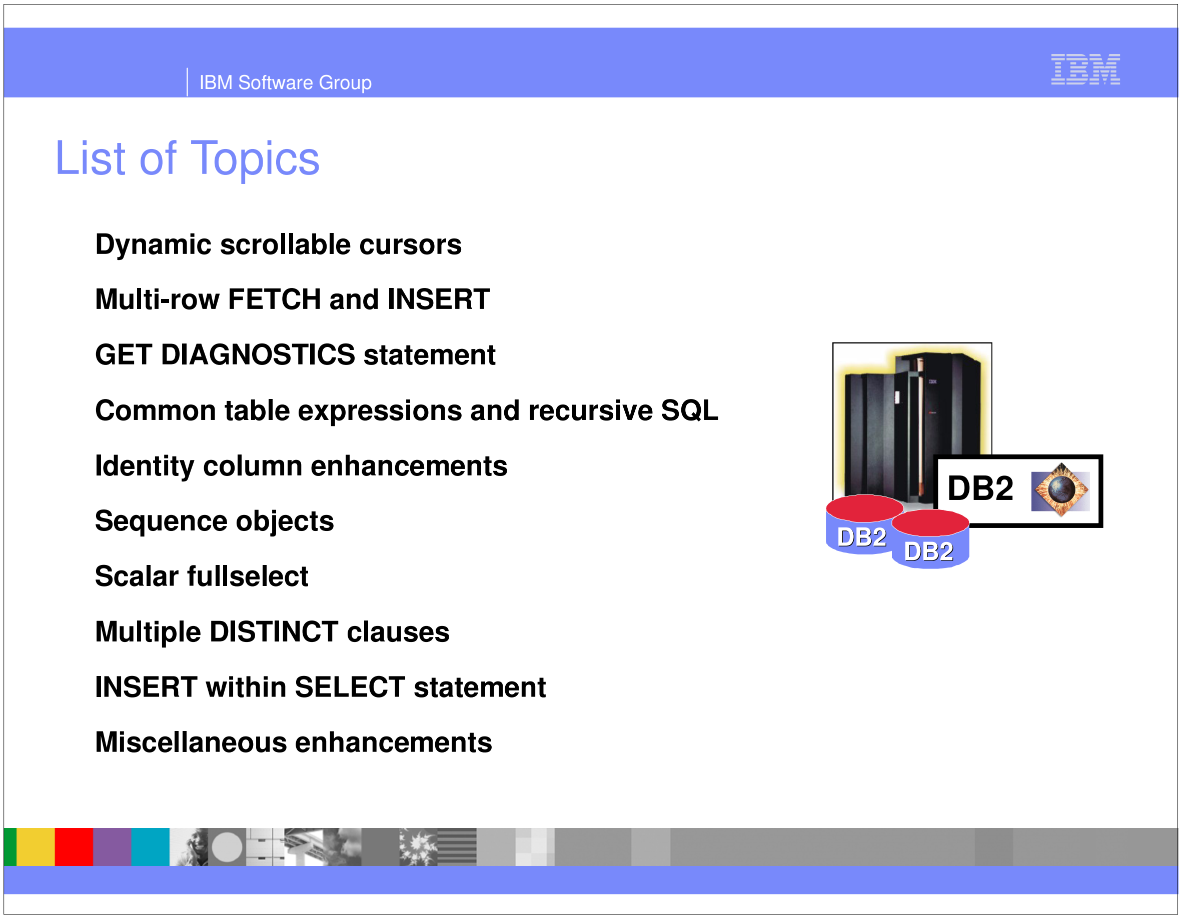
DB2 V8: SQL enhancements

Cécile Benhamou
Technical Sales DB2 z/OS et Tools DB2
cecile_benhamou@fr.ibm.com



List of Topics

Dynamic scrollable cursors

Multi-row FETCH and INSERT

GET DIAGNOSTICS statement

Common table expressions and recursive SQL

Identity column enhancements

Sequence objects

Scalar fullselect

Multiple DISTINCT clauses

INSERT within SELECT statement

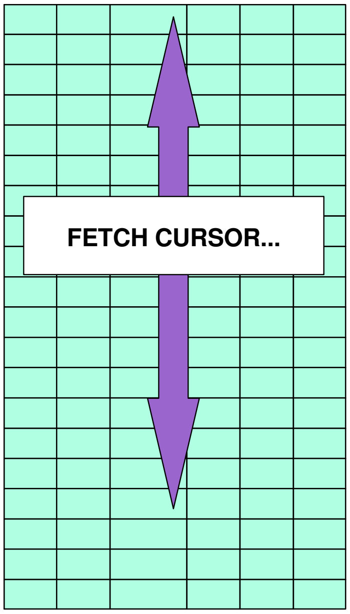
Miscellaneous enhancements

Static Scrollable Cursors - V7 Review

Cursors can be scrolled

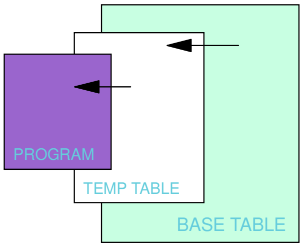
- Backwards
- Forwards
- To an absolute position
- To a position relative to the current cursor
- Before/after position

Result table in TEMP database



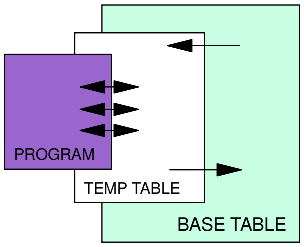
Sensitive and Insensitive Cursors - V7 Review

```
DECLARE C1 INSENSITIVE
  SCROLL..
      ...
FETCH INSENSITIVE...
```



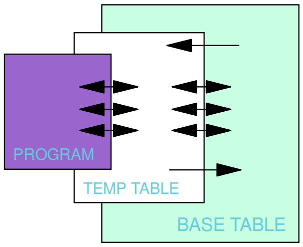
- Read only cursor
- Not aware of updates or deletes in base table

```
DECLARE C1 SENSITIVE
  STATIC SCROLL..
      ...
FETCH INSENSITIVE...
```



- Updateable cursor
- Aware of own updates or deletes within cursor
- Other changes to base table not visible to cursor
- Any inserts not recognized

```
DECLARE C1 SENSITIVE
  STATIC SCROLL..
      ...
FETCH SENSITIVE...
```



- Updateable cursor
- Aware of own updates and deletes within cursor
- Sees all committed updates and deletes
- Any inserts not recognized

New in V8 - Dynamic Scrollable Cursors

Scrollable cursor that provides access to the base table rather than a workfile

-- allows visibility of updates and ***inserts*** done by you or other users

```
DECLARE C1 SENSITIVE DYNAMIC SCROLL
    CURSOR FOR
        SELECT C1, C2
          FROM T1;
```

Declare Cursor - New Attributes

SENSITIVE DYNAMIC

- Specifies that size of result table is not fixed at OPEN cursor time
- Cursor has complete visibility to changes
  - All committed **inserts**, updates, deletes by other application processes
  - All positioned updates and deletes within cursor
  - All **inserts**, updates, deletes by same application processes, but outside cursor
- FETCH executed against base table since **no temporary result table** created

ASENSITIVE

- DB2 determines sensitivity of cursor
- If read-only...
  - Cursor is INSENSITIVE if SELECT statement does not allow it to be SENSITIVE (UNION, UNION ALL, FOR FETCH ONLY, FOR READ ONLY)
  - It behaves as an insensitive cursor
- If not read-only, SENSITIVE DYNAMIC is used for maximum sensitivity
- Mainly for Client applications that do not care whether or not the server supports the sensitivity or scrollability

Implications on FETCH

INSENSITIVE not allowed with FETCH statement (SQLCODE -244) if

- The associated cursor is declared as SENSITIVE DYNAMIC SCROLL
- The cursor is declared ASENSITIVE and DB2 chooses the maximum allowable sensitivity of SENSITIVE DYNAMIC SCROLL

There are no "holes" as there is no temporary result table

- Special case: If FETCH CURRENT or FETCH RELATIVE +0 requested but row on which cursor is positioned was deleted or updated so that it no longer meets selection criteria (SQLCODE +231)

  For example, can occur with ISOLATION(CS) and CURRENTDATA(NO)

Inserts by the application itself are immediately visible -- inserts by others are visible after commit

Order is always maintained

- If current row is updated, the cursor is positioned before the next row of the original location and there is no current row

Dynamic Scrollable Cursors Benefits

- Enhance usability and power of SQL
- Facilitates portability
- Performance improved by sort elimination
- Elimination of workfile (temporary table)
- Immediate visibility of commited updates, deletes, inserts

Cursor Type Comparison

Cursor Type	Result Table	Visibility of Own Changes	Visibility of Others' Changes	Updatability (*)
Non-Scrollable (SQL contains a Join or Sort, etc)	Fixed, workfile	No	No	No
Non-Scrollable	No workfile, base table access	Yes	Yes	Yes
INSENSITIVE SCROLL	Fixed, declared temp table	No	No	No
SENSITIVE STATIC SCROLL	Fixed, declared temp table	Yes (INSERTs not allowed)	Yes (Not INSERTs)	Yes
SENSITIVE DYNAMIC SCROLL	No declared temp table, base table access	Yes	Yes	Yes

Multi-Row FETCH and INSERT

What is it? .....

- Multi-row FETCH:
  - A single FETCH statement can retrieve multiple rows of data from the result table of a query as a rowset
    - ➡ A rowset is a group of rows of data that are grouped together and operated on as a set
    - ➡ Supports dynamic and static SQL (Fetch always static)
- Multi-row INSERT:
  - A single SQL statement can insert one or more rows into a table or view
  - Multi-row INSERT can be implemented as either static or dynamic SQL

Benefits .....

- Enhances usability and power of SQL
- Performance is improved by eliminating multiple trips between application and database engine; for distributed access, reduced network traffic

DECLARE CURSOR and FETCH Examples

Declare C1 as the cursor of a query to retrieve a rowset from table EMP

```
EXEC SQL
  DECLARE C1 CURSOR
  WITH ROWSET POSITIONING
  FOR SELECT * FROM EMP;
```

WITH ROWSET POSITIONING specifies whether multiple rows of data can be accessed as a rowset on a single FETCH statement

Fetch 3 rows starting with row 20 regardless of the current position of the cursor

```
EXEC SQL
     FETCH ROWSET STARTING AT ABSOLUTE 20
         FROM C1 FOR 3 ROWS INTO...
```

Rowsets

A ROWSET is a group of rows from the result table of a query, which are returned by a single FETCH statement (or inserted by a single (multi-row) INSERT statement)

The program controls how many rows are returned in a rowset (it controls the size of the rowset)

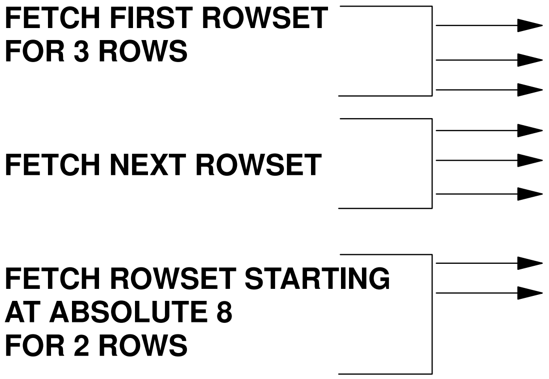
- Can be specified on the FETCH ... FOR n ROWS statement (n is the rowset size and can be up to 32767)

Each group of rows is operated on as a rowset

Ability to intertwine single row and multiple row fetches for a multi-fetch cursor

```
FETCH FIRST ROWSET STARTING AT ABSOLUTE 10
   FROM CURS1
   FOR 6 ROWS INTO :hva1, :hva2;
```

Cursor Positioning: Rowset Positioned Fetches



FETCH FIRST ROWSET FOR 3 ROWS

FETCH NEXT ROWSET

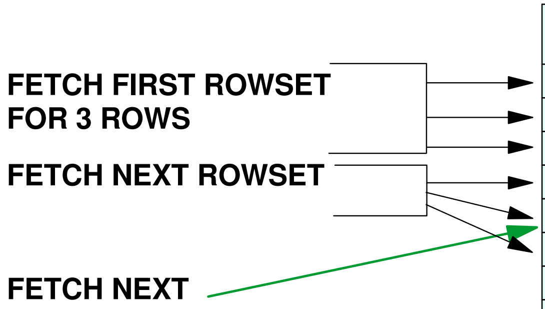
FETCH ROWSET STARTING AT ABSOLUTE 8 FOR 2 ROWS

Result table

CUST_NO	CUST_TYP	CUST_NAME
1	P	Ian
2	P	Mark
3	P	John
4	P	Karen
5	P	Sarah
6	M	Florence
7	M	Dylan
8	M	Bert
9	M	Jo
10	R	Karen
11	R	Gary
12	R	Bill
13	R	Geoff
14	R	Julia
15	R	Sally

Note : Cursor is positioned on ALL rows in current rowset

Mixing Row and Rowset Positioning



FETCH FIRST ROWSET FOR 3 ROWS

FETCH NEXT ROWSET

FETCH NEXT

Result table

CUST_NO	CUST_TYP	CUST_NAME
1	P	Ian
2	P	Mark
3	P	John
4	P	Karen
5	P	Sarah
6	M	Florence
7	M	Dylan
8	M	Bert
9	M	Jo
10	R	Karen
11	R	Gary
12	R	Bill
13	R	Geoff
14	R	Julia
15	R	Sally

Note : FETCH NEXT is relative to the FIRST row in the current rowset

Multi-Row INSERT

New third form of insert

- INSERT via VALUES is used for inserting a single row into the table or view using values provided or referenced
- INSERT via SELECT is used for inserting one or more rows into the table or view using values from other tables or views
- INSERT with FOR "n" ROWS is used to insert multiple rows into the table or view using values provided in a host variable array

FOR "n" ROWS

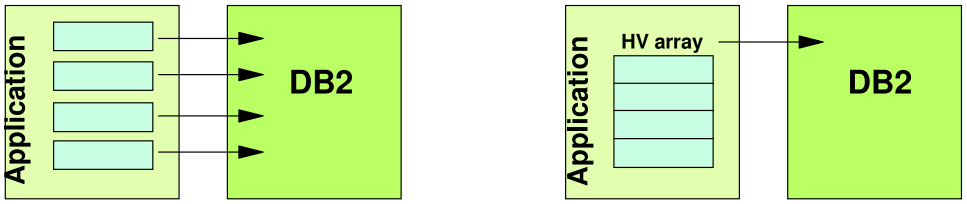
- For static, specify FOR "n" ROWS on the INSERT statement (for dynamic INSERT, specify FOR "n" ROWS on the EXECUTE statement)
- Input provided with host variable array -- each array represents cells for multiple rows of a single column

VALUES clause allows specification of multiple rows of data

- Host variable arrays used to provide values for a column on INSERT

Using Multi-Row INSERT

Single row versus multi-row INSERT



Multi-row INSERT statement - special case

INSERT INTO TAB1 VALUES (’my string’ ,   :hva   , CURRENT DATE)
FOR 4 ROWS

- Program contains

my string	:hva	03-08-2004

- DB2 INSERTS

my string	:hva	03-08-2004
my string		03-08-2004
my string		03-08-2004
my string		03-08-2004

ATOMIC / NOT ATOMIC

ATOMIC (default)

- If the insert for any row fails, all changes made to database by that INSERT statement are undone

NOT ATOMIC CONTINUE ON SQLEXCEPTION

- Inserts are processed independently
- If errors occur during execution of INSERT, processing continues
- Diagnostics are available for each failed row through GET DIAGNOSTICS
- SQLCODE indicates if:
  - All failed
  - All were successful
  - At least one failed

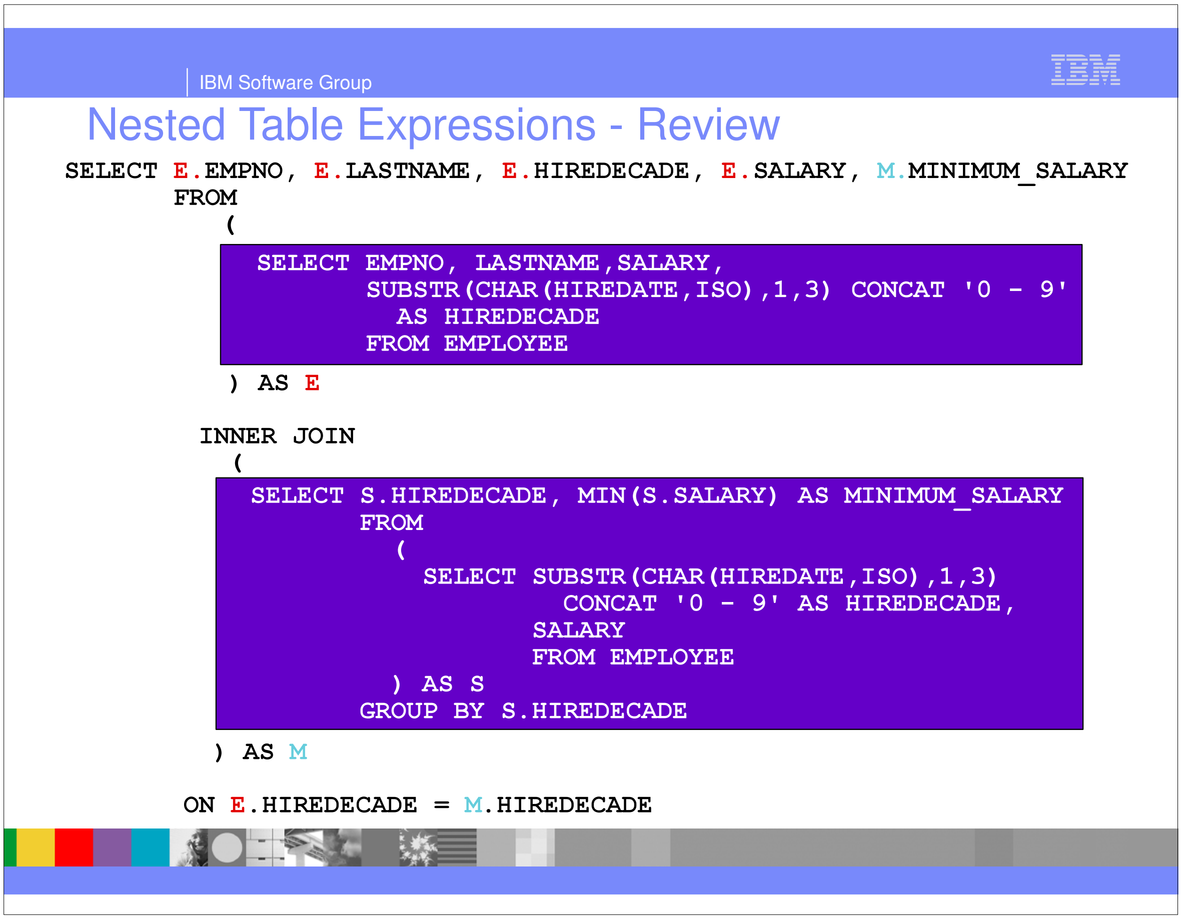
GET DIAGNOSTICS

- Enables more diagnostic information to be returned than can be contained into SQLCA
- Returns SQL error information
  - For overall statement
  - For each condition (when multiple errors occur)
- Supports SQL error message tokens greater than 70 bytes (SQLCA limitation)

To handle multiple SQL errors during a NOT ATOMIC multi-row insert

```
INSERT INTO T1 FOR 5 ROWS VALUES(:ARRAY);

GET DIAGNOSTICS :ERR_COUNT = NUMBER;
   DO II = 1 TO ERR_COUNT;
      GET DIAGNOSTICS CONDITION :II
          :RC = RETURNED_SQLSTATE;
   END;
```

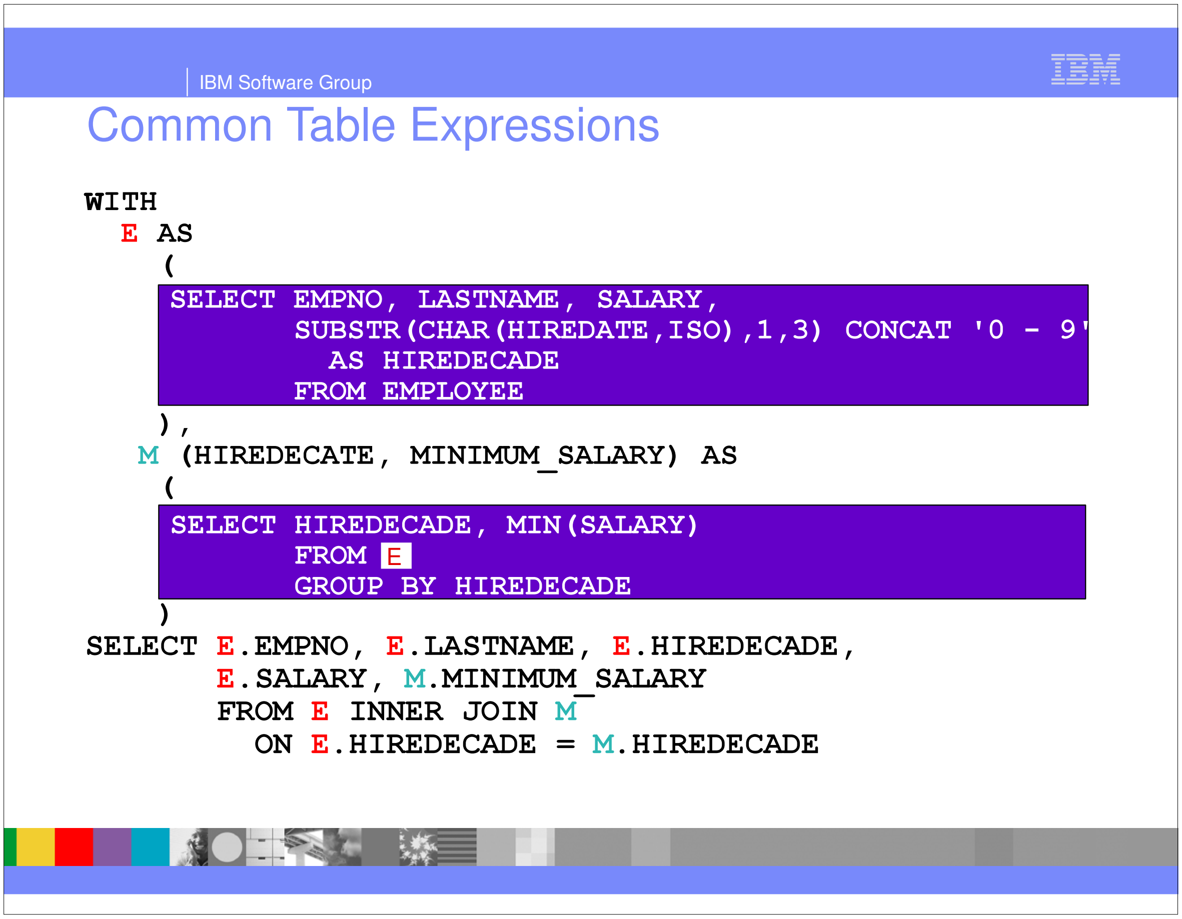


Nested Table Expressions - Review

```
SELECT E.EMPNO, E.LASTNAME, E.HIREDECADE, E.SALARY, M.MINIMUM_SALARY
       FROM
          (
            SELECT EMPNO, LASTNAME,SALARY,
                   SUBSTR(CHAR(HIREDATE,ISO),1,3) CONCAT '0 - 9'
                     AS HIREDECADE
                   FROM EMPLOYEE
          ) AS E

        INNER JOIN
          (
           SELECT S.HIREDECADE, MIN(S.SALARY) AS MINIMUM_SALARY
                  FROM
                    (
                      SELECT SUBSTR(CHAR(HIREDATE,ISO),1,3)
                               CONCAT '0 - 9' AS HIREDECADE,
                             SALARY
                             FROM EMPLOYEE
                    ) AS S
                  GROUP BY S.HIREDECADE
         ) AS M

        ON E.HIREDECADE = M.HIREDECADE
```



Common Table Expressions

```
WITH
  E AS
    (
    SELECT EMPNO, LASTNAME, SALARY,
           SUBSTR(CHAR(HIREDATE,ISO),1,3) CONCAT '0 - 9'
             AS HIREDECADE
           FROM EMPLOYEE
    ),
   M (HIREDECATE, MINIMUM_SALARY) AS
    (
    SELECT HIREDECADE, MIN(SALARY)
           FROM E
           GROUP BY HIREDECADE
    )
SELECT E.EMPNO, E.LASTNAME, E.HIREDECADE,
       E.SALARY, M.MINIMUM_SALARY
       FROM E INNER JOIN M
         ON E.HIREDECADE = M.HIREDECADE
```

Recursive SQL

```
WITH
  RPL (PART, SUBPART, QUANTITY) AS
    (
```

Initialization Select

```
      SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
             FROM PARTLIST ROOT
             WHERE ROOT.PART = '01'
```

UNION ALL

Iterative Select

```
      SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
             FROM RPL PARENT, PARTLIST CHILD
             WHERE PARENT.SUBPART = CHILD.PART
     )
```

Main Select

```
  SELECT PART, SUBPART, SUM(QUANTITY) AS QUANTITY
         FROM RPL
         GROUP BY PART, SUBPART
```

Recursive SQL- Initialization SELECT

```
SELECT ROOT.PART, ROOT.SUBPART,
ROOT.QUANTITY
        FROM PARTLIST ROOT
        WHERE ROOT.PART = '01'
```

PART	SUBPART	QUANTITY
00	01	5
00	05	3
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

PARTLIST Table

PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3

RPL

Recursive SQL - First Iteration

```
SELECT CHILD.PART, CHILD.SUBPART,
CHILD.QUANTITY
       FROM RPL PARENT, PARTLIST CHILD
       WHERE PARENT.SUBPART = CHILD.PART
```

PART	SUBPART	QUANTITY
00	01	5
00	05	3
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

PARTLIST Table

PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
06	12	10
06	13	10

RPL

Recursive SQL - Second Iteration

```
SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
       FROM RPL PARENT, PARTLIST CHILD
       WHERE PARENT.SUBPART = CHILD.PART
```

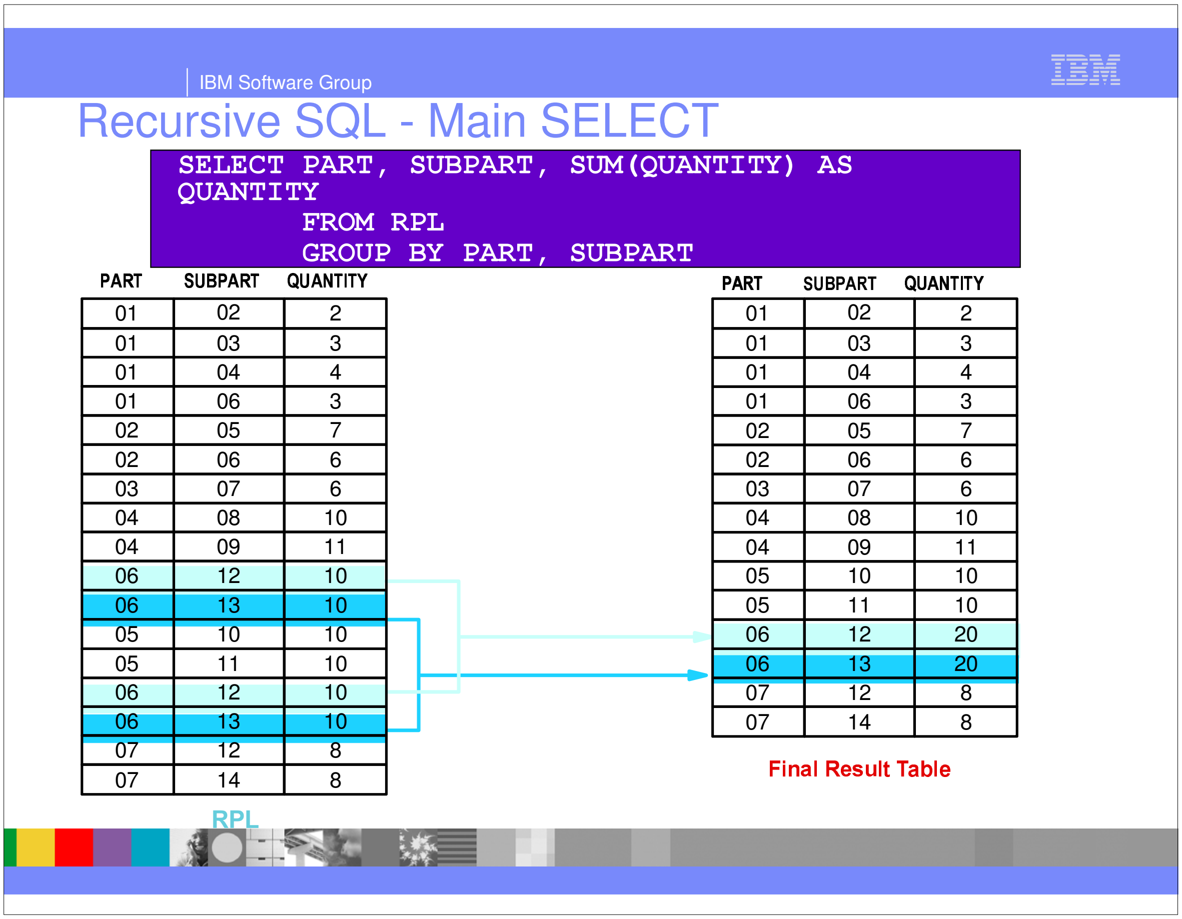
PART	SUBPART	QUANTITY
00	01	5
00	05	3
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

PARTLIST Table

No correspondence in PARTLIST table

PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
06	12	10
06	13	10
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

RPL



Recursive SQL - Main SELECT

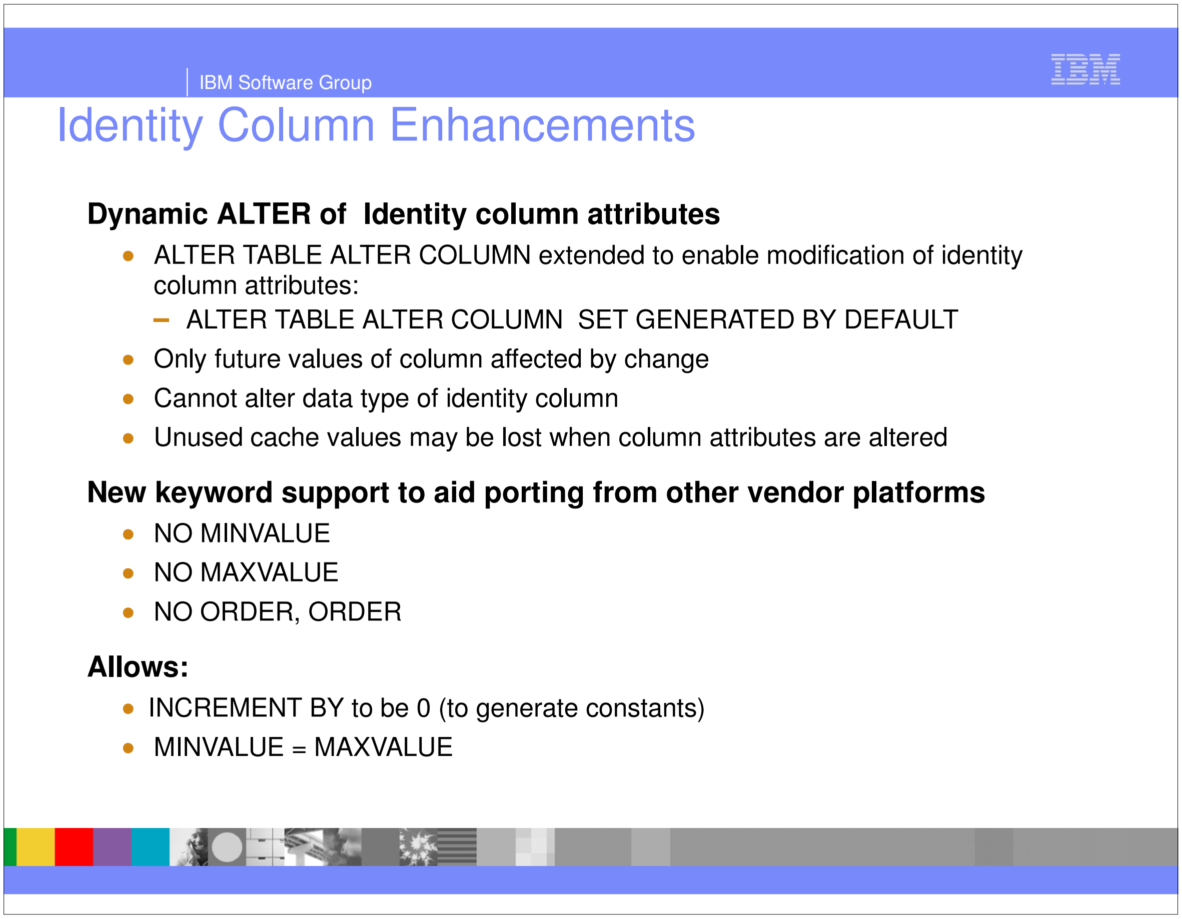
```
SELECT PART, SUBPART, SUM(QUANTITY) AS
QUANTITY
         FROM RPL
         GROUP BY PART, SUBPART
```

PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
06	12	10
06	13	10
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

RPL

PART	SUBPART	QUANTITY
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	20
06	13	20
07	12	8
07	14	8

Final Result Table



Identity Column Enhancements

Dynamic ALTER of Identity column attributes

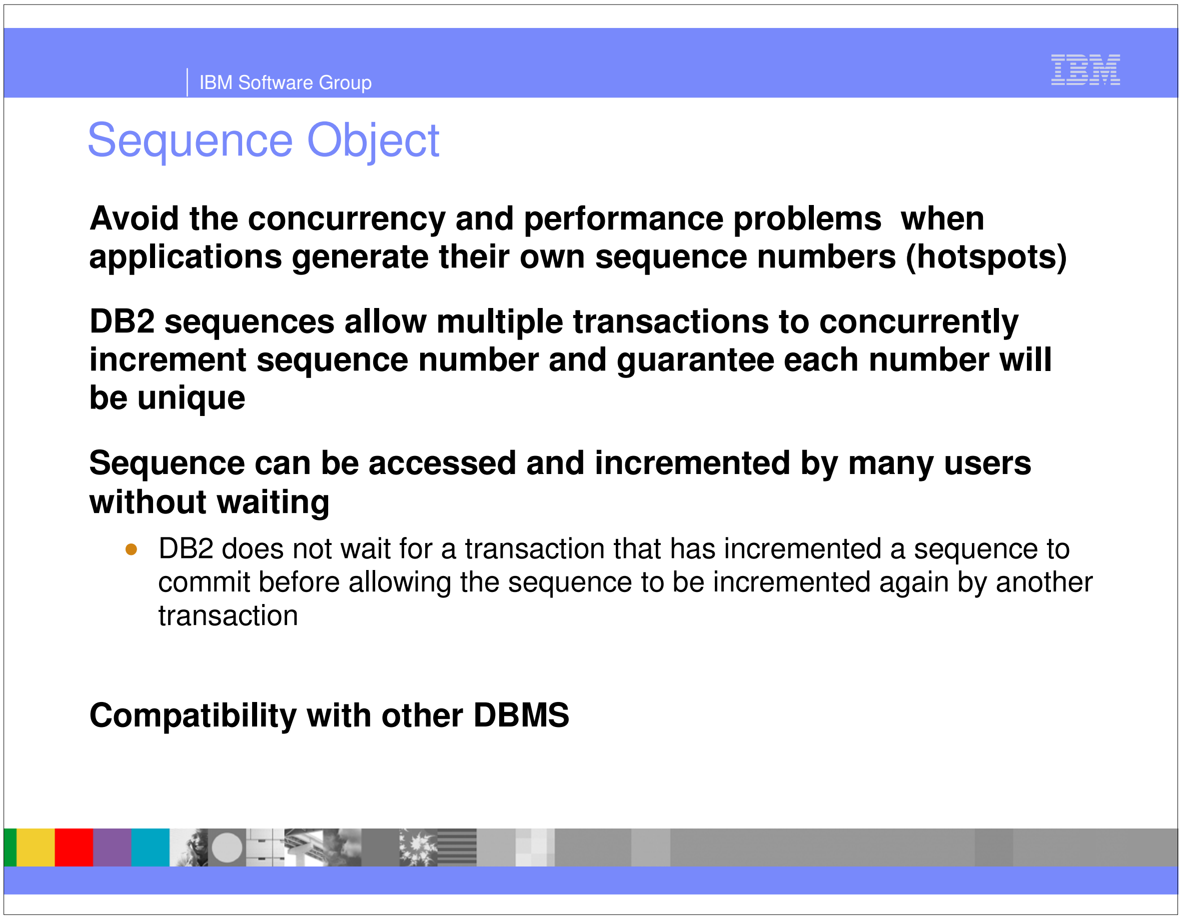
- ALTER TABLE ALTER COLUMN extended to enable modification of identity column attributes:
  - ALTER TABLE ALTER COLUMN SET GENERATED BY DEFAULT
- Only future values of column affected by change
- Cannot alter data type of identity column
- Unused cache values may be lost when column attributes are altered

New keyword support to aid porting from other vendor platforms

- NO MINVALUE
- NO MAXVALUE
- NO ORDER, ORDER

Allows:

- INCREMENT BY to be 0 (to generate constants)
- MINVALUE = MAXVALUE



Sequence Object

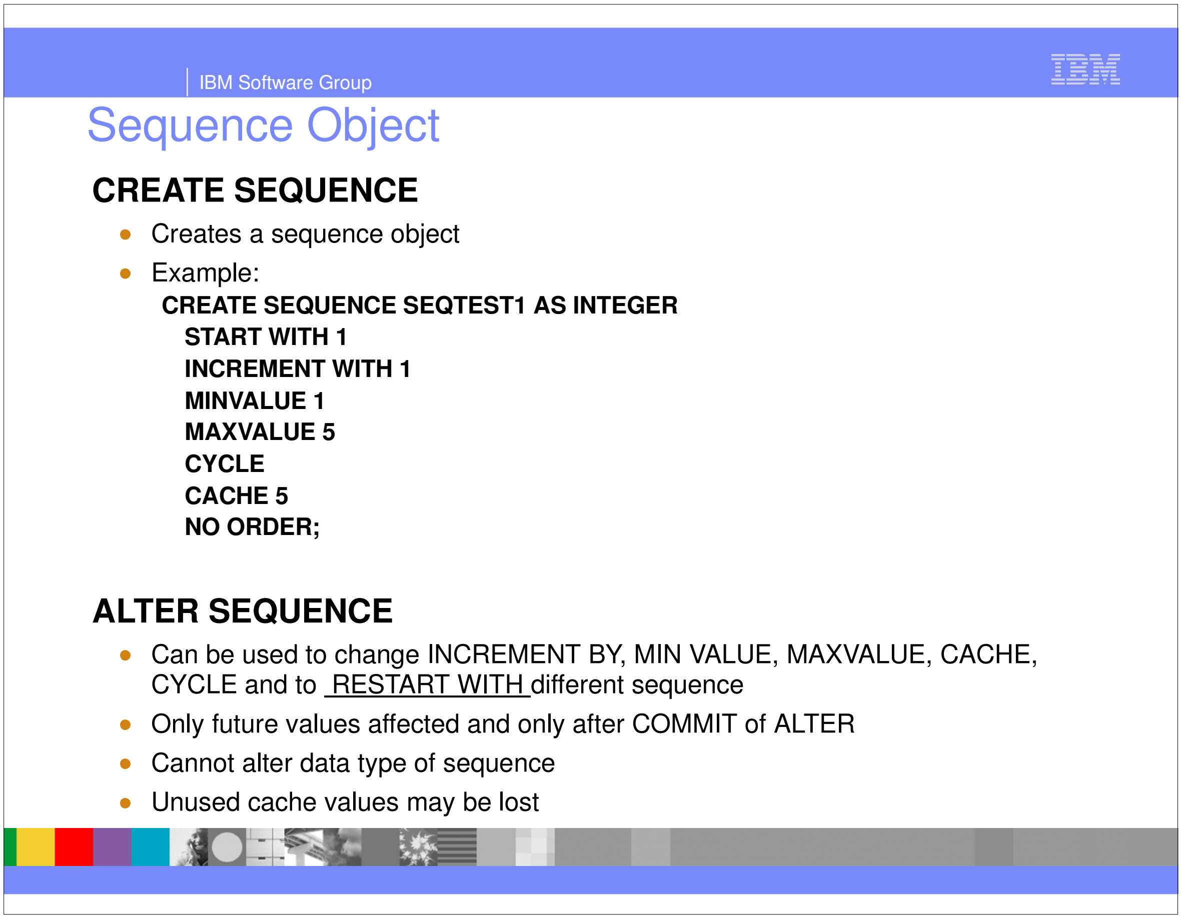
Avoid the concurrency and performance problems when applications generate their own sequence numbers (hotspots)

DB2 sequences allow multiple transactions to concurrently increment sequence number and guarantee each number will be unique

Sequence can be accessed and incremented by many users without waiting

- DB2 does not wait for a transaction that has incremented a sequence to commit before allowing the sequence to be incremented again by another transaction

Compatibility with other DBMS



Sequence Object

CREATE SEQUENCE

- Creates a sequence object
- Example:

```
CREATE SEQUENCE SEQTEST1 AS INTEGER
   START WITH 1
   INCREMENT WITH 1
   MINVALUE 1
   MAXVALUE 5
   CYCLE
   CACHE 5
   NO ORDER;
```

ALTER SEQUENCE

- Can be used to change INCREMENT BY, MIN VALUE, MAXVALUE, CACHE, CYCLE and to RESTART WITH different sequence
- Only future values affected and only after COMMIT of ALTER
- Cannot alter data type of sequence
- Unused cache values may be lost

Next and Previous Values

Applications can refer to the named sequence object to get its current or next value

- NEXT VALUE FOR < sequence- name >
- PREVIOUS VALUE FOR < sequence-name >
  - Returns most recently generated value for sequence for previous statement within current session
  - NEXT VALUE must have been invoked within current session

Examples:

1) *Assume sequence created with START WITH 1, INCREMENT BY 1*

```
SELECT NEXT VALUE FOR MYSEQ FROM SYSIBM.SYSDUMMY1;  Generates Value of 1
SELECT NEXT VALUE FOR MYSEQ FROM SYSIBM.SYSDUMMY1;  Generates Value of 2
COMMIT;
SELECT PREVIOUS VALUE FOR MYSEQ FROM SYSIBM.SYSDUMMY1;
                    Returns most recently generated value (2)
```

2) *Viewing sequence while inserting*

```
SELECT * FROM FINAL TABLE
  (     INSERT INTO TESTTAB (KEYVALUE, TESTSEQ)
        VALUES ( NEXT VALUE FOR SEQTEST1, NEXT VALUE FOR SEQTEST2 )    )     ;
```

Comparing Identity Columns and Sequences

Sequences	Identity columns
Stand-alone object	Tied to a table
Can use one sequence for many tables or many sequences in one table	One to one relationship between identity and tables
Retrieved via NEXT VALUE FOR / PREVIOUS VALUE FOR expressions	Retrieved via IDENTITY_VAL_LOCAL function - within agents scope only
Can be altered via ALTER SEQUENCE	Can be altered via ALTER TABLE (ALTER COLUMN) Prior to V8 could not be altered

Scalar Fullselect

What is it? .....

- A scalar fullselect is a **fullselect**, enclosed in parentheses, that returns a single value
- Allows scalar fullselect where expressions were previously supported
- Example:

```
SELECT PRODUCT, PRICE
           FROM PRODUCTS
                       WHERE PRICE <= 0.7 * (SELECT AVG(PRICE)
                                                FROM   PRODUCTS);
```

Benefits .....

- Enhances usability and power of SQL
- Facilitates portability
- Conforms with SQL standards

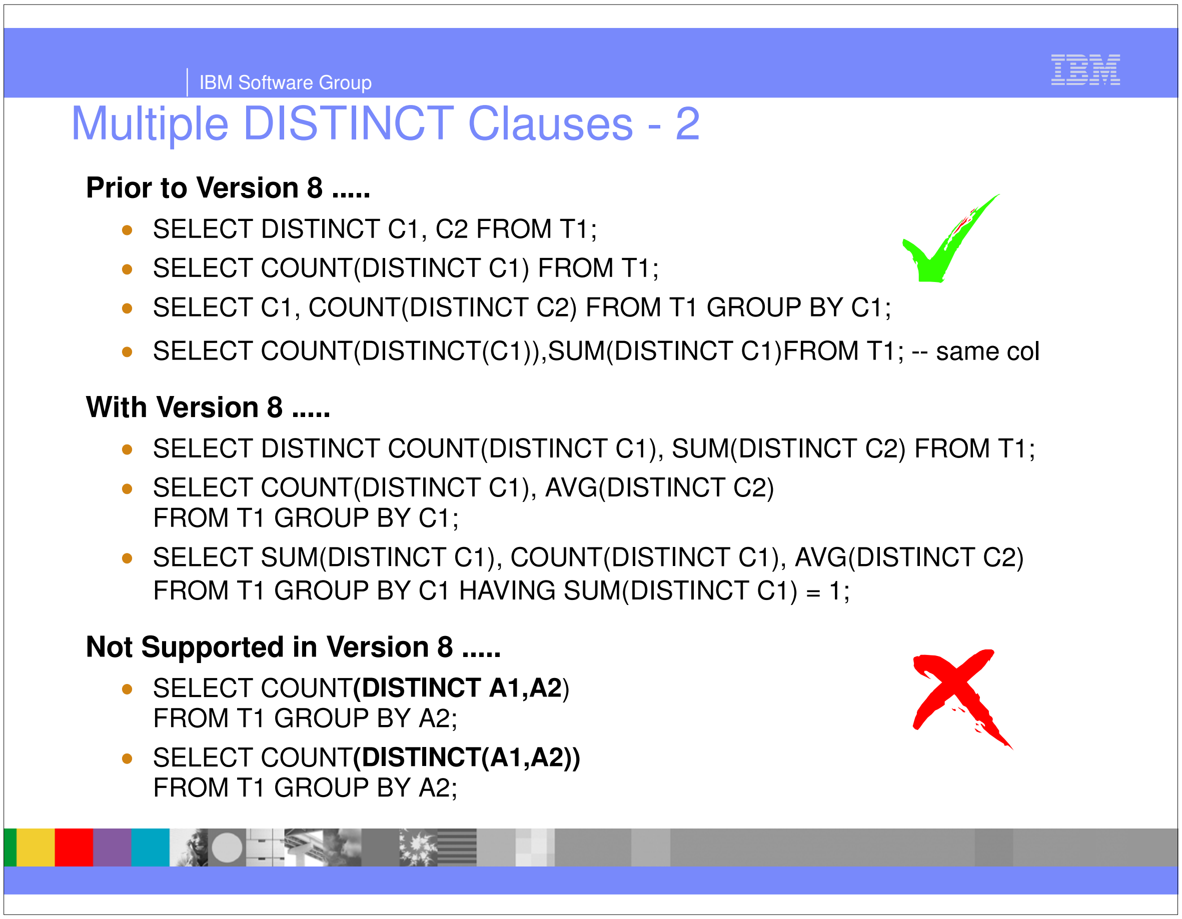
Multiple DISTINCT Clauses

What is it? .....

- Allows more than one DISTINCT keyword on the SELECT or HAVING clause for a query

Benefits .....

- Enhances usability and power of SQL
- DB2 Family compatibility
- Previously you would get an SQLCODE -127



Multiple DISTINCT Clauses - 2

Prior to Version 8 ..... ✓

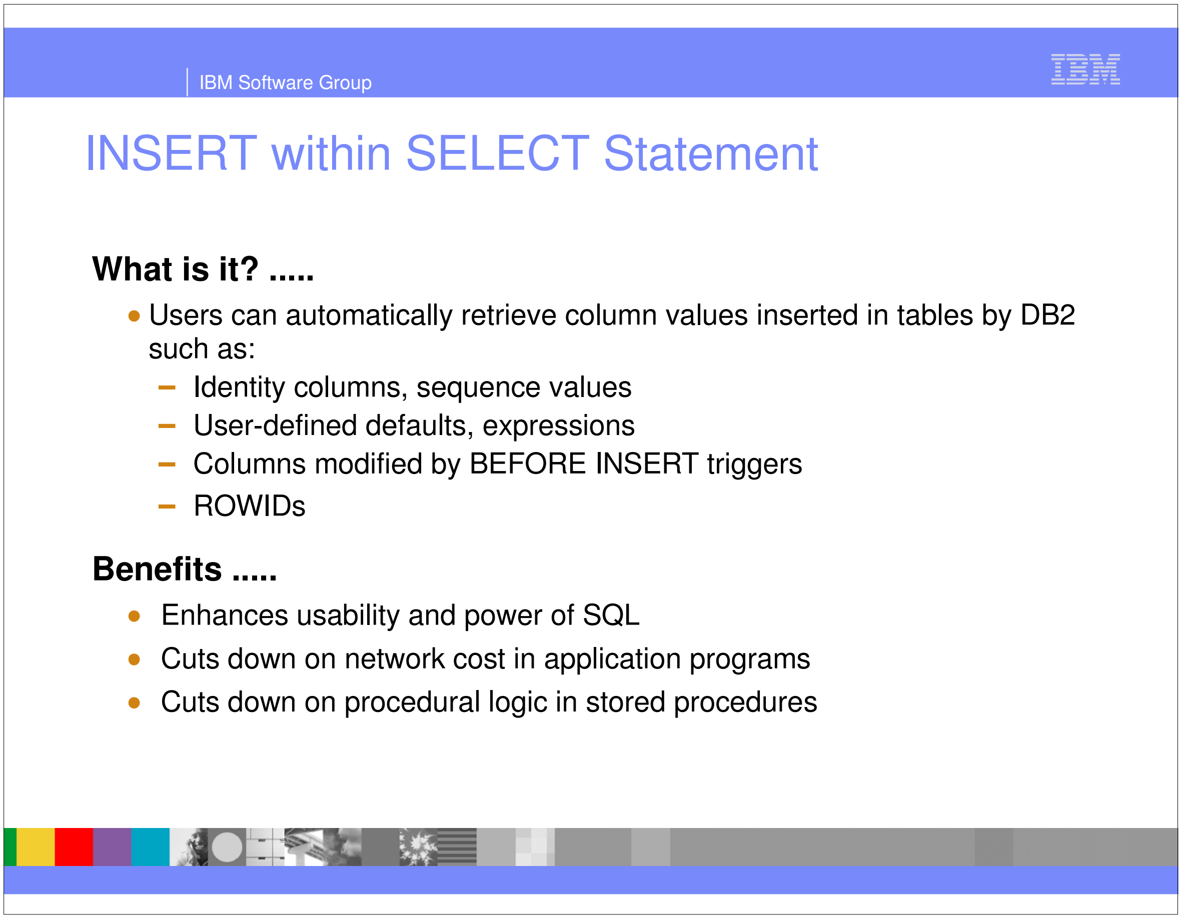
- SELECT DISTINCT C1, C2 FROM T1;
- SELECT COUNT(DISTINCT C1) FROM T1;
- SELECT C1, COUNT(DISTINCT C2) FROM T1 GROUP BY C1;
- SELECT COUNT(DISTINCT(C1)),SUM(DISTINCT C1)FROM T1; -- same col

With Version 8 .....

- SELECT DISTINCT COUNT(DISTINCT C1), SUM(DISTINCT C2) FROM T1;
- SELECT COUNT(DISTINCT C1), AVG(DISTINCT C2)
  FROM T1 GROUP BY C1;
- SELECT SUM(DISTINCT C1), COUNT(DISTINCT C1), AVG(DISTINCT C2)
  FROM T1 GROUP BY C1 HAVING SUM(DISTINCT C1) = 1;

Not Supported in Version 8 ..... ✗

- SELECT COUNT(**DISTINCT A1,A2**)
  FROM T1 GROUP BY A2;
- SELECT COUNT**(DISTINCT(A1,A2))**
  FROM T1 GROUP BY A2;



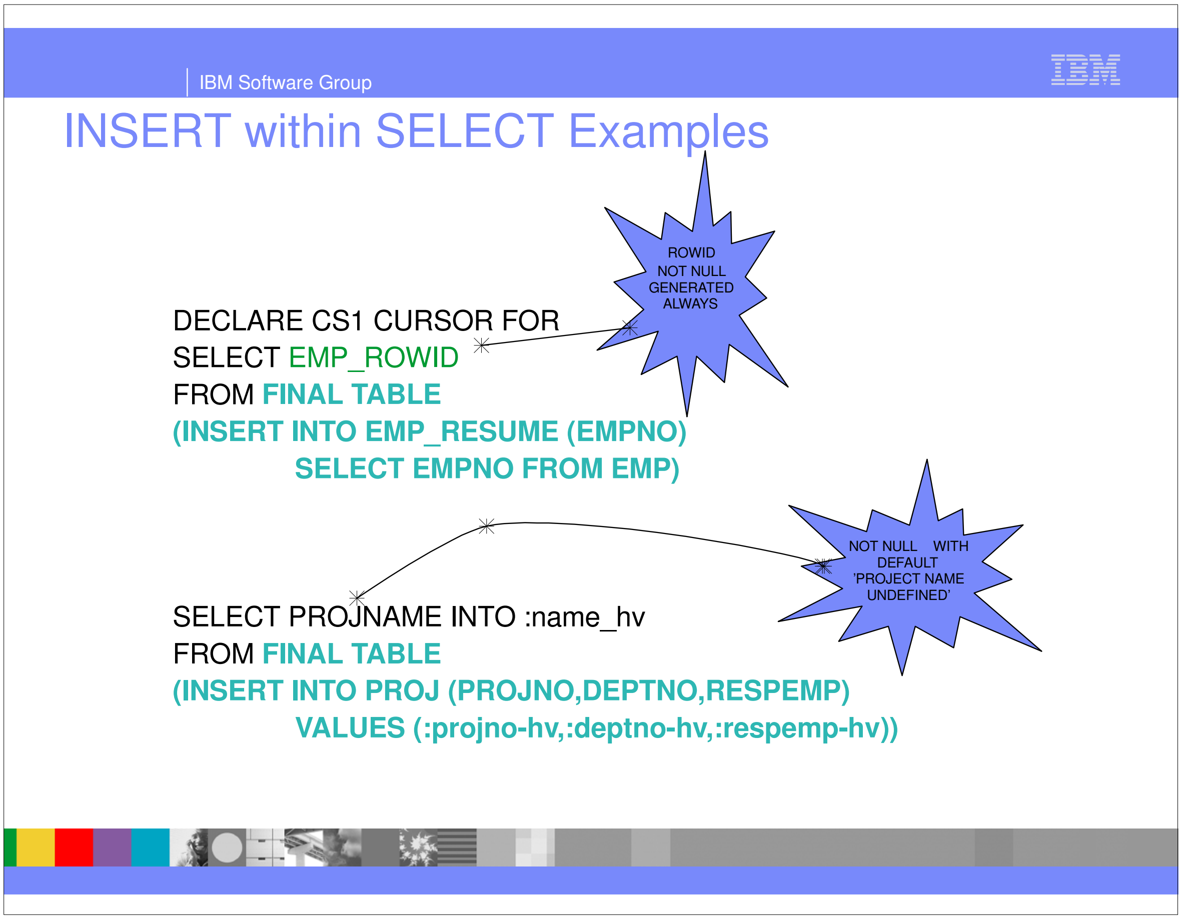
INSERT within SELECT Statement

What is it? .....

- Users can automatically retrieve column values inserted in tables by DB2 such as:
  - Identity columns, sequence values
  - User-defined defaults, expressions
  - Columns modified by BEFORE INSERT triggers
  - ROWIDs

Benefits .....

- Enhances usability and power of SQL
- Cuts down on network cost in application programs
- Cuts down on procedural logic in stored procedures



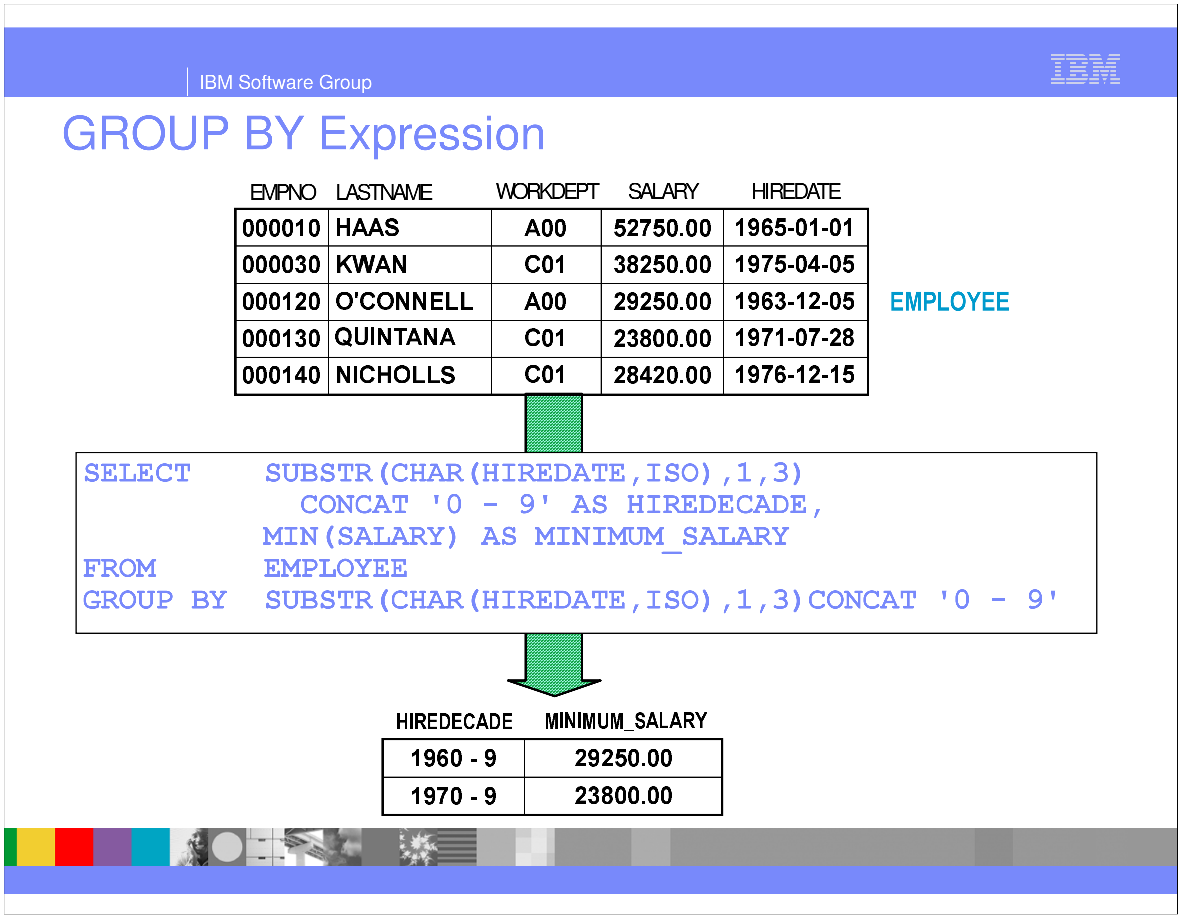
INSERT within SELECT Examples

```
DECLARE CS1 CURSOR FOR
SELECT EMP_ROWID
FROM FINAL TABLE
(INSERT INTO EMP_RESUME (EMPNO)
          SELECT EMPNO FROM EMP)
```

ROWID NOT NULL GENERATED ALWAYS

```
SELECT PROJNAME INTO :name_hv
FROM FINAL TABLE
(INSERT INTO PROJ (PROJNO,DEPTNO,RESPEMP)
          VALUES (:projno-hv,:deptno-hv,:respemp-hv))
```

NOT NULL WITH DEFAULT 'PROJECT NAME UNDEFINED'



GROUP BY Expression

EMPLOYEE

EMPNO	LASTNAME	WORKDEPT	SALARY	HIREDATE
000010	HAAS	A00	52750.00	1965-01-01
000030	KWAN	C01	38250.00	1975-04-05
000120	O'CONNELL	A00	29250.00	1963-12-05
000130	QUINTANA	C01	23800.00	1971-07-28
000140	NICHOLLS	C01	28420.00	1976-12-15

```
SELECT     SUBSTR(CHAR(HIREDATE,ISO),1,3)
             CONCAT '0 - 9' AS HIREDECADE,
           MIN(SALARY) AS MINIMUM_SALARY
FROM       EMPLOYEE
GROUP BY   SUBSTR(CHAR(HIREDATE,ISO),1,3)CONCAT '0 - 9'
```

HIREDECADE	MINIMUM_SALARY
1960 - 9	29250.00
1970 - 9	23800.00

Qualified Column Names in INSERT and UPDATE

Column names can be qualified with a table name, or a schema followed by a table name in INSERT

Column names in the SET clause of an UPDATE statement can be qualified

These enhancements provide for more DB2 family compatibility

For example:

```
UPDATE T1 SET T1.C1 = C1 + 10 WHERE C1 = 1

UPDATE T1 T SET T.C1 = C1 + 10 WHERE C1 = 2
```

IS NOT DISTINCT FROM

SQL uses three-valued logic where any given comparison can return: TRUE, FALSE, or NULL

Applications can use IS NOT DISTINCT FROM to obtain a TRUE result instead of NULL when comparing NULL values

```
SELECT C1 FROM T1 WHERE
C1 IS NOT DISTINCT FROM :hv;
```

C1 value	:hv value	RESULT
NULL	'ABC'	FALSE
NULL	NULL	**TRUE**
'ABC'	'ABC'	TRUE
'ABC'	NULL	FALSE
'ABC'	'DEF'	FALSE

Returned by query above

REOPT(ONCE)

Bind option that controls when the Optimizer builds the access path information for dynamic SQL applications.

- **By default, access path is calculated at PREPARE.**
- **REOPT(VARS)**
  - **defers access path selection until OPEN**
  - **values of host variables on OPEN are used to calculate access path**
  - **resulting access path is cached in the global prepare cache**
    - **done at every execution**
- **REOPT (ONCE)**
  - **same as REOPT(VARS) BUT**
    - **access path is only calculated the first time is it executed**

Transparent ROWID

Eliminates the need to explicitly declare a ROWID column in tables that include LOBs

DB2 generates a "hidden" ROWID column, which is not visible on SELECT *

Simplifies porting of LOB applications from other platforms

Acknowledgments

This presentation is based on the following ’Redbook’:

DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More (SG24-6079)

Redbooks

Other information

IBM DB2 Universal Database SQL Reference for Cross Platform Development

z/OS OS/390 OS/400 AIX HP-UX Solaris Linux Windows

A new SQL Reference book for the DB2 UDB family, not just one platform.

ftp://ftp.software.ibm.com/ps/products/db2/info/xplatsql/pdf/en_US/cpsqlrv2.pdf