System i

ฐานข้อมูล
การโปรแกรม SQL

เวอร์ชัน 6 รีลีส 1

System i

ฐานข้อมูล
การโปรแกรม SQL

เวอร์ชัน 6 รีลีส 1

หมายเหตุ

ก่อนที่จะใช้ข้อมูลนี้และผลิตภัณฑ์ที่ข้อมูลนี้สนับสนุน โปรดอ่านข้อมูลใน “คำประกาศ”, ในหน้า 355

การพิมพ์ครั้งนี้ใช้ได้กับ IBM i5/OS เวอร์ชัน 6 รีลีส 1 โมดิฟิเคชัน 0 (หมายเลขผลิตภัณฑ์ 5761-SS1) และใช้กับรีลีสและโมดิฟิเคชันถัดจากนี้ไป จนกว่าจะมีการระบุเป็นอย่างอื่นในการพิมพ์ครั้งใหม่ เวอร์ชันนี้ไม่สามารถรันบนโมเดล reduced instruction set computer (RISC) และโมเดล CISC ได้ทุกรุ่น

สารบัญ

การโปรแกรม SQL

ฐานข้อมูล DB2® for i5/OS® มี การสนับสนุนที่หลากหลายสำหรับ Structured Query Language (SQL)

ตัวอย่างคำสั่ง SQL ที่แสดงไว้ในหัวข้อนี้อิงจากตารางตัวอย่าง โดยถือว่ามีลักษณะดังนี้:

- ตัวอย่างดังกล่าวแสดงไว้ในสภาวะแวดล้อม SQL แบบโต้ตอบ หรือถูกบันทึกไว้ใน ILE C หรือใน COBOL. ใช้ EXEC SQL และ END-EXEC เพื่อคั่นคำสั่ง SQL ในโปรแกรม COBOL.
- SQL แต่ละตัวอย่างแสดงไว้บนบรรทัดต่างๆ โดยคำสั่งแต่ละคำสั่งจะอยู่คนละบรรทัดกัน.
- คีย์เวิร์ด SQL จะถูกไฮไลต์ไว้.
- ชื่อของตารางตัวอย่างนี้ใช้แบบแผน CORPDATA. สำหรับชื่อตารางซึ่งไม่พบในตารางตัวอย่าง ควรใช้แบบแผนที่คุณสร้างขึ้น.
- คอลัมน์ที่มีการคำนวณจะอยู่ในวงเล็บ () และวงเล็บก้ามปู []
- มีการใช้หลักการตั้งชื่อ SQL.
- มีการใช้อ็อพชันพรีคอมไพเลอร์ APOST และ APOSTSQL แม้ว่าจะไม่ใช่อ็อพชันดีฟอลต์ในCOBOL. literal ของสตริงอักขระภายในคำสั่ง SQL และคำสั่งภาษาโฮสต์ถูกคั่นด้วยเครื่องหมายอัญประกาศเดี่ยว (').
- มีการใช้การเรียงลำดับ *HEX, เว้นแต่จะมีการแจ้งเป็นอย่างอื่น.

เมื่อใดก็ตามที่ตัวอย่างแตกต่างไปจากสมมติฐานเหล่านี้, จะมีการแจ้งไว้.

เนื่องจากหัวข้อนี้มีไว้สำหรับแอ็พพลิเคชันโปรแกรมเมอร์, ตัวอย่างส่วนใหญ่จึงแสดงไว้ในลักษณะที่บันทึกไว้ในแอ็พพลิเคชันโปรแกรม. อย่างไรก็ตาม, ตัวอย่างจำนวนมาก สามารถทำการเปลี่ยนแปลงได้เล็กน้อยและรันแบบโต้ตอบด้วยการใช้ SQL แบบโต้ตอบ. ไวยากรณ์ของคำสั่ง SQL, เมื่อใช้ SQL แบบโต้ตอบ, จะแตกต่างเล็กน้อยจากรูปแบบคำสั่งเดียวกันเมื่อใส่อยู่ในโปรแกรม.

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ "สิทธิในรหัส และข้อมูลถ้อยแถลง" ในหน้า 353.

หลักการที่เกี่ยวข้อง

Embedded SQL programming

สิ่งอ้างอิงที่เกี่ยวข้อง

"DB2 for i5/OS ตารางตัวอย่าง" ในหน้า 329
ตารางตัวอย่างต่อไปนี้ถูกอ้างอิงและใช้งานใน หัวข้อการโปรแกรม SQL และการอ้างอิง SQL
การอ้างอิง DB2 สำหรับ i5/OS SQL

มีอะไรใหม่ใน V6R1

อ่านเกี่ยวกับข้อมูลใหม่หรือข้อมูลที่เปลี่ยนแปลงอย่างมาก สำหรับกลุ่มหัวข้อ SQL programming

คอลัมน์ Row change timestamp

เมื่อคุณสร้างตาราง คุณสามารถกำหนดคอลัมน์ในตารางให้เป็น คอลัมน์ row change timestamp ได้ แต่ละครั้งที่แถวถูกใส่เพิ่มหรือเปลี่ยนแปลงในตาราง ค่าในคอลัมน์ row change timestamp จะถูกตั้งเป็น timestamp ที่สัมพันธ์กับเวลาที่ทำการแทรกหรืออัพเดต หากต้องการข้อมูลเพิ่มเติม โปรดดู “การสร้างคอลัมน์ row change timestamp” ในหน้า 26

นิพจน์การเปลี่ยนแถว

นิพจน์ ROW CHANGE TIMESTAMP และ ROW CHANGE TOKEN สามารถ ใช้เพื่อระบุว่าแถวถูกเปลี่ยนแปลงล่าสุดเมื่อไร หากต้องการข้อมูลเพิ่มเติม โปรดดู “นิพจน์การเปลี่ยนแถว” ในหน้า 60

Full outer join

DB2 for i5/OS สนับสนุน full outer join ก่อนหน้านี้ คุณสามารถจำลอง full outer join โดยใช้ left outer join และ right exception join เท่านั้น สำหรับตัวอย่างของ full outer join โปรดดู “Full outer join” ในหน้า 73

SELECT จาก INSERT

เมื่อคุณแทรกแถวหนึ่งหรือมากกว่าลงในตาราง คุณสามารถเลือก แถวผลลัพธ์ของการแทรก โดยระบุคำสั่ง INSERT ใน FROM clause ของคำสั่ง SELECT หากต้องการข้อมูลเพิ่มเติม โปรดดู “การเลือกค่าที่แทรก” ในหน้า 101

การสนับสนุนทศนิยม

DB2 for i5/OS SQL สนับสนุนชนิดข้อมูล decimal floating-point:

- “การแปลงชนิดข้อมูลที่อนุญาต” ในหน้า 33
- “ข้อผิดพลาดในการดึงข้อมูล” ในหน้า 95
- “พารามิเตอร์ที่ผ่านหลักการสำหรับสำหรับโพรซีเดอร์ที่เก็บไว้และฟังก์ชันแบบผู้ใช้กำหนดเอง” ในหน้า 173

การสนับสนุนไฟล์ stream ต้นฉบับ

ไฟล์ต้นฉบับย่อยหรือไฟล์ stream ต้นฉบับสามารถมี ต้นฉบับสำหรับแอ็พพลิเคชันโปรแกรม DB2 for i5/OS ตัวประมวลผลคำสั่ง SQL ยอมให้คำสั่งทำงานได้จากรายการ ต้นฉบับย่อย หรือไฟล์ stream ต้นฉบับ หากต้องการข้อมูลเพิ่มเติม โปรดดูหัวข้อต่อไปนี้:

- “แนวคิด SQL” ในหน้า 4
- “อ็อบเจ็กต์แอ็พพลิเคชันโปรแกรม” ในหน้า 14
- “ไฟล์ต้นฉบับของผู้ใช้” ในหน้า 16
- “การใช้ตัวประมวลผลคำสั่ง SQL” ในหน้า 298
- “การแสดงรายการต้นฉบับสำหรับตัวประมวลผลคำสั่ง SQL” ในหน้า 300

การเปลี่ยนแปลงฟังก์ชันอื่นๆ สำหรับข้อมูล SQL programming

- ค่าดีฟอลต์, แอ็ตทริบิวต์ที่ซ่อน และแอ็ตทริบิวต์ row change timestamp ยังสามารถรวมไว้ใน LIKE clause หรือ AS clause ของคำสั่ง CREATE TABLE โปรดดู “การสร้างตารางโดยใช้ LIKE” ในหน้า 23 และ “การสร้างตารางโดยใช้ AS” ในหน้า 23

- สามารถใช้นิพจน์ส่วนใหญ่ที่อนุญาตโดย SQL ใน definition ของคอลัมน์หลักสำหรับดรรชนี สำหรับตัวอย่าง โปรดดู “การสร้างดรรชนี” ในหน้า 41
- มีการเพิ่มเรจิสเตอร์พิเศษใหม่ๆ ไว้ในหัวข้อ “เรจิสเตอร์พิเศษในคำสั่ง SQL” ในหน้า 57
- หัวข้อ “Concurrency” ในหน้า 131 ได้ถูกอัพเดต เพื่อรวม ข้อควรพิจารณาสำหรับการใช้ SKIP LOCKED DATA clause
- หัวข้อ “ตัวแปรตัวบ่งชี้และโพรซีเดอร์ที่เก็บไว้” ในหน้า 179 ได้ถูกอัพเดต เพื่อ บันทึกข้อมูลการสนับสนุนตัวบ่งชี้ที่ขยาย

การเปลี่ยนแปลงที่ไม่ใช้ด้านเทคนิคสำหรับข้อมูล SQL programming

- รายละเอียดเกี่ยวกับการเข้าใช้ข้อมูล DB2 for i5/OS ผ่านทาง อินเตอร์เฟสไคลเอ็นต์ ได้ถูกย้ายจากกลุ่มหัวข้อ SQL programming ไปยังกลุ่มหัวข้อ การดูแลฐานข้อมูล
- ตัวอย่างของทริกเกอร์ภายนอกได้ถูกย้ายไปยังหัวข้อ ตัวอย่าง: ทริกเกอร์โปรแกรม ใน กลุ่มหัวข้อโปรแกรมมิงฐานข้อมูล

วิธีการดูหัวข้อมีอะไรใหม่ หรือที่เปลี่ยนแปลง

เพื่อช่วยคุณให้ดูที่ซึ่งมีการเปลี่ยนแปลงทางด้านเทคนิค, information center จะใช้:

- รูปภาพ » เพื่อทำเครื่องหมายว่า ที่ใดคือข้อมูลใหม่ หรือข้อมูลที่เปลี่ยนแปลง.
- รูปภาพ « เพื่อทำเครื่องหมายว่า ที่ใดคือจุดสิ้นสุดของข้อมูลใหม่ หรือข้อมูลที่เปลี่ยนแปลง.

ในไฟล์ PDF คุณอาจเห็น revision bar (|) ที่ ระยะขอบด้านซ้ายของข้อมูลใหม่และข้อมูลที่เปลี่ยนแปลง

หากต้องการค้นหาข้อมูลเกี่ยวกับหัวข้อมีอะไรใหม่ หรือที่เปลี่ยนแปลงในรีลีสนี้, โปรดดู บันทึกข้อความถึงผู้ใช้.

ไฟล์ PDF สำหรับการโปรแกรม SQL

คุณสามารถดูและพิมพ์ไฟล์ PDF ของข้อมูลนี้ได้

ถ้า ต้องการดูหรือดาวน์โหลดเอกสารนี้ในเวอร์ชัน PDF ให้เลือก SQL programming (มีขนาดประมาณ 3900 KB)

การบันทึกไฟล์ PDF

ถ้าต้องการบันทึกไฟล์ PDF ลงที่เวิร์กสเตชันของคุณเพื่อนำมาอ่านหรือพิมพ์ภายหลัง:

1. คลิกขวาที่ลิงก์ PDF ในบราวเซอร์ของคุณ
2. คลิกที่อ็อพชันที่จะบันทึก PDF แบบโลคัล.
3. นำทางไปจนถึงไดเร็กทอรีที่คุณต้องการบันทึกไฟล์ PDF.
4. เลือก **Save**.

การดาวน์โหลด Adobe Reader

คุณจำเป็นต้องติดตั้ง Adobe® Reader บนระบบของคุณ เพื่อดูหรือพิมพ์ไฟล์ PDF เหล่านี้ คุณสามารถดาวน์โหลดโปรแกรมได้จาก เว็บไซต์ของ Adobe (www.adobe.com/products/acrobat/readstep.html)

บทนำสู่ DB2 for i5/OS Structured Query Language

Structured Query Language (SQL) เป็นภาษามาตรฐาน สำหรับการกำหนดและจัดการข้อมูลในฐานข้อมูลเชิงสัมพันธ์ หัวข้อเหล่านี้อธิบายถึงการปรับใช้ SQL ของ System i™ โดย ใช้ฐานข้อมูล DB2 for i5/OS และ ไลเซนส์โปรแกรม IBM® DB2 Query Manager and SQL Development Kit for i5/OS

SQL จัดการข้อมูลซึ่งอยู่ในแบบจำลองข้อมูลเชิงสัมพันธ์. สามารถนำคำสั่ง SQL ฝังลงในภาษาชั้นสูง, หรือนำมาจัดเตรียมแบบไดนามิกก่อนรัน หรือนำมารันแบบโต้ตอบ. สำหรับข้อมูลเกี่ยวกับ SQL แบบฝัง ให้ดู Embedded SQL programming

SQL จะประกอบด้วยคำสั่งและข้อความ ที่อธิบายถึงสิ่งที่คุณต้องดำเนินการกับข้อมูลในฐานข้อมูลและเงื่อนไขต่างๆ ที่ทำให้คุณต้องการดำเนินการดังกล่าว.

SQL สามารถเข้าถึงข้อมูลในฐานข้อมูลเชิงสัมพันธ์แบบรีโมต โดยใช้ IBM Distributed Relational Database Architecture™ (DRDA®)

หลักการที่เกี่ยวข้อง

โปรแกรมมิงฐานข้อมูลแบบกระจาย

สิ่งอ้างอิงที่เกี่ยวข้อง

"ฟังก์ชันของฐานข้อมูลเชิงสัมพันธ์แบบกระจายและ SQL" ในหน้า 302
ฐานข้อมูลเชิงสัมพันธ์แบบกระจาย ประกอบไปด้วยชุด SQL อ็อบเจ็กต์ที่กระจายอยู่บนระบบคอมพิวเตอร์ที่เชื่อมต่อถึงกันและกัน.

แนวคิด SQL

DB2 for i5/OS SQL ประกอบด้วยส่วนหลักๆ หลายส่วน เช่น การสนับสนุนรันไทม์ของ SQL, พรีคอมไพเลอร์ และ SQL แบบโต้ตอบ

- ตัวสนับสนุนรันไทม์ SQL

  รันไทม์ SQL จะวิเคราะห์คำสั่ง SQL และรันคำสั่ง SQL ใดๆ การสนับสนุนนี้คือ ส่วนหนึ่งของไลเซนส์โปรแกรม i5/OS ที่อนุญาตให้แอ็พพลิเคชันที่มีคำสั่ง SQL สามารถรันได้บนระบบ โดยไม่ต้องติดตั้งไลเซนส์โปรแกรม IBM DB2 Query Manager and SQL Development Kit for i5/OS
- SQL พรีคอมไพเลอร์

  SQL พรีคอมไพเลอร์สนับสนุนคำสั่ง SQL ที่ฝังอยู่แบบพรีคอมไพล์ในภาษาโฮสต์. ภาษาต่อไปนี้เป็นภาษาที่ได้รับการสนับสนุน:
  - ILE C
  - ILE C++
  - ILE COBOL
  - COBOL
  - PL/I
  - RPG III (ส่วนหนึ่งของ RPG)
  - ILE RPG

SQL พรีคอมไพเลอร์ของภาษาโฮสต์จะเตรียมแอ็พพลิเคชันโปรแกรมที่มีคำสั่ง SQL คอมไพเลอร์ภาษาโฮสต์จึงคอมไพล์ซอร์สโปรแกรมโฮสต์แบบพรีคอมไพล์. สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ พรีคอมไพเลอร์โปรดดูหัวข้อ การเตรียมและ การรันโปรแกรมด้วยคำสั่ง SQL ใน Embedded SQL programming ตัวสนับสนุนพรีคอมไพเลอร์คือส่วนหนึ่งของไลเซนส์โปรแกรม IBM DB2 Query Manager and SQL Development Kit for i5/OS.

- อินเตอร์เฟสแบบโต้ตอบของ SQL

  อินเตอร์เฟสแบบโต้ตอบของ SQL ทำให้คุณสามารถสร้างและรันคำสั่ง SQL หากต้องการทราบข้อมูลเพิ่มเติมเกี่ยวกับ SQL แบบโต้ตอบ, โปรดดูที่ “การใช้ SQL แบบโต้ตอบ” ในหน้า 285. SQL แบบโต้ตอบคือส่วนหนึ่งของไลเซนส์โปรแกรม IBM DB2 Query Manager and SQL Development Kit for i5/OS.

- Run SQL Scripts

  หน้าต่าง Run SQL Scripts ใน System i Navigator จะให้คุณสร้าง, แก้ไข, รัน, และแก้ปัญหาเบื้องต้นเกี่ยวกับสคริปต์คำสั่ง SQL.

- คำสั่ง CL สำหรับ Run SQL Statements (RUNSQLSTM)

  คำสั่ง RUNSQLSTM สามารถใช้เพื่อรันชุดคำสั่ง SQL ที่เก็บไว้ใน ไฟล์ต้นฉบับหรือไฟล์ stream ต้นฉบับ หากต้องการข้อมูลเพิ่มเติมเกี่ยวกับคำสั่ง RUNSQLSTM โปรดดูที่ “การใช้ตัวประมวลผลคำสั่ง SQL” ในหน้า 298

- DB2 Query Manager

  DB2 Query Manager มีอินเตอร์เฟสแบบโต้ตอบที่ทำงานโดยคำสั่ง ซึ่งอนุญาตให้คุณสร้างข้อมูล, เพิ่มข้อมูล, รักษาข้อมูล และรันรายงานบนฐานข้อมูลได้ Query Manager เป็น ส่วนหนึ่งของไลเซนส์โปรแกรม IBM DB2 Query Manager and SQL Development Kit for i5/OS หากต้องการข้อมูลเพิ่มเติม โปรดดูที่ Query Manager Use 

- อินเตอร์เฟส SQL REXX™

  อินเตอร์เฟส SQL REXX จะทำให้คุณรันคำสั่ง SQL ในโพรซีเดอร์ REXX หากต้องการข้อมูลเพิ่มเติมเกี่ยวกับการใช้คำสั่ง SQL ในโพรซีเดอร์ REXX โปรดดูที่ การเขียนโค้ดคำสั่ง SQL ในแอ็พพลิเคชัน REXX ใน Embedded SQL programming

- SQL call level interface

  ฐานข้อมูล DB2 for i5/OS สนับสนุน SQL call level interface ซึ่งจะยอมให้ผู้ใช้ภาษา ILE ใดๆ สามารถเรียกใช้ฟังก์ชัน SQL โดยตรงผ่านการเรียกใช้ เซอร์วิสโปรแกรมที่จัดเตรียมโดยระบบ เมื่อใช้ SQL call level interface คุณจะสามารถใช้งานฟังก์ชัน SQL ทั้งหมดได้โดยไม่จำเป็นต้องพรีคอมไพล์ นี่คือชุดคำสั่งมาตรฐานใช้เรียกเพื่อเตรียมข้อความ SQL, รันข้อความ SQL, ดึงแถวของข้อมูลออกมา และทำแม้กระทั่งฟังก์ชันระดับสูง เช่น การเรียกใช้ แคตาล็อก และการเชื่อมโยงตัวแปรโปรแกรมไปยังคอลัมน์ของเอาต์พุต

  สำหรับ รายละเอียดที่สมบูรณ์ของฟังก์ชันทั้งหมดที่มีอยู่และไวยากรณ์ โปรดดูหัวข้อ SQL call level interface ใน ส่วนของฐานข้อมูลของ i5/OS Information Center

- Process Extended Dynamic SQL (QSQPRCED) API

  application programming interface (API) นี้จะทำให้ SQL สามารถทำงานแบบ dynamic ได้ คุณสามารถจัดเตรียมคำสั่ง SQL ไว้ใน SQL แพ็กเกจ และรันคำสั่งโดยการใช้ API นี้ คำสั่งที่ถูกจัดเตรียมเป็นแพ็กเกจด้วย API นี้จะยังคงอยู่ จนกระทั่งแพ็กเกจหรือคำสั่งถูกลบออกไป สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ QSQPRCED API โปรดดูหัวข้อ Process Extended Dynamic SQL (QSQPRCED) API สำหรับข้อมูลทั่วไปเกี่ยวกับ API โปรดดูหัวข้อ Application programming interfaces

- Syntax Check SQL Statement (QSQCHKS) API

  ไวยากรณ์ API จะตรวจสอบคำสั่ง SQL. สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ QSQCHKS API โปรดดูหัวข้อ Syntax Check SQL Statement (QSQCHKS) API สำหรับข้อมูลทั่วไปเกี่ยวกับ API โปรดดูหัวข้อ Application programming interfaces

- DB2 Multisystem

คุณลักษณะนี้ของระบบปฏิบัติการจะทำให้ข้อมูลของคุณกระจายอย่างทั่วถึงบนหลายๆ ระบบ หากต้องการข้อมูลเพิ่มเติม โปรดดู DB2 Multisystem

- DB2 Symmetric Multiprocessing

  คุณลักษณะนี้ของระบบปฏิบัติการจะมี query optimizer และวิธีการเพิ่มเติมในการเรียกข้อมูลด้วยการประมวลผลแบบขนาน. Symmetric multiprocessing (SMP) เป็นรูปแบบของการเข้าใช้งานแบบขนานบนระบบเดี่ยว ที่ซึ่งโพรเซสเซอร์หลายตัว (CPU และตัวประมวลผล I/O) ที่แบ่งใช้งานรีซอร์สของหน่วยความจำ และ ดิสก์ทำงานร่วมกัน เพื่อให้ได้มาซึ่งผลลัพธ์อย่างเดียวในตอนสุดท้าย. การประมวลผลแบบขนานหมายความว่าตัวจัดการฐานข้อมูลสามารถรองรับการสืบค้นจากตัวประมวลผลระบบได้มากกว่าหนึ่งตัว (หรือทั้งหมด) พร้อมๆ กัน. หากต้องการข้อมูลเพิ่มเติม โปรดดู การควบคุม การประมวลผลแบบขนานสำหรับเคียวรี่ ในกลุ่มหัวข้อ ประสิทธิภาพการทำงานของฐานข้อมูลและ การใช้งานเคียวรี่ให้ได้ผลดีที่สุด

ฐานข้อมูลเชิงสัมพันธ์ SQL และศัพท์เฉพาะระบบ

ในแบบจำลองข้อมูลเชิงสัมพันธ์ จะถือว่าข้อมูลทั้งหมดมีอยู่ในตาราง DB2 for i5/OS อ็อบเจ็กต์ ถูกสร้างและรักษาไว้ในฐานะเป็นอ็อบเจ็กต์ระบบ

ตารางต่อไปนี้จะแสดงความสัมพันธ์ระหว่างศัพท์เฉพาะของระบบ และศัพท์เฉพาะของฐานข้อมูลเชิงสัมพันธ์.

ตารางที่ 1. ความสัมพันธ์ของศัพท์เฉพาะระบบกับศัพท์เฉพาะ SQL

ศัพท์เฉพาะระบบ	ศัพท์เฉพาะ SQL
ไลบรารี. จัดกลุ่มอ็อบเจ็กต์ที่สัมพันธ์กันและให้คุณสามารถค้นหาอ็อบเจ็กต์ได้ ด้วยชื่อของอ็อบเจ็กต์.	**แบบแผน**. ประกอบด้วยไลบรารี, เจอร์นัล, journal receiver, แค็ตตาล็อก SQL , และอาจรวมถึงพจนานุกรมข้อมูล. แบบแผนจะจัดกลุ่มอ็อบเจ็กต์ที่สัมพันธ์กันและให้คุณค้นหาอ็อบเจ็กต์ได้ด้วยชื่อ.
ไฟล์ฟิสิคัล. เซ็ตของเร็กคอร์ด.	**ตาราง**. เซ็ตของคอลัมน์และแถว.
เร็กคอร์ด. เซ็ตของฟิลด์.	**แถว**. ส่วนที่เป็นแนวนอนของตาราง ซึ่งเซ็ตของคอลัมน์เรียงลำดับอยู่.
ฟิลด์. อักขระหนึ่งตัวขึ้นไปของข้อมูลที่เกี่ยวข้องในประเภทข้อมูลหนึ่งประเภท.	**คอลัมน์**. ส่วนที่เป็นแนวตั้งของตารางประเภทข้อมูลหนึ่งประเภท.
ไฟล์ลอจิคัล. กลุ่มย่อยของฟิลด์และเร็คคอร์ดของไฟล์ฟิสิคัลหนึ่งไฟล์ขึ้นไป.	**มุมมอง**. เซ็ตย่อยของคอลัมน์และแถวของตารางหนึ่งตารางขึ้นไป.
แพ็กเกจ SQL ชนิดของอ็อบเจ็กต์ ที่มีการใช้ในการรันคำสั่ง SQL .	**แพ็กเกจ**. ชนิดของอ็อบเจ็กต์ ที่มีการใช้ในการรันคำสั่ง SQL .
โปรไฟล์ผู้ใช้	**ชื่อหรือรหัสที่ได้รับอนุญาต**

หลักการที่เกี่ยวข้อง

โปรแกรมมิงฐานข้อมูลแบบกระจาย

SQL และหลักการตั้งชื่อของระบบ

คุณสามารถใช้หลักการตั้งชื่อของระบบ (*SYS) หรือ SQL (*SQL) ในการเขียนโปรแกรม DB2 for i5/OS

| หลักการตั้งชื่อที่ใช้จะมีผลต่อวิธีการคัดเลือกชื่อไฟล์และตารางและศัพท์เฉพาะที่ใช้งานบนหน้าจอ SQL แบบโต้ตอบ. หลัก
| การตั้งชื่อที่ใช้งานจะถูกเลือกโดยพารามิเตอร์บนคำสั่ง SQL หรือใช้คำสั่ง SET OPTION

การตั้งชื่อระบบ (*SYS)

ในหลักการตั้งชื่อระบบ, ตาราง และ SQL อ็อบเจ็กต์อื่นๆ ในคำสั่ง SQL ที่ถูกต้องจะต้องอยู่ในรูปแบบ:

```
แบบแผน/ตาราง
```

การตั้งชื่อ SQL (*SQL)

ในหลักการตั้งชื่อ SQL ตารางและ SQL อ็อบเจ็กต์อื่นๆ ในคำสั่ง SQL ที่ถูกต้องตามชื่อ schema จะต้องอยู่ในรูปแบบ:

```
แบบแผน.ตาราง
```

สิ่งอ้างอิงที่เกี่ยวข้อง

คุณสมบัติของชื่ออ็อบเจ็กต์ที่ไม่ตรงตามเกณฑ์

ประเภทของคำสั่ง SQL

มีประเภทของคำสั่ง SQL พื้นฐานอยู่หลายประเภท ซึ่งคำสั่งเหล่านั้นจะถูกแสดงไว้ที่นี่ตามฟังก์ชัน

- คำสั่งแบบแผน SQL, ซึ่งรู้จักกันว่า คำสั่ง data definition language (DDL)
- ข้อมูล SQL และคำสั่งการเปลี่ยนข้อมูล, ซึ่งรู้จักกันว่า คำสั่ง data manipulation language (DML)
- คำสั่ง SQL แบบ Dynamic
- คำสั่งภาษาโฮสต์ SQL แบบฝัง

คำสั่งแบบแผน SQL

ALTER FUNCTION
ALTER PROCEDURE
ALTER SEQUENCE
ALTER TABLE
COMMENT ON
CREATE ALIAS
CREATE DISTINCT TYPE
CREATE FUNCTION
CREATE INDEX
CREATE PROCEDURE
CREATE SCHEMA
CREATE SEQUENCE
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
DROP ALIAS
DROP DISTINCT TYPE
DROP FUNCTION
DROP INDEX
DROP PACKAGE
DROP PROCEDURE
DROP SEQUENCE
DROP SCHEMA
DROP TABLE
DROP TRIGGER
DROP VIEW
GRANT DISTINCT TYPE
GRANT FUNCTION
GRANT PACKAGE
GRANT PROCEDURE
GRANT SEQUENCE
GRANT TABLE
LABEL ON
RENAME
REVOKE DISTINCT TYPE
REVOKE FUNCTION
REVOKE PACKAGE
REVOKE PROCEDURE
REVOKE SEQUENCE
REVOKE TABLE

คำสั่งข้อมูล SQL

CLOSE
DECLARE CURSOR
DELETE
FETCH
FREE LOCATOR
HOLD LOCATOR
INSERT
LOCK TABLE
OPEN
REFRESH TABLE
SELECT INTO
SET variable
UPDATE
VALUES INTO

คำสั่งการเปลี่ยนข้อมูล SQL

DELETE
INSERT
UPDATE

คำสั่งการเชื่อมต่อ SQL

CONNECT
DISCONNECT
RELEASE
SET CONNECTION

คำสั่งรายการ SQL
COMMIT
RELEASE SAVEPOINT
ROLLBACK
SAVEPOINT
SET TRANSACTION

คำสั่งเซสชัน SQL
DECLARE GLOBAL TEMPORARY TABLE
SET CURRENT DECFLOAT ROUNDING MODE
SET CURRENT DEGREE
SET ENCRYPTION PASSWORD
SET PATH
SET SCHEMA
SET SESSION AUTHORIZATION

คำสั่ง SQL แบบ Dynamic
ALLOCATE DESCRIPTOR
DEALLOCATE DESCRIPTOR
DESCRIBE
DESCRIBE INPUT
DESCRIBE TABLE
EXECUTE
EXECUTE IMMEDIATE
GET DESCRIPTOR
PREPARE
SET DESCRIPTOR

คำสั่งภาษาโฮสต์ SQL แบบฝัง
BEGIN DECLARE SECTION
DECLARE PROCEDURE
DECLARE STATEMENT
DECLARE VARIABLE
END DECLARE SECTION
GET DIAGNOSTICS
INCLUDE
SET OPTION
SET RESULT SETS
SIGNAL
WHENEVER

SQL control statement
CALL

คำสั่ง SQL สามารถปฏิบัติการกับอ็อบเจ็กต์ที่ถูกสร้างโดย SQL รวมทั้งฟิสิคัลไฟล์ชนิด externally described และโลจิคัลไฟล์ชนิด single-format คำสั่ง SQL จะไม่อ้างอิงถึงคำนิยามในพจนานุกรม interactive data definition utility (IDDU) สำหรับไฟล์ที่อธิบายด้วยโปรแกรม ไฟล์ที่อธิบายด้วยโปรแกรมจะปรากฏเป็นตารางที่มีคอลัมน์เดี่ยว.

หลักการที่เกี่ยวข้อง

"Data definition language" ในหน้า 18
Data definition language (DDL) คือส่วน ของ SQL ที่สร้าง เปลี่ยน และลบอ็อบเจ็กต์ฐานข้อมูล อ็อบเจ็กต์ฐานข้อมูล เหล่านี้ประกอบด้วยแบบแผน ตาราง มุมมอง ลำดับ แคตาล็อก ดรรชนี และ alias

"ภาษาสำหรับการดำเนินการข้อมูล" ในหน้า 44
ภาษาสำหรับการดำเนินการข้อมูล (DML) คือส่วนของ SQL ที่ดำเนินการ หรือควบคุมข้อมูล

สิ่งอ้างอิงที่เกี่ยวข้อง

การอ้างอิง DB2 สำหรับ i5/OS SQL

พื้นที่สื่อสาร SQL

พื้นที่สื่อสาร SQL (SQLCA) เป็นชุดตัวแปร ที่จัดหาข้อมูลเกี่ยวกับการรันคำสั่ง SQL ให้แก่ แอ็พพลิเคชันโปรแกรม SQLCA จะถูกอัพเดตเมื่อสิ้นสุดการรัน คำสั่ง SQL ทุกคำสั่ง

หลักการที่เกี่ยวข้อง

SQLCA (SQL communication area)

การจัดการค่าส่งคืนข้อผิดพลาด SQL โดยใช้ SQLCA

พื้นที่วินิจฉัย SQL

พื้นที่วินิจฉัย SQL คือชุดของข้อมูลที่ดูแลโดยตัวจัดการ ฐานข้อมูลที่เกี่ยวกับคำสั่ง SQL ซึ่งมีการรันงาน ครั้งล่าสุด แอ็พพลิเคชันโปรแกรมของคุณสามารถเข้าถึงพื้นที่วินิจฉัย SQL โดยใช้ คำสั่ง GET DIAGNOSTICS

หลักการที่เกี่ยวข้อง

การใช้พื้นที่วินิจฉัย SQL

สิ่งอ้างอิงที่เกี่ยวข้อง

GET DIAGNOSTICS statement

SQL อ็อบเจ็กต์

SQL *อ็อบเจ็กต์* คือแบบแผน, เจอร์นัล, แคตาล็อก, ตาราง, aliase, มุมมอง, ดรรชนี, ข้อจำกัด, ทริกเกอร์, ลำดับ, โพรซีเดอร์ที่เก็บไว้, ฟังก์ชันแบบผู้ใช้กำหนดเอง, ประเภทแบบผู้ใช้กำหนดเอง, และ SQL แพ็กเกจ. SQL จะสร้างและรักษาอ็อบเจ็กต์เหล่านี้ให้เป็นอ็อบเจ็กต์ระบบ.

แบบแผน (Schemas)

แบบแผนจะมีการจัดกลุ่มของ SQL อ็อบเจ็กต์แบบโลจิคัล *แบบแผน* ประกอบด้วยไลบรารี, เจอร์นัล, journal receiver, แคตาล็อก และอาจมีพจนานุกรมข้อมูล

การสร้าง, ย้าย, หรือเก็บตาราง, มุมมอง และอ็อบเจ็กต์ระบบ (เช่นโปรแกรม) อาจไว้ในไลบรารี, ระบบใดๆ ก็ได้ . ไฟล์ระบบทั้งหมดอาจถูกสร้างหรือย้ายไปยังแบบแผน SQL หากแบบแผน SQL ไม่มีพจนานุกรมข้อมูล. หากแบบแผน SQL มีพจนานุกรมข้อมูลแล้ว:

- source physical ไฟล์หรือ nonsource physical ไฟล์ที่มีต้นทางที่มีรายการย่อยเดียวสามารถถูกสร้าง, ย้าย, หรือ เก็บไว้ในแบบแผน SQL ได้.
- เราไม่สามารถวางโลจิคัลไฟล์ในแบบแผน SQL ได้เนื่องจากไฟล์เหล่านั้นไม่สามารถอธิบายอยู่ใน พจนานุกรมข้อมูลได้.

คุณสามารถสร้างและเป็นเจ้าของแบบแผนได้หลายอัน.

เจอร์นัลและตัวรับเจอร์นัล

เจอร์นัล และ *journal receiver* จะใช้ บันทึกการเปลี่ยนแปลงของตารางและมุมมองในฐานข้อมูล

เจอร์นัลและ journal receivers จึงใช้ในการประมวลผล คำสั่ง SQL COMMIT, ROLLBACK, SAVEPOINT และ RELEASE SAVEPOINT เจอร์นัล และ journal receivers ยังสามารถใช้งานเป็นหลักฐานการตรวจสอบ หรือใช้สำหรับ forward recovery หรือ backward recovery

หลักการที่เกี่ยวข้อง

การจัดการเจอร์นัล

Commitment control

แคตตาล็อก

แคตาล็อก SQL คือชุด ตาราง และมุมมอง ซึ่งอธิบายตาราง, มุมมอง, ดรรชนี, แพ็กเกจ, โพรซีเดอร์, ฟังก์ชัน, ไฟล์, ลำดับ, ทริกเกอร์, และข้อจำกัด

ข้อมูลนี้อยู่ในชุดตาราง cross-reference ในไลบรารี QSYS และ QSYS2. ในแบบแผน SQL จะมีชุดมุมมองซึ่งถูกสร้างขึ้นจากตารางแคตาล็อกซึ่งมีข้อมูลเกี่ยวกับตาราง, มุมมอง, ดรรชนี, แพ็กเกจ, ไฟล์, และข้อจำกัดในแบบแผน.

แคตาล็อกจะถูกสร้างขึ้นโดยอัตโนมัติเมื่อคุณสร้างแบบแผน. คุณไม่สามารถลบหรือเปลี่ยนแคตาล็อก.

สิ่งอ้างอิงที่เกี่ยวข้อง

แคตาล็อก

ตาราง, แถว, และคอลัมน์

ตาราง เป็นการจัดการข้อมูลแบบสองด้าน ซึ่งประกอบด้วย *แถว* และ *คอลัมน์*

แถวคือส่วนที่เป็นแนวนอนซึ่งประกอบด้วยคอลัมน์ตั้งแต่หนึ่งคอลัมน์ขึ้นไป. คอลัมน์คือส่วนที่เป็นแนวดิ่งซึ่งประกอบด้วยแถวตั้งแต่หนึ่งแถวขึ้นไปในประเภทข้อมูลหนึ่งประเภท. ข้อมูลทั้งหมดในหนึ่งคอลัมน์ต้องเป็นข้อมูลประเภทเดียวกัน. ตารางใน SQL คือฟิสิคัลไฟล์ที่มีคีย์ หรือไม่มีคีย์ก็ได้.

materialized query table คือตารางที่ใช้ในการเก็บ ข้อมูลที่เป็นสื่อพิมพ์ที่ซึ่งแปลงมาจากตารางต้นฉบับหนึ่งหรือหลายตารางที่ระบุโดย select-statement.

partitioned table คือตารางที่มีข้อมูลที่เป็นส่วนประกอบใน โลคัลพาร์ติชัน (เมมเบอร์)หนึ่งหรือมากกว่าหนึ่ง.

หลักการที่เกี่ยวข้อง

DB2 Multisystem

สิ่งอ้างอิงที่เกี่ยวข้อง

ชนิดข้อมูล

"การสร้างและเปลี่ยนตาราง materialized query" ในหน้า 24
ตาราง materialized query เป็น ตารางที่มี definition อยู่บนพื้นฐานของผลลัพธ์ของเคียวรี่ และข้อมูลอยู่ในรูปแบบ ของผลลัพธ์ที่คำนวณไว้ล่วงหน้า ซึ่งรับมาจากตารางที่ definition ของตาราง materialized query ใช้อ้างอิง

Alias

alias คืออีกชื่อหนึ่งของตารางหรือมุมมอง.

คุณสามารถใช้ alias เพื่ออ้างถึงตารางหรือมุมมองในกรณีที่ทำได้. นอกจากนี้, คุณยังสามารถใช้ aliase เพื่อรวมตารางเข้าไว้ด้วยกัน.

สิ่งอ้างอิงที่เกี่ยวข้อง

Alias

มุมมอง

มุมมอง จะเป็นเหมือนตารางสำหรับ แอ็พพลิเคชันโปรแกรม อย่างไรก็ตาม มุมมองไม่มีข้อมูลใดๆ แต่จะแสดงเพียงแค่ ตารางหนึ่งหรือหลายตารางที่ข้อมูลนั้นถูกสร้างขึ้น

มุมมองสามารถรองรับคอลัมน์และแถวทั้งหมดของ ตารางหรือชุดย่อยของตาราง คอลัมน์ในมุมมองสามารถจัดวางให้ต่างไปจากที่เป็นอยู่ในตารางที่นำคอลัมน์นั้นมาได้. มุมมองใน SQL คือรูปแบบพิเศษของโลจิคัลไฟล์ที่ไม่มีคีย์.

สิ่งอ้างอิงที่เกี่ยวข้อง

มุมมอง

ดรรชนี (Index)

ดรรชนี SQL คือ ชุดข้อมูลย่อยในคอลัมน์ของตารางที่จัดเรียงตามลำดับการเรียงจากมากไปหาน้อยหรือจากน้อยไปหามาก ตามความเหมาะสม.

ดรรชนีแต่ละตัวมีการจัดเรียงที่แยกจากกัน. การจัดเรียงเหล่านี้ได้แก่ การเรียงลำดับ (ORDER BY clause), การจัดกลุ่ม (GROUP BY clause), และการเชื่อมโยง. ดรรชนี SQL คือ โลจิคัลไฟล์แบบมีคีย์.

ระบบจะใช้ดรรชนีเพื่อให้ดึงข้อมูลออกมาได้รวดเร็วขึ้น. คุณสามารถเลือกได้ว่าจะสร้างหรือไม่สร้างดรรชนีก็ได้. คุณสามารถสร้างดรรชนีจำนวนเท่าใดก็ได้. นอกจากนี้ คุณอาจสร้างหรือลบดรรชนีได้ตลอดเวลา. ดรรชนีจะถูกรักษาไว้โดยระบบโดยอัตโนมัติ. อย่างไรก็ตาม เนื่องจากดรรชนีจะถูกเก็บไว้บนระบบ ดังนั้นหากมี ดรรชนีจำนวนมากจะส่งผลต่อประสิทธิภาพการทำงานของแอ็พพลิเคชัน ที่เปลี่ยนตาราง

หลักการที่เกี่ยวข้อง

ยุทธวิธีในการสร้างดรรชนี

ข้อจำกัด

ข้อจำกัด เป็นกฎที่บังคับใช้โดยตัวจัดการฐานข้อมูล เพื่อจำกัดค่าที่สามารถแทรก ลบ หรืออัพเดต ในตาราง

DB2 for i5/OS สนับสนุนข้อจำกัดดังต่อไปนี้:

- ข้อจำกัดแบบเฉพาะ (Unique constraints)

  ข้อจำกัดแบบเฉพาะ คือ กฎที่บังคับว่าค่าของคีย์จะถูกต้องก็ต่อเมื่อค่านั้นเป็นค่าเฉพาะ. คุณสามารถ สร้างข้อจำกัดเฉพาะโดยใช้คำสั่ง CREATE TABLE หรือ ALTER TABLE แม้ว่าคำสั่ง CREATE INDEX สามารถสร้างดรรชนีแบบเฉพาะซึ่งจะไม่ซ้ำกัน แต่ดรรชนีดังกล่าวก็ไม่ถือเป็นข้อจำกัด

  ข้อจำกัดแบบเฉพาะจะถูกนำมาบังคับใช้เมื่อมีการรันคำสั่ง INSERT และ UPDATE. ข้อจำกัด PRIMARY KEY คือ รูปแบบข้อจำกัด UNIQUE ความแตกต่างคือ PRIMARY KEY ต้องไม่มีคอลัมน์ที่เป็น null.

- ข้อจำกัดแบบอ้างอิง (Referential constraints)

  ข้อจำกัดแบบอ้างอิง คือ กฎที่บังคับว่าค่าของคีย์ foreign จะถูกต้องก็ต่อเมื่อ:

  - ค่าเหล่านั้นปรากฏเป็นค่าของคีย์หลัก (parent key)
  - ส่วนประกอบบางตัวของคีย์ foreign เป็น null.

  ข้อจำกัดแบบอ้างอิงจะถูกบังคับใช้เมื่อมีการรันคำสั่ง INSERT, UPDATE, และ DELETE .

- ข้อจำกัดการตรวจสอบ

  ข้อจำกัดการตรวจสอบ คือ กฎที่จำกัดค่าที่ใช้ในคอลัมน์หรือกลุ่มคอลัมน์ คุณสามารถ สร้างข้อจำกัดการตรวจสอบ โดยใช้คำสั่ง CREATE TABLE หรือ ALTER TABLE ข้อจำกัดการตรวจสอบจะถูกบังคับใช้เมื่อมีการรันคำสั่ง INSERT และ UPDATE. หากต้องการปฏิบัติตามข้อจำกัด แถวข้อมูลแต่ละแถวที่แทรกเข้าไปหรือถูกอัพเดตในตาราง ต้องทำให้เงื่อนไขที่ระบุไว้เป็นแบบ TRUE หรือ ไม่รู้จัก (เนื่องจากเป็น null)

  สิ่งอ้างอิงที่เกี่ยวข้อง

  "ข้อจำกัด" ในหน้า 141

  ฐานข้อมูล DB2 for i5/OS สนับสนุน ข้อจำกัดเฉพาะ, ข้อจำกัดในการอ้างอิง และข้อจำกัดในการตรวจสอบ

ทริกเกอร์

ทริกเกอร์ คือชุดของ action ที่รันโดยอัตโนมัติ เมื่อใดก็ตามที่มีเหตุการณ์ที่ระบุไว้เกิดขึ้นกับตารางหรือมุมมองที่ระบุไว้

เหตุการณ์ดังกล่าวอาจเป็น การแทรก การอัพเดต การลบ หรือการอ่าน ทริกเกอร์สามารถรันก่อนหรือหลังเหตุการณ์นี้ DB2 for i5/OS จะสนับสนุน การแทรก SQL, อัพเดต และทริกเกอร์ที่ใช้ลบ และทริกเกอร์ภายนอก

งานที่เกี่ยวข้อง

การทำทริกเกอร์เหตุการณ์แบบอัตโนมัติในฐานข้อมูลของคุณ

โพรซีเดอร์ที่เก็บไว้

*โพรซีเดอร์ที่เก็บไว้*คือ โปรแกรม ที่อาจถูกเรียกโดยใช้คำสั่ง SQL CALL

DB2 for i5/OS สนับสนุน โพรซีเดอร์ที่เก็บไว้แบบภายนอกและ SQL โพรซีเดอร์ โพรซีเดอร์ภายนอกอาจเป็นโปรแกรม ระบบ, เซอร์วิสโปรแกรม หรือโพรซีเดอร์ REXX แต่ไม่สามารถเป็นโปรแกรมหรือโพรซีเดอร์ System/36™ SQL โพรซีเดอร์จะถูกกำหนดไว้ทั้งหมดใน SQL และอาจมีคำสั่ง SQL รวมทั้ง SQL control statements

หลักการที่เกี่ยวข้อง

"สตอร์โพรซีเดอร์" ในหน้า 146

โพรซีเดอร์ (ซึ่งมักเรียกว่า โพรซีเดอร์ที่เก็บไว้) คือโปรแกรมที่สามารถเรียกขึ้นมาเพื่อปฏิบัติงาน โพรซีเดอร์อาจประกอบด้วย คำสั่งภาษาโฮสต์และคำสั่ง SQL โพรซีเดอร์ใน SQL มีข้อดีเหมือนกับโพรซีเดอร์ในภาษาโฮสต์

ลำดับ (Sequences)

sequence คืออ็อบเจ็กต์พื้นที่ข้อมูล ที่กำหนดให้วิธีการที่ง่ายและรวดเร็วในการสร้างหมายเลขเฉพาะ.

คุณสามารถใช้ลำดับในการแทนที่คอลัมน์ identity หรือ คอลัมน์ตัวเลขที่ผู้ใช้สร้างขึ้นมาได้ ลำดับมีลักษณะการใช้งานคล้ายกับทางเลือกเหล่านี้

สิ่งอ้างอิงที่เกี่ยวข้อง

"การสร้างและการใช้ลำดับ" ในหน้า 28

ลำดับจะเหมือนกับคอลัมน์ identity ในเรื่องที่ว่าทั้งคู่จะสามารถสร้างค่าที่เป็น unique อย่างไรก็ตาม ลำดับเป็นอ็อบเจ็กต์ที่ขึ้นอยู่กับตาราง คุณสามารถใช้ลำดับเพื่อสร้างค่าได้อย่างรวดเร็ว และง่ายดาย

ฟังก์ชันแบบผู้ใช้กำหนด (User-defined functions)

ฟังก์ชันแบบผู้ใช้กำหนด คือโปรแกรมที่อาจถูกเรียกทำงาน ได้เหมือนกับฟังก์ชันในตัวอื่นๆ

DB2 for i5/OS สนับสนุนฟังก์ชันแบบภายนอก, ฟังก์ชัน SQL, และฟังก์ชันแบบต้นทาง (sourced functions). ฟังก์ชันแบบภายนอกอาจเป็นโปรแกรม ILE ของระบบใดๆ หรือเซอร์วิสโปรแกรม ฟังก์ชัน SQL ถูกกำหนดไว้ทั้งหมดใน SQL และอาจมีคำสั่ง SQL รวมทั้งคำสั่ง SQL control. ฟังก์ชันแบบต้นทางจะถูกสร้างขึ้นมาบนฟังก์ชันในตัวใด (built-in) หรือ บนฟังก์ชันแบบผู้ใช้กำหนดที่มีอยู่. คุณสามารถสร้างฟังก์ชัน scalar หรือฟังก์ชันตารางให้เป็นฟังก์ชัน SQL หรือฟังก์ชันแบบภายนอก

หลักการที่เกี่ยวข้อง

"การใช้ฟังก์ชันแบบผู้ใช้กำหนดเอง" ในหน้า 182

ในการเขียนแอ็พพลิเคชัน SQL คุณสามารถเลือก ปฏิบัติการหรือดำเนินการบางอย่างในแบบฟังก์ชันที่ผู้ใช้กำหนดเอง (UDF) หรือแบบรูทีนย่อย ในแอ็พพลิเคชันของคุณ ถึงแม้ว่ามันอาจจะดูง่ายกว่าในการเลือกการดำเนินการใหม่แบบ รูทีนย่อยในแอ็พพลิเคชันของคุณ คุณอาจต้องพิจารณาถึงประโยชน์ของการใช้งาน UDF แทน

ประเภทที่ผู้ใช้กำหนด (User-defined types)

ประเภทที่ผู้ใช้กำหนด คือ ประเภท ข้อมูลจำเพาะที่ผู้ใช้สามารถกำหนดได้โดยไม่ขึ้นกับประเภทข้อมูลที่มีอยู่ ในระบบจัดการฐานข้อมูล

ชนิดข้อมูลจำเพาะแม็พกับประเภทฐานข้อมูลที่มีอยู่แบบหนึ่งต่อหนึ่ง.

หลักการที่เกี่ยวข้อง

"User-defined distinct type" ในหน้า 237
user-defined distinct type (UDT) คือกลไก ที่ขยายความสามารถของ DB2 ให้มี ชนิดข้อมูลมากไปกว่าที่มีอยู่

SQL แพ็กเกจ

SQL แพ็กเกจ คืออ็อบเจ็กต์ที่มีโครงสร้างควบคุมซึ่งเกิดขึ้นเมื่อมีการเชื่อมโยงคำสั่ง SQL ในแอ็พพลิเคชันโปรแกรมเข้ากับระบบจัดการฐานข้อมูลเชิงสัมพันธ์แบบรีโมต (DBMS).

DBMS จะใช้โครงสร้างควบคุมเพื่อประมวลผลคำสั่ง SQL ที่พบขณะรันแอ็พพลิเคชันโปรแกรม.

SQL แพ็กเกจจะถูกสร้างขึ้นเมื่อระบุชื่อฐานข้อมูลเชิงสัมพันธ์ (พารามิเตอร์ RDB) บนคำสั่ง Create SQL (CRTSQL*xxx*) และเมื่อสร้างอ็อบเจ็กต์โปรแกรม. นอกจากนี้ยังสามารถสร้างแพ็กเกจได้ ด้วยคำสั่ง Create SQL Package (CRTSQLPKG)

หมายเหตุ: *xxx* ในคำสั่งนี้หมายถึง ตัวบ่งชี้ภาษาโฮสต์ ซึ่งได้แก่: CI สำหรับภาษา ILE C, CPPI สำหรับภาษา ILE C++, CBL สำหรับภาษา COBOL, CBLI สำหรับภาษา ILE COBOL, PLI สำหรับภาษา PL/I, RPG for RPG/400® และ RPGI สำหรับภาษา ILE RPG

SQL แพ็กเกจสามารถสร้างขึ้นได้โดยใช้คำสั่ง Process Extended Dynamic SQL (QSQPRCED) API ทั้งนี้ SQL แพ็กเกจที่กล่าวถึงในกลุ่มหัวข้อนี้จะหมายถึง แพ็กเกจ distributed program SQL เท่านั้น QSQPRCED API จะใช้ SQL แพ็กเกจ เพื่อให้การสนับสนุน extended dynamic SQL

สิ่งอ้างอิงที่เกี่ยวข้อง

"ฟังก์ชันของฐานข้อมูลเชิงสัมพันธ์แบบกระจายและ SQL" ในหน้า 302
ฐานข้อมูลเชิงสัมพันธ์แบบกระจาย ประกอบไปด้วยชุด SQL อ็อบเจ็กต์ที่กระจายอยู่บนระบบคอมพิวเตอร์ที่เชื่อมต่อถึงกันและกัน.

Process Extended Dynamic SQL (QSQPRCED) API

อ็อบเจ็กต์แอ็พพลิเคชันโปรแกรม

หลายๆ อ็อบเจ็กต์ถูกสร้างขึ้นเมื่อแอ็พพลิเคชันโปรแกรม DB2 for i5/OS ถูกพรีคอมไพล์

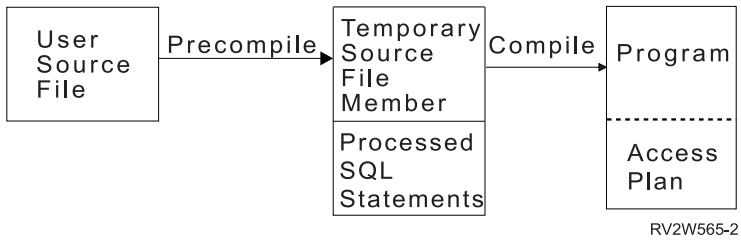
DB2 for i5/OS สนับสนุนพรีคอมไพเลอร์แบบไม่มี ILE และแบบ ILE. แอ็พพลิเคชันโปรแกรมอาจเป็นแบบกระจายหรือไม่กระจายก็ได้

ด้วยฐานข้อมูล DB2 for i5/OS คุณอาจต้องจัดการอ็อบเจ็กต์ต่อไปนี้:

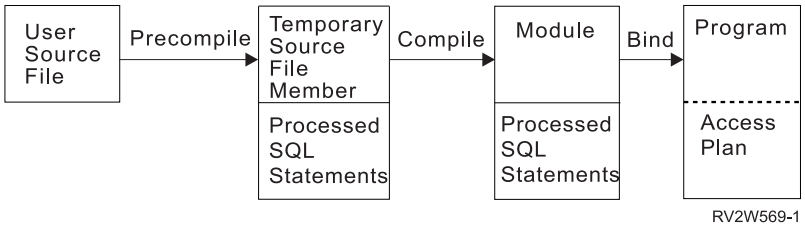
- ซอร์สต้นฉบับ
- หรือ, อ็อบเจ็กต์โมดูลสำหรับโปรแกรม ILE
- โปรแกรมหรือเซอร์วิสโปรแกรม
- SQL แพ็กเกจสำหรับโปรแกรมแบบกระจาย

ด้วยโปรแกรม DB2 for i5/OS แบบไม่มี ILE และแบบไม่กระจาย คุณต้องจัดการเฉพาะซอร์สต้นฉบับ และโปรแกรมผลลัพธ์ ภาพด้านล่างนี้แสดง อ็อบเจ็กต์ที่เกี่ยวข้องและขั้นตอนที่เกิดขึ้นระหว่างกระบวนการพรีคอมไพล์ และคอมไพล์สำหรับโปรแกรม DB2 for i5/OS ที่ไม่มี ILE แบบไม่กระจาย ไฟล์ต้นฉบับของผู้ใช้จะพรีคอมไพล์ต้นฉบับไปยังเมมเบอร์ไฟล์ต้น

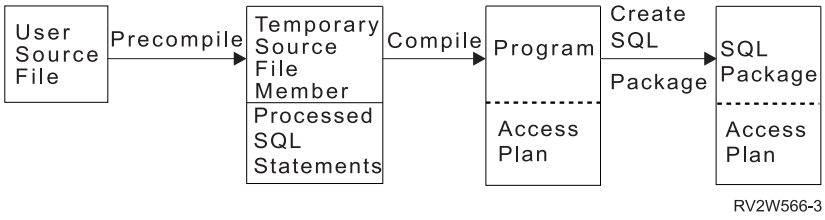
| ฉบับชั่วคราว หลังจากนั้นรายการย่อยนี้จะถูกคอมไพล์เข้าไปยังโปรแกรม.



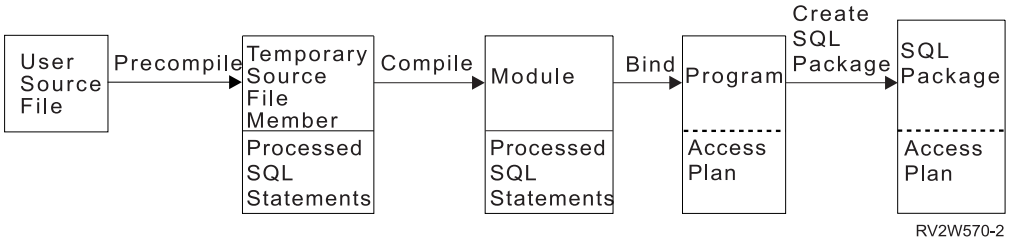
| ด้วยโปรแกรม DB2 for i5/OS ที่มี ILE แบบไม่กระจาย คุณอาจต้องจัดการซอร์สต้นฉบับ โมดูล และโปรแกรมผลลัพธ์ หรือเ
| ซอร์วิสโปรแกรม ภาพต่อไปนี้จะแสดงอ็อบเจ็กต์ที่เกี่ยวข้อง และขั้นตอนที่เกิดขึ้นระหว่างกระบวนการพรีคอมไพล์ และ
| คอมไพล์สำหรับโปรแกรม DB2 for i5/OS ที่มี ILE แบบไม่กระจาย เมื่อระบุ OBJTYPE(*PGM) ในคำสั่งพรีคอมไพล์ ไฟล์
| ต้นฉบับของผู้ใช้จะพรีคอมไพล์ต้นฉบับไปยังเมมเบอร์ไฟล์ต้นฉบับชั่วคราว หลังจากนั้นรายการย่อยนั้นจะถูกคอมไพล์ไปยัง
| โมดูลซึ่งเชื่อมกับโปรแกรม



| ด้วยโปรแกรม DB2 for i5/OS ที่ไม่มี ILE แบบกระจาย, คุณต้องจัดการซอร์สต้นฉบับ, โปรแกรมผลลัพธ์, และแพ็กเกจผล
| ลัพธ์. ภาพต่อไปนี้จะแสดงอ็อบเจ็กต์ที่เกี่ยวข้องและขั้นตอนที่เกิดขึ้นระหว่างกระบวนการพรีคอมไพล์และคอมไพล์
| สำหรับโปรแกรม DB2 for i5/OS ที่ไม่มี ILE แบบกระจาย ไฟล์ต้นฉบับของผู้ใช้จะพรีคอมไพล์ต้นฉบับไปยังเมมเบอร์ไฟล์ต้น
| ฉบับชั่วคราว หลังจากนั้นรายการย่อยนี้จะถูกคอมไพล์เข้าไปยังโปรแกรม. เมื่อสร้างโปรแกรมแล้ว SQL แพ็กเกจจะถูกสร้าง
| ขึ้นเพื่อรองรับโปรแกรม



| ด้วยโปรแกรม DB2 for i5/OS ที่มี ILE แบบกระจาย, คุณต้องจัดการซอร์สต้นฉบับ, อ็อบเจ็กต์โมดูล, โปรแกรมผลลัพธ์ หรือเ
| ซอร์วิสโปรแกรม, และแพ็กเกจผลลัพธ์. SQL แพ็กเกจสามารถสร้างขึ้นสำหรับโมดูลแบบกระจายแต่ละโมดูลในโปรแกรมที่มี
| ILE แบบกระจายหรือเซอร์วิสโปรแกรม. ภาพต่อไปนี้จะแสดงอ็อบเจ็กต์ที่เกี่ยวข้องและขั้นตอนที่เกิดขึ้นระหว่างกระบวนกา
| รพรีคอมไพล์และคอมไพล์สำหรับโปรแกรม DB2 for i5/OS ที่มี ILE แบบกระจาย ไฟล์ต้นฉบับของผู้ใช้จะพรีคอมไพล์ต้น
| ฉบับไปยังเมมเบอร์ไฟล์ต้นฉบับชั่วคราว หลังจากนั้นรายการย่อยนั้นจะถูกคอมไพล์ไปยังโมดูลซึ่งเชื่อมกับโปรแกรม เมื่อสร้าง
| โปรแกรมแล้ว SQL แพ็กเกจจะถูกสร้างขึ้นเพื่อรองรับโปรแกรม



หมายเหตุ: แผนการเข้าใช้งานที่เชื่อมโยงกับอ็อบเจ็กต์โปรแกรม DB2 for i5/OS แบบกระจายจะไม่ถูกสร้างขึ้นจนกว่าจะรันโปรแกรมในระบบ.

งานที่เกี่ยวข้อง

การเตรียมและรันโปรแกรมด้วยคำสั่ง SQL

ไฟล์ต้นฉบับของผู้ใช้

ไฟล์ต้นฉบับย่อยหรือไฟล์ stream ต้นฉบับจะมี ภาษาแอ็พพลิเคชันและคำสั่ง SQL คุณสามารถสร้าง และปรับปรุงรายการไฟล์ต้นฉบับย่อยโดยใช้ source entry utility (SEU), ซึ่งเป็นส่วนหนึ่งของไลเซนส์โปรแกรม IBM WebSphere® Development Studio for System i.

เอาต์พุตรายการไฟล์ต้นฉบับย่อย (Output source file member)

ตามค่าดีฟอลต์ กระบวนการพรีคอมไพล์จะสร้างไฟล์ต้นฉบับ ชั่วคราว QSQLT*xxxxx* ในไลบรารี QTEMP อย่างไรก็ตาม คุณสามารถระบุเอาต์พุตไฟล์ต้นฉบับให้เป็นชื่อไฟล์ถาวรได้บนคำสั่งพรีคอมไพล์

หากกระบวนการพรีคอมไพล์ใช้งานไลบรารี QTEMP ระบบจะลบไฟล์ออกอัตโนมัติเมื่องานเสร็จสมบูรณ์ รายการย่อยที่มีชื่อเดียวกันกับชื่อโปรแกรมจะถูกเพิ่มเข้าไปยัง เอาต์พุตไฟล์ต้นฉบับ. รายการย่อยนี้จะมีไอเท็มต่อไปนี้:

- การเรียกไปยังตัวสนับสนุนรันไทม์ SQL ซึ่งได้แทนที่คำสั่ง SQL ที่ฝังอยู่
- คำสั่ง SQL ที่ได้รับการวิเคราะห์คำและตรวจสอบซินแทกซ์

ตามค่าดีฟอลต์, พรีคอมไพเลอร์จะเรียกคอมไพเลอร์ภาษาโฮสต์.

งานที่เกี่ยวข้อง

การเตรียมและรันโปรแกรมด้วยคำสั่ง SQL

โปรแกรม

โปรแกรม คืออ็อบเจ็กต์ที่ถูกสร้างขึ้นจากกระบวนการคอมไพล์สำหรับการคอมไพล์แบบ non-ILE หรือกระบวนการเชื่อมโยงสำหรับคอมไพล์แบบ ILE

แผนการเข้าใช้งาน คือ ชุดโครงสร้างภายในและข้อมูลซึ่งบอก SQL ว่าจะรันคำสั่ง SQL แบบฝังอยู่ให้เกิดประสิทธิภาพสูงสุดได้อย่างไร แผนการจะถูกสร้างขึ้นเมื่อสร้างโปรแกรมสำเร็จแล้ว แผนการเข้าใช้งานจะไม่ถูกสร้างขึ้นมาระหว่างการสร้างโปรแกรมสำหรับคำสั่ง SQL หากคำสั่งอ้างถึงอ็อบเจ็กต์ เช่น ตาราง หรือมุมมอง ที่ไม่สามารถค้นพบได้ หรือที่คุณไม่มีสิทธิเข้าใช้งาน

แผนการเข้าใช้งานของคำสั่งดังกล่าวนั้นจะถูกสร้างขึ้นเมื่อรันโปรแกรม. หาก, ในเวลานั้น, ยังไม่พบตารางหรือมุมมองหรือคุณยังไม่มีสิทธิ์เข้าใช้งาน, SQLCODE ค่าลบจะถูกส่งคืนมา. แผนการเข้าใช้งานจะถูกเก็บไว้และรักษาไว้ในอ็อบเจ็กต์โปรแกรมสำหรับโปรแกรม SQL แบบไม่กระจาย และใน SQL แพ็กเกจสำหรับโปรแกรม SQL แบบกระจาย

SQL แพ็กเกจ

SQL แพ็กเกจจะมีแผนการเข้าใช้งานสำหรับโปรแกรม SQL แบบกระจาย.

SQL แพ็กเกจคือ อ็อบเจ็กต์ที่ถูกสร้างขึ้นเมื่อ:

- คุณสร้างโปรแกรม SQL แบบกระจาย โดยระบุ พารามิเตอร์ฐานข้อมูลเชิงสัมพันธ์ (RDB) บนคำสั่ง CREATE SQL (CRTSQL*xxx*)
- คุณรันคำสั่ง Create SQL Package (CRTSQLPKG)

เมื่อโปรแกรม SQL แบบกระจายถูกสร้างขึ้น, ชื่อของ SQL แพ็กเกจและ โทเค็นตรวจสอบความสอดคล้องกันภายในจะถูกบันทึกไว้ในโปรแกรม. และจะถูกใช้งานในช่วงรันไทม์เพื่อค้นหา SQL แพ็กเกจ และเพื่อตรวจสอบว่า SQL แพ็กเกจถูกต้องสำหรับโปรแกรมนี้หรือไม่ เนื่องจากชื่อของ SQL แพ็กเกจสำคัญมากสำหรับการรันโปรแกรม SQL แบบกระจาย, SQL แพ็กเกจจะต้องไม่มีการ:

- ย้าย
- เปลี่ยนชื่อ
- ทำซ้ำ
- เรียกคืนไปไว้ยังไลบรารีอื่น

โมดูล

โมดูล คืออ็อบเจ็กต์ชนิด Integrated Language Environment® (ILE) ที่คุณสร้างโดยการคอมไพล์ซอร์สโค้ดโดยใช้คำสั่ง Create Module (CRT*xxx*MOD) (หรือคำสั่งใดๆ ของ Create Bound Program (CRTBND*xxx*) โดยที่ *xxx* คือ C, CBL, CPP หรือ RPG)

คุณสามารถรันโมดูลได้ก็ต่อเมื่อคุณใช้คำสั่ง Create Program (CRTPGM) เพื่อเชื่อมโยงโมดูลนั้นเข้ากับโปรแกรม. ตามปกติคุณสามารถเชื่อมโยงโมดูลหลายตัวเข้าด้วยกันได้, แต่คุณอาจเชื่อมโยงโมดูลเข้ากับตัวมันเองก็ได้ด้วยเช่นกัน. โมดูลจะมีข้อมูลเกี่ยวกับคำสั่ง SQL อย่างไรก็ตาม แผนการเข้าใช้งาน SQL จะไม่ถูกสร้างขึ้นจนกว่าจะเชื่อมโยงโมดูลเข้ากับโปรแกรม หรือเซอร์วิสโปรแกรม.

สิ่งอ้างอิงที่เกี่ยวข้อง

คำสั่งการสร้างโปรแกรม (CRTPGM)

เซอร์วิสโปรแกรม

เซอร์วิสโปรแกรม คืออ็อบเจ็กต์ชนิด Integrated Language Environment (ILE) ซึ่งจัดเตรียมค่ากลุ่มของแพ็กเกจ ที่สนับสนุนคอลเลอร์รูทีนภายนอก (ฟังก์ชัน หรือโพรซีเดอร์) ลงใน อ็อบเจ็กต์แต่ละตัว

โปรแกรมที่เกี่ยวข้องและเซอร์วิสโปรแกรมอื่นๆ สามารถเข้าใช้งานรูทีนเหล่านี้ได้โดยการ resolve รายการอิมปอร์ตของรูทีนไปยังรายการเอ็กซปอร์ตของเซอร์วิสโปรแกรม. การเชื่อมต่อเข้ากับเซอร์วิสเหล่านี้จะเกิดขึ้นเมื่อมีการสร้างโปรแกรมเรียกทำงาน. วิธีการนี้จะปรับปรุงประสิทธิภาพในการเรียกใช้งานรูทีนเหล่านี้โดยไม่ต้องใส่โค้ดไว้ในโปรแกรมเรียกทำงาน.

Data definition language

Data definition language (DDL) คือส่วน ของ SQL ที่สร้าง เปลี่ยน และลบอ็อบเจ็กต์ฐานข้อมูล อ็อบเจ็กต์ฐานข้อมูล เหล่านี้ประกอบด้วยแบบแผน ตาราง มุมมอง ลำดับ แคตาล็อก ดรรชนี และ alias

หลักการที่เกี่ยวข้อง

"ประเภทของคำสั่ง SQL" ในหน้า 7

มีประเภทของคำสั่ง SQL พื้นฐานอยู่หลายประเภท ซึ่งคำสั่งเหล่านั้นจะถูกแสดงไว้ที่นี่ตามฟังก์ชัน

งานที่เกี่ยวข้อง

การเริ่มต้นด้วย SQL

การสร้างแบบแผน (schema)

แบบแผนจะมีการจัดกลุ่มของ SQL อ็อบเจ็กต์แบบโลจิคัล ในการสร้างแบบแผน ให้ใช้คำสั่ง CREATE SCHEMA

แบบแผนประกอบด้วยไลบรารี, เจอร์นัล, journal receiver, แคตาล็อก, และอาจรวมพจนานุกรมข้อมูล. สามารถสร้าง ย้าย หรือเรียกคืนตาราง มุมมอง และอ็อบเจ็กต์ระบบ (เช่น โปรแกรม) ไว้ในไลบรารีระบบใดๆ ก็ได้ ไฟล์ระบบทั้งหมดอาจถูกสร้างหรือย้ายไปยังแบบแผน SQL หากแบบแผน SQL ไม่มีพจนานุกรมข้อมูล. หากแบบแผน SQL มีพจนานุกรมข้อมูลแล้ว:

- source physical ไฟล์หรือ nonsource physical ไฟล์ที่มีต้นทางที่มีรายการย่อยเดียวสามารถถูกสร้าง, ย้าย, หรือ เก็บไว้ในแบบแผน SQL ได้.
- เราไม่สามารถวางโลจิคัลไฟล์ในแบบแผน SQL ได้เนื่องจากไฟล์เหล่านั้นไม่สามารถอธิบายอยู่ใน พจนานุกรมข้อมูลได้.

คุณสามารถสร้างและเป็นเจ้าของแบบแผนได้หลายรายการ.

คุณสามารถ สร้างแบบแผนโดยใช้คำสั่ง CREATE SCHEMA ตัวอย่างเช่น สร้างแบบแผน ที่ชื่อว่า DBTEMP:

```
CREATE SCHEMA DBTEMP
```

สิ่งอ้างอิงที่เกี่ยวข้อง

CREATE SCHEMA

การสร้างตาราง

ตารางสามารถแสดงการจัดเรียงข้อมูลเป็นสองมิติ ซึ่งประกอบด้วยแถวและคอลัมน์ ในการสร้างตาราง ให้ใช้คำสั่ง CREATE TABLE

แถวคือส่วนที่เป็นแนวนอนซึ่งประกอบด้วยคอลัมน์ตั้งแต่หนึ่งคอลัมน์ขึ้นไป. คอลัมน์คือส่วนที่เป็นแนวดิ่งซึ่งประกอบด้วยแถวตั้งแต่หนึ่งแถวขึ้นไปในประเภทข้อมูลหนึ่งประเภท. ข้อมูลทั้งหมดในหนึ่งคอลัมน์ต้องเป็นข้อมูลประเภทเดียวกัน. ตารางใน SQL คือฟิสิคัลไฟล์ที่มีคีย์ หรือไม่มีคีย์ก็ได้.

คุณสามารถสร้างตาราง โดยใช้คำสั่ง CREATE TABLE definition ต้องมีชื่อ definition และชื่อและแอ็ตทริบิวต์ของคอลัมน์ด้วย. definition อาจรวมถึงแอ็ตทริบิวต์ อื่นๆ ของตาราง เช่น คีย์หลัก

ตัวอย่าง: สมมติว่าคุณมีสิทธิระดับผู้ดูแลระบบให้สร้างตารางชื่อ 'INVENTORY' โดยมีคอลัมน์ต่อไปนี้:

- หมายเลขชิ้นส่วน: จำนวนเต็มระหว่าง 1 ถึง 9999, และต้องไม่เป็นค่า null

- รายละเอียด: อักขระต้องมีความยาวตั้งแต่ 0 ถึง 24
- จำนวนที่มีอยู่: จำนวนเต็มตั้งแต่ 0 ถึง 100000

คีย์หลักคือ PARTNO.

```
CREATE TABLE INVENTORY
                (PARTNO         SMALLINT     NOT NULL,
                 DESCR          VARCHAR(24 ),
                 QONHAND        INT,
                 PRIMARY KEY(PARTNO))
```

หลักการที่เกี่ยวข้อง

ชนิดข้อมูล

การเพิ่มและลบข้อจำกัด

สามารถเพิ่มข้อจำกัดเข้าในตารางใหม่ หรือ ตารางที่มีอยู่เดิมได้ เมื่อต้องการเพิ่มคีย์หลักหรือคีย์เฉพาะ, ข้อจำกัดแบบอ้างอิง หรือข้อจำกัดเพื่อการตรวจสอบ ให้ใช้คำสั่ง CREATE TABLE หรือ ALTER TABLE เมื่อต้องการลบข้อจำกัด ให้ใช้คำสั่ง ALTER TABLE

ตัวอย่างเช่น เพิ่มคีย์หลักลงในตารางที่มีอยู่เดิม โดยใช้คำสั่ง ALTER TABLE:

```
ALTER TABLE CORPDATA.DEPARTMENT
  ADD PRIMARY KEY (DEPTNO)
```

หากต้องการให้คีย์นี้เป็นคีย์แบบเฉพาะ, ให้ใส่แทนที่คีย์เวิร์ด PRIMARY ด้วย UNIQUE.

คุณสามารถลบข้อจำกัดออกโดยใช้คำสั่ง ALTER TABLE แบบเดียวกัน:

```
ALTER TABLE CORPDATA.DEPARTMENT
  DROP PRIMARY KEY (DEPTNO)
```

Referential integrity และตาราง

Referential integrity คือเงื่อนไข สำหรับชุดของตารางในฐานข้อมูลซึ่งใช้ในการอ้างอิงทั้งหมดจากตารางหนึ่งไปยัง อีกตารางหนึ่ง

พิจารณาตัวอย่างนี้:

- CORPDATA.EMPLOYEE ใช้เป็นรายการหลักของพนักงานทั้งหมด.
- CORPDATA.DEPARTMENT ทำหน้าที่เป็นเสมือนรายการหลักของจำนวนแผนกทั้งหมดที่ถูกต้อง.
- CORPDATA.EMP_ACT ให้รายการหลักของกิจกรรมที่ดำเนินการในโครงการต่างๆ.

ตารางอื่นๆ จะอ้างอิงถึง entity เดียวกับที่อธิบายไว้ในตารางเหล่านี้. เมื่อตารางประกอบด้วยข้อมูลที่มีรายการหลัก, ข้อมูลดังกล่าวควรปรากฏในรายการหลักนั้น, ไม่เช่นนั้นแสดงว่าการอ้างอิงไม่ถูกต้อง. ตารางซึ่งประกอบด้วยรายการหลักคือ *ตาราง parent*, และตารางที่อ้างอิงถึงตาราง parent คือ *ตาราง dependent* . เมื่อการอ้างอิงจาก ตาราง dependent ไปยังตาราง parent ถูกต้อง, สภาวะของชุดตารางจะถูกเรียกว่า *ความสมบูรณ์ในการอ้างอิง*.

หรือจะกล่าวอีกนัยหนึ่ง, ความสมบูรณ์ในการอ้างอิงคือสภาวะของฐานข้อมูลโดยที่ค่าของ foreign key ทั้งหมดถูกต้อง. ค่าแต่ละค่าของ foreign key จะต้องมีอยู่ใน parent key เช่นกันหรือมีค่าเป็น null. Definition ของความสมบูรณ์ในการอ้างอิงนี้จะต้องมีความเข้าใจในคำศัพท์ต่อไปนี้:

- *unique key* คือคอลัมน์หรือชุดคอลัมน์ในตารางซึ่งระบุแถวเป็นการเฉพาะ แม้ว่าในหนึ่งตารางสามารถมี unique key ได้เป็นจำนวนมาก, แต่แถวสองแถวในหนึ่งตารางจะต้องไม่มีค่า unique key ค่าเดียวกัน.
- *primary key*คือ unique key ซึ่งต้องไม่มีค่าเป็น null. ในหนึ่งตารางจะต้องมี primary key เดียวเท่านั้น.
- *parent key* คือ unique key หรือ primary key ซึ่ง ถูกอ้างอิงในข้อจำกัดการอ้างอิง
- *foreign key* คือ คอลัมน์หรือชุดคอลัมน์ซึ่งค่าต้องตรงกับค่าของ parent key. หากค่าคอลัมน์ใดๆ ที่ใช้ในการ สร้าง foreign key เท่ากับ null กฎดังกล่าวก็จะใช้ไม่ได้
- *parent table* คือ ตารางซึ่งประกอบด้วย parent key.
- *dependent table* คือ ตารางซึ่งประกอบด้วย foreign key.
- *descendent table* คือ ตาราง dependent หรือตารางที่อยู่ในลำดับถัดจากตาราง dependent.

การบังคับใช้ความสมบูรณ์ในการอ้างอิงเป็นการละเมิดกฎที่ระบุว่า ทุกๆ foreign key ที่ไม่ใช่ null ต้องมี parent key ที่ตรงกัน.

SQL สนับสนุนแนวคิดเรื่องความสมบูรณ์ในการอ้างอิงด้วยคำสั่ง CREATE TABLE และ ALTER TABLE.

สิ่งอ้างอิงที่เกี่ยวข้อง

"DB2 for i5/OS ตารางตัวอย่าง" ในหน้า 329
ตารางตัวอย่างต่อไปนี้ถูกอ้างอิงและใช้งานใน หัวข้อการโปรแกรม SQL และการอ้างอิง SQL

CREATE TABLE

ALTER TABLE

การเพิ่มและลบข้อจำกัดในการอ้างอิง:

คุณสามารถใช้คำสั่ง CREATE TABLE หรือ คำสั่ง ALTER TABLE เพื่อเพิ่มข้อจำกัดในการอ้างอิง เมื่อต้องการลบข้อจำกัดในการอ้างอิง ให้ใช้คำสั่ง ALTER TABLE

ข้อจำกัด คือ กฎที่รับรองว่าการอ้างอิงจากตารางหนึ่ง, หรือตาราง dependent, ไปยังข้อมูลในอีกตารางหนึ่ง, หรือตาราง parent, ถูกต้อง. คุณใช้ข้อจำกัดในการอ้างอิงเพื่อตรวจสอบความสมบูรณ์ในการอ้างอิง.

ด้วยข้อจำกัดในการอ้างอิง, ค่าที่ไม่ใช่ null ของ foreign key จะใช้ได้ก็ต่อเมื่อค่าเหล่านั้น ปรากฏขึ้นเป็นเสมือนค่าของ parent key. เมื่อคุณกำหนดข้อจำกัดในการอ้างอิง, ให้คุณระบุ:

- primary หรือ unique key
- foreign key
- ลบและอัพเดตกฎที่ระบุการดำเนินงานที่เกิดขึ้นซึ่งเกี่ยวข้องกับแถว dependent เมื่อแถว parent ถูกลบออก หรือมีการอัพเดต.

หรือ, คุณสามารถระบุชื่อให้กับข้อจำกัดได้. หากไม่มีการระบุชื่อ, ระบบจะสร้างชื่อให้โดยอัตโนมัติ.

หลังจากมีการกำหนดข้อจำกัดในการอ้างอิงแล้ว ระบบจะบังคับให้ใช้ข้อจำกัดดังกล่าวกับทุกๆ การปฏิบัติคำสั่ง INSERT, DELETE, และ UPDATE โดยกระทำผ่าน SQL หรืออินเตอร์เฟสอื่นๆ ซึ่งรวมถึง System i Navigator, คำสั่ง CL, ยูทิลิตี หรือข้อความภาษาชั้นสูง

สิ่งอ้างอิงที่เกี่ยวข้อง

CREATE TABLE

ตัวอย่าง: การเพิ่มข้อจำกัดแบบอ้างอิง:

คุณกำหนดข้อจำกัดแบบอ้างอิงที่ว่า ทุกๆ หมายเลขแผนกที่แสดงไว้ในตารางตัวอย่างพนักงานจะต้องปรากฏในตารางแผนกนั้น ข้อจำกัดแบบอ้างอิงนี้เป็นการรับรองว่าพนักงานทุกคนอยู่ในแผนกที่มีอยู่

คำสั่ง SQL ต่อไปนี้เป็นการสร้างตาราง CORPDATA.DEPARTMENT และตาราง CORPDATA.EMPLOYEE ซึ่งมีการกำหนดความสัมพันธ์ของข้อจำกัดเหล่านั้น.

```
CREATE TABLE CORPDATA.DEPARTMENT
      (DEPTNO    CHAR(3)     NOT NULL PRIMARY KEY,
       DEPTNAME  VARCHAR(29) NOT NULL,
       MGRNO     CHAR(6),
       ADMRDEPT  CHAR(3)     NOT NULL
                             CONSTRAINT REPORTS_TO_EXISTS
                               REFERENCES CORPDATA.DEPARTMENT (DEPTNO)
                               ON DELETE CASCADE)

CREATE TABLE CORPDATA.EMPLOYEE
      (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
       FIRSTNME    VARCHAR(12)     NOT NULL,
       MIDINIT     CHAR(1)         NOT NULL,
       LASTNAME    VARCHAR(15)     NOT NULL,
       WORKDEPT  CHAR(3)     CONSTRAINT WORKDEPT_EXISTS
                               REFERENCES CORPDATA.DEPARTMENT (DEPTNO)
                               ON DELETE SET NULL ON UPDATE RESTRICT,
       PHONENO   CHAR(4),
       HIREDATE  DATE,
       JOB       CHAR(8),
       EDLEVEL   SMALLINT    NOT NULL,
       SEX       CHAR(1),
       BIRTHDATE DATE,
       SALARY    DECIMAL(9,2),
       BONUS     DECIMAL(9,2),
       COMM      DECIMAL(9,2),
       CONSTRAINT  UNIQUE_LNAME_IN_DEPT UNIQUE (WORKDEPT, LASTNAME))
```

ในกรณีนี้, ตาราง DEPARTMENT มีคอลัมน์ของจำนวนแผนกเฉพาะ (DEPTNO) ซึ่งฟังก์ชันเป็น primary key, และเป็นตาราง parent ในความสัมพันธ์สองข้อของข้อจำกัด:

REPORTS_TO_EXISTS

คือ ข้อจำกัดในการอ้างอิงด้วยตนเองโดยที่ตาราง DEPARTMENT เป็นทั้ง parent และ dependent ในความสัมพันธ์เดียวกัน. ทุกๆ ค่าของ ADMRDEPT ที่ไม่เป็นค่า null ต้องตรงกับค่าของ DEPTNO. แผนกต้องรายงานไปยังแผนกที่มีอยู่ในฐานข้อมูล. กฎ DELETE CASCADE แสดงว่าหากแถวที่มีค่า DEPTNO *n* ถูกลบออก, ทุกๆ แถวในตารางที่ ADMRDEPT เท่ากับ *n* ต้องถูกลบออกเช่นกัน.

WORKDEPT_EXISTS

สร้างตาราง EMPLOYEE เป็นตาราง dependent, และคอลัมน์การกำหนดแผนกพนักงาน (WORKDEPT) เป็น foreign key. ดังนั้น, ทุกๆ ค่าของ WORKDEPT ต้องตรงกับค่าของ DEPTNO. The กฎ DELETE SET NULL กล่าวว่า หากแถวถูกลบออกจาก DEPARTMENT โดยที่ค่าของ DEPTNO เท่ากับ *n*, ค่าของ WORKDEPT ใน

EMPLOYEE จะถูกตั้งค่าให้เป็น null ในทุกๆ แถวที่มีค่าเป็น *n*. กฎ UPDATE RESTRICT กล่าวว่า ค่าของ DEPTNO ใน DEPARTMENT ไม่สามารถอัพเดตได้ หากมีค่าของ WORKDEPT ใน EMPLOYEE ที่ตรงกับค่าปัจจุบันของ DEPTNO.

ข้อจำกัด UNIQUE_LNAME_IN_DEPT ในตาราง EMPLOYEE เป็นสาเหตุทำให้ LASTNAME ที่อยู่ภายใน department เป็น unique หรือต้องไม่ซ้ำกัน. ขณะที่ข้อจำกัดนี้ไม่เป็นเช่นนั้น, ข้อจำกัดนั้นจะอธิบายวิธีการที่ข้อจำกัดซึ่งสร้างคอลัมน์จำนวนมากถูกกำหนดที่ระดับตาราง.

ตัวอย่าง: การลบข้อจำกัด

เมื่อคุณลบ primary key บนคอลัมน์ DEPTNO ในตาราง DEPARTMENT ตารางอื่นๆ ก็จะได้รับผลกระทบ

นอกจากนี้คุณยังลบข้อจำกัด REPORTS_TO_EXISTS ที่ ถูกกำหนดบนตาราง DEPARTMENT และข้อจำกัด WORKDEPT_EXISTS ที่ ถูกกำหนดบนตาราง EMPLOYEE เนื่องจาก primary key ที่คุณลบออก เป็น parent key ในความสัมพันธ์ของข้อจำกัดนั้น

```
ALTER TABLE CORPDATA.EMPLOYEE DROP PRIMARY KEY
```

คุณยังสามารถลบข้อจำกัดตามชื่อ, ดังตัวอย่างนี้:

```
ALTER TABLE CORPDATA.DEPARTMENT
            DROP CONSTRAINT UNIQUE_LNAME_IN_DEPT
```

การระงับการตรวจสอบ

ข้อจำกัดในการอ้างอิงและข้อจำกัดในการตรวจสอบสามารถอยู่ในสถานะการระงับการตรวจสอบ โดยที่มีความเป็นไปได้ที่จะมีการละเมิดข้อจำกัดอยู่

ในส่วนข้อจำกัดในการอ้างอิง, การละเมิดเกิดขึ้นได้เมื่อมีความไม่ตรงกันที่อาจเกิดขึ้นระหว่าง parent key และ foreign key. ในส่วนข้อจำกัด ในการตรวจสอบ การละเมิดเกิดขึ้นได้เมื่อค่าที่อาจเกิดขึ้นอยู่ในคอลัมน์ซึ่งถูกจำกัดโดยข้อจำกัดในการตรวจสอบ เมื่อระบบตัดสินแล้วว่าอาจมีการละเมิด ข้อจำกัด (อย่างเช่นหลังการดำเนินการกู้คืน) ข้อจำกัดนั้น จะถูกทำเครื่องหมายว่าเป็นระงับการตรวจสอบ เมื่อเกิดกรณีเช่นนี้ขึ้น, จะมีการใช้ข้อบังคับในการใช้ตารางที่เกี่ยวข้องกับข้อจำกัดดังกล่าว. ในส่วนของข้อจำกัดในการอ้างอิง, มีการใช้ข้อบังคับต่อไปนี้:

- ไม่อนุญาตให้อินพุตหรือเอาต์พุตไฟล์ dependent.
- อนุญาตเฉพาะการอ่านและแทรกบนไฟล์ parent.

เมื่อข้อจำกัดในการตรวจสอบอยู่ในสถานะการระงับการตรวจสอบ, จะใช้ข้อบังคับต่อไปนี้:

- ไม่อนุญาตให้อ่านไฟล์.
- อนุญาตให้มีการแทรกและอัพเดตและบังคับใช้ข้อจำกัด

เพื่อลบข้อจำกัดออกจากการระงับการตรวจสอบ ให้ปฏิบัติตามขั้นตอนต่อไปนี้:

1. ยกเลิกความสัมพันธ์ด้วยคำสั่ง CL Change Physical File Constraint (CHGPFCST).
2. แก้ไขข้อมูลคีย์ (foreign, parent, หรือทั้งสอง) ของข้อจำกัดในการอ้างอิงหรือข้อมูลคอลัมน์ สำหรับข้อจำกัดในการตรวจสอบ.
3. ใช้งานข้อจำกัดอีกครั้งด้วยคำสั่ง CL CHGPFCST.

คุณสามารถระบุแถวที่ละเมิดข้อจำกัดด้วยคำสั่ง CL Display Check Pending Constraint (DSPCPCST).

หลักการที่เกี่ยวข้อง

สถานะรอการตรวจสอบในข้อจำกัดในการอ้างอิง

งานที่เกี่ยวข้อง

การทำงานกับข้อจำกัดที่อยู่ในสถานะรอการตรวจสอบ

การสร้างตารางโดยใช้ LIKE

คุณสามารถสร้างตารางที่เหมือนตารางอื่น. นั่นคือ, คุณสามารถสร้างตารางที่แทรก definition ของคอลัมน์ทั้งหมดจากตารางที่มีอยู่.

definition ต่อไปนี้จะถูกก็อปปี:

- ชื่อคอลัมน์ (และชื่อคอลัมน์ระบบ)
- ประเภทข้อมูล, ความยาว, ความแม่นยำ และมาตราส่วน
- CCSID

หาก LIKE clause อยู่ตามหลังชื่อตาราง ในทันทีและไม่ได้ปิดท้ายด้วยวงเล็บ แอ็ตทริบิวต์ต่อไปนี้จะถูกแทรกเข้าไว้:

- ข้อความของคอลัมน์ (LABEL ON)
- หัวคอลัมน์ (LABEL ON)
- ค่าดีฟอลต์
- แอ็ตทริบิวต์ที่ซ่อน
- แอ็ตทริบิวต์ Identity
- ความเป็นศูนย์
- แอ็ตทริบิวต์ Row change timestamp

ถ้าตารางหรือมุมมองที่กำหนดไว้ ประกอบด้วย identity column คุณต้องระบุอ็อพชัน INCLUDING IDENTITY บน คำสั่ง CREATE TABLE ถ้าหากคุณต้องการให้มี identity column เกิดขึ้นในตารางใหม่ การทำงานซึ่งเป็นค่าดีฟอลต์ของ CREATE TABLE คือ EXCLUDING IDENTITY. มีอ็อพชันที่คล้ายคลึงกันซึ่งประกอบด้วยค่าดีฟอลต์, แอ็ตทริบิวต์ที่ซ่อน และ แอ็ตทริบิวต์ row change timestamp หากตารางที่ระบุหรือมุมมองคือไฟล์แบบฟิสิคัลที่สร้างขึ้นแบบไม่มี SQL หรือไฟล์แบบลอจิคัล, แอ็ตทริบิวต์แบบไม่มี SQL จะถูกลบออก

สร้างตาราง EMPLOYEE2 ที่รวมคอลัมน์ทั้งหมดไว้ใน EMPLOYEE:

```
CREATE TABLE EMPLOYEE2 LIKE EMPLOYEE
```

สิ่งอ้างอิงที่เกี่ยวข้อง

CREATE TABLE

การสร้างตารางโดยใช้ AS

คุณสามารถสร้างตารางจากผลลัพธ์ของคำสั่ง SELECT ในการสร้างตารางชนิดนี้ ให้ใช้คำสั่ง CREATE TABLE AS

สามารถใช้งานนิพจน์ทั้งหมดซึ่งสามารถใช้ในคำสั่ง SELECT ในคำสั่ง CREATE TABLE AS ได้. คุณสามารถแทรกข้อมูลทั้งหมดจากตารางหรือตารางที่คุณเลือกจากได้.

ตัวอย่างเช่น, สร้างตารางที่ชื่อ EMPLOYEE3 ซึ่งรวมเอา definition คอลัมน์ทั้งหมดจาก EMPLOYEE ที่ซึ่ง DEPTNO = D11.

```
CREATE TABLE EMPLOYEE3 AS
   (SELECT PROJNO, PROJNAME, DEPTNO
    FROM EMPLOYEE
    WHERE DEPTNO = 'D11') WITH NO DATA
```

ถ้าตารางหรือมุมมองที่กำหนดไว้ ประกอบด้วย identity column คุณต้องระบุอ็อพชัน INCLUDING IDENTITY บน คำสั่ง CREATE TABLE ถ้าหากคุณต้องการให้มี identity column เกิดขึ้นในตารางใหม่ การทำงานซึ่งเป็นค่าดีฟอลต์ของ CREATE TABLE คือ EXCLUDING IDENTITY. มีอ็อพชันที่คล้ายคลึงกันซึ่งประกอบด้วยค่าดีฟอลต์, แอ็ตทริบิวต์ที่ซ่อน และ แอ็ตทริบิวต์ row change timestamp ประโยค WITH NO DATA ชี้ให้เห็นว่า definition ของคอลัมน์ ได้ถูกก็อปปีไปโดยไม่มีข้อมูล. ถ้าคุณต้องการใส่เพิ่มข้อมูล ลงในตารางใหม่ EMPLOYEE3 ให้ใส่ WITH DATA clause หากเคียวรี่ที่ระบุไว้มีฟิสิคัลไฟล์ที่สร้างขึ้นแบบไม่มี SQL หรือโลจิคัลไฟล์ แอ็ตทริบิวต์ที่เป็นผลลัพธ์ซึ่งไม่มี SQL จะถูกลบออก

หลักการที่เกี่ยวข้อง

"การดึงข้อมูลโดยใช้คำสั่ง SELECT" ในหน้า 44
คำสั่ง SELECT ปรับแต่งเคียวรี่ของคุณเพื่อรวบรวม ข้อมูล คุณสามารถใช้คำสั่ง SELECT เพื่อดึงแถวที่เฉพาะเจาะจงหรือดึง ข้อมูลในลักษณะที่เฉพาะเจาะจง

สิ่งอ้างอิงที่เกี่ยวข้อง

CREATE TABLE

การสร้างและเปลี่ยนตาราง materialized query

ตาราง materialized query เป็น ตารางที่มี definition อยู่บนพื้นฐานของผลลัพธ์ของเคียวรี่ และข้อมูลอยู่ในรูปแบบ ของผลลัพธ์ที่คำนวณไว้ล่วงหน้า ซึ่งรับมาจากตารางที่ definition ของตาราง materialized query ใช้อ้างอิง

ถ้า optimizer พบว่าเคียวรี่ที่ ใช้ตาราง materialized query รันได้เร็วกว่าตารางพื้นฐาน เคียวรี่ก็จะรันโดยใช้ตาราง materialized query คุณสามารถทำเคียวรี่ได้โดยตรงกับตาราง materialized query. สำหรับข้อมูลเพิ่มเติมเกี่ยวกับวิธีที่ optimizer ใช้ตาราง materialized query โปรดดูหัวข้อ ประสิทธิภาพ ในการทำงานของฐานข้อมูลและการ optimization เคียวรี่

สมมุติว่ามีตาราง transaction ขนาดใหญ่มากชื่อ TRANS ประกอบด้วย transaction ในแต่ละแถว ที่ประมวลผลต่อหนึ่งบริษัท. ตารางถูกกำหนดให้มีหลายคอลัมน์. ให้สร้างตาราง materialized query สำหรับตาราง TRANS ที่ประกอบด้วยข้อมูลสรุปรายวันสำหรับวันที่ และจำนวนของ transaction โดยใช้คำสั่งต่อไปนี้:

```
CREATE TABLE STRANS
  AS (SELECT YEAR AS SYEAR, MONTH AS SMONTH, DAY AS SDAY, SUM(AMOUNT) AS SSUM
      FROM TRANS
      GROUP BY YEAR, MONTH, DAY )
  DATA INITIALLY DEFERRED
  REFRESH DEFERRED
  MAINTAINED BY USER
```

ตาราง materialized query นี้ระบุว่าตารางนี้ไม่ได้มีอยู่ ณ เวลาที่มันถูกสร้างขึ้นมาโดย โดยการใช้ประโยค DATA INITIALLY DEFERRED. REFRESH DEFERRED ชี้ให้เห็นว่าการเปลี่ยนแปลงที่เกิดขึ้นกับ TRANS ไม่มีผลกระทบใน STRANS. นอกจากนี้ ตารางนี้ได้รับการดูแลจากผู้ใช้ ทำให้ ผู้ใช้สามารถใช้คำสั่ง ALTER, INSERT, DELETE และ UPDATE ได้

เพื่อที่จะให้ตาราง materialized query คงอยู่ หรือรีเฟรช ตารางนั้น หลังจากที่เกิดขึ้นแล้ว ให้ใช้คำสั่ง REFRESH TABLE ซึ่งจะทำให้ เคียวรี่ที่เชื่อมโยงกับตาราง materialized query ทำงาน และ ทำให้ผลลัพธ์ของเคียวรี่ถูกใส่ลงในตาราง เพื่อให้ตาราง STRANS คงสภาพอยู่ตลอด ให้รันคำสั่งต่อไปนี้:

```
REFRESH TABLE STRANS
```

คุณสามารถสร้างตาราง materialized query จากตารางฐานที่เกิดขึ้นแล้ว ตราบเท่าที่ ผลลัพธ์ของ select-statement ได้เตรียมกลุ่มของคอลัมน์ที่ตรงกับคอลัมน์ ในตารางที่เกิดขึ้นก่อนแล้ว (จำนวนคอลัมน์เท่ากัน และ definitions ของคอลัมน์เข้ากันได้). ตัวอย่างเช่น, ให้สร้างตาราง TRANSCOUNT. แล้ว, เปลี่ยน ตารางฐาน TRANSCOUNT ไปเป็นตาราง materialized query:

การสร้างตาราง:

```
CREATE TABLE TRANSCOUNT
    (ACCTID SMALLINT NOT NULL,
     LOCID SMALLINT,
     YEAR DATE
     CNT INTEGER)
```

คุณสามารถเปลี่ยนตารางนี้ไปเป็นตาราง materialized query:

```
ALTER TABLE TRANSCOUNT
  ADD MATERIALIZED QUERY
      (SELECT  ACCTID, LOCID, YEAR, COUNT(*) AS CNT
      FROM TRANS
      GROUP BY ACCTID, LOCID, YEAR )
  DATA INITIALLY DEFERRED
  REFRESH DEFERRED
  MAINTAINED BY USER
```

ท้ายที่สุด, คุณก็ยังสามารถเปลี่ยนตาราง materialized query กลับไปเป็นตารางฐานเหมือนเดิมได้. ตัวอย่างเช่น:

```
ALTER TABLE TRANSCOUNT
      DROP MATERIALIZED QUERY
```

ในตัวอย่างนี้, ตาราง TRANSCOUNT ไม่ได้ถูกเอาออกไป, แต่มันไม่เได้เป็นตาราง materialized query อีกต่อไป.

หลักการที่เกี่ยวข้อง

"ตาราง, แถว, และคอลัมน์" ในหน้า 11
ตาราง เป็นการจัดการข้อมูลแบบสองด้าน ซึ่งประกอบด้วย *แถว* และ *คอลัมน์*

การประกาศตารางชั่วคราวแบบโกลบอล

คุณสามารถสร้างตารางชั่วคราวเพื่อใช้งานกับเซสชัน ปัจจุบันได้ เมื่อต้องการสร้างตารางชั่วคราว ให้ใช้คำสั่ง DECLARE GLOBAL TEMPORARY TABLE

ตารางชั่วคราวจะไม่ปรากฏขึ้นในแค็ตตาล็อกระบบและไม่สามารถใช้งานร่วมกับเซสชันอื่นๆ ได้. เมื่อคุณสิ้นสุดเซสชัน, แถวของตารางจะถูกลบทิ้ง และตารางจะเลื่อนลงมา.

ไวยากรณ์ของคำสั่งนี้คล้ายกับ คำสั่ง CREATE TABLE และสามารถมี LIKE หรือ AS clause ได้

ตัวอย่างเช่น, สร้างตารางชั่วคราว ORDERS:

```
DECLARE GLOBAL TEMPORARY TABLE ORDERS
              (PARTNO   SMALLINT NOT NULL,
               DESCR    VARCHAR(24),
               QONHAND  INT)
       ON COMMIT DELETE ROWS
```

ตารางนี้จะถูกสร้างขึ้นใน QTEMP. หากต้องการอ้างอิงถึงตารางที่ใช้ชื่อแบบแผน,ให้ใช้ SESSION หรือ QTEMP. คุณส มารถใช้คำสั่ง SELECT, INSERT, UPDATE, และ DELETE กับตารางนี้, เช่นเดียวกับตารางอื่นๆทั่วไป. คุณสามารถเลื่อนตารางนี้ได้โดยใช้คำสั่ง DROP TABLE:

```
DROP TABLE ORDERS
```

สิ่งอ้างอิงที่เกี่ยวข้อง

DECLARE GLOBAL TEMPORARY TABLE

การสร้างคอลัมน์ row change timestamp

แต่ละครั้งที่แถวถูกใส่เพิ่มหรือเปลี่ยนแปลงในตารางที่มีคอลัมน์ row change timestamp ค่าในคอลัมน์ row change timestamp จะถูกตั้งเป็น timestamp ที่สัมพันธ์กับเวลาที่ทำการแทรกหรืออัพเดต

ชนิดข้อมูลของคอลัมน์ row change timestamp จะต้องเป็น TIMESTAMP คุณสามารถกำหนดคอลัมน์ row change timestamp ได้เพียงหนึ่งคอลัมน์ในตาราง

เมื่อคุณสร้างตาราง, คุณสามารถกำหนดคอลัมน์ในตารางให้เป็น คอลัมน์ row change timestamp ได้ ตัวอย่างเช่น ให้สร้างตาราง ORDERS ที่มี คอลัมน์ที่มีชื่อว่า ORDERNO, SHIPPED_TO, ORDER_DATE, STATUS และ CHANGE_TS กำหนด CHANGE_TS ให้เป็นคอลัมน์ row change timestamp

```
CREATE TABLE ORDERS
   (ORDERNO SMALLINT,
    SHIPPED_TO VARCHAR (36) ,
    ORDER_DATE DATE,
    STATUS CHAR(1),
    CHANGE_TS TIMESTAMP FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP)
```

เมื่อ แถวถูกแทรกไว้ในตาราง ORDERS คอลัมน์ CHANGE_TS สำหรับแถว จะถูกตั้งเป็น timestamp ของการแทรก เมื่อใดก็ตามที่แถวใน ORDERS ถูกอัพเดต คอลัมน์ CHANGE_TS สำหรับแถวนั้นจะถูกแก้ไข เพื่อแสดง timestamp ของการอัพเดต

คุณสามารถเลื่อนแอ็ตทริบิวต์ row change timestamp จากคอลัมน์ได้:

```
ALTER TABLE ORDER
  ALTER COLUMN CHANGE_TS
  DROP ROW CHANGE TIMESTAMP
```

คอลัมน์ CHANGE_TS ยังคง เป็นคอลัมน์ TIMESTAMP ในตาราง แต่ระบบไม่อัพเดต ค่า timestamp สำหรับคอลัมน์นี้โดยอัตโนมัติอีกต่อไป

การสร้างและเปลี่ยน identity column

ทุกครั้งที่เพิ่มแถวใหม่เข้าไปยังตารางด้วย identity column ระบบจะสร้างค่าของ identity column สำหรับแถวใหม่นี้

เฉพาะคอลัมน์ของประเภท SMALLINT, INTEGER, BIGINT, DECIMAL, หรือ NUMERIC ที่สามารถถูกสร้างเป็น identity column ได้. คุณมีสิทธิ์สร้าง identity column ได้หนึ่งคอลัมน์ต่อตาราง. เมื่อคุณเปลี่ยน definition ของตาราง, สามารถระบุเฉพาะคอลัมน์ที่คุณจะเพิ่มเป็น identity column ได้; ไม่สามารถระบุคอลัมน์ที่มีอยู่เดิมได้.

เมื่อคุณสร้างตาราง, คุณสามารถกำหนดคอลัมน์ในตารางให้เป็น identity column ได้. ตัวอย่างเช่น ให้สร้างตาราง ORDERS โดยมีสามคอลัมน์ ที่มีชื่อว่า ORDERNO, SHIPPED_TO และ ORDER_DATE กำหนด ORDERNO ให้เป็น identity column.

```
CREATE TABLE ORDERS
   (ORDERNO SMALLINT NOT NULL
   GENERATED ALWAYS AS IDENTITY
   (START WITH 500
   INCREMENT BY 1
   CYCLE),
   SHIPPED_TO VARCHAR (36) ,
   ORDER_DATE DATE)
```

คอลัมน์นี้จะถูกกำหนดด้วยค่าเริ่มต้นของ 500 โดยเพิ่มขึ้นทีละหนึ่ง 1 เมื่อแทรกแถวใหม่ และจะหมุนเวียนกลับมาใช้ใหม่เมื่อถึงค่าสูงสุด ในตัวอย่างนี้, ค่าสูงสุดสำหรับ identity column คือค่าสูงสุดสำหรับประเภทข้อมูล. เพราะประเภทข้อมูลถูกกำหนดไว้เป็น SMALLINT ช่วงของค่าที่สามารถกำหนดให้กับ ORDERNO ได้จึงอยู่ระหว่าง 500 ถึง 32 767 เมื่อค่าของคอลัมน์มีถึง 32 767 ค่านั้นจะกลับมาเริ่มต้นใหม่ที่ 500 อีกครั้ง หาก 500 ยังคงถูกกำหนดให้กับคอลัมน์ และมีการระบุคีย์แบบเฉพาะบน identity column ก็จะมีข้อผิดพลาดคีย์ซ้ำซ้อนเกิดขึ้น การแทรกครั้งต่อไปจะพยายามใช้ 501 หากคุณไม่มีคีย์แบบเฉพาะที่ระบุไว้สำหรับ identity column, 500 จะถูกนำมาใช้อีกครั้ง, โดยไม่สนใจว่าค่านั้นจะปรากฏกี่ครั้งในตาราง.

สำหรับช่วงค่าที่กว้างกว่า ให้ระบุคอลัมน์ที่จะเป็น INTEGER หรือแม้แต่ BIGINT ถ้าคุณต้องการให้ค่าของคอลัมน์ identity ลดลง ให้ระบุค่าที่เป็นลบ สำหรับตัวเลือก INCREMENT เป็นไปได้ที่ระบุช่วงจำนวนที่ถูกต้องโดยการใช้ MINVALUE และ MAXVALUE.

คุณสามารถดัดแปลงแอ็ตทริบิวต์ของ identity column ที่มีอยู่เดิมโดยใช้ข้อความ ALTER TABLE. ตัวอย่างเช่น หากคุณต้องการรีสตาร์ท identity column ด้วยค่าใหม่:

```
ALTER TABLE ORDER
  ALTER COLUMN ORDERNO
  RESTART WITH 1
```

คุณสามารถเลื่อน identity attribute จากคอลัมน์ได้:

```
ALTER TABLE ORDER
  ALTER COLUMN ORDERNO
  DROP IDENTITY
```

คอลัมน์ ORDERNO ยังคงเป็นคอลัมน์ SMALLINT, แต่ identity attribute จะถูกลบออกไป. ระบบจะไม่สร้างค่าสำหรับคอลัมน์นี้อีกแล้ว.

สิ่งอ้างอิงที่เกี่ยวข้อง

“การเปรียบเทียบคอลัมน์ identity และลำดับ” ในหน้า 30
ขณะที่คอลัมน์ identity และลำดับมีลักษณะเหมือนกันในหลายๆทาง แต่ก็ยังมีที่ แตกต่างกันบ้าง

“การแทรกค่าเข้าในคอลัมน์ identity” ในหน้า 101
คุณสามารถแทรกค่าเข้าในคอลัมน์ identity หรืออนุญาตให้ระบบแทรกค่าให้คุณ.

"การอัพเดต identity column" ในหน้า 106
คุณสามารถอัพเดตค่าใน identity column ให้เป็นค่าที่ระบุ หรือให้ระบบสร้างค่าใหม่.

การใช้ ROWID

การใช้ ROWID เป็นอีกวิธีหนึ่งที่ทำให้ระบบ กำหนดค่าเฉพาะให้กับคอลัมน์ ROWID เหมือนกับคอลัมน์ identity แต่ แทนที่จะเป็นแอ็ตทริบิวต์ของคอลัมน์ตัวเลข กลับเป็นชนิดข้อมูล ที่แยกต่างหาก

วิธีการสร้างตารางที่คล้ายกับตัวอย่างคอลัมน์ identity:

```
CREATE TABLE ORDERS
   (ORDERNO ROWID
   GENERATED ALWAYS,
   SHIPPED_TO VARCHAR (36) ,
   ORDER_DATE DATE)
```

การสร้างและการใช้ลำดับ

ลำดับจะเหมือนกับคอลัมน์ identity ในเรื่องที่ว่าทั้งคู่จะสามารถสร้างค่าที่เป็น unique อย่างไรก็ตาม ลำดับเป็นอ็อบเจ็กต์ที่ขึ้นอยู่กับตาราง คุณสามารถใช้ลำดับเพื่อสร้างค่าได้อย่างรวดเร็ว และง่ายดาย

ลำดับไม่ผูกติดกับตาราง และสามารถเรียกใช้งานแยกต่างหากได้ นอกจากนี้, มันจะไม่ถูกควบคุมให้เป็นส่วนหนึ่งส่วนใดของ transaction ของงานของหน่วยนั้นๆ.

คุณสามารถสร้างลำดับโดยการใช้ข้อความ CREATE SEQUENCE. สำหรับตัวอย่างจะคล้ายกับตัวอย่างของคอลัมน์ identity, การสร้างลำดับ ORDER_SEQ:

```
CREATE SEQUENCE ORDER_SEQ
START WITH 500
INCREMENT BY 1
MAXVALUE 1000
CYCLE
CACHE 24
```

ลำดับนี้ถูกกำหนดให้เริ่มต้นค่าที่ 500 และเพิ่มขึ้นทีละ 1 ทุกๆครั้งที่ใช้งาน และจะรีไซเคิลเมื่อถึงค่าสูงสุด ในตัวอย่างนี้, ค่าสูงสุดสำหรับลำดับคือ 1000. เมื่อค่านั้ถึง 1000, มันจะกลับมาเริ่มต้นใหม่ที่ 500 อีกครั้ง.

ดังนั้นหลังจากที่มีการสร้างลำดับขึ้น คุณจะสามารถแทรกค่าลงในคอลัมน์โดยการใช้ลำดับ ตัวอย่างเช่น, แทรกค่าถัดไปของลำดับ ORDER_SEQ ลงในตาราง ORDERS โดยมีคอลัมน์ ORDERNO และ CUSTNO.

ก่อนอื่น ให้สร้างตาราง ORDERS:

```
CREATE TABLE ORDERS
(ORDERNO SMALLINT NOT NULL,
CUSTNO SMALLINT);
```

แล้ว, แทรกค่าลำดับ:

```
INSERT INTO ORDERS (ORDERNO, CUSTNO)
VALUES (NEXT VALUE FOR ORDER_SEQ, 12)
```

การรันคำสั่งต่อไปนี้ จะส่งกลับค่าลงมาในคอลัมน์:

```
SELECT *
FROM ORDERS
```

ตารางที่ 2. ผลลัพธ์สำหรับ SELECT จากตาราง ORDERS

ORDERNO	CUSTNO
500	12

ในตัวอย่างนี้ ค่าถัดมาสำหรับลำดับ ORDER ได้ถูกแทรก ลงไปในคอลัมน์ ORDERNO ให้เรียกข้อความ INSERT อีกครั้ง. แล้วรันคำสั่ง SELECT

ตารางที่ 3. ผลลัพธ์สำหรับ SELECT จากตาราง ORDERS

ORDERNO	CUSTNO
500	12
501	12

คุณยังสามารถแทรกค่าก่อนหน้านี้สำหรับลำดับ ORDER โดยใช้ นิพจน์ PREVIOUS VALUE คุณสามารถใช้ NEXT VALUE และ PREVIOUS VALUE ในนิพจน์ ต่อไปนี้:

- ภายใน *select-clause* ของคำสั่ง SELECT หรือคำสั่ง SELECT INTO ตราบเท่าที่คำสั่งนั้นไม่ได้ประกอบด้วยคีย์เวิร์ด DISTINCT, GROUP BY clause, ORDER BY clause, คีย์เวิร์ด UNION, คีย์เวิร์ด INTERSECT หรือคีย์เวิร์ด EXCEPT
- ภายในประโยค VALUES ของข้อความ INSERT
- ภายใน *select-clause* ของ fullselect ของ ข้อความ INSERT
- ภายในประโยค SET ของการค้นหา หรือ ตำแหน่งข้อความ UPDATE , ถึงแม้ว่า NEXT VALUE ไม่สามารถระบุลงใน *select-clause* ของ subselect ของนิพจน์ในประโยค SET

คุณสามารถเปลี่ยนแปลงลำดับโดยการใช้ข้อความ ALTER SEQUENCE. ลำดับ สามารถเปลี่ยนแปลงได้โดยแนวทางต่อไปนี้:

- การเริ่มทำ ลำดับ ต่อ
- เปลี่ยนส่วนเพิ่มระหว่างค่าลำดับที่จะเกิดขึ้นข้างหน้า
- การตั้ง หรือ การปรับค่า ต่ำสุด หรือ สูงสุด
- การเปลี่ยนจำนวนเลขแคชของลำดับ
- การเปลี่ยนแอ็ตทริบิวต์ที่กำหนดว่า ลำดับ จะเป็นวัฏจักร หรือไม่
- การเปลี่ยนว่า เลขลำดับต้องถูกสร้างขึ้นตามลำดับที่ร้องขอหรือไม่

ตัวอย่างเช่น, เปลี่ยนส่วนเพิ่มของค่าของลำดับ ORDER จาก 1 ถึง 5:

```
ALTER SEQUENCE ORDER_SEQ
INCREMENT BY 5
```

หลังจากการเปลี่ยนแปลงเสร็จสิ้น ให้รันคำสั่ง INSERT อีกครั้ง แล้วใช้ คำสั่ง SELECT ตอนนี้ตารางจะประกอบด้วยคอลัมน์ ต่อไปนี้

ตารางที่ 4. ผลลัพธ์สำหรับ SELECT จากตาราง ORDERS

ORDERNO	CUSTNO
500	12
501	12
528	12

โปรดสังเกตุว่าค่าถัดไปที่ ลำดับใช้คือ 528. ในตอนแรก, หมายเลขนี้จะปรากฏไม่ถูกต้อง. อย่างไรก็ตาม, เมื่อมองตามเหตุการณ์ที่ นำไปสู่การมอบหมายนี้. ขั้นแรก, เมื่อ ลำดับถูกสร้างขึ้นตามปกติ, ค่าของแคชจะถูกกำหนดเป็น 24. ระบบจะกำหนดค่า 24 ตัวแรกสำหรับ แคชนี้. ถัดมา, ลำดับจะมีการเปลี่ยนแปลง. เมื่อมีการใช้ข้อความ ALTER SEQUENCE, ระบบจะยกเลิกค่าที่กำหนดไว้ และเริ่มงานใหม่อีกครั้งด้วย ค่าที่มีอยู่ต่อไป; ในกรณีนี้ ค่าเริ่มต้นของแคชที่ 24 , จะบวกเพิ่มขึ้น ไปอีก, 5. ถ้าเดิมข้อความ CREATE SEQUENCE ไม่มีประโยค CACHE , ระบบจะกำหนดค่าแคชดีฟอลต์เป็น 20 โดยอัตโนมัติ. ถ้าลำดับนั้นมีการเปลี่ยนแปลง, แล้วค่าที่ใช้ได้จะเป็น 25.

หลักการที่เกี่ยวข้อง

"ลำดับ (Sequences)" ในหน้า 13
sequence คืออ็อบเจ็กต์พื้นที่ข้อมูล ที่กำหนดให้วิธีการที่ง่ายและรวดเร็วในการสร้างหมายเลขเฉพาะ.

การเปรียบเทียบคอลัมน์ identity และลำดับ

ขณะที่คอลัมน์ identity และลำดับมีลักษณะเหมือนกันในหลายๆทาง แต่ก็ยังมีที่ แตกต่างกันบ้าง

ให้พิจารณาความแตกต่างเหล่านี้ ก่อนที่คุณจะตัดสินใจเลือกใช้.

คอลัมน์ identity มีลักษณะเฉพาะตัวดังต่อไปนี้:

- เราสามารถกำหนดคอลัมน์ identity เป็นเพียงส่วนหนึ่งของตารางได้เมื่อมีการสร้าง ตารางขึ้น. หลังจากที่ตารางถูกสร้างขึ้น คุณจะไม่สามารถเปลี่ยนตารางเพื่อเพิ่มคอลัมน์ identity ได้ (อย่างไรก็ตาม ลักษณะเฉพาะตัวของคอลัมน์ identity ที่มีอยู่นั้นอาจเปลี่ยนแปลงได้)
- คอลัมน์ identity จะสร้างค่าสำหรับตารางเดี่ยวโดยอัตโนมัติ.
- เมื่อคอลัมน์ identity ถูกกำหนดเป็น GENERATED ALWAYS, ค่าที่นำไปใช้ จะถูกสร้างโดยตัวจัดการฐานข้อมูลเสมอ. แอ็พพลิเคชันจะไม่สามารถ ใช้ค่าของตัวมันเองได้ เมื่อมีการเปลี่ยนแปลงเนื้อหาของ ตาราง.
- เราสามารถใช้ฟังก์ชัน IDENTITY_VAL_LOCAL เพื่อดูค่าที่ถูกกำหนดล่าสุดสำหรับ คอลัมน์ identity.

ลำดับ มีลักษณะเฉพาะตัวดังนี้:

- ลำดับ เป็นอ็อบเจ็กต์ระบบชนิด *DTAARA ที่ไม่ผูกติดกับตาราง.
- ลำดับ จะเป็นตัวสร้างค่าเรียงลำดับที่สามารถนำไปใช้ในคำสั่ง SQL ใดๆ.
- มีนิพจน์อยู่สองแบบที่ใช้สำหรับเรียกค่าถัดไปใน ลำดับ ออกมา และ ใช้มองหาค่าก่อนหน้าที่ถูกกำหนดไว้สำหรับ ลำดับ. นิพจน์ PREVIOUS VALUE จะส่งกลับค่าที่ถูกกำหนดล่าสุดสำหรับลำดับที่ระบุ สำหรับข้อความก่อนหน้าในเซสชันปัจจุบัน. นิพจน์ NEXT VALUE จะส่งกลับค่าถัดไปสำหรับ ลำดับที่ระบุ. การใช้นิพจน์เหล่านี้ จะอนุญาตให้ค่าที่เหมือนกันถูกนำไปใช้ข้ามข้อความ SQL ได้หลายๆข้อความ ในหลายๆตาราง.

แม้ว่าสิ่งเหล่านี้ไม่ใช่ลักษณะเฉพาะตัวทั้งหมดของคอลัมน์และลำดับ แต่ลักษณะเฉพาะตัวเหล่านี้จะช่วยคุณในการตัดสินใจว่าจะใช้อะไร ขึ้นอยู่กับการออกแบบฐานข้อมูลของคุณ และแอ็พพลิเคชันที่ใช้ฐานข้อมูลนั้น.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การสร้างและเปลี่ยน identity column" ในหน้า 26
ทุกครั้งที่เพิ่มแถวใหม่เข้าไปยังตารางด้วย identity column ระบบจะสร้างค่าของ identity column สำหรับแถวใหม่นี้

การสร้างเลเบลอธิบายโดยใช้คำสั่ง LABEL ON

บางครั้งข้อความอธิบายมีประโยชน์ สำหรับอ็อบเจ็กต์ (เช่น ตาราง หรือดรรชนี) หรือใช้เป็นข้อความคอลัมน์หรือส่วนหัวคอลัมน์ คุณสามารถสร้างเลเบลคำอธิบายได้มากกว่านี้สำหรับชื่อเหล่านี้โดยใช้ คำสั่ง LABEL ON

สามารถเห็นเลเบลเหล่านี้ในแคตาล็อก SQL ในคอลัมน์ LABEL .

คำสั่ง LABEL ON จะเป็นดังนี้:

```
LABEL ON
  TABLE CORPDATA.DEPARTMENT IS 'Department Structure Table'

LABEL ON
  COLUMN CORPDATA.DEPARTMENT.ADMRDEPT IS 'Reports to Dept.'
```

หลังจากคำสั่งเหล่านี้ถูกรันแล้ว ตารางที่ชื่อ DEPARTMENT จะแสดงผลรายละเอียดข้อความเป็น *Department Structure Table* และคอลัมน์ที่ชื่อ *ADMRDEPT* จะแสดงผลส่วนหัว *Reports to Dept* เลเบลสำหรับอ็อบเจ็กต์หรือคอลัมน์ไม่สามารถมีความยาวเกิน 50 ไบต์ และเลเบล สำหรับส่วนหัวคอลัมน์ไม่สามารถมีความยาวเกิน 60 ไบต์ (รวมถึงช่องว่าง) ต่อไปนี้คือตัวอย่างของคำสั่ง LABEL ON สำหรับส่วนหัวของคอลัมน์:

คำสั่ง LABEL ON มีส่วนหัวคอลัมน์ 1 และส่วนหัวคอลัมน์ 2:

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.EMPNO IS
          'หมายเลขประจำตัว            พนักงาน'
```

คำสั่ง LABEL ON มีส่วนหัวคอลัมน์สามระดับสำหรับคอลัมน์ SALARY:

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.SALARY IS
           'เงินเดือน               ประจำปี               (เป็นดอลลาร์)'
```

คำสั่ง LABEL ON นี้จะลบส่วนหัวคอลัมน์ของ SALARY:

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.SALARY IS ''
```

คำสั่ง LABEL ON นี้จัดเตรียมส่วนหัวคอลัมน์ DBCS ที่มี ระดับสองระดับที่ระบุไว้:

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.SALARY IS
           '<AABBCCDD>          <EEFFGG>'
```

คำสั่ง LABEL ON จะมีข้อความคอลัมน์สำหรับคอลัมน์ EDLEVEL:

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.EDLEVEL TEXT IS
   'จำนวนปีการศึกษาภาคบังคับ'
```

สิ่งอ้างอิงที่เกี่ยวข้อง

LABEL

การอธิบาย SQL อ็อบเจ็กต์โดยใช้ COMMENT ON

หลังจากที่คุณสร้างอ็อบเจ็กต์ SQL เช่น ตาราง หรือมุมมอง คุณจะสามารถจัดหาข้อมูลอ็อบเจ็กต์สำหรับการอ้างอิงในอนาคตโดยใช้ คำสั่ง COMMENT ON

ข้อมูลอาจเป็นวัตถุประสงค์ของอ็อบเจ็กต์, ที่ใช้, และเป็นสิ่งอื่นๆ ที่ผิดปรกติ หรือเป็นสิ่งพิเศษ. คุณสามารถแทรกข้อมูลที่คล้ายคลึงกันเกี่ยวกับแต่ละคอลัมน์ของตารางหรือมุมมองได้. ข้อสังเกตุจะมีประโยชน์มากหากชื่อของคุณไม่ได้ระบุเนื้อหาของคอลัมน์หรืออ็อบเจ็กต์ไว้อย่างชัดเจน. ในกรณีนั้น, ใช้หมายเหตุเพื่ออธิบายเนื้อหาเฉพาะของคอลัมน์ หรืออ็อยเจ็กต์. โดยปกติ, หมายเหตุของคุณต้องมีอักขระไม่เกิน 2000 อักขระ, แต่สำหรับลำดับ จะมีความยาวมากสุด 500 อักขระ. ถ้าอ็อบเจ็กต์มีข้อสังเกตุอยู่แล้ว ข้อสังเกตุเดิมจะถูกแทนที่ด้วยข้อสังเกตุใหม่

ตัวอย่างการใช้ COMMENT ON มีดังนี้:

```
COMMENT ON TABLE CORPDATA.EMPLOYEE IS
   'ตารางพนักงาน.  แต่ละแถวในตารางนี้จะแสดง
    พนักงานหนึ่งคนของบริษัท.'
```

การรับข้อสังเกตุหลังจากรันคำสั่ง COMMENT ON

หลังจากรันคำสั่ง COMMENT ON สำหรับตาราง ข้อสังเกตุของคุณ จะถูกเก็บไว้ในคอลัมน์ *LONG_COMMENT* ของ SYSTABLES ข้อสังเกตุสำหรับอ็อบเจ็กต์อื่นๆ จะถูกเก็บไว้ในคอลัมน์ LONG_COMMENT ของตารางแค็ตตาล็อกที่เหมาะสม. ตัวอย่างต่อไปนี้จะรับข้อสังเกตุที่เพิ่มโดยคำสั่ง COMMENT ON ในตัวอย่างก่อนหน้านี้:

```
SELECT LONG_COMMENT
  FROM CORPDATA.SYSTABLES
  WHERE NAME = 'EMPLOYEE'
```

สิ่งอ้างอิงที่เกี่ยวข้อง

COMMENT

การเปลี่ยน definition ตาราง

คุณสามารถเปลี่ยน definition ของตารางได้โดยการเพิ่ม คอลัมน์, การเปลี่ยน definition ของคอลัมน์ที่มีอยู่เดิม เช่น ความยาว หรือ ค่าดีฟอลต์, การปล่อยคอลัมน์ที่มีอยู่เดิม, การเพิ่มข้อจำกัด หรือการลบข้อจำกัด

เมื่อต้องการเปลี่ยน definition ของตาราง ให้ใช้คำสั่ง SQL ALTER TABLE

คุณสามารถเพิ่ม, เปลี่ยน, หรือลบคอลัมน์และเพิ่มหรือลบข้อจำกัดทั้งหมดด้วยคำสั่ง ALTER TABLE. อย่างไรก็ตาม, สามารถอ้างอิงถึงคอลัมน์เดี่ยวเพียงหนึ่งครั้งใน ADD COLUMN, ALTER COLUMN, และ DROP COLUMN clause. นั่นคือ, คุณไม่สามารถเพิ่มคอลัมน์และเปลี่ยนคอลัมน์นั้นในคำสั่ง ALTER TABLE เดียวกัน.

สิ่งอ้างอิงที่เกี่ยวข้อง

ALTER TABLE

การเพิ่มคอลัมน์

เมื่อคุณเพิ่มคอลัมน์ใหม่เข้ายังตาราง, คอลัมน์จะถูก initialize ด้วยค่าดีฟอลต์สำหรับแถวทั้งหมดที่มีอยู่เดิม. หากไม่ได้ระบุว่าไม่ใช่ศูนย์, จะต้องระบุค่าดีฟอลต์ด้วย.

คุณสามารถเพิ่มคอลัมน์ไปยังตารางโดยใช้ ADD COLUMN clause ของคำสั่ง SQL ALTER TABLE.

ตารางที่เปลี่ยนไปอาจประกอบด้วยคอลัมน์ไม่เกิน 8000. จำนวนการนับไบต์ของคอลัมน์ต้องไม่เกิน 32766 หรือ, หากมีการระบุคอลัมน์ VARCHAR หรือ VARGRAPHIC, 32740. หากมีการระบุคอลัมน์ LOB, จำนวนของการนับไบต์ของเร็กคอร์ดข้อมูลของคอลัมน์ต้องมีขนาดไม่เกิน 15 728 640.

สิ่งอ้างอิงที่เกี่ยวข้อง

ALTER TABLE

การเปลี่ยนคอลัมน์

คุณสามารถเปลี่ยน definition คอลัมน์ในตารางได้โดยใช้ ALTER COLUMN clause ของคำสั่ง ALTER TABLE.

เมื่อคุณเปลี่ยนประเภทข้อมูลของคอลัมน์ที่มีอยู่, แอ็ตทริบิวต์เดิมและใหม่ต้องทำงานร่วมกันได้. คุณสามารถเปลี่ยนอักขระ, กราฟิก, หรือ คอลัมน์ไบนารี ได้เสมอจาก ความยาวคงที่ เป็นความยาวไม่คงที่ หรือ LOB; หรือ จากความยาวไม่คงที่ หรือ LOB เป็น ความยาวคงที่.

เมื่อคุณแปลงไปเป็นประเภทข้อมูลโดยมีความยาวเพิ่มขึ้น ข้อมูลจะถูกเติมเต็มด้วยแพ็ดอักขระที่เหมาะสม เมื่อคุณแปลงไปเป็นประเภทข้อมูลซึ่งมีความยาวน้อยกว่า ข้อมูลอาจหายเพราะเกิดการตัดปลาย ข้อความสอบถามจะถามให้คุณยืนยันการร้องขอ.

หากคุณมีคอลัมน์ที่ไม่อนุญาตให้มีค่าเป็นศูนย์และคุณต้องการเปลี่ยนให้เป็นคอลัมน์ที่อนุญาตให้มีค่าเป็นศูนย์, ให้ใช้ DROP NOT NULL clause. หากคุณมีคอลัมน์ที่อนุญาตให้มีค่าศูนย์และคุณต้องการป้องกันการใช้ค่าศูนย์, ให้ใช้ SET NOT NULL clause. หากค่าใดค่าหนึ่งของค่าที่มีอยู่ในคอลัมน์นั้นเป็นค่าศูนย์, ALTER TABLE จะไม่ถูกเรียกทำงานและจะเกิด SQLCODE of -190.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การแปลงชนิดข้อมูลที่อนุญาต"
เมื่อคุณเปลี่ยนประเภทข้อมูลของคอลัมน์ที่มีอยู่, แอ็ตทริบิวต์เดิมและใหม่ต้องทำงานร่วมกันได้.

ข้อมูลที่เกี่ยวข้อง

ALTER TABLE

การแปลงชนิดข้อมูลที่อนุญาต

เมื่อคุณเปลี่ยนประเภทข้อมูลของคอลัมน์ที่มีอยู่, แอ็ตทริบิวต์เดิมและใหม่ต้องทำงานร่วมกันได้.

ตารางที่ 5. การแปลงที่ได้รับอนุญาต

จากชนิดข้อมูล	เป็นชนิดข้อมูล
Decimal	Numeric
Decimal	Bigint, Integer, Smallint
Decimal	Decfloat
Decimal	Float
Numeric	Decimal
Numeric	Bigint, Integer, Smallint

ตารางที่ 5. การแปลงที่ได้รับอนุญาต (ต่อ)

จากชนิดข้อมูล	เป็นชนิดข้อมูล
Numeric	Decfloat
Numeric	Float
Bigint, Integer, Smallint	Decimal
Bigint, Integer, Smallint	Numeric
Bigint, Integer, Smallint	Decfloat
Bigint, Integer, Smallint	Float
Float	Decimal
Float	Numeric
Float	Bigint, Integer, Smallint
Float	Decfloat
Character	DBCS-open
Character	UCS-2 or UTF-16 graphic
DBCS-open	Character
DBCS-open	UCS-2 or UTF-16 graphic
DBCS-either	Character
DBCS-either	DBCS-open
DBCS-either	UCS-2 or UTF-16 graphic
DBCS-only	DBCS-open
DBCS-only	DBCS graphic
DBCS-only	UCS-2 or UTF-16 graphic
DBCS graphic	UCS-2 or UTF-16 graphic
UCS-2 or UTF-16 graphic	Character
UCS-2 or UTF-16 graphic	DBCS-open
UCS-2 or UTF-16 graphic	DBCS graphic
distinct type	*source type*
source type	*distinct type*

เมื่อคุณเปลี่ยนแปลงคอลัมน์ที่มีอยู่ เฉพาะแอ็ตทริบิวต์ที่คุณระบุไว้เท่านั้นที่จะเปลี่ยนไป แอ็ตทริบิวต์อื่นๆ ทั้งหมดจะไม่ถูกเปลี่ยนแปลง ตัวอย่างเช่น คุณมีตารางที่มี definition ของตารางดังต่อไปนี้:

```
CREATE TABLE EX1 (COL1 CHAR(10) DEFAULT 'COL1',
                  COL2 VARCHAR(20) ALLOCATE(10) CCSID 937,
                  COL3 VARGRAPHIC(20) ALLOCATE(10)
                       NOT NULL WITH DEFAULT)
```

หลังจากที่คุณรันคำสั่ง ALTER TABLE ต่อไปนี้ COL2 จะยังคงมีความยาวที่ถูกจัดสรรเท่ากับ 10 และ CCSID 937 และ COL3 ยังคงมี ขนาดความยาวที่จัดสรรเป็น 10

```
ALTER TABLE EX1 ALTER COLUMN COL2 SET DATA TYPE VARCHAR(30)
                ALTER COLUMN COL3 DROP NOT NULL
```

สิ่งอ้างอิงที่เกี่ยวข้อง

"การเปลี่ยนคอลัมน์" ในหน้า 33

คุณสามารถเปลี่ยน definition คอลัมน์ในตารางได้โดยใช้ ALTER COLUMN clause ของคำสั่ง ALTER TABLE.

การลบคอลัมน์

คุณสามารถลบคอลัมน์โดยใช้ DROP COLUMN clause ของคำสั่ง ALTER TABLE.

การลบคอลัมน์จะลบคอลัมน์นั้นออกจาก definition ตาราง. หากมีการระบุ CASCADE, มุมมองใดๆ , ดรรชนีใดๆ , และข้อจำกัดใดๆ ที่ขึ้นอยู่กับคอลัมน์นั้นจะถูกลบออกไปเช่นกัน. หากมีการระบุ RESTRICT, และมุมมองใดๆ, ดรรชนีใดๆ, หรือข้อจำกัดที่ขึ้นอยู่กับคอลัมน์, คอลัมน์จะไม่ถูกลบออกไป และจะมีการออกคำสั่ง SQLCODE เป็น -196.

```
ALTER TABLE DEPT
  DROP COLUMN NUMDEPT
```

สิ่งอ้างอิงที่เกี่ยวข้อง

ALTER TABLE

ลำดับการดำเนินการของคำสั่ง ALTER TABLE

การดำเนินการสำหรับคำสั่ง ALTER TABLE ถูกทำ ตามลำดับที่กำหนด

คำสั่ง ALTER TABLE จะถูกใช้งานตาม ชุดขั้นตอนต่อไปนี้:

1. ลบข้อจำกัด
2. ลบตาราง materialized query
3. ลบข้อมูลพาร์ติชัน
4. ลบคอลัมน์ที่มีการระบุอ็อพชัน RESTRICT
5. เปลี่ยน definition คอลัมน์ (หมายรวมถึงการเพิ่มคอลัมน์และการลบคอลัมน์ที่มีการระบุอ็อพชัน CASCADE)
6. ใส่เพิ่ม หรือเปลี่ยนตาราง materialized query
7. ใส่เพิ่มพาร์ติชันในตาราง
8. เพิ่มข้อจำกัด

ภายในแต่ละขั้นตอน, ลำดับที่คุณระบุ clause คือลำดับที่คุณดำเนินการ, โดยมี exception หนึ่งข้อ. หากคอลัมน์ใดคอลัมน์หนึ่งถูกลบออก, การดำเนินการนั้นจะเสร็จสิ้นแบบโลจิคัลก่อนที่ definition ของคอลัมน์ใดๆ จะถูกเพิ่มหรือเปลี่ยนไป, ในกรณีที่ความยาวเร็กคอร์ดเพิ่มขึ้นเนื่องมาจากคำสั่ง ALTER TABLE.

การสร้างและการใช้งานชื่อ ALIAS

เมื่อคุณอ้างถึงตารางหรือมุมมองที่มีอยู่เดิม หรือถึงไฟล์ฟิสิคัลที่ประกอบด้วยเมมเบอร์จำนวนมาก คุณสามารถเลี่ยงการใช้การบันทึกทับไฟล์โดยการสร้าง alias ในการสร้าง alias ให้ใช้คำสั่ง CREATE ALIAS

คุณสามารถสร้าง alias สำหรับ:

- ตารางหรือมุมมอง
- *เมมเบอร์*ของตาราง

alias ของตารางจะกำหนดชื่อสำหรับไฟล์, รวมถึงชื่อเมมเบอร์เฉพาะ. คุณสามารถใช้ชื่อ alias นี้ในคำสั่ง SQL โดยวิธีเดียวกับที่ใช้ใน ชื่อของตาราง. ต่างจากการบันทึกทับค่าเดิม, ชื่อ alias คืออ็อบเจ็กต์ที่มีอยู่จนกว่าจะถูกลบทิ้ง.

ตัวอย่างเช่น, หากมีไฟล์เมมเบอร์จำนวนมาก MYLIB.MYFILE พร้อมด้วยเมมเบอร์ MBR1 และ MBR2, สามารถสร้าง alias ไว้สำหรับเมมเบอร์ที่สอง เพื่อที่ว่า SQL จะสะดวกในการอ้างถึงเมมเบอร์ที่สองนั้นได้.

```
CREATE ALIAS MYLIB.MYMBR2_ALIAS FOR MYLIB.MYFILE (MBR2)
```

เมื่อมีการระบุ alias MYLIB.MYMBR2_ALIAS ไว้บนคำสั่งการแทรก ต่อไปนี้ ค่าจะถูกแทรกลงในเมมเบอร์ MBR2 ใน MYLIB.MYFILE:

```
INSERT INTO MYLIB.MYMBR2_ALIAS VALUES('ABC', 6)
```

สามารถระบุชื่อ alias ไว้บนคำสั่ง DDL. สมมติว่า MYLIB.MYALIAS เป็น alias สำหรับตาราง MYLIB.MYTABLE คำสั่ง DROP ต่อไปนี้จะลบตาราง MYLIB.MYTABLE:

```
DROP TABLE MYLIB.MYALIAS
```

หากคุณต้องการลบชื่อ alias แทน, โปรดระบุคีย์เวิร์ด ALIAS ไว้บนคำสั่ง:

```
DROP ALIAS MYLIB.MYALIAS
```

สิ่งอ้างอิงที่เกี่ยวข้อง

CREATE ALIAS

การสร้างและการใช้มุมมอง

สามารถใช้มุมมองเพื่อเข้าใช้งานข้อมูลในตารางหนึ่งตารางหรือมากกว่าหรือมุมมองหนึ่งมุมมองหรือมากกว่าได้. คุณสามารถสร้างมุมมองได้โดยใช้คำสั่ง SELECT.

ตัวอย่างเช่น ให้สร้างมุมมองที่เลือกเฉพาะนามสกุลและแผนกของผู้จัดการทั้งหมด:

```
CREATE VIEW CORPDATA.EMP_MANAGERS AS
   SELECT LASTNAME, WORKDEPT FROM  CORPDATA.EMPLOYEE
   WHERE JOB = 'MANAGER'
```

หลังจากที่คุณสร้างมุมมองแล้ว คุณสามารถใช้งานมุมมองนั้นในคำสั่ง SQL เหมือนกับตาราง คุณสามารถเปลี่ยนข้อมูลในตารางฐานผ่านทางมุมมอง คำสั่ง SELECT ต่อไปนี้จะแสดงเนื้อหาของ EMP_MANAGERS:

```
SELECT *
      FROM CORPDATA.EMP_MANAGERS
```

ผลที่ได้คือ

LASTNAME	WORKDEPT
THOMPSON	B01
KWAN	C01
GEYER	E01
STERN	D11
PULASKI	D21
HENDERSON	E11
SPENSER	E21

หากรายการให้เลือกมีส่วนประกอบนอกเหนือจากคอลัมน์เช่นนิพจน์, ฟังก์ชัน, ค่าคงที่, หรือการลงทะเบียนพิเศษ, และ AS clause ไม่ได้ถูกใช้งานเพื่อตั้งชื่อคอลัมน์ , ต้องระบุรายการคอลัมน์สำหรับมุมมอง. ในตัวอย่างต่อไปนี้, คอลัมน์ของมุมมองคือ LASTNAME และ YEARSOFSERVICE.

```
CREATE VIEW CORPDATA.EMP_YEARSOFSERVICE
    (LASTNAME, YEARSOFSERVICE) AS
   SELECT LASTNAME, YEAR (CURRENT DATE - HIREDATE)
   FROM CORPDATA.EMPLOYEE
```

เนื่องจากผลลัพธ์ของการสอบถามมุมมองนี้เปลี่ยนตามการเปลี่ยนปีปัจจุบัน ซึ่งไม่ได้รวมอยู่ใน ที่นี้

คุณสามารถกำหนดมุมมองก่อนหน้าได้โดยการใช้ AS clause ในรายการให้เลือกเพื่อตั้งชื่อคอลัมน์ในมุมมอง ตัวอย่างเช่น:

```
CREATE VIEW CORPDATA.EMP_YEARSOFSERVICE AS
    SELECT LASTNAME,
           YEARS (CURRENT_DATE - HIREDATE) AS YEARSOFSERVICE
    FROM CORPDATA.EMPLOYEE
```

การใช้คีย์เวิร์ด UNION, คุณสามารถรวมการเลือกย่อยสองรายการหรือมากกว่าเพื่อสร้างมุมมองเดี่ยว. ตัวอย่างเช่น:

```
CREATE VIEW D11_EMPS_PROJECTS AS
  (SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
   UNION
   SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO = 'MA2112' OR
        PROJNO = 'MA2113' OR
        PROJNO = 'AD3111')
```

มุมมองนี้มีข้อมูลต่อไปนี้

ตารางที่ 6. ผลลัพธ์ของการสร้างมุมมองแบบ UNION

EMPNO
000060
000150
000160
000170
000180
000190
000200
000210
000220
000230
000240
200170
200220

สามารถสร้างมุมมองด้วยการเรียงลำดับที่มีทำงานก็ต่อเมื่อคำสั่ง CREATE VIEW ถูกรัน. ลำดับการจัดเรียงประยุกต์ใช้กับอักขระทั้งหมด, หรือ UCS-2 หรือกราฟิก UTF-16 เปรียบเทียบในการเลือกย่อยของคำสั่ง CREATE VIEW.

นอกจากนี้ คุณสามารถสร้างมุมมองได้โดยใช้ WITH CHECK OPTION clause เพื่อระบุระดับของการตรวจสอบเมื่อมีการแทรกหรืออัพเดตข้อมูลผ่านมุมมอง

หลักการที่เกี่ยวข้อง

"การดึงข้อมูลโดยใช้คำสั่ง SELECT" ในหน้า 44
คำสั่ง SELECT ปรับแต่งเคียวรี่ของคุณเพื่อรวบรวม ข้อมูล คุณสามารถใช้คำสั่ง SELECT เพื่อดึงแถวที่เฉพาะเจาะจงหรือดึง ข้อมูลในลักษณะที่เฉพาะเจาะจง

"ลำดับการเรียง และ normalization ใน SQL" ในหน้า 121
ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การใช้คีย์เวิร์ด UNION เพื่อรวมการเลือกย่อย" ในหน้า 84
การใช้คีย์เวิร์ด UNION, คุณสามารถรวมการเลือกย่อยมากกว่าสองได้ เพื่อสร้าง fullselect.

CREATE VIEW

WITH CHECK OPTION บนมุมมอง

WITH CHECK OPTION คือ clause ที่เป็นทางเลือกบน คำสั่ง CREATE VIEW โดยจะระบุระดับของการตรวจสอบที่ควรดำเนินการ เมื่อแทรกข้อมูลหรืออัพเดตข้อมูลผ่านมุมมอง

หากมีการระบุ WITH CHECK OPTION ทุกๆ แถวที่ถูกแทรก หรืออัพเดตผ่านมุมมองจะต้องตรงตาม definition ของ มุมมองนั้น ไม่สามารถระบุอ็อพชันดังกล่าวได้หากมุมมองเป็นแบบอ่านอย่างเดียว definition ของมุมมองจะต้องไม่รวมการสืบค้นย่อย.

หากมุมมองถูกสร้างขึ้นโดยไม่มี WITH CHECK OPTION clause, การแทรกและการอัพเดตที่กระทำบนมุมมอง จะไม่ถูกตรวจสอบว่าตรงตาม definition ของมุมมองหรือไม่. แต่อาจมีการตรวจสอบบางอย่างอยู่หากมุมมอง ขึ้นโดยตรงหรือโดยอ้อมกับมุมมองอื่นซึ่งประกอบด้วย WITH CHECK OPTION. เนื่องจากไม่ได้ใช้ definition ของมุมมอง ดังนั้น แถวที่ไม่ตรงกับ definition ของมุมมองจึงอาจถูกแทรกหรืออัพเดตผ่านมุมมองดังกล่าว นั่นหมายความว่า แถวไม่สามารถถูกเลือกได้อีกครั้งโดยผ่านมุมมอง

สิ่งอ้างอิงที่เกี่ยวข้อง

CREATE VIEW

WITH CASCADED CHECK OPTION:

WITH CASCADED CHECK OPTION clause ระบุ ว่าทุกๆ แถวที่ถูกแทรก หรืออัพเดตผ่านมุมมองจะต้องตรงตาม definition ของมุมมอง

นอกจากนี้, เงื่อนไขการค้นหามุมมอง dependent ทั้งหมดจะถูกตรวจสอบเมื่อมีการแทรกหรืออัพเดตแถว. หากแถวไม่ตรงตาม definition ของมุมมอง จะไม่สามารถเรียกแถวดังกล่าวออกมาได้โดยผ่านทางมุมมอง

สำหรับตัวอย่าง, ให้พิจารณามุมมองที่แก้ไขได้ดังต่อไปนี้:

```
CREATE VIEW V1 AS SELECT COL1
        FROM T1 WHERE COL1 > 10
```

เนื่องจากไม่มีการระบุ WITH CHECK OPTION, คำสั่ง INSERT ต่อไปนี้จึงใช้ได้ แม้ว่าค่าที่ถูกแทรกจะไม่เป็นไปตามเงื่อนไขการค้นหาของมุมมอง.

```
INSERT INTO V1 VALUES (5)
```

สร้างอีกมุมมองหนึ่งที่ V1 โดยระบุ WITH CASCADED CHECK OPTION clause:

```
CREATE VIEW V2 AS SELECT COL1
          FROM V1 WITH CASCADED CHECK OPTION
```

ข้อความ INSERT ดังต่อไปนี้ล้มเหลวเนื่องจากมีการสร้างแถวซึ่งไม่เป็นไปตาม คำจำกัดความของ V2:

```
INSERT INTO V2 VALUES (5)
```

พิจารณาหนึ่งมุมมองหรือมากกว่าที่สร้างขึ้นที่ V2:

```
CREATE VIEW V3 AS SELECT COL1
         FROM V2 WHERE COL1 < 100
```

คำสั่ง INSERT ต่อไปนี้ใช้ไม่ได้เนื่องจาก V3 ต้องอิงกับ V2, และ V2 มี WITH CASCADED CHECK OPTION.

```
INSERT INTO V3 VALUES (5)
```

อย่างไรก็ตาม, คำสั่ง INSERT ต่อไปนี้ใช้ได้เนื่องจากตรงตาม definition ของ V2. เนื่องจาก V3 ไม่มี WITH CASCADED CHECK OPTION, จึงไม่ใช่เรื่องสำคัญที่ว่าคำสั่งดังกล่าว จะไม่ตรงตาม definition ของ V3.

```
INSERT INTO V3 VALUES (200)
```

WITH LOCAL CHECK OPTION:

WITH LOCAL CHECK OPTION clause เหมือนกับ WITH CASCADED CHECK OPTION clause เว้นแต่คุณสามารถอัพเดตแถวได้ ซึ่ง ทำให้ไม่สามารถเรียกแถวออกมาผ่านทางมุมมองอีกต่อไป กรณีนี้จะเกิดขึ้นได้ต่อเมื่อมุมมองนั้นต้องอิงโดยตรงหรือโดยอ้อมกับมุมมองที่ถูกกำหนดโดยไม่มี WITH CHECK OPTION clause

ตัวอย่างเช่น, ให้พิจารณามุมมองที่อัพเดตได้เหมือนกัน ซึ่งใช้อยู่ในตัวอย่างก่อนหน้านี้:

```
CREATE VIEW V1 AS SELECT COL1
          FROM T1 WHERE COL1 > 10
```

สร้างมุมมองที่สองบน V1, โดยครั้งนี้ ให้ระบุ WITH LOCAL CHECK OPTION:

```
CREATE VIEW V2 AS SELECT COL1
          FROM V1 WITH LOCAL CHECK OPTION
```

คำสั่ง INSERT เดียวกันที่ใช้ไม่ได้ในตัวอย่าง CASCADED CHECK OPTION ก่อนหน้านี้ จะสามารถใช้ได้ในครั้งนี้ เนื่องจาก V2 ไม่มีเงื่อนไขการค้นหาใดๆ, และเงื่อนไขการค้นหาของ V1 ไม่จำเป็นต้องถูกตรวจสอบ เนื่องจาก V1 ไม่ได้ระบุอ็อพชันการตรวจสอบ.

```
INSERT INTO V2 VALUES (5)
```

ให้พิจารณาหนึ่งมุมมอง หรือมากกว่าที่สร้างขึ้นบน V2:

```
CREATE VIEW V3 AS SELECT COL1
     FROM V2 WHERE COL1 < 100
```

INSERT ต่อไปนี้จะใช้ได้อีกครั้งเพราะเงื่อนไขการค้นหาบน V1 ไม่ได้ถูกตรวจสอบเนื่องจากมี WITH LOCAL CHECK OPTION บน V2, เทียบกับ WITH CASCADED CHECK OPTION ในตัวอย่างก่อนหน้านี้.

```
INSERT INTO V3 VALUES (5)
```

ความแตกต่างระหว่าง LOCAL และ CASCADED CHECK OPTION อยู่ที่จำนวนครั้งในการตรวจสอบเงื่อนไขการค้นหาของมุมมอง dependent เมื่อมีการแทรกหรืออัพเดตแถว.

- WITH LOCAL CHECK OPTION ระบุว่าเงื่อนไขการตรวจสอบเฉพาะมุมมอง dependent ที่มี WITH LOCAL CHECK OPTION หรือ WITH CASCADED CHECK OPTION จะถูกตรวจสอบเมื่อมีการแทรกหรืออัพเดตแถว.
- WITH CASCADED CHECK OPTION ระบุว่าเงื่อนไขการตรวจสอบของ dependent view ทั้งหมดจะถูกตรวจสอบ เมื่อมีการแทรกหรืออัพเดตแถว.

ตัวอย่าง: อ็อพชันการตรวจสอบแบบต่อเรียง:

ตัวอย่างนี้แสดงวิธีการใช้อ็อพชันการตรวจสอบ กับมุมมองแบบ dependent ซึ่งถูกกำหนดโดยใช้หรือไม่ใช้อ็อพชันการตรวจสอบ

ใช้ตารางและมุมมองต่อไปนี้:

```
CREATE TABLE T1 (COL1 CHAR(10))

CREATE VIEW V1 AS SELECT COL1
      FROM T1 WHERE COL1 LIKE 'A%'

CREATE VIEW V2 AS SELECT COL1
      FROM V1 WHERE COL1 LIKE '%Z'
                           WITH LOCAL CHECK OPTION

CREATE VIEW V3 AS SELECT COL1
      FROM V2 WHERE COL1 LIKE 'AB%'

CREATE VIEW V4 AS SELECT COL1
      FROM V3 WHERE COL1 LIKE '%YZ'
                            WITH CASCADED CHECK OPTION

CREATE VIEW V5 AS SELECT COL1
      FROM V4 WHERE COL1 LIKE 'ABC%'
```

ระบบจะตรวจสอบเงื่อนไขการค้นหาต่างๆ โดยขึ้นอยู่กับว่ามุมมองใดที่ทำงานอยู่โดยใช้คำสั่ง INSERT หรือ UPDATE

- หาก V1 ทำงานอยู่, จะไม่มีการตรวจสอบเงื่อนไขใดๆ เนื่องจาก V1 ไม่ได้ระบุ WITH CHECK OPTION.
- หาก V2 ทำงานอยู่,
  - COL1 ต้องสิ้นสุดด้วยตัวอักษร Z, แต่มันไม่จำเป็นต้องเริ่มต้นด้วยตัวอักษร A. ทั้งนี้เนื่องจากตัวเลือกการตรวจสอบเป็น LOCAL, และมุมมอง V1 ไม่ได้มีการระบุ ตัวเลือกการตรวจสอบ.
- หาก V3 ทำงานอยู่,
  - COL1 จะต้องจบด้วยตัวอักษร Z, แต่ไม่จำเป็นต้องขึ้นต้นด้วยตัวอักษร A, V3 ไม่ได้ระบุอ็อพชันการตรวจสอบไว้, ดังนั้น จึงไม่จำเป็นต้องเป็นไปตามเงื่อนไขการค้นหาของตนเอง. อย่างไรก็ตาม ต้องมีการตรวจสอบเงื่อนไขการค้นหาสำหรับ V2 เนื่องจาก V3 ถูกกำหนดไว้บน V2 และ V2 มีอ็อพชันการตรวจสอบ
- หาก V4 ทำงานอยู่,
  - COL1 ต้องขึ้นต้นด้วย 'AB' และจบด้วย 'YZ' เนื่องจาก V4 มีการระบุ WITH CASCADED CHECK OPTION, ทุกๆ เงื่อนไขการค้นหาสำหรับทุกมุมมองที่ V4 ต้องยึดตามจึงต้องถูกตรวจสอบ.
- หาก V5 ทำงานอยู่,
  - COL1 ต้องขึ้นต้นด้วย 'AB', ไม่จำเป็นต้องเป็น 'ABC'. ที่เป็นเช่นนี้เพราะ V5 ไม่ได้ระบุอ็อพชันการตรวจสอบ, ดังนั้น จึงไม่จำเป็นต้องตรวจสอบเงื่อนไขการค้นหาของตนเอง. อย่างไรก็ตาม, เนื่องจาก V5 ถูกระบุไว้บน V4, และ V4 มีอ็อพชันการตรวจสอบแบบต่อเรียง, ดังนั้นจึงต้องมีการตรวจสอบเงื่อนไขการค้นหาทั้งหมดของ V4, V3, V2, และ V1. กล่าวคือ, COL1 ต้องขึ้นต้นด้วย 'AB' และจบด้วย 'YZ'.

หาก V5 ถูกสร้างขึ้นด้วย WITH LOCAL CHECK OPTION, การทำงานบน V5 ย่อมหมายถึงว่า COL1 ต้องขึ้นต้นด้วย 'ABC' และจบด้วย 'YZ'. LOCAL CHECK OPTION ได้ใส่ข้อกำหนดเพิ่มเติมว่าอักขระตัวที่สามต้องเป็น 'C'.

การสร้างดรรชนี

คุณสามารถใช้ดรรชนี เพื่อเรียงลำดับ และเลือกข้อมูล. นอกจากนี้, ดรรชนียังช่วย ระบบให้เรียกข้อมูลออกมาได้เร็วขึ้น เพื่อประสิทธิภาพของการสอบถามที่ดีกว่าเดิม.

การใช้ข้อความ CREATE INDEX เพื่อสร้างดรรชนี. ตัวอย่างต่อไปนี้จะสร้างดรรชนีของคอลัมน์ *LASTNAME* ในตาราง CORPDATA.EMPLOYEE:

```
CREATE INDEX CORPDATA.INX1 ON CORPDATA.EMPLOYEE (LASTNAME)
```

นอกจากนี้คุณยังสามารถสร้างดรรชนีที่ไม่ตรงกับ ข้อมูลสำหรับคอลัมน์ในตาราง ตัวอย่างเช่น คุณสามารถสร้างดรรชนีที่ใช้เวอร์ชันตัวพิมพ์ใหญ่ของชื่อพนักงาน

```
CREATE INDEX CORPDATA.INX2 ON CORPDATA.EMPLOYEE (UPPER(LASTNAME))
```

สามารถใช้นิพจน์ส่วนใหญ่ที่อนุญาต โดย SQL ใน definition ของคอลัมน์หลัก

คุณสามารถสร้างดรรชนีจำนวนเท่าใดก็ได้. อย่างไรก็ตาม, เนื่องจากระบบมีการปรับปรุง ดรรชนี, ดรรชนีที่มีขนาดใหญ่สามารถส่งผลกระทบต่อประสิทธิภาพการทำงานได้. ดรรชนีประเภทหนึ่ง, ซึ่งก็คือดรรชนี vector แบบเข้ารหัส (EVI), ทำให้สแกนได้อย่างรวดเร็วซึ่งทำให้ประมวลผลแบบขนานได้ง่าย.

หากดรรชนีที่ถูกสร้างมีแอ็ตทริบิวต์เดียวกันกับดรรชนีที่มีอยู่เดิม, ดรรชนีใหม่จะใช้งาน binary tree ของดรรชนีที่มีอยู่เดิมร่วมกัน. มิฉะนั้น, binary tree อื่นจะถูกสร้างขึ้นมา. หากแอ็ตทริบิวต์ของดรรชนีใหม่เป็นอันเดียวกันกับอีกดรรชนีหนึ่ง, เว้นเสียแต่ว่าดรรชนีใหม่มีคอลัมน์น้อยลง, binary tree จะยังคงถูกสร้างขึ้นมา. มันยังคงถูกสร้างเนื่องจากคอลัมน์พิเศษ จะป้องกันดรรชนีจากการใช้งานโดยเคอร์เซอร์หรือข้อความ UPDATE ที่อัพเดต คอลัมน์พิเศษเหล่านั้น.

ดรรชนีจะถูกสร้างขึ้นด้วยการเรียงลำดับที่ทำงานอยู่ขณะที่ข้อความ CREATE INDEX ถูกรัน. การเรียงลำดับจะใช้งานกับฟิลด์แบบอักขระ SBCS ทั้งหมด, หรือ UCS-2 หรือ ฟิลด์กราฟิก UTF-16 ของดรรชนี.

หลักการที่เกี่ยวข้อง

"ลำดับการเรียง และ normalization ใน SQL" ในหน้า 121
ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

ยุทธวิธีในการสร้างดรรชนี

สิ่งอ้างอิงที่เกี่ยวข้อง

CREATE INDEX

แคตาล็อกในการออกแบบฐานข้อมูล

แคตาล็อกจะถูกสร้างขึ้นโดยอัตโนมัติเมื่อคุณสร้างแบบแผน. มีแคตาล็อกที่ใช้งานระบบซึ่งอยู่ในไลบรารี QSYS2 เสมอ.

เมื่อ SQL อ็อบเจ็กต์ถูกสร้างขึ้นมาในแบบแผน, ข้อมูลจะถูกเพิ่มเข้าไปยังทั้งตารางแคตาล็อกระบบและตารางแคตาล็อกแบบแผน. เมื่อ SQL อ็อบเจ็กต์ถูกสร้างขึ้นมาในไลบรารี, จะมีเฉพาะแคตาล็อก QSYS2 ที่ถูกอัพเดต. ตารางที่สร้างด้วย DECLARE GLOBAL TEMPORARY TABLE จะไม่ถูกเพิ่มเข้ายังแคตาล็อก.

เมื่อตัวอย่างต่อไปนี้แสดงให้เห็น, คุณสามารถแสดงผลข้อมูลแคตาล็อกได้. คุณไม่สามารถแทรก ลบ หรืออัพเดตข้อมูลแคตาล็อกได้ คุณต้องมี privilege SELECT ในมุมมองแคตาล็อกเพื่อรันตัวอย่างต่อไปนี้.

สิ่งอ้างอิงที่เกี่ยวข้อง

มุมมองแคตาล็อก DB2 for i5/OS

การรับข้อมูลแคตาล็อกเกี่ยวกับตาราง

มุมมอง SYSTABLES จะมีแถวสำหรับทุกตาราง และทุกมุมมองในแบบแผน SQL มุมมอง SYSTABLES จะให้ข้อมูลต่างๆ เช่น ชนิดของอ็อบเจ็กต์ (ตารางหรือมุมมอง), ชื่ออ็อบเจ็กต์, เจ้าของอ็อบเจ็กต์ และอ็อบเจ็กต์นั้นอยู่ในแบบแผนใด

ตัวอย่างคำสั่งต่อไปนี้จะแสดงข้อมูล สำหรับตาราง CORPDATA.DEPARTMENT:

```
SELECT *
  FROM CORPDATA.SYSTABLES
  WHERE TABLE_NAME = 'DEPARTMENT'
```

การรับข้อมูลแคตาล็อกเกี่ยวกับคอลัมน์

มุมมอง SYSCOLUMNS จะมีแถวสำหรับแต่ละคอลัมน์ ของตารางและมุมมองใน schema

ตัวอย่างคำสั่งต่อไปนี้จะแสดง ชื่อคอลัมน์ทั้งหมดในตาราง CORPDATA.DEPARTMENT:

```
SELECT *
  FROM CORPDATA.SYSCOLUMNS
  WHERE TABLE_NAME = 'DEPARTMENT'
```

ผลลัพธ์ ของตัวอย่างคำสั่งก่อนหน้านี้คือ แถวของข้อมูลสำหรับแต่ละ คอลัมน์ในตาราง

สำหรับข้อมูลที่เฉพาะเจาะจงเกี่ยวกับแต่ละคอลัมน์ ให้ระบุ คำสั่งเพื่อเลือก เช่น:

```
SELECT COLUMN_NAME, TABLE_NAME, DATA_TYPE, LENGTH, HAS_DEFAULT
  FROM CORPDATA.SYSCOLUMNS
  WHERE TABLE_NAME = 'DEPARTMENT'
```

นอกเหนือจากชื่อคอลัมน์ของแต่ละคอลัมน์, ข้อความเพื่อเลือกจะแสดง:

- ชื่อของตารางที่มีคอลัมน์อยู่
- ประเภทข้อมูลของคอลัมน์
- แอ็ตทริบิวต์ความยาวของคอลัมน์
- หากคอลัมน์อนุญาตให้มีค่าดีฟอลต์

ผลลัพธ์จะมีลักษณะเช่นนี้

COLUMN_NAME	TABLE_NAME	DATA_TYPE	LENGTH	HAS_DEFAULT
DEPTNO	DEPARTMENT	CHAR	3	N
DEPTNAME	DEPARTMENT	VARCHAR	29	N
MGRNO	DEPARTMENT	CHAR	6	Y
ADMRDEPT	DEPARTMENT	CHAR	3	N

การลบอ็อบเจ็กต์ฐานข้อมูล

คำสั่ง DROP จะลบอ็อบเจ็กต์ออก. ขึ้นอยู่กับ action ที่ร้องขอ, อ็อบเจ็กต์ใดๆ ที่ขึ้นอยู่กับอ็อบเจ็กต์นั้นทั้งทางตรงและทางอ้อม อาจถูกลบทิ้ง หรืออาจถูกป้องกันไม่ให้ถูกลบ.

ตัวอย่างเช่น, หากคุณลบตาราง, aliase ใดๆ, ข้อจำกัด, ทริกเกอร์, มุมมอง, หรือดรรชนีที่เชื่อมโยงกับตารางนั้นจะถูกลบออกด้วย. เมื่อใดก็ตามที่อ็อบเจ็กต์ถูกลบทิ้ง, รายละเอียดของอ็อบเจ็กต์นั้นจะถูกลบออกจากแคตาล็อก.

ตัวอย่างเช่น, หากต้องการลบตาราง EMPLOYEE, ให้ใช้คำสั่งต่อไปนี้:

```
DROP TABLE EMPLOYEE RESTRICT
```

สิ่งอ้างอิงที่เกี่ยวข้อง

DROP

ภาษาสำหรับการดำเนินการข้อมูล

ภาษาสำหรับการดำเนินการข้อมูล (DML) คือส่วนของ SQL ที่ดำเนินการ หรือควบคุมข้อมูล

หลักการที่เกี่ยวข้อง

"ประเภทของคำสั่ง SQL" ในหน้า 7
มีประเภทของคำสั่ง SQL พื้นฐานอยู่หลายประเภท ซึ่งคำสั่งเหล่านั้นจะถูกแสดงไว้ที่นี่ตามฟังก์ชัน

การดึงข้อมูลโดยใช้คำสั่ง SELECT

คำสั่ง SELECT ปรับแต่งเคียวรี่ของคุณเพื่อรวบรวม ข้อมูล คุณสามารถใช้คำสั่ง SELECT เพื่อดึงแถวที่เฉพาะเจาะจงหรือดึงข้อมูลในลักษณะที่เฉพาะเจาะจง

ถ้า SQL ไม่สามารถหาแถวที่ตรงกับเงื่อนไขการค้นหา, SQLCODE ที่มีค่าเป็น +100 จะถูกคืนค่ากลับมา.

ถ้า SQL เจอข้อผิดพลาดขณะรันคำสั่ง Select, SQLCODE ที่มีค่าเป็นลบจะถูกคืนค่ากลับมา. ถ้า SQL เจอตัวแปรโฮสต์ที่มากกว่าผลลัพธ์, ค่า +326 จะถูกส่งคืนกลับมา.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การสร้างตารางโดยใช้ AS" ในหน้า 23
คุณสามารถสร้างตารางจากผลลัพธ์ของคำสั่ง SELECT ในการสร้างตารางชนิดนี้ ให้ใช้คำสั่ง CREATE TABLE AS

"การสร้างและการใช้มุมมอง" ในหน้า 36
สามารถใช้มุมมองเพื่อเข้าใช้งานข้อมูลในตารางหนึ่งตารางหรือมากกว่าหรือมุมมองหนึ่งมุมมองหรือมากกว่าได้. คุณสามารถสร้างมุมมองได้โดยใช้คำสั่ง SELECT.

ข้อความ SELECT ระดับต้น

รูปแบบและไวยากรณ์ระดับต้นของคำสั่ง SELECT ประกอบด้วย clause ที่จำเป็นและที่เป็นทางเลือก

คุณสามารถเขียนคำสั่ง SQL ได้หลายบรรทัด. สำหรับคำสั่ง SQL ในโปรแกรมพรีคอมไพล์, กฎสำหรับการต่อบรรทัดที่ใช้จะเป็นกฎเดียวกับของภาษาโฮสต์ (ภาษาที่ใช้เขียนโปรแกรม). นอกจากนี้คุณสามารถเรียกใช้คำสั่ง SELECT จากเคอร์เซอร์ในโปรแกรมได้. และสุดท้าย, คำสั่ง SELECT สามารถสร้างในแอ็พพลิเคชันแบบ dynamic ได้.

หมายเหตุ:

1. คำสั่ง SQL ที่อธิบายในส่วนนี้สามารถใช้งานได้กับตาราง วิว, และไฟล์ฐานข้อมูลทั้งแบบฟิสิคัลและโลจิคัล.

2. สตริงอักขระที่ถูกระบุในคำสั่ง SQL (เช่น ค่าที่ถูกใช้ด้วย WHERE หรือ VALUES clause) จะตรงตามตัวอักษรพิมพ์ใหญ่หรือพิมพ์เล็ก นั่นคือ ตัวอักษรพิมพ์ใหญ่ต้องถูกป้อนในแบบตัวพิมพ์ใหญ่และตัวอักษรพิมพ์เล็กต้องถูกป้อนด้วยตัวพิมพ์เล็ก

   ```
   WHERE ADMRDEPT='a00'      (จะไม่คืนค่าผลลัพธ์)

   WHERE ADMRDEPT='A00'      (จะคืนค่าหมายเลขแผนกที่ถูกต้อง)
   ```

   การดำเนินการเปรียบเทียบอาจจะไม่เป็นแบบตรงตามตัวอักษรพิมพ์ใหญ่พิมพ์เล็กถ้าใช้การเรียงแบบ shared-weight ซึ่งจะถือว่าตัวอักษรพิมพ์ใหญ่และตัวอักษรพิมพ์เล็กเป็น ตัวอักขระเดียวกัน

คำสั่ง SELECT อาจประกอบด้วยสิ่งต่อไปนี้:

1. ชื่อของแต่ละคอลัมน์ที่คุณต้องการรวมในผลลัพธ์.
2. ชื่อของตารางหรือมุมมองที่มีข้อมูลอยู่.
3. เงื่อนไขการค้นหาที่ระบุแถวซึ่งมีข้อมูลที่คุณต้องการ.
4. ชื่อของแต่ละคอลัมน์ที่ใช้เพื่อจัดกลุ่มข้อมูลของคุณ.
5. เงื่อนไขการค้นหาที่ระบุกลุ่มที่มีข้อมูลที่คุณต้องการ.
6. ลำดับการเรียงผลลัพธ์เพื่อให้ส่งคืนแถวที่ต้องการซึ่งอยู่ท่ามกลางข้อมูลที่ซ้ำกัน.

คำสั่ง SELECT จะเป็นดังนี้:

```
SELECT ชื่อคอลัมน์
  FROM table or view name
  WHERE search condition
  GROUP BY column names
  HAVING search condition
  ORDER BY column-name
```

SELECT และ FROM clause จะต้องถูกระบุ. clause อื่นจะเป็นตัวเลือกว่าจะระบุหรือไม่ก็ได้.

การใช้ SELECT clause, จะทำให้คุณระบุชื่อของแต่ละคอลัมน์ที่คุณต้องการดึงค่าได้. ตัวอย่างเช่น:

```
SELECT EMPNO, LASTNAME, WORKDEPT
```

คุณสามารถระบุว่าให้ดึงข้อมูลแค่คอลัมน์เดียวเท่านั้น, หรือได้มากสุดถึง 8000 คอลัมน์. ค่าของแต่ละคอลัมน์ที่คุณเลือกจะถูกดึงข้อมูลในลำดับที่ระบุใน SELECT clause.

ถ้าคุณต้องการดึงค่าคอลัมน์ทั้งหมด (ในลำดับเดียวกับที่ปรากฏใน definition ของตาราง), ให้ใช้เครื่องหมายดอกจัน (*) แทนการใช้ชื่อคอลัมน์:

```
SELECT *
```

FROM clause จะระบุตารางที่คุณต้องการเลือกข้อมูล*จาก*. คุณสามารถเลือกคอลัมน์จากตารางมากกว่าหนึ่งตารางได้. เมื่อเรียกใช้คำสั่ง SELECT, คุณต้องระบุ FROM clause ด้วย. ใช้คำสั่งต่อไปนี้:

```
SELECT *
       FROM EMPLOYEE
```

ผลลัพธ์คือ คอลัมน์ และแถวทั้งหมดจากตาราง EMPLOYEE

รายการ SELECT อาจมีนิพจน์, ซึ่งรวมถึงค่าคงที่, เรจิสเตอร์พิเศษ, และ scalar fullselect. AS clause ยังสามารถใช้เพื่อตั้งชื่อคอลัมน์ผลลัพธ์. ตัวอย่างเช่น, ใช้คำสั่งต่อไปนี้:

```
SELECT LASTNAME, SALARY * .05 AS RAISE
       FROM EMPLOYEE
       WHERE EMPNO = '200140'
```

ผลลัพธ์ของคำสั่งนี้คือ

ตารางที่ 7. ผลลัพธ์สำหรับการสืบค้น

LASTNAME	RAISE
NATZ	1421

การระบุเงื่อนไขการค้นหาโดยใช้ WHERE clause

WHERE clause จะระบุเงื่อนไขการค้นหา ที่บ่งชี้ถึงแถวที่คุณต้องการดึงค่า, อัพเดต หรือลบ

จำนวนแถวที่คุณประมวลผลด้วยคำสั่ง SQL จะขึ้นอยู่กับจำนวนแถวที่ตรงกับ **เงื่อนไขการค้นหา** ของ WHERE clause. เงื่อนไขการค้นหาประกอบด้วย**เพรดิเคต**หนึ่งตัวขึ้นไป. เพรดิเคตจะระบุการทดสอบที่คุณต้องการให้ SQL ใช้กับแถวในตารางที่ระบุ.

ในตัวอย่างต่อไปนี้, WORKDEPT = 'C01' คือเพรดิเคต, WORKDEPT และ 'C01' คือนิพจน์, และเครื่องหมายเท่ากับ (=) คือ comparison operator. โปรดสังเกตว่าค่าตัวอักขระจะถูกล้อมด้วย apostrophe (') ; ส่วนค่าตัวเลขจะไม่ถูกล้อมด้วยค่านี้. วิธีการนี้จะใช้กับค่าคงที่ทั้งหมดที่ถูกโค้ดภายในคำสั่ง SQL. ตัวอย่างเช่น, เมื่อต้องการระบุว่า คุณสนใจแถวที่มีหมายเลขแผนกคือ C01, ให้ใช้คำสั่งต่อไปนี้:

```
... WHERE WORKDEPT = 'C01'
```

ในกรณีนี้, เงื่อนไขการค้นหาประกอบด้วยเพรดิเคตหนึ่งตัว คือ: WORKDEPT = 'C01'.

เพื่อแสดงการใช้ WHERE ต่อไป, ให้พิจรณาเมื่อใช้กับคำสั่ง SELECT. สมมติว่าแต่ละแผนกที่อยู่ในตาราง CORPDATA. DEPARTMENT มีหมายเลขแผนกไม่ซ้ำกัน. คุณต้องการดึงค่าชื่อแผนกและหมายเลขผู้จัดการจากตาราง CORPDATA. DEPARTMENT สำหรับแผนก C01. ใช้คำสั่งต่อไปนี้:

```
SELECT DEPTNAME, MGRNO
       FROM CORPDATA.DEPARTMENT
       WHERE DEPTNO = 'C01'
```

ผลลัพธ์ของคำสั่งนี้คือหนึ่งแถว

ตารางที่ 8. ตารางผลลัพธ์

DEPTNAME	MGRNO
INFORMATION CENTER	000030

ถ้าเงื่อนไขการค้นหามีอักขระ, หรือเพรดิเคตคอลัมน์ที่เป็นกราฟิกแบบ UCS-2 หรือ UTF-16 , ลำดับการเรียงของผลลัพธ์การทำเคียวรี่จะส่งผลต่อค่าเพรดิเคตด้วย. ถ้าไม่มีการใช้ลำดับการเรียง, ค่าคงที่ที่เป็นตัวอักษรจะต้องระบุในแบบตัวพิมพ์ใหญ่หรือตัวพิมพ์เล็กเพื่อให้ตรงกับคอลัมน์หรือประโยคที่ค่าเหล่านั้นกำลังเปรียบเทียบอยู่.

หลักการที่เกี่ยวข้อง

"ลำดับการเรียง และ normalization ใน SQL" ในหน้า 121
ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การนิยามเงื่อนไขการค้นหาที่ซับซ้อน" ในหน้า 61
นอกจากเพรดิเคตการเปรียบเทียบพื้นฐานแล้ว เช่น = และ > เงื่อนไขการค้นหาสามารถมีเพรดิเคต BETWEEN, IN, EXISTS, IS NULL และ LIKE

"หลายเงื่อนไขการค้นหาภายใน WHERE clause" ในหน้า 63
คุณสามารถทำการร้องขอเพิ่มเติม โดยโค้ดเงื่อนไขการค้นหาที่ประกอบด้วยหลายเพรดิเคต.

นิพจน์ใน WHERE clause:

นิพจน์ใน WHERE clause ใช้ตั้งชื่อหรือระบุ สิ่งที่คุณต้องการเปรียบเทียบกับสิ่งอื่น

นิพจน์ที่คุณระบุสามารถเป็น:

- **ชื่อคอลัมน์**คือการระบุคอลัมน์. ตัวอย่างเช่น:

  ```
  ... WHERE EMPNO = '000200'
  ```

  EMPNO ระบุคอลัมน์ที่ถูกนิยามด้วยค่าอักขระชนิด 6-ไบต์.
- **นิพจน์** ระบุค่าสองค่าที่จะถูกเพิ่ม (+), ลบ (-), คูณ (*), หาร (/), ยกกำลัง (**), หรือเชื่อมต่อ (CONCAT หรือ ||) กับผลลัพธ์ในค่า. operand พื้นฐานของนิพจน์คือ:
  - ค่าคงที่
  - คอลัมน์
  - ตัวแปรโฮสต์
  - ฟังก์ชัน
  - register พิเศษ
  - scalar fullselect
  - นิพจน์อื่น

  ตัวอย่างเช่น:

  ```
  ... WHERE INTEGER(PRENDATE - PRSTDATE) > 100
  ```

  เมื่อลำดับของการประเมินผลไม่ได้ระบุโดยเครื่องหมายวงเล็บแล้ว, นิพจน์จะถูกประเมินผลในลำดับดังต่อไปนี้:
  1. โอเปอเรเตอร์ prefix
  2. การยกกำลัง
  3. การคูณ, การหาร, และการเชื่อมต่อ
  4. การบวกและการลบ

  โอเปอเรเตอร์ที่อยู่ระดับเดียวกันจะทำงานจากซ้ายมาขวา.
- **ค่าคงที่**จะระบุค่า literal สำหรับนิพจน์. ตัวอย่างเช่น:

  ```
  ... WHERE 40000 < SALARY
  ```

SALARY ระบุ คอลัมน์ที่ถูกนิยามเป็นค่าทศนิยมแบบ 9 หลัก (DECIMAL(9,2)) และจะถูกเปรียบเทียบค่ากับค่าคงที่ 40000.

- **ตัวแปรโฮสต์**ระบุตัวแปรในแอ็พพลิเคชันโปรแกรม. ตัวอย่างเช่น:

  ```
  ... WHERE EMPNO = :EMP
  ```

- **register พิเศษ**ระบุค่าพิเศษที่นิยามโดยผู้จัดการฐานข้อมูล. ตัวอย่างเช่น:

  ```
  ... WHERE LASTNAME = USER
  ```

- ค่า **NULL** ระบุค่าที่ไม่ทราบค่า.

  ```
  ... WHERE DUE_DATE IS NULL
  ```

- scalar fullselect.

เงื่อนไขการค้นหาสามารถระบุเพรดิเคตหลายตัวที่แยกโดย AND และ OR. โดยไม่ว่าเงื่อนไขการค้นหาจะซับซ้อนอย่างไร, มันจะให้ค่า TRUE หรือ FALSE เมื่อทำการประเมินผลกับแถว. และยังมีค่าความจริงเป็น *Unknown*, ซึ่งมีผลเป็น False. นั่นคือ, ถ้าค่าของแถวเป็น null แล้ว, ค่า null นี้จะไม่ถูกส่งคืนค่าเป็นผลลัพธ์ของการค้นหาเพราะว่าค่านี้ไม่ใช่คาที่น้อยกว่า, เท่ากับ, หรือมากกว่าค่าที่ระบุในเงื่อนไขการค้นหา.

เพื่อให้เข้าใจการใช้ WHERE clause, คุณจำเป็นต้องรู้ลำดับที่ SQL ทำการประเมินผลเงื่อนไขการค้นหาและเพรดิเคต, และวิธีที่ SQL เปรียบเทียบค่าของนิพจน์. หัวข้อนี้จะอธิบายไว้ในกลุ่มหัวข้อ DB2 for i5/OS การอ้างอิง SQL

หลักการที่เกี่ยวข้อง

"การใช้เคียวรี่ย่อย" ในหน้า 112
คุณสามารถใช้เคียวรี่ย่อยในเงื่อนไขการค้นหาเพื่อเป็นอีกทางหนึ่งในการ เลือกข้อมูล เคียวรี่ย่อยสามารถใช้ได้ทุกที่ที่นิพจน์สามารถใช้งานได้.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การนิยามเงื่อนไขการค้นหาที่ซับซ้อน" ในหน้า 61
นอกจากเพรดิเคตการเปรียบเทียบพื้นฐานแล้ว เช่น = และ > เงื่อนไขการค้นหาสามารถมีเพรดิเคต BETWEEN, IN, EXISTS, IS NULL และ LIKE

นิพจน์

ตัวดำเนินการเปรียบเทียบ:

SQL สนับสนุน comparison operator เหล่านี้

Comparison operator	รายละเอียด
<> or ¬= or !=	Not equal to
=	Equal to
<	Less than
>	Greater than
<= or ¬> or !>	Less than or equal to (or not greater than)
>= or ¬< or !<	Greater than or equal to (or not less than)

NOT keyword:

คุณสามารถเติมหน้าเพรดิเคตด้วยคีย์เวิร์ด NOT เพื่อระบุว่า คุณต้องการค่าตรงกันข้ามกับค่าของเพรดิเคต (นั่นคือ, TRUE ถ้าเพรดิเคตคือ FALSE)

NOT จะใช้กับเพรดิเคตที่อยู่ต่อจากมันเท่านั้น, ไม่ได้ใช้กับเพรดิเคตทั้งหมดใน WHERE clause. ตัวอย่างเช่น เมื่อต้องการระบุว่าคุณสนใจในพนักงานทั้งหมดยกเว้นพนักงานที่ทำงานในแผนก C01 คุณอาจจะใช้คำสั่ง:

```
... WHERE NOT WORKDEPT = 'C01'
```

ซึ่งจะเท่ากันกับ:

```
... WHERE WORKDEPT <> 'C01'
```

GROUP BY clause

GROUP BY clause อนุญาตให้คุณค้นหาคุณลักษณะของกลุ่มของแถวมากกว่าที่จะค้นหาแถวเดี่ยว.

เมื่อคุณระบุ GROUP BY clause แล้ว, SQL จะแบ่งแถวที่เลือกให้เป็นกลุ่ม ซึ่งแถวของแต่ละกลุ่มจะมีค่าที่ตรงกับค่าในคอลัมน์หรือนิพจน์. ต่อมา, SQL จะดำเนินการกับแต่ละกลุ่มเพื่อสร้างผลลัพธ์แบบแถวเดียวให้กลุ่ม. คุณสามารถระบุคอลัมน์หรือนิพจน์ได้มากกว่าหนึ่งใน GROUP BY clause เพื่อจัดกลุ่มแถว. รายการที่คุณระบุในคำสั่ง SELECT จะเป็นคุณสมบัติของแต่ละกลุ่มของแถว, ไม่ใช่คุณสมบัติของแถวเดี่ยวในตารางหรือมุมมอง.

เมื่อไม่มี GROUP BY clause, แอ็พพลิเคชันของฟังก์ชันโดยรวมของ SQL จะคืนค่า*หนึ่ง*แถว. เมื่อ GROUP BY ถูกใช้, ฟังก์ชันจะถูกใช้กับ*แต่ละ*กลุ่ม, ดังนั้นจะคืนค่าแถวมามากเท่ากับจำนวนกลุ่มอที่มีอยู่.

ตัวอย่างเช่น, ตาราง CORPDATA.EMPLOYEE มีแถวอยู่หลายชุด, และแต่ละชุดจะมีแถวที่อธิบายข้อมูลสมาชิกของแต่ละแผนก. เมื่อต้องการหาค่าเฉลี่ยเงินเดือนของพนักงานในแต่ละแผนก, คุณสามารถใช้คำสั่ง:

```
SELECT WORKDEPT, DECIMAL (AVG(SALARY),5,0)
       FROM CORPDATA.EMPLOYEE
       GROUP BY WORKDEPT
```

ผลลัพธ์จะมีหลายแถว, หนึ่งแถวสำหรับแต่ละแผนก.

WORKDEPT	AVG-SALARY
A00	40850
B01	41250
C01	29722
D11	25147
D21	25668
E01	40175
E11	21020
E21	24086

หมายเหตุ:

1. การจัดกลุ่มแถวไม่ได้หมายความว่าจะเรียงลำดับแถวเหล่านั้นด้วย. การจัดกลุ่มจะดึงแถวที่เลือกเข้ามา , เพื่อให้ SQL จัดการประมวลผลหาค่าความเป็นลักษณะเฉพาะออกมา. การเรียงลำดับแถวจะใส่แถวทั้งหมดเข้าไปในตารางผลลัพธ์ด้วยลำดับการเรียงแบบจากน้อยไปมากหรือจากมากไปน้อย. กลุ่มของผลลัพธ์อาจจะเรียงลำดับก็ได้ ทั้งนี้ขึ้นอยู่กับวิธีปฏิบัติที่ผู้จัดการฐานข้อมูลเลือก
2. ถ้ามีค่า null ในคอลัมน์ที่คุณระบุใน GROUP BY clause, ผลลัพธ์แบบแถวเดียวจะถูกสร้างเป็นข้อมูลในแถวด้วยค่า null.
3. ถ้ามีการจัดกลุ่มกับอักขระ, หรือคอลัมน์กราฟิกประเภท USC-2 หรือ UTF-16, การเรียงลำดับที่เกิดจากการรันเคียวรี่จะมีผลต่อการทำกลุ่มด้วย.

เมื่อคุณใช้GROUP BY, รบบจะแสดงรายชื่อคอลัมน์หรือนิพจน์ที่คุณต้องการให้ SQL ใช้ในการจัดกลุ่มแถว. ตัวอย่างเช่น สมมติว่าคุณต้องการรายการจำนวนคนที่ทำงานในแต่ละโครงการสำคัญซึ่งอธิบายไว้ในตาราง CORPDATA.PROJECT คุณสามารถใช้คำสั่ง:

```
SELECT SUM(PRSTAFF), MAJPROJ
       FROM CORPDATA.PROJECT
       GROUP BY MAJPROJ
```

ผลลัพธ์คือ รายการโครงการสำคัญปัจจุบันของบริษัท และจำนวนคนที่ทำงานในแต่ละโครงการ

SUM(PRSTAFF)	MAJPROJ
6	AD3100
5	AD3110
10	MA2100
8	MA2110
5	OP1000
4	OP2000
3	OP2010
32.5	?

คุณยังสามารถระบุว่าคุณต้องการแถวที่จัดกลุ่มโดยคอลัมน์หรือนิพจน์มากกว่าหนึ่งค่าได้. ตัวอย่างเช่น คุณอาจใช้คำสั่ง Select เพื่อหาค่าเฉลี่ยเงินเดือนสำหรับผู้ชายและผู้หญิงในแต่ละแผนก โดยใช้ตาราง CORPDATA.EMPLOYEE เมื่อต้องการทำเช่นนี้, ให้คุณระบุ:

```
SELECT WORKDEPT, SEX, DECIMAL(AVG(SALARY),5,0) AS AVG_WAGES
       FROM CORPDATA.EMPLOYEE
       GROUP BY WORKDEPT, SEX
```

ผลลัพธ์เป็นดังนี้

WORKDEPT	SEX	AVG_WAGES
A00	F	49625

WORKDEPT	SEX	AVG_WAGES
A00	M	35000
B01	M	41250
C01	F	29722
D11	F	25817
D11	M	24764
D21	F	26933
D21	M	24720
E01	M	40175
E11	F	22810
E11	M	16545
E21	F	25370
E21	M	23830

เนื่องจากคุณไม่ได้รวม WHERE clause เข้าไปในตัวอย่างนี้ SQL จึงตรวจสอบและดำเนินการกับทุกแถวในตาราง CORPDATA.EMPLOYEE แถวจะถูกจัดกลุ่มโดยหมายเลขแผนกก่อนแล้วต่อด้วยเพศ (ภายในแต่ละแผนก) ก่อนที่ SQL จะหาค่า SALARY เฉลี่ยสำหรับแต่ละกลุ่ม.

หลักการที่เกี่ยวข้อง

"ลำดับการเรียง และ normalization ใน SQL" ในหน้า 121
ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

สิ่งอ้างอิงที่เกี่ยวข้อง

"ORDER BY clause" ในหน้า 52
ORDER BY clause ระบุลำดับเฉพาะที่ คุณต้องการส่งคืนแถวที่เลือกไว้ ลำดับจะถูกเรียงลำดับการเรียงของค่าคอลัมน์หรือค่านิพจน์จากน้อยไปมาก หรือจากมากไปน้อย

HAVING clause

HAVING clause ระบุเงื่อนไขการค้นหา สำหรับกลุ่มที่เลือกได้โดยใช้ GROUP BY clause

HAVING clause เป็นการบอกว่าคุณต้องการ*เฉพาะ*กลุ่มที่เป็นไปตามเงื่อนไขใน clause นั้น. ดังนั้นแล้ว, เงื่อนไขการค้นหาที่คุณระบุใน HAVING clauseจะต้องทดสอบคุณสมบัติของแต่ละกลุ่มมากกว่าที่จะทดสอบคุณสมบัติของแถวเดี่ยวในกลุ่ม.

HAVING clause จะอยู่ต่อจาก GROUP BY clause และ สามารถประกอบด้วยประเภทของเงื่อนไขการค้นหาเดียวกันกับที่คุณสามารถระบุใน WHERE clause นอกเหนือจากนั้น, คุณสามารถระบุฟังก์ชันการรวมใน HAVING clause. ตัวอย่างเช่น สมมติว่าคุณต้องการดึงค่าเงินเดือนเฉลี่ยของผู้หญิงในแต่ละแผนก หากต้องการทำสิ่งนี้, ให้ใช้ฟังก์ชันการรวม AVG และจัดกลุ่มแถวที่ได้ด้วย WORKDEPT และระบุ WHERE clause โดยที่กำหนด SEX = 'F'.

เมื่อต้องการระบุว่าคุณต้องการข้อมูลนี้เฉพาะเมื่อพนักงานผู้หญิงทั้งหมดในแผนกที่ถูกเลือกมีระดับการศึกษาเท่ากับหรือมากกว่า 16 (การศึกษาระดับวิทยาลัย), ให้ใช้ HAVING clause. HAVING clause จะทดสอบคุณสมบัติของกลุ่ม. ในกรณีนี้, การทดสอบคือ MIN(EDLEVEL), ซึ่งก็คือคุณสมบัติของกลุ่ม:

```
SELECT WORKDEPT, DECIMAL(AVG(SALARY),5,0) AS AVG_WAGES, MIN(EDLEVEL) AS MIN_EDUC
       FROM CORPDATA.EMPLOYEE
       WHERE SEX='F'
       GROUP BY WORKDEPT
       HAVING MIN(EDLEVEL)>=16
```

ผลลัพธ์เป็นดังนี้

WORKDEPT	AVG_WAGES	MIN_EDUC
A00	49625	18
C01	29722	16
D11	25817	17

คุณสามารถใช้เพรดิเคตหลายตัวใน HAVING clause ได้โดยเชื่อมโยงค่าเหล่านั้นด้วย AND และ OR, และคุณสามารถใช้ NOT สำหรับเพรดิเคตใดๆ ของเงื่อนไขการค้นหา.

หมายเหตุ: ถ้าคุณตั้งใจอัพเดตคอลัมน์หรือลบแถว, คุณไม่สามารถรวม GROUP BY clause หรือ HAVING clause เข้าไปในคำสั่ง SELECT ภายในคำสั่ง DECLARE CURSOR ได้. clause เหล่านี้ทำให้เคอร์เซอร์เป็นแบบอ่านได้อย่างเดียว.

เพรดิเคตที่มีอากิวเมนต์ที่ไม่ใช่ฟังก์ชันรวมสามารถโค้ดใน WHERE หรือ HAVING clause. ปกติการโค้ดเงื่อนไขการเลือกไว้ใน WHERE clause จะทำให้การค้นหามีประสิทธิภาพมากขึ้นเนื่องจากเงื่อนไขจะถูกจัดการก่อนในกระบวนการเคียวรี่. การเลือก HAVING จะถูกทำในขั้นตอนหลังประมวลผลตารางผลลัพธ์.

ถ้าเงื่อนไขการค้นหามีอักขระ, หรือเพรดิเคตคอลัมน์ที่เป็นกราฟิกแบบ UCS-2 หรือ UTF-16 , ลำดับการเรียงของผลลัพธ์การทำเคียวรี่จะส่งผลต่อค่าเพรดิเคตด้วย.

หลักการที่เกี่ยวข้อง

"ลำดับการเรียง และ normalization ใน SQL" ในหน้า 121
ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การใช้เคอร์เซอร์" ในหน้า 252
เมื่อ SQL รันคำสั่ง SELECT แถวผลลัพธ์ จะประกอบขึ้นจากตารางผลลัพธ์ เคอร์เซอร์จะแสดงวิธีการเข้าถึงตารางผลลัพธ์

ORDER BY clause

ORDER BY clause ระบุลำดับเฉพาะที่ คุณต้องการส่งคืนแถวที่เลือกไว้ ลำดับจะถูกเรียงลำดับการเรียงของค่าคอลัมน์ หรือค่านิพจน์จากน้อยไปมาก หรือจากมากไปน้อย

ตัวอย่างเช่น, เมื่อต้องการดึงชื่อและหมายเลขแผนกของพนักงานหญิงโดยให้หมายเลขแผนกเรียงตามลำดับตัวอักษร, คุณควรใช้คำสั่ง Select ดังนี้:

```
SELECT LASTNAME,WORKDEPT
       FROM CORPDATA.EMPLOYEE
       WHERE SEX='F'
       ORDER BY WORKDEPT
```

ผลลัพธ์เป็นดังนี้

LASTNAME	WORKDEPT
HAAS	A00
HEMMINGER	A00
KWAN	C01
QUINTANA	C01
NICHOLLS	C01
NATZ	C01
PIANKA	D11
SCOUTTEN	D11
LUTZ	D11
JOHN	D11
PULASKI	D21
JOHNSON	D21
PEREZ	D21
HENDERSON	E11
SCHNEIDER	E11
SETRIGHT	D11
SCHWARTZ	E11
SPRINGER	E11
WONG	E21

หมายเหตุ: ค่า null จะถูกเรียงลำดับเป็นค่าที่สูงที่สุด.

คอลัมน์ที่ถูกระบุใน ORDER BY clause ไม่จำเป็นต้องถูกรวมเข้าไปใน SELECT clause. ตัวอย่างเช่น, คำสั่งต่อไปนี้จะคืนค่าพนักงานหญิงทั้งหมด ซึ่งเรียงลำดับด้วยเงินเดือนที่มากที่สุดตามลำดับ:

```
SELECT LASTNAME,FIRSTNME
       FROM CORPDATA.EMPLOYEE
       WHERE SEX='F'
       ORDER BY SALARY DESC
```

ถ้า AS clause ถูกระบุเพื่อตั้งชื่อคอลัมน์ผลลัพธ์ในรายการที่เลือก, ชื่อนี้สามารถถูกระบุใน ORDER BY clause ได้. ชื่อที่ระบุใน AS clause จะต้องไม่ซ้ำกันในรายการที่เลือก. ตัวอย่างเช่น ถ้าต้องการดึงชื่อเต็มของพนักงานให้เรียงลำดับตามตัวอักษร คุณสามารถใช้คำสั่ง Select ต่อไปนี้:

```
SELECT LASTNAME CONCAT FIRSTNME AS FULLNAME
  FROM CORPDATA.EMPLOYEE
  ORDER BY FULLNAME
```

คำสั่ง select นี้อาจเขียนได้เป็น:

```
SELECT LASTNAME CONCAT FIRSTNME
  FROM CORPDATA.EMPLOYEE
  ORDER BY LASTNAME CONCAT FIRSTNME
```

แทนที่จะตั้งชื่อคอลัมน์เพื่อเรียงลำดับผลลัพธ์, คุณสามารถใช้หมายเลขได้. ตัวอย่างเช่น, ORDER BY 3 จะระบุว่า คุณต้องการผลลัพธ์ที่ถูกเรียงลำดับโดยคอลัมน์ *ที่สาม*ของตารางผลลัพธ์, ที่ระบุโดยรายการที่เลือก. ใช้ตัวเลขเพื่อเรียงลำดับแถวของตารางผลลัพธ์เมื่อค่าที่เรียงเป็นคอลัมน์ที่ไม่มีชื่อ.

คุณยังสามารถระบุว่าคุณต้องการให้ SQL เรียงลำดับแถวในลำดับจากน้อยไปมาก (ASC) หรือจากมากไปน้อย (DESC). การเรียงลำดับจากน้อยไปมากจะเป็นค่าดีฟอลต์. ในคำสั่ง Select ก่อนหน้านี้, SQL จะคืนค่าแถวด้วยนิพจน์ *FULLNAME* ที่ต่ำที่สุดก่อน (ตามตัวอักษรและตัวเลข), ตามด้วยแถวที่มีค่ามากกว่า. เมื่อต้องการเรียงลำดับแถวในลำดับการเรียงจากมากไปน้อยโดยยึดจากชื่อนี้, ให้ระบุ:

```
... ORDER BY FULLNAME DESC
```

คุณสามารถระบุลำดับการเรียงสำรอง (หรือลำดับการเรียงอีกหลายระดับ) และลำดับการเรียงแรกได้. ในตัวอย่างก่อนหน้านี้, คุณอาจต้องการให้แถวเรียงลำดับโดยหมายเลขแผนกก่อน, และภายในแต่ละแผนก, ให้เรียงลำดับโดยชื่อพนักงาน. เมื่อต้องการทำเช่นนี้, ให้ระบุ:

```
... ORDER BY WORKDEPT, FULLNAME
```

ถ้าคอลัมน์แบบตัวอักขระ, หรือคอลัมน์แบบกราฟิก UCS-2 ถูกใช้ใน ORDER BY แล้ว, การเรียงลำดับของคอลัมน์เหล่านี้จะขึ้นกับการจัดลำดับที่กำหนดในการรันเคียวรี่.

หลักการที่เกี่ยวข้อง

"ลำดับการเรียง และ normalization ใน SQL" ในหน้า 121
ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

สิ่งอ้างอิงที่เกี่ยวข้อง

"GROUP BY clause" ในหน้า 49
GROUP BY clause อนุญาตให้คุณค้นหาคุณลักษณะของกลุ่มของแถวมากกว่าที่จะค้นหาแถวเดี่ยว.

คำสั่ง SELECT แบบ Static

สำหรับคำสั่ง SELECT แบบ static (ซึ่งฝังตัวอยู่ในโปรแกรม SQL), INTO clause จะต้องระบุไว้หน้า FROM clause.

INTO clause จะตั้งชื่อตัวแปรโฮสต์ (ตัวแปรในโปรแกรมของคุณซึ่งถูกใช้เพื่อเก็บค่าคอลัมน์ที่ดึงค่ามา). ค่าของคอลัมน์ผลลัพธ์แรกที่ระบุใน SELECT clause จะถูกใส่ค่าเข้าไปในตัวแปรโฮสต์ตัวแรกที่มีชื่อในค INTO clause; ค่าที่สองจะถูกใส่ค่าเข้าไปในตัวแปรโฮสต์ตัวที่สอง, เช่นนี้ไปเรื่อยๆ .

ตารางผลลัพธ์สำหรับ SELECT INTO ควรมีแค่หนึ่งแถวเท่านั้น. ตัวอย่างเช่น, แต่ละแถวในตาราง CORPDATA.EMPLOYEE จะมีคอลัมน์ EMPNO (หมายเลขพนักงาน) ที่ไม่ซ้ำกัน. ผลลัพธ์ของคำสั่ง SELECT INTO สำหรับตารางนี้ ถ้า WHERE clause มีการเปรียบเทียบแบบเท่ากันกับคอลัมน์ EMPNO ควรจะมี แถวเดียวเท่านั้น (หรือไม่มีแถวเลย) หากผลการค้นหามีมากกว่าหนึ่งแถวแสดงว่ามีข้อผิดพลาด, แต่ว่าจะมีแถวหนึ่งที่ถูกคืนค่ามา. คุณสามารถควบคุมว่าแถวไหนที่จะถูกคืนค่ามาในสภาวะที่ผิดพลาดเช่นนี้โดยการใช้ ORDER BY clause. ถ้าคุณใช้ ORDER BY clause, แถวแรกในตารางผลลัพธ์จะถูกคืนค่ากลับมา.

ถ้าคุณต้องการผลลัพธ์ของคำสั่ง SELECT INTO มากกว่าหนึ่งแถว, ให้ใช้คำสั่ง DECLARE CURSOR เพื่อเลือกแถว, แล้วจึงตามด้วยคำสั่ง FETCH เพื่อย้ายค่าคอลัมน์ไปไว้ในตัวแปรโฮสต์ครั้งละหนึ่งแถวขึ้นไป.

เมื่อใช้คำสั่ง Select ในแอ็พพลิเคชันโปรแกรม, ให้ทำรายชื่อคอลัมน์เพื่อทำให้โปรแกรมของคุณมีข้อมูลที่เป็นอิสระมากขึ้น. การทำเช่นนี้มีเหตุผล 2 ประการด้วยกัน:

1. เมื่อคุณดูคำสั่งในซอร์สโค้ด, คุณจะเห็นชื่อคอลัมน์ใน SELECT clauseและตัวแปรโฮสต์ที่มีชื่อใน INTO clause จะตรงกันแบบรายการต่อรายการ.
2. ถ้าคอลัมน์ถูกเพิ่มเข้าไปในตารางหรือมุมมองที่คุณเข้าถึง และคุณใช้ "SELECT * ...," และคุณสร้างโปรแกรมอีกครั้งจากต้นฉบับนี้, INTO clause จะไม่มีชื่อตัวแปรโฮสต์ที่ตรงกับคอลัมน์ใหม่. คอลัมน์พิเศษนี้จะเป็นเหตุให้คุณได้รับการเตือน (ไม่ใช่ข้อผิดพลา) ใน SQLCA (SQLWARN3 จะมีค่า "W"). เมื่อใช้คำสั่ง GET DIAGNOSTICS , RETURNED_SQLSTATE จะมีค่าเท่ากับ '01503'.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การใช้เคอร์เซอร์" ในหน้า 252
เมื่อ SQL รันคำสั่ง SELECT แถวผลลัพธ์ จะประกอบขึ้นจากตารางผลลัพธ์ เคอร์เซอร์จะแสดงวิธีการเข้าถึงตารางผลลัพธ์

การจัดการค่า null

ค่า null จะระบุว่าไม่มีค่าคอลัมน์ใน แถว ค่า null ไม่ใช่ค่าเดียวกับค่าศูนย์หรือ ว่างเปล่า

เราสามารถใช้ค่า null เป็นเงื่อนไขใน WHERE และ HAVING clause ได้. ตัวอย่างเช่น, WHERE clause สามารถระบุคอลัมน์ที่, เก็บค่า null อยู่, ในบางแถว. การเปรียบเทียบเพรดิเคตขั้นพื้นฐานโดยใช้คอลัมน์ที่เก็บค่า null จะไม่เลือกแถวที่คอลัมน์มีค่า null. ที่เป็นเช่นนั้นเนื่องจากค่า null จะไม่น้อยกว่า, เท่ากับ หรือมากกว่าค่าที่ระบุในเงื่อนไข เพรดิเคต IS NULL จะถูกใช้เพื่อตรวจสอบค่า null. เมื่อต้องการเลือกค่าสำหรับทุกแถวที่มีค่าหมายเลขผู้จัดการเป็น null, คุณอาจระบุ:

```
SELECT DEPTNO, DEPTNAME, ADMRDEPT
  FROM CORPDATA.DEPARTMENT
  WHERE MGRNO IS NULL
```

ผลลัพธ์เป็นดังนี้

DEPTNO	DEPTNAME	ADMRDEPT
D01	DEVELOPMENT CENTER	A00
F22	BRANCH OFFICE F2	E01

DEPTNO	DEPTNAME	ADMRDEPT
G22	BRANCH OFFICE G2	E01
H22	BRANCH OFFICE H2	E01
I22	BRANCH OFFICE I2	E01
J22	BRANCH OFFICE J2	E01

เมื่อต้องการหาแถวที่หมายเลขของผู้จัดการไม่เป็น null , คุณควรเปลี่ยน WHERE clause ให้เป็นดังนี้:

```
WHERE MGRNO IS NOT NULL
```

DISTINCT เป็นเพรดิเคตที่มีประโยชน์สามารถในการเปรียบเทียบค่าที่เป็น null ได้. การเปรียบเทียบสองคอลัมน์ด้วยเครื่องหมายเท่ากับ (COL1 = COL2) จะให้ค่าเป็น true ถ้าทั้งสองคอลัมน์มีค่าเท่ากันและไม่เป็น null . แต่ถ้าทั้งสองคอลัมน์มีค่าเป็น null ผลที่ได้จะเป็น false เพราะค่า null จะไม่เท่ากับค่าใดทั้งนั้น แม้กระทั่ง null ด้วยกัน สำหรับเพรดิเคต DISTINCT , ค่า null จะถือว่าเท่ากัน. ดังนั้นประโยค COL1 is NOT DISTINCT from COL2 จะให้ผลเป็น true เมื่อทั้งสองคอลัมน์มีค่าเท่ากันทั้งในกรณีที่ทั้งคู่ไม่ใช่ค่า null หรือมีค่าเป็น null

ตัวอย่างเช่น สมมติว่า คุณต้องการเลือกข้อมูลจากสองตารางที่มีค่า null ตารางแรก T1 มีคอลัมน์ C1 ที่มีค่าต่อไปนี้

C1
2
1
null

ตารางที่สอง T2 มีคอลัมน์ C2 ที่มีค่าต่อไปนี้

C2
2
null

รันคำสั่ง SELECT ต่อไปนี้:

```
SELECT *
    FROM T1, T2
    WHERE C1 IS DISTINCT FROM C2
```

ผลลัพธ์เป็นดังนี้

C1	C2
1	2
1	-

C1	C2
2	-
-	2

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการใช้ค่า null โปรดดูกลุ่มหัวข้อ DB2 for i5/OS การอ้างอิง SQL

เรจิสเตอร์พิเศษในคำสั่ง SQL

คุณสามารถระบุเรจิสเตอร์พิเศษในคำสั่ง SQL ได้ *เรจิสเตอร์พิเศษ* เช่น CURRENT DATE ประกอบด้วย ข้อมูลที่สามารถอ้างถึงได้ในคำสั่ง SQL

สำหรับคำสั่ง SQL ที่รันแบบโลคัล เรจิสเตอร์พิเศษ และเนื้อหาจะแสดงอยู่ในตารางต่อไปนี้

เรจิสเตอร์พิเศษ	เนื้อหา
CURRENT CLIENT_ACCTNG	สตริงบัญชีผู้ใช้สำหรับการเชื่อมต่อไคลเอ็นต์
CURRENT CLIENT_APPLNAME	ชื่อแอ็พพลิเคชันสำหรับการเชื่อมต่อไคลเอ็นต์
CURRENT CLIENT_PROGRAMID	ID โปรแกรมสำหรับการเชื่อมต่อไคลเอ็นต์
CURRENT CLIENT_USERID	ID ID ผู้ใช้ไคลเอ็นต์สำหรับการเชื่อมต่อไคลเอ็นต์
CURRENT CLIENT_WRKSTNNAME	ชื่อเวิร์กสเตชันสำหรับการเชื่อมต่อไคลเอ็นต์
CURRENT DATE CURRENT_DATE	วันที่ปัจจุบัน.
CURRENT DEBUG MODE	ดีบักโหมดที่จะใช้เมื่อสร้างหรือปรับเปลี่ยน รูทีน
CURRENT DECFLOAT ROUNDING MODE	วิธีการปัดเศษที่จะใช้เมื่อทำงานกับค่า ทศนิยม
CURRENT DEGREE	จำนวนของงานที่ตัวจัดการฐานข้อมูลควรรันแบบขนาน
CURRENT PATH CURRENT_PATH CURRENT FUNCTION PATH	พาธ SQL ที่ใช้ resolve ชื่อของชนิดข้อมูล, ชื่อโพรซีเดอร์, และชื่อฟังก์ชันที่ไม่ครบตามเกณฑ์ในคำสั่ง SQL ที่เตรียมไว้แบบ dynamic.
CURRENT SCHEMA	ชื่อแบบแผนที่ใช้เพื่อทำให้อ้างอิงอ็อบเจ็กต์ฐานข้อมูลครบตามเกณฑ์ ซึ่งสามารถใช้ได้ในคำสั่ง SQL ที่เตรียมไว้แบบ dynamic.
CURRENT SERVER CURRENT_SERVER	ชื่อของฐานข้อมูลเชิงสัมพันธ์ที่ถูกใช้อยู่ขณะนี้.
CURRENT TIME CURRENT_TIME	เวลาปัจจุบัน.
CURRENT TIMESTAMP CURRENT_TIMESTAMP	วันที่และเวลาปัจจุบันที่อยู่ในรูปของ timestamp.

เรจิสเตอร์พิเศษ	เนื้อหา
CURRENT TIMEZONE CURRENT_TIMEZONE	ช่วงระยะเวลาที่เชื่อมโยงเวลาท้องถิ่นเข้ากับ Universal Time Coordinated (UTC) โดยใช้สูตร: เวลาท้องถิ่น - CURRENT TIMEZONE = UTC ซึ่งจะถูกนำมาจากค่ากำหนดของระบบ QUTCOFFSET.
SESSION_USER USER	ตัวบ่งชี้การให้สิทธิ (โปรไฟล์ผู้ใช้) ขณะรันไทม์ของงาน.
SYSTEM_USER	ตัวบ่งชี้การให้สิทธิ (โปรไฟล์ผู้ใช้) ของผู้ใช้ที่เชื่อมต่อกับฐานข้อมูล.

ถ้าคำสั่งเดียวมีการอ้างอิงไปยังเรจิสเตอร์พิเศษ CURRENT DATE, CURRENT TIME, หรือ CURRENT TIMESTAMP, หรือฟังก์ชันแบบสเกลาร์ CURDATE, CURTIME, หรือ NOW มากกว่าหนึ่งการอ้างอิงแล้ว, ค่าทั้งหมดจะอยู่บนพื้นฐานของการอ่านข้อมูลจากนาฬิกาครั้งเดียว.

สำหรับคำสั่ง SQL ที่ทำงานแบบรีโมต ค่าสำหรับเรจิสเตอร์พิเศษจะถูกกำหนดที่ระบบรีโมต

เมื่อทำการสืบค้นกับตารางแบบกระจายที่อ้างอิงถึง register พิเศษ, เนื้อหาของ register พิเศษบนระบบที่ร้องขอการสืบค้นจะถูกใช้. สำหรับข้อมูลเพิ่มเติมเกี่ยวกับตารางแบบกระจาย โปรดดูกลุ่มหัวข้อ DB2 Multisystem

การแปลงชนิดข้อมูล

ในบางครั้ง คุณอาจจำเป็นต้องแปลง หรือเปลี่ยนชนิดของนิพจน์ไปเป็นชนิดข้อมูลอื่น หรือชนิดข้อมูลเดิมที่มีความยาว ความแม่นยำ หรือมาตราส่วนที่ต่างออกไป

ตัวอย่างเช่น ถ้าคุณต้องการเปรียบเทียบสองคอลัมน์ที่ต่างชนิดกัน เช่น ชนิดที่ผู้ใช้กำหนดเองที่เป็นแบบอักขระ และแบบจำนวนเต็ม คุณสามารถเปลี่ยนอักขระให้เป็นจำนวนเต็ม หรือจำนวนเต็มให้เป็นอักขระได้ เพื่อให้เปรียบเทียบกันได้ ประเภทข้อมูลที่สามารถถูกเปลี่ยนไปเป็นประเภทอื่นคือ*สามารถแปลงประเภท*จากประเภทข้อมูลต้นฉบับไปเป็นประเภทข้อมูลปลายทาง.

คุณสามารถใช้ฟังก์ชันการแปลง หรือค่ากำหนด CAST เพื่อทำการแปลงชนิดข้อมูลไปเป็นชนิดข้อมูลแบบอื่นได้โดยตรง. ตัวอย่างเช่น, ถ้าคุณมีคอลัมน์วันที่ (BIRTHDATE) ที่นิยามเป็น DATE และต้องการแปลงชนิดข้อมูลไปเป็น CHARACTER โดยมีความยาวคงที่คือ 10, ให้ป้อนค่าดังนี้:

```
SELECT CHAR (BIRTHDATE,USA)
   FROM CORPDATA.EMPLOYEE
```

นอกจากนี้ คุณสามารถใช้ค่ากำหนด CAST เพื่อแปลงชนิดข้อมูลได้โดยตรง:

```
SELECT CAST(BIRTHDATE AS CHAR(10))
   FROM CORPDATA.EMPLOYEE
```

สิ่งอ้างอิงที่เกี่ยวข้อง

การแปลงชนิดข้อมูล

ประเภทข้อมูลวันที่, เวลา, และ timestamp

วันที่, เวลา และ timestamp คือ ชนิดข้อมูลที่มีการแทนค่าในรูปแบบภายในซึ่งผู้ใช้ SQL ไม่เห็น

วันที่, เวลา, และ timestamp สามารถถูกแทนค่าโดยค่าสตริงอักขระ และสามารถถูกกำหนดค่าให้กับตัวแปรสตริงอักขระได้. ผู้จัดการฐานข้อมูลจะยอมรับสิ่งต่อไปนี้เป็นค่าวันที่, เวลา, และ Timestamp:

- ค่าที่คืนค่าโดยฟังก์ชันแบบสเกลาร์ คือ DATE, TIME หรือ TIMESTAMP
- ค่าที่คืนค่าโดย register พิเศษ คือ CURRENT DATE, CURRENT TIME หรือ CURRENT TIMESTAMP
- ค่าของสตริงอักขระในรูปแบบวันที่, เวลา, หรือ timestamp มาตรฐาน ANSI/ISO, ตัวอย่างเช่น, **DATE** '1950-01-01'.
- สตริงอักขระเมื่อใช้เป็น Operand ของนิพจน์ทางคณิตศาสตร์หรือการเปรียบเทียบ *และ* Operand อื่นเป็น Date, Time, หรือ Timestamp. ตัวอย่างเช่น, ในเพรดิเคต:

  ```
  ... WHERE HIREDATE < '1950-01-01'
  ```

  ถ้า HIREDATE คือคอลัมน์ข้อมูล, สตริงอักขระ '1950-01-01' จะถูกตีความให้เป็นวันที่.
- ตัวแปรหรือค่าคงที่แบบสตริงอักขระที่ถูกใช้เพื่อตั้งค่าคอลัมน์วันที่, เวลา, หรือ Timestamp ใน SET clauseของคำสั่ง UPDATE, หรือใน VALUES clause ของคำสั่ง INSERT.

สิ่งอ้างอิงที่เกี่ยวข้อง

ชนิดข้อมูล

การระบุค่าวันที่และเวลาปัจจุบัน:

คุณสามารถระบุวันที่ เวลา หรือ timestamp ในนิพจน์ โดยใช้หนึ่งในเรจิสเตอร์พิเศษเหล่านี้: CURRENT DATE, CURRENT TIME และ CURRENT TIMESTAMP

ค่าของแต่ละตัวจะอยู่บนพื้นฐานของการอ่านนาฬิกาบอกเวลาที่ได้รับขณะกำลังรันคำสั่ง. การอ้างอิงหลายตัวที่อ้างไปยัง CURRENT DATE, CURRENT TIME, หรือ CURRENT TIMESTAMP ภายในคำสั่ง SQL เดียวกันจะใช้ค่าเดียวกัน. คำสั่งต่อไปนี้จะคืนค่าอายุ (เป็นปี) ของพนักงานแต่ละคนในตาราง EMPLOYEE เมื่อรันคำสั่ง:

```
SELECT YEAR(CURRENT DATE - BIRTHDATE)
  FROM CORPDATA.EMPLOYEE
```

Register พิเศษ CURRENT TIMEZONE อนุญาตให้เวลาท้องถิ่นถูกแปลงไปเป็น Universal Time Coordinated (UTC) ได้. ตัวอย่างเช่น ถ้าคุณมีตารางชื่อ DATETIME ซึ่งเก็บประเภทคอลัมน์เวลาในชื่อ STARTT และคุณต้องการแปลง STARTT ไปเป็น UTC คุณสามารถใช้คำสั่งต่อไปนี้:

```
SELECT STARTT - CURRENT TIMEZONE
  FROM DATETIME
```

การคำนวณวันที่/เวลา:

การบวกและการลบ คือเครื่องหมายคำนวณที่สามารถใช้กับค่าวันที่, เวลา, และ timestamp เท่านั้น.

คุณสามารถเพิ่มค่าและลดค่าวันที่, เวลา, หรือ timestamp เป็นช่วงระยะเวลาได้; หรือลบวันที่จากวันที่, ลบเวลาออกจากเวลา, หรือ ลบ timestamp จาก timestamp.

สิ่งอ้างอิงที่เกี่ยวข้อง

การคำนวณ Datetime ใน SQL

นิพจน์การเปลี่ยนแถว

นิพจน์ ROW CHANGE TIMESTAMP และ ROW CHANGE TOKEN สามารถ ใช้เพื่อระบุว่าแถวถูกเปลี่ยนแปลงล่าสุดเมื่อไร

ในการใช้นิพจน์ ROW CHANGE TIMESTAMP สำหรับตาราง ตารางจะต้อง ถูกกำหนดให้มีคอลัมน์ row change timestamp

เคียวรี่ ต่อไปนี้สามารถค้นหาคำสั่งซื้อทั้งหมดที่มีอายุอย่างน้อยสี่สัปดาห์ และ สามารถแสดงรายการว่าคำสั่งซื้อเหล่านี้ถูกแก้ไขครั้งล่าสุดเมื่อไร

```
SELECT ORDERNO, ROW CHANGE TIMESTAMP FOR ORDERS
   FROM ORDERS
   WHERE ORDER_DATE < CURRENT DATE - 4 WEEKS
```

นิพจน์ ROW CHANGE TOKEN สามารถใช้สำหรับทั้งตารางที่มี row change timestamp และตารางที่ไม่มี โดยจะแสดงจุดที่มีการแก้ไขสำหรับ แถว ถ้าตารางมี row change timestamp ก็จะถูกนำมาจาก timestamp ดังกล่าว ถ้าตารางไม่มี row change timestamp ก็จะอ้างอิงเวลาแก้ไข ภายใน ซึ่งไม่ได้ขึ้นกับแถว ดังนั้นจึงมีความแม่นยำน้อยกว่า ตารางที่มี row change timestamp

การจัดการกับแถวซ้ำ

เมื่อ SQL ประเมินผลคำสั่ง select หลายแถว อาจจะมีคุณสมบัติพอที่จะอยู่ในตารางผลลัพธ์ขึ้นอยู่กับจำนวนของแถว ที่ตรงกับเงื่อนไขการค้นหาของคำสั่ง select บางแถว ในตารางผลลัพธ์อาจซ้ำกันได้

คุณสามารถระบุว่า คุณไม่ต้องการข้อมูลซ้ำกันโดยใช้คีย์เวิร์ด DISTINCT, ตามด้วยรายชื่อนิพจน์:

```
SELECT DISTINCT JOB, SEX
...
```

DISTINCT หมายความว่าคุณต้องการเลือกเฉพาะแถวที่ไม่ซ้ำเท่านั้น ถ้าแถวที่ถูกเลือกมีค่าซ้ำกับแถวอื่นในตารางผลลัพธ์, แถวที่ซ้ำจะถูกข้ามไป (ไม่นำมาใส่ในตารางผลลัพธ์). ตัวอย่างเช่น, สมมติว่าคุณต้องการรายการรหัสงานของพนักงาน. คุณไม่จำเป็นต้องรู้ว่าพนักงานคนไหนมีรหัสงานอะไร. เนื่องจากบางทีอาจมีหลายคนในแผนกที่มีรหัสงานเดียวกัน, ดังนั้นคุณสามารถใช้ DISTINCT เพื่อให้มั่นใจว่าตารางผลลัพธ์จะมีเฉพาะค่าที่ไม่ซ้ำเท่านั้น.

ตัวอย่างต่อไปนี้แสดงวิธีการดังที่กล่าวมา:

```
SELECT DISTINCT JOB
    FROM CORPDATA.EMPLOYEE
    WHERE WORKDEPT = 'D11'
```

ผลลัพธ์คือสองแถว

JOB
DESIGNER
MANAGER

ถ้าคุณไม่รวม DISTINCT ไว้ใน SELECT clause, คุณอาจพบกับแถวที่ซ้ำกันในผลลัพธ์ของคุณ, เนื่องจาก SQL ส่งคืนค่าคอลัมน์ *JOB* สำหรับแต่ละแถวที่ตรงกับเงื่อนไขการค้นหา. ค่า null จะถูกตีความเป็นแถวที่ซ้ำกันสำหรับ DISTINCT.

ถ้าคุณรวม DISTINCT ไว้ใน SELECT และคุณยังรวมลำดับการจัดเรียงแบบ shared-weight, อาจมีค่าส่งคืนเพียงเล็กน้อย. ลำดับการเรียงเป็นสาเหตุให้ค่าที่เก็บตัวอักขระเดียวกันมีน้ำหนักเท่ากัน. ถ้า 'MGR', 'Mgr' และ 'mgr' อยู่ในตารางเดียวกันทั้งหมด เฉพาะค่าใดค่าหนึ่งในนั้น จะถูกส่งคืน

หลักการที่เกี่ยวข้อง

"ลำดับการเรียง และ normalization ใน SQL" ในหน้า 121
ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

การนิยามเงื่อนไขการค้นหาที่ซับซ้อน

นอกจากเพรดิเคตการเปรียบเทียบพื้นฐานแล้ว เช่น = และ > เงื่อนไขการค้นหาสามารถมีเพรดิเคต BETWEEN, IN, EXISTS, IS NULL และ LIKE

เงื่อนไขการค้นหาสามารถมี scalar fullselect.

สำหรับอักขระ , หรือเพรดิเคตคอลัมน์ที่เป็นกราฟิกแบบ UCS-2 หรือ UTF-16 , จะมีการเรียงลำดับในส่วนของ Operand ก่อนที่จะไปทำกับส่วนที่เป็นเพรดิเคต BETWEEN, IN, EXISTS, และ LIKE clauses.

คุณยังสามารถทำเงื่อนไขการค้นหาได้หลายครั้ง.

- **BETWEEN ... AND ...** ถูกใช้เพื่อระบุเงื่อนไขการค้นหา ซึ่งเงื่อนไขจะถูกก็ต่อเมื่อค่านั้นอยู่ระหว่างสองค่านี้. ตัวอย่างเช่น, เมื่อต้องการค้นหาพนักงานทั้งหมดที่รับเข้ามาในปี 1987, คุณสามารถเขียนคำสั่งได้ดังนี้:

  ```
  ... WHERE HIREDATE BETWEEN '1987-01-01' AND '1987-12-31'
  ```

  คีย์เวิร์ด BETWEEN จะถูกรวมเข้าไปด้วย. เงื่อนไขการค้นหาที่ซับซ้อนขึ้น, แต่ตรงตัว, ซึ่งสร้างผลลัพธ์เดียวกันคือ:

  ```
  ... WHERE HIREDATE >= '1987-01-01' AND HIREDATE <= '1987-12-31'
  ```

- **IN** จะบอกว่าคุณสนใจแถวที่มีค่านิพจน์ที่คุณต้องการอยู่ระหว่างค่าที่คุณทำรายการไว้. ตัวอย่างเช่น, เมื่อต้องการค้นหาชื่อของพนักงานทั้งหมดในแผนก A00, C01, and E21, คุณสามารถระบุว่า:

  ```
  ... WHERE WORKDEPT IN ('A00', 'C01', 'E21')
  ```

- **EXISTS** จะบอกว่าคุณสนใจที่จะทดสอบแถวบางแถวมีอยู่หรือไม่. ตัวอย่างเช่น, เมื่อต้องการค้นหาว่า มีพนักงานคนใดที่มีเงินเดือนมากกว่า 60000, คุณอาจระบุ:

  ```
  EXISTS (SELECT * FROM EMPLOYEE WHERE SALARY > 60000)
  ```

- **IS NULL** จะบอกว่าคุณสนใจที่จะทดสอบเพื่อหาค่า null. ตัวอย่างเช่น, เมื่อต้องการค้นหาว่า มีพนักงานคนใดที่ไม่มีเบอร์โทรศัพท์, คุณอาจระบุว่า:

  ```
  ... WHERE EMPLOYEE.PHONE IS NULL
  ```

- **LIKE** จะบอกว่าคุณสนใจแถวที่มีค่านิพจน์เหมือนกับค่าที่คุณจัดหาได้. เมื่อคุณใช้ LIKE, แล้ว SQL จะค้นหาสตริงอักขระที่เหมือนกับค่าที่คุณระบุ. ระดับของความเหมือนจะถูกพิจารณาโดยอักขระพิเศษสองตัวที่ใช้ในสตริงที่คุณรวมเข้าไปในเงื่อนไขการค้นหา:

  _ ตัวอักขระขีดเส้นใต้แทนตัวอักขระเดี่ยวใดๆ.

  % เครื่องหมายเปอร์เซนต์แทนสตริงอักขระ 0 หรือมากกว่าที่ไม่รู้ค่า. ถ้าเครื่องหมายเปอร์เซนต์อยู่ตอนต้นของสตริงที่ใช้ค้นหา, แล้ว SQL จะอนุญาตให้ตัวอักขระ 0 ตัวหรือมากกว่านั้นมาอยู่นำหน้าค่าที่ตรงกันในคอลัมน์. มิฉะนั้นแล้ว, สตริงที่ใช้ค้นหาต้องเริ่มต้นที่ตำแหน่งแรกของคอลัมน์.

หมายเหตุ: ถ้าคุณดำเนินการกับข้อมูลแบบ MIXED, ลักษณะพิเศษดังต่อไปนี้จะใช้ได้: ตัวอักขระที่ขีดเส้นใต้ SBCS จะอ้างอิงไปยังอักขระ SBCS หนึ่งตัว. ข้อจำกัดนี้ใช้ไม่ได้กับเครื่องหมายเปอร์เซนต์; นั่นคือ, เครื่องหมายเปอร์เซนต์จะอ้างอิงถึงตัวอักขระ SBCS หรือ DBCS กี่ตัวก็ได้. โปรดดูกลุ่มหัวข้อ DB2 for i5/OS การอ้างอิง SQL สำหรับข้อมูลเพิ่มเติมเกี่ยวกับเพรดิเคต LIKE และข้อมูล MIXED

ใช้อักขระขีดเส้นใต้ หรือเครื่องหมายเปอร์เซนต์ เมื่อคุณไม่รู้ หรือไม่สนใจเกี่ยวกับอักขระทั้งหมดของค่าคอลัมน์. ตัวอย่างเช่น, เมื่อต้องการค้นหาพนักงานที่อาศัยอยู่ใน Minneapolis, คุณอาจระบุว่า:

```
... WHERE ADDRESS LIKE '%MINNEAPOLIS%'
```

SQL จะคืนค่าแถวใดๆ ที่มีสตริง MINNEAPOLIS ในคอลัมน์ ADDRESS, โดยไม่สนใจตำแหน่งของสตริง.

ตัวอย่างถัดมา, เมื่อต้องการแสดงรายชื่อเมืองที่ขึ้นต้นด้วย 'SAN', คุณอาจระบุ:

```
... WHERE TOWN LIKE 'SAN%'
```

ถ้าคุณต้องการค้นหาที่อยู่ใดๆ ที่ชื่อถนนไม่ได้อยู่ในรายการชื่อถนนหลักของคุณ, คุณสามารถใช้นิพจน์ในนิพจน์ LIKE. ในตัวอย่างนี้, คอลัมน์ STREET ในตารางจะถูกสมมติว่าเป็นตัวพิมพ์ใหญ่.

```
... WHERE UCASE (:address_variable) NOT LIKE '%'||STREET||'%'
```

ถ้าคุณต้องการค้นหาสตริงอักขระที่มีตัวอักขระขีดเส้นใต้หรือตัวอักขระเครื่องหมายเปอร์เซนต์อย่างใดอย่างหนึ่งแล้ว, ให้ใช้ ESCAPE clause เพื่อระบุตัวอักขระที่ต้องการหลีกเลี่ยง. ตัวอย่างเช่น, เมื่อต้องการดูธุรกิจทั้งหมดที่มีเปอร์เซนต์อยู่ในชื่อ, คุณอาจระบุ:

```
... WHERE BUSINESS_NAME LIKE '%@%%' ESCAPE '@'
```

อักขระเปอร์เซนต์ตัวแรกและตัวสุดท้ายในสตริง LIKE จะถูกตีความเป็นอักขระเปอร์เซนต์ LIKE ตามปกติ. การใช้ '@%' ปนเข้าไปด้วยจะถูกพิจารณาว่าเป็นอักขระเปอร์เซนต์.

หลักการที่เกี่ยวข้อง

"การใช้เคียวรี่ย่อย" ในหน้า 112
คุณสามารถใช้เคียวรี่ย่อยในเงื่อนไขการค้นหาเพื่อเป็นอีกทางหนึ่งในการ เลือกข้อมูล เคียวรี่ย่อยสามารถใช้ได้ทุกที่ที่นิพจน์สามารถใช้งานได้.

"ลำดับการเรียง และ normalization ใน SQL" ในหน้า 121
ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การระบุเงื่อนไขการค้นหาโดยใช้ WHERE clause" ในหน้า 46
WHERE clause จะระบุเงื่อนไขการค้นหา ที่บ่งชี้ถึงแถวที่คุณต้องการดึงค่า, อัพเดต หรือลบ

"นิพจน์ใน WHERE clause" ในหน้า 47
นิพจน์ใน WHERE clause ใช้ตั้งชื่อหรือระบุ สิ่งที่คุณต้องการเปรียบเทียบกับสิ่งอื่น

เพรดิเคต

ข้อพิจารณาพิเศษสำหรับ LIKE:

นี่คือข้อพิจารณาพิเศษบางข้อสำหรับการใช้ เพรดิเคต LIKE

- เมื่อตัวแปรโฮสต์ถูกใช้แทนค่าคงที่สตริงในรูปแบบการค้นหาแล้ว, คุณควรพิจารณาการใช้ตัวแปรโฮสต์ที่มีความยาวต่างๆ กัน. ซึ่งทำให้คุณสามารถ:
  - กำหนดค่าคงที่สตริงที่ถูกใช้ก่อนหน้านี้ให้กับตัวแปรโฮสต์โดยไม่ต้องเปลี่ยนอะไร

– รับค่าเงื่อนไขการเลือกและผลลัพธ์เดียวกันเหมือนกับว่าค่าคงที่สตริงถูกใช้

- เมื่อตัวแปรโฮสต์ที่ความยาวคงที่ถูกใช้แทนค่าคงที่สตริงในรูปแบบการค้นหา, คุณควรตรวจสอบให้แน่ใจว่าค่าที่ถูกระบุในตัวแปรโฮสต์มีค่าตรงกับรูปแบบที่ค่าคงที่สตริงใช้ในครั้งก่อน. ตัวอักขระทั้งหมดในตัวแปรโฮสต์ที่ไม่ได้ถูกกำหนดค่าจะถูกกำหนดค่าเริ่มต้นด้วยช่องว่าง.

  ตัวอย่างเช่น ถ้าคุณค้นหาโดยใช้รูปแบบสตริง 'ABC%' ในตัวแปรโฮสต์แบบความยาวแปรผัน ตัวอย่างค่าที่ได้กลับมาจะมีดังนี้:

  ```
  'ABCD      '  'ABCDE'    'ABCxxx'      'ABC '
  ```

  อย่างไรก็ตาม ถ้าคุณค้นหาโดยใช้รูปแบบการค้นหา 'ABC%' ที่อยู่ในตัวแปรโฮสต์ที่มีความยาวคงที่เท่ากับ 10 นี่คือบางค่าที่อาจ ได้กลับมา ถ้าสมมติว่าคอลัมน์มีความยาวเท่ากับ 12:

  ```
  'ABCDE       ' 'ABCD        ' 'ABCxxx      ' 'ABC         '
  ```

  หมายเหตุ: ค่าที่คืนกลับมาทั้งหมดจะเริ่มต้นด้วย 'ABC' และสิ้นสุดด้วยช่องว่างอย่างน้อย 6 ตัว ช่องว่างจะถูกใช้แทนเนื่องจากอักขระ 6 ตัวสุดท้ายในตัวแปรโฮสต์จะไม่ถูกกำหนดด้วยค่าที่เฉพาะ

  ถ้าคุณต้องการค้นหา โดยใช้ตัวแปรโฮสต์ที่ความยาวคงที่โดยมีอักขระ 7 ตัวสุดท้าย สามารถเป็นค่าใดก็ได้ ให้ค้นหาเป็นลักษณะ 'ABC%%%%%%%' ค่าที่ได้คืนค่ากลับมาจะเป็น:

  ```
  'ABCDEFGHIJ'  'ABCXXXXXXX'  'ABCDE'      'ABCDD'
  ```

หลายเงื่อนไขการค้นหาภายใน WHERE clause:

คุณสามารถทำการร้องขอเพิ่มเติม โดยโค้ดเงื่อนไขการค้นหาที่ประกอบด้วยหลายเพรดิเคต.

เงื่อนไขการค้นหาที่คุณระบุสามารถมี comparison operators หรือเพรดิเคต BETWEEN, DISTINCT, IN, LIKE, EXISTS, IS NULL, และ IS NOT NULL.

คุณสามารถรวมสองเพรดิเคตใดๆ เข้ากันด้วยตัวเชื่อม AND และ OR. นอกจากนี้, คุณสามารถใช้คีย์เวิร์ด NOT เพื่อระบุว่าเงื่อนไขการค้นหาที่ต้องการคือค่าตรงข้ามกับเงื่อนไขการค้นหาที่ระบุ. WHERE clause สามารถมีได้หลายเพรดิเคตตามที่คุณต้องการ.

- **AND** จะบอกว่า, เพื่อให้แถวข้อมูลถูกต้อง, แถวจะต้องตรงกับเพรดิเคตทั้งคู่ของเงื่อนไขการค้นหา. ตัวอย่างเช่น, เมื่อต้องการค้นหาพนักงานในแผนก D21 ที่รับเข้ามาทำงานหลังจาก 31 ธันวาคม, 1987, คุณอาจระบุ:

  ```
  ...
    WHERE WORKDEPT = 'D21' AND HIREDATE > '1987-12-31'
  ```

- **OR** จะบอกว่า, เพื่อให้แถวข้อมูลถูกต้อง, แถวต้องตรงกับเงื่อนไขที่ตั้งค่าโดยเพรดิเคตของเงื่อนไขการค้นหา เงื่อนไขใดเงื่อนไขหนึ่งหรือทั้งคู่. ตัวอย่างเช่น, เมื่อต้องการค้นหาพนักงานที่อยู่ในแผนก C01 หรือแผนก D11, คุณอาจระบุ:

  ```
  ...
    WHERE WORKDEPT = 'C01' OR WORKDEPT = 'D11'
  ```

  หมายเหตุ: คุณยังสามารถใช้ IN เพื่อระบุในการร้องขอ: WHERE WORKDEPT IN ('C01', 'D11').

- **NOT** จะบอกว่า, เพื่อให้ถูกต้องตามเกณฑ์, แถวต้องไม่ตรงกับเกณฑ์ที่ตั้งขึ้นโดยเงื่อนไขการค้นหาหรือเพรดิเคตที่ตามหลัง NOT. ตัวอย่างเช่น เมื่อ ต้องการค้นหาพนักงานทุกคนในแผนก E11 ยกเว้นพวกที่มีรหัสงานเท่ากับ "analyst" คุณอาจระบุ:

  ```
  ...
    WHERE WORKDEPT = 'E11' AND NOT JOB = 'ANALYST'
  ```

เมื่อ SQL ประเมินผลเงื่อนไขการค้นหาที่มีตัวเชื่อมเหล่านี้, SQL จะทำในลำดับเฉพาะ. และจะประเมินผล NOT clause ก่อน, ต่อมาจึงประเมินผล AND clause, ตามด้วย OR clause.

คุณสามารถเปลี่ยนแปลงลำดับของการประเมินผลโดยการใช้วงเล็บ. เงื่อนไขการค้นหาที่อยู่ในวงเล็บจะถูกประเมินผลก่อน. ตัวอย่างเช่น, เมื่อต้องการค้นหาพนักงานในแผนก E11 และ E21 ที่มีการศึกษาสูงกว่าระดับ 12, คุณสามารถระบุว่า:

```
...
  WHERE EDLEVEL > 12 AND
     (WORKDEPT = 'E11' OR WORKDEPT = 'E21')
```

วงเล็บจะกำหนดความหมายของเงื่อนไขการค้นหา. ในตัวอย่างนี้, คุณต้องการแถวทั้งหมดที่มี:

- ค่า WORKDEPT เป็น E11 หรือ E21, และ
- ค่า EDLEVEL ที่มากกว่า 12

ถ้าคุณไม่ได้ใช้วงเล็บ:

```
...
  WHERE EDLEVEL > 12 AND WORKDEPT = 'E11'
    OR WORKDEPT = 'E21'
```

ผลลัพธ์ของคุณจะแตกต่างออกไป. แถวที่ถูกเลือกคือแถวที่มี:

- WORKDEPT = E11 and EDLEVEL > 12, or
- WORKDEPT = E21, โดยไม่สนใจค่าของ EDLEVEL

ถ้าคุณกำลังรวมการเปรียบเทียบการเท่ากันหลายๆ ครั้ง, คุณสามารถเขียนเพรดิเคตด้วย AND ตามที่แสดงไว้ในตัวอย่างต่อไปนี้:

```
...
  WHERE WORKDEPT = 'E11' AND EDLEVEL = 12 AND JOB = 'CLERK'
```

คุณสามารถเปรียบเทียบรายการสองรายการ, ตัวอย่างเช่น:

```
...
  WHERE (WORKDEPT, EDLEVEL, JOB) = ('E11', 12, 'CLERK')
```

เมื่อรายการสองรายการถูกใช้, ไอเท็มแรกในรายการแรกจะถูกเปรียบเทียบกับไอเท็มแรกในรายการที่สอง, และเป็นเช่นนี้ตลอดจนรายการทั้งสอง. ดังนั้น, แต่ละรายการต้องมีจำนวนของ entry ตรงกัน. การใช้รายการเฉพาะเพื่อเขียนเคียวรี่ด้วย AND. รายการอาจถูกใช้ด้วย comparison operator เท่ากับ หรือไม่เท่ากับเท่านั้น.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การระบุเงื่อนไขการค้นหาโดยใช้ WHERE clause" ในหน้า 46
WHERE clause จะระบุเงื่อนไขการค้นหา ที่บ่งชี้ถึงแถวที่คุณต้องการดึงค่า, อัพเดต หรือลบ

การใช้ค่ากำหนด OLAP

ค่ากำหนด Online analytical processing (OLAP) จะถูกใช้เพื่อส่งคืนลำดับหมายเลข และหมายเลขแถวที่เป็นผลลัพธ์แถวของเคียวรี่ คุณ สามารถระบุ RANK, DENSE_RANK และ ROW_NUMBER

ตัวอย่าง: การจัดลำดับและการกำหนดหมายเลขแถว

สมมติว่า คุณต้องการรายการของเงินเดือน 10 อันดับแรกตามลำดับ เคียวรี่ต่อไปนี้จะจัดลำดับหมายเลขให้คุณ:

```
SELECT EMPNO, SALARY,
       RANK() OVER(ORDER BY SALARY DESC),
       DENSE_RANK() OVER(ORDER BY SALARY DESC),
       ROW_NUMBER() OVER(ORDER BY SALARY DESC)
FROM EMPLOYEE
FETCH FIRST 10 ROWS ONLY
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 9. ผลลัพธ์ของเคียวรี่ก่อนหน้า

EMPNO	SALARY	RANK	DENSE_RANK	ROW_NUMBER
000010	52,750.00	1	1	1
000110	46,500.00	2	2	2
200010	46,500.00	2	2	3
000020	41,250.00	4	3	4
000050	40,175.00	5	4	5
000030	38,250.00	6	5	6
000070	36,170.00	7	6	7
000060	32,250.00	8	7	8
000220	29,840.00	9	8	9
200220	29,840.00	9	8	10

ในตัวอย่างนี้, SALARY จะเรียงจากมากไปน้อยด้วย 10 แรกที่ถูกส่งคืนค่า. คอลัมน์ RANK จะแสดงการจัดลำดับของแต่ละเงินเดือน. สังเกตว่า มีอยู่สองแถวที่มีเงินเดือนเท่ากันที่ตำแหน่ง 2. แต่ละแถวจะถูกกำหนดค่าลำดับที่เหมือนกัน. แถวต่อไปนี้จะถูกกำหนดค่าเป็น 4. RANK จะส่งคืนค่าของแถวที่มากกว่าจำนวนของแถวทั้งหมดที่อยู่ก่อนหน้าแถวนั้น. ซึ่งจะมีช่องว่างอยู่ระหว่างลำดับหมายเลข ไม่ว่าลำดับหมายเลขเหล่านั้นจะซ้ำกันหรือไม่ก็ตาม.

ในทางตรงกันข้าม, คอลัมน์ DENSE_RANK จะแสดงค่า 3 สำหรับแถวโดยตรง หลังจากแถวที่ซ้ำกัน. DENSE_RANK ส่งคืนค่าสำหรับแถวซึ่งเป็นค่าที่มากกว่าจำนวนของค่าแถวที่ต่างกันซึ่งอยู่ก่อนหน้าแถวนั้น. และจะไม่มีช่องว่างในลำดับหมายเลข.

ROW_NUMBER ส่งคืนหมายเลขเฉพาะสำหรับแต่ละแถว. สำหรับแต่ละแถวที่มีค่าซ้ำกันตามลำดับที่ระบุ, การกำหนดหมายเลขแถวจะไม่มีเกณฑ์; หมายเลขแถวสามารถถูกกำหนดด้วยลำดับที่แตกต่างกันสำหรับแถวที่ซ้ำกัน เมื่อรันเคียวรี่อีกครั้ง.

ตัวอย่าง: กลุ่มการจัดลำดับ

สมมติว่าคุณต้องการ ค้นหาแผนกที่มีเงินเดือนโดยเฉลี่ยสูงที่สุด เคียวรี่ต่อไปนี้จะจัดกลุ่มข้อมูลตามแผนก คำนวณเงินเดือนโดยเฉลี่ยสำหรับแต่ละแผนก และจัดลำดับค่าเฉลี่ยที่ได้.

```
SELECT WORKDEPT, INT(AVG(SALARY)) AS AVERAGE,
        RANK() OVER(ORDER BY AVG(SALARY) DESC) AS AVG_SALARY
   FROM EMPLOYEE
   GROUP BY  WORKDEPT
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 10. ผลลัพธ์ของเคียวรี่ก่อนหน้านี้

WORKDEPT	AVERAGE	AVG_SALARY
B01	41,250	1
A00	40,850	2
E01	40,175	3
C01	29,722	4
D21	25,668	5
D11	25,147	6
E21	24,086	7
E11	21,020	8

ตัวอย่าง: การจัดลำดับภายในแผนก

สมมติว่า คุณต้องการรายการของพนักงานพร้อมด้วยการจัดลำดับโบนัสของพนักงานเหล่านั้นภายในแผนก การใช้ PARTITION BY clause, คุณสามารถระบุกลุ่มโดยแยกหมายเลขต่างหาก.

```
SELECT LASTNAME, WORKDEPT, BONUS,
       DENSE_RANK() OVER(PARTITION BY WORKDEPT ORDER BY BONUS DESC)
         AS BONUS_RANK_IN_DEPT
   FROM EMPLOYEE
   WHERE WORKDEPT LIKE 'E%'
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 11. ผลลัพธ์ของเคียวรี่ก่อนหน้า

LASTNAME	WORKDEPT	BONUS	BONUS_RANK_in_DEPT
GEYER	E01	800.00	1
HENDERSON	E11	600.00	1
SCHNEIDER	E11	500.00	2
SCHWARTZ	E11	500.00	2
SMITH	E11	400.00	3
PARKER	E11	300.00	4
SETRIGHT	E11	300.00	4

ตารางที่ 11. ผลลัพธ์ของเคียวรี่ก่อนหน้า (ต่อ)

LASTNAME	WORKDEPT	BONUS	BONUS_RANK_in_DEPT
SPRINGER	E11	300.00	4
SPENSER	E21	500.00	1
LEE	E21	500.00	1
GOUNOT	E21	500.00	1
WONG	E21	500.00	1
ALONZO	E21	500.00	1
MENTA	E21	400.00	2

ตัวอย่าง: การจัดลำดับและการเรียงลำดับด้วยผลลัพธ์ของนิพจน์ตาราง

สมมติว่า คุณต้องการค้นหาพนักงานห้าคนแรกที่มีเงินเดือนสูงที่สุดพร้อมด้วยชื่อแผนก ชื่อแผนกอยู่ในตาราง *แผนก* ดังนั้นจึงจำเป็นต้องใช้การรวม เนื่องจากการเรียงลำดับถูกทำในนิพจน์ตารางที่ซ้อนกัน การเรียงลำดับจังสามารถใช้เพื่อกำหนดค่าของ ROW_NUMBER ได้ ORDER BY ORDER OF *table* clause จะถูกใช้ในกรณีนี้.

```
SELECT ROW_NUMBER() OVER(ORDER BY ORDER OF EMP),
            EMPNO, SALARY, DEPTNO, DEPTNAME
   FROM (SELECT EMPNO, WORKDEPT, SALARY
            FROM EMPLOYEE
            ORDER BY SALARY DESC
            FETCH FIRST 5 ROWS ONLY) EMP,
        DEPARTMENT
   WHERE DEPTNO = WORKDEPT
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 12. ผลลัพธ์ของเคียวรี่ก่อนหน้า

ROW_NUMBER	EMPNO	SALARY	DEPTNO	DEPTNAME
1	000010	52,750.00	A00	SPIFFY COMPUTER SERVICE DIV.
2	000110	46,500.00	A00	SPIFFY COMPUTER SERVICE DIV.
3	200010	46,500.00	A00	SPIFFY COMPUTER SERVICE DIV.
4	000020	41,250.00	B01	PLANNING
5	000050	40,175.00	E01	SUPPORT SERVICES

การรวมข้อมูลจากตารางมากกว่าหนึ่งตาราง

บางครั้งข้อมูลที่คุณต้องการดูไม่ได้อยู่ในตารางเดียว. เมื่อต้องการสร้างแถวของตารางผลลัพธ์, คุณอาจต้องการดึงค่าบางคอลัมน์จากตารางหนึ่งและบางคอลัมน์จากตารางอื่น. คุณสามารถดึงค่าคอลัมน์และรวมค่าคอลัมน์จากสองตารางหรือมากกว่าเข้าไปอยู่ในแถวเดียวได้.

การรวมหลายประเภทได้รับการรองรับจาก DB2 for i5/OS: inner join, left outer join, right outer join, left exception join, right exception join, และ cross join.

การใช้หมายเหตุบนการดำเนินการรวม

เมื่อคุณรวมตารางตั้งแต่สองตารางขึ้นไป, ให้พิจารณารายการต่อไปนี้:

- ถ้ามีชื่อคอลัมน์ร่วมกัน, คุณจะต้องระบุแต่ละชื่อที่ตรงกันด้วยชื่อของตาราง (หรือชื่อที่สัมพันธ์กัน). ชื่อคอลัมน์ที่มีชื่อไม่ซ้ำไม่จำเป็นต้องระบุชื่อตาราง. อย่างไรก็ตาม, สามารถใช้ USING clause ในการรวม เพื่ออนุญาตให้คุณสามารถแยกแยะคอลัมน์ที่ซ้ำกันในตารางทั้งสองโดยไม่ต้องระบุชื่อของตาราง.
- ถ้าคุณไม่ได้ทำรายการชื่อคอลัมน์ที่คุณต้องการ, แต่ใช้ SELECT * แทน, SQL จะคืนค่าแถวที่ประกอบด้วยทุกคอลัมน์ของตารางแรก, ตามด้วยทุกคอลัมน์ของตารางที่สอง, เช่นนี้ไปเรื่อยๆ .
- คุณจะต้องมีสิทธิในการเลือกแถวจากตารางหรือมุมมองที่ระบุใน FROM clause.
- ลำดับการเรียงจะถูกใช้กับคอลัมน์แบบอักขระ , หรือคอลัมน์กราฟิก UCS-2 หรือ UTF-16 ทั้งหมดที่ถูกรวม.

Inner join:

inner join จะคืนค่าเฉพาะแถวจากแต่ละ ตารางที่มีค่าที่จับคู่กัยในคอลัมน์ร่วม แถวใดๆ ที่ไม่มี ค่าที่ตรงกันระหว่างตารางจะไม่ปรากฏในตารางผลลัพธ์

โดยการใช้ Inner Join, ค่าคอลัมน์จากหนึ่งแถวของตารางจะถูกรวมกับค่าคอลัมน์จากอีกแถวหนึ่งของตารางอื่น (หรือตารางเดียวกัน) เพื่อสร้างแถวข้อมูลเดี่ยว. SQL จะพิจารณาตารางทั้งคู่ที่ถูกระบุสำหรับการรวม เพื่อดึงค่าจากทุกแถวที่ตรงกับเงื่อนไขการค้นหาสำหรับการรวม. จะมีอยู่สองวิธีในการระบุ Inner Join: โดยการใช้ซินแทกซ์ JOIN, และโดยการใช้ WHERE clause.

สมมติว่าคุณต้องการดึงค่าหมายเลขพนักงาน, ชื่อ, และหมายเลขโครงการ สำหรับพนักงานทั้งหมดที่ต้องรับผิดชอบโครงการ. หรือพูดอีกอย่างหนึ่งว่า, คุณต้องการคอลัมน์ *EMPNO* และ *LASTNAME* จากตาราง CORPDATA.EMPLOYEE และคอลัมน์ *PROJNO* จากตาราง CORPDATA.PROJECT. เฉพาะพนักงานที่มีนามสกุลเริ่มต้นด้วย 'S' หรือต่อมาเท่านั้นที่จะถูกพิจารณา. เมื่อต้องการค้นหาข้อมูลนี้, คุณจำเป็นต้องรวมสองตารางเข้าด้วยกัน.

Inner join โดยใช้ไวยากรณ์ JOIN:

โดยการใช้ไวยากรณ์ inner join, ตารางทั้งสองที่คุณกำลังรวมกันจะถูกแสดงใน FROM clause, พร้อมด้วยเงื่อนไขการรวมที่ใช้กับตารางเหล่านั้น.

เงื่อนไขการรวมจะถูกระบุหลังคีย์เวิร์ด ON และจะใช้ในการพิจารณาว่าตารางทั้งสองจะเปรียบเทียบกันเพื่อสร้างผลลัพธ์การรวมได้อย่างไร. เงื่อนไขอาจเป็นตัวดำเนินการเปรียบเทียบใดๆ ก็ได้; ไม่จำเป็นต้องเป็นตัวดำเนินการเท่ากับเท่านั้น. เงื่อนไขการรวมหลายตัวสามารถระบุใน ON clause ได้โดยแยกกันด้วยคีย์เวิร์ด AND. เงื่อนไขเพิ่มเติมใดๆ ที่ไม่เกี่ยวกับการรวมจะถูกระบุใน WHERE clause หรือระบุเป็นส่วนของการรวมใน ON clause.

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE INNER JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

ในตัวอย่างนี้, การรวมจะทำกับตารางสองตารางโดยใช้คอลัมน์ EMPNO และ RESPEMP จากตาราง. เนื่องจากเฉพาะพนักงานที่มีนามสกุลขึ้นต้นด้วย 'S' อย่างน้อยหนึ่งตัวเท่านั้นที่จะถูกคืนค่ากลับมา, เงื่อนไขเพิ่มเติมนี้จะถูกจัดเตรียมใน WHERE clause.

เคียวรี่นี้จะคือค่าผลลัพธ์ดังนี้

EMPNO	LASTNAME	PROJNO
000250	SMITH	AD3112
000060	STERN	MA2110
000100	SPENSER	OP2010
000020	THOMPSON	PL2100

Inner join โดยใช้ WHERE clause:

ในการใช้ WHERE clause เพื่อให้เกิดการรวมแบบเดียวกัน กับที่คุณทำโดยใช้ไวยากรณ์ INNER JOIN ให้ใส่เงื่อนไขการรวมและ เงื่อนไขการเลือกเพิ่มเติมใน WHERE clause

ตารางที่จะถูกรวมจะถูกแสดงรายชื่อใน FROM clause, แยกกันด้วยเครื่องหมายจุลภาค.

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE, CORPDATA.PROJECT
  WHERE EMPNO = RESPEMP
  AND LASTNAME > 'S'
```

เคียวรี่นี้จะคืนค่าผลลัพธ์เดียวกับตัวอย่างที่แล้ว.

การรวมข้อมูลด้วย USING clause:

คุณสามารถใช้ USING clause เพื่อกำหนด เงื่อนไขการรวมอย่างรวดเร็ว USING clause เหมือนกับเงื่อนไขการรวม ซึ่งแต่ละคอลัมน์จากตารางด้านซ้ายจะถูกเปรียบเทียบกับคอลัมน์ที่มี ชื่อเดียวกันที่อยู่ในตารางด้านขวา

ตัวอย่าง, การใช้ USING clause :

```
SELECT EMPNO, ACSTDATE
     FROM CORPDATA.PROJACT INNER JOIN CORPDATA.EMPPROJACT
     USING (PROJNO, ACTNO)
     WHERE ACSDATE > '1982-12-31';
```

ข้อความด้านบนมีไวยากรณ์ที่ถูกต้อง และจะให้ผลเหมือนกับเงื่อนไขการรวมในประโยคด้านล่างนี้:

```
SELECT EMPNO, ACSTDATE
      FROM CORPDATA.PROJACT INNER JOIN CORPDATA.EMPPROJACT
           ON CORPDATA.PROJACT.PROJNO = CORPDATA.EMPPROJACT.PROJNO AND
              CORPDATA.PROJACT.ACTNO = CORPDATA.EMPPROJACT.ACTNO
      WHERE ACSTDATE > '1982-12-31';
```

Left outer join:

left outer join จะคืนค่าทุกแถวที่ inner join คืนค่ามา บวกกับอีกหนึ่งแถวสำหรับแถวอื่นแต่ละแถวในตารางแรก ที่ไม่มีค่าที่จับคู่กับตารางที่สอง

สมมติว่าคุณต้องการค้นหาพนักงานทั้งหมดและโครงการที่พนักงานคนนั้นกำลังรับผิดชอบอยู่ในปัจจุบัน. คุณต้องการดูพนักงานที่ไม่ได้ทำโครงการใดอยู่ด้วยเหมือนกัน. เคียวรี่ต่อไปนี้จะคืนรายชื่อของพนักงานทั้งหมดที่มีชื่อมากกว่า 'S' , พร้อมกับโครงการที่ได้รับมอบหมาย.

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE LEFT OUTER JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

ผลลัพธ์ของการสืบค้นนี้มีข้อมูลพนักงานบางคนที่ไม่มีหมายเลขโครงการอยู่. เขาจะถูกแสดงชื่ออยู่ในการสืบค้น, แต่จะมีค่า null ที่คืนค่ากลับมาสำหรับหมายเลขโครงการของเขา.

EMPNO	LASTNAME	PROJNO
000020	THOMPSON	PL2100
000060	STERN	MA2110
000100	SPENSER	OP2010
000170	YOSHIMURA	-
000180	SCOUTTEN	-
000190	WALKER	-
000250	SMITH	AD3112
000280	SCHNEIDER	-
000300	SMITH	-
000310	SETRIGHT	-
200170	YAMAMOTO	-
200280	SCHWARTZ	-
200310	SPRINGER	-
200330	WONG	-

หมายเหตุ: การใช้ฟังก์ชันแบบสเกลาร์ชื่อ RRN เพื่อคืนค่าหมายเลขเร็กคอร์ดเชิงสัมพันธ์สำหรับคอลัมน์ในตารางด้านขวามือของ Left Outer Loin หรือ Exception Join จะคืนค่าเป็น 0 สำหรับแถวที่ไม่มีค่าที่จับคู่กัน.

Right outer join:

right outer join จะคืนค่าทุกแถว ที่ inner join คืนค่ามา บวกกับหนึ่งแถวสำหรับแถวอื่นแต่ละแถวในตาราง ที่สองที่ไม่ค่าที่จับคู่กับตารางแรก ลักษณะนี้จะเหมือนกับการทำ Left Outer Join ด้วยตารางที่ระบุในลำดับที่ตรงกันข้ามกัน

เคียวรี่ที่ใช้เป็นตัวอย่างของ Left Outer Join สามารถนำมาเขียนใหม่ด้วย Right Outer Join ได้ดังนี้:

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.PROJECT RIGHT OUTER JOIN CORPDATA.EMPLOYEE
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

ผลลัพธ์ของเคียวรี่นี้จะเหมือนกันทุกประการกับผลลัพธ์จากเคียวรี่แบบ Left Outer Join.

Exception join:

left exception join จะคืนค่าเฉพาะแถวจากตารางแรกที่ **ไม่มี**ค่าที่จับคู่กับตารางที่สอง.

โดยการใช้ตารางเดียวกันเหมือนก่อนหน้านี้, จะคืนค่าพนักงานที่ไม่ได้รับผิดชอบโครงการใดๆ อยู่.

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE EXCEPTION JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

การรวมนี้จะคืนค่าผลลัพธ์

EMPNO	LASTNAME	PROJNO
000170	YOSHIMURA	-
000180	SCOUTTEN	-
000190	WALKER	-
000280	SCHNEIDER	-
000300	SMITH	-
000310	SETRIGHT	-
200170	YAMAMOTO	-
200280	SCHWARTZ	-
200310	SPRINGER	-
200330	WONG	-

Exception Join ยังสามารถเขียนให้เป็นการสืบค้นย่อยโดยใช้เพรดิเคต NOT EXISTS ได้. เคียวรีที่แล้วสามารถเขียนใหม่ได้ดังนี้:

```
SELECT EMPNO, LASTNAME
  FROM CORPDATA.EMPLOYEE
  WHERE LASTNAME > 'S'
    AND NOT EXISTS
       (SELECT * FROM CORPDATA.PROJECT
          WHERE EMPNO = RESPEMP)
```

ข้อแตกต่างเดียวเท่านั้นในการสืบค้นนี้ก็คือจะไม่มีการคืนค่าจากตาราง PROJECT.

นั่นคือ Right Exception Join, ด้วย, ที่ทำงานคล้าย Left Exception Join แต่ทำงานด้วยตารางที่กลับด้านกัน.

Cross join:

Cross join หรือ ผลรวมคาร์ทีเซียน คืนค่าตารางผลลัพธ์ที่เกิดจากการรวมค่าแต่ละแถวจากตารางแรก กับแต่ละแถวจากตารางที่สอง

จำนวนของแถวในตารางผลลัพธ์จะเป็นผลคูณของจำนวนแถวในแต่ละตาราง. ถ้าตารางมีขนาดใหญ่, การรวมนี้อาจใช้เวลานานมาก.

Cross Join สามารถระบุได้ด้วยสองวิธี: โดยการใช้ซินแทกซ์ JOIN หรือโดยการแสดงรายชื่อตารางใน FROM clauseที่แบ่งโดยเครื่องหมายจุลภาคโดยไม่ใช้ WHERE clause เพื่อเรียกเงื่อนไขการรวม.

สมมติมีตารางดังต่อไปนี้

ตารางที่ 13. ตาราง A

ACOL1	ACOL2
A1	AA1
A2	AA2
A3	AA3

ตารางที่ 14. ตาราง B

BCOL1	BCOL2
B1	BB1
B2	BB2

คำสั่ง Select ดังต่อไปนี้จะสร้างผลลัพธ์ที่เท่ากันทุกประการ.

```
SELECT * FROM A CROSS JOIN B
SELECT * FROM A, B
```

ตารางผลลัพธ์ของคำสั่ง SELECT อย่างใดอย่างหนึ่ง จะมีลักษณะดังนี้

ACOL1	ACOL2	BCOL1	BCOL2
A1	AA1	B1	BB1
A1	AA1	B2	BB2
A2	AA2	B1	BB1
A2	AA2	B2	BB2
A3	AA3	B1	BB1
A3	AA3	B2	BB2

Full outer join:

Full Outer Join จะคืนค่าแถวที่จับคู่กันจากตารางทั้งคู่ เหมือนกับ Left และ Right Outer Join อย่างไรก็ตาม full outer join ยังได้คืนค่าแถวที่ไม่จับคู่กันจากตารางทั้งคู่ด้วย

สมมติว่าคุณต้องการค้นหาพนักงานทั้งหมดและโครงการทั้งหมดของพนักงาน คุณต้องการดูพนักงานที่ไม่ได้ทำโครงการใดอยู่ด้วย รวมถึงโครงการใดๆ ที่ยังไม่ได้มอบหมายให้พนักงาน เคียวรี่ต่อไปนี้จะคืนรายชื่อของพนักงานทั้งหมดที่มีชื่อมากกว่า 'S' พร้อมกับโครงการที่ได้รับมอบหมาย:

```
SELECT EMPNO, LASTNAME, PROJNO
     FROM CORPDATA.EMPLOYEE FULL OUTER JOIN CORPDATA.PROJECT
          ON EMPNO = RESPEMP
     WHERE LASTNAME > 'S'
```

เนื่องจากไม่มีโครงการใดๆ ที่ไม่ได้มอบหมายให้แก่พนักงาน ดังนั้นเคียวรี่จึงคืนค่าแถวที่เหมือนกับ left outer join ผลลัพธ์เป็นดังนี้

EMPNO	LASTNAME	PROJNO
000020	THOMPSON	PL2100
000060	STERN	MA2110
000100	SPENSER	OP2010
000170	YOSHIMURA	-
000180	SCOUTTEN	-
000190	WALKER	-
000250	SMITH	AD3112
000280	SCHNEIDER	-
000300	SMITH	-
000310	SETRIGHT	-

EMPNO	LASTNAME	PROJNO
200170	YAMAMOTO	-
200280	SCHWARTZ	-
200310	SPRINGER	-
200330	WONG	-

ประเภทการรวมหลายประเภทในหนึ่งคำสั่ง:

มีบางครั้งที่จำเป็นต้องรวมตารางมากกว่าสองตาราง เพื่อให้ เกิดผลตามต้องการ

ถ้าคุณต้องการรายชื่อพนักงาน, ชื่อแผนกของ พนักงาน และโปรเจ็กต์ที่พนักงานคนนั้นรับผิดชอบ จึงจำเป็นต้องรวม ตาราง EMPLOYEE, ตาราง DEPARTMENT และตาราง PROJECT เพื่อให้ได้ ข้อมูลตามต้องการ ตัวอย่างต่อไปนี้จะแสดงเคียวรี่และผลลัพธ์:

```
SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE INNER JOIN CORPDATA.DEPARTMENT
       ON WORKDEPT = DEPTNO
     LEFT OUTER JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

EMPNO	LASTNAME	DEPTNAME	PROJNO
000020	THOMPSON	PLANNING	PL2100
000060	STERN	MANUFACTURING SYSTEMS	MA2110
000100	SPENSER	SOFTWARE SUPPORT	OP2010
000170	YOSHIMURA	MANUFACTURING SYSTEMS	-
000180	SCOUTTEN	MANUFACTURING SYSTEMS	-
000190	WALKER	MANUFACTURING SYSTEMS	-
000250	SMITH	ADMINISTRATION SYSTEMS	AD3112
000280	SCHNEIDER	OPERATIONS	-
000300	SMITH	OPERATIONS	-
000310	SETRIGHT	OPERATIONS	-

การใช้นิพจน์ตาราง

คุณสามารถใช้นิพจน์ตารางเพื่อระบุตารางผลลัพธ์ชั่วคราวได้.

นิพจน์ตารางสามารถถูกใช้แทนมุมมองเพื่อหลีกเลี่ยงการสร้างมุมมอง เมื่อไม่จำเป็นต้องใช้มุมมอง. นิพจน์ตารางประกอบด้วยนิพจน์ตารางที่ซ้อนกัน (ซึ่งเรียกว่าตารางลูก) และนิพจน์ตารางกลาง.

นิพจน์ตารางที่ซ้อนกันจะถูกระบุภายในวงเล็บใน FROM clause. ตัวอย่างเช่น, สมมติว่าคุณต้องการตารางผลลัพธ์ที่แสดงหมายเลขผู้จัดการ, หมายเลขแผนก, และเงินเดือนสูงสุดสำหรับแต่ละแผนก. หมายเลขผู้จัดการอยู่ในตาราง DEPARTMENT, หมายเลขแผนกอยู่ในทั้งตาราง DEPARTMENT และ EMPLOYEE, และเงินเดือนอยู่ในตาราง EMPLOYEE. คุณสามารถใช้นิพจน์ตารางใน FROM clause เพื่อเลือกเงินเดือนสูงสุดสำหรับแต่ละแผนก. คุณยังสามารถเพิ่มชื่อที่สัมพันธ์กัน, T2, ตามด้วยนิพจน์ที่ซ้อนกันเพื่อตั้งชื่อตารางลูก. คำสั่ง Select ด้านนอกจะใช้ T2 เพื่อระบุคอลัมน์ที่ถูกเลือกจากตารางลูก, ในกรณีนี้คือ MAXSAL และ WORKDEPT. โปรดสังเกตว่าคอลัมน์ MAX(SALARY) ที่ถูกเลือกในนิพจน์ตารางที่ซ้อนกันจะต้องถูกตั้งชื่อเพื่อสามารถถูกอ้างถึงจากคำสั่ง Select ด้านนอกได้. ซึ่งทำได้โดยใช้ AS clause.

```
SELECT MGRNO, T1.DEPTNO, MAXSAL
FROM CORPDATA.DEPARTMENT T1,
      (SELECT MAX(SALARY) AS MAXSAL, WORKDEPT
       FROM CORPDATA.EMPLOYEE E1
       GROUP BY WORKDEPT) T2
WHERE T1.DEPTNO = T2.WORKDEPT
ORDER BY DEPTNO
```

ผลลัพธ์ของเคียวรี่นี้คือ

MGRNO	DEPTNO	MAXSAL
000010	A00	52750.00
000020	B01	41250.00
000030	C01	38250.00
000060	D11	32250.00
000070	D21	36170.00
000050	E01	40175.00
000090	E11	29750.00
000100	E21	26150.00

นิพจน์ตารางกลางสามารถระบุก่อน Full-Select ในคำสั่ง SELECT, คำสั่ง INSERT statement, หรือคำสั่ง CREATE VIEW . และนิพจน์แบบนี้สามารถใช้เมื่อตารางผลลัพธ์เดียวกันถูกใช้ร่วมกันใน Full-Select. นิพจน์ตารางกลางจะนำหน้าด้วยคีย์เวิร์ด WITH.

ตัวอย่างเช่น, สมมติว่าคุณต้องการตารางที่แสดงค่าต่ำสุดและค่าสูงสุดของเงินเดือนเฉลี่ยในกลุ่มแผนกที่ต้องการ. ตัวอักขระตัวแรกของหมายเลขแผนกจะมีความหมายบางอย่าง และคุณต้องการค่าต่ำสุดและสูงสุดสำหรับแผนกที่ขึ้นต้นด้วยตัวอักษร 'D' และแผนกที่ขึ้นต้นด้วยตัวอักษร 'E'. คุณสามารถใช้นิพจน์ตารางกลางเพื่อเลือกเงินเดือนเฉลี่ยสำหรับแต่ละแผนก. คุณต้องตั้งชื่อตารางลูกด้วย; ในกรณีนี้, ชื่อคือ DT, อีกครั้งหนึ่ง. คุณสามารถระบุคำสั่ง SELECT โดยใช้ WHERE clause เพื่อจำกัดการเลือกให้เลือกเฉพาะแผนกที่เริ่มต้นด้วยตัวอักษรบางค่าได้. ระบุค่าต่ำสุดและค่าสูงสุดของคอลัมน์ AVGSAL จากตารางลูก DT. ระบุ UNION เพื่อให้ได้ผลลัพธ์สำหรับตัวอักษร 'E' และผลลัพธ์สำหรับตัวอักษร 'D'.

```
WITH DT AS (SELECT E.WORKDEPT AS DEPTNO, AVG(SALARY) AS AVGSAL
            FROM CORPDATA.DEPARTMENT D , CORPDATA.EMPLOYEE E
            WHERE D.DEPTNO = E.WORKDEPT
            GROUP BY E.WORKDEPT)
 SELECT 'E', MAX(AVGSAL), MIN(AVGSAL) FROM DT
```

```
WHERE DEPTNO LIKE 'E%'
UNION
SELECT 'D', MAX(AVGSAL), MIN(AVGSAL) FROM DT
WHERE DEPTNO LIKE 'D%'
```

ผลลัพธ์ของเคียวรี่นี้คือ

	MAX(AVGSAL)	MIN(AVGSAL)
E	40175.00	21020.00
D	25668.57	25147.27

สมมติว่า คุณต้องการเขียนเคียวรี่ที่ทำงานกับฐานข้อมูลรายการสั่งซื้อของคุณ ซึ่งจะคืนค่ารายการสูงสุด 5 รายการ (ในจำนวนรายการสั่งซื้อทั้งหมด) ภายในรายการสั่งซื้อ 1000 รายการล่าสุดจากลูกค้าที่สั่งซื้อรายการ 'XXX' ด้วย

```
WITH X AS (SELECT ORDER_ID, CUST_ID
                  FROM ORDERS
                  ORDER BY ORD_DATE DESC
                  FETCH FIRST 1000 ROWS ONLY),
     Y AS (SELECT CUST_ID, LINE_ID, ORDER_QTY
                  FROM X, ORDERLINE
                  WHERE X.ORDER_ID = ORDERLINE.ORDER_ID)
SELECT LINE_ID
     FROM (SELECT LINE_ID
                  FROM Y
                  WHERE Y.CUST_ID IN (SELECT DISTINCT CUST_ID
                                             FROM Y
                                             WHERE LINE.ID = 'XXX' )
                  GROUP BY LINE_ID
                  ORDER BY SUM(ORDER_QTY) DESC)
   FETCH FIRST 5 ROWS ONLY
```

นิพจน์ตารางกลาง (X) จะคืนค่าหมายเลขรายการสั่งซื้อ 1000 รายการล่าสุด. ผลลัพธ์จะถูกเรียงลำดับโดยวันที่ในลำดับจากมากไปน้อย แล้วเฉพาะ 1000 แถวแรกเท่านั้นที่จะถูกคืนค่าเป็นตารางผลลัพธ์.

นิพจน์ตารางกลางตัวที่สอง (Y) จะทำการรวมรายการสั่งซื้อ 1000 รายการล่าสุดเข้ากับตารางบรรทัดรายการ และคืนค่าลูกค้า (สำหรับแต่ละรายการสั่งซื้อทั้ง 1000 รายการ), บรรทัดรายการ, และจำนวนของบรรทัดรายการสำหรับรายการสั่งซื้อนั้น.

ตารางลูกในคำสั่ง Select หลักจะคืนค่าบรรทัดรายการสำหรับลูกค้าที่อยู่ในรายการสั่งซื้อ 1000 อันดับแรกที่สั่งซื้อรายการ XXX. ผลลัพธ์สำหรับลูกค้าทั้งหมดที่สั่งซื้อ XXX จะถูกจัดกลุ่มโดยบรรทัดรายการ และกลุ่มจะเรียงลำดับโดยจำนวนทั้งหมดของบรรทัดรายการ.

ท้ายสุด, คำสั่ง Select ภายนอกจะเลือกเฉพาะ 5 แถวแรกจากรายการที่ถูกเรียงลำดับแล้วซึ่งตารางลูกคืนค่ากลับมา.

การใช้เคียวรี่แบบเรียกซ้ำ

บางแอ็พพลิเคชันทำงานกับข้อมูลที่เรียกซ้ำโดยหน้าที่ของแอ็พพลิเคชันเอง. ในการ เคียวรี่ข้อมูลประเภทนี้ คุณสามารถใช้นิพจน์ตารางแบบเรียกซ้ำ หรือ มุมมองแบบเรียกซ้ำ

ตัวอย่างเช่น แอ็พพลิเคชัน Bill of Materials (BOM) จะ ทำงานกับการขยายของชิ้นส่วนและส่วนประกอบ ตัวอย่างเช่น, เก้าอี้อาจประกอบด้วยส่วนที่นั่งของเก้าอีก และชิ้นส่วนของขา. ส่วนที่นั่งของเก้าอี้อาจประกอบด้วยที่นังและพนักวางแขนสองข้าง. แต่ละชิ้นส่วนเหล่านี้สามารถแตกลงเป็นชิ้นส่วนย่อยๆ จนกระทั่งได้รายการของชิ้นส่วนทั้งหมดที่ใช้ในการสร้างเก้าอี้.

ในตัวอย่างการเดินทางโดยเครื่องบิน, เที่ยวบินและการเชื่อมต่อกับรถไฟจะถูกใช้เพื่อค้นหาเส้นทางการเดินทางระหว่างเมือง. definition ตารางและข้อมูลต่อไปนี้จะถูกใช้ในตัวอย่างนี้.

```
CREATE TABLE FLIGHTS (DEPARTURE CHAR(20),
                      ARRIVAL CHAR(20),
                      CARRIER CHAR(15),
                      FLIGHT_NUMBER CHAR(5),
                      PRICE INT)


INSERT INTO FLIGHTS VALUES('New York', 'Paris', 'Atlantic', '234', 400)
INSERT INTO FLIGHTS VALUES('Chicago', 'Miami', 'NA Air', '2334', 300)
INSERT INTO FLIGHTS VALUES('New York', 'London', 'Atlantic', '5473', 350)
INSERT INTO FLIGHTS VALUES('London', 'Athens'  , 'Mediterranean', '247', 340)
INSERT INTO FLIGHTS VALUES('Athens', 'Nicosia' , 'Mediterranean', '2356', 280)
INSERT INTO FLIGHTS VALUES('Paris', 'Madrid' , 'Euro Air',  '3256', 380)
INSERT INTO FLIGHTS VALUES('Paris', 'Cairo' , 'Euro Air', '63', 480)
INSERT INTO FLIGHTS VALUES('Chicago', 'Frankfurt', 'Atlantic', '37', 480)
INSERT INTO FLIGHTS VALUES('Frankfurt', 'Moscow', 'Asia Air', '2337', 580)
INSERT INTO FLIGHTS VALUES('Frankfurt', 'Beijing', 'Asia Air',  '77', 480)
INSERT INTO FLIGHTS VALUES('Moscow', 'Tokyo', 'Asia Air', '437', 680)
INSERT INTO FLIGHTS VALUES('Frankfurt', 'Vienna', 'Euro Air', '59', 200)
INSERT INTO FLIGHTS VALUES('Paris', 'Rome', 'Euro Air', '534', 340)
INSERT INTO FLIGHTS VALUES('Miami', 'Lima', 'SA Air', '5234', 530)
INSERT INTO FLIGHTS VALUES('New York', 'Los Angeles', 'NA Air', '84', 330)
INSERT INTO FLIGHTS VALUES('Los Angeles', 'Tokyo', 'Pacific Air', '824', 530)
INSERT INTO FLIGHTS VALUES('Tokyo', 'Hong Kong', 'Asia Air', '94', 330)
INSERT INTO FLIGHTS VALUES('Washington', 'Toronto', 'NA Air', '104', 250)

CREATE TABLE TRAINS(DEPARTURE CHAR(20),
                    ARRIVAL CHAR(20),
                    RAILLINE CHAR(15),
                    TRAIN CHAR(5),
                    PRICE INT)

INSERT INTO TRAINS VALUES('Chicago', 'Washington', 'UsTrack', '323', 90)
INSERT INTO TRAINS  VALUES('Madrid', 'Barcelona', 'EuroTrack', '5234', 60)
INSERT INTO TRAINS  VALUES('Washington' , 'Boston' , 'UsTrack', '232', 50)
```

ถึงตอนนี้ ตารางจะถูกสร้าง, ข้อมูลจะถูกเคียวรี่เพื่อค้นหาข้อมูลเกี่ยวกับเน็ตเวิร์กสายการบิน. สมมติว่า คุณต้องการค้นหาเมืองที่คุณต้องการบินไป หากคุณเริ่มต้นที่ Chicago, และจำนวนเที่ยวบินที่จะไปเมืองนั้น. เคียวรี่ต่อไปนี้จะแสดงข้อมูลนี้ให้คุณ.

```
WITH destinations (origin, departure, arrival, flight_count) AS
    (SELECT a.departure, a.departure, a.arrival, 1
            FROM flights a
            WHERE a.departure = 'Chicago'
     UNION ALL
     SELECT r.origin, b.departure, b.arrival, r.flight_count + 1
```

```
            FROM destinations r, flights b
            WHERE r.arrival = b.departure)
SELECT origin, departure, arrival, flight_count
    FROM destinations
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 15. ผลลัพธ์ของเคียวรี่ก่อนหน้า

ORIGIN	DEPARTURE	ARRIVAL	FLIGHT_COUNT
Chicago	Chicago	Miami	1
Chicago	Chicago	Frankfurt	1
Chicago	Miami	Lima	2
Chicago	Frankfurt	Moscow	2
Chicago	Frankfurt	Beijing	2
Chicago	Frankfurt	Vienna	2
Chicago	Moscow	Tokyo	3
Chicago	Tokyo	Hong Kong	4

เคียวรี่แบบเริกซ้ำนี้จะถูกเขียนออกเป็นสองส่วน. ส่วนแรกของนิพจน์ตารางจะเรียกว่า *intialization fullselect*. ซึ่งจะเลือกแถวแรกสำหรับชุดผลลัพธ์ของนิพจน์ตาราง. ในตัวอย่างนี้, เคียวรี่จะเลือกแถวสองแถวในตาราง*เที่ยวบิน* ซึ่งจะบอกคุณถึงสถานที่อื่นจากชิคาโกได้โดยตรง. และยังเตรียมข้อมูลเบื้องต้นเกี่ยวกับจำนวนของเที่ยวบินสำหรับแต่ละแถวที่เลือก.

ส่วนที่สองของเคียวรี่แบบเรียกซ้ำจะรวมแถวจากชุดผลลัพธ์ปัจจุบันของนิพจน์ตารางด้วยแถวอื่นจากตารางเดิม. ซึ่งเรียกว่า *iterative fullselect*. ทั้งหมดนี้คือการเรียกซ้ำ. สังเกตว่า แถวที่ถูกเลือกเป็นชุดผลลัพธ์จะถูกอ้างอิงโดยใช้ชื่อของนิพจน์ตารางเป็นชื่อตาราง และชื่อคอลัมน์ผลลัพธ์นิพจน์ตารางเป็นชื่อคอลัมน์.

ในส่วนของเคียวรี่แบบเรียกซ้ำ, แถวใดๆ จากตารางต้นฉบับที่คุณสามารถรับค่าเมืองที่จะไปถึงที่เลือกไว้ก่อนหน้าถูกเลือกไว้. เมืองที่จะไปถึงภายในแถวที่เลือกจะกลายเป็นเมืองที่จะออกเดินทางใหม่. แต่ละแถวจากการเลือกแบบเรียกซ้ำนี้จะเพิ่ม การนับเที่ยวบินไปยังปลายทางด้วยเที่ยวบินอื่น. แถวใหม่เหล่านี้จะถูกเพิ่มไปยังชุดผลลัพธ์นิพจน์ตาราง, ซึ่งจะส่งไปยัง iterative fullselect เพื่อสร้างแถวของชุดผลลัพธ์เพิ่มเติม. ในข้อมูลที่ได้จากผลลัพธ์สุดท้าย, คุณจะเห็นว่า จำนวนของเที่ยวบินทั้งหมดคือจำนวนของการรวมแบบเรียกซ้ำทั้งหมด (บวก 1) เพื่อที่จะได้เมืองที่จะไปถึง.

มุมมองแบบเรียกซ้ำดูคล้ายกับนิพจน์ตารางแบบเรียกซ้ำ. คุณสามารถเขียนนิพจน์ตารางก่อนหน้านี้เป็นมุมมองแบบเรียกซ้ำได้ดังนี้:

```
CREATE VIEW destinations (origin, departure, arrival, flight_count) AS
     SELECT departure, departure, arrival, 1
            FROM flights
            WHERE departure = 'Chicago'
     UNION ALL
     SELECT r.origin, b.departure, b.arrival, r.flight_count + 1
            FROM destinations r, flights b
            WHERE r.arrival = b.departure)
```

ส่วนของ definition มุมมองแบบ interactive fullselect นี้อ้างอิงถึงตัวของมุมมองเอง. การเลือกจากมุมมองนี้จะส่งคืนแถวที่เหมือนกันกับที่คุณได้มาจากนิพจน์ตารางแบบเรียกซ้ำก่อนหน้านี้.

ตัวอย่าง: จุดเริ่มต้นของเมืองสองเมือง

ถึงตอนนี้, เพื่อให้เคียวรี่ซับซ้อนยิ่งขึ้น, สมมติว่า คุณกำลังบินจาก Chicago หรือ New York, และคุณต้องการรู้สถานที่ที่คุณจะไป และค่าใช้จ่ายที่เกิดขึ้น.

```
WITH destinations (departure, arrival, connections, cost) AS
    (SELECT a.departure, a.arrival, 0, price
            FROM flights a
            WHERE a.departure = 'Chicago' OR
                  a.departure = 'New York'
     UNION ALL
     SELECT r.departure, b.arrival, r.connections + 1,
                  r.cost + b.price
            FROM destinations r, flights b
            WHERE r.arrival = b.departure)
SELECT departure, arrival, connections, cost
       FROM destinations
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 16. ผลลัพธ์ของเคียวรี่ก่อนหน้า

DEPARTURE	ARRIVAL	CONNECTIONS	COST
Chicago	Miami	0	300
Chicago	Frankfurt	0	480
New York	Paris	0	400
New York	London	0	350
New York	Los Angeles	0	330
Chicago	Lima	1	830
Chicago	Moscow	1	1,060
Chicago	Beijing	1	960
Chicago	Vienna	1	680
New York	Madrid	1	780
New York	Cairo	1	880
New York	Rome	1	740
New York	Athens	1	690
New York	Tokyo	1	860
Chicago	Tokyo	2	1,740

ตารางที่ 16. ผลลัพธ์ของเคียวรี่ก่อนหน้า (ต่อ)

DEPARTURE	ARRIVAL	CONNECTIONS	COST
New York	Nicosia	2	970
New York	Hong Kong	2	1,190
Chicago	Hong Kong	3	2,070

สำหรับแต่ละแถวที่ถูกส่งคืนค่า, ผลลัพธ์จะแสดงจุดเริ่มต้นของเมืองต้นทาง และเมืองปลายทางสุดท้าย. ซึ่งจะนับจำนวนของการเชื่อมต่อที่จำเป็น แทนที่จะเป็นจำนวนของเที่ยวบินทั้งหมด และเพิ่มข้อมูลค่าใช้จ่ายทั้งหมดสำหรับแต่ละเที่ยวบิน.

ตัวอย่าง: สองตารางที่ใช้สำหรับการเรียกซ้ำ

ถึงตอนนี้, สมมติว่า คุณเริ่มต้นจาก Chicago แต่ต้องเดินทางโดยรถไฟ แล้วเดินทางต่อด้วยเที่

เคียวรี่ต่อไปนี้จะส่งคืนข้อมูลนี้:

```
WITH destinations (departure, arrival, connections, flights, trains, cost) AS
  (SELECT f.departure, f.arrival, 0, 1, 0, price
          FROM  flights f
          WHERE f.departure = 'Chicago'
   UNION ALL
   SELECT t.departure, t.arrival, 0, 0, 1, price
          FROM trains t
          WHERE t.departure = 'Chicago'
   UNION ALL
   SELECT r.departure, b.arrival, r.connections + 1 , r.flights + 1, r.trains,
             r.cost + b.price
          FROM  destinations r, flights b
          WHERE r.arrival = b.departure
   UNION ALL
   SELECT r.departure, c.arrival, r.connections + 1 ,
             r.flights, r.trains + 1, r.cost + c.price
          FROM destinations r, trains c
          WHERE r.arrival = c.departure)
SELECT departure, arrival, connections, flights, trains, cost
       FROM destinations
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 17. ผลลัพธ์ของเคียวรี่ก่อนหน้า

DEPARTURE	ARRIVAL	CONNECTIONS	FLIGHTS	TRAINS	COST
Chicago	Miami	0	1	0	300
Chicago	Frankfurt	0	1	0	480
Chicago	Washington	0	0	1	90
Chicago	Lima	1	2	0	830
Chicago	Moscow	1	2	0	1,060

ตารางที่ 17. ผลลัพธ์ของเคียวรี่ก่อนหน้า (ต่อ)

DEPARTURE	ARRIVAL	CONNECTIONS	FLIGHTS	TRAINS	COST
Chicago	Beijing	1	2	0	960
Chicago	Vienna	1	2	0	680
Chicago	Toronto	1	1	1	340
Chicago	Boston	1	0	2	140
Chicago	Tokyo	2	3	0	1,740
Chicago	Hong Kong	3	4	0	2,070

ในตัวอย่างนี้, มีสองส่วนของนิพจน์ตารางที่กำหนดค่าเริ่มต้นให้กับเคียวรี่: ส่วนแรกสำหรับเที่ยวบิน และอีกหนึ่งส่วนสำหรับรถไฟ. สำหรับแต่ละแถวของผลลัพธ์, มีสองการอ้างอิงแบบเรียกซ้ำเพื่อดึงข้อมูลจากตำแหน่งที่ไปถึงก่อนหน้านี้ให้เป็นสถานที่ปลายทางถัดไปที่เป็นไปได้: การอ้างอิงแรกสำหรับการเดินทางต่อด้วยเครื่องบิน, การอ้างอิงถัดมาสำหรับการเดินทางต่อด้วยรถไฟ. ในผลลัพธ์สุดท้าย, คุณจะเห็นจำนวนของการเชื่อมต่อที่จำเป็น และจำนวนของสายการบินหรือเที่ยวรถไฟที่สามารถไปได้. .

ตัวอย่าง: อ็อพชัน DEPTH FIRST และ BREADTH FIRST

ตัวอย่างสองตัวอย่างนี้ จะแสดงลำดับแถวของชุดผลลัพธ์ที่แตกต่างกัน ขึ้นอยู่กับการเรียกซ้ำที่ถูกประมวลผลด้วย depth first หรือ breadth first.

หมายเหตุ: search clause ไม่สนับสนุนสำหรับมุมมองแบบเรียกซ้ำ. คุณสามารถกำหนดมุมมองที่มีนิพจน์ตารางแบบเรียกซ้ำเพื่อเรียกใช้ฟังก์ชันนี้.

อ็อพชันที่จะเป็นตัวกำหนดผลลัพธ์โดยใช้ breadth first หรือ depth first คือการเรียงลำดับความสัมพันธ์แบบเรียกซ้ำ ขึ้นอยู่กับคอลัมน์การรวมแบบเรียกซ้ำที่ระบุสำหรับ SEARCH BY clause. เมื่อการเรียกซ้ำถูกจัดการแบบ breadth first, ลูกทั้งหมดจะถูกประมวลผลครั้งแรก, แล้วตามด้วยหลาน, แล้วตามด้วยเหลน. เมื่อการเรียกซ้ำถูกจัดการแบบ depth first, กลุ่มของเร็กคอร์ดที่สัมพันธ์กันของเด็กหนึ่งคนแบบเรียกซ้ำแบบเต็มจะถูกประมวลผลก่อนที่จะไปยังเด็กคนต่อไป.

ในทั้งสองกรณี, คุณระบุชื่อคอลัมน์เพิ่มเติม ซึ่งจะถูกใช้โดยกระบวนการเรียกซ้ำ เพื่อเก็บแทร็กของการเรียงลำดับแบบ depth first หรือ breadth first. คอลัมน์นี้ต้องใช้ใน ORDER BY clause ของเคียวรี่แบบ outer เพื่อดึงแถวกลับเข้าในลำดับที่ระบุ. หากคอลัมน์นี้ไม่ถูกใช้ใน ORDER BY, อ็อพชันการประมวลผลแบบ DEPTH FIRST หรือ BREADTH FIRST จะถูกละเลย.

การเลือกว่า คอลัมน์ใดจะใช้สำหรับคอลัมน์ SEARCH BY เป็นสิ่งที่สำคัญ. เพื่อให้ผลลัพธ์มีความหมาย, จึงต้องเป็นคอลัมน์ที่ถูกใช้ใน iterative fullselect เพื่อรวมจาก initialization fullselect. ในตัวอย่างนี้, ARRIVAL คือคอลัมน์ที่ใช้.

เคียวรี่ต่อไปนี้จะส่งคืนข้อมูลนี้:

```
WITH destinations (departure, arrival, connections, cost) AS
    (SELECT f.departure, f.arrival, 0, price
            FROM flights f
            WHERE f.departure = 'Chicago'
     UNION ALL
     SELECT r.departure, b.arrival, r.connections + 1,
```

```
            r.cost + b.price
          FROM destinations r, flights b
          WHERE r.arrival = b.departure)
   SEARCH DEPTH FIRST BY arrival SET ordcol
SELECT *
   FROM destinations
   ORDER BY ordcol
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 18. ผลลัพธ์ของเคียวรี่ก่อนหน้า

DEPARTURE	ARRIVAL	CONNECTIONS	COST
Chicago	Miami	0	300
Chicago	Lima	1	830
Chicago	Frankfurt	0	480
Chicago	Moscow	1	1,060
Chicago	Tokyo	2	1,740
Chicago	Hong Kong	3	2,070
Chicago	Beijing	1	960
Chicago	Vienna	1	680

ในข้อมูลผลลัพธ์นี้, คุณจะเห็นสถานที่ปลายทางทั้งหมดที่สร้างจากแถว Chicago-to-Miami ซึ่งถูกแสดงก่อนสถานที่ปลายทางจากแถว Chicago-to-Frankfort.

ถัดไป, คุณสามารถรันเคียวรี่เดียวกันนี้ แต่ร้องขอให้มีการเรียงลำดับผลลัพธ์แบบ breadth first.

```
WITH destinations (departure, arrival, connections, cost) AS
    (SELECT f.departure, f.arrival, 0, price
          FROM flights f
          WHERE f.departure='Chicago'
     UNION ALL
     SELECT r.departure, b.arrival, r.connections + 1,
            r.cost + b.price
          FROM destinations r, flights b
          WHERE r.arrival = b.departure)
    SEARCH BREADTH FIRST BY arrival SET ordcol
SELECT *
  FROM destinations
  ORDER BY ordcol
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 19. ผลลัพธ์ของเคียวรี่ก่อนหน้า

DEPARTURE	ARRIVAL	CONNECTIONS	COST
Chicago	Miami	0	300
Chicago	Frankfurt	0	480
Chicago	Lima	1	830
Chicago	Moscow	1	1,060
Chicago	Beijing	1	960
Chicago	Vienna	1	680
Chicago	Tokyo	2	1,740
Chicago	Hong Kong	3	2,070

ในข้อมูลผลลัพธ์นี้, คุณจะเห็นการเชื่อมต่อโดยตรงทั้งหมดจากชิคาโกที่ถูกแสดงไว้ก่อนการเชื่อมต่อเที่ยวบิน. ข้อมูลจะเหมือนกับกับผลลัพธ์ที่ได้จากเคียวรี่ก่อนหน้า, แต่อยู่ในลำดับการเรียงแบบ breadth first.

ตัวอย่าง: การวน

คีย์สำหรับการประมวลผลแบบเรียกซ้ำ, ไม่ว่าจะเป็น algorithm โปรแกรมมิงแบบเรียกซ้ำ หรือการเคียวรี่ข้อมูลแบบเรียกซ้ำ, คือการเรียกซ้ำต้องถูกจำกัด. ถ้าไม่, คุณจะไม่จบจากการวนซ้ำ. อ็อพชัน CYCLE จะอนุญาตให้คุณป้องกันการวนซ้ำข้อมูล. ไม่เพียงแต่การยกเลิกการวนซ้ำ แต่จะอนุญาตให้คุณออกเอาต์พุตตัวบ่งชี้การมาร์กให้วนรอบ ซึ่งอาจนำคุณไปยังข้อมูลที่วนซ้ำ.

หมายเหตุ: cycle clause ไม่สนับสนุนสำหรับมุมมองแบบเรียกซ้ำ. คุณสามารถกำหนดมุมมองที่มีนิพจน์ตารางแบบเรียกซ้ำเพื่อเรียกใช้ฟังก์ชันนี้.

สำหรับตัวอย่างสุดท้าย, สมมติว่า เรามีข้อมูลแบบวัฏจักร. โดยการเพิ่มแถวเข้าไปหนึ่งแถวในตาราง, ซึ่งจะเป็นเที่ยวบินจาก Cairo ไปยัง Paris และจาก Paris ไปยัง Cairo. ไม่มีเหตุผลสำหรับการมีข้อมูลวัฏจักรแบบนี้, ซึ่งง่ายต่อการเคียวรี่โดยที่จะไปยังการประมวลผลการวนรอบแบบไม่รู้จบ.

เคียวรี่ต่อไปนี้จะส่งคืนข้อมูลนี้:

```
INSERT INTO FLIGHTS VALUES('Cairo', 'Paris', 'Euro Air', '1134', 440)


WITH destinations (departure, arrival, connections, cost, itinerary) AS
    (SELECT f.departure, f.arrival, 1, price,
                CAST(f.departure CONCAT f.arrival AS VARCHAR(2000))
            FROM flights f
            WHERE f.departure = 'New York'
     UNION ALL
     SELECT r.departure, b.arrival, r.connections + 1 ,
                r.cost + b.price, CAST(r.itinerary CONCAT b.arrival AS VARCHAR(2000))
            FROM destinations r, flights b
```

```
            WHERE r.arrival = b.departure)
    CYCLE arrival SET cyclic_data TO '1' DEFAULT '0'
SELECT departure, arrival, itinerary, cyclic_data
      FROM destinations
      ORDER BY cyclic_data
```

เคียวรี่นี้ส่งคืนข้อมูลต่อไปนี้.

ตารางที่ 20. ผลลัพธ์ของเคียวรี่ก่อนหน้า

DEPARTURE	ARRIVAL	ITINERARY	CYCLIC_DATA
New York	Paris	New York Paris	0
New York	London	New York London	0
New York	Los Angeles	New York Los Angeles	0
New York	Madrid	New York Paris Madrid	0
New York	Cairo	New York Paris Cairo	0
New York	Rome	New York Paris Rome	0
New York	Athens	New York London Athens	0
New York	Tokyo	New York Los Angeles Tokyo	0
New York	Paris	New York Paris Cairo Paris	1
New York	Nicosia	New York London Athens Nicosia	0
New York	Hong Kong	New York Los Angeles Tokyo Hong Kong	0

ในตัวอย่างนี้, คอลัมน์ ARRIVAL จะถูกกำหนดใน CYCLE clause เป็นคอลัมน์ที่ใช้สำหรับตรวจการวนของข้อมูล. เมื่อพบการวนของข้อมูล, คอลัมน์พิเศษ, ในกรณีนี้คือ CYCLIC_DATA, จะถูกตั้งค่าอักขระเป็น '1' สำหรับแถวที่วนในชุดผลลัพธ์. แถวอื่นๆ ทั้งหมดจะยังคงมีค่าดีฟอลต์เป็น '0'. เมื่อพบการวนบนคอลัมน์ ARRIVAL, การประมวลผลจะไม่ดำเนินการใดๆ ต่อ ดังนั้นการวนรอบแบบไม่รู้จบจะไม่เกิดขึ้น. เมื่อต้องการดู หากข้อมูลมีการอ้างอิงแบบวนรอบ, คอลัมน์ CYCLIC_DATA จะถูกอ้างอิงในเคียวรี่แบบ outer.

การใช้คีย์เวิร์ด UNION เพื่อรวมการเลือกย่อย

การใช้คีย์เวิร์ด UNION, คุณสามารถรวมการเลือกย่อยมากกว่าสองได้ เพื่อสร้าง fullselect.

เมื่อ SQL เจอคีย์เวิร์ด UNION, SQL จะดำเนินการกับการเลือกย่อยแต่ละตัวเพื่อสร้างตารางผลลัพธ์ชั่วคราว, จากนั้นจึงรวมตารางผลลัพธ์ชั่วคราวของการเลือกย่อยแต่ละครั้ง และลบแถวที่ซ้ำกันเพื่อสร้างตารางผลลัพธ์ที่ถูกรวมแล้ว. คุณสามารถใช้ clause และเทคนิคได้หลากหลาย เมื่อเขียนโค้ดคำสั่ง select.

คุณสามารถใช้ UNION เพื่อลบรายการที่ซ้ำกันเมื่อรวมรายการของค่าที่รับมาจากหลายตาราง. ตัวอย่างเช่น, คุณสามารถรับรายการรวมของหมายเลขพนักงานที่มี:

- คนในแผนก D11
- คนที่ได้รับมอบงานในโครงการ MA2112, MA2113, และ AD3111

รายการรวมจะรับค่ามาจากตารางสองตารางและเก็บค่าที่ไม่ซ้ำกัน. เมื่อต้องการทำสิ่งนี้, ให้ระบุ:

```
SELECT EMPNO
   FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = 'D11'
 UNION
 SELECT EMPNO
   FROM CORPDATA.EMPPROJACT
   WHERE PROJNO = 'MA2112' OR
      PROJNO = 'MA2113' OR
      PROJNO = 'AD3111'
 ORDER BY EMPNO
```

เมื่อต้องการเข้าใจผลลัพธ์จากคำสั่ง SQL ยิ่งขึ้น, ลองจินตนาการว่า SQL จะทำงานผ่านกระบวนการดังต่อไปนี้:

ขั้นตอน 1. SQL จะดำเนินการกับคำสั่ง SELECT แรก:

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
```

เคียวรี่ส่งคืน ตารางผลลัพธ์ชั่วคราวต่อไปนี้

EMPNO from CORPDATA.EMPLOYEE
000060
000150
000160
000170
000180
000190
000200
000210
000220
200170
200220

ขั้นตอน 2. SQL จะดำเนินการกับคำสั่ง SELECT ที่สอง:

```
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
```

เคียวรี่ส่งคืน ตารางผลลัพธ์ชั่วคราวอีกตารางหนึ่ง

EMPNO from CORPDATA.EMPPROJACT
000230
000230
000240
000230
000230
000240
000230
000150
000170
000190
000170
000190
000150
000160
000180
000170
000210
000210

ขั้นตอน 3. SQL จะรวมตารางผลลัพธ์ชั่วคราวทั้งสองเข้าด้วยกัน, เอาแถวที่ซ้ำกันออก, และเรียงลำดับผลลัพธ์:

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
UNION
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                PROJNO= 'MA2113' OR
                PROJNO= 'AD3111'
ORDER BY EMPNO
```

เคียวรี่ส่งคืนตาราง ผลลัพธ์รวมซึ่งมีค่าในลำดับจากน้อยไปมาก

EMPNO
000060
000150
000160
000170
000180
000190
000200
000210
000220
000230
000240
200170
200220

เมื่อคุณใช้ UNION:

- ทุก ORDER BY clause ต้องปรากฏอยู่หลังการเลือกย่อยสุดท้ายที่เป็นส่วนหนึ่งของ Union. ในตัวอย่างนี้, ผลลัพธ์จะเรียงลำดับตามคอลัมน์ที่ถูกเลือกตัวแรก, *EMPNO*. ORDER BY clauseจะระบุว่าตารางผลลัพธ์รวมจะอยู่ในรูปแบบที่มีลำดับการเรียง. ORDER BY จะไม่อนุญาตให้ใช้ในวิว.
- ชื่อสามารถถูกระบุในอนุประโยค ORDER BY ได้ถ้าคอลัมน์ผลลัพธ์ถูกตั้งชื่อ. คอลัมน์ผลลัพธ์จะถูกตั้งชื่อถ้าคอลัมน์ที่สัมพันธ์กันในแต่ละคำสั่ง Select ที่ถูกรวมมีชื่อเดียวกัน. AS clause สามารถใช้เพื่อกำหนดชื่อให้กับคอลัมน์ในรายการที่เลือกได้.

  ```
  SELECT A + B AS X ...
  UNION
  SELECT X ... ORDER BY X
  ```

  ถ้าคอลัมน์ผลลัพธ์ไม่ได้ถูกตั้งชื่อ, ให้ใช้จำนวนเต็มบวกเพื่อเรียงลำดับผลลัพธ์. หมายเลขจะอ้างอิงไปยังตำแหน่งของนิพจน์ในรายการของนิพจน์ที่คุณรวมเข้าไปในการเลือกย่อยของคุณ.

  ```
  SELECT A + B ...
  UNION
  SELECT X ... ORDER BY 1
  ```

เมื่อต้องการระบุว่าการเลือกย่อยแต่ละแถวมาจากไหน, คุณสามารถรวมค่าคงที่เข้าไปที่ท้ายสุดของรายการเลือกสำหรับแต่ละการเลือกย่อยใน Union. เมื่อ SQL คืนค่าผลลัพธ์ของคุณกลับมา, คอลัมน์สุดท้ายจะเก็บค่าคงที่สำหรับการเลือกย่อยที่เป็นต้นฉบับของแถวนั้น. ตัวอย่างเช่น, คุณสามารถระบุ:

```
SELECT A, B, 'A1' ...
UNION
SELECT X, Y, 'B2'...
```

เมื่อแถวถูกส่งคืนค่ากลับมา, ค่าดังกล่าวจะรวมค่า (A1 หรือ B2) เพื่อบ่งชี้ถึงตารางที่มีต้นฉบับของค่าแถว. ถ้าชื่อคอลัมน์ใน Union ต่างกัน, SQL จะใช้ชุดของชื่อคอลัมน์ที่ระบุในการเลือกย่อยเมื่อ SQL แบบโต้ตอบแสดงหรือพิมพ์ผลลัพธ์, หรือใน SQLDA ที่เป็นผลลัพธ์จากการดำเนินการคำสั่ง DESCRIBE ของ SQL.

หมายเหตุ: การเรียงลำดับจะถูกใช้หลังจากฟิลด์ระหว่าง UNION ถูกทำให้เข้ากันได้แล้ว. การเรียงลำดับถูกใช้สำหรับการดำเนินการเฉพาะที่เกิดขึ้นโดยตรงขณะดำเนินการ UNION.

หลักการที่เกี่ยวข้อง

"ลำดับการเรียง และ normalization ใน SQL" ในหน้า 121
ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การสร้างและการใช้มุมมอง" ในหน้า 36
สามารถใช้มุมมองเพื่อเข้าใช้งานข้อมูลในตารางหนึ่งตารางหรือมากกว่าหรือมุมมองหนึ่งมุมมองหรือมากกว่าได้. คุณสามารถสร้างมุมมองได้โดยใช้คำสั่ง SELECT.

การระบุคีย์เวิร์ด UNION ALL:

ถ้าคุณต้องการเก็บค่าที่ซ้ำกันในผลลัพธ์ ของการดำเนินการ UNION ให้ระบุคีย์เวิร์ด UNION ALL แทนที่จะระบุแค่ UNION

หัวข้อนี้จะใช้ขั้นตอนและตัวอย่างเดียวกันกับ "การใช้คีย์เวิร์ด UNION เพื่อรวมการเลือกย่อย" ในหน้า 84

ขั้นตอน 3. SQL จะรวมตารางผลลัพธ์ชั่วคราวทั้งสอง:

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
 UNION ALL
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
   ORDER BY EMPNO
```

เคียวรี่ส่งคืนตารางผลลัพธ์ที่มีการเรียงลำดับซึ่งรวมข้อมูลที่ซ้ำด้วย

EMPNO
000060
000150
000150
000150

EMPNO
000160
000160
000170
000170
000170
000170
000180
000180
000190
000190
000190
000200
000210
000210
000210
000220
000230
000230
000230
000230
000230
000240
000240
200170
200220

การดำเนินการ UNION ALL ทำให้เกิดการเชื่อมโยง, ตัวอย่างเช่น:

```
(SELECT PROJNO FROM CORPDATA.PROJECT
UNION ALL
SELECT PROJNO FROM CORPDATA.PROJECT)
UNION ALL
SELECT PROJNO FROM CORPDATA.EMPPROJACT
```

คำสั่งนี้ยังสามารถเขียนได้เป็น:

```
SELECT PROJNO FROM CORPDATA.PROJECT
UNION ALL
(SELECT PROJNO FROM CORPDATA.PROJECT
UNION ALL
SELECT PROJNO FROM CORPDATA.EMPPROJACT)
```

เมื่อคุณรวม UNION ALL เข้าไปในคำสั่ง SQL เดียวกับตัวดำเนินการ UNION, อย่างไรก็ตาม, ผลลัพธ์ของการดำเนินการจะขึ้นอยู่กับลำดับของการประเมินผล. เมื่อไม่มีวงเล็บ, การประเมินผลจะทำจากซ้ายไปขวา. หากมีวงเล็บ, การเลือกย่อยที่อยู่ในวงเล็บจะถูกประเมินผลก่อน, ตามด้วย, จากซ้ายไปขวา, โดยส่วนอื่นของคำสั่ง.

การใช้คีย์เวิร์ด EXCEPT

คีย์เวิร์ด EXCEPT จะส่งค่ากลับมาโดยนำผลที่ได้จากการเลือกย่อยชุดแรกลบด้วยแถวที่ซ้ำกันในผลของการเลือกย่อยชุดที่สอง.

สมมติว่า คุณต้องการค้นหารายการหมายเลขพนักงาน ที่ประกอบด้วยบุคคลในแผนก D11 ยกเว้นผู้ที่ได้รับมอบหมายงานในโครงการ MA2112, MA2113 และ AD3111

ผลจากเคียวรี่จะได้เป็น พนักงานทั้งหมดที่อยู่ในแผนก D11 ผู้ที่ *ไม่ได้* ทำงานในโครงการ MA2112, MA2113 และ AD3111:

```
SELECT EMPNO
   FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = 'D11'
 EXCEPT
 SELECT EMPNO
   FROM CORPDATA.EMPPROJACT
   WHERE PROJNO = 'MA2112' OR
      PROJNO = 'MA2113' OR
      PROJNO = 'AD3111'
 ORDER BY EMPNO
```

เมื่อต้องการเข้าใจผลลัพธ์จากคำสั่ง SQL ยิ่งขึ้น, ลองจินตนาการว่า SQL จะทำงานผ่านกระบวนการดังต่อไปนี้:

ขั้นตอน 1. SQL จะดำเนินการกับคำสั่ง SELECT แรก:

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
```

เคียวรี่นี้ ส่งคืนตารางผลลัพธ์ชั่วคราวตารางหนึ่ง

EMPNO from CORPDATA.EMPLOYEE
000060
000150
000160
000170

EMPNO from CORPDATA.EMPLOYEE
000180
000190
000200
000210
000220
200170
200220

ขั้นตอน 2. SQL จะดำเนินการกับคำสั่ง SELECT ที่สอง:

```
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
```

เคียวรี่นี้ ส่งคืนตารางผลลัพธ์ชั่วคราวอีกตารางหนึ่ง

EMPNO from CORPDATA.EMPPROJACT
000230
000230
000240
000230
000230
000240
000230
000150
000170
000190
000170
000190
000150
000160
000180

EMPNO from CORPDATA.EMPPROJACT
000170
000210
000210

ขั้นตอน 3. SQL จะนำตารางผลลัพธ์ชั่วคราวตารางแรก, ลบแถวทั้งหมดที่มีเหมือนกับตารางผลลัพธ์ชั่วคราวที่สอง, ลบแถวที่ซ้ำกันออก, และเรียงลำดับผลลัพธ์:

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
EXCEPT
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
ORDER BY EMPNO
```

เคียวรี่นี้ส่งคืนตาราง ผลลัพธ์รวมซึ่งมีค่าในลำดับจากน้อยไปมาก

EMPNO
000060
000200
000220
200170
200220

การใช้คีย์เวิร์ด INTERSECT

คีย์เวิร์ด INTERSECT จะส่งค่าผลลัพธ์รวมที่ประกอบด้วยแถวที่ปรากฏอยู่ในผลลัพธ์ทั้งสองชุด.

สมมติว่า คุณต้องการค้นหารายการหมายเลขพนักงาน ที่ประกอบด้วยบุคคลในแผนก D11 และบุคคลที่ได้รับมอบหมายงานใน โครงการ MA2112, MA2113 และ AD3111

INTERSECT จะ ให้ผลเป็นรายการหมายเลขพนักงานทั้งหมดที่ปรากฏอยู่ในผลลัพธ์ทั้งสองชุด หรือพูดอีกอย่างก็คือ ผลจากเคียวรี่จะได้เป็นพนักงานทั้งหมดที่อยู่ในแผนก D11 ที่ทำงานอยู่ในโครงการ MA2112, MA2113 และ AD3111 ด้วย:

```
SELECT EMPNO
   FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = 'D11'
 INTERSECT
 SELECT EMPNO
   FROM CORPDATA.EMPPROJACT
```

```
   WHERE PROJNO = 'MA2112' OR
      PROJNO = 'MA2113' OR
      PROJNO = 'AD3111'
 ORDER BY EMPNO
```

เมื่อต้องการเข้าใจผลลัพธ์จากคำสั่ง SQL ยิ่งขึ้น, ลองจินตนาการว่า SQL จะทำงานผ่านกระบวนการดังต่อไปนี้:

ขั้นตอน 1. SQL จะดำเนินการกับคำสั่ง SELECT แรก:

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
```

เคียวรี่นี้ ส่งคืนตารางผลลัพธ์ชั่วคราวตารางหนึ่ง

EMPNO from CORPDATA.EMPLOYEE
000060
000150
000160
000170
000180
000190
000200
000210
000220
200170
200220

ขั้นตอน 2. SQL จะดำเนินการกับคำสั่ง SELECT ที่สอง:

```
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
```

เคียวรี่นี้ ส่งคืนตารางผลลัพธ์ชั่วคราวอีกตารางหนึ่ง

EMPNO from CORPDATA.EMPPROJACT
000230
000230
000240

EMPNO from CORPDATA.EMPPROJACT
000230
000230
000240
000230
000150
000170
000190
000170
000190
000150
000160
000180
000170
000210
000210

ขั้นตอน 3. SQL จะนำตารางผลลัพธ์ชั่วคราวตารางแรก, เปรียบเทียบกับตารางผลลัพธ์ชั่วคราวตารางที่สอง, และส่งคืนแถวที่ปรากฏในตารางทั้งสองยกเว้นแถวที่ซ้ำกัน, และเรียงลำดับผลลัพธ์.

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
INTERSECT
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
ORDER BY EMPNO
```

เคียวรี่นี้ส่งคืนตาราง ผลลัพธ์รวมซึ่งมีค่าในลำดับจากน้อยไปมาก

EMPNO
000150
000160
000170

EMPNO
000180
000190
000210

ข้อผิดพลาดในการดึงข้อมูล

ใช้ข้อมูลนี้เพื่อทำความเข้าใจเกี่ยวกับวิธีการที่ SQL จัดการ ข้อผิดพลาดที่เกิดขึ้นเมื่อดึงข้อมูล

ถ้า SQL พบว่า คอลัมน์อักขระ หรือคอลัมน์กราฟิกที่มีความยาวเกินกว่าที่จะใส่เข้าไปในตัวแปรโฮสต์แล้ว, SQL จะทำสิ่งต่อไปนี้:

- ตัดข้อมูลขณะที่กำหนดค่าให้กับตัวแปรโฮสต์.
- ตั้งค่า SQLWARN0 และ SQLWARN1 ใน SQLCA ให้เป็นค่า 'W' หรือตั้งค่า RETURNED_SQLSTATE ให้เป็น '01004' ในพื้นที่วินิจฉัย SQL
- ตั้งค่าตัวแปร Indicator, ถ้าถูกเตียมไว้, ให้เป็นความยาวของค่าก่อนที่จะถูกตัด.

ถ้า SQL เจอข้อผิดพลาดในการจับคู่ข้อมูลในขณะที่รันคำสั่งอยู่, SQL จะดำเนินการอย่างหนึ่งอย่างใดต่อไปนี้:

- ถ้าข้อผิดพลาดเกิดขึ้นในนิพจน์รายการ SELECT และตัวแปร Indicator ถูกเตรียมไว้สำหรับนิพจน์ที่มีข้อผิดพลาด:
  - SQL จะคืนค่า -2 สำหรับตัวแปรตัวบ่งชี้ที่สัมพันธ์กับนิพจน์ที่มีข้อผิดพลาด.
  - SQL คืนค่าข้อมูลที่ถูกทั้งหมดสำหรับแถวนั้น.
  - SQL คืนค่า SQLCODE เป็นบวก.
- ถ้าไม่มีตัวแปร Indicator มาให้, SQL จะคืนค่า SQLCODE ที่เป็นค่าลบ.

ข้อผิดพลาดในการจับคู่ข้อมูลประกอบด้วย:

- +138 - อากิวเมนต์ของฟังก์ชันที่ใช้ตัดสตริงมีค่าไม่ถูกต้อง.
- +180 - ซินแทกซ์สำหรับสตริงที่แทนวัน, เวลา, หรือ timestamp มีค่าไม่ถูกต้อง.
- +181 - สตริงที่เป็นตัวแทนวัน, เวลา, หรือ timestamp ไม่ใช่ค่าที่ถูกต้อง.
- +183 - ผลลัพธ์ที่ไม่ถูกต้องจากนิพจน์วัน/เวลา. วันหรือ timestamp ที่ได้ไม่ได้อยู่ในช่วงวันรือ timestamp ที่ถูกต้อง.
- +191 - รูปแบบข้อมูล MIXED ไม่ถูกต้อง.
- +304 - ข้อผิดพลาดในการแปลงตัวเลข (ตัวอย่างเช่น, โอเวอร์โฟลว์, อันเดอร์โฟลว์, หรือหารด้วยศูนย์).
- +331 - ตัวอักขระไม่สามารถถูกแปลงได้.
- +364 - ข้อผิดพลาดในการคำนวณ DECFLOAT
- +420 - ตัวอักขระในอากิวเมนต์ของ CAST มีค่าไม่ถูกต้อง.
- +802 - การแปลงข้อมูลหรือการจับคู่ข้อมูลมีข้อผิดพลาด.

สำหรับการแปลงข้อมูลมีข้อผิดพลาด, SQLCA จะรายงานเฉพาะข้อผิดพลาดสุดท้ายที่ตรวจเจอ. ตัวแปรตัวบ่งชี้ที่สัมพันธ์กับคอลัมน์ผลลัพธ์แต่ละคอลัมน์ที่มีข้อผิดพลาดจะถูกตั้งค่าเป็น -2.

สำหรับข้อผิดพลาดจากการแม็พข้อมูลในการดึงข้อมูลแบบหลายแถว, ข้อผิดพลาดการแม็พที่รายงานเป็นการเตือนแบบ SQLSTATE จะมีพื้นที่เงื่อนไขแยกออกไปในส่วนพื้นที่วินิจฉัยของ SQL . โปรดสังเกตุว่า SQL จะหยุดทำงานเมื่อเกิดข้อผิดพลาดครั้งแรก, ดังนั้นข้อผิดพลาดการแม็พที่ถูกรายงานเป็นข้อผิดพลาด SQLSTATE เพียงอันเดียวจะถูกส่งไปในพื้นที่กาวินิจฉัยของ SQL .

สำหรับข้อความ SQL อื่นๆ, เฉพาะคำเตือน SQLSTATE อันสุดท้ายจะถูกรายงานในพื้นที่การวินิจฉัยของ SQL.

ถ้า Full-Select มี DISTINCT ในรายการการเลือก และคอลัมน์ในรายการที่เลือกมีข้อมูลตัวเลขที่มีค่าไม่ถูกต้องแล้ว, ข้อมูลจะถูกพิจารณาให้เท่ากับค่า null ถ้าการสืบค้นมีการเรียงลำดับ. ถ้าดรรชนีที่มีอยู่ถูกใช้แล้ว, ข้อมูลจะไม่ถูกพิจารณาให้มีค่าเท่า null.

ผลกระทบของข้อผิดพลาดในการจับคู่ข้อมูลใน ORDER BY clause จะขึ้นอยู่กับสถานการณ์:

- ถ้าข้อผิดพลาดในการจับคู่ข้อมูลเกิดขึ้นในขณะที่ข้อมูลถูกกำหนดค่าให้กับตัวแปรโฮสต์ในคำสั่ง SELECT INTO หรือ FETCH, และนิพจน์เดียวกันถูกใช้ใน ORDER BY clause, เร็กคอร์ดผลลัพธ์จะถูกเรียงลำดับอยู่บนพื้นฐานของค่านิพจน์. โดยไม่จัดลำดับเหมือนมีค่าเป็น null (มีค่ามากกว่าค่าอื่นทั้งหมด). ที่เป็นเช่นนี้เนื่องจากนิพจน์ถูกประเมินผลก่อนที่จะทำการกำหนดค่าให้กับตัวแปรโฮสต์.
- ถ้าข้อผิดพลาดการจับคู่ของข้อมูลเกิดขึ้นในขณะที่นิพจน์ในรายการที่เลือกกำลังถูกประเมินผล และนิพจน์เดียวกันนี้ถูกใช้ใน ORDER BY clause, คอลัมน์ผลลัพธ์จะถูกเรียงลำดับตามปกติเหมือนมีค่าเป็น null (มีค่ามากกว่าค่าอื่นทั้งหมด). ถ้า ORDER BY clause ถูกดำเนินการโดยใช้การเรียงลำดับ, คอลัมน์ผลลัพธ์จะถูกเรียงลำดับเหมือนมีค่าเป็น null. ถ้า ORDER BY clause ถูกปฏิบัติโดยใช้ดรรชนีที่มีอยู่, ในกรณีดังต่อไปนี้, คอลัมน์ผลลัพธ์จะถูกเรียงลำดับอยู่บนพื้นฐานของค่าจริงของนิพจน์ในดรรชนี:
  - นิพจน์เป็นคอลัมน์วันที่ที่มีรูปแบบวันที่เป็น *MDY, *DMY, *YMD, หรือ *JUL, และข้อผิดพลาดการแปลงวันที่เกิดขึ้นเพราะวันที่ไม่ได้อยู่ในช่วงวันที่ที่ถูกต้อง.
  - นิพจน์เป็นคอลัมน์ตัวอักขระ และตัวอักขระไม่สามารถแปลงได้.
  - นิพจน์เป็นคอลัมน์เลขทศนิยมและค่าตัวเลขที่ไม่ถูกต้องถูกตรวจพบ.

การแทรกแถวโดยใช้คำสั่ง INSERT

ในการเพิ่มแถวเดียวหรือหลายแถวลงในตาราง หรือมุมมอง ให้ใช้ฟอร์มของคำสั่ง INSERT

คุณสามารถใช้คำสั่ง INSERT เพื่อเพิ่มแถวใหม่เข้าในตารางหรือมุมมอง ด้วยวิธีใดวิธีหนึ่งต่อไปนี้:

- ให้เพิ่มการระบุค่าสำหรับคอลัมน์ในคำสั่ง INSERT.
- การรวมคำสั่ง select ในคำสั่ง INSERT เพื่อแจ้งแก่ SQL ว่า มีข้อมูลสำหรับแถวใหม่ข้อมูลใดอยู่ในตารางหรือมุมมองอื่น.
- การระบุลักษณะบล็อกของคำสั่ง INSERT เพื่อเพิ่มหลายแถว.

สำหรับทุกแถวที่คุณแทรก, คุณต้องให้ค่าสำหรับแต่ละคอลัมน์ที่กำหนดโดยมีแอ็ตทริบิวต์ NOT NULL หากคอลัมน์นั้นไม่มีค่าดีฟอลต์. คำสั่ง INSERT สำหรับเพิ่มแถวเข้าในตารางหรือมุมมองอาจมีลักษณะเช่นนี้:

```
INSERT INTO table-name
   (column1, column2, ... )
VALUES (value-for-column1, value-for-column2, ... )
```

clause INTO จะให้ชื่อคอลัมน์ที่คุณระบุค่า. clause VALUES จะระบุค่าสำหรับแต่ละคอลัมน์ที่มีชื่อใน clause INTO. ค่าที่คุณระบุอาจเป็น:

- **ค่าคงที่.** แทรกค่าที่มีให้ใน clause VALUES .
- **ค่า null.** แทรกค่า null, โดยใช้ คีย์เวิร์ด NULL. คอลัมน์จะต้องถูกกำหนดเป็นแบบมีค่า null ได้ มิฉะนั้นจะเกิดข้อผิดพลาด.
- **ตัวแปรโฮสต์.** แทรกเนื้อหาของ ตัวแปรโฮสต์.
- **เรจิสเตอร์พิเศษ.** แทรกค่าเรจิสเตอร์พิเศษ; ตัวอย่างเช่น, USER.
- **นิพจน์.** แทรกค่าที่เป็นผลลัพธ์จาก นิพจน์.
- **scalar fullselect** แทรกค่าที่เป็นผลลัพธ์ของการรันคำสั่ง select.
- คีย์เวิร์ด **DEFAULT**. แทรกค่าดีฟอลต์ของ คอลัมน์. คอลัมน์ต้องมีค่าดีฟอลต์ ที่กำหนดไว้ให้ หรือยอมให้มีค่า NULL, มิฉะนั้นจะเกิดข้อผิดพลาด.

คุณต้องให้ค่าใน VALUES clause สำหรับแต่ละคอลัมน์ที่มีชื่อในรายการคอลัมน์ของคำสั่ง INSERT. รายการชื่อคอลัมน์อาจข้ามได้ หากคอลัมน์ทั้งหมดในตารางมีค่าที่กำหนดให้ใน clause VALUES. หาก คอลัมน์มีค่าดีฟอลต์, คีย์เวิร์ด DEFAULT อาจใช้เป็นค่าใน VALUES clause. ซึ่งทำให้ค่าดีฟอลต์สำหรับคอลัมน์ถูกใส่ ในคอลัมน์.

ควรจะใส่ชื่อคอลัมน์ทั้งหมดลงในคอลัมน์ที่คุณจะแทรกค่า เนื่องจาก:
- คำสั่ง INSERT ของคุณอธิบายได้ดีกว่า.
- คุณสามารถตรวจสอบได้ว่าคุณจะให้ค่าในลำดับที่เหมาะสม โดยยึดตามชื่อคอลัมน์.
- คุณมีข้อมูลที่เป็นแบบไม่ต้องพึ่งพา (independence) มากขึ้น. ลำดับที่กำหนดคอลัมน์ ในตารางไม่มีผลต่อคำสั่ง INSERT ของคุณ.

หากคอลัมน์ถูกกำหนดให้ยอมให้มีค่า null หรือมีค่าดีฟอลต์, คุณ ไม่จำเป็นต้องใส่ชื่อในรายการชื่อคอลัมน์หรือระบุค่าให้. มันจะมีค่าเป็นดีฟอลต์. หากคอลัมน์ถูกกำหนดให้มีค่าดีฟอลต์, ค่าดีฟอลต์จะถูกใส่ในคอลัมน์. หากมีการระบุ DEFAULT สำหรับ column definition โดยไม่มีค่าดีฟอลต์ที่ชัดเจน, SQL จะใส่ค่าดีฟอลต์ สำหรับประเภทข้อมูลนั้นในคอลัมน์. หากคอลัมน์ไม่มีค่าดีฟอลต์กำหนดไว้ให้, แต่ยอมให้มีค่า null (ไม่มีการระบุ NOT NULL ในคอลัมน์ definition), SQL จะใส่ค่า null ไว้ในคอลัมน์.
- สำหรับคอลัมน์ตัวเลข, ค่าดีฟอลต์คือ 0.
- สำหรับคอลัมน์อักขระความยาวคงที่หรือกราฟิก, ค่าดีฟอลต์คือว่าง.
- สำหรับคอลัมน์ไบนารีความยาวคงที่, ค่าดีฟอลต์คือเลขฐานสิบหกที่เป็นศูนย์.
- สำหรับคอลัมน์อักขระความยาวผันแปร, กราฟิก, หรือไบนารี และคอลัมน์ LOB, ค่าดีฟอลต์คือสตริงความยาวศูนย์.
- สำหรับคอลัมน์วันที่, เวลา, และ timestamp, ค่าดีฟอลต์คือ วันที่, เวลา, หรือ timestamp ปัจจุบัน. เมื่อแทรกกลุ่มเร็กคอร์ด, ค่าวันที่/เวลาที่เป็นดีฟอลต์ จะถูกคัดลอกจากระบบเมื่อมีการเขียนบล็อก. ซึ่งหมายความว่า คอลัมน์จะถูกกำหนดค่าเดียวกันสำหรับแต่ละแถวในบล็อก.
- สำหรับคอลัมน์ DataLink, ค่าดีฟอลต์จะตรงกับ DLVALUE(’’,’URL’,’’).
- สำหรับคอลัมน์แยกประเภท (distinct-type), ค่าดีฟอลต์จะเป็นค่าดีฟอลต์ของ ประเภทซอร์สที่ตรงกัน.
- สำหรับคอลัมน์ ROWID หรือคอลัมน์ที่กำหนด AS IDENTITY, ตัวจัดการฐานข้อมูลจะสร้างค่าดีฟอลต์.

เมื่อโปรแกรมของคุณพยายามแทรกแถวที่ซ้ำกับแถวอื่น ซึ่งมีอยู่ในตารางแล้ว, อาจเกิดข้อผิดพลาดขึ้น. ค่า null หลายค่าอาจถือเป็นค่าซ้ำกันหรือไม่ก็ได้, ทั้งนี้ขึ้นอยู่กับอ็อพชันที่ใช้ เมื่อสร้างดรรชนี.
- หากตารางมีคีย์หลัก, คีย์เฉพาะ, หรือดรรชนีเฉพาะ, จะไม่มีการ แทรกแถว. แต่, SQL จะส่งคืนค่า SQLCODE เป็น -803.
- หากตารางไม่มีคีย์หลัก, คีย์เฉพาะ, หรือดรรชนีเฉพาะ, จะสามารถ แทรกแถวได้โดยไม่เกิดข้อผิดพลาด.

หาก SQL พบข้อผิดพลาดขณะรันคำสั่ง INSERT, จะหยุดการแทรก ข้อมูล. หากคุณระบุ COMMIT(*ALL), COMMIT (*CS), COMMIT(*CHG), หรือ COMMIT(*RR), จะไม่มีการแทรกแถวใดๆ. แถวที่ถูกแทรกโดยคำสั่งนี้แล้ว, ในกรณีที่เป็น INSERT ซึ่งมีคำสั่ง select หรือการแทรกแบบบล็อก, จะถูกลบออก. หากคุณระบุ COMMIT(*NONE), แถวใดๆ ที่ถูกแทรกแล้วจะ *ไม่*ถูกลบออก.

ตารางที่สร้างโดย SQL จะถูกสร้างด้วยพารามิเตอร์ Reuse Deleted Records ซึ่งเป็น *YES. ซึ่งทำให้ตัวจัดการฐานข้อมูลสามารถใช้ซ้ำแถวใดๆ ในตารางที่ ถูกทำเครื่องหมายว่าลบออก. คำสั่ง CHGPF สามารถใช้เปลี่ยนแอ็ตทริบิวต์ เป็น *NO ซึ่งทำให้ INSERT เพิ่มแถวที่ส่วนท้ายของตารางเสมอ

ลำดับของแถวที่แทรกจะไม่รับรองว่าเป็นลำดับ ที่จะถูกเรียกออกมา.

หากมีการแทรกแถวโดยไม่เกิดข้อผิดพลาด, ฟิลด์ SQLERRD(3) ของ SQLCA จะมีค่าเป็น 1.

หมายเหตุ: สำหรับ INSERT ที่ถูกบล็อกหรือสำหรับ INSERT ที่มีคำสั่ง select, จะสามารถแทรกได้มากกว่าหนึ่งแถว. จำนวนแถวที่แทรกจะแสดงอยู่ใน SQLERRD(3) ใน SQLCA. มันยังมีปรากฏจากรายการวินิจฉัย ROW_COUNT ในคำสั่ง GET DIAGNOSTICS.

สิ่งอ้างอิงที่เกี่ยวข้อง

INSERT

การแทรกแถวโดยใช้ VALUES clause

คุณใช้ VALUES clause ในคำสั่ง INSERT เพื่อแทรกแถวเดี่ยวหรือหลายแถวลงในตาราง

ตัวอย่างในที่นี้เป็นการเพิ่มแทรกแถวใหม่ลงในตาราง DEPARTMENT. คอลัมน์สำหรับแถวใหม่แสดงดังต่อไปนี้:

- หมายเลข Department (DEPTNO) คือ 'E31'
- ชื่อ Department (DEPTNAME) คือ 'ARCHITECTURE'
- หมายเลข Manager (MGRNO) คือ '00390'
- รายงานสำหรับ (ADMRDEPT) ฝ่าย 'E01'

ข้อความ INSERT สำหรับแถวใหม่นี้แสดงดังต่อไปนี้:

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, MGRNO, ADMRDEPT)
   VALUES('E31', 'ARCHITECTURE', '00390', 'E01')
```

ท่าน ยังสามารถเพิ่มแทรกแถวหลายแถวลงในตารางโดยใช้ VALUES clause. ตัวอย่างต่อไปนี้ แสดงการแทรกแถวสองแถวลงในตาราง PROJECT. ค่าสำหรับ หมายเลข Project (PROJNO) , ชื่อ Project (PROJNAME), หมายเลข Department (DEPTNO), และ พนักงานที่ รับผิดชอบ (RESPEMP) ถูกกำหนดอยู่ในรายการแสดงค่า. ค่าสำหรับวันเริ่มต้น Project (PRSTDATE) ใช้วันที่ปัจจุบัน. คอลัมน์ที่เหลืออยู่ในตาราง ซึ่งไม่ได้แสดงไว้ในรายการคอลัมน์ได้ถูกกำหนดเป็นค่าดีฟอลต์.

```
INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTDATE)
   VALUES('HG0023', 'NEW NETWORK', 'E11', '200280', CURRENT DATE),
         ('HG0024', 'NETWORK PGM', ''E11", '200310', CURRENT DATE)
```

การแทรกแถวลงในตารางโดยใช้คำสั่ง select

คุณสามารถใช้คำสั่ง select ภายในคำสั่ง INSERT เพื่อแทรกเพิ่มศูนย์แถว, หนึ่งแถว, หรือ มากกว่า ลงในตาราง จากตารางผลลัพธ์ของคำสั่ง select.

ประโยชน์ประการหนึ่งสำหรับคำสั่ง INSERT ประเภทนี้คือเพื่อย้ายข้อมูลเข้าในตาราง ที่คุณสร้างสำหรับข้อมูลสรุป. ตัวอย่างเช่น, สมมติว่า คุณต้องการตารางที่แสดงระยะเวลาการทำงานในโครงการของพนักงานแต่ละคน. สร้างตารางชื่อ EMPTIME โดยมีคอลัมน์ *EMPNUMBER*, *PROJNUMBER*, *STARTDATE*, และ *ENDDATE* และ แล้วใช้คำสั่ง INSERT ต่อไปนี้เพื่อเติมลงในตาราง:

```
INSERT INTO CORPDATA.EMPTIME
   (EMPNUMBER, PROJNUMBER, STARTDATE, ENDDATE)
SELECT EMPNO, PROJNO, EMSTDATE, EMENDATE
  FROM CORPDATA.EMPPROJACT
```

คำสั่ง select ที่อยู่ในคำสั่ง INSERT ไม่ต่างจาก คำสั่ง select ที่คุณใช้เพื่อเรียกข้อมูล. ยกเว้นกรณีที่มี clause ของ FOR READ ONLY, FOR UPDATE, หรือ OPTIMIZE, คุณสามารถใช้คีย์เวิร์ด, คอลัมน์ ฟังก์ชัน, และเทคนิคทั้งหมดที่ใช้เพื่อเรียกข้อมูล. SQL จะแทรกแถวทั้งหมด ที่ตรงตามเงื่อนไขการค้นหาเข้าในตารางที่คุณระบุ. การแทรกแถว จากตารางหนึ่งไปยังอีกตารางหนึ่งไม่มีผลต่อแถวที่มีอยู่ใน ตารางต้นทางหรือตารางเป้าหมาย.

คุณควรพิจารณาสิ่งต่อไปนี้เมื่อแทรกหลายแถวเข้าในตาราง:

หมายเหตุ:

1. จำนวนคอลัมน์ที่แสดงโดยแฝงหรือชัดเจนในคำสั่ง INSERT จะต้องเท่ากับจำนวนคอลัมน์ที่แสดงในคำสั่ง select.
2. ข้อมูลในคอลัมน์ที่คุณเลือกต้องเข้ากันได้กับคอลัมน์ที่คุณจะแทรกคำสั่ง เมื่อใช้ INSERT กับคำสั่ง select.
3. ในกรณีที่คำสั่ง select ซึ่งผนวกรวมใน INSERT ไม่ให้แถวใดๆ, จะมี SQLCODE เป็น 100 เพื่อเตือนคุณว่าไม่มีการแทรกแถว. หาก คุณแทรกแถวได้สำเร็จ, ฟิลด์ SQLERRD(3) ของ SQLCA จะมีจำนวนเต็ม ที่แทนจำนวนแถวซึ่ง SQL แทรกตามจริง. ค่านี้ยังมีปรากฏจากรายการวิเคราะห์ ROW_COUNT ในคำสั่ง GET DIAGNOSTICS .
4. หาก SQL พบข้อผิดพลาดขณะรันข้อความ INSERT, SQL จะหยุด การดำเนินการ. หากคุณระบุ COMMIT (*CHG), COMMIT(*CS), COMMIT (*ALL), หรือ COMMIT(*RR), จะไม่มีการแทรกสิ่งใดเข้าในตารางและจะได้รับ SQLCODE ค่าลบ. หาก คุณระบุ COMMIT(*NONE), แถวใดๆ ที่แทรกก่อนเกิดข้อผิดพลาดจะยังคงอยู่ใน ตาราง.

การแทรกแถวโดยใช้คำสั่ง INSERT ที่ถูกบล็อก

เมื่อใช้คำสั่ง INSERT ที่ถูกบล็อก คุณจะสามารถแทรก หลายแถวเข้าในตารางที่มีคำสั่ง INSERT เพียงคำสั่งเดียว

ข้อความ INSERT แบบบล็อกจะสนับสนุนในทุกภาษา ยกเว้น REXX ข้อมูลที่แทรกเข้าในตารางจะต้องอยู่ใน host structure array. หากใช้ตัวแปรตัวบ่งชี้กับ INSERT ที่ถูกบล็อก, ตัวแปรเหล่านี้ ต้องอยู่ใน host structure array ด้วย.

ตัวอย่างเช่น, ในการเพิ่มพนักงานสิบคนเข้าในตาราง CORPDATA.EMPLOYEE:

```
INSERT INTO CORPDATA.EMPLOYEE
                (EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT)
          10 ROWS VALUES(:DSTRUCT:ISTRUCT)
```

DSTRUCT เป็น host structure array ที่มีองค์ประกอบห้าส่วนซึ่งแสดงใน โปรแกรม. องค์ประกอบห้าส่วนตรงกับ EMPNO, FIRSTNME, MIDINIT, LASTNAME, และ WORKDEPT. DSTRUCT มีขนาดอย่างน้อยสิบเพื่อบรรจุ แถวที่แทรกสิบแถว. ISTRUCT เป็น host structure array ที่แสดงในโปรแกรม. ISTRUCT มีขนาดอย่างน้อยสิบฟิลด์จำนวนเต็มขนาดเล็กสำหรับตัวบ่งชี้.

คำสั่ง INSERT แบบบล็อกจะสนับสนุนสำหรับแอ็พพลิเคชัน SQL แบบไม่กระจาย และสำหรับแอ็พพลิเคชันแบบกระจาย ซึ่งทั้งแอ็พพลิเคชันเซิร์ฟเวอร์ และ application requester เป็นระบบ System i

หลักการที่เกี่ยวข้อง

Embedded SQL programming

การแทรกข้อมูลลงในตารางด้วยข้อจำกัดในการอ้างอิง

เมื่อคุณแทรกข้อมูลลงในตารางด้วยข้อจำกัด ในการอ้างอิง คุณต้องพิจารณากฎเหล่านี้

หากคุณแทรกข้อมูลลงใน parent table ด้วย parent key, SQL จะไม่อนุญาต:

- ให้ทำสำเนาค่า parent key
- หาก parent key คือ primary key, ค่าในคอลัมน์ใดๆ ของ primary key จะเป็นค่า null.

หากคุณแทรกข้อมูลลงในตาราง dependent ด้วย foreign key:

- แต่ละค่าที่ไม่ใช่ค่า null ที่คุณแทรกลงในคอลัมน์ foreign key จะต้องเท่ากับค่าบางค่าใน parent key ที่ตรงกันในตาราง parent.
- หากคอลัมน์ใดๆ ใน foreign key เป็น null, foreign key ทั้งหมดก็จะถูกพิจารณาว่าเป็น null เช่นกัน. หาก foreign key ทั้งหมดที่ประกอบด้วยคอลัมน์เป็น null, คำสั่ง INSERT จะทำงานสำเร็จ (ตราบใดที่ไม่มีการละเมิดตรรชนีเฉพาะ).

เปลี่ยนตารางโครงการแอ็พพลิเคชันตัวอย่าง (PROJECT) เพื่อกำหนด สอง foreign key:

- foreign key บนหมายเลขแผนก (DEPTNO) ซึ่งอ้างอิงถึงตารางแผนก
- foreign key บนหมายเลขพนักงาน (RESPEMP) ซึ่งอ้างอิงถึงตารางพนักงาน.

```
ALTER TABLE CORPDATA.PROJECT ADD CONSTRAINT RESP_DEPT_EXISTS
              FOREIGN KEY (DEPTNO)
              REFERENCES CORPDATA.DEPARTMENT
              ON DELETE RESTRICT

ALTER TABLE CORPDATA.PROJECT ADD CONSTRAINT RESP_EMP_EXISTS
              FOREIGN KEY (RESPEMP)
              REFERENCES CORPDATA.EMPLOYEE
              ON DELETE RESTRICT
```

โปรดสังเกตว่าคอลัมน์ตาราง parent ไม่ได้ถูกระบุไว้ใน REFERENCES clause. ไม่จำเป็นต้องระบุคอลัมน์ดังกล่าว ตราบที่ตารางซึ่งอ้างอิงถึงมี primary key หรือ unique key ซึ่งสามารถใช้เป็น parent key ได้.

ทุกแถวที่ถูกแทรกลงในตาราง PROJECT ต้องมีค่า DEPTNO ที่เท่ากับค่าบางค่าของ DEPTNO ในตารางแผนก. (ยกเว้นค่า null เนื่องจาก DEPTNO ในตารางโปรเจ็กต์ถูกระบุให้เป็น NOT NULL.) แถวนั้นต้องมีค่า RESPEMP ซึ่งเท่ากับค่าบางค่าของ EMPNO ในตารางพนักงานหรือเป็นค่า null.

คำสั่ง INSERT ต่อไปนี้ใช้ไม่ได้เนื่องจากไม่มีค่า DEPTNO ที่ตรงกัน ('A01') ในตาราง DEPARTMENT.

```
INSERT INTO  CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
       VALUES  ('AD3120', 'BENEFITS ADMIN', 'A01', '000010')
```

ในทำนองเดียวกัน คำสั่ง INSERT ต่อไปนี้จะใช้ไม่ได้ เนื่องจาก ไม่มีค่า EMPNO เป็น '000011' ในตาราง EMPLOYEE

```
INSERT INTO CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
       VALUES ('AD3130', 'BILLING', 'D21', '000011')
```

คำสั่ง INSERT ต่อไปนี้สมบูรณ์อย่างครบถ้วนเนื่องจากมีค่า DEPTNO ของ 'E01' ที่ตรงกันในตาราง DEPARTMENT และมีค่า EMPNO ของ'000010' ที่ตรงกันในตาราง EMPLOYEE.

```
INSERT INTO CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
       VALUES ('AD3120', 'BENEFITS ADMIN', 'E01', '000010')
```

การแทรกค่าเข้าในคอลัมน์ identity

คุณสามารถแทรกค่าเข้าในคอลัมน์ identity หรืออนุญาตให้ระบบแทรกค่าให้คุณ.

ตัวอย่างเช่น, ตารางมีคอลัมน์ที่ชื่อ ORDERNO (คอลัมน์ identity), SHIPPED_TO (varchar(36)), และ ORDER_DATE (date). คุณสามารถแทรกแถวเข้าในตารางนี้ได้โดยใช้ข้อความต่อไปนี้:

```
INSERT INTO ORDERS (SHIPPED_TO, ORDER_DATE)
    VALUES ('BME TOOL', 2002-02-04)
```

ในกรณีนี้, ระบบจะสร้างค่าสำหรับ identity column โดยอัตโนมัติ. คุณยังสามารถเขียนข้อความนี้ได้โดยใช้คีย์เวิร์ด DEFAULT:

```
INSERT INTO ORDERS (SHIPPED_TO, ORDER_DATE, ORDERNO)
    VALUES ('BME TOOL', 2002-02-04, DEFAULT)
```

หลังจากแทรก, คุณสามารถใช้ฟังก์ชัน IDENTITY_VAL_LOCAL เพื่อกำหนดค่าที่ระบบกำหนดให้กับคอลัมน์.

บางครั้ง ผู้ใช้จะระบุค่าสำหรับคอลัมน์ identity, เช่น ในคำสั่ง INSERT โดยใช้ SELECT:

```
INSERT INTO ORDERS OVERRIDING USER VALUE
   (SELECT * FROM TODAYS_ORDER)
```

ในกรณีนี้, OVERRIDING USER VALUE จะสั่งให้ระบบไม่สนใจค่าที่มีสำหรับ identity column จาก SELECT และให้สร้างค่าใหม่สำหรับ identity column. ต้องใช้ OVERRIDING USER VALUE หาก identity column สร้างด้วย clause GENERATED ALWAYS; แต่ไม่จำเป็นถ้าใช้ clause GENERATED BY DEFAULT. หากไม่ได้ระบุ OVERRIDING USER VALUE สำหรับ identity column ที่ระบุเป็น GENERATED BY DEFAULT, ค่าที่กำหนดใน SELECT จะนำมาแทรกลงใน identity column.

คุณสามารถบังคับให้ระบบใช้ค่าจาก SELECT ใน identity column แบบ GENERATED ALWAYS ได้โดยระบุ OVERRIDING SYSTEM VALUE. ตัวอย่างเช่น, ใช้คำสั่งต่อไปนี้:

```
INSERT INTO ORDERS OVERRIDING SYSTEM VALUE
   (SELECT * FROM TODAYS_ORDER)
```

คำสั่ง INSERT นี้ใช้ค่าจาก SELECT; ซึ่งจะไม่สร้างค่าใหม่สำหรับ identity column. คุณไม่สามารถให้ค่าสำหรับ identity column ที่สร้างโดยใช้ GENERATED ALWAYS โดยไม่ใช้ clause OVERRIDING SYSTEM VALUE.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การสร้างและเปลี่ยน identity column" ในหน้า 26
ทุกครั้งที่เพิ่มแถวใหม่เข้าไปยังตารางด้วย identity column ระบบจะสร้างค่าของ identity column สำหรับแถวใหม่นี้
ฟังก์ชัน Scalar

การเลือกค่าที่แทรก

คุณสามารถดึงค่าสำหรับแถวที่ถูกแทรกโดย การระบุคำสั่ง INSERT ใน FROM clause ของคำสั่ง SELECT

เมื่อคุณแทรกแถวหนึ่งหรือมากกว่าลงในตาราง คุณสามารถเลือก แถวผลลัพธ์ของการแทรก แถวเหล่านั้นประกอบด้วยค่าใดค่าหนึ่ง ต่อไปนี้:

- ค่าของคอลัมน์ใดๆ ที่สร้างขึ้น เช่น identity, ROWID หรือคอลัมน์ row change timestamp
- ค่าดีฟอลต์ใดๆ ที่ใช้สำหรับคอลัมน์
- ค่าทั้งหมดสำหรับแถวทั้งหมดที่แทรกโดยการแทรกแบบหลายแถว
- ค่าที่ถูกเปลี่ยนโดยทริกเกอร์ก่อนการแทรก

ตัวอย่างต่อไปนี้ใช้ตารางที่ถูกกำหนดดังนี้:

```
CREATE TABLE EMPSAMP
  (EMPNO     INTEGER GENERATED ALWAYS AS IDENTITY,
   NAME      CHAR(30),
   SALARY    DECIMAL(10,2),
   DEPTNO    SMALLINT,
   LEVEL     CHAR(30),
   HIRETYPE  VARCHAR(30) NOT NULL DEFAULT 'New Employee',
   HIREDATE  DATE NOT NULL WITH DEFAULT)
```

เมื่อต้องการแทรกแถวสำหรับพนักงานใหม่ และดู ค่าที่ใช้สำหรับ EMPNO, HIRETYPE และ HIREDATE ให้ใช้คำสั่ง ต่อไปนี้:

```
SELECT EMPNO, HIRETYPE, HIREDATE
  FROM FINAL TABLE ( INSERT INTO EMPSAMP (NAME, SALARY, DEPTNO, LEVEL)
                              VALUES('Mary Smith', 35000.00, 11, 'Associate'))
```

ค่าที่ส่งคืนเป็นค่าที่ถูกสร้างขึ้นสำหรับ EMPNO, 'New Employee' สำหรับ HIRETYPE และวันที่ปัจจุบันสำหรับ HIREDATE

การเปลี่ยนข้อมูลในตารางโดยใช้คำสั่ง UPDATE

ในการอัพเดตข้อมูลในตาราง ให้ใช้คำสั่ง UPDATE

ด้วยคำสั่ง UPDATE, คุณสามารถเปลี่ยนค่าของคอลัมน์ได้มากกว่าหนึ่งค่าในแต่ละแถว ซึ่งตรงตามเงื่อนไขการค้นหาของ WHERE clause. ผลลัพธ์ของคำสั่ง UPDATE จะมีค่าคอลัมน์ที่เปลี่ยนแปลงมากกว่าหนึ่งคอลัมน์ในจำนวนศูนย์ หรือมากกว่าของตาราง (ขึ้นอยู่กับจำนวนแถวที่ตรงกับเงื่อนไขการค้นหาที่ระบุใน WHERE clause). คำสั่ง UPDATE จะเป็นดังนี้:

```
UPDATE table-name
  SET column-1 = value-1,
      column-2 = value-2, ...
  WHERE search-condition ...
```

สมมติว่ามีการย้ายพนักงาน ในการอัพเดตตาราง CORPDATA.EMPLOYEE เพื่อให้สอดคล้องกับการย้ายดังกล่าว ให้รันคำสั่งต่อไปนี้:

```
UPDATE CORPDATA.EMPLOYEE
  SET JOB = :PGM-CODE,
      PHONENO = :PGM-PHONE
  WHERE EMPNO = :PGM-SERIAL
```

ใช้ SET clause เพื่อระบุค่าใหม่สำหรับแต่ละคอลัมน์ ที่คุณต้องการอัพเดต SET clause จะแสดงชื่อคอลัมน์ที่คุณต้องการให้อัพเดตและให้ค่า ที่คุณต้องการให้เปลี่ยนเป็นค่านั้น คุณสามารถระบุ ชนิดค่าดังต่อไปนี้:

- **ชื่อคอลัมน์.** แทนที่ค่าปัจจุบันของคอลัมน์ด้วยเนื้อหาของอีกคอลัมน์ในแถวเดียวกัน.
- **ข้อจำกัด.** แทนที่ค่าปัจจุบันของคอลัมน์ด้วยค่าที่มีให้ใน SET clause.
- **ค่า null.** แทนที่ค่าปัจจุบันของคอลัมน์ด้วยค่า null, โดยใช้คีย์เวิร์ด NULL. คอลัมน์จะต้องถูกกำหนดเป็น แบบมีค่า null ได้เมื่อมีการสร้างตาราง, มิฉะนั้นจะเกิด ข้อผิดพลาด.
- **ตัวแปรโฮสต์.** แทนที่ค่าปัจจุบันของคอลัมน์ด้วยเนื้อหาของตัวแปรโฮสต์.
- **เรจิสเตอร์พิเศษ.** แทนที่ค่าปัจจุบันของคอลัมน์ด้วยค่าเรจิสเตอร์พิเศษ; ตัวอย่างเช่น, USER.
- **นิพจน์.** แทนที่ค่าปัจจุบันของคอลัมน์ด้วยค่าที่เป็นผลลัพธ์จากนิพจน์.
- **scalar fullselect.** แทนที่ค่าปัจจุบันของคอลัมน์ ด้วยค่าที่ subquery ให้มา.
- คีย์เวิร์ด **DEFAULT**. แทนที่ค่าปัจจุบันของคอลัมน์ ด้วยค่าดีฟอลต์ของคอลัมน์. คอลัมน์ต้องมีค่าดีฟอลต์ ที่กำหนดไว้ให้ หรือยอมให้มีค่า NULL, มิฉะนั้นจะเกิดข้อผิดพลาด.

คำสั่ง UPDATE ต่อไปนี้ใช้ค่าที่แตกต่างกันหลายค่า:

```
UPDATE WORKTABLE
   SET COL1 = 'ASC',
       COL2 = NULL,
       COL3 = :FIELD3,
       COL4 = CURRENT TIME,
       COL5 = AMT - 6.00,
       COL6 = COL7
    WHERE EMPNO = :PGM-SERIAL
```

ในการระบุแถวที่จะอัพเดต, ให้ใช้ clause WHERE:

- ในการอัพเดตแถวเดียว, ให้ใช้ WHERE clause ที่เลือกเพียงแถวเดียว.
- ในการอัพเดตหลายแถว, ให้ใช้ clause WHERE ที่เลือกเฉพาะแถว ที่คุณต้องการอัพเดต.

คุณสามารถละเว้น clause WHERE. หากคุณเว้น, SQL จะอัพเดตแต่ละแถวในตาราง หรือมุมมองซึ่งมีค่าที่คุณให้.

หากตัวจัดการฐานข้อมูลพบข้อผิดพลาดขณะรันข้อความ UPDATE ของคุณ, ตัวจัดการจะหยุดและให้ SQLCODE ที่เป็นค่าลบ. หากคุณระบุ COMMIT(*ALL), COMMIT(*CS), COMMIT(*CHG), หรือ COMMIT(*RR), จะไม่มีการเปลี่ยนแถวใดในตาราง (แถวที่ถูกเปลี่ยนโดยข้อความนี้แล้ว, หากมี, จะถูกเรียกคืนเป็นค่า ก่อนหน้านี้). หากระบุ COMMIT(*NONE), แถวใดๆ ที่ถูกเปลี่ยนแล้วจะ*ไม่*ถูกเรียกคืนเป็นค่าก่อนหน้านี้.

หากตัวจัดการฐานข้อมูลไม่พบแถวใดที่ตรงตามเงื่อนไขการค้นหา, จะได้ SQLCODE เป็น +100.

หมายเหตุ: ข้อความ UPDATE อาจได้อัพเดต มากกว่าหนึ่งแถว. จำนวนของแถวที่ถูกอัพเดตถูกแสดงเป็น SQLERRD(3) ของ SQLCA. ค่านี้ยังมีปรากฏจากรายการวิเคราะห์ ROW_COUNT ในข้อความ GET DIAGNOSTICS .

clause SET ของข้อความ UPDATE สามารถใช้ได้หลายวิธีเพื่อกำหนด ค่าแท้จริงที่จะกำหนดในแต่ละแถวที่กำลังอัพเดต. ตัวอย่างต่อไปนี้แสดงแต่ละคอลัมน์และค่าของมัน:

```
UPDATE EMPLOYEE
  SET WORKDEPT = 'D11',
      PHONENO = '7213',
      JOB = 'DESIGNER'
  WHERE EMPNO = '000270'
```

นอกจากนี้ คุณ ยังสามารถเขียนคำสั่ง UPDATE นี้ได้โดยระบุคอลัมน์ทั้งหมด แล้วระบุค่าทั้งหมด:

```
UPDATE EMPLOYEE
  SET (WORKDEPT, PHONENO, JOB)
      = ('D11', '7213', 'DESIGNER')
  WHERE EMPNO = '000270'
```

สิ่งอ้างอิงที่เกี่ยวข้อง

UPDATE

การอัพเดตตารางโดยใช้ scalar-subselect

เมื่อใช้ scalar-subselect คุณจะสามารถอัพเดตคอลัมน์หนึ่งคอลัมน์ หรือมากกว่าได้โดยกำหนดเป็นค่าหนึ่งหรือมากกว่า ซึ่งเลือกได้จากอีกตารางหนึ่ง

ในตัวอย่างต่อไปนี้, พนักงานย้ายไปแผนกอื่นแต่ ยังทำงานโปรเจ็กต์เดิมต่อไป. ตารางพนักงานถูกอัพเดต เพื่อให้มีหมายเลขแผนกใหม่. ขณะนี้จำเป็นต้องอัพเดตตารางโปรเจ็กต์ เพื่อแสดงหมายเลขแผนกใหม่ของพนักงานคนนี้ (หมายเลขพนักงานคือ '000030').

```
UPDATE PROJECT
  SET DEPTNO =
        (SELECT WORKDEPT FROM EMPLOYEE
           WHERE PROJECT.RESPEMP = EMPLOYEE.EMPNO)
  WHERE RESPEMP='000030'
```

เทคนิคเดียวกันนี้ สามารถใช้เพื่ออัพเดตรายการของคอลัมน์ที่มีค่าหลายค่าซึ่งได้มาจากการเลือก ครั้งเดียว.

การอัพเดตตารางที่มีแถวจากตารางอื่น

คุณสามารถอัพเดตแถวทั้งหมดในตารางหนึ่งที่มี ค่าจากแถวหนึ่งในตารางอื่น

สมมติว่าตารางกำหนดการคลาสหลัก จำเป็นต้องอัพเดตความเปลี่ยนแปลงที่เกิดขึ้นในสำเนาของตาราง สำเนางานมีการเปลี่ยนแปลงและผนวกรวมเข้าในตารางหลัก ทุกคืน. ตารางสองตารางมีคอลัมน์ต่างๆ ที่เหมือนกัน และคอลัมน์ CLASS_CODE เป็นคีย์คอลัมน์เฉพาะ.

```
UPDATE CL_SCHED
  SET ROW =
       (SELECT * FROM MYCOPY
          WHERE CL_SCHED.CLASS_CODE = MYCOPY.CLASS_CODE)
```

การอัพเดตนี้จะอัพเดตแถวทั้งหมดใน CL_SCHED ด้วยค่า จาก MYCOPY.

การอัพเดตตารางด้วยข้อจำกัดในการอ้างอิง

หากคุณอัพเดตตาราง parent คุณไม่สามารถ แก้ไข primary key ที่มีแถว dependent อยู่ได้

การเปลี่ยนคีย์เป็นการละเมิดข้อจำกัดในการอ้างอิงของตาราง dependent และทำให้แถวบางแถวไม่มี parent. นอกจากนี้, คุณไม่สามารถระบุค่า null ในส่วนใดๆ ของ primary key.

กฎการอัพเดต

การดำเนินการที่เกิดขึ้นบนตาราง dependent เมื่อมีการใช้ UPDATE บนตาราง parent ขึ้นอยู่กับกฎการอัพเดต ที่ระบุให้กับข้อจำกัดในการอ้างอิง. หากไม่มีการระบุกฎการอัพเดตสำหรับข้อจำกัดในการอ้างอิง, กฎ UPDATE NO ACTION ก็จะถูกนำมาใช้.

UPDATE NO ACTION

ระบุว่าแถวในตาราง parent สามารถอัพเดตได้หากไม่มีแถวอื่นที่ต้องพึ่งพิงแถวนั้น. หากมีแถว dependent อยู่ในความสัมพันธ์, UPDATE ก็จะใช้ไม่ได้. ระบบจะตรวจสอบแถว dependent เมื่อรันคำสั่งเสร็จ.

UPDATE RESTRICT

ระบุว่าแถวในตาราง parent สามารถอัพเดตได้หากไม่มีแถวอื่นที่ต้องพึ่งพิงแถวนั้น. หากมีแถว dependent อยู่ในความสัมพันธ์, UPDATE ก็จะใช้ไม่ได้. ระบบจะตรวจสอบแถว dependent ทันที.

ความแตกต่างอย่างชัดเจนระหว่างกฎ RESTRICT และกฎ NO ACTION สามารถเห็นได้อย่างง่ายดาย เมื่อดูจากปฏิกิริยาโต้ตอบของทริกเกอร์และข้อจำกัดในการอ้างอิง. คุณสามารถระบุทริกเกอร์ให้สั่งงานก่อนหรือหลังการปฏิบัติการ (ซึ่งก็คือคำสั่ง UPDATE, ในกรณีนี้). ค่า *ก่อนทริกเกอร์* จะสั่งงานก่อน UPDATE จะทำงานและก่อนการตรวจสอบข้อจำกัดใดๆ. ค่า*หลังทริกเกอร์* ถูกสั่งงานหลังจากที่ UPDATE ทำงานแล้ว, และหลังกฎข้อจำกัด RESTRICT (โดยที่มีการตรวจสอบทันที), แต่สั่งงานก่อนกฎข้อจำกัด NO ACTION (โดยที่มีการตรวจสอบเมื่อสิ้นสุดคำสั่ง). ทริกเกอร์และกฎจะเกิดขึ้นตามลำดับต่อไปนี้:

1. ค่า*ก่อนทริกเกอร์* จะถูกสั่งงานก่อน UPDATE และก่อนกฎข้อจำกัด RESTRICT หรือ NO ACTION.
2. ค่า*หลังทริกเกอร์* จะถูกสั่งงานหลังกฎข้อจำกัด RESTRICT, แต่สั่งงานก่อนกฎ NO ACTION.

หากคุณอัพเดตตาราง*dependent*, ค่า foreign key ใดๆ ที่ไม่ใช่ null ที่คุณเปลี่ยนต้องตรงกับ primary key สำหรับแต่ละความสัมพันธ์โดยที่ตารางเป็นแบบ dependent. ตัวอย่างเช่น, หมายเลขแผนกในตารางพนักงานขึ้นอยู่กับหมายเลขแผนกในตารางแผนก. คุณสามารถกำหนดค่าพนักงานให้เป็นไม่มีแผนกได้ (ค่า null), แต่กำหนดค่าพนักงานให้แผนกที่ไม่มีอยู่จริงไม่ได้.

หาก UPDATE ตารางโดยใช้ข้อจำกัดในการอ้างอิงล้มเหลว, การเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นระหว่างการอัพเดตจะไม่สมบูรณ์.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การทำเจอร์นัล" ในหน้า 132

การสนับสนุนเจอร์นัลของ DB2 for i5/OS จะช่วย เป็นหลักฐานการตรวจสอบและการกู้คืนทั้งแบบ forward และ backward

"Commitment control" ในหน้า 133

การสนับสนุน DB2 for i5/OS commitment control จะรวบรวมวิธีการประมวลผลกลุ่มของการเปลี่ยนแปลงของฐานข้อมูล เช่น อัพเดต, แทรก หรือลบการดำเนินการ หรือการดำเนินการ data definition language (DDL) ใน ลักษณะของหน่วยการทำงานเดี่ยว (หรือที่เรียกว่า *transaction*)

ตัวอย่าง: กฎ UPDATE:

ตัวอย่างเหล่านี้อธิบายถึงกฎ UPDATE สำหรับ ตารางที่มีข้อจำกัดการอ้างอิง

ตัวอย่างเช่น คุณไม่สามารถอัพเดตหมายเลขแผนก จากตาราง DEPARTMENT ได้ หากตารางดังกล่าวยังมีชื่อบางโครงการอยู่ซึ่งอธิบายด้วยแถว dependent ในตาราง PROJECT

คำสั่ง UPDATE ต่อไปนี้ใช้ไม่ได้เนื่องจาก ตาราง PROJECT มีแถวซึ่งต้องอิงกับ DEPARTMENT.DEPTNO ที่ประกอบด้วยค่า 'D01' (แถวดังกล่าวถูกวางเป้าหมายด้วยคำสั่ง WHERE) ถ้าคำสั่ง UPDATE ใช้ได้ ข้อจำกัดที่อ้างถึงระหว่างตาราง PROJECT และ DEPARTMENT จะยกเลิกไป

```
UPDATE CORPDATA.DEPARTMENT
       SET DEPTNO = 'D99'
       WHERE DEPTNAME = 'DEVELOPMENT CENTER'
```

คำสั่งต่อไปนี้ใช้ไม่ได้เนื่องจากเป็นการละเมิดข้อจำกัดในการอ้างอิงที่มีอยู่ระหว่าง primary key DEPTNO ใน DEPARTMENT และ foreign key DEPTNO ใน PROJECT:

```
UPDATE CORPDATA.PROJECT
       SET DEPTNO = 'D00'
       WHERE DEPTNO = 'D01';
```

คำสั่งพยายามที่จะเปลี่ยนหมายเลขแผนก D01 ทั้งหมดเป็นหมายเลขแผนก D00. เนื่องจาก D00 ไม่ใช่ค่า primary key DEPTNO ใน DEPARTMENT คำสั่งจึงใช้ไม่ได้

การอัพเดต identity column

คุณสามารถอัพเดตค่าใน identity column ให้เป็นค่าที่ระบุ หรือให้ระบบสร้างค่าใหม่.

ตัวอย่างเช่น, เมื่อใช้ตารางที่มีคอลัมน์ชื่อ ORDERNO (identity column), SHIPPED_TO (varchar(36)), และ ORDER_DATE (date), คุณสามารถอัพเดตค่าใน identity column ได้โดยใช้คำสั่งต่อไปนี้:

```
UPDATE ORDERS
  SET (ORDERNO, ORDER_DATE)=
       (DEFAULT, 2002-02-05)
  WHERE SHIPPED_TO = 'BME TOOL'
```

ระบบจะสร้างค่า สำหรับ identity column โดยอัตโนมัติ. คุณสามารถแทนทับค่าที่ระบบสร้างให้โดยใช้ clause OVERRIDING SYSTEM VALUE:

```
UPDATE ORDERS OVERRIDING SYSTEM VALUE
  SET (ORDERNO, ORDER_DATE)=
       (553, '2002-02-05')
  WHERE SHIPPED_TO = 'BME TOOL'
```

สิ่งอ้างอิงที่เกี่ยวข้อง

"การสร้างและเปลี่ยน identity column" ในหน้า 26

ทุกครั้งที่เพิ่มแถวใหม่เข้าไปยังตารางด้วย identity column ระบบจะสร้างค่าของ identity column สำหรับแถวใหม่นี้

การอัพเดตข้อมูลตามที่ดึงมาจากตาราง

คุณสามารถอัพเดตแถวของข้อมูลตามที่คุณได้ดึงออกมาโดยใช้เคอร์เซอร์.

ในข้อความเลือก, ให้ใช้ FOR UPDATE OF ตามด้วยรายการคอลัมน์ที่อัพเดตได้. แล้ว ใช้ข้อความ UPDATE ที่ควบคุมด้วยเคอร์เซอร์. clause WHERE CURRENT OF จะระบุชื่อ เคอร์เซอร์ซึ่งชี้ไปยังแถวที่คุณต้องการอัพเดต. หากไม่ได้ระบุ clause FOR UPDATE OF, ORDER BY, FOR READ ONLY, หรือ SCROLL แบบที่ไม่มี clause DYNAMIC ไว้, คอลัมน์ทั้งหมดสามารถอัพเดตได้.

หากมีการระบุและรันข้อความ FETCH แบบหลายแถว, เคอร์เซอร์ จะอยู่ที่แถวสุดท้ายของบล็อก. ดังนั้น, หากมีการระบุ clause WHERE CURRENT OF ในข้อความ UPDATE, แถวสุดท้ายในบล็อก จะถูกอัพเดต. หากต้องอัพเดตแถวภายในบล็อก, ก่อนอื่นโปรแกรมต้อง วางเคอร์เซอร์ไว้ที่แถวนั้น. แล้วจึงระบุ UPDATE WHERE CURRENT OF ได้. พิจารณาตัวอย่างนี้:

ตารางที่ 21. การอัพเดตตาราง

Scrollable Cursor SQL Statement	Comments
EXEC SQL **DECLARE** THISEMP **DYNAMIC SCROLL CURSOR FOR** **SELECT** EMPNO, WORKDEPT, BONUS **FROM** CORPDATA.EMPLOYEE **WHERE** WORKDEPT = 'D11' **FOR UPDATE OF** BONUS END-EXEC.	
EXEC SQL **OPEN** THISEMP END-EXEC.	
EXEC SQL **WHENEVER NOT FOUND** GO TO CLOSE-THISEMP END-EXEC.	
EXEC SQL **FETCH NEXT FROM** THISEMP **FOR** 5 **ROWS** **INTO** :DEPTINFO :IND-ARRAY END-EXEC.	DEPTINFO และ IND-ARRAY ถูกประกาศใน โปรแกรมเป็น host structure array และ indicator array.
... ตรวจสอบว่ามีพนักงานในแผนก D11 รับโบนัสน้อยกว่า 500.00 ดอลลาร์หรือไม่. หากเป็นเช่นนั้น, ให้อัพเดตเร็กคอร์ดนั้นใหม่ให้ค่า น้อยที่สุดเป็น 500.00 ดอลลาร์.	
EXEC SQL **FETCH** RELATIVE :NUMBACK **FROM** THISEMP END-EXEC.	... ระบุตำแหน่งเร็กคอร์ดในบล็อกเพื่ออัพเดตโดยดึงข้อมูลออกในลำดับกลับกัน.
EXEC SQL **UPDATE** CORPDATA.EMPLOYEE **SET** BONUS = 500 **WHERE CURRENT OF** THISEMP END-EXEC.	... อัพเดตโบนัสสำหรับพนักงานในแผนก D11 ผู้ได้น้อยกว่าอัตราใหม่ที่น้อยที่สุด 500.00 ดอลลาร์.

ตารางที่ 21. การอัพเดตตาราง (ต่อ)

Scrollable Cursor SQL Statement	Comments
EXEC SQL **FETCH** RELATIVE :NUMBACK **FROM** THISEMP **FOR** 5 **ROWS** **INTO** :DEPTINFO :IND-ARRAY END-EXEC.	... วางตำแหน่งที่ส่วนเริ่มต้นของ บล็อกเดียวกันที่ถูกดึงข้อมูลออกมาแล้ว และดึงข้อมูลออกจากบล็อกอีกครั้ง. (NUMBACK -(5 - NUMBACK - 1))
... แบรนช์ (branch) กลับเพื่อพิจารณาว่ามี พนักงานอื่นในกลุ่มเร็กคอร์ดมีโบนัสน้อยกว่า 500.00 ดอลลาร์อีกหรือไม่. ... แบรนช์กลับ เพื่อดึงข้อมูลออกและดำเนินการกับบล็อกต่อไปของแถว.	
CLOSE-THISEMP. EXEC SQL **CLOSE** THISEMP END-EXEC.	

สิ่งอ้างอิงที่เกี่ยวข้อง

"การใช้เคอร์เซอร์" ในหน้า 252

เมื่อ SQL รันคำสั่ง SELECT แถวผลลัพธ์ จะประกอบขึ้นจากตารางผลลัพธ์ เคอร์เซอร์จะแสดงวิธีการเข้าถึงตารางผลลัพธ์

การลบแถวออกจากตารางโดยใช้คำสั่ง DELETE

ในการลบแถวออกจากตาราง, ให้ใช้คำสั่ง DELETE.

เมื่อคุณ delete แถว คุณจะลบออกทั้งแถว DELETE จะไม่ลบเพียงบางคอลัมน์ ออกจากแถว ผลของคำสั่ง DELETE คือการลบแถวจำนวนศูนย์แถวหรือ มากกว่าออกจากตาราง ทั้งนี้ขึ้นอยู่กับว่ามีแถวที่ตรงตามเงื่อนไขการค้นหา ที่ระบุใน WHERE clause เท่าใด หากคุณละเว้น WHERE clause จากคำสั่ง DELETE, SQL จะลบแถวทั้งหมดออกจากตาราง คำสั่ง DELETE มีลักษณะเช่นนี้:

```
DELETE FROM table-name
  WHERE search-condition ...
```

ตัวอย่างเช่น สมมติว่าแผนก D11 ถูกย้ายไปที่อื่น คุณลบ แต่ละแถวในตาราง CORPDATA.EMPLOYEE ที่มีค่า WORKDEPT ของ D11 ดังต่อไปนี้:

```
DELETE FROM CORPDATA.EMPLOYEE
  WHERE WORKDEPT = 'D11'
```

clause WHERE จะบอก SQL ว่าแถวใดที่คุณต้องการลบออกจากตาราง. SQL จะลบแถวทั้งหมดที่ตรงตามเงื่อนไขการค้นหาจากตารางฐาน. การลบแถวจากมุมมองจะเป็นการลบแถวออกจากตารางฐาน. คุณสามารถละเว้น clause WHERE, แต่ไม่ควรจะรวมไว้, เนื่องจากคำสั่ง DELETE ที่ไม่มี clause WHERE จะลบแถวทั้งหมดจากตารางหรือมุมมอง. ในการลบ definition ตาราง และเนื้อหาตาราง, ให้ใช้คำสั่ง DROP.

หาก SQL พบข้อผิดพลาดขณะรันคำสั่ง DELETE ของคุณ, จะหยุดการลบข้อมูลและให้ SQLCODE ที่เป็นลบ. หากคุณระบุ COMMIT(*ALL), COMMIT(*CS), COMMIT(*CHG), หรือ COMMIT(*RR), จะไม่มีการลบแถวใดในตาราง (แถวที่ถูกลบด้วยคำสั่งนี้แล้ว, หากมี, จะถูกเรียกคืนเป็นค่าก่อนหน้านี้). หากไม่มีการระบุ COMMIT(*NONE), แถวใดๆ ที่ถูกลบแล้วจะ*ไม่*ถูกเรียกคืนเป็นค่าก่อนหน้านี้.

หาก SQL ไม่พบแถวใดที่ตรงตามเงื่อนไขการค้นหา, จะได้ SQLCODE เป็น +100.

หมายเหตุ: คำสั่ง DELETE อาจใช้ลบแถวออกมากกว่าหนึ่งแถว . จำนวนของแถวที่ถูกลบจะแสดงอยู่ใน SQLERRD(3) ของ SQLCA. ค่านี้ยังมีปรากฏจากรายการวิเคราะห์ ROW_COUNT ในคำสั่ง GET DIAGNOSTICS .

สิ่งอ้างอิงที่เกี่ยวข้อง

DROP

DELETE

การลบแถวออกจากตารางด้วยข้อจำกัดในการอ้างอิง

ถ้าตารางมี primary key แต่ไม่มี dependent หรือถ้าตารางมีเพียงแค่ foreign key เท่านั้น แต่ไม่มี primary key คำสั่ง DELETE จะ ทำงานเหมือนกับกรณีของตารางที่ไม่มีข้อจำกัดในการอ้างอิง ถ้าตารางมี primary key และเป็นตาราง dependent คำสั่ง DELETE จะทำงาน ตามกฎการลบที่ระบุสำหรับตารางนั้น

การดำเนินการจะต้องเป็นไปตามกฎการลบทั้งหมดของความสัมพันธ์ที่ได้รับผลกระทบเพื่อให้การลบสำเร็จ. หากมีการละเมิดข้อจำกัดในการ อ้างอิง DELETE จะใช้ไม่ได้

สิ่งที่จะดำเนินการบนตาราง dependent เมื่อ DELETE ทำงานบนตาราง parent ขึ้นอยู่กับกฎการลบที่ระบุสำหรับข้อจำกัดในการอ้างอิง. หากไม่มีการกำหนดกฎการลบ, กฎ DELETE NO ACTION จะถูกนำมาใช้.

DELETE NO ACTION

ให้ระบุว่าแถวในตารางหลักสามารถลบออกได้ ถ้าไม่มีแถวอื่นๆ อ้างอิงถึงมัน. หากมีแถว dependent อยู่ในความสัมพันธ์ , DELETE จะใช้ไม่ได้. ระบบจะตรวจสอบแถว dependent เมื่อรันคำสั่งเสร็จ.

DELETE RESTRICT

ให้ระบุว่าแถวในตารางหลักสามารถลบออกได้ ถ้าไม่มีแถวอื่นๆ อ้างอิงถึงมัน. หากมีแถว dependent อยู่ในความสัมพันธ์ , DELETE จะใช้ไม่ได้. ระบบจะตรวจสอบแถว dependent ทันที.

ตัวอย่างเช่น, คุณไม่สามารถลบแผนกจากตารางแผนกได้ หากตารางดังกล่าวยังมีชื่อบางโครงการซึ่งอธิบายด้วยแถว dependent ในตารางโครงการ.

DELETE CASCADE

ระบุว่าแถวที่ถูกกำหนดในตาราง parent จะถูกลบออกเป็นอันดับแรก. จากนั้น, แถว dependent จะถูกลบออก.

ตัวอย่างเช่น, คุณสามารถลบแผนกได้ด้วยการลบแถวของแผนกในตารางแผนกออก. การลบแถวออกจากตารางแผนกยังเป็นการลบ:

- แถวของทุกแผนกที่รายงานมายังตารางแผนก
- ทุกแผนกที่รายงานมายังแผนกเหล่านั้นและต่อจากนั้น.

DELETE SET NULL

ระบุว่าแต่ละคอลัมน์ที่มีค่าเป็น null ได้ใน foreign key ของแต่ละแถว dependent ถูกตั้งให้เป็นค่าดีฟอลต์. หมายความ

ว่าคอลัมน์จะถูกตั้งเฉพาะให้เป็นค่าดีฟอลต์หากคอลัมน์นั้นเป็นเมมเบอร์ของคีย์แปลกปลอมที่อ้างอิงถึงแถวที่ถูกลบออก. เฉพาะแถว dependent ที่อยู่ในชั้นถัดมาเท่านั้นที่ได้รับผลกระทบ.

DELETE SET DEFAULT

ระบุว่าแต่ละคอลัมน์ของ foreign key ในแต่ละแถว dependent ถูกตั้งให้เป็นค่าดีฟอลต์. หมายความว่าคอลัมน์จะถูกตั้งเฉพาะให้เป็นค่าดีฟอลต์หากคอลัมน์นั้นเป็นเมมเบอร์ของคีย์แปลกปลอมที่อ้างอิงถึงแถวที่ถูกลบออก. เฉพาะแถว dependent ที่เป็นอยู่ในชั้นถัดมาเท่านั้น ที่ได้รับผลกระทบ.

ตัวอย่างเช่น, คุณสามารถลบพนักงานออกจากตารางพนักงาน (EMPLOYEE) ได้แม้ว่าพนักงานนั้นจะบริหารบางแผนกก็ตาม. ในกรณีนี้, ค่า MGRNO สำหรับพนักงานแต่ละคนซึ่งรายงานไปยังผู้จัดการแผนกคนนี้จะถูกตั้งเป็นค่าเปล่าในตารางแผนก (DEPARTMENT). หากมีการระบุค่าดีฟอลต์อื่นๆ บางค่าในการสร้างตาราง, ค่านั้นจะถูกนำไปใช้.

เนื่องจากมีการระบุข้อจำกัด REPORTS_TO_EXISTS สำหรับตารางแผนกไว้.

หากตารางในลำดับชั้นถัดไปมีกฎการลบ RESTRICT หรือ NO ACTION และพบแถวที่ว่าไม่สามารถลบแถวในลำดับชั้นถัดมาได้, DELETE ทั้งหมดจะใช้ไม่ได้.

เมื่อรันคำสั่งนี้ด้วยโปรแกรม, จำนวนแถวที่ถูกลบออกจะถูกส่งคืนมาใน SQLERRD(3) ใน SQLCA. จำนวนนี้มีเฉพาะ จำนวนแถวที่ถูกลบออกในตารางซึ่งระบุในคำสั่ง DELETE. แต่ไม่รวมถึงแถวที่ถูกลบออกตามกฎ CASCADE. SQLERRD(5) ใน SQLCA ประกอบด้วยแถวที่ได้รับผลกระทบโดยข้อจำกัดในการอ้างอิงในตารางทั้งหมด. ค่า SQLERRD(3) ยังมีปรากฏจากรายการ ROW_COUNT ในข้อความ GET DIAGNOSTICS . ค่า SQLERRD(5) ยังมีปรากฏจาก รายการ DB2_ROW_COUNT_SECONDARY .

ความแตกต่างอย่างชัดเจนระหว่างกฎ RESTRICT และ NO ACTION สามารถเห็นได้อย่างง่ายดาย เมื่อดูที่ปฏิกิริยาโต้ตอบของทริกเกอร์และข้อจำกัดในการอ้างอิง. คุณสามารถระบุทริกเกอร์ให้สั่งงานก่อนหรือหลัง การปฏิบัติการ (ซึ่งก็คือคำสั่ง DELETE, ในกรณีนี้). ค่า *ก่อนทริกเกอร์* จะสั่งงานก่อน DELETE จะทำงานและก่อนการตรวจสอบข้อจำกัดใดๆ. ค่า *หลังทริกเกอร์* ถูกสั่งงานหลังจากที่ DELETE ทำงาน, และหลังกฎข้อจำกัด RESTRICT (โดยที่มีการตรวจสอบทันที), แต่ก่อนกฎข้อจำกัด NO ACTION (โดยที่มีการตรวจสอบเมื่อสิ้นสุดคำสั่ง). ทริกเกอร์และกฎจะเกิดขึ้นตามลำดับต่อไปนี้:

1. ค่า*ก่อนทริกเกอร์* จะถูกสั่งงานก่อน DELETE และก่อนกฎข้อจำกัด RESTRICT หรือ NO ACTION.
2. ค่า*หลังทริกเกอร์* จะถูกสั่งงานหลังกฎข้อจำกัด RESTRICT, แต่สั่งงานก่อนกฎ NO ACTION.

ตัวอย่าง: กฎ DELETE:

สมมติว่าการลบแผนกออกจากตาราง DEPARTMENT เป็นการตั้งค่า WORKDEPT ในตาราง EMPLOYEE ให้เป็น null สำหรับพนักงานทุกคนที่ถูกมอบหมาย ให้กับแผนกนั้น

พิจารณาคำสั่ง DELETE ต่อไปนี้:

```
DELETE FROM CORPDATA.DEPARTMENT
       WHERE DEPTNO = 'E11'
```

ดูจากตาราง และข้อมูลใน “DB2 for i5/OS ตารางตัวอย่าง” ในหน้า 329, มีแถวหนึ่งแถวที่ถูกลบออกจากตาราง DEPARTMENT, และตาราง EMPLOYEE ได้รับการอัพเดตเพื่อให้ตั้งค่า WORKDEPT เป็นค่าเริ่มต้นโดยที่ค่าเท่ากับ 'E11'. เครื่องหมายคำถาม ('?') ในข้อมูลตัวอย่างต่อไปนี้แสดงถึงค่า null ผลลัพธ์ปรากฏขึ้นดังต่อไปนี้:

ตารางที่ 22. ตาราง DEPARTMENT. เนื้อหาของตารางหลังจากที่คำสั่ง DELETE เสร็จสมบูรณ์.

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER	?	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E01	SUPPORT SERVICES	000050	A00
E21	SOFTWARE SUPPORT	000100	E01
F22	BRANCH OFFICE F2	?	E01
G22	BRANCH OFFICE G2	?	E01
H22	BRANCH OFFICE H2	?	E01
I22	BRANCH OFFICE I2	?	E01
J22	BRANCH OFFICE J2	?	E01

โปรดสังเกตว่า ไม่มีการลบแบบต่อเรียงในตาราง DEPARTMENT เนื่องจากไม่มีแผนกใดที่รายงานไปยังแผนก 'E11'.

ต่อไปนี้คือข้อมูลเก็บจากหน่วยความจำของส่วนหนึ่งของตาราง EMPLOYEE ที่ได้รับผลกระทบก่อนและหลังจากที่คำสั่ง DELETE จะเสร็จสมบูรณ์

ตารางที่ 23. ตาราง EMPLOYEE บางส่วน. เนื้อหาบางส่วนก่อนหน้าคำสั่ง DELETE.

EMPNO	FIRSTNME	MI	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1960-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	E11	0997	1967-03-24
000290	JOHN	R	PARKER	E11	4502	1980-05-30
000300	PHILIP	X	SMITH	E11	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07

ตารางที่ 23. ตาราง EMPLOYEE บางส่วน (ต่อ). เนื้อหาบางส่วนก่อนหน้าคำสั่ง DELETE.

EMPNO	FIRSTNME	MI	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5696	1947-05-05

ตารางที่ 24. ตาราง EMPLOYEE บางส่วน. เนื้อหาบางส่วนหลังคำสั่ง DELETE.

EMPNO	FIRSTNME	MI	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1960-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	?	0997	1967-03-24
000290	JOHN	R	PARKER	?	4502	1980-05-30
000300	PHILIP	X	SMITH	?	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	?	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5696	1947-05-05

การใช้เคียวรี่ย่อย

คุณสามารถใช้เคียวรี่ย่อยในเงื่อนไขการค้นหาเพื่อเป็นอีกทางหนึ่งในการ เลือกข้อมูล เคียวรี่ย่อยสามารถใช้ได้ทุกที่ที่นิพจน์สามารถใช้งานได้.

ตามหลักการแล้ว จะมีการประเมินผลเคียวรี่ เมื่อใดก็ตามที่แถวหรือกลุ่มแถวใหม่ ถูกประมวลผล ที่จริงแล้ว, หากเคียวรี่ย่อยของทุกแถวหรือทุกกลุ่มเป็นแบบเดียวกัน, เคียวรี่ย่อยนั้นก็จะถูกประเมินผลเพียงครั้งเดียว. เคียวรี่ย่อยแบบนี้เรียกว่า **ภาวะที่ไม่สัมพันธ์กัน.**

เคียวรี่ย่อยบางอย่างจะส่งคืนค่าที่แตกต่างกันจากแถวสู่แถวหรือจากกลุ่มสู่กลุ่ม. กลไกข้างต้นนี้เรียกว่า **ภาวะที่สัมพันธ์กัน**, และเคียวรี่ย่อยดังกล่าวเรียกว่า เคียวรี่ย่อย**ที่สัมพันธ์กัน.**

สิ่งอ้างอิงที่เกี่ยวข้อง

"นิพจน์ใน WHERE clause" ในหน้า 47
นิพจน์ใน WHERE clause ใช้ตั้งชื่อหรือระบุ สิ่งที่คุณต้องการเปรียบเทียบกับสิ่งอื่น

“การนิยามเงื่อนไขการค้นหาที่ซับซ้อน” ในหน้า 61
นอกจากเพรดิเคตการเปรียบเทียบพื้นฐานแล้ว เช่น = และ > เงื่อนไขการค้นหาสามารถมีเพรดิเคต BETWEEN, IN, EXISTS, IS NULL และ LIKE

เคียวรี่ย่อยในคำสั่ง SELECT

เคียวรี่ย่อยช่วยกรองเงื่อนไขการค้นหาเพิ่มเติม

ใน WHERE และ HAVING clause แบบง่ายๆ, คุณสามารถระบุเงื่อนไขการค้นหาด้วยการใช้ ค่า literal, ชื่อคอลัมน์, นิพจน์, หรือ register พิเศษ. ในเงื่อนไขการค้นหาเหล่านั้น, คุณทราบว่าคุณกำลังค้นหาค่าเฉพาะ. แต่ในบางครั้ง, คุณไม่สามารถป้อนค่านั้นได้จนกว่าจะได้เรียกข้อมูลอื่นๆ ออกมาจากตารางก่อน. ตัวอย่างเช่น, สมมติว่าคุณต้องการรายการหมายเลขพนักงาน, ชื่อ, และโค้ดงานของพนักงานทั้งหมดที่ทำงานในแต่ละโครงการ, เช่นหมายเลขโครงการ MA2100. คุณสามารถเขียนส่วนแรกของคำสั่งได้อย่างง่ายดายดังนี้:

```
SELECT EMPNO, LASTNAME, JOB
FROM CORPDATA.EMPLOYEE
WHERE EMPNO ...
```

แต่คุณไม่สามารถดำเนินการต่อไปได้เนื่องจากตาราง CORPDATA.EMPLOYEE ไม่ได้รวมข้อมูลหมายเลขโครงการไว้ด้วย. คุณไม่ทราบว่าพนักงานคนใดทำงานโครงการ MA2100 อยู่ หากไม่ได้ใช้คำสั่ง SELECT อีกคำสั่งหนึ่งให้กับตาราง CORPDATA.EMP_ACT.

ด้วยการใช้ SQL, คุณสามารถซ้อนคำสั่ง SELECT หนึ่งคำสั่งภายในอีกหนึ่งคำสั่งเพื่อแก้ปัญหานี้. คำสั่ง SELECT ภายในเรียกว่า **การสืบค้นย่อย** . คำสั่ง SELECT ที่อยู่นอกการสืบค้นย่อยเรียกว่า **outer- level SELECT**. เมื่อใช้การเคียวรี่ย่อย คุณจะสามารถใช้คำสั่ง SQL เพียงคำสั่งเดียวในการดึงข้อมูลหมายเลขพนักงาน, ชื่อ และโค้ดงานของพนักงานซึ่งทำงานโครงการ MA2100 ได้พร้อมกัน:

```
SELECT EMPNO, LASTNAME, JOB
FROM CORPDATA.EMPLOYEE
WHERE EMPNO IN
   (SELECT EMPNO
      FROM CORPDATA.EMPPROJACT
      WHERE PROJNO = 'MA2100')
```

เพื่อให้เกิดความเข้าใจดียิ่งขึ้นว่าจะเกิดผลลัพธ์อะไรขึ้นจากคำสั่ง SQL, ให้จินตนาการว่า SQL เป็นไปตามกระบวนการต่อไปนี้:

ขั้นที่ 1: SQL จะประเมินผลเคียวรี่ย่อยเพื่อรับรายการค่า EMPNO:

```
(SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO= 'MA2100')
```

ผลลัพธ์ใน ตารางผลลัพธ์ชั่วคราวเป็นดังนี้

EMPNO from CORPDATA.EMPPROJACT
000010
000110

ขั้นที่ 2: จากนั้นตารางผลลัพธ์ชั่วคราวจะทำหน้าที่เป็นเสมือนรายการในเงื่อนไขการค้นหาของคำสั่ง outer-level SELECT ซึ่งมีความสำคัญ เนื่องจากคำสั่งนี้เป็นคำสั่งที่รันอยู่:

```
SELECT EMPNO, LASTNAME, JOB
     FROM CORPDATA.EMPLOYEE
     WHERE EMPNO IN
           ('000010', '000110')
```

ตารางผลลัพธ์ขั้นสุดท้ายจะเป็นดังนี้

EMPNO	LASTNAME	JOB
000010	HAAS	PRES
000110	LUCCHESSI	SALESREP

เคียวรี่ย่อยและเงื่อนไขการค้นหา:

เคียวรี่ย่อยอาจเป็นส่วนหนึ่งของเงื่อนไขการค้นหา ซึ่งอยู่ในรูป *operand operator operand* ทั้งนี้ operand อาจเป็นเคียวรี่ย่อยก็ได้

ดังตัวอย่างต่อไปนี้, **operand แรก**คือ EMPNO และ **operator** คือ IN. เงื่อนไขการค้นหาอาจเป็นส่วนหนึ่งของ WHERE หรือ HAVING clause. ในแต่ละ clause อาจประกอบด้วยเงื่อนไขการค้นหาที่มีเคียวรี่ย่อยอยู่มากกว่าหนึ่งเงื่อนไข. เงื่อนไขการค้นหาที่มีเคียวรี่ย่อยอยู่อาจอยู่ระหว่างวงเล็บ, นำหน้าด้วยคีย์เวิร์ด NOT หรือเชื่อมโยงไปยังเงื่อนไขการค้นหาอื่นๆ โดยใช้คีย์เวิร์ด AND และ OR เช่นเดียวกันกับเงื่อนไขการค้นหาอื่นๆ ตัวอย่าง, WHERE clause ของเคียวรี่อาจมีลักษณะดังนี้:

```
WHERE  (subquery1) = X AND (Y > SOME (subquery2) OR Z = 100)
```

เคียวรี่ย่อยนั้นอาจปรากฏอยู่ในเงื่อนไขการค้นหาของเคียวรี่ย่อยอื่นๆ. ลักษณะดังกล่าวเรียกว่าเคียวรี่ย่อย **แบบซ้อนภายใน**ที่อาจเกิดขึ้นในการซ้อนภายในบางระดับ. ตัวอย่างเช่น, เคียวรี่ย่อยซึ่งอยู่ภายในเคียวรี่ย่อยใน outer- level SELECT จะถูกซ้อนอยู่ในระดับที่สอง. คำสั่ง SQL ยอมให้มีการซ้อนภายในได้ทั้งสิ้น 32 ระดับ.

หมายเหตุการใช้บนเคียวรี่ย่อย:

ต่อไปนี้เป็นข้อควรพิจารณาบางประการเกี่ยวกับการใช้เคียวรี่ย่อย เพื่อปรับแต่งเงื่อนไขการค้นหาของคุณ

1. เมื่อทำการซ้อนภายในคำสั่ง SELECT, คุณสามารถใช้เคียวรี่ย่อยได้หลายๆ ครั้งตามต้องการ (1 ถึง 255 เคียวรี่ย่อย), แม้การเพิ่มเคียวรี่ย่อยขึ้นมาแต่ละครั้งจะทำให้ระบบทำงานช้าลง.
2. สำหรับเพรดิเคตที่ใช้คีย์เวิร์ด ALL, ANY, SOME, หรือ EXISTS, จำนวน แถวที่ถูกส่งคืนจากเคียวรี่ย่อยสามารถเป็นได้ตั้งแต่ค่าศูนย์จนถึงค่าหลายๆ ค่า สำหรับเคียวรี่ย่อยอื่นๆ ทั้งหมด, จำนวนแถวที่ถูกส่งคืนจะต้องเป็นศูนย์หรือหนึ่ง.
3. สำหรับเพรดิเคตต่อไปนี้, การเลือกแถวแบบเต็มสามารถใช้กับเคียวรี่ย่อยได้. นี่หมายความว่า เคียวรี่ย่อยสามารถส่งคืนค่าได้มากกว่าหนึ่งค่าสำหรับแถว.
   - เพรดิเคตพื้นฐานกับการเปรียบเทียบที่เท่ากันและไม่เท่ากัน
   - เพรดิเคตที่เป็นจำนวนโดยใช้ =ANY, =ALL, และ =SOME
   - เพรดิเคต IN และ NOT IN

   ถ้าใช้การเลือกแถวแบบเต็ม:
   - รายการที่เลือกต้องไม่มี SELECT *. ซึ่งต้องระบุค่าที่แน่นอน.

- การเลือกแถวแบบเต็มต้องถูกเปรียบเทียบกับนิพจน์แถว. นิพจน์แถว คือรายการของค่าที่อยู่ในวงเล็บ. ซึ่งต้องเป็นจำนวนของค่าเดียวกันกับค่าที่ส่งคืนจากเคียวรี่ย่อยซึ่งอยู่ในนิพจน์แถว.
- นิพจน์แถวสำหรับเพรดิเคต IN หรือ NOT IN ไม่สามารถมีตัวทำเครื่องหมายพารามิเตอร์ที่ไม่ถูกพิมพ์. ใช้ CAST เพื่อจัดหาชนิดข้อมูลผลลัพธ์สำหรับตัวทำเครื่องหมายพารามิเตอร์เหล่านี้.
- เคียวรี่ย่อยไม่สามารถมี UNION, EXCEPT, หรือ INTERSECT หรือการอ้างอิงที่เกี่ยวพันกัน.

4. เคียวรี่ย่อยไม่สามารถมี ORDER BY, FOR READ ONLY, FETCH FIRST *n* ROWS, UPDATE หรือ OPTIMIZE clause

การรวมเคียวรี่ย่อยไว้ใน WHERE หรือ HAVING clause:

คุณสามารถรวมเคียวรี่ย่อยไว้ใน WHERE หรือ HAVING clause โดยใช้การเปรียบเทียบพื้นฐานหรือการเปรียบเทียบเชิงปริมาณ, คีย์เวิร์ด IN หรือคีย์เวิร์ด EXISTS

การดำเนินการเปรียบเทียบพื้นฐาน

คุณสามารถใช้เคียวรี่ย่อยก่อนหรือหลัง comparison operator ใดๆ ก็ได้. เคียวรี่ย่อยสามารถส่งคืนค่าได้เพียงหนึ่งแถวเท่านั้น. และสามารถส่งคืนค่าแถวได้หลายๆ ค่า หากใช้โอเปอเรเตอร์เท่ากับและไม่เท่ากับ. SQL จะเปรียบเทียบค่าแต่ละค่าจากแถวของเคียวรี่ย่อยด้วยค่าที่สอดคล้องกันกับค่าที่อยู่อีกด้านหนึ่งของ comparison operator. ตัวอย่างเช่น เมื่อคุณต้องการค้นหาหมายเลขพนักงาน, ชื่อ และเงินเดือนสำหรับพนักงานที่มีระดับการศึกษาสูงกว่าระดับการศึกษาโดยเฉลี่ยทั่วไปในบริษัท

```
SELECT EMPNO, LASTNAME, SALARY
FROM CORPDATA.EMPLOYEE
WHERE EDLEVEL >
   (SELECT AVG(EDLEVEL)
      FROM CORPDATA.EMPLOYEE)
```

SQL จะประเมินผลเคียวรี่ย่อยก่อน จากนั้นจะแทนที่ผลลัพธ์ใน WHERE clause ของคำสั่ง SELECT. ดังตัวอย่าง, ผลลัพธ์ที่ได้คือระดับการศึกษาโดยเฉลี่ย ของทั้งบริษัท. นอกจากการส่งคืนค่าเพียงแถวเดียวแล้ว, เคียวรี่ย่อยอาจไม่ส่งคืนค่าเลยก็เป็นได้. หากเป็นเช่นนั้น, จะไม่ทราบผลลัพธ์ของการเปรียบเทียบ.

การเปรียบเทียบเชิงปริมาณ (ALL, ANY, และ SOME)

คุณสามารถใช้เคียวรี่ย่อยที่อยู่หลัง comparison operator ต่อด้วยคีย์เวิร์ด ALL, ANY, หรือ SOME. ด้วยวิธีนี้, เคียวรี่ย่อยอาจไม่ส่งคืนค่าแถว, ส่งคืนหนึ่งแถว, หรือส่งคืนมากกว่าหนึ่งแถว, ซึ่งรวมถึงค่า null ด้วย. คุณสามารถใช้ ALL, ANY, และSOME ดังวิธีต่อไปนี้:

- ใช้ ALL เพื่อแสดงว่า ค่าที่คุณป้อนต้องถูกนำไปเปรียบเทียบตามวิธีการข้างต้นกับแถวที่ได้รับจากเคียวรี่ย่อยที่เรียกว่า **ALL**. ตัวอย่างเช่น, สมมติว่าคุณใช้ comparison operator มากกว่า พร้อมกับ ALL:

  ```
  ... WHERE expression > ALL (subquery)
  ```

  เพื่อให้เป็นไปตาม WHERE clause, ค่าของนิพจน์จะต้องมากกว่าผลลัพธ์สำหรับแต่ละแถว (กล่าวคือ, ต้องมากกว่าค่าสูงสุด) ที่ส่งคืนค่าโดยเคียวรี่ย่อย. หากเคียวรี่ย่อยส่งคืนชุดที่ว่างเปล่า (กล่าวคือ, ไม่มีการเลือกแถว), ก็จะถือว่าเป็นไปตามเงื่อนไขเช่นกัน.
- ใช้ ANY หรือ SOME เพื่อแสดงว่า ค่าที่คุณป้อนต้องนำไปเปรียบเทียบตามวิธีการข้างต้นกับแถว *อย่างน้อยหนึ่ง*แถวที่เคียวรี่ย่อยส่งคืน. ตัวอย่างเช่น, ถ้าคุณใช้ comparison operator มากกว่า พร้อมกับ **ANY**:

  ```
  ... WHERE expression > ANY (subquery)
  ```

เพื่อให้เป็นไปตาม WHERE, ค่าในนิพจน์ต้องมากกว่าค่าที่ถูกส่งคืนจากเคียวรี่ย่อยอย่างน้อยหนึ่งแถว (นั่นคือ, ต้องมากกว่าค่าที่ต่ำสุด). หากค่าที่เคียวรี่ย่อยส่งคืนคือชุดค่าที่ว่างเปล่า, ก็ถือว่าไม่เป็นไปตามเงื่อนไข.

หมายเหตุ: ผลลัพธ์ที่ได้เมื่อเคียวรี่ย่อยส่งคืนค่าศูนย์หนึ่งค่า หรือมากกว่า อาจทำให้คุณประหลาดใจ, เว้นแต่ว่า คุณคุ้นเคยกับตรรกะตามรูปแบบอยู่แล้ว.

คีย์เวิร์ด IN

คุณสามารถใช้ IN เพื่อแสดงว่า ค่าในนิพจน์ต้องอยู่ระหว่างแถวที่ส่งคืนโดยเคียวรี่ย่อย. การใช้ IN เท่ากับการใช้ *=ANY* หรือ *=SOME*. ซึ่งได้อธิบายไว้ก่อนหน้านี้แล้ว. คุณยังสามารถใช้คีย์เวิร์ด IN กับคีย์เวิร์ด NOT เพื่อใช้เลือกแถว เมื่อค่าไม่มีอยู่ในแถวที่ส่งคืนโดยเคียวรี่ย่อย. ตัวอย่างเช่น, คุณสามารถระบุ:

```
... WHERE WORKDEPT NOT IN (SELECT ...)
```

คีย์เวิร์ด EXISTS

ในเคียวรี่ย่อยที่นำเสนอมาตั้งแต่ต้น, SQL จะประเมินผลเคียวรี่ย่อยและใช้ผลลัพธ์เป็นส่วนหนึ่งของ WHERE clause ของ outer- level SELECT. ในทางตรงกันข้าม , เมื่อคุณใช้คีย์เวิร์ด EXISTS, SQL จะตรวจสอบว่าเคียวรี่ย่อยส่งคืนค่าตั้งแต่หนึ่งแถวขึ้นไปหรือไม่. หากเป็นเช่นนั้น, แสดงว่าเป็นไปตามเงื่อนไข. หากไม่ส่งคืนแถวเลย, แสดงว่าไม่เป็นไปตามเงื่อนไข. ตัวอย่างเช่น:

```
SELECT EMPNO,LASTNAME
FROM CORPDATA.EMPLOYEE
WHERE EXISTS
   (SELECT *
            FROM CORPDATA.PROJECT
      WHERE PRSTDATE > '1982-01-01');
```

ในตัวอย่าง, เงื่อนไขการค้นหาจะเป็นจริงหากโครงการใดๆ ที่นำเสนอในตาราง CORPDATA.PROJECT มีวันที่เริ่มต้นโดยประมาณซึ่งเป็นหลังวันที่ 1 มกราคม, 1982. โปรดสังเกตว่าตัวอย่างนี้ไม่ได้แสดงประสิทธิภาพที่สมบูรณ์แบบของ EXIST , เนื่องจากผลลัพธ์ที่ได้จะเป็นแบบเดียวกันเสมอสำหรับทุกแถวถ้าตรวจสอบด้วยคำสั่ง outer-level SELECT . ดังนั้น, ผลลัพธ์ที่ได้อาจประกอบข้อมูลทุกแถว, หรือไม่มีเลยแม้แต่แถวเดียว. สำหรับตัวอย่างที่ดีกว่า, ตัวเคียวรี่ย่อยเองควรสัมพันธ์กัน, และเปลี่ยนจากแถวไปยังแถว.

ดังที่แสดงไว้ในตัวอย่าง, คุณไม่จำเป็นต้องระบุชื่อคอลัมน์ในรายการการเลือกของเคียวรี่ย่อยสำหรับ EXISTS clause. แต่, คุณควรโค้ด SELECT *.

คุณยังสามารถใช้คีย์เวิร์ด EXISTS คู่กับคีย์เวิร์ด NOT เพื่อเลือกแถวเมื่อไม่มีข้อมูลหรือเงื่อนไขที่คุณระบุอยู่. คุณสามารถใช้คำสั่งต่อไปนี้:

```
... WHERE NOT EXISTS (SELECT ...)
```

เคียวรี่ย่อยที่สัมพันธ์กัน

เคียวรี่ย่อยที่สัมพันธ์กัน เป็นเคียวรี่ย่อย ที่ SQL จำเป็นต้องใช้ประเมินผลขณะที่ตรวจสอบแถวใหม่แต่ละแถว (WHERE clause) หรือกลุ่มของแถว (HAVING clause) ในคำสั่ง outer-level SELECT

ชื่อและการอ้างอิงที่สัมพันธ์กัน:

การอ้างอิงที่สัมพันธ์กันอาจปรากฏในเงื่อนไขการค้นหาในการสืบค้นย่อย. การอ้างอิงจะอยู่ในรูปของ X.C เสมอ โดยที่ X คือชื่อที่มีความหมายเหมือนกัน และ C คือชื่อของคอลัมน์ในตารางที่ X อ้างถึง

คุณสามารถกำหนดชื่อที่มีความหมายเหมือนกันสำหรับตารางใดๆ ที่ปรากฏอยู่ใน FROM clause. ชื่อการสืบค้นที่สัมพันธ์กันจะถือเป็นชื่อเฉพาะของตารางในเคียวรี่. ชื่อตารางเดียวกันสามารถใช้ได้หลายครั้งภายในเคียวรี่และการเลือกย่อยแบบซ้อนภายในของตาราง. การระบุชื่อต่างๆ ที่มีความหมายเหมือนกันสำหรับการอ้างอิงตารางแต่ละข้อ อาจเป็นการกำหนดตารางเฉพาะที่คอลัมน์อ้างอิงถึงได้.

ชื่อที่มีความหมายเหมือนกันจะถูกกำหนดใน FROM clause ของเคียวรี่. เคียวรี่นี้อาจเป็น outer-level SELECT, หรือเป็นเคียวรี่ย่อยใดๆ ที่มีเคียวรี่ที่อ้างอิง. ตัวอย่าง , สมมติว่า, เคียวรี่ประกอบด้วยเคียวรี่ย่อย A, B, และ C, ซึ่ง A ประกอบด้วย B และ B ประกอบด้วย C. ชื่อที่มีความหมายเหมือนกันที่ใช้ใน C ก็ควรถูกกำหนดใน B, A, หรือ outer-level SELECT เช่นกัน. หากต้องการกำหนดชื่อที่มีความหมายเหมือนกัน, ให้ใส่ชื่อที่มีความหมายเหมือนกันไว้หลังชื่อตารางT. เว้นช่องว่างไว้หนึ่งช่องหรือมากกว่าระหว่างชื่อตารางและชื่อที่มีความหมายเหมือนกันของตาราง, และใส่เครื่องหมายจุลภาคไว้หลังชื่อที่มีความหมายเหมือนกัน หากชื่อนั้นตามด้วยชื่อตารางอีกชื่อหนึ่ง. FROM clause ต่อไปนี้เป็นการกำหนดชื่อที่สัมพันธ์กันคือ TA และ TB สำหรับตาราง TABLEA และ TABLEB, และไม่มีชื่อที่สัมพันธ์กันสำหรับตาราง TABLEC.

```
FROM TABLEA TA, TABLEC, TABLEB TB
```

เคียวรี่ย่อยอาจประกอบด้วยจำนวนการอ้างอิงที่สัมพันธ์กัน. ตัวอย่างเช่น,ชื่อที่สัมพันธ์กันในเงื่อนไขการค้นหาสามารถกำหนดใน outer-level SELECT, ขณะที่อีกชื่อหนึ่งสามารถกำหนดในเคียวรี่ย่อย.

ก่อนที่จะใช้งานเคียวรี่ย่อย, ค่าจากคอลัมน์ที่อ้างอิงจะถูกแทนที่สำหรับการอ้างอิงที่สัมพันธ์กันเสมอ.

ตัวอย่าง: เคียวรี่ย่อยที่สัมพันธ์กันใน WHERE clause:

สมมติว่าคุณต้องการรายชื่อพนักงานทั้งหมดที่มีระดับการศึกษาสูงกว่าระดับการศึกษาโดยเฉลี่ยในแต่ละแผนก. ถ้าต้องการได้ข้อมูลนี้, SQL ต้องค้นหาตาราง CORPDATA.EMPLOYEE.

ในส่วนของพนักงานแต่ละคนในตาราง, SQL ต้องเปรียบเทียบระดับการศึกษาของพนักงานกับระดับการศึกษาโดยเฉลี่ยสำหรับแผนกของพนักงาน. ในการเคียวรี่ย่อย, คุณแจ้งให้ SQL คำนวณระดับการศึกษาโดยเฉลี่ยสำหรับหมายเลขแผนกในแถวปัจจุบัน. ตัวอย่างเช่น:

```
SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
 FROM CORPDATA.EMPLOYEE X
 WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
       FROM CORPDATA.EMPLOYEE
       WHERE WORKDEPT = X.WORKDEPT)
```

เคียวรี่ย่อยที่สัมพันธ์กันจะคล้ายกับเคียวรี่ย่อยที่ไม่สัมพันธ์กัน, ยกเว้นการปรากฏขึ้นของการอ้างอิงที่สัมพันธ์กันหนึ่งครั้งหรือมากกว่า. ในตัวอย่าง, การอ้างอิงเดียวที่สัมพันธ์กันคือ X.WORKDEPT ใน FROM clause ของการเลือกย่อย. ในที่นี้, qualifier *X* คือชื่อที่มีความหมายเหมือนกันที่ถูกกำหนดไว้ใน FROM clause ของคำสั่ง outer SELECT. ใน clause ดังกล่าว, *X* จะเป็นชื่อที่สัมพันธ์กันของตาราง CORPDATA.EMPLOYEE.

ตอนนี้, ให้พิจารณาว่าจะเกิดอะไรขึ้นเมื่อเรียกใช้งานเคียวรี่ย่อยสำหรับแถว CORPDATA.EMPLOYEE ที่ให้มา. ก่อนที่จะเรียกใช้งาน, X.WORKDEPT จะถูกแทนที่ด้วยค่าของคอลัมน์ WORKDEPT สำหรับแถวนั้น. ตัวอย่าง, สมมติว่า, แถวดังกล่าวคือแถวสำหรับ CHRISTINE I HAAS. แผนกงานของเธอคือ A00, ซึ่งเท่ากับค่า WORKDEPT สำหรับแถวนี้. เคียวรี่ย่อยที่ถูกเรียกใช้งานสำหรับแถวนี้คือ:

```
(SELECT AVG(EDLEVEL)
   FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = 'A00')
```

ดังนั้น, สำหรับแถวที่ถูกพิจารณา, เคียวรี่ย่อยจะแสดงระดับการศึกษาโดยเฉลี่ยของแผนกของ Christine. จากนั้น ค่านี้จะถูกนำไปเปรียบเทียบในคำสั่ง outer กับระดับการศึกษาของ Christine. สำหรับแถวอื่นบางแถวที่ WORKDEPT มีค่าอื่น, ค่านั้นจะปรากฏขึ้นในเคียวรี่ย่อยแทนที่ A00. ตัวอย่างเช่น, ในแถวสำหรับ MICHAEL L THOMPSON, ค่านี้คือ B01, และเคียวรี่ย่อยสำหรับแถวของเขาจะแสดงระดับการศึกษาโดยเฉลี่ยสำหรับแผนก B01.

ตารางผลลัพธ์จากเคียวรี่มีค่า ดังต่อไปนี้

ตารางที่ 25. ชุดผลลัพธ์สำหรับเคียวรี่ก่อนหน้านี้

EMPNO	LASTNAME	WORKDEPT	EDLEVEL
000010	HAAS	A00	18
000030	KWAN	C01	20
000070	PULASKI	D21	16
000090	HENDERSON	E11	16
000110	LUCCHESSI	A00	19
000160	PIANKA	D11	17
000180	SCOUTTEN	D11	17
000210	JONES	D11	17
000220	LUTZ	D11	18
000240	MARINO	D21	17
000260	JOHNSON	D21	16
000280	SCHNEIDER	E11	17
000320	MEHTA	E21	16
000340	GOUNOT	E21	16
200010	HEMMINGER	A00	18
200220	JOHN	D11	18
200240	MONTEVERDE	D21	17
200280	SCHWARTZ	E11	17
200340	ALONZO	E21	16

ตัวอย่าง: เคียวรี่ย่อยที่สัมพันธ์กันใน HAVING clause:

สมมติว่าคุณต้องการรายการแผนกทั้งหมดซึ่งมีเงินเดือนโดยเฉลี่ยสูงกว่าเงินเดือนโดยเฉลี่ยของหน่วยงาน ในส่วนนี้ (แผนกทั้งหมดที่ WORKDEPT ขึ้นต้นด้วยตัวอักษรเดียวกันจะอยู่ในส่วนเดียวกัน) ถ้าต้องการได้ข้อมูลนี้ SQL ต้องค้นหาตาราง CORPDATA.EMPLOYEE

สำหรับแต่ละแผนกในตาราง, SQL จะเปรียบเทียบเงินเดือนโดยเฉลี่ยของแผนกกับเงินเดือนโดยเฉลี่ยของหน่วยงาน. ในเคียวรี่ย่อย, SQL จะคำนวณเงินเดือนโดยเฉลี่ยสำหรับหน่วยงานของแผนกในกลุ่มปัจจุบัน. ตัวอย่างเช่น:

```
SELECT WORKDEPT, DECIMAL(AVG(SALARY),8,2)
 FROM CORPDATA.EMPLOYEE X
 GROUP BY WORKDEPT
 HAVING AVG(SALARY) >
    (SELECT AVG(SALARY)
       FROM CORPDATA.EMPLOYEE
       WHERE SUBSTR(X.WORKDEPT,1,1) = SUBSTR(WORKDEPT,1,1))
```

ให้พิจารณาว่าจะเกิดอะไรขึ้นเมื่อรันเคียวรี่ย่อยของแผนก CORPDATA.EMPLOYEE. ก่อนที่จะถูกรัน, X.WORKDEPT จะถูกแทนที่ด้วยค่าของคอลัมน์ WORKDEPT สำหรับกลุ่มดังกล่าว. สมมติว่า, ตัวอย่างเช่น, กลุ่มแรกที่ถูกเลือกค่า WORKDEPT เป็น A00. เคียวรี่ย่อยที่ถูกเรียกใช้งานสำหรับกลุ่มนี้คือ:

```
(SELECT AVG(SALARY)
   FROM CORPDATA.EMPLOYEE
   WHERE SUBSTR('A00',1,1) = SUBSTR(WORKDEPT,1,1))
```

ดังนั้น, สำหรับกลุ่มที่ถูกพิจารณา, เคียวรี่ย่อยจะแสดงเงินเดือนโดยเฉลี่ยสำหรับหน่วยงาน. จากนั้น ค่านี้จะถูกนำไปเปรียบเทียบในคำสั่ง outer กับเงินเดือนโดยเฉลี่ยของแผนก 'A00'. สำหรับกลุ่มอื่นที่ WORKDEPT เท่ากับ 'B01', เคียวรี่ย่อยจะส่งค่าเงินเดือนเฉลี่ยสำหรับ หน่วยงานที่มีแผนก B01 อยู่ด้วย.

ตารางผลลัพธ์จากเคียวรี่มีค่า ดังต่อไปนี้

WORKDEPT	AVG SALARY
D21	25668.57
E01	40175.00
E21	24086.66

ตัวอย่าง: เคียวรี่ย่อยที่สัมพันธ์กันใน select-list:

สมมติว่า คุณต้องการรายการของแผนกทั้งหมด รวมถึงชื่อแผนก หมายเลข และชื่อผู้จัดการแผนก

ชื่อและหมายเลขแผนกหาได้จากตาราง CORPDATA.DEPARTMENT. อย่างไรก็ตาม DEPARTMENT มีเฉพาะหมายเลขผู้จัดการแผนก แต่ไม่มีชื่อ ผู้จัดการแผนก เพื่อค้นหาชื่อผู้จัดการแต่ละแผนก, คุณต้องค้นหาหมายเลขพนักงานจากตาราง EMPLOYEE ที่ตรงกับหมายเลขผู้จัดการในตาราง DEPARTMENT และส่งคืนแถวที่ตรงกัน. โดยจะส่งคืนเฉพาะแผนกที่มีผู้จัดการประจำอยู่เท่านั้นในปัจจุบัน รัน คำสั่ง SQL ต่อไปนี้:

```
SELECT DEPTNO, DEPTNAME,
    (SELECT FIRSTNME CONCAT ' ' CONCAT
         MIDINIT CONCAT ' ' CONCAT LASTNAME
       FROM EMPLOYEE X
       WHERE X.EMPNO = Y.MGRNO) AS MANAGER_NAME
   FROM DEPARTMENT Y
   WHERE MGRNO IS NOT NULL
```

แต่ละแถวที่ถูกส่งคืนสำหรับ DEPTNO และ DEPTNAME, ระบบจะค้นหา EMPNO = MGRNO และส่งคืนชื่อผู้จัดการ. ตารางผลลัพธ์จากเคียวรี่มีค่า ดังต่อไปนี้

ตารางที่ 26. ชุดผลลัพธ์สำหรับการสืบค้นก่อนหน้านี้

DEPTNO	DEPTNAME	MANAGER_NAME
A00	SPIFFY COMPUTER SERVICE DIV.	CHRISTINE I HAAS
B01	PLANNING	MICHAEL L THOMPSON
C01	INFORMATION CENTER	SALLY A KWAN
D11	MANUFACTURING SYSTEMS	IRVING F STERN
D21	ADMINISTRATION SYSTEMS	EVA D PULASKI
E01	SUPPORT SERVICES	JOHN B GEYER
E11	OPERATIONS	EILEEN W HENDERSON
E21	SOFTWARE SUPPORT	THEODORE Q SPENSER

ตัวอย่าง: เคียวรี่ย่อยที่สัมพันธ์กันในคำสั่ง UPDATE:

เมื่อคุณใช้เคียวรี่ย่อยที่สัมพันธ์กันในคำสั่ง UPDATE ชื่อที่มีความหมายเหมือนกันจะหมายถึงแถว ซึ่งคุณต้องการที่จะอัพเดต

ตัวอย่างเช่น, หากกิจกรรมทั้งหมดของโครงการต้องทำให้เสร็จสมบูรณ์ก่อนเดือนกันยายน 1983, แผนกของคุณจะพิจารณาว่าโครงการนั้นคือโครงการที่มีความสำคัญในอันดับต้นๆ. คุณสามารถใช้คำสั่ง SQL ข้างล่างนี้เพื่อประเมินผล โครงการในตาราง CORPDATA.PROJECT และใส่ค่า 1 (แฟล็กแสดงถึง ระดับความสำคัญ) ในคอลัมน์ PRIORITY (คอลัมน์ที่คุณเพิ่มให้กับ CORPDATA.PROJECT สำหรับ วัตถุประสงค์นี้) สำหรับแต่ละโครงการที่มีความสำคัญ

```
UPDATE CORPDATA.PROJECT X
  SET PRIORITY = 1
  WHERE '1983-09-01' >
     (SELECT MAX(EMENDATE)
         FROM CORPDATA.EMPPROJACT
         WHERE PROJNO = X.PROJNO)
```

เมื่อ SQL ตรวจสอบแต่ละแถวในตาราง CORPDATA.EMPPROJACT, จะกำหนดวันที่สิ้นสุดกิจกรรมเป็นอย่างมากที่สุด (EMENDATE) สำหรับกิจกรรมทั้งหมดของโครงการ (จากตาราง CORPDATA.PROJECT). หากวันที่สิ้นสุดของแต่ละกิจกรรมที่เกี่ยวข้องกับโครงการ คือ วันก่อนเดือนกันยายน 1983, แถวปัจจุบันในตาราง CORPDATA.PROJECT จะเป็นไปตามเกณฑ์และถูกอัพเดต.

อัพเดตตารางลำดับหลักเมื่อเกิดการเปลี่ยนแปลงใดๆ ขึ้นกับปริมาณตามลำดับ. หากไม่ได้เซ็ตปริมาณในตารางลำดับ (ค่า NULL), ให้เก็บค่าที่อยู่ในตารางลำดับหลักเอาไว้.

```
UPDATE MASTER_ORDERS X
   SET QTY=(SELECT COALESCE (Y.QTY, X.QTY)
            FROM ORDERS Y
            WHERE X.ORDER_NUM = Y.ORDER_NUM)
   WHERE X.ORDER_NUM IN (SELECT ORDER_NUM
            FROM ORDERS)
```

ในตัวอย่างนี้, แต่ละแถวของตาราง MASTER_ORDERS จะถูกตรวจสอบเพื่อดูว่ามีแถวที่ตรงกันในตาราง ORDERS หรือไม่. หากมีแถวที่ตรงกันในตาราง ORDERS, ฟังก์ชัน COALESCE จะถูกนำมาใช้เพื่อส่งคืนค่าสำหรับคอลัมน์ QTY. หาก QTY ในตาราง ORDERS มีค่าที่ไม่ใช่ null, ค่านั้นจะถูกนำไปใช้เพื่ออัพเดตคอลัมน์ QTY ในตาราง MASTER_ORDERS. หากค่า QTY ในตาราง ORDERS เท่ากับ NULL, คอลัมน์ MASTER_ORDERS QTY จะถูกอัพเดตด้วยค่าของมันเอง.

ตัวอย่าง: เคียวรี่ย่อยที่สัมพันธ์กันในคำสั่ง DELETE:

เมื่อคุณใช้เคียวรี่ย่อยที่สัมพันธ์กันในคำสั่ง DELETE ชื่อที่มี ความหมายเหมือนกันจะหมายถึงแถวที่คุณลบออกไป SQL จะประเมินเคียวรี่ย่อยที่สัมพันธ์กันหนึ่งครั้งสำหรับแต่ละแถวในตารางที่ปรากฏชื่อในคำสั่ง DELETE เพื่อตัดสินว่าจะลบแถวนั้นออกหรือไม่

สมมติว่าแถวในตาราง CORPDATA.PROJECT ถูก ลบออก แถวที่เกี่ยวข้องกับโครงการที่ถูกลบออกในตาราง CORPDATA.EMPPROJACT จะต้องถูกลบออกด้วย. ในการทำสิ่งนี้ ให้รันคำสั่งต่อไปนี้:

```
DELETE FROM CORPDATA.EMPPROJACT X
  WHERE NOT EXISTS
     (SELECT *
              FROM CORPDATA.PROJECT
        WHERE PROJNO = X.PROJNO)
```

SQL จะตัดสินว่า, แต่ละแถวในตาราง CORPDATA.EMP_ACT, แถวที่มีหมายเลขโครงการเดียวกันมีอยู่แล้วในตาราง CORPDATA.PROJECT หรือไม่. หากไม่มีอยู่, แถว CORPDATA.EMP_ACT จะถูกลบทิ้ง.

ลำดับการเรียง และ normalization ใน SQL

ลำดับการจัดเรียงกำหนดความสัมพันธ์ของอักขระในชุดอักขระ เมื่อมีการเปรียบเทียบหรือจัดลำดับ. Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

ลำดับการจัดเรียงใช้สำหรับอักขระทั้งหมดและการเปรียบเทียบกราฟิก UCS-2 และ UTF-16 ในคำสั่ง SQL. มีตารางลำดับการจัดเรียงสำหรับข้อมูลอักขระทั้งแบบไบต์เดียว และไบต์คู่. ตารางลำดับการจัดเรียงแบบไบต์เดียวแต่ละตารางจะมีตารางลำดับการจัดเรียงแบบไบต์คู่ที่สัมพันธ์กันเสมอ การแปลงค่าระหว่างสองตารางจะเริ่มขึ้นเมื่อจำเป็นต้องดำเนินการสืบค้น. นอกจากนี้, คำสั่ง CREATE INDEX มีลำดับการจัดเรียง (มีผลเมื่อมีการรันคำสั่ง) ซึ่งใช้กับคอลัมน์อักขระที่อ้างอิงถึงในดรรชนี.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การสร้างและการใช้มุมมอง" ในหน้า 36
สามารถใช้มุมมองเพื่อเข้าใช้งานข้อมูลในตารางหนึ่งตารางหรือมากกว่าหรือมุมมองหนึ่งมุมมองหรือมากกว่าได้. คุณสามารถสร้างมุมมองได้โดยใช้คำสั่ง SELECT.

“การสร้างดรรชนี” ในหน้า 41
คุณสามารถใช้ดรรชนี เพื่อเรียงลำดับ และเลือกข้อมูล. นอกจากนี้, ดรรชนียังช่วย ระบบให้เรียกข้อมูลออกมาได้เร็วขึ้น เพื่อประสิทธิภาพของการสอบถามที่ดีกว่าเดิม.

“การระบุเงื่อนไขการค้นหาโดยใช้ WHERE clause” ในหน้า 46
WHERE clause จะระบุเงื่อนไขการค้นหา ที่บ่งชี้ถึงแถวที่คุณต้องการดึงค่า, อัพเดต หรือลบ

“GROUP BY clause” ในหน้า 49
GROUP BY clause อนุญาตให้คุณค้นหาคุณลักษณะของกลุ่มของแถวมากกว่าที่จะค้นหาแถวเดี่ยว.

“HAVING clause” ในหน้า 51
HAVING clause ระบุเงื่อนไขการค้นหา สำหรับกลุ่มที่เลือกได้โดยใช้ GROUP BY clause

“ORDER BY clause” ในหน้า 52
ORDER BY clause ระบุลำดับเฉพาะที่ คุณต้องการส่งคืนแถวที่เลือกไว้ ลำดับจะถูกเรียงลำดับการเรียงของค่าคอลัมน์ หรือค่านิพจน์จากน้อยไปมาก หรือจากมากไปน้อย

“การจัดการกับแถวซ้ำ” ในหน้า 60
เมื่อ SQL ประเมินผลคำสั่ง select หลายแถว อาจจะมีคุณสมบัติพอที่จะอยู่ในตารางผลลัพธ์ขึ้นอยู่กับจำนวนของแถว ที่ตรงกับเงื่อนไขการค้นหาของคำสั่ง select บางแถว ในตารางผลลัพธ์อาจซ้ำกันได้

“การนิยามเงื่อนไขการค้นหาที่ซับซ้อน” ในหน้า 61
นอกจากเพรดิเคตการเปรียบเทียบพื้นฐานแล้ว เช่น = และ > เงื่อนไขการค้นหาสามารถมีเพรดิเคต BETWEEN, IN, EXISTS, IS NULL และ LIKE

“การใช้คีย์เวิร์ด UNION เพื่อรวมการเลือกย่อย” ในหน้า 84
การใช้คีย์เวิร์ด UNION, คุณสามารถรวมการเลือกย่อยมากกว่าสองได้ เพื่อสร้าง fullselect.

ลำดับการเรียง

ลำดับการจัดเรียงที่ใช้กับ ORDER BY และการเลือกแถว

ตัวอย่างเหล่านี้แสดงวิธีการเรียงลำดับแถว และการเลือกแถว สำหรับลำดับการจัดเรียงที่ใช้

ค่าในคอลัมน์ JOB มีทั้งตัวพิมพ์ใหญ่และเล็ก คุณสามารถดูค่า ’Mgr’, ’MGR’, และ ’mgr’.

ตารางที่ 27. ตาราง STAFF

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
20	Pernal	20	Sales	8	18171.25	612.45
30	Merenghi	38	MGR	5	17506.75	0
40	OBrien	38	Sales	6	18006.00	846.55
50	Hanes	15	Mgr	10	20659.80	0
60	Quigley	38	SALES	0	16808.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
80	James	20	Clerk	0	13504.60	128.20

ตารางที่ 27. ตาราง STAFF (ต่อ)

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
90	Koonitz	42	sales	6	18001.75	1386.70
100	Plotz	42	mgr	6	18352.80	0

ในตัวอย่างต่อไปนี้, ผลลัพธ์จะแสดงสำหรับคำสั่งแต่ละคำสั่งโดยใช้:

- ลำดับการจัดเรียง *HEX
- ลำดับการจัดเรียงแบบเฉลี่ยน้ำหนักโดยใช้ language identifier ENU
- ลำดับการจัดเรียงแบบน้ำหนักเฉพาะโดยใช้ language identifier ENU

หมายเหตุ: ENU ถูกเลือกเป็น language identifier โดยระบุ SRTSEQ(*LANGIDUNQ), หรือ SRTSEQ(*LANGIDSHR) และ LANGID(ENU), บนคำสั่ง CRTSQLxxx, STRSQL, หรือ RUNSQLSTM, หรือโดยใช้คำสั่ง SET OPTION.

ลำดับการจัดเรียงและ ORDER BY

ลำดับการจัดเรียงส่งผลกระทบต่อการสั่งซื้อที่ดำเนินการโดย ORDER BY clause

คำสั่ง SQL ต่อไปนี้ทำให้ตารางผลลัพธ์ถูกจัดเรียงโดยใช้ค่าในคอลัมน์ JOB:

```
SELECT * FROM STAFF ORDER BY JOB
```

ตารางต่อไปนี้แสดงผลลัพธ์โดยใช้ลำดับการจัดเรียง *HEX. แถวต่างๆ ถูกจัดเรียงโดยยึดตามค่า EBCDIC ในคอลัมน์ JOB. ในกรณีนี้, อักษรตัวพิมพ์เล็กทั้งหมดจะจัดเรียงก่อนอักษรตัวพิมพ์ใหญ่.

*ตารางที่ 28. ผลลัพธ์ของการใช้ลำดับการเรียง *HEX*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
100	Plotz	42	mgr	6	18352.80	0
90	Koonitz	42	sales	6	18001.75	1386.70
80	James	20	Clerk	0	13504.60	128.20
10	Sanders	20	Mgr	7	18357.50	0
50	Hanes	15	Mgr	10	20659.80	0
30	Merenghi	38	MGR	5	17506.75	0
20	Pernal	20	Sales	8	18171.25	612.45
40	OBrien	38	Sales	6	18006.00	846.55
70	Rothman	15	Sales	7	16502.83	1152.00
60	Quigley	38	SALES	0	16808.30	650.25

ตารางต่อไปนี้แสดงวิธีการจัดเรียงสำหรับลำดับการจัดเรียงแบบน้ำหนักเฉพาะ. หลังจากใช้ลำดับการจัดเรียงกับค่าในคอลัมน์ JOB, แถวต่างๆ จะถูกจัดเรียง. โปรดสังเกตุว่า หลังจากการจัดเรียง, อักษรตัวพิมพ์เล็กจะมาก่อนตัวอักษรตัวเดียวกันที่เป็นอักษรตัวพิมพ์ใหญ่, และค่า 'mgr', 'Mgr', และ 'MGR' จะอยู่ติดกัน.

ตารางที่ 29. ผลลัพธ์ของการใช้ลำดับการจัดเรียงแบบน้ำหนักเฉพาะสำหรับ ENU language identifier

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
80	James	20	Clerk	0	13504.60	128.20
100	Plotz	42	mgr	6	18352.80	0
10	Sanders	20	Mgr	7	18357.50	0
50	Hanes	15	Mgr	10	20659.80	0
30	Merenghi	38	MGR	5	17506.75	0
90	Koonitz	42	sales	6	18001.75	1386.70
20	Pernal	20	Sales	8	18171.25	612.45
40	OBrien	38	Sales	6	18006.00	846.55
70	Rothman	15	Sales	7	16502.83	1152.00
60	Quigley	38	SALES	0	16808.30	650.25

ตารางต่อไปนี้แสดงวิธีการจัดเรียงสำหรับลำดับการจัดเรียงแบบน้ำหนักเฉลี่ย. หลังจากใช้ลำดับการจัดเรียงกับค่าในคอลัมน์ JOB, แถวต่างๆ จะถูกจัดเรียง. สำหรับการเปรียบเทียบการจัดเรียง, อักษรตัวพิมพ์เล็กแต่ละตัว จะถือว่าเหมือนกับอักษรตัวพิมพ์ใหญ่ที่ตรงกัน. ในตารางนี้, โปรดสังเกตุว่า ค่า 'MGR', 'mgr' และ 'Mgr' ทั้งหมดถูกรวมไว้ด้วยกัน.

ตารางที่ 30. ผลลัพธ์ของการใช้ลำดับการจัดเรียงแบบเฉลี่ยสำหรับ ENU language identifier

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
80	James	20	Clerk	0	13504.60	128.20
10	Sanders	20	Mgr	7	18357.50	0
30	Merenghi	38	MGR	5	17506.75	0
50	Hanes	15	Mgr	10	20659.80	0
100	Plotz	42	mgr	6	18352.80	0
20	Pernal	20	Sales	8	18171.25	612.45
40	OBrien	38	Sales	6	18006.00	846.55
60	Quigley	38	SALES	0	16808.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
90	Koonitz	42	sales	6	18001.75	1386.70

ลำดับการจัดเรียงและการเลือกแถว

ลำดับการจัดเรียงส่งผลกระทบต่อการเลือกข้อมูล

คำสั่ง SQL ต่อไปนี้จะเลือกแถวที่มีค่า 'MGR' ในคอลัมน์ JOB:

```
SELECT * FROM STAFF WHERE JOB='MGR'
```

ตารางแรกแสดงวิธีการเลือกแถวที่มีลำดับการจัดเรียงแบบ *HEX. แถวที่ตรงตามเกณฑ์การเลือกแถวสำหรับคอลัมน์ JOB จะถูกเลือกตามที่ระบุไว้ในคำสั่ง select. เฉพาะ 'MGR' ที่เป็นตัวพิมพ์ใหญ่เท่านั้นที่ถูกเลือก.

*ตารางที่ 31. ผลลัพธ์ของการใช้ลำดับการเรียง *HEX*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

ตาราง 2 แสดงวิธีการเลือกแถวที่มีลำดับการจัดเรียงแบบน้ำหนักเฉพาะ. ตัวอักษรตัวพิมพ์เล็กและตัวพิมพ์ใหญ่จะมีลักษณะเฉพาะ. 'mgr' ที่เป็นตัวพิมพ์เล็กจะถือว่าไม่เหมือนกับ 'MGR' ที่เป็นตัวพิมพ์ใหญ่. ดังนั้น, 'mgr' ที่เป็นตัวพิมพ์เล็กจะไม่ถูกเลือก.

ตารางที่ 32. ผลลัพธ์ของการใช้ลำดับการจัดเรียงแบบน้ำหนักเฉพาะสำหรับ ENU language identifier

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

ตารางต่อไปนี้แสดงวิธีการเลือกแถวที่มีลำดับการจัดเรียงแบบน้ำหนักเฉลี่ย. แถวที่ตรงตามเกณฑ์การเลือกสำหรับคอลัมน์ 'JOB' จะถูกเลือกโดยถือว่าตัวอักษรตัวพิมพ์ใหญ่เหมือนกับตัวอักษรตัวพิมพ์เล็ก. โปรดสังเกตว่า ค่า 'mgr', 'Mgr' และ 'MGR' ทั้งหมดจะถูกเลือก.

ตารางที่ 33. ผลลัพธ์ของการใช้ลำดับการจัดเรียงแบบเฉลี่ยสำหรับ ENU language identifier

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
30	Merenghi	38	MGR	5	17506.75	0
50	Hanes	15	Mgr	10	20659.80	0
100	Plotz	42	mgr	6	18352.80	0

ลำดับการจัดเรียงและมุมมอง

เมื่อมีการรันคำสั่ง CREATE VIEW มุมมองจะถูกสร้างขึ้นด้วยลำดับการจัดเรียงที่ระบุไว้

เมื่อมีการอ้างถึงมุมมองใน FROM clause, ลำดับการจัดเรียงนั้นจะถูกใช้สำหรับการเปรียบเทียบอักขระใดๆ ในการเลือกย่อยของ CREATE VIEW. ในขณะนั้น, ตารางผลลัพธ์ระดับกลางจะถูกสร้างจาก การเลือกย่อยของมุมมอง. จากนั้นลำดับการจัดเรียงซึ่งมีผลเมื่อรันเคียวรี่ จะถูกใช้กับอักขระและการเปรียบเทียบกราฟิก UCS-2 ทั้งหมด (รวมทั้งการเปรียบเทียบที่เกี่ยวข้องกับการแปลงแบบ implicit เป็นอักขระ, หรือกราฟิก UCS-2 หรือกราฟิก UTF-16) ที่ระบุในเคียวรี่.

คำสั่ง SQL และตารางต่อไปนี้จะแสดงวิธีการทำงานของมุมมองและลำดับการจัดเรียง. มุมมอง V1, ซึ่งใช้ในตัวอย่างต่อไปนี้, ถูกสร้างด้วยลำดับการจัดเรียงแบบเฉลี่ยน้ำหนักของ SRTSEQ(*LANGIDSHR) และ LANGID(ENU). คำสั่ง CREATE VIEW จะเป็นดังนี้:

```
CREATE VIEW V1 AS SELECT *
  FROM STAFF
  WHERE JOB = 'MGR' AND ID < 100
```

*ตารางที่ 34. "SELECT * FROM V1"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
30	Merenghi	38	MGR	5	17506.75	0
50	Hanes	15	Mgr	10	20659.80	0

เคียวรี่ใดๆ ที่รันกับมุมมอง V1 จะรันกับตารางผลลัพธ์ที่แสดงไว้ด้านบน. เคี่ยวรี่ที่แสดงด้านล่างรันกับลำดับการจัดเรียง ของ SRTSEQ(*LANGIDUNQ) และ LANGID(ENU).

*ตารางที่ 35. "SELECT * FROM V1 WHERE JOB = 'MGR'" โดยใช้ลำดับการจัดเรียงแบบน้ำหนักเฉพาะสำหรับ ENU language identifier*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

ลำดับการจัดเรียงและคำสั่ง CREATE INDEX

ดรรชนีจะถูกสร้างขึ้นด้วยลำดับการจัดเรียงที่ ระบุไว้ เมื่อมีการรันคำสั่ง CREATE INDEX

รายการจะเพิ่มเข้าในดรรชนีทุกครั้งที่มีการแทรก เข้าในตารางที่มีการกำหนดดรรชนีไว้. รายการของดรรชนีมีค่าถ่วงน้ำหนักสำหรับคอลัมน์คีย์อักขระ, และคอลัมน์คีย์กราฟิกแบบ UCS-2 และ UTF-16. ระบบได้รับค่าถ่วงน้ำหนักโดยการแปลงค่าคีย์ ซึ่งยึดตาม ลำดับการจัดเรียงของดรรชนี.

เมื่อมีการเลือกโดยใช้ลำดับการเลือกนั้นและดรรชนีนั้น, ไม่จำเป็นต้องแปลงคีย์อักขระ, หรือคีย์กราฟิกแบบ UCS-2 หรือ UTF-16 ก่อนเปรียบเทียบ. ซึ่งจะทำให้ประสิทธิภาพการสืบค้นให้ดีขึ้น.

หลักการที่เกี่ยวข้อง

การใช้ดรรชนีที่มีลำดับการจัดเรียง

ลำดับการจัดเรียงและข้อจำกัด

ข้อจำกัดเฉพาะจะดำเนินการด้วยดรรชนี หากตารางที่มีการเพิ่มข้อจำกัดเฉพาะถูกกำหนดโดยใช้ลำดับ การจัดเรียง ดรรชนีจะถูกสร้างขึ้นด้วยลำดับการจัดเรียงเดียวกัน

หากระบุข้อจำกัดอ้างอิง, ลำดับการจัดเรียงระหว่างตาราง parent และ dependent ต้องตรงกัน.

ลำดับการจัดเรียงที่ใช้ขณะกำหนดข้อจำกัดการตรวจสอบ จะเป็น ลำดับการจัดเรียงเดียวกันกับที่ระบบใช้เพื่อตรวจสอบความถูกต้องของการปฏิบัติตาม ข้อจำกัดในขณะ INSERT หรือ UPDATE.

ลำดับการจัดเรียง ICU

เมื่อตารางลำดับการจัดเรียง International Components for Unicode (ICU) ถูกใช้ ฐานข้อมูลจะใช้กฎทางภาษาเพื่อ กำหนดการถ่วงน้ำหนักของข้อมูลตามโลแคลของตาราง

ตารางลำดับการจัดเรียง ICU มีชื่อว่า en_us (United States locale) สามารถเรียงลำดับข้อมูลต่างไปจากตาราง ICU อื่นที่ชื่อว่า fr_FR (French locale) ตัวอย่างเช่น.

การสนับสนุน ICU (i5/OS อ็อพชัน 39) สามารถจัดการข้อมูลที่ไม่ถูกทำให้เป็นมาตรฐานได้อย่างเหมาะสม ทำให้เกิดผลลัพธ์ที่เหมือนกันกับ ข้อมูลที่ได้รับการทำให้เป็นมาตรฐานแล้ว ตารางลำดับการจัดเรียง ICU สามารถเรียงลำดับตัวอักขระทั้งหมด, กราฟิก, และข้อมูลยูนิโค้ด (UTF-8, UTF-16 และ UCS-2)

ตัวอย่างเช่น คอลัมน์ของอักขระ UTF-8 ที่ชื่อว่า NAME จะมีชื่อต่อไปนี้ (พร้อมค่าตัวเลขฐานสิบหกของคอลัมน์)

NAME	HEX (NAME)
Gómez	47C3B36D657A
Gomer	476F6D6572
Gumby	47756D6279

ลำดับการจัดเรียง *HEX จะจัดเรียงค่าของ NAME ดังนี้

NAME
Gomer
Gumby
Gómez

ตารางลำดับการจัดเรียง ICU ที่ชื่อ en_us จะทำให้การจัดลำดับค่า NAME เป็นไปอย่างถูกต้องดังนี้

NAME
Gomer
Gómez
Gumby

เมื่อตารางลำดับการจัดเรียง ICU ถูกระบุ ประสิทธิภาพการทำงานของคำสั่ง SQL ที่ใช้ตารางนั้นอาจช้าลงมากกว่าประสิทธิภาพของคำสั่ง SQL ที่ใช้ตารางลำดับการจัดเรียงที่ไม่ใช่ ICU หรือใช้ลำดับการจัดเรียงแบบ *HEX ประสิทธิภาพการทำงานที่ช้าลงเป็นผลมาจากการเรียกใช้การสนันสนุน ICU เพื่อที่จะได้ค่าการถ่วงน้ำหนักของข้อมูลแต่ละชิ้นที่จะถูกจัดเรียง ตารางลำดับการจัดเรียง ICU ช่วยให้ฟังก์ชันในการเรียงลำดับเพิ่มมากขึ้นแต่จะทำให้การรันคำสั่ง SQL ช้าลง. อย่างไรก็ตาม ดรรชนีที่ถูกสร้าง

ด้วย ตารางลำดับการจัดเรียง ICU สามารถถูกสร้างบนคอลัมน์เพื่อช่วยลดความต้องการในการเรียกไปยังการสนับสนุน ICU ในกรณีนี้คีย์ดรรชนีควรจะมีค่าการถ่วงน้ำหนัก ICU อยู่แล้วซึ่งจะทำให้ไม่ต้องมีการเรียกใช้การสนับสนุน ICU

หลักการที่เกี่ยวข้อง

International Components for Unicode

Normalization

Normalization อนุญาตให้คุณเปรียบเทียบสตริงที่มีอักขระแบบผสม.

ข้อมูลที่มีป้าย UTF-8 หรือ UTF-16 CCSID สามารถเก็บอักขระแบบผสมได้. อักขระแบบผสมอนุญาตให้อักขระที่เป็นผลลัพธ์สามารถประกอบขึ้นจากอักขระมากกว่าหนึ่งอักขระ. ในข้อมูลสตริง หลังจากอักขระตัวแรกของอักขระแบบเชิงซ้อน, สามารถตามได้ด้วยอักขระชนิดที่ไม่ใช่เว้นวรรค ตัวอย่างเช่น เครื่องหมายแสดงสำเนียง. ถ้าอักขระผลลัพธ์เป็นหนึ่งในกลุ่มอักขระที่ถูกกำหนดไว้แล้ว, การ normalization ของสตริงจะเป็นผลทำให้ตัวอักขระแบบผสมหลายตัวถูกแทนที่ด้วยค่าของอักขระที่ถูกกำหนดไว้แล้ว. ตัวอย่างเช่น, ถ้าสตริงของคุณประกอบด้วยตัวอักษร 'a' ตามด้วย '..', สตริงจะถูกทำให้เป็นมาตรฐาน ซึ่งประกอบด้วยอักขระ 'ä' ตัวเดียว.

การ Normalization ทำให้เกิดความเป็นไปได้ในการเปรียบเทียบสตริงอย่างแม่นยำ. ถ้าข้อมูลไม่ถูกทำให้เป็นมาตรฐาน, สตริงสองชุดที่ดูเหมือนกันบนหน้าจออาจจะเปรียบเทียบได้ไม่เท่ากันเพราะการแทนค่าที่ถูกเก็บไว้อาจจะแตกต่างกัน. เมื่อข้อมูลสตริง UTF-8 และ UTF-16 ไม่ถูกทำให้เป็นมาตรฐาน, เป็นไปได้ว่า คอลัมน์ในตารางสามารถมีแถวหนึ่งแถวที่มีตัวอักษร 'a' ตามด้วยอักขระเน้นเสียง และอีกแถวที่เป็นอักขระผสม 'ä'. สองค่านี้มีค่าไม่เท่ากันเมื่อเปรียบเทียบเพรดิเคต: WHERE C1 = 'ä'. ด้วยเหตุผลนี้, จึงเป็นการสนับสนุนว่าคอลัมน์สตริงทั้งหมดในตารางควรจะเก็บในรูปแบบที่ทำให้เป็นมาตรฐานแล้ว.

คุณสามารถทำข้อมูลของคุณให้เป็นมาตรฐานก่อนที่จะทำการแทรกหรือทำการอัพเดต, หรือคุณสามารถกำหนดคอลัมน์ในตารางของฐานข้อมูลให้เป็นมาตรฐานแบบอัตโนมัติ. เพื่อให้ฐานข้อมูลทำการ normalization, ระบุ NORMALIZED เป็นส่วนของการนิยามคอลัมน์. อ็อพชันนี้อนุญาตให้เฉพาะคอลัมน์ที่มีป้าย CCSID เป็น 1208 (UTF-8) หรือ 1200 (UTF-16). ฐานข้อมูลทำเสมือนว่าทุกคอลัมน์ในตารางถูกทำให้เป็นมาตรฐานแล้ว.

NORMALIZED clause สามารถระบุไว้สำหรับพารามิเตอร์ของฟังก์ชันและโพรซิเดอร์. ถ้าระบุสำหรับอินพุตพารามิเตอร์, การ normalization จะถูกกระทำโดยฐานข้อมูลสำหรับค่าของพารามิเตอร์ก่อนที่จะเรียกฟังก์ชันหรือโพรซิเดอร์. ถ้าระบุสำหรับเอาต์พุตพารามิเตอร์, clause จะไม่ถูกบังคับ; จะเป็นเสมือนว่าค่าที่ได้รับคืนจากรูทีนของผู้ใช้นั้นถูกทำให้เป็นมาตรฐานแล้ว.

อ็อพชัน NORMALIZE_DATA ในไฟล์ QAQQINI ถูกใช้เพื่อชี้ว่าระบบควรจะทำการ normalization หรือไม่เมื่อทำงานกับข้อมูลแบบ UTF-8 และ UTF-16. อ็อพชันนี้ควบคุมว่าเมื่อไรควรจะทำให้เป็นมาตรฐานก่อนการใช้ใน SQL สำหรับตัวอักษร, ตัวแปรโฮสต์, มาร์คเกอร์พารามิเตอร์, และนิพจน์ที่ประกอบด้วยสตริง. อ็อพชันถูกกำหนดในตอนเริ่มแรกว่าไม่ต้องทำการ normalization. ซึ่งเป็นค่าที่ถูกต้องสำหรับคุณถ้าข้อมูลในตารางของคุณ และค่าตัวอักษรใดๆในแอ็พพลิเคชั่นของคุณถูกทำให้เป็นมาตรฐานอยู่แล้วโดยกลไกอื่น หรือไม่มีอักขระที่จะต้องถูกทำให้เป็นมาตรฐาน. ถ้าเป็นกรณีนี้คุณจะต้องการหลีกเลี่ยงภาระของการทำ normalization ของระบบในการเคียวรีของคุณ. ถ้าข้อมูลของคุณไม่ถูกทำให้เป็นมาตรฐาน, คุณจะต้องการเปลี่ยนค่าของอ็อพชันนี้เพื่อให้ระบบทำการ normalization ให้กับคุณ.

งานที่เกี่ยวข้อง

การควบคุมเคียวรี่แบบ dynamic ด้วยไฟล์อ็อพชันเคียวรี่ QAQQINI

การปกป้องข้อมูล

ฐานข้อมูล DB2 for i5/OS ให้วิธีการที่หลากหลายสำหรับการปกป้องข้อมูล SQL จากผู้ใช้ที่ไม่มีสิทธิในการใช้งาน และ การตรวจสอบ data integrity

การรักษาความปลอดภัยสำหรับ SQL อ็อบเจ็กต์

อ็อบเจ็กต์ทั้งหมดในระบบ รวมถึง SQL อ็อบเจ็กต์ จะถูกจัดการโดยฟังก์ชันความปลอดภัยของระบบ

ผู้ใช้สามารถให้สิทธิในการใช้งาน SQL อ็อบเจ็กต์ได้โดยผ่านทาง คำสั่ง SQL GRANT และ REVOKE หรือผ่านทางคำสั่ง CL อันได้แก่ Edit Object Authority (EDTOBJAUT), Grant Object Authority (GRTOBJAUT), and Revoke Object Authority (RVKOBJAUT)

คำสั่ง SQL GRANT และ REVOKE จะปฏิบัติการบนฟังก์ชัน SQL, SQL แพ็กเกจ, SQL โพรซีเดอร์, ชนิดของการจำแนก, ลำดับ, ตาราง, มุมมอง, และในแต่ละคอลัมน์ของตารางและมุมมอง. นอกเหนือไปจากนั้น, คำสั่ง SQL GRANT และ REVOKE จะยอมรับแต่เพียงสิทธิในการใช้งานแบบ private และ public เท่านั้น. ในบางกรณี, มีความจำเป็นที่จะต้องใช้ EDTOBJAUT, GRTOBJAUT, และ RVKOBJAUT ในการมอบสิทธิในการใช้งาน อ็อบเจ็กต์อื่นๆให้กับผู้ใช้, ยกตัวอย่างเช่น คำสั่ง และ โปรแกรมต่างๆ.

สิทธิในการใช้งานคำสั่ง SQL จะได้รับการตรวจสอบหรือไม่ ขึ้นอยู่กับว่าในขณะนั้นคำสั่งอยู่ในสถานะ static, dynamic, หรือ กำลังรันแบบโต้ตอบอยู่.

สำหรับคำสั่ง SQL แบบ static:

- ถ้าค่าของ USRPRF เป็น *USER, สิทธิในการใช้งานเพื่อที่จะรันคำสั่ง SQL ในระบบจะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ของผู้ที่กำลังรันโปรแกรมอยู่ในขณะนั้น. สิทธิในการใช้งานเพื่อที่จะรันคำสั่ง SQL แบบรีโมตนั้นจะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ที่อยู่ในแอ็พพลิเคชันเซิร์ฟเวอร์. *USER เป็นค่าโดยปกติในการตั้งชื่อให้ระบบ (*SYS).
- ถ้าค่าของ USRPRF เป็น *OWNER, สิทธิในการใช้งานเพื่อที่จะรันคำสั่ง SQL ในระบบจะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ของผู้ที่กำลังรันโปรแกรมอยู่ในขณะนั้น และโปรไฟล์ผู้ใช้ของเจ้าของโปรแกรมนั้น. สิทธิในการใช้งานเพื่อที่จะรันคำสั่ง SQL แบบรีโมตจะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ของงานในแอ็พพลิเคชันเซิร์ฟเวอร์และโปรไฟล์ผู้ใช้ของเจ้าของแพ็คเกจ SQL นั้น. สิทธิที่มีขั้นสูงกว่าคือสิทธิในการใช้งานที่ถูกนำมาใช้. *OWNER เป็นค่าปกติสำหรับการตั้งชื่อให้ SQL (*SQL).

สำหรับคำสั่ง SQL แบบ dynamic:

- ถ้าค่าของ USRPRF เป็น *USER, สิทธิในการใช้งานเพื่อที่จะรันคำสั่ง SQL ในระบบ จะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ของผู้ที่กำลังรันโปรแกรมอยู่ในขณะนั้น. สิทธิในการใช้งานเพื่อรันคำสั่ง SQL แบบรีโมตจะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ของงานในแอ็พพลิเคชันเซิร์ฟเวอร์.
- ถ้าค่าของ USRPRF เป็น *OWNER และ DYNUSRPRF เป็น *USER, สิทธิในการใช้งานเพื่อที่จะรันคำสั่ง SQL ในระบบจะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ของผู้ที่กำลังรันโปรแกรมอยู่ในขณะนั้น. สิทธิในการใช้งานเพื่อรันคำสั่ง SQL แบบรีโมตจะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ของงานในแอ็พพลิเคชันเซิร์ฟเวอร์.
- ถ้าค่าของ USRPRF เป็น *OWNER และ DYNUSRPRF เป็น *OWNER, สิทธิในการใช้งานเพื่อที่จะรันคำสั่ง SQL ในระบบจะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ของผู้ที่กำลังรันโปรแกรมอยู่ในขณะนั้น และโปรไฟล์ผู้ใช้ของเจ้าของโปรแกรม. สิทธิในการใช้งานเพื่อที่จะรันคำสั่ง SQL แบบรีโมตจะได้รับการตรวจสอบโดยใช้โปรไฟล์ผู้ใช้ของงานในแอ็พพลิเคชันเ

ซิร์ฟเวอร์และโปรไฟล์ผู้ใช้ของเจ้าของแพ็คเกจ SQL นั้น. สิทธิที่มีขั้นสูงที่สุดคือสิทธิในการใช้งานที่ถูกนำมาใช้. เพื่อคำนึงถึงความปลอดภัย, จึงควรที่จะใช้ค่าของพารามิเตอร์ *OWNER สำหรับ DYNUSRPRF อย่างระมัดระวัง. ตัวเลือกนี้จะให้สิทธิในการใช้งานของเจ้าของโปรแกรมหรือ package ให้กับผู้ที่จะรันโปรแกรม.

สำหรับคำสั่ง SQL แบบโต้ตอบ, สิทธิในการใช้งานจะได้รับการตรวจสอบในลักษณะตรงกันข้ามกันกับ สิทธิของผู้ที่กำลังทำการประมวลผลคำสั่งในขณะนั้น. สิทธิที่รับมาจะไม่สามารถใช้ได้กับคำสั่ง SQL แบบโต้ตอบ.

สิ่งอ้างอิงที่เกี่ยวข้อง

การอ้างอิงความปลอดภัย

GRANT (Table หรือ View Privileges)

REVOKE (Table หรือ View Privileges)

รหัสแสดงสิทธิการใช้งาน

รหัสแสดงสิทธิการใช้งาน เป็นอ็อบเจ็กต์โปรไฟล์ผู้ใช้ที่ระบุผู้ใช้เพียงหนึ่งเดียว คุณสามารถใช้คำสั่ง Create User Profile (CRTUSRPRF) เพื่อสร้างรหัสแสดงสิทธิการใช้งาน

มุมมอง

มุมมองสามารถป้องกันผู้ใช้ที่ไม่มีสิทธิจากการ เข้าถึงข้อมูลที่สำคัญ

แอ็พพลิเคชันโปรแกรมสามารถเข้าถึงข้อมูลที่ต้องการได้จากตาราง, แต่จะไม่สามารถเข้าถึงข้อมูลที่สำคัญ หรือข้อมูลที่ถูกจำกัดเอาไว้ในตาราง. มุมมอง สามารถจำกัดการเข้าถึงข้อมูลในคอลัมน์จำเพาะได้โดยการไม่ระบุคอลัมน์เหล่านั้นลงในรายการ SELECT (ตัวอย่างเช่น, เงินเดือนพนักงาน). มุมมองยังสามารถจำกัดการเข้าถึงแถวจำเพาะในตารางโดยการระบุ WHERE clause (ตัวอย่างเช่น, การอนุญาตให้เข้าถึงข้อมูลเฉพาะแถวที่เชื่อมโยงกับหมายเลขแผนกจำเพาะเท่านั้น).

การตรวจสอบ

ฐานข้อมูล DB2 for i5/OS ได้ถูกออกแบบมาให้ใช้ได้กับระดับการรักษาความปลอดภัย C2 ของรัฐบาลสหรัฐอเมริกา คุณสมบัติหลัก ของระดับนั้นก็คือความสามารถในการตรวจสอบการทำงานของระบบ

DB2 for i5/OS ใช้ เครื่องมือช่วยในการตรวจสอบซึ่งควบคุมโดยฟังก์ชันความปลอดภัยของระบบ การตรวจสอบ สามารถกระทำได้ในระดับของอ็อบเจ็กต์, ระดับของผู้ใช้ หรือระดับของระบบ ค่ากำหนดของระบบที่เป็นค่า QAUDCTL จะเป็นตัวควบคุมให้การตรวจสอบถูกกระทำในระดับของ อ็อบเจ็กต์หรือผู้ใช้. คำสั่ง Change User Audit (CHGUSRAUD) และคำสั่ง Change Object Audit (CHGOBJAUD) ระบุว่าผู้ใช้และอ็อบเจ็กต์ใดถูกตรวจสอบ ค่ากำหนดของระบบ QAUDLVL ควบคุมประเภทของการทำงานที่ถูกตรวจสอบ (ตัวอย่างเช่น ความล้มเหลวใน การให้สิทธิ และการสร้าง การลบออก การยอมรับ หรือการเรียกคืน)

DB2 for i5/OS ยังสามารถ ตรวจสอบการเปลี่ยนแปลงของแถวได้โดยใช้การสนับสนุนเจอร์นัลของ DB2 for i5/OS

ในบางกรณี, รายการในบันทึกการตรวจสอบจะไม่เป็นไปในลำดับตามที่เกิดขึ้นจริง. ตัวอย่างเช่น, งานที่รันภายใต้ commitment control ที่ทำการลบตาราง, สร้างตารางใหม่โดยใช้ชื่อเดียวกับตารางที่ลบไป, แล้วทำการ commit. สิ่งเหล่านี้จะถูกบันทึกลงในเจอร์นัลตรวจสอบในลักษณะของการสร้างแล้วตามด้วยการลบออก. เนื่องจาก อ็อบเจ็กต์ที่ถูกสร้างขึ้นจะถูกบันทึกทันทีทันใด. อ็อบเจ็กต์ที่ถูกลบออกภายใต้ commitment control จะถูกซ่อนเอาไว้และยังไม่ถูกลบจริงจนกระทั่งทำการ commit เรียบร้อยแล้ว. ทันทีที่ทำการ commit เสร็จเรียบร้อย, การทำงานนั้นจะถูกบันทึกเอาไว้.

สิ่งอ้างอิงที่เกี่ยวข้อง

การอ้างอิงความปลอดภัย

Data integrity

Data integrity ป้องกันข้อมูลจากการถูกทำลาย หรือเปลี่ยนแปลงโดยผู้ที่ไม่มีสิทธิในการใช้งาน, การดำเนินการของระบบ หรือความขัดข้องของฮาร์ดแวร์ (เช่น ความเสียหายที่เกิดกับดิสก์ในด้านฟิสิคัล), ข้อผิดพลาดในการเขียนโปรแกรม, การขัดจังหวะก่อนที่งานจะเสร็จสมบูรณ์ (เช่น ความล้มเหลวในการจ่ายกระแสไฟฟ้า), หรือสอดแทรกจากการรันแอ็พพลิเคชันอื่นในเวลาเดียวกัน (เช่น ปัญหาที่เกิดต่อเนื่องกันไป).

สิ่งอ้างอิงที่เกี่ยวข้อง

XA APIs

Concurrency

Concurrency เป็นความสามารถในการรองรับผู้ใช้หลายคนในการเข้าถึงและเปลี่ยนแปลงข้อมูลในตารางเดียวกันหรือ มุมมองเดียวกันในเวลาเดียวกันโดยปราศจากความเสี่ยงในการสูญเสีย data integrity.

ความสามารถนี้จะถูกจัดไว้ให้โดยอัตโนมัติโดยตัวจัดการฐานข้อมูลDB2 for i5/OS. ล็อกเป็นสิ่งที่จำเป็นสำหรับตารางและแถว ซึ่งมีไว้เพื่อปกป้องไม่ให้ผู้ใช้หลายคนทำการเปลี่ยนแปลงข้อมูลชุดเดียวกันในเวลาที่ตรงกันพอดี.

โดยปกติ DB2 for i5/OS จำเป็น ที่จะต้องมีล็อกบนแถวเพื่อยืนยัน integrity อย่างไรก็ตาม, ในบางสถานการณ์ต้องการให้ DB2 for i5/OS มีการล็อกในระดับตารางแทนที่จะเป็นการล็อกแถว

ตัวอย่างเช่น, ล็อกอัพเดต (แบบเฉพาะ) ของแถวซึ่งถูกระงับการทำงานเคอร์เซอร์ตัวหนึ่งสามารถถูกเรียกใช้โดยเคอร์เซอร์อีกตัวหนึ่งในโปรแกรมเดียวกัน (หรือใน DELETE หรือ UPDATE คำสั่งที่ไม่เกี่ยวข้องกับเคอร์เซอร์นั้น). การทำแบบนี้จะป้องกันคำสั่ง UPDATE หรือ DELETE ที่ถูกกำหนดไว้ซึ่งเป็นตัวอ้างถึงเคอร์เซอร์ตัวแรก ไปจนกระทั่ง มีการ FETCH ครั้งต่อไปเกิดขึ้น. ล็อคแบบอ่าน (ไม่มีการอัพเดตร่วมกัน) ของแถวซึ่งถูกระงับการทำงาน โดยเคอร์เซอร์ตัวหนึ่ง จะไม่มีการป้องกันเคอร์เซอร์ตัวอื่นจากโปรแกรมเดียวกัน (หรือคำสั่ง DELETE หรือ UPDATE) จากการเรียกใช้ล็อกในแถวเดียวกัน .

สามารถใช้ค่า lock-wait timeout ที่เป็นค่าดีฟอลต์ และค่าที่ผู้ใช้กำหนดได้ DB2 for i5/OS สร้างตาราง, มุมมอง, และดรรชนีด้วยค่าดีฟอลต์ของ record wait time (60 วินาที) และค่าปกติของ file wait time (*IMMED) โดย wait time ของล็อกนี้จะถูกนำมาใช้สำหรับคำสั่ง data manipulation language (DML) สามารถทำการเปลี่ยนแปลงค่าเหล่านี้โดยใช้คำสั่ง CL นั่นคือ Change Physical File (CHGPF), Change Logical File (CHGLF), และ Override Database File (OVRDBF).

wait time ของล็อกที่ใช้ในคำสั่ง data definition language (DDL) ทั้งหมด และคำสั่ง LOCK TABLE จะเป็นค่า wait time ปกติของงาน (DFTWAIT) คุณสามารถเปลี่ยนค่านี้ได้โดยใช้คำสั่ง CL ที่เป็น Change Job (CHGJOB) หรือ Change Class (CHGCLS)

หากมีการระบุค่า record wait time ไว้มากๆ จะมีการเตรียมการตรวจหา deadlock เอาไว้ด้วย ตัวอย่างเช่น สมมติว่างานชิ้นหนึ่งมีล็อกเฉพาะอยู่บนแถวที่ 1 และงานอีกชิ้นหนึ่งมีล็อกเฉพาะอยู่บนแถวที่ 2 ถ้างานชิ้นแรกพยายามที่จะล็อกแถวที่ 2 จะต้องรอเนื่องจากงานชิ้นที่ 2 มีล็อกอยู่บนแถวที่ 2 นั้น ถ้างานชิ้นที่สองพยายามที่จะล็อกแถวที่ 1 DB2 for i5/OS จะตรวจพบว่ามีงานสองงานอยู่ใน deadlock และข้อผิดพลาดจะถูกส่งไปยังงานชิ้นที่ 2

คุณสามารถป้องกันผู้ใช้คนอื่นๆ จากการใช้ตารางในเวลาเดียวกันโดยใช้คำสั่ง SQL LOCK TABLE. การใช้ COMMIT(*RR) จะช่วยป้องกันผู้ใช้คนอื่นๆ จากการใช้ตารางร่วมกันในระหว่างการทำงานแต่ละครั้ง.

เพื่อปรับปรุงประสิทธิภาพการทำงาน DB2 for i5/OS จะปล่อยให้ open data path (ODP) เปิดอยู่เสมอ คุณสมบัติในการดำเนินการนี้จะปล่อยล็อกไว้บนตารางที่อ้างอิงโดย ODP, แต่จะไม่ทิ้งล็อกใดๆ ไว้บนแถว. ล็อกที่ถูกทิ้งไว้บนตารางจะป้องกันไม่ให้งานอื่นดำเนินการบนตารางนั้น อย่างไรก็ตาม โดยมากแล้ว DB2 for i5/OS จะตรวจพบว่ามีงานอื่นๆ มีล็อกอยู่และจะมีการส่งสัญญาณเกี่ยวกับเหตุการณ์ต่างๆ ไปยังงานเหล่านั้น เหตุการณ์นี้ทำให้ DB2 for i5/OS ปิด ODP ใดๆ (และทำการปล่อยล็อกในตาราง) ที่เชื่อมโยงกับตารางนั้น และขณะนี้จะเปิดเฉพาะในกรณีที่มีเหตุผลเกี่ยวกับการดำเนินงานเท่านั้น.

หมายเหตุ: wait timeout ของล็อกจะต้องมีค่ามากพอสำหรับการส่งสัญญาณต่อเหตุการณ์และงานอื่นๆ เพื่อที่จะปิด ODP หรือส่งข้อผิดพลาดกลับมา

นอกจากจะใช้คำสั่ง LOCK TABLE ในการเรียกใช้ล็อกของตาราง, หรือใช้ COMMIT(*ALL) หรือ COMMIT(*RR), ข้อมูลที่ถูกอ่านโดยงานงานหนึ่งอาจถูกเปลี่ยนแปลงได้โดยงานอีกงานหนึ่ง. โดยปกติ, ข้อมูลที่อ่านในขณะที่ คำสั่ง SQL ทำงานจะเป็นข้อมูลที่เป็นปัจจุบันมาก (ตัวอย่างเช่น, ระหว่างใช้คำสั่ง FETCH). อย่างไรก็ตาม, ในกรณีตัวอย่างนี้, ข้อมูลถูกอ่านขึ้นมาก่อนการทำงานของคำสั่ง SQL ซึ่งเป็นผลให้ได้ข้อมูลที่ไม่เป็นปัจจุบันได้ (ตัวอย่าง, ระหว่างคำสั่ง OPEN).

- ถ้าระบุค่า ALWCPYDTA(*OPTIMIZE) optimizer จะกำหนดว่าการทำสำเนาข้อมูลจะทำงานได้ดีกว่าการไม่ทำสำเนา.
- การสืบค้นบางอย่างต้องอาศัยตัวจัดการฐานข้อมูลในการสร้างตารางผลลัพธ์ชั่วคราว. ข้อมูลที่อยู่ในตารางผลลัพธ์ชั่วคราวนี้จะไม่แสดงการเปลี่ยนแปลงที่เกิดขึ้นหลังจากที่เคอร์เซอร์ถูกเปิด สำหรับข้อมูลเพิ่มเติมเกี่ยวกับกรณีที่ตารางผลลัพธ์ชั่วคราว ถูกสร้างขึ้น โปรดดู DECLARE CURSOR ในกลุ่มหัวข้อการอ้างอิง SQL
- การสืบค้นย่อยระดับต้นจะถูกประเมินผลเมื่อการสืบค้นนั้นถูกเปิด.

คุณสามารถใช้ SKIP LOCKED DATA clause เพื่อลด transaction wait time ให้น้อยลงอย่างมาก โดยอนุญาตให้ transaction ข้ามแถว ที่ถูกล็อกอย่างเข้ากันไม่ได้โดย transaction อื่นๆ โดยไม่รอให้มีการ ปลดล็อก ฟังก์ชันนี้ให้ความยืดหยุ่นและเพิ่มประสิทธิภาพ สำหรับแอ็พพลิเคชันที่กำหนดให้มีการส่งคืนข้อมูลที่มีอยู่และถูก commit แล้วเท่านั้น ควรใช้ SKIP LOCKED DATA เฉพาะในกรณีที่แอ็พพลิเคชันของคุณไม่ต้องการ ข้อมูลทั้งหมดจากเคียวรี่

- สามารถระบุ SKIP LOCKED DATA clause สำหรับคำสั่ง select, SELECT INTO, PREPARE, searched UPDATE หรือ searched DELETE โดย SKIP LOCKED DATA จะใช้กับตารางทั้งหมดที่ถูกอ้างถึงในคำสั่ง ถ้ามีหลายตาราง ที่เกี่ยวข้องในเคียวรี่ จำนวนแถวที่ถูก commit ที่สามารถส่งคืนได้ อาจจะลดลง เพราะแถวในหลายๆ ตารางอาจถูกล็อกโดย transaction อื่นๆ
- หากระดับการแยกสำหรับคำสั่งคือ COMMIT(*NONE), COMMIT(*CHG) หรือ COMMIT(*RR) ก็จะละเว้น SKIP LOCKED DATA clause

สิ่งอ้างอิงที่เกี่ยวข้อง

LOCK TABLE

การทำเจอร์นัล

การสนับสนุนเจอร์นัลของ DB2 for i5/OS จะช่วย เป็นหลักฐานการตรวจสอบและการกู้คืนทั้งแบบ forward และ backward

Forward recovery หรือ การกู้คืนแบบส่งต่อ คือการนำตารางเวอร์ชันที่เก่ากว่ามาทำการเปลี่ยนแปลงข้อมูลตามบันทึกของเจอร์นัล. Backward recovery หรือการกู้คืนแบบเรียกคืน คือการยกเลิกการเปลี่ยนแปลงตามที่ได้บันทึกไว้ในเจอร์นัล.

เมื่อแบบแผน SQL ถูกสร้างขึ้น, เจอร์นัลและ receiver จะถูกสร้างขึ้นในแบบแผน. เมื่อ SQL สร้างเจอร์นัลและ เจอร์นัลreceiver ขึ้น, ทั้งหมดจะถูกสร้างขึ้นในส่วนของพูลหน่วยความจำสำรอง (ASP) ของผู้ใช้ ถ้ามีการระบุ ASP clause ในคำสั่ง CREATE SCHEMA. อย่างไรก็ตาม, เนื่องจากการกำหนด journal receiver ในส่วนของ ASP ทำให้ประสิทธิภาพการทำงานดีขึ้น, ผู้ที่จัดการเกี่ยวกับเจอร์นัลนั้นอาจจะต้องการที่จะสร้าง journal receiver ที่จะมีในอนาคตในส่วนของ ASP ที่แยกออกไปต่างหาก .

เมื่อตารางถูกสร้างขึ้นในแบบแผน จะมีการบันทึกเกิดขึ้นโดยอัตโนมัติในเจอร์นัล ที่ DB2 for i5/OS สร้างขึ้น ในแบบแผน (QSQJRN) ตารางที่สร้างขึ้นในไลบรารีจะมีการทำเจอร์นัล เริ่มขึ้นถ้ามีเจอร์นัลที่ชื่อ QSQJRN ปรากฏในไลบรารี หลังจากจุดนี้ไป, จะเป็นความรับผิดชอบของคุณที่จะใช้ฟังก์ชันของบันทึกในการจัดการเจอร์นัล, journal receiver, และการทำเจอร์นับตารางลงในเจอร์นัล. ตัวอย่างเช่น, ถ้าตารางถูกย้ายไปอยู่ในแบบแผน, ระบบจะไม่ทำการเปลี่ยนแปลงสถานะการทำเจอร์นัลโดยอัตโนมัติ. ถ้าตารางถูกเรียกคืนมา, กฎของเจอร์นัลปกติจะถูกนำมาใช้. นั่นคือ ถ้า ตารางถูกทำเจอร์นัลในเวลาที่บันทึกข้อมูล ตารางจะถูกบันทึกลงในเจอร์นัลเดียวกันกับ เวลาที่เรียกคืนข้อมูล ถ้าตารางไม่ถูกทำเจอร์นัลในเวลาที่บันทึกข้อมูล เจอร์นัลจะไม่ถูกบันทึกในเวลาที่เรียกคืน

เจอร์นัลที่ถูกสร้างขึ้นใน SQL schema โดยปกติจะเป็นเจอร์นัลที่ถูกใช้สำหรับบันทึกการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นกับตาราง SQL อย่างไรก็ตาม, คุณสามารถ, ใช้ฟังก์ชันเจอร์นัลของระบบในการบันทึกตาราง SQL ลงในเจอร์นัลที่ต่างกัน.

ผู้ใช้สามารถหยุดการทำเจอร์นัลตารางใดๆ โดยใช้ฟังก์ชันเจอร์นัล, แต่การทำดังกล่าวจะป้องกันไม่ให้แอ็พพลิเคชันรันภายใต้ commitment control. ถ้าการทำเจอร์นัลถูกหยุดในตารางที่เป็น parent ของข้อจำกัดที่อ้างอิงถึง โดยกฎการลบของ NO ACTION, CASCADE, SET NULL, หรือ SET DEFAULT, การดำเนินการอัพเดตและการลบทั้งหมดจะไม่เกิดขึ้น. มิฉะนั้น, แอ็พพลิเคชันจะยังสามารถทำงานได้ ถ้าคุณได้ระบุ COMMIT(*NONE); อย่างไรก็ตาม, การทำเช่นนี้จะไม่ทำให้เกิด integrity ระดับเดียวกันกับที่ได้จากการทำเจอร์นัล และ commitment control.

หลักการที่เกี่ยวข้อง

การจัดการเจอร์นัล

สิ่งอ้างอิงที่เกี่ยวข้อง

“การอัพเดตตารางด้วยข้อจำกัดในการอ้างอิง” ในหน้า 104

หากคุณอัพเดตตาราง parent คุณไม่สามารถ แก้ไข primary key ที่มีแถว dependent อยู่ได้

Commitment control

การสนับสนุน DB2 for i5/OS commitment control จะรวบรวมวิธีการประมวลผลกลุ่มของการเปลี่ยนแปลงของฐานข้อมูล เช่น อัพเดต, แทรก หรือลบการดำเนินการ หรือการดำเนินการ data definition language (DDL) ใน ลักษณะของหน่วยการทำงานเดี่ยว (หรือที่เรียกว่า *transaction*)

commit operation รับประกันว่ากลุ่มของการปฏิบัติการได้ถูกดำเนินการอย่างสมบูรณ์. การทำ rollback รับประกันว่ากลุ่มของการปฏิบัติการได้ถูกส่งกลับออกมา. Savepoint สามารถนำมาใช้ในการแบ่ง transaction ให้เป็นหน่วยที่เล็กลงทำให้สามารถทำการ roll back ได้. Commit operation สามารถรับคำสั่งผ่านทางอินเตอร์เฟซที่แตกต่างกันได้หลายทาง . ตัวอย่างเช่น,

- SQL COMMIT statement
- คำสั่ง CL COMMIT
- คำสั่ง language commit(เช่น คำสั่ง RPG COMMIT)

การทำ rollback สามารถรับคำสั่งผ่านอินเตอร์เฟซที่ต่างกันได้หลายทาง. ตัวอย่างเช่น,

- SQL ROLLBACK statement
- คำสั่ง CL ROLLBACK
- คำสั่ง language rollback (เช่น คำสั่ง RPG ROLBK)

มีคำสั่ง SQL เพียงชุดเดียวที่ไม่สามารถทำการ commit หรือ roll back ได้นั่นคือ:

- DROP SCHEMA

- GRANT หรือ REVOKE ถ้ามีผู้ถือสิทธิในการใช้งานปรากฏอยู่ในอ็อบเจ็กต์ที่ได้ระบุเอาไว้

ถ้า commitment control ยังไม่ได้เริ่มทำงาน เมื่อ มีการเรียกใช้คำสั่ง SQL ด้วยระดับการแยกกันอื่นๆ ที่ไม่ใช่ COMMIT (*NONE) หรือเมื่อเรียกใช้คำสั่ง RELEASE, DB2 for i5/OS จะ ตั้งค่าสภาวะแวดล้อมของ commitment control โดยการเรียกคำสั่ง CL นั่นคือ Start Commitment Control (STRCMTCTL) DB2 for i5/OS ระบุ พารามิเตอร์ NFYOBJ(*NONE) และ CMTSCOPE(*ACTGRP) พร้อมกับพารามิเตอร์ LCKLVL บนคำสั่ง STRCMTCTL พารามิเตอร์ LCKLVL ที่ระบุเป็นระดับของล็อกบนพารามิเตอร์ COMMIT ของคำสั่ง Create SQL (CRTSQL*xxx*), Start SQL Interactive Session (STRSQL) หรือ Run SQL Statements (RUNSQLSTM) ใน REXX พารามิเตอร์ LCKLVL ที่ระบุเป็นระดับของล็อกบนคำสั่ง SET OPTION คุณสามารถใช้คำสั่ง STRCMTCTL เพื่อระบุพารามิเตอร์ CMTSCOPE, NFYOBJ, หรือ LCKLVL ที่แตกต่าง ถ้าคุณระบุ CMTSCOPE(*JOB) เพื่อเริ่มการทำงาน job-level commitment definition, DB2 for i5/OS จะใช้ job-level commitment definition นั้นสำหรับโปรแกรมใน activation group นั้น

หมายเหตุ:

- เมื่อใช้ commitment control ตารางที่ถูกอ้างถึงในแอ็พพลิเคชันโปรแกรม โดยคำสั่ง data manipulation language (DML) จะต้องมีการทำบันทึกลง เจอร์นัลไว้
- พารามิเตอร์ LCKLVL ที่ระบุไว้เป็นเพียงแค่ค่าปกติของระดับ ของล็อก หลังจาก commitment control เริ่มทำงาน คำสั่ง SET TRANSACTION SQL และ ระดับของล็อกที่ระบุเอาไว้ในพารามิเตอร์ COMMIT ในคำสั่ง CRTSQL*xxx*, STRSQL หรือ RUNSQLSTM จะแทนที่ทับค่าเดิมซึ่งเป็นค่าปกติของระดับล็อก นอกจากนี้ พารามิเตอร์ LCKLVL จะใช้กับการดำเนินการ commitment control ที่ถูกร้องขอ ผ่านทางอินเตอร์เฟสระบบแบบเก่าของ i5/OS (ไม่ใช่ SQL) เท่านั้น ระดับล็อกที่ระบุเอาไว้บนพารามิเตอร์ LCKLVL จะไม่ได้รับผลกระทบจากการเปลี่ยนแปลงในภายหลังต่อระดับการแยกกันของ SQL ที่ ถูกกระทำ โดยใช้คำสั่ง SET TRANSACTION เป็นต้น

สำหรับเคอร์เซอร์ที่ใช้ฟังก์ชันการรวม, GROUP BY หรือ HAVING และกำลังรันภายใต้ commitment control, ROLLBACK HOLD จะไม่มีผลใดๆ ต่อตำแหน่งเคอร์เซอร์ นอกจากนั้น, สิ่งต่างๆ ต่อไปนี้จะเกิดขึ้นภายใต้ commitment control:

- ถ้า COMMIT(*CHG) และ (ALWBLK(*NO) หรือ (ALWBLK(*READ)) ถูกระบุสำหรับเคอร์เซอร์ใดต่อไปนี้, ข้อความ (CPI430B) จะถูกส่งออกไปเพื่อแจ้งว่า COMMIT(*CHG) ถูกร้องขอแต่ไม่ได้รับอนุญาต.
- ถ้า COMMIT(*ALL), COMMIT(*RR) หรือ COMMIT(*CS) พร้อมกับ KEEP LOCKS clause ถูกระบุเอาไว้สำหรับเคอร์เซอร์อันใดอันหนึ่ง DB2 for i5/OS จะล็อก ตารางที่ถูกอ้างถึงทั้งหมดให้อยู่ในโหมดใช้ร่วมกัน (*SHRNUP) การล็อกจะป้องกันการประมวลผลพร้อมกันเนื่องจากการเรียกใช้ปฏิบัติการใดๆ ยกเว้นปฏิบัติการแบบ read-only ในตารางที่ระบุชื่อไว้ ข้อความ (SQL7902 หรือ CPI430A อย่างใดอย่างหนึ่ง) จะถูกส่งออกไปว่า COMMIT(*ALL), COMMIT(*RR), หรือ COMMIT(*CS) พร้อมกับ KEEP LOCKS clause ได้ถูกระบุไว้สำหรับเคอร์เซอร์ใดถูกร้องขอแต่ไม่ได้รับอนุญาต. ข้อความ SQL0595 อาจถูกส่งไปด้วยเช่นกัน

สำหรับเคอร์เซอร์ที่มี COMMIT(*ALL), COMMIT(*RR) หรือ COMMIT(*CS) พร้อมกันกับ KEEP LOCKS clause ที่ได้ระบุเอาไว้อยู่ และ ไฟล์แค็ตตาล็อกไฟล์ใดไฟล์หนึ่งถูกใช้ หรือจำเป็นต้องใช้ตารางผลลัพธ์ชั่วคราว, DB2 for i5/OS จะล็อกตารางที่ถูกอ้างถึงทั้งหมดให้อยู่ในโหมดใช้งานร่วมกัน (*SHRNUP) การล็อกจะป้องกันการประมวลผลพร้อมกันเนื่องจากการเรียกใช้ปฏิบัติการใดๆ ยกเว้นปฏิบัติการแบบ read-only ในตารางที่ระบุชื่อไว้ ข้อความ (SQL7902 หรือ CPI430Aอย่างใดอย่างหนึ่ง) จะถูกส่งออกมาว่า COMMIT(*ALL) ถูกร้องขอแต่ไม่ได้รับอนุญาต. ข้อความ SQL0595 อาจถูกส่งไปด้วยเช่นกัน

ถ้ามีการระบุ ALWBLK(*ALLREAD) และ COMMIT(*CHG) , ตอนที่โปรแกรมถูกพรีคอมไพล์, เคอร์เซอร์ที่เป็นแบบ read-only ทั้งหมดจะอนุญาติให้มีการจัดเป็นกลุ่มบล็อกของแถว โดยที่ ROLLBACK HOLD จะไม่ย้อนตำแหน่งของเคอร์เซอร์กลับมา.

ถ้า COMMIT(*RR) ถูกร้องของ, ตารางจะถูกล็อกจนกระทั่งปิดการสืบค้น. ถ้าเคอร์เซอร์เป็นแบบ read-only, ตารางจะถูกล็อก (*SHRNUP). ถ้าเคอร์เซอร์อยู่ในโหมดอัพเดต, ตารางจะถูกล็อก (*EXCLRD). เนื่องจากผู้ใช้คนอื่นๆ จะถูกล็อกอยู่นอกตาราง, การรันด้วยการอ่านแบบอ่านซ้ำได้จะช่วยป้องกันการเข้าถึงตารางพร้อมกันได้.

| ในสภาวะแวดล้อมที่มีความขัดแย้งสูง ซึ่งใช้ COMMIT(*RR) แอ็พพลิเคชันอาจจำเป็นต้องลองดำเนินการอีกครั้ง หลังจากที่
| ได้รับ SQL0913 เพื่ออนุญาตให้กลไกการปลดล็อกได้ทำงาน

หากมีการระบุระดับ isolation อื่นๆ ที่ไม่ใช่COMMIT(*NONE) และแอ็พพลิเคชันได้ทำการออกคำสั่ง ROLLBACK หรือ activation group สิ้นสุดการทำงานแบบไม่ปกติ (และ commitment definition ไม่ใช่ *JOB), ปฏิบัติการอัพเดต, การแทรก, การลบออก, และ DDL ทั้งหมดที่ถูกกระทำภายในช่วงหน่วยการทำงานจะถูกยับยั้งเอาไว้. ถ้าแอ็พพลิเคชันออกคำสั่ง COMMIT หรือ activation สิ้นสุดการทำงานแบบไม่ปกติ, ปฏิบัติการอัพเดต, การแทรก, การลบออก, และ DDL ทั้งหมดที่ถูกกระทำอยู่ในช่วงหน่วยการทำงานจะถูก commit เอาไว้.

DB2 for i5/OS ใช้ล็อกกับแถวเพื่อที่จะป้องกันงานอื่นๆ ไม่ให้เข้าถึงข้อมูลที่ถูกเปลี่ยนแปลงก่อนที่หน่วยการทำงานนั้นจะเสร็จสิ้น ถ้า COMMIT(*ALL) ถูกระบุเอาไว้, ล็อกการอ่านในแถวที่ถูกเรียกข้อมูลจะถูกใช้ป้องกันงานอื่นๆ จากการเปลี่ยนแปลงข้อมูลที่ถูกอ่านก่อนที่หน่วยการทำงานจะเสร็จสิ้น การทำเช่นนี้เป็นการป้องกันงานอื่นๆ จากการอ่านแถวที่ไม่มีการเปลี่ยนแปลง ซึ่งทำให้มั่นใจว่า, ถ้าหน่วยการทำงานเดียวกันทำการอ่านแถวนั้นซ้ำ, จะทำให้ได้ค่าผลลัพธ์เช่นเดิม. ล็อกป้องกันการอ่านจะไม่สามารถป้องกันงานอื่นจากการดึงข้อมูลในแถวเดียวกันออกมาใช้.

Commitment control สามารถจัดการกับการเปลี่ยนแปลงข้อมูลในแถวได้ถึง 500 ล้านแถวในหนึ่งหน่วยการทำงาน. ถ้า COMMIT(*ALL) หรือ COMMIT(*RR) ถูกระบุไว้, แถวที่ถูกอ่านทั้งหมดจะถูกรวมอยู่ในขีดจำกัดด้วย. (ถ้าแถวข้อมูลแถวหนึ่งถูกเปลี่ยนแปลงหรือถูกอ่านมากกว่าหนึ่งครั้งในหนึ่งหน่วยการทำงาน, จะถูกนับเป็นครั้งเดียวจากที่จำกัดไว้.) การมีล็อกเป็นจำนวนมากจะมีผลต่อประสิทธิภาพการทำงานของระบบและจะทำให้ผู้ใช้หลายคนเข้ามาใช้งานแถวที่ถูกล็อกไว้ในหน่วยการทำงานเดียวกันไม่ได้จนกระทั่งหน่วยนั้นทำงานเสร็จ. เรื่องที่ต้องคำนึงถึงที่สุดก็คือ ทำอย่างไรให้มีจำนวนแถวที่ถูกประมวลผลในหนึ่งหน่วยการทำงานมีค่าน้อยๆ.

COMMIT HOLD และ ROLLBACK HOLD อนุญาตให้เปิดเคอร์เซอร์ทิ้งไว้ได้และเริ่มการทำงานของหน่วยการทำงานอื่นโดยไม่จำเป็นต้องออกคำสั่ง OPEN อีกครั้งหนึ่ง ค่าของ HOLD จะไม่ปรากฏในกรณีที่คุณเชื่อมต่อกับฐานข้อมูลแบบรีโมตที่ไม่ได้อยู่ในแพล็ตฟอร์ม System i อย่างไรก็ตาม คุณสามารถใช้ตัวเลือก WITH HOLD ใน DECLARE CURSOR ในการทำให้เคอร์เซอร์เปิดหลังจากการ commit เคอร์เซอร์ชนิดนี้จะสนับสนุนการทำงานเมื่อเชื่อมต่อกับฐานข้อมูลแบบรีโมตที่ไม่ได้อยู่ในแพล็ตฟอร์ม System i เคอร์เซอร์นี้จะปิดในช่วงการ rollback.

ตารางที่ 36. ช่วงระยะเวลาในการล็อกของแถว

คำสั่ง SQL	พารามิเตอร์ COMMIT (ดูในหมายเหตุ 5)	ช่วงระยะเวลาของการล็อกของแถว	ชนิดของการล็อก
SELECT INTO SET variable VALUES INTO	*NONE *CHG *CS (ดูในหมายเหตุ 6) *ALL (ดูในหมายเหตุ 2 และ 7)	ไม่มีล็อก ไม่มีล็อก แถวจะถูกล็อกเมื่อถูกอ่านแล้วจะปล่อยล็อก ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	 READ READ

ตารางที่ 36. ช่วงระยะเวลาในการล็อกของแถว (ต่อ)

คำสั่ง SQL	พารามิเตอร์ COMMIT (ดูในหมายเหตุ 5)	ช่วงระยะเวลาของการล็อกของแถว	ชนิดของการล็อก
FETCH (cursor แบบ read-only)	*NONE	ไม่มีล็อก	
	*CHG	ไม่มีล็อก	
	*CS (ดูในหมายเหตุ 6)	ตั้งแต่การอ่านจนถึงการ FETCH ครั้งต่อไป	READ
	*ALL (ดูในหมายเหตุ 2 และ 7)	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	READ
FETCH (เคอร์เซอร์ที่สามารถอัพเดตหรือลบออกได้) (ดูในหมายเหตุ 1)	*NONE	เมื่อแถวข้อมูลไม่ได้ถูกอัพเดตหรือลบออก	UPDATE
		ตั้งแต่การอ่านจนถึงการ FETCH ครั้งต่อไป	
		เมื่อแถวข้อมูลถูกอัพเดต	
		ตั้งแต่การอ่านจนถึงการ FETCH ครั้งต่อไป	
		เมื่อแถวข้อมูลถูกลบออก	UPDATE
		ตั้งแต่การอ่านจนถึงการ DELETE ครั้งต่อไป	
	*CHG	เมื่อแถวข้อมูลไม่ได้ถูกอัพเดตหรือลบออก	
		ตั้งแต่การอ่านจนถึงการ FETCH ครั้งต่อไป	
		เมื่อแถวข้อมูลถูกอัพเดตหรือลบออก	UPDATE
		ตั้งแต่การอ่านจนถึง COMMIT หรือ ROLLBACK	
	*CS	เมื่อแถวข้อมูลไม่ได้ถูกอัพเดตหรือลบออก	
		ตั้งแต่การอ่านจนถึงการ FETCH ครั้งต่อไป	
		เมื่อแถวข้อมูลถูกอัพเดตหรือลบออก	UPDATE
		ตั้งแต่การอ่านจนถึง COMMIT หรือ ROLLBACK	
	*ALL	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	
INSERT (ตารางเป้าหมาย)	*NONE	ไม่มีล็อก	
	*CHG	ตั้งแต่การแทรกจนถึง ROLLBACK หรือ COMMIT	UPDATE
	*CS	ตั้งแต่การแทรกจนถึง ROLLBACK หรือ COMMIT	UPDATE
	*ALL	ตั้งแต่การแทรกจนถึง ROLLBACK หรือ COMMIT	UPDATE[3]
INSERT (ตารางในการเลือกย่อย)	*NONE	ไม่มีล็อก	
	*CHG	ไม่มีล็อก	
	*CS	แต่ละแถวจะถูกล็อกในขณะที่ถูกอ่าน	READ
	*ALL	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	READ
UPDATE (non-cursor)	*NONE	แต่ละแถวจะถูกล็อกในขณะที่ถูกอัพเดต	UPDATE
	*CHG	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
	*CS	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
	*ALL	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
DELETE (ไม่ใช้เคอร์เซอร์)	*NONE	แต่ละแถวจะถูกล็อกในขณะที่ถูกลบออก	UPDATE
	*CHG	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
	*CS	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
	*ALL	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
UPDATE (ใช้เคอร์เซอร์)	*NONE	ตั้งแต่การอ่านจนถึงการ FETCH ครั้งต่อไป	UPDATE
	*CHG	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
	*CS	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
	*ALL	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
DELETE (ใช้เคอร์เซอร์)	*NONE	ล็อกจะถูกปล่อยเมื่อแถวถูกลบออกไป	UPDATE
	*CHG	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
	*CS	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE
	*ALL	ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	UPDATE

ตารางที่ 36. ช่วงระยะเวลาในการล็อกของแถว (ต่อ)

คำสั่ง SQL	พารามิเตอร์ COMMIT (ดูในหมายเหตุ 5)	ช่วงระยะเวลาของการล็อกของแถว	ชนิดของการล็อก
การสืบค้นย่อย (เคอร์เซอร์ที่สามารถอัพเดตหรือลบออกได้ หรือ UPDATE หรือ DELETE โดยไม่ใช้เคอร์เซอร์)	*NONE *CHG *CS *ALL (ดูในหมายเหตุ 2)	ตั้งแต่การอ่านจนถึงการ FETCH ครั้งต่อไป ตั้งแต่การอ่านจนถึงการ FETCH ครั้งต่อไป ตั้งแต่การอ่านจนถึงการ FETCH ครั้งต่อไป ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	READ READ READ READ
เคียวรี่ย่อย (เคอร์เซอร์แบบ read-only หรือ SELECT INTO)	*NONE *CHG *CS *ALL	ไม่มีล็อก ไม่มีล็อก แต่ละแถวจะถูกล็อกในขณะที่ถูกอ่าน ตั้งแต่การอ่านจนกระทั่ง ROLLBACK หรือ COMMIT	 READ READ

หมายเหตุ:

1. เคอร์เซอร์จะเปิดไว้และอนุญาตให้ UPDATE หรือ DELETE ถ้าตารางผลลัพธ์ไม่ได้เป็นแบบ read-only และถ้าหนึ่งในสิ่งต่อไปนี้เป็นจริง:
   - เคอร์เซอร์ถูกระบุด้วย FOR UPDATE clause.
   - เคอร์เซอร์ไม่ถูกระบุด้วย FOR UPDATE, FOR READ ONLY, หรือ ORDER BY clause และโปรแกรมมีสิ่งต่างๆ ต่อไปนี้อย่างน้อยหนึ่งอย่าง:
     - Cursor UPDATE ที่อ้างถึงชื่อเดียวกัน
     - Cursor DELETE ที่อ้างถึงชื่อเดียวกัน
     - EXECUTE หรือ EXECUTE IMMEDIATEคำสั่งและ ALWBLK(*READ) หรือ ALWBLK(*NONE) ที่ถูกระบุในคำสั่ง CRTSQLxxx.
2. ตารางหรือ มุมมองสามารถถูกล็อกได้ต่างหากเพื่อที่จะให้รองรับคำสั่ง COMMIT(*ALL). ถ้าการเลือกย่อยถูกประมวลผลและรวม UNION, หรือถ้าการประมวลผลของการสืบค้นนั้นต้องการใช้ตารางผลลัพธ์ชั่วคราว, จำเป็นที่จะต้องใช้ล็อกต่างหากเพื่อไม่ให้คุณเห็นการเปลี่ยนแปลงที่ไม่มีการ commit.
3. ล็อกแบบ UPDATE ในแถวของตารางเป้าหมายและล็อกแบบ READ ในแถวของตารางการเลือกย่อย.
4. ตารางหรือ มุมมองสามารถถูกล็อกไว้ต่างหากได้เพื่อที่จะรองรับการอ่านแบบซ้ำๆ. การล็อกแถวจะยังคงดำเนินต่อไปในขณะที่มีการอ่านแบบซ้ำๆ. ล็อกเป็นสิ่งจำเป็นและมีช่วงระยะเวลาการทำงานเท่ากับ *ALL.
5. การล็อกแถวที่มีการอ่านแบบซ้ำๆ (*RR) จะเป็นเช่นเดียวกับล็อกที่ระบุไว้สำหรับ *ALL.
6. ถ้า KEEP LOCKS clause ถูกระบุไว้ด้วย *CS, ล็อกในการอ่านใดๆ จะถูกพักไว้จนกระทั่งปิดเคอร์เซอร์หรือการทำ COMMIT หรือ ROLLBACK เสร็จสิ้นลง. ถ้าไม่มีเคอร์เซอร์ใดที่เกี่ยวข้องกับ isolation clause แล้ว, ล็อกจะถูกพักไว้จนกระทั่งคำสั่ง SQL สิ้นสุดการทำงาน.
7. ถ้ามีการระบุ USE AND KEEP EXCLUSIVE LOCKS clause พร้อมด้วย *RS หรือระดับการแยก *RR, การล็อกของ UPDATE บนแถวจะถูกกระทำแทนการล็อกของ READ.

หลักการที่เกี่ยวข้อง

Commitment control

สิ่งอ้างอิงที่เกี่ยวข้อง

DECLARE CURSOR

ระดับการแยกกัน

XA APIs

Savepoints

savepoint คือ entity ที่ถูกตั้งชื่อ เพื่อแสดงสถานะของข้อมูล และแบบแผนที่จุดเวลาภายใน หน่วยของงาน คุณสามารถสร้าง savepoint ภายใน transaction ได้. ถ้า transaction ทำการ rolls back การเปลี่ยนแปลงต่างๆ จะไม่สำเร็จ และกลับไปยัง savepoint ที่ระบุไว้ แทนที่จะ กลับไปที่จุดเริ่มต้นของ transaction

คุณสามารถตั้ง savepoint โดยใช้คำสั่ง SAVEPOINT SQL ตัวอย่างเช่น, การสร้าง savepoint ที่ชื่อ STOP_HERE:

```
SAVEPOINT STOP_HERE
       ON ROLLBACK RETAIN CURSORS
```

ตรรกะของโปรแกรมในแอ็พพลิเคชันจะกำหนดว่าจะใช้ชื่อของ savepoint อีกครั้งในการระบุความก้าวหน้าของแอ็พพลิเคชัน, หรือถ้าชื่อของ savepoint ระบุตำแหน่งการทำงาน เฉพาะในแอ็พพลิเคชันก็ไม่ควรที่จะนำกลับมาใช้อีกครั้งหนึ่ง.

ถ้า savepoint นั้นแทนตำแน่งของงานเพียงหนึ่งเดียวที่ไม่สมควรจะถูกย้ายไปด้วยคำสั่ง SAVEPOINT อื่นๆ, ให้ระบุคีย์เวิร์ด UNIQUE. การทำเช่นนี้เป็นการป้องกันการนำชื่อมาใช้ใหม่โดยไม่ได้ตั้งใจซึ่งสามารถเกิดขึ้นโดยการเรียก stored procedure ซึ่งใช้ชื่อเดียวกับsavepoint นี้ในคำสั่ง SAVEPOINT. อย่างไรก็ตาม, ถ้าคำสั่ง SAVEPOINT ถูกนำไปใช้ใน loop, ก็ไม่ควรใช้คีย์เวิร์ด UNIQUE. คำสั่ง SQL ต่อไปนี้ตั้งชื่อ savepoint ที่ไม่ซ้ำกันที่ชื่อ START_OVER.

```
SAVEPOINT START_OVER UNIQUE
       ON ROLLBACK RETAIN CURSORS
```

ในการ rollback ไปยัง savepoint, ให้ใช้คำสั่ง ROLLBACK ตามด้วย TO SAVEPOINT clause. ตัวอย่างต่อไปนี้แสดงการใช้ SAVEPOINT และคำสั่ง ROLLBACK TO SAVEPOINT:

ตรรกะของแอ็พพลิเคชันนี้จะจองที่นั่งของสายการบินในวันที่ต้องการ, และจะทำการจองโรงแรม. ถ้าไม่มีที่พักว่างในโรงแรม, จะ roll back การจองที่นั่งของสายการบินและทำการดำเนินการอีกครั้งสำหรับวันอื่นแทน. ระบบจะพยายามทำเช่นนี้ 3 ครั้ง.

```
got_reservations =0;
EXEC SQL SAVEPOINT START_OVER UNIQUE ON ROLLBACK RETAIN CURSORS;

if (SQLCODE != 0) return;

for (i=0; i<3 & got_reservations == 0; ++i)
{
  Book_Air(dates(i), ok);
  if (ok)
  {
    Book_Hotel(dates(i), ok);
    if (ok) got_reservations = 1;
    else
    {
      EXEC SQL ROLLBACK TO SAVEPOINT START_OVER;
      if (SQLCODE != 0) return;
    }
  }
}

EXEC SQL RELEASE SAVEPOINT START_OVER;
```

Savepoint จะถูกยกเลิกโดยคำสั่ง RELEASE SAVEPOINT. ถ้าไม่ใช้คำสั่ง RELEASE SAVEPOINT เพื่อยกเลิก savepoint อย่างชัดเจน, คำสั่งดังกล่าวจะถูกเรียกเมื่อสิ้นสุดระดับ savepoint ปัจจุบันหรือสิ้นสุด transaction. คำสั่งต่อไปนี้จะยกเลิก savepoint START_OVER.

```
RELEASE SAVEPOINT START_OVER
```

Savepoint จะถูกเลิกใช้เมื่อ transaction ถูก commit หรือถูก roll back. ทันทีที่ชื่อของ savepoint ถูกยกเลิก, คุณจะ rollback กลับไปยังชื่อของ savepoint นั้นไม่ได้อีก. คำสั่ง COMMIT หรือ ROLLBACK จะยกเลิกชื่อ savepoint ทั้งหมดที่กำหนดไว้ใน transaction. เนื่องจากชื่อของ savepoint ทั้งหมดจะถูกยกเลิกภายใน transaction, ชื่อเหล่านั้นจึงสามารถนำไปใช้อีกครั้งในการ commit หรือ rollback.

Savepoint จะถูกกำหนดขอบเขตไว้สำหรับการเชื่อมต่อแบบเดี่ยวเท่านั้น. ในทันทีที่ savepoint ถูกกำหนดขึ้น, savepoint จะไม่ถูกกระจายไปยังฐานข้อมลแบบรีโมตทั้งหมดที่เชื่อมต่อกับแอ็พพลิเคชันนั้น. Savepoint จะใช้ได้กับฐานข้อมูลปัจจุบันที่แอ็พพลิเคชันนั้นเชื่อมต่ออยู่เมื่อ savepoint ถูกกำหนดขึ้นเท่านั้น.

คำสั่งเดี่ยวสามารถเรียกใช้ ฟังก์ชันที่ผู้ใช้กำหนด, ทริกเกอร์, หรือกระบวนการที่บันทึกไว้ได้ทั้งโดยนัยหรือโดยชัดเจน. โดยรู้จักกันในชื่อของ nesting หรือ การซ้อนภายใน. ในบางกรณีเมื่อการซ้อนภายในระดับใหม่ถูกเริ่มขึ้น, ระดับของ savepoint ใหม่จะถูกเริ่มตามไปด้วย. ระดับของ savepoint ใหม่นั้นจะแยกการเรียกใช้แอ็พพลิเคชันจากการทำงานใดๆ ของ savepoint โดยใช้รูทีนหรือ ทริกเกอร์ระดับที่ต่ำกว่า.

Savepoint สามารถถูกอ้างอิงได้เฉพาะในระดับของ savepoint เดียวกัน (หรือขอบเขตเดียวกัน) กับที่ระบุ ไว้ savepoint นั้นไว้. คุณไม่สามารถนำคำสั่ง ROLLBACK TO SAVEPOINT มาใช้ในการ rollback ไปยัง savepoint ที่กำหนดไว้ภายนอกระดับปัจจุบันของ savepoint นั้น. ในลักษณะเดียวกัน, คำสั่ง RELEASE SAVEPOINT ไม่สามารถถูกนำมาใช้ในการเรียกใช้ savepoint ที่กำหนดไว้ภายนอกระดับปัจจุบันของ savepoint นั้น. ตารางต่อไปนี้จะช่วยสรุปว่าเมื่อใดที่ระดับของ savepoint จะถูกเริ่มขึ้นและสิ้นสุดลง:

ระดับใหม่ของ savepoint จะเริ่มขึ้นเมื่อ:	**ระดับ savepoint นั้นสิ้นสุดลงเมื่อ:**
หน่วยการทำงานใหม่เริ่มทำงาน	มีการเรียกใช้ COMMIT หรือ ROLLBACK
มีการเรียกใช้ทริกเกอร์	ทริกเกอร์เสร็จสิ้นลง
มีการเรียกใช้ฟังก์ชันที่ผู้ใช้ระบุ	มีการส่งฟังก์ชันที่ผู้ใช้ระบุกลับคืนไปยังผู้เรียกใช้งาน
กระบวนการที่ถูกบันทึกไว้จะถูกเรียกมาใช้, และกระบวนการนั้นจะถูกสร้างขึ้นโดย NEW SAVEPOINT LEVEL clause	กระบวนการที่ถูกบันทึกไว้จะถูกส่งกลับไปยังผู้เรียกใช้งาน
จะมี BEGIN สำหรับคำสั่ง ATOMIC compound SQL	จะมี END สำหรับคำสั่ง ATOMIC compound

savepoint ที่ถูกกำหนดขึ้นในระดับ savepoint จะยกเลิกโดยนัยเมื่อระดับ savepoint นั้นสิ้นสุดลง.

Atomic operations

ในการรันภายใต้ COMMIT(*CHG), COMMIT(*CS), or COMMIT(*ALL), ปฏิบัติการทั้งหมดถือว่าเป็นแบบ atomic.

นั่นคือ, ปฏิบัติการเหล่านั้นจะเสร็จสิ้นการทำงานหรือเสมือนว่าไม่มีการเริ่มการทำงาน . ลักษณะดังกล่าวจะเป็นจริงโดยไม่ต้องคำนึงถึงว่าฟังก์ชันถูกทำให้สิ้นสุดลงหรือถูกขัดจังหวะเมื่อไรหรืออย่างไร (ได้แก่ ไฟฟ้าขัดข้อง, การสิ้นสุดการทำงานแบบไม่ปกติ, หรือถูกยกเลิก).

ถ้า COMMIT (*NONE) ถูกระบุ, อย่างไรก็ตาม, ฐานข้อมูลที่เป็นรากฐานของฟังก์ชันของ data definition บางตัวจะไม่นับเป็น atomic. คำสั่ง SQL data definition ต่อไปนี้ถือว่าเป็นแบบ atomic:

- ALTER TABLE (ดูในหมายเหตุ 1)
- COMMENT ON (ดูในหมายเหตุ 2)

- LABEL ON (ดูในหมายเหตุ 2)
- GRANT (ดูในหมายเหตุ 3)
- REVOKE (ดูในหมายเหตุ 3)
- DROP TABLE (ดูในหมายเหตุ 4)
- DROP VIEW (ดูในหมายเหตุ 4)
- DROP INDEX
- DROP PACKAGE
- REFRESH TABLE

หมายเหตุ:

1. ถ้าจำเป็นต้องเพิ่มหรือลบข้อจำกัดใดๆ, รวมทั้งการเปลี่ยน definition ของคอลัมน์, ปฏิบัติการเหล่านั้นจะถูกประมวลผลครั้งละหนึ่งปฏิบัติการ, ดังนั้นคำสั่ง SQL ทั้งหมดไม่เป็นแบบ atomic. ลำดับของปฏิบัติการจะเป็นดังนี้:
   - ลบข้อจำกัดออก
   - ละคอลัมน์ที่ถูกระบุตัวเลือก RESTRICT เอาไว้
   - Definition ของคอลัมน์อื่นๆ มีการเปลี่ยนแปลง (DROP COLUMN CASCADE, ALTER COLUMN, ADD COLUMN)
   - เพิ่มข้อจำกัด
2. ถ้าคอลัมน์หลายๆ คอลัมน์ถูกระบุในคำสั่ง COMMENT ON หรือ LABEL ON, คอลัมน์เหล่านั้นจะถูกประมวลผลครั้งละหนึ่งคอลัมน์, ดังนั้นคำสั่ง SQL ทั้งหมดจึงไม่เป็น atomic, แต่ COMMENT ON หรือ LABEL ON ในแต่ละคอลัมน์หรืออ็อบเจ็กต์จะถือว่าเป็น atomic.
3. ถ้าตารางหลายตาราง, แพ็คเก็จ SQL หลายแพ็คเก็จ, หรือ ผู้ใช้หลายๆ คนถูกระบุไว้ในคำสั่ง GRANT หรือ REVOKE, ตารางจะถูกประมวลผลครั้งละหนึ่งตาราง, ดังนั้นคำสั่ง SQL ทั้งหมดจึงไม่เป็น atomic, แต่ GRANT หรือ REVOKE ของแต่ละตารางจะถือว่าเป็น atomic.
4. ถ้าต้องมีการละมุมมอง dependent ทิ้งไปในขณะที่ DROP TABLE หรือ DROP VIEW, แต่ละมุมมอง dependent จะถูกประมวลผลครั้งละหนึ่งมุมมอง, ดังนั้นคำสั่ง SQL ทั้งหมดจึงไม่เป็น atomic.

คำสั่ง data definition ต่อไปนี้ไม่เป็นแบบ atomic เนื่องจากคำสั่งเหล่านั้นเกี่ยวพันกับปฏิบัติการทางฐานข้อมูลมากกว่าหนึ่งฐานข้อมูล:

- ALTER FUNCTION
- ALTER PROCEDURE
- ALTER SEQUENCE
- CREATE ALIAS
- CREATE DISTINCT TYPE
- CREATE FUNCTION
- CREATE INDEX
- CREATE PROCEDURE
- CREATE SCHEMA

- CREATE SEQUENCE
- CREATE TABLE
- CREATE TRIGGER
- CREATE VIEW
- DROP ALIAS
- DROP DISTINCT TYPE
- DROP FUNCTION
- DROP PROCEDURE
- DROP SCHEMA
- DROP SEQUENCE
- DROP TRIGGER
- RENAME (ดูใน note 1)

หมายเหตุ: RENAME เป็น atomic ก็ต่อเมื่อชื่อหรือชื่อของระบบถูกเปลี่ยนแปลง. เมื่อสองสิ่งนี้ถูกเปลี่ยนไป, RENAME ไม่ถือว่าเป็น atomic.

ตัวอย่างเช่น คำสั่ง CREATE TABLE สามารถถูกขัดจังหวะ หลังจากที่ไฟล์ของ DB2 for i5/OS ถูกสร้างขึ้น ก่อนที่รายการย่อยจะถูกเพิ่มเข้าไป ดังนั้นในกรณี ของคำสั่งที่เกี่ยวกับการสร้าง ถ้าปฏิบัติการใดสิ้นสุดลงโดยไม่ปกติ คุณอาจ จำเป็นต้องละอ็อบเจ็กต์ทิ้งไป แล้วสร้างขึ้นมาใหม่อีกครั้ง ในกรณีของคำสั่ง DROP SCHEMA คุณจำเป็นต้องละทิ้ง schema อีกครั้ง หรือใช้คำสั่ง CL command Delete Library (DLTLIB) เพื่อลบส่วนที่ค้างอยู่ของ schema

ข้อจำกัด

ฐานข้อมูล DB2 for i5/OS สนับสนุน ข้อจำกัดเฉพาะ, ข้อจำกัดในการอ้างอิง และข้อจำกัดในการตรวจสอบ

ข้อจำกัดที่เป็นเอกลักษณ์นั้นเป็นกฎที่รับประกันว่าค่าของคีย์มีเพียงหนึ่งเดียว. ข้อจำกัดที่อ้างอิงได้นั้น คือ กฎที่ว่า foreign key ที่ไม่มีค่าเป็น null ทั้งหมดในตาราง dependent นั้น มี parent key ที่เกี่ยวข้องกันในตาราง parent. ข้อจำกัดการตรวจสอบ คือ กฎที่จำกัดค่าที่ใช้ในคอลัมน์หรือกลุ่มคอลัมน์.

DB2 for i5/OS จะกำหนด ความถูกต้องของข้อจำกัดในช่วงคำสั่งภาษาที่ใช้ในการดำเนินการข้อมูล (DML) ใดๆ อย่างไรก็ตาม ปฏิบัติการบางอย่าง (เช่น การบันทึกตาราง dependent) ทำให้ความถูกต้องของข้อจำกัดนั้นไม่เป็นที่ทราบแน่ชัด ในกรณีนี้ คำสั่ง DML จะถูกป้องกันจนกระทั่ง DB2 for i5/OS ได้รับการตรวจสอบความถูกต้องของข้อจำกัดนั้น

- ข้อจำกัดเฉพาะจะดำเนินการพร้อมด้วยดรรชนี. ถ้าดรรชนีหนึ่งที่ใช้กับข้อจำกัดเฉพาะนั้นไม่ถูกต้อง คำสั่ง Edit Rebuild of Access Paths (EDTRBDAP) จะถูกนำมาใช้ในการแสดงดรรชนีใดๆ ที่จำเป็นต้องสร้างขึ้นใหม่
- ถ้าในขณะนั้น DB2 for i5/OS ไม่ทราบว่า ข้อจำกัดที่อ้างอิงได้หรือข้อจำกัดที่ใช้ตรวจสอบได้มีค่าที่ถูกต้อง, ข้อจำกัดจะถูกนับว่าอยู่ในสถานะรอการตรวจสอบ. คำสั่ง Edit Check Pending Constraints (EDTCPCST) สามารถใช้ในการแสดงผลดรรชนีใดๆ ที่จำเป็นต้องสร้างใหม่ในขณะนั้น

หลักการที่เกี่ยวข้อง

"ข้อจำกัด" ในหน้า 12

ข้อจำกัด เป็นกฎที่บังคับใช้โดยตัวจัดการฐานข้อมูล เพื่อจำกัดค่าที่สามารถแทรก ลบ หรืออัพเดต ในตาราง

การเพิ่ม และการใช้ข้อจำกัดในการตรวจสอบ:

ข้อจำกัดในการตรวจสอบ รับรองถึง ความถูกต้องของข้อมูลระหว่างการแทรกและอัพเดต ด้วยการจำกัดค่าที่ใช้ได้ในคอลัมน์หรือกลุ่มคอลัมน์

ใช้คำสั่ง SQL CREATE TABLE และ ALTER TABLE เพื่อเพิ่ม หรือลบข้อจำกัดในการตรวจสอบ.

ในตัวอย่างนี้, คำสั่งต่อไปนี้เป็นการสร้างตารางที่มีสามคอลัมน์ และหนึ่งข้อจำกัดในการตรวจสอบบน COL2 ซึ่งจำกัดค่าที่ใช้ได้ในคอลัมน์นั้นให้เป็นจำนวนเต็มบวก:

```
CREATE TABLE T1 (COL1 INT, COL2 INT CHECK (COL2>0), COL3 INT)
```

จากตาราง, คำสั่งต่อไปนี้:

```
INSERT INTO T1 VALUES (-1, -1, -1)
```

ใช้ไม่ได้เนื่องจากค่าที่ต้องใส่ใน COL2 ไม่เป็นไปตามข้อจำกัดในการตรวจสอบ; กล่าวคือ, -1 ไม่ได้มากกว่า 0.

คำสั่งต่อไปนี้ใช้ได้:

```
INSERT INTO T1 VALUES (1, 1, 1)
```

แต่เมื่อจะแทรกแถว, คำสั่งต่อไปนี้จะใช้ไม่ได้:

```
ALTER TABLE T1 ADD CONSTRAINT C1 CHECK (COL1=1 AND COL1<COL2)
```

คำสั่ง ALTER TABLE พยายามที่จะเพิ่มข้อจำกัดในการตรวจสอบข้อที่สองซึ่งจำกัดค่าที่ใช้ได้ใน COL1 ไว้ที่ 1 และยังบังคับว่าค่าใน COL2 ต้องมากกว่า 1. ข้อจำกัดนี้ใช้ไม่ได้เนื่องจากส่วนที่สองของข้อจำกัด ไม่ตรงตามข้อมูลที่มีอยู่ (ค่า '1' ใน COL2 ไม่น้อยกว่าค่า '1' ใน COL1).

สิ่งอ้างอิงที่เกี่ยวข้อง

ALTER TABLE

CREATE TABLE

ฟังก์ชันการบันทึกและเรียกคืน

ฟังก์ชันการบันทึกและเรียกคืนของ i5/OS ใช้ ในการบันทึก SQL อ็อบเจ็กต์ ไปยังดิสก์ (save file) หรือ ไปยังสื่อบันทึกภายนอก

เวอร์ชันที่ถูกบันทึกสามารถถูกเรียกคืนมาไว้ในระบบปฏิบัติการ i5/OS ใดๆ ในภายหลัง ฟังก์ชันการบันทึก/เรียกคืนจะอนุญาตให้บันทึก schema ทั้งหมด, อ็อบเจ็กต์ที่ถูกเลือก หรืออ็อบเจ็กต์ที่ถูกเปลี่ยนแปลงตั้งแต่วันที่และเวลาที่กำหนดไว้ทั้งหมด ข้อมูลทั้งหมดที่จำเป็นในการเรียกคืนของ อ็อบเจ็กต์หนึ่งๆ ไปยังสถานะก่อนหน้านี้จะถูกบันทึกไว้. ฟังก์ชันนี้สามารถใช้ในการกู้ข้อมูลคืนจากความเสียหายในตารางแต่ละส่วนโดยเรียกคืนข้อมูลของตารางเวอร์ชันก่อนหน้านี้หรือ schema ทั้งหมด

โปรแกรมหรือเซอร์วิสโปรแกรมที่ถูกสร้างสำหรับ SQL โพรซีเดอร์, สำหรับฟังก์ชัน SQL , หรือสำหรับฟังก์ชันที่เป็นซอร์สที่ถูกเรียกคืนมา, จะถูกเพิ่มเข้าไปโดยอัตโนมัติในแค็ตตาล็อกของ SYSROUTINES และ SYSPARMS, ถ้าไม่มีโพรซีเดอร์หรือฟังก์ชันที่มีลายมือหรือชื่อโปรแกรมเดียวกันนั้นอยู่ก่อน. โปรแกรม SQL ที่ถูกสร้างขึ้นใน QSYS จะไม่ถูกสร้างให้เป็นโพรซีเดอร์ของ SQL เมื่อถูกเรียกคืน. นอกจากนั้น, โปรแกรมภายนอกหรือเซอร์วิสโปรแกรมที่ถูกอ้างอิงในคำสั่ง CREATE PROCEDURE หรือ CREATE FUNCTION จะเก็บข้อมูลที่จำเป็นในการลงทะเบียนรูทีนใน SYSROUTINES. ถ้าข้อมูลปรากฏโดยมีลายมือชื่อเดียวกัน, ฟังก์ชันหรือโพรซีเดอร์จะถูกเพิ่มเข้าไปใน SYSROUTINES และ SYSPARMS เมื่อถูกเรียกคืน.

เมื่อตาราง SQL ถูกเรียกคืนมา, definition ของ SQL ทริกเกอร์ที่ถูกกำหนดสำหรับตารางนั้นก็จะถูกเรียกคืนมาด้วย. definition ของ SQL ทริกเกอร์จะถูกเพิ่มเข้าไปโดยอัตโนมัติในแค็ตตาล็อกของ SYSTRIGGERS, SYSTRIGDEP, SYSTRIGCOL, และ SYSTRIGUPD. อ็อบเจ็กต์ของโปรแกรมที่ถูกสร้างขึ้นจากคำสั่ง SQL CREATE TRIGGER จะต้องถูกบันทึกและเรียกคืนเมื่อตาราง SQL ถูกบันทึกและเรียกคืน. การบันทึกและการเรียกคืนอ็อบเจ็กต์ของโปรแกรมไม่ได้ถูกทำให้เป็นอัตโนมัติโดยตัวจัดการฐานข้อมูล. ควรทบทวนข้อควรระวังสำหรับทริกเกอร์แบบอ้างอิงถึงตัวเองเมื่อมีการเรียกคืนตาราง SQL ไปยังไลบรารีใหม่.

เมื่ออ็อบเจ็กต์ *SQLUDT ถูกเรียกคืนสำหรับประเภทที่ผู้ใช้กำหนด, ประเภทดังกล่าวจะถูกเพิ่มเข้าโดยอัตโนมัติในแค็ตตาล็อก SYSTYPES. ฟังก์ชันที่เหมาะสมสำหรับการแปลงประเภท ที่ผู้ใช้กำหนดและประเภทต้นฉบับจำเป็นต้องถูกสร้างขึ้นด้วย, ตราบใดที่ไม่ปรากฏชนิดและฟังก์ชันอยู่แล้ว.

เมื่อค่า *DTAARA สำหรับลำดับนั้นถูกกู้คืน, ลำดับนั้นจะถูกเพิ่มเข้าไปในแค็ตตาล็อก SYSSEQUENCES อย่างอัตโนมัติ. ถ้าการอัพเดตแค็ตตาล็อกไม่เป็นผลสำเร็จ, จะมีการดัดแปลง *DTAARA เพื่อไม่ให้นำมาใช้เป็นลำดับ และจะมีข้อความ SQL9020 ปรากฏอยู่ในบันทึกการใช้งาน.

โปรแกรม SQL แบบกระจายหรือแพ็คเก็จ SQL แบบเชื่อมโยงสามารถบันทึกและเรียกคืนให้กับระบบกี่ระบบก็ได้. ลักษณะนี้จะอนุญาตให้สำเนาโปรแกรมของโปรแกรม SQL เท่าไรก็ได้จากระบบต่างๆ กันสามารถเข้าถึงแพ็คเก็จ SQL เดียวกันบนแอ็พพลิเคชันเซิร์ฟเวอร์เดียวกัน. ทำให้โปรแกรม SQL แบบกระจาย สามารถเชื่อมต่อกับแอ็พพลิเคชันเซิร์ฟเวอร์จำนวนเท่าใดก็ได้ที่เรียกคืนแพ็คเก็จ SQL มาแล้ว (สามารถใช้ CRTSQLPKG ได้เช่นกัน). แพ็คเก็จ SQL ไม่สามารถเรียกคืนไปยังไลบรารีที่ต่างกันได้.

หมายเหตุ: การเรียกคืนแบบแผนหนึ่งไปยังไลบรารีที่มีอยู่หรือไปยังแบบแผนที่มีชื่อต่างกันจะไม่ทำการเรียกคืนเจอร์นัล, journal receiver, หรือ IDDU dictionary (ถ้ามีปรากฏอยู่). ถ้าแบบแผนถูกเรียกคืนมายังแบบแผนที่มีชื่อต่างกัน, มุมมองของแค็ตตาล็อกในแบบแผนนั้นจะแสดงให้เห็นเพียงอ็อบเจ็กต์ในแบบแผนเก่าเท่านั้น. อย่างไรก็ตาม, มุมมองของแค็ตตาล็อกใน QSYS2, จะแสดงอ็อบเจ็กต์ทั้งหมดอย่างเหมาะสม.

การต้านทานความเสียหาย

ฐานข้อมูล DB2 for i5/OS มีวิธีลดหรือจำกัดความเสียหายที่เกิดจากความผิดพลาดของดิสก์ได้หลายวิธีด้วยกัน

ตัวอย่างเช่น การทำ mirror, checksum, และการทำ RAID ดิสก์เพื่อลดโอกาสเกิดปัญหาเกี่ยวกับดิสก์ ฟังก์ชัน DB2 for i5/OS มีค่าความต้านทานความเสียหายที่เกิดจากความผิดพลาดของดิสก์หรือความผิดพลาดของระบบได้ในระดับหนึ่ง.

ปฏิบัติการ DROP มักประสบความสำเร็จอยู่เสมอ, โดยไม่จำเป็นต้องคำนึงถึงความเสียหายใดๆ . นี่เป็นการตรวจสอบให้แน่ใจว่าความเสียหายที่เกิดขึ้น, อย่างน้อยตาราง, มุมมอง, แพ็กเก็จ SQL, ดรรชนี, โพรซีเดอร์, ฟังก์ชัน, หรือ ประเภทที่แตกต่างกันจะลบออกได้และเรียกกลับคืนมาหรือสร้างใหม่ได้อีกครั้ง.

ในกรณีที่ความผิดพลาดของดิสก์ทำความเสียหายให้กับส่วนเล็กๆ ของ แถวข้อมูลในตาราง ตัวจัดการฐานข้อมูลของ DB2 for i5/OS จะ อนุญาตให้อ่านแถวเหล่านั้นได้

Index recovery

ฐานข้อมูล DB2 for i5/OS สนับสนุน หลายๆ ฟังก์ชันที่ใช้ในการกู้คืนดรรชนี

- ระบบจัดการการปกป้องดรรชนี

คำสั่ง CL ที่เป็น Edit Recovery for Access Paths (EDTRCYAP) อนุญาตให้ผู้ใช้สามารถออกคำสั่งให้ DB2 for i5/OS รับประกัน ว่าเมื่อเกิดความผิดพลาดของระบบหรือไฟฟ้า ระยะเวลาที่ใช้ในการกู้คืนดรรชนี ทั้งหมดในระบบจะต้องต่ำกว่าเวลาที่ระบุไว้ ระบบจะทำการบันทึกข้อมูลที่เพียงพอลงในเจอร์นัลของระบบโดยอัตโนมัติเพื่อจำกัดเวลาในการกู้คืนให้อยู่ในจำนวนที่ระบุไว้.

- การทำเจอร์นัลดรรชนี

  DB2 for i5/OS สนับสนุนฟังก์ชันการทำเจอร์นัลดรรชนีที่ทำให้คุณไม่ต้องสร้างดรรชนีทั้งหมดใหม่เนื่องจากกระแสไฟฟ้าขัดข้องหรือระบบขัดข้อง ถ้าดรรชนีถูกบันทึก, การสนับสนุนฐานข้อมูลของระบบจะตรวจสอบให้แน่ใจว่าดรรชนีนั้นเชื่อมโยงกับข้อมูลในตารางโดยไม่ต้องสร้างใหม่จากจุดเริ่มต้น (scratch) ดรรชนี SQL จะไม่มีการทำเจอร์นัลโดยอัตโนมัติ อย่างไรก็ตาม, คุณสามารถ, ใช้คำสั่ง CL ที่เป็น Start Journal Access Path (STRJRNAP) ในการทำเจอร์นัลดรรชนีใดๆ ที่ถูกสร้างขึ้นโดย DB2 for i5/OS.

- Index rebuild

  ดรรชนีทั้งหมดในระบบมีอ็อพชันในการดูแลรักษาซึ่งจะระบุว่าจะมีการรักษาดรรชนีเมื่อใดบ้าง. ดรรชนีของ SQL จะถูกสร้างด้วยแอ็ตทริบิวต์ในการรักษา *IMMED.

  ในกรณีที่ไฟฟ้าขัดข้องหรือระบบขัดข้องแบบไม่ปกติ ถ้าดรรชนีไม่ได้ถูกปกป้องไว้โดยเทคนิคใดที่อธิบายมาก่อนหน้านี้ ดรรชนีเหล่านั้นในขั้นตอนของการเปลี่ยนแปลงอาจจำเป็นต้องถูกสร้างใหม่โดยตัวจัดการฐานข้อมูลเพื่อให้ถูกต้องตรงกับข้อมูลจริง ดรรชนีทั้งหมดในระบบจะมีอ็อพชันการกู้คืนซึ่งจะระบุว่าดรรชนีจะถูกสร้างขึ้นใหม่ถ้าจำเป็น. ดรรชนี SQL ทั้งหมดที่มีแอ็ตทริบิวต์ UNIQUE ถูกสร้างขึ้นด้วยแอ็ตทริบิวต์การกู้คืนของ *IPL (หมายความว่า ดรรชนี้เหล่านี้ถูกสร้างขึ้นใหม่ ก่อนที่ระบบปฏิบัติการ i5/OS จะเริ่มทำงาน) ดรรชนี SQL อื่นๆ ทั้งหมดถูกสร้างขึ้นอ็อพชันด้วยตัวเลือกการกู้คืน *AFTIPL (หมายความว่าหลังจากระบบปฏิบัติการเริ่มทำงาน ดรรชนีจะถูกสร้างขึ้นใหม่ในเวลาที่ต่างกัน) ในระหว่าง IPL โอเปอเรเตอร์จะเห็นจอแสดงผลแสดงดรรชนีที่จำเป็นต้องถูกสร้างใหม่และอ็อพชันการกู้คืน โอเปอเรเตอร์สามารถทำการแทนทับค่าเดิมของอ็อพชันการกู้คืนได้.

- การบันทึกและการเรียกคืนดรรชนี

  ฟังก์ชันบันทึก/เรียกคืน อนุญาตให้บันทึกดรรชนีได้เมื่อตารางถูกบันทึกโดยการใช้ ACCPTH(*YES) ใน Save Object (SAVOBJ) หรือ Save Library (SAVLIB) ที่เป็นคำสั่ง CL . ในกรณีที่มีการเรียกคืนเมื่อดรรชนีถูกบันทึกไว้ด้วย, ไม่มีความจำเป็นที่จะต้องสร้างดรรชนีเหล่านั้นขึ้นใหม่. ดรรชนีใดๆ ที่ไม่ได้ถูกบันทึกไว้และเรียกคืนก่อนหน้านี้จะถูกสร้างขึ้นใหม่โดยอัตโนมัติในเวลาที่ต่างกันโดยตัวจัดการฐานข้อมูล.

ความสมบูรณ์ของแคตาล็อก

เพื่อให้แน่ใจว่าข้อมูลในแคตาล็อกมีความถูกต้อง อยู่เสมอ ฐานข้อมูล DB2 for i5/OS จึง ป้องกันไม่ให้ผู้ใช้เปลี่ยนแปลงข้อมูลในแคตาล็อกอย่างชัดแจ้ง และ คงข้อมูลเอาไว้โดยนัย เมื่ออ็อบเจ็กต์ SQL ที่ระบุไว้ในแคตาล็อก ถูกเปลี่ยน

ความสมบูรณ์ของแคตาล็อกจะถูกรักษาไว้ไม่ว่าอ็อบเจ็กต์ในแบบแผนจะถูกเปลี่ยนโดยคำสั่ง SQL, คำสั่ง i5/OS CL, คำสั่ง CL ในสภาวะแวดล้อม System/38™, ฟังก์ชันในสภาวะแวดล้อม System/36 หรือผลิตภัณฑ์อื่นๆ หรือยูทิลิตีอื่นๆ บนแพล็ตฟอร์ม System i ตัวอย่างเช่น คุณสามารถลบตารางได้ด้วยการรันคำสั่ง SQL DROP, ออกคำสั่ง i5/OS Delete File (DLTF) CL, ออกคำสั่ง System/38 Delete File (DLTF) CL หรือใช้อ็อพชัน 4 บนจอแสดงผล WRKF หรือ WRKOBJ โดยไม่จำเป็นต้องคำนึงถึงอินเตอร์เฟสที่ใช้ในการลบตารางออก ตัวจัดการฐานข้อมูลจะทิ้งรายละเอียดของตารางออกจากแคตาล็อกเมื่อตารางถูกลบ ตารางต่อไปนี้แสดงรายการฟังก์ชันต่างๆ และผลกระทบที่เกี่ยวข้องที่เกิดขึ้นกับ แคตาล็อก

ตารางที่ 37. ผลกระทบของฟังก์ชันต่างๆ กับแคตาล็อก

ฟังก์ชัน	ผลกระทบกับแคตาล็อก
เพิ่มข้อจำกัดเข้าไปในตาราง	ข้อมูลถูกเพิ่มเข้าไปในแคตาล็อก

ตารางที่ 37. ผลกระทบของฟังก์ชันต่างๆ กับแคตาล็อก (ต่อ)

ฟังก์ชัน	ผลกระทบกับแคตาล็อก
ลบข้อจำกัดออกจากตาราง	ข้อมูลที่เกี่ยวข้องจะถูกลบออกจากแคตาล็อก
สร้างอ็อบเจ็กต์ขึ้นในแบบแผน	ข้อมูลถูกเพิ่มเข้าไปในแคตาล็อก
ลบอ็อบเจ็กต์ออกจากแบบแผน	ข้อมูลที่เกี่ยวข้องจะถูกลบออกจากแคตาล็อก
เรียกคืนอ็อบเจ็กต์เข้ามาในแบบแผน	ข้อมูลถูกเพิ่มเข้าไปในแคตาล็อก
การเปลี่ยนแปลงของข้อสังเกตแบบยาวของอ็อบเจ็กต์	ข้อสังเกตถูกอัพเดตในแคตาล็อก
การเปลี่ยนแปลงเลเบล (ข้อความ) ของอ็อบเจ็กต์	เลเบลถูกอัพเดตในแคตาล็อก
การเปลี่ยนแปลงเจ้าของอ็อบเจ็กต์	เจ้าของถูกอัพเดตในแคตาล็อก
การลบอ็อบเจ็กต์ออกจากแบบแผน	ข้อมูลที่เกี่ยวข้องจะถูกลบออกจากแคตาล็อก
การย้ายอ็อบเจ็กต์เข้าไปในแบบแผน	ข้อมูลถูกเพิ่มเข้าไปในแคตาล็อก
การเปลี่ยนชื่ออ็อบเจ็กต์	ชื่ออ็อบเจ็กต์ถูกอัพเดตในแคตาล็อก

User auxiliary storage pool

คุณสามารถสร้าง schema ใน user auxiliary storage pool (ASP) โดยใช้ ASP clause ในคำสั่ง CREATE SCHEMA

คำสั่ง Create Library (CRTLIB) สามารถใช้ในการสร้าง ไลบรารีใน ASP ของผู้ใช้ได้เช่นกัน ไลบรารีดังกล่าวสามารถใช้ในการรับตาราง SQL, มุมมอง, และดรรชนีได้.

หลักการที่เกี่ยวข้อง

การสำรองข้อมูลและการกู้คืน

การจัดการดิสก์

Independent auxiliary storage pool

ดิสก์พูลแบบอิสระถูกนำมาใช้ในการติดตั้งฐานข้อมูลของผู้ใช้ บนระบบ

ดิสก์พูลอิสระแบ่งเป็น 3 ประเภทด้วยกันได้แก่ : primary, secondary, และ user-defined file system (UDFS). ดิสก์พูลหลักแบบอิสระถูก นำมาใช้ในการติดตั้งฐานข้อมูล

คุณสามารถทำงานกับฐานข้อมูลหลายชุด ระบบ มีฐานข้อมูลของระบบ (มักเรียกว่า SYSBAS) และความสามารถในการ ทำงานกับฐานข้อมูลของผู้ใช้ได้หลายฐานข้อมูล ฐานข้อมูลของผู้ใช้จะถูกจัดการด้วย ระบบ โดยใช้ดิสก์พูลอิสระ ซึ่งถูกติดตั้งไว้ในฟังก์ชัน disk management ของ System i Navigator หลังจากที่ดิสก์พูลอิสระถูกติดตั้ง มันจะปรากฏเป็นฐานข้อมูลอีกอันหนึ่งภายใต้ โฟลเดอร์ Databases ของ System i Navigator

หลักการที่เกี่ยวข้อง

การจัดการดิสก์

รูทีน

รูทีน คือโค้ดหรือโปรแกรมที่คุณสามารถเรียกใช้งานได้.

สตอร์โพรซีเดอร์

โพรซีเดอร์ (ซึ่งมักเรียกว่า โพรซีเดอร์ที่เก็บไว้) คือโปรแกรมที่สามารถเรียกขึ้นมาเพื่อปฏิบัติงาน โพรซีเดอร์อาจประกอบด้วยคำสั่งภาษาโฮสต์และคำสั่ง SQL โพรซีเดอร์ใน SQL มีข้อดีเหมือนกับโพรซีเดอร์ในภาษาโฮสต์

DB2 สนับสนุนโพรซีเดอร์ที่เก็บไว้ เพื่อให้แอ็พพลิเคชัน SQL สามารถ กำหนด และเรียกโพรซีเดอร์ผ่านทางคำสั่ง SQL โพรซีเดอร์ ที่เก็บไว้สามารถนำมาใช้ทั้งในแอ็พพลิเคชัน DB2 แบบกระจายและไม่กระจาย ข้อดีข้อหนึ่งของการใช้โพรซีเดอร์ที่เก็บไว้คือสำหรับแอ็พพลิเคชันแบบกระจายแล้ว, การใช้ข้อความ CALL หนึ่งข้อความบน application requester หรือไคลเอ็นต์ สามารถทำงานในปริมาณเท่าใดก็ตามบนแอ็พพลิเคชันเซิร์ฟเวอร์

คุณอาจนิยามโพรซีเดอร์ว่าเป็น SQL โพรซีเดอร์หรือโพรซีเดอร์ภายนอก. โพรซีเดอร์ภายนอกสามารถเป็นโปรแกรมภาษาชั้นสูงที่สนับสนุนใดๆ ก็ตาม (ยกเว้นโปรแกรมและโพรซีเดอร์ System/36) หรือโพรซีเดอร์ REXX โพรซีเดอร์ดังกล่าวไม่จำเป็นต้องมี ข้อความ SQL, แต่อาจมีข้อความ SQL ได้. SQL โพรซีเดอร์ถูกกำหนดไว้ทั้งหมดใน SQL, และสามารถมีข้อความ SQL ที่รวมเอา SQL control statement ไว้ได้.

การโค้ดโพรซีเดอร์ที่เก็บไว้นั้นผู้ใช้จำเป็นต้องเข้าใจดังนี้:

- บันทึก definition ของโพรซีเดอร์ด้วยข้อความ CREATE PROCEDURE
- บันทึกการเรียกโพรซีเดอร์ด้วยข้อความ CALL
- หลักการผ่านพารามิเตอร์
- วิธีการย้อนกลับสถานะที่สมบูรณ์ไปยังโปรแกรมที่เรียกโพรซีเดอร์.

คุณอาจนิยามโพรซีเดอร์ที่เก็บไว้ด้วยการใช้ข้อความ CREATE PROCEDURE. ข้อความ CREATE PROCEDURE จะเป็นการเพิ่ม definition ของโพรซีเดอร์และพารามิเตอร์ ให้กับตารางแคตาล็อก SYSROUTINES และ SYSPARMS. definition เหล่านี้สามารถเข้าไปได้โดย ข้อความ SQL CALL ใดๆ บนระบบ.

ในการสร้างโพรซีเดอร์ภายนอก หรือ SQL โพรซีเดอร์, คุณสามารถใช้คำสั่ง SQL CREATE PROCEDURE.

ส่วนต่อไปนี้เป็นการอธิบายคำสั่ง SQL ที่ใช้เพื่อกำหนดและเรียกโพรซีเดอร์ที่เก็บไว้, ข้อมูลเกี่ยวกับการผ่านพารามิเตอร์ไปยังโพรซีเดอร์ที่เก็บไว้, และตัวอย่างการใช้โพรซีเดอร์ที่เก็บไว้.

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับโพรซีเดอร์ที่เก็บไว้โปรดดู โพรซีเดอร์ที่เก็บไว้, ทริกเกอร์ และฟังก์ชันแบบผู้ใช้กำหนดเองบน DB2 Universal Database™ สำหรับ iSeries™

หลักการที่เกี่ยวข้อง

"โพรซีเดอร์ที่เก็บไว้" ในหน้า 13
*โพรซีเดอร์ที่เก็บไว้*คือ โปรแกรม ที่อาจถูกเรียกโดยใช้คำสั่ง SQL CALL

Java SQL routines

สิ่งอ้างอิงที่เกี่ยวข้อง

"ข้อควรพิจารณาสำหรับโพรซีเดอร์ของ DRDA ที่บันทึกไว้" ในหน้า 328
เซิร์ฟเวอร์ i5/OS Distributed Relational Database Architecture (DRDA) สนับสนุนการส่งคืนชุดผลลัพธ์หนึ่งชุด หรือมากกว่านั้นในโพรซีเดอร์ที่เก็บไว้

CREATE PROCEDURE

การกำหนดโพรซีเดอร์ภายนอก

คำสั่ง CREATE PROCEDURE สำหรับโพรซีเดอร์ ภายนอกที่ตั้งชื่อโพรซีเดอร์ จะกำหนดพารามิเตอร์และแอ็ตทริบิวต์ของโพรซีเดอร์ และให้ข้อมูลอื่นๆ เกี่ยวกับโพรซีเดอร์ที่ระบบใช้เมื่อ เรียกโพรซีเดอร์นั้น

พิจารณาตัวอย่างนี้:

```
CREATE PROCEDURE P1
        (INOUT PARM1 CHAR(10))
        EXTERNAL NAME MYLIB.PROC1
        LANGUAGE C
        GENERAL WITH NULLS
```

คำสั่ง CREATE PROCEDURE:

- ตั้งชื่อโพรซีเดอร์ P1
- กำหนดหนึ่งพารามิเตอร์ซึ่งถูกใช้เป็นทั้งอินพุตพารามิเตอร์และเอาต์พุตพารามิเตอร์. พารามิเตอร์คือ ฟิลด์แบบอักขระที่มีความยาวสิบตัวอักษร. สามารถกำหนดพารามิเตอร์ให้เป็นประเภท IN, OUT, หรือ INOUT. จะกำหนดประเภทพารามิเตอร์เมื่อค่าสำหรับพารามิเตอร์ ผ่านไปยังและผ่านจากโพรซีเดอร์.
- กำหนดชื่อของโปรแกรมที่สอดคล้องกับโพรซีเดอร์, คือ PROC1 ใน MYLIB. MYLIB.PROC1 คือโปรแกรมซึ่งถูกเรียกเมื่อมีการเรียกโพรซีเดอร์ด้วยคำสั่ง CALL.
- แสดงว่าโพรซีเดอร์ P1 (โปรแกรม MYLIB.PROC1) ถูกบันทึกลงใน C. ภาษาเป็นสิ่งสำคัญเนื่องจากมีผลต่อประเภทพารามิเตอร์ที่สามารถผ่านไปได้. และยังส่งผลต่อวิธีการที่พารามิเตอร์ ถูกส่งไปยังโพรซีเดอร์(ตัวอย่างเช่น, สำหรับโพรซีเดอร์ ILE C, NUL-terminator จะถูกส่งด้วยอักขระ,กราฟิก, วันที่, เวลา, และพารามิเตอร์ timestamp).
- กำหนดประเภท CALL ให้เป็น GENERAL WITH NULLS. แสดงว่าพารามิเตอร์สำหรับโพรซีเดอร์อาจมีค่า NULL อยู่, ดังนั้นจึงต้องการให้มีอากิวเมนต์เพิ่มเติมผ่านไปยังโพรซีเดอร์บนข้อความ CALL. อากิวเมนต์เพิ่มเติมคืออะเรย์ของจำนวนเต็ม N ที่สั้น, โดยที่ N คือจำนวนพารามิเตอร์ ซึ่งประกาศในคำสั่ง CREATE PROCEDURE. ในตัวอย่างนี้, อะเรย์มีเพียงหนึ่งองค์ประกอบ เนื่องจากมีเพียงพารามิเตอร์เท่านั้น.

เป็นเรื่องสำคัญที่ต้องสังเกตว่าไม่จำเป็นต้องกำหนดโพรซีเดอร์เพื่อเรียกใช้งาน. อย่างไรก็ตาม, หากไม่พบ definition ของโพรซีเดอร์, จาก CREATE PROCEDURE ก่อนหน้านี้หรือจาก DECLARE PROCEDURE ในโปรแกรมนี้, แสดงว่ามีการตั้งข้อบังคับและสมมุติฐานบางอย่างเมื่อเรียกโพรซีเดอร์บนคำสั่ง CALL. ตัวอย่างเช่น, อากิวเมนต์ ตัวบ่งชี้ NULL จะไม่สามารถผ่านไปได้.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การใช้คำสั่ง CALL แบบฝังโดยไม่มี definition ของโพรซีเดอร์อยู่" ในหน้า 154
คำสั่ง CALL แบบ static ที่ไม่มีคำสั่ง CREATE PROCEDURE ที่ตรงกันถูกประมวลผลด้วยกฎต่อไปนี้

การกำหนด SQL โพรซีเดอร์

คำสั่ง CREATE PROCEDURE สำหรับ SQL โพรซีเดอร์ จะตั้งชื่อโพรซีเดอร์, กำหนดพารามิเตอร์และแอ็ตทริบิวต์ และให้ข้อมูลอื่นๆ เกี่ยวกับโพรซีเดอร์ที่ใช้ เมื่อเรียกใช้โพรซีเดอร์ และกำหนดโครงโพรซีเดอร์

*โครงโพรซีเดอร์*คือส่วนที่สามารถเรียกทำงานได้ของโพรซีเดอร์ และเป็นคำสั่ง SQL แบบเดี่ยว.

พิจารณาตัวอย่างง่ายๆ ต่อไปนี้ ซึ่งใช้อินพุตเป็นหมายเลขพนักงาน และอัพเดตเงินเดือนพนักงาน:

```
CREATE PROCEDURE UPDATE_SALARY_1
  (IN EMPLOYEE_NUMBER CHAR(10),
   IN RATE DECIMAL(6,2))
   LANGUAGE SQL MODIFIES SQL DATA
   UPDATE CORPDATA.EMPLOYEE
     SET SALARY = SALARY * RATE
     WHERE EMPNO = EMPLOYEE_NUMBER
```

คำสั่ง CREATE PROCEDURE นี้:

- ตั้งชื่อโพรซีเดอร์เป็น UPDATE_SALARY_1.
- กำหนดพารามิเตอร์ EMPLOYEE_NUMBER ซึ่งเป็นอินพุตพารามิเตอร์และเป็นประเภทข้อมูลอักขระ ความยาว 6 ตัวอักษรและพารามิเตอร์ RATE ซึ่งเป็นอินพุตพารามิเตอร์และจัดอยู่ในประเภทข้อมูลทศนิยม.
- แสดงว่าโพรซีเดอร์นี้คือ SQL โพรซีเดอร์ซึ่งแก้ไขข้อมูล SQL.
- กำหนดโครงโพรซีเดอร์เป็นข้อความ UPDATE แบบเดี่ยว. เมื่อมีการเรียกโพรซีเดอร์, ข้อความ UPDATE จะถูกเรียกขึ้นมาด้วยการใช้ค่าที่ส่งต่อสำหรับ EMPLOYEE_NUMBER และ RATE.

แทนที่จะใช้คำสั่ง UPDATE แบบเดี่ยว สามารถเพิ่มตรรกะนี้ให้กับ SQL โพรซีเดอร์ด้วยการใช้ SQL control statement SQL control statement ประกอบด้วยคำสั่งต่อไปนี้:

- คำสั่งการกำหนด
- คำสั่ง CALL
- คำสั่ง CASE
- คำสั่งผสม
- คำสั่ง FOR
- คำสั่ง GET DIAGNOSTICS
- คำสั่ง GOTO
- คำสั่ง IF
- คำสั่ง ITERATE
- คำสั่ง LEAVE
- คำสั่ง LOOP
- คำสั่ง REPEAT
- คำสั่ง RESIGNAL
- คำสั่ง RETURN
- คำสั่ง SIGNAL
- คำสั่ง WHILE

ใช้ตัวอย่างต่อไปนี้เป็นอินพุตของหมายเลขพนักงานและการจัดอันดับซึ่งได้รับจากการประเมิน ล่าสุด. โพรซีเดอร์ใช้คำสั่ง CASE เพื่อกำหนดการเพิ่มและโบนัสสำหรับอัพเดต.

```
CREATE PROCEDURE UPDATE_SALARY_2
  (IN EMPLOYEE_NUMBER CHAR(6),
  IN RATING INT)
  LANGUAGE SQL MODIFIES SQL DATA
     CASE RATING
        WHEN 1 THEN
          UPDATE CORPDATA.EMPLOYEE
            SET SALARY = SALARY * 1.10,
             BONUS = 1000
            WHERE EMPNO = EMPLOYEE_NUMBER;
        WHEN 2 THEN
          UPDATE CORPDATA.EMPLOYEE
            SET SALARY = SALARY * 1.05,
            BONUS = 500
            WHERE EMPNO = EMPLOYEE_NUMBER;
       ELSE
           UPDATE CORPDATA.EMPLOYEE
             SET SALARY = SALARY * 1.03,
             BONUS = 0
             WHERE EMPNO = EMPLOYEE_NUMBER;
      END CASE
```

คำสั่ง CREATE PROCEDURE นี้:

- ตั้งชื่อโพรซีเดอร์เป็น UPDATE_SALARY_2.
- กำหนดพารามิเตอร์ EMPLOYEE_NUMBER ซึ่งเป็นอินพุตพารามิเตอร์และเป็นประเภทข้อมูลอักขระ ความยาว 6 ตัวอักษรและพารามิเตอร์ RATING ซึ่งเป็นอินพุตพารามิเตอร์และจัดอยู่ในประเภทข้อมูลจำนวนเต็ม.
- แสดงว่าโพรซีเดอร์นี้คือ SQL โพรซีเดอร์ซึ่งแก้ไขข้อมูล SQL.
- กำหนดโครงโพรซีเดอร์. เมื่อมีการเรียกโพรซีเดอร์นี้, อินพุตพารามิเตอร์ RATING จะถูกตรวจสอบและข้อความอัพเดตที่เหมาะสมจะถูกเรียกใช้งาน.

สามารถเพิ่มข้อความจำนวนมากให้กับโครงโพรซีเดอร์ด้วยการเพิ่มข้อความผสม. ภายในข้อความผสม, สามารถระบุจำนวนคำสั่ง SQL ใดๆ ก็ได้. นอกจากนี้, ยังสามารถประกาศตัวแปร SQL, เคอร์เซอร์, และ handler ได้.

ใช้ตัวอย่างต่อไปนี้เป็นอินพุตหมายเลขแผนก. ซึ่งจะรายงานเงินเดือนโดยรวมของพนักงานทั้งหมดในแผนกนั้น รวมทั้งจำนวนพนักงานในแผนกที่ได้โบนัส.

```
CREATE PROCEDURE RETURN_DEPT_SALARY
        (IN DEPT_NUMBER CHAR(3),
         OUT DEPT_SALARY DECIMAL(15,2),
         OUT DEPT_BONUS_CNT INT)
        LANGUAGE SQL READS SQL DATA
        P1: BEGIN
         DECLARE EMPLOYEE_SALARY DECIMAL(9,2);
         DECLARE EMPLOYEE_BONUS DECIMAL(9,2);
         DECLARE TOTAL_SALARY DECIMAL(15,2)DEFAULT 0;
         DECLARE BONUS_CNT INT DEFAULT 0;
         DECLARE END_TABLE INT DEFAULT 0;
         DECLARE C1 CURSOR FOR
          SELECT SALARY, BONUS FROM CORPDATA.EMPLOYEE
           WHERE WORKDEPT = DEPT_NUMBER;
         DECLARE CONTINUE HANDLER FOR NOT FOUND
```

```
    SET END_TABLE = 1;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
    SET DEPT_SALARY = NULL;
  OPEN C1;
  FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
  WHILE END_TABLE = 0 DO
     SET TOTAL_SALARY = TOTAL_SALARY + EMPLOYEE_SALARY + EMPLOYEE_BONUS;
     IF EMPLOYEE_BONUS > 0 THEN
       SET BONUS_CNT = BONUS_CNT + 1;
     END IF;
     FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
   END WHILE;
   CLOSE C1;
   SET DEPT_SALARY = TOTAL_SALARY;
   SET DEPT_BONUS_CNT = BONUS_CNT;
 END P1
```

คำสั่ง CREATE PROCEDURE นี้:

- ตั้งชื่อโพรซีเดอร์เป็น RETURN_DEPT_SALARY.
- กำหนดพารามิเตอร์ DEPT_NUMBER ซึ่งเป็นอินพุตพารามิเตอร์และเป็นประเภทข้อมูลอักขระ ความยาว 3 ตัวอักษร, พารามิเตอร์ DEPT_SALARY ซึ่งเป็นเอาต์พุตพารามิเตอร์และจัดอยู่ในประเภทข้อมูล ทศนิยม, และพารามิเตอร์ DEPT_BONUS_CNT ซึ่งเป็นเอาต์พุตพารามิเตอร์และเป็นประเภทข้อมูลจำนวนเต็ม.
- แสดงว่าโพรซีเดอร์นี้คือ SQL โพรซีเดอร์ซึ่งอ่านข้อมูล SQL
- กำหนดโครงโพรซีเดอร์.
  - ประกาศให้ SQL ผันแปรกับ EMPLOYEE_SALARY และ TOTAL_SALARY เป็นฟิลด์แบบทศนิยม.
  - ประกาศให้ SQL ผันแปรกับ BONUS_CNT และ END_TABLE ซึ่งเป็นจำนวนเต็มและถูก initialize เป็น 0.
  - ประกาศ C1 ซึ่งเลือกคอลัมน์จากตารางพนักงาน.
  - ประกาศ handler แบบต่อเนื่องสำหรับ NOT FOUND, ซึ่ง, เมื่อถูกเรียกจะกำหนดค่าผันแปร END_TABLE เป็น 1. handler นี้จะถูกเรียกเมื่อ FETCH ไม่มีแถวเหลืออยู่สำหรับส่งคืน. หาก handler นี้ถูกเรียกขึ้นมา, SQLCODE และ SQLSTATE จะถูก initialize ใหม่เป็น 0.
  - ประกาศ exit handler สำหรับ SQLEXCEPTION. หากถูกเรียก, DEPT_SALARY จะถูกตั้งให้เป็น NULL และการประมวลผลข้อความผสมถูกทำให้จบ. handler นี้จะถูกเรียกเมื่อมีข้อผิดพลาดเกิดขึ้น, กล่าวคือ, คลาส SQLSTATE ไม่ใช่ '00', '01' หรือ '02'. เนื่องจากตัวบ่งชี้จะถูกส่งไปยัง SQL โพรซีเดอร์เสมอ, ค่าตัวบ่งชี้สำหรับ DEPT_SALARY จึงเท่ากับ -1 เมื่อโพรซีเดอร์ส่งคืนค่า. หาก handler นี้ถูกเรียกขึ้นมา, SQLCODE และ SQLSTATE จะถูก initializ ใหม่เป็น 0.

    หากไม่มีการระบุ handler สำหรับ SQLEXCEPTION และเกิดข้อผิดพลาดขึ้นซึ่งไม่ได้รับการจัดการในอีก handler หนึ่ง, การใช้ข้อความผสมจะถูกยกเลิกและข้อผิดพลาด จะถูกส่งคืนใน SQLCA. เช่นเดียวกับตัวบ่งชี้, SQLCA จะถูกส่งคืนจาก SQL โพรซีเดอร์เสมอ.
  - ประกอบด้วย OPEN, FETCH, และ CLOSE ของเคอร์เซอร์ C1. หากไม่มีการระบุ CLOSE ของเคอร์เซอร์, เคอร์เซอร์นั้นจะถูกปิดในตอนท้ายของคำสั่งผสม เนื่องจากไม่มีการระบุ SET RESULT SETS ในคำสั่ง CREATE PROCEDURE.
  - ประกอบด้วยข้อความ WHILE ซึ่งจะวนซ้ำจนกว่าเร็กคอร์ดล่าสุดจะถูกดึงข้อมูลออก. สำหรับแต่ละแถวที่ถูกเรียกออกมา, TOTAL_SALARY จะเพิ่มขึ้นและ, หากโบนัสของพนักงานมากกว่า 0, BONUS_CNT ก็จะเพิ่มขึ้นเช่นกัน.
  - ส่งคืน DEPT_SALARY และ DEPT_BONUS_CNT เป็นเอาต์พุตพารามิเตอร์.

คำสั่งผสมสามารถทำเป็นแบบ atomic เพื่อที่หากเกิดข้อผิดพลาดแบบไม่คาดคิดขึ้น, คำสั่งที่อยู่ภายในคำสั่งแบบ atomic จะสามารถย้อนกลับได้. คำสั่งผสมแบบ atomic ถูกนำมาใช้ด้วยการใช้ SAVEPOINTS. หากคำสั่งผสมสำเร็จ, transaction จะถูก commit.

ใช้ตัวอย่างต่อไปนี้เป็นอินพุตหมายเลขแผนก. ซึ่งเป็นการรับประกันว่าตาราง EMPLOYEE_BONUS ยังคงมีอยู่, และใส่ชื่อของพนักงานทั้งหมดในแผนกที่ได้โบนัส. โพรซีเดอร์จะส่งคืนการนับโดยรวมของพนักงานทั้งหมดที่ได้โบนัส.

```
CREATE PROCEDURE CREATE_BONUS_TABLE
         (IN DEPT_NUMBER CHAR(3),
          INOUT CNT INT)
          LANGUAGE SQL MODIFIES SQL DATA
           CS1: BEGIN ATOMIC
           DECLARE NAME VARCHAR(30) DEFAULT NULL;
           DECLARE CONTINUE HANDLER FOR SQLSTATE '42710'
              SELECT COUNT(*) INTO CNT
              FROM DATALIB.EMPLOYEE_BONUS;
           DECLARE CONTINUE HANDLER FOR SQLSTATE '23505'
              SET  CNT = CNT - 1;
           DECLARE UNDO HANDLER FOR SQLEXCEPTION
              SET CNT = NULL;
           IF DEPT_NUMBER IS NOT NULL THEN
             CREATE TABLE DATALIB.EMPLOYEE_BONUS
                  (FULLNAME VARCHAR(30),
                   BONUS DECIMAL(10,2),
                  PRIMARY KEY (FULLNAME));
           FOR_1:FOR V1 AS C1 CURSOR FOR
               SELECT FIRSTNME, MIDINIT, LASTNAME, BONUS
                  FROM CORPDATA.EMPLOYEE
                  WHERE WORKDEPT = CREATE_BONUS_TABLE.DEPT_NUMBER
               DO
               IF BONUS > 0 THEN
                 SET NAME = FIRSTNME CONCAT ' ' CONCAT
                            MIDINIT  CONCAT ' 'CONCAT LASTNAME;
                 INSERT INTO DATALIB.EMPLOYEE_BONUS
                    VALUES(CS1.NAME, FOR_1.BONUS);
                 SET CNT = CNT + 1;
               END IF;
             END FOR FOR_1;
           END IF;
           END CS1
```

คำสั่ง CREATE PROCEDURE นี้:

- ตั้งชื่อโพรซีเดอร์เป็น CREATE_BONUS_TABLE.
- กำหนดพารามิเตอร์ DEPT_NUMBER ซึ่งเป็นอินพุตพารามิเตอร์และจัดอยู่ในประเภทข้อมูลอักขระ ความยาว 3 ตัวอักษร และพารามิเตอร์ CNT ซึ่งเป็นพารามิเตอร์อินพุต/เอาต์พุต และเป็นประเภทข้อมูลจำนวนเต็ม.
- แสดงว่าโพรซีเดอร์นี้คือ SQL โพรซีเดอร์ซึ่งแก้ไขข้อมูล SQL
- กำหนดโครงโพรซีเดอร์.
  - ประกาศ SQL variable NAME ให้เป็นอักขระที่เปลี่ยนแปลงได้.

- ประกาศ handler แบบต่อเนื่องสำหรับ SQLSTATE 42710, มีตารางอยู่แล้ว. หากมีตาราง EMPLOYEE_BONUS อยู่แล้ว, handler จะถูกเรียกใช้งานและเรียกจำนวนเร็กคอร์ดในตารางออกมา. SQLCODE และ SQLSTATE ถูกรีเซ็ตเป็น 0 และมีการประมวลผลอย่างต่อเนื่องด้วยข้อความ FOR.
- ประกาศ handler แบบต่อเนื่องสำหรับ SQLSTATE 23505, ทำซ้ำคีย์. ถ้าโพรซีเดอร์ พยายามใส่ชื่อซึ่งมีอยู่ในตารางอยู่แล้ว, handler จะถูกเรียกใช้งานและไปลดส่วน CNT ลง. มีการประมวลผลอย่างต่อเนื่องบนข้อความ SET ซึ่งอยู่หลังข้อความ INSERT.
- ประกาศ UNDO handler สำหรับ SQLEXCEPTION. ถ้าถูกเรียกใช้งาน, ข้อความก่อนหน้านี้จะย้อนกลับ, CNT จะถูกตั้งเป็น 0, และมีการประมวลผลต่อหลังข้อความผสม. ในกรณีนี้, เนื่องจากไม่มีข้อความตามหลังข้อความผสม,โพรซีเดอร์จึงส่งคืน.
- ใช้ข้อความ FOR เพื่อประกาศเคอร์เซอร์ C1 ให้อ่านเร็กคอร์ดจากตาราง EMPLOYEE. ภายในข้อความ FOR, ชื่อคอลัมน์จากลิสต์ที่เลือกจะถูกใช้เป็นตัวแปร SQL ซึ่งประกอบด้วยข้อมูลจากแถวที่ถูกดึงข้อมูลออก. สำหรับแต่ละแถว, ข้อมูลจากคอลัมน์ FIRSTNME, MIDINIT, และ LASTNAME จะถูกเชื่อมต่อเข้าด้วยกัน ด้วยที่ว่างในระหว่างและผลลัพธ์จะถูกใส่ไว้ใน SQL variable NAME. SQL variables NAME และ BONUS จะถูกใส่ไว้ในตาราง EMPLOYEE_BONUS. เนื่องจากต้องรู้ประเภทข้อมูลของไอเท็มลิสต์ที่เลือก เมื่อมีการสร้างโพรซีเดอร์, ตารางที่ระบุในคำสั่ง FOR จะต้องมีอยู่ เมื่อสร้างโพรซีเดอร์.

  ชื่อตัวแปร SQL สามารถทำให้ถูกกฎเกณฑ์ได้ด้วยการใช้ชื่อเลเบลของข้อความ FOR หรือข้อความผสมที่ชื่อตัวแปรนั้นระบุอยู่. ในตัวอย่าง, FOR_1.BONUS หมายถึงตัวแปร SQL ที่มีค่าของคอลัมน์ BONUS สำหรับแต่ละแถวที่เลือก. CS1.NAME คือตัวแปร NAME ซึ่งกำหนดไว้ในข้อความผสมขึ้นต้นด้วยเลเบล CS1. นอกจากนี้ ชื่อพารามิเตอร์สามารถทำให้ถูกกฎเกณฑ์ได้ด้วยการใช้ชื่อโพรซีเดอร์. CREATE_BONUS_TABLE.DEPT_NUMBER คือพารามิเตอร์ DEPT_NUMBER สำหรับโพรซีเดอร์ CREATE_BONUS_TABLE. หากมีการใช้ชื่อตัวแปร SQL ที่ไม่ถูกกฎเกณฑ์ในข้อความ SQL โดยที่มีการอนุญาตใช้ชื่อคอลัมน์เช่นกัน, และชื่อตัวแปรเหมือนกับชื่อคอลัมน์, ชื่อนั้นก็จะถูกใช้เพื่ออ้างถึงคอลัมน์.

คุณสามารถใช้ SQL แบบไดนามิกใน SQL โพรซีเดอร์. ตัวอย่างต่อไปเป็นการสร้างตาราง ซึ่งประกอบด้วยพนักงานทั้งหมดในแผนกเฉพาะ. หมายเลขแผนกจะถูกส่งต่อเป็นอินพุตไปยังโพรซีเดอร์ และถูกเชื่อมต่อเข้าด้วยกันกับชื่อตาราง.

```
CREATE PROCEDURE CREATE_DEPT_TABLE (IN P_DEPT CHAR(3))
       LANGUAGE SQL
  BEGIN
    DECLARE STMT CHAR(1000);
    DECLARE MESSAGE CHAR(20);
    DECLARE TABLE_NAME CHAR(30);
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
       SET MESSAGE = 'ok';
    SET TABLE_NAME = 'CORPDATA.DEPT_' CONCAT P_DEPT CONCAT '_T';
    SET STMT = 'DROP TABLE ' CONCAT TABLE_NAME;
    PREPARE S1 FROM STMT;
    EXECUTE S1;
       SET STMT = 'CREATE TABLE ' CONCAT TABLE_NAME CONCAT
      '( EMPNO CHAR(6) NOT NULL,
         FIRSTNME    VARCHAR(12)     NOT NULL,
         MIDINIT     CHAR(1)         NOT NULL,
         LASTNAME CHAR(15) NOT NULL,
         SALARY DECIMAL(9,2))';
    PREPARE S2 FROM STMT;
    EXECUTE S2;
    SET STMT = 'INSERT INTO ' CONCAT TABLE_NAME CONCAT
```

```
      'SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
        FROM CORPDATA.EMPLOYEE
        WHERE  WORKDEPT = ?';
    PREPARE S3 FROM STMT;
    EXECUTE S3 USING P_DEPT;
END
```

คำสั่ง CREATE PROCEDURE นี้:

- ตั้งชื่อโพรซีเดอร์เป็น CREATE_DEPT_TABLE
- กำหนดพารามิเตอร์ P_DEPT ซึ่งเป็นอินพุตพารามิเตอร์และจัดอยู่ในประเภทข้อมูล อักขระความยาว 3 ตัวอักษร.
- แสดงว่าโพรซีเดอร์นี้คือ SQL โพรซีเดอร์.
- กำหนดโครงโพรซีเดอร์.
  - ประกาศ SQL variable STMT และ SQL variable TABLE_NAME เป็นอักขระ.
  - ประกาศ CONTINUE handler. โพรซีเดอร์พยายามที่จะ DROP ตารางในกรณีที่มีอยู่แล้ว. ถ้าไม่มีตารางอยู่, EXECUTE แรกจะล้มเหลว. ด้วยการใช้ handler, การประมวลผลจะดำเนินต่อไป.
  - ตั้งตัวแปร TABLE_NAME ให้เป็น 'DEPT_' ตามด้วยอักขระที่ถูกส่งผ่านมาในพารามิเตอร์ P_DEPT, ตามด้วย '_T'.
  - ตั้งตัวแปร STMT ให้เป็นข้อความ DROP, จากนั้นให้เตรียมและเรียกใช้งานข้อความ.
  - เซ็ตตัวแปร STMT ให้เป็นข้อความ CREATE, จากนั้นให้เตรียมและเรียกใช้งานข้อความ.
  - ตั้งตัวแปร STMT ให้เป็นข้อความ INSERT, จากนั้นให้เตรียมและเรียกใช้งานข้อความ. มีการระบุเครื่องหมายพารามิเตอร์ใน clause. เมื่อเรียกใช้งานคำสั่ง, ตัวแปร P_DEPT จะถูกส่งต่อไปบน USING clause.

หากมีการเรียกโพรซีเดอร์ผ่านค่า 'D21' สำหรับแผนก, ตาราง DEPT_D21_T ก็จะถูกสร้างขึ้นและตารางจะถูก initialize ด้วยพนักงานทั้งหมดในแผนก 'D21'.

การเรียกโพรซีเดอร์ที่เก็บไว้

คำสั่ง SQL CALL เรียกโพรซีเดอร์ที่เก็บไว้.

ที่ข้อความ CALL, จะมีการระบุชื่อของโพรซีเดอร์ที่เก็บไว้และอากิวเมนต์ใดๆ. อากิวเมนต์อาจเป็นจำนวนคงที่, เรจิสเตอร์พิเศษ, หรือตัวแปรโฮสต์. โพรซีเดอร์ที่เก็บไว้ภายนอกที่ระบุในข้อความ CALL ไม่จำเป็นต้องมีข้อความ CREATE PROCEDURE ที่ตรงกัน. โปรแกรมที่สร้างขึ้นโดย SQL โพรซีเดอร์จะสามารถเรียกได้ด้วยการเรียกใช้งาน ชื่อโพรซีเดอร์ที่ระบุบนข้อความ CREATE PROCEDURE เท่านั้น.

แม้ว่าโพรซีเดอร์จะเป็นอ็อบเจ็กต์โปรแกรมระบบ, การใช้คำสั่ง CALL ในชุดคำสั่ง CL เพื่อเรียกโพรซีเดอร์จะไม่สามารถใช้ได้. คำสั่ง CALL ในชุดคำสั่ง CL ไม่ได้ใช้ definition ของโพรซีเดอร์เพื่อแม็พพารามิเตอร์อินพุตและเอาต์พุต, และไม่ได้ส่งพารามิเตอร์ต่อไปยังโปรแกรมโดยใช้รูปแบบพารามิเตอร์ของโพรซีเดอร์.

จำเป็นต้องกล่าวถึงคำสั่ง CALL ประเภทต่างๆ ต่อไปนี้ เนื่องจากฐานข้อมูล DB2 for i5/OS มีกฎที่แตกต่างกันสำหรับแต่ละประเภท:

- คำสั่ง CALL แบบ dynamic หรือที่ใส่อยู่ซึ่งมี definition ของโพรซีเดอร์อยู่
- คำสั่ง CALL ที่ใส่อยู่โดยที่ไม่มี definition ของโพรซีเดอร์อยู่
- คำสั่ง CALL แบบไดนามิกโดยที่ไม่มี CREATE PROCEDURE อยู่

หมายเหตุ:

Dynamic ในที่นี้หมายถึง:

- คำสั่ง CALL ที่ถูกเตรียม และเรียกใช้งานแบบ dynamic.
- คำสั่ง CALL ที่มีอยู่ในสภาวะแวดล้อมแบบโต้ตอบ (ตัวอย่างเช่น, ผ่าน STRSQL หรือ Query Manager).
- คำสั่ง CALL ถูกเรียกใช้งานในคำสั่ง EXECUTE IMMEDIATE.

การใช้คำสั่ง CALL โดยที่มี definition ของโพรซีเดอร์อยู่:

คำสั่ง CALL ประเภทนี้จะอ่านข้อมูลทั้งหมดเกี่ยวกับโพรซีเดอร์ และแอ็ตทริบิวต์อากิวเมนต์จาก definition ของแคตาล็อก CREATE PROCEDURE.

ตัวอย่าง PL/I ต่อไปนี้จะแสดงคำสั่ง CALL ซึ่งตรงกับคำสั่ง CREATE PROCEDURE ที่ได้แสดงไว้.

```
DCL HV1 CHAR(10);
DCL IND1 FIXED BIN(15);
   :
EXEC SQL CREATE P1 PROCEDURE
          (INOUT PARM1 CHAR(10))
          EXTERNAL NAME MYLIB.PROC1
          LANGUAGE C
          GENERAL WITH NULLS;
   :
EXEC SQL CALL P1 (:HV1 :IND1);
   :
```

เมื่อมีการเรียกคำสั่ง CALL นี้ขึ้นมา, จะมีการเรียกไปยังโปรแกรม MYLIB/PROC1 และ ส่งต่อสองอากิวเมนต์. เนื่องจากภาษาของโปรแกรมคือ ILE C อากิวเมนต์แรกจึงเป็นสตริงอักขระความยาว 11 ตัวอักษรที่จบด้วย C NUL ซึ่งมีเนื้อหาของตัวแปรโฮสต์ HV1 อยู่ ในการเรียก ไปยังโพรซีเดอร์ ILE C, SQL จะเพิ่มอักขระหนึ่งตัวให้กับการประกาศพารามิเตอร์ ถ้าพารามิเตอร์นั้นถูกประกาศให้เป็นอักขระ, กราฟิก, วันที่, เวลา หรือตัวแปร timestamp อากิวเมนต์ที่สองคืออะเรย์ตัวบ่งชี้. ในกรณีนี้ พารามิเตอร์นี้ เป็นจำนวนเต็มแบบสั้นหนึ่งจำนวน เนื่องจากมีเพียงหนึ่งพารามิเตอร์เท่านั้นในคำสั่ง CREATE PROCEDURE อากิวเมนต์นี้ประกอบด้วยเนื้อหาของตัวแปรตัวบ่งชี้ IND1 บน entry ไปยังโพรซีเดอร์.

เนื่องจากพารามิเตอร์แรกถูกประกาศให้เป็น INOUT, SQL จึงอัพเดตตัวแปรโฮสต์ HV1 และตัวแปรตัวบ่งชี้ IND1 ด้วยค่าที่ส่งคืนมาจาก MYLIB.PROC1 ก่อนที่จะส่งคืนกลับไปยังโปรแกรมผู้ใช้.

หมายเหตุ:

1. ชื่อโพรซีเดอร์ที่ระบุในคำสั่ง CREATE PROCEDURE และ CALL จะต้องตรงกัน อย่างแท้จริงตามลำดับ เพื่อให้มีการลิงก์ระหว่างทั้งสองชื่อระหว่างที่ SQL คอมไพล์โปรแกรมล่วงหน้า.
2. สำหรับคำสั่ง CALL ที่ใส่อยู่โดยที่มีคำสั่งทั้ง CREATE PROCEDURE และ DECLARE PROCEDURE อยู่, คำสั่ง DECLARE PROCEDURE จะถูกนำมาใช้.

การใช้คำสั่ง CALL แบบฝังโดยไม่มี definition ของโพรซีเดอร์อยู่:

คำสั่ง CALL แบบ static ที่ไม่มีคำสั่ง CREATE PROCEDURE ที่ตรงกันถูกประมวลผลด้วยกฎต่อไปนี้

- อากิวเมนต์ตัวแปรโฮสต์ทั้งหมดจะถูกปฏิบัติในฐานะพารามิเตอร์ประเภท INOUT.

- ประเภท CALL คือ GENERAL (ไม่มีการส่งต่ออาถิวเมนต์ตัวบ่งชี้).
- มีการกำหนดโปรแกรมที่จะเรียกใช้งานโดยอิงจากชื่อโพรซีเดอร์ที่ระบุใน CALL, และ, หากจำเป็น, ให้อิงจากหลักการตั้งชื่อ.
- ภาษาของโปรแกรมที่จะเรียกใช้งานถูกกำหนดโดยอิงจากข้อมูลที่เรียกออกมาจาก ระบบเกี่ยวกับโปรแกรม.

ตัวอย่าง: คำสั่ง CALL แบบฝังที่ไม่มี definition ของโพรซีเดอร์อยู่

ตัวอย่าง PL/I ต่อไปนี้แสดงคำสั่ง CALL แบบฝังที่ไม่มี definition ของโพรซีเดอร์อยู่:

```
DCL HV2 CHAR(10);
   :
EXEC SQL CALL P2 (:HV2);
   :
```

เมื่อมีการเรียกคำสั่ง CALL, SQL จะพยายามที่จะ ค้นหาโปรแกรมโดยยึดตามหลักการตั้งชื่อ SQL สำหรับตัวอย่าง ข้างบน ให้ตั้งสมมุติฐานว่าอ็อพชันการตั้งชื่อของ *SYS (การตั้งชื่อระบบ) จะถูกใช้และ ไม่มีการระบุพารามิเตอร์ DFTRDBCOL บนคำสั่ง Create SQL PL/I Program (CRTSQLPLI) ในกรณีนี้, รายชื่อไลบรารีจะค้นหาโปรแกรมที่ชื่อ P2. เนื่องจากประเภทการเรียกคือ GENERAL จึงไม่มีการส่งอาถิวเมนต์เพิ่มเติมไปยังโปรแกรมสำหรับตัวแปรตัวบ่งชี้

หมายเหตุ: หากมีการระบุตัวแปรตัวบ่งชี้บนคำสั่ง CALL และค่าตัวแปรดังกล่าวน้อยกว่าศูนย์เมื่อ เรียกใช้งานคำสั่ง CALL, จะเกิดข้อผิดพลาดขึ้นเนื่องจากไม่มีทางที่จะส่งตัวบ่งชี้ ผ่านไปยังโพรซีเดอร์ได้.

สมมติว่า พบโปรแกรม P2 ในรายชื่อไลบรารี, เนื้อหาของตัวแปรโฮสต์ HV2 จะถูกส่งต่อไปยังโปรแกรมบน CALL และอาถิวเมนต์ที่ถูกส่งคืนจาก P2 จะถูกแม็พกลับไปยังตัวแปรโฮสต์ หลังจากที่ P2 ทำงานเสร็จสมบูรณ์แล้ว.

สำหรับจำนวนคงที่ที่เป็นตัวเลขที่ถูกส่งไปยังคำสั่ง CALL, จะมีการใช้กฎต่อไปนี้:

- จำนวนคงที่ที่เป็นจำนวนเต็มทั้งหมดจะถูกส่งผ่านเป็นจำนวนเต็มแบบ fullword binary.
- จำนวนคงที่ที่เป็นทศนิยมทั้งหมดจะถูกส่งผ่านเป็นค่า packed decimal. จำนวนเลขโดดและมาตราส่วน จะถูกกำหนดโดยอิงจากค่าคงที่. ตัวอย่างเช่น, ค่า 123.45 จะถูกส่งผ่านเป็น packed decimal(5,2). เช่นเดียวกัน, ค่า 001.01 ก็จะถูกส่งผ่านด้วยจำนวนเลขโดด และมาตราส่วน 5 และ 2 ตามลำดับ.
- จำนวนคงที่ทศนิยมลอยตัวทั้งหมดจะถูกส่งผ่านเป็นทศนิยมลอยตัวที่มีความเที่ยงตรงสองเท่า.

| เรจิสเตอร์พิเศษที่ระบุบนคำสั่ง CALL แบบ dynamic จะถูกส่งผ่านตามชนิดข้อมูลและความยาวที่กำหนดดังนี้:

| **CURRENT DATE**
| ส่งผ่านเป็นสตริงอักขระขนาด 10 ไบต์ในฟอร์แม็ต ISO.

| **CURRENT TIME**
| ส่งผ่านเป็นสตริงอักขระขนาด 8 ไบต์ในฟอร์แม็ต ISO.

| **CURRENT TIMESTAMP**
| ส่งผ่านสตริงอักขระขนาด 26 ไบต์ในฟอร์แม็ต IBM SQL

การใช้คำสั่ง CALL ที่ฝังด้วย SQLDA:

ใน CALL ประเภทฝัง (โดยที่ definition ของโพรซีเดอร์อาจจะมีหรือไม่มีอยู่) SQLDA อาจถูกส่งผ่าน แทนที่ลิสต์ของพารามิเตอร์

ตัวอย่าง C ต่อไปนี้อธิบายถึงสิ่งนี้. สมมุติว่าโพรซีเดอร์ที่เก็บไว้คาดว่ามี 2 พารามิเตอร์, ประเภทแรกคือ SHORT INT และประเภทที่สองคือ CHAR ซึ่งมีความยาวอักขระ 4 ตัวอักษร.

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ "สิทธิในรหัส และข้อมูลถ้อยแถลง" ในหน้า 353.

```
#define SQLDA_HV_ENTRIES 2
#define SHORTINT 500
#define NUL_TERM_CHAR 460

exec sql include sqlca;
exec sql include sqlda;
...
typedef struct sqlda Sqlda;
typedef struct sqlda* Sqldap;
...
main()
{
 Sqldap dap;
 short col1;
 char col2[4];
 int bc;
 dap = (Sqldap) malloc(bc=SQLDASIZE(SQLDA_HV_ENTRIES));
        /* SQLDASIZE is a macro defined in the sqlda include */
 col1 = 431;
 strcpy(col2,"abc");
 strncpy(dap->sqldaid,"SQLDA   ",8);
 dap->sqldabc = bc;         /* bc set in the malloc statement above */
 dap->sqln = SQLDA_HV_ENTRIES;
 dap->sqld = SQLDA_HV_ENTRIES;
 dap->sqlvar[0].sqltype = SHORTINT;
 dap->sqlvar[0].sqllen  = 2;
 dap->sqlvar[0].sqldata = (char*) &col1;
 dap->sqlvar[0].sqlname.length = 0;
 dap->sqlvar[1].sqltype = NUL_TERM_CHAR;
 dap->sqlvar[1].sqllen  = 4;
 dap->sqlvar[1].sqldata = col2;
 ...
 EXEC SQL CALL P1 USING DESCRIPTOR :*dap;
 ...
}
```

ชื่อของโพรซีเดอร์ที่เก็บไว้อาจเก็บไว้ในตัวแปรโฮสต์ และตัวแปรโฮสต์ที่ใช้ในคำสั่ง CALL , แทนที่ชื่อโพรซีเดอร์แบบ hard-code. ตัวอย่างเช่น:

```
...
main()
{
 char proc_name[15];
 ...
 strcpy (proc_name, "MYLIB.P3");
 ...
 EXEC SQL CALL :proc_name ...;
 ...
}
```

ในตัวอย่างข้างบน หาก MYLIB.P3 คาดถึงพารามิเตอร์ ลิสต์พารามิเตอร์หรือ SQLDA ซึ่งถูกส่งผ่านด้วย USING DESCRIPTOR อาจถูกใช้งานอยู่ ตามที่แสดงไว้ในตัวอย่างก่อนหน้านี้

เมื่อตัวแปรโฮสต์ที่มีชื่อโพรซีเดอร์ถูกใช้ในคำสั่ง CALL และมี CREATE PROCEDURE catalog definition อยู่, ตัวแปรโฮสต์นั้นจะถูกใช้. ชื่อโพรซีเดอร์ไม่สามารถระบุเป็นเครื่องหมายพารามิเตอร์ได้.

การใช้คำสั่ง CALL แบบ dynamic โดยที่ไม่มี CREATE PROCEDURE อยู่:

กฎต่อไปนี้จะเกี่ยวข้องกับการประมวลผลคำสั่ง CALL แบบ dynamic เมื่อไม่มี definition ของ CREATE PROCEDURE อยู่

- อากิวเมนต์ทั้งหมดจะถูกจัดให้เป็นพารามิเตอร์ประเภท IN.
- ประเภท CALL คือ GENERAL (ไม่มีการส่งต่ออากิวเมนต์ตัวบ่งชี้).
- มีการกำหนดโปรแกรมที่จะเรียกขึ้นมาโดยอิงจากชื่อโพรซีเดอร์ที่ระบุใน CALL และหลักการตั้งชื่อ.
- ภาษาของโปรแกรมที่จะเรียกใช้งานถูกกำหนดโดยอิงจากข้อมูลที่เรียกออกมาจาก ระบบเกี่ยวกับโปรแกรม.

ตัวอย่าง: คำสั่ง CALL แบบ dynamic โดยไม่มี CREATE PROCEDURE อยู่

ต่อไปนี้คือตัวอย่าง C ของคำสั่ง CALL แบบ dynamic:

```
char hv3[10],string[100];
    :
strcpy(string,"CALL MYLIB.P3 ('P3 TEST')");
EXEC SQL EXECUTE IMMEDIATE :string;
    :
```

ตัวอย่างนี้จะแสดงคำสั่ง CALL แบบ dynamic ที่ถูกเรียกใช้งานผ่านคำสั่ง EXECUTE IMMEDIATE. มีการเรียก ไปยังโปรแกรม MYLIB.P3 โดยมีการส่งต่อหนึ่งพารามิเตอร์ในฐานะที่เป็นตัวแปรอักขระซึ่งมี 'P3 TEST' อยู่.

เมื่อเรียกใช้งานคำสั่ง CALL และผ่านจำนวนคงที่แล้ว, ดังตัวอย่างก่อนหน้านี้, จะต้องจดจำความยาวของอากิวเมนต์ที่คาดไว้ในโปรแกรม. ถ้าโปรแกรม MYLIB.P3 คาดหวังอากิวเมนต์ที่มีความยาวอักขระเพียง 5 ตัวอักษร, อักขระ 2 ตัวสุดท้ายของค่าคงที่ที่ระบุไว้ในตัวอย่าง จะต้องเสียให้กับโปรแกรม.

หมายเหตุ: ด้วยสาเหตุนี้, จึงเป็นการปลอดภัยกว่าเสมอที่จะใช้ ตัวแปรโฮสต์บนคำสั่ง CALL เพื่อที่แอ็ตทริบิวต์ของโพรซีเดอร์จะได้ตรงกันแน่นอน และเพื่อที่อักขระจะไม่หายไป. สำหรับ SQL แบบ dynamic, สามารถระบุตัวแปรโฮสต์สำหรับอากิวเมนต์คำสั่ง CALL ได้หากคำสั่ง PREPARE และ EXECUTE ถูกนำมาใช้เพื่อประมวลผล SQL ดังกล่าว.

ตัวอย่าง: คำสั่ง CALL:

ตัวอย่างเหล่านี้แสดงถึงวิธีการที่อากิวเมนต์ของคำสั่ง CALL ถูกส่งผ่านไปยังโพรซีเดอร์สำหรับหลายภาษา และวิธีการที่ได้รับอากิวเมนต์ เข้าสู่ตัวแปรโลคัลในโพรซีเดอร์

ตัวอย่าง 1: ILE C และโพรซีเดอร์ PL/I ที่ถูกเรียกจากโปรแกรม ILE C:

ตัวอย่างนี้แสดงโปรแกรม ILE C ที่ใช้ CREATE PROCEDURE definition เพื่อเรียกโพรซีเดอร์ P1 และ P2 โพรซีเดอร์ P1 ถูกบันทึกลงใน ILE C และมี 10 พารามิเตอร์ โพรซีเดอร์ P2 ถูกบันทึกลงใน PL/I และมี 10 พารามิเตอร์เช่นกัน

การนิยามโพรซีเดอร์ P1 และ P2

```
EXEC SQL CREATE PROCEDURE P1 (INOUT PARM1 CHAR(10),
                              INOUT PARM2 INTEGER,
                              INOUT PARM3 SMALLINT,
                              INOUT PARM4 FLOAT(22),
                              INOUT PARM5 FLOAT(53),
                              INOUT PARM6 DECIMAL(10,5),
                              INOUT PARM7 VARCHAR(10),
                              INOUT PARM8 DATE,
                              INOUT PARM9 TIME,
                              INOUT PARM10 TIMESTAMP)
          EXTERNAL NAME  TEST12.CALLPROC2
          LANGUAGE C GENERAL WITH NULLS

EXEC SQL CREATE PROCEDURE P2 (INOUT PARM1 CHAR(10),
                              INOUT PARM2 INTEGER,
                              INOUT PARM3 SMALLINT,
                              INOUT PARM4 FLOAT(22),
                              INOUT PARM5 FLOAT(53),
                              INOUT PARM6 DECIMAL(10,5),
                              INOUT PARM7 VARCHAR(10),
                              INOUT PARM8 DATE,
                              INOUT PARM9 TIME,
                              INOUT PARM10 TIMESTAMP)
          EXTERNAL NAME  TEST12.CALLPROC
          LANGUAGE PLI GENERAL WITH NULLS
```

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

การเรียกโพรซีเดอร์ P1 และ P2

```
/*****************************************************************/
/*********** START OF SQL C Application **********************/

#include <stdio.h>
#include <string.h>
#include <decimal.h>
main()
{
 EXEC SQL INCLUDE SQLCA;
 char PARM1[10];
 signed long int PARM2;
 signed short int PARM3;
 float PARM4;
 double PARM5;
 decimal(10,5) PARM6;
 struct { signed short int parm7l;
          char parm7c[10];
        } PARM7;
char PARM8[10];        /* FOR DATE */
char PARM9[8];         /* FOR TIME */
char PARM10[26];       /* FOR TIMESTAMP */
```

```
/*******************************************************/
/* Initialize variables for the call to the procedures */
/*******************************************************/
strcpy(PARM1,"PARM1");
PARM2 = 7000;
PARM3 = -1;
PARM4 = 1.2;
PARM5 = 1.0;
PARM6 = 10.555;
PARM7.parm7l = 5;
strcpy(PARM7.parm7c,"PARM7");
strncpy(PARM8,"1994-12-31",10);         /* FOR DATE      */
strncpy(PARM9,"12.00.00",8);            /* FOR TIME      */
strncpy(PARM10,"1994-12-31-12.00.00.000000",26);
                                        /* FOR TIMESTAMP */
/*********************************************/
/* Call the C procedure                      */
/*                                           */
/*                                           */
/*********************************************/
EXEC SQL CALL P1 (:PARM1, :PARM2, :PARM3,
                  :PARM4, :PARM5, :PARM6,
                  :PARM7, :PARM8, :PARM9,
                  :PARM10 );
if (strncmp(SQLSTATE,"00000",5))
 {
 /* Handle error or warning returned on CALL statement */
 }

/* Process return values from the CALL.                */
:


/*********************************************/
/* Call the PLI procedure                    */
/*                                           */
/*                                           */
/*********************************************/
/* Reset the host variables before making the CALL      */
/*                                                      */
:
EXEC SQL CALL P2 (:PARM1, :PARM2, :PARM3,
                  :PARM4, :PARM5, :PARM6,
                  :PARM7, :PARM8, :PARM9,
                  :PARM10 );
if (strncmp(SQLSTATE,"00000",5))
 {
  /* Handle error or warning returned on CALL statement */
}
/* Process return values from the CALL.                */
:
}

/******** END OF C APPLICATION *********************************/
/***************************************************************/
```

โพรซีเดอร์ P1

```
/******** START OF C PROCEDURE P1 ******************************/
/*        PROGRAM TEST12/CALLPROC2                              */
/*************************************************************/

#include <stdio.h>
#include <string.h>
#include <decimal.h>
main(argc,argv)
  int argc;
  char *argv[];
  {
    char parm1[11];
    long int parm2;
    short int parm3,i,j,*ind,ind1,ind2,ind3,ind4,ind5,ind6,ind7,
              ind8,ind9,ind10;
    float parm4;
    double parm5;
    decimal(10,5) parm6;
    char parm7[11];
    char parm8[10];
    char parm9[8];
    char parm10[26];
    /* ********************************************************/
    /* Receive the parameters into the local variables -       */
    /* Character, date, time, and timestamp are passed as      */
    /* NUL terminated strings - cast the argument vector to    */
    /* the proper data type for each variable. Note that       */
    /* the argument vector can be used directly instead of     */
    /* copying the parameters into local variables - the copy  */
    /* is done here just to illustrate the method.             */
    /* ********************************************************/

    /* Copy 10 byte character string into local variable      */
    strcpy(parm1,argv[1]);

    /* Copy 4 byte integer into local variable                */
    parm2 = *(int *) argv[2];

    /* Copy 2 byte integer into local variable                */
    parm3 = *(short int *) argv[3];

    /* Copy floating point number into local variable         */
    parm4 = *(float *) argv[4];

    /* Copy double precision number into local variable       */
    parm5 = *(double *) argv[5];

    /* Copy decimal number into local variable                */
    parm6 = *(decimal(10,5) *) argv[6];
/*********************************************************/
/* Copy NUL terminated string into local variable.         */
/* Note that the parameter in the CREATE PROCEDURE was   */
/* declared as varying length character. For C, varying   */
```

```
/* length are passed as NUL terminated strings unless     */
/* FOR BIT DATA is specified in the CREATE PROCEDURE     */
/*********************************************************/
strcpy(parm7,argv[7]);

/*********************************************************/
/* Copy date into local variable.                        */
/* Note that date and time variables are always passed in */
/* ISO format so that the lengths of the strings are     */
/* known. strcpy works here just as well.                */
/*********************************************************/
strncpy(parm8,argv[8],10);

/* Copy time into local variable                         */
strncpy(parm9,argv[9],8);

/*********************************************************/
/* Copy timestamp into local variable.                   */
/* IBM SQL timestamp format is always passed so the length*/
/* of the string is known.                               */
/*********************************************************/
strncpy(parm10,argv[10],26);

/*********************************************************/
/* The indicator array is passed as an array of short    */
/* integers. There is one entry for each parameter passed */
/* on the CREATE PROCEDURE (10 for this example).       */
/* Below is one way to set each indicator into separate  */
/* variables.                                            */
/*********************************************************/
   ind = (short int *) argv[11];
   ind1 = *(ind++);
   ind2 = *(ind++);
   ind3 = *(ind++);
   ind4 = *(ind++);
   ind5 = *(ind++);
   ind6 = *(ind++);
   ind7 = *(ind++);
   ind8 = *(ind++);
   ind9 = *(ind++);
   ind10 = *(ind++);
  :
/* Perform any additional processing here                */
  :
return;
}
/******** END OF C PROCEDURE P1 *****************************/
```

โพรซีเดอร์ P2

```
/******** START OF PL/I PROCEDURE P2 *************************/
/******** PROGRAM TEST12/CALLPROC ****************************/
/*************************************************************/


CALLPROC :PROC( PARM1,PARM2,PARM3,PARM4,PARM5,PARM6,PARM7,
```

```
                      PARM8,PARM9,PARM10,PARM11);
 DCL  SYSPRINT FILE STREAM OUTPUT EXTERNAL;
 OPEN FILE(SYSPRINT);
 DCL PARM1 CHAR(10);
 DCL PARM2 FIXED BIN(31);
 DCL PARM3 FIXED BIN(15);
 DCL PARM4 BIN FLOAT(22);
 DCL PARM5 BIN FLOAT(53);
 DCL PARM6 FIXED DEC(10,5);
 DCL PARM7 CHARACTER(10) VARYING;
 DCL PARM8 CHAR(10);      /* FOR DATE */
 DCL PARM9 CHAR(8);       /* FOR TIME */
 DCL PARM10 CHAR(26);     /* FOR TIMESTAMP */
 DCL PARM11(10) FIXED BIN(15);   /* Indicators */

 /* PERFORM LOGIC - Variables can be set to other values for  */
 /* return to the calling program.                            */

 :

 END CALLPROC;
```

ตัวอย่าง 2: โพรซีเดอร์ REXX ที่ถูกเรียกจากโปรแกรม ILE C:

ตัวอย่างนี้แสดงโพรซีเดอร์ REXX ที่ถูกเรียกจากโปรแกรม ILE C

การนิยามโพรซีเดอร์ REXX

```
EXEC SQL CREATE PROCEDURE REXXPROC
              (IN PARM1 CHARACTER(20),
               IN PARM2 INTEGER,
               IN PARM3 DECIMAL(10,5),
               IN PARM4 DOUBLE PRECISION,
               IN PARM5 VARCHAR(10),
               IN PARM6 GRAPHIC(4),
               IN PARM7 VARGRAPHIC(10),
               IN PARM8 DATE,
               IN PARM9 TIME,
               IN PARM10 TIMESTAMP)
          EXTERNAL NAME 'TEST.CALLSRC(CALLREXX)'
          LANGUAGE REXX GENERAL WITH NULLS
```

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

การเรียกโพรซีเดอร์ REXX

```
/*****************************************************************/
/************ START OF SQL C Application **********************/

        #include <decimal.h>
        #include <stdio.h>
        #include <string.h>
        #include <wcstr.h>
        /*---------------------------------------------------------*/
        exec sql include sqlca;
```

```
        exec sql include sqlda;
        /* ************************************************************/
        /* Declare host variable for the CALL statement              */
        /* ************************************************************/
        char parm1[20];
        signed long int parm2;
        decimal(10,5) parm3;
        double parm4;
        struct {  short  dlen;
                  char dat[10];
                } parm5;
        wchar_t parm6[4] = { 0xC1C1, 0xC2C2, 0xC3C3, 0x0000 };
        struct {  short  dlen;
                  wchar_t dat[10];
                } parm7 = {0x0009, 0xE2E2,0xE3E3,0xE4E4, 0xE5E5, 0xE6E6,
                           0xE7E7, 0xE8E8, 0xE9E9, 0xC1C1, 0x0000 };

        char parm8[10];
        char parm9[8];
        char parm10[26];
        main()
        {
        /* ************************************************************/
        /* Call the procedure - on return from the CALL statement the  */
        /* SQLCODE should be 0. If the SQLCODE is non-zero,            */
        /* the procedure detected an error.                            */
        /* ************************************************************/
        strcpy(parm1,"TestingREXX");
        parm2 = 12345;
        parm3 = 5.5;
        parm4 = 3e3;
        parm5.dlen = 5;
        strcpy(parm5.dat,"parm6");
        strcpy(parm8,"1994-01-01");
        strcpy(parm9,"13.01.00");
        strcpy(parm10,"1994-01-01-13.01.00.000000");

         EXEC SQL CALL REXXPROC (:parm1, :parm2,
                                 :parm3,:parm4,
                                 :parm5, :parm6,
                                 :parm7,
                                 :parm8, :parm9,
                                 :parm10);

         if (strncpy(SQLSTATE,"00000",5))
           {
           /* handle error or warning returned on CALL   */
           :
           }
                :
       }

/****** END OF SQL C APPLICATION   ***********************************/
/*********************************************************************/
```

```
/*********************************************************************/
/****** START OF REXX MEMBER TEST/CALLSRC CALLREXX ********************/
/*********************************************************************/
       /* REXX source member TEST/CALLSRC CALLREXX              */
       /* Note the extra parameter being passed for the indicator*/
       /* array.                                                */
       /*                                                       */
       /* ACCEPT THE FOLLOWING INPUT VARIABLES SET TO THE       */
       /* SPECIFIED VALUES :                                    */
       /* AR1        CHAR(20)          =  'TestingREXX'         */
       /* AR2        INTEGER           = 12345                  */
       /* AR3        DECIMAL(10,5)     = 5.5                    */
       /* AR4        DOUBLE PRECISION  = 3e3                    */
       /* AR5        VARCHAR(10)       = 'parm6'                */
       /* AR6        GRAPHIC           = G'C1C1C2C2C3C3'        */
       /* AR7        VARGRAPHIC        =                        */
       /*             G'E2E2E3E3E4E4E5E5E6E6E7E7E8E8E9E9EAEA'   */
       /* AR8        DATE              = '1994-01-01'           */
       /* AR9        TIME              = '13.01.00'             */
       /* AR10       TIMESTAMP         =                        */
       /*                       '1994-01-01-13.01.00.000000'    */
       /* AR11       INDICATOR ARRAY   = +0+0+0+0+0+0+0+0+0+0   */

       /*********************************************************/
       /* Parse the arguments into individual parameters        */
       /*********************************************************/
       parse arg ar1 ar2 ar3 ar4 ar5 ar6 ar7 ar8 ar9 ar10 ar11

       /*********************************************************/
       /* Verify that the values are as expected                */
       /*********************************************************/
       if ar1<>"'TestingREXX'" then signal ar1tag
       if ar2<>12345 then signal ar2tag
       if ar3<>5.5 then signal ar3tag
       if ar4<>3e3 then signal ar4tag
       if ar5<>"'parm6'" then signal ar5tag
       if ar6 <>"G'AABBCC'" then signal ar6tag
       if ar7 <>"G'SSTTUUVVWWXXYYZZAA'" then ,
        signal ar7tag
       if ar8 <> "'1994-01-01'" then signal ar8tag
       if ar9 <> "'13.01.00'" then signal ar9tag
       if ar10 <> "'1994-01-01-13.01.00.000000'" then signal ar10tag
       if ar11 <> "+0+0+0+0+0+0+0+0+0+0" then signal ar11tag
       /*********************************************************/
       /* Perform other processing as necessary ..              */
       /*********************************************************/
          :
       /*********************************************************/
       /* Indicate the call was successful by exiting with a    */
       /* return code of 0                                      */
       /*********************************************************/
       exit(0)

       ar1tag:
       say "ar1 did not match" ar1
```

```
exit(1)
ar2tag:
say "ar2 did not match" ar2
exit(1)
:
:

/************ END OF REXX MEMBER   *********************************/
```

การส่งคืนเซ็ตของผลลัพธ์จากโพรซีเดอร์ที่เก็บไว้

นอกจากการส่งกลับเอาต์พุตพารามิเตอร์แล้ว โพรซีเดอร์ที่เก็บไว้ยังมีอีกคุณลักษณะหนึ่งซึ่งสามารถส่งกลับเซ็ตของผลลัพธ์ (นั่นคือ ตารางผลลัพธ์ที่เกี่ยวข้อง กับเคอร์เซอร์ที่ถูกเปิดไว้ในโพรซีเดอร์ที่เก็บไว้) ไปยังแอ็พพลิเคชันที่ส่ง คำสั่ง CALL จากนั้น แอ็พพลิเคชันนั้นจะใช้การร้องขอดึงข้อมูลเพื่ออ่านแถวของเคอร์เซอร์ของเซ็ตของผลลัพธ์

เซ็ตของผลลัพธ์ถูกส่งกลับตามแอ็ตทริบิวต์ความสามารถในการส่งกลับของเคอร์เซอร์. แอ็ตทริบิวต์ความสามารถในการส่งกลับของเคอร์เซอร์สามารถกำหนดได้ในคำสั่ง DECLARE CURSOR หรือเป็นค่าดีฟอลต์. คำสั่ง SET RESULT SETS ยังอนุญาตให้กำหนดว่า เซ็ตของผลลัพธ์ควรถูกส่งกลับไปที่ใด. โดยดีฟอลต์, เคอร์เซอร์ซึ่งถูกเปิดไว้ในโพรซิเดอร์ที่เก็บไว้ถูกกำหนดให้มีแอ็ตทริบิวต์ของ RETURN TO CALLER. การส่งกลับเซ็ตของผลลัพธ์พร้อมด้วยเคอร์เซอร์ไปยังแอ็พพลิเคชันซึ่งอยู่ชั้นนอกสุดของสแต็กการเรียก, แอ็ตทริบิวต์ความสามารถในการส่งกลับของ RETURN TO CLIENT ถูกกำหนดไว้ในข้อความ DECLARE CURSOR. ซึ่งจะอนุญาตให้โพรซิเดอร์ชั้นในสามารถส่งกลับเซ็ตของผลลัพธ์เมื่อแอ็พพลิเคชันเรียกโพรซิเดอร์แบบซับซ้อน. สำหรับเคอร์เซอร์ของเซ็ตของผลลัพธ์ที่ไม่เคยถูกส่งกลับไปยังผู้เรียกหรือไคลเอ็นต์, แอ็ตทริบิวต์ความสามารถในการส่งกลับของ WITHOUT RETURN ถูกระบุไว้บนข้อความ DECLARE CURSOR.

หมายเหตุ: เมื่อคุณใช้ COBOL เซ็ตของผลลัพธ์ จะถูกปิดโดยอัตโนมัติ เนื่องจากการตั้งค่าโปรแกรม COBOL เปลี่ยน คำสั่ง EXIT PROGRAM เป็น EXIT PROGRAM AND CONTINUE RUN UNIT และเซ็ตของผลลัพธ์ ควรจะถูกส่งคืน

มีหลายกรณีที่การเปิดเคอร์เซอร์ในโพรซิเดอร์ที่เก็บไว้และการรับเซ็ตของผลลัพธ์กลับมีข้อดีหลายข้อเหนือการเปิดเคอร์เซอร์โดยตรงจากแอ็พพลิเคชัน. ตัวอย่าง, ความปลอดภัยต่อตารางที่เคียวรี่อ้างถึงสามารถรับสิทธิ์การใช้งานได้จากโพรซิเดอร์ที่เก็บไว้ ดังนั้นจึงไม่มีความจำเป็นต้องให้สิทธิ์การใช้งานตารางต่างๆกับผู้ใช้แอ็พพลิเคชั่นโดยตรง. เพียงแค่, มีการให้สิทธิ์ในการใช้งานที่จะเรียกโพรซิเดอร์ที่เก็บไว้กับผู้ใช้, ซึ่งได้รับการคอมไพล์พร้อมกับสิทธิ์ในการใช้งานที่เพียงพอในการเข้าถึงตาราง. ข้อดีของการเปิดเคอร์เซอร์ในโพรซิเดอร์ที่เก็บไว้อีกอันหนึ่งคือการที่การเรียกไปยังโพรซิเดอร์ที่เก็บไว้ครั้งหนึ่งสามารถรับค่าเซ็ตผลลัพธ์กลับได้หลายชุด, ซึ่งทำให้มีประสิทธิภาพมากกว่าการเปิดเคอร์เซอร์แยกกันจากแอ็พพลิเคชันที่เรียก. ยิ่งไปกว่านั้น, การเรียกไปที่โพรซิเดอร์ที่เก็บไว้อันเดียวกันในแต่ละครั้งอาจได้เซ็ตผลลัพธ์ที่แตกต่างกัน, ทำให้เกิดความคล่องตัวของแอ็พพลิเคชัน.

อินเตอร์เฟสที่สามารถใช้งานร่วมกับเซ็ตผลลัพธ์ของโพรซิเดอร์ที่เก็บไว้ได้แก่ JDBC, CLI, และ ODBC. ตัวอย่างสำหรับวิธีการใช้อินเตอร์เฟส API เหล่านี้สำหรับการทำงานกับเซ็ตของผลลัพธ์ของโพรซีเดอร์ที่เก็บไว้ซึ่งอยู่ในตัวอย่างต่อไปนี้

ตัวอย่าง 1: การเรียกโพรซีเดอร์ที่เก็บไว้ ซึ่งส่งกลับชุดผลลัพธ์หนึ่งชุด:

ตัวอย่างแสดงการเรียก API ซึ่งแอ็พพลิเคชัน Open Database Connectivity (ODBC) สามารถใช้ในการเรียกโพรซีเดอร์ที่เก็บไว้ เพื่อส่งกลับ ชุดผลลัพธ์หนึ่งชุด

สังเกตว่าไม่มีการระบุความสามารถในการส่งกลับอย่างชัดแจ้งในข้อความ DECLARE CURSOR. เมื่อมีเพียงหนึ่งโพรซิเดอร์ที่เก็บไว้ในสแต็กการเรียก, แอ็ตทริบิวต์ความสามารถในการส่งกลับของ RETURN TO CALLER และของ RETURN TO

CLIENT จะทำให้เซ็ตผลลัพธ์พร้อมสำหรับผู้ที่เรียกใช้แอ็พพลิเคชัน. โปรดสังเกตด้วยว่าโพรซิเดอร์ที่เก็บไว้ถูกกำหนดด้วย DYNAMIC RESULT SETS clause. สำหรับโพรซิเดอร์ SQL, clause นี้จำเป็นต้องมีถ้าต้องการให้โพรซิเดอร์ที่เก็บไว้ส่งกลับเซ็ตผลลัพธ์.

การนิยามโพรซีเดอร์ที่เก็บไว้:

```
PROCEDURE prod.resset

CREATE PROCEDURE prod.resset () LANGUAGE SQL
 DYNAMIC RESULT SETS 1
 BEGIN
 DECLARE C1 CURSOR FOR SELECT * FROM QIWS.QCUSTCDT;
 OPEN C1;
 RETURN;
 END
```

แอ็พพลิเคชัน ODBC

หมายเหตุ: ตรรกะบางตัวได้ถูกลบออก.

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

```
 :
strcpy(stmt,"call prod.resset()");
rc = SQLExecDirect(hstmt,stmt,SQL_NTS);
if (rc == SQL_SUCCESS)
 {
   // CALL statement has executed successfully. Process the result set.
   // Get number of result columns for the result set.
   rc = SQLNumResultCols(hstmt, &wNum);
   if (rc == SQL_SUCCESS)
       // Get description of result columns in result set
      { rc = SQLDescribeCol(hstmt,ä);
        if (rc == SQL_SUCCESS)
             :

       {
       // Bind result columns based on attributes returned
       //
         rc = SQLBindCol(hstmt,ä);
       :
      // FETCH records until EOF is returned

        rc = SQLFetch(hstmt);
        while (rc == SQL_SUCCESS)
          { // process result returned on the SQLFetch
                :
            rc = SQLFetch(hstmt);
          }
         :
          }
        // Close the result set cursor when done with it.
        rc = SQLFreeStmt(hstmt,SQL_CLOSE);
       :
```

ตัวอย่าง 2: การเรียกโพรซีเดอร์ที่เก็บไว้ซึ่งส่งกลับเซ็ตผลลัพธ์หนึ่งชุดจาก โพรซีเดอร์ที่ซ้อนอยู่:

ตัวอย่างนี้แสดงวิธีที่โพรซีเดอร์ที่เก็บไว้ที่ซ้อนอยู่สามารถเปิดและส่งกลับเซ็ตผลลัพธ์ไปยังโพรซีเดอร์ที่อยู่นอกสุด.

การส่งกลับเซ็ตผลลัพธ์ไปยังโพรซีเดอร์ที่อยู่นอกสุดในสภาพแวดล้อมที่มีโพรซีเดอร์ที่เก็บไว้ซ้อนอยู่, แอ็ตทริบิวต์ความสามารถในการส่งกลับ RETURN TO CLIENT ควรจะถูกใช้ในข้อความ DECLARE CURSOR หรือในข้อความ SET RESULT SETS เพื่อที่จะแสดงความต้องการที่จะส่งกลับเคอร์เซอร์ไปยังแอ็พพลิเคชันที่เรียกมาที่โพรซีเดอร์ชั้นนอกสุด. โปรดสังเกตว่าการเรียกโพรซีเดอร์ที่ซ้อนกันนี้จะส่งเซ็ตผลลัพธ์สองชุดไปยังไคลเอ็นต์; ชุดแรก, ชุดผลลัพธ์แบบอะเรย์, และชุดที่สองคือชุดผลลัพธ์แบบเคอร์เซอร์. ไคลเอ็นต์แอ็พพลิเคชันแบบ ODBC และแบบ JDBC พร้อมด้วยโพรซีเดอร์ที่เก็บไว้ถูกแสดงไว้ข้างล่างนี้.

การนิยามโพรซีเดอร์ที่เก็บไว้

```
CREATE PROCEDURE prod.rtnnested () LANGUAGE CL DYNAMIC RESULT SET 2
        EXTERNAL NAME prod.rtnnested GENERAL

CREATE PROCEDURE prod.rtnclient () LANGUAGE RPGLE
        EXTERNAL NAME prod.rtnclient GENERAL
```

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

ซอร์ส CL สำหรับโพรซีเดอร์ที่เก็บไว้ prod.rtnnested

```
PGM
            CALL       PGM(PROD/RTNCLIENT)
```

ซอร์ส ILE RPG สำหรับโพรซีเดอร์ที่เก็บไว้ prod.rtnclient

```
DRESULT           DS                  OCCURS(20)
D COL1                    1     16A
C     1             DO        10            X                 2 0
C     X             OCCUR     RESULT
C                   EVAL      COL1='array result set'
C                   ENDDO
C                   EVAL      X=X-1
C/EXEC SQL DECLARE C2 CURSOR WITH RETURN TO CLIENT
C+ FOR SELECT LSTNAM FROM QIWS.QCUSTCDT FOR FETCH ONLY
C/END-EXEC
C/EXEC SQL
C+  OPEN C2
C/END-EXEC
C/EXEC SQL
C+ SET RESULT SETS FOR RETURN TO CLIENT ARRAY :RESULT FOR :X ROWS,
C+  CURSOR C2
C/END-EXEC
C                   SETON                                         LR
C                   RETURN
```

แอ็พพลิเคชัน ODBC

```
//*********************************************************************
//
// Module:
//    Examples.C
```

```
//
// Purpose:
//    Perform calls to stored procedures to get back result sets.

//
// ********************************************************************


#include "common.h"
#include "stdio.h"


// ********************************************************************
//
// Local function prototypes.
//
// ********************************************************************

SWORD   FAR PASCAL RetClient(lpSERVERINFO lpSI);
BOOL    FAR PASCAL Bind_Params(HSTMT);
BOOL    FAR PASCAL Bind_First_RS(HSTMT);
BOOL    FAR PASCAL Bind_Second_RS(HSTMT);

// ********************************************************************
//
// Constant strings definitions for SQL statements used in
// the auto test.
//
// ********************************************************************
//
// Declarations of variables global to the auto test.
//
// ********************************************************************
#define ARRAYCOL_LEN 16
#define LSTNAM_LEN 8
char   stmt[2048];
char   buf[2000];

UDWORD rowcnt;
char arraycol[ARRAYCOL_LEN+1];
char lstnam[LSTNAM_LEN+1];
SDWORD cbcol1,cbcol2;

lpSERVERINFO lpSI;  /* Pointer to a SERVERINFO structure. */


// ********************************************************************
//
//  Define the auto test name and the number of test cases
//     for the current auto test.  These informations will
//     be returned by AutoTestName().
//
// ********************************************************************

LPSTR szAutoTestName =  CREATE_NAME("Result Sets Examples");
```

```
UINT  iNumOfTestCases = 1;



// ********************************************************************
//
//  Define the structure for test case names, descriptions,
//     and function names for the current auto test.
//     Test case names and descriptions will be returned by
//     AutoTestDesc().  Functions will be run by
//     AutoTestFunc() if the bits for the corresponding test cases
//     are set in the rglMask member of the SERVERINFO
//     structure.
//
// ********************************************************************
struct TestCase  TestCasesInfo[] =
{
        "Return to Client",
        "2 result sets   ",
         RetClient

};


// ********************************************************************
//
// Sample return to Client:
//      Return to Client result sets. Call a CL program which in turn
//      calls an RPG program which returns 2 result sets. The first
//      result set is an array result set and the second is a cursor
//      result set.
//
//
// ********************************************************************
SWORD FAR PASCAL RetClient(lpSERVERINFO lpSI)
{
   SWORD      sRC = SUCCESS;
   RETCODE    returncode;
   HENV       henv;
   HDBC       hdbc;
   HSTMT      hstmt;



   if (FullConnect(lpSI, &henv, &hdbc, &hstmt) == FALSE)
   {
      sRC = FAIL;
      goto ExitNoDisconnect;
   }
   // ******************************************************
   // Call CL program PROD.RTNNESTED, which in turn calls RPG
   // program RTNCLIENT.
   // ******************************************************
   strcpy(stmt,"CALL PROD.RTNNESTED()");
   // ************************************************************
```

```
// Call the CL program prod.rtnnested. This program will in turn
// call the RPG program proc.rtnclient, which will open 2 result
// sets for return to this ODBC application.
// *************************************************************
returncode = SQLExecDirect(hstmt,stmt,SQL_NTS);
if (returncode != SQL_SUCCESS)
{
   vWrite(lpSI, "CALL PROD.RTNNESTED is not Successful", TRUE);
}
else
{
   vWrite(lpSI, "CALL PROC.RTNNESTED was Successful", TRUE);
}
// *************************************************************
// Bind the array result set output column. Note that the result
// sets are returned to the application in the order that they
// are specified on the SET RESULT SETS statement.
// *************************************************************
if (Bind_First_RS(hstmt) == FALSE)
{
   myRETCHECK(lpSI, henv, hdbc, hstmt, SQL_SUCCESS,
              returncode, "Bind_First_RS");
   sRC = FAIL;
   goto ErrorRet;
}
else
{
   vWrite(lpSI, "Bind_First_RS Complete...", TRUE);
}
// *************************************************************
// Fetch the rows from the array result set. After the last row
// is read, a returncode of SQL_NO_DATA_FOUND will be returned to
// the application on the SQLFetch request.
// *************************************************************
returncode = SQLFetch(hstmt);
while(returncode == SQL_SUCCESS)
{
 wsprintf(stmt,"array column = %s",arraycol);
 vWrite(lpSI,stmt,TRUE);
 returncode = SQLFetch(hstmt);
}
if (returncode == SQL_NO_DATA_FOUND) ;
else {
       myRETCHECK(lpSI, henv, hdbc, hstmt, SQL_SUCCESS_WITH_INFO,
                  returncode, "SQLFetch");
       sRC = FAIL;
       goto ErrorRet;
     }
// *********************************************************
// Get any remaining result sets from the call. The next
// result set corresponds to cursor C2 opened in the RPG
// Program.
// *********************************************************
returncode = SQLMoreResults(hstmt);
if (returncode != SQL_SUCCESS)
```

```
 {
  myRETCHECK(lpSI, henv, hdbc, hstmt, SQL_SUCCESS, returncode, "SQLMoreResults");
  sRC = FAIL;
  goto ErrorRet;
 }
// *************************************************************
// Bind the cursor result set output column. Note that the result
// sets are returned to the application in the order that they
// are specified on the SET RESULT SETS statement.
// *************************************************************

if (Bind_Second_RS(hstmt) == FALSE)
{
   myRETCHECK(lpSI, henv, hdbc, hstmt, SQL_SUCCESS,
              returncode, "Bind_Second_RS");
   sRC = FAIL;
   goto ErrorRet;
}
else
{
   vWrite(lpSI, "Bind_Second_RS Complete...", TRUE);
}
// *************************************************************
// Fetch the rows from the cursor result set. After the last row
// is read, a returncode of SQL_NO_DATA_FOUND will be returned to
// the application on the SQLFetch request.
// *************************************************************
 returncode = SQLFetch(hstmt);
while(returncode == SQL_SUCCESS)
{
 wsprintf(stmt,"lstnam = %s",lstnam);
 vWrite(lpSI,stmt,TRUE);
 returncode = SQLFetch(hstmt);
}
if (returncode == SQL_NO_DATA_FOUND) ;
else {
       myRETCHECK(lpSI, henv, hdbc, hstmt, SQL_SUCCESS_WITH_INFO,
                  returncode, "SQLFetch");
       sRC = FAIL;
       goto ErrorRet;
     }

returncode = SQLFreeStmt(hstmt,SQL_CLOSE);
if (returncode != SQL_SUCCESS)
{
   myRETCHECK(lpSI, henv, hdbc, hstmt, SQL_SUCCESS,
                    returncode, "Close statement");
   sRC = FAIL;
   goto ErrorRet;
}
else
{
   vWrite(lpSI, "Close statement...", TRUE);
}
```

```
ErrorRet:
   FullDisconnect(lpSI, henv, hdbc, hstmt);
   if (sRC == FAIL)
   {
      // a failure in an ODBC function that prevents completion of the
      // test - for example, connect to the server
      vWrite(lpSI, "\t\t *** Unrecoverable RTNClient Test FAILURE ***",  TRUE);
   } /* endif */

ExitNoDisconnect:

   return(sRC);
} // RetClient
```

```
BOOL FAR PASCAL Bind_First_RS(HSTMT hstmt)
{
 RETCODE rc = SQL_SUCCESS;
   rc = SQLBindCol(hstmt,1,SQL_C_CHAR,arraycol,ARRAYCOL_LEN+1, &cbcol1);
   if (rc != SQL_SUCCESS) return FALSE;
   return TRUE;
}
BOOL FAR PASCAL Bind_Second_RS(HSTMT hstmt)
{
 RETCODE rc = SQL_SUCCESS;
   rc = SQLBindCol(hstmt,1,SQL_C_CHAR,lstnam,LSTNAM_LEN+1,&dbcol2);
   if (rc != SQL_SUCCESS) return FALSE;
   return TRUE;
}
```

แอ็พพลิเคชัน JDBC

```
//-------------------------------------------------------
// Call Nested procedures which return result sets to the
// client, in this case a JDBC client.
//-------------------------------------------------------
import java.sql.*;
public class callNested
{
  public static void main (String argv[])               // Main entry point
  {
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException e) {
      e.printStackTrace();
    }

   try {
      Connection jdbcCon =
```

```
DriverManager.getConnection("jdbc:db2:lp066ab","Userid","xxxxxxx");
      jdbcCon.setAutoCommit(false);
      CallableStatement cs = jdbcCon.prepareCall("CALL PROD.RTNNESTED");
      cs.execute();
      ResultSet rs1 = cs.getResultSet();
      int r = 0;
while (rs1.next())
      {
      r++;
      String s1 = rs1.getString(1);
      System.out.print("Result set 1 Row: " + r + ": ");
      System.out.print(s1 + "   " );
      System.out.println();
      }
      cs.getMoreResults();
      r = 0;
      ResultSet rs2 = cs.getResultSet();
      while (rs2.next())
      {
      r++;
      String s2 = rs2.getString(1);
      System.out.print("Result set 2 Row: " + r + ": ");
      System.out.print(s2 + "   ");
      System.out.println();
      }
   }
   catch ( SQLException e ) {
      System.out.println( "SQLState: " + e.getSQLState() );
      System.out.println( "Message : " + e.getMessage() );
      e.printStackTrace();
   }
  }   // main
}
```

พารามิเตอร์ที่ผ่านหลักการสำหรับสำหรับโพรซีเดอร์ที่เก็บไว้และฟังก์ชันแบบผู้ใช้กำหนดเอง

คำสั่ง CALL และการเรียกใช้ฟังก์ชันสามารถส่งผ่านอากิวเมนต์ไปยังโปรแกรมที่บันทึกในภาษาโฮสต์ทั้งหมดที่สนับสนุนและโพรซีเดอร์ REXX

แต่ละภาษาจะสนับสนุนประเภทข้อมูลที่ต่างกันที่ถูกปรับแต่งสำหรับภาษานั้นๆ ดังแสดงในตารางต่อไปนี้. ประเภทข้อมูล SQL ถูกใส่ไว้ในคอลัมน์ด้านซ้ายสุดของแต่ละตาราง คอลัมน์อื่นๆ ในแถวนั้นมีการบ่งชี้ ว่ามีการสนับสนุนประเภทข้อมูลหรือไม่ในฐานะที่เป็นประเภทพารามิเตอร์สำหรับภาษาเฉพาะ. หากคอลัมน์ว่างเปล่า, ชนิดข้อมูลจะไม่สนับสนุนในฐานะเป็นประเภทพารามิเตอร์สำหรับภาษานั้น. การประกาศตัวแปรโฮสต์ แสดงว่า DB2 for i5/OS สนับสนุน ประเภทข้อมูลนี้ในฐานะที่เป็นพารามิเตอร์ในภาษานี้ การประกาศแสดงว่าตัวแปรโฮสต์จะต้องประกาศอย่างไรเพื่อให้ได้รับและตั้งอย่างถูกต้องด้วยโพรซีเดอร์หรือฟังก์ชัน. เมื่อเรียก SQL โพรซีเดอร์หรือฟังก์ชัน ประเภทข้อมูล SQL ทั้งหมดจะถูกสนับสนุน ดังนั้นจึงไม่มีคอลัมน์อยู่ในตาราง

ตารางที่ 38. ชนิดข้อมูลของพารามิเตอร์

ชนิดข้อมูล SQL	C และ C++	CL	COBOL และ ILE COBOL
SMALLINT	short		PIC S9(4) BINARY

ตารางที่ 38. ชนิดข้อมูลของพารามิเตอร์ (ต่อ)

ชนิดข้อมูล SQL	C และ C++	CL	COBOL และ ILE COBOL
INTEGER	long		PIC S9(9) BINARY
BIGINT	long long		PIC S9(18) BINARY **หมายเหตุ:** สนับสนุน สำหรับ ILE COBOL เท่านั้น
DECIMAL(p,s)	decimal(p,s)	TYPE(*DEC) LEN(p s)	PIC S9(p-s)V9(s) PACKED-DECIMAL **หมายเหตุ:** ความแม่นยำ ต้องไม่เกิน 18
NUMERIC(p,s)			PIC S9(p-s)V9(s) DISPLAY SIGN LEADING SEPARATE **หมายเหตุ:** ความแม่นยำ ต้องไม่เกิน 18
DECFLOAT	_Decimal32, _Decimal64, _Decimal128 **หมายเหตุ:** สนับสนุน สำหรับ C เท่านั้น		
REAL หรือ FLOAT(p)	float		COMP-1 **หมายเหตุ:** สนับสนุนสำหรับ ILE COBOL เท่านั้น
DOUBLE PRECISION หรือ FLOAT หรือ FLOAT(p)	double		COMP-2 **หมายเหตุ:** สนับสนุนสำหรับ ILE COBOL เท่านั้น
CHARACTER(n)	char ... [n+1]	TYPE(*CHAR) LEN(n)	PIC X(n)
VARCHAR(n)	char ... [n+1]		สตริงอักขระความยาวแปรผัน
VARCHAR(n) FOR BIT DATA	รูปแบบโครงสร้าง VARCHAR		สตริงอักขระความยาวแปรผัน
CLOB	รูปแบบโครงสร้าง CLOB		รูปแบบโครงสร้าง CLOB **หมายเหตุ:** สนับสนุนสำหรับ ILE COBOL เท่านั้น
GRAPHIC(n)	wchar_t ... [n+1]		PIC G(n) DISPLAY-1 or PIC N(n) **หมายเหตุ:** สนับสนุนสำหรับ ILE COBOL เท่านั้น

ตารางที่ 38. ชนิดข้อมูลของพารามิเตอร์ (ต่อ)

ชนิดข้อมูล SQL	C และ C++	CL	COBOL และ ILE COBOL
VARGRAPHIC(n)	รูปแบบโครงสร้าง VARGRAPHIC		กราฟิกสตริงความยาวแปรผัน **หมายเหตุ:** สนับสนุนสำหรับ ILE COBOL เท่านั้น
DBCLOB	รูปแบบโครงสร้าง DBCLOB		รูปแบบโครงสร้าง DBCLOB **หมายเหตุ:** สนับสนุนสำหรับ ILE COBOL เท่านั้น
BINARY	รูปแบบโครงสร้าง BINARY		รูปแบบโครงสร้าง BINARY
VARBINARY	รูปแบบโครงสร้าง VARBINARY		รูปแบบโครงสร้าง VARBINARY
BLOB	รูปแบบโครงสร้าง BLOB		รูปแบบโครงสร้าง BLOB **หมายเหตุ:** สนับสนุนสำหรับ ILE COBOL เท่านั้น
DATE	char ... [11]	TYPE(*CHAR) LEN(10)	PIC X(10) **หมายเหตุ:** สำหรับ ILE COBOL เท่านั้น, FORMAT DATE
TIME	char ... [9]	TYPE(*CHAR) LEN(8)	PIC X(8) **หมายเหตุ:** สำหรับ ILE COBOL เท่านั้น, FORMAT TIME
TIMESTAMP	char ... [27]	TYPE(*CHAR) LEN(26)	PIC X(26) **หมายเหตุ:** สำหรับ ILE COBOL เท่านั้น, FORMAT TIMESTAMP
ROWID	รูปแบบโครงสร้าง ROWID		รูปแบบโครงสร้าง ROWID
DataLink			
ตัวแปรตัวบ่งชี้	short		PIC S9(4) BINARY

ตารางที่ 39. ชนิดข้อมูลของพารามิเตอร์

ชนิดข้อมูล SQL	ลักษณะพารามิเตอร์ Java™ JAVA	ลักษณะพารามิเตอร์ Java DB2GENERAL	PL/I
SMALLINT	short	short	FIXED BIN(15)
INTEGER	int	int	FIXED BIN(31)

ตารางที่ 39. ชนิดข้อมูลของพารามิเตอร์ (ต่อ)

ชนิดข้อมูล SQL	ลักษณะพารามิเตอร์ Java™ JAVA	ลักษณะพารามิเตอร์ Java DB2GENERAL	PL/I
BIGINT	long	long	
DECIMAL(p,s)	BigDecimal	BigDecimal	FIXED DEC(p,s)
NUMERIC(p,s)	BigDecimal	BigDecimal	
DECFLOAT			
REAL หรือ FLOAT(p)	float	float	FLOAT BIN(p)
DOUBLE PRECISION หรือ FLOAT หรือ FLOAT(p)	double	double	FLOAT BIN(p)
CHARACTER(n)	String	String	CHAR(n)
VARCHAR(n)	String	String	CHAR(n) VAR
VARCHAR(n) FOR BIT DATA	byte[ ]	com.ibm.db2.app.Blob	CHAR(n) VAR
CLOB	java.sql.Clob	com.ibm.db2.app.Clob	รูปแบบโครงสร้าง CLOB
GRAPHIC(n)	String	String	
VARGRAPHIC(n)	String	String	
DBCLOB	java.sql.Clob	com.ibm.db2.app.Clob	รูปแบบโครงสร้าง DBCLOB
BINARY	byte[ ]	com.ibm.db2.app.Blob	รูปแบบโครงสร้าง BINARY
VARBINARY	byte[ ]	com.ibm.db2.app.Blob	รูปแบบโครงสร้าง VARBINARY
BLOB	java.sql.Blob	com.ibm.db2.app.Blob	รูปแบบโครงสร้าง BLOB
DATE	Date	String	CHAR(10)
TIME	Time	String	CHAR(8)
TIMESTAMP	Timestamp	String	CHAR(26)
ROWID	byte[]	com.ibm.db2.app.Blob	รูปแบบโครงสร้าง ROWID
DataLink			
ตัวแปรตัวบ่งชี้			FIXED BIN(15)

ตารางที่ 40. ชนิดข้อมูลของพารามิเตอร์

ชนิดข้อมูล SQL	REXX	RPG	ILE RPG
SMALLINT		โครงสร้างข้อมูลที่ประกอบด้วย ฟิลด์ย่อยหนึ่งฟิลด์. ***B*** ในตำแหน่ง 43, ความยาวต้องเป็น 2, และ 0 ในตำแหน่ง 52 ของค่ากำหนดฟิลด์ย่อย.	ค่ากำหนดข้อมูล. ***B*** ใน ตำแหน่ง 40, ความยาวต้องเท่ากับ <=4, และ 00 ในตำแหน่ง 41–42 ของค่ากำหนดฟิลด์ย่อย. หรือ ค่ากำหนดข้อมูล. ***I*** ในตำแหน่ง 40, ความยาวต้องเท่ากับ 5, และ 00 ในตำแหน่ง 41–42 ของ ค่ากำหนดฟิลด์ย่อย.
INTEGER	สตริงตัวเลขที่ไม่มีทศนิยม (และเครื่องหมายนำหน้าเสริม)	โครงสร้างข้อมูลที่ประกอบด้วย ฟิลด์ย่อยหนึ่งฟิลด์. ***B*** ในตำแหน่ง 43, ความยาวต้อง เท่ากับ 4, และ 0 ในตำแหน่ง 52 ของค่ากำหนดฟิลด์ย่อย.	ค่ากำหนดข้อมูล. ***B*** ในตำแหน่ง 40, ความยาวต้องเป็น <=09 และ >=05, และ 00 ในตำแหน่ง 41–42 ของค่ากำหนดฟิลด์ย่อย. หรือ ค่ากำหนดข้อมูล. ***I*** ในตำแหน่ง 40, ความยาวต้องเท่ากับ 10, และ 00 ในตำแหน่ง 41–42 ของค่ากำหนดฟิลด์ย่อย.
BIGINT			ค่ากำหนดข้อมูล. ***I*** ในตำแหน่ง 40, ความยาวต้องเท่ากับ 20, และ 00 ในตำแหน่ง 41–42 ของ ค่ากำหนดฟิลด์ย่อย.
DECIMAL(p,s)	สตริงตัวเลขที่มีทศนิยม (และเครื่องหมายนำหน้าเสริม)	โครงสร้างข้อมูลที่ประกอบด้วย ฟิลด์ย่อยหนึ่งฟิลด์. ***P***ในตำแหน่ง 43 และ 0 ถึง 9 ในตำแหน่ง 52 ของค่ากำหนดฟิลด์ย่อย. หรือฟิลด์ใส่ข้อมูลตัวเลขหรือฟิลด์ผลการคำนวณ.	ค่ากำหนดข้อมูล. ***P*** ในตำแหน่ง 40 และ 00 ถึง 31 ในตำแหน่ง 41–42 ของค่ากำหนดฟิลด์ย่อย.
NUMERIC(p,s)		โครงสร้างข้อมูลที่ประกอบด้วย ฟิลด์ย่อยหนึ่งฟิลด์. ***Blank*** ในตำแหน่ง 43 และ 0 ถึง 9 ในตำแหน่ง 52 ของค่ากำหนดฟิลด์ย่อย.	ค่ากำหนดข้อมูล. ***S*** ในตำแหน่ง 40, หรือ ***Blank*** ในตำแหน่ง 40 และ 00 ถึง 31 ในตำแหน่ง 41–42 ของค่ากำหนดฟิลด์ย่อย.
DECFLOAT			
REAL หรือ FLOAT(p)	สตริงที่มีดิจิต, ตามด้วย E, (และตามด้วยเครื่องหมายเสริม), ตามด้วยดิจิต		ค่ากำหนดข้อมูล. ***F*** ในตำแหน่ง 40, ความยาวต้องเท่ากับ 4.

ตารางที่ 40. ชนิดข้อมูลของพารามิเตอร์ (ต่อ)

ชนิดข้อมูล SQL	REXX	RPG	ILE RPG
DOUBLE PRECISION หรือ FLOAT หรือ FLOAT(p)	สตริงที่มีดิจิต, ตามด้วย E, (และตามด้วยเครื่องหมายเสริม), ตามด้วยดิจิต		ค่ากำหนดข้อมูล. ***F*** ในตำแหน่ง 40, ความยาวต้องเท่ากับ 8.
CHARACTER(n)	สตริงที่มีอักขระ n ตัว ที่อยู่ภายในเครื่องหมายย่อสองตัว	ฟิลด์โครงสร้างข้อมูลที่ไม่มีฟิลด์ย่อย หรือโครงสร้างข้อมูลที่ประกอบด้วยฟิลด์ย่อยหนึ่งฟิลด์. ***Blank*** ในตำแหน่ง 43 และ 52 ของค่ากำหนดฟิลด์ย่อย. หรือฟิลด์ใส่ข้อมูลอักขระ A หรือฟิลด์ผลการคำนวณ.	ค่ากำหนดข้อมูล. ***A*** ในตำแหน่ง 40, หรือ***Blank*** ในตำแหน่ง 40 และ 41-42 ของค่ากำหนดฟิลด์ย่อย.
VARCHAR(n)	สตริงที่มีอักขระ n ตัว ที่อยู่ภายในเครื่องหมายย่อสองตัว		ค่ากำหนดข้อมูล. ***A*** ในตำแหน่ง 40, หรือ ***Blank*** ในตำแหน่ง 40 และ 41-42 ของค่ากำหนดฟิลด์ย่อยและคีย์เวิร์ด VARYING ในตำแหน่ง 44-80.
VARCHAR(n) FOR BIT DATA	สตริงที่มีอักขระ n ตัว ที่อยู่ภายในเครื่องหมายย่อสองตัว		ค่ากำหนดข้อมูล. ***A*** ในตำแหน่ง 40, หรือ ***Blank*** ในตำแหน่ง 40 และ 41-42 ของค่ากำหนดฟิลด์ย่อยและคีย์เวิร์ด VARYING ในตำแหน่ง 44-80.
CLOB			รูปแบบโครงสร้าง CLOB
GRAPHIC(n)	สตริงขึ้นต้นด้วย G', ตามด้วยอักขระ n ที่มีไบต์สองเท่า, ตามด้วย '		ค่ากำหนดข้อมูล. ***G*** ในตำแหน่ง 40 ของค่ากำหนดฟิลด์ย่อย.
VARGRAPHIC(n)	สตริงขึ้นต้นด้วย G', ตามด้วยอักขระ n ที่มีไบต์สองเท่า, ตามด้วย '		ค่ากำหนดข้อมูล. ***G*** ในตำแหน่ง 40 ของค่ากำหนดฟิลด์ย่อยและคีย์เวิร์ด VARYING ในตำแหน่ง 44-80.
DBCLOB			รูปแบบโครงสร้าง DBCLOB
BINARY			รูปแบบโครงสร้าง BINARY
VARBINARY			รูปแบบโครงสร้าง VARBINARY
BLOB			รูปแบบโครงสร้าง BLOB
DATE	สตริงที่มีอักขระ 10 ตัว ที่อยู่ภายในเครื่องหมายย่อสองตัว	ฟิลด์โครงสร้างข้อมูลที่ไม่มีฟิลด์ย่อย หรือโครงสร้างข้อมูลที่ประกอบด้วยฟิลด์ย่อยหนึ่งฟิลด์. ***Blank*** ในตำแหน่ง 43 และ 52 ของค่ากำหนดฟิลด์ย่อย. ความยาวเท่ากับ 10. หรือฟิลด์ใส่ข้อมูลอักขระ A หรือฟิลด์ผลการคำนวณ.	ค่ากำหนดข้อมูล. ***D*** ในตำแหน่ง 40 ของค่ากำหนดฟิลด์ย่อย. DATFMT (*ISO) ในตำแหน่ง 44-80.

ตารางที่ 40. ชนิดข้อมูลของพารามิเตอร์ (ต่อ)

ชนิดข้อมูล SQL	REXX	RPG	ILE RPG
TIME	สตริงที่มีอักขระ 8 ตัว ที่อยู่ภายในเครื่องหมายย่อสองตัว	ฟิลด์โครงสร้างข้อมูลที่ไม่มีฟิลด์ย่อย หรือโครงสร้างข้อมูลที่ประกอบด้วยฟิลด์ย่อยหนึ่งฟิลด์. ***Blank*** ในตำแหน่ง 43 และ 52 ของค่ากำหนดฟิลด์ย่อย. ความยาวเท่ากับ 8. หรือฟิลด์ใส่ข้อมูลอักขระ A หรือฟิลด์ผลการคำนวณ.	ค่ากำหนดข้อมูล. ***T*** ในตำแหน่ง 40 ของค่ากำหนดฟิลด์ย่อย. TIMFMT (*ISO) ในตำแหน่ง 44–80.
TIMESTAMP	สตริงที่มีอักขระ 26 ตัว ที่อยู่ภายในเครื่องหมายย่อสองตัว	ฟิลด์โครงสร้างข้อมูลที่ไม่มีฟิลด์ย่อย หรือโครงสร้างข้อมูลที่ประกอบด้วยฟิลด์ย่อยหนึ่งฟิลด์. ***Blank*** ในตำแหน่ง 43 และ 52 ของค่ากำหนดฟิลด์ย่อย. ความยาวเท่ากับ 26. หรือฟิลด์ใส่ข้อมูลอักขระ A หรือฟิลด์ผลการคำนวณ.	ค่ากำหนดข้อมูล. ***Z*** ในตำแหน่ง 40 ของค่ากำหนดฟิลด์ย่อย.
ROWID			รูปแบบโครงสร้าง ROWID
DataLink			
ตัวแปรตัวบ่งชี้	สตริงตัวเลขที่ไม่มีทศนิยม (และเครื่องหมายนำหน้าเสริม)	โครงสร้างข้อมูลที่ประกอบด้วย ฟิลด์ย่อยหนึ่งฟิลด์. ***B*** ในตำแหน่ง 43, ความยาวต้องเป็น 2, และ 0 ในตำแหน่ง 52 ของค่ากำหนดฟิลด์ย่อย.	ค่ากำหนดข้อมูล. ***B*** ใน ตำแหน่ง 40, ความยาวต้องเท่ากับ <=4, และ 00 ในตำแหน่ง 41–42 ของค่ากำหนดฟิลด์ย่อย.

หลักการที่เกี่ยวข้อง

Embedded SQL programming

Java SQL routines

ตัวแปรตัวบ่งชี้และโพรซีเดอร์ที่เก็บไว้

สามารถใช้ตัวแปรโฮสต์ที่มีตัวแปรตัวบ่งชี้ ด้วยคำสั่ง CALL เพื่อส่งผ่านข้อมูลเพิ่มเติมไปยังและจาก โพรซีเดอร์

เพื่อแสดงให้เห็นว่าตัวแปรโฮสต์ที่เกี่ยวข้อง ประกอบด้วยค่า null ตัวแปรตัวบ่งชี้จะถูกตั้งเป็นค่าลบ นั่นคือ –1, –2, –3, –4 หรือ –6 ข้อความ CALL ที่มีตัวแปรตัวบ่งชี้ถูกประมวลผลดังนี้:

- ถ้าตัวบ่งชี้เป็นค่าลบ จะมีการส่งค่าดีฟอลต์ สำหรับตัวแปรโฮสต์ที่เกี่ยวข้องบนคำสั่ง CALL และตัวแปรตัวบ่งชี้ถูกส่งผ่านเหมือนเดิม
- ถ้าตัวแปรตัวบ่งชี้เป็นค่าลบ ตัวแปรโฮสต์และตัวแปรตัวบ่งชี้จะถูกส่งผ่านเหมือนเดิม

เมื่อมีการเรียกโพรซีเดอร์ SQL หรือโพรซีเดอร์ภายนอก ที่ถูกคอมไพล์โดยไม่มีอ็อพชัน *EXTIND จะไม่สามารถส่งผ่านค่าตัวบ่งชี้ที่ขยาย –5 และ –7 ได้ จากนั้นจะมีการส่งข้อผิดพลาดบน คำสั่ง CALL เมื่อมีการเรียกโพรซีเดอร์ภายนอกที่ถูกคอมไพล์ด้วยอ็อพชัน *EXTIND จะสามารถส่งผ่านค่าตัวบ่งชี้ที่ขยายได้

กฎการประมวลผลเหล่านี้เหมือนกับกฎสำหรับอินพุตพารามิเตอร์ที่ไปยังโพรซีเดอร์ และเอาต์พุตพารามิเตอร์ที่ส่งคืนจากโพรซีเดอร์ เมื่อตัวแปรตัวบ่งชี้ถูกใช้งาน วิธีการที่ถูกต้องในการโค้ดคือ การตรวจสอบค่าตัวแปรตัวบ่งชี้ก่อน ที่จะใช้ตัวแปรโฮสต์ที่เกี่ยวข้อง

ตัวอย่างต่อไปนี้แสดงถึงการจัดการตัวแปรตัวบ่งชี้ในข้อความ CALL. โปรดสังเกตว่าตรรกะจะตรวจสอบค่าตัวแปร ตัวบ่งชี้ก่อนที่จะใช้ตัวแปรที่เกี่ยวข้อง. และสังเกตเพิ่มเติมถึงวิธีการที่ตัวแปรตัวบ่งชี้ ถูกส่งผ่านเข้าไปยังโพรซีเดอร์ PROC1 (ในฐานะที่เป็นอากิวเมนต์ที่สามซึ่งประกอบด้วยอะเรย์ ของค่าขนาด 2 ไบต์)

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

สมมติว่าโพรซีเดอร์ถูกกำหนดดังนี้ โปรแกรม ILE RPG ถูกคอมไพล์โดยไม่อนุญาตให้ใช้ตัวบ่งชี้ที่ขยาย

```
 CREATE PROCEDURE PROC1
    (INOUT DECIMALOUT DECIMAL(7,2), INOUT DECOUT2 DECIMAL(7,2))
    EXTERNAL NAME LIB1.PROC1 LANGUAGE RPGLE
    GENERAL WITH NULLS)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
โปรแกรม CRPG
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    D INOUT1          S              7P 2
    D INOUT1IND       S              4B 0
    D INOUT2          S              7P 2
    D INOUT2IND       S              4B 0
    C                   EVAL      INOUT1 = 1
    C                   EVAL      INOUT1IND  = 0
    C                   EVAL      INOUT2 = 1
    C                   EVAL      INOUT2IND = -2
    C/EXEC SQL CALL PROC1 (:INOUT1 :INOUT1IND , :INOUT2
    C+                         :INOUT2IND)
    C/END-EXEC
    C                   EVAL      INOUT1 = 1
    C                   EVAL      INOUT1IND  = 0
    C                   EVAL      INOUT2 = 1
    C                   EVAL      INOUT2IND = -2
    C/EXEC SQL CALL PROC1 (:INOUT1 :INOUT1IND , :INOUT2
    C+                         :INOUT2IND)
    C/END-EXEC
    C     INOUT1IND     IFLT      0
    C*                   :
    C*                  HANDLE NULL INDICATOR
    C*                   :
    C                   ELSE
    C*                   :
    C*                  INOUT1 CONTAINS VALID DATA
    C*                   :
    C                   ENDIF
    C*                   :
    C*                  HANDLE ALL OTHER PARAMETERS
    C*                  IN A SIMILAR FASHION
    C*                   :
    C                   RETURN
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
สิ้นสุด PROGRAM CRPG
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
โปรแกรม PROC1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    D INOUTP          S              7P 2
    D INOUTP2         S              7P 2
    D NULLARRAY       S              4B 0 DIM(2)
    C     *ENTRY        PLIST
    C                   PARM                    INOUTP
    C                   PARM                    INOUTP2
    C                   PARM                    NULLARRAY
    C     NULLARRAY(1)  IFLT      0
    C*                   :
    C*                  CODE FOR INOUTP DOES NOT CONTAIN MEANINGFUL DATA
    C*                   :
    C                   ELSE
    C*                   :
    C*                  CODE FOR INOUTP CONTAINS MEANINGFUL DATA
    C*                   :
    C                   ENDIF
    C*                  PROCESS ALL REMAINING VARIABLES
    C*
    C*                  BEFORE RETURNING, SET OUTPUT VALUE FOR FIRST
    C*                  PARAMETER AND SET THE INDICATOR TO A NON-NEGATIVE
    C*                  VALUE SO THAT THE DATA IS RETURNED TO THE CALLING
    C*                  PROGRAM
    C*
    C                   EVAL      INOUTP2 = 20.5
    C                   EVAL      NULLARRAY(2) = 0
    C*
    C*                  INDICATE THAT THE SECOND PARAMETER IS TO CONTAIN
    C*                  THE NULL VALUE UPON RETURN. THERE IS NO POINT
    C*                  IN SETTING THE VALUE IN INOUTP SINCE IT WON'T BE
    C*                  PASSED BACK TO THE CALLER.
    C                   EVAL      NULLARRAY(1) = -1
    C                   RETURN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
สิ้นสุด PROGRAM PROC1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

การย้อนกลับสถานะที่สมบูรณ์ไปยังโปรแกรมการเรียก

SQL และโพรซีเดอร์ภายนอกส่งคืนข้อมูลสถานะ ไปยังโปรแกรมการเรียกด้วยวิธีการต่างๆ

สำหรับ SQL โพรซีเดอร์, ข้อผิดพลาดใดๆ ที่ไม่ได้รับการจัดการในโพรซีเดอร์จะถูกส่งคืนมาที่ตัวเรียกใน SQLCA. สามารถ SIGNAL และ RESIGNAL control statement เพื่อส่งข้อมูลข้อผิดพลาดได้เช่นกัน.

สำหรับโพรซีเดอร์ภายนอก, มีสองวิธีการในการส่งข้อมูลสถานะกลับ. วิธีการที่หนึ่งในการส่งคืนสถานะไปที่โปรแกรม SQL ที่ส่งข้อความ CALL คือ ให้โค้ดพารามิเตอร์ประเภท INOUT พิเศษและเซ็ตไว้ก่อนที่จะกลับคืนจากโพรซีเดอร์ดังกล่าว. เมื่อโพรซีเดอร์ที่เรียกคือโปรแกรมที่มีอยู่แล้ว, วิธีการข้างต้นย่อมเป็นไปไม่ได้.

วิธีการที่สองในการส่งคืนสถานะไปที่โปรแกรม SQL ที่ส่งข้อความ CALL คือให้ส่ง escape message ไปยังโปรแกรมการเรียก (โปรแกรมระบบปฏิบัติการ QSQCALL) ที่เรียกโพรซีเดอร์. โปรแกรมการเรียกที่เรียกใช้งานโพรซีเดอร์คือ QSQCALL. แต่ละภาษามีวิธีการสำหรับเงื่อนไขการส่งสัญญาณ และการส่งข้อความ. โปรดดูที่การอ้างอิงแต่ละภาษาเพื่อกำหนดวิธีการที่เหมาะสมในการส่งสัญญาณข้อความ. เมื่อมีการส่งสัญญาณข้อความ, QSQCALL จะแปลงข้อผิดพลาดเป็น SQLCODE/SQLSTATE -443/38501.

สิ่งอ้างอิงที่เกี่ยวข้อง

SQL control statements

การใช้ฟังก์ชันแบบผู้ใช้กำหนดเอง

ในการเขียนแอ็พพลิเคชัน SQL คุณสามารถเลือก ปฏิบัติการหรือดำเนินการบางอย่างในแบบฟังก์ชันที่ผู้ใช้กำหนดเอง (UDF) หรือแบบรูทีนย่อย ในแอ็พพลิเคชันของคุณ ถึงแม้ว่ามันอาจจะดูง่ายกว่าในการเลือกการดำเนินการใหม่แบบ รูทีนย่อยในแอ็พพลิเคชันของคุณ คุณอาจต้องพิจารณาถึงประโยชน์ของการใช้งาน UDF แทน

ตัวอย่างเช่น, ถ้าการดำเนินการใหม่เป็นสิ่งที่ผู้ใช้งานหรือโปรแกรมอื่นๆ สามารถได้รับประโยชน์, รูปแบบ UDF สามารถช่วยในการนำมาใช้งานได้อีก. นอกจากนี้, เราสามารถเรียกฟังก์ชันได้โดยตรงใน SQL ในที่ใดก็ตามที่สามารถใช้นิพจน์ได้. ฐานข้อมูลจะดูแลชนิดข้อมูลทั้งหลายของฟังก์ชันอากิวเมนต์ให้โดยอัตโนมัติ. ตัวอย่างเช่น, จาก DECIMAL ไปเป็น DOUBLE, ฐานข้อมูลอนุญาตให้ฟังก์ชันของคุณใช้ชนิดข้อมูลที่แตกต่างไปได้, แต่ต้องทำงานร่วมกันได้.

ในบางกรณี, การเรียก UDF โดยตรงจากเอ็นจินฐานข้อมูลแทนที่จะเรียกจากแอ็พพลิเคชันของคุณสามารถทำให้ประสิทธิภาพดีขึ้นอย่างมาก. คุณอาจจะสังเกตุเห็นประสิทธิภาพที่เพิ่มขึ้นได้ในกรณีที่ฟังก์ชันถูกใช้ในการตรวจสอบข้อมูลสำหรับการประมวลผลครั้งต่อไป. กรณีนี้จะเกิดขึ้นก็ต่อเมื่อฟังก์ชันถูกใช้ในกระบวนการเลือกแถว.

พิจารณาสถานการณ์ง่ายๆ เมื่อคุณต้องการดำเนินการกับบางข้อมูล. คุณอาจเจอเงื่อนไขการเลือกบางอย่างที่สามารถแสดงเป็นแบบฟังก์ชัน SELECTION_CRITERIA() ได้. แอ็พพลิเคชันของคุณสามารถเรียกใช้คำสั่ง select ดังต่อไปนี้:

```
SELECT A, B, C FROM T
```

เมื่อได้รับข้อมูลแต่ละแถวแล้ว, จะเรียกใช้ฟังก์ชัน SELECTION_CRITERIA กับข้อมูลเหล่านั้นเพื่อตัดสินใจว่าข้อมูลนั้นเป็นที่สนใจในการประมวลผลข้อมูลต่อไปหรือไม่. นั่นคือ, ทุกแถวของตาราง T ต้องถูกส่งกลับไปยังแอ็พพลิเคชัน. แต่, ถ้า SELECTION_CRITERIA() ถูกสร้างเป็นแบบ UDF แล้ว, แอ็พพลิเคชันของคุณสามารถเรียกใช้คำสั่งดังต่อไปนี้ได้:

```
SELECT C FROM T WHERE SELECTION_CRITERIA(A,B)=1
```

ในกรณีนี้, มีเพียงแถวที่อยู่ในคอลัมน์เดียวที่สนใจเท่านั้นที่จะถูกส่งข้ามไปมาระหว่างอินเตอร์เฟสของแอ็พพลิเคชันและฐานข้อมูล.

กรณีอื่นๆ ที่ UDF สามารถช่วยเพิ่มประสิทธิภาพก็คือเมื่อต้องทำงานกับอ็อบเจ็กต์ขนาดใหญ่ (LOB) สมมติว่าคุณมีฟังก์ชันที่ดึงข้อมูลจากค่าของ LOB คุณสามารถทำการดึงข้อมูลนี้บนเซิร์ฟเวอร์ฐานข้อมูลแล้วส่งผ่านเฉพาะข้อมูลที่ดึงออกมาไปยังแอ็พพลิเคชันได้. วิธีนี้จะมีประสิทธิภาพมากกว่าการส่งผ่านค่า LOB ทั้งหมดกลับไปยังแอ็พพลิเคชันแล้วค่อยทำการดึงข้อมูลออกมา. ค่าประสิทธิภาพของการจัดแพ็กเกจฟังก์ชันนี้เป็นแบบ UDF อาจจะมีค่าสูงมาก, ซึ่งจะขึ้นอยู่กับสถานการณ์เฉพาะ.

หลักการที่เกี่ยวข้อง

"ฟังก์ชันแบบผู้ใช้กำหนด (User-defined functions)" ในหน้า 13

ฟังก์ชันแบบผู้ใช้กำหนด คือโปรแกรมที่อาจถูกเรียกทำงาน ได้เหมือนกับฟังก์ชันในตัวอื่นๆ

แนวคิดของ UDF

ฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) เป็นฟังก์ชัน ที่ถูกกำหนดให้กับระบบฐานข้อมูล DB2 โดยใช้คำสั่ง CREATE FUNCTION และ สามารถอ้างถึงได้ในคำสั่ง SQL ทั้งนี้ UDF อาจเป็นฟังก์ชันภายนอก หรือฟังก์ชัน SQL

ชนิดของฟังก์ชัน

มีชนิดของฟังก์ชันอยู่หลายชนิด:

- *ในตัว(Built-in).* คือฟังก์ชันที่ถูกจัดเตรียมให้และมาพร้อมกับฐานข้อมูล. ตัวอย่างคือ SUBSTR().
- *ระบบสร้างให้(System-generated).* ฟังก์ชันนี้จะถูกสร้างโดยตรงโดยเอ็นจินฐานข้อมูลเมื่อ DISTINCT TYPE ถูกสร้างขึ้นมา. ฟังก์ชันนี้จัดเตรียมตัวดำเนินการเปลี่ยนชนิดข้อมูลระหว่าง DISTINCT TYPE และชนิดพื้นฐานของ DISTINCT TYPE นั้น.
- *ผู้ใช้กำหนดเอง(User-defined).* ฟังก์ชันนี้ถูกสร้างโดยผู้ใช้แล้วจึงลงทะเบียนไปที่ฐานข้อมูล.

นอกจากนี้ แต่ละฟังก์ชันสามารถถูกจัดหมวดหมู่เป็นฟังก์ชันแบบ *Scalar*, ฟังก์ชันแบบ *Column*, หรือฟังก์ชันแบบ *Table*

ฟังก์ชันแบบ Scalar จะคืนค่าผลลัพธ์เดียวในแต่ละครั้งที่เรียกใช้. ตัวอย่างเช่น, ฟังก์ชันในตัว SUBSTR() คือฟังก์ชันแบบ Scalar, ดังเช่นฟังก์ชันในตัวหลายๆฟังก์ชัน. ฟังก์ชันที่ระบบสร้างให้(System-generated function)จะเป็นฟังก์ชันแบบ Scalar เสมอ. Scalar UDFs สามารถเป็นได้ทั้งส่วนภานนอก (โค้ดในภาษาโปรแกรมเช่น C), เขียนใน SQL, หรือ ในต้นฉบับ (การใช้ในการนำไปปฏิบัติของฟังก์ชันที่มีอยู่แล้ว).

ฟังก์ชันการรวม จะรับชุดของค่าที่คล้ายกัน (คอลัมน์ของข้อมูล) และคืนค่าผลลัพธ์เดียวจากชุดของค่าเหล่านั้น ฟังก์ชันในตัวบางฟังก์ชันก็เป็นฟังก์ชันการรวม ตัวอย่างของฟังก์ชันการรวมก็คือ ฟังก์ชันในตัว AVG() UDF ภายนอกไม่สามารถกำหนดให้เป็นฟังก์ชันการรวมได้ อย่างไรก็ตาม UDF ต้นฉบับจะถูกกำหนดให้เป็นฟังก์ชันการรวม ถ้าฟังก์ชันต้นฉบับเป็นฟังก์ชันการรวมแบบในตัว อย่างหลังสุดจะใช้ประโยชน์ได้มากสำหรับ Distinct Types. ตัวอย่างเช่น ถ้ามี distinct type ชื่อ SHOESIZE อยู่และถูกกำหนดค่าพื้นฐาน เป็น INTEGER คุณสามารถกำหนด UDF, AVG(SHOESIZE) ให้เป็น ฟังก์ชันการรวมต้นฉบับบนฟังก์ชันการรวมแบบในตัวที่มีอยู่ นั่นคือ AVG(INTEGER)

ฟังก์ชันแบบ Table จะส่งคืนค่าตารางให้กับคำสั่ง SQL ที่อ้างอิงถึงฟังก์ชันนั้น. มันต้องถูกอ้างอิงในอนุประโยค FROM ของ SELECT. ฟังก์ชันตารางสามารถ นำมาใช้เพิ่มความสามารถในการประมวลผลภาษา SQL กับข้อมูลที่ไม่ได้เป็นข้อมูลชนิด DB2 , หรือเพื่อแปลงข้อมูล นั้นไปเป็นรูปแบบตาราง DB2 . ซึ่งสามารถ, ยกตัวอย่างเช่น, ดึงไฟล์ และแปลงเป็นรูปแบบตาราง, ข้อมูลตัวอย่างจากใน World Wide Web และนำมาจัดเรียงเป็นตาราง, หรือเข้าถึงฐานข้อมูล Lotus Notes® และส่งกลับข้อมูลเกี่ยวกับข้อความอีเมล, เช่น วันที่, ผู้ส่ง, และเนื้อหาของข้อความนั้น. ข้อมูลเหล่านี้สามารถเชื่อมกับตารางอื่นๆในฐานข้อมูลได้. ฟังก์ชันแบบ Table สามารถถูกกำหนดให้เป็นแบบฟังก์ชันภายนอกหรือฟังก์ชัน SQL ได้; แต่ว่าจะไม่สามารถถูกกำหนดให้เป็นฟังก์ชันต้นฉบับได้.

ชื่อเต็มของฟังก์ชัน

ชื่อเต็มของฟังก์ชันที่ใช้การตั้งชื่อของ *SQL คือ <schema-name>.<function-name>.

ชื่อเต็มของฟังก์ชันในการตั้งชื่อของ *SYS คือ <schema-name>/<function-name>. ชื่อฟังก์ชันไม่สามารถครบตามเกณฑ์ถ้าใช้การตั้งชื่อโดย *SYS ในคำสั่ง DML.

คุณสามารถใช้ชื่อเต็มนี้ได้ในทุกที่ที่คุณอ้างอิงถึงฟังก์ชัน. ตัวอย่างเช่น:

```
QGPL.SNOWBLOWER_SIZE    SMITH.FOO    QSYS2.SUBSTR    QSYS2.FLOOR
```

อย่างไรก็ตาม, คุณยังอาจละเว้น <schema-name>., ในบางกรณีที่, DB2 ต้องพิจารณาว่า ฟังก์ชันใดที่คุณกำลังอ้างอิงอยู่. ตัวอย่างเช่น:

```
SNOWBLOWER_SIZE    FOO    SUBSTR    FLOOR
```

พาธ

แนวคิดเกี่ยวกับ*พาธ* จะมุ่งเน้นไปที่ความชัดเจนของ DB2 ในการอ้างอิง*ที่ไม่แน่นอน* ซึ่งเกิดขึ้นเมื่อไม่ได้มีการระบุชื่อแบบแผนที่ชัดเจน. พาธ เป็นลำดับรายการของรูปแบบชื่อ ที่ใช้สำหรับ การแก้ไขการอ้างอิงที่ไม่ชัดเจนกับ UDFs และ UDTs. ในกรณีที่การอ้างอิงฟังก์ชันไปตรงกับฟังก์ชันมากกว่าหนึ่งรูปแบบในพาธ, ลำดับของรูปแบบในพาธจะถูกใช้เพื่อแก้ปัญหาการตรงกันนี้. พาธถูกกำหนดขึ้นโดยตัวเลือก SQLPATH บนคำสั่งพรีคอมไพล์ สำหรับ static SQL. พาธถูกตั้งค่าโดยคำสั่ง SET PATH สำหรับ dynamic SQL. เมื่อคำสั่ง SQL แรกที่ทำงานใน activation group ซึ่งทำงานด้วยการตั้งชื่อของ SQL, พาธจะมีค่าดีฟอลต์ดังต่อไปนี้:

```
"QSYS","QSYS2","<ID>"
```

ค่านี้จะใช้ได้ใน SQL ทั้งแบบ static และ dynamic, โดยที่ <ID> คือ authorization ID ของคำสั่งปัจจุบัน.

เมื่อคำสั่ง SQL แรกที่ทำงานใน activation group ทำงานด้วยการตั้งชื่อของระบบ, ค่าดีฟอลต์คือ *LIBL.

ชื่อฟังก์ชันที่ถูก Overloaded

ชื่อฟังก์ชันสามารถ *Overloaded* ได้. การ Overloaded หมายความว่าหลายฟังก์ชัน, แม้ว่าจะอยู่ใน Schema เดียวกัน, สามารถใช้ชื่อเดียวกันได้. อย่างไรก็ตาม, สองฟังก์ชันไม่สามารถ, มี*signature*เหมือนกันได้. Signature ของฟังก์ชันสามารถกำหนดให้เป็นค่าชื่อฟังก์ชันที่ถูกตามเกณฑ์เชื่อมต่อเข้ากับชนิดข้อมูลของพารามิเตอร์ของฟังก์ชันทั้งหมดตามลำดับที่พารามิเตอร์เหล่านั้นถูกนิยาม.

การแก้ปัญหาของฟังก์ชัน

อัลกอริธึมการแก้ปัญหาฟังก์ชัน คือสิ่งที่นำมาใช้สำหรับการ Overloading และฟังก์ชันพาธ เพื่อเลือกชื่อที่*เหมาะที่สุด*สำหรับทุกการอ้างอิงฟังก์ชัน, โดยไม่สนใจว่าการอ้างอิงครบตามเกณฑ์หรือไม่. ทุกฟังก์ชัน, รวมถึงฟังก์ชันในตัวด้วย, จะถูกดำเนินการด้วยอัลกอริธึมการเลือกฟังก์ชัน. อัลกอริธึมการแก้ปัญหาฟังก์ชันไม่ได้ถูกใช้ในการแก้ปัญหาชนิดของฟังก์ชัน. ดังนั้นฟังก์ชันตารางอาจถูก resolve ได้ราวกับเป็นฟังก์ชัน*ที่เหมาะสมที่สุด*, แม้ว่า การใช้การอ้างอิงจะต้องการฟังก์ชัน scalar, หรือในทำนองกลับกัน.

ระยะเวลาที่ UDF รัน

เราเรียกใช้ UDFs จากการทำงานภายในข้อความ SQL , ซึ่งโดยปกติการปฏิบัติการ query ซึ่งมีศักยภาพในการทำงานกับจำนวนแถวนับพันแถวในตารางได้. ด้วยเหตุนี้, จึงจำเป็นต้องเรียกใช้ UDF จากฐานข้อมูลระดับต่ำ.

ผลของการถูกเรียกใช้งานจากระดับต่ำดังกล่าว, ทำให้รีซอร์สบางตัว (การล็อกและการยึด) ถูกระงับการทำงานชั่วคราว ณ. ขณะที่มีการเรียกใช้ UDF และในระหว่างการทำงานของ UDF. รีซอร์สเหล่านี้คือตัวล็อกหลักบนตารางและดรรชนีใดๆ ที่เกี่ยวข้องกับคำสั่ง SQL ซึ่งกำลังเรียก UDF ทำงาน. เนื่องจากรีซอร์สถูกระงับการทำงาน, UDF จึงไม่ควรดำเนินการที่อาจใช้ระยะเวลาเกินไป (หลายนาทีหรือหลายชั่วโมง). เนื่องมาจากลักษณะที่สำคัญในการระงับการทำงานของรีซอร์สเป็นเวลานาน, ฐานข้อมูลจะคอยสักระยะเวลาหนึ่งเพื่อให้ UDF ทำงานเสร็จสิ้นก่อน. ถ้า UDF ทำงานไม่เสร็จสิ้นภายในเวลาที่กำหนดให้, คำสั่ง SQL ที่กำลังเรียกใช้งาน UDF อยู่จะล้มเหลวลง.

ระยะเวลาที่ฐานข้อมูลรอ UDF ซึ่งเป็นค่าดีฟอลต์นั้นควรจะนานเกินกว่าเวลาที่ใช้จริงเพื่อให้ UDF แบบปกติรันให้เสร็จสิ้น. อย่างไรก็ตาม, ถ้าคุณมีการรัน UDF ที่ยาวนาน และต้องการเพิ่มเวลาในการรอ, คุณสามารถทำได้โดยการใช้ตัวเลือก UDF_TIME_OUT ในไฟล์สอบถาม INI. อย่างไรก็ตาม, โปรดจำว่า, ฐานข้อมูลจะใช้เวลาได้ไม่เกินข้อจำกัดสูงสุดที่กำหนดไว้, ไม่ว่าค่าที่ระบุไว้สำหรับ UDF_TIME_OUT จะเป็นเท่าไรก็ตาม.

เนื่องจากรีซอร์สถูกระงับการทำงานขณะที่รัน UDF, UDF จึงไม่ดำเนินการบนตารางหรือดรรชนีเดียวกันซึ่งถูกกำหนดให้กับคำสั่ง SQL ต้นฉบับหรือ, หากว่า UDF ดำเนินการไปแล้ว, UDF จะไม่ดำเนินการที่ขัดกับการดำเนินการที่กำลังปฏิบัติการอยู่ในคำสั่ง SQL. โดยเฉพาะอย่างยิ่ง, UDF จะไม่พยายามดำเนินการแทรก, อัพเดต, หรือลบการดำเนินการของแถวในตารางเหล่านั้น.

งานที่เกี่ยวข้อง

การควบคุมเคียวรี่แบบ dynamic ด้วยไฟล์อ็อพชันเคียวรี่ QAQQINI

การเขียน UDF เป็นฟังก์ชัน SQL

ฟังก์ชัน SQL เป็นฟังก์ชันแบบ ผู้ใช้กำหนดเอง (UDF) ที่คุณกำหนด เขียน และลงทะเบียน โดยใช้คำสั่ง CREATE FUNCTION

ฟังก์ชัน SQL เขียนด้วยภาษา SQL เท่านั้น และ definition ของฟังก์ชันนั้นถูกเก็บไว้ภายในคำสั่ง CREATE FUNCTION ขนาดใหญ่เพียงคำสั่งเดียว การสร้างฟังก์ชัน SQL จะทำให้ UDF ได้รับการลงทะเบียน, สร้างโค้ดที่ทำงานได้สำหรับฟังก์ชัน, และกำหนดรายละเอียดการส่งผ่านพารามิเตอร์ให้กับฐานข้อมูล.

ตัวอย่าง: SQL scalar UDF:

ตัวอย่างนี้แสดงฟังก์ชันสเกลาร์ที่ส่งคืน ระดับความสำคัญโดยดูจากวันที่

```
CREATE FUNCTION PRIORITY(indate DATE) RETURNS CHAR(7)
LANGUAGE SQL
BEGIN
RETURN(
      CASE   WHEN indate>CURRENT DATE-3 DAYS THEN 'HIGH'
         WHEN indate>CURRENT DATE-7 DAYS THEN 'MEDIUM'
         ELSE 'LOW'
      END
      );
END
```

ฟังก์ชันจะถูกเรียกทำงานเป็น:

```
SELECT ORDERNBR, PRIORITY(ORDERDUEDATE) FROM ORDERS
```

ตัวอย่าง: ตาราง SQL UDF:

ตัวอย่างนี้อธิบายถึงฟังก์ชันตารางที่ส่งคืน ข้อมูลโดยดูจากวันที่

```
CREATE FUNCTION PROJFUNC(indate DATE)
  RETURNS TABLE (PROJNO CHAR(6), ACTNO SMALLINT, ACTSTAFF DECIMAL(5,2),
           ACSTDATE DATE, ACENDATE DATE)
  LANGUAGE SQL
```

```
BEGIN
 RETURN SELECT * FROM PROJACT
   WHERE ACSTDATE<=indate;
END
```

ฟังก์ชันจะถูกเรียกทำงานเป็น:

```
SELECT * FROM TABLE(PROJFUNC(:datehv)) X
```

ฟังก์ชันตาราง SQL จะต้องมีคำสั่ง RETURN หนึ่งคำสั่งเท่านั้น.

การเขียน UDF ให้เป็นฟังก์ชันภายนอก

คุณสามารถเขียนโค้ดที่รันได้ของฟังก์ชันที่ ผู้ใช้กำหนดเอง (UDF) ในภาษาอื่นที่นอกเหนือจาก SQL

ขณะที่วิธีนี้จะยุ่งยากกว่าฟังก์ชันแบบ SQL , แต่วิธีนี้จะมีความยืดหยุ่นให้คุณได้ใช้ภาษาใดก็ได้ที่มีประสิทธิภาพที่สุดสำหรับคุณ. สามารถเก็บโค้ดที่ใช้งานได้ในโปรแกรม หรือเซอร์วิสโปรแกรม.

ฟังก์ชันแบบภายนอกสามารถถูกเขียนเป็น จาวา.

หลักการที่เกี่ยวข้อง

Java SQL routines

การลงทะเบียน UDF:

ฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) ต้องถูกลงทะเบียน ในฐานข้อมูลก่อนที่ฟังก์ชันจะรู้จัก และถูกใช้โดย SQL คุณ สามารถลงทะเบียน UDF โดยใช้คำสั่ง CREATE FUNCTION

คำสั่งอนุญาตให้คุณระบุภาษาและชื่อของโปรแกรมได้, รวมทั้งตัวเลือกอย่างเช่น DETERMINISTIC, ALLOW PARALLEL, และ RETURNS NULL ON NULL INPUT. ตัวเลือกเหล่านี้จะช่วยให้ฐานข้อมูลระบุเป้าหมายของฟังก์ชันได้ตรงขึ้น และช่วยระบุว่า วิธีการเรียกไปยังฐานข้อมูลสามารถทำการปรับปรุงประสิทธิภาพได้อย่างไร.

คุณควรเรจิสเตอร์ UDF แบบภายนอก หลังจากที่คุณได้เขียนและทดสอบโค้ดจริงได้ เสร็จสมบูรณ์. และเป็นไปได้ที่จะกำหนด UDF ก่อนที่จะเขียนจริง. อย่างไรก็ตาม, เพื่อหลีกเลี่ยงปัญหาเกี่ยวกับการรัน UDF ของคุณ, คุณควรเขียนและทดสอบให้ครอบคลุมก่อนที่จะทำการลงทะเบียน.

สิ่งอ้างอิงที่เกี่ยวข้อง

CREATE FUNCTION

ตัวอย่าง: การยกกำลัง:

สมมติว่า คุณได้เขียนฟังก์ชันภายนอกเพื่อ ทำการยกกำลังค่าตัวเลขทศนิยม และต้องการลงทะเบียน ในแบบแผน MATH

```
CREATE FUNCTION MATH.EXPON (DOUBLE, DOUBLE)
  RETURNS DOUBLE
  EXTERNAL NAME 'MYLIB/MYPGM(MYENTRY)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
```

```
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURNS NULL ON NULL INPUT
  ALLOW PARALLEL
```

ในตัวอย่างนี้, มีการระบุ RETURNS NULL ON NULL INPUT เนื่องจากคุณต้องการ ผลลัพธ์เป็น NULL ถ้าค่าอากิวเมนต์อันใดอันหนึ่งเป็น NULL. และเนื่องจากไม่มีเหตุผลใดที่ EXPON จะไม่สามารถทำงานคู่ขนานได้, ดังนั้นค่า ALLOW PARALLEL จึงถูกระบุ.

ตัวอย่าง: การค้นหาสตริง:

สมมติว่า คุณเขียน user-defined function (UDF) เพื่อมองหาสตริงที่ระบุ ซึ่งส่งผ่านเป็นอากิวเมนต์ ภายในค่า character large object (CLOB) ที่ถูกกำหนดไว้ซึ่งส่งผ่านเป็นอากิวเมนต์เช่นกัน UDF จะคืนค่า ตำแหน่งของสตริงภายใน CLOB ถ้าเจอสตริงนั้น หรือคืนค่าศูนย์ ถ้าหาไม่เจอ

มีการเขียนโปรแกรมภาษา C เพื่อให้ผลลัพธ์เป็น FLOAT กลับมา. สมมุติว่าคุณรู้ว่าเมื่อ UDF นี้ถูกใช้ใน SQL, มันจะคืนค่า INTEGER เสมอ. คุณสามารถสร้างฟังก์ชันดังต่อไปนี้:

```
CREATE FUNCTION FINDSTRING (CLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC FINDSTRING
  EXTERNAL NAME 'MYLIB/MYPGM(FINDSTR)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURNS NULL ON NULL INPUT
```

โปรดสังเกตุว่าประโยค CAST FROM ใช้ระบุว่าโปรแกรม UDF ได้ส่งกลับค่า FLOAT จริง, แต่คุณต้องการแปลงค่านี้ให้เป็น INTEGER ก่อนการส่งกลับค่า มาที่ข้อความ SQL ซึ่งเรียกใช้ UDFนั้น. ดังนั้น, คุณต้องเตรียมชื่อเฉพาะของคุณเอง สำหรับฟังก์ชัน. เนื่องจาก UDF ไม่ได้ถูกเขียนมาเพื่อจัดการค่า NULL ได้, คุณต้องใช้ RETURNS NULL ON NULL INPUT.

ตัวอย่าง: การค้นหาสตริง BLOB:

สมมติว่า คุณต้องการให้ฟังก์ชัน FINDSTRING ทำงาน บน binary large object (BLOB) และ character large object (CLOB) หากต้องการทำสิ่งนี้ คุณต้องกำหนด FINDSTRING อีกหนึ่งตัว ให้นำ BLOB มาเป็นพารามิเตอร์แรก

```
CREATE FUNCTION FINDSTRING (BLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC FINDSTRING_BLOB
  EXTERNAL NAME 'MYLIB/MYPGM(FINDSTR)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURNS NULL ON NULL INPUT
```

ตัวอย่างนี้แสดงให้เห็นการ Overloading ของชื่อ UDF และแสดงให้เห็นว่า UDF หลายตัวสามารถใช้เนื้อหาร่วมกันได้. โปรดสังเกตุว่าถึงแม้ว่า BLOB ไม่สามารถถูกกำหนดค่าให้กับ CLOBได้, แต่ว่าสามารถใช้ซอร์สโค้ดเดียวกันได้. ไม่มีปัญหาการโปรแกรมมิง ในตัวอย่างข้างต้นเนื่องด้วย อินเตอร์เฟสสำหรับ BLOB และ CLOB ระหว่าง DB2 และโปรแกรม UDF เหมือนกัน: คือความยาวตามด้วยข้อมูล.

ตัวอย่าง: การค้นหาสตริงบน user-defined type (UDT):

สมมติว่า คุณพอใจกับฟังก์ชัน FINDSTRING จาก การค้นหาสตริง binary large object (BLOB) แต่ตอนนี้คุณต้องการ นิยาม distinct type ชื่อ BOAT ด้วยชนิดต้นฉบับคือ BLOB

และคุณต้องการให้ FINDSTRING จัดการกับค่าที่มีชนิดข้อมูลเป็น BOAT ได้, ดังนั้นคุณจึงสร้างฟังก์ชัน FINDSTRING มาอีกหนึ่งฟังก์ชัน. ฟังก์ชันนี้จะใช้ต้นฉบับของ FINDSTRING ที่จัดการกับค่า BLOB. โปรดสังเกตว่า มีการ overload ของ FINDSTRING ในตัวอย่างนี้:

```
CREATE FUNCTION FINDSTRING (BOAT, VARCHAR(200))
  RETURNS INT
  SPECIFIC "slick_fboat"
  SOURCE SPECIFIC FINDSTRING_BLOB
```

โปรดสังเกตุว่าฟังก์ชัน FINDSTRING นี้จะมี Signature ต่างจากฟังก์ชัน FINDSTRING ใน “ตัวอย่าง: การค้นหาสตริง BLOB” ในหน้า 187, ดังนั้นจึงไม่มีปัญหาในการ Overloading ของชื่อฟังก์ชัน. เนื่องจากคุณใช้อนุประโยค SOURCE, ดังนั้นคุณไม่สามารถใช้อนุประโยค EXTERNAL NAME หรือคีย์เวิร์ดอื่นที่ใช้ระบุฟังก์ชันแอ็ตทริบิวต์ได้. แอ็ตทริบิวต์เหล่านี้จะนำมาจากฟังก์ชันต้นฉบับ. สุดท้าย, สังเกตุว่าในการระบุฟังก์ชันต้นฉบับนั้นก็คือคุณกำลังใช้ชื่อฟังก์ชันเฉพาะโดยตรง ซึ่งฟังก์ชันเหล่านี้จัดเตรียมไว้ใน “ตัวอย่าง: การค้นหาสตริง BLOB” ในหน้า 187. เพราะว่านี่เป็นการอ้างอิงที่ไม่ครบตามเกณฑ์, Schema ที่ฟังก์ชันต้นฉบับนี้เก็บอยู่จะต้องอยู่ในฟังก์ชันพาธ, มิฉะนั้นการอ้างอิงนี้จะหาไม่เจอ.

ตัวอย่าง: AVG บน user-defined type (UDT):

ตัวอย่างนี้จะใช้ฟังก์ชันการรวม AVG บน Distinct Type ชนิด CANADIAN_DOLLAR

การเข้มงวดเรื่องชนิดจะช่วยป้องกันคุณจากการใช้ฟังก์ชันในตัว AVG บน Distinct Type ได้. ซึ่งกลายเป็นว่า ชนิดต้นฉบับของ CANADIAN_DOLLAR ก็คือ DECIMAL, ดังนั้นคุณจึงสร้าง AVG โดยใช้ต้นฉบับของฟังก์ชันในตัว AVG(DECIMAL).

```
CREATE FUNCTION AVG (CANADIAN_DOLLAR)
  RETURNS CANADIAN_DOLLAR
  SOURCE "QSYS2".AVG(DECIMAL(9,2))
```

โปรดสังเกตว่า SOURCE clause ต้องใช้ชื่อฟังก์ชันที่ครบตามเกณฑ์, เผื่อในกรณีที่อาจมีฟังก์ชัน AVG อื่นแฝงอยู่ใน SQL พาธของคุณ.

ตัวอย่าง: การนับ:

ฟังก์ชันนับจำนวนอย่างง่ายของคุณจะคืนค่า 1 ในครั้งแรกที่เรียกใช้และจะเพิ่มผลลัพธ์ทีละหนึ่งในแต่ละครั้งที่เรียกใช้ ฟังก์ชันนี้ไม่ได้รับอากิวเมนต์ SQL และโดยนิยามแล้ว ฟังก์ชันนี้คือฟังก์ชันแบบ NOT DETERMINISTIC เนื่องจากผลลัพธ์จะเปลี่ยนไปในการเรียกใช้แต่ละครั้ง

มันจะใช้ SCRATCHPAD เพื่อบันทึกค่าที่คืนค่ามาล่าสุด. แต่ละครั้งที่ถูกเรียก ฟังก์ชันจะเพิ่มค่านี้และคืนค่านี้ไป.

```
CREATE FUNCTION COUNTER ()
  RETURNS INT
  EXTERNAL NAME 'MYLIB/MYFUNCS(CTR)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC
  NOT FENCED
  SCRATCHPAD 4
  DISALLOW PARALLEL
```

โปรดสังเกตุว่าไม่จำเป็นต้องกำหนดพารามิเตอร์, มีแค่วงเล็บว่าง. ฟังก์ชันด้านบนนี้ระบุ SCRATCHPAD และใช้ค่าดีฟอลต์ของ NO FINAL CALL. ในกรณีนี้, ขนาดของกระดาษทดจะถูกตั้งค่าให้เป็นขนาด 4 ไบต์เท่านั้น, ซึ่งก็เพียงพอแล้วสำหรับการนับ. เนื่องจากฟังก์ชัน COUNTER ต้องการใช้หนึ่งกระดาษทดเท่านั้น ในการทำงานให้ถูกต้อง, DISALLOW PARALLEL จึงถูกเพิ่มเข้าไปเพื่อป้องกัน DB2 จากการทำงานแบบขนาน.

ตัวอย่าง: ฟังก์ชันแบบ Table ที่คืนค่า document ID:

สมมุติว่าคุณเขียนฟังก์ชันแบบ Table ที่ คืนค่าแถวที่ประกอบด้วยคอลัมน์เดียวที่มี document identifier สำหรับแต่ละ เอกสารที่ตรงกับพื้นที่หัวข้อที่ให้มา (พารามิเตอร์แรก) และ มีสตริงที่ให้มา (พารามิเตอร์ที่สอง)

ฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) นี้จะระบุ เอกสารอย่างรวดเร็ว:

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
  RETURNS TABLE (DOC_ID CHAR(16))
  EXTERNAL NAME 'DOCFUNCS/UDFMATCH(udfmatch)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  NOT FENCED
  SCRATCHPAD
  NO FINAL CALL
  DISALLOW PARALLEL
  CARDINALITY 20
```

ภายในบริบทของเซสชันเดียวแล้ว ฟังก์ชันนี้จะคืนค่าตารางเดียวกันเสมอ, ดังนั้นมันจึงถูกกำหนดให้เป็นแบบ DETERMINISTIC. RETURNS clause กำหนดเอาต์พุตจาก DOCMATCH, รวมถึงชื่อคอลัมน์ DOC_ID. FINAL CALL ไม่จำเป็นต้องระบุสำหรับฟังก์ชัน Table นี้. คีย์เวิร์ด DISALLOW PARALLEL จำเป็นเนื่องจากฟังก์ชันตารางไม่สามารถทำงาน แบบขนานได้. ถึงแม้ว่าขนาดของเอาต์พุตจาก DOCMATCH จะเป็นตารางขนาด ใหญ่, แต่ค่า CARDINALITY 20 จะเป็นค่าแทนที่, และจะถูกระบุเพื่อช่วย ให้ตัว optimizer ตัดสินใจได้ดีขึ้น.

โดยทั่วไป, ฟังก์ชันแบบ Table นี้อาจจะถูกใช้ในการเชื่อมโยงกับตารางที่เก็บข้อความเอกสาร, ดังด้านล่างนี้:

```
SELECT T.AUTHOR, T.DOCTEXT
  FROM DOCS AS T, TABLE(DOCMATCH('MATHEMATICS', 'ZORN''S LEMMA')) AS F
  WHERE T.DOCID = F.DOC_ID
```

โปรดสังเกตุไวยากรณ์พิเศษ (คีย์เวิร์ด TABLE) สำหรับระบุฟังก์ชันแบบ Table ในอนุประโยค FROM. ในการพยายามนี้, ฟังก์ชันแบบ Table ที่ชื่อ DOCMATCH() จะคืนค่าแถวที่เก็บคอลัมน์ DOC_ID สำหรับแต่ละเอกสาร MATHEMATICS ที่อ้างอิงไปยัง ZORN'S LEMMA. ค่า DOC_ID จะถูกเชื่อมโยงกับตารางเอกสารหลัก, และใช้ดึงชื่อผู้แต่งและข้อความเอกสาร.

การส่งผ่านอากิวเมนต์จาก DB2 ไปยังฟังก์ชันภายนอก:

DB2 ได้จัดเตรียมหน่วยเก็บสำหรับพารามิเตอร์ทุกตัวที่ส่งผ่าน ไปยังฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) ดังนั้นพารามิเตอร์จะถูกส่งผ่านไปยังฟังก์ชัน แบบภายนอกด้วยแอดเดรส

นี่คือวิธีการส่งผ่านพารามิเตอร์แบบปกติสำหรับโปรแกรม. สำหรับเซอร์วิสโปรแกรม, โปรดตรวจสอบให้แน่ใจว่าพารามิเตอร์ถูกกำหนดไว้อย่างถูกต้องในฟังก์ชันโค้ด.

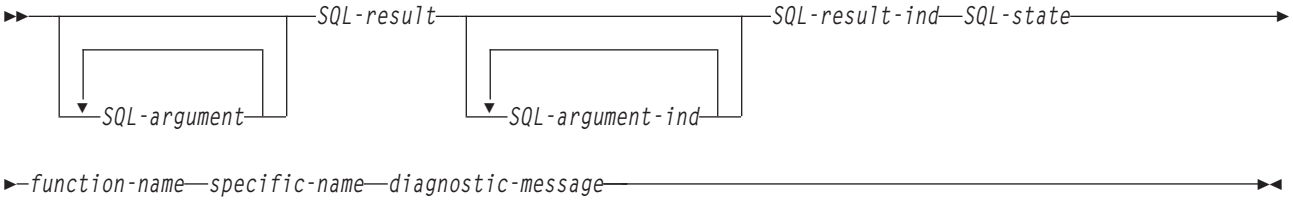
เมื่อกำหนดและใช้งานพารามิเตอร์ใน UDF แล้ว, ควรดูแลเพื่อให้แน่ใจว่าไม่มีการอ้างอิงถึงหน่วยเก็บสำหรับพารามิเตอร์ที่กำหนดให้มากไปกว่าที่ถูกกำหนดให้สำหรับพารามิเตอร์นั้น. พารามิเตอร์ถูกเก็บไว้ทั้งหมดในเนื้อที่เดียวกันและการใช้พื้นที่หน่วยเก็บของพารามิเตอร์เกินที่กำหนดให้จะบันทึกทับค่าของพารามิเตอร์อื่น. ในทางกลับกัน, วิธีนี้, สามารถทำให้ฟังก์ชันนี้ได้เห็นข้อมูลอินพุตที่ไม่ถูกต้องหรือทำให้ค่าถูกส่งคืนไปยังฐานข้อมูลที่ไม่ถูกต้อง.

พารามิเตอร์ที่ได้รับการสนับสนุนซึ่งใช้ได้กับ UDFs แบบภายนอกมีอยู่หลายลักษณะด้วยกัน. ส่วนใหญ่, ลักษณะที่ต่างกัน คือจำนวนพารามิเตอร์ที่ถูกส่งผ่านไปยังโปรแกรมภายนอกหรือเซอร์วิสโปรแกรม.

รูปแบบพารามิเตอร์ SQL:

รูปแบบพารามิเตอร์ SQL เป็นไปตาม SQL มาตรฐานอุตสาหกรรม รูปแบบพารามิเตอร์นี้สามารถใช้งานได้กับฟังก์ชันกำหนดเองแบบสเกลาร์ (UDFs)

ด้วยลักษณะพารามิเตอร์ SQL, พารามิเตอร์จะถูกส่งผ่านไปยังโปรแกรมภายนอกดังนี้ (ตามลำดับที่ระบุ):



SQL-argument

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. ค่านี้จะทำซ้ำ *n* ครั้ง, โดยที่ค่า *n* เป็นจำนวนของ อากิวเมนต์ที่ระบุอยู่ในการอ้างอิงฟังก์ชัน. ค่าของแต่ละอากิวเมนต์เหล่านี้จะถูกนำมาจากนิพจน์ที่ระบุไว้ในการเรียกฟังก์ชันทำงาน. ค่าจะถูกแสดงออกในประเภทข้อมูลของพารามิเตอร์ที่กำหนดไว้ในคำสั่งฟังก์ชันการสร้าง. หมายเหตุ: พารามิเตอร์เหล่านี้จะนำไปใช้เป็นอินพุตเท่านั้น; การเปลี่ยนค่าพารามิเตอร์ใดๆ ที่กระทำโดย UDF จะละเลยข้ามไปโดย DB2.

SQL-result

อากิวเมนต์นี้ถูกเซ็ตค่าโดย UDF ก่อนการส่งกลับไปยัง DB2. ฐานข้อมูลมีหน่วยเก็บสำหรับค่าส่งคืน. เนื่องจากพารามิเตอร์ถูกส่งผ่านโดยแอดเดรส, แอดเดรสจึงเป็นหน่วยเก็บค่าส่งคืน. ฐานข้อมูลจะมีหน่วยเก็บมากเท่าที่จำเป็นสำหรับค่าส่งคืนตามที่กำหนดไว้บนคำสั่ง CREATE FUNCTION. ถ้าหากมีการใช้ประโยค CAST FROM ในข้อ

ความ CREATE FUNCTION , DB2 ให้สันนิษฐานไว้ก่อนว่า UDF ได้ส่งกลับค่าที่ได้กำหนดไว้ในประโยค CAST FROM , หรือมิฉะนั้น DB2 ให้สันนิษฐานว่า UDF ส่งกลับค่าที่ได้กำหนดไว้ใน ประโยค RETURNS แทน.

SQL-argument-ind

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. อากิวเมนต์สามารถถูกใช้งานโดย UDF เพื่อกำหนดว่า *SQL-argument* ที่ต้องการเป็น Null หรือไม่. The *n*th *SQL-argument-ind* corresponds to the *n*th *SQL-argument*, described previously. ตัวบ่งชี้แต่ละตัวถูกกำหนดเป็นจำนวนเต็มขนาดสองไบต์. ตัวบ่งชี้นั้นจะถูกกำหนดให้มีค่าใดค่าหนึ่งต่อไปนี้:

0 มีอากิวเมนต์อยู่และไม่เป็นศูนย์.

-1 อากิวเมนต์เป็นศูนย์ .

หากฟังก์ชันถูกกำหนดด้วย RETURNS NULL ON NULL INPUT, UDF จะไม่ต้องตรวจสอบเพื่อหาค่าที่เป็น null . อย่างไรก็ตาม, หากฟังก์ชันถูกกำหนดด้วย CALLS ON NULL INPUT, อากิวเมนต์ใดๆ สามารถเป็น NULL และ UDF ควรตรวจสอบเพื่อหาอินพุตที่เป็น null . หมายเหตุ: พารามิเตอร์เหล่านี้จะนำไปใช้เป็นอินพุตเท่านั้น; การเปลี่ยนค่าพารามิเตอร์ใดๆ ที่กระทำโดย UDF จะละเลยข้ามไปโดย DB2.

SQL-result-ind

อากิวเมนต์นี้ถูกเซ็ตค่าโดย UDF ก่อนการส่งกลับไปยัง DB2. ฐานข้อมูลมีหน่วยเก็บสำหรับค่าส่งคืน. อากิวเมนต์ถูกกำหนดไว้เป็นจำนวนเต็มขนาดสองไบต์. หากกำหนดให้เป็นค่าลบ, ฐานข้อมูลจะตีความผลของฟังก์ชันเป็นค่าศูนย์ . หากกำหนดค่าให้เป็น null หรือค่าบวก, ฐานข้อมูลจะใช้งานค่าที่ถูกส่งคืนใน *SQL-result*. ฐานข้อมูล ได้จัดเตรียมหน่วยเก็บสำหรับตัวบ่งชี้ค่าส่งคืน. เนื่องจากพารามิเตอร์ถูกส่งผ่านโดยแอดเดรส, แอดเดรสจึงเป็นหน่วยเก็บค่าตัวบ่งชี้.

SQL-state

อากิวเมนต์ คือค่า CHAR(5) ซึ่งแทนค่า SQLSTATE.

พารามิเตอร์นี้จะถูกส่งผ่านในรูปของฐานข้อมูลที่ตั้งค่าเป็น '00000' และสามารถตั้งค่าโดยฟังก์ชันให้เป็นสถานะผลลัพธ์ของฟังก์ชัน. ขณะที่ปกติแล้ว SQLSTATE ไม่ได้ถูกกำหนดค่าโดยฟังก์ชัน, แต่สามารถใช้งาน SQLSTATE เพื่อส่งสัญญาณข้อผิดพลาดหรือแจ้งเตือนไปยังฐานข้อมูลได้ดังต่อไปนี้:

01Hxx ฟังก์ชันโค้ดตรวจพบการแจ้งเตือน. การตรวจพบนี้ก่อให้เกิดการแจ้งเตือนแบบ SQL, ซึ่ง *xx* ในที่นี้อาจเป็นหนึ่งในหลายๆ สตริงที่อาจพบได้.

38xxx ฟังก์ชันโค้ดตรวจพบข้อผิดพลาด. การตรวจพบนี้ก่อให้เกิดข้อผิดพลาด SQL . ซึ่ง *xxx* ในที่นี้อาจเป็นหนึ่งในหลายๆ สตริงที่อาจพบได้.

function-name

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. มันคือค่า VARCHAR(139) ที่ประกอบด้วยชื่อของฟังก์ชันที่ถูกเรียกใช้งานเสมือนเป็น ฟังก์ชันโค้ด.

รูปแบบของชื่อฟังก์ชันที่ถูกส่งผ่านคือ:

```
<schema-name>.<function-name>
```

พารามิเตอร์นี้มีประโยชน์เมื่อฟังก์ชันโค้ดถูกใช้งานโดย definition จำนวนมากของ UDF ดังนั้นโค้ดจะสามารถแยกแยะได้ว่า definition ไหนถูกเรียกใช้งาน. หมายเหตุ: พารามิเตอร์นี้จะถูกใช้เป็นเพียงอินพุตเท่านั้น; การเปลี่ยนค่าพารามิเตอร์ใดๆ ที่กระทำโดย UDF จะถูกละเลยข้ามไปโดย DB2.

specific-name

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. มันคือค่า VARCHAR(128) ที่ประกอบด้วยชื่อของฟังก์ชันที่ถูกเรียกใช้งานเสมือนเป็น ฟังก์ชันโค้ด.

เช่นเดียวกับชื่อฟังก์ชัน, พารามิเตอร์นี้มีประโยชน์ เมื่อฟังก์ชันโค้ดถูกใช้งานโดย definition จำนวนมากของ UDF เพื่อที่โค้ดจะสามารถแยกแยะได้ว่า definition ใดจะถูกเรียกใช้งาน. หมายเหตุ: พารามิเตอร์นี้จะถูกใช้เป็นเพียงอินพุตเท่านั้น; การเปลี่ยนค่าพารามิเตอร์ใดๆ ที่กระทำโดย UDF จะถูกละเลยข้ามไปโดย DB2.

diagnostic-message

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. ค่าของอากิวเมนต์คือค่า VARCHAR(70) ซึ่งสามารถถูกใช้งานโดย UDF เพื่อส่งข้อความกลับเมื่อ UDF แจ้งการเตือนและข้อผิดพลาดของ SQLSTATE.

อากิวเมนต์จะถูก initialize โดยฐานข้อมูลบนอินพุตไปยัง UDF และอาจถูกกำหนดค่าโดย UDF โดยมีข้อมูลอธิบายรายละเอียด. ข้อความ Message จะถูกละเลยไปโดย DB2 เว้นแต่พารามิเตอร์ SQL-state ถูกกำหนดโดย UDF.

สิ่งอ้างอิงที่เกี่ยวข้อง

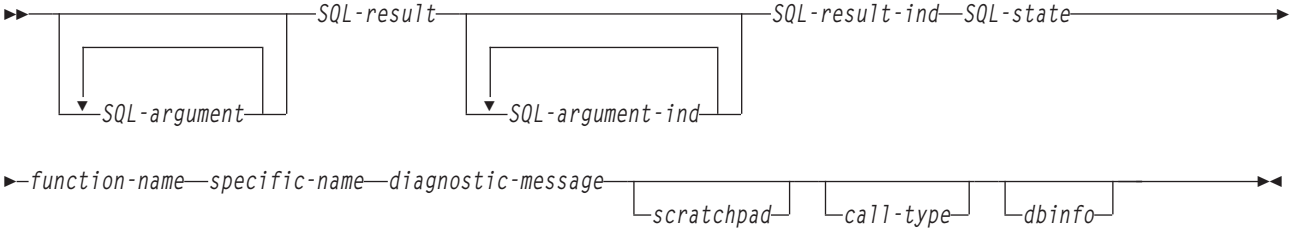
ข้อความและโค้ด SQL

รูปแบบพารามิเตอร์ DB2SQL:

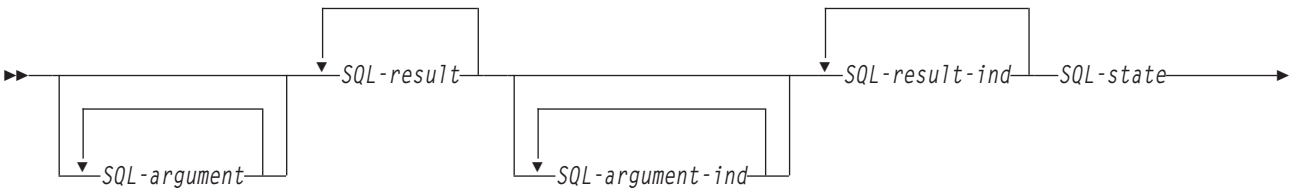
ด้วยรูปแบบพารามิเตอร์ DB2SQL, พารามิเตอร์เดียวกัน และลำดับพารามิเตอร์เดียวกันจะถูกส่งผ่านไปยังโปรแกรมภายนอกหรือเซอร์วิสโปรแกรม อย่างที่ถูกส่งผ่านไปสำหรับรูปแบบพารามิเตอร์ SQL อย่างไรก็ตาม DB2SQL อนุญาตให้ ส่งผ่านพารามิเตอร์ตัวเลือกเสริมได้เช่นกัน

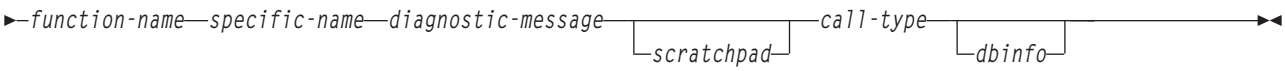
หากมีการระบุพารามิเตอร์ตัวเลือกเสริมมากกว่าหนึ่งตัวใน definition UDF , พารามิเตอร์เหล่านั้นจะถูกส่งผ่านไปยัง UDF ตามลำดับที่กำหนดไว้ด้านล่าง. โปรดศึกษาจากลักษณะพารามิเตอร์ SQL เพื่อดูพารามิเตอร์ทั่วไป. ลักษณะพารามิเตอร์สามารถใช้งานได้กับ UDF แบบสเกลาร์และตาราง.

สำหรับฟังก์ชันสเกลาร์:



สำหรับฟังก์ชันตาราง:





scratchpad

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. อากิวเมนต์จะถูกแสดงก็ต่อเมื่อคำสั่ง CREATE FUNCTION สำหรับ UDF ระบุคีย์เวิร์ด SCRATCHPAD. อากิวเมนต์นี้ คือโครงสร้างซึ่งมีส่วนประกอบต่อไปนี้:

- INTEGER แสดงความยาวของ scratchpad.
- scratchpad ที่แท้จริง, จะถูกเตรียมข้อมูลเบื้องต้นไปยังทุกไบนารี 0 โดย DB2 ก่อนการเรียกใช้งาน UDF ในครั้งแรก.

UDF สามารถใช้งาน scratchpad ให้เป็นทั้งหน่วยเก็บใช้งาน หรือหน่วยเก็บถาวร, เนื่องจาก scratchpad จะถูกเก็บไว้เมื่อมีการเรียก UDF ทำงาน.

สำหรับฟังก์ชันตาราง, scratchpad จะถูก เตรียมข้อมูลเบื้องต้นตามที่กล่าวไว้ข้างต้นก่อนการเรียกทำงาน FIRST ไปยัง UDF หากมีการระบุ FINAL CALL ไว้บน CREATE FUNCTION. หลังจากการเรียกนี้แล้ว, เนื้อหา scratchpad จะอยู่ภายใต้การควบคุมของฟังก์ชันตารางทั้งหมด. DB2 ไม่ได้ทดสอบ หรือ เปลี่ยนเนื้อหาของ scratchpad หลังจากนั้น. scratchpad จะถูกส่งผ่านไปยังฟังก์ชันในการเรียกทำงานแต่ละครั้ง. ฟังก์ชันสามารถถูกป้อนกลับเข้าไปใหม่, และ DB2 จะเก็บเตรียมข้อมูลสถานะของตัวเองไว้ใน scratchpad.

หากมีการระบุ NO FINAL CALL หรือถูกกำหนดเป็นค่าดีฟอลต์สำหรับฟังก์ชันตาราง, scratchpad จะถูก initialize ตามที่กล่าวไว้ด้านบนสำหรับการเรียก OPEN แต่ละครั้ง, และเนื้อหา scratchpad จะอยู่ภายใต้การควบคุมของฟังก์ชันตารางทั้งหมดระหว่างการเรียก OPEN. นี่เป็นเรื่องสำคัญมากสำหรับฟังก์ชันตารางที่ใช้งานร่วมกันหรือในการสืบค้นย่อย. หากจำเป็นต้องรักษาเนื้อหาของ scratchpad ทุกครั้งที่มีการเรียก OPEN , คุณต้องระบุ FINAL CALL ในคำสั่ง CREATE FUNCTION ของคุณ. เมื่อระบุ FINAL CALL, เพิ่มเติมนอกเหนือจากการเรียก OPEN, FETCH, และ CLOSE, ฟังก์ชันตารางจะได้รับการเรียกแบบ FIRST และ FINAL เช่นกัน, เพื่อรักษา scratchpad และการปล่อยรีซอร์ส.

call-type

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. สำหรับฟังก์ชันแบบสเกลาร์, อากิวเมนต์จะถูกแสดงก็ต่อเมื่อคำสั่ง CREATE FUNCTION สำหรับ UDF ระบุคีย์เวิร์ด FINAL. อย่างไรก็ตาม, ฟังก์ชันตาราง อากิวเมนต์จะ *แสดงอยู่เสมอ*. อากิวเมนต์จะอยู่ตามหลังอากิวเมนต์ *scratchpad*; หรืออาร์กิวเมนต์ *ข้อความวินิจฉัย* หากอาร์กิวเมนต์ scratchpad ไม่ปรากฏขึ้นมา. อากิวเมนต์นี้จะใช้รูปแบบของค่า INTEGER.

สำหรับฟังก์ชันสเกลาร์:

-1 นี่คือ*การเรียกครั้งแรก* ให้กับ UDF สำหรับคำสั่งนี้. การเรียกครั้งแรกคือ*การเรียกแบบปกติ* ในค่าอากิวเมนต์ SQL ทั้งหมดที่ถูกส่งผ่าน.

0 นี่คือ *การเรียกแบบปกติ*. (ค่าอากิวเมนต์อินพุตแบบปกติทั้งหมดจะถูกส่งผ่าน).

1 นี่คือ *การเรียกครั้งสุดท้าย*. ไม่มีการส่งผ่านค่า*SQL-argument* หรือค่า *SQL-argument-ind*. UDF ไม่ควรส่งคืนคำตอบใดๆ โดยใช้ SQL-result, อากิวเมนต์ SQL-result-ind, SQL-state, หรือ ข้อความวินิจฉัย. ระบบจะละเลยข้ามอากิวเมนต์เหล่านี้ เมื่อมีการส่งคืนมาจาก UDF.

สำหรับฟังก์ชันตาราง:

-2 นี่คือ*การเรียกครั้งแรก* ให้กับ UDF สำหรับคำสั่งนี้. การเรียกครั้งแรกคือ*การเรียกแบบปกติ* ในค่าอากิวเมนต์ SQL ทั้งหมดที่ถูกส่งผ่าน.

-1 นี่คือ*การเรียกแบบเปิด* ไปยัง UDF สำหรับคำสั่งนี้. scratchpad จะถูก initialize หากไม่มีการระบุ FINAL CALL , แต่ไม่จำเป็นมากนัก. ค่าอากิวเมนต์ SQL ทั้งหมดจะถูกส่งผ่าน.

0 นี่คือ *การเรียกแบบดึงข้อมูลออก*. DB2 คาดหวังว่าฟังก์ชันตารางจะส่งกลับ แถวซึ่งประกอบด้วยชุดของค่าส่งคืน, หรือ เงื่อนไขการสิ้นสุดตารางที่บ่งชี้โดย SQLSTATE ที่มีค่า '02000' อย่างใดอย่างหนึ่ง.

1 นี่คือ*การเรียกแบบปิด*. การเรียกแบบนี้จะสร้างสมดุลให้กับการเรียก OPEN, และสามารถถูกใช้ประมวลผล CLOSE แบบภายนอกและปล่อยรีซอร์ส.

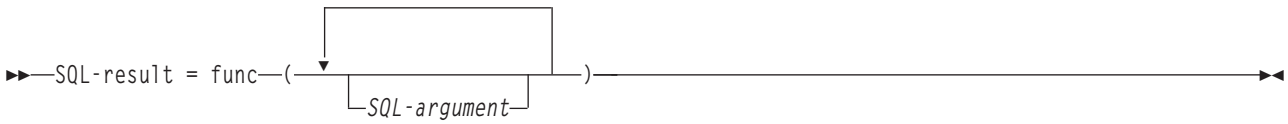
2 นี่คือ *การเรียกครั้งสุดท้าย*. ไม่มีการส่งผ่านค่า*SQL-argument* หรือค่า *SQL-argument-ind*. UDF ไม่ควรส่งคืนคำตอบใดๆ โดยใช้ SQL-result, อากิวเมนต์ SQL-result-ind, SQL-state, หรือ ข้อความวินิจฉัย. ระบบจะละเลยข้ามอากิวเมนต์เหล่านี้ เมื่อมีการส่งคืนมาจาก UDF.

dbinfo อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. อากิวเมนต์จะถูกแสดงก็ต่อเมื่อคำสั่ง CREATE FUNCTION สำหรับ UDF ระบุคีย์เวิร์ด DBINFO. อากิวเมนต์คือโครงสร้างที่มี definition อยู่ในการสอดแทรกคำสั่ง `sqludf`.

รูปแบบพารามิเตอร์ GENERAL:

ด้วยรูปแบบพารามิเตอร์ GENERAL พารามิเตอร์จะถูก ส่งผ่านไปในเซอร์วิสโปรแกรมภายนอกเหมือนกันกับที่พารามิเตอร์เหล่านั้นถูกระบุไว้ใน คำสั่ง CREATE FUNCTION รูปแบบพารามิเตอร์สามารถใช้งานได้กับ ฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) แบบสเกลาร์

ฟอร์แม็ตคือ:



SQL-argument

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. ค่านี้จะทำซ้ำ *n* ครั้ง, โดยที่ค่า *n* เป็นจำนวนของ อากิวเมนต์ที่ระบุอยู่ในการอ้างอิงฟังก์ชัน. ค่าของแต่ละอากิวเมนต์เหล่านี้จะถูกนำมาจากนิพจน์ที่ระบุไว้ในการเรียกฟังก์ชันทำงาน. ค่าจะถูกแสดงออกในประเภทข้อมูลของพารามิเตอร์ที่กำหนดไว้ในคำสั่ง CREATE FUNCTION. หมายเหตุ: พารามิเตอร์เหล่านี้จะนำไปใช้เป็นอินพุตเท่านั้น; การเปลี่ยนค่าพารามิเตอร์ใดๆ ที่กระทำโดย UDF จะละเลยข้ามไปโดย DB2.

SQL-result

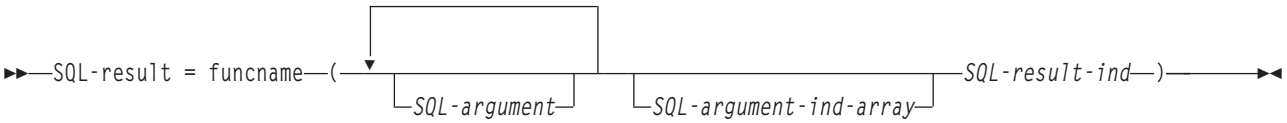
ค่านี้จะถูกส่งกลับโดย UDF. DB2 ก็อปปีค่าลงในหน่วยเก็บฐานข้อมูล. เพื่อจะส่งคืนค่าอย่างถูกต้อง, ฟังก์ชันโค้ดต้องเป็นฟังก์ชันการส่งคืนค่า. ฐานข้อมูลจะคัดลอกเฉพาะค่าที่กำหนดไว้ให้มากที่สุดสำหรับค่าส่งคืนซึ่งระบุไว้บนคำสั่ง CREATE FUNCTION . ถ้าหากมีการใช้ CAST FROM clause ในคำสั่ง CREATE FUNCTION, DB2 จะสันนิษฐานว่า UDF ได้ส่งกลับค่าที่ได้กำหนด ไว้ใน CAST FROM clause หรือมิฉะนั้น DB2 จะสันนิษฐานว่า UDF ส่งกลับค่าที่ได้กำหนดไว้ใน RETURNS clause

เนื่องจากข้อกำหนดที่ว่าฟังก์ชันโค้ดต้องเป็นฟังก์ชันการส่งคืนค่า, คุณต้องสร้างฟังก์ชันโค้ดใดๆ ให้กับลักษณะพารามิเตอร์ GENERAL ลงในเซอร์วิสโปรแกรม.

รูปแบบพารามิเตอร์ GENERAL WITH NULLS:

รูปแบบพารามิเตอร์ GENERAL WITH NULLS สามารถ ใช้งานได้กับฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) แบบสเกลาร์และตาราง

ด้วยลักษณะพารามิเตอร์, พารามิเตอร์จะถูกส่งผ่านไปยังเซอร์วิสโปรแกรมดังต่อไปนี้ (ตามลำดับที่ระบุไว้):



SQL-argument

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. ค่านี้จะทำซ้ำ *n* ครั้ง, โดยที่ค่า *n* เป็นจำนวนของ อากิวเมนต์ที่ระบุอยู่ในการอ้างอิงฟังก์ชัน. ค่าของแต่ละอากิวเมนต์เหล่านี้จะถูกนำมาจากนิพจน์ที่ระบุไว้ในการเรียกฟังก์ชันทำงาน. ค่าจะถูกแสดงออกในประเภทข้อมูลของพารามิเตอร์ที่กำหนดไว้ในคำสั่ง CREATE FUNCTION. หมายเหตุ: พารามิเตอร์เหล่านี้จะนำไปใช้เป็นอินพุตเท่านั้น; การเปลี่ยนค่าพารามิเตอร์ใดๆ ที่กระทำโดย UDF จะละเลยข้ามไปโดย DB2.

SQL-argument-ind-array

อากิวเมนต์นี้ ถูกกำหนดโดย DB2 ก่อนการเรียกใช้ UDF. อากิวเมนต์สามารถถูกใช้งานโดย UDF เพื่อกำหนดว่า *SQL-arguments* ตั้งแต่หนึ่งรายการขึ้นไปมีค่าเป็น null หรือไม่. อากิวเมนต์คือ array ของจำนวนเต็มขนาดสองไบต์ (ตัวบ่งชี้). อากิวเมนต์ array ลำดับที่ *n* เชื่อมโยงกับ *SQL-argument* ลำดับที่ *n*. แต่ละ array entry ถูกตั้งค่าให้เป็นค่าใดค่าหนึ่งต่อไปนี้:

0 มีอากิวเมนต์อยู่และไม่เป็นศูนย์.

-1 อากิวเมนต์เป็นศูนย์ .

UDF ควรตรวจสอบอินพุตที่มีค่า null . หมายเหตุ: พารามิเตอร์นี้จะถูกใช้เป็นเพียงอินพุตเท่านั้น; การเปลี่ยนค่าพารามิเตอร์ที่กระทำโดย UDF จะถูกละเลยโดย DB2.

SQL-result-ind

อากิวเมนต์นี้ถูกเซ็ตค่าโดย UDF ก่อนการส่งกลับไปยัง DB2. ฐานข้อมูลมีหน่วยเก็บสำหรับค่าส่งคืน. อากิวเมนต์ถูกกำหนดไว้เป็นจำนวนเต็มขนาดสองไบต์. หากกำหนดให้เป็นค่าลบ, ฐานข้อมูลจะตีความผลของฟังก์ชันเป็นค่าศูนย์ . หากกำหนดค่าให้เป็น null หรือค่าบวก, ฐานข้อมูลจะใช้งานค่าที่ถูกส่งคืนใน *SQL-result*. ฐานข้อมูล ได้จัดเตรียมหน่วยเก็บสำหรับตัวบ่งชี้ค่าส่งคืน. เนื่องจากพารามิเตอร์ถูกส่งผ่านโดยแอดเดรส, แอดเดรสจึงเป็นหน่วยเก็บค่าตัวบ่งชี้.

SQL-result

ค่านี้จะถูกส่งกลับโดย UDF. DB2 ก็อปปีค่าลงในหน่วยเก็บฐานข้อมูล. เพื่อจะส่งคืนค่าอย่างถูกต้อง, ฟังก์ชันโค้ดต้องเป็นฟังก์ชันการส่งคืนค่า. ฐานข้อมูลจะคัดลอกเฉพาะค่าที่กำหนดไว้ให้มากที่สุดสำหรับค่าส่งคืนซึ่งระบุไว้บนคำ

สั่ง CREATE FUNCTION . ถ้าหากมีการใช้ CAST FROM clause ในคำสั่ง CREATE FUNCTION , DB2 ให้สันนิษฐานไว้ก่อนว่า UDF ได้ส่งกลับค่าที่ได้กำหนดไว้ใน CAST FROM clause, หรือมิฉะนั้น DB2 ให้สันนิษฐานว่า UDF ส่งกลับค่าที่ได้กำหนดไว้ใน RETURNS clause แทน.

เนื่องจากข้อกำหนดที่ว่า ฟังก์ชันโค้ด คือฟังก์ชันการส่งคืนค่า, ฟังก์ชันใดๆ ที่ใช้สำหรับรูปแบบพารามิเตอร์ GENERAL WITH NULLS ต้องถูกสร้างขึ้นภายในเซอร์วิสโปรแกรม.

หมายเหตุ:

1. คุณสามารถระบุชื่อภายนอกที่ระบุไว้บนคำสั่ง CREATE FUNCTION ได้ด้วยการใส่เครื่องหมายอัญประกาศ หรือไม่ใส่เครื่องหมายอัญประกาศ. หากชื่อไม่มีเครื่องหมายอัญประกาศ, ชื่อจะถูกทำเป็นอักษรตัวพิมพ์ใหญ่ก่อนที่จะถูกเก็บไว้; หากมีเครื่องหมายอัญประกาศ, ชื่อจะถูกเก็บไว้ตามที่ระบุ. เรื่องนี้สำคัญเมื่อตั้งชื่อโปรแกรม, เพราะฐานข้อมูลจะค้นหาโปรแกรมที่มีชื่อตรงกันกับชื่อที่เก็บไว้ด้วย definition ฟังก์ชัน. ตัวอย่างเช่น, หากฟังก์ชันถูกสร้างขึ้นเป็น:

   ```
   CREATE FUNCTION X(INT) RETURNS INT
     LANGUAGE C
     EXTERNAL NAME 'MYLIB/MYPGM(MYENTRY)'
   ```

   แลซอร์สสำหรับโปรแกรมคือ:

   ```
   void myentry(
         int*in
         int*out,
         .
         .
         . .
   ```

   ฐานข้อมูลจะไม่พบ entry เพราะ entry จะเป็นตัวอักษรตัวพิมพ์เล็ก*myentry* และฐานข้อมูลจะถูกสร้างขึ้นเพื่อดูตัวอักษรตัวพิมพ์ใหญ่ *MYENTRY*.
2. สำหรับเซอร์วิสโปรแกรมและโมดูล C++, โปรดแน่ใจว่าในซอร์สโค้ด C++ อยู่ก่อนหน้า definition ฟังก์ชันโปรแกรมโดยมี *extern "C"*. มิฉะนั้น, คอมไพเลอร์ C++ จะดำเนินการ 'name mangling' ของชื่อฟังก์ชันและฐานข้อมูลจะไม่พบชื่อนั้น.

รูปแบบพารามิเตอร์ DB2GENERAL:

รูปแบบพารามิเตอร์ DB2GENERAL ถูกใช้โดยฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) ของ Java

หลักการที่เกี่ยวข้อง

Java SQL routines

รูปแบบพารามิเตอร์ Java:

รูปแบบพารามิเตอร์ Java เป็นรูปแบบที่กำหนดโดย SQLJ Part 1: มาตรฐาน SQL Routines.

หลักการที่เกี่ยวข้อง

Java SQL routines

ข้อควรพิจารณาฟังก์ชันแบบ Table:

ฟังก์ชันตารางภายนอก คือ user-defined function (UDF) ที่ส่งตารางไปยังคำสั่ง SQL ที่ถูกอ้างถึง การอ้างถึงฟังก์ชันตารางจะสามารถใช้งานได้เฉพาะใน FROM clause ของคำสั่ง SELECT

เมื่อใช้งานฟังก์ชันตาราง, โปรดสังเกตดังต่อไปนี้:

- ถึงแม้ว่าฟังก์ชันตารางจะส่งค่าตาราง, อินเตอร์เฟสฟิสิคัล ระหว่าง DB2 และ UDF จะเป็นแบบ ครั้งละหนึ่งแถว. การเรียกไปยังฟังก์ชันตารางมี 5 ประเภท: OPEN, FETCH, CLOSE, FIRST, และ FINAL. การเรียกแบบ FIRST และ FINAL ขึ้นอยู่กับวิธีที่คุณกำหนด UDF. ระบบ *ประเภทการเรียก* ที่สามารถใช้กับฟังก์ชันแบบสเกลาร์จะถูกใช้งานเพื่อแยกแยะการเรียกเหล่านี้.
- มาตรฐานอินเตอร์เฟสที่ใช้ระหว่าง DB2 และ ฟังก์ชัน scalar ที่กำหนดโดยผู้ใช้จะถูกขยายออก เพื่อรองรับฟังก์ชันตาราง. อากิวเมนต์ *SQL-result* จะทำงานซ้ำสำหรับฟังก์ชันตาราง; ซึ่งก็คือแต่ละ instance ที่เชื่อมโยงกับคอลัมน์ที่จะถูกส่งคืนตามที่กำหนดใน RETURNS TABLE ของคำสั่ง CREATE FUNCTION. อากิวเมนต์ *SQL-result-ind* จะทำซ้ำ, แต่ละ instance ที่เชื่อมโยงกับ instance *SQL-result* ที่เกี่ยวข้อง.
- คอลัมน์ผลลัพธ์บางคอลัมน์ที่กำหนดใน RETURNS clause ของคำสั่ง CREATE FUNCTION สำหรับฟังก์ชันตารางเท่านั้นที่จะถูกส่งคืน. คีย์เวิร์ด DBINFO ของ CREATE FUNCTION, และอากิวเมนต์ *dbinfo* ที่เกี่ยวข้องเปิดใช้งาน optimization ที่คอลัมน์เหล่านั้นต้องการสำหรับการอ้างอิงของฟังก์ชันตารางที่ต้องส่งคืน.
- ค่าคอลัมน์แต่ละคอลัมน์ที่ถูกส่งคืนสอดคล้องกับฟอร์แม็ตของค่าที่ถูกส่งคืนจากฟังก์ชันแบบสเกลาร์.
- คำสั่ง CREATE FUNCTION สำหรับฟังก์ชันตารางมีค่ากำหนด CARDINALITY *n*. ข้อกำหนดคุณสมบัตินี้จะเปิดทางให้ definer ได้แจ้งให้ DB2 optimizer ถึงขนาดโดยประมาณของผลลัพธ์ เพื่อว่า optimizer สามารถทำการตัดสินใจได้ดีขึ้นเมื่อมีการอ้างถึงฟังก์ชัน. โดยไม่ต้องคำนึงว่ามีการระบุอะไรเป็น CARDINALITY ของฟังก์ชันตาราง, โปรดปฏิบัติตามข้อควรระวังในการเขียนฟังก์ชันด้วย infinite cardinality; นั่นคือ, ฟังก์ชันที่มักจะส่งคืนแถวบนการเรียกแบบ FETCH. DB2 คาดหวังในเงื่อนไข *end-of-table* , เพื่อเป็นเสมือนตัวเร่งการทำงานภายใน query processing. ดังนั้นฟังก์ชันตารางที่ไม่เคยส่งคืนเงื่อนไข end-of-table (SQL-state value '02000') จะก่อให้เกิดการประมวลผลแบบวนซ้ำไม่สิ้นสุด.

การประมวลผลข้อผิดพลาดของ UDFs:

เมื่อเกิดข้อผิดพลาดในการประมวลผลฟังก์ชันแบบ ผู้ใช้กำหนดเอง (UDF) ระบบจะทำตามโมเดลที่ระบุเพื่อจัดการข้อผิดพลาด

การประมวลผลข้อผิดพลาดฟังก์ชันตาราง

การประมวลผลข้อผิดพลาดสำหรับการเรียกฟังก์ชันตารางมีดังต่อไปนี้:

1. หากการเรียก FIRST ล้มเหลว, จะไม่มีการเรียกครั้งต่อไป.
2. หากการเรียก FIRST สำเร็จ, จะมีการเรียก OPEN, FETCH, และ CLOSE แบบ nested, และจะมีการเรียกแบบ FINAL เสมอ.
3. หากการเรียก OPEN ล้มเหลว, จะไม่มีการเรียก FETCH หรือ CLOSE.
4. หากการเรียก OPEN ดำเนินต่อไป, จะมีการเรียกแบบ FETCH และ CLOSE.
5. หากการเรียกแบบ FETCH ล้มเหลว, จะไม่มีการเรียกแบบ FETCH อีกต่อไป, แต่จะมีการเรียก CLOSE.

หมายเหตุ: แบบจำลองนี้จะอธิบายการประมวลผลข้อผิดพลาดแบบธรรมดาสำหรับตาราง UDF. ในกรณีที่ระบบล้มเหลวหรือมีปัญหาในการสื่อสาร, อาจไม่มีการเรียกที่ระบุโดยแบบจำลองการประมวลผลข้อผิดพลาด.

การประมวลผลข้อผิดพลาดฟังก์ชัน Scalar

แบบจำลองการประมวลผลข้อผิดพลาดสำหรับ UDF แบบ scalar, ซึ่งถูกกำหนดด้วยค่ากำหนด FINAL CALL, มีดังต่อไปนี้:

1. หากการเรียก FIRST ล้มเหลว, จะไม่มีการเรียกครั้งต่อไป.
2. หากการเรียก FIRST ดำเนินต่อไป, จะมีการเรียก NORMAL ต่อไปตามที่ได้รับประกันจากการประมวลผลคำสั่ง, และจะมีการเรียก FINAL เสมอ.
3. หากการเรียก NORMAL ล้มเหลว, จะไม่มีการเรียกแบบ NORMAL อีกต่อไป, แต่จะมีการเรียก FINAL (หากคุณระบุ FINAL CALL). ซึ่งหมายความว่าหากมีการส่งคืนข้อผิดพลาดบนการเรียก FIRST, UDF ต้องลบทิ้งก่อนการส่งคืน, เพราะไม่มีการเรียก FINAL.

หมายเหตุ: แบบจำลองนี้จะอธิบายการประมวลผลข้อผิดพลาดแบบธรรมดาสำหรับ UDF แบบสเกลาร์. ในกรณีที่ระบบล้มเหลวหรือมีปัญหาในการสื่อสาร, อาจไม่มีการเรียกที่ระบุโดยแบบจำลองการประมวลผลข้อผิดพลาด.

ข้อควรพิจารณา Threads:

ฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) ที่ถูกกำหนด เป็น FENCED รันในงานเดียวกันกับคำสั่ง SQL ที่เรียกฟังก์ชันดังกล่าว อย่างไรก็ตาม UDF รันใน thread ของระบบ แยกจาก thread ที่รัน คำสั่ง SQL

เนื่องจาก UDF รันในงานเดียวกันกับคำสั่ง SQL , UDF จึงอยู่ในสภาพแวดล้อมเดียวกันกับคำสั่ง SQL . อย่างไรก็ตาม, เนื่องจาก UDF รันภายใต้ thread ที่แยกต่างหาก, จึงควรพิจารณาเรื่อง thread ต่อไปนี้:

- UDF จะขัดแย้งกับรีซอร์สระดับ thread ที่เกิดจาก thread ของคำสั่ง SQL. แรกเริ่ม, ข้อมูลเหล่านี้คือรีซอร์สตารางที่ถูกกล่าวถึงด้านบน.
- UDFs จะไม่รับช่วงสิทธิที่รับมาจากโปรแกรมที่อาจแอ็คทีฟขณะที่คำสั่ง SQL ถูกเรียกทำงาน. สิทธิ UDF มาจากสิทธิในการใช้งานที่เชื่อมโยงกับตัวโปรแกรม UDF หรือมาจากสิทธิในการใช้งานของการรันคำสั่ง SQL ของผู้ใช้.
- UDF ไม่สามารถปฏิบัติการใดๆ ที่ถูกบล็อกไว้ไม่ให้รันใน thread รอง.
- โปรแกรม UDF ต้องถูกสร้างให้สามารถรันภายใต้ activation group ที่มีชื่อหรือใน activation group ของตัวเรียก (พารามิเตอร์ACTGRP). โปรแกรมที่ระบุ ACTGRP(*NEW) จะไม่ได้รับอนุญาตให้รันเป็น UDFs.

สิ่งอ้างอิงที่เกี่ยวข้อง

แอ็พพลิเคชันแบบ Multithreaded

"ข้อควรพิจารณาเรื่องเฟนซ์ (Fenced) หรือ อันเฟนซ์ (unfenced)" ในหน้า 199
เมื่อสร้างฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) โปรดพิจารณาว่า จะสร้าง UDF หรือ unfenced UDF

การประมวลผลแบบขนาน:

คุณสามารถกำหนดฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) เพื่อให้ ประมวลผลแบบขนานได้

หมายความว่าโปรแกรม UDF เดียวกันสามารถรันในหลายๆ thread ในเวลาเดียวกันได้. ดังนั้น, หาก ALLOW PARALLEL ถูกระบุไว้สำหรับ UDF, โปรดตรวจสอบให้แน่ใจว่าการทำงานดังกล่าวไม่เป็นผลเสียต่อ thread.

ฟังก์ชันตารางแบบผู้ใช้กำหนดไม่สามารถรันแบบขนานได้; ดังนั้น, จึงต้องระบุ DISALLOW PARALLEL เมื่อสร้างฟังก์ชัน

สิ่งอ้างอิงที่เกี่ยวข้อง

แอ็พพลิเคชันแบบ Multithreaded

ข้อควรพิจารณาเรื่องเฟนซ์ (Fenced) หรือ อันเฟนซ์ (unfenced):

เมื่อสร้างฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) โปรดพิจารณาว่า จะสร้าง UDF หรือ unfenced UDF

ตามค่าดีฟอลต์, UDF จะถูกสร้างขึ้นเป็น fenced UDF. Fenced จะแสดงว่าฐานข้อมูลควรรัน UDF ใน thread ต่างหาก. สำหรับ UDF ที่ซับซ้อน, การแยกนี้มีความสำคัญเพราะช่วยเลี่ยงปัญหาที่อาจจะเกิดขึ้นได้ เช่น การสร้างชื่อเคอร์เซอร์ SQL แบบเฉพาะ. การไม่ต้องกังวลเรื่องความขัดแย้งของรีซอร์สคือเหตุผลหนึ่งที่ต้องใช้ค่าดีฟอลต์และสร้าง UDF เป็น UDF แบบ fenced. UDF ที่ถูกสร้างขึ้นด้วยอ็อพชัน 'NOT FENCED' จะระบุกับฐานข้อมูลว่าผู้ใช้กำลังร้องขอให้ UDF รันภายใน thread เดียวกับที่เริ่มต้นใช้งาน UDF. ขอแนะนำให้ใช้ Unfenced กับฐานข้อมูล, ซึ่งยังคงตัดสินใจให้รัน UDF ด้วยวิธีเดียวกันกับ fenced UDF.

```
CREATE FUNCTION QGPL.FENCED (parameter1 INTEGER)
RETURNS INTEGER LANGUAGE SQL
BEGIN
RETURN parameter1 * 3;
END;

CREATE FUNCTION QGPL.UNFENCED1 (parameter1 INTEGER)
RETURNS INTEGER LANGUAGE SQL NOT FENCED
-- สร้าง UDF เพื่อร้องขอการทำงานที่รวดเร็วขึ้นผ่านทางอ็อพชัน NOT FENCED
BEGIN
RETURN parameter1 * 3;
END;
```

สิ่งอ้างอิงที่เกี่ยวข้อง

"ข้อควรพิจารณา Threads" ในหน้า 198

ฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) ที่ถูกกำหนด เป็น FENCED รันในงานเดียวกันกับคำสั่ง SQL ที่เรียกฟังก์ชันดังกล่าว อย่างไรก็ตาม UDF รันใน thread ของระบบ แยกจาก thread ที่รัน คำสั่ง SQL

ข้อควรพิจารณาในการบันทึกและการกู้คืน:

เมื่อฟังก์ชันภายนอกที่เชื่อมโยงกับโปรแกรมภายนอก ILE หรือเซอร์วิสโปรแกรมถูกสร้างขึ้นมา จะมีการพยายามบันทึกแอ็ตทริบิวต์ของฟังก์ชันในโปรแกรม หรืออ็อบเจ็กต์เซอร์วิสโปรแกรมที่เชื่อมโยงอยู่

ถ้าอ็อบเจ็กต์ *PGM หรือ *SRVPGM ถูกบันทึกแล้วถูกเรียกคืนให้กับระบบนี้หรือระบบอื่นแล้ว, แคตาล็อจะอัพเดตแอ็ตทริบิวต์เหล่านั้นให้โดยอัตโนมัติ. ถ้าแอ็ตทริบิวต์ของฟังก์ชันไม่สามารถบันทึกได้, แล้วแคตาล็อกจะไม่อัพเดตให้โดยอัตโนมัติ และผู้ใช้ต้องสร้างฟังก์ชันภายนอกไว้บนระบบใหม่. แอ็ตทริบิวต์สามารถบันทึกสำหรับฟังก์ชันภายนอกได้ภายใต้ข้อจำกัดดังต่อไปนี้:

- โปรแกรมไลบรารีภายนอกต้องไม่ใช่ QSYS หรือ QSYS2.
- โปรแกรมภายนอกต้องมีอยู่จริงเมื่อคำสั่ง CREATE FUNCTION ถูกเรียกใช้งาน.
- โปรแกรมภายนอกต้องเป็นอ็อบเจ็กต์ ILE *PGM หรือ *SRVPGM.
- โปรแกรมภายนอกหรือเซอร์วิสโปรแกรมต้องมีคำสั่ง SQL อย่างน้อยหนึ่งคำสั่ง.

ถ้าอ็อบเจ็กต์โปรแกรมไม่สามารถอัพเดตได้, ฟังก์ชันจะยังถูกสร้างอยู่.

ตัวอย่าง: โค้ด UDF

ตัวอย่างต่อไปนี้จะแสดงวิธีการใช้โค้ด user-defined function (UDF) โดยใช้ฟังก์ชัน SQL และฟังก์ชันภายนอก

ตัวอย่าง: Square ของ UDF หมายเลข:

สมมติว่า คุณต้องการฟังก์ชันที่ส่งคืนค่ายกกำลังของตัวเลข

คำสั่งเคียวรี่คือ:

```
SELECT SQUARE(myint) FROM mytable
```

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ "สิทธิในรหัส และข้อมูลถ้อยแถลง" ในหน้า 353.

ตัวอย่างต่อไปนี้แสดงวิธีการกำหนด UDF หลายๆ วิธี

การใช้ฟังก์ชัน SQL

คำสั่ง CREATE FUNCTION:

```
CREATE FUNCTION SQUARE( inval INT) RETURNS INT
LANGUAGE SQL
SET OPTION DBGVIEW=*SOURCE
BEGIN
RETURN(inval*inval);
END
```

โปรแกรมนี้จะสร้างฟังก์ชัน SQL ที่คุณสามารถดีบักได้.

การใช้ฟังก์ชันภายนอก, ลักษณะของพารามิเตอร์ SQL

คำสั่ง CREATE FUNCTION:

```
CREATE FUNCTION SQUARE(INT) RETURNS INT CAST FROM FLOAT
LANGUAGE C
EXTERNAL NAME 'MYLIB/MATH(SQUARE)'
DETERMINISTIC
NO SQL
NO EXTERNAL ACTION
PARAMETER STYLE SQL
ALLOW PARALLEL
```

โค้ด:

```
void SQUARE(int *inval,
double *outval,
short *inind,
short *outind,
char *sqlstate,
char *funcname,
char *specname,
char *msgtext)
 {
if (*inind<0)
```

```
  *outind=-1;
 else
   {
    *outval=*inval;
    *outval=(*outval)*(*outval);
    *outind=0;
   }
 return;
 }
```

วิธีการสร้างเซอร์วิสโปรแกรมภายนอกเพื่อให้ดีบักได้:

```
CRTCMOD MODULE(mylib/square) DBGVIEW(*SOURCE)
CRTSRVPGM SRVPGM(mylib/math) MODULE(mylib/square)
  EXPORT(*ALL) ACTGRP(*CALLER)
```

การใช้ฟังก์ชันภายนอก, ลักษณะของพารามิเตอร์ GENERAL

คำสั่ง CREATE FUNCTION:

```
CREATE FUNCTION SQUARE(INT) RETURNS INT CAST FROM FLOAT
LANGUAGE C
EXTERNAL NAME 'MYLIB/MATH(SQUARE)'
DETERMINISTIC
NO SQL
NO EXTERNAL ACTION
PARAMETER STYLE GENERAL
ALLOW PARALLEL
```

โค้ด:

```
double SQUARE(int *inval)
{
  double outval;
  outval=*inval;
  outval=outval*outval;
 return(outval);
 }
```

วิธีการสร้างเซอร์วิสโปรแกรมภายนอกเพื่อให้ดีบักได้:

```
CRTCMOD MODULE(mylib/square) DBGVIEW(*SOURCE)

   CRTSRVPGM SRVPGM(mylib/math) MODULE(mylib/square)
  EXPORT(*ALL) ACTGRP(*CALLER)
```

ตัวอย่าง: ตัวนับ:

สมมติว่าคุณต้องการกำหนดจำนวนแถวในคำสั่ง SELECT ดังนั้นคุณจึงเขียน user-defined function (UDF) ซึ่งเพิ่ม และส่งคืนตัวนับ

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ "สิทธิในรหัส และข้อมูลถ้อยแถลง" ในหน้า 353.

ตัวอย่างนี้มีการใช้ ฟังก์ชันภายนอกกับรูปแบบพารามิเตอร์ DB2 SQL และ scratchpad.

```
CREATE FUNCTION COUNTER ()
       RETURNS INT
       SCRATCHPAD
       NOT DETERMINISTIC
       NO SQL
       NO EXTERNAL ACTION
       LANGUAGE C
       PARAMETER STYLE DB2SQL
       EXTERNAL NAME 'MYLIB/MATH(ctr)'
       DISALLOW PARALLEL

/* structure scr defines the passed scratchpad for the function "ctr" */
struct scr {
   long len;
   long countr;
   char not_used[92];
};

void ctr (
   long *out,                          /* output answer (counter) */
   short *outnull,                     /* output NULL indicator */
   char *sqlstate,                     /* SQL STATE */
   char *funcname,                     /* function name */
   char *specname,                     /* specific function name */
   char *mesgtext,                     /* message text insert */
   struct scr *scratchptr) {           /* scratch pad */

   *out = ++scratchptr->countr;      /* increment counter & copy out */
   *outnull = 0;
   return;
}
/* end of UDF : ctr */
```

สำหรับ UDF นี้, โปรดสังเกตว่า:

- ไม่มีการระบุอากิวเมนต์ SQL อินพุต, แต่มีการส่งคืนค่า.
- UDF จะอยู่ต่อท้ายอากิวเมนต์อินพุต scratchpad หลังจากอากิวเมนต์การติดตามมาตรฐาน 4 อากิวเมนต์, ซึ่งได้แก่ *SQL-state*, *function-name*, *specific-name*, และ *message-text*.
- UDF จะรวมถึง definition โครงสร้างเพื่อแม็พ scratchpad ที่ถูกส่งผ่าน.
- ไม่มีการกำหนดพารามิเตอร์อินพุต. การดำเนินการนี้สอดคล้องกับโค้ด.
- SCRATCHPAD ถูกโค้ด, ทำให้ DB2 มีการจัดสรร, initialize และส่งผ่าน อากิวเมนต์ scratchpad ได้อย่างเหมาะสม.
- คุณต้องระบุ scratchpad ให้เป็น NOT DETERMINISTIC, เพราะ scratchpad ขึ้นอยู่กับอากิวเมนต์อินพุต SQL มากกว่า, (ไม่ใช่ในกรณีนี้).
- คุณระบุ DISALLOW PARALLEL ไว้อย่างถูกต้องแล้ว, เพราะฟังก์ชันการแก้ไขของ UDF ขึ้นอยู่กับ single scratchpad.

ตัวอย่าง: ฟังก์ชันตารางอากาศ:

สมมติว่าคุณเขียนฟังก์ชันตารางที่ รายงานข้อมูลอากาศของเมืองต่างๆ ในสหรัฐอเมริกา

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

วันที่บันทึกสภาพอากาศสำหรับเมืองเหล่านี้จะถูกอ่านจากไฟล์ภายนอก, ตามที่ระบุไว้ในข้อสังเกตซึ่งอยู่ในโปรแกรมตัวอย่าง. ข้อมูลจะรวมถึงชื่อของเมืองตามด้วยข้อมูลอากาศของเมืองนั้น. โดยจะใช้รูปแบบเดียวกันนี้สำหรับเมืองอื่นๆ ด้วย.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sqludf.h> /* for use in compiling User Defined Function */


#define   SQL_NOTNULL   0   /* Nulls Allowed - Value is not Null */
#define   SQL_ISNULL   -1   /* Nulls Allowed - Value is Null */
#define   SQL_TYP_VARCHAR 448
#define   SQL_TYP_INTEGER 496
#define   SQL_TYP_FLOAT 480

/* Short and long city name structure */
typedef struct {
  char * city_short ;
  char * city_long ;
} city_area ;

/* Scratchpad data */ (See note 1)
/* Preserve information from one function call to the next call */
typedef struct {
  /* FILE * file_ptr; if you use weather data text file */
  int file_pos ;  /* if you use a weather data buffer */
} scratch_area ;

/* Field descriptor structure */
typedef struct {
  char fld_field[31] ;                    /* Field data */
  int  fld_ind ;            /* Field null indicator data */
  int  fld_type ;                         /* Field type */
  int  fld_length ;  /* Field length in the weather data */
  int  fld_offset ;  /* Field offset in the weather data */
} fld_desc ;

/* Short and long city name data */
city_area cities[] = {
  { "alb", "Albany, NY"                },
  { "atl", "Atlanta, GA"               },
  .
  .
  .
  { "wbc", "Washington DC, DC"         },
  /* You may want to add more cities here */

  /* Do not forget a null termination */
  { ( char * ) 0, ( char * ) 0         }
} ;

/* Field descriptor data */
fld_desc fields[] = {
  { ", SQL_ISNULL, SQL_TYP_VARCHAR, 30,  0 }, /* city          */
  { ", SQL_ISNULL, SQL_TYP_INTEGER,  3,  2 }, /* temp_in_f     */
```

```
  { ", SQL_ISNULL, SQL_TYP_INTEGER,  3,  7 }, /* humidity      */
  { ", SQL_ISNULL, SQL_TYP_VARCHAR,  5, 13 }, /* wind          */
  { ", SQL_ISNULL, SQL_TYP_INTEGER,  3, 19 }, /* wind_velocity */
  { ", SQL_ISNULL, SQL_TYP_FLOAT,    5, 24 }, /* barometer     */
  { ", SQL_ISNULL, SQL_TYP_VARCHAR, 25, 30 }, /* forecast      */
  /* You may want to add more fields here */

  /* Do not forget a null termination */
  { ( char ) 0, 0, 0, 0, 0 }
} ;

/* Following is the weather data buffer for this example. You */
/* may want to keep the weather data in a separate text file. */
/* Uncomment the following fopen() statement.  Note that you  */
/* need to specify the full path name for this file.          */
char * weather_data[] = {
   "alb.forecast",
   "   34   28%    wnw   3  30.53 clear",
   "atl.forecast",
   "   46   89%   east  11  30.03 fog",
   .
   .
   .
   "wbc.forecast",
   "   38   96%    ene  16  30.31 light rain",
   /* You may want to add more weather data here */

   /* Do not forget a null termination */
   ( char * ) 0
} ;

#ifdef __cplusplus
extern "C"
#endif
/* This is a subroutine. */
/* Find a full city name using a short name */
int get_name( char * short_name, char * long_name ) {

    int name_pos = 0 ;

    while ( cities[name_pos].city_short != ( char * ) 0 ) {
       if (strcmp(short_name, cities[name_pos].city_short) == 0) {
          strcpy( long_name, cities[name_pos].city_long ) ;
          /* A full city name found */
          return( 0 ) ;
       }
       name_pos++ ;
    }
    /* can not find such city in the city data */
    strcpy( long_name, "Unknown City" ) ;
    return( -1 ) ;

}

#ifdef __cplusplus
```

```
extern "C"
#endif
/* This is a subroutine. */
/* Clean all field data and field null indicator data */
int clean_fields( int field_pos ) {

    while (fields[field_pos].fld_length !=0 ) {
       memset( fields[field_pos].fld_field, '\0', 31 ) ;
       fields[field_pos].fld_ind = SQL_ISNULL ;
       field_pos++ ;
    }
    return( 0 ) ;

}

#ifdef __cplusplus
extern "C"
#endif
/* This is a subroutine. */
/* Fills all field data and field null indicator data ... */
/* ... from text weather data */
int get_value( char * value, int field_pos ) {

    fld_desc * field ;
    char field_buf[31] ;
    double * double_ptr ;
    int * int_ptr, buf_pos ;

    while ( fields[field_pos].fld_length != 0 ) {
       field = &fields[field_pos] ;
       memset( field_buf, '\0', 31 ) ;
       memcpy( field_buf,
               ( value + field->fld_offset ),
               field->fld_length ) ;
       buf_pos = field->fld_length ;
       while ( ( buf_pos > 0 ) &&
               ( field_buf[buf_pos] == ' ' ) )
          field_buf[buf_pos--] = '\0' ;
       buf_pos = 0 ;
       while ( ( buf_pos < field->fld_length ) &&
               ( field_buf[buf_pos] == ' ' ) )
          buf_pos++ ;
       if ( strlen( ( char * ) ( field_buf + buf_pos ) ) > 0 ||
            strcmp( ( char * ) ( field_buf + buf_pos ), "n/a") != 0 ) {
          field->fld_ind = SQL_NOTNULL ;

          /* Text to SQL type conversion */
          switch( field->fld_type ) {
            case SQL_TYP_VARCHAR:
                 strcpy( field->fld_field,
                         ( char * ) ( field_buf + buf_pos ) ) ;
                 break ;
            case SQL_TYP_INTEGER:
                 int_ptr = ( int * ) field->fld_field ;
                 *int_ptr = atoi( ( char * ) ( field_buf + buf_pos ) ) ;
```

```
                break ;
            case SQL_TYP_FLOAT:
                double_ptr = ( double * ) field->fld_field ;
                *double_ptr = atof( ( char * ) ( field_buf + buf_pos ) ) ;
                break ;
            /* You may want to add more text to SQL type conversion here */
          }

       }
       field_pos++ ;
    }
    return( 0 ) ;

}

#ifdef __cplusplus
extern "C"
#endif
void SQL_API_FN weather( /* Return row fields */
              SQLUDF_VARCHAR * city,
              SQLUDF_INTEGER * temp_in_f,
              SQLUDF_INTEGER * humidity,
              SQLUDF_VARCHAR * wind,
              SQLUDF_INTEGER * wind_velocity,
              SQLUDF_DOUBLE  * barometer,
              SQLUDF_VARCHAR * forecast,
              /* You may want to add more fields here */

              /* Return row field null indicators */
              SQLUDF_NULLIND * city_ind,
              SQLUDF_NULLIND * temp_in_f_ind,
              SQLUDF_NULLIND * humidity_ind,
              SQLUDF_NULLIND * wind_ind,
              SQLUDF_NULLIND * wind_velocity_ind,
              SQLUDF_NULLIND * barometer_ind,
              SQLUDF_NULLIND * forecast_ind,
              /* You may want to add more field indicators here */

              /* UDF always-present (trailing) input arguments */
              SQLUDF_TRAIL_ARGS_ALL
            ) {

  scratch_area * save_area ;
  char line_buf[81] ;
  int line_buf_pos ;

  /* SQLUDF_SCRAT is part of SQLUDF_TRAIL_ARGS_ALL */
  /* Preserve information from one function call to the next call */
  save_area = ( scratch_area * ) ( SQLUDF_SCRAT->data ) ;

  /* SQLUDF_CALLT is part of SQLUDF_TRAIL_ARGS_ALL */
  switch( SQLUDF_CALLT ) {

    /* First call UDF: Open table and fetch first row */
    case SQL_TF_OPEN:
```

```
     /* If you use a weather data text file specify full path */
     /* save_area->file_ptr = fopen("tblsrv.dat","r"); */
     save_area->file_pos = 0 ;
     break ;

/* Normal call UDF: Fetch next row */ (See note 2)
case SQL_TF_FETCH:
     /* If you use a weather data text file */
     /* memset(line_buf, '\0', 81); */
     /* if (fgets(line_buf, 80, save_area->file_ptr) == NULL) { */
     if ( weather_data[save_area->file_pos] == ( char * ) 0 ) {

        /* SQLUDF_STATE is part of SQLUDF_TRAIL_ARGS_ALL */
        strcpy( SQLUDF_STATE, "02000" ) ;

        break ;
     }
     memset( line_buf, '\0', 81 ) ;
     strcpy( line_buf, weather_data[save_area->file_pos] ) ;
     line_buf[3] = '\0' ;

     /* Clean all field data and field null indicator data */
     clean_fields( 0 ) ;

     /* Fills city field null indicator data */
     fields[0].fld_ind = SQL_NOTNULL ;

     /* Find a full city name using a short name */
     /* Fills city field data */
     if ( get_name( line_buf, fields[0].fld_field ) == 0 ) {
        save_area->file_pos++ ;
        /* If you use a weather data text file */
        /* memset(line_buf, '\0', 81); */
        /* if (fgets(line_buf, 80, save_area->file_ptr) == NULL) { */
        if ( weather_data[save_area->file_pos] == ( char * ) 0 ) {
           /* SQLUDF_STATE is part of SQLUDF_TRAIL_ARGS_ALL */
           strcpy( SQLUDF_STATE, "02000" ) ;
           break ;
        }
        memset( line_buf, '\0', 81 ) ;
        strcpy( line_buf, weather_data[save_area->file_pos] ) ;
        line_buf_pos = strlen( line_buf ) ;
        while ( line_buf_pos > 0 ) {
           if ( line_buf[line_buf_pos] >= ' ' )
              line_buf_pos = 0 ;
           else {
              line_buf[line_buf_pos] = '\0' ;
              line_buf_pos-- ;
           }
        }
     }

     /* Fills field data and field null indicator data ... */
     /* ... for selected city from text weather data */
     get_value( line_buf, 1 ) ;  /* Skips city field */
```

```
      /* Builds return row fields */
      strcpy( city, fields[0].fld_field ) ;
      memcpy( (void *) temp_in_f,
              fields[1].fld_field,
              sizeof( SQLUDF_INTEGER ) ) ;
      memcpy( (void *) humidity,
              fields[2].fld_field,
              sizeof( SQLUDF_INTEGER ) ) ;
      strcpy( wind, fields[3].fld_field ) ;
      memcpy( (void *) wind_velocity,
              fields[4].fld_field,
              sizeof( SQLUDF_INTEGER ) ) ;
      memcpy( (void *) barometer,
              fields[5].fld_field,
              sizeof( SQLUDF_DOUBLE ) ) ;
      strcpy( forecast, fields[6].fld_field ) ;

      /* Builds return row field null indicators */
      memcpy( (void *) city_ind,
              &(fields[0].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) temp_in_f_ind,
              &(fields[1].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) humidity_ind,
              &(fields[2].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) wind_ind,
              &(fields[3].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) wind_velocity_ind,
              &(fields[4].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) barometer_ind,
              &(fields[5].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) forecast_ind,
              &(fields[6].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;

      /* Next city weather data */
      save_area->file_pos++ ;

      break ;

  /* Special last call UDF for clean up (no real args!): Close table */ (See note 3)
  case SQL_TF_CLOSE:
      /* If you use a weather data text file */
      /* fclose(save_area->file_ptr); */
      /* save_area->file_ptr = NULL; */
      save_area->file_pos = 0 ;
      break ;
```

```
  }
}
```

เมื่ออ้างถึงหมายเหตุที่ฝังอยู่ในโค้ด UDF นี้ โปรดรับทราบประเด็นต่อไปนี้:

1. Scratchpad ถูกกำหนดนิยามไว้แล้ว. ตัวแปร `แถว` ถูก initialize ในการเรียกแบบ OPEN, และ `iptr` array และตัวแปร `nbr_rows` ถูกเติมเต็มด้วยฟังก์ชัน *mystery* ขณะเปิดใช้.
2. FETCH จะสำรวจ `iptr` array, โดยการใช้แถวเป็นดรรชนี, และย้ายค่าผลตอบแทนจากส่วนประกอบปัจจุบันของ `iptr` ไปยังตำแหน่งที่ถูกชี้โดยตัวชี้ค่าผลลัพธ์ `out_c1`, `out_c2`, และ `out_c3`.
3. CLOSE จะลบข้อมูลในหน่วยเก็บของ OPEN และ anchor ไว้ใน scratchpad

ต่อไปนี้คือคำสั่ง CREATE FUNCTION สำหรับ UDF นี้:

```
CREATE FUNCTION tfweather_u()
  RETURNS TABLE (CITY VARCHAR(25),
                 TEMP_IN_F INTEGER,
                 HUMIDITY INTEGER,
                 WIND VARCHAR(5),
                 WIND_VELOCITY INTEGER,
                 BAROMETER FLOAT,
                 FORECAST VARCHAR(25))
  SPECIFIC tfweather_u
  DISALLOW PARALLEL
  NOT FENCED
  DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  SCRATCHPAD
  NO FINAL CALL
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  EXTERNAL NAME 'LIB1/WEATHER(weather)';
```

สำหรับ definition ของฟังก์ชันตารางนี้ โปรดรับทราบ ประเด็นต่อไปนี้:

- คำสั่งนี้ไม่ได้ใช้งานอินพุตใดๆ และส่งคืนคอลัมน์เอาต์พุตเจ็ด คอลัมน์
- มีการระบุ SCRATCHPAD ดังนั้น DB2 ได้จัดสรร เตรียมข้อมูลเบื้องต้น และ ส่งผ่านอากิวเมนต์ scratchpad อย่างเหมาะสม
- มีการระบุ NO FINAL CALL.
- ฟังก์ชันถูกระบุเป็น NOT DETERMINISTIC, เพราะฟังก์ชันนั้นขึ้นอยู่กับสิ่งอื่นที่มากกว่าอากิวเมนต์อินพุต SQL. นั่นคือ, ฟังก์ชันนี้ขึ้นอยู่กับฟังก์ชัน mystery และเราเข้าใจว่าข้อมูลอาจแตกต่างกันไปในแต่ละการทำงาน.
- CARDINALITY 100 ซึ่งเป็นค่าดีฟอลต์ เป็นจำนวนแถวที่คาดว่าจะส่งกลับโดยประมาณ ที่จัดเตรียมไว้สำหรับ DB2 optimizer
- DBINFO ไม่ถูกใช้งาน, และไม่มีการ optimization เพื่อส่งคืนคอลัมน์ที่จำเป็นสำหรับคำสั่งเฉพาะที่อ้างอิงถึงฟังก์ชันนี้.

หากต้องการเลือกแถวทั้งหมดที่สร้างขึ้นโดยฟังก์ชันตารางนี้, โปรดใช้การสืบค้นต่อไปนี้:

```
SELECT *
    FROM TABLE (tfweather_u())x
```

การใช้ UDF ในคำสั่ง SQL

สามารถเรียกใช้งานฟังก์ชันที่ผู้ใช้กำหนดเอง (UDF) แบบ Scalar และแบบคอลัมน์ภายในคำสั่ง SQL เกือบทุกๆที่ ที่ใช้งานนิพจน์ได้ เราสามารถเรียกใช้ตาราง UDF ใน FROM clause ของคำสั่ง SELECT ด้านล่างนี้คือข้อจำกัดบางประการสำหรับการใช้ UDF

- UDF และฟังก์ชันที่ระบบสร้างขึ้นจะไม่สามารถถูกระบุในการตรวจสอบข้อจำกัดได้. การตรวจสอบข้อจำกัดยังคงไม่สามารถบรรจุการอ้างอิงถึงฟังก์ชันในตัวบางอย่าง ซึ่งปฏิบัติโดยระบบเช่นเดียวกับ UDFs.
- UDF ภายนอก, SQL UDFS และฟังก์ชันในตัว DLVALUE, DLURLPATH, DLURLPATHONLY, DLURLSCHEME, DLURLCOMPLETE, และ DLURLSERVER ไม่สามารถอ้างถึงได้จากอนุประโยค ORDER BY หรือ GROUP BY, นอกจากว่าคำสั่ง SQL จะอ่านได้อย่างเดียวเท่านั้นและมีการอนุญาตให้ประมวลผล (ALWCPYDTA(*YES) หรือ (*OPTIMIZE))ได้ชั่วคราว.

การใช้ตัวทำเครื่องหมายพารามิเตอร์ หรือค่า NULL ในอากิวเมนต์ฟังก์ชัน:

คุณไม่สามารถใช้ตัวทำเครื่องหมายพารามิเตอร์หรือค่า NULL เป็น ฟังก์ชันอากิวเมนต์ เนื่องจากการ resolve ฟังก์ชันจะไม่รู้ว่าชนิดข้อมูลของอากิวเมนต์ จะมีค่าเป็นอะไร ดังนั้นจึงไม่สามารถ resolve การอ้างอิงได้

คุณไม่สามารถโค้ดสิ่งต่อไปนี้:

```
BLOOP(?)
```

หรือ

```
BLOOP(NULL)
```

คุณสามารถใช้ค่ากำหนด CAST เพื่อเตรียมชนิดข้อมูลสำหรับตัวทำเครื่องหมายพารามิเตอร์ หรือค่า NULL ที่ฟังก์ชันสามารถใช้ได้:

```
BLOOP(CAST(? AS INTEGER))
```

หรือ

```
BLOOP(CAST(NULL AS INTEGER))
```

การใช้การอ้างอิงฟังก์ชันตามเกณฑ์:

ถ้าคุณใช้การอ้างอิงฟังก์ชันที่ครบตามเกณฑ์, คุณต้องจำกัดการค้นหาสำหรับฟังก์ชันที่ตรงกันกับแบบแผนที่ระบุ

ตัวอย่างเช่น, คุณมีคำสั่งต่อไปนี้:

```
SELECT PABLO.BLOOP(COLUMN1) FROM T
```

เฉพาะฟังก์ชัน BLOOP ใน Schema PABLO เท่านั้นที่ถูกพิจารณา. มันไม่สำคัญว่าผู้ใช้คือ SERGE เป็นผู้นิยามฟังก์ชัน BLOOP, หรือฟังก์ชันนั้นคือฟังก์ชันในตัว BLOOP หรือไม่. สมมติว่า ผู้ใช้คือ PABLO เป็นผู้นิยามฟังก์ชัน BLOOP สองฟังก์ชันในแบบแผนของเขา:

```
CREATE FUNCTION BLOOP (INTEGER) RETURNS ...
CREATE FUNCTION BLOOP (DOUBLE) RETURNS ...
```

ดังนั้น BLOOP จะมีการโหลดมากเกินไปใน PABLO schema, และ algorithm ของการเลือก ฟังก์ชัน จะเลือก BLOOP ที่ดีที่สุด, โดยขึ้นอยู่กับชนิดข้อมูลของอากิวเมนต์, COLUMN1. ในกรณีนี้, PABLO.BLOOP ทั้งสองฟังก์ชันรับอากิวเมนต์ที่เป็นตัว

เลข, และถ้า COLUMN1 ไม่ใช่ชนิดข้อมูลแบบตัวเลขแล้ว, คำสั่งนี้จะทำงานไม่สำเร็จ. ในทางตรงกันข้าม ถ้า COLUMN1 เป็น SMALLINT หรือ INTEGER อย่างใดอย่างหนึ่งแล้ว การเลือกฟังก์ชัน จะ resolve ตัว BLOOP ตัวแรก ในขณะที่ถ้า COLUMN1 เป็น DECIMAL หรือ DOUBLE แล้ว BLOOP ตัวที่สองจะถูกเลือก

มีจุดที่น่าสนใจหายอย่างเกี่ยวกับตัวอย่างนี้:

1. มันแสดงการเลื่อนระดับของอากิวเมนต์. ฟังก์ชัน BLOOP ถูกนิยามด้วยพารามิเตอร์แบบ INTEGER, แต่คุณสามารถผ่านค่าเป็นอากิวเมนต์แบบ SMALLINT ได้. algorithm การเลือกฟังก์ชันจะสนับสนุนการส่งเสริมระหว่างชนิดข้อมูลในตัว และ DB2 ที่ทำการแปลงค่าข้อมูลที่เหมาะสม.
2. ถ้าในบางเหตุผลที่คุณต้องการเรียกฟังก์ชัน BLOOP ตัวที่สองด้วยอากิวเมนต์แบบ SMALLINT หรือ INTEGER แล้ว, คุณต้องกระทำโดยตรงในคำสั่งของคุณดังต่อไปนี้:

   ```
   SELECT PABLO.BLOOP( DOUBLE(COLUMN1)) FROM T
   ```

3. ถ้าคุณต้องการเรียกใช้ BLOOP แรกด้วยอากิวเมนต์แบบ DECIMAL หรือ DOUBLE, คุณมีตัวเลือกในการกระทำโดยตรง, โดยขึ้นอยู่กับจุดประสงค์ของคุณ:

   ```
   SELECT PABLO.BLOOP(INTEGER(COLUMN1)) FROM T
   SELECT PABLO.BLOOP(FLOOR(COLUMN1)) FROM T
   ```

สิ่งอ้างอิงที่เกี่ยวข้อง

"การใช้การอ้างอิงฟังก์ชันที่ไม่ครบตามเกณฑ์"
คุณสามารถใช้การอ้างอิงฟังก์ชันที่ไม่ครบตามเกณฑ์แทนการอ้างอิงฟังก์ชันที่ครบตามเกณฑ์ได้ เมื่อค้นหาฟังก์ชันที่ตรงกันโดยปกติแล้ว DB2 จะใช้ ฟังก์ชันพาธเพื่อให้เหมาะสมกับการอ้างอิง

"การนิยาม UDT" ในหน้า 238
คุณกำหนด user-defined type (UDT) โดยใช้ คำสั่ง CREATE DISTINCT TYPE

การใช้การอ้างอิงฟังก์ชันที่ไม่ครบตามเกณฑ์:

คุณสามารถใช้การอ้างอิงฟังก์ชันที่ไม่ครบตามเกณฑ์แทนการอ้างอิงฟังก์ชันที่ครบตามเกณฑ์ได้ เมื่อค้นหาฟังก์ชันที่ตรงกันโดยปกติแล้ว DB2 จะใช้ ฟังก์ชันพาธเพื่อให้เหมาะสมกับการอ้างอิง

ในกรณีของคำสั่ง DROP FUNCTION หรือ COMMENT ON FUNCTION การอ้างอิงจะถูกทำให้ครบตามเกณฑ์โดยใช้ Authorization ID ปัจจุบัน ถ้าการอ้างนั้นไม่ครบตามเกณฑ์สำหรับการตั้งชื่อ *SQL หรือการตั้งชื่อ *LIBL สำหรับการตั้งชื่อ *SYS ดังนั้น, มันจึงสำคัญที่คุณจะรู้ว่าพาธฟังก์ชันของคุณเป็นอะไร, และ ฟังก์ชันที่ขัดแย้งอะไร, ถ้ามีอยู่, ที่เกิดขึ้นในแบบแผนของพาธฟังก์ชันในปัจจุบัน ของคุณ. ตัวอย่างเช่น สมมติว่า คุณเป็น PABLO และคำสั่ง SQL แบบ static ของคุณเป็นดังนี้, โดยที่ COLUMN1 คือชนิดข้อมูลแบบ INTEGER:

```
SELECT BLOOP(COLUMN1) FROM T
```

คุณได้สร้างฟังก์ชัน BLOOP สองฟังก์ชันในส่วนของการใช้การอ้างอิงฟังก์ชันตามเกณฑ์, และคุณต้องการ และคาดว่า หนึ่งในฟังก์ชันนั้นจะถูกเลือก. ถ้าดีฟอลต์ฟังก์ชันพาธถูกใช้, BLOOP ตัวแรกจะถูกเลือก (เนื่องจาก COLUMN1 เป็น INTEGER), ถ้าไม่มี BLOOP ที่ขัดแย้งใน QSYS หรือ QSYS2:

```
"QSYS","QSYS2","PABLO"
```

อย่างไรก็ตาม, สมมุติว่าคุณลืมไปว่าคุณใช้สคริปต์สำหรับพรีคอมไพล์(precompile)และไบด์(bind)กับสิ่งที่คุณเขียนไว้เพื่อจุดประสงค์อื่น. ในสคริปต์นี้, คุณเขียนพารามิเตอร์ SQLPATH ของคุณโดยตรง เพื่อระบุฟังก์ชันพาธต่อไปนี้ เพื่อใช้ในเหตุผลอื่นที่ไม่สามารถใช้กับงานปัจจุบันของคุณได้:

```
"KATHY","QSYS","QSYS2","PABLO"
```

ถ้ามีฟังก์ชัน BLOOP ในแบบแผน KATHY, การเลือกฟังก์ชันสามารถ resolve ฟังก์ชันนั้นได้เป็นอย่างดี, และคำสั่งของคุณปฏิบัติงานโดยไม่มีข้อผิดพลาด. คุณจะไม่ได้รับการแจ้งบอก เนื่องจาก DB2 คิดว่าคุณรู้อยู่แล้วว่าคุณกำลังทำอะไร. เพราะมันเป็นความรับผิดชอบของคุณในการหาเอาต์พุตที่ผิดจากคำสั่งของคุณแล้วทำให้ถูกต้อง.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การใช้การอ้างอิงฟังก์ชันตามเกณฑ์" ในหน้า 210
ถ้าคุณใช้การอ้างอิงฟังก์ชันที่ครบตามเกณฑ์, คุณต้องจำกัดการค้นหาสำหรับฟังก์ชันที่ตรงกันกับแบบแผนที่ระบุ

ข้อสรุปของการอ้างอิงฟังก์ชัน:

สำหรับการอ้างอิงฟังก์ชันทั้งแบบครบตามเกณฑ์และไม่ครบตามเกณฑ์, algorithm การเลือกฟังก์ชันจะมองหาฟังก์ชันที่ใช้ได้ทั้งหมด ทั้งแบบในตัวและแบบผู้ใช้กำหนดเอง ที่มีชื่อ จำนวนของพารามิเตอร์ที่กำหนดเป็นอากิวเมนต์เหมือนกัน แต่ละพารามิเตอร์จะเหมือน หรือ สนับสนุนชนิดของอากิวเมนต์ที่ตรงกัน

ฟังก์ชันประยุกต์ หมายถึงฟังก์ชันในแบบแผนที่ตั้งชื่อสำหรับการอ้างอิงที่ครบตามเกณฑ์, หรือฟังก์ชันในแบบแผนของฟังก์ชันพาธสำหรับการอ้างอิงที่ไม่ครบตามเกณฑ์. algorithm จะมองหาฟังก์ชันที่ตรงที่สุด, หรือถ้าหาไม่เจอ, จึงค่อยหาฟังก์ชันที่ใกล้เคียงที่สุด. ฟังก์ชันพาธปัจจุบันจะถูกใช้, ในกรณีของการอ้างอิงที่ไม่ครบตามเกณฑ์เท่านั้น, ซึ่งเป็นปัจจัยในการตัดสิน ถ้าพบฟังก์ชันที่เหมือนกันสองฟังก์ชันในแบบแผนที่ต่างกัน.

คุณลักษณะพิเศษที่น่าสนใจก็คือ ฟังก์ชันการอ้างอิงสามารถซ้อนกันได้ ถึงแม้ว่าจะมีการอ้างอิงฟังก์ชันตัวเดียวกัน นี่เป็นความจริงสำหรับฟังก์ชันในตัว รวมถึง UDF ด้วย. อย่างไรก็ตาม มีข้อจำกัดบางอย่าง ในกรณีที่มีฟังก์ชันการรวมเข้ามาเกี่ยวข้อง

การกล่นกรองตัวอย่างก่อนหน้านี้:

```
CREATE FUNCTION BLOOP (INTEGER) RETURNS INTEGER ...
CREATE FUNCTION BLOOP (DOUBLE) RETURNS INTEGER ...
```

ดังนั้น ควรพิจารณาถึงคำสั่งต่อไปนี้:

```
SELECT BLOOP( BLOOP(COLUMN1)) FROM T
```

ถ้า COLUMN1 คือคอลัมน์ชนิด DECIMAL หรือ DOUBLE แล้ว, การอ้างอิง BLOOP ด้านในจะเรียกใช้ฟังก์ชัน BLOOP ตัวที่สองซึ่งถูกนิยามไว้ด้านบน. เนื่องจาก BLOOP นี้คืนค่าเป็น INTEGER, ดังนั้น BLOOP ด้านนอกจะเรียกใช้ BLOOP ตัวแรก.

หรืออีกทางหนึ่ง, ถ้า COLUMN1 คือคอลัมน์ชนิด SMALLINT หรือ INTEGER แล้ว, การอ้างอิง BLOOP ด้านในจะเรียกใช้ BLOOP ตัวแรกซึ่งถูกนิยามไว้ด้านบน. เนื่องจาก BLOOP นี้คืนค่าเป็น INTEGER, ดังนั้น BLOOP ด้านนอกก็เรียกใช้ BLOOP ตัวแรกด้วย. ในกรณีนี้, คุณจะมองเห็นว่ามีการอ้างอิงไปยังฟังก์ชันเดียวกันซ้อนกันอยู่.

จุดสำคัญเพิ่มเติมอีกเล็กน้อยสำหรับการอ้างอิงฟังก์ชันคือ:

- คุณสามารถนิยามฟังก์ชันด้วยชื่อของหนึ่งในตัวดำเนินการ SQL. ตัวอย่างเช่น, สมมุติว่าคุณสามารถแนบบางความหมายให้กับตัวดำเนินการ "+" สำหรับค่าที่มี Distinct Type ชนิด BOAT อยู่. คุณสามารถนิยาม UDF ดังต่อไปนี้:

  ```
  CREATE FUNCTION "+" (BOAT, BOAT) RETURNS ...
  ```

  Then you can write the following valid SQL statement:

```
SELECT "+"(BOAT_COL1, BOAT_COL2)
FROM BIG_BOATS
WHERE BOAT_OWNER = 'Nelson Mattos'
```

คุณไม่ได้รับสิทธิให้โหลดตัวดำเนินการที่มีเงื่อนไขในตัวมากเกินไป เช่น >, =, LIKE, IN, และอื่น, ในทำนองเดียวกันนี้.

- อัลกอริธึมการเลือกฟังก์ชันจะไม่พิจารณาบริบทของการอ้างอิงในการเลือกฟังก์ชัน. ดูที่ฟังก์ชัน BLOOP นี้, ซึ่งถูกดัดแปลงจากก่อนหน้านี้เล็กน้อย:

  ```
  CREATE FUNCTION BLOOP (INTEGER) RETURNS INTEGER ...
  CREATE FUNCTION BLOOP (DOUBLE) RETURNS CHAR(10)...
  ```

  แล้วสมมุติว่าคุณเขียนคำสั่ง SELECT ดังต่อไปนี้:

  ```
  SELECT 'ABCDEFG' CONCAT BLOOP(SMALLINT_COL) FROM T
  ```

  เนื่องจากการจับคู่ที่เหมาะสมที่สุด, เลือกโดยใช้อากิวเมนต์ SMALLINT, จะได้ BLOOP ตัวแรกที่นิยามไว้ด้านบน, ซึ่ง operand ตัวที่สองของ CONCAT จะแปลงเป็นชนิดข้อมูล INTEGER. คำสั่งอาจไม่ส่งกลับผลลัพธ์ที่คาดหวังไว้เนื่องจากจำนวนเต็มที่ส่งกลับจะถูกทำให้เป็น VARCHAR ก่อนจะปฏิบัติคำสั่ง CONCAT . ถ้า BLOOP ตัวแรกไม่ได้ปรากฏอยู่, BLOOP ตัวอื่นจะถูกเลือกและการเรียกใช้คำสั่งอาจจะทำได้สำเร็จ.

- UDFs สามารถนิยามให้มีพารามิเตอร์หรือมีผลลัพธ์ที่มีชนิด LOB: BLOB, CLOB, หรือ DBCLOB ได้. ระบบจะ materialize ค่า LOB ทั้งหมดใน หน่วยเก็บก่อนการเรียกใช้ฟังก์ชันใดๆ, ถึงแม้ว่าต้นทางของค่าจะเป็น ตัวแปรโฮสต์ *LOB locator*. ตัวอย่างเช่น, พิจารณาส่วนของแอ็พพลิเคชันภาษา C ดังต่อไปนี้:

  ```
  EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS CLOB(150K) clob150K ;        /* LOB host var */
    SQL TYPE IS CLOB_LOCATOR clob_locator1;  /* LOB locator host var */
    char                    string[40];      /* string host var */
  EXEC SQL END DECLARE SECTION;
  ```

  ทั้งตัวแปรโฮสต์ :clob150K หรือ :clob_locator1 จะเป็นอากิวเมนต์ที่ถูกต้องสำหรับฟังก์ชันที่พารามิเตอร์ที่สอดคล้องกันถูกนิยามไว้เป็น CLOB(500K). การอ้างอิงถึง FINDSTRING ที่กำหนดไว้ใน “ตัวอย่าง: การค้นหาสตริง” ในหน้า 187 ทั้งสอง ตัวอย่างดังต่อไปนี้จะใช้ได้ในโปรแกรม:

  ```
  ... SELECT FINDSTRING (:clob150K, :string) FROM ...
  ... SELECT FINDSTRING (:clob_locator1, :string) FROM ...
  ```

- พารามิเตอร์แบบ UDF ภายนอกหรือผลลัพธ์ซึ่งมีชนิด LOB อย่างน้อยหนึ่งชนิดสามารถ ถูกสร้างด้วย modifier ชนิด AS LOCATOR. ในกรณีนี้, ค่า LOB ทั้งหมดจะไม่ถูกดึงค่าไปเก็บก่อนที่จะเรียกใช้. แทนดังนั้น, LOB LOCATOR จะถูกส่งผ่านไปยัง UDF.

  คุณยังสามารถใช้ความสามารถนี้บนพารามิเตอร์ UDF หรือผลลัพธ์ที่มีชนิดเป็น Distinct Type ที่อยู่บนพื้นฐานของ LOB ได้. ความสามารถนี้จะจำกัดใช้ได้กับ UDFs แบบภายนอก. โปรดสังเกตุว่าอากิวเมนต์ของฟังก์ชันสามารถเป็นค่า LOB ของชนิดที่นิยามไว้ใดๆก็ได้; ไม่จำเป็นว่าต้องเป็นตัวแปรโฮสต์ที่นิยามให้เป็นหนึ่งในชนิด LOCATOR. การใช้ Host variable Locators เป็นอากิวเมนต์จะไม่เกี่ยวข้องใดๆกับการใช้ AS LOCATOR ในพารามิเตอร์ของ UDF และการนิยามผลลัพธ์.

- UDFs สามารถถูกนิยามโดยใช้ Distinct Types เป็นพารามิเตอร์หรือเป็นผลลัพธ์ได้. DB2 จะส่งผ่าน ค่าไปยัง UDF ในรูปแบบของชนิดข้อมูลต้นฉบับของ distinct type.

  ค่า Distinct type ที่อยู่ในตัวแปรโฮสต์และที่ถูกใช้เป็นอากิวเมนต์ของ UDF ที่มีพารามิเตอร์ที่สอดคล้องกันถูกนิยามเป็นชนิด distinct ต้องให้ผู้ใช้แปลงข้อมูลให้เป็น distinct type โดยตรง. จะไม่มีชนิดบนภาษาโฮสต์สำหรับ Distinct Type. การกำหนดชนิดที่ strong ของ DB2 มีความจำเป็นอย่างยิ่งในที่นี่. มิฉะนั้นแล้วผลลัพธ์ที่คุณได้อาจจะคลุมเคลือ. ดังนั้น, ให้พิจารณาชนิด Distinct BOAT ที่ถูกกำหนดให้กับ BLOB ที่รับอ็อบเจ็กต์ของชนิด BOAT เป็นอากิวเมนต์. ในส่วนของแอ็พพลิเคชันภาษา C ดังต่อไปนี้, ตัวแปรโฮสต์ :ship จะเก็บค่า BLOB ที่จะถูกส่งผ่านไปยังฟังก์ชัน BOAT_COST:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB(150K) ship;
EXEC SQL END DECLARE SECTION;
```

Both of the following statements correctly resolve to the BOAT_COST function, because both cast the `:ship` host variable to type BOAT:

```
... SELECT BOAT_COST (BOAT(:ship)) FROM ...
... SELECT BOAT_COST (CAST(:ship AS BOAT)) FROM ...
```

ถ้ามี Distinct Type ชื่อ BOAT หลายตัวในฐานข้อมูล, หรือมีหลาย BOAT UDF ใน Schema อื่น, คุณต้องระมัดระวังฟังก์ชันพาธของคุณให้ดี. มิฉะนั้นแล้วผลลัพธ์ที่คุณได้อาจจะคลุมเคลือ.

ทริกเกอร์

*ทริกเกอร์*คือชุดของ action ที่รัน โดยอัตโนมัติ เมื่อมีการเปลี่ยนแปลงที่ระบุบน ตารางหรือมุมมองที่ระบุ

การเปลี่ยนแปลงดังกล่าวสามารถเป็นได้ทั้งคำสั่ง SQL INSERT, UPDATE, หรือ DELETE, หรือเป็นการแทรก, อัพเดต, หรือลบคำสั่งภาษาชั้นสูงในแอ็พพลิเคชันโปรแกรม ทริกเกอร์เป็นประโยชน์สำหรับงานอย่าง เช่น การบังคับใช้กฎธุรกิจ, การตรวจสอบความถูกต้องของข้อมูลอินพุต, และการเก็บหลักฐานการตรวจสอบ.

ทริกเกอร์สามารถกำหนดให้เป็น SQL หรือภายนอก.

สำหรับทริกเกอร์ใช้ภายนอก, จะมีการใช้คำสั่ง CRTPFTRG CL. โปรแกรมซึ่งประกอบด้วยชุดการทำงานของทริกเกอร์สามารถถูกกำหนดด้วยภาษาชั้นสูงใดที่สนับสนุนก็ได้. ทริกเกอร์ภายนอกสามารถแทรก, อัพเดต, ลบ, หรืออ่านทริกเกอร์.

สำหรับ SQL ทริกเกอร์, จะมีการใช้คำสั่ง CREATE TRIGGER. ทริกเกอร์โปรแกรมถูกกำหนดทั้งหมดโดยใช้ SQL. SQL ทริกเกอร์สามารถแทรก, อัพเดต, หรือลบทริกเกอร์.

เมื่อทริกเกอร์ถูกเชื่อมโยงเข้ากับตารางหรือมุมมอง, ตัวสนับสนุนทริกเกอร์จะเรียกทริกเกอร์โปรแกรมขึ้นมาทุกครั้งที่มีการเปลี่ยนแปลงเกิดขึ้นกับตารางหรือมุมมอง, หรือโลจิคัลไฟล์หรือมุมมองใดๆ ที่กสร้างขึ้นบนตารางหรือมุมมอง. คุณสามารถกำหนด SQL ทริกเกอร์และทริกเกอร์ใช้ภายนอกสำหรับตารางเดียวกันได้. เฉพาะ SQL ทริกเกอร์เท่านั้นที่สามารถกำหนดสำหรับมุมมองได้. ซึ่งสามารถกำหนดได้มากถึง 200 ทริกเกอร์สำหรับตารางหรือมุมมองเดียว.

การเปลี่ยนแปลงตารางแต่ละครั้งสามารถเรียกทริกเกอร์ได้ก่อนหรือหลังจากที่เกิดการเปลี่ยนแปลง. นอกจากนี้, คุณสามารถเพิ่มทริกเกอร์ใช้ *อ่าน* ซึ่งจะถูกเรียกขึ้นมาทุกครั้งที่เข้าถึงตาราง. ดังนั้น, ตารางจึงสามารถเชื่อมโยงกับทริกเกอร์ได้หลายประเภท.

- ทริกเกอร์ก่อนลบ
- ทริกเกอร์ก่อนแทรก
- ทริกเกอร์ก่อนอัพเดต
- ทริกเกอร์หลังลบ
- ทริกเกอร์หลังแทรก
- ทริกเกอร์หลังอัพเดต
- ทริกเกอร์อ่านอย่างเดียว (ทริกเกอร์ใช้ภายนอกอย่างเดียว)

การเปลี่ยนแปลงมุมมองแต่ละครั้งสามารถเรียกโปรแกรมอื่นแทนทริกเกอร์ซึ่งจะทำชุดของ action บางชุดแทนการแทรก, อัพเดต, หรือลบ. มุมมองสามารถเชื่อมโยงกับ:

- โปรแกรมแทนทริกเกอร์ใช้ลบ
- โปรแกรมแทนทริกเกอร์ใช้แทรก
- โปรแกรมแทนทริกเกอร์ใช้อัพเดต

งานที่เกี่ยวข้อง

การทำทริกเกอร์เหตุการณ์แบบอัตโนมัติในฐานข้อมูลของคุณ

SQL ทริกเกอร์

คำสั่ง SQL CREATE TRIGGER จะแสดงวิธีการสำหรับระบบจัดการ ฐานข้อมูล เพื่อให้ควบคุม, มอนิเตอร์ และจัดการกลุ่มตารางและมุมมองอย่างได้ผล ทุกครั้งที่มีการแทรก, อัพเดต หรือลบออก

คำสั่งที่ระบุใน SQL ทริกเกอร์จะถูกเรียกใช้งานทุกครั้งที่มีการแทรก, อัพเดต, หรือลบ SQL. SQL ทริกเกอร์อาจเรียกโพรซีเดอร์ที่เก็บไว้หรือฟังก์ชันที่ผู้ใช้กำหนดให้ทำการประมวลผลเพิ่มเติมเมื่อเรียกใช้งานทริกเกอร์.

ตรงกันข้ามกับโพรซีเดอร์ที่เก็บไว้, SQL ทริกเกอร์ไม่สามารถเรียกโดยตรงจากแอ็พพลิเคชันได้. แต่, SQL ทริกเกอร์จะถูกเรียกโดยระบบจัดการฐานข้อมูลเมื่อใช้การแทรก, อัพเดต, หรือลบทริกเกอร์. definition ของ SQL ทริกเกอร์จะถูกเก็บไว้ในระบบการจัดการฐานข้อมูล และถูกเรียกใช้งานโดยระบบการจัดการฐานข้อมูลเช่นกัน เมื่อตารางหรือมุมมอง SQL ที่มีการกำหนดทริกเกอร์ไว้, ถูกแก้ไข.

คุณสามารถสร้าง SQL ทริกเกอร์ได้ด้วยการระบุคำสั่ง CREATE TRIGGER SQL. อ็อบเจ็กต์ทั้งหมดที่อ้างถึงคำสั่ง CREATE TRIGGER (เช่น ตาราง และฟังก์ชัน) ต้องมีอยู่; มิฉะนั้น, ทริกเกอร์จะไม่ถูกสร้าง. คำสั่งในส่วนรูทีนของ SQL ทริกเกอร์จะถูกแปลงโดย SQL ให้เป็นอ็อบเจ็กต์โปรแกรม (*PGM). โปรแกรมจะถูกสร้างในแบบแผนที่ระบุโดย qualifier ของชื่อทริกเกอร์. ทริกเกอร์ที่ระบุจะถูกลงทะเบียนในแคตาล็อก SYSTRIGGERS, SYSTRIGDEP, SYSTRIGCOL, และ SYSTRIGUPD SQL.

หลักการที่เกี่ยวข้อง

"การดีบักรูทีน SQL" ในหน้า 222
ด้วยการระบุ SET OPTION DBGVIEW = *SOURCE ใน คำสั่ง CREATE PROCEDURE, CREATE FUNCTION หรือ CREATE TRIGGER คุณสามารถ ดีบักโปรแกรมหรือโมดูลที่สร้างขึ้นที่ระดับคำสั่ง SQL

สิ่งอ้างอิงที่เกี่ยวข้อง

SQL control statements

CREATE TRIGGER

ทริกเกอร์ BEFORE SQL:

ทริกเกอร์ BEFORE อาจไม่สามารถแก้ไขตารางได้, แต่สามารถใช้เพื่อตรวจสอบค่าอินพุตคอลัมน์, และเพื่อแก้ไขค่าคอลัมน์ที่ถูกแทรก หรืออัพเดตในตาราง

ในตัวอย่างต่อไปนี้, ทริกเกอร์ถูกใช้เพื่อกำหนดปีการเงินรายไตรมาสสำหรับบริษัท ก่อนการแทรกแถวลงในตารางเป้าหมาย.

```
CREATE TABLE TransactionTable (DateOfTransaction DATE, FiscalQuarter SMALLINT)

CREATE TRIGGER TransactionBeforeTrigger BEFORE INSERT ON TransactionTable
```

```
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
BEGIN
  DECLARE newmonth SMALLINT;
SET newmonth = MONTH(new_row.DateOfTransaction);
  IF newmonth < 4 THEN
    SET new_row.FiscalQuarter=3;
  ELSEIF newmonth < 7 THEN
    SET new_row.FiscalQuarter=4;
  ELSEIF newmonth < 10 THEN
    SET new_row.FiscalQuarter=1;
  ELSE
    SET new_row.FiscalQuarter=2;
  END IF;
END
```

ในส่วนคำสั่งแทรก SQL ข้างล่างนี้, คอลัมน์ "FiscalQuarter" ควรถูกกำหนดเป็น 2, ถ้าวันที่ปัจจุบันคือ 14 พฤศจิกายน, 2000.

```
INSERT INTO TransactionTable(DateOfTransaction)
           VALUES(CURRENT DATE)
```

SQL ทริกเกอร์ได้เข้าถึงและสามารถใช้ user-defined types (UDTs) และโพรซีเดอร์ที่เก็บไว้ในตัวอย่างต่อไปนี้, SQL ทริกเกอร์จะเรียกโพรซีเดอร์ที่เก็บไว้ขึ้นมา เพื่อเรียกใช้งานตรรกะทางธุรกิจบางข้อซึ่งได้ถูกกำหนดไว้ก่อน, ในกรณีนี้, เพื่อตั้งคอลัมน์ให้เป็นค่าซึ่งได้ถูกกำหนดไว้ก่อนสำหรับธุรกิจ.

```
CREATE DISTINCT TYPE enginesize AS DECIMAL(5,2) WITH COMPARISONS

CREATE DISTINCT TYPE engineclass AS VARCHAR(25) WITH COMPARISONS

CREATE PROCEDURE SetEngineClass(IN SizeInLiters enginesize,
                           OUT CLASS engineclass)
LANGUAGE SQL CONTAINS SQL
BEGIN
  IF SizeInLiters<2.0 THEN
    SET CLASS = 'Mouse';
  ELSEIF SizeInLiters<3.1 THEN
    SET CLASS ='Economy Class';
  ELSEIF SizeInLiters<4.0 THEN
    SET CLASS ='Most Common Class';
  ELSEIF SizeInLiters<4.6 THEN
    SET CLASS = 'Getting Expensive';
  ELSE
    SET CLASS ='Stop Often for Fillups';
  END IF;
END

CREATE TABLE EngineRatings (VariousSizes enginesize, ClassRating engineclass)

CREATE TRIGGER SetEngineClassTrigger BEFORE INSERT ON EngineRatings
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
  CALL SetEngineClass(new_row.VariousSizes, new_row.ClassRating)
```

สำหรับคำสั่งแทรก SQL ข้างล่างนี้, คอลัมน์ "ClassRating" ถูกกำหนดเป็น "Economy Class", ถ้าคอลัมน์ "VariousSizes" มีค่าเท่ากับ 3.0.

```
INSERT INTO EngineRatings(VariousSizes) VALUES(3.0)
```

SQL กำหนดให้ต้องมีตาราง, ฟังก์ชันที่ผู้ใช้กำหนด, โพรซีเดอร์ และประเภทที่ผู้ใช้กำหนดทั้งหมดขึ้นมา ก่อนจะมีการสร้าง SQL ทริกเกอร์. ในตัวอย่างข้างบน, ตาราง, โพรซีเดอร์ที่เก็บไว้, และประเภทที่ผู้ใช้กำหนดทั้งหมดจะถูกกำหนดก่อนที่จะสร้างทริกเกอร์.

ทริกเกอร์ AFTER SQL:

ทริกเกอร์ after จะทำงานหลังจากที่มีการใช้การแทรก การอัพเดต หรือการลบการเปลี่ยนแปลงที่เกี่ยวข้องกับตาราง

คุณสามารถใช้เงื่อนไข WHEN ใน SQL ทริกเกอร์เพื่อระบุเงื่อนไขได้. หากเงื่อนไขประเมินผลว่าถูกต้อง คำสั่ง SQL ในส่วนรูทีนของ SQL ทริกเกอร์จะถูกรัน หากเงื่อนไขประเมินผลว่าผิด, คำสั่ง SQL ในส่วนของรูทีนของ SQL ทริกเกอร์จะไม่ถูกรัน, และการควบคุมจะถูกส่งกลับไปที่ระบบฐานข้อมูล.

ในตัวอย่างต่อไปนี้, จะมีการประเมินผลเคียวรี่เพื่อตัดสินว่า ควรมีการรันคำสั่งในส่วนรูทีนของทริกเกอร์หรือไม่ เมื่อทริกเกอร์ถูกเรียกทำงาน.

```
CREATE TABLE TodaysRecords(TodaysMaxBarometricPressure FLOAT,
  TodaysMinBarometricPressure FLOAT)

CREATE TABLE OurCitysRecords(RecordMaxBarometricPressure FLOAT,
  RecordMinBarometricPressure FLOAT)

CREATE TRIGGER UpdateMaxPressureTrigger
AFTER UPDATE OF TodaysMaxBarometricPressure ON TodaysRecords
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
WHEN (new_row.TodaysMaxBarometricPressure>
     (SELECT MAX(RecordMaxBarometricPressure) FROM
     OurCitysRecords))
  UPDATE OurCitysRecords
         SET RecordMaxBarometricPressure =
             new_row.TodaysMaxBarometricPressure

CREATE TRIGGER UpdateMinPressureTrigger
AFTER UPDATE OF TodaysMinBarometricPressure
ON TodaysRecords
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
WHEN(new_row.TodaysMinBarometricPressure<
     (SELECT MIN(RecordMinBarometricPressure) FROM
     OurCitysRecords))
  UPDATE OurCitysRecords
         SET RecordMinBarometricPressure =
             new_row.TodaysMinBarometricPressure
```

ก่อนอื่นค่าปัจจุบันจะถูก initialize สำหรับตาราง.

```
INSERT INTO TodaysRecords VALUES(0.0,0.0)
INSERT INTO OurCitysRecords VALUES(0.0,0.0)
```

ในส่วนคำสั่งอัพเดต SQL ข้างล่างนี้, RecordMaxBarometricPressure ใน OurCitysRecords จะถูกอัพเดตโดย UpdateMaxPressureTrigger.

```
UPDATE TodaysRecords SET TodaysMaxBarometricPressure = 29.95
```

แต่ในอนาคต, หาก TodaysMaxBarometricPressure เท่ากับ 29.91 เท่านั้น, RecordMaxBarometricPressure จะไม่ได้รับการอัพเดต.

```
UPDATE TodaysRecords SET TodaysMaxBarometricPressure = 29.91
```

SQL จะยอมรับ definition ของทริกเกอร์สำหรับการทำงานของทริกเกอร์แบบเดี่ยว. ในตัวอย่างก่อนหน้านี้, มีทริกเกอร์ AFTER UPDATE สองประเภทได้แก่: UpdateMaxPressureTrigger และ UpdateMinPressureTrigger. ทริกเกอร์เหล่านี้จะปฏิบัติงานก็ต่อเมื่อมีการอัพเดตคอลัมน์เฉพาะของตาราง TodaysRecords

ทริกเกอร์ AFTER อาจแก้ไขตาราง. ในตัวอย่างข้างบน, การดำเนินการ UPDATE จะถูกนำมาใช้กับตารางที่สอง. โปรดสังเกตว่าควรหลีกเลี่ยงการแทรกและการอัพเดตแบบเรียกซ้ำ. ระบบจัดการฐานข้อมูลจะจบการทำงาน หากถึงระดับการซ้อนภายในสูงสุดของทริกเกอร์. คุณสามารถหลีกเลี่ยงการเรียกซ้ำได้ด้วยการเพิ่มตรรกะแบบมีเงื่อนไข เพื่อที่จะออกจากการแทรกหรืออัพเดตก่อนที่จะถึงระดับการซ้อนภายในสูงสุด. คุณควรหลีกเลี่ยงสถานการณ์เดียวกันนี้ในเครือข่ายทริกเกอร์ซึ่งมีการต่อเรียงแบบเรียกซ้ำผ่านเครือข่ายทริกเกอร์.

ทริกเกอร์ INSTEAD OF SQL:

ทริกเกอร์ INSTEAD OF คือ SQL ทริกเกอร์ที่ถูกประมวลผล “แทนที่” คำสั่ง SQL UPDATE, DELETE หรือ INSERT. ไม่เหมือนกับทริกเกอร์ SQL BEFORE และ AFTER, ทริกเกอร์ INSTEAD OF อาจถูกกำหนดไว้บนมุมมอง ไม่ใช่ตาราง

ทริกเกอร์ INSTEAD OF อนุญาตให้มุมมอง, ซึ่งไม่ใช่ความสามารถในการแทรก, ความสามารถในการอัพเดต, หรือความสามารถในการลบ, ถูกแทรก, อัพเดต, หรือลบทิ้ง. โปรดดู CREATE VIEW สำหรับข้อมูลเพิ่มเติมเกี่ยวกับความสามารถในการลบ, ความสามารถในการอัพเดต, และความสามารถในการแทรกมุมมอง.

หลังจากที่ทริกเกอร์ SQL INSTEAD OF ถูกเพิ่มไปยังมุมมองแล้ว, มุมมองซึ่งก่อนหน้านี้สามารถอ่านได้อย่างเดียว ก็จะสามารถใช้เพื่อแทรก, อัพเดต, หรือลบการดำเนินการ. ทริกเกอร์ INSTEAD OF จะกำหนดการดำเนินงานที่จำเป็นต้องทำเพื่อคงไว้ซึ่งมุมมอง.

มุมมองสามารถใช้เพื่อควบคุมการเข้าถึงตาราง. ทริกเกอร์ INSTEAD OF สามารถดูแลรักษาการควบคุมการเข้าถึงตารางได้โดยง่าย.

การใช้ทริกเกอร์ INSTEAD OF

definition ของมุมมอง V1 ต่อไปนี้ คือ สามารถอัพเดตได้, สามารถลบได้, และสามารถแทรกได้:

```
CREATE TABLE T1 (C1 VARCHAR(10), C2 INT)
CREATE VIEW V1(X1) AS SELECT C1 FROM T1 WHERE C2 > 10
```

สำหรับคำสั่งการแทรก, C1 ลงในตาราง T1 จะถูกกำหนดค่าเป็น ’A’. C2 จะถูกกำหนดค่า NULL. ค่า NULL จะเป็นสาเหตุทำให้แถวใหม่ไม่ตรงกับเกณฑ์การเลือก C2 > 10 สำหรับมุมมอง V1

```
INSERT INTO V1 VALUES('A')
```

การเพิ่มทริกเกอร์ INSTEAD OF IOT1 สามารถแสดงค่าที่แตกต่างกันสำหรับแถวที่จะถูกเลือกโดยมุมมอง. :

```
CREATE TRIGGER IOT1 INSTEAD OF INSERT ON V1
REFERENCING NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
INSERT INTO T1 VALUES(NEW_ROW.X1, 15)
```

การทำให้มุมมองสามารถลบได้

definition ของมุมการรวม V3 ไม่สามารถอัพเดตได้, ไม่สามารถลบได้, หรือไม่สามารถแทรกได้:

```
CREATE TABLE A (A1 VARCHAR(10), A2 INT)
CREATE VIEW V1(X1) AS SELECT A1 FROM A

CREATE TABLE B (B1 VARCHAR(10), B2 INT)
CREATE VIEW V2(Y1) AS SELECT  B1 FROM B

CREATE VIEW V3(Z1, Z2) AS SELECT V1.X1, V2.Y1 FROM V1, V2 WHERE V1.X1 = 'A' AND V2.Y1 > 'B'
```

การเพิ่มทริกเกอร์ INSTEAD OF IOT2 จะทำให้มุมมอง V3 สามารถลบได้:

```
CREATE TRIGGER IOT2 INSTEAD OF DELETE ON V3
REFERENCING OLD AS OLD_ROW
FOR EACH ROW MODE DB2SQL
BEGIN
  DELETE FROM A WHERE A1 = OLD_ROW.Z1;
  DELETE FROM B WHERE B1 = OLD_ROW.Z2;
END
```

ด้วยทริกเกอร์นี้, คำสั่ง DELETE ต่อไปนี้จะสามารถใช้ได้. คำสั่งนี้จะลบแถวทั้งหมดออกจากตาราง A โดยที่ A1 มีค่าเป็น 'A', และ แถวทั้งหมดจากตาราง B โดยที่ B1 มีค่า 'X'.

```
DELETE FROM V3 WHERE Z1 = 'A' AND Z2 = 'X'
```

ทริกเกอร์ INSTEAD OF ที่มีมุมมองจะถูกกำหนดไว้บนมุมมอง

definition ของมุมมอง V2 ต่อไปนี้ที่ถูกกำหนดไว้บน V1 จะไม่สามารถแทรกได้, ไม่สามารถอัพเดตได้, หรือไม่สามารถลบได้:

```
CREATE TABLE T1 (C1 VARCHAR(10), C2 INT)
CREATE TABLE T2 (D1 VARCHAR(10), D2 INT)
CREATE VIEW V1(X1, X2) AS SELECT C1, C2 FROM T1
UNION SELECT D1, D2 FROM T2

CREATE VIEW V2(Y1, Y2) AS SELECT X1, X2 FROM V1
```

การเพิ่ม ทริกเกอร์ INSTEAD OF IOT1 ลงใน V1 จะไม่ได้ทำให้ V2 สามารถอัพเดตได้:

```
CREATE TRIGGER IOT1 INSTEAD OF UPDATE ON V1
REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
FOR EACH ROW MODE DB2SQL
BEGIN
  UPDATE T1 SET C1 = NEW_ROW.X1, C2 = NEW_ROW.X2 WHERE
    C1 = OLD_ROW.X1 AND C2 = OLD_ROW.X2;
  UPDATE T2 SET D1 = NEW_ROW.X1, D2 = NEW_ROW.D2 WHERE
    D1 = OLD_ROW.X1 AND D2 = OLD_ROW.X2;
END
```

มุมมอง V2 ยังคงไม่สามารถอัพเดตได้ เนื่องจาก definition ของมุมมอง V2 แบบเดิมยังคงไม่สามารถอัพเดตได้.

การใช้ทริกเกอร์ INSTEAD OF พร้อมด้วยทริกเกอร์ BEFORE และ AFTER

การเพิ่มของทริกเกอร์ INSTEAD OF ในมุมมองไม่ได้เป็นสาเหตุทำให้เกิดความขัดแย้งใดๆ กับทริกเกอร์ BEFORE และ AFTER ที่ถูกกำหนดบนตารางหลัก:

```
CREATE TABLE T1 (C1 VARCHAR(10), C2 DATE)
CREATE TABLE T2 (D1 VARCHAR(10))

CREATE TRIGGER AFTER1 AFTER DELETE ON T1
REFERENCING OLD AS OLD_ROW
FOR EACH ROW MODE DB2SQL
  DELETE FROM T2 WHERE D1 = OLD_ROW.C1

CREATE VIEW V1(X1, X2) AS SELECT SUBSTR(T1.C1, 1, 1), DAYOFWEEK_ISO(T1.C2) FROM T1

CREATE TRIGGER IOT1 INSTEAD OF DELETE ON V1
REFERENCING OLD AS OLD_ROW
FOR EACH ROW MODE DB2SQL
  DELETE FROM T1 WHERE C1 LIKE (OLD_ROW.X1 CONCAT '%')
```

การดำเนินการลบใดๆ สำหรับมุมมอง V1 เป็นผลทำให้ทริกเกอร์ AFTER DELETE AFTER1 ถูกเรียกใช้งานด้วย เนื่องจากทริกเกอร์ IOT1 จะทำการลบบนตาราง T1. การลบตาราง T1 เป็นสาเหตุทำให้ทริกเกอร์ AFTER1 ถูกเรียกใช้.

มุมมองที่ถูกอ้างอิง และทริกเกอร์ INSTEAD OF

เมื่อเพิ่มทริกเกอร์ INSTEAD OF ลงในมุมมอง, หาก definition ของมุมมองอ้างอิงกับมุมมองที่มีทริกเกอร์ INSTEAD OF กำหนดอยู่ด้วย, คุณควรจะกำหนดทริกเกอร์ INSTEAD OF สำหรับการดำเนินงานสามอย่างคือ, UPDATE, DELETE, และ INSERT, เพื่อหลีกเลี่ยงความสับสนว่า ความสามารถใดที่มุมมองถูกกำหนดให้มี กับความสามารถใดที่มุมมองที่ถูกอ้างอิงต้องมี.

Handlers ใน SQL ทริกเกอร์:

handler ใน SQL ทริกเกอร์เพิ่มความสามารถให้กับ SQL ทริกเกอร์ในการกู้คืนจากข้อผิดพลาด หรือข้อมูลบันทึกการทำงานเกี่ยวกับข้อผิดพลาดที่เกิดขึ้น ขณะที่ประมวลผลคำสั่ง SQL ในส่วนรูทีนของทริกเกอร์.

ในตัวอย่างต่อไปนี้, มีการกำหนด handler 2 อย่าง ได้แก่: handler แรกเพื่อจัดการกับสภาวะโอเวอร์โฟลว์ และ handler ที่สองเพื่อจัดการกับ SQL exception.

```
CREATE TABLE ExcessInventory(Description VARCHAR(50), ItemWeight SMALLINT)

CREATE TABLE YearToDateTotals(TotalWeight SMALLINT)

CREATE TABLE FailureLog(Item VARCHAR(50), ErrorMessage VARCHAR(50), ErrorCode INT)

CREATE TRIGGER InventoryDeleteTrigger
AFTER DELETE ON ExcessInventory
REFERENCING OLD AS old_row
FOR EACH ROW MODE DB2ROW
BEGIN
  DECLARE sqlcode INT;
```

```
  DECLARE invalid_number condition FOR '22003';
  DECLARE exit handler FOR invalid_number
  INSERT INTO FailureLog VALUES(old_row.Description,
        'Overflow occurred in YearToDateTotals', sqlcode);
  DECLARE exit handler FOR sqlexception
  INSERT INTO FailureLog VALUES(old_row.Description,
        'SQL Error occurred in InventoryDeleteTrigger', sqlcode);
  UPDATE YearToDateTotals SET TotalWeight=TotalWeight +
        old_row.itemWeight;
END
```

ก่อนอื่น, ค่าปัจจุบันสำหรับตารางจะถูก initialize.

```
INSERT INTO ExcessInventory VALUES('Desks',32500)
INSERT INTO ExcessInventory VALUES('Chairs',500)
INSERT INTO YearToDateTotals VALUES(0)
```

เมื่อคำสั่งการลบ SQL คำสั่งแรกข้างล่างนี้ถูกเรียกใช้งาน, ItemWeight สำหรับไอเท็ม "Desks" จะถูกเพิ่มเข้าไปในคอลัมน์ทั้งหมดสำหรับ TotalWeight ในตาราง YearToDateTotals. เมื่อคำสั่งการลบ SQL คำสั่งที่สองถูกเรียกใช้งาน, จะเกิดการโอเวอร์โฟลว์ขึ้นเมื่อ ItemWeight สำหรับไอเท็ม "Chairs" ถูกเพิ่มเข้าไปในคอลัมน์ทั้งหมดสำหรับ TotalWeight, เนื่องจากคอลัมน์จะจัดการเฉพาะค่าสูงสุดเท่ากับ 32767. เมื่อเกิดการโอเวอร์โฟลว์ขึ้น, exit handler ซึ่งมีหมายเลขที่ไม่ถูกต้องจะถูกเรียกใช้งาน และแถวจะถูกบันทึกลงในตาราง FailureLog. ตัวอย่างเช่น, sqlexception exit handler จะทำงาน, หากตาราง YearToDateTotals ถูกลบออกโดยบังเอิญ. ในตัวอย่างนี้, handler จะถูกใช้ในการบันทึกการทำงานเพื่อให้วินิจฉัยปัญหาได้ในภายหลัง.

```
DELETE FROM ExcessInventory WHERE Description='Desks'
DELETE FROM ExcessInventory WHERE Description='Chairs'
```

ตารางการถ่ายโอน SQL ทริกเกอร์:

SQL ทริกเกอร์อาจต้องอ้างอิงถึงแถวทั้งหมดที่ได้รับผลกระทบสำหรับการแทรก, อัพเดต หรือลบ SQL ตัวอย่างเช่น หากทริกเกอร์จำเป็นต้องใช้ฟังก์ชันแบบรวม เช่น MIN หรือ MAX กับคอลัมน์เฉพาะของแถวที่ได้รับผลกระทบ คุณสามารถใช้ตารางการถ่ายโอน OLD_TABLE และ NEW_TABLE เพื่อการนี้ได้

ในตัวอย่างต่อไปนี้, ทริกเกอร์จะใช้ฟังก์ชัน MAX แบบรวมกับแถวทั้งหมดของตาราง StudentProfiles ที่ได้รับผลกระทบ.

```
CREATE TABLE StudentProfiles(StudentsName VARCHAR(125),
      StudentsYearInSchool SMALLINT, StudentsGPA DECIMAL(5,2))

CREATE TABLE CollegeBoundStudentsProfile
      (YearInSchoolMin SMALLINT, YearInSchoolMax SMALLINT, StudentGPAMin
      DECIMAL(5,2), StudentGPAMax DECIMAL(5,2))

CREATE TRIGGER UpdateCollegeBoundStudentsProfileTrigger
AFTER UPDATE ON StudentProfiles
REFERENCING NEW_TABLE AS ntable
FOR EACH STATEMENT MODE DB2SQL
BEGIN
  DECLARE maxStudentYearInSchool SMALLINT;
  SET maxStudentYearInSchool =
       (SELECT MAX(StudentsYearInSchool) FROM ntable);
  IF maxStudentYearInSchool >
       (SELECT MAX (YearInSchoolMax) FROM
```

```
          CollegeBoundStudentsProfile) THEN
    UPDATE CollegeBoundStudentsProfile SET YearInSchoolMax =
         maxStudentYearInSchool;
  END IF;
END
```

ในตัวอย่างก่อนหน้านี้, ทริกเกอร์จะถูกเรียกใช้งานหนึ่งครั้ง หลังจากประมวลผลคำสั่งอัพเดตทริกเกอร์ เนื่องจากถูกกำหนดให้เป็นทริกเกอร์ FOR EACH STATEMENT. คุณจำเป็นต้องพิจารณาการประมวลผลทั่วไปที่กำหนดโดยระบบจัดการฐานข้อมูลสำหรับการบรรจุตารางการถ่ายโอน เมื่อคุณกำหนดทริกเกอร์ซึ่งอ้างอิงตารางการถ่ายโอน.

ทริกเกอร์ใช้ภายนอก

สำหรับทริกเกอร์ภายนอก โปรแกรมที่มี ชุด action ของทริกเกอร์ในภาษาชั้นสูงที่สนับสนุน ซึ่งใช้สร้างอ็อบเจ็กต์ *PGM

ทริกเกอร์โปรแกรมสามารถมี SQL ที่ใส่อยู่ในทริกเกอร์นั้น. เพื่อกำหนด ทริกเกอร์ภายนอก คุณต้องสร้างทริกเกอร์โปรแกรมและเพิ่มลงในตาราง โดยใช้คำสั่ง Add Physical File Trigger (ADDPFTRG) CL หรือคุณสามารถเพิ่มลงใน ตารางโดยใช้ System i Navigator ในการเพิ่ม ทริกเกอร์ให้กับตาราง คุณต้องทำสิ่งต่างๆ ดังต่อไปนี้:

- จำแนกตาราง
- จำแนกประเภทการดำเนินการ
- จำแนกโปรแกรมซึ่งทำงานตามที่ต้องการ.

งานที่เกี่ยวข้อง

การทำทริกเกอร์เหตุการณ์แบบอัตโนมัติในฐานข้อมูลของคุณ

การดีบักรูทีน SQL

ด้วยการระบุ SET OPTION DBGVIEW = *SOURCE ใน คำสั่ง CREATE PROCEDURE, CREATE FUNCTION หรือ CREATE TRIGGER คุณสามารถ ดีบักโปรแกรมหรือโมดูลที่สร้างขึ้นที่ระดับคำสั่ง SQL

และคุณยังสามารถระบุ DBGVIEW(*SOURCE) เป็นพารามิเตอร์บนคำสั่ง RUNSQLSTM และจะนำมาใช้กับรูทีนทั้งหมดที่อยู่ภายใน RUNSQLSTM.

มุมมองซอร์สจะถูกสร้างขึ้นโดยระบบที่มาจากโครงรูทีนเดิม โดยไว้อยู่ในซอร์สไฟล์ QSQDSRC ในไลบรารีของรูทีน. แต่ถ้าไม่สามารถกำหนดไลบราลีได้, QSQDSRC จะถูกสร้างอยู่ใน QTEMP. มุมมองซอร์สไม่ได้ถูกบันทึก ด้วยโปรแกรมหรือเซอร์วิสโปรแกรม. มุมมองนั้นจะถูกแยกออกเป็นบรรทัดซึ่งตรงกับที่ที่คุณสามารหยุดดีบักได้. ข้อความ, รวมทั้งพารามิเตอร์และชื่อตัวแปร, จะถูกปิดด้วยตัวพิมพ์ใหญ่.

ตัวแปรและพารามิเตอร์ทั้งหมดถูกสร้างขึ้นให้เป็นส่วนหนึ่งของโครงสร้าง. ต้องใช้ชื่อโครงสร้าง เมื่อประเมินผลตัวแปรในดีบัก. ตัวแปรจะเป็นไปตามเกณฑ์ด้วยการใช้ชื่อเลเบลปัจจุบัน. พารามิเตอร์จะเป็นไปตามเกณฑ์ด้วยการใช้โพรซีเดอร์หรือชื่อฟังก์ชัน. ตัวแปรการส่งผ่านในทริกเกอร์ จะเป็นไปตามเกณฑ์ด้วยการใช้ชื่อความสัมพันธ์ที่เหมาะสม. ขอแนะนำให้คุณระบุชื่อเลเบล สำหรับแต่ละข้อความผสมหรือข้อความ FOR. หากคุณไม่ได้ระบุชื่อ, ระบบจะสร้างชื่อให้คุณเอง. ซึ่งจะทำให้ใกล้เคียงต่อการประเมินผลตัวแปร. โปรดจำไว้ว่าพารามิเตอร์และตัวแปรทั้งหมดจะต้องถูกประเมินผลเป็นชื่อตัวพิมพ์ใหญ่. และคุณยังสามารถ ประเมินผลชื่อของโครงสร้างได้ด้วย. ซึ่งจะแสดงตัวแปรทั้งหมดภายในโครงสร้าง. หากตัวแปรหรือพารามิเตอร์เป็นศูนย์, ตัวบ่งชี้สำหรับตัวแปรหรือพารามิเตอร์นั้นก็จะตามหลังตัวแปรหรือพารามิเตอร์นั้นในโครงสร้างทันที.

เนื่องจากรูทีน SQL ถูกสร้างขึ้นใน C, จึงมีข้อจำกัดบางอย่างใน C ที่ส่งผลต่อการดีบักซอร์ส SQL. ชื่อที่ถูกคั่นซึ่งระบุในโครงรูทีน SQL ไม่สามารถระบุใน C ได้. ชื่อต่างๆ ถูกสร้างขึ้นสำหรับชื่อเหล่านี้, ซึ่งทำให้ยากต่อการดีบักหรือประเมินผล. ในการประเมินผลเนื้อหาของ ตัวแปรอักขระใดๆ, ให้ระบุ * ก่อนชื่อตัวแปร.

เนื่องจากระบบจะสร้างตัวบ่งชี้สำหรับชื่อตัวแปรและพารามิเตอร์ส่วนใหญ่, จึงไม่มีทางที่จะตรวจสอบโดยตรงเพื่อดูว่าตัวแปรมีค่า SQL ที่เป็นศูนย์หรือไม่. การประเมินผลตัวแปรจะแสดงค่าเสมอ, แม้ว่าจะมีการตั้งตัวบ่งชี้ให้แสดงค่าศูนย์ก็ตาม.

ในการพิจารณาว่า handler ถูกเรียกขึ้นมาหรือไม่นั้น, ให้เซ็ตจุดพัก ที่ข้อความแรกภายใน handler. ตัวแปรที่ถูกประกาศในข้อความผสม หรือคำสั่ง FOR ภายใน handler สามารถนำมาประเมินผลได้.

การปรับปรุงประสิทธิภาพของโพรซีเดอร์ และฟังก์ชัน

เมื่อมีการสร้างโพรซีเดอร์ที่เก็บไว้ หรือฟังก์ชันแบบ ผู้ใช้กำหนดเอง (UDF) ตัวประมวลผลภาษา SQL เชิงโพรซีเดอร์จะไม่สร้าง โค้ดที่มีประสิทธิภาพสูงสุดได้ อย่างไรก็ตาม คุณสามารถลดจำนวนการเรียกใช้เอ็นจินฐานข้อมูล และปรับปรุงประสิทธิภาพการทำงาน

ซึ่งคุณสามารถแก้ไขได้ทั้งในส่วนการออกแบบรูทีน และในส่วนของการนำไปปฏิบัติ. ตัวอย่างเช่น , ช่วงเวลาระหว่างการจัดการตัวแปรโฮสต์ในคอมไพเลอร์ภาษา C กับช่วงที่ตัวประมวลผลโพรซีเดอร์ของ SQL ต้องรอการจัดการตัวแปรโฮสต์ สามารถทำให้เกิดการเรียกใช้ส่วนเอ็นจินฐานข้อมูลได้หลายครั้ง. การเรียกใช้ส่วนเอ็นจินฐานข้อมูลเหล่านี้ต้องใช้ทรัพยากรมาก, และเมื่อต้องทำหลายๆ ครั้ง, จะส่งผลทำให้ประสิทธิภาพของเครื่องลดลงอย่างมาก.

การปรับปรุงประสิทธิภาพการทำงานของโพรซีเดอร์และฟังก์ชัน

เทคนิคการเขียนโค้ดพื้นฐานที่สามารถช่วยลด เวลาในการประมวลผลของฟังก์ชันหรือโพรซีเดอร์ได้

คำแนะนำเหล่านี้สำคัญอย่างมากสำหรับ ฟังก์ชัน เพราะฟังก์ชันมีแนวโน้มที่จะถูกเรียกใช้บ่อยครั้งจากหลาย โพรซีเดอร์:

- ให้ใช้อ็อพชัน NOT FENCED เพื่อให้ UDF รันใน thread เดียวกันกับผู้ที่เรียกใช้
- ให้ใช้อ็อพชัน DETERMINISTIC กับโพรซิเดอร์และ UDF ที่ให้ค่าผลลัพธ์เหมือนเดิมทุกครั้งเมื่อใส่ค่าอินพุตเดียวกัน. การใช้อ็อพชันนี้จะทำให้ optimizer สามารถแคชค่าผลลัพธ์ของการเรียกฟังก์ชันหรือแคชลำดับการเรียกฟังก์ชันในช่วงกระแสที่โปรแกรมทำงานเอาไว้ได้เพื่อร่นเวลารันไทม์.
- ใช้อ็อพชัน NO EXTERNAL ACTION กับ UDF ที่ไม่รับงานนอกขอบเขตของฟังก์ชัน ตัวอย่างของงานนอกขอบเขตได้แก่ ฟังก์ชันที่ต้องสร้างโพรเซสขึ้นมาใหม่เพื่อมารองรับ request การทำ transaction.

เทคนิคการเขียนโค้ดสำหรับส่วนรูทีนของ SQL จะได้ผลทางประสิทธิภาพช่วงรันไทม์อย่างมากเมื่อมีการสร้างเป็นโปรแกรมภาษา C ออกมา. ถ้าคุณหมั่นใช้ภาษา C ในการกำหนดค่าและการเปรียบเทียบในรูทีนของคุณ , คุณจะสามารถลดจำนวนประโยค SQL ที่ต้องใช้ลงได้. คำแนะนำหล่านี้จะช่วยให้รูทีนของคุณสร้างโค้ดภาษา C มากขึ้น และลดจำนวนคำสั่ง SQL ลง.

- ควรประกาศตัวแปรโฮสต์เป็นแบบ NOT NULL . มันจะทำให้โค้ดที่ได้ไม่ต้องคอยไปตรวจสอบและเซ็ตค่าแฟล็กสำหรับ null. ไม่ควรเซ็ตค่าตัวแปรทั้งหมดเป็น NOT NULL โดยอัตโนมัติ. การที่คุณระบุเป็น NOT NULL, คุณต้องมีค่าดีฟอลต์เตรียมไว้ด้วย. ถ้าตัวแปรนั้นถูกเรียกใช้ในรูทีนเป็นประจำ, การใช้ค่าดีฟอลต์จะช่วยได้. แต่ถ้า, ตัวแปรนั้นไม่ได้ถูกเรียกใช้อย่างสม่ำเสมอ, การตั้งค่าเป็นดีฟอลต์จะทำให้เกิด overhead ที่ไม่จำเป็น. ค่าดีฟอลต์นั้นเหมาะกับค่าตัวเลขที่สุด , ทำให้ไม่จำเป็นต้องเรียกใช้ฐานข้อมูลเพื่อการโพรเซสการกำหนดค่าดีฟอลต์อีก.
- หลีกเลี่ยงการใช้กับข้อมูลแบบอักขระและแบบวันที่. ตัวอย่าง นี่คือการใช้ตัวแปรเป็นค่าแฟล็กโดยกำหนดค่าเป็น 0, 1, 2, หรือ 3. ถ้าตัวแปรนี้ถูกประกาศเป็นตัวแปรแบบอักขระเดียวแทนที่จะเป็นจำนวนเต็ม, มันต้องมีการเรียกใช้เอ็นจินของฐานข้อมูลซึ่งควรหลีกเลี่ยง.

- ใช้จำนวนเต็ม อย่าใช้ทศนิยมที่มีสเกลเป็นศูนย์, โดยเฉพาะอย่างยิ่งเมื่อตัวแปรนั้นทำหน้าที่เป็นตัวนับ.
- อย่าใช้ตัวแปรชั่วคราว. พิจารณาตัวอย่างต่อไปนี้:

```
IF M_days<=30 THEN
 SET I = M_days-7;
 SET J = 23
 RETURN decimal(M_week_1 + ((M_month_1 - M_week_1)*I)/J,16,7);
END IF
```

ตัวอย่างนี้สามารถเขียนใหม่โดยไม่ต้องใช้ตัวแปรชั่วคราว:

```
IF M_days<=30 THEN
 Return decimal(M-week_1 + ((M_month_1 - M_week_1)* (M_days-7))/23,16,7);
END IF
```

- รวมคำสั่ง SET ที่มีลำดับซับซ้อนเป็นคำสั่งเดียว. ให้ใช้คำสั่งนี้กับคำสั่งที่ไม่สามารถสร้างเป็นภาษา C ได้เพราะมี CCSIDS หรือชนิดข้อมูลนั้นอยู่.

```
SET var1 = function1(var2);
SET var2 = function2();
```

สามารถเขียนรวมเป็นคำสั่งเดียวได้คือ:

```
SET var1 = function1(var2), var2 = function2();
```

- ใช้รูปแบบเป็น IF ( ) ELSE IF ( ) ... ELSE ... แทนที่จะใช้เป็น IF (x AND y) เพื่อเลี่ยงการเปรียบเทียบอันไม่จำเป็น.
- ทำให้ได้มากที่สุดในคำสั่ง SELECT:

```
SELECT A INTO Y FROM B;
 SET Y=Y||'X';
```

เขียนใหม่เป็น:

```
SELECT A || 'X' INTO Y FROM B
```

- หลีกเลี่ยงการเปรียบเทียบอักขระหรือวันที่ภายในลูป. บางครั้งการวนลูปสามารถเขียนใหม่ให้การเปรียบเทียบไปอยู่นอกลูปโดยให้การเปรียบเทียบนั้นตั้งค่าตัวแปรจำนวนเต็มเพื่อนำมาใช้ในลูปต่อไป. วิธีนี้ทำให้มีการประเมินผลข้อความที่ซับซ้อนเพียงครั้งเดียว. การเปรียบเทียบค่าจำนวนเต็มภายในลูปจะมีประสิทธิภาพมากกว่าเพราะมันสามารถทำได้กับโค้ดภาษา C ที่สร้างออกมา.
- หลีกเลี่ยงการตั้งค่าตัวแปรที่ไม่ได้ใช้. ตัวอย่าง, ถ้ามีการเซ็ตค่าตัวแปรนอกประโยค IF, ต้องมั่นใจว่าตัวแปรนั้นถูกนำมาใช้จริงกับ instance ทั้งหมดของประโยค IF. มิฉะนั้น, ให้ตั้งค่าตัวแปรเฉพาะในส่วนของประโยค IF ที่ได้ใช้จริงๆ.
- ถ้าเป็นไปได้ แทนที่ส่วนของโค้ดด้วยประโยค SELECT ประโยคเดียว. พิจารณาตัวอย่างโค้ดต่อไปนี้:

```
SET vnb_decimal = 4;
cdecimal:
   FOR vdec AS cdec CURSOR FOR
   SELECT nb_decimal
   FROM K$FX_RULES
   WHERE first_currency=Pi_curl AND second_currency=P1_cur2
   DO
    SET vnb_decimal=SMALLINT(cdecimal.nb_decimal);
END FOR cdecimal;

 IF vnb_decimal IS NULL THEN
```

```
 SET vnb_decimal=4;
END IF;
SET vrate=ROUND(vrate1/vrate2,vnb_decimal);
RETURN vrate;
```

โค้ดตัวอย่างข้างต้นสามารถทำให้มีประสิทธิภาพดีขึ้นได้โดยเขียนเป็น:

```
RETURN( SELECT
   CASE
    WHEN MIN(nb_decimal) IS NULL THEN ROUND(Vrate1/Vrate2,4)
   ELSE ROUND(Vrate1/Vrate2,SMALLINT(MIN(nb_decimal)))
   END
   FROM K$FX_RULES
   WHERE first_currency=Pi_curl AND second_currency=Pi_cur2);
```

- โค้ดภาษา C สามารถใช้สำหรับการตั้งค่าและการเปรียบเทียบค่าของข้อมูลแบบอักขระได้ถ้า CCSID ของ operand ทั้งสองฝั่งนั้นเหมือนกัน, ถ้า CCSID ตัวหนึ่งเท่ากับ 65535, ถ้า CCSID ไม่ใช่ UTF8, และถ้าการ truncate ข้อมูลอักระไม่สามารถทำได้. ถ้าไม่มีการระบุ CCSID ให้กับตัวแปร , จะไม่มีการกำหนดค่า CCSID จนกว่าจะมีการเรียกใช้โพรซีเดอร์. ในกรณีเช่นนี้, ต้องสร้างโค้ดให้ทำการกำหนดและเปรียบเทียบค่า CCSID ณ เวลารันไทม์. แต่ถ้ามีการกำหนดลำดับการเรียงทางเลือกไว้ หรือกำหนดค่า *JOBRUN, คุณจะไม่สามารถสร้างโค้ดภาษา C เพื่อใช้เปรียบเทียบอักขระได้.
- สำหรับการกำหนดค่าตัวแปรแบบตัวเลขทั้งหมด ให้ใช้ชนิดข้อมูลเดียวกัน , มีความยาวเท่ากันและสเกลเดียวกัน. การสร้างโค้ดภาษา C จะทำได้ก็ต่อเมื่อไม่สามารถทำการ truncate ได้.

```
DECLARE v1, v2 INT;
SET v1 = 100;
SET v1 =  v2;
```

การออกแบบรูทีนใหม่เพื่อเพิ่มประสิทธิภาพในการทำงาน

ถึงแม้จะทำตามคำแนะนำทั้งหมดแล้วก็ตาม บางทีโพรซีเดอร์หรือฟังก์ชันก็ยังทำงานไม่ได้ตามต้องการ ในกรณีนี้ คุณต้องย้อนกลับไปดูที่การออกแบบโพรซีเดอร์หรือฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) และ พิจารณาว่าจะสามารถแก้ไขอะไรเพื่อเพิ่มประสิทธิภาพการทำงานได้บ้าง

การเปลี่ยนแปลงการออกแบบมีหลายประเภทที่คุณควรพิจารณา

อย่างแรกคือการลดจำนวนการเรียกใช้ฐานข้อมูลหรือฟังก์ชันของโพรซีเดอร์, ซึ่งมีขั้นตอนคล้ายกับการค้นหาส่วนของโค้ดที่สามารถเปลงเป็นประโยค SQL . บ่อยครั้งที่คุณสามารถลดจำนวนการเรียกลงได้โดยการเพิ่มตรรกะเข้าไปในโค้ดของคุณ.

ส่วนอีกวิธีการที่ยากกว่าก็คือการปรับโครงสร้างของฟังก์ชันทั้งหมดโดยยึดตามผลลัพธ์เดิมแต่เปลี่ยนวิธีการใหม่. ตัวอย่างเช่น, ฟังก์ชันของคุณใช้ประโยค SELECT เพื่อใช้หาหนทางที่ตรงตามเงื่อนไขที่กำหนด จากนั้นก็เรียกใช้งานประโยคนั้นแบบไดนามิกส์. โดยพิจารณาจากวิธีที่ฟังก์ชันทำงาน, คุณอาจเปลี่ยนตรรกะการทำงานของฟังก์ชัน เพื่อให้ฟังก์ชันสามารถใช้เคียวรี่ SELECT แบบสแตติกในการค้นหาคำตอบ, ซึ่งส่งผลให้ได้ประสิทธิภาพที่ดีขึ้น.

คุณควรใช้ประโยค compound ในลักษณะที่ซ้อนกันเพื่อให้การ handle ของ exception และเคอร์เซอร์เป็นแบบ local. ถ้ามีการกำหนด handler เฉพาะเป็นจุดๆ, โค้ดที่ได้จะตรวจดูว่ามีข้อผิดพลาดหลังประโยคนั้นๆหรือไม่. โค้ดที่เกิดขึ้นจะสามารถปิดเคอร์เซอร์และเริมขั้นตอนของจุดช่วยเหลือถ้าเกิดข้อผิดพลาดในประโยคcompund นั้น. สำหรับรูทีนที่มีประโยคcompund เดียวแต่มีหลาย handler และหลายเคอร์เซอร์, โค้ดที่ได้จะจัดการทุก handler และเคอร์เซอร์หลังจากทุกประโยค SQL. ถ้าคุณกำหนดขอบเขตุของ handler และ เคอร์เซอร์กับในประโยคcompundแบบซ้อน, handler และ เคอร์เซอร์จะถูกตรวจสอบภายในประโยคcompundแบบซ้อนเท่านั้น.

ในรูทีนตัวอย่างนี้, โค้ดการตรวจสอบข้อผิดพลาด SQLSTATE '22H11' จะถูกสร้างขึ้นสำหรับประโยคที่อยู่ภายในประโยค compound ที่ชื่อ lab2 เท่านั้น. จะไม่มีการตรวจสอบข้อผิดพลาดเฉพาะนี้กับประโยคที่อยู่ในรูทีนนอกบล็อก lab2 . จะมีการสร้างโค้ดการตรวจสอบข้อผิดพลาด SQLEXCEPTION สำหรับทุกประโยคที่อยู่ในบล็อก lab1 และ lab2 . ในทำนองเดียวกัน, การจัดการข้อผิดพลาดสำหรับการปิดเคอร์เซอร์ c1 จะถูกจำกัดเฉพาะประโยคในบล็อก lab2 เท่านั้น.

```
Lab1: BEGIN
   DECLARE var1 INT;
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
         RETURN -3;
   lab2:  BEGIN
      DECLARE EXIT HANDLER FOR SQLSTATE '22H11'
           RETURN -1;
      DECLARE c1 CURSOR FOR SELECT col1 FROM table1;
      OPEN c1;
      CLOSE c1;
   END lab2;
END Lab1
```

เนื่องจากการออกแบบรูทีนใหม่ทั้งหมดเป็นการใช้เวลาอย่างมาก , ให้ตรวจสอบเฉพาะรูทีนหลักๆที่จะทำให้เกิดปัญหาทางด้านประสิทธิภาพแทนที่จะดูแอ็พพลิเคชันทั้งหมด. แต่สิ่งที่สำคัญกว่าการแก้ปัญหาด้วยการออกแบบใหม่ก็คือ การพยายามทบทวนถึงผลด้านประสิทธิภาพเป็นหลักตั้งแต่ช่วงที่ทำการออกแบบ. ถ้าคุณพยายามเน้นในส่วนของแอ็พพลิเคชันที่คาดว่าจะถูกใช้งานอย่างหนัก และมั่นใจว่าได้ออกแบบส่วนนั้นโดยคำนึงถึงประสิทธิภาพ คุณก็จะปลอดภัยจากการที่ต้องมาออกแบบส่วนดังกล่าวใหม่ภายหลัง.

การประมวลผลชนิดข้อมูลพิเศษ

ชนิดข้อมูลส่วนใหญ่ เช่น INTEGER และ CHARACTER ไม่ต้องใช้การประมวลผลที่มีคุณลักษณะพิเศษ อย่างไรก็ดี อาจมีชนิดข้อมูลบางประเภท ที่ต้องใช้ฟังก์ชันพิเศษ หรือ locator ในการเรียกใช้ข้อมูลเหล่านั้นอย่างมีประสิทธิภาพ

อ็อบเจ็กต์ขนาดใหญ่

อ็อบเจ็กต์ขนาดใหญ่ (LOB) เป็นชนิดข้อมูลสตริงที่มี ขนาดตั้งแต่ 0 ไบต์ ถึง 2 GB (GB เท่ากับ 1 073 741 824 ไบต์)

ชนิดข้อมูล VARCHAR, VARGRAPHIC, และ VARBINARY จำกัดเนื้อที่การจัดเก็บได้เพียง 32 KB (โดยที่ KB มีค่าเท่ากับ 1024 ไบต์) ของหน่วยเก็บ. แม้ว่าข้อจำกัดนี้อาจ เพียงพอสำหรับข้อความที่มีขนาดเล็กถึงปานกลาง แต่ แอ็พพลิเคชันต้องการเก็บเอกสารที่มีข้อความขนาดใหญ่ ซึ่งแอ็พพลิเคชันเหล่านี้อาจต้องการเก็บชนิดข้อมูลอื่นๆ อีกหลายชนิดเพิ่มเติม เช่น เสียง, วิดีโอ, รูปวาด, ข้อความผสมกับกราฟิก, และรูปภาพ ชนิดข้อมูลบางชนิดสามารถจัดเก็บอ็อบเจ็กต์ข้อมูลเหล่านี้เป็นสตริงที่มีขนาดสูงสุด 2 GB

ชนิดข้อมูลเหล่านี้ได้แก่ binary large objects (BLOBs), single-byte character large objects (CLOBs) และ double-byte character large objects (DBCLOBs) แต่ละตารางอาจมีข้อมูล LOB เป็นจำนวนมาก ถึงแม้ว่าแถวหนึ่งจะเก็บค่า LOB ได้ไม่เกิน 3.5 GB แต่ตารางอาจมีข้อมูล LOB เกือบถึง 256 GB

คุณสามารถอ้างถึงและะดำเนินการกับ LOBs โดยใช้ตัวแปรโฮสต์ได้ เหมือนกับที่คุณทำกับชนิดข้อมูลอื่นๆ อย่างไรก็ตาม ตัวแปรโฮสต์ใช้หน่วยความจำจากโปรแกรม ซึ่งอาจจะมีขนาดไม่ใหญ่พอที่จะเก็บค่า LOB ดังนั้นคุณอาจต้องจัดการค่าขนาด

ใหญ่ด้วยวิธีการอื่นๆ *Locators* ใช้เพื่อระบุและดำเนินการ กับอ็อบเจ็กต์ขนาดใหญ่ในเซิร์ฟเวอร์ฐานข้อมูลและใช้สำหรับดึงค่าของ LOB *ตัวแปรที่อ้างอิงถึงไฟล์* ใช้เพื่อย้ายค่าอ็อบเจ็กต์ขนาดใหญ่(หรือส่วนที่ใหญ่ของอ็อบเจ็กต์นั้น)ไปยังไคลเอนต์หรือย้ายค่านั้นมาจากไคลเอนต์.

ชนิดข้อมูลอ็อบเจ็กต์ขนาดใหญ่

นี่คือ definition ของ binary large objects (BLOBs), character large objects (CLOBs) และ double-byte character large objects (DBCLOBs)

- Binary large object (BLOB) — คือสตริงแบบไบนารีที่ถูกสร้างมาจากข้อมูลไบต์ที่ไม่มีการเชื่อมโยงกับโค้ดเพจ. ชนิดข้อมูลนี้สามารถเก็บข้อมูลไบนารีที่มีขนาดใหญ่กว่าชนิด VARBINARY (ถูกจำกัดไว้ที่32K). ชนิดข้อมูลนี้เหมาะสำหรับเก็บรูปภาพ, เสียง, กราฟิก, และข้อมูลเฉพาะทางธุรกิจหรือแอ็พพลิเคชันอื่นๆ.
- Character large object (CLOB) — คือสตริงอักขระที่ถูกสร้างมาจากอักขระแบบไบต์เดียวที่มีการเชื่อมโยงกับโค้ดเพจ. ชนิดข้อมูลนี้เหมาะสมสำหรับการเก็บข้อมูลลักษณะที่เป็นตัวอักษรโดยขนาดของข้อมูลอาจเพิ่มจนเกินขีดจำกัดของชนิดข้อมูล VARCHAR (จำนวนสูงสุดไม่เกิน 32 กิโลไบต์). ข้อมูลชนิดนี้รองรับการแปลงโค้ดเพจได้
- Double-byte character large object (DBCLOBs) — คือสตริงอักขระที่ถูกสร้างมาจากอักขระแบบสองไบต์ที่มีการเชื่อมโยงกับโค้ดเพจ. ชนิดข้อมูลนี้เหมาะสำหรับเก็บข้อมูลลักษณะที่เป็นตัวอักษรซึ่งใช้ชุดอักขระแบบสองไบต์. อีกครั้ง, ข้อมูลชนิดนี้รองรับการแปลงโค้ดเพจได้.

Large object locator

Large object (LOB) locator เป็น ค่าขนาดเล็กที่จัดการง่าย ใช้สำหรับอ้างอิงค่าที่ใหญ่กว่ามาก

ถ้าชี้ชัดลงไป LOB locator ก็คือค่าขนาด 4 ไบต์ที่เก็บอยู่ในตัวแปรโฮสต์ที่โปรแกรมใช้อ้างอิงไปสู่ค่า LOB ที่อยู่ในเซิร์ฟเวอร์ฐานข้อมูล โดยการใช้ LOB locator แล้ว โปรแกรมสามารถจัดการกับค่า LOB เหมือนกับว่าค่าดังกล่าวเก็บอยู่ในตัวแปรโฮสต์ปรกติ เมื่อคุณใช้ LOB locator, จึงไม่จำเป็นต้องส่งค่า LOB จากเซิร์ฟเวอร์ไปยังแอ็พพลิเคชัน (และอาจจะส่งค่ากลับมาอีกครั้ง).

LOB locator จะเชื่อมโยงกับค่า LOB , ไม่ได้เชื่อมโยงกับแถวหรือตำแหน่งที่เก็บข้อมูลในฐานข้อมูล. ดังนั้น, หลังจากที่กำหนดค่า LOB ให้กับ locator แล้ว, คุณไม่สามารถทำอะไรกับแถวหรือตารางต้นฉบับที่จะมีผลกับค่าที่ถูกอ้างอิงจาก locator ได้. ค่าที่สัมพันธ์กับ locator จะยังถูกต้องจนกระทั่งหน่วยการทำงานสิ้นสุด, หรือเมื่อ locator ถูกปล่อยค่าโดยตรง, อยู่ที่ว่าเหตุการณ์ใดเกิดขึ้นก่อน. คำสั่ง FREE LOCATOR จะปลด locator จากค่าที่มันเชื่อมโยงอยู่. ในทำนองเดียวกัน, คำสั่ง commit หรือ rollback จะปลด LOB locators ที่ผูกกับ transaction ทั้งหมดออก.

LOB locators สามารถถูกผ่านค่าไปมากับ UDFs ได้. ใน UDF, ฟังก์ชันที่ใช้ข้อมูล LOB สามารถนำไปจัดการค่า LOB โดยใช้ LOB locators ได้.

เมื่อเลือกค่า LOB, คุณมี 3 ตัวเลือกคือ.

- เลือกค่า LOB ทั้งหมดไปที่ตัวแปรโฮสต์. ค่า LOB ทั้งหมดจะถูกทำสำเนาไปยังตัวแปรโฮสต์.
- เลือกค่า LOB ไปที่ LOB locator. ค่า LOB จะยังอยู่ที่เซิร์ฟเวอร์; ค่า LOB จะไม่ถูกทำสำเนาไปที่ตัวแปรโฮสต์.
- เลือกค่า LOB ทั้งหมดไปที่ตัวแปรที่อ้างอิงไปยังไฟล์. ค่า LOB จะถูกย้ายไปยังไฟล์ระบบไฟล์รวม

ลักษณะการใช้ค่า LOB ภายในโปรแกรมสามารถช่วยโปรแกรมเมอร์ตัดสินใจว่าวิธีการใดเหมาะสมที่สุด. ถ้าค่า LOB มีขนาดใหญ่มากและจำเป็นต้องใช้เป็นค่าอินพุตสำหรับคำสั่ง SQL ที่ตามมาเท่านั้น, ให้เก็บค่าไว้ใน locator.

ถ้าโปรแกรมจำเป็นต้องใช้ค่า LOB ทั้งหมดโดยไม่สนใจเรื่องขนาด, คงไม่มีทางเลือกอื่นนอกจากจะถ่ายโอน LOB เท่านั้น. แม้ในกรณีนี้, ก็ยังมีตัวเลือกสำหรับคุณ. คุณสามารถเลือกค่าทั้งหมดไปที่ตัวแปรโฮสต์ปรกติหรือไปที่ตัวแปรโฮสต์ที่อ้างอิงไปยังไฟล์. คุณยังสามารถเลือกค่า LOB ให้กับ locator และอ่านค่านั้นทีละส่วนจาก locator ไปยังตัวแปรโฮสต์ปรกติได้.

สิ่งอ้างอิงที่เกี่ยวข้อง

"ตัวแปรที่อ้างอิงถึงไฟล์ LOB" ในหน้า 232
ตัวแปรที่อ้างอิงถึงไฟล์จะคล้ายกับตัวแปร โฮสต์ ยกเว้นว่า ตัวแปรนั้นถูกใช้เพื่อโอนย้ายข้อมูลไปมาระหว่างไฟล์ integrated (ไม่ใช่โอนย้ายระหว่างบัฟเฟอร์หน่วยความจำ)

"ตัวอย่าง: การใช้ locator เพื่อทำงานกับค่า CLOB"
สมมติว่า คุณต้องการให้แอ็พพลิเคชันโปรแกรม เรียก locator สำหรับค่า character large object (CLOB) แล้วใช้ locator ดังกล่าวเพื่อดึงข้อมูลจากค่า CLOB

ตัวอย่าง: การใช้ locator เพื่อทำงานกับค่า CLOB

สมมติว่า คุณต้องการให้แอ็พพลิเคชันโปรแกรม เรียก locator สำหรับค่า character large object (CLOB) แล้วใช้ locator ดังกล่าวเพื่อดึงข้อมูลจากค่า CLOB

โดยใช้วิธีนี้โปรแกรมจัดสรรที่เก็บข้อมูล ให้พอสำหรับข้อมูล CLOB เพียงหนึ่งชิ้นเท่านั้น (ขนาดที่เก็บจะพิจารณาโดยโปรแกรม) นอกเหนือจากนี้, โปรแกรมสามารถออกคำสั่งดึงข้อมูลโดยใช้เคอร์เซอร์เพียงครั้งเดียวเท่านั้น.

โปรแกรมตัวอย่าง LOBLOC ทำงานอย่างไร

โปรแกรม ตัวอย่างมีสามส่วนที่ทำงานกับตัวแปร CLOB locator:

1. **การประกาศตัวแปรโฮสต์.** ตัวแปรโฮสต์ CLOB locator จะถูกประกาศ
2. **การดึงค่า CLOB ไปที่ตัวแปรโฮสต์ของ locator** รูทีน CURSOR และ FETCH ถูกใช้เพื่อรับค่าตำแหน่งของฟิลด์ CLOB ใน ฐานข้อมูลไปไว้ที่ตัวแปรโฮสต์ locator
3. **ปล่อยค่า CLOB locators** CLOB locators ที่ถูกใช้ในตัวอย่างนี้จะถูกปล่อยค่า, เป็นการปล่อยค่า locator จากค่าที่ก่อนหน้านี้เชื่อมโยงอยู่

หลักการที่เกี่ยวข้อง

"Large object locator" ในหน้า 227
Large object (LOB) locator เป็น ค่าขนาดเล็กที่จัดการง่าย ใช้สำหรับอ้างอิงค่าที่ใหญ่กว่ามาก

ตัวอย่าง: LOBLOC ใน C:

โปรแกรมตัวอย่างนี้ ซึ่งเขียนเป็นภาษา C ใช้ locator เพื่อเรียกค่า CLOB

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ "สิทธิในรหัส และข้อมูลถ้อยแถลง" ในหน้า 353.

```
| #include <stdio.h>
| #include <stdlib.h>
| #include <string.h>
|
| EXEC SQL INCLUDE SQLCA;
|
| int main(int argc, char *argv[]) {
|
|    EXEC SQL BEGIN DECLARE SECTION;
```

```
|        char number[7];
|        long deptInfoBeginLoc;
|        long deptInfoEndLoc;
|        SQL TYPE IS CLOB_LOCATOR resume;     [1]
|        SQL TYPE IS CLOB_LOCATOR deptBuffer;
|        short lobind;
|        char buffer[1000]="";
|        char userid[9];
|        char passwd[19];
|     EXEC SQL END DECLARE SECTION;
|
|     printf( "Sample C program: LOBLOC\n" );
|
|     if (argc == 1) {
|        EXEC SQL CONNECT TO sample;
|     }
|     else if (argc == 3) {
|        strcpy (userid, argv[1]);
|        strcpy (passwd, argv[2]);
|        EXEC SQL CONNECT TO sample USER :userid USING :passwd;
|     }
|     else {
|        printf ("\nUSAGE: lobloc [userid passwd]\n\n");
|        return 1;
|     } /* endif */
|
|     /* พนักงาน A10030 ไม่ถูกรวมอยู่ในรายการทีเลือกต่อไปนี้, เพราะว่า
|        โปรแกรม lobeval จะดำเนินการกับเร็กคอร์ดของ A10030 เพื่อที่ว่า
|        เร็กคอร์ดนั้นจะทำงานร่วมกับ lobloc ได้ */
|
|     EXEC SQL DECLARE c1 CURSOR FOR
|              SELECT empno, resume FROM emp_resume WHERE resume_format='ascii'
|              AND empno <> 'A00130';
|
|     EXEC SQL OPEN c1;
|
|     do {
|        EXEC SQL FETCH c1 INTO :number, :resume :lobind;  [2]
|        if (SQLCODE != 0) break;
|        if (lobind < 0) {
|           printf ("NULL LOB indicated\n");
|        } else {
|           /* EVALUATE the LOB LOCATOR */
|           /* Locate the beginning of "Department Information" section */
|           EXEC SQL VALUES (POSSTR(:resume, 'Department Information'))
|              INTO :deptInfoBeginLoc;
|
|           /* Locate the beginning of "Education" section (end of "Dept.Info" */
|           EXEC SQL VALUES (POSSTR(:resume, 'Education'))
|              INTO :deptInfoEndLoc;
|
|           /* Obtain ONLY the "Department Information" section by using SUBSTR */
|           EXEC SQL VALUES(SUBSTR(:resume, :deptInfoBeginLoc,
|              :deptInfoEndLoc - :deptInfoBeginLoc)) INTO :deptBuffer;
|
```

```
|           /* Append the "Department Information" section to the :buffer var. */
|           EXEC SQL VALUES(:buffer || :deptBuffer) INTO :buffer;
|         } /* endif */
|     } while ( 1 );
|
|     printf ("%s\n",buffer);
|
|     EXEC SQL FREE LOCATOR :resume, :deptBuffer; [3]
|
|     EXEC SQL CLOSE c1;
|
|     EXEC SQL CONNECT RESET;
|     return 0;
|  }
|  /* end of program : LOBLOC */
```

ตัวอย่าง: LOBLOC ใน COBOL:

โปรแกรมตัวอย่างนี้ ซึ่งเขียนเป็นภาษา COBOL ใช้ locator เพื่อเรียกค่า CLOB

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

```
|   Identification Division.
|   Program-ID. "lobloc".
|
|   Data Division.
|   Working-Storage Section.
|       EXEC SQL INCLUDE SQLCA END-EXEC.
|
|       EXEC SQL BEGIN DECLARE SECTION END-EXEC.
|   01 userid            pic x(8).
|   01 passwd.
|     49 passwd-length   pic s9(4) comp-5 value 0.
|     49 passwd-name     pic x(18).
|   01 empnum            pic x(6).
|   01 di-begin-loc      pic s9(9) comp-5.
|   01 di-end-loc        pic s9(9) comp-5.
|   01 resume            USAGE IS SQL TYPE IS CLOB-LOCATOR.   [1]
|   01 di-buffer         USAGE IS SQL TYPE IS CLOB-LOCATOR.
|   01 lobind            pic s9(4) comp-5.
|   01 buffer            USAGE IS SQL TYPE IS CLOB(1K).
|       EXEC SQL END DECLARE SECTION END-EXEC.
|
|   Procedure Division.
|   Main Section.
|       display "Sample COBOL program: LOBLOC".
|
|  * Get database connection information.
|       display "Enter your user id (default none): "
|            with no advancing.
|       accept userid.
|
|       if userid = spaces
|         EXEC SQL CONNECT TO sample END-EXEC
|       else
```

```
|         display "Enter your password : " with no advancing
|         accept passwd-name.
|
|  * Passwords in a CONNECT statement must be entered in a VARCHAR
|  * format with the length of the input string.
|       inspect passwd-name tallying passwd-length for characters
|          before initial " ".
|
|       EXEC SQL CONNECT TO sample USER :userid USING :passwd
|           END-EXEC.
|
|  * Employee A10030 is not included in the following select, because
|  * the lobeval program manipulates the record for A10030 so that it is
|  * not compatible with lobloc
|
|       EXEC SQL DECLARE c1 CURSOR FOR
|                 SELECT empno, resume FROM emp_resume
|                 WHERE resume_format = 'ascii'
|                 AND empno <> 'A00130' END-EXEC.
|
|       EXEC SQL OPEN c1 END-EXEC.
|
|       Move 0 to buffer-length.
|
|       perform Fetch-Loop thru End-Fetch-Loop
|          until SQLCODE not equal 0.
|
|  * display contents of the buffer.
|       display buffer-data(1:buffer-length).
|
|       EXEC SQL FREE LOCATOR :resume, :di-buffer END-EXEC.  [3]
|
|       EXEC SQL CLOSE c1 END-EXEC.
|
|       EXEC SQL CONNECT RESET END-EXEC.
|   End-Main.
|          go to End-Prog.
|
|   Fetch-Loop Section.
|       EXEC SQL FETCH c1 INTO :empnum, :resume :lobind   [2]
|          END-EXEC.
|
|       if SQLCODE not equal 0
|          go to End-Fetch-Loop.
|
|  * check to see if the host variable indicator returns NULL.
|       if lobind less than 0 go to NULL-lob-indicated.
|
|  * Value exists.  Evaluate the LOB locator.
|  * Locate the beginning of "Department Information" section.
|       EXEC SQL VALUES (POSSTR(:resume, 'Department Information'))
|                 INTO :di-begin-loc END-EXEC.
|
|  * Locate the beginning of "Education" section (end of Dept.Info)
|       EXEC SQL VALUES (POSSTR(:resume, 'Education'))
```

```
|                INTO :di-end-loc END-EXEC.
|
|       subtract di-begin-loc from di-end-loc.
|
|  * Obtain ONLY the "Department Information" section by using SUBSTR
|       EXEC SQL VALUES (SUBSTR(:resume, :di-begin-loc,
|                :di-end-loc))
|                INTO :di-buffer END-EXEC.
|
|  * Append the "Department Information" section to the :buffer var
|       EXEC SQL VALUES (:buffer || :di-buffer) INTO :buffer
|                END-EXEC.
|
|       go to End-Fetch-Loop.
|
|   NULL-lob-indicated.
|       display "NULL LOB indicated".
|
|   End-Fetch-Loop. exit.
|
|   End-Prog.
|             stop run.
```

ตัวแปรตัวบ่งชี้ และ LOB locator

สำหรับตัวแปรโฮสต์ในแอ็พพลิเคชันปรกติแล้ว, เมื่อเลือกค่า NULL ให้กับตัวแปรโฮสต์แล้ว, ค่าลบจะถูกกำหนดให้กับตัวแปรตัวบ่งชี้ เพื่อเป็นความหมายว่าค่าคือ NULL. อย่างไรก็ตาม, ในกรณีของ LOB locator, ความหมายของตัวแปรตัวบ่งชี้จะต่างไปเล็กน้อย.

เนื่องจากตัวแปรโฮสต์ไม่สามารถมีค่าเป็น NULL ได้, ดังนั้นตัวแปรตัวบ่งชี้ที่มีค่าเป็นลบจะทำให้รู้ว่าค่า LOB ที่อ้างถึงโดย LOB locator มีค่าเป็น NULL. ข้อมูล NULL จะถูกเก็บไว้ที่ไคลเอ็นต์โดยใช้ค่าตัวแปรตัวบ่งชี้. เซิร์ฟเวอร์จะไม่แทร็กค่า NULL ด้วย locator ที่ถูกต้อง.

ตัวแปรที่อ้างอิงถึงไฟล์ LOB

ตัวแปรที่อ้างอิงถึงไฟล์จะคล้ายกับตัวแปร โฮสต์ ยกเว้นว่า ตัวแปรนั้นถูกใช้เพื่อโอนย้ายข้อมูลไปมาระหว่างไฟล์ integrated (ไม่ใช่โอนย้ายระหว่างบัฟเฟอร์หน่วยความจำ)

ตัวแปรที่อ้างอิงถึงไฟล์จะแทนค่าไฟล์(มากกว่าที่จะเก็บไฟล์), คล้ายกับที่ LOB locator แทนค่า LOB(มากกว่าที่จะเก็บค่า LOB). เคียวรี่, อัพเดต, และแทรกฐานข้อมูลอาจใช้ตัวแปรที่อ้างอิงถึงไฟล์ เพื่อเก็บ, หรือดึง, ค่า LOB หนึ่งค่า.

สำหรับอ็อบเจ็กต์ที่มีขนาดใหญ่มาก, ไฟล์คือที่เก็บตามปรกติ. มันเหมือนกับว่า LOBs ส่วนใหญ่เริ่มมาจากข้อมูลนั้นถูกเก็บไว้ในไฟล์บนไคลเอนต์ก่อนที่จะย้ายข้อมูลนั้นไปที่ฐานข้อมูลบนเซิร์ฟเวอร์. การใช้ตัวแปรที่อ้างอิงถึงไฟล์ จะช่วยในการย้ายข้อมูล LOB โปรแกรมใช้ตัวแปรที่อ้างอิงถึงไฟล์เพื่อโอนถ่ายข้อมูล LOB จากไฟล์ integrated ไปยังเอ็นจินฐานข้อมูลโดยตรง เมื่อต้งการย้ายข้อมูล LOB, แอ็พพลิเคชันไม่จำเป็นต้องเขียนยูทิลิตีรูทีน เพื่ออ่านและเขียนไฟล์โดยใช้ตัวแปรโฮสต์.

หมายเหตุ: ไฟล์ที่ถูกอ้างถึงจะต้องสามารถเข้าถึงได้จาก(แต่ไม่จำเป็นต้องเก็บอยู่ใน)ระบบที่โปรแกรมนั้นทำงานอยู่. สำหรับโพรซีเดอร์ที่เก็บไว้, จะอยู่ที่เซิร์ฟเวอร์.

ตัวแปรที่อ้างอิงถึงไฟล์มีชนิดข้อมูลเป็น BLOB, CLOB, หรือ DBCLOB. และตัวแปรนี้ถูกใช้ให้เป็นแหล่งข้อมูล(อินพุต) หรือไม่ก็เป็นข้อมูลปลายทาง(เอาต์พุต). ตัวแปรที่อ้างอิงถึงไฟล์อาจจะเป็นชื่อไฟล์แบบอ้างอิง(relative file name) หรือชื่อไฟล์

แบบสมบูรณ์(complete path name) ก็ได้ (แนะนำให้ใช้อย่างหลัง). ความยาวของชื่อไฟล์ถูกระบุโดยแอ็พพลิเคชันโปรแกรม. ส่วนความยาวข้อมูลของตัวแปรที่อ้างอิงถึงไฟล์จะไม่ถูกใช้ขณะอินพุต. ขณะเอาต์พุต, ความยาวข้อมูลจะถูกตั้งค่าโดยโค้ดของ application requester ให้มีความยาวของข้อมูลใหม่เพื่อค่านี้จะได้ถูกเขียนลงไปในไฟล์.

เมื่อใช้ตัวแปรที่อ้างอิงถึงไฟล์จะมีหลายตัวเลือกที่ต่างกันสำหรับทั้งอินพุตและเอาต์พุต. คุณต้องเลือกการกระทำสำหรับไฟล์ โดยการตั้งค่าฟิลด์ `file_options` ใน structure ของตัวแปรที่อ้างอิงถึงไฟล์. ตัวเลือกสำหรับการกำหนดค่าให้กับฟิลด์ซึ่งครอบคลุมทั้งค่าอินพุตและค่าเอาต์พุตถูกแสดงไว้ด้านล่างนี้.

ค่า (แสดงสำหรับภาษา C) และอ็อพชัน เมื่อใช้ตัวแปรที่อ้างอิงถึงไฟล์มีค่าดังต่อไปนี้:

- **SQL_FILE_READ** (ไฟล์ปกติ) — อ็อพชันนี้มีค่าเป็น 2. นี่คือไฟล์ที่สามารถเปิด, อ่าน, และปิดได้. DB2 กำหนดความยาวของข้อมูลในไฟล์(เป็นไบต์)ตอนเปิดไฟล์. แล้ว DB2 จึงผ่านค่าความยาวของข้อมูลไว้ที่ฟิลด์ `data_lengthของ` structure ตัวแปรที่อ้างอิงถึงไฟล์. ค่าสำหรับ COBOL คือ SQL-FILE-READ.

ค่าและอ็อพชัน เมื่อใช้ตัวแปรที่อ้างอิงถึงไฟล์แบบเอาต์พุตมีค่าดังนี้:

- **SQL_FILE_CREATE** (สร้างไฟล์) — อ็อพชันนี้มีค่าเป็น 8. อ็อพชันนี้จะทำการสร้างไฟล์ใหม่. ถ้ามีไฟล์นี้อยู่แล้ว, จะส่งข้อความแสดงความผิดพลาดกลับมา. ค่าสำหรับ COBOL คือ SQL-FILE-CREATE.
- **SQL_FILE_OVERWRITE** (เขียนทับไฟล์) — อ็อพชันนี้มีค่าเป็น 16. ตัวเลือกนี้จะทำการสร้างไฟล์ขึ้นมาใหม่ถ้าไฟล์นั้นไม่เคยมีอยู่. แต่ถ้าไฟล์นั้นมีอยู่แล้ว, ข้อมูลใหม่จะเขียนทับข้อมูลเดิมในไฟล์นั้น. ค่าสำหรับ COBOL คือ SQL-FILE-OVERWRITE.
- **SQL_FILE_APPEND** (ต่อท้ายไฟล์) — อ็อพชันนี้มีค่าเป็น 32. ตัวเลือกนี้จะส่งผลลัพธ์ผนวกเข้าไปต่อท้ายไฟล์, ถ้าไฟล์นั้นมีอยู่. มิฉะนั้น, จะทำการสร้างไฟล์ขึ้นมาใหม่. ค่าสำหรับ COBOL คือ SQL-FILE-APPEND.

หมายเหตุ: ถ้าตัวแปรที่อ้างอิงถึงไฟล์ LOB ถูกใช้ในคำสั่ง OPEN , ห้ามลบไฟล์ที่เชื่อมโยงกับตัวแปรที่อ้างอิงถึงไฟล์ LOB จนกว่าเคอร์เซอร์จะถูกปิด.

หลักการที่เกี่ยวข้อง

"Large object locator" ในหน้า 227

Large object (LOB) locator เป็น ค่าขนาดเล็กที่จัดการง่าย ใช้สำหรับอ้างอิงค่าที่ใหญ่กว่ามาก

ระบบไฟล์รวม

ตัวอย่าง: การดึงข้อมูล CLOB ไปยังไฟล์

สมมติว่า คุณจำเป็นต้องดึงส่วนประกอบอักขระอ็อบเจ็กต์ ขนาดใหญ่ (CLOB) จากตารางไปเก็บไว้ที่ไฟล์ภายนอก

โปรแกรมตัวอย่าง LOBFILE ทำงานอย่างไร

โปรแกรม ตัวอย่างมีสามส่วนที่ทำงานกับตัวแปรอ้างอิงไฟล์ CLOB:

1. **การประกาศตัวแปรโฮสต์.** ตัวแปรโฮสต์การอ้างอิงไฟล์ CLOB จะถูก ประกาศ การประกาศถูกขยายเป็นโครงสร้างที่รวมการประกาศ สำหรับชื่อไฟล์, ความยาวชื่อไฟล์ และอ็อพชันของไฟล์
2. **ตัวแปรโฮสต์การอ้างอิงไฟล์ CLOB จะถูกตั้งค่า** แอ็ตทริบิวต์ของ การอ้างอิงไฟล์จะถูกตั้งค่า ชื่อไฟล์ที่ไม่ได้ประกาศพาธแบบเต็ม, โดยดีฟอลต์, จะถูกเก็บไว้ในไดเร็กทอรีปัจจุบันของผู้ใช้. ถ้าชื่อพาธไม่ได้ขึ้นต้นด้วยอักขระ forward slash (/), ชื่อนั้นจะใช้ไม่ได้.
3. **เลือกไปที่ตัวแปรโฮสต์การอ้างอิงไฟล์ CLOB** ข้อมูลจากฟิลด์ `resume` ถูกกำหนดไปที่ชื่อไฟล์ที่ถูกอ้างอิงจากตัวแปรโฮสต์.

ตัวอย่าง: LOBFILE ใน C:

โปรแกรมตัวอย่างนี้ ซึ่งเขียนด้วยภาษา C จะดึงข้อมูล CLOB จากตารางไปยังไฟล์ภายนอก

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sql.h>

EXEC SQL INCLUDE SQLCA;

int main(int argc, char *argv[]) {

   EXEC SQL BEGIN DECLARE SECTION;
      SQL TYPE IS CLOB_FILE resume;     [1]
      short lobind;
      char userid[9];
      char passwd[19];
   EXEC SQL END DECLARE SECTION;

   printf( "Sample C program: LOBFILE\n" );

   if (argc == 1) {
      EXEC SQL CONNECT TO sample;

   else if (argc == 3) {
      strcpy (userid, argv[1]);
      strcpy (passwd, argv[2]);
      EXEC SQL CONNECT TO sample USER :userid USING :passwd;

   else {
      printf ("\nUSAGE: lobfile [userid passwd]\n\n");
      return 1;
   } /* endif */

   strcpy (resume.name, "RESUME.TXT");         [2]
   resume.name_length = strlen("RESUME.TXT");
   resume.file_options = SQL_FILE_OVERWRITE;

   EXEC SQL SELECT resume INTO :resume :lobind FROM emp_resume  [3]
      WHERE resume_format='ascii' AND empno='000130';

   if (lobind < 0) {
      printf ("NULL LOB indicated \n");
   } else {
      printf ("Resume for EMPNO 000130 is in file : RESUME.TXT\n");
   } /* endif */

   EXEC SQL CONNECT RESET;
   return 0;
}
/* end of program : LOBFILE */
```

ตัวอย่าง: LOBFILE ใน COBOL:

โปรแกรมตัวอย่างนี้ ซึ่งเขียนด้วยภาษา C จะดึงข้อมูล CLOB จากตารางไปยังไฟล์ภายนอก

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ "สิทธิในรหัส และข้อมูลถ้อยแถลง" ในหน้า 353.

```
|
|   Identification Division.
|   Program-ID. "lobfile".
|
|   Data Division.
|   Working-Storage Section.
|       EXEC SQL INCLUDE SQLCA END-EXEC.
|
|       EXEC SQL BEGIN DECLARE SECTION END-EXEC.
|   01 userid            pic x(8).
|   01 passwd.
|     49 passwd-length   pic s9(4) comp-5 value 0.
|     49 passwd-name     pic x(18).
|   01 resume            USAGE IS SQL TYPE IS CLOB-FILE.  [1]
|   01 lobind            pic s9(4) comp-5.
|       EXEC SQL END DECLARE SECTION END-EXEC.
|
|   Procedure Division.
|   Main Section.
|       display "Sample COBOL program: LOBFILE".
|
|  * Get database connection information.
|       display "Enter your user id (default none): "
|            with no advancing.
|       accept userid.
|
|       if userid = spaces
|         EXEC SQL CONNECT TO sample END-EXEC
|       else
|         display "Enter your password : " with no advancing
|         accept passwd-name.
|
|  * Passwords in a CONNECT statement must be entered in a VARCHAR
|  * format with the length of the input string.
|       inspect passwd-name tallying passwd-length for characters
|          before initial " ".
|
|       EXEC SQL CONNECT TO sample USER :userid USING :passwd
|           END-EXEC.
|
|       move "RESUME.TXT" to resume-NAME.               [2]
|       move 10 to resume-NAME-LENGTH.
|       move SQL-FILE-OVERWRITE to resume-FILE-OPTIONS.
|
|       EXEC SQL SELECT resume INTO :resume :lobind     [3]
|                FROM emp_resume
|                WHERE resume_format = 'ascii'
|                AND empno = '000130' END-EXEC.
|       if lobind less than 0 go to NULL-LOB-indicated.
```

```
|
|       display "Resume for EMPNO 000130 is in file : RESUME.TXT".
|       go to End-Main.
|
|    NULL-LOB-indicated.
|       display "NULL LOB indicated".
|
|    End-Main.
|       EXEC SQL CONNECT RESET END-EXEC.
|    End-Prog.
|               stop run.
```

ตัวอย่าง: การแทรกข้อมูลลงในคอลัมน์ CLOB

ตัวอย่างนี้แสดงวิธีการแทรกข้อมูลจาก ไฟล์ปกติที่ถูกอ้างอิงโดย : `hv_text_file` ลงในคอลัมน์ character large object (CLOB)

ในส่วนคำจำกัดความของเซ็กเมนต์โปรแกรมภาษา C ดังต่อไปนี้:

- `userid` เป็นตัวแทนไดเร็กทอรีของหนึ่งในผู้ใช้ของคุณ.
- `dirname` เป็นตัวแทนชื่อไดเร็กทอรีย่อยของ “userid”.
- `filnam.1` สามารถแปลงเป็นชื่อของเอกสารที่คุณต้องการแทรกเข้าไปในตาราง.
- `clobtab` คือชื่อของตารางที่มีชนิดข้อมูล CLOB.

```
strcpy(hv_text_file.name, "/home/userid/dirname/filnam.1");
hv_text_file.name_length = strlen("/home/userid/dirname/filnam.1");
hv_text_file.file_options = SQL_FILE_READ; /* this is a 'regular' file */

EXEC SQL INSERT INTO CLOBTAB
  VALUES(:hv_text_file);
```

การแสดงโครงร่างของคอลัมน์ LOB

เมื่อคุณใช้คำสั่ง CL เช่น Display Physical File Member (DSPPFM) เพื่อแสดงแถวข้อมูลจากตารางที่มีคอลัมน์ large object (LOB) ข้อมูล LOB ที่เก็บไว้ในแถวนั้นจะไม่ถูกแสดง แต่ฐานข้อมูล จะแสดงค่าพิเศษสำหรับคอลัมน์ LOB แทน

โครงร่างของค่าพิเศษนี้มีค่าดังต่อไปนี้:

- 13 ถึง 28 ไบต์จะเป็นค่าศูนย์ในเลขฐานสิบหก.
- 16 ไบต์จะขึ้นต้นด้วย *POINTER และตามด้วยช่องว่าง.

จำนวนของไบต์ในส่วนแรกของค่าถูกตั้งค่าด้วยจำนวนที่จำเป็นในการจัดตำแหน่งทีละ 16 ไบต์กับส่วนที่สองของค่า.

ตัวอย่างเช่น, ถ้าคุณมีตารางที่เก็บ 3 คอลัมน์: ColumnOne Char(10), ColumnTwo CLOB(40K), และ ColumnThree BLOB (10M). ถ้าคุณใช้คำสั่ง DSPPFM กับตารางนี้, แต่ละแถวของข้อมูลจะมีค่าดังนี้.

- สำหรับ ColumnOne: 10 ไบต์จะถูกเติมค่าด้วยข้อมูลอักขระ.
- สำหรับ ColumnTwo: 22 ไบต์จะถูกเติมค่าด้วยศูนย์ในเลขฐานสิบหก และ 16 ไบต์จะถูกเติมค่าด้วย '*POINTER '.
- สำหรับ ColumnThree: 16 ไบต์จะถูกเติมค่าด้วยศูนย์ในเลขฐานสิบหก และ 16 ไบต์จะถูกเติมค่าด้วย '*POINTER '.

ชุดของคำสั่งทั้งหมดที่แสดงคอลัมน์ LOB ด้วยวิธีนี้คือ:

- Display Physical File Member (DSPPFM)
- Copy File (CPYF) เมื่อค่า *PRINT ถูกระบุสำหรับคีย์เวิร์ด TOFILE
- Display Journal (DSPJRN)
- Retrieve Journal Entry (RTVJRNE)
- Receive Journal Entry (RCVJRNE) เมื่อค่า *TYPE1, *TYPE2, *TYPE3 และ *TYPE4 ถูกระบุสำหรับคีย์เวิร์ด ENTFMT.

การแสดงโครงร่าง Journal entry ของคอลัมน์ LOB

คำสั่งเหล่านี้จะส่งคืนบัฟเฟอร์ที่ให้ผู้ใช้สามารถเข้าถึงข้อมูล large object (LOB) ที่ได้ผ่านการเจอร์นัลแล้ว

- Receive Journal Entry (RCVJRNE) CL command, เมื่อค่า *TYPEPTR ถูกระบุสำหรับคีย์เวิร์ด ENTFMT
- Retrieve Journal Entries (QjoRetrieveJournalEntries) API

โครงร่างของคอลัมน์ LOB ใน entry นี้จะเป็นดังต่อไปนี้:

- 0 ถึง 15 ไบต์เป็นค่าศูนย์ในฐานสิบหก
- 1 ไบต์ของข้อมูลระบบตั้งค่าเป็น '00'x
- 4 ไบต์เก็บความยาวของข้อมูล LOB ที่จัดการโดยตัวชี้, ด้านล่าง
- 8 ไบต์ของค่าศูนย์ในฐานสิบหก
- 16 ไบต์เก็บค่าตัวชี้ไปยังข้อมูล LOB ที่เก็บไว้ใน Journal Entry.

ส่วนแรกของโครงร่างนี้เจตนาให้จัดตำแหน่งทีละ 16 ไบต์กับตัวชี้ไปยังข้อมูล LOB. จำนวนไบต์ในส่วนนี้จะขึ้นอยู่กับความยาวของคอลัมน์ที่ดำเนินการกับคอลัมน์ LOB. สำหรับตัวอย่างของวิธีการคำนวณความยาวของส่วนแรกนี้ให้อ้างอิงถึงส่วนด้านบนที่เกี่ยวกับการแสดงโครงร่างของคอลัมน์ LOB .

User-defined distinct type

user-defined distinct type (UDT) คือกลไก ที่ขยายความสามารถของ DB2 ให้มี ชนิดข้อมูลมากไปกว่าที่มีอยู่

User-defined distinct types ทำให้คุณสามารถกำหนดชนิดข้อมูลสำหรับ DB2 ขึ้นมาใหม่ ซึ่งให้ความสามารถที่มากขึ้นเนื่องจากคุณไม่จำเป็นต้องถูกจำกัดให้ใช้แค่ชนิดข้อมูลในตัวที่ระบบจัดเตรียมให้ในการจำลองแบบทางธุรกิจและโครงสร้างข้อมูลอีกต่อไป ชนิดข้อมูลแบบ Distinct อนุญาตให้คุณจับคู่แบบหนึ่งต่อหนึ่งกับชนิดที่มีอยู่แล้วในฐานข้อมูลได้.

มีประโยชน์หลายอย่างที่เชื่อมโยงกับ UDT:

- **ความสามารถในการต่อขยาย.**
  ด้วยการนิยามชนิดขึ้นมาใหม่, คุณสามารถเพิ่มชุดของชนิดใน DB2 เพื่อสนับสนุนแอ็พพลิเคชันของคุณได้อย่างไม่จำกัด.
- **ความยืดหยุ่น.**
  คุณสามารถระบุความหมายและพฤติกรรมของชนิดใหม่ของคุณได้โดยใช้ User-defined Functions (UDFs) เพื่อเพิ่มความหลากหลายของชนิดที่ใช้ได้ในระบบ.
- **พฤติกรรมที่ไม่เปลี่ยนแปลง.**
  ความเข้มงวดในเรื่องชนิดจะทำให้มั่นใจว่า UDTs ของคุณจะทำงานอย่างเหมาะสม ซึ่งจะรับประกันว่าเฉพาะฟังก์ชันที่นิยามบน UDT ของคุณเท่านั้นที่จะถูกใช้กับ Instance ของ UDT.

- **การห่อหุ้ม (Encapsulation).**

  พฤติกรรมของ UDT ของคุณจะถูกจำกัดโดยฟังก์ชันและตัวดำเนินการที่ใช้ได้กับ UDT ของคุณเท่านั้น. สิ่งนี้ทำให้เกิดความยืดหยุ่นในการนำไปปฏิบัติเนื่องจากการทำงานของแอ็พพลิเคชันไม่ได้ขึ้นอยู่กับการแทนค่าภายในที่คุณเลือกสำหรับชนิดข้อมูลของคุณ.
- **พฤติกรรมที่สามารถขยายได้.**

  การนิยามของ User-defined Function บนชนิดสามารถเพิ่มหน้าที่ซึ่งจัดเตรียมไว้เพื่อดำเนินการกับ UDT ได้ตลอดเวลา.
- **พื้นฐานสำหรับส่วนขยาย object-oriented.**

  UDTs คือส่วนขยายของคุณลักษณะที่สำคัญของ object-oriented. มันคือก้าวที่สำคัญสู่การเป็นส่วนขยายของ.

  หลักการที่เกี่ยวข้อง

  "ประเภทที่ผู้ใช้กำหนด (User-defined types)" ในหน้า 13
  ประเภทที่ผู้ใช้กำหนด คือ ประเภท ข้อมูลจำเพาะที่ผู้ใช้สามารถกำหนดได้โดยไม่ขึ้นกับประเภทข้อมูลที่มีอยู่ในระบบจัดการฐานข้อมูล

การนิยาม UDT

คุณกำหนด user-defined type (UDT) โดยใช้ คำสั่ง CREATE DISTINCT TYPE

สำหรับคำสั่ง CREATE DISTINCT TYPE, โปรดสังเกตุว่า:

1. ชื่อของ UDT ใหม่อาจเป็นชื่อที่ครบตามเกณฑ์หรือไม่ครบตามเกณฑ์ก็ได้.
2. ชนิดต้นฉบับของ UDT ถูกใช้โดยระบบเพื่อแทนค่า UDT ภายใน. ด้วยเหตุผลนี้, จึงจำเป็นต้องเป็นชนิดข้อมูลในตัว. UDT ที่นิยามขึ้นมาก่อนหน้าจะไม่สามารถใช้เป็นชนิดต้นฉบับของ UDT ได้.

เนื่องจากเป็นส่วนของการนิยาม UDT, ระบบจะสร้างฟังก์ชันที่แปลงชนิดข้อมูลให้เสมอเพื่อ:

- แปลงชนิดข้อมูลจาก UDT ไปเป็นชนิดต้นฉบับ, โดยใช้ชื่อมาตรฐานของชนิดต้นฉบับ. ตัวอย่างเช่น, ถ้าคุณสร้าง Distinct Type โดยอยู่บนพื้นฐานของ FLOAT, แล้วฟังก์ชันการแปลงชนิดข้อมูลที่ชื่อ DOUBLE จะถูกสร้างขึ้นมา.
- แปลงจากชนิดต้นฉบับไปเป็น UDT.

ฟังก์ชันนี้จะสำคัญมากสำหรับการดำเนินการของ UDT ในเคียวรี่.

ฟังก์ชันพาธถูกใช้เพื่อแก้ปัญหาการอ้างอิงใดๆ ไปยังชื่อชนิดหรือฟังก์ชันที่ไม่ถูกต้อง, ยกเว้นแต่ชื่อชนิดหรือฟังก์ชันนั้นเป็นอ็อบเจ็กต์หลักของคำสั่ง CREATE, DROP, หรือ COMMENT ON.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การใช้การอ้างอิงฟังก์ชันตามเกณฑ์" ในหน้า 210
ถ้าคุณใช้การอ้างอิงฟังก์ชันที่ครบตามเกณฑ์, คุณต้องจำกัดการค้นหาสำหรับฟังก์ชันที่ตรงกันกับแบบแผนที่ระบุ

CREATE TYPE

ตัวอย่าง: เงินตรา:

สมมติว่า คุณกำลังเขียนแอ็พพลิเคชันที่ จัดการกับระบบเงินตราหลายๆ ระบบ และต้องการตรวจสอบว่า DB2 ไม่อนุญาตให้ระบบเงินตราเหล่านี้ถูก เปรียบเทียบ หรือนำมาใช้ร่วมกันโดยตรงกับในเคียวรี่อื่น

โปรดจำไว้ว่าการแปลงค่าจะมีความจำเป็นถ้าคุณต้องการเปรียบเทียบค่าของระบบเงินตราที่ต่างกัน. ดังนั้นคุณนิยาม UDT ได้มากเท่าที่คุณต้องการ; หนึ่งในแต่ละระบบเงินตราที่คุณอาจจำเป็นต้องแทนค่าคือ:

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL (9,2)
CREATE DISTINCT TYPE CANADIAN_DOLLAR AS DECIMAL (9,2)
CREATE DISTINCT TYPE EURO AS DECIMAL (9,2)
```

ตัวอย่าง: ประวัติย่อ:

สมมติว่า คุณต้องการเก็บแบบฟอร์มสมัครงาน ที่เติมข้อมูลโดยผู้สมัครงานกับบริษัทของคุณไว้ในตาราง และ คุณกำลังจะใช้ฟังก์ชันเพื่อดึงข้อมูลจากแบบฟอร์มเหล่านี้

เนื่องจากฟังก์ชันเหล่านี้ไม่สามารถใช้ได้กับสตริงอักขระทั่วไป (เพราะว่าไม่สามารถค้นหาข้อมูลที่ควรจะคืนค่ามาได้), คุณจึงนิยาม UDT เพื่อแทนค่าฟอร์มที่เติมข้อมูลแล้ว:

```
CREATE DISTINCT TYPE PERSONAL.APPLICATION_FORM AS CLOB(32K)
```

การนิยามตารางด้วย UDT

หลังจากที่คุณได้นิยามชนิดแบบผู้ใช้กำหนดเอง (UDT) หลายชนิดแล้ว คุณสามารถเริ่มนิยามตารางด้วยคอลัมน์ที่มีชนิดเป็น UDT ได้

ตัวอย่าง: การขาย:

สมมติว่า คุณต้องการนิยามตารางเพื่อเก็บ ยอดขายของบริษัทของคุณในประเทศต่างๆ

คุณสร้างตารางได้ดังนี้:

```
CREATE TABLE US_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         US_DOLLAR)

CREATE TABLE CANADIAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         CANADIAN_DOLLAR)

CREATE TABLE GERMAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         EURO)
```

UDT ในตัวอย่างด้านบนนี้ถูกสร้างโดย ใช้คำสั่ง CREATE DISTINCT TYPE เดียวกันใน “ตัวอย่าง: เงินตรา” ในหน้า 238 โปรดสังเกตว่า ตัวอย่างด้านบนจะใช้ข้อจำกัดการตรวจสอบ

ตัวอย่าง: แบบฟอร์มสมัครงาน:

สมมติว่า คุณต้องการนิยามตารางเพื่อเก็บ ฟอร์มที่กรอกโดยผู้สมัคร

สร้างตารางได้ดังนี้:

```
CREATE TABLE APPLICATIONS
  (ID               INTEGER,
   NAME             VARCHAR (30),
   APPLICATION_DATE DATE,
   FORM             PERSONAL.APPLICATION_FORM)
```

คุณต้องใช้ชื่อ UDT แบบครบถ้วนตามเกณฑ์เนื่องจาก qualifier ไม่ใช่ Authorization ID เดียวกับคุณ และคุณไม่ได้เปลี่ยนแปลงดีฟอลต์ฟังก์ชันพาธ. โปรดจำไว้ว่าเมื่อใดก็ตามที่ชื่อชนิดหรือชื่อฟังก์ชันไม่ถูกต้องตามเกณฑ์แล้ว, DB2 จะค้นหาในรายชื่อ schemas ของฟังก์ชันพาธปัจจุบันเพื่อหาชื่อชนิดหรือชื่อฟังก์ชันที่ใกล้เคียงกัน.

การจัดการ UDT

ความเข้มงวดในเรื่องชนิด เป็นแนวคิดที่สำคัญที่เกี่ยวข้อง กับชนิดที่ผู้ใช้กำหนดเอง (UDT) ความเข้มงวดในเรื่องชนิดจะรับประกันว่า เฉพาะฟังก์ชันและตัวดำเนินการที่ถูกนิยามบน UDT เท่านั้นที่สามารถใช้ได้กับ instance ของมัน

ความเข้มงวดในเรื่องชนิดจะสำคัญมากในการทำให้มั่นใจว่า instance ของ UDT ของคุณนั้นถูกต้อง. ตัวอย่างเช่น, ถ้าคุณได้นิยามฟังก์ชันเพื่อแปลงดอลลาร์สหรัฐไปเป็นดอลลาร์แคนาดาตามอัตราแลกเปลี่ยนปัจจุบัน, คุณไม่ต้องการให้ฟังก์ชันเดียวกันนี้ถูกใช้ในการแปลค่าเงินยูโรไปเป็นแคนาดา เนื่องจากฟังก์ชันนี้จะคืนค่าที่ผิดมาอย่างแน่นอน.

ผลที่ตามมาของการเข้มงวดในเรื่องชนิด, DB2 จะไม่อนุญาตให้คุณเขียนเคียวรี่ที่ทำการเปรียบเทียบ, อย่างเช่น, ระหว่าง instance ของ UDT กับ instance ของ UDT ต้นฉบับ. ด้วยเหตุผลเดียวกัน, DB2 จะไม่อนุญาตให้คุณใช้ฟังก์ชันที่ถูกนิยามบนชนิดอื่นกับ UDTs. ถ้าคุณต้องการเปรียบเทียบ instances ของ UDT กับ instance ชนิดอื่น, คุณจำเป็นต้องทำการแปลง instance ใด instance หนึ่ง. ในทำนองเดียวกัน, คุณจำเป็นต้องแปลง instance ของ UDT ให้เป็นชนิดของพารามิเตอร์ของฟังก์ชันที่ไม่ได้ถูกนิยามบน UDT ถ้าคุณต้องการใช้ฟังก์ชันนี้.

ตัวอย่าง: การใช้ UDT

ตัวอย่างต่อไปนี้แสดงการใช้ user-defined type (UDT) ในสถานการณ์ต่างๆ

ตัวอย่าง: การเปรียบเทียบระหว่าง UDTs และค่าคงที่:

สมมติว่า คุณต้องการรู้ว่า สินค้าไหนที่ขายไป มากกว่า US $100 000.00 ในสหรัฐ เมื่อเดือนกรกฎาคม 1998

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > US_DOLLAR (100000)
  AND    month = 7
  AND    year  = 1998
```

เนื่องจากคุณไม่สามารถเปรียบเทียบดอลลาร์สหรัฐกับ instance ต้นฉบับของดอลลาร์สหรัฐ (ซึ่งคือ, DECIMAL) ได้โดยตรง, คุณจึงใช้ฟังก์ชันการแปลงที่จัดเตรียมโดย DB2 เพื่อแปลงจาก DECIMAL ให้เป็นดอลลาร์สหรัฐ. คุณยังสามารถใช้ฟังก์ชันการแปลงอื่นที่จัดเตรียมโดย DB2 (ซึ่งคือ, ตัวที่ใช้แปลงจากดอลลาร์สหรัฐไปเป็น DECIMAL) และแปลงคอลัมน์ผลรวมไปเป็น DECIMAL. ไม่ว่าคุณจะใช้การแปลงอย่างใด, แปลงไปหรือแปลงกลับเป็น UDT, คุณสามารถใช้สัญญลักษณ์ค่ากำหนดการแปลงเพื่อทำการแปลงได้, หรือใช้สัญญลักษณ์หน้าที่. คุณสามารถเขียนเคียวรี่ข้างบนให้เป็นดังนี้:

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > CAST (100000 AS us_dollar)
  AND    MONTH = 7
  AND    YEAR  = 1998
```

ตัวอย่าง: การแปลงระหว่าง UDT:

สมมติว่า คุณต้องการนิยาม user-defined function (UDF) ที่ทำการแปลงดอลลาร์แคนาดาไปเป็นดอลลาร์สหรัฐ

คุณสามารถนำค่าอัตราแลกเปลี่ยนปัจจุบันมาจากไฟล์ที่อยู่ ภายนอก DB2 จากนั้นให้นิยาม UDF ที่รับค่าในแบบดอลลาร์แคนาดา เรียกดูไฟล์อัตราแลกเปลี่ยนเงินตรา และคืนค่าเป็นจำนวนเงินในหน่วยดอลลาร์สหรัฐ

ในครั้งแรกที่ดู, UDF นี้อาจรู้สึกว่าเขียนได้ง่าย. อย่างไรก็ตาม, คอมไพลเลอร์ภาษา C ไม่ทุกตัวที่สนับสนุนค่า DECIMAL. UDTs ที่เป็นตัวแทนระบบเงินตราต่างๆอาจถูกนิยามให้เป็นแบบ DECIMAL. UDF ของคุณจึงอาจจำเป็นต้องรับและคืนค่าเป็นค่า DOUBLE, เนื่องจากค่านี้เป็นชนิดข้อมูลเดียวเท่านั้นที่จัดเตรียมโดยภาษา C ซึ่งอนุญาตให้ใช้แทนค่า DECIMAL ได้โดยไม่สูญเสียความแม่นยำของทศนิยม. UDF ของคุณจึงควรนิยามดังนี้:

```
CREATE FUNCTION CDN_TO_US_DOUBLE(DOUBLE) RETURNS DOUBLE
  EXTERNAL NAME 'MYLIB/CURRENCIES(C_CDN_US)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC
```

อัตราแลกเปลี่ยนเงินตราระหว่างดอลลาร์แคนาดาและดอลลาร์สหรัฐอาจมีการเปลี่ยนแปลงในระหว่างการเรียก UDF สองครั้ง, ดังนั้นคุณจึงประกาศให้เป็น NOT DETERMINISTIC.

คำถามคือ, คุณจะทำการส่งผ่านค่าดอลลาร์แคนาดาไปยัง UDF นี้และรับค่าดอลลาร์สหรัฐจาก UDF นี้ได้อย่างไร? ค่าดอลลาร์แคนาดาต้องถูกแปลงชนิดให้เป็นค่า DECIMAL. ค่า DECIMAL จะต้องถูกแปลงชนิดให้เป็น DOUBLE. และคุณยังจำเป็นต้องคืนค่า DOUBLE ที่ถูกแปลงชนิดให้เป็น DECIMAL และค่า DECIMAL ที่ถูกแปลงชนิดให้เป็นดอลลาร์สหรัฐ.

การแปลงจะทำให้อย่างอัตโนมัติโดย DB2 ทุกครั้งที่คุณนิยาม UDF ต้นฉบับ, โดยที่พารามิเตอร์และค่าคืนกลับมีชนิดไม่ตรงกับพารามิเตอร์และค่าที่คืนกลับของฟังก์ชันต้นฉบับ. ดังนั้น, คุณจึงจำเป็นต้องนิยาม UDF ต้นฉบับสองตัว. ตัวแรกจะนำค่า DOUBLE แล้วแทนค่าเป็น DECIMAL. ตัวที่สองจะนำค่า DECIMAL แล้วแทนค่าเป็น UDT. นิยามได้ดังต่อไปนี้:

```
CREATE FUNCTION CDN_TO_US_DEC (DECIMAL(9,2)) RETURNS DECIMAL(9,2)
  SOURCE CDN_TO_US_DOUBLE (DOUBLE)

CREATE FUNCTION US_DOLLAR (CANADIAN_DOLLAR) RETURNS US_DOLLAR
  SOURCE CDN_TO_US_DEC (DECIMAL())
```

โปรดสังเกตุว่าการเรียกของฟังก์ชัน US_DOLLAR เป็นแบบ US_DOLLAR(C1), ซึ่ง C1 คือคอลัมน์ที่ชนิดคือดอลลาร์แคนาดา, จะมีผลเช่นเดียวกับการเรียก:

```
US_DOLLAR (DECIMAL(CDN_TO_US_DOUBLE (DOUBLE (DECIMAL (C1)))))
```

นั่นคือ, C1 (ในดอลลาร์แคนาดา)จะถูกแปลงชนิดให้เป็น DECIMAL ซึ่งจะถูกแปลงให้เป็นค่า DOUBLE อีกทีหนึ่งก่อนที่จะผ่านค่าไปยังฟังก์ชัน CDN_TO_US_DOUBLE. ฟังก์ชันนี้จะใช้ไฟล์อัตราแลกเปลี่ยนเงินตราและคือค่า DOUBLE(ที่แทนจำนวนดอลลาร์สหรัฐ)ที่จะแปลงชนิดไปเป็น DECIMAL, แล้วจึงแปลงชนิดไปเป็นดอลลาร์สหรัฐอีกทีหนึ่ง.

ฟังก์ชันที่แปลงค่าเงินยูโรไปเป็นดอลลาร์สหรัฐจะคล้ายกับตัวอย่างข้างบน:

```
CREATE FUNCTION EURO_TO_US_DOUBLE(DOUBLE)
  RETURNS DOUBLE
  EXTERNAL NAME 'MYLIB/CURRENCIES(C_EURO_US)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC

CREATE FUNCTION EURO_TO_US_DEC (DECIMAL(9,2))
  RETURNS DECIMAL(9,2)
  SOURCE EURO_TO_US_DOUBLE(DOUBLE)

CREATE FUNCTION US_DOLLAR(EURO) RETURNS US_DOLLAR
  SOURCE EURO_TO_US_DEC (DECIMAL())
```

ตัวอย่าง: การเปรียบเทียบที่มี UDT รวมอยู่ด้วย:

สมมติว่า คุณต้องการรู้ว่า ผลิตภัณฑ์ไหนที่ขายในสหรัฐอเมริกามากกว่าในแคนาดา และเยอรมันสำหรับเดือนมีนาคม 2003

ใช้คำสั่ง SELECT ต่อไปนี้:

```
SELECT US.PRODUCT_ITEM, US.TOTAL
  FROM US_SALES AS US, CANADIAN_SALES AS CDN, GERMAN_SALES AS GERMAN
  WHERE US.PRODUCT_ITEM = CDN.PRODUCT_ITEM
  AND US.PRODUCT_ITEM = GERMAN.PRODUCT_ITEM
  AND US.TOTAL > US_DOLLAR (CDN.TOTAL)
  AND US.TOTAL > US_DOLLAR (GERMAN.TOTAL)
  AND US.MONTH = 3
  AND US.YEAR  = 2003
  AND CDN.MONTH = 3
  AND CDN.YEAR  = 2003
  AND GERMAN.MONTH = 3
  AND GERMAN.YEAR  = 2003
```

เนื่องจากคุณไม่สามารถทำการเปรียบเทียบดอลลาร์สหรัฐกับดอลลาร์แคนาดา หรือค่าเงินยูโร, คุณจึงใช้ UDF เพื่อแปลงจำนวนดอลลาร์แคนาดาให้เป็นดอลลาร์สหรัฐ, UDF เพื่อแปลงจำนวนในค่าเงินยูโรให้เป็นดอลลาร์สหรัฐ. คุณไม่สามารถแปลงค่าทั้งหมดให้เป็น DECIMAL แล้วเปรียบเทียบค่า DECIMAL ที่ถูกแปลงแล้วได้เนื่องจากจำนวนเงินนี้ไม่สามารถเปรียบเทียบกันได้เพราะว่าไม่ได้อยู่ในระบบเงินตราเดียวกัน.

ตัวอย่าง: UDF ต้นฉบับที่มี UDT รวมอยู่ด้วย:

สมมติว่า คุณได้นิยามฟังก์ชันที่ผู้ใช้กำหนด (UDF) บนฟังก์ชันในตัว SUM เพื่อสนับสนุน SUM ของค่าเงินยูโร

คำสั่งฟังก์ชันจะเป็นดังนี้:

```
CREATE FUNCTION SUM (EURO)
  RETURNS EURO
  SOURCE SYSIBM.SUM (DECIMAL())
```

คุณต้องการรู้ยอดรวมของการขายในเยอรมันสำหรับแต่ละผลิตภัณฑ์ในปี 2004. และคุณต้องการยอดรวมการขายในสกุลเงินดอลลาร์สหรัฐ:

```
SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM GERMAN_SALES
  WHERE YEAR = 2004
  GROUP BY PRODUCT_ITEM
```

คุณไม่สามารถใช้ SUM (US_DOLLAR (TOTAL)), ได้จนกว่าคุณจะกำหนดฟังก์ชัน SUM ของดอลลาร์สหรัฐในแบบเดียวกับด้านบน.

สิ่งอ้างอิงที่เกี่ยวข้อง

"ตัวอย่าง: การกำหนดค่าที่มี UDT ที่ต่างกันรวมอยู่ด้วย" ในหน้า 244
สมมติว่า คุณได้นิยามฟังก์ชันที่ผู้ใช้ กำหนด (UDF) บนฟังก์ชันในตัว SUM เพื่อสนับสนุน SUM ของดอลลาร์สหรัฐ และดอลลาร์แคนาดา

ตัวอย่าง: การกำหนดค่าที่มี UDT รวมอยู่ด้วย:

สมมติว่า คุณต้องการเก็บฟอร์มที่ กรอกโดยผู้สมัครใหม่เข้าไปในฐานข้อมูล

คุณได้นิยามตัวแปรโฮสต์ที่เก็บค่าสตริงอักขระที่ใช้เพื่อเป็นตัวแทนฟอร์มที่ถูกกรอกแล้ว:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS CLOB(32K) hv_form;
EXEC SQL END DECLARE SECTION;

/* Code to fill hv_form */

INSERT INTO APPLICATIONS
  VALUES (134523, 'Peter Holland', CURRENT DATE, :hv_form)
```

คุณไม่ได้เรียกฟังก์ชันการแปลงเพื่อแปลงสตริงอักขระไปเป็น UDT personal.application_form โดยตรง. นั่นก็เพราะ DB2 ยอมให้คุณกำหนดค่า instance ของซอร์สชนิด UDT ให้กับปลายทางที่มี UDT นั้นได้.

สิ่งอ้างอิงที่เกี่ยวข้อง

"ตัวอย่าง: การกำหนดค่าใน SQL แบบไดนามิก"
ถ้าคุณต้องการเก็บแบบฟอร์มใบสมัครโดยใช้ SQL แบบไดนามิก คุณสามารถใช้ตัวทำเครื่องหมายพารามิเตอร์

ตัวอย่าง: การกำหนดค่าใน SQL แบบไดนามิก:

ถ้าคุณต้องการเก็บแบบฟอร์มใบสมัครโดยใช้ SQL แบบไดนามิก คุณสามารถใช้ตัวทำเครื่องหมายพารามิเตอร์

คำสั่งเป็นดังนี้:

```
EXEC SQL BEGIN DECLARE SECTION;
  long id;
  char name[30];
  SQL TYPE IS CLOB(32K) form;
  char command[80];
EXEC SQL END DECLARE SECTION;

/* Code to fill host variables */

strcpy(command,"INSERT INTO APPLICATIONS VALUES");
strcat(command,"(?, ?, CURRENT DATE, ?)");
```

```
EXEC SQL PREPARE APP_INSERT FROM :command;
EXEC SQL EXECUTE APP_INSERT USING :id, :name, :form;
```

คุณใช้ค่ากำหนดการแปลงของ DB2 เพื่อบอกกับ DB2 ว่าประเภทของตัวทำเครื่องหมายเป็นแบบ CLOB(32K), ซึ่งเป็นชนิดที่สามารถกำหนดให้กับคอลัมน์ UDT ได้. โปรดจำไว้ว่าคุณไม่สามารถประกาศตัวแปรโฮสต์ของชนิด UDT ได้, เนื่องจากภาษาโฮสต์ไม่ได้สนับสนุน UDT. ดังนั้น, คุณไม่สามารถระบุชนิดของตัวทำเครื่องหมายพารามิเตอร์ให้เป็น UDT ได้.

สิ่งอ้างอิงที่เกี่ยวข้อง

"ตัวอย่าง: การกำหนดค่าที่มี UDT รวมอยู่ด้วย" ในหน้า 243

สมมติว่า คุณต้องการเก็บฟอร์มที่ กรอกโดยผู้สมัครใหม่เข้าไปในฐานข้อมูล

ตัวอย่าง: การกำหนดค่าที่มี UDT ที่ต่างกันรวมอยู่ด้วย:

สมมติว่า คุณได้นิยามฟังก์ชันที่ผู้ใช้ กำหนด (UDF) บนฟังก์ชันในตัว SUM เพื่อสนับสนุน SUM ของดอลลาร์สหรัฐ และดอลลาร์แคนาดา

```
CREATE FUNCTION SUM (CANADIAN_DOLLAR)
  RETURNS CANADIAN_DOLLAR
  SOURCE SYSIBM.SUM (DECIMAL())

CREATE FUNCTION SUM (US_DOLLAR)
  RETURNS US_DOLLAR
  SOURCE SYSIBM.SUM (DECIMAL())
```

สมมติว่า หัวหน้าของคุณร้องขอให้คุณเก็บยอดรวมการขายทั้งปีในแบบดอลลาร์สหรัฐของแต่ละผลิตภัณฑ์ในแต่ละประเทศ, ในตารางที่แยกกัน:

```
CREATE TABLE US_SALES_04
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

CREATE TABLE GERMAN_SALES_04
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

CREATE TABLE CANADIAN_SALES_04
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

INSERT INTO US_SALES_04
  SELECT PRODUCT_ITEM, SUM (TOTAL)
  FROM US_SALES
  WHERE YEAR = 2004
  GROUP BY PRODUCT_ITEM

INSERT INTO GERMAN_SALES_04
  SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM GERMAN_SALES
  WHERE YEAR = 2004
  GROUP BY PRODUCT_ITEM
```

```
INSERT INTO CANADIAN_SALES_04
  SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM CANADIAN_SALES
  WHERE YEAR = 2004
  GROUP BY PRODUCT_ITEM
```

คุณจึงทำการแปลงจำนวนเงินในแบบดอลลาร์แคนาดา และค่าเงินยูโรไปเป็นดอลลาร์สหรัฐ เนื่องจาก UDT ที่ต่างกันจะไม่สามารถกำหนดค่าให้ UDT ตัวอื่นได้. คุณไม่สามารถใช้ไวยากรณ์ค่ากำหนดการแปลงได้ เนื่องจาก UDT สามารถถูกแปลงให้เป็นชนิดต้นฉบับได้เท่านั้น.

สิ่งอ้างอิงที่เกี่ยวข้อง

"ตัวอย่าง: UDF ต้นฉบับที่มี UDT รวมอยู่ด้วย" ในหน้า 242
สมมติว่า คุณได้นิยามฟังก์ชันที่ผู้ใช้กำหนด (UDF) บนฟังก์ชันในตัว SUM เพื่อสนับสนุน SUM ของค่าเงินยูโร

ตัวอย่าง: การใช้ UDT ในคำสั่ง UNION:

สมมติว่า คุณต้องการให้ผู้ใช้ในสหรัฐอเมริกา ใช้เคียวรี่ที่แสดงยอดขายของแต่ละผลิตภัณฑ์ของบริษัทของคุณ

คำสั่ง SELECT จะเป็นดังนี้:

```
SELECT PRODUCT_ITEM, MONTH, YEAR, TOTAL
FROM US_SALES
UNION
SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR (TOTAL)
FROM CANADIAN_SALES
UNION
SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR (TOTAL)
FROM GERMAN_SALES
```

คุณสามารถแปลงดอลลาร์แคนาดาให้เป็นดอลลาร์สหรัฐ และแปลงค่าเงินยูโรให้เป็นดอลลาร์สหรัฐได้ เนื่องจาก UDT สามารถรวมเข้ากันได้กับ UDT เดียวกันเท่านั้น. คุณต้องใช้สัญญลักษณ์เพื่อแปลงชนิดระหว่าง UDT เนื่องจากค่ากำหนดการแปลงจะอนุญาตให้คุณแปลงระหว่าง UDT และชนิดต้นฉบับของมันเท่านั้น.

ตัวอย่าง: การใช้ UDT, UDF, และ LOB

ตัวอย่างต่อไปนี้แสดงวิธีการใช้ประเภทแบบผู้ใช้กำหนดเอง (UDT), ฟังก์ชันแบบผู้ใช้กำหนดเอง (UDF) และอ็อบเจ็กต์ขนาดใหญ่ (LOB) พร้อมกันใน แอ็พพลิเคชันที่ซับซ้อน

ตัวอย่าง: การนิยาม UDT และ UDF

สมมติว่าคุณต้องการเก็บจดหมาย อิเล็กทรอนิก (อีเมล) ที่ส่งมายังบริษัทของคุณไว้ในตาราง

โดยไม่สนใจเรื่องความเป็นส่วนตัวแล้ว, คุณวางแผนที่จะเขียนเคียวรี่กับอีเมลเพื่อหาหัวข้อ, ความบ่อยของอีเมลเซอร์วิสที่ถูกใช้รับคำสั่งซื้อของลูกค้า, และอื่นๆ. อีเมลสามารถมีขนาดใหญ่ได้, และจะมีโครงสร้างภายในที่ซับซ้อน (ผู้ส่ง, ผู้รับ, เรื่อง, วันที่, และเนื้อหาอีเมล). ดังนั้น, คุณตัดสินใจเก็บอีเมลโดยใช้ UDT ที่ชนิดต้นฉบับคืออ็อบเจ็กต์ขนาดใหญ่. คุณนิยามชุดของ UDFs บนชนิดอีเมลของคุณ, เช่น ฟังก์ชันเพื่อดึงข้อมูลชื่อเรื่องของอีเมล, ชื่อผู้ส่ง, วันที่, และอื่นๆ. และคุณยังได้นิยามฟังก์ชันที่สามารถทำการค้นหาข้อความของอีเมลได้. คุณทำดังด้านบนโดยการใช้คำสั่ง CREATE ดังต่อไปนี้:

```
CREATE DISTINCT TYPE E_MAIL AS BLOB (1M)

CREATE FUNCTION SUBJECT (E_MAIL)
```

```
  RETURNS VARCHAR (200)
  EXTERNAL NAME 'LIB/PGM(SUBJECT)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION SENDER (E_MAIL)
  RETURNS VARCHAR (200)
  EXTERNAL NAME 'LIB/PGM(SENDER)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION RECEIVER (E_MAIL)
  RETURNS VARCHAR (200)
  EXTERNAL NAME 'LIB/PGM(RECEIVER)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION SENDING_DATE (E_MAIL)
  RETURNS DATE CAST FROM VARCHAR(10)
  EXTERNAL NAME 'LIB/PGM(SENDING_DATE)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION CONTENTS (E_MAIL)
  RETURNS BLOB (1M)
  EXTERNAL NAME 'LIB/PGM(CONTENTS)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION CONTAINS (E_MAIL, VARCHAR (200))
  RETURNS INTEGER
  EXTERNAL NAME 'LIB/PGM(CONTAINS)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
```

```
CREATE TABLE ELECTRONIC_MAIL
  (ARRIVAL_TIMESTAMP TIMESTAMP,
   MESSAGE E_MAIL)
```

ตัวอย่าง: การใช้ฟังก์ชัน LOB เพื่อใส่ค่าเข้าไปในฐานข้อมูล

สมมติว่า คุณใส่ค่าเข้าไปในตารางของคุณโดยการย้าย อีเมลของคุณที่เก็บไว้ในไฟล์เข้าไปในฐานข้อมูล DB2 for i5/OS

รันคำสั่ง INSERT ต่อไปนี้หลายๆ ครั้ง โดยใช้ค่า HV_EMAIL_FILE ที่ต่างกันจนกว่าคุณจะจัดเก็บอีเมลได้ทั้งหมด:

```
EXEC SQL BEGIN DECLARE SECTION
  SQL TYPE IS BLOB_FILE HV_EMAIL_FILE;

EXEC SQL END DECLARE SECTION
  strcpy (HV_EMAIL_FILE.NAME, "/u/mail/email/mbox");
  HV_EMAIL_FILE.NAME_LENGTH = strlen(HV_EMAIL_FILE.NAME);
  HV_EMAIL_FILE.FILE_OPTIONS = 2;

EXEC SQL INSERT INTO ELECTRONIC_MAIL
  VALUES (CURRENT TIMESTAMP, :hv_email_file);
```

ฟังก์ชันทั้งหมดที่มีใน DB2 ฟังก์ชันที่เกี่ยวกับ LOB สามารถใช้ได้กับ UDTs ที่มีชนิดต้นฉบับเป็นเ LOBs. ดังนั้น, คุณได้ใช้ตัวแปรอ้างอิงไฟล์ของ LOB เพื่อกำหนดค่าของไฟล์ให้กับคอลัมน์ของ UDT. คุณไม่ได้ใช้ฟังก์ชันการแปลงเพื่อแปลงค่าของชนิด BLOB ไปเป็นชนิดอีเมลของคุณ. นี่เป็นเพราะว่า DB2 อนุญาตให้คุณกำหนดค่าของชนิดต้นฉบับของ distinct type ให้กับปลายทางของ distinct type.

ตัวอย่าง: การใช้ UDF เพื่อเคียวรี่ instance ของ UDT

สมมติว่า คุณต้องการรู้จำนวนอีเมลที่ ส่งไปให้ลูกค้าเกี่ยวกับคำสั่งซื้อของลูกค้า และคุณมีอีเมลแอดเดรส ของลูกค้าในตารางลูกค้า

คำสั่งเป็นดังนี้:

```
SELECT COUNT (*)
  FROM ELECTRONIC_MAIL AS EMAIL, CUSTOMERS
  WHERE SUBJECT (EMAIL.MESSAGE) = 'customer order'
  AND CUSTOMERS.EMAIL_ADDRESS = SENDER (EMAIL.MESSAGE)
  AND CUSTOMERS.NAME = 'Customer X'
```

คุณได้ใช้ UDFs นิยามบน UDT ในเคียวรี่ SQL นี้เนื่องจากเป็นวิธีเดียวเท่านั้นที่ใช้ดำเนินการกับ UDT. ในกรณีนี้, อีเมลแบบ UDT ของคุณจะถูกห่อหุ้ม(encapsulated)อย่างสมบูรณ์. การแทนค่าและโครงสร้างภายในจะถูกซ่อนและสามารถถูกดำเนินการโดย UDFs ที่นิยามไว้เท่านั้น. UDF เหล่านี้จะรู้วิธีตีความข้อมูลโดยไม่จำเป็นต้องเปิดเผยค่าที่แทนที่อยู่.

สมมติว่า คุณต้องการรู้รายละเอียดอีเมลทั้งหมดที่บริษัทของคุณได้รับในปี 1994 ที่ทำให้ต้องปรับปรุงประสิทธิภาพผลิตภัณฑ์ของคุณในตลาด.

```
SELECT SENDER (MESSAGE), SENDING_DATE (MESSAGE), SUBJECT (MESSAGE)
  FROM ELECTRONIC_MAIL
  WHERE CONTAINS (MESSAGE,
  '"performance" AND "products" AND "marketplace"') = 1
```

คุณได้ใช้ contains UDF ที่สามารถวิเคราะห์เนื้อหาของการค้นหาข้อความสำหรับคีย์เวิร์ด หรือคำเหมือนที่ตรงประเด็น.

ตัวอย่าง: การใช้ LOB locator เพื่อจัดการกับ instance ของ UDT

สมมติว่า คุณต้องการดึงข้อมูลของอีเมลที่กำหนด แต่ ไม่ต้องการย้ายอีเมลทั้งหมดไปยังตัวแปรโฮสต์ในแอ็พพลิเคชัน โปรแกรมของคุณ

โปรดจำไว้ว่า อีเมลสามารถมีขนาดใหญ่ได้ เนื่องจาก user-defined type (UDT) ถูกนิยามเป็น large object (LOB) คุณจึงสามารถใช้ LOB locators สำหรับจุดประสงค์นั้นได้:

```
EXEC SQL BEGIN DECLARE SECTION
  long hv_len;
  char hv_subject[200];
  char hv_sender[200];
  char hv_buf[4096];
  char hv_current_time[26];
  SQL TYPE IS BLOB_LOCATOR hv_email_locator;
EXEC SQL END DECLARE SECTION

EXEC SQL SELECT MESSAGE
  INTO :hv_email_locator
  FROM ELECTRONIC MAIL
  WHERE ARRIVAL_TIMESTAMP = :hv_current_time;

EXEC SQL VALUES (SUBJECT (E_MAIL(:hv_email_locator))
  INTO :hv_subject;
.... code that checks if the subject of the e_mail is relevant ....
.... if the e_mail is relevant, then.............................

EXEC SQL VALUES (SENDER (CAST (:hv_email_locator AS E_MAIL)))
  INTO :hv_sender;
```

เนื่องจากตัวแปรโฮสต์ของคุณเป็นชนิด BLOB locator (ชนิดต้นฉบับของ UDT), คุณจึงทำการแปลง BLOB locator ไปเป็น UDT ของคุณโดยตรง, เมื่อใดก็ตามที่มันถูกใช้เป็นอากิวเมนต์ของ UDF ที่ถูกนิยามบน UDT.

การใช้ DataLink

ชนิดข้อมูล DataLink เป็นหนึ่งในส่วนพื้นฐานของการขยายชนิดของข้อมูลที่สามารถเก็บไว้ในไฟล์ฐานข้อมูลได้. แนวคิดของ DataLink คือข้อมูลที่เก็บจริงในคอลัมน์ที่เป็นตัวชี้ไปยังอ็อบเจ็กต์.

อ็อบเจ็กต์ที่ว่านี้สามารถเป็นอะไรก็ได้, ไฟล์รูปภาพ, เสียงที่บันทึกไว้, ไฟล์ข้อความ, และอื่นๆ. วิธีที่ใช้สำหรับอ้างอิงไปหาอ็อบเจ็กต์คือการเก็บ Uniform Resource Locator (URL). นี่ก็หมายความว่าแถวในตารางสามารถถูกใช้เก็บข้อมูลเกี่ยวกับอ็อบเจ็กต์ในชนิดข้อมูลแบบเดิม, และตัวอ็อบเจ็กต์เองก็สามารถถูกอ้างอิงถึงโดยใช้ชนิดข้อมูล DataLink ได้. ผู้ใช้สามารถใช้ฟังก์ชันแบบ Scalar ของ SQL เพื่อดึงค่าพาธของอ็อบเจ็กต์และเซิร์ฟเวอร์ที่เก็บอ็อบเจ็กต์นั้น (ดูที่เรื่องฟังก์ชันในตัว ในส่วนอ้างอิงของ SQL). ด้วยชนิดข้อมูล DataLink, จะมีความสัมพันธ์แบบหลวมๆ ระหว่างแถวและอ็อบเจ็กต์. สำหรับ instance, การลบแถวจะตัดความสัมพันธ์ไปยังอ็อบเจ็กต์ที่อ้างอิงถึงโดย DataLink, แต่ตัวอ็อบเจ็กต์เองอาจจะไม่ถูกลบ.

ตารางที่ถูกสร้างโดยมีคอลัมน์ DataLink สามารถใช้เก็บข้อมูลเกี่ยวกับอ็อบเจ็กต์ได้, โดยไม่จำเป็นต้องเก็บอ็อบเจ็กต์นั้นจริง. แนวคิดนี้ให้ความยืดหยุ่นแก่ผู้ใช้ในเรื่องชนิดของข้อมูลที่สามารถจัดการได้โดยใช้ตาราง. ถ้า, ตัวอย่างเช่น, ผู้ใช้มีวิดีโอคลิปอยู่หลายพันที่เก็บไว้ในระบบไฟล์รวมของเซิร์ฟเวอร์ของพวกเขา, พวกเขาอาจจะต้องการใช้ตาราง SQL เพื่อเก็บข้อมูลเกี่ยวกับวิดีโอคลิปเหล่านั้น. เนื่องจากผู้ใช้มีอ็อบเจ็กต์ที่เก็บอยู่ในไดเร็กทอรีอยู่แล้ว, พวกเขาจึงต้องการตาราง SQL เพื่อใช้อ้างอิงไปยังอ็อบเจ็กต์เท่านั้น, ไม่ได้ต้องการไว้เป็นที่เก็บข้อมูลจริง. ทางออกที่ดีก็คือการใช้ DataLinks. ตาราง SQL อาจใช้ชนิดข้อมูล

SQL แบบเดิมเพื่อเก็บข้อมูลของแต่ละคลิป, เช่น ชื่อเรื่อง, ความยาว, วันที่, เป็นต้น. แต่ตัวคลิปเองจะถูกอ้างอิงจากคอลัมน์ของ DataLink. แต่ละแถวในตารางจะเก็บค่า URL ของอ็อบเจ็กต์และหมายเหตุต่างๆ. ดังนั้นแอ็พพลิเคชันที่ทำงานกับคลิปสามารถดึงค่า URL โดยใช้อินเตอร์เฟสของ SQL ได้, แล้วจึงใช้บราวเซอร์หรือซอฟต์แวร์อื่นเพื่อทำงานกับ URL และแสดงผลวิดีโอคลิป.

มีประโยชน์หลายอย่างในการใช้เทคนิคนี้:

- ระบบไฟล์รวมสามารถเก็บ stream file ใดๆ ก็ได้.
- ระบบไฟล์รวมสามารถเก็บอ็อบเจ็กต์ที่มีขนาดใหญ่มากๆได้, ซึ่งไม่พอถ้าเป็นคอลัมน์ชนิดตัวอักขระ, หรือแม้กระทั้งคอลัมน์ชนิด LOB
- ธรรมชาติของความเป็นลำดับชั้นของระบบไฟล์รวมเหมาะสมดีกับการจัดระบบและการทำงานกับอ็อบเจ็กแบบ stream file.
- ด้วยการปล่อยให้อ็อบเจ็กจริงอยู่ภายนอกฐานข้อมูลและอยู่ภายในระบบไฟล์รวม, แอ็พพลิเคชันสามารถได้รับประสิทธิภาพที่ดีกว่าโดยการให้รันไทม์เอ็นจินของ SQL จัดการกับเคียวรี่และรายงาน, และให้ระบบไฟล์จัดการกับวิดีโอ, การแสดงภาพ, ข้อความ, และอื่นๆ.

การใช้ DataLinks ยังให้การควบคุมบนอ็อบเจ็กต์ในขณะที่สถานะของอ็อบเจ็กต์คือ "linked" ได้. คอลัมน์ DataLink สามารถถูกสร้างให้อ็อบเจ็กต์ที่ถูกอ้างอิงถึงไม่สามารถถูกลบ, ถูกย้าย, หรือถูกเปลี่ยนชื่อในขณะที่แถวในตาราง SQL นั้นกำลังอ้างอิงถึงอ็อบเจ็กต์อยู่. จะถือว่าอ็อบเจ็กต์นี้ถูกเชื่อมโยงอยู่. เมื่อแถวที่มีการอ้างอิงอยู่ถูกลบออก, อ็อบเจ็กต์จะถูกยกเลิกการเชื่อมโยง. เมื่อต้องการทำความเข้าใจกับแนวคิดนี้ทั้งหมด, คุณควรเข้าใจระดับของตัวควบคุมที่สามารถถูกระบุ เมื่อทำการสร้างคอลัมน์ DataLink ด้วย.

สิ่งอ้างอิงที่เกี่ยวข้อง

ชนิดข้อมูล

ระดับของการควบคุมลิงก์ใน DataLinks

คุณสามารถสร้างคอลัมน์ DataLink ได้ด้วยการควบคุมลิงก์ที่แตกต่างกัน.

NO LINK CONTROL:

ถ้าคอลัมน์ DataLink ถูกสร้างด้วย NO LINK CONTROL จะไม่มีการเชื่อมโยงเกิดขึ้น เมื่อแถวถูกเพิ่มเข้าไปยังตาราง SQL

URL จะถูกตรวจสอบความถูกต้องของไวยากรณ์, แต่จะไม่มีการตรวจสอบเพื่อให้แน่ใจว่าเซิร์ฟเวอร์สามารถเข้าถึงได้หรือไม่, หรือไม่ตรวจสอบว่าไฟล์นั้นมีอยู่จริงหรือไม่.

FILE LINK CONTROL พร้อมด้วยสิทธิสำหรับ FS:

เมื่อคอลัมน์ DataLink ถูกสร้างเป็นระดับ FILE LINK CONTROL พร้อมด้วยสิทธิสำหรับระบบไฟล์ (FS) ระบบจะตรวจสอบว่า ค่า DataLink ใดๆ เป็น URL ที่ถูกต้องหรือไม่ ด้วยชื่อเซิร์ฟเวอร์ และชื่อไฟล์ที่ถูกต้อง

ไฟล์ต้องมีอยู่จริงในเวลาที่แถวถูกแทรก เข้าไปในตาราง SQL เมื่อพบอ็อบเจ็กต์ อ็อบเจ็กต์จะถูกทำเครื่องหมายให้เป็นถูกเชื่อมโยงอยู่ นั่นก็ หมายความว่าอ็อบเจ็กต์ไม่สามารถถูกย้าย ลบ หรือเปลี่ยนชื่อ ในขณะที่อ็อบเจ็กต์นี้ ถูกเชื่อมโยงอยู่ได้ ด้วยเหมือนกัน, อ็อบเจ็กต์ไม่สามารถถูกเชื่อมโยงได้มากกว่าหนึ่งการเชื่อมโยง. ถ้าส่วนของชื่อเซิร์ฟเวอร์ของ URL ระบุถึงระบบทางไกล (remote system), ระบบนั้นต้องสามารถเข้าถึงได้. ถ้าแถวที่เก็บค่า DataLink ถูกลบ อ็อบเจ็กต์จะถูกยกเลิกการเชื่อมโยง ถ้าค่า DataLink ถูกอัพเดตให้เป็นค่าอื่น อ็อบเจ็กต์เดิมจะถูกยกเลิกการเชื่อมโยง และอ็อบเจ็กต์ใหม่จะถูกเชื่อมโยง

ระบบไฟล์รวมยังคงรับภาระสำหรับการจัดการการอนุญาตสำหรับอ็อบเจ็กต์ที่ถูกเชื่อมโยง. การอนุญาตจะไม่ถูกเปลี่ยนแปลงขณะดำเนินการเชื่อมต่อหรือยกเลิกการเชื่อมต่อ. ตัวเลือกนี้ให้การควบคุมการมีอยู่ของอ็อบเจ็กต์สำหรับช่วงระยะเวลาที่อ็อบเจ็กต์ถูกเชื่อมโยงอยู่.

FILE LINK CONTROL พร้อมด้วยสิทธิสำหรับ DB:

เมื่อคอลัมน์ DataLink ถูกสร้างขึ้นเป็นแบบ FILE LINK CONTROL พร้อมด้วยสิทธิสำหรับฐานข้อมูล (DB), URL จะถูกตรวจสอบความถูกต้อง และสิทธิ ที่มีอยู่ทั้งหมดของอ็อบเจ็กต์จะถูกลบออก

ความเป็นเจ้าของของอ็อบเจ็กต์จะถูกเปลี่นให้กับโปรไฟล์ผู้ใช้ชนิดพิเศษที่ระบบจัดเตรียมไว้ให้. ช่วงระยะเวลาที่อ็อบเจ็กต์ถูกเชื่อมโยงอยู่, สามารถเข้าถึงอ็อบเจ็กต์ได้ทางเดียวเท่านั้นโดยการดึงค่า URL จากตาราง SQL ที่มีอ็อบเจ็กต์ที่ถูกเชื่อมโยงอยู่. การทำเช่นนี้จะถูกจัดการโดยใช้โทเค็นพิเศษที่ใช้ในการเข้าถึงซึ่งจะถูกผนวกต่อท้ายให้กับ URL ที่คืนค่าโดย SQL. ถ้าไม่มีโทเค็นที่ใช้ในการเข้าถึงแล้ว, การพยายามเข้าถึงอ็อบเจ็กต์ทั้งหมดจะไม่สำเร็จเนื่องจากจะเป็นการละเมิดสิทธิในการใช้งาน. ถ้า URL ที่มีโทเค็นที่ใช้ในการเข้าถึงถูกดึงค่ามาจากตาราง SQL โดยวิธีปกติ(FETCH, SELECT INTO, อื่นๆ.) แล้วตัวกรองของระบบไฟล์จะตรวจสอบโทเค็นที่ใช้ในการเข้าถึงและจึงอนุญาตให้เข้าถึงอ็อบเจ็กต์ได้.

ตัวเลือกนี้ได้ในการควบคุมในการป้องกันการเปลี่ยนแปลงอ็อบเจ็กต์ที่ถูกเชื่อมโยงอยู่สำหรับผู้ใช้ที่พยายามเข้าถึงอ็อบเจ็กต์โดยตรง. เนื่องจากสามารถเข้าถึงอ็อบเจ็กได้ทางเดียวโดยการดึงค่าโทเค็นที่ใช้ในการเข้าถึงมาจากการดำเนินการ SQL, ผู้ดูแลระบบสามารถควบคุมการเข้าถึงอ็อบเจ็กต์ที่ถูกเชื่อมโยงอยู่ได้อย่างมีประสิทธิภาพโดยใช้การอนุญาตของฐานข้อมูลกับตาราง SQL ที่มีคอลัมน์ DataLink.

การทำงานกับ DataLinks

ในการทำงานกับ DataLinks คุณต้องเข้าใจ สภาวะแวดล้อมการประมวลผลของ DataLink

การสนับสนุนสำหรับชนิดข้อมูล DataLink สามารถแบ่งย่อยได้เป็นส่วนประกอบสามส่วนดังนี้:

1. สำหรับ DB2 มีชนิดข้อมูลที่เรียกว่า DATALINK. ชนิดข้อมูลนี้สามารถระบุในคำสั่ง SQL เช่น CREATE TABLE และ ALTER TABLE ได้. คอลัมน์จะไม่สามารถมีค่าดีฟอต์อื่นนอกจาก NULL. การเข้าถึงข้อมูลต้องใช้อินเตอร์เฟสของ SQL เท่านั้น. ที่เป็นเช่นนี้เนื่องจากตัว DataLink เองจะเข้ากันไม่ได้กับชนิดข้อมูลโฮสต์ใดๆ ก็ตาม. ฟังก์ชันแบบ Scalar ของ SQL สามารถถูกใช้เพื่อดึงค่า DataLink ในรูปแบบอักขระได้. จะมีฟังก์ชันแบบ Scalar ชื่อ DLVALUE ที่ต้องใช้ใน SQL เพื่อ INSERT และ UPDATE ค่าในคอลัมน์.
2. DataLink File Manager (DLFM) คือส่วนประกอบที่ดูแลสถานะของลิงก์สำหรับไฟล์หรือเซิร์ฟเวอร์, ตรวจดูการเปลี่ยแปลงของ metadata สำหรับแต่ละไฟล์ โค้ดนี้จะจัดการกับการเชื่อมโยง, การยกเลิกการเชื่อมโยง, และคำสั่งเกี่ยวกับ commitment control. ด้านที่สำคัญของ DataLinks คือ DLFM ไม่จำเป็นต้องอยู่บนระบบเดียวกันกับตาราง SQL ที่เก็บคอลัมน์ DataLink ดังนั้นตาราง SQL สามารถเชื่อมโยงไปยังอ็อบเจ็กต์ที่อยู่ในระบบไฟล์รวมของระบบเดียวกัน หรืออยู่ในระบบไฟล์รวมของรีโมตเซิร์ฟเวอร์ก็ได้
3. ตัวกรองของ DataLink จะต้องทำงานเมื่อระบบไฟล์พยายามดำเนินการกับไฟล์ที่อยู่ในไดเร็กทอรีที่ใช้เก็บอ็อบเจ็กต์ที่ถูกเชื่อมโยง. ส่วนประกอบนี้จะพิจารณาว่าไฟล์ถูกเชื่อมโยงหรือไม่, และโดยทางเลือกแล้ว, จะพิจารณาว่าผู้ใช้มีสิทธิในการเข้าถึงไฟล์หรือไม่. ถ้าชื่อไฟล์รวมโทเค็นที่ใช้ในการเข้าถึงไว้ด้วย, โทเค็นนั้นจะถูกตรวจสอบด้วย. เนื่องจากกระบวนการกรองนี้จะใช้เวลามากเป็นพิเศษ จึงต้องทำงานเมื่ออ็อบเจ็กต์ที่ถูกใช้อยู่ในหนึ่งในไดเร็กทอรีภายใน DataLink prefix ให้ดูคำอธิบายต่อไปนี้เกี่ยวกับ 'prefix'

เมื่อทำงานกับ DataLinks, จะมีหลายขั้นตอนที่ต้องทำเพื่อตั้งค่าระบบให้เหมาะสม:

- TCP/IP จะต้องถูกตั้งค่าบนระบบใดๆ ที่จะถูกใช้เมื่อทำงานกับ DataLinks. ซึ่งรวมถึงระบบที่กำลังสร้างตาราง SQL ที่มีคอลัมน์ DataLink , หรือระบบบที่จะมีอ็อบเจ็กต์ที่จะถูกเชื่อม. ในกรณีส่วนใหญ่แล้ว, จะอยู่บนระบบเดียวกัน. เนื่องจาก URL ที่ถูกใช้ในการอ้างอิงไปยังอ็อบเจ็กต์จะเก็บชื่อเซิร์ฟเวอร์แบบ TCP/IP, ดังนั้นชื่อนี้ต้องรู้จักโดยระบบที่กำลังจะเก็บ. คำสั่ง CFGTCP สามารถถูกใช้เพื่อตั้งค่าชื่อ TCP/IP ได้, หรือใช้เพื่อลงทะเบียนชื่อเซิร์ฟเวอร์แบบ TCP/IP.
- ระบบที่เก็บตาราง SQL จะต้องมี Relational Database Directory ที่ถูกปรับปรุงเพื่อส่งผลกับระบบฐานข้อมูลโลคัล, หรือระบบรีโมตทางเลือกใดๆ. คำสั่ง WRKRDBDIRE สามารถถูกใช้เพื่อเพิ่มหรือแก้ไขข้อมูลในไดเร็กทอรีนี้. สำหรับผลที่ตามมา, แนะนำว่าชื่อเซิร์ฟเวอร์แบบ TCP/IP และชื่อ Relational Database ควรจะใช้ชื่อเดียวกัน.
- เซิร์ฟเวอร์ DLFM จะต้องถูกเรียกทำงานบนระบบใดๆ ที่จะใช้เก็บอ็อบเจ็กต์ที่ถูกเชื่อมโยง. คำสั่ง STRTCPSVR *DLFM สามารถถูกใช้เพื่อเริ่มทำงานเซิร์ฟเวอร์ DLFM ได้. เซิร์ฟเวอร์ DLFM สามารถถูกจบการทำงานได้โดยใช้คำสั่ง CL คือ ENDTCPSVR *DLFM.

เมื่อ DLFM ถูกเรียกใช้งานแล้ว, มีบางขั้นตอนที่จำเป็นในการตั้งค่า DLFM. ฟังก์ชัน DLFM นี้จะใช้ได้โดยสคริปต์ที่สามารถทำงานได้ ซึ่งสคริปต์นั้นสามารถใส่ค่าได้จากอินเตอร์เฟสของ QShell. เมื่อต้องการเข้าถึงเชลล์อินเตอร์เฟสแบบโต้ตอบ, ให้ใช้คำสั่ง CL ชื่อ QSH. คำสั่งนี้จะแสดงหน้าจอป้อนคำสั่งขึ้นมาซึ่งทำให้คุณสามารถป้อนคำสั่งสคริปต์ DLFM ได้. คำสั่งสคริปต์ ″dfmadmin -help″ สามารถใช้เพื่อแสดงข้อความวิธีใช้และไดอะแกรมของไวยากรณ์. สำหรับฟังก์ชันที่ถูกใช้บ่อยแล้ว, คำสั่ง CL ยังได้ถูกจัดเตรียมไว้ด้วย. โดยการใช้คำสั่ง CL แล้ว, การตั้งค่า DLFM ทั้งหมดหรือเกือบทั้งหมดสามารถทำให้สำเร็จได้โดยไม่ต้องใช้สคริปต์อินเตอร์เฟส. ขึ้นอยู่กับความชอบของคุณ, คุณสามารถเลือกได้ว่าจะใช้คำสั่งสคริปต์จากหน้าจอป้อนคำสั่งของ QSH หรือว่าเลือกใช้คำสั่ง CL จากหน้าจอป้อนคำสั่งของ CL.

เนื่องจากฟังก์ชันเหล่านั้นจำเป็นสำหรับผู้ดูแลระบบหรือผู้ดูแลฐานข้อมูล, ดังนั้นจึงต้องการสิทธิในการใช้แบบ *IOSYSCFG.

การเพิ่ม prefix

prefix คือพาธหรือไดเร็กทอรีที่จะเก็บอ็อบเจ็กต์ที่ถูกเชื่อมโยง. เมื่อเริ่มติดตั้ง Data Links File Manager (DLFM) บนระบบ, ผู้ดูแลระบบต้องเพิ่ม prefix ใดๆ ที่จะถูกใช้สำหรับ DataLink ด้วย. คำสั่งสคริปต์ ″dfmadmin -add_prefix″ จะถูกใช้เพื่อเพิ่ม ″prefixe″. คำสั่ง CL ที่ใช้ในการเพิ่ม prefix คือ คำสั่ง Add Prefix to DataLink File Manager (ADDPFXDLFM).

ตัวอย่างเช่น, บนเซิร์ฟเวอร์ TESTSYS1, มีไดเร็กทอรี่ชื่อ /mydir/datalinks/ ที่เก็บอ็อบเจ็กต์ที่จะถูกเชื่อมโยง. ผู้ดูแลระบบใช้คำสั่ง ADDPFXDLFM PREFIX('/mydir/datalinks/') เพื่อเพิ่ม prefix ลิงก์ของ URL ต่อไปนี้ถูกต้องเพราะมีพาธที่มีค่า prefix ถูกต้อง:

```
http://TESTSYS1/mydir/datalinks/videos/file1.mpg
```

หรือ

```
file://TESTSYS1/mydir/datalinks/text/story1.txt
```

และยังเป็นไปได้ที่จะเอา prefix ออกโดยใช้คำสั่งสคริปต์ ″dfmadmin -del_prefix″. แต่ว่าฟังก์ชันนี้ไม่ได้ถูกใช้บ่อย เนื่องจากฟังก์ชันสามารถถูกเรียกใช้ได้ก็ต่อเมื่อไม่มีอ็อบเจ็กต์ที่ถูกเชื่อมโยงเหลืออยู่ในโครงสร้างไดเร็กทอรีที่เก็บอยู่ภายในชื่อ prefix.

หมายเหตุ:

1. ไดเร็กทอรีต่อไปนี้, รวมถึงไดเร็กทอรีย่อย, ไม่ควรนำมาใช้เป็น prefix สำหรับ DataLink:
   - /QIBM
   - /QReclaim
   - /QSR

- /QFPNWSSTG

2. นอกจากนี้แล้ว, ไม่ควรนำไดเร็กทอรีพื้นฐานเหล่านี้มาใช้เว้นแต่มี prefix เป็นไดเร็กทอรีย่อยอยู่ในไดเร็กทอรีเหล่านี้:
   - /home
   - /dev
   - /bin
   - /etc
   - /tmp
   - /usr
   - /lib

การเพิ่มฐานข้อมูลโฮสต์

ฐานข้อมูลโฮสต์ คือระบบฐานข้อมูลเชิงสัมพันธ์ซึ่งการร้องขอลิงก์เริ่มมาจากที่นี่. ถ้า DLFM อยู่บนระบบเดียวกันกับตาราง SQL ที่จะเก็บ DataLinks แล้ว, เฉพาะชื่อฐานข้อมูลโลคัลเท่านั้นที่จำเป็นในการใส่เพิ่ม. ถ้า DLFM จะมีการร้องขอลิงก์ที่มาจากระบบรีโมตแล้ว, ชื่อของระบบทั้งหมดต้องถูกลงทะเบียนด้วย DLFM. คำสั่งสคริปต์ที่ใช้ในการเพิ่มฐานข้อมูลโฮสต์คือ dfmadmin -add_db และ คำสั่ง CL คือคำสั่ง Add Host Database to DataLink File Manager (ADDHDBDLFM). ฟังก์ชันนี้ยังต้องการให้ไลบารีที่เก็บตาราง SQL ถูกลงทะเบียนด้วย.

ตัวอย่างเช่น บนเซิร์ฟเวอร์ TESTSYS1 ที่คุณได้เพิ่ม "prefix" /mydir/datalinks/ เข้าไปแล้ว คุณต้องการให้ตาราง SQL บนระบบโลคัลในไลบารี TESTDB หรือ PRODDB ได้รับอนุญาตเพื่อเชื่อมโยงอ็อบเจ็กต์บนเซิร์ฟเวอร์นี้ได้ ใช้คำสั่งต่อไปนี้:

```
ADDHDBDLFM HOSTDBLIB((TESTDB) (PRODDB)) HOSTDB(TESTSYS1)
```

เมื่อ DLFM ถูกเรียกทำงานแล้ว, และ "prefixe" และชื่อฐานข้อมูลโฮสต์ได้ถูกลงทะเบียนแล้ว, คุณสามารถเริ่มเชื่อมโยงอ็อบเจ็กต์ในระบบไฟล์ได้.

การใช้ SQL ในสภาพแวดล้อมที่แตกต่างกัน

คุณสามารถใช้ SQL ในสภาพแวดล้อมที่แตกต่างกันได้ในหลายกรณี

การใช้เคอร์เซอร์

เมื่อ SQL รันคำสั่ง SELECT แถวผลลัพธ์ จะประกอบขึ้นจากตารางผลลัพธ์ เคอร์เซอร์จะแสดงวิธีการเข้าถึงตารางผลลัพธ์

โดยจะถูกใช้ภายในโปรแกรม SQL เพื่อรักษาตำแหน่งในตารางผลลัพธ์. SQL จะใช้เคอร์เซอร์ให้ทำงานร่วมกับแถวในตารางผลลัพธ์และเพื่อทำให้แถวเหล่านั้นใช้ได้กับโปรแกรมของคุณ. โปรแกรมของคุณสามารถมีหลายเคอร์เซอร์, แม้ว่าแต่ละเคอร์เซอร์ต้องมีชื่อเฉพาะก็ตาม.

ข้อความที่เกี่ยวกับการใช้เคอร์เซอร์ประกอบด้วย:

- ข้อความ DECLARE CURSOR เพื่อกำหนดและตั้งชื่อเคอร์เซอร์และระบุแถวที่ต้องเรียกออกมาด้วยข้อความที่ใส่อยู่ที่เลือกไว้.

- ข้อความ OPEN และ CLOSE เพื่อเปิดและปิดเคอร์เซอร์สำหรับใช้ภายในโปรแกรม. ต้องเปิดเคอร์เซอร์ก่อนที่จะเรียกแถวใดๆ ออกมา.
- คำสั่ง FETCH เพื่อค้นหาแถวจากตารางผลลัพธ์ของเคอร์เซอร์ หรือเพื่อวางตำแหน่งเคอร์เซอร์บนอีกแถวหนึ่ง.
- ข้อความ UPDATE ... WHERE CURRENT OF เพื่ออัพเดตแถวปัจจุบันของเคอร์เซอร์.
- ข้อความ DELETE ... WHERE CURRENT OF เพื่อลบแถวปัจจุบันของเคอร์เซอร์.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การอัพเดตข้อมูลตามที่ดึงมาจากตาราง" ในหน้า 106
คุณสามารถอัพเดตแถวของข้อมูลตามที่คุณได้ดึงออกมาโดยใช้เคอร์เซอร์.

CLOSE

DECLARE CURSOR

DELETE

FETCH

UPDATE

ประเภทเคอร์เซอร์

SQL สนับสนุนเคอร์เซอร์แบบอนุกรมและแบบเลื่อนขึ้นลง ประเภท ของเคอร์เซอร์จะกำหนดวิธีการวางตำแหน่ง ซึ่งสามารถนำมาใช้ร่วมกับ เคอร์เซอร์ได้

เคอร์เซอร์แบบอนุกรมr

เคอร์เซอร์แบบอนุกรมคือเคอร์เซอร์ที่ถูกกำหนดโดยไม่มีคีย์เวิร์ด SCROLL.

สำหรับเคอร์เซอร์แบบอนุกรมนั้น, ตารางผลลัพธ์แต่ละแถวสามารถดึงข้อมูลออกมาได้เพียงหนึ่งครั้งต่อการเปิดเคอร์เซอร์. เมื่อเคอร์เซอร์ถูกเปิด, จะถูกวางตำแหน่งหน้าแถวแรกในตารางผลลัพธ์. เมื่อมีการใส่ข้อความ FETCH , เคอร์เซอร์จะถูกย้ายไปยังแถวถัดไปในตารางผลลัพธ์. แถวนั้นจะกลายเป็นแถวปัจจุบัน. หากมีการระบุตัวแปรโฮสต์ (ด้วย INTO clause บนคำสั่ง FETCH), SQL จะย้ายเนื้อหาของแถวปัจจุบันเข้าไปไว้ในตัวแปรโฮสต์ของโปรแกรม.

จะมีการทำซ้ำลำดับดังกล่าวทุกครั้งที่มีการใส่คำสั่ง FETCH จนกว่าจะสิ้นสุดข้อมูล (SQLCODE = 100). เมื่อสิ้นสุดข้อมูลแล้ว, ให้ปิดเคอร์เซอร์. คุณไม่สามารถเข้าถึงแถวใดๆ ในตารางผลลัพธ์หลังจากที่สิ้นสุดข้อมูลแล้ว. ในการใช้เคอร์เซอร์แบบอนุกรมซ้ำ, ก่อนอื่นคุณต้องปิดเคอร์เซอร์ จากนั้นจึงใส่ คำสั่ง OPEN อีกครั้ง. คุณไม่สามารถสำรองข้อมูลได้ด้วยการใช้เคอร์เซอร์แบบอนุกรม.

เคอร์เซอร์แบบเลื่อนได้

สำหรับเคอร์เซอร์แบบเลื่อนขึ้นลงนั้น, สามารถดึงข้อมูลออกจากแถวของตารางผลลัพธ์ได้หลายครั้ง. เคอร์เซอร์จะเคลื่อนผ่านตารางผลลัพธ์โดยยึดตามอ็อพชันตำแหน่งที่ระบุบนคำสั่ง FETCH. เมื่อเคอร์เซอร์ถูกเปิด, จะถูกวางตำแหน่งหน้าแถวแรกในตารางผลลัพธ์. เมื่อมีการใส่คำสั่ง FETCH, เคอร์เซอร์จะถูกวางตำแหน่งที่แถวในตารางผลลัพธ์ซึ่งถูกระบุโดยอ็อพชันตำแหน่ง. แถวนั้นจะกลายเป็นแถวปัจจุบัน. หากมีการระบุตัวแปรโฮสต์ (ด้วย INTO clause บนคำสั่ง FETCH), SQL จะย้ายเนื้อหาของแถวปัจจุบันเข้าไปไว้ในตัวแปรโฮสต์ของโปรแกรม. ไม่สามารถระบุตัวแปรโฮสต์สำหรับอ็อพชันตำแหน่ง BEFORE และ AFTER ได้.

จะมีการทำซ้ำลำดับดังกล่าวทุกครั้งที่มีการใส่คำสั่ง FETCH. ไม่จำเป็นต้องปิดเคอร์เซอร์ เมื่อสิ้นสุดข้อมูลหรือเริ่มต้นข้อมูล. อ็อพชันตำแหน่งทำให้โปรแกรมสามารถดึงข้อมูลแถวออกมาจากตารางได้อย่างต่อเนื่อง.

มีการใช้อ็อพชันการเลื่อนต่อไปนี้เพื่อวางตำแหน่งเคอร์เซอร์เมื่อใส่คำสั่ง FETCH . ตำแหน่งเหล่านี้จะสัมพันธ์กับตำแหน่งเคอร์เซอร์ปัจจุบันในตารางผลลัพธ์.

NEXT	วางตำแหน่งเคอร์เซอร์บนแถวถัดไป. นี่คือค่าเริ่มต้นหากไม่มีการระบุตำแหน่ง.
PRIOR	วางตำแหน่งเคอร์เซอร์บนแถวก่อนหน้านี้.
FIRST	วางตำแหน่งเคอร์เซอร์บนแถวแรก.
LAST	วางตำแหน่งเคอร์เซอร์บนแถวสุดท้าย.
BEFORE	วางตำแหน่งเคอร์เซอร์หน้าแถวแรก.
AFTER	วางตำแหน่งเคอร์เซอร์หลังแถวสุดท้าย.
CURRENT	ไม่ได้เปลี่ยนตำแหน่งเคอร์เซอร์.
RELATIVE n	ประเมินตัวแปรโฮสต์ หรือจำนวนเต็ม *n* ที่สัมพันธ์กับตำแหน่งปัจจุบันของเคอร์เซอร์. ตัวอย่าง เช่น, ถ้า *n* เท่ากับ –1, เคอร์เซอร์จะถูกวางตำแหน่งบนแถวก่อนหน้านี้ของตารางผลลัพธ์. ถ้า *n* เท่ากับ +3, เคอร์เซอร์จะถูกวางตำแหน่งสามแถวหลังแถวปัจจุบัน.

สำหร บเคอร์เซอร์แบบเลื่อนขึ้นลงนั้น, สามารถกำหนดจุดสิ้นสุดของตารางได้โดยใช้ :

```
FETCH AFTER FROM C1
```

เมื่อวางตำแหน่งเคอร์เซอร์ที่จุดสิ้นสุดของตารางแล้ว, โปรแกรมสามารถใช้อ็อพชันเลื่อน PRIOR หรือ RELATIVE เพื่อวางตำแหน่งและดึงข้อมูลออกมาโดยเริ่มต้นจากจุดสิ้นสุดของตาราง.

ตัวอย่าง: การใช้เคอร์เซอร์

ตัวอย่างต่อไปนี้แสดงคำสั่ง SQL ที่คุณ ควรรวมอยู่ในโปรแกรม เพื่อกำหนดและใช้เคอร์เซอร์แบบอนุกรมและแบบ เลื่อนขึ้นลง

สมมุติว่าโปรแกรมของคุณกำลังตรวจสอบข้อมูลเกี่ยวกับพนักงาน ในแผนก D11 คุณสามารถใช้เคอร์เซอร์แบบอนุกรมหรือแบบเลื่อนขึ้นลงอย่างใดอย่างหนึ่งเพื่อรับ ข้อมูลเกี่ยวกับแผนกจากตาราง CORPDATA.EMPLOYEE

สำหรับตัวอย่างเคอร์เซอร์แบบอนุกรม, โปรแกรมจะประมวลผลทุกแถวจากตาราง, อัพเดตงานสำหรับสมาชิกทั้งหมดของแผนก D11 และลบเร็กคอร์ดของพนักงานจากแผนกอื่นๆ.

ตารางที่ 41. ตัวอย่างเคอร์เซอร์แบบอนุกรม

คำสั่งเคอร์เซอร์ SQL แบบอนุกรม	อธิบายไว้ในส่วนนี้
EXEC SQL **DECLARE** THISEMP **CURSOR FOR** **SELECT** EMPNO, LASTNAME, WORKDEPT, JOB **FROM** CORPDATA.EMPLOYEE **FOR UPDATE OF** JOB END-EXEC.	"ขั้นที่ 1: การกำหนดเคอร์เซอร์" ในหน้า 257.
EXEC SQL **OPEN** THISEMP END-EXEC.	"ขั้นที่ 2: การเปิดเคอร์เซอร์" ในหน้า 258.
EXEC SQL **WHENEVER NOT FOUND** **GO TO** CLOSE-THISEMP END-EXEC.	"ขั้นที่ 3: การระบุสิ่งที่ต้องทำเมื่อถึงจุดสิ้นสุดของข้อมูล" ในหน้า 258.
EXEC SQL **FETCH** THISEMP **INTO** :EMP-NUM, :NAME2, :DEPT, :JOB-CODE END-EXEC.	"ขั้นที่ 4: การดึงค่าแถวโดยใช้เคอร์เซอร์" ในหน้า 259.
... for all employees in department D11, update the JOB value: EXEC SQL **UPDATE** CORPDATA.EMPLOYEE **SET** JOB = :NEW-CODE **WHERE CURRENT OF** THISEMP END-EXEC. ... then print the row.	"ขั้นที่ 5a: การอัพเดตแถวปัจจุบัน" ในหน้า 260.

ตารางที่ 41. ตัวอย่างเคอร์เซอร์แบบอนุกรม (ต่อ)

คำสั่งเคอร์เซอร์ SQL แบบอนุกรม	อธิบายไว้ในส่วนนี้
... for other employees, delete the row: EXEC SQL **DELETE FROM** CORPDATA.EMPLOYEE **WHERE CURRENT OF** THISEMP END-EXEC.	"ขั้นที่ 5b: การลบแถวปัจจุบัน" ในหน้า 260.
Branch back to fetch and process the next row.	
CLOSE-THISEMP. EXEC SQL **CLOSE** THISEMP END-EXEC.	"ขั้นที่ 6: การปิดเคอร์เซอร์" ในหน้า 261.

สำหรับตัวอย่างเคอร์เซอร์แบบเลื่อนขึ้นลง, โปรแกรมจะใช้อ็อพชันตำแหน่ง RELATIVE เพื่อรับตัวอย่างเงินเดือนจากแผนก D11.

ตารางที่ 42. ตัวอย่างเคอร์เซอร์แบบเลื่อนขึ้นลง

คำสั่งเคอร์เซอร์ SQL แบบเลื่อนขึ้นลง	อธิบายไว้ในส่วนนี้
EXEC SQL **DECLARE** THISEMP **DYNAMIC SCROLL CURSOR FOR** **SELECT** EMPNO, LASTNAME, SALARY **FROM** CORPDATA.EMPLOYEE **WHERE** WORKDEPT = 'D11' END-EXEC.	"ขั้นที่ 1: การกำหนดเคอร์เซอร์" ในหน้า 257.
EXEC SQL **OPEN** THISEMP END-EXEC.	"ขั้นที่ 2: การเปิดเคอร์เซอร์" ในหน้า 258.
EXEC SQL **WHENEVER NOT FOUND** **GO TO** CLOSE-THISEMP END-EXEC.	"ขั้นที่ 3: การระบุสิ่งที่ต้องทำเมื่อถึงจุดสิ้นสุดของข้อมูล" ในหน้า 258.

ตารางที่ 42. ตัวอย่างเคอร์เซอร์แบบเลื่อนขึ้นลง (ต่อ)

คำสั่งเคอร์เซอร์ SQL แบบเลื่อนขึ้นลง	อธิบายไว้ในส่วนนี้
	"ขั้นที่ 4: การดึงค่าแถวโดยใช้เคอร์เซอร์" ในหน้า 259.
...initialize program summation salary variable EXEC SQL **FETCH RELATIVE** 3 **FROM** THISEMP **INTO** :EMP-NUM, :NAME2, :JOB-CODE END-EXEC. ...add the current salary to program summation salary ...branch back to fetch and process the next row.	
...calculate the average salary	
	"ขั้นที่ 6: การปิดเคอร์เซอร์" ในหน้า 261.
CLOSE-THISEMP. EXEC SQL **CLOSE** THISEMP END-EXEC.	

ขั้นที่ 1: การกำหนดเคอร์เซอร์:

ในการกำหนดเคอร์เซอร์เพื่อให้เข้าถึงตารางผลลัพธ์ให้ใช้คำสั่ง DECLARE CURSOR

ข้อความ DECLARE CURSOR จะตั้งชื่อเคอร์เซอร์และระบุข้อความที่เลือก. ข้อความที่เลือกจะกำหนดชุดแถวซึ่ง, ตามแนวคิดแล้ว, คือการสร้างตารางผลลัพธ์. สำหรับเคอร์เซอร์แบบอนุกรม, คำสั่งจะเป็นดังนี้ (FOR UPDATE OF clause สามารถเลือกได้):

```
EXEC SQL
 DECLARE cursor-name CURSOR FOR
 SELECT column-1, column-2 ,...
   FROM table-name , ...
   FOR UPDATE OF column-2 ,...
END-EXEC.
```

สำหรับเคอร์เซอร์แบบเลื่อนขึ้นลง, คำสั่งจะเป็นดังนี้ (WHERE clause สามารถเลือกได้):

```
EXEC SQL
 DECLARE cursor-name  SCROLL CURSOR FOR
 SELECT column-1, column-2 ,...
   FROM table-name ,...
   WHERE column-1 = expression ...
END-EXEC.
```

คำสั่ง select ซึ่งแสดงไว้ในที่นี้ค่อนข้างง่ายๆ. อย่างไรก็ตาม, คุณสามารถโค้ด clause ประเภทอื่นๆ ในข้อความที่เลือกไว้ภายในคำสั่ง DECLARE CURSOR สำหรับเคอร์เซอร์แบบอนุกรมและแบบเลื่อนขึ้นลง.

หากคุณประสงค์ที่จะอัพเดตคอลัมน์ใดๆ ในแถวใดๆ หรือทุกแถวของตารางที่ระบุ (ตารางจะถูกตั้งชื่อใน FROM clause), รวมทั้ง FOR UPDATE OF clause. ตารางจะตั้งชื่อแต่ละคอลัมน์ที่คุณต้องการอัพเดต. หากคุณไม่ได้ระบุชื่อคอลัมน์, และระบุ ORDER BY clause หรือ FOR READ ONLY clause, จะมีการส่งคืน SQLCODE แบบลบหากมีความพยายามในการอัพเดต . ถ้าคุณไม่ระบุ FOR UPDATE OF clause, FOR READ ONLY clause, ORDER BY clause, และตารางผลลัพธ์ที่ไม่ได้เป็นแบบอ่านอย่างเดียว และเคอร์เซอร์เป็นแบบที่เลื่อนไม่ได้, คุณสามารถอัพเดตคอลัมน์ใดๆ ก็ได้ในตารางที่ระบุ.

คุณสามารถอัพเดตคอลัมน์ตารางที่ระบุได้แม้ว่าคอลัมน์นั้นจะไม่ใช่ส่วนหนึ่งของตารางผลลัพธ์ก็ตาม. ในกรณีนี้, คุณไม่ต้องตั้งชื่อคอลัมน์ในข้อความ SELECT. เมื่อเคอร์เซอร์ค้นหาแถว (โดยใช้ FETCH) ซึ่งมีค่าคอลัมน์ที่คุณต้องการอัพเดต, คุณสามารถใช้ UPDATE ... WHERE CURRENT OF เพื่ออัพเดตแถว.

ตัวอย่างเช่น, สมมุติว่าตารางผลลัพธ์แต่ละแถวประกอบด้วยคอลัมน์ *EMPNO*, *LASTNAME*, และ *WORKDEPT* จากตาราง CORPDATA.EMPLOYEE. หากคุณต้องการอัพเดตคอลัมน์ *JOB* (หนึ่งในคอลัมน์ในแต่ละแถวของตาราง CORPDATA.EMPLOYEE), คำสั่ง DECLARE CURSOR ควรจะประกอบด้วย FOR UPDATE OF JOB ... แม้ว่า JOB จะถูกตัดออกจากคำสั่ง SELECT ก็ตาม.

สำหรับข้อมูลเพิ่มเติมเกี่ยวกับกรณีที่ตารางผลลัพธ์และ เคอร์เซอร์เป็นแบบอ่านอย่างเดียว โปรดดู DECLARE CURSOR ในกลุ่มหัวข้อการอ้างอิง SQL

ขั้นที่ 2: การเปิดเคอร์เซอร์:

เพื่อเริ่มการประมวลผลแถวของตารางผลลัพธ์, ให้ใส่คำสั่ง OPEN.

เมื่อโปรแกรมของคุณใส่คำสั่ง OPEN, SQL จะประมวลผลคำสั่งที่เลือกภายในคำสั่ง DECLARE CURSOR เพื่อระบุชุดแถว, เรียกตารางผลลัพธ์, โดยใช้ค่าปัจจุบันของตัวแปรโฮสต์ใดๆ ที่ระบุในคำสั่งที่เลือก. ตารางผลลัพธ์สามารถประกอบด้วยแถวศูนย์, หนึ่ง, หรือหลายแถว, ขึ้นอยู่กับขอบเขตความต้องการของเงื่อนไขการค้นหา. คำสั่ง OPEN จะเป็นดังนี้:

```
EXEC SQL
 OPEN cursor-name
END-EXEC.
```

ขั้นที่ 3: การระบุสิ่งที่ต้องทำเมื่อถึงจุดสิ้นสุดของข้อมูล:

เงื่อนไขจุดสิ้นสุดของข้อมูลเกิดขึ้นเมื่อคำสั่ง FETCH ทำการค้นหาแถวสุดท้ายในตารางผลลัพธ์ และโปรแกรมของคุณ ออกคำสั่ง FETCH

เพื่อค้นหาว่า ตารางผลลัพธ์จะสิ้นสุดเมื่อใด ให้ทดสอบฟิลด์ SQLCODE สำหรับค่า 100 หรือทดสอบฟิลด์ SQLSTATE สำหรับค่า '02000' (ซึ่งก็คือ จุดสิ้นสุดของข้อมูล)

ตัวอย่างเช่น:

```
...
IF SQLCODE =100 GO TO DATA-NOT-FOUND.
```

หรือ

```
IF SQLSTATE ='02000' GO TO DATA-NOT-FOUND.
```

อีกทางเลือกหนึ่งของเทคนิคนี้คือให้โค้ดคำสั่ง WHENEVER. การใช้ WHENEVER NOT FOUND ทำให้เกิดการแยกสาขาไปยังอีกส่วนหนึ่งของโปรแกรม, ซึ่งมีการใส่ข้อความ CLOSE. คำสั่ง WHENEVER จะเป็นดังนี้:

```
EXEC SQL
 WHENEVER NOT FOUND GO TO symbolic-address
END-EXEC.
```

โปรแกรมของคุณควรคาดการณ์ถึงเงื่อนไขการสิ้นสุดของข้อมูลเมื่อใดก็ตามที่เคอร์เซอร์ถูกใช้เพื่อดึงข้อมูลแถวออกมา, และควรเตรียมการเพื่อจัดการเมื่อเกิดสถานการณ์เช่นนี้ขึ้น.

เมื่อคุณกำลังใช้เคอร์เซอร์แบบอนุกรม และถึงจุดสิ้นสุดข้อมูล ทุกๆ คำสั่ง FETCH ที่ตามมาจะย้อนกลับไปที่เงื่อนไขการสิ้นสุดข้อมูล คุณไม่สามารถกำหนดตำแหน่งเคอร์เซอร์บนแถวซึ่งถูกประมวลผลเรียบร้อยแล้ว. คำสั่ง CLOSE คือการดำเนินการเพียงอย่างเดียวที่สามารถทำงานบนเคอร์เซอร์ได้.

เมื่อคุณกำลังใช้เคอร์เซอร์แบบเลื่อนขึ้นลง และถึงจุดสิ้นสุดข้อมูล ตารางผลลัพธ์ยังสามารถประมวลผลข้อมูลเพิ่มเติมได้อีก คุณสามารถกำหนดตำแหน่งเคอร์เซอร์ได้ทุกที่ในตารางผลลัพธ์ด้วยการใช้การผสมของอ็อพชันตำแหน่ง. คุณไม่จำเป็นต้องปิดเคอร์เซอร์เมื่อสิ้นสุดข้อมูล

ขั้นที่ 4: การดึงค่าแถวโดยใช้เคอร์เซอร์:

ในการย้ายเนื้อหาของแถวที่เลือกเข้าไว้ในตัวแปรโฮสต์ของโปรแกรม ให้ใช้คำสั่ง FETCH

คำสั่ง SELECT ภายในคำสั่ง DECLARE CURSOR เป็นการจำแนกแถวที่ประกอบด้วยค่าคอลัมน์ที่โปรแกรมต้องการ. อย่างไรก็ตาม, SQL จะไม่นำข้อมูลใดๆ ออกมาสำหรับแอ็พพลิเคชันโปรแกรมจนกว่าจะมีการใส่คำสั่ง FETCH.

เมื่อโปรแกรมของคุณใส่คำสั่ง FETCH, SQL จะใช้ตำแหน่งเคอร์เซอร์ปัจจุบันเป็นจุดเริ่มต้นเพื่อหาตำแหน่งแถวที่ร้องขอในตารางผลลัพธ์. ซึ่งจะเป็นการเปลี่ยนแถวดังกล่าวให้เป็น **แถวปัจจุบัน**. หากมีการระบุ INTO clause, SQL จะย้ายเนื้อหาของแถวปัจจุบันเข้าไว้ในตัวแปรโฮสต์ของโปรแกรม. จะมีการซ้ำลำดับนี้ทุกครั้งที่มีการใส่คำสั่ง FETCH.

SQL จะคงไว้ซึ่งตำแหน่งของแถวปัจจุบัน (กล่าวคือ, เคอร์เซอร์จะชี้ไปที่แถวปัจจุบัน) จนกว่าจะมีการใส่คำสั่ง FETCH ถัดไปเพื่อให้มีการออกเคอร์เซอร์. คำสั่ง UPDATE ไม่ได้เป็นการเปลี่ยนตำแหน่งของแถวปัจจุบันภายในตารางผลลัพธ์, แม้ว่าคำสั่ง DELETE ก่อให้เกิดการเปลี่ยนตำแหน่งก็ตาม.

คำสั่ง FETCH สำหรับเคอร์เซอร์แบบอนุกรมจะเป็นดังนี้:

```
EXEC SQL
 FETCH cursor-name
   INTO :host variable-1[, :host variable-2] ...
END-EXEC.
```

คำสั่ง FETCH สำหรับเคอร์เซอร์แบบเลื่อนขึ้นลงจะเป็นดังนี้:

```
EXEC SQL
 FETCH RELATIVE integer
   FROM cursor-name
   INTO :host variable-1[, :host variable-2] ...
END-EXEC.
```

ขั้นที่ 5a: การอัพเดตแถวปัจจุบัน:

เมื่อโปรแกรมของคุณกำหนดตำแหน่งเคอร์เซอร์ ที่อยู่บนแถว คุณสามารถอัพเดตข้อมูลได้ด้วยการใช้ข้อความ UPDATE ที่มี WHERE CURRENT OF clause ทั้งนี้ WHERE CURRENT OF clause จะระบุ เคอร์เซอร์ที่ชี้ไปยังแถวที่คุณต้องการอัพเดต

คำสั่ง UPDATE ... WHERE CURRENT OF จะเป็นดังนี้:

```
EXEC SQL
 UPDATE table-name
   SET column-1 = value [, column-2 = value] ...
   WHERE CURRENT OF cursor-name
END-EXEC.
```

เมื่อใช้ร่วมกับเคอร์เซอร์, คำสั่ง UPDATE:

- จะอัพเดตเฉพาะแถวเดียว—แถวปัจจุบัน
- จะระบุเคอร์เซอร์ซึ่งชี้ไปที่แถวที่จะอัพเดต
- ต้องการให้คอลัมน์ที่ถูกอัพเดตมีการตั้งชื่อก่อนหน้านี้ใน FOR UPDATE OF clause ของข้อความ DECLARE CURSOR, หากมีการระบุ ORDER BY ด้วยเช่นกัน

หลังจากอัพเดตแถวแล้ว, ตำแหน่งของเคอร์เซอร์จะยังคงอยู่บนแถวนั้น (กล่าวคือ, แถวปัจจุบันของเคอร์เซอร์ไม่ได้เปลี่ยนไป) จนกว่าคุณจะใช้คำสั่ง FETCH สำหรับแถวถัดไป.

ขั้นที่ 5b: การลบแถวปัจจุบัน:

เมื่อโปรแกรมของคุณเรียกแถวปัจจุบันออกมา คุณสามารถลบแถวโดยการใช้คำสั่ง DELETE พร้อมกับประโยค WHERE CURRENT OF. ประโยค WHERE CURRENT OF จะระบุเคอร์เซอร์ที่ชี้ไปยังแถวที่คุณต้องการลบ

คำสั่ง DELETE ... WHERE CURRENT OF จะเป็นดังนี้:

```
EXEC SQL
 DELETE FROM table-name
   WHERE CURRENT OF cursor-name
END-EXEC.
```

เมื่อใช้ร่วมกับเคอร์เซอร์, คำสั่ง DELETE:

- จะลบเฉพาะแถวเดียว—แถวปัจจุบัน
- จะใช้ WHERE CURRENT OF clause เพื่อระบุเคอร์เซอร์ซึ่งชี้ไปที่แถวที่ต้องถูกลบออก

หลังจากลบแถวแล้ว, คุณไม่สามารถอัพเดตหรือลบอีกแถวหนึ่งได้โดยใช้เคอร์เซอร์ดังกล่าวจนกว่าคุณจะใส่ข้อความ FETCH เพื่อกำหนดตำแหน่งเคอร์เซอร์.

คุณสามารถใช้คำสั่ง DELETE เพื่อลบแถวทั้งหมดที่ตรงตามเงื่อนไขการค้นหาเฉพาะ คุณสามารถใช้คำสั่ง FETCH และ DELETE ... WHERE CURRENT OF เมื่อคุณต้องการรับสำเนาของแถว, ตรวจสอบ แล้วลบแถวออก

ขั้นที่ 6: การปิดเคอร์เซอร์:

หากคุณประมวลผลแถวของตารางผลลัพธ์โดยใช้เคอร์เซอร์แบบอนุกรม และคุณต้องการใช้เคอร์เซอร์อีกครั้ง ให้ใส่คำสั่ง CLOSE เพื่อปิดเคอร์เซอร์ก่อนที่จะเปิดใหม่

คำสั่งจะเป็นดังนี้:

```
EXEC SQL
 CLOSE cursor-name
END-EXEC.
```

หากคุณประมวลผลแถวของตารางผลลัพธ์ และไม่ต้องการใช้เคอร์เซอร์นั้นอีก, คุณสามารถปล่อยให้ระบบปิดเคอร์เซอร์เอง. ระบบจะปิดเคอร์เซอร์โดยอัตโนมัติเมื่อ:

- มีการใส่ COMMIT โดยไม่มีข้อความ HOLD และไม่ได้ประกาศเคอร์เซอร์ด้วยการใช้ WITH HOLD clause.
- มีการใส่ ROLLBACK โดยไม่มีข้อความ HOLD.
- สิ้นสุดงาน.
- สิ้นสุด activation group ends และมีการระบุ CLOSQLCSR(*ENDACTGRP) บนพรีคอมไพล์.
- โปรแกรม SQL แรกใน call stack สิ้นสุดลงและไม่มีการระบุทั้ง CLOSQLCSR(*ENDJOB) หรือ CLOSQLCSR (*ENDACTGRP) เมื่อโปรแกรมถูกคอมไพล์ล่วงหน้า.
- การเชื่อมต่อไปยังแอ็พพลิเคชันเซิร์ฟเวอร์ถูกทำให้สิ้นสุดลงด้วยการใช้ข้อความ DISCONNECT.
- การเชื่อมต่อไปยังแอ็พพลิเคชันเซิร์ฟเวอร์ถูกรีลีสและเกิด COMMIT ที่สมบูรณ์.
- เกิด *RUW CONNECT.

  เนื่องจากเคอร์เซอร์แบบเปิดยังคงรักษาการล็อกบนการอ้างอิงถึงตารางหรือบนมุมมอง, คุณจึงควรปิดเคอร์เซอร์แบบเปิดใดๆ อย่างชัดเจนทันทีที่ไม่จำเป็นต้องใช้งานแล้ว.

การใช้คำสั่ง FETCH แบบหลายแถว

คำสั่ง FETCH แบบหลายแถวสามารถนำมาใช้เพื่อเรียกแถวจำนวนมากออกมาจากตาราง หรือดูคำสั่ง FETCH เดียว โปรแกรมจะควบคุมการบล็อกแถวตามจำนวนแถวที่ร้องขอบนคำสั่ง FETCH (คำสั่ง Override Database File (OVRDBF) ไม่มีผลกระทบ).

จำนวนแถวสูงสุดที่สามารถร้องขอบนการเรียกดึงข้อมูลออกแบบเดี่ยวคือ 32 767 หลังจากที่นำข้อมูลออกมาแล้ว เคอร์เซอร์จะถูกวางตำแหน่งบนแถวสุดท้ายที่ถูกเรียกออกมา

มีสองวิธีการในการกำหนดหน่วยเก็บในที่ซึ่งแถวที่ถูกดึงข้อมูลออกมาตั้งอยู่: อะเรย์โครงสร้างโฮสต์หรือพื้นที่หน่วยเก็บของแถวซึ่งมี descriptor ที่เกี่ยวข้อง. สามารถโค้ดได้ทั้งสองวิธีในทุกภาษาที่พรีคอมไพล์ของ SQL สนับสนุนอยู่ ยกเว้นอะเรย์โครงสร้างโฮสต์ใน REXX รูปแบบทั้งสองของคำสั่ง FETCH แบบหลายแถวยอมให้แอ็พพลิเคชันโค้ดอะเรย์ตัวบ่งชี้ที่แยกต่างหาก. อะเรย์ตัวบ่งชี้ควรมีตัวบ่งชี้หนึ่งตัว indicator สำหรับแต่ละตัวแปรโฮสต์ซึ่งเป็น null.

คำสั่ง FETCH แบบหลายแถวสามารถนำมาใช้ร่วมกับทั้งเคอร์เซอร์แบบอนุกรมและแบบเลื่อนขึ้นลง. การดำเนินการที่ถูกใช้เพื่อกำหนด, เปิด, และปิดเคอร์เซอร์สำหรับคำสั่ง FETCH แบบหลายแถวยังคงเหมือนเดิม. เฉพาะคำสั่ง FETCH เท่านั้นที่เปลี่ยนเพื่อระบุจำนวนแถวที่จะเรียกออกมาและจำนวนหน่วยเก็บที่ซึ่งมีแถวตั้งอยู่.

หลังคำสั่ง FETCH แบบหลายแถวแต่ละข้อความ, ข้อมูลจะถูกส่งคืนไปยังโปรแกรมผ่าน SQLCA. นอกเหนือจากฟิลด์ SQLCODE และ SQLSTATE แล้ว, SQLERRD จะให้ข้อมูลต่อไปนี้:

- SQLERRD3 ประกอบด้วยจำนวนแถวที่ถูกเรียกออกมาบนข้อความ FETCH แบบหลายแถว. หาก SQLERRD3 น้อยกว่าจำนวนแถวที่ร้องขอ, จะเกิดข้อผิดพลาดหรือเงื่อนไขการสิ้นสุดข้อมูลขึ้น.
- SQLERRD4 ประกอบด้วยความยาวของแต่ละแถวที่ถูกเรียกออกมา.
- SQLERRD5 มีการบ่งชี้ที่ว่าแถวสุดท้ายในตารางถูกดึงข้อมูลออกมา. ซึ่งสามารถนำไปใช้เพื่อตรวจพบเงื่อนไขการสิ้นสุดข้อมูลในตารางที่ถูกดึงข้อมูลออกมาเมื่อเคอร์เซอร์ไม่มีความสามารถในการรับรู้ทันทีเพื่ออัพเดต. เคอร์เซอร์ซึ่งมีความสามารถทันทีในการอัพเดตควรดึงข้อมูลออกอย่างต่อเนื่องจนกว่าจะได้รับ SQLCODE +100 เพื่อตรวจพบเงื่อนไขการสิ้นสุดข้อมูล.

หลักการที่เกี่ยวข้อง

Embedded SQL programming

การดึงข้อมูลออก (FETCH) แบบหลายแถวโดยใช้อะเรย์โครงสร้างโฮสต์:

การใช้ FETCH แบบหลายแถวที่มีอะเรย์โครงสร้างโฮสต์ แอ็พพลิเคชันต้องกำหนดอะเรย์โครงสร้างโฮสต์ที่ SQL สามารถใช้ได้

แต่ละภาษาจะมีระเบียบและกฎของตัวมันเองสำหรับการกำหนดอะเรย์โครงสร้างโฮสต์. สามารถกำหนดอะเรย์โครงสร้างโฮสต์ได้ด้วยการใช้การประกาศที่เปลี่ยนแปลงได้หรือใช้คำสั่งคอมไพเลอร์เพื่อเรียก External File Descriptions ออกมา (อย่างเช่นคำสั่ง COBOL COPY).

อะเรย์โครงสร้างโฮสต์ประกอบด้วยอะเรย์ของโครงสร้าง. แต่ละโครงสร้างจะตรงกับแถวหนึ่งแถวของตารางผลลัพธ์. โครงสร้างแรกในอะเรย์จะตรงกับแถวแรก, โครงสร้างที่สองในอะเรย์จะตรงกับแถวที่สอง, เป็นอาทิ. SQL จะพิจารณาแอ็ตทริบิวต์ของไอเท็มชั้นต้นในอะเรย์โครงสร้างโฮสต์โดยยึดตามการประกาศอะเรย์โครงสร้างโฮสต์. เพื่อประสิทธิภาพการทำงานสูงสุด, แอ็ตทริบิวต์ของไอเท็มซึ่งสร้างอะเรย์โครงสร้างโฮสต์ควรตรงกับแอ็ตทริบิวต์ของคอลัมน์ที่ถูกเรียกออกมา.

พิจารณาตัวอย่าง COBOL ต่อไปนี้:

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

```
     EXEC SQL INCLUDE SQLCA
     END-EXEC.

...

     01 TABLE-1.
         02 DEPT  OCCURS 10 TIMES.
            05  EMPNO   PIC   X(6).
            05 LASTNAME.
               49 LASTNAME-LEN  PIC S9(4) BINARY.
               49 LASTNAME-TEXT PIC X(15).
            05  WORKDEPT PIC  X(3).
            05  JOB      PIC  X(8).
     01 TABLE-2.
        02 IND-ARRAY   OCCURS 10 TIMES.
           05 INDS PIC  S9(4) BINARY OCCURS 4 TIMES.
```

```
...
      EXEC SQL
      DECLARE D11 CURSOR FOR
      SELECT EMPNO, LASTNAME, WORKDEPT, JOB
       FROM CORPDATA.EMPLOYEE
        WHERE WORKDEPT = "D11"
      END-EXEC.

...

      EXEC SQL
       OPEN D11
      END-EXEC.
      PERFORM FETCH-PARA UNTIL SQLCODE NOT EQUAL TO ZERO.
  ALL-DONE.
        EXEC SQL CLOSE D11 END-EXEC.

...

      FETCH-PARA.
         EXEC SQL WHENEVER NOT FOUND GO TO ALL-DONE END-EXEC.
      EXEC SQL FETCH D11 FOR 10 ROWS INTO :DEPT :IND-ARRAY
      END-EXEC.

...
```

ในตัวอย่างนี้, มีการกำหนดเคอร์เซอร์สำหรับตาราง CORPDATA.EMPLOYEE เพื่อเลือกแถวทั้งหมดที่คอลัมน์ WORKDEPT มีค่าเท่ากับ 'D11'. ตารางผลลัพธ์ประกอบด้วยแปดแถว. คำสั่ง DECLARE CURSOR และ OPEN ไม่มีซินแทกซ์พิเศษใดๆ เมื่อถูกใช้ร่วมกับคำสั่ง FETCH แบบหลายแถว. คำสั่ง FETCH อีกคำสั่งหนึ่งซึ่งส่งคืนแถวเดี่ยวมาให้กับเคอร์เซอร์เดียวกันสามารถโค้ดได้ที่จุดอื่นในโปรแกรม. คำสั่ง FETCH แบบหลายแถวถูกนำมาใช้เพื่อเรียกแถวทั้งหมดในตารางผลลัพธ์ออกมา. หลัง FETCH, ตำแหน่งเคอร์เซอร์ยังคงอยู่ที่แถวสุดท้ายซึ่งถูกเรียกออกมา.

มีการกำหนดอะเรย์โครงสร้างโฮสต์ DEPT และอะเรย์ตัวบ่งชี้ IND-ARRAY ที่เกี่ยวข้องในแอ็พพลิเคชัน. ทั้งสองอะเรย์มีสิบด้าน. อะเรย์ตัวบ่งชี้มี entry สำหรับแต่ละคอลัมน์ในตารางผลลัพธ์.

แอ็ตทริบิวต์ของประเภทและความยาวของไอเท็มชั้นต้นของอะเรย์โครงสร้างโฮสต์ DEPT ตรงกับ คอลัมน์ซึ่งถูกเรียกออกมา.

เมื่อคำสั่ง FETCH แบบหลายแถวเสร็จสมบูรณ์, อะเรย์โครงสร้างโฮสต์ จะประกอบด้วยข้อมูลสำหรับแถวทั้งหมดแปดแถว. อะเรย์ตัวบ่งชี้, IND_ARRAY, ประกอบด้วยค่าศูนย์สำหรับทุกคอลัมน์ในแถวทุกแถวเนื่องจากไม่มีการส่งคืนค่าศูนย์.

SQLCA ซึ่งถูกส่งคืนมายังแอ็พพลิเคชันประกอบด้วยข้อมูลต่อไปนี้:

- SQLCODE ประกอบด้วย 0
- SQLSTATE ประกอบด้วย '00000'
- SQLERRD3 ประกอบด้วย 8, ซึ่งคือจำนวนแถวที่ถูกดึงข้อมูลออกมา
- SQLERRD4 ประกอบด้วย 34, ซึ่งคือความยาวของแต่ละแถว
- SQLERRD5 ประกอบด้วย +100, แสดงว่าแถวสุดท้ายในตารางผลลัพธ์อยู่ในบล็อก

สิ่งอ้างอิงที่เกี่ยวข้อง

SQLCA (SQL communication area)

การดึงข้อมูลออก (FETCH) แบบหลายแถวโดยใช้พื้นที่หน่วยเก็บของแถว:

ก่อนที่จะใช้คำสั่ง FETCH แบบหลายแถวที่มี พื้นที่หน่วยเก็บของแถว แอ็พพลิเคชันจะต้องกำหนดพื้นที่หน่วยเก็บของแถว และ พื้นที่รายละเอียดที่เกี่ยวข้อง

พื้นที่หน่วยเก็บของแถว คือตัวแปรโฮสต์ซึ่งถูกกำหนดใน แอ็พพลิเคชัน พื้นที่หน่วยเก็บของแถวประกอบด้วยผลลัพธ์ของ คำสั่ง FETCH แบบหลายแถว พื้นที่หน่วยเก็บของแถวสามารถเป็นตัวแปรอักขระที่มีไบต์ มากพอต่อการรักษาแถวที่ร้องขอทั้งหมดไว้บนคำสั่ง FETCH แบบ หลายแถว

SQLDA ซึ่งประกอบด้วย SQLTYPE และ SQLLEN สำหรับแต่ละคอลัมน์ที่ถูกส่งคืนมาถูกกำหนดโดย descriptor ที่เกี่ยวข้อง ซึ่งถูกใช้บนรูปแบบพื้นที่หน่วยเก็บของแถวของ FETCH แบบหลายแถว. ข้อมูลที่ให้ไว้ใน descriptor จะกำหนดข้อมูลที่แม็พจากฐานข้อมูลไปยังพื้นที่หน่วยเก็บของแถว. เพื่อประสิทธิภาพการทำงานสูงสุด, ข้อมูลแอ็ตทริบิวต์ใน descriptor ควรตรงกับแอ็ตทริบิวต์ของคอลัมน์ที่ถูกเรียกออกมา.

พิจารณาตัวอย่าง PL/I ต่อไปนี้:

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353.

```
      *....+....1....+....2....+....3....+....4....+....5....+....6....+....7...*
     EXEC SQL INCLUDE SQLCA;
     EXEC SQL INCLUDE SQLDA;

...

    DCL DEPTPTR PTR;
    DCL 1 DEPT(20) BASED(DEPTPTR),
        3 EMPNO CHAR(6),
        3 LASTNAME CHAR(15) VARYING,
        3 WORKDEPT CHAR(3),
        3 JOB CHAR(8);
    DCL I BIN(31) FIXED;
    DEC J BIN(31) FIXED;
    DCL ROWAREA CHAR(2000);

...

    ALLOCATE SQLDA SET(SQLDAPTR);
    EXEC SQL
     DECLARE D11 CURSOR FOR
     SELECT EMPNO, LASTNAME, WORKDEPT, JOB
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11';
...

    EXEC SQL
      OPEN D11;
    /* SET UP THE DESCRIPTOR FOR THE MULTIPLE-ROW FETCH */
    /* 4 COLUMNS ARE BEING FETCHED                */
    SQLD = 4;
    SQLN = 4;
    SQLDABC = 366;
    SQLTYPE(1) = 452; /* FIXED LENGTH CHARACTER -  */
```

```
                  /* NOT NULLABLE             */
  SQLLEN(1) = 6;
  SQLTYPE(2) = 456; /*VARYING LENGTH CHARACTER   */
                  /* NOT NULLABLE             */
  SQLLEN(2) = 15;
  SQLTYPE(3) = 452; /* FIXED LENGTH CHARACTER -  */
  SQLLEN(3) = 3;
  SQLTYPE(4) = 452; /* FIXED LENGTH CHARACTER -  */
                  /* NOT NULLABLE             */
  SQLLEN(4) = 8;
  /*ISSUE THE MULTIPLE-ROW FETCH STATEMENT TO RETRIEVE*/
  /*THE DATA INTO THE DEPT ROW STORAGE AREA      */
  /*USE A HOST VARIABLE TO CONTAIN THE COUNT OF  */
  /*ROWS TO BE RETURNED ON THE MULTIPLE-ROW FETCH     */

  J = 20;        /*REQUESTS 20 ROWS ON THE FETCH */
...
  EXEC SQL
    WHENEVER NOT FOUND
    GOTO FINISHED;
  EXEC SQL
    WHENEVER SQLERROR
    GOTO FINISHED;
  EXEC SQL
    FETCH D11 FOR :J ROWS
    USING DESCRIPTOR :SQLDA INTO :ROWAREA;
  /* ADDRESS THE ROWS RETURNED                  */
  DEPTPTR = ADDR(ROWAREA);
  /*PROCESS EACH ROW RETURNED IN THE ROW STORAGE */
  /*AREA BASED ON THE COUNT OF RECORDS RETURNED  */
  /*IN SQLERRD3.                                 */
  DO I = 1 TO SQLERRD(3);
   IF EMPNO(I) = '000170'  THEN
     DO;
    :
     END;
  END;
  IF SQLERRD(5) = 100 THEN
    DO;
      /* PROCESS END OF FILE  */
    END;
  FINISHED:
```

ในตัวอย่างนี้, มีการกำหนดเคอร์เซอร์สำหรับตาราง CORPDATA.EMPLOYEE เพื่อเลือกแถวทั้งหมดที่คอลัมน์ WORKDEPT มีค่าเท่ากับ 'D11'. ตาราง EMPLOYEE ตัวอย่างใน Sample Tables แสดงตารางผลลัพธ์ที่มีหลายแถว. ข้อความ DECLARE CURSOR และ OPEN ไม่มีซินแทกซ์พิเศษเมื่อถูกใช้ร่วมกับข้อความ FETCH แบบหลายแถว. คำสั่ง FETCH อีกคำสั่งหนึ่งซึ่งส่งคืนแถวเดี่ยวมาให้กับเคอร์เซอร์เดียวกันสามารถโค้ดได้ที่จุดอื่นในโปรแกรม. คำสั่ง FETCH แบบหลายแถวถูกนำมาใช้เพื่อเรียกแถวทั้งหมดในตารางผลลัพธ์ออกมา. หลัง FETCH, ตำแหน่งเคอร์เซอร์ยังคงอยู่บนแถวสุดท้ายในบล็อก.

พื้นที่แถว, ROWAREA, ถูกกำหนดให้เป็นอะเรย์อักขระ. ข้อมูลจากตารางผลลัพธ์ถูกใส่ไว้ในตัวแปรโฮสต์. ในตัวอย่างนี้, ตัวแปรตัวชี้ (pointer) จะถูกกำหนดให้กับแอดเดรสของ ROWAREA. แต่ละไอเท็มในแถวซึ่งถูกส่งคืนจะถูกตรวจสอบและถูกใช้ด้วยโครงสร้าง DEPT พื้นฐาน.

แอ็ตทริบิวต์ (ประเภทและความยาว) ของไอเท็มใน descriptor ตรงกับคอลัมน์ที่ถูกเรียกออกมา. ในกรณีนี้, ไม่มีการให้พื้นที่ตัวบ่งชี้ใดๆ.

หลังทำคำสั่ง FETCH เสร็จสิ้นแล้ว, ROWAREA จะประกอบด้วยแถวทั้งหมดที่มีค่าเท่ากับ 'D11', ในกรณีนี้จะมี 11 แถว. SQLCA ที่ถูกส่งคืนมายังแอ็พพลิเคชันประกอบด้วยข้อมูลต่อไปนี้:

- SQLCODE ประกอบด้วย 0
- SQLSTATE ประกอบด้วย '00000'
- SQLERRD3 ประกอบด้วย 11, ซึ่งคือจำนวนแถวที่ถูกส่งคืน
- SQLERRD4 ประกอบด้วย 34, สำหรับความยาวของแถวที่ถูกดึงข้อมูลออกมา
- SQLERRD5 ประกอบด้วย +100, แสดงว่าแถวสุดท้ายในตารางผลลัพธ์ถูกดึงข้อมูลออกมา

ในตัวอย่างนี้, แอ็พพลิเคชันได้ประโยชน์จากข้อเท็จจริงที่ว่า SQLERRD5 มีการบ่งชี้ว่าสิ้นสุดไฟล์แล้ว. ผลที่ได้คือ, แอ็พพลิเคชันไม่ต้องเรียก SQL อีกครั้งเพื่อพยายามเรียกแถวออกมามากขึ้น. หากเคอร์เซอร์มีความสามารถในการรับรู้ทันทีต่อการแทรก, คุณควรเรียก SQL ในกรณีที่มีการใส่เพิ่มเร็กคอร์ดใดๆ. เคอร์เซอร์มีความสามารถในการรับรู้ทันทีเมื่อระดับ commitment control เป็นระดับอื่นที่ไม่ใช่ *RR.

สิ่งอ้างอิงที่เกี่ยวข้อง

"DB2 for i5/OS ตารางตัวอย่าง" ในหน้า 329
ตารางตัวอย่างต่อไปนี้ถูกอ้างอิงและใช้งานใน หัวข้อการโปรแกรม SQL และการอ้างอิง SQL

SQLDA (SQL descriptor area)

ยูนิตงานและเคอร์เซอร์ที่เปิดอยู่

เมื่อโปรแกรมของคุณทำยูนิตงานเสร็จสมบูรณ์แล้ว โปรแกรมควร commit หรือ roll back การเปลี่ยนแปลงที่คุณได้ทำไว้

เว้นแต่คุณจะระบุ HOLD บนคำสั่ง COMMIT หรือ ROLLBACK, เคอร์เซอร์ทั้งหมดที่เปิดอยู่จะถูกปิดอัตโนมัติโดย SQL. เคอร์เซอร์ที่ถูกประกาศด้วย WITH HOLD clause จะไม่ถูกปิดโดยอัตโนมัติบน COMMIT. แต่จะถูกปิดโดยอัตโนมัติบน ROLLBACK (WITH HOLD clause ซึ่งถูกระบุบนข้อความ DECLARE CURSOR จะถูกข้ามไป).

หากคุณต้องการประมวลผลต่อจากตำแหน่งเคอร์เซอร์ปัจจุบันหลัง COMMIT หรือ ROLLBACK, คุณต้องระบุ COMMIT HOLD หรือ ROLLBACK HOLD. เมื่อมีการระบุ HOLD, เคอร์เซอร์ใดๆ ที่เปิดอยู่จะถูกเปิดค้างไว้และรักษาตำแหน่งเคอร์เซอร์ไว้เพื่อที่จะเริ่มต้นการประมวลผลใหม่. บนข้อความ COMMIT, ตำแหน่งเคอร์เซอร์จะถูกคงไว้. บนคำสั่ง ROLLBACK, ตำแหน่งเคอร์เซอร์จะถูกเรียกคืนไปไว้ที่ข้างหลังของแถวสุดท้ายที่ถูกเรียกออกมาจากยูนิตงานก่อนหน้านี้. ล็อกของเร็กคอร์ดทั้งหมดยังคงถูกรีลีส.

หลังจากใส่คำสั่ง COMMIT หรือ ROLLBACK โดยไม่มี HOLD, ล็อกทั้งหมดจะถูกรีลีสและเคอร์เซอร์ทั้งหมดจะถูกปิด. คุณสามารถเปิดเคอร์เซอร์ได้อีกครั้ง, แต่คุณจะต้องเริ่มการประมวลผลที่แถวแรกของตารางผลลัพธ์.

หมายเหตุ: ค่ากำหนดของพารามิเตอร์ ALWBLK(*ALLREAD) ของคำสั่ง Create SQL (CRTSQL*xxx*) สามารถเปลี่ยนกลับสู่สภาพเดิมของ ตำแหน่งเคอร์เซอร์สำหรับเคอร์เซอร์แบบอ่านอย่างเดียว สำหรับข้อมูลเกี่ยวกับการใช้ พารามิเตอร์ ALWBLK และระดับการทำงานอื่นๆ ที่เกี่ยวข้องกับอ็อพชันบนคำสั่ง CRTSQL*xxx* โปรดดู "แอ็พพลิเคชัน Dynamic SQL" ในหน้า 267

หลักการที่เกี่ยวข้อง

Commitment control

แอ็พพลิเคชัน Dynamic SQL

Dynamic SQL อนุญาตให้แอ็พพลิเคชันกำหนด และรันคำสั่ง SQL ที่เวลารันไทม์ของโปรแกรม แอ็พพลิเคชันที่ใช้ dynamic SQL จะยอมรับคำสั่ง SQL เป็นอินพุต หรือสร้างคำสั่ง SQL ในรูปของสตริงอักขระ แอ็พพลิเคชันไม่จำเป็นต้องทราบชนิดของคำสั่ง SQL

แอ็พพลิเคชัน:

- สร้าง หรือ รับเป็นอินพุตของคำสั่ง SQL
- เตรียมคำสั่ง SQL สำหรับการรัน
- ทำการรันคำสั่ง
- จัดการกับค่าส่งคืนของ SQL

หมายเหตุ:

- การประมวลผล SQL แบบ dynamic จะมีค่าใช้จ่ายสูงกว่าการประมวลผล SQL แบบ static เนื่องจากคำสั่งอาจต้องใช้การประมวลผลแบบเต็ม ณ เวลารันไทม์ในกรณีที่เลวร้ายที่สุด, คำสั่งต้องถูกเตรียม, ถูกผูก, และถูก optimize แบบเต็มที่โดยฐานข้อมูลก่อนที่จะรัน. ในกรณีอื่นๆ, ถ้ารันคำสั่งก่อน, ส่วนของการประมวลผลจะสามารถข้ามได้ เนื่องจาก algorithm ที่ถูกใช้ และแคชที่เก็บไว้โดยฐานข้อมูล. คุณลักษณะเหล่านี้อนุญาตให้ฐานข้อมูล DB2 for i5/OS เตรียม ประสิทธิภาพในการทำงานที่ดีสำหรับคำสั่ง SQL แบบ dynamic ถ้าประสิทธิภาพในการทำงานของแอ็พพลิเคชันแบบ dynamic ของคุณอยู่ในขั้นเลวร้าย ให้พิจารณาถึงการใช้ความสามารถในการทำงานที่ถูกขยายแบบ dynamic โดยใช้ Process Extended Dynamic SQL (QSQPRCED) API คุณลักษณะพิเศษนี้อนุญาตให้แอ็พพลิเคชันคงไว้ซึ่งแคชของคำสั่ง SQL และลดค่าใช้จ่ายในช่วงรันไทม์ เมื่อรันแอ็พพลิเคชัน
- โปรแกรมที่มี EXECUTE หรือ EXECUTE IMMEDIATE statement อยู่และใช้ FOR READ ONLY clause ในการทำให้เคอร์เซอร์สำหรับอ่านอย่างเดียวมีการทำงานที่ดีขึ้น เนื่องจากมีการใช้การจัดเป็นกลุ่มบล็อกเพื่อทำการเรียกแถวข้มูลออกมาสำหรับเคอร์เซอร์.

  ตัวเลือก ALWBLK(*ALLREAD) CRTSQLxxx จะแสดงความหมายโดยนัยของการประกาศ FOR READ ONLY สำหรับเคอร์เซอร์ ทั้งหมดที่ไม่ได้แสดงโค้ดไว้อย่างชัดเจนว่า FOR UPDATE OF หรือมีการระบุการลบออกหรือการอัพเดต ที่อ้างถึงในเคอร์เซอร์. เคอร์เซอร์ที่มีการประกาศโดยนัยว่า FOR READ ONLY จะได้รับผลประโยชน์จากรายการที่สองในรายชื่อนี้.

คำสั่ง SQL แบบ dynamic บางคำสั่ง จำเป็นต้องใช้ตัวแปรที่เป็นตัวชี้ โปรแกรม RPG/400 ต้องการ ความช่วยเหลือของ PL/I, COBOL, C หรือโปรแกรม ILE RPG เพื่อจัดการกับตัวแปร ที่เป็นตัวชี้

หลักการที่เกี่ยวข้อง

"การใช้ SQL แบบโต้ตอบ" ในหน้า 285

SQL แบบโต้ตอบอนุญาตให้โปรแกรมเมอร์ หรือผู้ดูแลฐานข้อมูล มีความรวดเร็วและสะดวกในการกำหนด, อัพเดต, ลบ, หรือสำรวจข้อมูลเพื่อทำการทดสอบ วิเคราะห์ปัญหา และการดูแลฐานข้อมูล

สิ่งอ้างอิงที่เกี่ยวข้อง

Action ที่อนุญาตให้ใช้บนคำสั่ง SQL

Process Extended Dynamic SQL (QSQPRCED) API

"ยูนิตงานและเคอร์เซอร์ที่เปิดอยู่" ในหน้า 266

เมื่อโปรแกรมของคุณทำยูนิตงานเสร็จสมบูรณ์แล้ว โปรแกรมควร commit หรือ roll back การเปลี่ยนแปลงที่คุณได้ทำไว้

การออกแบบและการรันแอ็พพลิเคชัน SQL แบบไดนามิก

เพื่อที่จะกำหนด dynamic SQL statement คุณจะต้องใช้ข้อความที่มี EXECUTE statement หรือ EXECUTE IMMEDIATE statement อย่างใดอย่างหนึ่ง เนื่องจากจัดเตรียม dynamic SQL statement ที่เวลารันไทม์ ไม่ใช่ที่เวลาคอมไพล์ล่วงหน้า

EXECUTE IMMEDIATE statement จัดเตรียม SQL statement และรันอย่างรวดเร็วในเวลารันไทม์ของโปรแกรม.

SQL statement สามารถแบ่งเป็นแบบพื้นฐานได้ 2 แบบด้วยกัน: SELECT statement และ non-SELECT statement. Non-SELECT statementจะประกอบด้วย statement อันได้แก่ DELETE, INSERT, และ UPDATE.

เซิร์ฟเวอร์แอ็พพลิเคชันไคลเอ็นต์ที่ใช้อินเตอร์เฟส เช่น Open Database Connectivity (ODBC) โดยทั่วไปจะใช้ dynamic SQL ในการเข้าถึง ฐานข้อมูล

หลักการที่เกี่ยวข้อง

System i Access for Windows: โปรแกรมมิง

CCSID ของคำสั่ง SQL แบบ dynamic

โดยปกติแล้วคำสั่ง SQL จะเป็นตัวแปรโฮสต์ ค่า coded character set identifier (CCSID) ของตัวแปรโฮสต์ จะถูกใช้ในรูป CCSID ของ statement text

Dynamic SQL statement จะถูกประมวลผลโดยใช้ CCSID ของ statement text ซึ่งจะส่งผลต่อ variant character ตัวอย่างเช่น เครื่องหมาย not (¬) จะถูกกำหนดให้อยู่ที่ 'BA'X ใน CCSID 500 ซึ่งหมายความว่า ถ้า CCSID ของข้อความคำสั่งของคุณคือ 500, SQL จะแสดงเครื่องหมาย not (¬) ในค่า 'BA'X

ถ้า CCSID ของข้อความคำสั่งเป็น 65535, SQL จะทำการประมวลผล variant character เหมือนกับว่ามีค่า CCSID เท่ากับ 37 ซึ่งหมายความว่า SQL จะทำการค้นหาเครื่องหมาย not (¬) ที่ '5F'X

การประมวลผลคำสั่ง non-SELECT

ก่อนที่จะสร้างคำสั่ง SQL non-SELECT แบบไดนามิก คุณจำเป็นต้องตรวจสอบว่าคำสั่ง SQL นี้สามารถรันแบบไดนามิกได้

การรัน SQL แบบไดนามิก คำสั่ง non-SELECT:

1. รันคำสั่ง SQL โดยใช้ EXECUTE IMMEDIATE, หรือทำการ PREPARE ให้กับคำสั่ง SQL, แล้วทำการ EXECUTE คำสั่งที่ถูกจัดเตรียมนั้น.
2. การจัดการกับค่าส่งคืนของ SQL ที่อาจเกิดขึ้น.

ตัวอย่างต่อไปนี้เป็นตัวอย่างของแอ็พพลิเคชันที่รัน SQL แบบ dynamic คำสั่ง non-SELECT (stmtstrg):

```
EXEC SQL
 EXECUTE IMMEDIATE :stmtstrg;
```

หลักการที่เกี่ยวข้อง

"การใช้ SQL แบบโต้ตอบ" ในหน้า 285

SQL แบบโต้ตอบอนุญาตให้โปรแกรมเมอร์ หรือผู้ดูแลฐานข้อมูล มีความรวดเร็วและสะดวกในการกำหนด, อัพเดต, ลบ, หรือสำรวจข้อมูลเพื่อทำการทดสอบ วิเคราะห์ปัญหา และการดูแลฐานข้อมูล

การใช้คำสั่ง PREPARE และ EXECUTE:

ถ้าคำสั่งที่ไม่ใช่ SELECT ไม่มี ตัวทำเครื่องหมายพารามิเตอร์ คุณจะสามารถรันคำสั่งแบบไดนามิก โดยใช้คำสั่ง EXECUTE IMMEDIATE อย่างไรก็ตาม ถ้าคำสั่งที่ไม่ใช่ SELECT มีตัวทำเครื่องหมายพารามิเตอร์ คุณจะต้องรันคำสั่ง โดยใช้คำสั่ง PREPARE และ EXECUTE

คำสั่ง PREPARE จะจัดเตรียมคำสั่งที่ไม่ใช่ SELECT (ตัวอย่างเช่น, คำสั่ง DELETE) และตั้งชื่อได้ตามความต้องการ. ถ้า DLYPRP (*YES) ถูกระบุว่าในคำสั่ง CRTSQLxxx, การจัดเตรียมจะถูกหน่วงไว้จนกระทั่งมีการใช้คำสั่งเป็นครั้งแรกในคำสั่ง EXECUTE หรือ DESCRIBE, นอกจากว่ามีการระบุ USING clause ไว้ในคำสั่ง PREPARE. หลังจากที่จัดเตรียมข้อความแล้ว ได้ถูกจัดทำขึ้น, มันจะถูกรันได้หลายครั้งภายในโปรแกรมเดียวกัน, โดยใช้ค่าต่างกันสำหรับเครื่องหมายพารามิเตอร์. ตัวอย่างต่อไปนี้เป็นคำสั่งที่ถูกจัดเตรียมไว้เพื่อรันได้หลายๆ ครั้ง:

```
DSTRING = 'DELETE FROM CORPDATA.EMPLOYEE WHERE EMPNO = ?';

/*The ? is a parameter marker which denotes
  ค่านี้เป็นตัวแปรโฮสต์ทีจะ
  ถูกแทนค่าทุกครั้งที่คำสั่งถูกรัน.*/

EXEC SQL PREPARE S1 FROM :DSTRING;

/*DSTRING เป็น delete statement ที่ PREPARE statement มีชื่อเป็น
  S1.*/

 DO UNTIL (EMP =0);
/*แอ็พพลิเคชันโปรแกรมอ่านค่าสำหรับ EMP จาก
  จอภาพ.*/
   EXEC SQL
     EXECUTE S1 USING :EMP;


END;
```

ลักษณะโดยทั่วไปที่คล้ายกันกับตัวอย่างที่กล่าวมาข้างต้น, คุณจะต้องทราบจำนวนของเครื่องหมายพารามิเตอร์ และชนิดข้อมูลของแต่ละตัว, เนื่องจากตัวแปรโฮสต์ที่ทำการจัดหาข้อมูลอินพุตนั้นจะถูกประกาศในช่วงที่โปรแกรมถูกเขียนขึ้น.

หมายเหตุ: คำสั่งที่ถูกจัดเตรียมทั้งหมดที่เชื่อมโยงกับแอ็พพลิเคชันเซิร์ฟเวอร์จะถูกทำลายลงเมื่อการเชื่อมต่อกับแอ็พพลิเคชันเซิร์ฟเวอร์สิ้นสุดลง. การเชื่อมต่อจะสิ้นสุดลงโดยการใช้คำสั่ง CONNECT (Type 1), คำสั่ง DISCONNECT, หรือ RELEASE ตามด้วย COMMIT.

การประมวลผลคำสั่ง SELECT และการใช้ descriptor

ประเภทพื้นฐานของคำสั่ง SELECT ได้แก่ fixed list และ varying list

เมื่อต้องการประมวลผลคำสั่ง SELECT แบบ fixed-list, จึงไม่จำเป็นต้องใช้ SQL descriptor.

เมื่อต้องการประมวลผลคำสั่ง SELECT แบบ varying-list, คุณต้องประกาศโครงสร้าง SQL descriptor area (SQLDA) เป็นอันดับแรก หรือ ALLOCATE SQLDA. ฟอร์มของ SQL descriptor ทั้งสองแบบสามารถใช้ส่งผ่านค่าอินพุตตัวแปรโฮสต์จาแอ็พพลิเคชันโปรแกรมไปยัง SQL และรับค่าเอาต์พุตจาก SQL. นอกจากนี้, ข้อมูลที่เกี่ยวกับนิพจน์ของรายการ SELECT สามารถส่งคืนกลับมาในคำสั่ง PREPARE หรือ DESCRIBE.

คำสั่ง SELECT แบบรายการคงที่:

ใน dynamic SQL คำสั่ง SELECT แบบรายการคงที่ จะเรียกข้อมูลที่ทราบค่าและชนิดของข้อมูล เมื่อใช้คำสั่งนี้ คุณจะสามารถคาดเดา และกำหนดตัวแปรโฮสต์ที่เหมาะสมกับข้อมูลที่ดึงออกมา ดังนั้น SQL descriptor area (SQLDA) จึงไม่มีความจำเป็น

ค่า FETCH ที่สำเร็จสมบูรณ์จะส่งคืนค่าตัวเลขที่เป็นค่าสุดท้ายกลับมาในแต่ละครั้ง, และค่าเหล่านี้จะมีรูปแบบเดียวกันกับค่าที่ส่งคืนมาสำหรับการ FETCH ในครั้งสุดท้าย. คุณสามารถระบุตัวแปรโฮสต์ได้เช่นเดียวกันกับแอ็พพลิเคชัน SQL.

คุณสามารถใช้ fixed-list dynamic SELECT statement กับแอ็พพลิเคชันโปรแกรมใดๆ ที่สนับสนุนการใช้งาน SQL.

การรัน fixed-list SELECT statement อย่างต่อเนื่อง, แอ็พพลิเคชันของคุณจะต้อง:

1. ใส่อินพุตคำสั่ง SQL ลงในตัวแปรโฮสต์.
2. ออกคำสั่ง PREPARE เพื่อตรวจสอบความถูกต้องของคำสั่ง SQL แบบ dynamic และใส่ลงไปใน form ที่สามารถถูกรันได้. ถ้า DLYPRP (*YES) ถูกระบุไว้ในคำสั่ง CRTSQLxxx, การจัดเตรียมจะถูกหน่วงไว้จนกระทั่ง statement ถูกใช้เป็นครั้งแรกในคำสั่ง EXECUTE หรือ DESCRIBE, นอกเสียจากได้ระบุ USING clause ไว้ในคำสั่ง PREPARE.
3. ประกาศเคอร์เซอร์สำหรับชื่อของคำสั่ง.
4. เปิดเคอร์เซอร์.
5. FETCH แถวเข้าไปใส่ไว้ใน fixed list ของตัวแปร (แทนที่จะไว้ใน descriptor area, เช่นเดียวกับที่คุณใช้คำสั่ง varying-list SELECT.
6. เมื่อมีการสิ้นสุดของข้อมูลเกิดขึ้น, ปิดเคอร์เซอร์.
7. จัดการกับค่าส่งคืนของ SQL ใดๆที่เกิดขึ้น.

ตัวอย่างเช่น:

```
MOVE 'SELECT EMPNO, LASTNAME FROM CORPDATA.EMPLOYEE WHERE EMPNO>?'
TO DSTRING.
EXEC SQL
 PREPARE S2 FROM :DSTRING END-EXEC.

EXEC SQL
 DECLARE C2 CURSOR FOR S2 END-EXEC.

EXEC SQL
 OPEN C2 USING :EMP END-EXEC.

PERFORM FETCH-ROW UNTIL SQLCODE NOT=0.

EXEC SQL
 CLOSE C2 END-EXEC.
STOP-RUN.
FETCH-ROW.
EXEC SQL
 FETCH C2 INTO :EMP, :EMPNAME END-EXEC.
```

หมายเหตุ: จำไว้ว่า เนื่องจากคำสั่ง SELECT, ในกรณีนี้, จะส่งคืนตัวเลข และชนิดของรายการข้อมูลเช่นเดียวกับที่รันคำสั่ง fixed-list SELECT, คุณไม่จำเป็นต้องใช้ SQL descriptor area.

คำสั่ง SELECT แบบ varying-list:

ใน dynamic SQL คำสั่ง SELECT แบบ varying-list ถูก ใช้เมื่อจำนวนและรูปแบบของคอลัมน์ผลลัพธ์ไม่สามารถคาดเดาได้ นั่นคือ คุณไม่ทราบชนิดข้อมูลหรือจำนวนตัวแปรที่คุณ ต้องการ

ดังนั้น, คุณจึงไม่สามารถกำหนดตัวแปรโฮสต์ล่วงหน้าได้ เพื่อที่จะให้เหมาะสมกับคอลัมน์ผลลัพธ์ที่จะถูกส่งคืนกลับมา.

หมายเหตุ: ใน REXX ขั้นตอนที่ 5.b, 6 และ 7 ไม่สามารถใช้ด้วยกันได้ REXX สนับสนุน เฉพาะ SQL descriptor ที่ถูกกำหนดโดยใช้โครงสร้าง SQLDA และไม่สนับสนุน SQL descriptor ที่จัดสรรแล้ว

ถ้าแอ็พพลิเคชันของคุณยอมรับคำสั่ง SELECT แบบ varying-list, โปรแกรมของคุณจะต้อง:

1. ใส่อินพุตคำสั่ง SQL ลงในตัวแปรโฮสต์.
2. ออกคำสั่ง PREPARE เพื่อตรวจสอบความถูกต้องของคำสั่ง SQL แบบ dynamic และใส่ลงไปใน form ที่สามารถถูกรันได้. ถ้า DLYPRP (*YES) ถูกระบุไว้ในคำสั่ง CRTSQLxxx, การจัดเตรียมจะถูกหน่วงไว้จนกระทั่งคำสั่งถูกใช้เป็นครั้งแรกในคำสั่ง EXECUTE หรือ DESCRIBE, นอกเสียจากได้ระบุ USING clause ไว้ในคำสั่ง PREPARE.
3. ประกาศเคอร์เซอร์สำหรับชื่อของข้อความ.
4. เปิดเคอร์เซอร์ (ที่ประกาศในขั้นตอนที่3) ที่มีชื่อของ dynamic SELECT statement อยู่.
5. สำหรับ SQL descriptor ที่จัดสรรแล้ว, ให้รันคำสั่ง ALLOCATE DESCRIPTOR เพื่อกำหนด descriptor ที่คุณต้องการใช้.
6. ออกคำสั่ง DESCRIBE เพื่อร้องขอข้อมูลจาก SQL เกี่ยวกับชนิดและขนาดของแต่ละคอลัมน์ในตาราง.

   หมายเหตุ:

   a. คุณสามารถโค้ดคำสั่ง PREPARE โดยใช้ INTO clause เพื่อดำเนินการฟังก์ชันของ PREPARE และ DESCRIBE โดยใช้คำสั่งเดี่ยว.

   b. ถ้า SQLDA และ SQLDA ไม่ใหญ่พอที่จะเก็บคอรายละเอียดคอลัมน์สำหรับคอลัมน์ที่ถูกเรียกออกมา, โปรแกรมต้องคำนวณว่า ต้องการเนื้อที่ว่างเท่าไร, และเตรียมที่เก็บให้มีขนาดเท่ากับที่ต้องการ, สร้าง SQLDA ตัวใหม่, และใช้คำสั่ง DESCRIBE.

      ถ้าการใช้ SQL descriptor ที่จัดสรรแล้วซึ่งไม่ใหญ่พอ, ให้ทำการจัดสรร descriptor ใหม่อีกครั้ง, จัดสรรด้วยจำนวน entry ขนาดใหญ่กว่า, และใช้คำสั่ง DESCRIBE ใหม่อีกครั้ง.
7. สำหรับ SQLDA descriptor, ให้จัดสรรจำนวนของหน่วยเก็บที่จำเป็น เพื่อเก็บแถวของข้อมูลที่ดึงออกมา.
8. สำหรับ SQLDA descriptor, ให้ใส่แอดเดรสหน่วยเก็บใน SQLDA เพื่อที่จะบอก SQL ให้ทราบว่าจะเก็บแต่ละไอเท็มของข้อมูลที่ดึงออกมาไว้ที่ไหน.
9. FETCH แถวข้อมูล.
10. ประมวลผลข้อมูลที่ส่งคืนใน SQL descriptor.
11. การจัดการกับค่าส่งคืนของ SQL ที่อาจเกิดขึ้น.
12. เมื่อมีการสิ้นสุดของข้อมูลเกิดขึ้น, ปิดเคอร์เซอร์.
13. สำหรับ SQL descriptor ที่จัดสรรแล้ว, ให้รันคำสั่ง DEALLOCATE DESCRIPTOR เพื่อลบ descriptor.

สิ่งอ้างอิงที่เกี่ยวข้อง

"ตัวอย่าง: คำสั่ง SELECT เพื่อจัดสรรหน่วยเก็บสำหรับ SQLDA" ในหน้า 275

สมมติว่า แอ็พพลิเคชันของคุณจำเป็นต้องจัดการกับคำสั่ง SELECT แบบ dynamic เมื่อค่าหนึ่งเปลี่ยนเป็นอีกค่าหนึ่งสำหรับใช้ต่อไป คำสั่งนี้สามารถอ่านได้จากจอแสดงผล, ซึ่งถูกส่งผ่านจากแอ็พพลิเคชันอื่น, หรือถูกสร้างขึ้นจากแอ็พพลิเคชันของคุณในขณะปฏิบัติงาน.

SQL descriptor area:

SQL แบบไดนามิก ใช้ SQL descriptor area เพื่อส่งผ่าน ข้อมูลเกี่ยวกับคำสั่ง SQL ระหว่าง SQL และแอ็พพลิเคชันของคุณ

descriptor เป็นสิ่งจำเป็นสำหรับการรันคำสั่ง DESCRIBE, DESCRIBE INPUT และ DESCRIBE TABLE, และยังสามารถใช้บนคำสั่ง PREPARE, OPEN, FETCH, CALL, และ EXECUTE ได้.

ความหมายของ ข้อมูลใน descriptor ขึ้นอยู่กับการใช้งาน ใน PREPARE และ DESCRIBE, descriptor จัดเตรียมข้อมูลให้กับแอ็พพลิเคชันโปรแกรมเกี่ยวกับคำสั่งที่ถูก จัดเตรียม ใน DESCRIBE INPUT, SQL descriptor area จัดเตรียมข้อมูลเกี่ยวกับตัวทำเครื่องหมายพารามิเตอร์ในคำสั่งที่ได้เตรียมไว้ให้กับแอ็พพลิเคชันโปรแกรม. ใน DESCRIBE TABLE, descriptor จัดเตรียมข้อมูลให้แอ็พพลิเคชันโปรแกรมเกี่ยวกับ คอลัมน์ในตารางหรือมุมมอง ใน OPEN, EXECUTE, CALL และ FETCH, descriptor จัดเตรียมข้อมูลเกี่ยวกับตัวแปรโฮสต์ ตัวอย่างเช่น คุณสามารถ อ่านค่ามาใส่ใน descriptor โดยใช้คำสั่ง DESCRIBE, เปลี่ยนค่าข้อมูลใน descriptor เพื่อใช้ตัวแปรโฮสต์ แล้วใช้ descriptor ตัวเดิมในคำสั่ง FETCH

ถ้าแอ็พพลิเคชันของคุณอนุญาตให้คุณมีหลายเคอร์เซอร์ที่เปิดอยู่ในเวลาเดียวกัน คุณสามารถโค้ดหลายๆ descriptor ได้ หนึ่งโค้ดสำหรับแต่ละคำสั่ง SELECT แบบไดนามิก

มี descriptor อยู่สองชนิด ชนิดแรกถูกนิยามด้วยคำสั่ง ALLOCATE DESCRIPTOR. ชนิดที่สองถูกนิยาม ด้วยโครงสร้าง SQLDA

ALLOCATE DESCRIPTOR ไม่สนับสนุนใน REXX SQLDAs สามารถใช้ได้ใน C, C++, COBOL, PL/I, REXX และ RPG เนื่องจาก RPG/400 ไม่ได้ จัดเตรียมวิธีการตั้งค่าตัวชี้ ดังนั้น SQLDA จะต้องถูกตั้งค่าอยู่ภายนอกโปรแกรม RPG/400 โดยโปรแกรม PL/I, C, C++, COBOL หรือ ILE RPG และโปรแกรมนั้นต้องเรียก โปรแกรม RPG/400

สิ่งอ้างอิงที่เกี่ยวข้อง

SQLCA (SQL communication area)

SQLDA (SQL descriptor area)

รูปแบบของ SQLDA:

SQL descriptor area (SQLDA) ประกอบด้วยตัวแปรสี่ตัวตามด้วยเลขเฉพาะของการเกิดขึ้นตามลำดับของกลุ่มตัวแปรหกตัวที่ชื่อ SQLVAR

หมายเหตุ: SQLDA ใน REXX จะแตกต่างออกไป

เมื่อ SQLDA ถูกใช้ใน OPEN, FETCH, CALL, และEXECUTE, แต่ละครั้งของการเกิดขึ้นของ SQLVAR จะช่วยอธิบายตัวแปรโฮสต์.

ฟิลด์ของ SQLDA มีดังนี้:

SQLDAID

SQLDAID จะเป็นเช่นเดียวกับการใช้ "eyecatcher" สำหรับดัมพ์หน่วยเก็บ. เป็นชุดอักขระ 8 ตัวที่มีค่า 'SQLDA' หลังจาก SQLDA ถูกเรียกใช้ใน PREPARE หรือ DESCRIBE statement. ตัวแปรนี้ไม่ได้ใช้สำหรับ FETCH, OPEN, CALL, หรือ EXECUTE.

ไบต์ที่ 7 สามารถใช้ในการพิจารณาว่าในแต่ละคอลัมน์มีความจำเป็นต้องใช้ SQLVAR entry มากกว่าหนึ่งหรือไม่. SQLVAR entry หลายๆตัวอาจเป็นที่ต้องการหากมี LOB หรือชนิดที่ต่างกันของคอลัมน์เกิดขึ้น. แฟล็กนี้จะถูกตั้งให้ว่างเอาไว้ถ้าไม่มี LOB หรือความต่างชนิดเกิดขึ้น.

SQLDAID ไม่สามารถใช้ได้ใน REXX

SQLDABC

SQLDABC ระบุความยาวบอง SQLDA. มันจะเป็นจำนวนเต็มแบบ 4 ไบต์ที่มีค่า SQLN*LENGTH(SQLVAR) + 16 หลังจากที่ SQLDA ถูกเรียกใช้ใน PREPARE หรือ DESCRIBE statement. SQLDABC จะต้องมีค่าเท่ากับหรือมากกว่า SQLN*LENGTH(SQLVAR) + 16 ก่อนการเรียกใช้โดย FETCH, OPEN, CALL, หรือ EXECUTE.

SQLABC ไม่สามารถใช้ได้ใน REXX

SQLN SQLN เป็นจำนวนเต็มแบบ 2 ไบต์ที่ระบุจำนวนที่เกิดขึ้นทั้งหมดของ SQLVAR. จะต้องมีการตั้งค่าก่อนที่จะถูกเรียกใช้โดย SQL statement ใดๆให้มีค่ามากกว่าหรือเท่ากับศูนย์.

SQLN ไม่สามารถใช้ได้ใน REXX

SQLD SQLD เป็นจำนวนเต็มแบบ 2 ไบต์ที่ระบุจำนวนการเกิดขึ้นของ SQLVAR, เรียกได้อีกอย่างว่า, จำนวนของตัวแปรโฮสต์หรือ คอลัมน์ที่อธิบายโดย SQLDA. ฟิลด์จะถูกตั้งค่าโดย SQL ใน DESCRIBE หรือ PREPARE statement. ใน statement อื่นๆ, ฟิลด์นี้จะถูกตั้งค่าก่อนที่จะใช้ให้มีค่ามากกว่าหรือเท่ากับศูนย์และน้อยกว่าหรือเท่ากับ SQLN.

SQLVAR

ตัวแปรกลุ่มนี้จะถูกทวนซ้ำหนึ่งครั้งสำหรับแต่ละตัวแปรโฮสต์หรือ คอลัมน์. ตัวแปรเหล่านี้จะถูกตั้งค่าโดย SQL ใน DESCRIBE หรือ PREPARE statement. ใน statement อื่นๆ, จะต้องถูกตั้งค่าก่อนการใช้. ตัวแปรเหล่านี้จะถูกกำหนดดังต่อไปนี้:

SQLTYPE

SQLTYPE เป็นจำนวนเต็มแบบ 2 ไบต์ ที่ระบุชนิดของข้อมูลของตัวแปรโฮสต์หรือคอลัมน์. โปรดดู SQLTYPE และ SQLLEN สำหรับตารางของค่าที่ถูกต้อง. จำนวนคี่ใน SQLTYPE แสดงให้เห็นว่า ตัวแปรโฮสต์มีตัวแปรชี้ที่เชื่อมโยงกันและจะถูกกำหนด address ให้โดย SQLIND.

SQLLEN

SQLLEN เป็นตัวแปรจำนวนเต็มแบบ 2 ไบต์ที่ระบุความยามของตัวแปรโฮสต์หรือคอลัมน์.

SQLRES

SQLRES เป็นเนื้อที่ 12 ไบต์ที่สำรองไว้สำหรับจุดประสงค์ในการการจัดตำแหน่งที่มีขอบเขตต่อกัน. ให้สังเกตว่า, ใน i5/OS, ตัวชี้จะ*ต้อง* อยู่ใน quad-word boundary.

SQLRES ไม่สามารถใช้ได้ใน REXX

SQLDATA

SQLDATA เป็นตัวแปรชี้แบบ 16 ไบต์ที่ระบุ address ของ ตัวแปรโฮสต์เมื่อ มีการใช้ SQLDA ใน OPEN, FETCH, CALL, และEXECUTE.

เมื่อ SQLDA ถูกใช้ใน PREPARE และ DESCRIBE, พื้นที่นี้จะซ้อนกันด้วยข้อมูลต่อไปนี้:

CCSID ของฟิลด์ตัวอักษรหรือ กราฟฟิค ที่บันทึกอยู่ในไบต์ที่สามและสี่ของ SQLDATA. สำหรับข้อมูล BIT , CCSID จะเป็น 65535. ใน REXX, CCSID จะถูกส่งคืนในรูปของตัวแปร SQLCCSID

SQLIND

SQLIND เป็นตัวชี้แบบ 16 ไบต์ที่ระบุaddress ของจำนวนเต็มจำนวนน้อยๆของตัวแปรโฮสต์ที่ใช้ในเป็นตัวระบุของ null หรือ notnull เมื่อ SQLDA ถูกใช้ใน OPEN, FETCH, CALL, และ EXECUTE. ค่าที่เป็นลบจะระบุ null แaะค่าทีไม่เป็นลบก็จะระบุ not null. ตัวชี้นี้จะถูกใช้เมื่อ SQLTYPE มีค่าเป็นจำนวนคี่เท่านั้น .

เมื่อ SQLDA ถูกใช้ใน PREPARE และ DESCRIBE, พื้นที่นั้นจะถูกจองไว้สำหรับการใช้ในครั้งต่อไป.

SQLNAME

SQLNAME เป็นตัวแปรแบบอักขระที่มีค่าความยาวผันแปรได้โดยความยาวสูงสุดคือ 30 ตัวอักษร. เมื่อทำการ PREPARE หรือ DESCRIBE, ตัวแปรนี้นจะมีชื่อของคอลัมน์, เลเบล, หรือ คอลัมน์ของระบบที่เลือกไว้. ใน OPEN, FETCH, EXECUTE, หรือ CALL, ตัวแปรนี้สามารถนำมาใช้ในการส่งผ่านค่า CCSID ของสตริงอักขระ. CCSID จะถูกส่งผ่านสำหรับตัวแปรโฮสต์แบบอักขระ และแบบกราฟิก.

ฟิลด์ SQLNAME ใน SQLVAR array entry ของอินพุต SQLDA สามารถตั้งค่าให้ระบุค่าของ CCSID ได้. โปรดดู ค่า CCSID ใน SQLDATA หรือ SQLNAME สำหรับผังข้อมูล CCSID ในฟิลด์นี้.

หมายเหตุ: ต้องจำไว้ว่า ฟิลด์ SQLNAME นั้นมีไว้สำหรับแทนทับค่าเดิมของ CCSID เท่านั้น. แอ็พพลิเคชันที่ใช้ค่าเดิมที่มีอยู่แล้วไม่จำเป็นต้องส่งผ่านข้อมูลของ CCSID. ถ้าค่าของ CCSID ไม่ถูกส่งผ่าน, จะใช้ค่าเดิมของ CCSID สำหรับงานนั้น.

ค่าเดิมของตัวแปรโฮสต์แบบกราฟิกเป็นค่าเชื่อมโยงของ CCSID แบบดับเบิ้ลไบต์สำหรับ CCSID ของงานนั้น. ถ้าค่าเชื่อมโยงของ CCSID แบบดับเบิ้ลไบต์ไม่ปรากฏ, จะใช้ค่า 65535 .

SQLVAR2

นี่เป็นโครงสร้างเพิ่มเติมของ SQLVAR ที่ประกอบด้วยฟิลด์ 3 ฟิลด์ด้วยกัน. Extended SQLVAR จำเป็นสำหรับคอลัมน์ทั้งหมดของผลลัพธ์ถ้าผลลัพธ์นั้นมีคอลัมน์ที่ต่างชนิดกันหรือ คอลัมน์ LOB อยู่. สำหรับชนิดที่ต่างกัน, จะมีชื่อเรียกที่ต่างกันด้วย. สำหรับ LOB, จะมีแอ็ตทริบิวต์ความยาวของตัวแปรโฮสต์และตัวชี้ไปที่บัฟเฟอร์ที่มีขนาดแน่นอน. ถ้าตัวบอกตำแหน่งถูกใช้ในการแสดง LOB, entry เหล่านี้จะไม่มีความจำเป็นอีก. จำนวนของการเกิด SQLVAR ที่จำเป็นนั้นขึ้นกับข้อความที่ SQLDA ได้ถูกจัดเตรียมมา และ ชนิดของข้อมูลของคอลัมน์หรือพารามิเตอร์ที่กำลังอธิบาย. ไบต์ที่ 7 ของ SQLDAID จะถูกตั้งให้เป็นจำนวนชุดของ SQLVAR ที่จำเป็นเสมอ.

ถ้า SQLD ไม่ได้ถูกตั้งค่าให้เพียงพอกับจำนวน SQLVAR ที่เกิดขึ้น:

- SQLD จะถูกตั้งค่าเป็นจำนวนรวมทั้งหมดของการเกิด SQLVAR ที่จำเป็นสำหรับทุกชุด.
- สัญญาณเตือน +237 จะถูกส่งคืนมาในฟิลด์ SQLCODE ของ SQLCA ถ้าอย่างน้อยถูกระบุไว้อย่างพอเพียงสำหรับ Base SQLVAR Entry. Base SQLVAR entry จะถูกส่งคืนมา แต่ไม่มี Extended SQLVAR ส่งคืนมา.
- สัญญาณเตือน +239 จะถูกส่งคืนมาในฟิลด์ของ SQLCODE ของ SQLCA ถ้าไม่ได้ระบุ SQLVAR อย่างเพียงพอถึงแม้จะเป็น สำหรับ Base SQLVAR Entry. ไม่มี SQLVAR entry ถูกส่งกลับมา.

SQLLONGLEN

SQLLONGLEN เป็นตัวแปรแบบจำนวนเต็ม 4 ไบต์ที่ระบุความยาวของ LOB (BLOB, CLOB, หรือ DBCLOB) ตัวแปรโฮสต์หรือ คอลัมน์.

SQLDATALEN

SQLDATALEN เป็นตัวแปรชี้แบบ 16ไบต์ที่ระบุ addressของความยาวของตัวแปรโฮสต์. ตัวแปรนี้จะใช้สำหรับ LOB (BLOB, CLOB, และ DBCLOB) ตัวแปรโฮสต์เท่านั้น. ไม่ได้ใช้เพื่อ DESCRIBE หรือ PREPARE.

ถ้าฟิลด์นี้เป็น NULL, แล้วความยาวที่แน่นอนของข้อมูลจะถูกบันทึกใน 4 ไบต์ทันที ก่อนที่จะเป็นส่วนเริ่มของข้อมูล, และ SQLDATA จะชี้ไปยังไบต์แรกของความยาวของฟิลด์นั้น. ความยาวระบุจำนวนของไบต์สำหรับ BLOB หรือ CLOB, และจำนวนตัวอักษรสำหรับ DBCLOB.

ถ้าฟิลด์นี้ไม่ได้มีค่าเป็น NULL, จะมีการเก็บค่าของตัวชี้ใน long buffer แบบ 4 ไบต์ที่มีความยาวที่แน่นอนในหน่วยของไบต์(แม้จะเป็นสำหรับ DBCLOB) ของข้อมูลในบัฟเฟอร์ที่ถูกชี้โดย ฟิลด์ SQLDATA ใน matching base SQLVAR.

SQLDATATYPE_NAME

SQLDATATYPE_NAME เป็นตัวแปรที่เป็นอักขระแบบความยาวผันแปรได้ด้วยความยาวสูงสุดเท่ากับ 30. ใช้สำหรับ DESCRIBE หรือ or PREPARE. ตัวแปรนี้จะถูกตั้งค่าให้เป็นค่าใดค่าหนึ่งต่อไปนี้:

- สำหรับคอลัมน์ต่างชนิดกัน, database manager ตั้งค่านี้ไว้เป็นชื่อที่แตกต่างกันอย่างสิ้นเชิง. ถ้าชื่อที่ตั้งไว้ยาวกว่า 30 ไบต์, ก็จะถูกตัดตอนปลายออกไป .
- สำหรับเลเบล, database manager ตั้งค่านี้ไว้ที่ 20 ไบต์แรกของเลเบล.
- สำหรับชื่อคอลัมน์, database manager จะตั้งค่านี้ไว้ที่ชื่อคอลัมน์.

งานที่เกี่ยวข้อง

การโค้ดคำสั่ง SQL ใน REXX applications

สิ่งอ้างอิงที่เกี่ยวข้อง

"ตัวอย่าง: คำสั่ง SELECT เพื่อจัดสรรหน่วยเก็บสำหรับ SQLDA"

สมมติว่า แอ็พพลิเคชันของคุณจำเป็นต้องจัดการกับคำสั่ง SELECT แบบ dynamic เมื่อค่าหนึ่งเปลี่ยนเป็นอีกค่าหนึ่งสำหรับใช้ต่อไป คำสั่งนี้สามารถอ่านได้จากจอแสดงผล, ซึ่งถูกส่งผ่านจากแอ็พพลิเคชันอื่น, หรือถูกสร้างขึ้นจากแอ็พพลิเคชันของคุณในขณะปฏิบัติงาน.

ตัวอย่าง: คำสั่ง SELECT เพื่อจัดสรรหน่วยเก็บสำหรับ SQLDA:

สมมติว่า แอ็พพลิเคชันของคุณจำเป็นต้องจัดการกับคำสั่ง SELECT แบบ dynamic เมื่อค่าหนึ่งเปลี่ยนเป็นอีกค่าหนึ่งสำหรับใช้ต่อไป คำสั่งนี้สามารถอ่านได้จากจอแสดงผล, ซึ่งถูกส่งผ่านจากแอ็พพลิเคชันอื่น, หรือถูกสร้างขึ้นจากแอ็พพลิเคชันของคุณในขณะปฏิบัติงาน.

พูดได้อีกอย่างว่า, คุณไม่ทราบแน่ชัดว่าข้อความนี้จะส่งค่าอะไรคืนกลับมาในทุกครั้ง. แอ็พพลิเคชันจำเป็นต้องจัดการกับจำนวนที่แตกต่างกันออกไปของคอลัมน์ผลลัพธ์ที่ไม่ทราบชนิดข้อมูลที่แน่นอนก่อนล่วงหน้า

ยกตัวอย่างเช่น, ข้อความต่อไปนี้จำเป็นจะต้องถูกประมวลผล:

```
SELECT WORKDEPT, PHONENO
  FROM CORPDATA.EMPLOYEE
  WHERE LASTNAME = 'PARKER'
```

หมายเหตุ: SELECT statement นี้*ไม่มี* INTO clause. Dynamic SELECT statement จะต้อง*ไม่* มี INTO clause, ถึงแม้ว่าจะส่งค่าคืนมาเพียงแถวเดียว.

ข้อความจะถูกกำหนดค่าให้กับตัวแปรโฮสต์. ตัวแปรโฮสต์, ในกรณีนี้มีชื่อว่า DSTRING, จะถูกทำการประมวลผลโดยใช้คำสั่ง PREPARE ตามที่ได้แสดงไว้ดังนี้:

```
EXEC SQL
PREPARE S1 FROM :DSTRING;
```

ขั้นถัดไป, คุณจำเป็นจะต้องหาค่าจำนวนของคอลัมน์ผลลัพธ์และชนิดของข้อมูล. ในการที่จะทำนั้น, ต้องอาศัย SQLDA.

ขั้นแรกในการกำหนด SQLDA ก็คือ การจัดสรรหน่วยเก็บ (ไม่จำเป็นต้องจัดสรรหน่วยเก็บใน REXX) เทคนิคสำหรับการจองเนื้อที่ขึ้นกับภาษาที่ใช้ SQLDA จะต้องได้รับจัดสรร (รีซอร์ส) เพื่อใช้งานในขอบเขต 16 ไบต์. SQLDA ประกอบด้วยส่วนหัวที่ความยาวคงที่ซึ่งมีขนาดความยาว 16 ไบต์. ส่วนหัวจะต่อท้ายด้วยส่วนของarrayที่ความยาวแปรผัน (SQLVAR), แต่ละส่วนประกอบจะมีความยาว.

จำนวนของเนื้อที่ที่ต้องการในการจัดสรร (รีซอร์ส) เพื่อใช้งานขึ้นอยู่กับจำนวนองค์ประกอบที่ต้องการจะมีใน SQLVAR array. แต่ละคอลัมน์ที่เลือกจะต้องมีความสัมพันธ์กับองค์ประกอบของ SQLVAR array. ดังนั้น, จำนวนของคอลัมน์ที่แสดงใน SELECT statementจะเป็นตัวบอกจำนวนองค์ประกอบของ SQLVAR array ที่จะต้องทำการจัดสรร (รีซอร์ส) เพื่อใช้งาน. เนื่องจากคำสั่ง SELECT ถูกระบุที่เวลารันไทม์จึง เป็นไปไม่ได้ที่จะรู้แน่ชัดว่าจะมีการเข้าไปใช้คอลัมน์กี่คอลัมน์ ดังนั้น, คุณจึงควรที่จะประเมินจำนวนของคอลัมน์. สมมติว่า, ในตัวอย่างนี้, จะมีคอลัมน์ได้ไม่เกิน 20 คอลัมน์ที่จะถูกเรียกใช้โดย single SELECT statement. ในกรณีนี้, SQLVAR array ควรจะมีมิติเป็น 20, เพื่อให้แน่ใจว่าแต่ละรายการใน select-list มี entry ที่เกี่ยวเนื่องกันในSQLVAR. จะทำให้ขนาดของ SQLDA เท่ากับ 20 x 80, หรือ 1600, บวก 16 สำหรับจำนวนไบต์ทั้งหมด 1616 ไบต์

การจัดสรร (รีซอร์ส) เพื่อใช้งานตามที่ประเมินให้เพียงพอสำหรับ SQLDA, จำเป็นที่จะต้องตั้งค่าในฟิลด์ SQLN ของ SQLDA ให้มีค่าเท่ากับจำนวนของ SQLVAR array element, ในที่นี้มีค่าเท่ากับ 20.

เมื่อทำการจัดสรรหน่วยเก็บ และเตรียมข้อมูลเบื้องต้นเกี่ยวกับขนาดแล้ว, คุณสามารถออกคำสั่ง DESCRIBE ได้.

```
EXEC SQL
DESCRIBE S1 INTO :SQLDA;
```

เมื่อ DESCRIBE statement ถูกรันn, SQL จะใส่ค่าใน SQLDA เพื่อให้ข้อมูลเกี่ยวกับ select-list สำหรับ statement ของคุณ. ตารางต่อไปนี้แสดงเนื้อหาของ the SQLDA หลังจากการรัน DESCRIBE . จะแสดงเฉพาะ entry ส่วนที่มีความหมายใน context นี้เท่านั้น.

ตารางที่ 43. ส่วนหัวของ SQLDA

รายละเอียด	**ค่า**
SQLAID	'SQLDA'
SQLDABC	1616
SQLN	20
SQLD	2

SQLDAID เป็นฟิลด์ identifier ที่ initialize โดย SQL เมื่อ DESCRIBE ถูกรัน. SQLDABC เป็นไบต์ที่นับหรือบอกขนาดของ SQLDA. ส่วนหัวของ SQLDA จะต่อท้ายด้วย 2 occurrence ของโครงสร้าง SQLVAR, แต่ละ occurrence สำหรับแต่ละคอลัมน์ในตารางผลลัพธ์ของคำสั่ง SELECT ซึ่งอธิบายได้ดังนี้:

ตารางที่ 44. SQLVAR องค์ประกอบ 1

รายละเอียด	**ค่า**
SQLTYPE	453

ตารางที่ 44. SQLVAR องค์ประกอบ 1 (ต่อ)

รายละเอียด	ค่า
SQLLEN	3
SQLDATA (3:4)	37
SQLNAME	8 WORKDEPT

ตารางที่ 45. SQLVAR องค์ประกอบ 2

รายละเอียด	ค่า
SQLTYPE	453
SQLLEN	4
SQLDATA(3:4)	37
SQLNAME	7 PHONENO

โปรแกรมของคุณอาจจะต้องปรับเปลี่ยนค่าของ SQLN ถ้า SQLDA มีขนาดไม่ใหญ่พอที่จะทำการเก็บ SQLVAR elements ที่ได้ทำการประกาศไว้แล้ว. ตัวอย่างเช่น, สมมติว่าแทนที่ค่าสูงสุดจะเป็น 20 คอลัมน์, SELECT statement ได้ทำการส่งคืนกลับมาเป็น 27. SQL ไม่สามารถอธิบาย select-list นี้ได้เนื่องจาก SQLVAR ต้องการ elementมากกว่าที่ระบุไว้ในเนื้อที่ที่กำหนด. แทนที่จะเป็นเช่นนั้น, SQL ตั้งค่าให้ SQLD เป็นตัวเลขที่แน่นอนของคอลัมน์ระบุโดย SELECT statement และส่วนที่เหลือของโครงสร้างก็จะถูกละเอาไว้. ดังนั้น, หลังจากการทำ DESCRIBE, ควรจะเปรียบเทียบค่าของ SQLN กับค่าของ SQLD. ถ้าค่าของ SQLD มากกว่าค่าของ SQLN, จะจัดสรร SQLDA ให้ใหญ่ขึ้นได้ยึดเอาค่า SQLD เป็นหลัก, ดังต่อไปนี้, และทำการ DESCRIBE อีกครั้ง:

```
EXEC SQL
    DESCRIBE S1 INTO :SQLDA;
IF SQLN <= SQLD THEN
DO;

/*จัดสรร (รีซอร์ส) ให้ SQLDA ใหญ่ขึ้นโดยใช้ค่าของ SQLD.*/
/*ปรับค่าของ SQLN ให้เป็นค่าที่มากกว่า.*/

EXEC SQL
    DESCRIBE S1 INTO :SQLDA;
END;
```

ถ้าใช้ DESCRIBE ในคำสั่ง non-SELECT, SQL จะตั้งค่า SQLD ให้เป็น 0 ดังนั้น, ถ้าโปรแกรมของคุณถูกออกแบบมาให้ประมวลผลทั้ง SELECT และ non SELECT statement, สามารถอธิบายแต่ละข้อความหลังจากที่จัดเตรียมแล้วเพื่อที่จะตรวจดูว่าเป็น SELECT statement หรือไม่. ตัวอย่างนี้ได้รับการออกแบบเพื่อให้ประมวลผลเฉพาะ SELECT statement; ค่าของ SQLD จะไม่ถูกตรวจสอบ.

โปรแกรมจะต้องวิเคราะห์ element ของ SQLVAR ที่ส่งกลับมาจากการ DESCRIBE ที่สมบูรณ์. รายการแรกในคือ WORKDEPT. ในฟิลด์ SQLTYPE, DESCRIBE จะส่งคืนค่าสำหรับชนิดข้อมูลของนิพจน์ และสามารถใช้ค่า null ได้หรือไม่.

ในตัวอย่างนี้, SQL จะตั้งค่า SQLTYPE เป็น 453 ใน SQLVAR องค์ประกอบ 1. ซึ่งระบุว่า WORKDEPT คือคอลัมน์ผลลัพธ์สตริงอักขระแบบความยาวคงที่ และค่า null จะถูกอนุญาตให้ใช้ในคอลัมน์.

SQL ตั้งค่า SQLLEN เป็นความยาวของคอลัมน์. เนื่องจากชนิดข้อมูลของ WORKDEPT เป็น CHAR, SQL จึงตั้งค่า SQLLEN ให้เท่ากับความยาวของคอลัมน์ตัวอักษร. สำหรับ WORKDEPT, ซึ่งมีความยาวเท่ากับ 3. ดังนั้น, เมื่อคำสั่ง SELECT ถูกรันในเวลาต่อมา, จึงจำเป็นที่จะต้องมีพื้นที่หน่วยเก็บที่เพียงพอในการเก็บชุดอักขระ CHAR(3) ได้.

เนื่องจากชนิดข้อมูลของ WORKDEPT เป็น CHAR FOR SBCS DATA, 4 ไบต์แรกของ SQLDATA จะถูกตั้งค่าให้เป็น CCSID ของคอลัมน์ตัวอักษร.

ฟิลด์สุดท้ายใน SQLVAR element จะเป็นสตริงอักขระแบบความยาวแปรผัน เรียกว่า SQLNAME. 2 ไบต์แรกของ SQLNAME จะเก็บค่าความยาวของข้อมูลตัวอักษรอยู่. ชื่อของข้อมูลตัวอักษรมักจะเป็นชื่อของคอลัมน์ที่ใช้ใน SELECT statement, ในกรณีนี้คือ WORKDEPT. exception ในกรณีนี้คือ รายการใน select-list ที่ไม่มีชื่อ, เช่น ฟังก์ชัน (ตัวอย่างเช่น, SUM(SALARY)), นิพจน์ (ตัวอย่างเช่น, A+B-C), และค่าคงที่. ในกรณีเหล่านี้, SQLNAME จะเป็นสตริงว่างเปล่า. SQLNAME สามารถเก็บค่าของเลเบลมากกว่าชื่อ. พารามิเตอร์ตัวหนึ่งที่เชื่อมโยงกับ PREPARE และ DESCRIBE statement คือ USING clause. สามารถระบุได้ดังนี้:

```
EXEC SQL
    DESCRIBE S1 INTO:SQLDA
        USING LABELS;
```

ถ้าระบุว่า:

NAMES (หรือละพารามิเตอร์ USING ทั้งหมด)
จะใส่แต่ชื่อของคอลัมน์ในฟิลด์ SQLNAME.

SYSTEM NAMES
จะใส่แต่ชื่อของคอลัมน์ของระบบลงในฟิลด์ SQLNAME.

LABELS
จะใส่แต่เลเบลที่เชื่อมโยงกับคอลัมน์ที่แสดงอยู่ใน SQL statement.

ANY เลเบลจะถูกใส่ลงในฟิลด์ SQLNAME สำหรับคอลัมน์ที่มีเลเบล; มิฉะนั้นจะใส่ชื่อคอลัมน์แทน .

BOTH ชื่อและเลเบลจะถูกใส่ในฟิลด์ด้วยความยาวที่เท่ากัน. จำไว้ว่าให้เพิ่มขนาดของ SQLVAR array เนื่องจากมีการใช้จำนวนของ element นั้นเป็นสองเท่า.

ALL ชื่อคอลัมน์, เลเบล, และ ชื่อคอลัมน์ของระบบจะถูกใส่ในฟิลด์ด้วยความยาวที่เท่ากัน. จำไว้ว่าจะต้องเพิ่มขนาด SQLVAR array เป็นสามเท่า

ในตัวอย่างนี้, element ที่สองของ SQLVAR จะเก็บข้อมูลสำหรับคอลัมน์ที่สองที่ใช้ใน select: PHONENO. รหัส 453 ใน SQLTYPE ระบุว่า PHONENO เป็นคอลัมน์ CHAR . SQLLEN ถูกตั้งค่าให้เป็น 4.

ตอนนี้จำเป็นต้องติดตั้งเพื่อที่จะใช้ SQLDA ในการเรียกค่าออกมาเมื่อในขณะรัน SELECT statement.

หลังจากที่วิเคราะห์ผลลัพธ์ของ DESCRIBE, สามารถจัดสรร (รีซอร์ส) เพื่อใช้งานเนื้อที่สำหรับตัวแปรที่จะเก็บผลลัพธ์ของ SELECT statement. สำหรับ WORKDEPT, ฟิลด์ตัวอักษรที่มีความยาวเท่ากับ 3 จะต้องได้รับการจัดสรร (รีซอร์ส) เพื่อใช้งาน; สำหรับ PHONENO, จะต้องมีการจัดสรร (รีซอร์ส) เพื่อใช้งานฟิลด์ตัวอักษรมีความยาวเท่ากับ 4 . เนื่องจากผลลัพธ์ทั้งสองแบบนี้สามารถมีค่า NULL, ตัวแปรตัวบ่งชี้ต้องถูกจัดสรรเพื่อใช้งานสำหรับแต่ละฟิลด์ด้วยเช่นกัน.

หลังจากที่ได้จัดสรร (รีซอร์ส) เพื่อใช้งาน, จะต้องตั้งค่าให้ SQLDATA และ SQLIND ให้ชี้ไปยังพื้นที่ของการจัดสรร (รีซอร์ส) เพื่อใช้งาน. สำหรับแต่ละ element ของ SQLVAR array, SQLDATA จะชี้ไปยังที่ที่ผลลัพธ์จะถูกนำไปเก็บไว้. SQLIND ชี้ไปที่ที่ค่าของ null indicator จะถูกเก็บเอาไว้. ตารางดังต่อไปนี้จะแสดงลักษณะโครงในขณะนี้. เฉพาะ entryที่มีความหมายจะถูกนำมาแสดงในบริบท นี้ :

ตารางที่ 46. ส่วนหัวของ SQLDA

รายละเอียด	**ค่า**
SQLAID	'SQLDA'
SQLDABC	1616
SQLN	20
SQLD	2

ตารางที่ 47. SQLVAR องค์ประกอบ 1

รายละเอียด	**ค่า**
SQLTYPE	453
SQLLEN	3
SQLDATA	ตัวชี้ไปยังพื้นที่ที่เก็บผลลัพธ์แบบ CHAR(3)
SQLIND	ตัวชี้ไปยังตัวบ่งชี้จำนวนเต็มแบบ 2 ไบต์สำหรับคอลัมน์ผลลัพธ์

ตารางที่ 48. SQLVAR องค์ประกอบ 2

รายละเอียด	**ค่า**
SQLTYPE	453
SQLLEN	4
SQLDATA	ตัวชี้สำหรับพื้นที่ที่เก็บผลลัพธ์แบบ CHAR(4)
SQLIND	ตัวชี้ไปยังตัวบ่งชี้จำนวนเต็มแบบ 2 ไบต์สำหรับคอลัมน์ผลลัพธ์

ตอนนี้ก็พร้อมที่จะเรียกดูผลลัพธ์ของ SELECT statements . การระบุ SELECT statement แบบ dynamic จะต้องไม่มี INTO statement. ดังนั้น, SELECT statement ที่กำหนดแบบ dynamic ต้องใช้เคอร์เซอร์. รูปแบบพิเศษของ DECLARE, OPEN, และ FETCH ถูกใช้สำหรับการระบุคำสั่ง SELECT แบบ dynamic.

คำสั่ง DECLARE สำหรับตัวอย่างนี้คือ:

```
EXEC SQL DECLARE C1 CURSOR FOR S1;
```

ดังที่ได้เห็นแล้วว่า, ความแตกต่างเพียงประการเดียวคือชื่อของ prepared SELECT statement (S1) จะถูกใช้แทนชื่อของตัว SELECT statement เอง. การดึงข้อมูลออกมาสำหรับแถวที่เป็นผลลัพธ์จะทำได้ดังนี้:

```
EXEC SQL
    OPEN C1;
EXEC SQL
    FETCH C1 USING DESCRIPTOR :SQLDA;
DO WHILE (SQLCODE = 0);
/*Process the results pointed to by SQLDATA*/
EXEC SQL
    FETCH C1 USING DESCRIPTOR :SQLDA;
END;
EXEC SQL
    CLOSE C1;
```

เคอร์เซอร์ถูกเปิด. แถวที่เป็นผลลัพธ์จาก SELECT จะถูกส่งคืนมาครั้งละหนึ่งแถวโดยใช้ FETCH statement. ใน FETCH statement,จะไม่มีรายชื่อของตัวแปรโฮสต์อยู่. แทนที่จะเป็นเช่นนั้น, FETCH statement จะบอกให้SQL ส่งคืนผลลัพธ์ไปในพื้นที่ที่ระบุโดย SQLDA. ผลลัพธ์จะถูกส่งคืนมาในพื้นที่จัดเก็บถูกชี้อยู่โดยฟิลด์ SQLDATA และ SQLIND ของ SQLVAR element. หลังจากที่ได้ทำการประมวลผล FETCH statement, ตัวชี้ SQLDATA สำหรับ WORKDEPT มีค่าอ้างอิงตั้งเป็น 'E11'. ซึ่งค่าของตัวบ่งชี้ที่เกี่ยวเนื่องกันเป็น 0 เนื่องจากค่า non-null ถูกส่งคืนมา. ตัวชี้ SQLDATA สำหรับ PHONENO มีค่าอ้างอิงเป็น '4502'. และค่าของตัวบ่งชี้ที่เกี่ยวเนื่องกันเป็น 0 เนื่องจากค่า non-null ถูกส่งคืนมา.

สิ่งอ้างอิงที่เกี่ยวข้อง

"คำสั่ง SELECT แบบ varying-list" ในหน้า 270
ใน dynamic SQL คำสั่ง SELECT แบบ varying-list ถูก ใช้เมื่อจำนวนและรูปแบบของคอลัมน์ผลลัพธ์ไม่สามารถคาดเดาได้ นั่นคือ คุณไม่ทราบชนิดข้อมูลหรือจำนวนตัวแปรที่คุณ ต้องการ

"รูปแบบของ SQLDA" ในหน้า 272
SQL descriptor area (SQLDA) ประกอบด้วยตัวแปรสี่ตัวตามด้วยเลขเฉพาะของการเกิดขึ้นตามลำดับของกลุ่มตัวแปรหกตัวที่ชื่อ SQLVAR

ตัวอย่าง: คำสั่ง SELECT ที่ใช้ SQL descriptor ที่ถูกจัดสรรแล้ว:

สมมติว่า แอ็พพลิเคชันของคุณจำเป็นต้องจัดการกับคำสั่ง SELECT แบบ dynamic เมื่อค่าหนึ่งเปลี่ยนเป็นอีกค่าหนึ่งสำหรับใช้ต่อไป คำสั่งนี้สามารถอ่านได้จากจอแสดงผล ซึ่งถูกส่งผ่านจากแอ็พพลิเคชันอื่น หรือถูกสร้างขึ้นจากแอ็พพลิเคชันของคุณแบบ dynamic

พูดได้อีกอย่างว่า, คุณไม่ทราบแน่ชัดว่า คำสั่งนี้จะส่งค่าอะไรคืนกลับมาในทุกครั้ง. แอ็พพลิเคชันจำเป็นต้องจัดการกับจำนวนที่แตกต่างกันออกไปของคอลัมน์ผลลัพธ์ที่ไม่ทราบชนิดข้อมูลที่แน่นอนก่อนล่วงหน้า

ยกตัวอย่างเช่น, คำสั่งต่อไปนี้จำเป็นจะต้องถูกประมวลผล:

```
SELECT WORKDEPT, PHONENO
  FROM CORPDATA.EMPLOYEE
  WHERE LASTNAME = 'PARKER'
```

หมายเหตุ: คำสั่ง SELECT นี้ไม่มี INTO clause. คำสั่ง SELECT แบบ dynamic จะต้อง*ไม่*มี INTO clause, ถึงแม้ว่าจะส่งค่าคืนมาเพียงแถวเดียว.

คำสั่งจะถูกกำหนดค่าให้กับตัวแปรโฮสต์. ตัวแปรโฮสต์, ในกรณีนี้มีชื่อว่า DSTRING, จะถูกทำการประมวลผลโดยใช้คำสั่ง PREPARE ตามที่ได้แสดงไว้ดังนี้:

```
EXEC SQL
PREPARE S1 FROM :DSTRING;
```

ขั้นถัดไป, คุณจำเป็นจะต้องหาค่าจำนวนของคอลัมน์ผลลัพธ์และชนิดของข้อมูล. หากต้องการทำสิ่งนี้, คุณจำเป็นต้องจัดสรรจำนวน entry ขนาดใหญ่สุดสำหรับ SQL descriptor ที่คุณคิดว่าคุณต้องการ. สมมติว่า มีคอลัมน์ไม่เกิน 20 คอลัมน์ถูกเรียกใช้โดยคำสั่ง SELECT เดียว.

```
EXEC SQL
ALLOCATE DESCRIPTOR 'mydescr' WITH MAX 20;
```

ถึงตอนนี้ descriptor จะถูกจัดสรร, คำสั่ง DESCRIBE สามารถเรียกใช้เพื่อรับข้อมูลคอลัมน์.

```
EXEC SQL
DESCRIBE S1 USING DESCRIPTOR 'mydescr';
```

เมื่อคำสั่ง DESCRIBE ถูกรัน, SQL จะใส่ค่าที่ได้เตรียมข้อมูลเกี่ยวกับรายการที่เลือกของคำสั่งเข้าไปใน SQL descriptor area ซึ่งถูกนิยามโดย 'mydescr'.

ถ้า DESCRIBE กำหนดว่า มี entry ไม่เพียงพอที่จะถูกจัดสรรใน descriptor, SQLCODE +239 จะถูกเรียกใช้. ส่วนหนึ่งของการวินิจฉัยนี้, การแทนที่ค่าข้อความที่สองจะบ่งชี้ถึงจำนวนของ entry ที่ต้องการ. ตัวอย่างโค้ดต่อไปนี้แสดงวิธีที่เงื่อนไขนี้สามารถตรวจพบ และแสดง descriptor ที่ถูกจัดสรรด้วยขนาดที่ใหญ่กว่า.

```
/* Determine the returned SQLCODE from the DESCRIBE statement */
EXEC SQL
  GET DIAGNOSTICS CONDITION 1: returned_sqlcode = DB2_RETURNED_SQLCODE;

if returned_sqlcode = 239 then do;

/* Get the second token for the SQLCODE that indicated
     not enough entries were allocated */

EXEC SQL
    GET DIAGNOSTICS CONDITION 1: token = DB2_ORDINAL_TOKEN_2;
  /* Move the token variable from a character host variable into an integer host variable */
  EXEC SQL
    SET :var1 = :token;
  /* Deallocate the descriptor that is too small */
  EXEC SQL
    DEALLOCATE DESCRIPTOR 'mydescr';
  /* Allocate the new descriptor to be the size indicated by the retrieved token */
  EXEC SQL
    ALLOCATE DESCRIPTOR 'mydescr' WITH MAX :var1;
  /* Perform the describe with the larger descriptor */
  EXEC SQL
   DESCRIBE s1 USING DESCRIPTOR 'mydescr';
end;
```

ถึงตอนนี้ descriptor จะมีข้อมูล เกี่ยวกับคำสั่ง SELECT และคุณพร้อมที่จะดึงผลลัพธีของคำสั่ง SELECT ออกมา. สำหรับ SQL แบบ dynamic, คำสั่ง SELECT INTO จะไม่อนุญาตให้ใช้. คุณต้องใช้เคอร์เซอร์.

```
EXEC SQL
  DECLARE C1 CURSOR FOR S1;
```

คุณจะสังเกตว่า ชื่อคำสั่งที่ถูกจัดเตรียมจะถูกใช้ในการประกาศเคอร์เซอร์แทนการทำคำสั่ง SELECT ให้สมบูรณ์. ถึงตอนนี้ คุณสามารถวนซ้ำแถวที่เลือก, ประมวลผลแถวเหล่านั้นตามที่คุณอ่านได้. ตัวอย่างโค้ดต่อไปนี้แสดงถึงวิธีการทำสิ่งนี้.

```
EXEC SQL
  OPEN C1;

EXEC SQL
    FETCH C1 USING SQL DESCRIPTOR 'mydescr';
do while not at end of data;

 /* process current data returned  (see below for discussion of doing this) */

/* then read the next row */

 EXEC SQL
    FETCH C1 USING SQL DESCRIPTOR 'mydescr';
end;

EXEC SQL
  CLOSE C1;
```

เคอร์เซอร์ถูกเปิด. แถวที่เป็นผลลัพธ์จากคำสั่ง SELECT จะถูกส่งคืนมาครั้งละหนึ่งแถวโดยใช้คำสั่ง FETCH. ในคำสั่ง FETCH, จะไม่มีรายชื่อของตัวแปรโฮสต์อยู่. แทนที่จะเป็นเช่นนั้น, คำสั่ง FETCH จะบอกให้ SQL ส่งคืนผลลัพธ์เข้าไปใน descriptor area.

หลังจากประมวลผล FETCH แล้ว, คุณสามารถใช้คำสั่ง GET DESCRIPTOR เพื่ออ่านค่าเหล่านั้น. อันดับแรก, คุณต้องอ่านค่าส่วนหัวที่บ่งชี้ถึงจำนวน descriptor entry ที่ถูกใช้.

```
EXEC SQL
  GET DESCRIPTOR 'mydescr' :count = COUNT;
```

หลังจากนั้น คุณสามารถอ่านข้อมูลเกี่ยวกับ descriptor entry แต่ละตัว. หลังจากที่คุณกำหนดชนิดข้อมูลของคอลัมน์ผลลัพธ์แล้ว, คุณสามารถทำให้ GET DESCRIPTOR อื่นส่งคืนค่าที่เป็นจริง. หากต้องการรับค่าของตัวบ่งชี้, ให้ระบุไอเท็ม INDICATOR. ถ้าค่าของไอเท็ม INDICATOR เป็นลบ, ค่าของไอเท็ม DATA จะไม่ถูกนิยาม. จนกว่า FETCH อื่นจะถูกทำ, ไอเท็ม descriptor จะยังคงรักษาค่าเหล่านั้นไว้.

```
do i = 1 to count;
  GET DESCRIPTOR 'mydescr' VALUE :i  /* set entry number to get */
                         :type = TYPE,                   /* get the data type */
                         :length = LENGTH,               /* length value */
                         :result_ind = INDICATOR;
 if result_ind >= 0 then
    if type = character
     GET DESCRIPTOR 'mydescr' VALUE :i
                        :char_result = DATA;       /* read data into character field */
    else
    if type = integer
       GET DESCRIPTOR 'mydescr' VALUE :i
                        :int_result = DATA;       /* read data into integer field */
  else
      /* continue checking and processing for all data types that might be returned */
end;
```

มีไอเท็ม descriptor อื่นๆ หลายไอเท็มที่คุณอาจต้องการตรวจสอบ เพื่อกำหนดวิธีการจัดการกับข้อมูลผลลัพธ์. PRECISION, SCALE, DB2_CCSID, และ DATETIME_INTERVAL_CODE อยู่ระหว่างกัน. ตัวแปรโฮสต์ที่มีการอ่านค่า DATA เข้าไปในตัวแปรต้องมีชนิดข้อมูลเดียวกัน และ CCSID ต้องเป็นข้อมูลที่อ่านได้. ถ้าชนิดข้อมูลมีความยาวผันแปร, ตัวแปรโฮสต์จะถูกประกาศความยาวได้ยาวกว่าข้อมูลจริง. สำหรับชนิดข้อมูลอื่นๆ ทั้งหมด, ความยาวต้องตรงกัน.

NAME, DB2_SYSTEM_COLUMN_NAME, และ DB2_LABEL จะถูกใช้เพื่อรับค่าชื่อที่สัมพันธ์กันสำหรับคอลัมน์ผลลัพธ์. โปรดดู GET DESCRIPTOR สำหรับข้อมูลเพิ่มเติมเกี่ยวกับไอเท็มที่ถูกส่งคืนสำหรับคำสั่ง GET DESCRIPTOR และสำหรับ definition ของค่า TYPE

ตัวทำเครื่องหมายพารามิเตอร์:

*ตัวทำเครื่องหมายพารามิเตอร์*เป็นเครื่องหมายคำถาม (?) ที่ปรากฏในสตริงคำสั่งแบบ dynamic เครื่องหมายคำถามอาจปรากฏในที่ที่ อาจมีตัวแปรโฮสต์ ถ้าสตริงคำสั่งเป็นคำสั่ง SQL แบบ static

ในตัวอย่างที่เราใช้กันนั้น คำสั่ง SELECT ที่ถูกรันแบบ dynamic มีค่าคงที่ใน WHERE clause:

```
WHERE LASTNAME = 'PARKER'
```

ถ้าต้องการรันคำสั่ง SELECT ตัวเดียวกันหลายๆ ครั้ง, โดยใช้ค่า LASTNAME ที่แตกต่างกัน, คุณสามารถใช้คำสั่ง SQL ที่มีลักษณะดังนี้:

```
SELECT WORKDEPT, PHONENO
    FROM CORPDATA.EMPLOYEE
    WHERE LASTNAME = ?
```

เมื่อใช้ตัวทำเครื่องหมายพารามิเตอร์, แอ็พพลิเคชันของคุณไม่จำเป็นต้องตั้งค่าชนิดข้อมูล และค่าสำหรับพารามิเตอร์ จนกว่ารันไทม์. ด้วยการระบุ descriptor บนคำสั่ง OPEN, คุณสามารถใช้ค่าแทนตัวทำเครื่องหมายพารามิเตอร์ในคำสั่ง SELECT ได้.

เมื่อต้องการโค้ดโปรแกรมแบบนี้, คุณจำเป็นต้องใช้คำสั่ง OPEN กับ descriptor clause. คำสั่ง SQL นี้ไม่เพียงแต่จะใช้ในการเปิดเคอร์เซอร์, แต่ยังใช้แทนตัวทำเครื่องหมายพารามิเตอร์แต่ละตัวด้วยค่าของ descriptor entry ที่เกี่ยวเนื่องกัน. ชื่อ descriptor ที่คุณระบุด้วยคำสั่งนี้ต้องระบุ descriptor ที่มีค่าของ definition ที่ถูกต้อง. descriptor นี้ไม่ได้ถูกใช้เพื่อส่งคืนข้อมูลเกี่ยวกับหน่วยข้อมูล ซึ่งเป็นส่วนหนึ่งของรายการ SELECT. และมีข้อมูลเกี่ยวกับค่าที่ถูกใช้เพื่อแทนตัวทำเครื่องหมายพารามิเตอร์ในคำสั่ง SELECT. ซึ่งจะได้รับข้อมูลมาจากแอ็พพลิเคชัน, ที่ต้องมีการออกแบบเพื่อแทนค่าที่เหมาะสมลงในฟิลด์ของ descriptor. descriptor จะพร้อมใช้งานโดย SQL เพื่อแทนตัวทำเครื่องหมายพารามิเตอร์ด้วยค่าที่แท้จริง.

เมื่อคุณใช้ SQLDA สำหรับอินพุตในคำสั่ง OPEN โดยใช้ USING DESCRIPTOR clause, คุณไม่จำเป็นต้องระบุค่าฟิลด์ทั้งหมด. โดยเฉพาะอย่างยิ่ง, SQLDAID, SQLRES, และ SQLNAME สามารถปล่อยว่างไว้ได้ (SQLNAME สามารถตั้งค่าได้ถ้าจำเป็นต้องใช้ค่าของ CCSID) ดังนั้น, เมื่อคุณใช้วิธีนี้เพื่อแทนตัวทำเครื่องหมายพารามิเตอร์ด้วยค่าเหล่านี้, คุณจำเป็นต้องกำหนด:

- จำนวนตัวทำเครื่องหมายที่ต้องมี
- ชนิดข้อมูลและแอ็ตทริบิวต์ของตัวทำเครื่องหมายพารามิเตอร์เหล่านี้ (SQLTYPE, SQLLEN, และ SQLNAME)
- จำเป็นต้องมีตัวแปรตัวบ่งชี้หรือไม่

นอกจากนั้น, ถ้ารูทีนจัดการทั้งคำสั่ง SELECT และ non-SELECT, คุณอาจต้องกำหนดประเภทของคำสั่งด้วย.

ถ้าแอ็พพลิเคชันของคุณใช้ตัวทำเครื่องหมายพารามิเตอร์, โปรแกรมของคุณจะต้องทำตามขั้นตอนต่อไปนี้. ซึ่งสามารถทำได้โดยใช้ SQLDA หรือ descriptor ที่จัดสรรแล้ว.

1. อ่านคำสั่งลงในตัวแปรโฮสต์ชื่อ DSTRING ซึ่งเป็นสตริงอักขระแบบความยาวแปรผัน .
2. หาค่าตัวเลขของเครื่องหมายพารามิเตอร์.
3. จัดสรรขนาดของ SQLDA หรือใช้ ALLOCATE DESCRIPTOR เพื่อจัดสรร descriptor ด้วยจำนวนของ entry. ไม่สามารถใช้ได้ใน REXX
4. สำหรับ SQLDA, ให้ตั้งค่า SQLN และ SQLD เป็นจำนวนของตัวทำเครื่องหมายพารามิเตอร์. SQLN ไม่สามารถใช้ได้ใน REXX สำหรับ descriptor ที่จัดสรรแล้ว, ให้ใช้ SET DESCRIPTOR เพื่อตั้งค่า COUNT entry เป็นจำนวนของตัวทำเครื่องหมายพารามิเตอร์.
5. สำหรับ SQLDA, ให้ตั้งค่า SQLDABC เท่ากับ SQLN*LENGTH(SQLVAR) + 16. ไม่สามารถใช้ได้ใน REXX
6. สำหรับตัวทำเครื่องหมายพารามิเตอร์:
   a. หาชนิดข้อมูล, ความยาว, และตัวบ่งชี้.
   b. สำหรับ SQLDA, ให้ตั้งค่า SQLTYPE และ SQLLEN สำหรับตัวทำเครื่องหมายพารามิเตอร์. สำหรับ descriptor ที่จัดสรรแล้ว, ให้ใช้ SET DESCRIPTOR เพื่อตั้งค่า entry สำหรับ TYPE, LENGTH, PRECISION, และ SCALE สำหรับตัวทำเครื่องหมายพารามิเตอร์แต่ละตัว.
   c. สำหรับ SQLDA, ให้จัดสรรหน่วยเก็บเพื่อจัดการกับค่าอินพุต.
   d. สำหรับ SQLDA, ให้ตั้งค่าเหล่านี้ในหน่วยเก็บ.
   e. สำหรับ SQLDA, ให้ตั้งค่า SQLDATA และ SQLIND (ถ้าสามารถใช้ได้) สำหรับตัวทำเครื่องหมายพารามิเตอร์แต่ละตัว. สำรหับ descriptor ที่จัดสรรแล้ว, ให้ใช้ SET DESCRIPTOR เพื่อตั้งค่า entry สำหรับ DATA และ INDICATOR (ถ้าสามารถใช้งานได้) สำหรับตัวทำเครื่องหมายพารามิเตอร์แต่ละตัว.
   f. ถ้ามีการใช้ตัวแปรอักขระ และมีค่าของ CCSID ที่ไม่ใช้ดีฟอลต์งาน CCSID, หรือตัวแปรกราฟิกที่ถูกใช้ และมี CCSID ที่ไม่ใช่ DBCS CCSID ที่ถูกเชื่อมโยงสำหรับงาน CCSID,
      - สำหรับ SQLDA ให้ตั้งค่า SQLNAME (SQLCCSID ใน REXX) ตามนั้น
      - สำหรับ SQL descriptor ที่จัดสรรแล้ว, ให้ใช้ SET DESCRIPTOR เพื่อตั้งค่า DB2_CCSID.
   g. ใช้คำสั่ง OPEN พร้อมด้วย USING DESCRIPTOR clause (สำหรับ SQLDA) หรือ USING SQL DESCRIPTOR clause (สำหรับ descriptor ที่จัดสรรแล้ว) เพื่อเปิดเคอร์เซอร์ และแทนค่าของตัวทำเครื่องหมายพารามิเตอร์แต่ละตัว.

จากนั้นจะนำข้อความมาประมวลผลได้ตามปกติ.

สิ่งอ้างอิงที่เกี่ยวข้อง

"ตัวอย่าง: คำสั่ง SELECT เพื่อจัดสรรหน่วยเก็บสำหรับ SQLDA" ในหน้า 275
สมมติว่า แอ็พพลิเคชันของคุณจำเป็นต้องจัดการกับคำสั่ง SELECT แบบ dynamic เมื่อค่าหนึ่งเปลี่ยนเป็นอีกค่าหนึ่งสำหรับใช้ต่อไป คำสั่งนี้สามารถอ่านได้จากจอแสดงผล, ซึ่งถูกส่งผ่านจากแอ็พพลิเคชันอื่น, หรือถูกสร้างขึ้นจากแอ็พพลิเคชันของคุณในขณะปฏิบัติงาน.

"ตัวอย่าง: คำสั่ง SELECT ที่ใช้ SQL descriptor ที่ถูกจัดสรรแล้ว" ในหน้า 280
สมมติว่า แอ็พพลิเคชันของคุณจำเป็นต้องจัดการกับคำสั่ง SELECT แบบ dynamic เมื่อค่าหนึ่งเปลี่ยนเป็นอีกค่าหนึ่งสำหรับใช้ต่อไป คำสั่งนี้สามารถอ่านได้จากจอแสดงผล ซึ่งถูกส่งผ่านจากแอ็พพลิเคชันอื่น หรือถูกสร้างขึ้นจากแอ็พพลิเคชันของคุณแบบ dynamic

การใช้ SQL แบบโต้ตอบ

SQL แบบโต้ตอบอนุญาตให้โปรแกรมเมอร์ หรือผู้ดูแลฐานข้อมูล มีความรวดเร็วและสะดวกในการกำหนด, อัพเดต, ลบ, หรือสำรวจข้อมูลเพื่อทำการทดสอบ วิเคราะห์ปัญหา และการดูแลฐานข้อมูล

โปรแกรมเมอร์, ที่ใช้ SQL แบบโต้ตอบ, สามารถแทรกแถวลงยังตารางและทดสอบข้อความ SQL ก่อนทำการรันข้อความเหล่านั้นในแอ็พพลิเคชันโปรแกรม. ผู้บริหารระบบฐานข้อมูลสามารถใช้ SQL แบบโต้ตอบเพื่อให้ privilege หรือ เรียกคืน privilege, สร้างหรือลบแบบแผน, ตาราง, หรือมุมมอง, หรือเลือกข้อมูลจากตารางแค็ตตาล็อกระบบ.

หลังจากข้อความ SQL แบบโต้ตอบถูกรัน, ข้อความแสดงการเสร็จสิ้นหรือข้อความแสดงข้อผิดพลาดจะปรากฏ. นอกจากนี้, โดยปกติแล้ว ข้อความแสดงสถานะจะปรากฏขึ้นระหว่างข้อความที่รันเป็นเวลานาน.

คุณสามารถดูคำอธิบายเกี่ยวกับข้อความได้โดยเลื่อนเคอร์เซอร์ไปไว้บนข้อความและกด F1 (Help)

ฟังก์ชันพื้นฐานของ SQL แบบโต้ตอบคือ:

- ฟังก์ชัน**entry ข้อความ** อนุญาตให้คุณ:
  - พิมพ์ข้อความ SQL แบบโต้ตอบและรันข้อความ.
  - เรียกข้อความออกมาและแก้ไขข้อความ.
  - พร็อมต์สำหรับข้อความ SQL.
  - เลื่อนไปยังข้อความ(คำสั่ง) และ ข้อความ (แสดงผล) ก่อนหน้านี้.
  - เรียกเซอร์วิสเซสชัน.
  - เรียกฟังก์ชันการเลือกรายการ.
  - ออกจาก SQL แบบโต้ตอบ.
- ฟังก์ชัน **พร้อมต์** อนุญาตให้คุณพิมพ์ ข้อความ SQL หรือข้อความ SQL บางส่วน ให้กด F4 (Prompt) และ คุณจะถูกถามให้ใส่ไวยากรณ์ของข้อความ นอกจากนี้คุณยังสามารถกด F4 เพื่อรับเมนูของคำสั่ง SQL ที่สนับสนุนได้ด้วย จากเมนูนี้, คุณสามารถเลือกข้อความและจะถูกถามให้ใส่ไวยากรณ์ของข้อความ.
- ฟังก์ชัน **การเลือกรายการ** อนุญาตให้คุณเลือกฐานข้อมูลเชิงสัมพันธ์, แบบแผน, ตาราง, มุมมอง, คอลัมน์, ข้อจำกัด หรือ SQL แพ็กเกจ จากรายการตามที่คุณมีสิทธิ

  รายการที่คุณเลือกจากรายการอาจนำมาแทรกลงในข้อความ SQL ที่ตำแหน่งของเคอร์เซอร์.
- ฟังก์ชัน **session services** อนุญาตให้คุณ:
  - เปลี่ยนแอ็ตทริบิวต์เซสชัน.
  - พิมพ์เซสชันปัจจุบัน.
  - ย้าย entry ทั้งหมดออกจากเซสชันปัจจุบัน.
  - บันทึกเซสชันในซอร์สไฟล์.

  หลักการที่เกี่ยวข้อง

  “แอ็พพลิเคชัน Dynamic SQL” ในหน้า 267

  Dynamic SQL อนุญาตให้แอ็พพลิเคชันกำหนด และรันคำสั่ง SQL ที่เวลารันไทม์ของโปรแกรม แอ็พพลิเคชันที่ใช้ dynamic SQL จะยอมรับคำสั่ง SQL เป็นอินพุต หรือสร้างคำสั่ง SQL ในรูปของสตริงอักขระ แอ็พพลิเคชันไม่จำเป็นต้องทราบชนิดของ คำสั่ง SQL

  สิ่งอ้างอิงที่เกี่ยวข้อง

"การประมวลผลคำสั่ง non-SELECT" ในหน้า 268
ก่อนที่จะสร้างคำสั่ง SQL non-SELECT แบบไดนามิก คุณจำเป็นต้องตรวจสอบว่าคำสั่ง SQL นี้สามารถรันแบบไดนามิกได้

การเริ่มต้นใช้งาน SQL แบบโต้ตอบ

เมื่อต้องการเริ่มต้นใช้งาน SQL แบบโต้ตอบ ให้ป้อน STRSQL จาก บรรทัดรับคำสั่ง i5/OS

จอแสดงผล Enter SQL Statements จะปรากฏขึ้นมา. นี่คือจอแสดงผลหลัก SQL แบบโต้ตอบ. จากจอแสดงผลนี้, คุณสามารถใส่คำสั่ง SQL และใช้:

- F4=prompt
- F13=Session services
- F16=Select collections
- F17=Select tables
- F18=Select columns

```
                           Enter SQL Statements

Type SQL statement, press Enter.
    มีการเชื่อมต่อปัจจุบันกับฐานข้อมูลเชิงสัมพันธ์ rdjacque.
===>_____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________

                                                                      Bottom
F3=Exit   F4=Prompt   F6=Insert line   F9=Retrieve   F10=Copy line
F12=Cancel            F13=Services     F24=More keys
```

กด F24=มีคีย์อื่นๆ เพื่อดูคีย์ฟังก์ชันที่เหลือ.

```
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________

                                                          Bottom

F14=Delete line   F15=Split line     F16=Select collections (libraries)
F17=Select tables                    F18=Select columns   F24=More keys
          (ไฟล์)                              (ฟิลด์)
```

หมายเหตุ: หากคุณกำลังใช้หลักการตั้งชื่อระบบ, ชื่อในวงเล็บจะปรากฏขึ้นมาแทนชื่อที่แสดงอยู่ด้านบน.

เซสชันแบบโต้ตอบประกอบด้วย:

- ค่าพารามิเตอร์ที่คุณระบุไว้สำหรับคำสั่ง STRSQL .
- ข้อความ SQL ที่คุณป้อนลงในเซสชันพร้อมกับข้อความโต้ตอบข้อความ SQL นั้น
- ค่าของพารามิเตอร์ใดๆ ที่คุณเปลี่ยนโดยใช้ฟังก์ชันเซอร์วิสเซสชัน
- รายการที่คุณเลือก

SQL แบบโต้ตอบมี session-ID แบบเฉพาะที่ประกอบด้วย user ID ของคุณและ workstation station ID ปัจจุบัน. session-ID นี้อนุญาตให้ผู้ใช้จำนวนมากที่มี user ID เดียวกันเข้าใช้งาน SQL แบบโต้ตอบจากเวิร์กสเตชันได้มากกว่าหนึ่งเวิร์กสเตชันในเวลาเดียวกัน. นอกจากนี้, สามารถรันเซสชัน SQL แบบโต้ตอบได้มากกว่าหนึ่งเซสชันได้จากเวิร์กสเตชันเดียวกันในเวลาเดียวกันจาก user ID เดียวกัน.

หากมีเซสชัน SQL และเซสชันนั้นถูกป้อนซ้ำ, พารามิเตอร์ใดๆ ที่ระบุไว้บนคำสั่ง STRSQL ถูกละเลย. พารามิเตอร์จากเซสชัน SQL ที่มีอยู่ถูกใช้งาน.

สิ่งอ้างอิงที่เกี่ยวข้อง

Start SQL Interactive Session (STRSQL) statement

การใช้ฟังชัน entry คำสั่ง

ฟังก์ชัน entry คำสั่ง คือฟังก์ชันที่คุณป้อนเข้าไปครั้งแรก เมื่อเลือก SQL แบบโต้ตอบ คุณจะกลับไปยัง entry คำสั่งหลังจากประมวลผลคำสั่ง SQL แบบโต้ตอบแต่ละอัน

ในฟังก์ชัน entry คำสั่ง, คุณจะพิมพ์หรือพร้อมต์สำหรับคำสั่ง SQL ทั้งหมดและส่งคำสั่งนั้นไปเพื่อการประมวลผลโดยกดคีย์ Enter.

คำสั่งที่คุณพิมพ์บนบรรทัดรับคำสั่งอาจยาวหนึ่งบรรทัดหรือมากกว่า. คุณสามารถพิมพ์หมายเหตุที่วงเล็บไว้ (`/**/`) ใน SQL แบบโต้ตอบ. อย่างไรก็ตาม, คุณไม่ควรใช้หมายเหตุธรรมดา (นั่นคือ, หมายเหตุที่เริ่มต้นด้วย `--`) ใน SQL แบบโต้ตอบ เนื่องจากหมายเหตุเหล่านี้จะประกอบด้วย ส่วนที่เหลือของคำสั่ง SQL ภายในหมายเหตุนั้น. เมื่อคำสั่งถูกประมวลผล, คำสั่งและข้อความผลลัพธ์จะถูกย้ายขึ้นไปด้านบนบนจอแสดงผล. หลังจากนั้นคุณสามารถป้อนคำสั่งอื่นได้.

หาก SQL พบข้อผิดพลาดด้านไวยากรณ์ของคำสั่งที่ป้อนเข้ามา ข้อความผลลัพธ์ (ข้อผิดพลาดด้านไวยากรณ์) จะถูก ย้ายไปด้านบนของจอแสดงผล. ในส่วนอินพุต, จะมีสำเนาคำสั่งพร้อมด้วยเคอร์เซอร์ที่วางอยู่ที่ข้อผิดพลาดทางไวยากรณ์. คุณสามารถวางเคอร์เซอร์ไว้บนข้อความ และกด F1=Help เพื่อดูข้อมูลเพิ่มเติมเกี่ยวกับข้อผิดพลาดได้.

คุณสามารถเลื่อนหน้าไปยังข้อความ (คำสั่ง), คำสั่ง, และข้อความ (แสดงผล) ก่อนหน้านี้ได้. ถ้าคุณกด F9=Retrieve ในขณะที่เคอร์เซอร์อยู่บนบรรทัดใส่คำสั่ง, คำสั่งก่อนหน้านี้จะถูกคัดลอกไว้ในพื้นที่อินพุต. กด F9 อีกครั้งจะทำให้เกิดการย้อนขึ้นไปอีกหนึ่งคำสั่ง และคัดลอกคำสั่งนั้นไว้ในพื้นที่อินพุต. การกด F9 ต่อไปทำให้คุณย้อนขึ้นไปเรื่อยๆทีละคำสั่งก่อนหน้าจนกระทั่งคุณพบคำสั่งที่คุณต้องการ. หากคุณต้องการเนื้อที่เพิ่มสำหรับพิมพ์คำสั่ง SQL ,ให้เลื่อนจอแสดงผลลง.

การออกคำสั่ง (Prompting)

ฟังก์ชันพร็อมต์ช่วยให้คุณหาข้อมูลที่จำเป็นสำหรับไวยากรณ์ของข้อความที่คุณต้องการใช้ ฟังก์ชันพร้อมต์อาจถูกใช้ในโหมดการประมวลผลคำสั่งเหล่านี้: *RUN, *VLD, และ *SYN. การออกคำสั่ง (Prompting) ไม่สามารถใช้งานได้สำหรับข้อความ SQL ทั้งหมด และไม่สมบูรณ์ สำหรับข้อความ SQL จำนวนมาก

คุณมีสองอ็อพชันเมื่อใช้งานตัวพร้อมต์:

- พิมพ์กิริยาของข้อความก่อนกด F4=Prompt.

  คำสั่งจะถูกวิเคราะห์และ clause ที่สมบูรณ์จะถูกกรอกในจอแสดงผลพร้อมต์.

  หากคุณพิมพ์ SELECT และกด F4=Prompt, จอแสดงผลต่อไปนี้จะปรากฏ:

```
                          Specify SELECT Statement

Type SELECT statement information.  Press F4 for a list.

  FROM tables . . . . . . . . _____________________________________________
  SELECT columns  . . . . . . _____________________________________________
  WHERE conditions  . . . . . _____________________________________________
  GROUP BY columns  . . . . . _____________________________________________
  HAVING conditions . . . . . _____________________________________________
  ORDER BY columns  . . . . . _____________________________________________
  FOR UPDATE OF columns . . .  _____________________________________________

                                                                      Bottom
Type choices, press Enter.

  DISTINCT rows in result table  . . . . . .  N   Y=Yes, N=No
  UNION with another SELECT  . . . . . . . .  N   Y=Yes, N=No
  Specify additional options . . . . . . . .  N   Y=Yes, N=No



F3=Exit         F4=Prompt   F5=Refresh   F6=Insert line   F9=Specify subquery
F10=Copy line   F12=Cancel  F14=Delete line  F15=Split line  F24=More keys
```

- กด F4=Prompt ก่อนพิมพ์ข้อมูลใดๆ ลงบนจอแสดงผล Enter SQL Statements. คุณจะเห็นรายการข้อความ. รายการข้อความจะต่างกันไปและขึ้นอยู่กับโหมดการประมวลผลข้อความ SQL แบบโต้ตอบในปัจจุบัน. สำหรับโหมดการตรวจสอบไวยากรณ์ด้วยภาษาอื่นนอกเหนือจาก *NONE, รายการจะรวมข้อความ SQL ทั้งหมดเข้าไว้ด้วย. สำหรับโหมดรันพร้อมตรวจสอบความถูกต้องนั้น, เฉพาะข้อความที่ถูกรันใน SQL แบบโต้ตอบเท่านั้นจะถูกแสดง. คุณสามารถเลือกจำนวนของข้อความที่คุณต้องการใช้งานได้. ระบบจะถามข้อความที่คุณเลือก.

  หากคุณกด F4=Prompt โดยไม่พิมพ์อะไร, จอแสดงผลจะแสดง:

```
                          เลือกข้อความ SQL

เลือกรายการใดรายการหนึ่งต่อไปนี้:

     1. ALTER TABLE
     2. CALL
     3. COMMENT ON
     4. COMMIT
     5. CONNECT
     6. CREATE ALIAS
     7. CREATE COLLECTION
     8. CREATE INDEX
     9. CREATE PROCEDURE
    10. CREATE TABLE
    11. CREATE VIEW
    12. DELETE
    13. DISCONNECT
    14. DROP ALIAS
                                                              อื่นๆ...
รายการที่เลือก
     __

F3=Exit   F12=Cancel
```

หากคุณกด F21=Display Statement บนจอแสดงผลพร็อมต์, ตัวพร็อมต์จะแสดงผลข้อความ SQL ที่ถูกฟอร์แม็ตเหมือนกับที่กรอกไว้ที่ตัวพร็อมต์.

เมื่อกด Enter ภายในการออกคำสั่ง, ข้อความซึ่งถูกสร้างด้วยหน้าจอพร็อมต์จะถูกแทรกลงยังเซสชัน. หากโหมดการประมวลผลข้อความคือ *RUN, ข้อความจะถูกรัน. ตัวพร็อมต์ยังคงอยู่ในการควบคุมหากพบข้อผิดพลาด.

การตรวจสอบไวยากรณ์:

ไวยากรณ์ของคำสั่ง SQL จะถูกตรวจสอบ เมื่อใส่ไวยากรณ์ในตัวพร้อมต์.

ตัวพร้อมต์ไม่รับคำสั่งที่ผิดพลาดทางไวยากรณ์. คุณต้องแก้ไขไวยากรณ์ หรือลบส่วนของคำสั่งที่ไม่ถูกต้อง มิฉะนั้นจะไม่อนุญาตให้ออกคำสั่ง.

โหมดการประมวลผลคำสั่ง:

สามารถเลือกโหมดการประมวลผลคำสั่งบนจอแสดงผล Change Session Attributes.

ในโหมด *RUN (run) หรือ *VLD (validate), เฉพาะคำสั่งที่อนุญาตให้รันใน SQL แบบโต้ตอบที่สามารถถามได้. ในโหมด *SYN (syntax check), ข้อความ SQL ทั้งหมดได้รับอนุญาต. ที่จริงแล้ว คำสั่งไม่ได้ถูกรันในโหมด *SYN หรือ *VLD; เฉพาะไวยากรณ์และอ็อบเจ็กต์จะถูกตรวจสอบ.

เคียวรี่ย่อย:

สามารถเลือกเคียวรี่ย่อยบนจอแสดงผลใดๆ ที่มี WHERE หรือ HAVING clause.

หากต้องการดูจอแสดงผลเคียวรี่ย่อย, โปรดกด F9=Specify subquery เมื่อเคอร์เซอร์อยู่ที่แถวอินพุต WHERE หรือ HAVING. จอแสดงผลจะแสดงผลให้คุณพิมพ์ข้อมูลการเลือกแบบย่อย. หากเคอร์เซอร์อยู่ภายในวงเล็บของเคียวรี่ย่อย เมื่อกด F9, ข้อมูลของเคียวรี่ย่อยจะถูกแสดงในจอผลถัดไป. หากเคอร์เซอร์อยู่นอกวงเล็บของเคียวรี่ย่อย, จอแสดงผลถัดไปจะว่างเปล่า.

การเรียกทำงาน CREATE TABLE:

คุณสามารถป้อนแต่ละ definition ของคอลัมน์ เมื่อคุณได้รับพร้อมต์สำหรับคำสั่ง CREATE TABLE

วางเคอร์เซอร์ของคุณในส่วน definition คอลัมน์ของจอแสดงผล, และกด F4=Prompt. จอแสดงผลที่มีเนื้อที่สำหรับการป้อนข้อมูลทั้งหมดสำหรับ definition ของหนึ่งคอลัมน์จะปรากฏขึ้นมา.

หากต้องการป้อนชื่อคอลัมน์ที่มีความยาวมากกว่า 18 อักขระ, ให้กด F20=Display entire name. หน้าต่างที่มีเนื้อที่พอสำหรับชื่อขนาด 30 อักขระจะปรากฏขึ้นมา.

คีย์แก้ไข, F6=Insert line, F10=Copy line, และ F14=Delete line, สามารถใช้งานเพื่อเพิ่ม และลบ entry ในรายการ definition คอลัมน์.

การป้อนข้อมูล DBCS:

กฎสำหรับการประมวลผลข้อมูล double-byte character set (DBCS) บนหลายๆ แถว เป็นกฎเดียวกันกับที่อยู่บนจอแสดงผล Enter SQL Statements และในตัวพร้อมต์ SQL

แต่ละแถวจะมีหมายเลขอักขระบนและล่างเดียวกัน. เมื่อทำการประมวลผลสตริงข้อมูล DBCS ซึ่งต้องการแถวข้อมูลมากกว่าหนึ่งแถวสำหรับป้อน, อักขระพิเศษบนและล่างจะถูกลบออกไป. หากคอลัมน์สุดท้ายบนแถวข้อมูลมีอักขระบนและคอลัมน์แรกของแถวข้อมูลถัดไปมีอักขระล่าง, อักขระบนและล่างจะถูกลบออกปโดยตัวพร็อมต์เมื่อแถวข้อมูลทั้งสองถูกแปลภาษา. หากสองคอลัมน์สุดท้ายของแถวข้อมูลมีอักขระบนที่ตามด้วยช่องว่าง และคอลัมน์แรกของแถวข้อมูลถัดไปมีอักขระบน, อักขระล่าง, ช่องว่าง, การเรียงอักขระล่างถูกลบออกไปเมื่อแถวข้อมูลถูกแปลภาษา. การลบนี้อนุญาตให้ข้อมูล DBCS ถูกอ่านเป็นสตริงอักขระต่อเนื่องได้.

ในตัวอย่าง, สมมติว่าป้อนเงื่อนไข WHERE ลงไป. อักขระด้านบน จะปรากฏขึ้นในส่วนสตริงเริ่มต้นและสิ้นสุดบนแต่ละแถวข้อมูลสองแถว.

```
                        Specify SELECT Statement

Type SELECT statement information.  Press F4 for a list.

  FROM tables . . . . . .  TABLE1_____________________________________
  SELECT columns  . . . . .  *_____________________________________
  WHERE conditions  . . . . .  COL1 = '<AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQ>
                           <RRSS>'_____________________________________
  GROUP BY columns  . . . . .  _____________________________________
  HAVING conditions . . . . .  _____________________________________
  ORDER BY columns  . . . . .  _____________________________________
  FOR UPDATE OF columns . . .  _____________________________________
```

เมื่อกด Enter, สตริงอักขระจะถูกดึงรวมกันไว้, โดยจะลบอักขระเสริมด้านบนออก. คำสั่งจะมีลักษณะเช่นนี้บนจอแสดงผล Enter SQL Statements:

```
SELECT * FROM TABLE1 WHERE COL1 = '<AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQRRSS>'
```

การใช้ฟังก์ชันรายการที่เลือก

คุณสามารถเข้าถึงฟังก์ชันการเลือกรายการ โดยกด F4 (Prompt) บนจอแสดงผลพร้อมต์ในบางจอ ในการเข้าถึงฟังก์ชันบนหน้าจอ Enter SQL Statements ให้กด F16 (Select collections), F17 (Select tables), หรือ F18 (Select columns)

หลังจากกดคีย์ฟังก์ชัน, คุณจะได้รับรายการฐานข้อมูลเชิงสัมพันธ์ที่ได้รับสิทธิ์, แบบแผน, ตาราง, มุมมอง, alias, คอลัมน์, ข้อจำกัด, โพรซีเดอร์, พารามิเตอร์, หรือแพ็กเกจที่ได้รับสิทธิ์ให้เลือก. หากคุณร้องขอรายการตาราง, แต่คุณไม่ได้เลือกแบบแผนไว้ก่อน, คุณจะถูกถามให้เลือกแบบแผนก่อน.

ในรายการ, คุณสามารถเลือกไอเท็มได้หนึ่งไอเท็มหรือมากกว่า, โดยระบุลำดับที่คุณต้องการให้ปรากฏในข้อความด้วยตัวเลข. เมื่อฟังก์ชันรายการออกจากการทำงาน, รายการที่คุณเลือกจะถูกแทรกลงที่ตำแหน่งที่วางเคอร์เซอร์บนจอแสดงผลที่คุณออกมา.

โปรดเลือกรายการที่คุณสนใจเป็นหลักเสมอ. ตัวอย่างเช่น หากคุณต้องการรายการคอลัมน์ แต่คุณเชื่อว่าคอลัมน์ที่คุณต้องการอยู่ในตารางที่ไม่ได้เลือกในปัจจุบัน ให้กด F18 แล้ว, จากรายการคอลัมน์,ให้กด F17 เพื่อเปลี่ยนตาราง. หากรายการตารางถูกเลือกเป็นอันดับแรก, ชื่อตารางจะถูกแทรกเข้าไปยังข้อความของคุณ. คุณจะไม่มีตัวเลือกสำหรับการเลือกคอลัมน์.

คุณสามารถร้องขอรายการเมื่อใดก็ได้ขณะพิมพ์ข้อความ SQL บนจอแสดงผล Enter SQL Statements. รายการที่คุณเลือกจากรายการจะถูกแทรกลงบนจอแสดงผล Enter SQL Statements. ข้อความจะถูกแทรกลงในที่ตำแหน่งที่เคอร์เซอร์วางอยู่ตามลำดับหมายเลขที่คุณระบุไว้บนจอแสดงผลรายการ. แม้ว่าข้อมูลของรายการที่คุณเลือกจะถูกเพิ่มเข้าไปแล้ว, คุณต้องพิมพ์คีย์เวิร์ดสำหรับข้อความด้วย.

ฟังก์ชันรายการพยายามจัดหาคุณสมบัติที่จำเป็นสำหรับคอลัมน์, ตาราง, และ SQL แพ็กเกจที่เลือก. อย่างไรก็ตาม, บางครั้งฟังก์ชันรายการจะไม่สามารถกำหนดจุดประสงค์ของข้อความ SQL ได้. คุณต้องตรวจสอบข้อความ SQL และตรวจสอบว่าคอลัมน์, ตาราง, และ SQL แพ็กเกจที่เลือกมีคุณสมบัติถูกต้อง.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การเริ่มต้นใช้งาน SQL แบบโต้ตอบ" ในหน้า 286
เมื่อต้องการเริ่มต้นใช้งาน SQL แบบโต้ตอบ ให้ป้อน STRSQL จาก บรรทัดรับคำสั่ง i5/OS

ตัวอย่าง: การใช้ฟังก์ชันรายการที่เลือก:

ตัวอย่างนี้แสดงวิธีการใช้ฟังก์ชัน รายการที่เลือก เพื่อสร้างคำสั่ง SELECT

สมมติว่า คุณมี:

- เพียงแค่ป้อน SQL แบบโต้ตอบโดยการพิมพ์ STRSQL บนบรรทัดรับคำสั่ง i5/OS.
- ไม่ต้องเลือกรายการหรือป้อน entry.
- เลือก *SQL เพื่อดูหลักการตั้งชื่อ.

หมายเหตุ: ตัวอย่างจะแสดงรายการที่ไม่อยู่บนเซิร์ฟเวอร์ของคุณ. รายการเหล่านั้นจะถูกใช้เพื่อเป็นตัวอย่างเท่านั้น.

เริ่มต้นใช้ SQL statements:

1. พิมพ์ SELECT บนแถว entry แรกของข้อความ.

2. พิมพ์ FROM บนแถว entry ที่สองของข้อความ.
3. ปล่อยเคอร์เซอร์ไว้หลังคำว่า FROM.

```
                          Enter SQL Statements

Type SQL statement, press Enter.
===> SELECT
     FROM _
```

4. กด F17=Select tables เพื่อดูรายการตาราง, เพราะคุณต้องใส่ชื่อตารางตามหลัง FROM.

   แทนที่รายการตารางจะปรากฏตามที่คุณคาดไว้, รายการคอลเล็คชันจะแสดง (จอแสดงผลคอลเล็คชัน Select and Sequence). คุณเพิ่งป้อนเซสชัน SQL และไม่ได้เลือกแบบแผนที่จะทำงานด้วย
5. พิมพ์ 1 ในคอลัมน์*Seq* ถัดจากแบบแผน YOURCOLL2.

```
                     Select and Sequence Collections


พิมพ์หมายเลขลำดับ (1-999) เพื่อเลือกคอลเล็คชัน, กด Enter.

ลำดับ   คอลเล็คชัน       ประเภท   ข้อความ
       YOURCOLL1         SYS     ผลประโยชน์ของบริษัท
1      YOURCOLL2         SYS     ข้อมูลส่วนตัวพนักงาน
       YOURCOLL3         SYS     การแบ่งประเภทงาน/ข้อกำหนดของงาน
       YOURCOLL4         SYS     การประกันภัยบริษัท
```

6. กด Enter.

   จอแสดงผล Select and Sequence Tables จะปรากฏขึ้นมา, โดยจะแสดงตารางที่อยู่ในแบบแผน YOURCOLL2.
7. พิมพ์ 1 ในคอลัมน์ *Seq* ถัดจากตาราง PEOPLE.

```
                      Select and Sequence Tables

พิมพ์หมายเลขลำดับ (1-999) เพื่อเลือกตาราง, กด Enter.

ลำดับ  ตาราง                  คอลเล็คชัน    ประเภท  ข้อความ
     EMPLCO                 YOURCOLL2     TAB   ข้อมูลบริษัทของพนักงาน
1    PEOPLE                 YOURCOLL2     TAB   ข้อมูลส่วนบุคคลของพนักงาน
     EMPLEXP                YOURCOLL2     TAB   ประสบการณ์พนักงาน
     EMPLEVL                YOURCOLL2     TAB   รายงานการประเมินผลพนักงาน
     EMPLBEN                YOURCOLL2     TAB   ระเบียนข้อมูลผลประโยชน์พนักงาน
     EMPLMED                YOURCOLL2     TAB   ระเบียนข้อมูลทางการแพทย์ของพนักงาน
     EMPLINVST              YOURCOLL2     TAB   ระเบียนข้อมูลการลงทุนของพนักงาน
```

8. กด Enter.

   จอแสดงผล Enter SQL Statements จะปรากฏขึ้นอีกครั้งพร้อมด้วยชื่อตาราง, YOURCOLL2.PEOPLE, จะถูกแทรกลงหลังจากFROM. ชื่อตารางจะถูกคัดเลือกโดยชื่อแบบแผนในหลักการตั้งชื่อ *SQL.

```
                          Enter SQL Statements

Type SQL statement, press Enter.
===> SELECT
     FROM YOURCOLL2.PEOPLE _
```

9. วางเคอร์เซอร์หลังจาก SELECT.
10. กด F18=Select columns เพื่อดูรายการคอลัมน์, เพราะคุณต้องให้ชื่อคอลัมน์ตามหลัง SELECT.

    จอแสดงผล Select and Sequence Columns จะปรากฏขึ้นมา, โดยจะแสดงตารางที่อยู่ในตาราง PEOPLE.
11. พิมพ์ 2 ในคอลัมน์ *Seq* ถัดจากคอลัมน์*NAME*.
12. พิมพ์ 1 ในคอลัมน์ *Seq* ถัดจากคอลัมน์ *SOCSEC*.

```
                       Select and Sequence Columns

พิมพ์หมายเลขลำดับ (1-999) เพื่อเลือกคอลัมน์, กด Enter.

ลำดับ  คอลัมน์              ตาราง                ประเภท          ดิจิต  ความยาว
2    NAME                  PEOPLE               CHARACTER                6
     EMPLNO                PEOPLE               CHARACTER               30
1    SOCSEC                PEOPLE               CHARACTER               11
     STRADDR               PEOPLE               CHARACTER               30
     CITY                  PEOPLE               CHARACTER               20
     ZIP                   PEOPLE               CHARACTER                9
     PHONE                 PEOPLE               CHARACTER               20
```

13. กด Enter.

    จอแสดงผล Enter SQL Statements จะปรากฏขึ้นมาพร้อมด้วย SOCSEC, NAME ซึ่งจะปรากฏหลังคำว่า SELECT.

```
                          Enter SQL Statements

Type SQL statement, press Enter.
===> SELECT SOCSEC, NAME
     FROM YOURCOLL2.PEOPLE
```

14. กด Enter.

    ข้อความที่คุณสร้างจะถูกรันในขณะนี้.

เมื่อคุณใช้งานฟังก์ชันรายการ, ค่าที่คุณเลือกไว้จะยังใช้งานได้อยู่จนกว่าคุณจะเปลี่ยนมันหรือจนกว่าคุณจะเปลี่ยนรายการของแบบแผนบนจอแสดงผล Change Session Attributes.

รายละเอียดเซอร์วิสเซสชัน

คุณสามารถเปลี่ยนแอ็ตทริบิวต์ของเซสชันได้จากจอแสดงผล Session Services นอกจากนี้คุณยังสามารถพิมพ์ ลบ หรือบันทึกเซสชันให้กับ ซอร์สไฟล์ได้อีกด้วย

ในการเข้าถึงจอแสดงผล Session Services ให้กด F13 (Services) บนจอแสดงผล Enter SQL Statements

Option 1 (เปลี่ยนแอ็ตทริบิวต์เซสชัน) จะแสดงจอแสดงผล Change Session Attributes, ซึ่งอนุญาตให้คุณเลือกค่าปัจจุบันซึ่งยังใช้งานได้อยู่สำหรับเซสชัน SQL แบบโต้ตอบ. อ็อพชันที่แสดงอยู่บนจอแสดงผลนี้จะเปลี่ยนไปตามอ็อพชันการประมวลผลของข้อความที่เลือก.

แอ็ตทริบิวต์เซสชันต่อไปนี้อาจถูกเปลี่ยนได้:

- แอ็ตทริบิวต์ Commitment control
- การควบคุมการประมวลผลข้อความ
- อุปกรณ์เอาต์พุต SELECT
- รายการแบบแผน
- ประเภทรายการสำหรับเลือกระบบของคุณ และ SQL อ็อบเจ็กต์ทั้งหมด หรือเฉพาะ SQL อ็อบเจ็กต์
- ข้อมูลจะรีเฟรชอ็อพชันเมื่อแสดงผลข้อมูล
- อ็อพชันอนุญาตให้ทำสำเนาข้อมูล
- อ็อพชันการตั้งชื่อ
- ภาษาโปรแกรม
- รูปแบบวันที่
- รูปแบบเวลา
- ตัวแยกวันที่
- ตัวแยกเวลา
- การแทนค่าจุดทศนิยม
- อักขระคั่นสตริง SQL
- ลำดับการจัดเรียง
- ตัวระบุภาษา (language identifier)
- กฎ SQL
- อ็อพชันรหัสผ่าน CONNECT

อ็อพชัน 2 (Print current session) จะเข้าใช้งานจอแสดงผล Change Printer ซึ่งจะให้คุณพิมพ์เซสชันปัจจุบันได้ทันทีและทำงานต่อไป คุณจะถูกถามให้ใส่ข้อมูลเครื่องพิมพ์ ข้อความ SQL ทั้งหมดที่คุณป้อนและข้อความที่แสดงผลจะถูกสั่งพิมพ์ออกมาเหมือนตอนที่ข้อความเหล่านั้นปรากฏบนจอแสดงผล Enter SQL Statements

อ็อพชัน 3 (Remove all entries from current session) จะให้คุณลบข้อความ SQL และข้อความจากจอแสดงผล Enter SQL Statements และประวัติเซสชัน คุณจะถูกถามเพื่อความแน่ใจว่าคุณต้องการลบข้อมูลนั้นจริงๆ

อ็อพชัน 4 (Save session in source file) จะเข้าใช้งานจอแสดงผล Change Source File ซึ่งให้คุณบันทึกเซสชันในซอร์สไฟล์ คุณจะถูกถามชื่อซอร์สไฟล์ ฟังก์ชันนี้จะให้คุณฝังซอร์สไฟล์ลงในโปรแกรมภาษาโฮสต์โดยการใช้ source entry utility (SEU)

หมายเหตุ: อ็อพชัน 4 จะให้คุณฝังข้อความ SQL ที่เป็นต้นแบบในโปรแกรมภาษาชั้นสูง (HLL) ที่ใช้งาน SQL ซอร์สไฟล์ที่ถูกสร้างโดยอ็อพชัน 4 อาจถูกแก้ไข และใช้เป็นซอร์สไฟล์อินพุตสำหรับคำสั่ง Run SQL Statements (RUNSQLSTM)

SQL แบบโต้ตอบที่มีอยู่

ในการเข้าถึงสภาพแวดล้อม SQL แบบโต้ตอบ ให้กด F3 (Exit) บนจอแสดงผล Enter SQL Statements คุณมีหลายอ็อพชันสำหรับการออก

- บันทึกและออกจากเซสชัน. ปล่อย SQL แบบโต้ตอบไว้. เซสชันปัจจุบันของคุณจะถูกบันทึกและใช้งานในครั้งต่อไปที่คุณเริ่มใช้งาน SQL แบบโต้ตอบ.
- ออกจากเซสชันโดยไม่ต้องบันทึกเซสชัน. ปล่อย SQL แบบโต้ตอบไว้โดยไม่ต้องบันทึกเซสชันของคุณ.
- เรียกเซสชันกลับสู่การทำงาน. ยังอยู่ใน SQL แบบโต้ตอบและกลับสู่จอแสดงผล Enter SQL Statements. พารามิเตอร์เซสชันปัจจุบันยังคงใช้งานได้อยู่.
- บันทึกเซสชันในซอร์สไฟล์. บันทึกเซสชันปัจจุบันในซอร์สไฟล์ จอแสดงผล Change Source File จะปรากฏขึ้นมาเพื่อให้คุณเลือกที่ที่จะบันทึกเซสชัน. คุณไม่สามารถเรียกคืนและทำงานกับเซสชันนี้ได้อีกใน SQL แบบโต้ตอบ.

หมายเหตุ:

1. อ็อพชัน 4 จะให้คุณฝังข้อความ SQL ต้นแบบในโปรแกรมภาษาชั้นสูง (HLL) ที่ใช้งาน SQL. ใช้งาน source entry utility (SEU) เพื่อทำสำเนาข้อความลงยังโปรแกรมของคุณ. สามารถแก้ไขซอร์ไฟล์ และถูกใช้เป็นซอร์สไฟล์อินพุตสำหรับคำสั่ง Run SQL Statements (RUNSQLSTM).
2. หากมีการเปลี่ยนแปลงของแถวและมีการล็อกงานนี้และคุณพยายามจะออกจาก SQL แบบโต้ตอบ, จะมีข้อความแจ้งเตือนปรากฏขึ้นมา.

การใช้เซสชัน SQL ที่มีอยู่

หากคุณบันทึกเซสชัน SQL แบบโต้ตอบเซสชัน โดยการเลือกอ็อพชัน 1 (เซสชัน Save and exit) บนจอแสดงผล Exit Interactive SQL คุณอาจต้องเรียกเซสชันนั้นกลับสู่การทำงานใหม่ที่เวิร์กสเตชันใดก็ได้

อย่างไรก็ตาม, หากคุณใช้อ็อพชัน 1 เพื่อบันทึกเซสชันสองเซสชันหรือมากกว่าบนเวิร์กสเตชันที่ต่างกัน, SQL แบบโต้ตอบจะพยายามเรียกเซสชันที่ตรงกับเวิร์กสเตชันของคุณให้กลับมาเริ่มต้นทำงานใหม่ก่อน. หากไม่มีเซสชันใดตรงกัน, SQL แบบโต้ตอบจะเพิ่มขอบเขตของการค้นหาในครอบคลุมถึงเซสชันทั้งหมดที่เป็นของ user ID คุณ. หากไม่มีเซสชันสำหรับ user ID ของคุณ, ระบบจะสร้างเซสชันใหม่สำหรับ user ID ของคุณและเวิร์กสเตชันปัจจุบัน.

ตัวอย่างเช่น, คุณได้บันทึกเซสชันไว้บนเวิร์กสเตชัน 1 และบันทึกอีกเซสชันหนึ่งไว้บนเวิร์กสเตชัน 2 และคุณยังคงทำงานอยู่บนเวิร์กสเตชัน 1. SQL แบบโต้ตอบจะพยายามเรียกเซสชันที่บันทึกไว้สำหรับเวิร์กสเตชัน 1 กลับมาทำงานใหม่. หากเซสชันนั้นกำลังอยู่ระหว่างใช้งาน, SQL แบบโต้ตอบจะพยายามเรียกเซสชันที่บันทึกไว้สำหรับเวิร์กสเตชัน 2 . หากเซสชันนั้นอยู่ระหว่างใช้งานเช่นกัน, ระบบจะสร้างเซสชันที่สองสำหรับเวิร์กสเตชัน 1.

อย่างไรก็ตาม สมมติว่าคุณกำลังทำงานที่เวิร์กสเตชัน 3 และต้องการ ใช้งานเซสชัน ISQL ที่เชื่อมโยงกับเวิร์กสเตชัน 2 คุณอาจต้องลบ เซสชันจากเวิร์กสเตชัน 1 เป็นลำดับแรกโดยใช้อ็อพชัน 2 (Exit without saving session) บนจอแสดงผล Exit Interactive SQL

การกู้คืนเซสชัน SQL

หากเซสชัน SQL ก่อนหน้านี้หยุดทำงานแบบผิดปกติ SQL แบบโต้ตอบจะปรากฏจอแสดงผล Recover SQL Session ตอนเริ่มต้นทำงาน ของเซสชันถัดไป เมื่อป้อนคำสั่ง Start SQL Interactive Session (STRSQL)

จากจอแสดงผลนี้, คุณสามารถเลือกเพื่อทำหนึ่งในสิ่งต่อไปนี้.

- กู้คืนเซสชันเดิมด้วยการเลือกอ็อพชัน 1 (พยายามเรียกเซสชัน SQL ที่มีอยู่เดิมกลับสู่การทำงานใหม่).

- ลบเซสชันเดิมและเริ่มต้นเซสชันใหม่ด้วยการเลือกอ็อพชัน 2 (ลบเซสชัน SQL ที่มีอยู่เดิมและเรียกใช้งานเซสชันใหม่).

หากคุณเลือกที่จะลบเซสชันเดิมออกและทำงานกับเซสชันใหม่ต่อไป, พารามิเตอร์ที่คุณระบุไว้เมื่อคุณป้อน STRSQL จะถูกใช้งาน. หากคุณเลือกที่จะกู้คืนเซสชันเดิม, หรือกำลังป้อนเซสชันที่บันทึกไว้ก่อนหน้า, พารามิเตอร์ที่คุณระบุไว้เมื่อคุณป้อน STRSQL จะถูกละเลยและพารามิเตอร์จากเซสชันเดิมจะถูกใช้งาน. ข้อความจะถูกส่งกลับเพื่อระบุว่าพารามิเตอร์ใดถูกเปลี่ยนไปจากค่าที่ระบุไว้ของค่าเซสชันเดิม.

การเข้าใช้งานฐานข้อมูลแบบรีโมตด้วย SQL แบบโต้ตอบ

ใน SQL แบบโต้ตอบ, คุณสามารถสื่อสารกับฐานข้อมูลเชิงสัมพันธ์แบบรีโมตด้วยการใช้ข้อความ SQL CONNECT . SQL แบบโต้ตอบจะใช้ซีแมนทิกส์ CONNECT (Type 2) (หน่วยงานแบบกระจาย) สำหรับคำสั่ง CONNECT.

SQL แบบโต้ตอบเชื่อมต่อกับฐานข้อมูลเชิงสัมพันธ์แบบโลคัลเมื่อเริ่มต้นใช้งานเซสชัน SQL เมื่อคำสั่ง CONNECT เสร็จสมบูรณ์, ข้อความจะแสดงการเชื่อมต่อของฐานข้อมูลเชิงสัมพันธ์ซึ่งถูกสร้างขึ้นมา. หาก SQL แบบโต้ตอบเริ่มเซสชันใหม่และไม่มีการระบุ COMMIT(*NONE) หรือหาก SQL แบบโต้ตอบเรียกคืนเซสชันที่บันทึกไว้ และระดับ commitment control ที่บันทึกไว้กับเซสชันไม่ใช่ *NONE การเชื่อมต่อก็จะถูกลงทะเบียนด้วย commitment control การเชื่อมต่อโดยนัยและการลงทะเบียน commitment control ที่เป็นไปได้ อาจส่งผลต่อการเชื่อมต่อในลำดับต่อมากับฐานข้อมูลแบบรีโมต ขอแนะนำให้คุณทำงานใดงานหนึ่งดังต่อไปนี้ ก่อนที่จะเชื่อมต่อกับฐานข้อมูลแบบรีโมต:

- เมื่อทำการเชื่อมต่อกับแอ็พพลิเคชันเซิร์ฟเวอร์ที่ไม่รองรับหน่วยการทำงานแบบกระจาย จะต้องมีการใช้คำสั่ง RELEASE ALL นำหน้าคำสั่ง COMMIT เพื่อสิ้นสุดการเชื่อมต่อครั้งก่อนหน้า รวมถึงการเชื่อมต่อโดยนัยกับฐานข้อมูลโลคัล
- เมื่อคุณทำการเชื่อมต่อกับแอ็พพลิเคชันเซิร์ฟเวอร์ที่ไม่ใช่ DB2 for i5/OS ให้ใช้คำสั่ง RELEASE ALL นำหน้าคำสั่ง COMMIT เพื่อสิ้นสุดการเชื่อมต่อ ครั้งก่อนหน้า รวมถึงการเชื่อมต่อโดยนัยไปยังฐานข้อมูลโลคัล และ เปลี่ยนระดับ commitment control ให้เป็น *CHG เป็นอย่างน้อย

เมื่อคุณทำการเชื่อมต่อกับแอ็พพลิเคชันเซิร์ฟเวอร์ที่ ไม่ใช่ DB2 for i5/OS เซสชันแอ็ตทริบิวต์บางตัวถูกเปลี่ยนไปเป็นแอ็ตทริบิวต์ ซึ่งถูกรองรับโดยแอ็พพลิเคชันเซิร์ฟเวอร์นั้น ตารางต่อไปนี้จะแสดงแอ็ตทริบิวต์ที่เปลี่ยนไป.

ตารางที่ 49. ตารางค่า

เซสชันแอ็ตทริบิวต์	ค่าต้นฉบับ	ค่าใหม่
รูปแบบวันที่	*YMD *DMY *MDY *JUL	*ISO *EUR *USA *USA
รูปแบบเวลา	*HMS ด้วยตัวแยก : *HMS ด้วยตัวแยกอื่นๆ	*JIS *EUR

ตารางที่ 49. ตารางค่า (ต่อ)

เซสชันแอ็ตทริบิวต์	ค่าต้นฉบับ	ค่าใหม่
Commitment control	*CHG, *NONE *ALL	*CS Repeatable Read
หลักการตั้งชื่อ	*SYS	*SQL
อนุญาตให้ทำสำเนาข้อมูล	*NO, *YES	*OPTIMIZE
รีเฟรชข้อมูล	*ALWAYS	*FORWARD
จุดทศนิยม	*SYSVAL	*PERIOD
เรียงลำดับ	ค่าใดๆ ที่นอกเหนือจาก *HEX	*HEX

หมายเหตุ: เมื่อทำการเชื่อมต่อกับ DB2 สำหรับ Linux®, UNIX® และ Windows® หรือ DB2 สำหรับแอ็พพลิเคชันเซิร์ฟเวอร์ z/OS® รูปแบบวันที่และเวลา ที่ระบุไว้ต้องเป็นรูปแบบเดียวกัน

หลังจากเชื่อมต่อสมบูรณ์แล้ว, ข้อความจะถูกส่งไปเพื่อระบุว่าเซสชันแอ็ตทริบิวต์ถูกเปลี่ยนไป. สามารถแสดงผลเซสชันแอ็ตทริบิวต์ที่เปลี่ยนไปด้วยการใช้จอแสดงผลเซอร์วิสเซสชัน. ขณะรัน SQL แบบโต้ตอบ, จะไม่สามารถสร้างการเชื่อมต่ออื่นสำหรับ activation group ที่เป็นค่าดีฟอลต์.

เมื่อเชื่อมต่อกับระบบรีโมตด้วย SQL แบบโต้ตอบ, โหมดการประมวลผลข้อความเฉพาะไวยากรณ์จะตรวจสอบไวยากรณ์ของข้อความกับไวยากรณ์ที่ระบบโลคัลสนับสนุน ไม่ใช่ไวยากรณ์ที่ระบบรีโมตสนับสนุน. อย่างคล้ายคลึงกัน, ตัวพร็อมต์ SQL และตัวสนับสนุนรายการจะใช้ไวยากรณ์ข้อความและหลักการตั้งชื่อซึ่งสนับสนุนโดยระบบโลคัล. อย่างไรก็ตาม, ข้อความจะถูกรัน, บนระบบรีโมต. เนื่องจากมีความแตกต่างในระดับของการสนับสนุน SQL ระหว่างทั้งสองระบบ, จึงอาจพบข้อผิดพลาดทางไวยากรณ์ในข้อความบนระบบรีโมตขณะอยู่ในรันไทม์.

รายการแบบแผนและตารางจะมีอยู่เมื่อคุณถูกเชื่อมต่อกับฐานข้อมูลเชิงสัมพันธ์แบบโลคัล. รายการคอลัมน์มีอยู่เฉพาะเมื่อคุณถูกเชื่อมต่อกับตัวจัดการฐานข้อมูลเชิงสัมพันธ์ที่สนับสนุนข้อความ DESCRIBE TABLE.

เมื่อคุณออกจาก SQL แบบโต้ตอบด้วยการเชื่อมต่อที่มีการเปลี่ยนแปลงค้างอยู่ในการเชื่อมต่อหรือการเชื่อมต่อนั้นใช้การสนทนา (ระหว่างโปรแกรม) แบบป้องกัน, การเชื่อมต่อจะยังคงทำงานอยู่ต่อไป. หากคุณไม่ทำงานอื่นๆ ขณะเชื่อมต่อ, การเชื่อมต่อจะสิ้นสุดในระหว่างการดำเนินงาน COMMIT หรือ ROLLBACK ถัดไป. คุณสามารถยุติการเชื่อมต่อได้โดยการใช้งาน RELEASE ALL และ COMMIT ก่อนออกจาก SQL แบบโต้ตอบ.

การใช้ SQL แบบโต้ตอบสำหรับการเข้าใช้งานแบบรีโมตกับแอ็พพลิเคชันเซิร์ฟเวอร์ ที่ไม่ใช่ DB2 for i5/OS อาจ ต้องการการตั้งค่าบางอย่าง

หมายเหตุ: ในเอาต์พุตของการติดตามการสื่อสาร, อาจมีการอ้างอิงถึงข้อความ 'CREATE TABLE XXX'. วิธีนี้ใช้เพื่อกำหนดการมีอยู่ของแพ็กเกจ;เป็นส่วนหนึ่งของการประมวลผลแบบปกติ, และอาจถูกละเลยได้.

หลักการที่เกี่ยวข้อง

โปรแกรมมิงฐานข้อมูลแบบกระจาย

สิ่งอ้างอิงที่เกี่ยวข้อง

“การจำแนกประเภทของการเชื่อมต่อ” ในหน้า 317
เมื่อการเชื่อมต่อ SQL แบบรีโมตเกิดขึ้น ระบบจะใช้การเชื่อมต่อระบบเครือข่ายแบบป้องกันหรือแบบไม่ได้ป้องกันอย่างใดอย่างหนึ่ง

การใช้ตัวประมวลผลคำสั่ง SQL

ตัวประมวลผลคำสั่ง SQL ยอมให้คำสั่ง SQL ทำงานได้จากรายการ ต้นฉบับย่อย หรือไฟล์ stream ต้นฉบับ คำสั่งในต้นฉบับสามารถรันซ้ำ หรือเปลี่ยนได้โดยไม่ต้องคอมไพล์ต้นฉบับ ซึ่งทำให้การตั้งค่าสภาวะแวดล้อมฐานข้อมูลง่ายดายขึ้น.

ตัวประมวลผลคำสั่ง SQL จะใช้งานได้โดยการใช้คำสั่ง Run SQL Statements (RUNSQLSTM)

คำสั่งที่ใช้ได้กับตัวประมวลผลคำสั่ง SQL ได้แก่:

- ALTER FUNCTION
- ALTER PROCEDURE
- ALTER SEQUENCE
- ALTER TABLE
- CALL
- COMMENT ON
- COMMIT
- CREATE ALIAS
- CREATE DISTINCT TYPE
- CREATE FUNCTION
- CREATE INDEX
- CREATE PROCEDURE
- CREATE SCHEMA
- CREATE SEQUENCE
- CREATE TABLE
- CREATE TRIGGER
- CREATE VIEW
- DECLARE GLOBAL TEMPORARY TABLE
- DELETE
- DROP
- GRANT
- INSERT
- LABEL ON
- LOCK TABLE

- REFRESH TABLE
- RELEASE SAVEPOINT
- RENAME
- REVOKE
- ROLLBACK
- SAVEPOINT
- SET CURRENT DECFLOAT ROUNDING MODE
- SET CURRENT DEGREE
- SET ENCRYPTION PASSWORD
- SET PATH
- SET SCHEMA
- SET TRANSACTION
- UPDATE

ในต้นฉบับ คำสั่งจะไม่ขึ้นต้นด้วย EXEC SQL แต่ละคำสั่งลงท้ายด้วยเซมิโคลอน สำหรับรายการต้นฉบับย่อย ระยะขอบดีฟอลต์ คือ 80 หากความยาวเร็กคอร์ดของรายการต้นฉบับย่อยเกินกว่า 80 ระบบจะอ่านได้เฉพาะอักขระ 80 ตัวแรกเท่านั้น คุณสามารถเปลี่ยนระยะขอบด้านขวาให้เป็น ค่าอื่น โดยใช้พารามิเตอร์ MARGINS บนคำสั่ง RUNSQLSTM สำหรับไฟล์ stream ต้นฉบับ ไฟล์ทั้งหมดจะถูกอ่าน และไม่มีการใช้ระยะขอบ

หมายเหตุในต้นฉบับอาจเป็นหมายเหตุแบบแถวหรือหมายเหตุแบบบล็อก หมายเหตุแบบแถวเริ่มต้นด้วยยติภังค์คู่ (--) และสิ้นสุดที่ท้ายแถว. หมายเหตุแบบบล็อกเริ่มต้นด้วย /* และมีต่อไปหลายแถวจนกว่าจะถึง */ ถัดไป. หมายเหตุแบบบล็อกสามารถซ้อนภายในได้. เฉพาะคำสั่ง SQL และ หมายเหตุเท่านั้นที่มีได้ในไฟล์ต้นฉบับ. การแสดงรายการเอาต์พุตและข้อความที่เป็นผลลัพธ์สำหรับคำสั่ง SQL จะถูกส่งไปยังไฟล์พิมพ์. ไฟล์พิมพ์ที่เป็นดีฟอลต์ คือ QSYSPRT.

ในการดำเนินการตรวจสอบไวยากรณ์เฉพาะกับคำสั่งทั้งหมด ในรายการต้นฉบับ โปรดระบุพารามิเตอร์ PROCESS(*SYN) ในคำสั่ง RUNSQLSTM หากต้องการดู รายละเอียดเพิ่มเติมสำหรับข้อความแสดงความผิดพลาดในรายการ ให้ระบุพารามิเตอร์ SECLVLTXT(*YES)

สิ่งอ้างอิงที่เกี่ยวข้อง

คำสั่ง Run SQL Statement (RUNSQLSTM)

การรันคำสั่งหลังเกิดข้อผิดพลาด

เมื่อคำสั่งเกิดข้อผิดพลาดที่มีค่าความรุนแรง สูงกว่าค่าที่ระบุในพารามิเตอร์ระดับความผิดพลาด (ERRLVL) ของคำสั่ง Run SQL Statements (RUNSQLSTM) แสดงว่าคำสั่งล้มเหลว

คำสั่งที่เหลือในต้นฉบับจะถูกวิเคราะห์คำเพื่อตรวจสอบข้อผิดพลาดของไวยากรณ์, และจะไม่ถูกรัน. ข้อผิดพลาดของ SQL ส่วนใหญ่มีค่าความรุนแรงระดับ 30. หากคุณต้องการดำเนินการประมวลผลต่อ หลังจากคำสั่ง SQL ล้มเหลว, ให้ตั้งค่าพารามิเตอร์ ERRLVL ของคำสั่ง RUNSQLSTM เป็น 30 หรือสูงกว่า. คำสั่ง DROP จะออกข้อผิดพลาดของความรุนแรงระดับ 20 หากไม่พบอ็อบเจ็กต์ที่จะลบ. การตั้งค่าพารามิเตอร์ ERRLVL ให้มีค่าเป็น 20 จะอนุญาตให้คุณข้ามข้อผิดพลาดของคำสั่ง DROP ในขณะที่ไม่อนุญาตให้การประมวลผลดำเนินการต่อ หากเป็นข้อผิดพลาดที่มีความรุนแรงสูงกว่า.

commitment control ในตัวประมวลผลคำสั่ง SQL

ระดับ commitment-control ถูกระบุใน คำสั่ง Run SQL Statements (RUNSQLSTM)

หากมีการระบุ ระดับ commitment-control อื่นที่ไม่ใช่ *NONE, คำสั่ง SQL จะรันภายใต้ commitment control. หากคำสั่งทั้งหมดดำเนินการสำเร็จ, COMMIT จะดำเนินการเมื่อตัวประมวลผลคำสั่ง SQL เสร็จสิ้น. มิฉะนั้น, จะดำเนินการ ROLLBACK. ระบบจะถือว่าคำสั่งสำเร็จหากค่าความรุนแรงของโค้ดที่ได้ต่ำกว่าหรือเท่ากับค่าที่ระบุในพารามิเตอร์ ERRLVL ของคำสั่ง RUNSQLSTM.

คำสั่ง SET TRANSACTION สามารถใช้ภายในรายการต้นฉบับย่อยเพื่อแทนที่ระดับ commitment control ที่ระบุในคำสั่ง RUNSQLSTM.

หมายเหตุ: งานจะต้องอยู่ที่ยูนิตของขอบเขตงานเพื่อใช้ตัวประมวลผลคำสั่ง SQL กับ commitment control.

การแสดงรายการต้นฉบับสำหรับตัวประมวลผลคำสั่ง SQL

| ตัวอย่างต่อไปนี้แสดงรายการต้นฉบับสำหรับตัวประมวลผลคำสั่ง SQL

หมายเหตุ: ด้วยการใช้โค้ดตัวอย่าง, คุณตกลงในเงื่อนไขของ “สิทธิในรหัส และข้อมูลถ้อยแถลง” ในหน้า 353

```
xxxxSS1 VxRxMx yymmdd         Run SQL Statements         SCHEMA      02/15/08 15:35:18   Page   1
Source file..............CORPDATA/SRC
Member..................SCHEMA
Commit..................*NONE
Naming..................*SYS
Generation level.........10
Date format..............*JOB
Date separator...........*JOB
Right margin.............80
Time format..............*HMS
Time separator ..........*JOB
Default collection.......*NONE
IBM SQL flagging.........*NOFLAG
ANS flagging.............*NONE
Decimal point............*JOB
Sort sequence............*JOB
Language ID..............*JOB
Printer file.............*LIBL/QSYSPRT
Source file CCSID.........65535
| Job CCSID.................37
Statement processing......*RUN
Allow copy of data........*OPTIMIZE
| Allow blocking............*ALLREAD
SQL rules.................*DB2
Decimal result options:
  Maximum precision.......31
  Maximum scale...........31
  Minimum divide scale....0
Source member changed on 11/01/07  11:54:10
```

รูปที่ 1. รายการ QSYSPRT listing สำหรับตัวประมวลผลคำสั่ง SQL

```
xxxxSS1 VxRxMx yymmdd         Run SQL Statements         SCHEMA      02/15/08 15:35:18   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
    1
    2     DROP SCHEMA DEPT;
    3     DROP SCHEMA MANAGER;
    4
    5     CREATE SCHEMA DEPT
    6          CREATE TABLE EMP (EMPNAME CHAR(50), EMPNBR INT)
    7                            -- EMP will be created in schema DEPT
    8          CREATE INDEX EMPIND ON EMP(EMPNBR)
    9                           -- EMPIND will be created in DEPT
   10          GRANT SELECT ON EMP TO PUBLIC;  -- grant authority
   11
   12     INSERT INTO DEPT/EMP VALUES('JOHN SMITH', 1234);
   13                             /* table must be qualified since no
   14                                longer in the schema */
   15
   16     CREATE SCHEMA AUTHORIZATION MANAGER
   17                              -- this schema will use MANAGER's
   18                              -- user profile
   19           CREATE TABLE EMP_SALARY (EMPNBR INT, SALARY DECIMAL(7,2),
   20                                    LEVEL CHAR(10))
   21           CREATE VIEW LEVEL AS SELECT EMPNBR, LEVEL
   22                FROM EMP_SALARY
   23           CREATE INDEX SALARYIND ON EMP_SALARY(EMPNBR,SALARY)
   24
   25           GRANT ALL ON LEVEL TO JONES GRANT SELECT ON EMP_SALARY TO CLERK
   26                    -- Two statements can be on the same line
* * * * *  E N D  O F  S O U R C E  * * * * *
```

รูปที่ 2. รายการ QSYSPRT สำหรับตัวประมวลผลคำสั่ง SQL (ต่อ)

```
xxxxSS1 VxRxMx yymmdd      Run SQL Statements          SCHEMA          02/15/08 15:35:18   Page   3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
MSG ID  SEV  RECORD  TEXT
SQL7953   0       1  Position 1 Drop of DEPT in QSYS complete.
SQL7953   0       3  Position 3 Drop of MANAGER in QSYS complete.
SQL7952   0       5  Position 3 Schema DEPT created.
SQL7950   0       6  Position 8 Table EMP created in DEPT.
SQL7954   0       8  Position 8 Index EMPIND created in DEPT on table EMP in
                     DEPT.
SQL7966   0      10  Position 8 GRANT of authority to EMP in DEPT completed.
SQL7956   0      10  Position 40 1 rows inserted in EMP in DEPT.
SQL7952   0      13  Position 28 Schema MANAGER created.
SQL7950   0      19  Position 9 Table EMP_SALARY created in schema
                     MANAGER.
SQL7951   0      21  Position 9 View LEVEL created in MANAGER.
SQL7954   0      23  Position 9 Index SALARYIND created in MANAGER on table
                     EMP_SALARY in MANAGER.
SQL7966   0      25  Position 9 GRANT of authority to LEVEL in MANAGER
                     completed.
SQL7966   0      25  Position 37 GRANT of authority to EMP_SALARY in MANAGER
                     completed.

Message Summary
Total    Info    Warning    Error    Severe    Terminal
   13     13          0        0         0           0
00 level severity errors found in source
* * * * *  E N D  O F  L I S T I N G  * * * * *
```

รูปที่ 3. รายการ QSYSPRT สำหรับตัวประมวลผลคำสั่ง SQL (ต่อ)

ฟังก์ชันของฐานข้อมูลเชิงสัมพันธ์แบบกระจายและ SQL

ฐานข้อมูลเชิงสัมพันธ์แบบกระจาย ประกอบไปด้วยชุด SQL อ็อบเจ็กต์ที่กระจายอยู่บนระบบคอมพิวเตอร์ที่เชื่อมต่อถึงกันและกัน.

ฐานข้อมูลเชิงสัมพันธ์เหล่านี้อาจเป็นชนิดเดียวกัน (ตัวอย่างเช่น ฐานข้อมูล DB2 for i5/OS) หรือต่างชนิดกัน (DB2 สำหรับ z/OS, DB2 สำหรับ VSE และ VM หรือระบบจัดการฐานข้อมูลแบบ non-IBM ที่ สนับสนุน Distributed Relational Database Architecture (DRDA)) แต่ละฐานข้อมูลเชิงสัมพันธ์นี้จะมีตัวจัดการฐานข้อมูลเชิงสัมพันธ์เพื่อจัดการกับตารางในสภาวะแวดล้อมของมันเอง. ตัวจัดการฐานข้อมูลทำการติดต่อและทำงานร่วมกับตัวจัดการฐานข้อมูลอื่นโดยที่จะอนุญาตให้ตัวจัดการข้อมูลที่มีอยู่สามารถรันคำสั่ง SQL ที่อยู่บนฐานข้อมูลเชิงสัมพันธ์ในระบบอื่นได้.

Application requester จะสนับสนุนการเชื่อมต่อของแอ็พพลิเคชัน. แอ็พพลิเคชันเซิร์ฟเวอร์จะเป็นฐานข้อมูลแบบโลคัลหรือแบบรีโมตให้กับ application requester ที่ถูกเชื่อมต่อ DB2 for i5/OS สนับสนุน DRDA เพื่อให้ application requester สามารถสื่อสารกับแอ็พพลิเคชันเซิร์ฟเวอร์ได้ นอกจากนั้น DB2 for i5/OS ยังสามารถเรียกร้องให้โปรแกรมทางออกอนุญาตการเข้าไปใช้ข้อมูล ระบบจัดการฐานข้อมูลอื่นๆ ที่ไม่สนับสนุน DRDA โปรแกรมทางออกเหล่านี้ถูกเรียกว่า *โปรแกรม application requester driver (ARD)*

DB2 for i5/OS สนับสนุน ฐานข้อมูลเชิงสัมพันธ์แบบกระจายซึ่งเป็นระดับต่อไปนี้:

- Remote unit of work (RUW)

หน่วยการทำงานแบบรีโมต คือ สถานการณ์ที่การเตรียมการและการรันคำสั่ง SQL หลายๆ คำสั่ง เกิดขึ้นในหนึ่งแอ็พพลิเคชันเซิร์ฟเวอร์เท่านั้นในช่วงของหนึ่งหน่วยการทำงาน DB2 for i5/OS สนับสนุน RUW บน Advanced Program-to-Program Communication (APPC) หรือ TCP/IP อย่างใดอย่างหนึ่ง

- Distributed unit of work (DUW)

  หน่วยการทำงานแบบกระจาย คือ สถานการณ์ที่การเตรียมการและการรันคำสั่ง SQL สามารถเกิดขึ้นได้ในหลายๆ แอ็พพลิเคชันเซิร์ฟเวอร์ในช่วงของหนึ่งหน่วยการทำงาน อย่างไรก็ตาม, คำสั่ง SQL แบบเดี่ยวสามารถอ้างอิงถึงได้เฉพาะอ็อบเจ็กต์ที่อยู่ในแอ็พพลิเคชันเซิร์ฟเวอร์แบบเดี่ยวเท่านั้น. DB2 for i5/OS สนับสนุน DUW บน APPC และ DUW บน TCP/IP

หลักการที่เกี่ยวข้อง

"บทนำสู่ DB2 for i5/OS Structured Query Language" ในหน้า 4
Structured Query Language (SQL) เป็นภาษามาตรฐาน สำหรับการกำหนดและจัดการข้อมูลในฐานข้อมูลเชิงสัมพันธ์ หัวข้อ เหล่านี้อธิบายถึงการปรับใช้ SQL ของ System i โดย ใช้ฐานข้อมูล DB2 for i5/OS และ ไลเซนส์โปรแกรม IBM DB2 Query Manager and SQL Development Kit for i5/OS

"SQL แพ็กเกจ" ในหน้า 14
SQL แพ็กเกจ คืออ็อบเจ็กต์ที่มีโครงสร้างควบคุมซึ่งเกิดขึ้นเมื่อมีการเชื่อมโยงคำสั่ง SQL ในแอ็พพลิเคชันโปรแกรมเข้ากับระบบจัดการฐานข้อมูลเชิงสัมพันธ์แบบรีโมต (DBMS).

โปรแกรมมิงฐานข้อมูลแบบกระจาย

DB2 for i5/OS การสนับสนุนฐานข้อมูลเชิงสัมพันธ์แบบกระจาย

ไลเซนส์โปรแกรม IBM DB2 Query Manager and SQL Development Kit for i5/OS สนับสนุนการเข้าถึงแบบโต้ตอบของฐานข้อมูลแบบกระจาย

DB2 Query Manager and SQL Development Kit for i5/OS จัดเตรียมการสนับสนุนฐานข้อมูลเชิงสัมพันธ์ ด้วยคำสั่ง SQL ต่อไปนี้:

- CONNECT
- SET CONNECTION
- DISCONNECT
- RELEASE
- DROP PACKAGE
- GRANT PACKAGE
- REVOKE PACKAGE

การสนับสนุนเพิ่มเติมจะมาจาก development kit ผ่านทางพารามิเตอร์ที่อยู่บนคำสั่งพรีคอมไพเลอร์ของ SQL:

- คำสั่งสร้าง SQL ILE C Object (CRTSQLCI)
- คำสั่งสร้าง SQL ILE C++ Object (CRTSQLCPPI)
- คำสั่งสร้าง SQL COBOL Program (CRTSQLCBL)
- คำสั่งสร้าง SQL ILE COBOL Object (CRTSQLCBLI)
- คำสั่งสร้าง SQL PL/I Program (CRTSQLPLI)
- คำสั่งสร้าง SQL RPG Program (CRTSQLRPG)

- คำสั่งสร้าง SQL ILE RPG Object (CRTSQLRPGI)

งานที่เกี่ยวข้อง

การเตรียมและรันโปรแกรมด้วยคำสั่ง SQL

สิ่งอ้างอิงที่เกี่ยวข้อง

"DB2 for i5/OSคำอธิบายของคำสั่ง CL" ในหน้า 353

DB2 for i5/OS จัดหา คำสั่ง CL เหล่านี้สำหรับ SQL

CONNECT (Type 1)

CONNECT (Type 2)

DISCONNECT

DROP

GRANT (Package Privileges)

REVOKE (Package Privileges)

RELEASE (Connection)

SET CONNECTION

DB2 for i5/OS โปรแกรมตัวอย่างฐานข้อมูลเชิงสัมพันธ์แบบกระจาย

โปรแกรมตัวอย่างฐานข้อมูลเชิงสัมพันธ์ที่ใช้ หน่วยการทำงานแบบรีโมตถูกรวมไว้กับผลิตภัณฑ์ SQL มีไฟล์และรายการย่อยหลายตัวอยู่ในไลบรารี QSQL เพื่อช่วยในการจัดเตรียมสภาวะแวดล้อมในการรันโปรแกรมตัวอย่าง DB2 for i5/OS แบบกระจาย

ในการใช้ไฟล์และรายการย่อยเหล่านี้, จำเป็นที่จะต้องรันงานแบ็ตช์ SETUP ที่อยู่ในไฟล์ QSQL/QSQSAMP. งานแบ็ตช์ SETUP อนุญาตให้คุณปรับแต่งตัวอย่างตามความต้องการเพื่อใช้ในการ:

- สร้างไลบรารี QSQSAMP ที่ตำแหน่ง local และ remote.
- ตั้งค่า directory entry ของฐานข้อมูลเชิงสัมพันธ์ที่ตำแหน่งโลคัลและรีโมต.
- สร้างแอ็พพลิเคชันพาเนลที่ตำแหน่ง local.
- ทำพรีคอมไพล์, คอมไพล์ และการรันโปรแกรมเพื่อที่จะสร้างแบบแผน, ตาราง, ดรรชนี และมุมมองของแอ็พพลิเคชันตัวอย่างแบบกระจาย
- โหลดข้อมูลเข้าตารางที่ตำแหน่ง local และ remote.
- โปรแกรมแบบ พรีคอมไพล์และ คอมไพล์.
- สร้าง SQL แพ็กเกจที่ตำแหน่งรีโมตสำหรับแอ็พพลิเคชันโปรแกรม.
- ทำพรีคอมไพล์, คอมไพล์, และรันโปรแกรมเพื่อที่จะอัพเดตคอลัมน์ตำแหน่งในตารางของแผนก.

ก่อนการรัน SETUP, อาจจำเป็นที่จะต้องแก้ไขรายการย่อยของ SETUP ที่อยู่ในไฟล์ QSQL/QSQSAMP. คำสั่งจะรวมอยู่ในรายการย่อยในรูปของความคิดเห็น. ในการรัน SETUP, ให้ระบุคำสั่งต่อไปนี้บนบรรทัดรับคำสั่ง:

```
========> SBMDBJOB QSQL/QSQSAMP SETUP
```

รอให้งานแบ็ตช์เสร็จสิ้น.

ในการใช้โปรแกรมตัวอย่าง ให้ระบุคำสั่งต่อไปนี้บนบรรทัดรับคำสั่ง:

```
========> ADDLIBLE QSQSAMP
```

ในการเรียกจอแสดงผลแรกที่อนุญาตให้คุณปรับแต่งโปรแกรมตัวอย่างได้ตามความต้องการ ให้ระบุคำสั่งต่อไปนี้บนบรรทัดรับคำสั่ง

```
========> CALL QSQ8HC3
```

ระบบจะแสดงหน้าจอต่อไปนี้ จากจอแสดงผลนี้ คุณสามารถปรับแต่งโปรแกรมตัวอย่างฐานข้อมูลได้ตามความต้องการ

```
SAMPLE ORGANIZATION APPLICATION


ACTION...........:   _      A  (ADD)                  E (ERASE)
                            D  (DISPLAY)              U (UPDATE)

OBJECT...........:   __     DE (DEPARTMENT)           EM (EMPLOYEE)
                            DS (DEPT STRUCTURE)

SEARCH CRITERIA..:   __     DI (DEPARTMENT ID)     MN (MANAGER NAME)
                            DN (DEPARTMENT NAME)   EI (EMPLOYEE ID)
                            MI (MANAGER ID)        EN (EMPLOYEE NAME)

LOCATION.........:   _____________  (BLANK IMPLIES LOCAL LOCATION)

DATA.............:




                                                        Bottom

F3=Exit
```

การสนับสนุนการใช้งานแพ็กเกจของ SQL

ระบบปฏิบัติการ i5/OS สนับสนุน SQL แพ็กเกจ ชนิดของอ็อบเจ็กต์คือ *SQLPKG

แพ็กเก็จ SQL ประกอบไปด้วยโครงสร้างควบคุมและแผนการเข้าไปใช้ข้อมูลที่จำเป็นในการประมวลผลคำสั่ง SQL บนแอ็พพลิเคชันเซิร์ฟเวอร์ในขณะที่รันโปรแกรมแบบกระจาย. SQL แพ็กเกจสามารถสร้างได้เมื่อ:

- พารามิเตอร์ RDB ถูกระบุไว้ในคำสั่ง CRTSQLxxx และอ็อบเจ็กต์ของโปรแกรมได้ถูกกำหนดไว้เรียบร้อยแล้ว. SQL แพ็กเกจจะถูกสร้างขึ้นบนระบบที่ระบุไว้ในพารามิเตอร์ RDB.

  ถ้าคอมไฟล์ไม่ผ่านหรือเพียงแค่สร้างอ็อบเจ็กต์โมดูลขึ้นมาเพียงอย่างเดียว, จะไม่มีการสร้าง SQL แพ็กเกจขึ้น.
- การใช้คำสั่ง CRTSQLPKG . CRTSQLPKG สามารถถูกนำมาใช้ในการสร้างแพ็กเกจเมื่อแพ็กเกจยังไม่ถูกสร้างขึ้นในช่วงพรีคอมไพล์ หรือถ้ามีความจำเป็นที่จะต้องใช้แพ็กเกจที่ RDB แทนคำสั่งที่ระบุไว้ในคำสั่งพรีคอมไพล์.

คำสั่ง Delete SQL Package (DLTSQLPKG) อนุญาตให้ลบ SQL แพ็กเกจที่อยู่บนระบบโลคัลได้.

SQL แพ็กเกจจะไม่ถูกสร้างขึ้นจนกว่าสิทธิพิเศษที่ถือครองโดย ID ที่ให้สิทธิไว้ซึ่งเกี่ยวข้องกับการสร้าง SQL แพ็กเกจรวมทั้งสิทธิที่เหมาะสมในการสร้างแพ็กเกจบนระบบรีโมต (แอ็พพลิเคชันเซิร์ฟเวอร์). ในการรันโปรแกรม, ID ที่ให้สิทธิไว้จะต้องรวมสิทธิพิเศษในการ EXECUTE ไว้ใน SQL แพ็กเกจด้วย. บนระบบปฏิบัติการ i5/OS สิทธิพิเศษในการ EXECUTE จะรวมเอาสิทธิในการใช้งานของระบบ อันได้แก่ *OBJOPR และ *EXECUTE เอาไว้ด้วย

สิ่งอ้างอิงที่เกี่ยวข้อง

Create SQL Package (CRTSQLPKG) command

คำสั่ง SQL ที่ถูกต้องในแพ็คเกจ SQL

โปรแกรมที่เชื่อมต่อกับเซิร์ฟเวอร์อื่นสามารถใช้คำสั่ง SQL ใดๆ ยกเว้นคำสั่ง SET TRANSACTION

โปรแกรมที่ถูกคอมไฟล์โดยใช้ DB2 for i5/OS ซึ่ง อ้างอิงถึงฐานข้อมูลแบบรีโมตที่ไม่ใช่ DB2 for i5/OS สามารถ ใช้คำสั่ง SQL ที่เรียกใช้งานได้ ซึ่งสนับสนุนโดยฐานข้อมูลแบบรีโมตนั้น พรีคอมไพเลอร์จะทำการส่งข้อความวินิจฉัยอย่างต่อเนื่องใน กรณีมีคำสั่งที่ไม่เข้าใจ คำสั่งเหล่านั้นจะถูกส่งไปยังระบบรีโมตในระหว่างการสร้างแพ็คเกจ SQL. การสนับสนุนรันไทม์จะส่ง คืนค่า SQLCODE เป็น -84 หรือ -525 เมื่อไม่สามารถรันคำสั่งบนแอ็พพลิเคชันเซิร์ฟเวอร์ปัจจุบัน ตัวอย่างเช่น การ FETCH แบบหลายแถว, การ INSERT แบบกลุ่มเร็กคอร์ด และเคอร์เซอร์แบบเลื่อนจะใช้ได้ เฉพาะในโปรแกรมแบบกระจายที่ทั้ง application requester และ แอ็พพลิเคชันเซิร์ฟเวอร์รันบน OS/400® V5R2 หรือ i5/OS V5R3 หรือหลังจากนั้น ด้วยข้อยก เว้นต่อไปนี้ application requester ที่ไม่ได้ รันบนระบบปฏิบัติการ i5/OS จะ สามารถออกคำสั่งการดำเนินการในลักษณะอ่าน อย่างเดียว, เคอร์เซอร์แบบเลื่อนได้ ที่ไม่ระบุ SENSITIVE บนแอ็พพลิเคชันเซิร์ฟเวอร์ i5/OS V5R3 ข้อจำกัดในการใช้คำสั่ง FETCH แบบหลายแถว, คำสั่ง INSERT กลุ่มเร็กคอร์ด และเคอร์เซอร์แบบเลื่อนได้คือ การไม่อนุญาตให้มีการส่งข้อมูลชนิด binary large object (BLOB), character large object (CLOB) และ double-byte character large object (DBCLOB) เมื่อมีการ ใช้ฟังก์ชันเหล่านี้

ข้อมูลที่เกี่ยวข้อง

ลักษณะของคำสั่ง SQL

ข้อควรพิจารณาในการสร้าง SQL แพ็กเกจ

เมื่อคุณสร้าง SQL แพ็กเกจ ให้พิจารณา แง่มุมต่อไปนี้

การให้สิทธิ CRTSQLPKG:

เมื่อคุณสร้าง SQL แพ็กเกจ บนระบบปฏิบัติการ i5/OS, authorization ID ที่ใช้จะต้องได้รับสิทธิ *USE สำหรับคำสั่ง Create SQL Package (CRTSQLPKG)

การสร้างแพ็กเกจบนฐานข้อมูลที่ไม่ใช่ DB2 for i5/OS:

เมื่อคุณสร้างโปรแกรมและ SQL แพ็กเกจบน ฐานข้อมูลอื่นที่ไม่ใช่ DB2 for i5/OS และพยายามที่จะใช้คำสั่ง SQL ที่ไม่ซ้ำกับ ฐานข้อมูลอื่นๆ คุณต้อง ตั้งค่าพารามิเตอร์ GENLVL บนคำสั่ง Create SQL (CRTSQL*xxx*) ไว้ที่ 30

โปรแกรมจะไม่ถูกสร้างขึ้นถ้าได้รับสัญญาณข้อความแสดงระดับค่าความรุนแรงมากกว่า 30. ถ้าสัญญาณข้อความถูกส่งออก มาด้วยระดับค่าความรุนแรงที่มีค่ามากกว่า 30, คำสั่ง อาจจะไม่สามารถใช้ได้กับฐานข้อมูลเชิงสัมพันธ์ใดๆ. ตัวอย่างเช่น, ตัวแปรโฮสต์ที่ไม่ได้ถูกระบุไว้หรือไม่สามารถใช้ได้หรือค่าคงที่ที่ไม่สามารถใช้ได้จะส่งสัญญาณข้อความ แสดงค่าความรุนแรงมากกว่า 30.

รายการพรีคอมไพเลอร์จะต้องถูกตรวจสอบหาสัญญาณข้อความผิดปกติเมื่อรันด้วยค่า GENLVL ที่มีค่ามากกว่า 10 เมื่อ คุณ ต้องการสร้างแพ็กเกจสำหรับฐานข้อมูล DB2 คุณจะต้องตั้งค่าพารามิเตอร์ GENLVL ให้ มีค่าน้อยกว่า 20

ถ้าพารามิเตอร์ RDB ระบุฐานข้อมูลอื่นที่ ไม่ใช่ DB2 for i5/OS ก็ไม่ควรใช้ อ็อพชันต่อไปนี้บนคำสั่ง CRTSQL*xxx*:

- COMMIT(*NONE)

- OPTION(*SYS)
- DATFMT(*MDY)
- DATFMT(*DMY)
- DATFMT(*JUL)
- DATFMT(*YMD)
- DATFMT(*JOB)
- DYNUSRPRF(*OWNER)
- TIMFMT(*HMS) ถ้า ระบุ TIMSEP(*BLANK) หรือ TIMSEP(',')
- SRTSEQ(*JOBRUN)
- SRTSEQ(*LANGIDUNQ)
- SRTSEQ(*LANGIDSHR)
- SRTSEQ(library-name/table-name)

หมายเหตุ: เมื่อคุณเชื่อมต่อกับเซิร์ฟเวอร์ DB2 จะต้องใช้กฎเพิ่มเติม ต่อไปนี้:

- รูปแบบวันที่และเวลาที่ระบุไว้จะต้องเป็นรูปแบบเดียวกัน
- ค่าของ *BLANK ต้องถูกนำมาใช้ในพารามิเตอร์ TEXT
- ไม่ใช้แบบแผนที่เป็นค่าดีฟอลต์ (DFTRDBCOL)
- CCSID ของซอร์สโปรแกรมจากแพ็กเกจที่กำลังถูกสร้างจะต้องไม่เป็น 65535; ถ้า 65535 ถูกใช้, แพ็กเกจเปล่าจะถูกสร้างขึ้น.

พารามิเตอร์ Target release (TGTRLS):

ในขณะที่สร้างแพ็กเกจ คำสั่ง SQL จะ ถูกตรวจสอบ เพื่อหาว่ารีลีสใดที่สามารถสนับสนุนฟังก์ชันได้

รีลีสนี้จะถูกตั้งค่าให้เป็นระดับเดิมของแพ็กเกจ. ตัวอย่างเช่น, ถ้าแพ็กเกจนั้นมีคำสั่ง CREATE TABLE ซึ่งเพิ่มข้อจำกัดของ FOREIGN KEY เข้าไปให้กับตาราง, จากนั้นค่าระดับที่เรียกคืนได้ของแพ็กเกจจะเป็น เวอร์ชัน 3 รีลีส 1, เนื่องจากไม่มีการสนับสนุนข้อจำกัดของ FOREIGN KEY ไว้ในรีลีสก่อนหน้านี้. สัญญาณข้อความ TGTRLS จะถูกระงับไว้เมื่อพารามิเตอร์ TGTRLS มีค่าเป็น *CURRENT.

SQL statement size:

ฟังก์ชันในการสร้าง SQL แพ็กเกจอาจจะไม่สามารถจัดการคำสั่ง SQL ที่มีขนาดเดียวกันกับที่พรีคอมไพเลอร์สามารถประมวลผลได้

ในขณะที่โปรแกรม SQL ถูกพรีคอมไพล์ คำสั่ง SQL จะ ถูกระบุให้อยู่ในพื้นที่ที่เชื่อมโยงกันของโปรแกรม เมื่อเหตุการณ์นี้เกิดขึ้น, แต่ละโทเค็นจะถูกแยกจากกันด้วยช่องว่าง. นอกจากนี้, เมื่อพารามิเตอร์ RDB ถูกระบุไว้, ตัวแปรโฮสต์ของคำสั่งต้นฉบับจะถูกแทนที่ด้วยตัวอักษร 'H'. ฟังก์ชันการสร้าง SQL แพ็กเกจจะส่งคำสั่งนี้ไปยังแอ็พพลิเคชันเซิร์ฟเวอร์, พร้อมกับรายชื่อตัวแปรโฮสต์ของคำสั่งนั้น. การเติม ช่องว่างระหว่างโทเค็นและการแทนที่ของตัวแปรโฮสต์อาจ ทำให้คำสั่งนั้นมีความยาวเกินความยาวสูงสุดของคำสั่ง SQL ได้ (SQL0101 เหตุผลที่ 5)

คำสั่งที่ไม่จำเป็นต้องใช้แพ็กเกจ:

ในบางกรณี อาจมีความพยายามที่จะสร้าง SQL แพ็กเกจ แต่ SQL แพ็กเกจนั้นจะไม่ถูกสร้างขึ้น และโปรแกรมจะยังรันอยู่ สถานการณ์นี้เกิดขึ้น เมื่อโปรแกรมมีแต่คำสั่ง SQL ที่ไม่จำเป็นต้องใช้ SQL แพ็กเกจในการรัน

ตัวอย่างเช่น, โปรแกรมที่มีแต่คำสั่ง SQL ที่เป็น DESCRIBE TABLE สร้างข้อความ SQL5041 ในระหว่างการสร้าง SQL แพ็กเกจ. คำสั่ง SQL ที่ไม่จำเป็นต้องใช้ SQL แพ็กเกจได้แก่:

- COMMIT
- CONNECT
- DESCRIBE TABLE
- DISCONNECT
- RELEASE
- RELEASE SAVEPOINT
- ROLLBACK
- SAVEPOINT
- SET CONNECTION

ชนิดอ็อบเจ็กต์แพ็กเกจ:

แพ็คเกจ SQL จะถูกสร้างขึ้นให้เป็นอ็อบเจ็กต์แบบ non-ILE เสมอและจะรันอยู่ใน activation group ปกติเสมอ.

โปรแกรม ILE และโปรแกรมเซอร์วิส:

โปรแกรม ILE และเซอร์วิสโปรแกรมที่เชื่อมโมดูลหลายๆ โมดูลที่มีคำสั่ง SQL จะต้องมี SQL แพ็กเกจแยกออกจากกันสำหรับแต่ละโมดูล .

การเชื่อมต่อโดยสร้างแพ็กเกจ:

ชนิดของการเชื่อมต่อที่ใช้สำหรับการสร้างแพ็คเกจ จะขึ้นอยู่กับชนิดของการเชื่อมต่อที่ระบุโดยพารามิเตอร์ RDBCNNMTH

ถ้า RDBCNNMTH(*DUW) ถูกระบุไว้, commitment control จะถูกนำมาใช้และการเชื่อมต่อจะเป็นแบบ read-only . ถ้าการเชื่อมต่อเป็นแบบ read-only, การสร้างแพ็กเกจจะล้มเหลว.

หน่วยของงาน:

เนื่องจากการสร้างแพ็กเกจจะเป็นการ commit หรือ rollback, commit definition จะต้องเป็นขอบเขตของหน่วยของการทำงานก่อนที่จะมีการสร้างแพ็กเกจ.

เงื่อนไขต่อไปนี้ต้องเป็นจริงทั้งหมด เพื่อที่จะให้ commit definition เป็นขอบเขตของหน่วยการทำงาน:

- SQL เป็นขอบเขตของหน่วยการทำงาน.
- ไม่มีไฟล์โลคัลหรือไฟล์ DDM ใดๆ ที่เปิดโดยใช้ commitment control และไม่มีการปิดไฟล์โลคัลหรือ DDM ใดขณะที่มีการเปลี่ยนแปลงค้างอยู่.
- ไม่มีรีซอร์สใดของ API ที่ถูก register ไว้.
- ไม่มีรีซอร์สใดของ LU 6.2 ที่ถูก register ไว้ที่ไม่เกี่ยวข้องกับ DRDA หรือ DDM.

การสร้างแพ็กเกจแบบโลคัล:

ชื่อที่ระบุบนพารามิเตอร์ RDB สามารถเป็นชื่อของระบบโลคัลได้.

ถ้าเป็นชื่อของระบบโลคัล SQL แพ็กเกจ จะถูกสร้างขึ้นบนระบบโลคัล SQL แพ็กเกจอาจถูกบันทึก (คำสั่ง Save Object (SAVOBJ)) และเรียกคืน (คำสั่ง Restore Object (RSTOBJ)) ไปยัง อีกระบบหนึ่งได้ เมื่อทำการรันโปรแกรมด้วยการเชื่อมต่อไปยังระบบโลคัล, แพ็คเกจ SQL จะไม่ถูกนำมาใช้. ถ้าทำการระบุ *LOCAL สำหรับพารามิเตอร์ RDB ไว้, อ็อบเจ็กต์ *SQLPKG จะไม่ถูกสร้างขึ้น, แต่ข้อมูลของแพ็กเกจจะถูกบันทึกไว้ในอ็อบเจ็กต์ *PGM.

เลเบล (Lebel):

คุณสามารถใช้คำสั่ง LABEL ON ในการสร้าง รายละเอียดสำหรับแพ็กเกจ SQL

โทเค็นที่สอดคล้องกัน:

โปรแกรมและ SQL แพ็กเกจที่เกี่ยวข้องจะมีโทเค็นที่มีความสอดคล้องกัน ซึ่งได้รับการตรวจสอบ เมื่อมีการเรียก SQL แพ็กเกจเกิดขึ้น.

โทเค็นที่มีความสอดคล้องกันนี้จะต้องมีค่าตรงกัน มิฉะนั้นจะไม่สามารถนำมาใช้ได้ อาจเป็นไปได้ที่โปรแกรมและแพ็คเกจ SQL จะไม่มีความสัมพันธ์กัน. สมมติว่า โปรแกรมและแอ็พพลิเคชันเซิร์ฟเวอร์อยู่บน ระบบปฏิบัติการ i5/OS สองชุด ที่แยกออกจากกัน โปรแกรมที่กำลังรันอยู่ในเซสชัน A และจะถูกสร้างขึ้นใหม่ในเซสชัน B (ซึ่ง SQL แพ็กเกจจะถูกสร้างขึ้นที่นี่ด้วย) การเรียกโปรแกรมในเซสชัน A ครั้งต่อไปอาจจะทำให้เกิดข้อผิดพลาดของโทเค็นที่มีความสอดคล้องกัน เพื่อหลีกเลี่ยงการระบุตำแหน่งของ SQL แพ็กเกจในการเรียกแต่ละครั้ง, SQL จะคงค่ารายการของแอดเดรสสำหรับ SQL แพ็กเกจที่ถูกใช้ในแต่ละเซสชันเอาไว้. เมื่อเซสชัน B ทำการสร้าง SQL แพ็กเกจขึ้นมาใหม่, SQL แพ็กเกจอันเก่าก็จะถูกย้ายไปยังไลบรารี QRPLOBJ. แอดเดรสไปยัง SQL แพ็กเกจในเซสชัน A จะยังคงเป็นค่าที่ใช้ได้อยู่. คุณสามารถหลีกเลี่ยงสถานการณ์เช่นนี้ได้ โดยการสร้างโปรแกรมและ SQL แพ็กเกจจากเซสชันที่กำลังรันโปรแกรมอยู่ หรือโดยการส่งคำสั่งรีโมตไปลบ SQL แพ็กเกจอันเก่าก่อนจะสร้างโปรแกรมใหม่

ในการที่จะใช้ SQL แพ็กเกจอันใหม่, คุณควรจบการเชื่อมต่อเข้ากับระบบรีโมตเสียก่อน. คุณสามารถเลือกอย่างใดอย่างหนึ่งระหว่างการออกจากเซสชันก่อนแล้วจึงเข้ามาใหม่, หรือคุณสามารถใช้คำสั่ง SQL แบบโต้ตอบ (STRSQL) ในการออกคำสั่ง DISCONNECT สำหรับการเชื่อมต่อของเครือข่ายที่ไม่มีการป้องกันหรือ คำสั่ง RELEASE ตามด้วย COMMIT สำหรับการเชื่อมต่อที่มีการป้องกัน. ดังนั้นจึงควรนำ RCLDDMCNV มาใช้ในการจบการเชื่อมต่อของเครือข่าย. จากนั้นจึงเรียกโปรแกรมอีกครั้งหนึ่ง.

SQL และการเรียกซ้ำ:

ถ้ามีการเรียก SQL จากโปรแกรมคีย์ Attention ในขณะที่กำลังทำการพรีคอมไพล์อยู่แล้วนั้น, จะทำให้ได้รับผลลัพธ์ที่ไม่ปรารถนาได้.

คำสั่ง Create SQL (CRTSQL*xxx*), Create SQL Package (CRTSQLPKG) และ Start SQL Interactive Session (STRSQL) และสภาวะแวดล้อมรันไทม์ SQL ไม่มีการเรียกซ้ำ ทั้งนี้จะทำให้เกิดผลลัพธ์ที่ไม่ปรารถนา ตามมา หากมีการพยายามทำการเรียกซ้ำ การเรียกซ้ำจะเกิดขึ้นถ้าในขณะที่คำสั่งใดคำสั่งหนึ่ง กำลังรันอยู่ (หรือในการรันโปรแกรมที่มีคำสั่ง SQL อยู่) งานนั้นเกิดขัดข้องก่อนที่คำสั่งจะทำงานเสร็จสมบูรณ์ และฟังก์ชัน SQL อีกอันหนึ่งจะถูก เริ่มการทำงานขึ้น

ข้อควรพิจารณาเกี่ยวกับ CCSID สำหรับ SQL

ถ้าคุณกำลังรันแอ็พพลิเคชันแบบกระจายอยู่ และ ระบบใดระบบหนึ่งไม่ใช่ระบบ System i เซิร์ฟเวอร์บนแพล็ตฟอร์ม System i ไม่สามารถตั้งค่า coded character set identifier (CCSID) ของงานไว้ที่ 65535

ก่อนการร้องขอให้ระบบรีโมตสร้าง แพ็คเกจ SQL, application requester จะทำการแปลงชื่อที่ระบุบน พารามิเตอร์ RDB, ชื่อของแพ็คเกจ SQL, ชื่อของไลบรารี, และเนื้อความ ที่อยู่ในแพ็คเกจจาก CCSID ไปเป็น CCSID 500 เสมอ การกระทำเช่นนี้จะถูกเรียกร้อง จาก Distributed Relational Database Architecture (DRDA) เมื่อระบบรีโมตเป็นระบบ System i ชื่อจะไม่ถูกแปลง จาก CCSID 500 เป็น CCSID ของงานนั้นๆ

คุณไม่ควรใช้ identifiers ที่ใช้สำหรับคั่นกับชื่อของตาราง, มุมมอง, ดรรชนี, แบบแผน, ไลบรารี, หรือแพ็คเกจ SQL. การแปลงชื่อระหว่างระบบที่มี CCSID ต่างกันจะทำไม่ได้. พิจารณาตัวอย่างดังต่อไปนี้ซึ่งระบบ A กำลังรันด้วยค่า CCSID เป็น 37 และระบบ B กำลังรันด้วยค่า CCSID เป็น 500.

- สร้างโปรแกรมที่สร้างตารางด้วยชื่อ ″a¬b|c″ บนระบบ A.
- บันทึกโปรแกรม ″a¬b|c″ บนระบบ A, จากนั้น ทำการเรียกคืนไปยังระบบ B.
- จุดของโค้ดสำหรับ ¬ ใน CCSID 37 คือ x'5F' ขณะที่ใน CCSID 500 คือ x'BA'.
- บนระบบ B ชื่อจะแสดงผลเป็น ″a[b]c″. ถ้าคุณได้สร้างโปรแกรมที่อ้างอิงถึงตารางซึ่งมีชื่อว่า ″a¬b|c.″, โปรแกรมจะหาตารางไม่พบ.

อักขระ (@), เครื่องหมาย pound (#), และเครื่องหมายดอลลาร์ ($) ไม่ควรนำมาใช้ในชื่อของ SQL อ็อบเจ็กต์. จำนวนจุดของโค้ดจะขึ้นอยู่ค่าของ CCSID ที่ใช้. ถ้าใช้ชื่อที่มีตัวคั่นหรือมีสาม national extender อยู่, อาจทำให้ฟังก์ชันการค้นหาชื่อรีลีสที่จะออกมาในอนาคตล้มเหลวได้.

การจัดการการเชื่อมต่อและ activation group

การเชื่อมต่อ SQL จะถูกควบคุมจัดการที่ระดับของ activation group. แต่ละ activation group ภายในหนึ่งงานจะควบคุมจัดการการเชื่อมต่อของตัวมันเองโดยไม่มีการใช้ข้าม activation group กัน.

ก่อนที่จะใช้ TCP/IP โดย Distributed Relational Database Architecture (DRDA) คำว่า *การเชื่อมต่อ* จะต้องไม่คลุมเครือ คำนี้หมายถึงการเชื่อมต่อจากมุมมองของ SQL นั่นคือ การเชื่อมต่อเริ่มต้นเมื่อคุณรันคำสั่ง CONNECT TO เพื่อเชื่อมต่อกับ ฐานข้อมูลเชิงสัมพันธ์ (RDB) และสิ้นสุดลงเมื่อคุณรันคำสั่ง DISCONNECT หรือ RELEASE ALL แล้วตามด้วยการดำเนินการ COMMIT ที่สำเร็จ การสนทนา (ระหว่างโปรแกรม) Advanced Program-to-Program Communication (APPC) อาจจะถูกเก็บหรือไม่เก็บเอาไว้ขึ้นอยู่กับ ค่าแอ็ตทริบิวต์ DDMCNV ของงาน และขึ้นอยู่กับว่าการสนทนา (ระหว่างโปรแกรม) นั้น เกิดขึ้นกับระบบ System i หรือ ระบบอื่นๆ

ศัพท์บัญญัติของ TCP/IP ไม่รวมถึงคำว่า *การสนทนา (ระหว่างโปรแกรม)* อย่างไรก็ตาม ได้มีการกล่าวถึงแนวคิดที่คล้ายกันไว้การสนับสนุน TCP/IP โดย DRDA, ทำให้การใช้งานของคำศัพท์ *การสนทนา (ระหว่างโปรแกรม)* ถูกแทนที่, ในหัวข้อนี้, ด้วยคำศัพท์ *การเชื่อมต่อ*, จนกว่าจะมีการกล่าวถึงการสนทนา (ระหว่างโปรแกรม) ของ APPC อย่างเฉพาะเจาะจง. ดังนั้น, จะมีการเชื่อมต่ออยู่สองประเภทที่ผู้อ่านต้องทราบ คือ: การเชื่อมต่อ SQL ของประเภทที่อธิบายมาแล้วข้างบน, และการเชื่อมต่อเน็ตเวิร์กที่ใช้แทนคำว่า *การสนทนา (ระหว่างโปรแกรม)*.

บางกรณีอาจเกิดความสับสนระหว่างการเชื่อมต่อสองประเภทนี้, ดังนั้น เราจะกล่างถึงการเชื่อมต่อพวกนี้ด้วยคำว่า SQL หรือเน็ตเวิร์ก เพื่อให้ผู้อ่านเข้าใจชัดเจนขึ้นว่าหมายถึงเรื่องใด.

ต่อไปนี้เป็นตัวอย่างของแอ็พพลิเคชันที่รันอยู่ในหลายๆ activation group. ซึ่งจะแสดงการโต้ตอบระหว่าง activation group, การจัดการการเชื่อมต่อ, และ commitment control. รูปแบบนี้ **ไม่** แนะนำให้ใช้ในการเขียนโค้ด

ซอร์สโค้ดสำหรับ PGM1

นี่คือซอร์สโค้ดสำหรับ PGM1.

```
...
EXEC SQL
  CONNECT TO SYSB
END-EXEC.
EXEC SQL
  SELECT ...
END-EXEC.
CALL PGM2.
...
```

รูปที่ 4. ซอร์สโค้ดสำหรับ PGM1

คำสั่งในการสร้างโปรแกรม และ SQL แพ็กเกจสำหรับ PGM1:

```
CRTSQLCBL PGM(PGM1) COMMIT(*NONE) RDB(SYSB)
```

ซอร์สโค้ดสำหรับ PGM2

นี่คือซอร์สโค้ดสำหรับ PGM2.

```
...
        EXEC SQL
          CONNECT TO SYSC;
        EXEC SQL
          DECLARE C1 CURSOR FOR
              SELECT ...;
        EXEC SQL
          OPEN C1;
        do {
             EXEC SQL
               FETCH C1 INTO :st1;
             EXEC SQL
               UPDATE ...
                   SET COL1 = COL1+10
                   WHERE CURRENT OF C1;
             PGM3(st1);
        } while SQLCODE == 0;
        EXEC SQL
         CLOSE C1;
        EXEC SQL COMMIT;
...
```

รูปที่ 5. ซอร์สโค้ดสำหรับ PGM2

คำสั่งที่ใช้ในการสร้างโปรแกรม และ SQL แพ็กเกจสำหรับ PGM2:

```
CRTSQLCI OBJ(PGM2) COMMIT(*CHG) RDB(SYSC) OBJTYPE(*PGM)
```

ซอร์สโค้ดสำหรับ PGM3

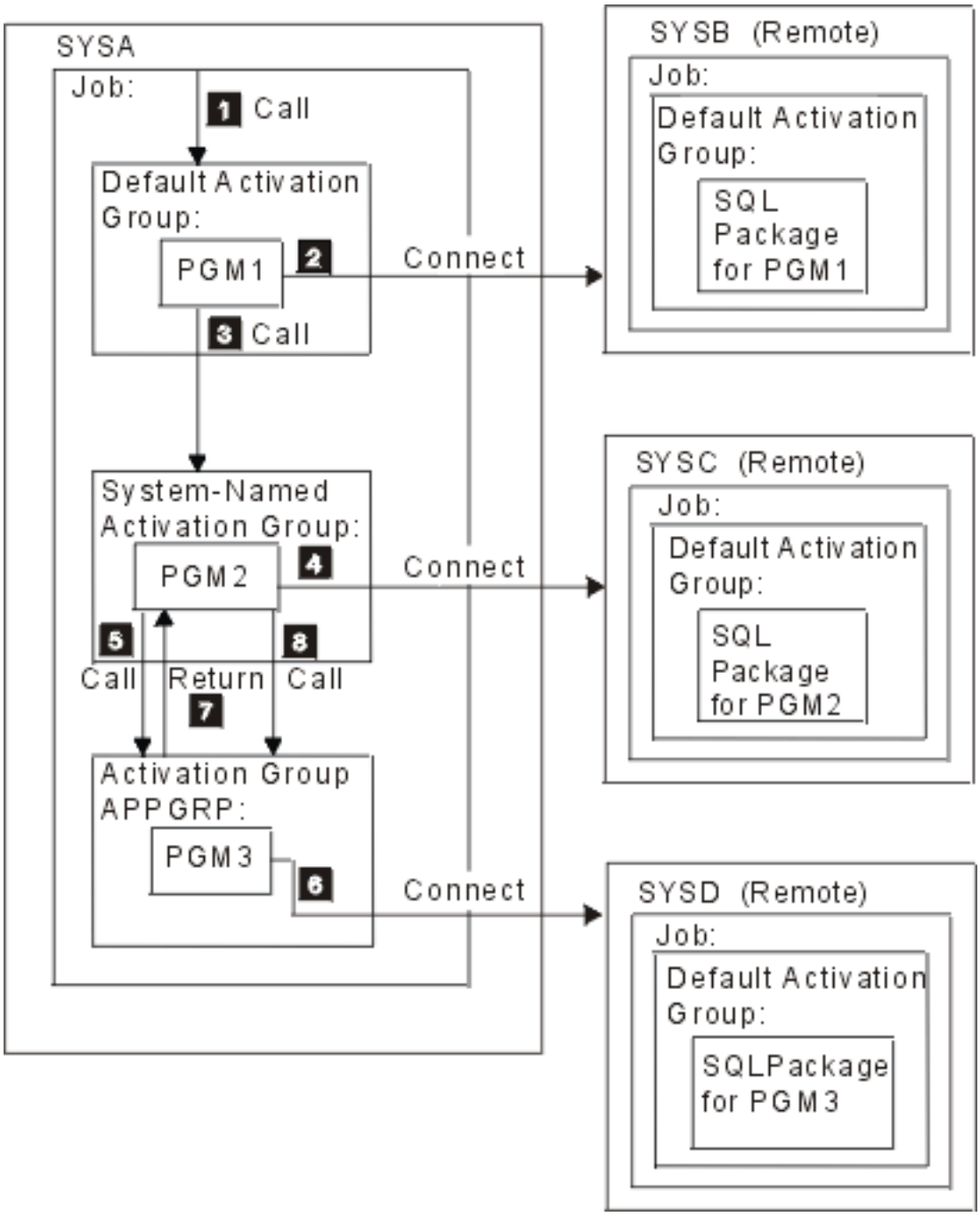
นี่คือซอร์สโค้ดสำหรับ PGM3.

```
...
 EXEC SQL
   INSERT INTO TAB VALUES(:st1);
 EXEC SQL COMMIT;
...
```

รูปที่ 6. ซอร์สโค้ดสำหรับ PGM3

คำสั่งในการสร้างโปรแกรม และ SQL แพ็กเกจสำหรับ PGM3:

```
CRTSQLCI OBJ(PGM3) COMMIT(*CHG) RDB(SYSD) OBJTYPE(*MODULE)
CRTPGM PGM(PGM3) ACTGRP(APPGRP)
CRTSQLPKG PGM(PGM3) RDB(SYSD)
```



ในตัวอย่างนี้, PGM1 เป็นโปรแกรม non-ILE ที่ถูกสร้างขึ้นโดยใช้คำสั่ง CRTSQLCBL. โปรแกรมนี้จะรันใน activation group ดีฟอลต์. PGM2 ถูกสร้างขึ้นโดยใช้คำสั่ง CRTSQLCI, และจะรันใน activation group ที่ได้รับการตั้งชื่อโดยระบบ. PGM3 ก็ถูกสร้างโดยใช้คำสั่ง CRTSQLCI เช่นกัน, แต่จะรันใน activation group ชื่อ APPGRP. เนื่องจาก APPGRP ไม่ได้เป็นค่าดีฟอลต์สำหรับพารามิเตอร์ ACTGRP, คำสั่ง CRTPGM ก็จะถูกรันแยกต่างหาก. คำสั่ง CRTPGM จะตามด้วยคำสั่ง CRTSQLPKG ที่สร้างอ็อบเจ็กต์แพ็คเกจ SQL อยู่บนฐานข้อมูลเชิงสัมพันธ์ SYSD. ในตัวอย่างนี้, ผู้ใช้ไม่ได้ทำการเรียกโปรแกรมทำงานของ definition ของ job level commitment ไว้โดยชัดเจน. SQL จะเรียกโปรแกรมทำงานของ commitment control โดยนัย.

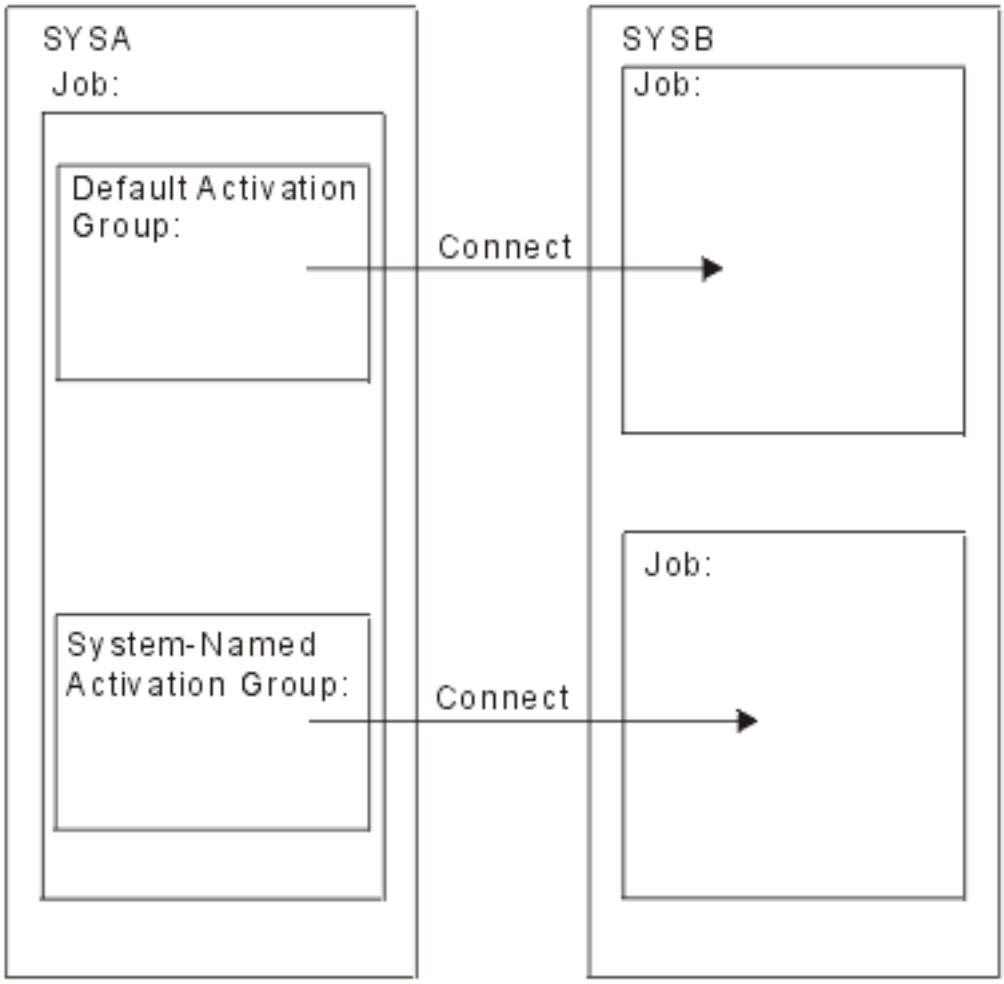
1. PGM1 ถูกเรียกและรันอยู่ใน activation group ดีฟอลต์.
2. PGM1 เชื่อมต่ออยู่กับฐานข้อมูลเชิงสัมพันธ์ SYSB และรันคำสั่ง SELECT.
3. PGM1 จะเรียก PGM2, ซึ่งรันใน activation group ที่ถูกตั้งชื่อโดยระบบ.

4. PGM2 ทำการเชื่อมต่อกับฐานข้อมูลเชิงสัมพันธ์ SYSC. เนื่องจาก PGM1 และ PGM2 จะอยู่ใน activation group ที่ต่างกัน, การเชื่อมต่อจาก PGM2 ใน activation group ที่ระบบตั้งชื่อให้ จะตัดการเชื่อมต่อจาก PGM1 ใน activation group ดีฟอลต์. การเชื่อมต่อทั้งสองนี้จะแอ็คทีฟทั้งคู่. PGM2 จะเปิดเคอร์เซอร์และดึงข้อมูลออกมา และจะอัพเดตแถวข้อมูล. โดยรันภายใต้ commitment control, ซึ่งอยู่ในระหว่างหน่วยการทำงาน, และไม่อยู่ที่สถานะที่จะเชื่อมต่อได้.
5. PGM2 เรียก PGM3, ซึ่งรันใน activation group APPGRP.
6. คำสั่ง INSERT เป็นคำสั่งแรกที่รันใน activation group APPGRP. คำสั่ง SQL จะทำให้เกิดการเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์ SYSD โดยนัย . แถวข้อมูลจะถูกแทรกเข้าไปในตาราง TAB ที่อยู่ในฐานข้อมูลเชิงสัมพันธ์ SYSD. จากนั้นการแทรกก็จะถูก commit. การเปลี่ยนแปลงที่ค้างอยู่ใน activation group ที่ตั้งชื่อโดยระบบ จะไม่ถูก commit, เนื่องจาก commitment control ถูกเรียกทำงานโดย SQL ด้วยขอบเขตการ commit ของ activation group.
7. จากนั้น PGM3 จะจบการทำงาน และการควบคุมจะกลับไปเป็นของ PGM2. PGM2 ทำการดึงข้อมูลและอัพเดตแถวข้อมูลอีกแถวหนึ่ง.
8. PGM3 ถูกเรียกขึ้นมาอีกครั้งเพื่อแทรกแถว. การเชื่อมต่อโดยนัยจะเสร็จสิ้นตั้งแต่มีการเรียกไปยัง PGM3 ครั้งแรก. โดยไม่มีการเชื่อมต่อในการเรียกครั้งต่อๆ มาอีกเนื่องจาก activation group ไม่ได้จบลงในระหว่างการเรียกไปยัง PGM3. ท้ายที่สุด, แถวข้อมูลทั้งหมดจะถูกประมวลผลโดย PGM2 และหน่วยการทำงานที่เชื่อมโยงกับ activation group ที่ถูกตั้งชื่อโดยระบบก็จะถูก commit.

การเชื่อมต่อหลายครั้งไปยังฐานข้อมูลเชิงสัมพันธ์เดียวกัน

หากมีหลาย activation group เชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์เดียวกัน การเชื่อมต่อ SQL แต่ละครั้งจะเชื่อมต่อเครือข่ายในตัวเอง และสร้างงานแอ็พพลิเคชันเซิร์ฟเวอร์ในตัวเองอีกด้วย

ถ้า activation group ถูกรันด้วย commitment control, การเปลี่ยนแปลงที่ถูก commit ใน activation group ใดจะไม่ commit การเปลี่ยนแปลงใน activation group จนกว่า definition ของ job-level commitment ถูกนำมาใช้.



การจัดการเชื่อมต่อโดยนัยสำหรับ activation group ดีฟอลต์

application requester สามารถเชื่อมต่อโดยนัย กับแอ็พพลิเคชันเซิร์ฟเวอร์ได้

การเชื่อมต่อ SQL โดยนัยเกิดขึ้นเมื่อ application requester ตรวจพบว่า คำสั่ง SQL แรกกำลังถูกใช้โดยโปรแกรม SQL แรกที่แอ็คทีฟใน activation group ดีฟอลต์ และรายการต่อไปนี้เป็นจริง:

- คำสั่ง SQL ที่ถูกเรียกนั้นไม่ใช่คำสั่ง CONNECT ที่มีพารามิเตอร์.
- SQL ไม่แอ็คทีฟใน activation group ดีฟอลต์.

สำหรับโปรแกรมแบบกระจาย, การเชื่อมต่อโดยนัยของ SQL เป็นการเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์ที่ระบุไว้บนพารามิเตอร์ RDB. สำหรับโปรแกรมที่เป็นแบบไม่กระจาย, การเชื่อมต่อโดยนัยของ SQL เป็นการเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์แบบโลคัล .

SQL จะสิ้นสุดการเชื่อมต่อที่แอ็คทีฟใน activation group ดีฟอลต์เมื่อ SQL เปลี่ยนสภาพเป็นไม่แอ็คทีฟ. SQL จะไม่แอ็คทีฟเมื่อ:

- Application requester ตรวจพบโปรแกรม SQL แรกที่แอ็คทีฟสำหรับกระบวนการนั้นได้สิ้นสุดลง และสิ่งต่อไปนี้เป็นจริงทั้งหมด:
  - ไม่มีการเปลี่ยนแปลง SQL ใดๆ ค้างอยู่
  - ไม่มีการเชื่อมต่อใดใช้การเชื่อมต่อที่ถูกป้องกันไว้
  - คำสั่ง SET TRANSACTION ไม่แอ็คทีฟ
  - ไม่มีการรันโปรแกรมใดๆ ที่ถูกพรีคอมไพล์ด้วย CLOSQLCSR(*ENDJOB).

  ถ้ามีการเปลี่ยนแปลงที่ค้างไว้, การเชื่อมต่อถูกป้องกันเอาไว้, หรือมีคำสั่ง SET TRANSACTION แอ็คทีฟ อยู่, SQL จะถูกระบุให้อยู่ในสถานะจบการทำงาน. ถ้ามีการรันโปรแกรมที่ถูกพรีคอมไพล์ด้วย CLOSQLCSR(*ENDJOB), SQL จะยังคงแอ็คทีฟสำหรับ activation group ดีฟอลต์จนกระทั่งงานนั้นสิ้นสุดลง.
- ในตอนท้ายของหน่วยของการทำงาน, ถ้า SQL อยู่ในสถานะจบการทำงาน. เหตุการณ์นี้จะเกิดขึ้นเมื่อทำการออกคำสั่ง COMMIT หรือ ROLLBACK จากนอกโปรแกรม SQL.
- ในตอนสิ้นสุดการทำงาน.

  สิ่งอ้างอิงที่เกี่ยวข้อง

  "การสิ้นสุดการเชื่อมต่อ" ในหน้า 323
  เนื่องจากการเชื่อมต่อ SQL แบบรีโมตมีการใช้รีซอร์ส ดังนั้นคุณ จึงควรจบการเชื่อมต่อที่ไม่ได้ใช้อีกต่อไปโดยเร็วที่สุดเท่าที่จะเป็นไปได้ คุณสามารถจบการเชื่อมต่อได้ไม่ว่าโดยนัยหรือโดยชัดเจน

การจัดการการเชื่อมต่อโดยนัยสำหรับ activation group ที่ไม่ใช่ดีฟอลต์

application requester สามารถเชื่อมต่อโดยนัย กับแอ็พพลิเคชันเซิร์ฟเวอร์ได้ การเชื่อมต่อ SQL โดยนัยเกิดขึ้นเมื่อ application requester ตรวจพบว่าคำสั่ง SQL แรกที่สั่งออกมาจาก activation group ไม่ใช่คำสั่ง CONNECT ที่มีพารามิเตอร์

สำหรับโปรแกรมแบบกระจาย, การเชื่อมต่อ SQL โดยนัยจะถูกกระทำกับฐานข้อมูลเชิงสัมพันธ์ที่ระบุในพารามิเตอร์ RDB. สำหรับโปรแกรมที่เป็นแบบไม่กระจาย, การเชื่อมต่อ SQL โดยนัยจะถูกกระทำกับฐานข้อมูลเชิงสัมพันธ์แบบโลคัล.

การตัดการเชื่อมต่อโดยนัยสามารถเกิดขึ้นได้ในกระบวนการในช่วงเวลาต่อไปนี้:

- เมื่อ activation group สิ้นสุดการทำงาน, ถ้า commitment control ไม่แอ็คทีฟ, commitment control ในระดับของ activation group จะแอ็คทีฟ, หรือ commitment definition ของระดับงานจะอยู่ที่ขอบเขตของหน่วยการทำงาน.

  ถ้า commitment definition ของระดับงานแอ็คทีฟ และไม่อยู่ที่ขอบเขตของหน่วยการทำงาน, SQL จะถูกระบุให้อยู่ในสถานะจบการทำงาน.
- ถ้า SQL อยู่ในสถานะจบการทำงาน, เมื่อ commitment definition ของระดับงานนั้นถูก commit หรือ roll back.
- ในตอนสิ้นสุดการทำงาน.

  สิ่งอ้างอิงที่เกี่ยวข้อง

  "การสิ้นสุดการเชื่อมต่อ" ในหน้า 323
  เนื่องจากการเชื่อมต่อ SQL แบบรีโมตมีการใช้รีซอร์ส ดังนั้นคุณ จึงควรจบการเชื่อมต่อที่ไม่ได้ใช้อีกต่อไปโดยเร็วที่สุดเท่าที่จะเป็นไปได้ คุณสามารถจบการเชื่อมต่อได้ไม่ว่าโดยนัยหรือโดยชัดเจน

การสนับสนุนแบบกระจาย

DB2 for i5/OS สนับสนุน ฐานข้อมูลเชิงสัมพันธ์แบบกระจายซึ่งมีสองระดับ

- Remote unit of work (RUW)

Remote unit of work คือ สถานการณ์ที่การเตรียมการและการรันคำสั่ง SQL หลายๆ คำสั่ง เกิดขึ้นในหนึ่งแอ็พพลิเคชันเซิร์ฟเวอร์เท่านั้นในช่วงของหนึ่งหน่วยการทำงาน. activation group พร้อมด้วยกระบวนการของแอ็พพลิเคชันที่ application requester สามารถเชื่อมต่อไปยังแอ็พพลิเคชันเซิร์ฟเวอร์, และรันคำสั่ง SQL แบบ static หรือ dynamic ที่อ้างถึงอ็อบเจ็กต์ที่อยู่บนแอ็พพลิเคชันเซิร์ฟเวอร์, ภายในหน่วยการทำงานตั้งแต่หนึ่งหน่วยขึ้นไป. Remote unit of work ก็ถูกอ้างถึงเช่นเดียวกันว่าเป็น DRDA ระดับ 1.

- Distributed unit of work (DUW)

  หน่วยการทำงานแบบกระจาย (Distributed unit of work)คือ สถานการณ์ที่การเตรียมการและการรันคำสั่ง SQL สามารถเกิดขึ้นได้ในหลายๆ แอ็พพลิเคชันเซิร์ฟเวอร์ในช่วงของหนึ่งหน่วยการทำงาน . อย่างไรก็ตาม, คำสั่ง SQL แบบเดี่ยวสามารถอ้างอิงถึงได้เฉพาะอ็อบเจ็กต์ที่อยู่ในแอ็พพลิเคชันเซิร์ฟเวอร์แบบเดี่ยวเท่านั้น. Distributed unit of work ก็ถูกอ้างถึงเช่นเดียวกันว่าเป็น DRDA ระดับ 2.

  หน่วยการทำงานแบบกระจายจะอนุญาตให้:

  – อัพเดตการเข้าใช้งานในหลายๆ แอ็พพลิเคชันเซิร์ฟเวอร์ในหนึ่งหน่วยการทำงานแบบโลจิคัล

    หรือ

  – อัพเดตการเข้าใช้งานในแอ็พพลิเคชันเซิร์ฟเวอร์แบบเดี่ยวโดยการเข้าไปอ่านข้อมูลในหลายแอ็พพลิเคชันเซิร์ฟเวอร์, ในหนึ่งหน่วยการทำงานแบบโลจิคัล.

  แอ็พพลิเคชันเซิร์ฟเวอร์หลายเซิร์ฟเวอร์จะถูกอัพเดตในหนึ่งหน่วยการทำงานได้หรือไม่จะขึ้นอยู่กับว่ามี sync point manager ที่ application requester หรือไม่, รวมทั้ง sync point manager ที่แอ็พพลิเคชันเซิร์ฟเวอร์, และการสนับสนุน commit protocol แบบสองเฟสระหว่าง application requester กับ application server.

  sync point manager เป็นส่วนประกอบของระบบที่ทำงานร่วมกับปฏิบัติการ commit และ rollback ท่ามกลางส่วนประกอบอื่นๆ ของ commit protocol แบบสองเฟส. เมื่อทำการรันการอัพเดตแบบกระจาย, sync point manager บนแต่ละระบบจะทำงานร่วมกันเพื่อตรวจสอบว่ามีรีซอร์สมากพอ. protocol และ flow ที่ถูกใช้โดย sync point manager จะถูกอ้างถึงว่าเป็น commit protocol แบบสองเฟสด้วยเช่นกัน. ถ้า commit protocol แบบสองเฟสจะถูกนำมาใช้, การเชื่อมต่อจะกลายเป็นรีซอร์สที่ถูกป้องกันไว้; มิฉะนั้น การเชื่อมต่อจะเป็นรีซอร์สที่ไม่ได้ป้องกันเอาไว้.

การจำแนกประเภทของการเชื่อมต่อ

เมื่อการเชื่อมต่อ SQL แบบรีโมตเกิดขึ้น ระบบจะใช้การเชื่อมต่อระบบเครือข่ายแบบป้องกันหรือแบบไม่ได้ป้องกันอย่างใดอย่างหนึ่ง

สำหรับการอัพเดตที่สามารถ commit ได้นั้น, การเชื่อมต่อของ SQL นี้อาจเป็นแบบอ่านได้อย่างเดียว, อัพเดตได้, หรือไม่ทราบว่าทำการอัพเดตได้หรือไม่เมื่อมีการเชื่อมต่อเกิดขึ้น. การอัพเดตที่สามารถ commit ได้นั้นจะเป็น การแทรก, การลบ, การอัพเดต, หรือคำสั่ง DDL ที่สามารถรันภายใต้ commitment control. ถ้าการเชื่อมต่อเป็นแบบอ่านเพียงอย่างเดียว, การเปลี่ยนแปลงที่ต้องใช้ COMMIT(*NONE) ยังรันได้. หลังจากการ CONNECT หรือ SET CONNECTION, SQLERRD(4) ของ SQLCA ได้ระบุชนิดของการเชื่อมต่อแล้ว.

DB2_CONNECTION_TYPE ค่าเฉพาะคือ:

1. การเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์แบบโลคัลและการเชื่อมต่อได้รับการปกป้อง.
2. การเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์แบบรีโมตและการเชื่อมต่อไม่ได้รับการปกป้อง.
3. การเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์แบบรีโมตและการเชื่อมต่อได้รับการปกป้อง.
4. การเชื่อมต่อไปยังไดร์เวอร์โปรแกรม application requester และการเชื่อมต่อได้รับการปกป้อง.

SQLERRD(4) ค่าเฉพาะคือ:

1. การเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์แบบรีโมตและการเชื่อมต่อไม่ได้รับการปกป้อง. การอัพเดตที่สามารถ commit ได้สามารถกระทำได้ในระหว่างการเชื่อมต่อ. ซึ่งจะเกิดขึ้นเมื่อข้อใดข้อหนึ่งต่อไปนี้เป็นจริง:
   - การเชื่อมต่อเกิดขึ้นโดยการใช้ remote unit of work (RUW).
   - ถ้าการเชื่อมต่อเกิดขึ้นโดยการใช้หน่วยการทำงานแบบกระจาย (DUW) แล้วสิ่งต่อไปนี้ทั้งหมดจะเป็นจริง:
     - การเชื่อมต่อนั้นไม่ได้เป็นแบบโลคัล.
     - แอ็พพลิเคชันเซิร์ฟเวอร์ไม่สนับสนุนหน่วยการทำงานแบบกระจาย.
     - ระดับของ commitment control ของโปรแกรมที่ทำการสร้างการเชื่อมต่อไม่ใช่ *NONE.
     - ไม่ว่าจะเป็นการที่ไม่มีการเชื่อมต่อใดๆ ไปยังแอ็พพลิเคชันเซิร์ฟเวอร์อื่นๆ (รวมทั้งโลคัล) ที่สามารถทำการอัพเดตที่ commit ได้หรือการเชื่อมต่อทั้งหมดเป็นการเชื่อมต่อแบบอ่านได้อย่างเดียวไปยังแอ็พพลิเคชันเซิร์ฟเวอร์ที่ไม่สนับสนุนหน่วยของการทำงานแบบกระจาย.
     - ไม่มีไฟล์โลคัลใดที่สามารถอัพเดตได้เปิดอยู่ภายใต้ commitment control สำหรับ commitment definition.
     - ไม่มีไฟล์ DDM ใดที่ทำการอัพเดตได้และเปิดอยู่โดยใช้การเชื่อมต่อที่แตกต่างกันภายใต้ commitment control สำหรับ commitment definition.
     - ไม่มี API commitment control รีซอร์สสำหรับ commitment definition.
     - ไม่มีการเชื่อมต่อแบบป้องกันไว้ถูกลงทะเบียนไว้สำหรับ commitment definition.

   ถ้าทำการรันด้วย commitment control, SQL จะลงทะเบียนรีซอร์ส DRDA ที่สามารถอัพเดตได้แบบเฟสเดียวสำหรับการเชื่อมต่อแบบรีโมต หรือรีซอร์ส DRDA ที่สามารถอัพเดตได้แบบสองเฟสสำหรับการเชื่อมต่อแบบโลคัลและ ARD.
2. การเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์แบบรีโมตและการเชื่อมต่อไม่ได้รับการปกป้อง. เนื่องจากการเชื่อมต่อเป็นแบบอ่านได้เพียงอย่างเดียว. ซึ่งจะเกิดขึ้นเมื่อสิ่งต่อไปนี้เป็นจริง:
   - การเชื่อมต่อนั้นไม่ได้เป็นแบบโลคัล.
   - แอ็พพลิเคชันเซิร์ฟเวอร์ไม่สนับสนุน หน่วยการทำงานแบบกระจาย
   - สิ่งต่างๆ ต่อไปนี้อย่างน้อยหนึ่งอย่างเป็นจริง:
     - ระดับของ commitment control ของโปรแกรมที่สั่งให้เชื่อมต่อเป็น *NONE.
     - การเชื่อมต่ออื่นปรากฏต่อแอ็พพลิเคชันเซิร์ฟเวอร์ที่ไม่สนับสนุน หน่วยการทำงานแบบกระจาย และแอ็พพลิเคชันเซิร์ฟเวอร์สามารถทำการอัพเดตที่ commit ได้
     - การเชื่อมต่ออื่นปรากฏต่อแอ็พพลิเคชันเซิร์ฟเวอร์ที่สนับสนุน หน่วยการทำงานแบบกระจาย (รวมไปจนถึงโลคัล).
     - มีไฟล์ที่สามารถอัพเดตได้เปิดอยู่ภายใต้ commitment control สำหรับ commitment definition.
     - มีไฟล์ DDM ใดที่ทำการอัพเดตได้เปิดอยู่โดยใช้การเชื่อมต่อที่แตกต่างกันภายใต้ commitment control สำหรับ commitment definition.
     - ไม่มี API commitment control รีซอร์สแบบเฟสเดียวสำหรับ commitment definition.
     - มีการเชื่อมต่อแบบป้องกันที่ลงทะเบียนไว้สำหรับ definition.

   ถ้าทำการรันด้วย commitment control, SQL จะลงทะเบียนรีซอร์ส DRDA แบบอ่านได้อย่างเดียวแบบเฟสเดียว.
3. การเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์แบบรีโมตและการเชื่อมต่อได้รับการปกป้อง. เราไม่สามารถทราบได้ว่าสามารถทำการอัพเดตแบบ commit ได้หรือไม่ . เหตุการณ์นี้จะเกิดขึ้นเมื่อทั้งหมดนี้เป็นจริง:
   - การเชื่อมต่อนั้นไม่ได้เป็นแบบโลคัล.

- ระดับของ commitment control ของโปรแกรมที่ทำการสร้างการเชื่อมต่อไม่ใช่ *NONE.
- แอ็พพลิเคชันเซิร์ฟเวอร์สนับสนุนทั้งหน่วยการทำงานแบบกระจาย และ commit protocol แบบสองเฟส (การเชื่อมต่อแบบป้องกัน).

ถ้าทำการรันด้วย commitment control, SQL จะลงทะเบียนรีซอร์ส DRDA ที่ไม่ทราบค่าแบบสองเฟส.

4. การเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์แบบรีโมตและการเชื่อมต่อไม่ได้รับการปกป้อง. เราไม่สามารถทราบได้ว่าสามารถทำการอัพเดตแบบ commit ได้หรือไม่ . เหตุการณ์นี้จะเกิดขึ้นเมื่อทั้งหมดนี้เป็นจริง:
   - การเชื่อมต่อนั้นไม่ได้เป็นแบบโลคัล.
   - แอ็พพลิเคชันเซิร์ฟเวอร์สนับสนุนหน่วยการทำงานแบบกระจาย
   - อาจเป็นเพราะแอ็พพลิเคชันเซิร์ฟเวอร์ไม่สนับสนุน commit protocol แบบสองเฟส (การเชื่อมต่อแบบป้องกัน) หรือระดับของ commitment control ของโปรแกรมที่ทำการสร้างการเชื่อมต่อเป็น *NONE อย่างใดอย่างหนึ่ง.

   ถ้าทำการรันด้วย commitment control, SQL จะลงทะเบียนรีซอร์ส DRDA ที่ไม่ทราบค่าแบบเฟสเดียว.
5. การเชื่อมต่อไปยังฐานข้อมูลเชิงสัมพันธ์แบบโลคัลหรือโปรแกรม application requester driver (ARD) และการเชื่อมต่อได้รับการปกป้อง. เราไม่สามารถทราบได้ว่าสามารถทำการอัพเดตแบบ commit ได้หรือไม่ . ถ้าทำการรันด้วย commitment control, SQL จะลงทะเบียนรีซอร์ส DRDA ที่ไม่ทราบค่าแบบสองเฟส.

ตารางต่อไปนี้สรุปประเภทของการเชื่อมต่อที่เป็นผลลัพธ์ของการเชื่อมต่อหน่วยการทำงานรีโมตแบบกระจาย. SQLERRD (4) จะถูกตั้งค่าในคำสั่ง CONNECT และ SET CONNECTION.

ตารางที่ 50. ข้อสรุปประเภทของการเชื่อมต่อ

เชื่อมต่อภายใต้ under commitment control	แอ็พพลิเคชันเซิร์ฟเวอร์สนับสนุน commit แบบสองระยะ	แอ็พพลิเคชันเซิร์ฟเวอร์สนับสนุนหน่วยการทำงานแบบกระจาย	รีซอร์สแบบหนึ่งระยะอื่นๆ ที่อัพเดตได้ถูกลงทะเบียน	SQLERRD(4)
ไม่	ไม่	ไม่	ไม่	2
ไม่	ไม่	ไม่	ใช่	2
ไม่	ไม่	ใช่	ไม่	4
ไม่	ไม่	ใช่	ใช่	4
ไม่	ใช่	ไม่	ไม่	2
ไม่	ใช่	ไม่	ใช่	2
ไม่	ใช่	ใช่	ไม่	4
ไม่	ใช่	ใช่	ใช่	4
ใช่	ไม่	ไม่	ไม่	1
ใช่	ไม่	ไม่	ใช่	2
ใช่	ไม่	ใช่	ไม่	4
ใช่	ไม่	ใช่	ใช่	4
ใช่	ใช่	ไม่	ไม่	ไม่มี [1]

ตารางที่ 50. ข้อสรุปประเภทของการเชื่อมต่อ (ต่อ)

เชื่อมต่อภายใต้ under commitment control	แอ็พพลิเคชันเซิร์ฟเวอร์สนับสนุน commit แบบสองระยะ	แอ็พพลิเคชันเซิร์ฟเวอร์สนับสนุนหน่วยการทำงานแบบกระจาย	รีซอร์สแบบหนึ่งระยะอื่นๆ ที่อัพเดตได้ถูกลงทะเบียน	SQLERRD(4)
ใช่	ใช่	ไม่	ใช่	ไม่มี [1]
ใช่	ใช่	ใช่	ไม่	3
ใช่	ใช่	ใช่	ใช่	3

[1]DRDA ไม่อนุญาตให้ใช้การเชื่อมต่อแบบป้องกันกับแอ็พพลิเคชันเซิร์ฟเวอร์ที่สนับสนุน remote unit of work (DRDA1) เท่านั้น รวมถึงการเชื่อมต่อ TCP/IP ของ DB2 ทั้งหมดสำหรับ i5/OS.

หลักการที่เกี่ยวข้อง

Commitment control

สิ่งอ้างอิงที่เกี่ยวข้อง

"การเข้าใช้งานฐานข้อมูลแบบรีโมตด้วย SQL แบบโต้ตอบ" ในหน้า 296
ใน SQL แบบโต้ตอบ, คุณสามารถสื่อสารกับฐานข้อมูลเชิงสัมพันธ์แบบรีโมตด้วยการใช้ข้อความ SQL CONNECT. SQL แบบโต้ตอบจะใช้ซีแมนทิกส์ CONNECT (Type 2) (หน่วยงานแบบกระจาย) สำหรับคำสั่ง CONNECT.

ข้อจำกัดของตัวควบคุมการเชื่อมต่อและตัวควบคุม Commitment

การเชื่อมต่อด้วย commitment control จะมีข้อจำกัดบางอย่าง นอกจากนี้ยังมีข้อจำกัดในกรณีที่ คุณพยายามรันคำสั่งโดยใช้ commitment control แต่คุณระบุ COMMIT(*NONE) บนการเชื่อมต่อ

ถ้ารีซอร์สที่ไม่ทราบค่าหรือสามารถอัพเดตได้แบบสองเฟสถูกลงทะเบียนเอาไว้หรือรีซอร์สที่สามารถอัพเดตได้แบบเฟสเดียวถูกลงทะเบียนเอาไว้แล้ว,รีซอร์สที่สามารถอัพเดตได้แบบเฟสเดียวอีกอันหนึ่งจะไม่สามารถถูกลงทะเบียนได้.

นอกจากนั้น, เมื่อการเชื่อมต่อกลายเป็น inactive และ attribute ของงาน DDMCNV เป็น *KEEP, การเชื่อมต่อที่ไม่ได้ใช้เหล่านี้จะทำให้คำสั่ง CONNECT ในโปรแกรมที่ถูกคอมไพล์ด้วยการจัดการการเชื่อมต่อแบบ RUW ทำงานล้มเหลว.

ถ้าทำการรันด้วยการจัดการการเชื่อมต่อแบบ RUW และใช้ definition ของ job-level commitment, ก็จะมีข้อจำกัดเพิ่มขึ้นมาด้วย.

- ถ้า definition ของ job-level commitment ถูกใช้โดย activation group มากกว่าหนึ่งกลุ่ม, การเชื่อมต่อแบบ RUW ทั้งหมดจะต้องไปยังฐานข้อมูลเชิงสัมพันธ์แบบโลคัล.
- ถ้าการเชื่อมต่อเป็นแบบรีโมต, จะมี activation group เพียงกลุ่มเดียวเท่านั้นที่สามารถใช้ definition ของ job-level commitment สำหรับการเชื่อมต่อแบบ RUW.

การจำแนกสถานะของการเชื่อมต่อ

คำสั่ง CONNECT ที่ปราศจากพารามิเตอร์สามารถใช้ค้นหาว่า การเชื่อมต่อปัจจุบันสามารถอัพเดตได้หรือเป็นแบบอ่านได้อย่างเดียว สำหรับหน่วยการทำงานปัจจุบัน

ค่าของ 1 หรือ 2 จะถูกส่งคืนใน SQLERRD(3) ใน SQLCA หรือ DB2_CONNECTION_STATUS ในพื้นที่วินิจฉัย SQL ผลลัพธ์จะปรากฏดังนี้:

1. การอัพเดตที่สามารถ commit ได้สามารถกระทำได้ในการเชื่อมต่อสำหรับหน่วยการทำงานนั้น.

ซึ่งจะเกิดขึ้นเมื่อข้อใดข้อหนึ่งต่อไปนี้เป็นจริง:

- การเชื่อมต่อเกิดขึ้นโดยการใช้ remote unit of work (RUW).
- ถ้าการเชื่อมต่อเกิดขึ้นโดยใช้หน่วยการทำงานแบบกระจาย (DUW) และสิ่งต่อไปนี้ทั้งหมดเป็นจริง:
  - ไม่มีการเชื่อมต่อใดเกิดขึ้นกับแอ็พพลิเคชันเซิร์ฟเวอร์ที่ไม่สนับสนุนหน่วยการทำงานแบบกระจายซึ่งอัพเดตแบบ commit ได้.
  - สิ่งใดต่อไปนี้เป็นจริง:
    - การอัพเดตที่ commit ได้จะถูกกระทำบนการเชื่อมต่อแบบป้องกัน, ฐานข้อมูลโลคัล, หรือการเชื่อมต่อไปยังโปรแกรม ARD .
    - มีไฟล์โลคัลที่สามารถอัพเดตได้เปิดอยู่ภายใต้ commitment control. .
    - มีไฟล์ DDM ที่สามารถอัพเดตได้เปิดอยู่และใช้การเชื่อมต่อแบบป้องกัน.
    - มีรีซอร์สของ API commitment control แบบสองเฟส.
    - ไม่ได้ทำการอัพเดตที่สามารถ commit ได้.
- ถ้าการเชื่อมต่อเกิดขึ้นโดยใช้หน่วยการทำงานแบบกระจาย (DUW) และสิ่งต่อไปนี้ทั้งหมดเป็นจริง:
  - ไม่มีการเชื่อมต่ออื่นเกิดขึ้นกับแอ็พพลิเคชันเซิร์ฟเวอร์ที่ไม่สนับสนุนหน่วยการทำงานแบบกระจาย ซึ่งอัพเดตแบบ commit ได้.
  - การอัพเดตแบบ commit ได้ครั้งแรกถูกกระทำบนการเชื่อมต่อนี้หรือไม่ได้ทำการอัพเดตแบบ commit ได้.
  - ไม่มีไฟล์ DDM ที่สามารถอัพเดตได้เปิดอยู่โดยใช้การเชื่อมต่อแบบป้องกัน.
  - ไม่มีมีไฟล์โลคัลที่สามารถอัพเดตได้เปิดอยู่ภายใต้ commitment control.
  - ไม่มีรีซอร์สใดๆ ของ API commitment control แบบสองเฟส.

2. ในการเชื่อมต่อสำหรับหน่วยการทำงานนี้ คุณจะไม่สามารถทำการอัพเดตที่ commit ได้.
   ซึ่งจะเกิดขึ้นเมื่อข้อใดข้อหนึ่งต่อไปนี้เป็นจริง:
   - ถ้าการเชื่อมต่อเกิดขึ้นโดยการใช้หน่วยการทำงานแบบกระจาย (DUW) และสิ่งใดสิ่งหนึ่งต่อไปนี้เป็นจริง:
     - การเชื่อมต่อเกิดขึ้นกับแอ็พพลิเคชันเซิร์ฟเวอร์แบบอัพเดตได้ที่สนับสนุนเพียงหน่วยการทำงานแบบรีโมต.
     - การอัพเดตแบบ commit ได้ครั้งแรกจะถูกกระทำบนการเชื่อมต่อแบบไม่ป้องกัน.
   - ถ้าการเชื่อมต่อเกิดขึ้นโดยการใช้หน่วยการทำงานแบบกระจาย (DUW) และสิ่งใดสิ่งหนึ่งต่อไปนี้เป็นจริง:
     - การเชื่อมต่อเกิดขึ้นกับแอ็พพลิเคชันเซิร์ฟเวอร์แบบอัพเดตได้ที่สนับสนุนเพียงหน่วยการทำงานแบบรีโมต.
     - การอัพเดตแบบ commit ได้ครั้งแรกไม่ได้ถูกกระทำบนการเชื่อมต่อนี้.
     - มีไฟล์ DDM ที่สามารถอัพเดตได้เปิดอยู่และใช้การเชื่อมต่อแบบป้องกัน.
     - มีไฟล์โลคัลที่สามารถอัพเดตได้เปิดอยู่ภายใต้ commitment control.
     - มีรีซอร์สของ API commitment control แบบสองเฟส.

ตารางต่อไปนี้สรุปว่าสถานะของการเชื่อมต่อถูกกำหนดค่าได้อย่างไรโดยยึดจากค่าของชนิดการเชื่อมต่อ, ถ้ามีการเชื่อมต่อแบบสามารถอัพเดตได้กับแอ็พพลิเคชันเซิร์ฟเวอร์ซึ่งสนับสนุนเพียงหน่วยการทำงานแบบรีโมตเท่านั้น, และที่ซึ่งการอัพเดตแบบ commit ได้เกิดขึ้นเป็นครั้งแรก.

ตารางที่ 51. ข้อสรุปการจำแนกค่าสถานะของการเชื่อมต่อ

วิธีการเชื่อมต่อ	มีการเชื่อมต่อไปยังแอ็พพลิเคชันเซิร์ฟเวอร์ของ หน่วยการทำงานแบบรีโมต	สถานการณ์ที่จะเกิดการอัพเดตครั้งแรกที่สามารถ commit ได้ [1]	SQLERRD(3) หรือ DB2_CONNECTION_STATUS
RUW			1
DUW	ใช่		2
DUW	ไม่	ไม่มีการอัพเดต	1
DUW	ไม่	เฟสเดียว	2
DUW	ไม่	การเชื่อมต่อนี้	1
DUW	ไม่	สองเฟส	1

[1] คำศัพท์ในคอลัมน์นี้จะถูกระบุเป็น:

- *No updates* ระบุว่าไม่มีการอัพเดตที่ commit ได้, ไม่มีไฟล์ DDM ใดๆ เปิดไว้สำหรับการอัพเดตโดยใช้การเชื่อมต่อแบบป้องกัน, ไม่มีไฟล์โลคัลใดๆ เปิดไว้สำหรับการอัพเดต, และไม่มี commitment control API ใดๆ ถูกลงทะเบียนไว้.
- *One-phase* ระบุว่าการอัพเดตแรกที่สามารถ commit ได้ถูกกระทำโดยใช้การเชื่อมต่อแบบไม่ป้องกัน หรือไฟล์ DDM เปิดไว้สำหรับการอัพเดตโดยใช้การเชื่อมต่อแบบไม่ป้องกัน.
- *Two-phase* ระบุว่ามีการอัพเดตที่สามารถ commit ได้บนแอ็พพลิเคชันเซิร์ฟเวอร์แบบสองเฟสที่มีหน่วยการทำงานแบบกระจาย, ไฟล์ DDM จะเปิดไว้สำหรับการอัพเดตโดยการใช้การเชื่อมต่อแบบป้องกัน, commitment control API ถูกลงทะเบียนไว้, หรือไฟล์โลคัลจะเปิดไว้สำหรับกอัพเดตภายใต้ commitment control.

ถ้ามีการพยายามอัพเดตที่สามารถ commit ได้บนการเชื่อมต่อ, หน่วยการทำงานจะถูกระบุให้อยู่ในสถานะที่ต้อมีการ rollback. ถ้าเป็นดังนั้น, คำสั่งเดียวที่ใช้ได้คือ ROLLBACK ; ส่วนคำสั่งอื่นๆ ทั้งหมดจะส่งผลลัพธ์เป็น SQLCODE -918.

ข้อควรพิจารณาในการเชื่อมต่อหน่วยการทำงานแบบกระจาย

เมื่อทำการเชื่อมต่อในแอ็พพลิเคชันที่มีหน่วยการทำงานแบบกระจาย ให้ พิจารณาประเด็นต่อไปนี้

- ถ้าหน่วยการทำงานจะกระทำการอัพเดตที่แอ็พพลิเคชันเซิร์ฟเวอร์มากกว่าหนึ่งเซิร์ฟเวอร์โดยมีการใช้ commitment control, การเชื่อมต่อทั้งหมดรวมทั้งการอัพเดตจะถูกกระทำโดยใช้ commitment control. ถ้าการเชื่อมต่อถูกทำขึ้นโดยไม่ใช้ commitment control และมีการอัพเดตแบบ commit ได้เกิดขึ้นในภายหลัง, ผลลัพธ์ที่ได้น่าจะเป็นการเชื่อมต่อแบบอ่านได้อย่างเดียว.
- รีซอร์สที่เป็น non-SQL commit อื่นๆ, เช่น ไฟล์โลคัล, ไฟล์ DDM, และรีซอร์ส commitment control API, จะส่งผลกระทบต่อสถานะในการอัพเดตและการอ่านได้เพียงอย่างเดียวของการเชื่อมต่อนั้น.
- ถ้าคุณเชื่อมต่อโดยใช้ commitment control ไปยังแอ็พพลิเคชันเซิร์ฟเวอร์ ที่ไม่สนับสนุนหน่วยการทำงานแบบกระจาย (ตัวอย่างเช่น ระบบ V4R5 ที่ใช้ TCP/IP) การเชื่อมต่อนั้นจะเป็นแบบสามารถอัพเดตได้ หรือแบบอ่านอย่างเดียว ถ้าการเชื่อมต่อนั้น สามารถอัพเดตได้ ก็จะเป็นเพียงการเชื่อมต่อที่สามารถอัพเดตได้เท่านั้น ตั้งแต่ V5R3 เป็นต้นมา commit operation แบบสองเฟสของ Distributed Relational Database Architecture (DRDA) คำนึงถึงการอัพเดตต่อไปนี้:
  - การอัพเดตที่เป็นผลมาจากทริกเกอร์
  - การอัพเดตที่เป็นผลมาจากฟังก์ชันแบบผู้ใช้กำหนดเองที่ถูกเรียกใช้ในระหว่างเคียวรี่ฐานข้อมูล

การสิ้นสุดการเชื่อมต่อ

เนื่องจากการเชื่อมต่อ SQL แบบรีโมตมีการใช้รีซอร์ส ดังนั้นคุณ จึงควรจบการเชื่อมต่อที่ไม่ได้ใช้อีกต่อไปโดยเร็วที่สุดเท่าที่จะเป็นไปได้ คุณสามารถจบการเชื่อมต่อได้ไม่ว่าโดยนัยหรือโดยชัดเจน

ส่วนการสิ้นสุดการเชื่อมต่ออย่างชัดเจนทำได้โดยการใช้คำสั่ง DISCONNECT หรือ RELEASE อย่างใดอย่างหนึ่งและตามด้วยการ COMMIT ที่สำเร็จสมบูรณ์. คำสั่ง DISCONNECT ใช้ได้กับการเชื่อมต่อแบบป้องกันหรือการเชื่อมต่อโลคัล. คำสั่ง DISCONNECT จะทำให้การเชื่อมต่อสิ้นสุดเมื่อรันคำสั่งนี้. คำสั่ง RELEASE ใช้กับการเชื่อมต่อแบบป้องกันหรือไม่ก็ได้. เมื่อรันคำสั่ง RELEASE, การเชื่อมต่อจะไม่สิ้นสุดลงแต่จะถูกระบุให้อยู่ในสถานะ release แทน. ซึ่งยังสามารถใช้การเชื่อมต่อนั้นได้อยู่. และการเชื่อมต่อนั้นจะไม่สิ้นสุดจนกว่าจะรัน COMMIT สำเร็จ. การ ROLLBACK หรือการ COMMIT ที่ไม่สำเร็จจะทำให้การเชื่อมต่อที่อยู่ในสถานะ release ไม่สิ้นสุดลง.

เมื่อเกิดการเชื่อมต่อ SQL แบบรีโมต การเชื่อมต่อเครือข่ายของ การจัดการข้อมูลแบบกระจาย (DDM) (การสนทนา (ระหว่างโปรแกรม) Advanced Program-to-Program Communication (APPC) หรือการเชื่อมต่อแบบ TCP/IP) จะถูกนำมาใช้ เมื่อการเชื่อมต่อของ SQL สิ้นสุดลง การ เชื่อมต่อเครือข่ายอาจจะอยู่ในสถานะที่ยังไม่ได้ใช้งาน หรือหลุดไป ไม่ว่าจะเป็นการเชื่อมต่อระบบเครือข่ายจะหลุดไปหรือถูกระบุให้อยู่ในสถานะที่ยังไม่ได้ใช้งานจะขึ้นอยู่กับแอททริบิวต์งานของ DDMCNV. ถ้าค่าของแอ็ตทริบิวต์งานเป็น *KEEP และมีการเชื่อมต่อไปยังเซิร์ฟเวอร์บนแพล็ตฟอร์ม System i การเชื่อมต่อนั้น จะอยู่ในสถานะไม่ใช้งาน ถ้าค่าของแอ็ตทริบิวต์งานเป็น *DROP และ มีการเชื่อมต่อไปยังเซิร์ฟเวอร์บนแพล็ตฟอร์ม System i การเชื่อมต่อนั้นจะหลุดไป ถ้ามีการเชื่อมต่อไปยังเซิร์ฟเวอร์บนแพล็ตฟอร์มที่ไม่ใช่ System i การเชื่อมต่อจะหลุดไปทุกครั้ง สถานะ *DROP ควรจะเกิดขึ้นในสถานการณ์ดังต่อไปนี้:

- เมื่อการรักษาการเชื่อมต่อที่ไม่ได้ใช้นั้นทำให้เสียค่าใช้จ่ายมากและมีแนวโน้มว่าจะไม่ได้ใช้การเชื่อมต่อนั้นในระยะเวลาอันใกล้.
- เมื่อทำการรันโปรแกรมหลายๆ โปรแกรมด้วยกัน, บางโปรแกรมจะคอมไพล์ด้วยการจัดการการเชื่อมต่อแบบ RUW และบางโปรแกรมจะ คอมไพล์ด้วยการจัดการการเชื่อมต่อแบบ DUW. การรันโปรแกรมที่ถูกคอมไพล์ด้วยการจัดการการเชื่อมต่อ RUW ไปยังตำแหน่งรีโมตจะล้มเหลวหากมีการเชื่อมต่อแบบป้องกันอยู่แล้ว.
- เมื่อทำการรันด้วยการเชื่อมต่อแบบป้องกันไว้โดยใช้ DDM หรือ DRDA อย่างใดอย่างหนึ่ง. จะทำให้เกิดค่าใช้จ่ายเพิ่มเติมในการ commit และ rollback สำหรับการเชื่อมต่อแบบป้องกันที่ไม่ได้ใช้งาน.

คำสั่งการเชื่อมต่อแบบ Reclaim DDM (RCLDDMCNV) สามารถใช้จบการเชื่อมต่อที่ไม่ใช้งาน , ถ้าอยู่ที่ขอบเขตของการ commit.

สิ่งอ้างอิงที่เกี่ยวข้อง

"การจัดการเชื่อมต่อโดยนัยสำหรับ activation group ดีฟอลต์" ในหน้า 315
application requester สามารถเชื่อมต่อโดยนัย กับแอ็พพลิเคชันเซิร์ฟเวอร์ได้
"การจัดการการเชื่อมต่อโดยนัยสำหรับ activation group ที่ไม่ใช่ดีฟอลต์" ในหน้า 316
application requester สามารถเชื่อมต่อโดยนัย กับแอ็พพลิเคชันเซิร์ฟเวอร์ได้ การเชื่อมต่อ SQL โดยนัยเกิดขึ้นเมื่อ application requester ตรวจพบว่าคำสั่ง SQL แรกที่สั่งออกมาจาก activation group ไม่ใช่คำสั่ง CONNECT ที่มีพารามิเตอร์

หน่วยการทำงานแบบกระจาย

Distributed unit of work (DUW) อนุญาต ให้มีการเข้าใช้งานแอ็พพลิเคชันเซิร์ฟเวอร์หลายเซิร์ฟเวอร์ภายในหน่วยการทำงานเดียวกัน

คำสั่ง SQL แต่ละคำสั่งสามารถเข้าใช้งานได้เพียงหนึ่งแอ็พพลิเคชันเซิร์ฟเวอร์เท่านั้น . ในขณะที่การใช้หน่วยการทำงานแบบกระจายจะอนุญาตให้ทำการเปลี่ยนแปลงที่หลายๆ แอ็พพลิเคชัน เซิร์ฟเวอร์ในการ commit หรือ rollback ภายในหน่วยการทำงานเดียว

การจัดการการเชื่อมต่อหน่วยการทำงานแบบกระจาย

คุณสามารถใช้คำสั่ง CONNECT, SET CONNECTION, DISCONNECT, และ RELEASE เพื่อจัดการการเชื่อมต่อในสภาวะแวดล้อมหน่วยการทำงานแบบกระจาย (DUW)

หน่วยการทำงานแบบกระจาย CONNECT จะถูกรันเมื่อโปรแกรมถูกพรีคอมไพล์โดยใช้ RDBCNNMTH(*DUW), ซึ่งเป็นค่าดีฟอลต์. รูปแบบของคำสั่ง CONNECT นี้จะไม่ตัดการเชื่อมต่อที่มีอยู่แต่จะระบุการเชื่อมต่อที่มีอยู่ก่อนให้อยู่ในสถานะที่ถูกระงับไว้แทน. ฐานข้อมูลเชิงสัมพันธ์ที่ระบุบนคำสั่ง CONNECT จะกลายเป็นการเชื่อมต่อปัจจุบัน . คำสั่ง CONNECT สามารถใช้ในการเริ่มการเชื่อมต่อใหม่; ถ้าต้องการที่จะสลับระหว่างการเชื่อมต่อที่มีอยู่, คำสั่ง SET CONNECTION จะต้องถูกนำมาใช้. เนื่องจากการเชื่อมต่อใช้รีซอร์สของระบบ, การเชื่อมต่อเหล่านั้นจึงควรที่จะจบลงเมื่อไม่จำเป็นที่จะต้องใช้อีก. คำสั่ง RELEASE หรือ DISCONNECT สามารถนำมาใช้จบการเชื่อมต่อได้. คำสั่ง RELEASE จะต้องตามด้วยการ commit ที่สำเร็จเพื่อจบการเชื่อมต่อ.

ต่อไปนี้เป็นตัวอย่างของโปรแกรมภาษาซีที่รันอยู่ใน DUW environment ที่ใช้ commitment control.

```
...
 EXEC SQL WHENEVER SQLERROR GO TO done;
 EXEC SQL WHENEVER NOT FOUND GO TO done;
...
 EXEC SQL
   DECLARE C1 CURSOR WITH HOLD FOR
       SELECT PARTNO, PRICE
            FROM PARTS
            WHERE SITES_UPDATED = 'N'
            FOR UPDATE OF SITES_UPDATED;
/*    Connect to the systems      */
EXEC SQL CONNECT TO LOCALSYS;
EXEC SQL CONNECT TO SYSB;
EXEC SQL CONNECT TO SYSC;
/* Make the local system the current connection */
EXEC SQL SET CONNECTION LOCALSYS;
/* Open the cursor */
EXEC SQL OPEN C1;

while (SQLCODE==0)
   {
    /* Fetch the first row */
    EXEC SQL FETCH C1 INTO :partnumber,:price;
    /* Update the row which indicates that the updates have been
       propagated to the other sites */
    EXEC SQL UPDATE PARTS SET SITES_UPDATED='Y'
                WHERE CURRENT OF C1;
    /* Check if the part data is on SYSB */
    if ((partnumber > 10) && (partnumber < 100))
       {
        /* Make SYSB the current connection and update the price */
        EXEC SQL SET CONNECTION SYSB;
        EXEC SQL UPDATE PARTS
                    SET PRICE=:price
                    WHERE PARTNO=:partnumber;
        }

     /* Check if the part data is on SYSC */
     if ((partnumber > 50) && (partnumber < 200))
        {
         /* Make SYSC the current connection and update the price */
         EXEC SQL SET CONNECTION SYSC;
         EXEC SQL UPDATE PARTS
                     SET PRICE=:price
                     WHERE PARTNO=:partnumber;
        }
     /* Commit the changes made at all 3 sites */
     EXEC SQL COMMIT;
     /* Set the current connection to local so the next row
        can be fetched */
     EXEC SQL SET CONNECTION LOCALSYS;
    }
 done:

 EXEC SQL WHENEVER SQLERROR CONTINUE;
 /* Release the connections that are no longer being used */
 EXEC SQL RELEASE SYSB;
 EXEC SQL RELEASE SYSC;
 /* Close the cursor */
 EXEC SQL CLOSE C1;
 /* Do another commit which will end the released connections.
    The local connection is still active because it was not
```

ในโปรแกรมนี้, มีแอ็พพลิเคชันเซิร์ฟเวอร์สามตัวที่แอ็คทีฟ: LOCALSYS ซึ่งเป็นระบบโลคัล, และระบบรีโมตอีกสองตัว, SYSB และ SYSC. SYSB และ SYSC สนับสนุนหน่วยการทำงานแบบกระจาย และ two-phase commit.

เริ่มแรกการเชื่อมต่อทั้งหมดจะถูกทำให้แอ็คทีฟโดยการใช้คำสั่ง CONNECT สำหรับแต่ละแอ็พพลิเคชันเซิร์ฟเวอร์ที่เกี่ยวข้องในการดำเนินงาน. เมื่อใช้ DUW, คำสั่ง CONNECT จะไม่ตัดการเชื่อมต่อที่มีอยู่ก่อนหน้านี้, แต่จะระบุการเชื่อมต่อที่มีอยู่ก่อนเป็นสถานะที่ถูกระงับไว้แทน. หลังจากที่แอ็พพลิเคชันเซิร์ฟวเอร์ทั้งหมดได้ถูกเชื่อมต่อแล้ว, การเชื่อมต่อแบบโลคัลจะถูกทำให้เป็นการเชื่อมต่อปัจจุบันโดยใช้คำสั่ง SET CONNECTION. เคอร์เซอร์ก็จะถูกเปิดและแถวข้อมูลแรกก็จะถูกดึงออกมา. จากนั้นระบบจะตรวจสอบว่าต้องอัพเดตข้อมูลที่แอ็พพลิเคชันเซิร์ฟเวอร์ใด. ถ้า SYSB ต้องการการอัพเดต, SYSB จะถูกทำให้เป็นการเชื่อมต่อปัจจุบันโดยใช้คำสั่ง SET CONNECTION และการอัพเดตจะถูกรัน. และจะมีการดำเนินการแบบเดียวกันนี้สำหรับ SYSC. การเปลี่ยนแปลงจะถูก commit หลังจากนั้น.

เนื่องจากการ commit แบบสองเฟสกำลังถูกใช้, การเปลี่ยนแปลงจะถูก commit ที่ระบบโลคัลและระบบรีโมตอีกสองระบบอย่างแน่นอน. เนื่องจากเคอร์เซอร์ถูกประกาศให้เป็น WITH HOLD, ทำให้ยังเปิดหลังจากที่ทำการ commit. จากนั้นการเชื่อมต่อปัจจุบันก็จะถูกเปลี่ยนเป็นระบบโลคัลทำให้แถวถัดไปของข้อมูลสามารถถูกดึงออกมาได้. การดึงข้อมูล, การอัพเดต, และการ commit แบบนี้จะถูกทำซ้ำไปเรื่อยๆ จนกระทั่งข้อมูลทั้งหมดถูกประมวลผล.

หลังจากข้อมูลทั้งหมดถูกดึงออกมา, การเชื่อมต่อสำหรับระบบรีโมตทั้งสองจะถูก release. โดยไม่จบการเชื่อมต่อเนื่องจากเป็นการเชื่อมต่อแบบป้องกันอยู่. หลังจากที่การเชื่อมต่อถูก release, จะมีการ commit เพื่อที่จะสิ้นสุดการเชื่อมต่อทั้งหมด. ระบบโลคัลยังคงเชื่อมต่อและทำการประมวลผลไปเรื่อยๆ.

การตรวจสอบสถานะของการเชื่อมต่อ

หากเป็นไปได้ที่จะมีการเชื่อมต่อแบบอ่านอย่างเดียว โปรแกรมของคุณควรจะตรวจสอบสถานะของการเชื่อมต่อ ก่อนที่จะทำการอัพเดต ที่สามารถ commit ได้ การทำเช่นนี้จะช่วยป้องกันหน่วยการทำงานจากการเข้าสู่งสถานะที่ต้องมีการ rollback

ตัวอย่างภาษา COBOL ต่อไปนี้แสดงวิธีการตรวจสอบสถานะการเชื่อมต่อ.

```
      ...
       EXEC SQL
        SET CONNECTION SYS5
       END-EXEC.
       ...
* Check if the connection is updatable.
     EXEC SQL CONNECT END-EXEC.
* If connection is updatable, update sales information otherwise
* inform the user.
     IF SQLERRD(3) = 1 THEN
       EXEC SQL
         INSERT INTO SALES_TABLE
             VALUES(:SALES-DATA)
        END-EXEC
      ELSE
        DISPLAY 'Unable to update sales information at this time'.
...
```

รูปที่ 8. ตัวอย่างของการตรวจสอบสถานะการเชื่อมต่อ

เคอร์เซอร์ และคำสั่งที่เตรียมไว้

เคอร์เซอร์และคำสั่งที่เตรียมไว้แล้วจะถูกระบุในหน่วยการคอมไพล์และการเชื่อมต่อด้วยเช่นกัน.

การกำหนดหน่วยการคอมไพล์หมายถึงการที่โปรแกรมซึ่งเรียกจากอีกโปรแกรมที่ถูกคอมไพล์แยกกันไม่สามารถใช้เคอร์เซอร์หรือคำสั่งที่เตรียมไว้ซึ่งถูกเปิดหรือเตรียมโดยการเรียกโปรแกรม. การระบุการเชื่อมต่อหมายถึงแต่ละการเชื่อมต่อภายในโปรแกรมสามารถมี instance ของเคอร์เซอร์หรือคำสั่งที่เตรียมไว้แยกจากกัน.

ตัวอย่างหน่วยการทำงานแบบกระจายต่อไปนี้แสดงวิธีเปิดเคอร์เซอร์ตัวเดียวกันสำหรับการเชื่อมต่อที่ต่างกัน, ทำให้เกิด instance ของเคอร์เซอร์ C1 สองอย่าง.

```
...
 EXEC SQL DECLARE C1 CURSOR FOR
             SELECT * FROM CORPDATA.EMPLOYEE;
 /* Connect to local and open C1 */
 EXEC SQL CONNECT TO LOCALSYS;
 EXEC SQL OPEN C1;
 /* Connect to the remote system and open C1 */
 EXEC SQL CONNECT TO SYSA;
 EXEC SQL OPEN C1;
 /* Keep processing until done */
 while (NOT_DONE) {
    /* Fetch a row of data from the local system */
    EXEC SQL SET CONNECTION LOCALSYS;
    EXEC SQL FETCH C1 INTO :local_emp_struct;
    /* Fetch a row of data from the remote system */
    EXEC SQL SET CONNECTION SYSA;
    EXEC SQL FETCH C1 INTO :rmt_emp_struct;
    /* Process the data */
    ...
 }
 /* Close the cursor on the remote system */
 EXEC SQL CLOSE C1;
 /* Close the cursor on the local system */
 EXEC SQL SET CONNECTION LOCALSYS;
 EXEC SQL CLOSE C1;
 ...
```

รูปที่ 9. ตัวอย่างของเคอร์เซอร์ในโปรแกรม DUW

ไดรเวอร์โปรแกรม application requester

เพื่อให้เข้าถึงฐานข้อมูลซึ่งเตรียมไว้โดยผลิตภัณฑ์ที่ใช้ Distributed Relational Database Architecture (DRDA) DB2 for i5/OS นำเสนอ อินเตอร์เฟสสำหรับการเขียนโปรแกรมทางออกบน application requester ของ DB2 for i5/OS ในการประมวลผล SQL request โปรแกรมทางออกดังกล่าวเรียกว่า *ไดร์เวอร์ application requester (ARD)*

ระบบทำการเรียกโปรแกรม ARD ในขณะปฏิบัติการ ต่อไปนี้:

- ในช่วงของการสร้างแพ็คเกจโดยการใช้คำสั่ง CRTSQLPKG หรือ CRTSQLxxx, เมื่อพารามิเตอร์ฐานข้อมูลเชิงสัมพันธ์ (RDB) ตรงกับชื่อ RDB ที่สอดคล้องกับโปรแกรม ARD.
- การประมวลลผลคำสั่ง SQL เมื่อมีการเชื่อมต่อปัจจุบันไปยังชื่อ RDB ที่สอดคล้องกับโปรแกรม ARD .

การเรียกเหล่านี้อนุญาตให้โปรแกรม ARD ส่งผ่านคำสั่ง SQL และข้อมูลเกี่ยวกับคำสั่งนั้นไปยังฐานข้อมูลเชิงสัมพันธ์แบบรีโมต และส่งผลลัพธ์กลับมายังระบบ. จากนั้นข้อมูลก็จะส่งค่ากลับมายังแอ็พพลิเคชันหรือผู้ใช้. การเข้าไปใช้ฐานข้อมูลเชิงสัมพันธ์โดยโปรแกรม ARD จะคล้ายกันกับการเข้าไปในแอ็พพลิเคชันเซิร์ฟเวอร์ DRDA ในสภาวะแวดล้อมที่ไม่เหมือนกัน. อย่างไรก็ตาม, สภาวะแวดล้อม ARD จะไม่สนับสนุนฟังก์ชัน DRDA. ตัวอย่างของฟังก์ชันที่ไม่ได้การสนับสนุน ได้แก่ อ็อบเจ็กต์ขนาดใหญ่(LOBs) และรหัสผ่านที่มีขนาดยาว (passphrases).

การรับมือกับปัญหา

กลยุทธ์หลักสำหรับการดักจับและรายงานผลข้อมูลที่ผิดพลาดสำหรับฟังก์ชันฐานข้อมูลแบบกระจายจะมีชื่อว่า*first failure data capture--FFDC*.

จุดประสงค์ของการสนับสนุน FFDC นี้ก็เพื่อเป็นการจัดเตรียมข้อมูล ที่ถูกต้องเกี่ยวกับข้อผิดพลาดที่ตรวจพบในส่วนประกอบ distributed data management (DDM) ของระบบปฏิบัติการ i5/OS ที่ สามารถสร้าง Authorized Program Analysis Report (APAR) ด้วยการทำงานของฟังก์ชันนี้ โครงสร้างหลักและ data stream ของ DDM จะถูกดัมพ์โดยอัตโนมัติไปยังไฟล์สพูล ข้อมูลที่ผิดพลาด 1024 ไบต์แรกจะถูกบันทึกไว้ในบันทึกข้อผิดพลาดของระบบ. การดัมพ์ข้อมูลข้อผิดพลาดอัตโนมัติเกี่ยวกับข้อผิดพลาดครั้งแรกหมายถึงความล้มเหลวจะต้องไม่ถูกสร้างขึ้นอีกเพื่อให้ลูกค้าแจ้งเข้า FFDC จะแอ็คทีฟทั้งในฟังก์ชัน application requester และแอ็พพลิเคชันเซิร์ฟเวอร์ในส่วนประกอบ DDM ของ i5/OS . อย่างไรก็ตาม, สำหรับข้อมูล FFDC ที่ต้องการบันทึกการทำงานไว้, ค่ากำหนดของระบบในส่วนของ QSFWERRLOG จะต้องถูกตั้งให้เป็น *LOG.

หมายเหตุ: ค่า SQLCODEs ที่เป็นลบจะถูกดัมพ์เป็นบางค่าเท่านั้น; ได้แก่ ค่าที่ใช้ในการผลิต APAR. สำหรับข้อมูลเพิ่มเติมเกี่ยวกับการรับมือกับปัญหาในปฏิบัติการของฐานข้อมูลเชิงสัมพันธ์แบบกระจาย โปรดดู *DRDA: แนวทางการแก้ปัญหา* SC26-4782 คู่มือนี้ มีอยู่ที่ IBM Publications Center ในรูปแบบของสิ่งพิมพ์ฮาร์ดก๊อปปี้ที่คุณสามารถสั่งซื้อ หรือในรูปแบบออนไลน์ที่คุณสามารถดาวน์โหลดได้ฟรี

เมื่อข้อผิดพลาดของ SQL ถูกตรวจพบ, SQLCODE พร้อมด้วย SQLSTATE ที่เกี่ยวข้องจะถูกส่งคืนมาใน SQLCA.

สิ่งอ้างอิงที่เกี่ยวข้อง

ข้อความและโค้ด SQL

ข้อควรพิจารณาสำหรับโพรซีเดอร์ของ DRDA ที่บันทึกไว้

เซิร์ฟเวอร์ i5/OS Distributed Relational Database Architecture (DRDA) สนับสนุนการส่งคืนชุดผลลัพธ์หนึ่งชุด หรือมากกว่านั้นในโพรซีเดอร์ที่เก็บไว้

คุณสามารถสร้างชุดผลลัพธ์ในโพรซีเดอร์ที่ถูกบันทึกเอาไว้โดยการเปิดเคอร์เซอร์ SQL จำนวนหนึ่งหรือมากกว่าที่เกี่ยวข้องกับคำสั่ง SELECT นอกเหนือจากนั้น, ค่าสูงสุดของอะเรย์ชุดผลลัพธ์สามารถถูกส่งค่าคืนมาได้เพียงหนึ่งชุด. ก่อนรุ่น V5R3, ณ เวลาใดเวลาหนึ่งโพรซิเดอร์ที่บันทึกไว้หนึ่งสามารถมีเปิดเพียงหนึ่ง instance ของเคียวรีเท่านั้น. ตอนนี้คุณสามารถเรียกโพรซีเดอร์ที่บันทึกไว้ได้หลายครั้งโดยไม่ต้องปิดชุดผลลัพธ์เคอร์เซอร์ ดังนั้นจึงสามารถเปิดเคียวรีได้มากกว่าหนึ่ง instance พร้อมกัน

หลักการที่เกี่ยวข้อง

"สตอร์โพรซีเดอร์" ในหน้า 146
โพรซีเดอร์ (ซึ่งมักเรียกว่า โพรซีเดอร์ที่เก็บไว้) คือโปรแกรมที่สามารถเรียกขึ้นมาเพื่อปฏิบัติงาน โพรซีเดอร์อาจประกอบด้วย คำสั่งภาษาโฮสต์และคำสั่ง SQL โพรซีเดอร์ใน SQL มีข้อดีเหมือนกับโพรซีเดอร์ในภาษาโฮสต์

โปรแกรมมิงฐานข้อมูลแบบกระจาย

สิ่งอ้างอิงที่เกี่ยวข้อง

SET RESULT SETS

CREATE PROCEDURE (SQL)

CREATE PROCEDURE (External)

การอ้างอิง

ข้อมูลอ้างอิงสำหรับโปรแกรมมิง SQL จะประกอบด้วยตัวอย่างตารางและคำสั่ง CL.

DB2 for i5/OS ตารางตัวอย่าง

ตารางตัวอย่างต่อไปนี้ถูกอ้างอิงและใช้งานใน หัวข้อการโปรแกรม SQL และการอ้างอิง SQL

ที่ให้มาพร้อมกับตารางนี้คือ คำสั่ง SQL สำหรับสร้างตาราง.

ตารางนี้ประกอบด้วยข้อมูลซึ่งให้รายละเอียดของพนักงาน, แผนก, โครงการ, และการดำเนินการ, ในลักษณะเป็นกลุ่ม. ข้อมูลนี้ มีโปรแกรมตัวอย่างซึ่งแสดงให้เห็นคุณลักษณะบางประการของไลเซนส์โปรแกรม IBM DB2 Query Manager and SQL Development Kit for i5/OS ตัวอย่างทั้งหมดที่อยู่ในตาราง จะอยู่ในแบบแผนที่ชื่อว่า CORPDATA (สำหรับข้อมูลบริษัท)

นอกจากนี้ยังมีการส่งโพรซีเดอร์ซึ่งเป็นส่วนหนึ่งของระบบที่ มีคำสั่ง data definition language (DDL) เพื่อสร้างตาราง ทั้งหมดนี้ และคำสั่ง INSERT เพื่อบรรจุตาราง โดย โพรซีเดอร์จะสร้างแบบแผนที่ระบุในการเรียกไปยังโพรซีเดอร์ เนื่องจาก โพรซีเดอร์เป็นแบบที่เก็บไว้ภายนอก จึงสามารถเรียกได้จากอินเตอร์เฟส SQL ใดๆ ก็ได้ รวมถึง SQL แบบโต้ตอบและ System i Navigator ในการเรียกโพรซีเดอร์ที่คุณจะสร้างแบบแผน SAMPLE ให้ใช้ คำสั่งต่อไปนี้:

```
CALL QSYS.CREATE_SQL_SAMPLE ('SAMPLE')
```

ต้องใช้ตัวพิมพ์ใหญ่ระบุชื่อแบบแผน. และต้องไม่มีแบบแผนนั้นอยู่ก่อน.

หมายเหตุ: ในตารางตัวอย่างเหล่านี้, เครื่องหมายคำถาม (?) แสดงถึงค่าศูนย์.

สิ่งอ้างอิงที่เกี่ยวข้อง

"Referential integrity และตาราง" ในหน้า 19
Referential integrity คือเงื่อนไข สำหรับชุดของตารางในฐานข้อมูลซึ่งใช้ในการอ้างอิงทั้งหมดจากตารางหนึ่งไปยัง อีกตารางหนึ่ง

"การดึงข้อมูลออก (FETCH) แบบหลายแถวโดยใช้พื้นที่หน่วยเก็บของแถว" ในหน้า 264
ก่อนที่จะใช้คำสั่ง FETCH แบบหลายแถวที่มี พื้นที่หน่วยเก็บของแถว แอ็พพลิเคชันจะต้องกำหนดพื้นที่หน่วยเก็บของแถว และ พื้นที่รายละเอียดที่เกี่ยวข้อง

ตารางแผนก (DEPARTMENT)

ตารางแผนกจะแสดงรายละเอียดของแต่ละแผนก ในบริษัท และระบุชื่อผู้จัดการ และแผนกที่ต้อง รายงาน

สร้างตารางแผนกด้วย คำสั่ง CREATE TABLE และ ALTER TABLE ต่อไปนี้:

```
CREATE TABLE DEPARTMENT
      (DEPTNO    CHAR(3)          NOT NULL,
       DEPTNAME  VARCHAR(36)      NOT NULL,
       MGRNO     CHAR(6)                  ,
       ADMRDEPT  CHAR(3)          NOT NULL,
       LOCATION  CHAR(16),
       PRIMARY KEY (DEPTNO))

ALTER TABLE DEPARTMENT
      ADD FOREIGN KEY ROD (ADMRDEPT)
          REFERENCES DEPARTMENT
          ON DELETE CASCADE
```

foreign key ต่อไปนี้จะถูกเพิ่มในภายหลัง.

```
ALTER TABLE DEPARTMENT
      ADD FOREIGN KEY RDE (MGRNO)
          REFERENCES EMPLOYEE
          ON DELETE SET NULL
```

ดรรชนีต่อไปนี้จะถูกสร้าง.

```
CREATE UNIQUE INDEX XDEPT1
       ON DEPARTMENT (DEPTNO)

CREATE INDEX XDEPT2
       ON DEPARTMENT (MGRNO)

CREATE INDEX XDEPT3
       ON DEPARTMENT (ADMRDEPT)
```

alias ต่อไปนี้จะถูกสร้างให้กับตาราง.

```
CREATE ALIAS DEPT FOR DEPARTMENT
```

ตารางต่อไปนี้จะแสดงเนื้อหาของคอลัมน์.

ตารางที่ 52. คอลัมน์ของตารางแผนก

ชื่อคอลัมน์	รายละเอียด
DEPTNO	หมายเลขแผนกหรือ ID.
DEPTNAME	ชื่อที่อธิบายถึงกิจกรรมทั่วไปของแผนก.
MGRNO	หมายเลขพนักงาน (EMPNO) ของผู้จัดการแผนก.
ADMRDEPT	แผนก (DEPTNO) ซึ่งเป็นหน่วยงานที่แผนกนี้รายงานตรงไปยัง; แผนกที่อยู่ในระดับสูงสุดจะรายงานตรงมายังแผนกของตนเอง.
LOCATION	ที่ตั้งของแผนก.

DEPARTMENT:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง DEPARTMENT

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	?
B01	PLANNING	000020	A00	?
C01	INFORMATION CENTER	000030	A00	?
D01	DEVELOPMENT CENTER	?	A00	?
D11	MANUFACTURING SYSTEMS	000060	D01	?
D21	ADMINISTRATION SYSTEMS	000070	D01	?
E01	SUPPORT SERVICES	000050	A00	?
E11	OPERATIONS	000090	E01	?
E21	SOFTWARE SUPPORT	000100	E01	?
F22	BRANCH OFFICE F2	?	E01	?
G22	BRANCH OFFICE G2	?	E01	?
H22	BRANCH OFFICE H2	?	E01	?
I22	BRANCH OFFICE I2	?	E01	?
J22	BRANCH OFFICE J2	?	E01	?

ตารางพนักงาน (EMPLOYEE)

ตารางพนักงานจะแสดงข้อมูลพนักงานทั้งหมด ตามหมายเลขพนักงาน และข้อมูลประจำตัวทั่วไป

ตารางพนักงานสามารถสร้างด้วยคำสั่งCREATE TABLE และ ALTER TABLE ต่อไปนี้:

```
CREATE TABLE EMPLOYEE
      (EMPNO     CHAR(6)          NOT NULL,
       FIRSTNME    VARCHAR(12)     NOT NULL,
       MIDINIT     CHAR(1)         NOT NULL,
       LASTNAME    VARCHAR(15)     NOT NULL,
       WORKDEPT    CHAR(3)                 ,
       PHONENO     CHAR(4)                 ,
       HIREDATE    DATE                    ,
       JOB         CHAR(8)                 ,
       EDLEVEL     SMALLINT        NOT NULL,
       SEX         CHAR(1)                 ,
       BIRTHDATE   DATE                    ,
       SALARY      DECIMAL(9,2)            ,
       BONUS       DECIMAL(9,2)            ,
       COMM        DECIMAL(9,2)
```

```
       PRIMARY KEY (EMPNO))

ALTER TABLE EMPLOYEE
      ADD FOREIGN KEY RED (WORKDEPT)
      REFERENCES DEPARTMENT
      ON DELETE SET NULL

ALTER TABLE EMPLOYEE
      ADD CONSTRAINT NUMBER
      CHECK (PHONENO >= '0000' AND PHONENO <= '9999')
```

มีการสร้างดรรชนีต่อไปนี้:

```
CREATE UNIQUE INDEX XEMP1
       ON EMPLOYEE (EMPNO)

CREATE INDEX XEMP2
       ON EMPLOYEE (WORKDEPT)
```

มีการสร้าง alias ต่อไปนี้ให้กับตาราง:

```
CREATE ALIAS EMP FOR EMPLOYEE
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ชื่อคอลัมน์	รายละเอียด
EMPNO	หมายเลขพนักงาน
FIRSTNME	ชื่อแรกของพนักงาน
MIDINIT	ตัวขึ้นต้นชื่อกลางของพนักงาน
LASTNAME	นามสกุลของพนักงาน
WORKDEPT	ID ของแผนที่พนักงานทำงานอยู่
PHONENO	หมายเลขโทรศัพท์ของพนักงาน
HIREDATE	วันที่ว่าจ้าง
JOB	ตำแหน่งงานของพนักงาน
EDLEVEL	จำนวนปีการศึกษาตามที่บังคับ
SEX	เพศของพนักงาน (ช หรือ ญ)
BIRTHDATE	วันเกิด
SALARY	เงินเดือนต่อปีเป็นดอลลาร์
BONUS	โบนัสต่อปีเป็นดอลลาร์
COMM	คอมมิชชันต่อปีเป็นดอลลาร์

EMPLOYEE:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง EMPLOYEE

EMP NO	FIRST NAME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIRE DATE	JOB	ED LEVEL	SEX	BIRTH DATE	SAL-ARY	BONUS	COMM
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01	PRES	18	F	1933-08-24	52750	1000	4220
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250	800	3300
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250	800	3060
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175	800	3214
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250	500	2580
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170	700	2893
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750	600	2380
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150	500	2092
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500	900	3720
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250	600	2340
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800	500	1904
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280	500	2022
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250	400	1780
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340	500	1707
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450	400	1636
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740	600	2217
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270	400	1462
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180	400	1774
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180	400	1534
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250	300	1380
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380	500	2190
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340	300	1227
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750	400	1420
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FILEREP	16	M	1932-08-11	19950	400	1596
000330	WING		LEE	E21	2103	1976-02-23	FILEREP	14	M	1941-07-18	25370	500	2030
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FILEREP	16	M	1926-05-17	23840	500	1907
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01	SALESREP	18	F	1933-08-14	46500	1000	4220
200120	GREG		ORLANDO	A00	2167	1972-05-05	CLERK	14	M	1942-10-18	29250	600	2340
200140	KIM	N	NATZ	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
200220	REBA	K	JOHN	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
200330	HELENA		WONG	E21	2103	1976-02-23	FIELDREP	14	F	1941-07-18	25370	500	2030
200340	ROY	R	ALONZO	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840	500	1907

ตารางภาพถ่ายพนักงาน (EMP_PHOTO)

ตารางภาพถ่ายพนักงานประกอบด้วยภาพถ่ายของพนักงาน ซึ่งระบุโดยหมายเลขพนักงาน

สร้างตารางภาพถ่ายพนักงานด้วยคำสั่ง CREATE TABLE และ ALTER TABLE ต่อไปนี้:

```
CREATE TABLE EMP_PHOTO
            (EMPNO     CHAR(6)          NOT NULL,
             PHOTO_FORMAT VARCHAR(10) NOT NULL,
             PICTURE BLOB(100K),
             EMP_ROWID CHAR(40) NOT NULL DEFAULT '',
             PRIMARY KEY (EMPNO,PHOTO_FORMAT))

ALTER TABLE EMP_PHOTO
```

```
            ADD COLUMN DL_PICTURE DATALINK(1000)
                LINKTYPE URL NO LINK CONTROL

ALTER TABLE EMP_PHOTO
              ADD FOREIGN KEY (EMPNO)
              REFERENCES EMPLOYEE
              ON DELETE RESTRICT
```

ดรรชนีต่อไปนี้จะถูกสร้าง:

```
CREATE UNIQUE INDEX XEMP_PHOTO
              ON EMP_PHOTO (EMPNO,PHOTO_FORMAT)
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ชื่อคอลัมน์	รายละเอียด
EMPNO	หมายเลขพนักงาน
PHOTO_FORMAT	รูปแบบสำหรับภาพที่เก็บไว้ใน PICTURE
PICTURE	ภาพถ่าย
EMP_ROWID	รหัสแถว, ไม่ได้ถูกใช้ในตอนนี้

EMP_PHOTO:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง EMP_PHOTO

EMPNO	PHOTO_FORMAT	PICTURE	EMP_ROWID
000130	bitmap	?	
000130	gif	?	
000140	bitmap	?	
000140	gif	?	
000150	bitmap	?	
000150	gif	?	
000190	bitmap	?	
000190	gif	?	

ตารางประวัติพนักงาน (EMP_RESUME)

ตารางประวัติพนักงานประกอบด้วยประวัติของ พนักงานแต่ละคน ซึ่งระบุโดยหมายเลขพนักงาน

ตารางประวัติพนักงานสร้างด้วยคำสั่ง CREATE TABLE และ ALTER TABLE ต่อไปนี้:

```
CREATE TABLE EMP_RESUME
            (EMPNO CHAR(6) NOT NULL,
             RESUME_FORMAT VARCHAR(10) NOT NULL,
```

```
              RESUME CLOB(5K),
              EMP_ROWID CHAR(40) NOT NULL DEFAULT '',
              PRIMARY KEY (EMPNO,RESUME_FORMAT))

ALTER TABLE EMP_RESUME
            ADD COLUMN DL_RESUME DATALINK(1000)
                LINKTYPE URL NO LINK CONTROL

ALTER TABLE EMP_RESUME
              ADD FOREIGN KEY (EMPNO)
              REFERENCES EMPLOYEE
              ON DELETE RESTRICT
```

ดรรชนีต่อไปนี้จะถูกสร้าง:

```
CREATE UNIQUE INDEX XEMP_RESUME
              ON EMP_RESUME (EMPNO,RESUME_FORMAT)
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ชื่อคอลัมน์	รายละเอียด
EMPNO	หมายเลขพนักงาน
RESUME_FORMAT	รูปแบบข้อความที่เก็บไว้ใน RESUME
RESUME	ประวัติ
EMP_ROWID	รหัสแถว, ไม่ได้ถูกใช้ในตอนนี้

EMP_RESUME:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง EMP_RESUME

EMPNO	RESUME_FORMAT	RESUME	EMP_ROWID
000130	ascii	?	
000130	html	?	
000140	ascii	?	
000140	html	?	
000150	ascii	?	
000150	html	?	
000190	ascii	?	
000190	html	?	

ตารางพนักงานต่อกิจกรรมโครงการ (EMPPROJACT)

ตารางพนักงานต่อกิจกรรมโครงการจะระบุถึง พนักงานที่ทำงานในแต่ละกิจกรรมสำหรับแต่ละโครงการ ระดับความเกี่ยวข้องของพนักงาน (ประจำหรือพาร์ทไทม์) และกำหนดการสำหรับกิจกรรมยังอยู่ในตารางนี้เช่นกัน

ตารางพนักงานต่อกิจกรรมโครงการสามารถสร้างขึ้นด้วยคำสั่ง CREATE TABLE และ ALTER TABLE ต่อไปนี้:

```
CREATE TABLE EMPPROJACT
      (EMPNO     CHAR(6)          NOT NULL,
       PROJNO    CHAR(6)          NOT NULL,
       ACTNO     SMALLINT         NOT NULL,
       EMPTIME   DECIMAL(5,2)             ,
       EMSTDATE  DATE                     ,
       EMENDATE  DATE                     )

ALTER TABLE EMPPROJACT
             ADD FOREIGN KEY REPAPA (PROJNO, ACTNO, EMSTDATE)
             REFERENCES PROJACT
             ON DELETE RESTRICT
```

aliase ต่อไปนี้จะถูกสร้างขึ้นให้กับตาราง:

```
CREATE ALIAS EMPACT FOR EMPPROJACT

CREATE ALIAS EMP_ACT FOR EMPPROJACT
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ตารางที่ 53. คอลัมน์ของพนักงานต่อกิจกรรมโครงการ

ชื่อคอลัมน์	รายละเอียด
EMPNO	รหัสพนักงาน
PROJNO	PROJNO ของโครงการที่มอบหมายให้พนักงาน
ACTNO	ID ของกิจกรรมภายในโครงการที่มอบหมายให้พนักงาน
EMPTIME	สัดส่วนเวลาเต็มของพนักงาน (ระหว่าง 0.00 ถึง 1.00) ที่ต้องใช้ในโครงการจาก EMSTDATE ถึง EMENDATE
EMSTDATE	วันเริ่มต้นกิจกรรม
EMENDATE	วันเสร็จสิ้นกิจกรรม

EMPPROJACT:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง EMPPROJACT

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000010	AD3100	10	.50	1982-01-01	1982-07-01
000070	AD3110	10	1.00	1982-01-01	1983-02-01

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000230	AD3111	60	1.00	1982-01-01	1982-03-15
000230	AD3111	60	.50	1982-03-15	1982-04-15
000230	AD3111	70	.50	1982-03-15	1982-10-15
000230	AD3111	80	.50	1982-04-15	1982-10-15
000230	AD3111	180	.50	1982-10-15	1983-01-01
000240	AD3111	70	1.00	1982-02-15	1982-09-15
000240	AD3111	80	1.00	1982-09-15	1983-01-01
000250	AD3112	60	1.00	1982-01-01	1982-02-01
000250	AD3112	60	.50	1982-02-01	1982-03-15
000250	AD3112	60	1.00	1983-01-01	1983-02-01
000250	AD3112	70	.50	1982-02-01	1982-03-15
000250	AD3112	70	1.00	1982-03-15	1982-08-15
000250	AD3112	70	.25	1982-08-15	1982-10-15
000250	AD3112	80	.25	1982-08-15	1982-10-15
000250	AD3112	80	.50	1982-10-15	1982-12-01
000250	AD3112	180	.50	1982-08-15	1983-01-01
000260	AD3113	70	.50	1982-06-15	1982-07-01
000260	AD3113	70	1.00	1982-07-01	1983-02-01
000260	AD3113	80	1.00	1982-01-01	1982-03-01
000260	AD3113	80	.50	1982-03-01	1982-04-15
000260	AD3113	180	.50	1982-03-01	1982-04-15
000260	AD3113	180	1.00	1982-04-15	1982-06-01
000260	AD3113	180	1.00	1982-06-01	1982-07-01
000270	AD3113	60	.50	1982-03-01	1982-04-01
000270	AD3113	60	1.00	1982-04-01	1982-09-01
000270	AD3113	60	.25	1982-09-01	1982-10-15
000270	AD3113	70	.75	1982-09-01	1982-10-15
000270	AD3113	70	1.00	1982-10-15	1983-02-01
000270	AD3113	80	1.00	1982-01-01	1982-03-01

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000270	AD3113	80	.50	1982-03-01	1982-04-01
000030	IF1000	10	.50	1982-06-01	1983-01-01
000130	IF1000	90	1.00	1982-10-01	1983-01-01
000130	IF1000	100	.50	1982-10-01	1983-01-01
000140	IF1000	90	.50	1982-10-01	1983-01-01
000030	IF2000	10	.50	1982-01-01	1983-01-01
000140	IF2000	100	1.00	1982-01-01	1982-03-01
000140	IF2000	100	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-10-01	1983-01-01
000010	MA2100	10	.50	1982-01-01	1982-11-01
000110	MA2100	20	1.00	1982-01-01	1983-03-01
000010	MA2110	10	1.00	1982-01-01	1983-02-01
000200	MA2111	50	1.00	1982-01-01	1982-06-15
000200	MA2111	60	1.00	1982-06-15	1983-02-01
000220	MA2111	40	1.00	1982-01-01	1983-02-01
000150	MA2112	60	1.00	1982-01-01	1982-07-15
000150	MA2112	180	1.00	1982-07-15	1983-02-01
000170	MA2112	60	1.00	1982-01-01	1983-06-01
000170	MA2112	70	1.00	1982-06-01	1983-02-01
000190	MA2112	70	1.00	1982-01-01	1982-10-01
000190	MA2112	80	1.00	1982-10-01	1983-10-01
000160	MA2113	60	1.00	1982-07-15	1983-02-01
000170	MA2113	80	1.00	1982-01-01	1983-02-01
000180	MA2113	70	1.00	1982-04-01	1982-06-15
000210	MA2113	80	.50	1982-10-01	1983-02-01
000210	MA2113	180	.50	1982-10-01	1983-02-01
000050	OP1000	10	.25	1982-01-01	1983-02-01
000090	OP1010	10	1.00	1982-01-01	1983-02-01

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000280	OP1010	130	1.00	1982-01-01	1983-02-01
000290	OP1010	130	1.00	1982-01-01	1983-02-01
000300	OP1010	130	1.00	1982-01-01	1983-02-01
000310	OP1010	130	1.00	1982-01-01	1983-02-01
000050	OP2010	10	.75	1982-01-01	1983-02-01
000100	OP2010	10	1.00	1982-01-01	1983-02-01
000320	OP2011	140	.75	1982-01-01	1983-02-01
000320	OP2011	150	.25	1982-01-01	1983-02-01
000330	OP2012	140	.25	1982-01-01	1983-02-01
000330	OP2012	160	.75	1982-01-01	1983-02-01
000340	OP2013	140	.50	1982-01-01	1983-02-01
000340	OP2013	170	.50	1982-01-01	1983-02-01
000020	PL2100	30	1.00	1982-01-01	1982-09-15

ตารางโครงการ (PROJECT)

ตารางโครงการจะอธิบายถึงโครงการแต่ละอย่างที่ธุรกิจดำเนินการอยู่ในปัจจุบัน. ข้อมูลที่อยู่ในแต่ละแถวประกอบด้วยหมายเลขโครงการ, ชื่อ, บุคคลที่รับผิดชอบ, และวันที่ตามกำหนดการ.

ตารางโครงการถูกสร้างด้วยคำสั่ง CREATE TABLE และ ALTER TABLE ต่อไปนี้:

```
CREATE TABLE PROJECT
      (PROJNO    CHAR(6)        NOT NULL,
       PROJNAME  VARCHAR(24)    NOT NULL DEFAULT,
       DEPTNO    CHAR(3)        NOT NULL,
       RESPEMP   CHAR(6)        NOT NULL,
       PRSTAFF   DECIMAL(5,2)          ,
       PRSTDATE  DATE                  ,
       PRENDATE  DATE                  ,
       MAJPROJ   CHAR(6)               ,
       PRIMARY KEY (PROJNO))

ALTER TABLE PROJECT
             ADD FOREIGN KEY (DEPTNO)
             REFERENCES DEPARTMENT
             ON DELETE RESTRICT

ALTER TABLE PROJECT
             ADD FOREIGN KEY (RESPEMP)
             REFERENCES EMPLOYEE
             ON DELETE RESTRICT
```

```
ALTER TABLE PROJECT
            ADD FOREIGN KEY  RPP (MAJPROJ)
            REFERENCES PROJECT
            ON DELETE CASCADE
```

มีการสร้างดรรชนีต่อไปนี้:

```
CREATE UNIQUE INDEX XPROJ1
            ON PROJECT (PROJNO)

CREATE INDEX XPROJ2
            ON PROJECT (RESPEMP)
```

มีการสร้าง alias ต่อไปนี้ให้กับตาราง:

```
CREATE ALIAS PROJ FOR PROJECT
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์:

ชื่อคอลัมน์	รายละเอียด
PROJNO	หมายเลขโครงการ
PROJNAME	ชื่อโครงการ
DEPTNO	หมายเลขแผนกของแผนกที่รับผิดชอบโครงการ
RESPEMP	หมายเลขพนักงานของพนักงานที่รับผิดชอบโครงการ
PRSTAFF	จำนวนพนักงานโดยประมาณ
PRSTDATE	วันที่เริ่มต้นโครงการโดยประมาณ
PRENDATE	วันที่สิ้นสุดโครงการโดยประมาณ
MAJPROJ	หมายเลขควบคุมโครงการสำหรับโครงการย่อย

PROJECT:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง PROJECT

PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
AD3100	ADMIN SERVICES	D01	000010	6.5	1982-01-01	1983-02-01	?
AD3110	GENERAL ADMIN SYSTEMS	D21	000070	6	1982-01-01	1983-02-01	AD3100
AD3111	PAYROLL PROGRAMMING	D21	000230	2	1982-01-01	1983-02-01	AD3110
AD3112	PERSONNEL PROGRAMMING	D21	000250	1	1982-01-01	1983-02-01	AD3110
AD3113	ACCOUNT PROGRAMMING	D21	000270	2	1982-01-01	1983-02-01	AD3110

PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
IF1000	QUERY SERVICES	C01	000030	2	1982-01-01	1983-02-01	?
IF2000	USER EDUCATION	C01	000030	1	1982-01-01	1983-02-01	?
MA2100	WELD LINE AUTOMATION	D01	000010	12	1982-01-01	1983-02-01	?
MA2110	W L PROGRAMMING	D11	000060	9	1982-01-01	1983-02-01	MA2100
MA2111	W L PROGRAM DESIGN	D11	000220	2	1982-01-01	1982-12-01	MA2110
MA2112	W L ROBOT DESIGN	D11	000150	3	1982-01-01	1982-12-01	MA2110
MA2113	W L PROD CONT PROGS	D11	000160	3	1982-02-15	1982-12-01	MA2110
OP1000	OPERATION SUPPORT	E01	000050	6	1982-01-01	1983-02-01	?
OP1010	OPERATION	E11	000090	5	1982-01-01	1983-02-01	OP1000
OP2000	GEN SYSTEMS SERVICES	E01	000050	5	1982-01-01	1983-02-01	?
OP2010	SYSTEMS SUPPORT	E21	000100	4	1982-01-01	1983-02-01	OP2000
OP2011	SCP SYSTEMS SUPPORT	E21	000320	1	1982-01-01	1983-02-01	OP2010
OP2012	APPLICATIONS SUPPORT	E21	000330	1	1982-01-01	1983-02-01	OP2010
OP2013	DB/DC SUPPORT	E21	000340	1	1982-01-01	1983-02-01	OP2010
PL2100	WELD LINE PLANNING	B01	000020	1	1982-01-01	1982-09-15	MA2100

ตารางกิจกรรมโครงการ (PROJACT)

ตารางกิจกรรมโครงการจะอธิบายถึงกิจกรรมโครงการแต่ละอย่างที่ธุรกิจดำเนินการอยู่ในปัจจุบัน ข้อมูลในแต่ละแถวประกอบด้วยหมายเลขโครงการ, หมายเลขกิจกรรม, และวันที่ตามกำหนดการ

สร้างตารางกิจกรรมโครงการ ด้วยคำสั่ง CREATE TABLE และ ALTER TABLE ต่อไปนี้:

```
CREATE TABLE PROJACT
            (PROJNO CHAR(6) NOT NULL,
             ACTNO SMALLINT NOT NULL,
             ACSTAFF DECIMAL(5,2),
             ACSTDATE DATE NOT NULL,
             ACENDATE DATE ,
```

```
            PRIMARY KEY (PROJNO, ACTNO, ACSTDATE))

ALTER TABLE PROJACT
              ADD FOREIGN KEY RPAP (PROJNO)
              REFERENCES PROJECT
              ON DELETE RESTRICT
```

มีการเพิ่มคีย์ foreign ต่อไปนี้ในภายหลัง:

```
ALTER TABLE PROJACT
            ADD FOREIGN KEY RPAA (ACTNO)
                REFERENCES ACT
                ON DELETE RESTRICT
```

ดรรชนีต่อไปนี้จะถูกสร้าง:

```
CREATE UNIQUE INDEX XPROJAC1
              ON PROJACT (PROJNO, ACTNO, ACSTDATE)
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์:

ชื่อคอลัมน์	รายละเอียด
PROJNO	หมายเลขโครงการ
ACTNO	หมายเลขกิจกรรม
ACSTAFF	จำนวนพนักงานโดยประมาณ
ACSTDATE	วันที่เริ่มกิจกรรม
ACENDATE	วันที่สิ้นสุดกิจกรรม

PROJACT:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง PROJACT

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3100	10	?	1982-01-01	?
AD3110	10	?	1982-01-01	?
AD3111	60	?	1982-01-01	?
AD3111	60	?	1982-03-15	?
AD3111	70	?	1982-03-15	?
AD3111	80	?	1982-04-15	?
AD3111	180	?	1982-10-15	?
AD3111	70	?	1982-02-15	?
AD3111	80	?	1982-09-15	?

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3112	60	?	1982-01-01	?
AD3112	60	?	1982-02-01	?
AD3112	60	?	1983-01-01	?
AD3112	70	?	1982-02-01	?
AD3112	70	?	1982-03-15	?
AD3112	70	?	1982-08-15	?
AD3112	80	?	1982-08-15	?
AD3112	80	?	1982-10-15	?
AD3112	180	?	1982-08-15	?
AD3113	70	?	1982-06-15	?
AD3113	70	?	1982-07-01	?
AD3113	80	?	1982-01-01	?
AD3113	80	?	1982-03-01	?
AD3113	180	?	1982-03-01	?
AD3113	180	?	1982-04-15	?
AD3113	180	?	1982-06-01	?
AD3113	60	?	1982-03-01	?
AD3113	60	?	1982-04-01	?
AD3113	60	?	1982-09-01	?
AD3113	70	?	1982-09-01	?
AD3113	70	?	1982-10-15	?
IF1000	10	?	1982-06-01	?
IF1000	90	?	1982-10-01	?
IF1000	100	?	1982-10-01	?
IF2000	10	?	1982-01-01	?
IF2000	100	?	1982-01-01	?
IF2000	100	?	1982-03-01	?
IF2000	110	?	1982-03-01	?
IF2000	110	?	1982-10-01	?

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
MA2100	10	?	1982-01-01	?
MA2100	20	?	1982-01-01	?
MA2110	10	?	1982-01-01	?
MA2111	50	?	1982-01-01	?
MA2111	60	?	1982-06-15	?
MA2111	40	?	1982-01-01	?
MA2112	60	?	1982-01-01	?
MA2112	180	?	1982-07-15	?
MA2112	70	?	1982-06-01	?
MA2112	70	?	1982-01-01	?
MA2112	80	?	1982-10-01	?
MA2113	60	?	1982-07-15	?
MA2113	80	?	1982-01-01	?
MA2113	70	?	1982-04-01	?
MA2113	80	?	1982-10-01	?
MA2113	180	?	1982-10-01	?
OP1000	10	?	1982-01-01	?
OP1010	10	?	1982-01-01	?
OP1010	130	?	1982-01-01	?
OP2010	10	?	1982-01-01	?
OP2011	140	?	1982-01-01	?
OP2011	150	?	1982-01-01	?
OP2012	140	?	1982-01-01	?
OP2012	160	?	1982-01-01	?
OP2013	140	?	1982-01-01	?
OP2013	170	?	1982-01-01	?
PL2100	30	?	1982-01-01	?

ตารางกิจกรรม (ACT)

ตารางกิจกรรมจะอธิบายรายละเอียดของแต่ละกิจกรรม.

สามารถสร้างตารางกิจกรรมด้วยคำสั่ง CREATE TABLE ต่อไปนี้:

```
CREATE TABLE ACT
            (ACTNO SMALLINT NOT NULL,
             ACTKWD CHAR(6) NOT NULL,
             ACTDESC VARCHAR(20) NOT NULL,
             PRIMARY KEY (ACTNO))
```

มีการสร้างดรรชนีต่อไปนี้:

```
CREATE UNIQUE INDEX XACT1
             ON ACT (ACTNO)

CREATE UNIQUE INDEX XACT2
             ON ACT (ACTKWD)
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ชื่อคอลัมน์	รายละเอียด
ACTNO	หมายเลขกิจกรรม
ACTKWD	คีย์เวิร์ดสำหรับกิจกรรม
ACTDESC	รายละเอียดของกิจกรรม

ACT:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง ACT

ACTNO	ACTKWD	ACTDESC
10	MANAGE	MANAGE/ADVISE
20	ECOST	ESTIMATE COST
30	DEFINE	DEFINE SPECS
40	LEADPR	LEAD PROGRAM/DESIGN
50	SPECS	WRITE SPECS
60	LOGIC	DESCRIBE LOGIC
70	CODE	CODE PROGRAMS
80	TEST	TEST PROGRAMS
90	ADMQS	ADM QUERY SYSTEM
100	TEACH	TEACH CLASSES
110	COURSE	DEVELOP COURSES

ACTNO	ACTKWD	ACTDESC
120	STAFF	PERS AND STAFFING
130	OPERAT	OPER COMPUTER SYS
140	MAINT	MAINT SOFTWARE SYS
150	ADMSYS	ADM OPERATING SYS
160	ADMDB	ADM DATA BASES
170	ADMDC	ADM DATA COMM
180	DOC	DOCUMENT

ตารางการกำหนดเวลาเรียน (CL_SCHED)

ตารางการกำหนดเวลาเรียนจะอธิบายแต่ละชั้นเรียน, เวลาเริ่มต้นของชั้นเรียน, เวลาสิ้นสุดของชั้นเรียน, และโค้ดของชั้นเรียน.

สามารถสร้างตารางกำหนดการเรียน ด้วยคำสั่ง CREATE TABLE ต่อไปนี้:

```
CREATE TABLE CL_SCHED
      (CLASS_CODE          CHAR(7),
       "DAY"               SMALLINT,
       STARTING            TIME,
       ENDING              TIME)
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ชื่อคอลัมน์	รายละเอียด
CLASS_CODE	โค้ดของชั้นเรียน (ห้อง:อาจารย์)
DAY	หมายเลขวันของกำหนดการ 4 วัน
STARTING	เวลาเริ่มต้นของชั้นเรียน
ENDING	เวลาสิ้นสุดชั้นเรียน

CL_SCHED:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง CL_SCHED

CLASS_CODE	DAY	STARTING	ENDING
042:BF	4	12:10:00	14:00:00
553:MJA	1	10:30:00	11:00:00
543:CWM	3	09:10:00	10:30:00
778:RES	2	12:10:00	14:00:00
044:HD	3	17:12:30	18:00:00

ตาราง In-tray (IN_TRAY)

ตาราง in-tray จะให้รายละเอียดของตะกร้ารับข้อมูลแบบอิเล็กทรอนิก ซึ่ง ประกอบด้วย timestamp แสดงเวลาที่รับข้อความ, user ID ของ บุคคลที่ส่งข้อความ และตัวข้อความ

สามารถสร้างตาราง in tray ด้วยคำสั่ง CREATE TABLE ต่อไปนี้:

```
CREATE TABLE IN_TRAY
    (RECEIVED           TIMESTAMP,
     SOURCE             CHAR(8),
     SUBJECT            CHAR(64),
     NOTE_TEXT          VARCHAR(3000))
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ชื่อคอลัมน์	รายละเอียด
RECEIVED	วันและเวลาที่ได้รับ
SOURCE	user ID ของบุคคลที่ส่งหมายเหตุ
SUBJECT	รายละเอียดโดยย่อของหมายเหตุ
NOTE_TEXT	หมายเหตุ

IN_TRAY:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง IN_TRAY

RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
1988-12-25-17.12.30.000000	BADAMSON	FWD: Fanstastic year! 4th Quarter Bonus.	To: JWALKER Cc: QUINTANA, NICHOLLS Jim, Looks like our hard work has paid off . I have some good beer in the fridge if you want to come over to celebrate a bit. Delores and Heather, are you interested as well ? Bruce <Forwarding from ISTERN> Subject: FWD: Fantastic year! 4th Quarter Bonus. To: Dept_D11 Congratulations on a job well done. Enjoy this year's bonus. Irv <Forwarding from CHAAS> Subject: Fantastic year! 4th Quarter Bonus. To: All_Managers Our 4th quarter results are in. We pulled together as a team and exceeded our plan! I am pleased to announce a bonus this year of 18%. Enjoy the holidays. Christine Haas

RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
1988-12-23-08.53.58.000000	ISTERN	FWD: Fanstastic year! 4th Quarter Bonus.	To: Dept_D11 Congratulations on a job well done. Enjoy this year's bonus. Irv <Forwarding from CHAAS> Subject: Fantastic year! 4th Quarter Bonus. To: All_Managers Our 4th quarter results are in. We pulled together as a team and exceeded our plan! I am pleased to announce a bonus this year of 18%. Enjoy the holidays. Christine Haas
1988-12-22-14.07.21.136421	CHAAS	Fantastic year! 4th Quarter Bonus.	To: All_Managers Our 4th quarter results are in. We pulled together as a team and exceeded our plan! I am pleased to announce a bonus this year of 18%. Enjoy the holidays. Christine Haas

ตารางโครงสร้าง (ORG)

ตารางโครงสร้างจะอธิบายถึงโครงสร้างของบริษัท.

สร้างตารางด้วยคำสั่ง CREATE TABLE ต่อไปนี้:

```
CREATE TABLE ORG
            (DEPTNUMB SMALLINT NOT NULL,
             DEPTNAME VARCHAR(14),
             MANAGER SMALLINT,
             DIVISION VARCHAR(10),
             LOCATION VARCHAR(13))
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ชื่อคอลัมน์	รายละเอียด
DEPTNUMB	หมายเลขแผนก
DEPTNAME	ชื่อแผนก
MANAGER	หมายเลขผู้จัดการของแผนก
DIVISION	หน่วยงานของแผนก
LOCATION	ที่ตั้งของแผนก

ORG:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง ORG

DEPTNUMB	DEPTNAME	MANAGER	DIVISION	LOCATION
10	Head Office	160	Corporate	New York
15	New England	50	Eastern	Boston
20	Mid Atlantic	10	Eastern	Washington
38	South Atlantic	30	Eastern	Atlanta
42	Great Lakes	100	Midwest	Chicago
51	Plains	140	Midwest	Dallas
66	Pacific	270	Western	San Francisco
84	Mountain	290	Western	Denver

ตารางพนักงาน (STAFF)

ตารางพนักงานจะให้รายละเอียดเกี่ยวกับข้อมูลแบ็กกราวน์ ของพนักงาน

สร้างตารางพนักงานด้วยคำสั่ง CREATE TABLE ต่อไปนี้:

```
CREATE TABLE STAFF
            (ID  SMALLINT NOT NULL,
             NAME VARCHAR(9),
             DEPT SMALLINT,
             JOB CHAR(5),
             YEARS SMALLINT,
             SALARY DECIMAL(7,2),
             COMM DECIMAL(7,2))
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ชื่อคอลัมน์	รายละเอียด
ID	หมายเลขพนักงาน
NAME	ชื่อพนักงาน
DEPT	หมายเลขแผนก
JOB	ตำแหน่งงาน
YEARS	จำนวนปีที่ทำงานกับบริษัท
SALARY	เงินเดือนต่อปีของพนักงาน
COMM	ค่านายหน้าของพนักงาน

STAFF:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง STAFF

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	?
20	Pernal	20	Sales	8	18171.25	612.45
30	Marenghi	38	Mgr	5	17506.75	?
40	O'Brien	38	Sales	6	18006.00	846.55
50	Hanes	15	Mgr	10	20659.80	?
60	Quigley	38	Sales	7	16508.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
80	James	20	Clerk	?	13504.60	128.20
90	Koonitz	42	Sales	6	18001.75	1386.70
100	Plotz	42	Mgr	7	18352.80	?
110	Ngan	15	Clerk	5	12508.20	206.60
120	Naughton	38	Clerk	?	12954.75	180.00
130	Yamaguchi	42	Clerk	6	10505.90	75.60
140	Fraye	51	Mgr	6	21150.00	?
150	Williams	51	Sales	6	19456.50	637.65
160	Molinare	10	Mgr	7	22959.20	?
170	Kermisch	15	Clerk	4	12258.50	110.10
180	Abrahams	38	Clerk	3	12009.75	236.50
190	Sneider	20	Clerk	8	14252.75	126.50
200	Scoutten	42	Clerk	?	11508.60	84.20
210	Lu	10	Mgr	10	20010.00	?
220	Smith	51	Sales	7	17654.50	992.80
230	Lundquist	51	Clerk	3	13369.80	189.65
240	Daniels	10	Mgr	5	19260.25	?
250	Wheeler	51	Clerk	6	14460.00	513.30
260	Jones	10	Mgr	12	21234.00	?
270	Lea	66	Mgr	9	18555.50	?
280	Wilson	66	Sales	9	18674.50	811.50
290	Quill	84	Mgr	10	19818.00	?

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
300	Davis	84	Sales	5	15454.50	806.10
310	Graham	66	Sales	13	21000.00	200.30
320	Gonzales	66	Sales	4	16858.20	844.00
330	Burke	66	Clerk	1	10988.00	55.50
340	Edwards	84	Sales	7	17844.00	1285.00
3650	Gafney	84	Clerk	5	13030.50	188.00

ตารางยอดขาย (SALES)

ตารางยอดขายอธิบายถึงข้อมูลของการขาย สำหรับพนักงานขายแต่ละคน

สร้างตารางยอดขายด้วยคำสั่ง CREATE TABLE ต่อไปนี้:

```
CREATE TABLE SALES
            (SALES_DATE DATE,
             SALES_PERSON VARCHAR(15),
             REGION VARCHAR(15),
             SALES INTEGER)
```

ตารางต่อไปนี้แสดงถึงรายละเอียดของคอลัมน์.

ชื่อคอลัมน์	รายละเอียด
SALES_DATE	วันที่ขาย
SALES_PERSON	บุคคลที่ขาย
REGION	ภูมิภาคที่มีการขาย
SALES	จำนวนยอดขาย

SALES:

รายการของข้อมูลที่สมบูรณ์ใน ตาราง SALES

SALES_DATE	SALES_PERSON	REGION	SALES
12/31/1995	LUCCHESSI	Ontario-South	1
12/31/1995	LEE	Ontario-South	3
12/31/1995	LEE	Quebec	1
12/31/1995	LEE	Manitoba	2
12/31/1995	GOUNOT	Quebec	1
03/29/1996	LUCCHESSI	Ontario-South	3

SALES_DATE	SALES_PERSON	REGION	SALES
03/29/1996	LUCCHESSI	Quebec	1
03/29/1996	LEE	Ontario-South	2
03/29/1996	LEE	Ontario-North	2
03/29/1996	LEE	Quebec	3
03/29/1996	LEE	Manitoba	5
03/29/1996	GOUNOT	Ontario-South	3
03/29/1996	GOUNOT	Quebec	1
03/29/1996	GOUNOT	Manitoba	7
03/30/1996	LUCCHESSI	Ontario-South	1
03/30/1996	LUCCHESSI	Quebec	2
03/30/1996	LUCCHESSI	Manitoba	1
03/30/1996	LEE	Ontario-South	7
03/30/1996	LEE	Ontario-North	3
03/30/1996	LEE	Quebec	7
03/30/1996	LEE	Manitoba	4
03/30/1996	GOUNOT	Ontario-South	2
03/30/1996	GOUNOT	Quebec	18
03/30/1996	GOUNOT	Manitoba	1
03/31/1996	LUCCHESSI	Manitoba	1
03/31/1996	LEE	Ontario-South	14
03/31/1996	LEE	Ontario-North	3
03/31/1996	LEE	Quebec	7
03/31/1996	LEE	Manitoba	3
03/31/1996	GOUNOT	Ontario-South	2
03/31/1996	GOUNOT	Quebec	1
04/01/1996	LUCCHESSI	Ontario-South	3
04/01/1996	LUCCHESSI	Manitoba	1
04/01/1996	LEE	Ontario-South	8
04/01/1996	LEE	Ontario-North	?

SALES_DATE	SALES_PERSON	REGION	SALES
04/01/1996	LEE	Quebec	8
04/01/1996	LEE	Manitoba	9
04/01/1996	GOUNOT	Ontario-South	3
04/01/1996	GOUNOT	Ontario-North	1
04/01/1996	GOUNOT	Quebec	3
04/01/1996	GOUNOT	Manitoba	7

DB2 for i5/OSคำอธิบายของคำสั่ง CL

DB2 for i5/OS จัดหา คำสั่ง CL เหล่านี้สำหรับ SQL

- คำสั่ง Create SQL Package (CRTSQLPKG)
- คำสั่ง Delete SQL Package (DLTSQLPKG)
- คำสั่ง Print SQL Information (PRTSQLINF)
- คำสั่ง Run SQL Statement (RUNSQLSTM)
- คำสั่ง Start SQL Interactive Session (STRSQL)

สิ่งอ้างอิงที่เกี่ยวข้อง

"DB2 for i5/OS การสนับสนุนฐานข้อมูลเชิงสัมพันธ์แบบกระจาย" ในหน้า 303
ไลเซนส์โปรแกรม IBM DB2 Query Manager and SQL Development Kit for i5/OS สนับสนุนการเข้าถึงแบบโต้ตอบของฐานข้อมูลแบบกระจาย

สิทธิในรหัส และข้อมูลถ้อยแถลง

IBM มอบสิทธิในลิขสิทธิ์แบบไม่เจาะจงในการใช้ตัวอย่างรหัสในการทำโปรแกรมทั้งหมดที่คุณสามารถใช้สร้างฟังก์ชันคล้ายคลึงกันที่ปรับให้เหมาะกับความต้องการเฉพาะของคุณได้

ขึ้นอยู่กับการรับประกันตามข้อบังคับใดที่ไม่สามารถยกเว้นได้ IBM, เจ้าหน้าที่พัฒนาโปรแกรมของ IBM และซัพพลายเออร์ไม่รับรอง หรือกำหนดเงื่อนไขโดยเปิดเผย หรือโดยนัย รวมทั้งแต่ไม่จำกัดเฉพาะการรับประกัน หรือเงื่อนไขโดยนัยในการสามารถซื้อขายได้ ความเหมาะสมกับวัตถุประสงค์บางประการ และการไม่ละเมิดเกี่ยวกับโปรแกรม หรือการสนับสนุนทางเทคนิค หากมี

ในทุกกรณี IBM, เจ้าหน้าที่พัฒนาโปรแกรมของ IBM หรือซัพพลายเออร์ไม่มีความรับผิดต่อลักษณะดังต่อไปนี้ แม้ว่าจะได้รับการบอกกล่าวถึงความเป็นไปได้ในการเกิดลักษณะดังกล่าวได้ก็ตาม:

1. การสูญหาย หรือความเสียหายของข้อมูล
2. ความเสียหายโดยตรง, พิเศษ, โดยบังเอิญ หรือทางอ้อม หรือความเสียหายที่เป็นผลจากเศรษฐกิจ หรือ
3. การสูญกำไร, ธุรกิจ, รายได้, ค่านิยม หรือเงินออมที่คาดว่าจะได้รับ

ในการพิจารณาคดีบางครั้งไม่อนุญาตให้ละเว้น หรือจำกัดความเสียหายโดยตรง, โดยบังเอิญ หรือที่เป็นผลตามมา ดังนั้นข้อจำกัด หรือข้อยกเว้นบางข้อ หรือทั้งหมดข้างต้นอาจไม่มีผลกับคุณ

ภาคผนวก. คำประกาศ

ข้อมูลนี้ถูกพัฒนาขึ้นสำหรับผลิตภัณฑ์และบริการที่เสนอขายในประเทศไทย

IBM อาจไม่สามารถจัดเตรียมผลิตภัณฑ์ บริการ หรือคุณลักษณะพิเศษที่กล่าวถึงในเอกสารนี้ในประเทศอื่นๆ ได้ โปรดปรึกษาตัวแทนของ IBM สำหรับข้อมูลที่เกี่ยวกับผลิตภัณฑ์และบริการที่เสนอขายอยู่ในท้องที่ของคุณ การอ้างอิงเกี่ยวกับผลิตภัณฑ์ โปรแกรม หรือบริการของ IBM มิได้มีเจตนาบอกกล่าว หรือแสดงนัยยะว่าเฉพาะผลิตภัณฑ์ โปรแกรม หรือบริการของ IBM เท่านั้นที่สามารถใช้ได้ ผลิตภัณฑ์ โปรแกรม หรือบริการที่ทำงานได้เท่าเทียมกัน ซึ่งไม่ละเมิดทรัพย์สินทางปัญญาของ IBM อาจสามารถใช้แทนกันได้ อย่างไรก็ตาม เป็นความรับผิดชอบของผู้ใช้ที่จะประเมินผล และตรวจสอบการทำงานของผลิตภัณฑ์ โปรแกรม หรือบริการที่ไม่ใช่ของ IBM

IBM อาจมีสิทธิบัตรหรือเอกสารซึ่งอยู่ระหว่างการดำเนินการขอสิทธิบัตรที่ครอบคลุมถึงประเด็นที่อธิบายไว้ในเอกสารนี้ การตกแต่งเอกสารนี้ใหม่ไม่ได้ทำให้คุณได้รับการอนุญาตจากสิทธิบัตรเหล่านั้น คุณสามารถสอบถามเกี่ยวกับการอนุญาตใช้สิทธิ์ได้ โดยเขียนส่งไปที่ :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

สำหรับการสอบถามการอนุญาตใช้สิทธิ์เกี่ยวกับข้อมูล double-byte (DBCS) ให้ติดต่อ IBM Intellectual Property Department ในประเทศของคุณ หรือส่งคำถามไปที่ :

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

ย่อหน้าต่อไปนี้ไม่มีผลบังคับใช้กับสหราชอาณาจักร หรือประเทศอื่นๆ ที่ซึ่งบริการและผลิตภัณฑ์ไม่สอดคล้องกับกฎหมายท้องถิ่น: INTERNATIONAL BUSINESS MACHINES CORPORATION จัดเตรียมข้อมูลนี้ "ตามสภาพที่เป็น" โดยไม่มีการรับประกันใดๆ ทั้งโดยชัดแจ้งหรือโดยนัย ตลอดจนไม่มีการรับประกันโดยนัยต่อความสามารถในการจัดจำหน่าย การไม่ละเมิด หรือความเหมาะสมสำหรับวัตถุประสงค์อย่างใดอย่างหนึ่ง ทั้งนี้ในบางรัฐไม่อนุญาตให้มีการจำกัดความรับผิดชอบในการรับประกันโดยชัดแจ้ง หรือโดยนัยในการทำธุรกรรมบางอย่าง ดังนั้น ข้อความข้างต้นอาจใช้ไม่ได้กับคุณ

ข้อมูลนี้อาจมีความไม่ถูกต้องทางเทคนิคหรือความผิดพลาดทางการพิมพ์ การเปลี่ยนแปลงข้อมูลในนี้จะมีเป็นระยะๆ ซึ่งจะสอดคล้องกับการตีพิมพ์ในครั้งใหม่ IBM อาจทำการปรับปรุง และ/หรือเปลี่ยนแปลงในผลิตภัณฑ์ และ/หรือโปรแกรมที่ได้อธิบายไว้ในเอกสารนี้ได้ตลอดเวลาโดยไม่ต้องแจ้งให้ทราบล่วงหน้า

การอ้างถึงเว็บไซต์ที่ไม่ใช่ของ IBM นั้นเป็นไปเพื่อวัตถุประสงค์ด้านความสะดวกเท่านั้น และไม่ได้เป็นการรับรองเว็บไซต์เหล่านั้น เนื้อหาที่อยู่ในเว็บไซต์เหล่านั้น ไม่ถือว่าเป็นส่วนหนึ่งของเนื้อหาสำหรับผลิตภัณฑ์ของ IBM นี้ และคุณต้องรับผิดชอบต่อความเสี่ยงในการใช้งานเว็บไซต์ดังกล่าว

IBM อาจใช้งานหรือเผยแพร่ส่วนหนึ่งส่วนใดของข้อมูลที่คุณให้ไว้ไม่ว่าจะด้วยวิธีการใดเมื่อเห็นสมควรโดยไม่ก่อให้เกิดข้อผูกพันใดๆ แก่คุณ

หากผู้ที่ได้รับอนุญาตให้ใช้โปรแกรมนี้ต้องการทราบข้อมูลเกี่ยวกับโปรแกรมเพื่อจุดประสงค์ในการใช้งาน: (1) แลกเปลี่ยนข้อมูลระหว่างโปรแกรมที่ถูกสร้างขึ้นอย่างเป็นอิสระ และโปรแกรมอื่น (รวมถึงโปรแกรมนี้) และ (2) ใช้ข้อมูลร่วมกันซึ่งมีการแลกเปลี่ยน ควรติดต่อ:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

ข้อมูลดังกล่าวอาจมีพร้อมให้ ภายใต้ข้อกำหนดและเงื่อนไขที่เหมาะสม รวมถึงในบางกรณี ที่ต้องมีการชำระค่าธรรมเนียม

โปรแกรมไลเซนส์ที่อธิบายไว้ในเอกสารนี้ รวมถึงข้อมูลทั้งหมดที่มีสำหรับโปรแกรม ถูกจัดเตรียมโดย IBM ภายใต้เงื่อนไขของ IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code หรือข้อตกลงอื่นที่เทียบเท่า

ข้อมูลของประสิทธิภาพการทำงานใดในที่นี้ เป็นข้อมูลที่ได้จากสภาพแวดล้อมที่ถูกควบคุม ดังนั้น ผลที่ได้ในสภาพแวดล้อมอื่นอาจแตกต่างกันได้ ค่าที่วัดได้บางอย่างอาจทำขึ้นในขั้นตอนการพัฒนาระบบ และไม่รับประกันว่าผลที่ได้เหล่านี้จะเหมือนกับระบบที่วางจำหน่าย ยิ่งไปกว่านั้น เกณฑ์บางอย่างอาจได้มาจากการประมาณโดยผ่านกระบวนการ extrapolation ค่าที่ได้จริงอาจแตกต่างกันได้ ผู้ใช้เอกสารนี้ควรตรวจสอบข้อมูลที่ใช้ได้สำหรับสภาพแวดล้อมเฉพาะสำหรับผู้ใช้

ข้อมูลเกี่ยวกับผลิตภัณฑ์ที่ไม่ได้จัดทำโดย IBM ได้รับมาจากซัพพลายเออร์ของผลิตภัณฑ์เหล่านั้น, ประกาศที่เผยแพร่หรือแหล่งข้อมูลที่เปิดเผยต่อสาธารณะ IBM ไม่ได้ทดสอบผลิตภัณฑ์เหล่านั้น และไม่ได้ยืนยันความถูกต้องของประสิทธิภาพการทำงาน ความเข้ากันได้ หรือคำกล่าวอ้างอื่นๆ เกี่ยวกับผลิตภัณฑ์ที่ไม่ได้จัดทำโดย IBM คำถามเกี่ยวกับความเข้ากันได้ของผลิตภัณฑ์ที่ไม่ได้ผลิตโดย IBM ควรแจ้งกับซัพพลายเออร์ของผลิตภัณฑ์เหล่านั้น

ข้อความใดๆ ที่เกี่ยวข้องกับทิศทางในอนาคตและเจตจำนงค์ของ IBM อาจมีการเปลี่ยนแปลง หรือเพิกถอนได้โดยไม่ต้องแจ้งล่วงหน้า และนำเสนอเฉพาะเป้าหมายและวัตถุประสงค์เท่านั้น

ข้อมูลนี้ประกอบด้วยตัวอย่างข้อมูลและรายงานที่ใช้ในการดำเนินธุรกิจประจำวัน เพื่อแสดงให้เห็นอย่างสมบูรณ์ที่สุดเท่าที่เป็นไปได้ ตัวอย่างเหล่านี้จึงประกอบด้วยชื่อของบุคคล บริษัท ตราสินค้า และผลิตภัณฑ์ ชื่อทั้งหมดเหล่านี้เป็นชื่อสมมติ และการคล้ายคลึงในชื่อและที่อยู่หน่วยธุรกิจที่มีอยู่จริงเป็นความบังเอิญทั้งสิ้น

COPYRIGHT LICENSE:

ข้อมูลนี้ประกอบด้วยแอ็พพลิเคชันตัวอย่างในภาษาต้นฉบับ ซึ่งแสดงเทคนิคในการเขียนโปรแกรมบนแพล็ตฟอร์มปฏิบัติการที่หลากหลาย คุณอาจคัดลอก ดัดแปลง หรือเผยแพร่โปรแกรมตัวอย่างเหล่านี้ในรูปแบบใดๆ โดยไม่ต้องจ่ายเงินให้กับ IBM สำหรับวัตถุประสงค์ของการพัฒนา การใช้งาน การตลาด หรือการเผยแพร่โปรแกรมแอ็พพลิเคชันที่ใช้ application programming interface สำหรับแพล็ตฟอร์มปฏิบัติการที่โปรแกรมตัวอย่างได้ถูกพัฒนาขึ้น ตัวอย่างเหล่านี้ ไม่ได้ผ่านการทดสอบภายใต้ทุกสถานการณ์ ดังนั้น IBM ไม่สามารถรับประกัน หรือกล่าวเป็นนัยถึงความเชื่อถือ ความสามารถในการให้บริการ หรือฟังก์ชันการทำงานของโปรแกรมเหล่านี้ได้

แต่ละสำเนาหรือบางส่วนของโปรแกรมตัวอย่าง หรืองานใดๆ ที่มาจากโปรแกรมเหล่านี้ ต้องมีข้อความแสดงลิขสิทธิ์ ดังนี้:

© (ชื่อบริษัทของคุณ) (ปี) บางส่วนของโค้ดนี้ถูกพัฒนามาจากโปรแกรมตัวอย่างของ IBM Corp. © Copyright IBM Corp. _ป้อนปี_. All rights reserved.

หากคุณกำลังอ่านข้อมูลนี้ในรูปแบบที่เป็น softcopy รูปภาพและภาพประกอบสีอาจไม่ปรากฏขึ้น

ข้อมูลเกี่ยวกับโปรแกรมมิงอินเตอร์เฟส

เอกสาร การโปรแกรม SQL เล่มนี้ จัดทำส่วนโปรแกรมมิงอินเตอร์เฟสซึ่งช่วยผู้ใช้ในการเขียนโปรแกรมติดต่อเซอร์วิสต่างๆ ของ IBM i5/OS

เครื่องหมายการค้า

ชื่อต่อไปนี้เป็นเครื่องหมายการค้าของ International Business Machines Corporation ในประเทศสหรัฐอเมริกา, ประเทศอื่น หรือทั้งสองกรณี:

DB2
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
i5/OS
IBM
IBM (logo)
Integrated Language Environment
iSeries
Lotus Notes
OS/400
REXX
RPG/400
System i
System/36
WebSphere
z/OS

Adobe, สัญลักษณ์ Adobe, PostScript และสัญลักษณ์ PostScript เป็นเครื่องหมายการค้าจดทะเบียน หรือเครื่องหมายการค้าของ Adobe Systems Incorporated ในประเทศสหรัฐอเมริกา และ/หรือประเทศอื่นๆ

Linux เป็นเครื่องหมายการค้าจดทะเบียนของ Linus Torvalds ในประเทศสหรัฐอเมริกา, ประเทศอื่น หรือทั้งสองกรณี

Microsoft, Windows, Windows NT และสัญลักษณ์ Windows เป็นเครื่องหมายการค้าของ Microsoft Corporation ในประเทศสหรัฐอเมริกา, ประเทศอื่น หรือทั้งสองกรณี

Java และเครื่องหมายการค้าที่เกี่ยวกับ Java เป็นเครื่องหมายการค้าของ Sun Microsystems, Inc. ในประเทศสหรัฐอเมริกา, ประเทศอื่น หรือทั้งสองกรณี

UNIX เป็นเครื่องหมายการค้าจดทะเบียนของ The Open ในประเทศสหรัฐอเมริกาและประเทศอื่น

ชื่อบริษัทอื่น, ชื่อผลิตภัณฑ์อื่น หรือชื่อบริการอื่น อาจเป็นเครื่องหมายการค้าหรือเครื่องหมายการบริการของผู้อื่น

ข้อกำหนดและเงื่อนไข

คำอนุญาตในการใช้เอกสารเหล่านี้เป็นไปตามข้อกำหนดและเงื่อนไขต่อไปนี้

การใช้งานเป็นการส่วนตัว: คุณสามารถจัดทำสำเนาของเอกสารเหล่านี้เพื่อใช้เป็นการส่วนตัว มิใช่เพื่อการพาณิชย์โดยมีเงื่อนไขว่าจะต้องคงข้อความประกาศความเป็นเจ้าของไว้โดยครบถ้วน คุณไม่สามารถแจกจ่าย แสดงหรือสร้างงานที่สืบเนื่องจากหนังสือเหล่านี้ หรือมาจากบางส่วนของหนังสือเหล่านี้ โดยไม่ได้รับความยินยอมอย่างชัดแจ้งจาก IBM

การใช้งานในเชิงพาณิชย์: คุณสามารถจัดทำสำเนา แจกจ่าย และแสดงเอกสารนี้ได้เฉพาะภายในองค์กรของคุณ โดยมีเงื่อนไขว่าจะต้องคงข้อความประกาศความเป็นเจ้าของไว้โดยครบถ้วน คุณไม่สามารถสร้างงานที่สืบเนื่องจากหนังสือเหล่านี้ หรือสร้างหนังสือเหล่านี้ใหม่ แจกจ่าย หรือแสดงหนังสือเหล่านี้ทั้งหมดหรือเป็นบางส่วนออกไปยังภายนอกองค์กรของคุณ โดยไม่ได้รับความยินยอมอย่างชัดแจ้งจาก IBM

นอกเหนือจากคำอนุญาตที่ได้แสดงไว้ในที่นี้ IBM ไม่ได้ให้อำนาจดำเนินการ ไลเซนส์หรือสิทธิ์อื่นใด ทั้งโดยชัดแจ้งและโดยนัยกับเอกสารเหล่านี้ รวมถึงสารสนเทศ ข้อมูล ซอฟต์แวร์ หรือทรัพย์สินทางปัญญาอื่นๆ ที่อยู่ภายในที่นี้

IBM ขอสงวนสิทธิ์ในการเพิกถอนคำอนุญาตที่ให้ไว้ในที่นี้ เมื่อใดก็ตามที่พิจารณาแล้วว่า การใช้เอกสารเหล่านี้ก่อให้เกิดความเสียหายต่อผลประโยชน์ของบริษัท หรือเมื่อ IBM ได้พิจารณาแล้วว่า ไม่มีการปฏิบัติตามข้อกำหนดข้างต้นอย่างเหมาะสม

คุณไม่สามารถดาวน์โหลด เอ็กซ์พอร์ตหรือทำการเอ็กซ์พอร์ตข้อมูลนี้ซ้ำได้ ยกเว้นจะได้ปฏิบัติตามกฎหมายและข้อบังคับที่กำหนดไว้ รวมไปถึงกฎหมายและข้อบังคับในการเอ็กซ์พอร์ตของสหรัฐอเมริกา

IBM ไม่ขอรับประกันเกี่ยวกับเนื้อหาของเอกสารเหล่านี้ เอกสารเหล่านี้นำเสนอเนื้อความ "ตามสภาพที่เป็น" โดยไม่มีการรับประกันใดๆ ไม่ว่าจะโดยชัดแจ้งหรือโดยนัย ตลอดจนไม่มีการรับประกันโดยนัยต่อความสามารถในการจำหน่าย การไม่ละเมิด และความเหมาะสมสำหรับวัตถุประสงค์อย่างใดอย่างหนึ่ง

พิมพ์ในสหรัฐอเมริกา