DB2 9

Technical Education Series

“SQL Enhancements”

Important Disclaimer

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.

IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANTY OR REPRESENTATION FROM IBM (OR ITS AFFILIATES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS); OR
- ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF IBM SOFTWARE.

List of topics

- INTERSECT, EXCEPT
- INSTEAD OF triggers
- MERGE
- SELECT FROM MERGE / UPDATE / DELETE
- TRUNCATE
- ORDER BY and FETCH FIRST in subselect
- New data types
- New SQL functions

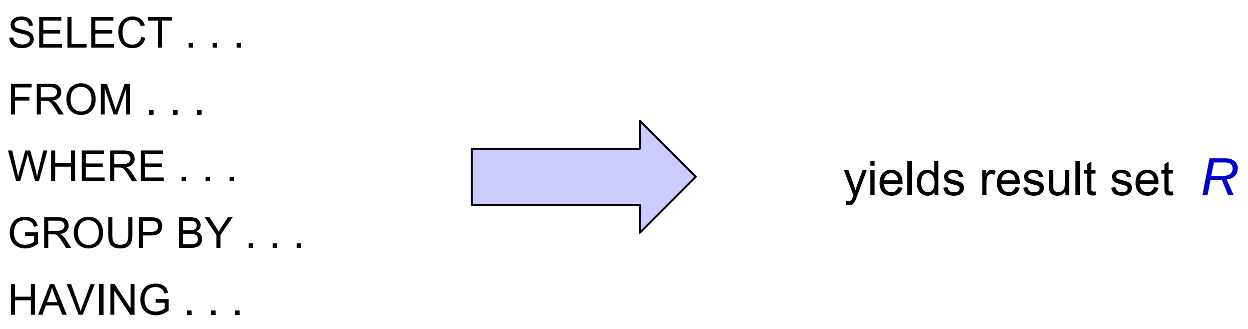
SWG BetaWorks

INTERSECT and EXCEPT

DB2 9 for z/OS Technical
Education Series

Review -- *subselect*

- A subselect specifies a result table derived from the result of its first FROM clause . . .
- Components of *subselect:*



SELECT . . .

FROM . . .

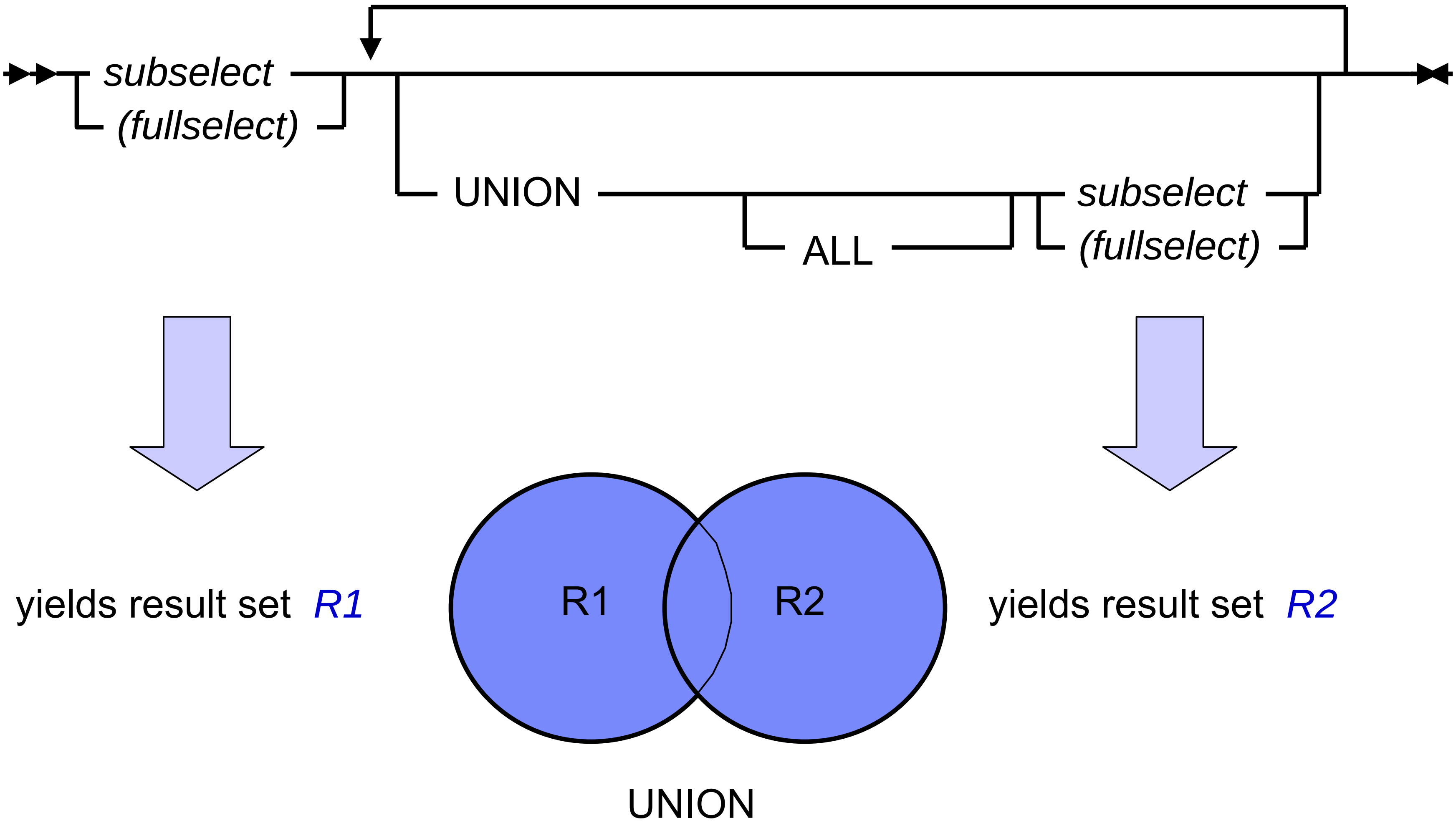
WHERE . . .

GROUP BY . . .

HAVING . . .

yields result set *R*

V8 *fullselect*



subselect

(fullselect)

UNION

ALL

subselect

(fullselect)

yields result set *R1*

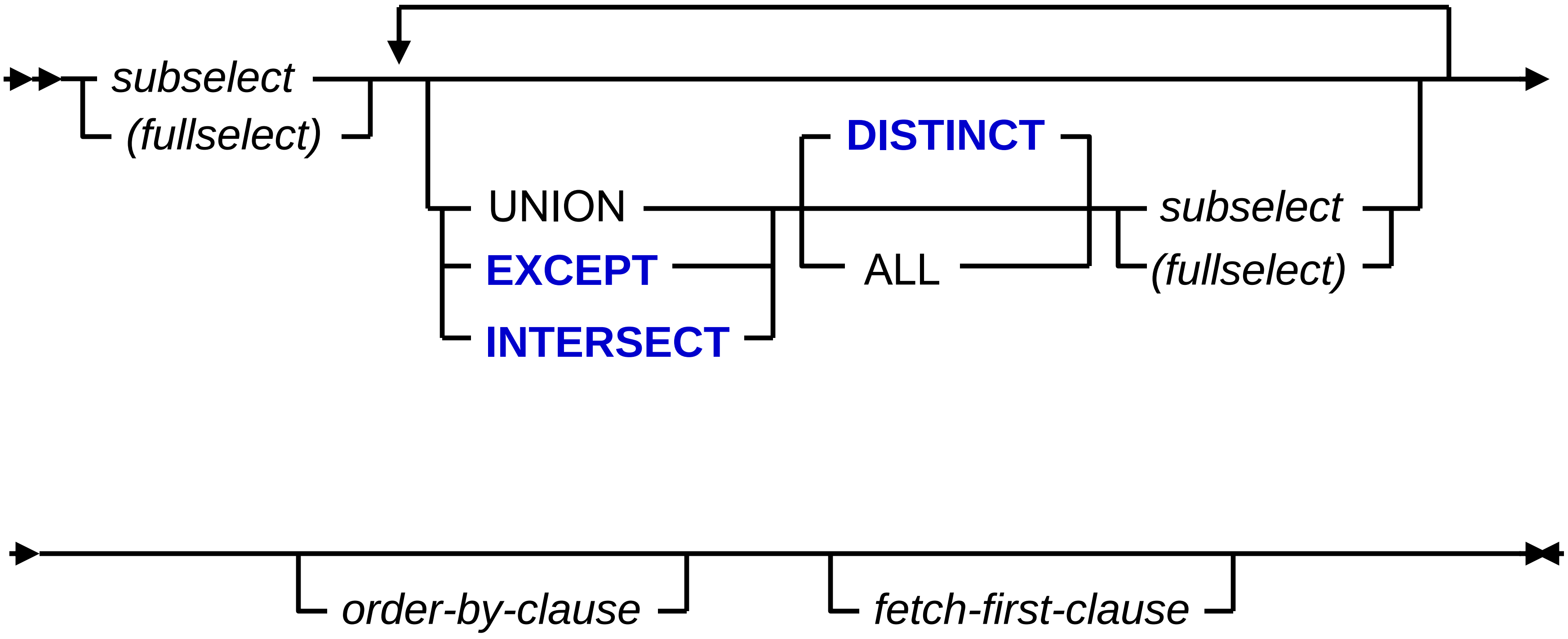
R1

R2

yields result set *R2*

UNION

DB2 9: *fullselect* -- new opportunities to combine sets



subselect

(fullselect)

UNION

EXCEPT

INTERSECT

DISTINCT

ALL

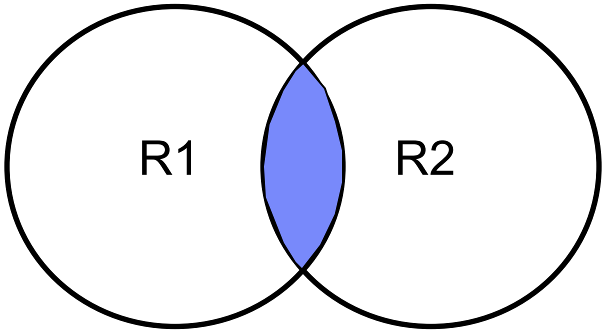
subselect

(fullselect)

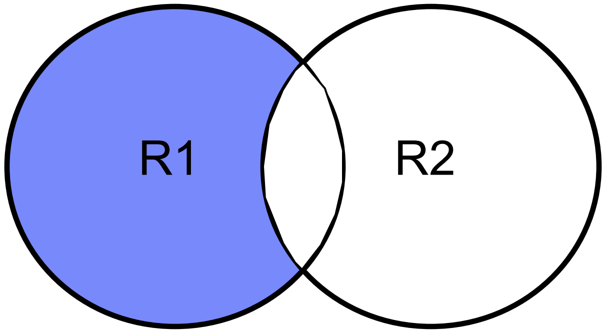
order-by-clause

fetch-first-clause

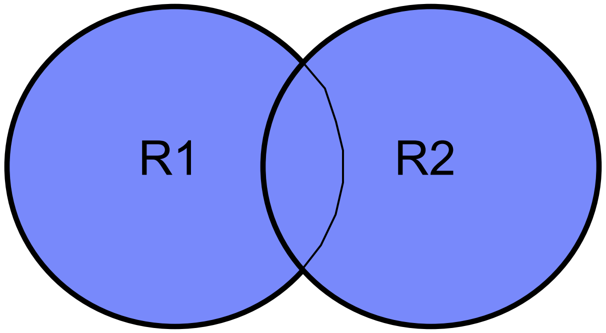
Options for joining sets



INTERSECT



EXCEPT



UNION

Columns participating in INTERSECT and EXCEPT

- **R1** and **R2** must have the same number of columns
  - data type for the *n*-th column of **R1** must be compatible with the *n*-th column of **R2**
  - data type must not be CLOB, BLOB, DBCLOB, XML, or distinct type based on these types
- If the *n*-th column of **R1** and the *n*-th column of **R2** have the same name, then the *n*-th column of the result table has the same name; else unnamed
- Qualified column names cannot be used in the ORDER BY clause when the set operators are specified

Example of EXCEPT

- *Example*: Assume that tables T1 and T2 exist, each containing the same number of columns named C1, C2, and so on. This example of EXCEPT operator produces all rows that are in T1 but not in T2, with redundant duplicate rows removed.

  ```
  (SELECT * FROM T1) EXCEPT DISTINCT (SELECT * FROM T2)
  ```

  If no NULL values are involved, this example returns the same result as:

  ```
  SELECT DISTINCT * FROM T1
  WHERE NOT EXISTS
          (SELECT * FROM T2 WHERE T1.C1 = T2.C1 AND T1.C2 AND ...)
  ```

  where the subquery contains an equal predicate for each pair of columns that exists in both tables.

Example of INTERSECT

- *Example*: Assume that the tables T1 and T2 from the prevoius example exist. This example of INTERSECT operator produces all rows that are in both tables T1 and T2, with redundant duplicate rows removed.

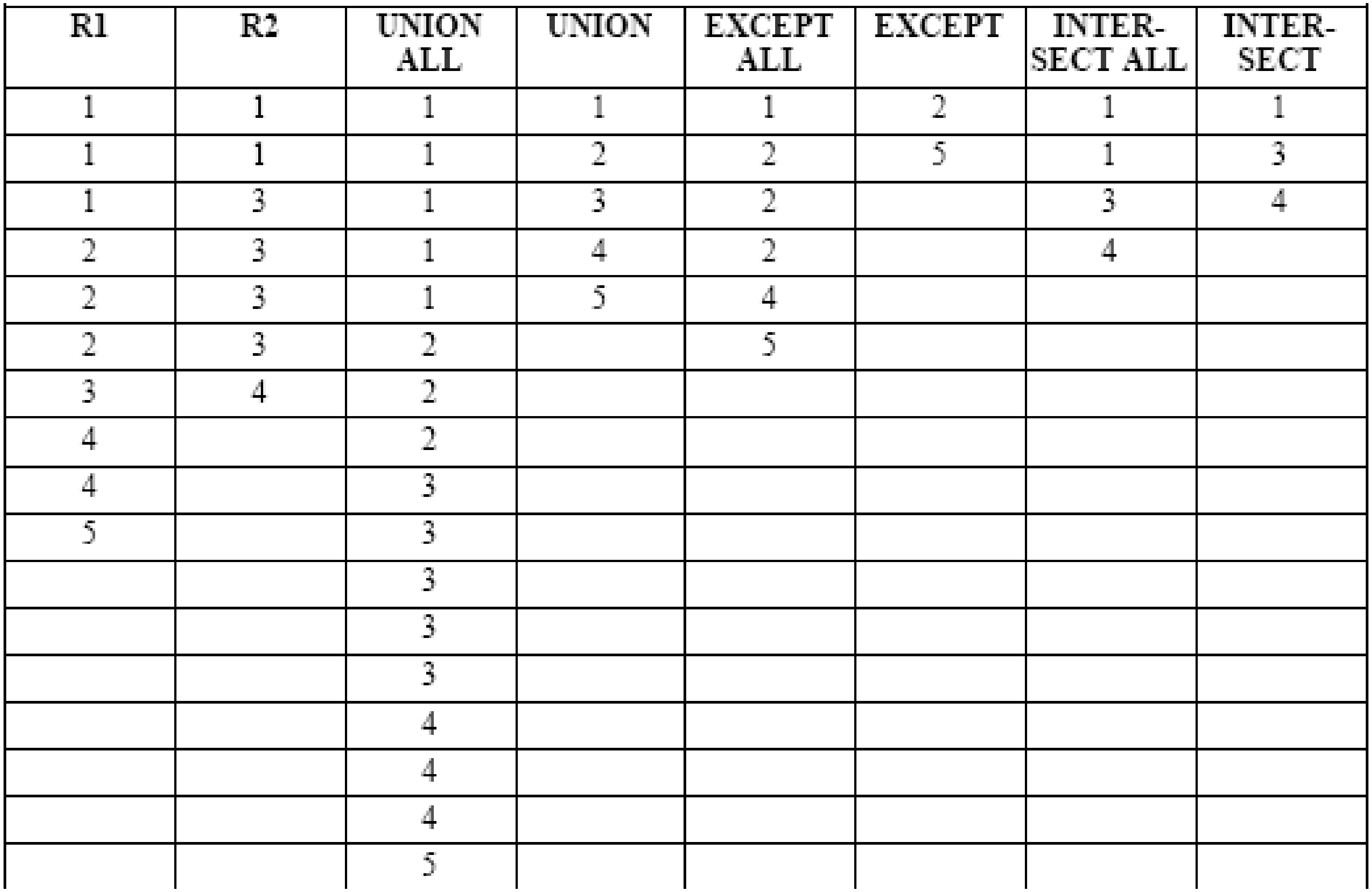
```
(SELECT * FROM T1) INTERSECT DISTINCT (SELECT * FROM T2)
```

If no NULL values are involved, this example returns the same results as:

```
SELECT DISTINCT * FROM T1
WHERE EXISTS
   (SELECT * FROM T2
   WHERE T1.C1 = T2.C2 AND T1.C2 = T2.C2 AND ...)
```

where the subquery contains an equal predicate for each pair of columns that exists in both tables.

Result of operations



R1	R2	UNION ALL	UNION	EXCEPT ALL	EXCEPT	INTERSECT ALL	INTERSECT
1	1	1	1	1	2	1	1
1	1	1	2	2	5	1	3
1	3	1	3	2		3	4
2	3	1	4	2		4	
2	3	1	5	4			
2	3	2		5			
3	4	2					
4		2					
4		3					
5		3					
		3					
		3					
		3					
		4					
		4					
		4					
		5					

SWG BetaWorks

INSTEAD OF triggers

DB2 9 for z/OS Technical
Education Series

INSTEAD OF triggers: current problem and goal

- Customers use views for read access control
- Many views are not updatable, so customers have to access base tables for data changes. Triggers can be used to help control updates.

- No INSERT / UPDATE / DELETE for read-only views

- Goal: to provide a mechanism to unify the target for all read / write access by an application (i.e., through views)

INSTEAD OF triggers

- A new type of trigger (~ BEFORE, AFTER triggers)
- Processed instead of the UPDATE, DELETE or INSERT statement that activated the trigger
- Can only be defined on views
  - provides an extension to the updatability of views
  - requested update operation against the view gets replaced by the trigger logic
  - application still believes all operations are performed against the view
  - applicable even for updatable views

Example

```
CREATE TABLE WEATHER (CITY VARCHAR(25), TEMPF DECIMAL(5,2))
CREATE VIEW CELCIUS_WEATHER (CITY, TEMPC) AS
    SELECT CITY, (TEMPF-32)*5.00/9.00 FROM WEATHER

CREATE TRIGGER CW_INSERT INSTEAD OF INSERT ON
     CELCIUS_WEATHER
REFERENCING NEW AS NEWCW DEFAULTS NULL
FOR EACH ROW MODE DB2SQL
  INSERT INTO WEATHER VALUES (NEWCW.CITY,
                               9.00/5.00*NEWCW.TEMPC+32)

CREATE TRIGGER CW_UPDATE INSTEAD OF UPDATE ON
     CELCIUS_WEATHER
REFERENCING NEW AS NEWCW OLD AS OLDCW DEFAULTS NULL
FOR EACH ROW MODE DB2SQL
  UPDATE WEATHER AS W
   SET W.CITY = NEWCW.CITY,
       W.TEMPF = 9.00/5.00*NEWCW.TEMPC+32
   WHERE W.CITY = OLDCW.CITY
```

The trigger can use . . .

- Can use transition variables, transition tables
- All SQL statements allowed in AFTER triggers
- Requires authorization similar to ALTER view

Restrictions

- Only 1 INSTEAD OF INSERT, UPDATE, DELETE per view
- View cannot be symmetric
- Only has row granularity
- No WHEN clause
- Cannot specify UPDATE OF column list
- new REFERENCING DEFAULTS NULL clause
- Cannot change transition variables
- Does not work with position UPDATE / DELETE
- No LOB, XML

- SELECT FROM UPDATE/DELETE/INSERT not supported
- MERGE into a view with INSTEAD OF trigger is not supported

DROP TRIGGER / VIEW

- DROP view also drops INSTEAD OF triggers
- DROP trigger invalidates other packages (including trigger packages) that depends on the dropped INSTEAD OF trigger

```
create trigger tr1 instead of update on v1
        begin ... end
create trigger tr2 after update on t1
        begin
                update v1...     --> tr2 depends on tr1
        end
drop trigger tr1     --> package tr2 is invalidated
```

Catalog changes

- 'I' for TRIGTIME column in SYSTRIGGERS
- non-0 OBID, DBID columns in SYSTABLES for triggering views
- New BTYPE value of ‘E’ in SYSPLANDEP, SYSPACKDEP to reflect dependency on INSTEAD OF trigger

SWG BetaWorks

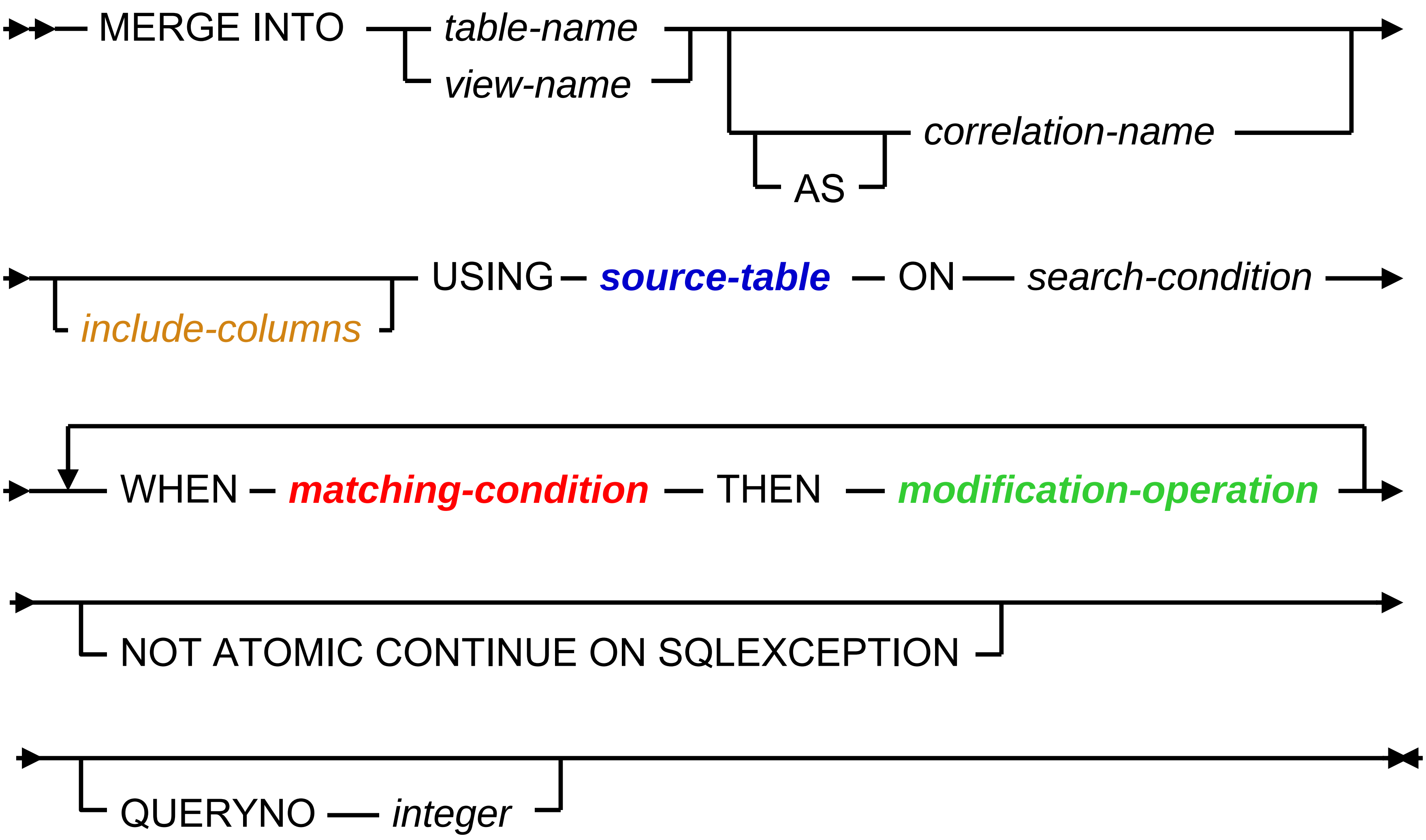
MERGE

DB2 9 for z/OS Technical
Education Series

MERGE

- Combine UPDATE and INSERT operation to a target table or view, from a input source of host-variable-arrays modeled as a source table
  - When source rows match to target, update target rows from source
  - When source rows do not match to target, insert source rows into target

MERGE



MERGE INTO *table-name* *view-name* AS *correlation-name*

include-columns USING **source-table** ON *search-condition*

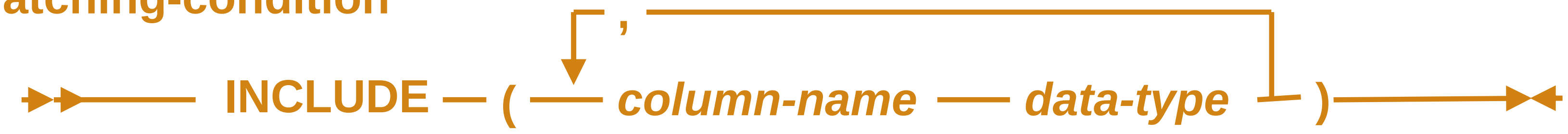
WHEN **matching-condition** THEN **modification-operation**

NOT ATOMIC CONTINUE ON SQLEXCEPTION

QUERYNO *integer*

include-columns and *source-table*

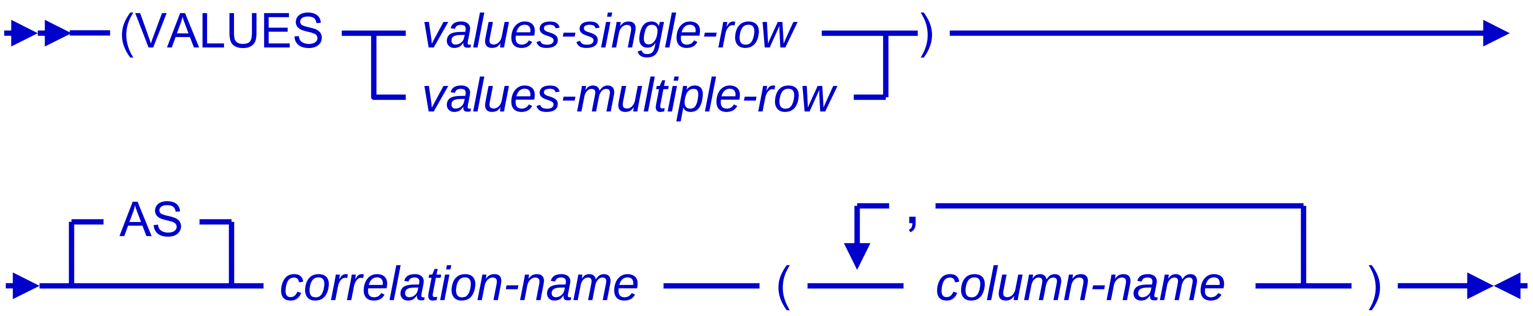
matching-condition



INCLUDE (*column-name* *data-type*)

,

source-table:



(VALUES *values-single-row*)

values-multiple-row

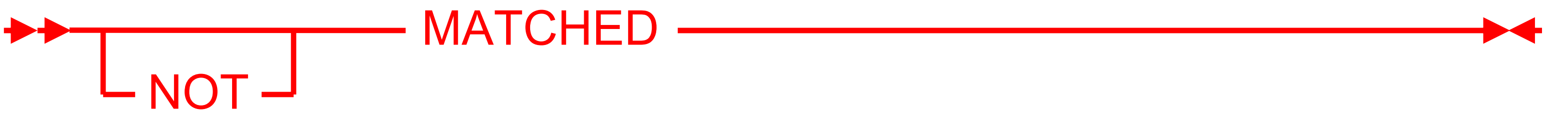
AS

correlation-name (*column-name*)

,

matching-condition

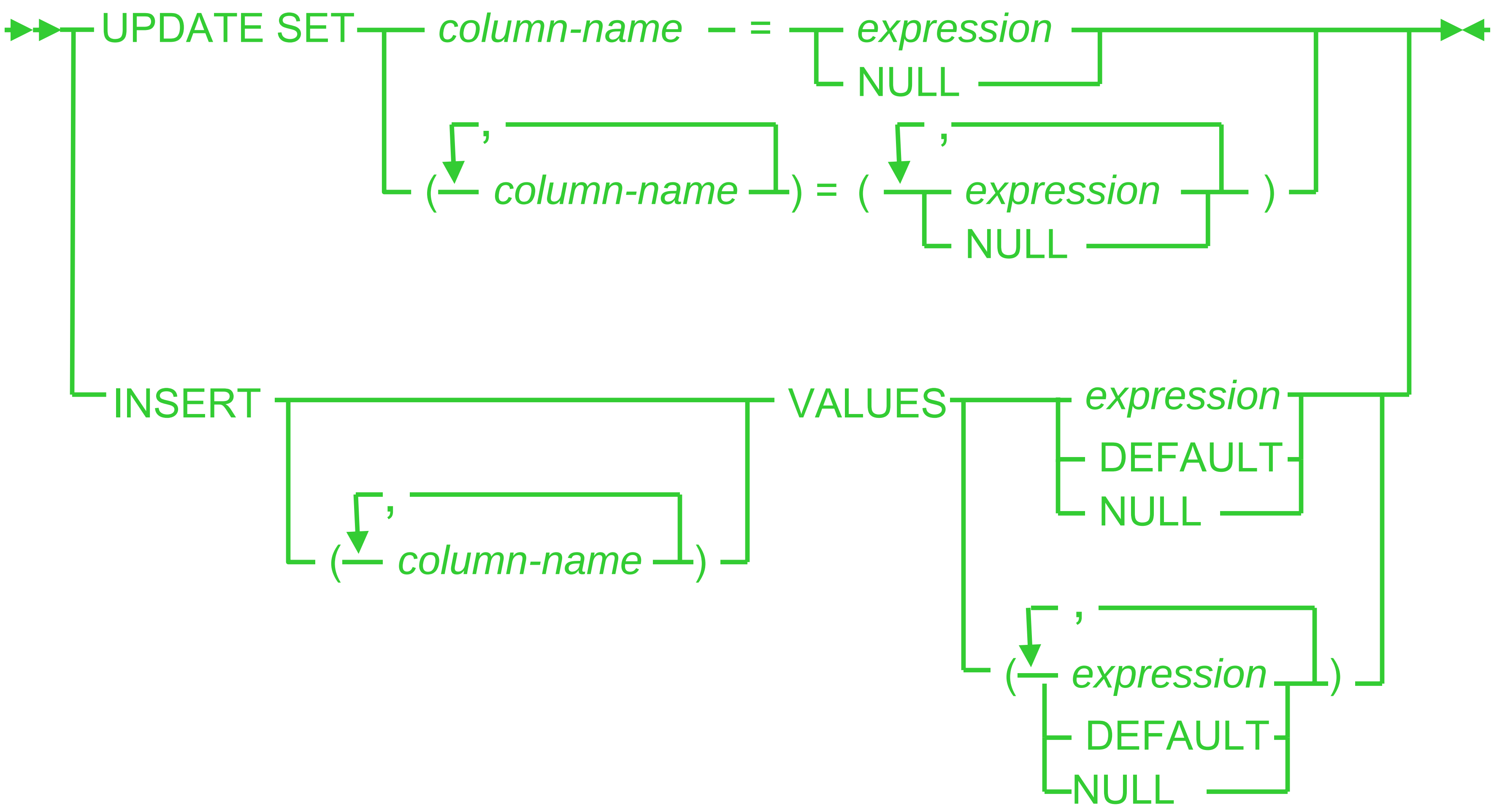
matching-condition



- WHEN MATCHED
  - only UPDATE SET allowed in THEN
  - can be specified at most once

- WHEN NOT MATCHED
  - only INSERT allowed in THEN
  - can be specified at most once

modification-operation



UPDATE SET *column-name* = *expression*

NULL

(, *column-name*) = (, *expression*)

NULL

INSERT (, *column-name*) VALUES *expression*

DEFAULT

NULL

(, *expression*)

DEFAULT

NULL

Example

Source

S.id	S.amt
1	30
5	10
10	40
5	20
1	50

Account - changed

T.id	balance
1	1030
5	10
10	540
5	30
1	1080

Target

Account - before

T.id	balance
1	1000
10	500
200	600
300	300
315	100
500	4000
...	

```
MERGE  INTO account AS T
USING VALUES (:hv_id, :hv_amt) FOR 5 ROWS  AS S(id,amt)
ON T.id = S.id
WHEN MATCHED THEN
   UPDATE SET balance = T.balance + S.amt
WHEN NOT MATCHED THEN
   INSERT (id, balance) VALUES (S.id, S.amt)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

Account - after

T.id	balance
1	1080
5	30
10	540
200	600
300	300
315	100
500	4000
...	

EXPLAIN changes

- Plan_table
  - New QBLOCK_TYPE : "MERGE"
  - MERGE is QB(1)
    - "Source table" left join "target table" - only nested loop join
  - UPDATE is QB(2)
  - INSERT is QB(3)

- DSN_Statement_table
  - New STMT_TYPE of "MERGE"

Sample EXPLAIN

QBLOCKNO	QBLOCK _TYPE	PLANNO	CORRELATI ON_NAME	TABLE _TYPE	JOIN_TYPE	METHOD	ACCESS TYPE
1	MERGE	1	S	B (1*)			V (2*)
1	MERGE	2	T	T	L	1 (3*)	(4*)
2	**UPDATE**	1	T	T			
3	INSERT	1	T	T			

1* : table_type of "B" is already supported in V8
2* : accesstype of "V" is already supported in V8
3* : Since we are doing "update in place", only Nested Loop Join is considered
4* : Since we are doing "update in place", if an index column is being updated, the index won't be considered for the table access to avoid problems

RID access ("I" with prefetch="L") won't be considered
Sparse index access ("T") won't be considered

No parallel support for MERGE.

MERGE notes

- Source data are piped into target
  - A row inserted into target is immediately available for update
  - A rows updated is immediately available for more update in the same statement
- NOT atomic - operation continues to next input row, even after the merge operation of an input row fails
- No MERGE trigger; UPDATE / INSERT triggers will be fired
- If target is a view with INSTEAD OF triggers, MERGE is not allowed

SWG BetaWorks

SELECT FROM MERGE, UPDATE, DELETE

DB2 9 for z/OS Technical
Education Series

Review: V8 -- SELECT FROM INSERT

- Benefits .....
  - Enhances usability and power of SQL
  - Enables user to immediately determine values inserted in tables by DB2 (identity, sequence, defaults, etc.) and before triggers
  - Cuts down on network cost in application programs
  - Cuts down on procedural logic in stored procedures
- What is it? .....
  - INSERT statement is now allowed in the FROM clause of a
    - SELECT statement that is a subselect
    - SELECT INTO statement
  - Users can automatically retrieve column values created by DB2 INSERT in single SELECT statement
    - Identity columns, sequence values
    - User-defined defaults, expressions
    - Columns modified by BEFORE INSERT triggers
    - ROWIDs

Example of SELECT FROM INSERT

ROWID NOT NULL GENERATED ALWAYS

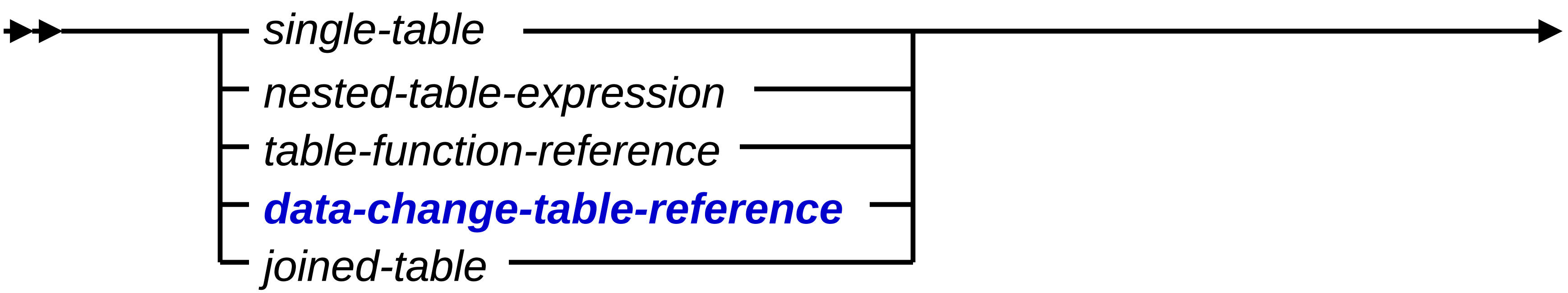
```
DECLARE CS1 CURSOR FOR
SELECT  EMP_ROWID
FROM FINAL TABLE
(INSERT INTO DSN8810.EMP_PHOTO_RESUME (EMPNO)
          SELECT EMPNO FROM DSN8810.EMP));
```

SELECT FROM UPDATE / DELETE / MERGE

- SELECT from UPDATE or DELETE will be implemented by allowing a searched UPDATE or searched DELETE statement in the FROM clause of a select-statement that is a subselect or in the SELECT INTO statement. By allowing a searched UPDATE or searched DELETE to appear in a select-statement or SELECT INTO statement, the database will allow the user to know which values were updated in a table and which rows were deleted from a table via a single SQL statement.
- SELECT FROM MERGE will return all the updated rows and inserted rows, including column values which are generated by DB2.
- An INCLUDE column specification is being introduced to allow the user to identify a new column for the select-list and as a method for sorting the data.

Syntax: SELECT FROM UPDATE / DELETE / MERGE

- Like the INSERT statement, the FROM clause of a SELECT statement will now allow an UPDATE or DELETE statement:



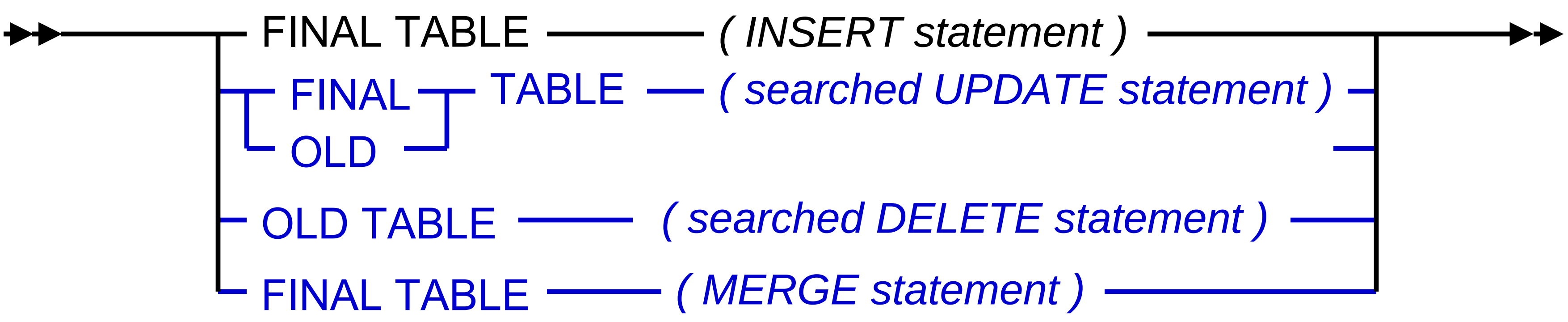
single-table

nested-table-expression

table-function-reference

data-change-table-reference

joined-table



FINAL TABLE *(INSERT statement)*

FINAL TABLE *(searched UPDATE statement)*

OLD TABLE *(searched UPDATE statement)*

OLD TABLE *(searched DELETE statement)*

FINAL TABLE *(MERGE statement)*

Notes

- **(Searched UPDATE statement)** Specifies a searched UPDATE statement as described under "UPDATE“ in the SQL Reference. A WHERE clause or a SET clause in the UPDATE statement cannot contain correlated references to columns outside of the UPDATE statement (SQLSTATE 42703, SQLCODE -206). The target of the UPDATE must be an updatable base table, an updatable symmetric view, or an updatable view where the view definition has no WHERE clause.
- **(Searched DELETE statement)** Specifies a searched DELETE statement as described under "DELETE“ in the SQL Reference. A WHERE clause in the DELETE statement cannot contain correlated references to columns outside of the DELETE statement (SQLSTATE 42703, SQLCODE -206). The target of the DELETE must be a deletable base table, a deletable symmetric view, or a deletable view where the view definition has no WHERE clause.
- **correlation-clause** (not shown on previous diagram; occurs after the UPDATE / DELETE / INSERT / MERGE statement) A correlation-name provides an alternative name that can be used when referencing columns of the intermediate result table. If no correlation-name is specified, then the exposed name is the name of the target table or view of the SQL data change statement. Otherwise, the exposed name is the *correlation-name*.

The content of the intermediate result table for a table reference containing an SQL data change statement is determined when the cursor is opened. The intermediate result table will contain all manipulated rows, including all of the columns in the specified target table or view. All of the columns of the target table or view of an SQL data change statement are accessible using the names from the target table or view unless they are renamed by the correlation clause. If a *correlation-clause* is not specified, the column names can be qualified by the target table or view name of the SQL data change statement. If an INCLUDE clause was specified as part of the SQL data change statement the intermediate result table will contain these additional columns.

Examples

- A user would like to know the sum of salaries of employees who are at level 'OPERATOR' and received a salary increase. In this scenario we can use FINAL TABLE with a searched UPDATE:

```
SELECT sum(salary) INTO :salary
FROM FINAL TABLE
   (UPDATE emp
   SET salary = salary * 1.05
   WHERE level = 'OPERATOR');
```

Examples . . . continued

- If a user would like to know the new salary of each employee who is at level 'OPERATOR' and received a salary increase, they could use FINAL TABLE with a searched UPDATE:

```
DECLARE CS1 CURSOR FOR
SELECT salary
FROM FINAL TABLE
   (UPDATE emp
   SET salary = salary * 1.05
   WHERE level = 'OPERATOR');

FETCH CS1 INTO :salary;
```

Examples . . . continued

- If a user would like to know the years of service of each employee who is at level 'OPERATOR' and was removed from the database, they could use OLD TABLE with a searched DELETE:

```
DECLARE CS1 CURSOR FOR
SELECT YEAR(CURRENT DATE - HIREDATE)
FROM OLD TABLE
   (DELETE FROM emp
   WHERE level = 'OPERATOR');

FETCH CS1 INTO :years_of_service;
```

Example – SELECT FROM FINAL TABLE (MERGE)

Source

S.id	S.amt
1	30
5	10
10	40
5	20
1	50
99	90

returned rows

T.id	balance	status
1	1030	upd
5	10	ins
10	540	upd
5	30	upd
1	1080	upd
99	90	ins

include columns

Account - target table

T.id	balance
1	1000
10	500
200	600
300	300
315	100
500	4000
…	

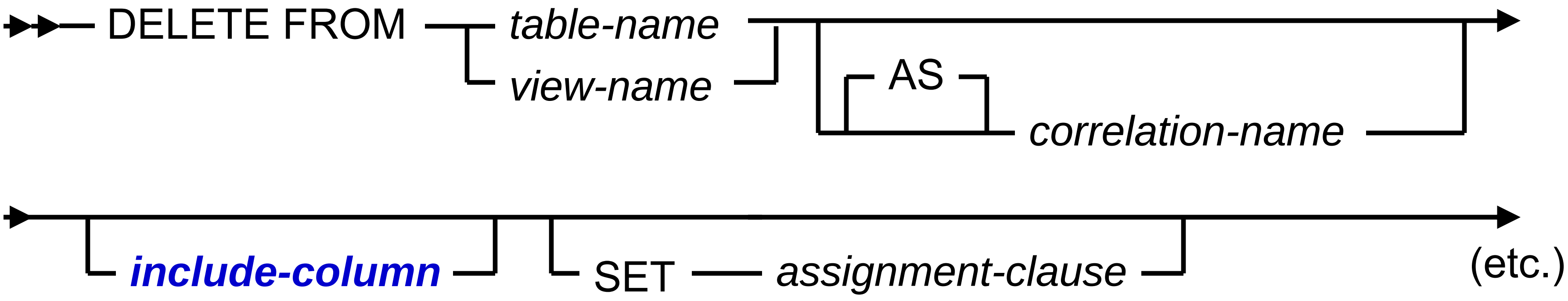
Account - after

T.id	balance
1	1080
5	30
10	540
99	90
200	600
300	300
315	100
500	4000
…	

```
SELECT   id, balance, status
FROM     FINAL TABLE (
MERGE INTO      account AS T   INCLUDE( status char(3))
USING  (VALUES  (:hv_id, :hv_amt) FOR 6 ROWS)  AS S (id,amt)
ON T.id = S.id
WHEN MATCHED THEN
   UPDATE SET balance = T.balance + S.amt,
                     status    = 'upd'
WHEN NOT MATCHED THEN
   INSERT (id, balance) VALUES (S.id, S.sum_amt,'ins')
NOT ATOMIC CONTINUE ON SQLEXCEPTION
)
```

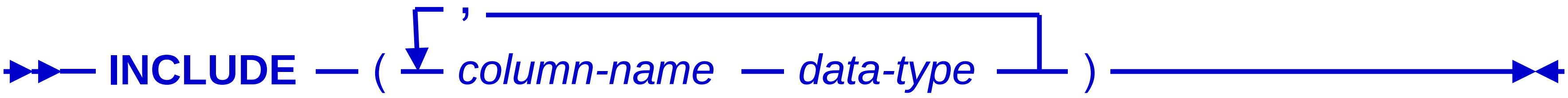
INCLUDE columns

- Introduces a list of columns to be included in the result table of the DELETE / INSERT / UPDATE / MERGE statement.
- The include columns are only available if the DELETE / INSERT / UPDATE / MERGE statement is nested in the FROM clause of a *select-statement* or SELECT INTO statement.



DELETE FROM *table-name* *view-name* AS *correlation-name*

include-column SET *assignment-clause* (etc.)



INCLUDE (, *column-name* *data-type*)

Examples

- The INCLUDE column is being introduced with the SQL data change statements. The INCLUDE column allows the user to specify an additional column on the select-list. Suppose a user would like to remove employees from a database and determine the salary and years of service for that employee:

```
DECLARE CS1 CURSOR FOR
SELECT salary, Years_Of_Service
FROM OLD TABLE
   (DELETE FROM emp INCLUDE ( Years_Of_Service INTEGER )
   SET Years_Of_Service = YEAR ( CURRENT DATE - Start_Date)
   WHERE Level = 'CONTRACTOR');
```

Examples . . . continued

- Here is an example using a searched UPDATE statement with an INCLUDE column where the user wants to see the salary of employees prior to updating the salary:

```
DECLARE CS1 CURSOR FOR
SELECT Name, Salary, Old_Salary
FROM FINAL TABLE
   (UPDATE emp INCLUDE ( Old_Salary DECIMAL(9,2) )
   SET Salary = Salary * 1.1, Old_Salary = Salary
   WHERE Level = 'ASSOCIATE');
```

Examples . . . continued

- In the next example, the PROJ table is being populated with employee department numbers and the user would like to see all of the managers for those departments:

```
DECLARE CS1 CURSOR FOR
SELECT manager_num, projname
FROM FINAL TABLE
   (INSERT INTO proj (deptno) INCLUDE ( manager_num CHAR(6) )
   SELECT deptno, mgrno FROM dept);
```

SWG BetaWorks

TRUNCATE

DB2 9 for z/OS Technical
Education Series

TRUNCATE TABLE customer requirements

- Delete rows from a table without firing DELETE triggers
- Have an option to LOAD REPLACE that works on a table level in a segmented table space with multiple tables

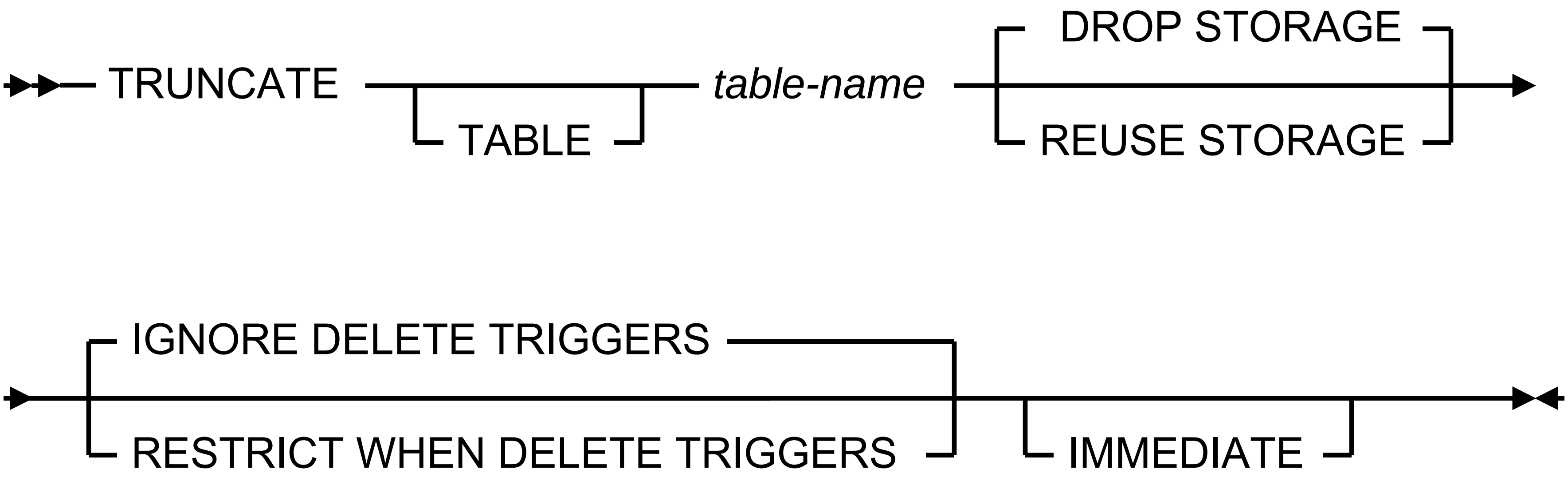
What TRUNCATE does

- Gives users an alternative way of emptying a table, with more flexibility over the current DELETE statement with no WHERE clause (i.e., a mass delete operation):
  - Delete all data rows in a designated DB2 table without activating DELETE triggers
  - DB2 catalog definition of the table (i.e., dropping and recreating of the delete triggers) is not needed for faster processing
  - Provides an option to allow the users to empty the designated DB2 table permanently without going through the current commit phase
  - Provides an option to reuse deallocated storage

Processing modes for TRUNCATE

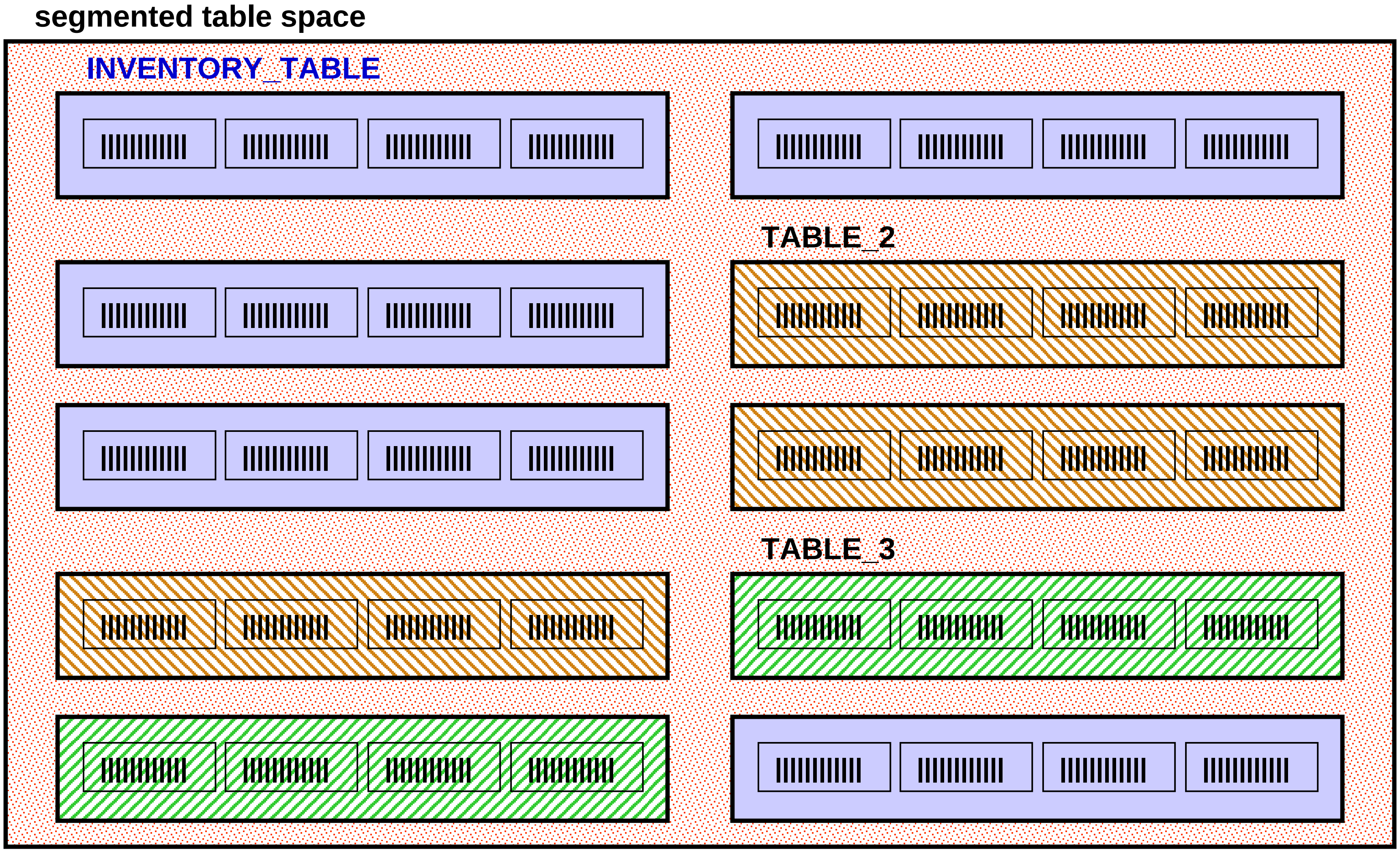
- *Normal way* truncate operation must process each data page to physically delete data records from the page
  - table in a simple table space
  - table in a partitioned table space
  - any table with table attributes
    - CDC-enabled (Change Data Capture)
    - MLS-enabled (Multiple Level Security)
    - VALIDPROC-enabled
- *Fast way* truncate operation deletes data records without physically processing each data page
  - table in a segmented table space or a universal table space without the above table attributes

TRUNCATE



TRUNCATE
TABLE
table-name
DROP STORAGE
REUSE STORAGE
IGNORE DELETE TRIGGERS
RESTRICT WHEN DELETE TRIGGERS
IMMEDIATE

Storage -- 3 tables in segmented table space



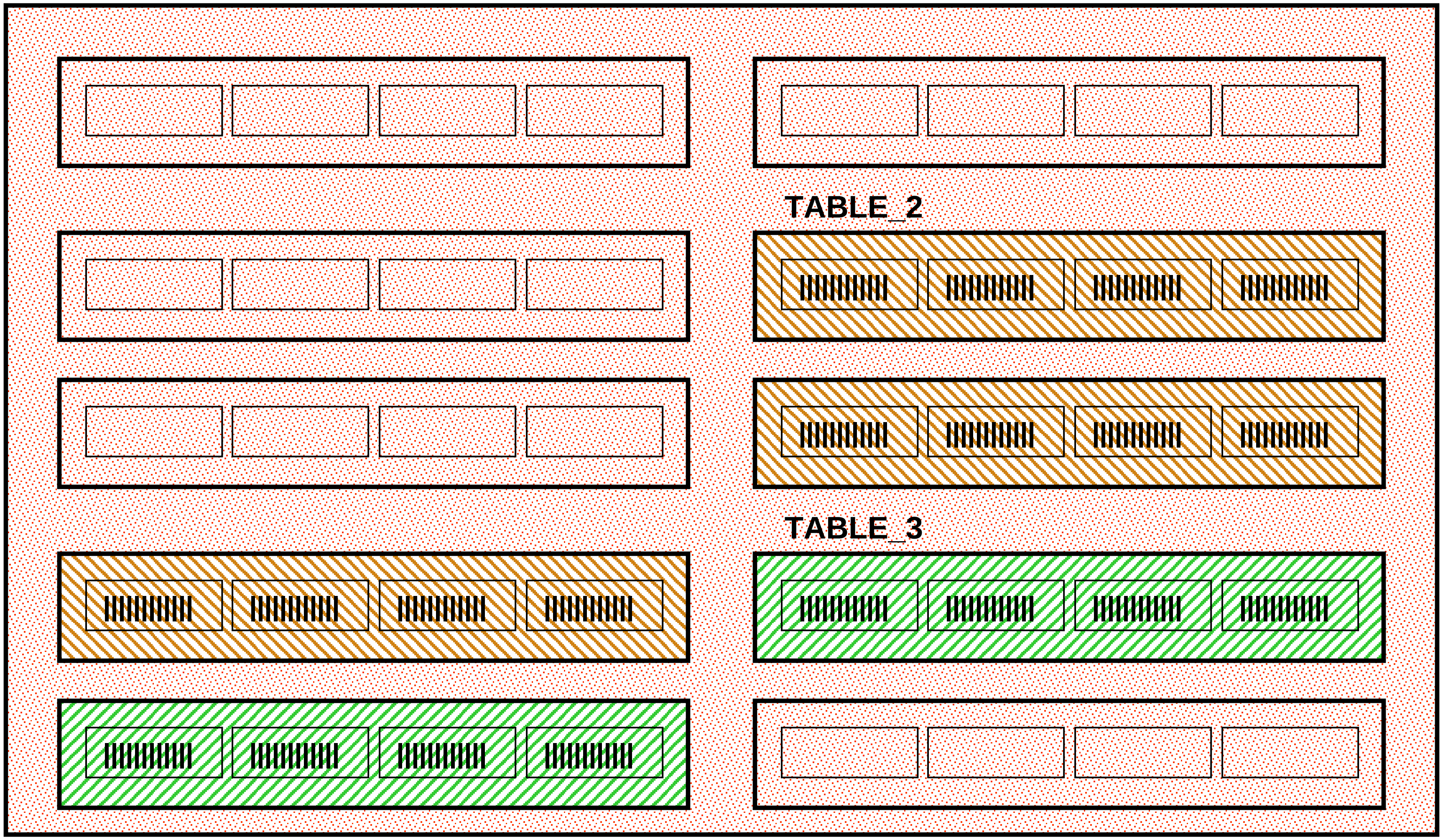
Example 1 . . . DROP STORAGE

- Let's say a user would like to empty an old inventory table regardless any existing delete triggers and also like to return its allocated space. In this scenario, the DROP STORAGE and IGNORE DELETE TRIGGERS clauses are used.

```
TRUNCATE INVENTORY_TABLE
   IGNORE DELETE TRIGGERS
   DROP STORAGE;
```

TRUNCATE INVENTORY_TABLE . . . DROP STORAGE

segmented table space



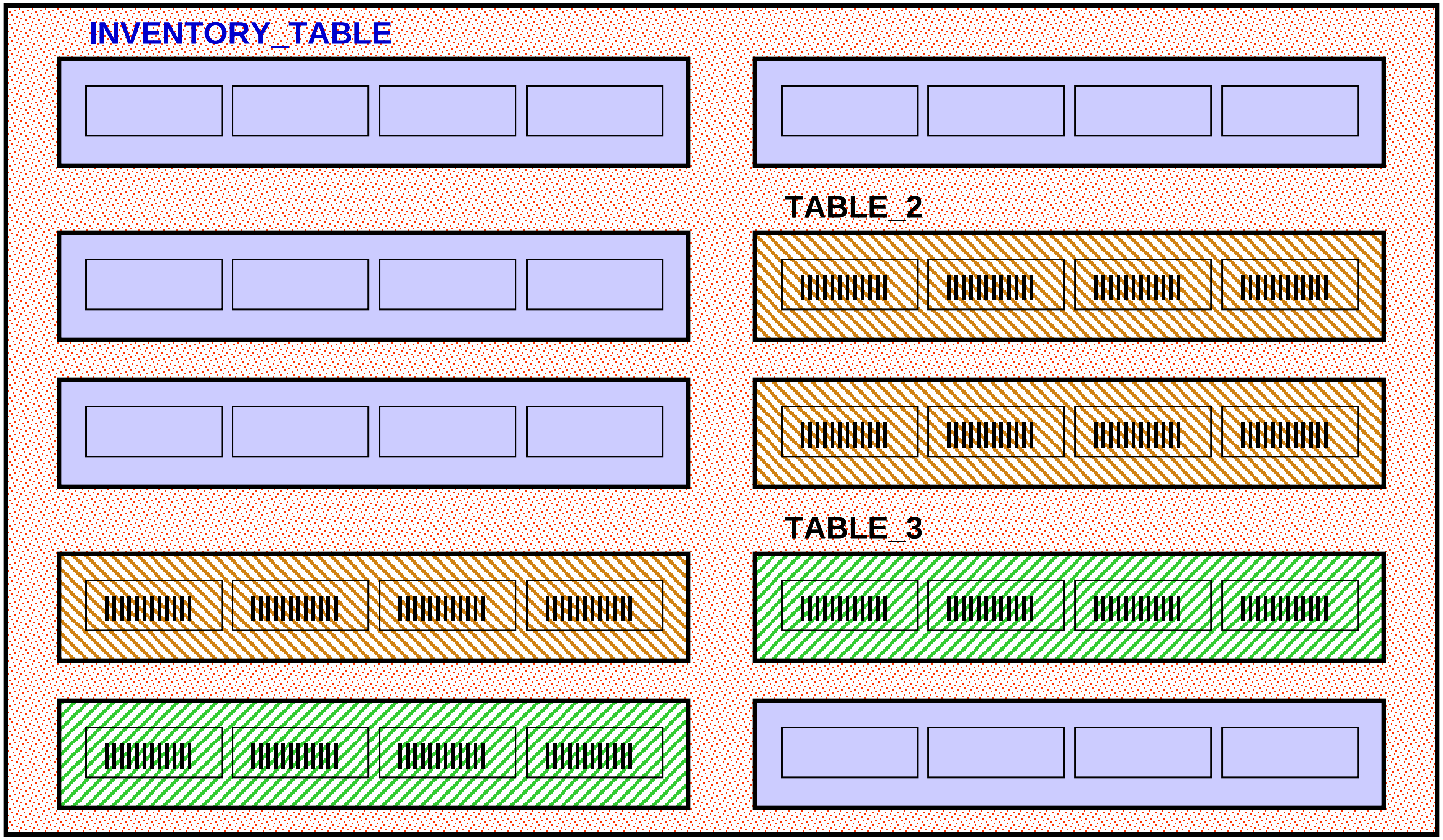
Example 2 . . . REUSE STORAGE

- If a user would like to empty an old inventory table regardless any existing delete triggers but also like to preserve its allocated space for later reuse. In this scenario, the REUSE STORAGE and IGNORE DELETE TRIGGERS clauses are used. At here, we are assuming the INVENTORY_TABLE is a table in the segmented table space.

```
TRUNCATE INVENTORY_TABLE
   REUSE STORAGE
   IGNORE DELETE TRIGGERS;
```

TRUNCATE INVENTORY_TABLE . . . REUSE STORAGE

segmented table space



TRUNCATE . . . IMMEDIATE

- Specifies that the truncate operation is processed immediately and cannot be undone.
- When IMMEDIATE option is specified, the table must not contain any uncommitted updates.
  - For a DGTT table object, the IMMEDIATE option does not apply to it. The truncate operation will fail since the table space contains a DGTT will be always in the update mode
  - No uncommitted DDL is allowed on the table prior to the TRUNCATE
- The truncated table is immediately available for use in the same unit of work.
- Although a ROLLBACK statement is allowed after the TRUNCATE statement, the truncate operation is not undone, and the table remains truncated. Other data changes following the TRUNCATE are rolled back.

SWG BetaWorks

ORDER BY and FETCH FIRST in subselect

DB2 9 for z/OS Technical
Education Series

Background

Prior to DB2 9, DB2 z/OS prohibit ORDER BY and FETCH FIRST n ROWS in a subselect.

i.e., one can write

```
SELECT * from T ORDER BY c1 FETCH FIRST 1 ROW ONLY;
```

but can not write

```
INSERT INTO temp
 (SELECT * from T ORDER BY c1 FETCH FIRST 1 ROW ONLY);
```

In DB2 9

- Allow all semantically relevant clauses of the select statement to be pushed into subqueries. The original query can be taken as is and wrapped by more SQL, such as shown in the example above
- Provides more function by being able to select, e.g., the top n rows in a leg of a join, a leg of union, or a subquery.

```
(SELECT * FROM T1
  ORDER BY C1 FIRST 3  ROW ONLY)
UNION
SELECT * FROM T2
```

Customer requirement

One customer has a huge table of which they want just the first 2000 rows sorted in a particular order. Unfortunately, the sort is done first, and the fetch first after. This would cause a huge sort for no reason. They had to code this using a temp table which is a lot more work than a simple select. The solution to this to allow FETCH FIRST n ROWS in subquery:

```
SELECT A, B, C FROM
   (SELECT A,B,C FROM TABLEA
    WHERE...
    FETCH FIRST 2000 ROWS ONLY ) AS TABLEB
ORDER BY C,B
```

Functional description

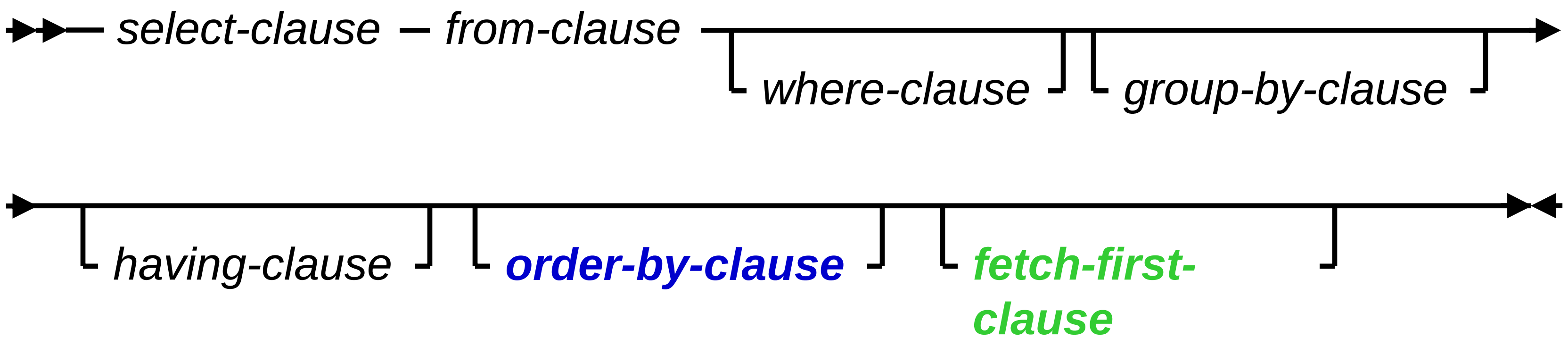
- Push down of ORDER BY into <fullselect> and <subselect>
- Push down of FETCH FIRST N ROWS into <fullselect> and <subselect>
- Addition of ORDER OF <table-identifier> to the ORDER BY clause
  The last addition allows the higher select to "pick up" the ordering of the derived table as shown in the following example

```
SELECT C1 FROM
  (SELECT C1 FROM T1
   UNION
   SELECT C1 FROM T2
   ORDER BY C1) AS UTABLE
ORDER BY ORDER OF UTABLE
```

In SQL there is otherwise no notion of guaranteeing an order to be maintained from a lower result set to a higher result set.

Syntax

subselect:



select-clause *from-clause*

where-clause

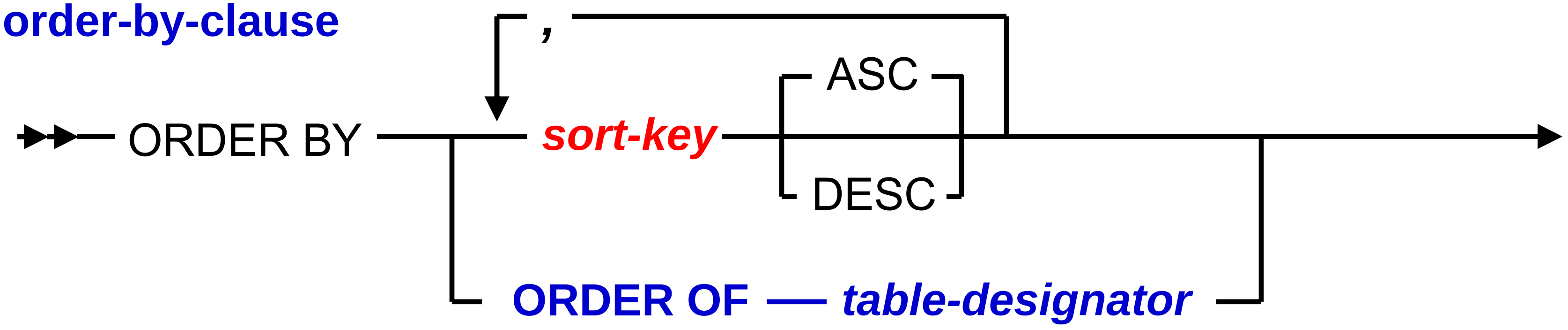
group-by-clause

having-clause

order-by-clause

fetch-first-clause

order-by-clause



,

ORDER BY

sort-key

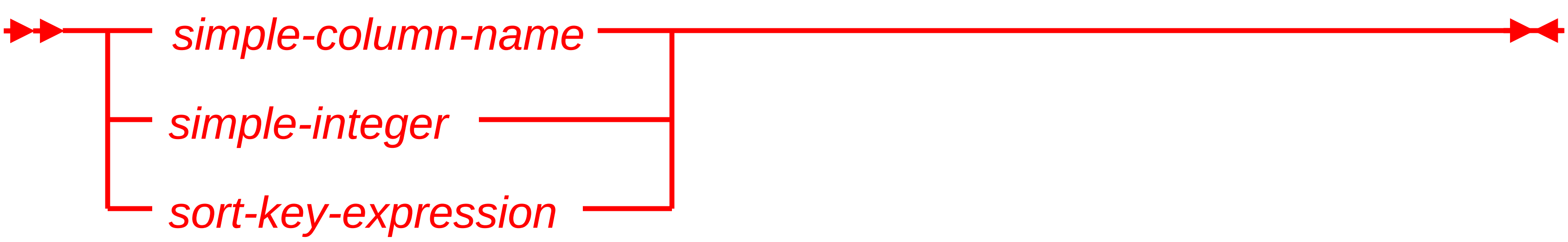
ASC

DESC

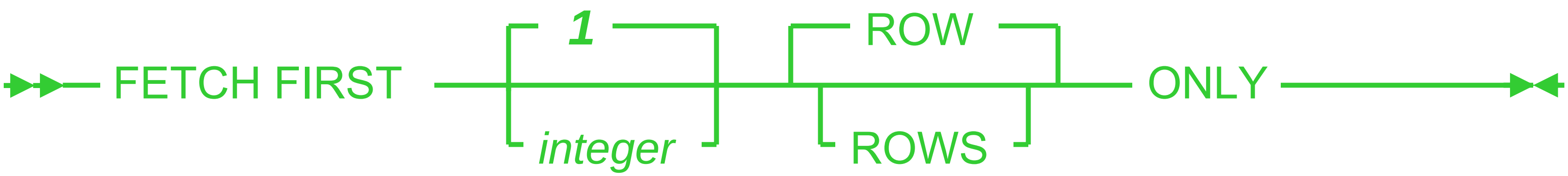
ORDER OF ***table-designator***

Syntax . . . continued

sort-key:



fetch-first-clause



Restrictions

- A subselect that contains an ORDER BY or FETCH FIRST clause cannot be specified:
  - In the outermost fullselect of a view.
  - In a materialized query table
  - Unless the subselect is enclosed in parenthesis

Example:

```
CREATE VIEW V1 AS
  (SELECT * FROM T1 ORDER BY C1);
```

SQLCODE -20211

```
SELECT * FROM T1
  ORDER BY C1
UNION
SELECT * FROM T2
  ORDER BY C2
```

SQLCODE -104

SWG BetaWorks

Data types and built-in functions

DB2 9 for z/OS Technical
Education Series

New data types

- BIGINT
- BINARY
- VARBINARY
- DECFLOAT

SWG BetaWorks

BIGINT

DB2 9 for z/OS Technical
Education Series

BIGINT

- An exact numeric capable of representing 63-bit integers
  - Range:
    - -9223372036854775808 to
    - 9223372036854775807
- Compatible with all numeric types

BIGINT-related functions

▶▶— BIGINT (*numeric-expression)* ————————————▶◀

- Example:

```
SELECT BIGINT (12345.6)
        FROM SYSIBM.SYSDUMMY1;
```

Returns 12345

BIGINT-related functions

►►— BIGINT (*string-expression)* ——————————————►◄

- Example:

```
SELECT BIGINT (‘00123456789012’)
        FROM SYSIBM.SYSDUMMY1;
```

Returns 123456789012

BIGINT extensions to existing functions

- CHAR
- DIGITS
- LENGTH
- MOD
- MULTIPLY_ALT
- POWER
- VARCHAR

SWG BetaWorks

BINARY and VARBINARY

DB2 9 for z/OS Technical
Education Series

BINARY and VARBINARY

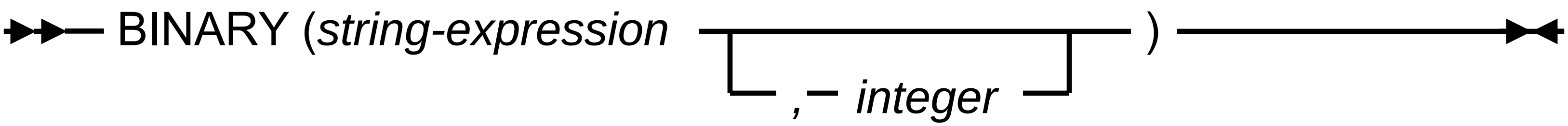
- BINARY fixed-length binary string
  - 1 to 255 bytes
- VARBINARY variable-length binary string
  - 1 to 32704 bytes; maximum length determined by the maximum record size associated with the table
- Both are compatible with BLOBs
- Neither are compatible with character string data types
  - Similar to FOR BIT DATA character strings
  - Can use CAST specification to change FOR BIT DATA character string into binary string
  - **There is a difference in padding characters:**
    - **[VAR]CHAR padded with spaces (X'40' for EBCDIC, X'20' for ASCII and Unicode)**
    - **BINARY padded with hex zeros (X'00')**
    - **VARBINARY not padded, even during comparisons**

Comparison of binary strings

- Two binary strings are equal only if the lengths are identical
- If two strings are equal up to the length of the shorter string length
  - the shorter string is considered less than the longer string
  - even when the remaining bytes in the longer string are hex zeros

Hex value of operand 1	relationship	Hex value of operand 2
X’4100’	<	X’410000’
X’4100’	<	X’42’
X’4100’	=	X’4100’
X’4100’	>	X’41’
X’4100’	>	X’400000’

BINARY-related functions



BINARY (*string-expression* , *integer*)

- Schema is SYSIBM
- Returns the fixed-length binary string representation
- ***string-expression*** returns a built-in character string, graphic string, vinary string, or row ID data type
- ***integer*** length of the resulting binary string; 1 to 255

Examples

- Following examples assume EBCDIC encoding of the input literal strings :

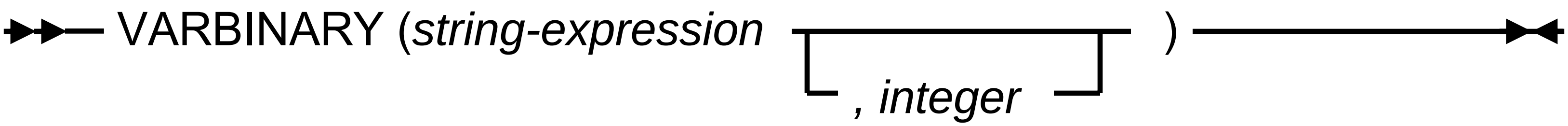
Example 1: The following function returns a fixed-length binary string with a length attribute 1 and a value BX'00'.

```
SELECT BINARY(",1) FROM SYSIBM.SYSDUMMY1;
```

Example 2: The following function returns a fixed-length binary string with a length attribute 5 and a value BX'D2C2C80000'

```
SELECT BINARY('KBH',5) FROM SYSIBM.SYSDUMMY1;
```

BINARY-related functions



VARBINARY (*string-expression* , *integer*)

- Schema is SYSIBM
- Returns the varying-length binary string representation
- ***string-expression*** returns a built-in character string, graphic string, vinary string, or row ID data type
- ***integer*** length of the resulting binary string; 1 to 32704

Examples

- Following examples assume EBCDIC encoding of the input literal strings:

Example 1: The following function returns a varying-length binary string with a length attribute 1, actual length 0, and a value of empty string.

```
SELECT VARBINARY(‘ ') FROM SYSIBM.SYSDUMMY1;
```

Example 2: The following function returns a varying-length binary string with a length attribute 5, actual length 3, and a value BX'D2C2C8'

```
SELECT VARBINARY('KBH',5) FROM SYSIBM.SYSDUMMY1;
```

Functions extended for BINARY and VARBINARY

- INSERT
- LEFT
- LTRIM
- POSSTR; POSITION does not support binary types
- REPEAT
- REPLACE
- RIGHT
- RTRIM
- STRIP
- SUBSTR

Online schema and binary columns

- **A column data type could be altered only to a compatible data type.**

- However, to ease the migration of existing applications, altering CHAR FOR BIT DATA or VARCHAR FOR BIT DATA column data types to BINARY or VARBINARY data types will be allowed (even though they are not considered to be compatible).

- When a CHAR FOR BIT DATA, or VARCHAR FOR BIT DATA column is altered to a BINARY or VARBINARY data type, and there is an index defined on that column, the index will be put in RBDP.

- Altering BINARY or VARBINARY data types to CHAR FOR BIT DATA or VARCHAR FOR BIT DATA will not be allowed.

Caution!!

- Caution should be taken when a CHAR FOR BIT DATA column is altered to a BINARY data type due to differences in padding.
- When a CHAR FOR BIT DATA column is altered to BINARY, the existing space characters in the table will not be changed to hexadecimal zeros (X'00). In addition, if the new length attribute is greater than current length attribute of the column, the values in the table are padded with hexadecimal zeros (X'00).

```
CREATE TABLE T1 ( C1 CHAR(5) FOR BIT DATA) CCSID EBCDIC;
INSERT INTO T1 VALUES(X'C1C2C3');
INSERT INTO T1 VALUES(X'C1C2C3C4C5');
COMMIT;

SELECT HEX( C1 ) FROM T1;
        returns:           C1C2C34040
                           C1C2C3C4C5
```

Caution!! . . . continued

```
ALTER  TABLE  T1 ALTER  COLUMN  C1  SET DATA TYPE BINARY(10) ;
COMMIT;

SELECT HEX( C1 ) FROM T1;
        returns:              C1C2C34040000000000000
                              C1C2C3C4C50000000000
```

Notes

- DSNTEP2, DSNTEP4, and DSNTIAUL are modified to support the BIGINT, VARBINARY and BINARY data types.
- PARTITION BY RANGE: Do not specify a VARBINARY column, a BINARY column, or a column with a distinct type that is based on a BINARY, or a VARBINARY data type as a column in the partitioning key.
- PADDED option can not be specified for indexes defined on VARBINARY columns.

SWG BetaWorks

BINARY and VARBINARY

Reference material

DB2 9 for z/OS Technical
Education Series

Declarations generated by DCLGEN

SQL data type	C	COBOL	PL/I
BIGINT	long long int	PIC S9(18) USAGE COMP	FIXED BIN(63)
BINARY(n)	SQL TYPE IS BINARY(n)	USAGE SQL TYPE IS BINARY(n)	SQL TYPE IS BINARY(n)
VARBINARY(n)	SQL TYPE IS VARBINARY(n)	USAGE SQL TYPE IS VARBINARY(n)	SQL TYPE IS VARBINARY(n)

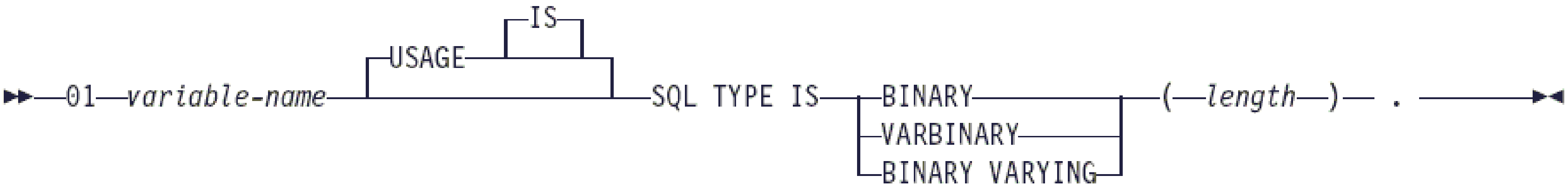
- BIGINT, BINARY, and VARBINARY will not be supported in FORTRAN

BIGINT, binary host variables in assembler

SQL data type	assembler equivalent	Notes
BIGINT	DS FD or DS FDL8	
BINARY(n)	DS XLn	1<=n<=255
VARBINARY(n)	DS HL2,XLn	1<=n<=255

Binary host variables in COBOL applications

- COBOL does not have variables that correspond to the SQL binary data types. To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a COBOL language structure in the output source member.



```
                         ┌─IS─┐
             ┌─USAGE─────┴────┤
>>──01──variable-name──┴───────────┴──SQL TYPE IS──┬─BINARY─────────┬──(──length──)── . ────────><
                                                   ├─VARBINARY──────┤
                                                   └─BINARY VARYING─┘
```

You declare this variable	DB2 generates this variable
01 BIN-VAR USAGE IS SQL TYPE IS BINARY(10).	01 BIN-VAR PIC X(10).
01 VBIN-VAR USAGE IS SQL TYPE IS VARBINARY(10).	01 VBIN-VAR. 49 VBIN-VAR-LEN PIC S9(4) COMP-5. 49 VBIN-VAR-TEXT PIC X(10).

Binary host variables in C and C++ applications

- C and C++ do not have variables that correspond to the SQL binary data types. To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a C language structure in the output source member.

- When you refer to a BINARY or VARBINARY host variable in an SQL statement, you must use the variable you specified in the SQL TYPE declaration.
- When you refer to the host variable in a host language statement, you must use the variable that DB2 generates.
- Note, however, that caution should be taken when operating on the binary host variable. DB2 uses C data type *char* but the declaration does not account for the null terminator, because binary strings are not null-terminated strings, in fact , binary string could contain some zeros (0x00) anywhere in it.

BIGINT, binary host variables in C and C++

You declare this variable	DB2 generates this variable
SQL TYPE IS BINARY(10) bin_var;	char bin_var[10];

SQL TYPE IS VARBINARY(10) vbin_var;	struct {
	    short length;
	    char data[10];
	  } vbin_var;

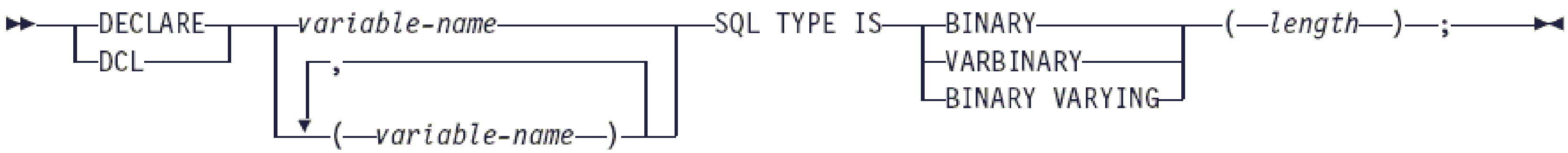
SQL data type	C equivalent
BIGINT	long long int long long sqlint64 ← use this for portability
BINARY(n)	SQL TYPE IS BINARY(n), 1 <= n <= 255
VARBINARY(n)	SQL TYPE IS VARBINARY(n)), 1 <= n <= 32704

Binary host variables in Java applications

SQL data type	Recommended Java data type or Java object type
BINARY(n), 1 <=n <= 255	byte[ ]
VARBINARY(n), 1 <= n <= 32704	byte[ ]

Binary host variables in PL/I applications

- PL/I does not have variables that correspond to the SQL binary data types. To create host variables that can be used with these data types, use the SQL TYPE IS clause. The SQL precompiler replaces this declaration with a PL/I language structure in the output source member.



PL/I data type	SQLTYPE of host variable	SQLLEN of host variable	SQL data type
SQL TYPE IS BINARY(n), 1<=n<=255	912	n	BINARY(n)
SQL TYPE IS VARBINARY(n), 1<=n<=32704	908	n	VARBINARY(n)

SWG BetaWorks

DECFLOAT

DB2 9 for z/OS Technical
Education Series

DECFLOAT

- Decimal floating point (DECFLOAT) is similar to both
  - Packed decimal (or binary coded decimal), and
  - Floating point (IEEE or hex)
- The main advantages that decimal floating point has over packed decimal or binary floating point (IEEE):
  - it can contain a larger number
    - in terms of digits of significance
    - in terms of exponent
- The rules for manipulation of DECFLOAT more closely follow the rules for manipulation of packed decimal:
  - DECFLOAT processing deals with exact numbers
  - IEEE floating point (binary) deals with numerical approximations

DECFLOAT details

- All numbers have a sign, a precision, and a scale
- The precision is the total number of decimal digits excluding the sign (either 16 or 34 for DECFLOAT)
- The scale is the total number of decimal digits to the right of the decimal point
- An exponent range of respectively 10^{-383} to 10^{+384} or 10^{-6143} to 10^{+6144}
- DECFLOAT(16):
  - Negative values: $-9.999999999999999 \times 10^{+384}$ to $-1.000000000000000 \times 10^{-383}$
  - Positive values: $1.000000000000000 \times 10^{-383}$ to $9.999999999999999 \times 10^{+384}$
- DECFLOAT(34):
  - Negative values: $-9.999999999999999999999999999999999 \times 10^{+6144}$ to $-1.000000000000000000000000000000000 \times 10^{-6143}$
  - Positive values: $1.000000000000000000000000000000000 \times 10^{-6143}$ to $9.999999999999999999999999999999999 \times 10^{+6144}$

Equality for DECFLOAT

- The DECFLOAT data type allows for multiple bit representations of the same number. Additionally, numbers with the same coefficient can have different exponents, and therefore different bit representations.
- For example 2.00 and 2.0 are two numbers with the same coefficient, but different exponent values.
- Thus, 2.00 <> 2.0 at a binary level, however the = (equal) predicate will return true for a comparison of 2.0 = 2.00. Given that 2.0 = 2.00 (the comparison is true), 2.0 < 2.00 is false. The behavior that is described here holds true whenever DB2 compares DECFLOAT data (such as for UNION, SELECT DISTINCT DECFLOAT_column, COUNT(DISTINCT DECFLOAT_column), basic predicates, IN predicates, etc)
- Example:

```
SELECT 2.0 FROM SYSIBM.SYSDUMMY1
UNION [DISTINCT]
SELECT 2.00 FROM SYSIBM.SYSDUMMY1              yields 1 row
```

Special representations

- The DECFLOAT data type supports the specification of negative and positive NaN (quiet and signalling), and negative and positive infinity. From an SQL perspective, Infinity = Infinity, NaN = NaN, and sNaN = sNaN.
- For "order-by-clause", the order for DECFLOAT data are as follow:

-NaN < -Infinity < negative numbers < 0 < positive numbers < Infinity < NaN

- NaN = not-a-number
  - **quiet NaN** - a value representing undefined results which does not cause an invalid number condition
  - **signaling NaN** - a value representing undefined results which will cause an invalid number condition if used in any operation defined in any numerical operation.
  - When a number has either one of these special values, its coefficient and exponent are undefined. The sign of an infinity is significant (that is, it is possible to have both positive and negative infinity). The sign of a NaN has no meaning for arithmetic operations.

Examples involving special values

Infinity + 1	==>	Infinity
inf + inf	==>	Infinity
inf + -Inf	==>	NaN (exception)
NaN + 1	==>	NaN
NaN + Infinity	==>	NaN
1 - Infinity	==>	-Infinity
inf - inf	==>	NaN (exception)
-inf - -inf	==>	NaN (exception)
-1 * Infinity	==>	-Infinity
1.0E1 / 0	==>	Infinity
-1.0E5 / 0.0	==>	-Infinity
Inf / -Inf	==>	NaN (exception)
Inf / 0	==>	Inf
-inf / 0	==>	-Infinity

Special representations

- The following DECFLOAT constants represent the numbers 1234567890123456789, sNAN, and negative Infinity:

  1234567890123456789E0   snan   -INF

- The special values positive and negative infinity and not-a-number have lexical representations INF, -INF and NaN, respectively. Lexical representations for zero may take a positive or negative sign.

Rounding DECFLOAT numbers

DSNHDECP	DECFLOAT ROUND MODE
BIND option	ROUNDING
Special register	CURRENT DECFLOAT ROUNDING MODE
Utilities	LOAD and UNLOAD

- Values:

ROUND_CEILING	round towards +infinity
ROUND_DOWN	round towards 0 (truncation)
ROUND_FLOOR	round towards -infinity
ROUND_HALF_DOWN	round to nearest; if equidistant, round down
ROUND_HALF_EVEN	round to nearest; if equidistant, round final digit even (IEEE default)
ROUND_HALF_UP	round to nearest; if equidistant, round up
ROUND_UP	round away from 0

Restrictions for DECFLOAT

- DECFLOAT (or a distinct type based on DECFLOAT) can not be used for:
  - PRIMARY KEY
  - UNIQUE key
  - A foreign key or parent key
  - An IDENTITY column
  - A column in the partitioning key (PARTITION BY RANGE)
  - A column used for index on expression
  - Have a FIELDPROC

SWG BetaWorks

DECFLOAT

Reference material

DB2 9 for z/OS Technical
Education Series

DECFLOAT host variables in Java applications

SQL data type	**Recommended Java data type or Java object type**
DECFLOAT(n), n = 16 or 34	java.math.BigDecimal
Other supported Java data types: long, int, short, byte, double, boolean, java.lang.String	

Notes

- Java run-time JDK 1.5 required
- DSNTEP2, DSNTEP4, and DSNTIAUL are modified to support the DECFLOAT data type
- SYSIBM.SYSENVIRONMENT - records the environment variables when an object is created.
  - ROUNDING CHAR(1) NOT NULL The ROUNDING MODE used when doing arithmetic and casting operation on DECFLOAT data (see ROUNDING bind option and CURRENT DECIMAL FLOATING POINT ROUNDING MODE special register for more detail):

C	ROUND_CEILING
D	ROUND_DOWN
F	ROUND_FLOOR
G	ROUND_HALF_DOWN
E	ROUND_HALF_EVEN
H	ROUND_HALF_UP
U	ROUND_UP

Notes . . . con't

- DSNHDECP DEF_DECFLOAT_ROUND_MODE
  - Acceptable values ROUND_CEILING, ROUND_DOWN, ROUND_FLOOR, ROUND_HALF_DOWN, ROUND_HALF_EVEN, ROUND_HALF_UP, and ROUND_UP Default ROUND_HALF_EVEN
  - Update Option 15 on panel DSNTIPB

- SQLCODE -574 Explanation text include a new bullet for the reasons:
  - A decimal floating point (DECFLOAT) constant is specified. decimal floating point cannot be restricted to zero scale numbers, and as such may not be used in IDENTITY columns.

Notes . . . con’t

- UNLOAD utility specification
  - DECFLOAT EXTERNAL (*length)* Specifies the total field length in bytes, including:
    - the first sign character
    - the decimal point
    - the E character
    - the second sign character (for the exponent)
    - and the exponent if they are in the string.
  - If the number of characters in the result is less than the specified or the default length, the result is padded to the right with blanks.
  - The default output field size is 23 if the source data type is the DECFLOAT(16); otherwise, the default is 42.

SWG BetaWorks

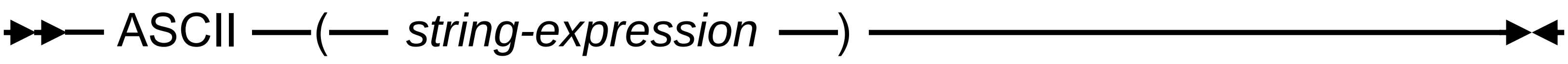
Built-in functions

DB2 9 for z/OS Technical
Education Series

New built-in functions

- ASCII
- TIMESTAMPDIFF
- SOUNDEX
- DIFFERENCE
- TIMESTAMP_ISO
- EXTRACT
- MONTHS_BETWEEN
- VARCHAR_FORMAT
- TIMESTAMP_FORMAT

ASCII function



ASCII (*string-expression*)

- Schema is SYSIBM
- Returns the ASCII code value of the leftmost character of the argument
- ***string-expression*** can be any character string type, except for CLOB or DBCLOB
- If ***string-expression*** is EBCDIC, Unicode, or graphic, first converted to SBCS ASCII (CCSID 367)
- Result is INTEGER (or null, if argument is null)

SET :hv = ASCII (‘A’); result is integer value 65 in :hv

TIMESTAMPDIFF function

```
▸▸—TIMESTAMPDIFF -(- numeric-expression - ,- string-expression —— ) ———▸◂
```

- Schema is SYSIBM
- ***numeric-expression*** defines the type of time interval
- ***string-expression*** equivalent of subtracting two timestamps and converting the result to a string of length 22

- Returns an integer which represents an *estimated number of intervals* of the type defined by ***numeric-expression***

Specifying the type of time interval

numeric-expression (INTEGER or SMALLINT)	**Value**
1	Microseconds
2	Seconds
4	Minutes
8	Hours
16	Days
32	Weeks
64	Months
128	Quarters
256	Years

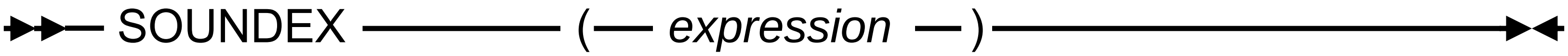
Specifying the timestamp difference

- Must be an expression that returns a value of a built-in character string or graphic string -- no CLOB or DBCLOB; if null, the result is null
- Assumptions used for estimating the number of intervals:
  - 365 days per year
  - 52 weeks per year
  - 12 months per year
  - 30 days in a month
  - 7 days in a week
  - 24 hours in a day
  - 60 minutes in an hour
  - 60 seconds in a minute
- Example:
  - interval 16 requested (number of days)
  - difference in timestamps: ‘1997-03-01-00.00.00’ and ‘1997-02-01-00.00.00’
  - result is 30 -- 1 month, so assume 30 days per month

Example

```
SELECT
  TIMESTAMPDIFF
        (  64,
        CAST (CURRENT_TIMESTAMP  -  CAST (BIRTHDATE AS TIMESTAMP)
                AS CHAR(22)
                )
        )
        AS AGE_IN_MONTHS
  FROM EMPLOYEE
```

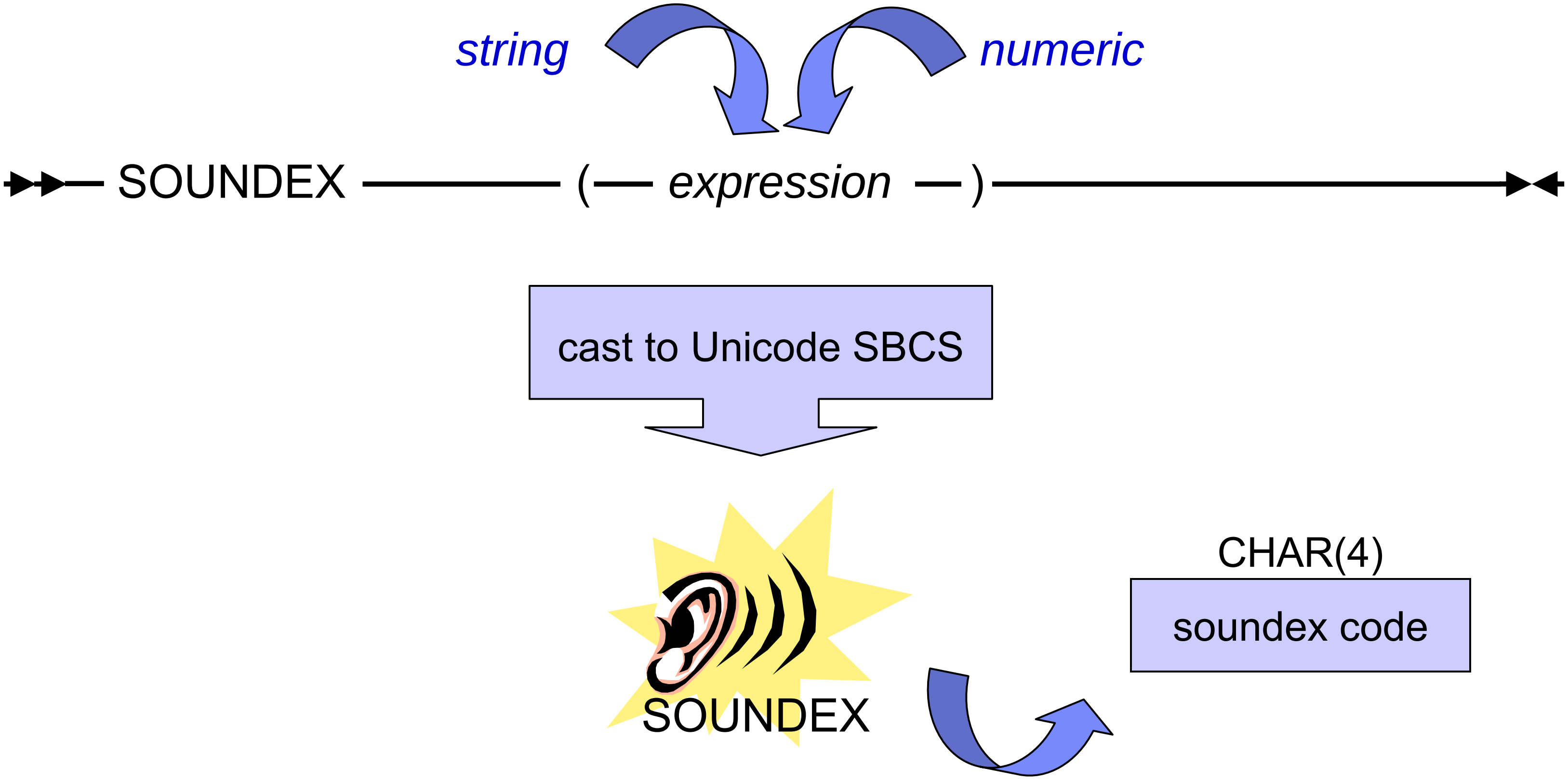
SOUNDEX function



SOUNDEX (*expression*)

- Schema is SYSIBM
- ***expression*** an expression which returns any built-in numeric or string data type other than CLOB or DBCLOB
- Returns a four-character code representing the sound of the words in the argument. The result can be compared with the sound of other strings
- Useful for finding strings in which the sound is known but precise spelling is not -- makes assumptions about sounds of letters and combinations of letters

Result of SOUNDEX



Example

```
SELECT EMPNO,LASTNAME
  FROM DSN910.EMPLOYEE
  WHERE SOUNDEX (LASTNAME) = SOUNDEX (“Loucesy”)
```

Returns the row:

```
        000110  LUCCHESSI;
```

DIFFERENCE function

```
▶▶—DIFFERENCE—— (expression-1 —— , ——expression-2 ——) ————————▶◀
```

- Schema is SYSIBM
- ***expression-1*** or ***expression-2*** an expression which returns any built-in numeric, graphic or string data type other than CLOB or DBCLOB

- Returns an integer 0 to 4 which represents the difference between the sounds of the two strings based on their SOUNDEX values; null if any argument is null.
- Value of 4 represents best possible match

Example 1

- Find the DIFFERENCE and SOUNDEX values for ‘CONSTRAINT’ and ‘CONSTANT’

```
SELECT DIFFERENCE (‘CONSTRAINT’, ‘CONSTANT’),
                SOUNDEX (‘CONSTRAINT’),
                SOUNDEX (‘CONSTANT’)
       FROM SYSIBM.SYSDUMMY1;
```

```
Returns:        4         C523    C523
```

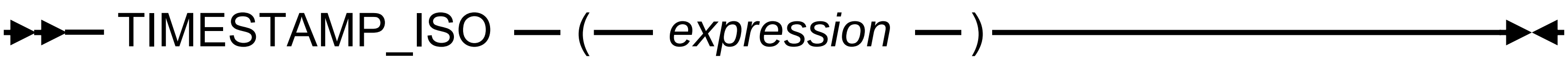
Example 2

- Find the DIFFERENCE and SOUNDEX values for ‘CONSTRAINT’ and ‘CONTRITE’

```
SELECT DIFFERENCE (‘CONSTRAINT’, ‘CONTRITE’),
                  SOUNDEX (‘CONSTRAINT’),
                  SOUNDEX (‘CONTRITE’)
        FROM SYSIBM.SYSDUMMY1;
```

```
Returns:          2          C523     C536
```

TIMESTAMP_ISO function



```
►►— TIMESTAMP_ISO — (— expression — )————————————————►◄
```

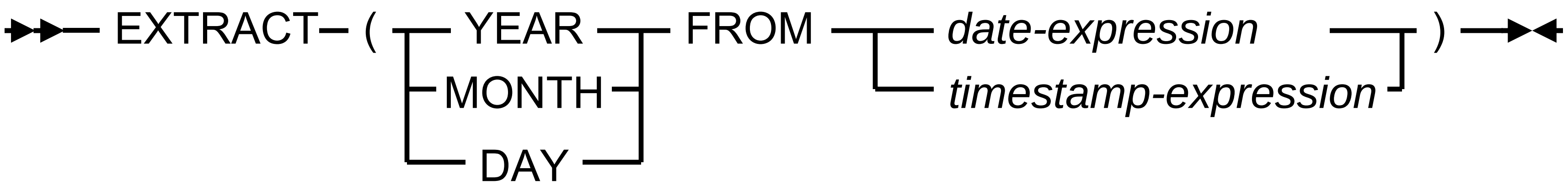
- Schema is SYSIBM
- ***expression*** an expression which returns any built-in timestamp, date, time, string, or graphic data type other than CLOB or DBCLOB. Value must be a valid date, time, or timestamp value.
- Results:
  - if ***expression*** is a date CURRENT DATE + zeros for time and microseconds
  - if ***expression*** is a time CURRENT DATE + zeros for microseconds
  - if ***expression*** is null null

Example

```
SELECT TIMESTAMP_ISO (  DATE (  ‘1965-07-27’  )  )
          FROM SYSIBM.SYSDUMMY1;
```

```
Returns:          ‘1965-07-27-00.00.00.000000’
```

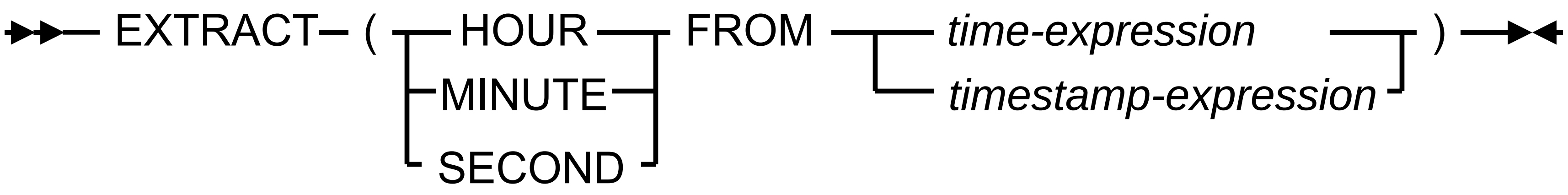
EXTRACT date values function



EXTRACT (YEAR / MONTH / DAY FROM *date-expression* / *timestamp-expression*)

- Schema is SYSIBM
- ***date-expression*** an expression which returns either a built-in date or a built-in character string data type other than CLOB or DBCLOB. Value must be a valid character-string representation of a date.
- ***timestamp-expression*** an expression which returns either a built-in timestamp or a built-in character string data type other than CLOB or DBCLOB. Value must be a valid character-string representation of a timestamp.

EXTRACT time values function



EXTRACT (HOUR / MINUTE / SECOND FROM *time-expression* / *timestamp-expression*)

- Schema is SYSIBM
- ***time-expression*** an expression which returns either a built-in time or a built-in character string data type other than CLOB or DBCLOB. Value must be a valid character-string representation of a time.
- ***timestamp-expression*** an expression which returns either a built-in timestamp or a built-in character string data type other than CLOB or DBCLOB. Value must be a valid character-string representation of a timestamp.

MONTHS_BETWEEN function

▸▸—MONTHS_BETWEEN —— (- *expression-1* —, — *expression-2* —) —▸◂

- Schema is SYSIBM
- ***expression-1 and expression-2*** expression2 which returns any of the following built-in data types: date, timestamp, character string, or graphic data type other than CLOB or DBCLOB. Value must be a valid date or timestamp value.
- Results:
  - Result is calculated based on a 31 day month, representing the difference between ***expression-1*** and ***expression-2*** with a fractional number
  - if ***expression-1 > expression-2*** result is positive
  - if ***expression-1 < expression-2*** result is negative
  - if ***expression-1*** and ***expression-2*** are within the same day of the month, or last day of the month, result is a whole number

Example

```
SELECT MONTHS_BETWEEN (  ’02-20-2005’ , ’01-17-2005’  )
        AS MONTHS_BETWEEN
        FROM SYSIBM.SYSDUMMY1;
```

Returns:

```
MONTHS_BETWEEN
---------------------
1.064516129032258
```

VARCHAR_FORMAT function

▶▶— VARCHAR_FORMAT *(expression, format string)* ————————▶◀

- Schema is SYSIBM
- ***expression*** an expression which returns any built-in timestamp data type
- ***format-string*** a character string which contains a template of how ***expression*** should be formatted.
- Results: returns a character representation of a timestamp in the format indicated by ***format-string***.

format-string values

format-string	**Value**
CC	First 2 digits of year
D	Day of week
DD	Day of month
DDD	Day of year
FF[n]	Fractional seconds; n (1 - 6)is the number of digits to be returned
HH24	Hour of day (00 – 24; when 24, minutes and seconds must be 0)
ID	ISO day of week (1 – 7)
IW	ISO week of year (1 – 53)
Etc.	. . .

Example

- Set the character variable TVAR to the timestamp value of CREATEDTS from SYSIBM.SYSDATABASE, using the character string format supported by the function to specify the format of the value for TVAR.

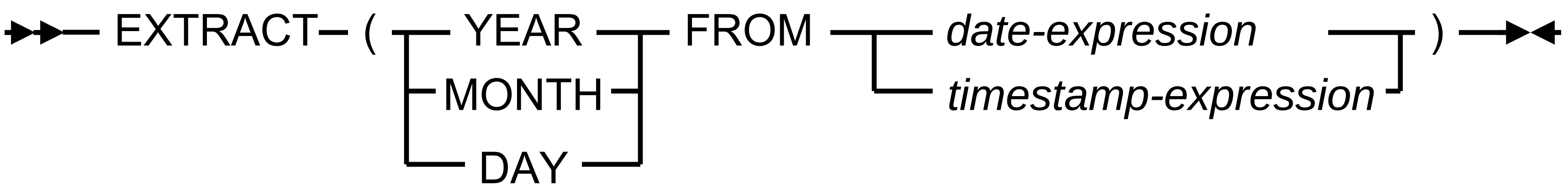
```
SELECT VARCHAR_FORMAT(CREATEDTS,’YYYY-MM-DD HH24:MI:SS’)
   INTO :TVAR
   FROM SYSIBM.SYSDATABASE;
```

TIMESTAMP_FORMAT function

▶▶— TIMESTAMP_FORMAT *(string-expression, format string)* ————▶◀

- Schema is SYSIBM
- ***string-expression*** an expression which returns any built-in character string or graphic string data type other than CLOB or DBCLOB.
- ***format-string*** possible formats*:*
  - ‘YYYY-MM-DD’
  - ‘YYYY-MM-DD-HH24-MI-SS’
  - ‘YYYY-MM-DD-HH24-MI-SS-NNNNNN’
  - Separators, in any combination: “-” “.” “/” “:” “ “
- Results: leading and trailing blanks are stripped, and the substring is formatted for return as a character representation of a timestamp in the format indicated by ***format-string***.

EXTRACT date values function



EXTRACT (YEAR MONTH DAY FROM date-expression timestamp-expression)

- Schema is SYSIBM
- ***expression*** an expression which returns any built-in timestamp, date, time, string, or graphic data type other than CLOB or DBCLOB. Value must be a valid date, time, or timestamp value.
- Results:

if ***expression*** is a date	CURRENT DATE + zeros for time and microseconds
if ***expression*** is a time	CURRENT DATE + zeros for microseconds
if ***expression*** is null	null

Impact on existing applications

- These functions result in an incompatible change for a customer who has created UDFs with these names AND has SYSIBM first in their PATH:
  - ASCII
  - TIMESTAMP
  - SOUNDEX
  - DIFFERENCE
  - TIMESTAMP_ISO
  - EXTRACT
  - MONTHS_BETWEEN
  - VARCHAR_FORMAT
  - TIMESTAMP_FORMAT

Thank You

Any Questions ?

ON DEMAND BUSINESS™