

DB2 UDB V7.2

SQL Cookbook

Graeme Birchall

11-Mar-2002

Introduction

These overhead slides can be used in conjunction my SQL Cookbook to teach basic DB2 SQL to a small audience. Based on my own teaching experiences, the class should take about two days to complete, but it might be better to do it over three days. There are no exercises.

As much as possible, I have tried to make every slide self-contained. Thus, it should have some sample SQL, the input data, and the answer. And with very few exceptions, I've laid out the slides in the same sequence as the examples in the related SQL Cookbook.

Disclaimer & Copyright

DISCLAIMER: What can I say, there are bound to be errors, use at your own risk.

COPYRIGHT: This document may not be copied in whole, or in part, without the explicit written permission of the distributor. Distribution does not imply publication. This copy was printed on 11 March, 2002. It is intended for personal use only. Secondary distribution for gain is not allowed. File name: CLASS12.DOC.

TRADEMARKS: AIX, DB2, and DB2 MVS, are registered trademarks of the IBM Corporation. Windows, Windows/NT, Windows/2000, Windows/ME, and Word for Windows are registered trademarks of the Microsoft Corporation. Adobe Acrobat is a registered trademark of the Adobe Corporation.

Author

Author: Graeme Birchall ©
Address: 1 River Court, Apt 1706
Jersey City NJ 07310-2007
Ph/Fax: (201)-963-0071
Email: Graeme_Birchall@compuserve.com
Web: http://ourworld.compuserve.com/homepages/Graeme_Birchall

Document History

Course Editions

This document is a work in progress. So far, it has been updated as follows:

- | | |
|-------------|---|
| 2000-12-04: | Initial edition of course notes uploaded. The following chapters were included: Joins, Sub-queries, Unions, Recursive SQL, Fun with SQL, and Quirks in SQL. |
| 2001-01-03: | All of the above chapters were updated, and two new chapters were added: Introduction to SQL, and Column Functions. |
| 2001-02-06: | Minor changes to include latest fixes to Cookbook. Also added new chapter on Scalar Functions (selected). |
| 2001-04-11: | Replace chapter on sub-queries. |
| 2001-11-21: | Update for DB2 V7.2. Also added chapter on Identity Columns. |
| 2001-12-07: | Added chapter on OLAP Functions and GROUP BY. |
| 2002-03-11: | Minor changes, mostly to section on precedence rules.. |

Please Share

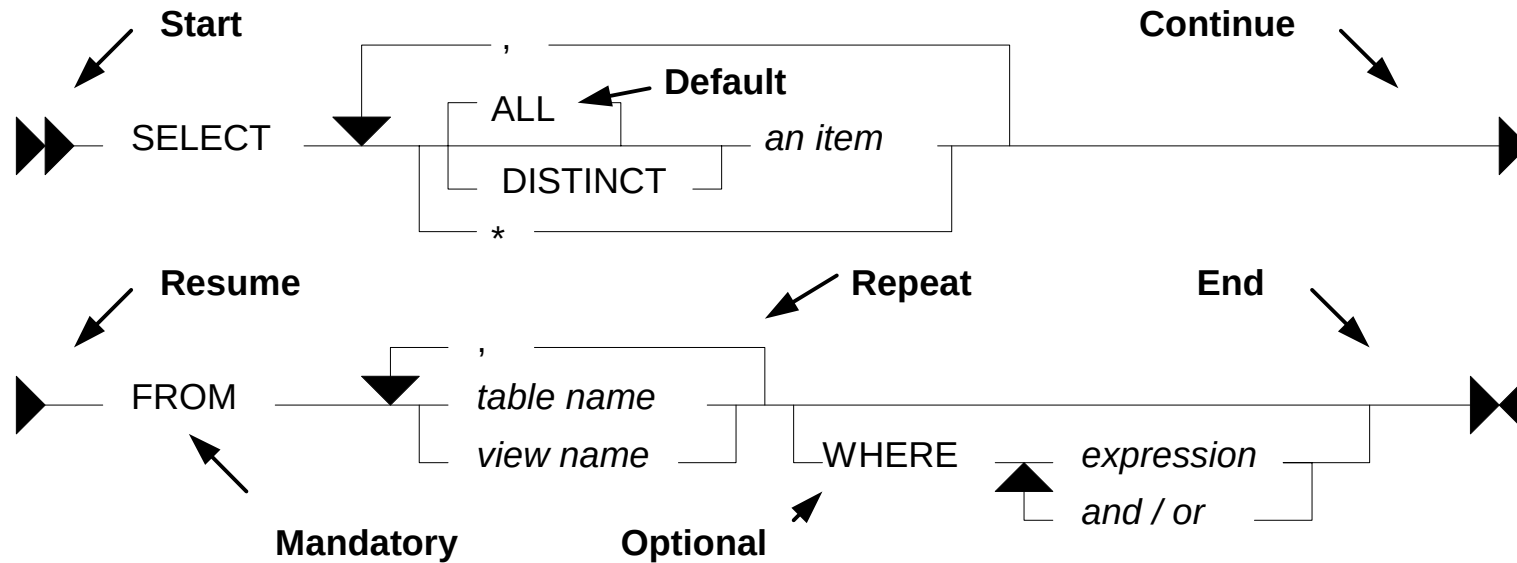
The more people that use this material, the more inclined I am to do more work keeping it up to date. If you like what you have, please tell other people about it.

Table of Contents

Introduction to SQL.....	5
Column Functions	57
OLAP Functions	74
Scalar Functions.....	99
Order By, Group By, and Having.....	119
Joins.....	133
Sub-Query.....	171
Union, Intersect, and Except.....	201
Identity Column	211
Recursive SQL.....	220
Fun with SQL	246
Quirks in SQL	268

Introduction to SQL

Syntax Diagram Conventions



Notes

- Upper case text is a SQL keyword
- Italic text is a placeholder, or explained elsewhere
- Backward arrows enable one to repeat phrases
- Branch line going above main line is default
- Branch line going below main line is optional

Table Definitions

```

CREATE TABLE EMPLOYEE
(EMPNO      CHARACTER (00006)      NOT NULL
, FIRSTNAME VARCHAR   (00012)      NOT NULL
, MIDINIT   CHARACTER (00001)      NOT NULL
, LASTNAME  VARCHAR   (00015)      NOT NULL
, WORKDEPT  CHARACTER (00003)
, PHONENO   CHARACTER (00004)
, HIREDATE  DATE
, JOB       CHARACTER (00008)
, EDLEVEL   SMALLINT              NOT NULL
, SEX       CHARACTER (00001)
, BIRTHDATE DATE
, SALARY    DECIMAL   (00009,02)
, BONUS     DECIMAL   (00009,02)
, COMM      DECIMAL   (00009,02)
)
DATA CAPTURE NONE;

```

DB2 Sample Tables

- Supplied by IBM
- See **SAMPLE Database**
- Used throughout this course

View Definitions

Queries Data Table

```
CREATE VIEW EMPLOYEE_VIEW AS
SELECT  A.EMPNO, A.FIRSTNME, A.SALARY, A.WORKDEPT
FROM    EMPLOYEE A
WHERE   A.SALARY >=
        (SELECT AVG(B.SALARY)
         FROM    EMPLOYEE B
         WHERE   A.WORKDEPT = B.WORKDEPT);
```

Makes 3 rows - Using VALUES Clause

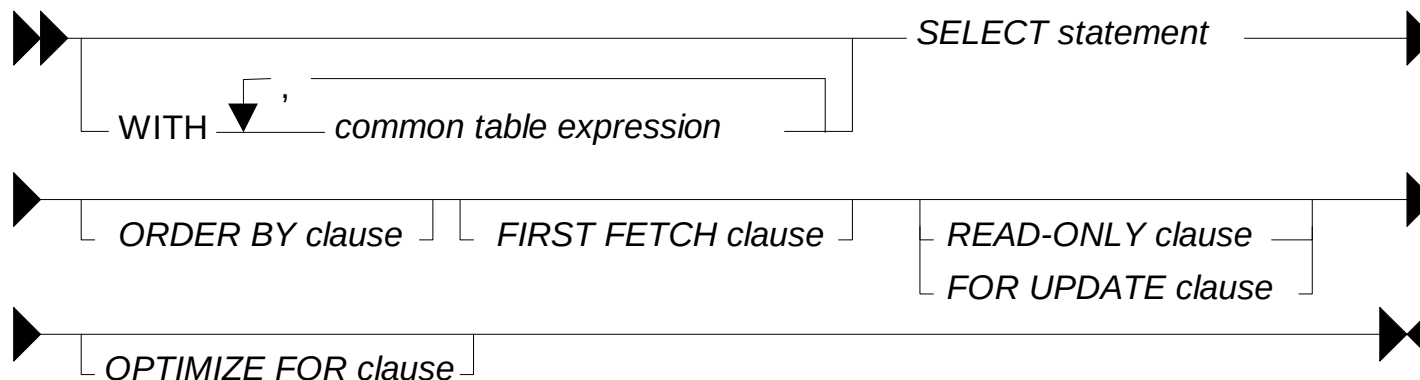
```
CREATE VIEW SILLY (C1, C2, C3)
AS VALUES (11, 'AAA', SMALLINT(22))
          , (12, 'BBB', SMALLINT(33))
          , (13, 'CCC', NULL);
```

Makes 10,000 rows - Using Recursion

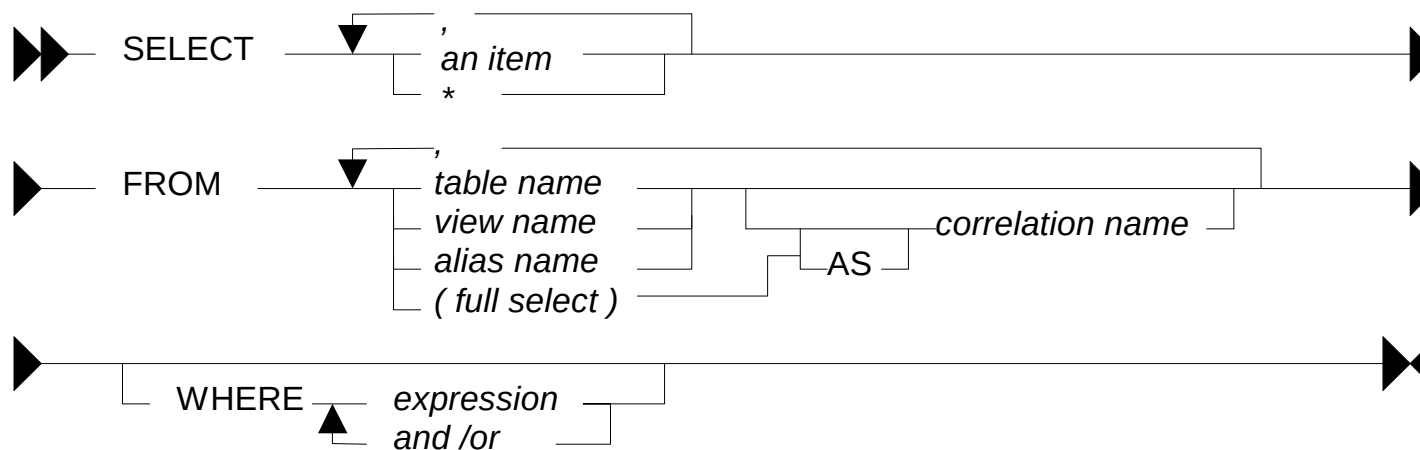
```
CREATE VIEW TEST_DATA AS
WITH TEMP1 (NUM1) AS
(VVALUES  (1)
 UNION ALL
 SELECT  NUM1 + 1
 FROM    TEMP1
 WHERE   NUM1 < 10000)
SELECT *
FROM    TEMP1;
```

SELECT Statement Syntax

General



SELECT Statement



Sample SELECT Statements (1 of 2)

Select Specific Columns

```
SELECT    DEPTNO
          , ADMRDEPT
          , 'ABC' AS ABC
FROM      DEPARTMENT
WHERE     DEPTNAME LIKE '%ING%'
ORDER BY 1;
```

```
ANSWER
=====
DEPTNO  ADMRDEPT  ABC
-----
B01     A00       ABC
D11     D01       ABC
```

Select all Columns

```
SELECT    *
FROM      DEPARTMENT
WHERE     DEPTNAME LIKE '%ING%'
ORDER BY 1;
```

```
ANSWER (part of)
=====
DEPTNO etc...
----->>>
B01     PLANNING
D11     MANUFACTU
```

Notes

- **SELECT "*" implies select all columns**

Sample SELECT Statements (2 of 2)

Select an Individual Column, and all Columns

```
SELECT  DEPTNO
        ,DEPARTMENT.*
FROM    DEPARTMENT
WHERE   DEPTNAME LIKE '%ING%'
ORDER BY 1;
```

ANSWER (part of)

```
=====
DEPTNO DEPTNO etc...
----- >>>
B01    B01    PLANNING
D11    D11    MANUFACTU
```

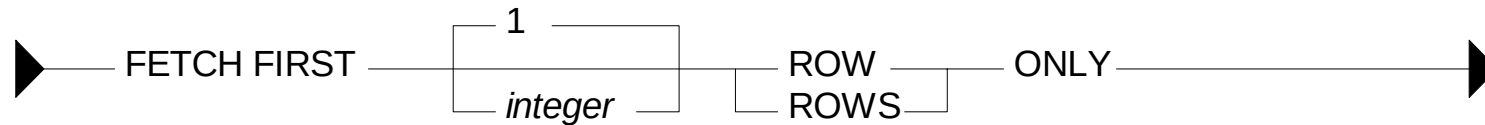
Select all Columns Twice

```
SELECT  DEPARTMENT.*
        ,DEPARTMENT.*
FROM    DEPARTMENT
WHERE   DEPTNAME LIKE '%NING%'
ORDER BY 1;
```

ANSWER (part of)

```
=====
DEPTNO etc...
----- >>>
B01    PLANNING
```

Fetch First Clause



Sample SQL, No ORDER BY, Gets Random Rows

```

SELECT  YEARS
        , NAME
        , ID
FROM    STAFF
FETCH FIRST 3 ROWS ONLY;
  
```

ANSWER

```

=====
YEARS  NAME      ID
-----
      7 Sanders   10
      8 Pernal    20
      5 Marenghi  30
  
```

Notes

- Stops fetching after "n" rows
- If no ORDER BY, gets first "n" random rows
- If ORDER BY not on unique field, also random results
- See other pages for more on this subject

FIRST FETCH and Unique Fields

ORDER BY not on unique field(s), Random rows returned

```
SELECT  YEARS
        , NAME
        , ID
FROM    STAFF
WHERE   YEARS IS NOT NULL
ORDER BY YEARS DESC
FETCH FIRST 3 ROWS ONLY;
```

```
ANSWER
=====
YEARS  NAME      ID
-----
      13 Graham   310
      12 Jones    260
      10 Hanes    50
```

ORDER BY on unique field(s)

```
SELECT  YEARS
        , NAME
        , ID
FROM    STAFF
WHERE   YEARS IS NOT NULL
ORDER BY YEARS DESC
        , ID      DESC
FETCH FIRST 3 ROWS ONLY;
```

```
ANSWER
=====
YEARS  NAME      ID
-----
      13 Graham   310
      12 Jones    260
      10 Quill    290
```

Notes

- There is no unique index on YEARS field

Correlation Name

In Nested Table Expression

```
SELECT  A.EMPNO
        ,A.LASTNAME
FROM    EMPLOYEE A
        , (SELECT MAX(EMPNO)AS EMPNO
          FROM    EMPLOYEE) AS B
WHERE   A.EMPNO = B.EMPNO;
```

```
ANSWER
=====
EMPNO  LASTNAME
-----
000340 GOUNOT
```

In Join

```
SELECT  A.EMPNO
        ,A.LASTNAME
        ,B.DEPTNO AS DEPT
FROM    EMPLOYEE A
        ,DEPARTMENT B
WHERE   A.WORKDEPT = B.DEPTNO
        AND A.JOB    <> 'SALESREP'
        AND B.DEPTNAME = 'OPERATIONS'
        AND A.SEX     IN ('M', 'F')
        AND B.LOCATION IS NULL
ORDER BY 1;
```

```
ANSWER
=====
EMPNO  LASTNAME  DEPT
-----
000090 HENDERSON  E11
000280 SCHNEIDER  E11
000290 PARKER    E11
000300 SMITH      E11
000310 SETRIGHT   E11
```

Notes

- If same table is accessed twice, correlation names are essential

Renaming Fields Using AS

SELECT Statement

```
SELECT    EMPNO      AS  E_NUM
          ,MIDINIT    AS "M INT"
          ,PHONENO    AS "... "
FROM      EMPLOYEE
WHERE     EMPNO < '000030'
ORDER BY 1;
```

```
ANSWER
=====
E_NUM    M INT    ...
-----
000010    I        3978
000020    L        3476
```

VIEW Definition

```
CREATE VIEW EMP2 AS
SELECT EMPNO      AS  E_NUM
          ,MIDINIT AS "M INT"
          ,PHONENO AS "... "
FROM      EMPLOYEE;

SELECT *
FROM      EMP2
WHERE     "... " = '3978';
```

```
ANSWER
=====
E_NUM    M INT    ...
-----
000010    I        3978
```

Notes

- Renamed fields can be referred to in ORDER BY, and outside of full-select
- Renamed fields cannot be referred to in GROUP BY, WHERE, HAVING

Null Values

AVG of Field Containing Null Values

```
SELECT    AVG(COMM)                AS A1
          , SUM(COMM) / COUNT(*) AS A2
FROM      STAFF
WHERE     ID < 100;
```

```
ANSWER
=====
A1      A2
-----
796.025 530.68
```

AVG of Field - Excluding Null Rows

```
SELECT    AVG(COMM)                AS A1
          , SUM(COMM) / COUNT(*) AS A2
FROM      STAFF
WHERE     ID < 100
        AND COMM IS NOT NULL;
```

```
ANSWER
=====
A1      A2
-----
796.025 796.02
```

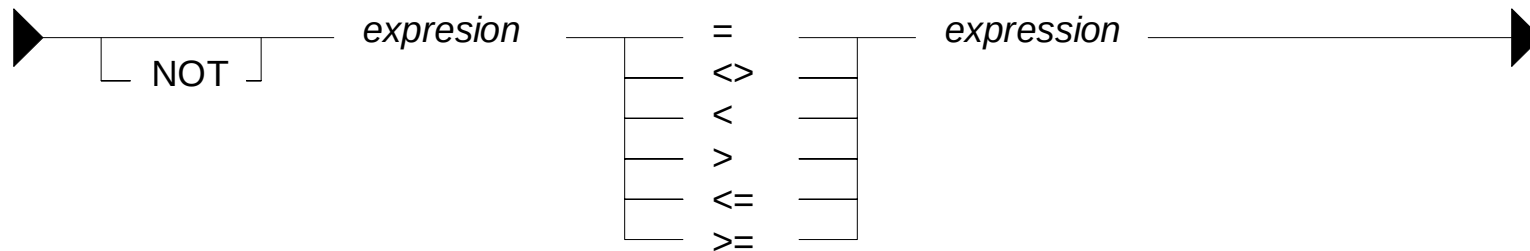
Getting NULL result from field defined NOT NULL

```
SELECT    COUNT(*)                AS NUM
          , MAX(LASTNAME) AS MAX
FROM      EMPLOYEE
WHERE     FIRSTNAME = 'FRED';
```

```
ANSWER
=====
NUM  MAX
---  ---
0    -
```

Basic Predicate

Syntax



Sample SQL

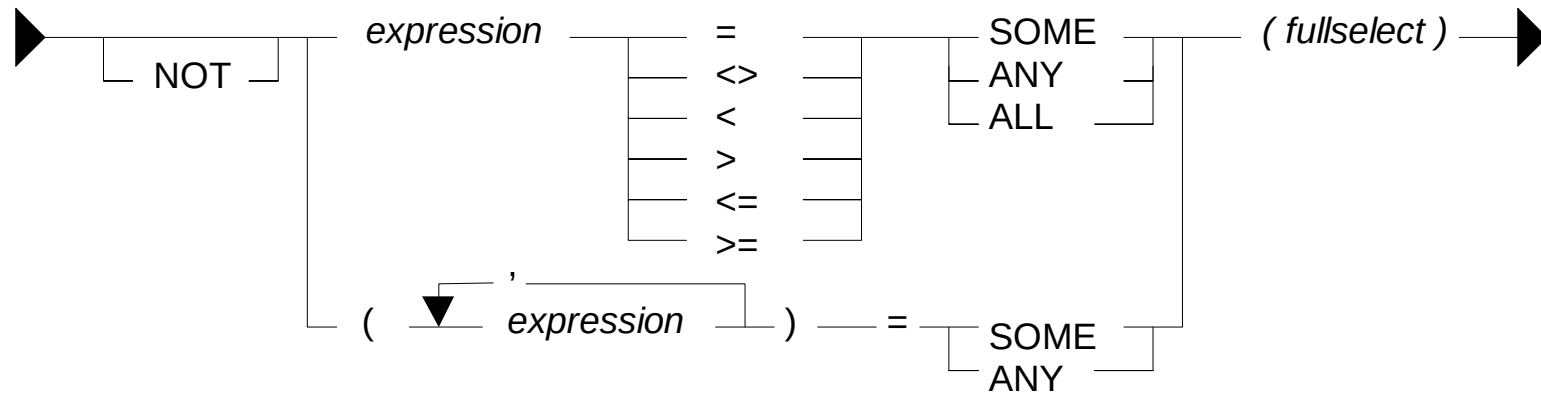
```
SELECT      ID, JOB, DEPT
FROM        STAFF
WHERE       JOB   =  'Mgr'
           AND NOT JOB <> 'Mgr'
           AND NOT JOB   =  'Sales'
           AND     ID   <> 100
           AND     ID   >=  0
           AND     ID   <= 150
           AND NOT DEPT =   50
ORDER BY    ID;
```

ANSWER

```
=====
ID      JOB      DEPT
---      - - - -
  10    Mgr       20
  30    Mgr       38
  50    Mgr       15
 140    Mgr       51
```

Quantified Predicate (1 of 3)

Syntax



Notes

- **Basic Predicate:** Compares two values
- **Quantified Predicate:** Compares "n" values with a collection of values (i.e. uses sub-query)

Quantified Predicate (2 of 3)

Two Single-value Checks

```
SELECT  ID, JOB
FROM    STAFF
WHERE   JOB  = ANY (SELECT JOB FROM STAFF)
      AND  ID  <= ALL (SELECT ID  FROM STAFF)
ORDER BY ID;
```

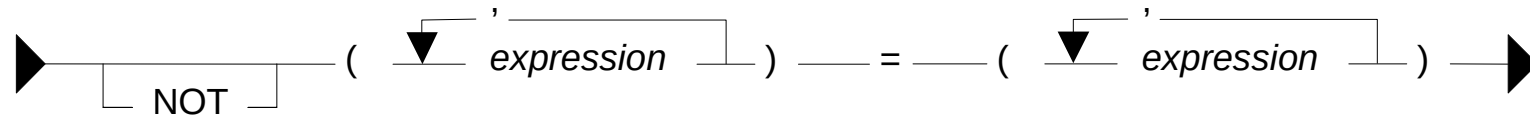
```
ANSWER
=====
ID  JOB
--- ---
 10 Mgr
```

Multi-value Check

```
SELECT  ID, DEPT, JOB
FROM    STAFF
WHERE   (ID,DEPT) = ANY
        (SELECT DEPT, ID
         FROM    STAFF)
ORDER BY 1;
```

```
ANSWER
=====
ID  DEPT  JOB
--- ---- -
 20   20 Sales
```

Quantified Predicate (3 of 3)



Multi-value Check, 1 of 2

```

SELECT  ID, DEPT, JOB
FROM    STAFF
WHERE   (ID, DEPT) = (30, 28)
        OR (ID, YEARS) = (90, 7)
        OR (DEPT, JOB) = (38, 'Mgr ')
ORDER BY 1;
  
```

```

ANSWER
=====
ID DEPT JOB
-- ---- ---
30   38 Mgr
  
```

Multi-value Check, 2 of 2

```

SELECT  ID, DEPT, JOB
FROM    STAFF
WHERE   (ID = 30 AND DEPT = 28)
        OR (ID = 90 AND YEARS = 7)
        OR (DEPT = 38 AND JOB = 'Mgr ')
ORDER BY 1;
  
```

```

ANSWER
=====
ID DEPT JOB
-- ---- ---
30   38 Mgr
  
```

BETWEEN Predicate

Syntax



Sample SQL

```
SELECT ID, JOB
FROM STAFF
WHERE ID BETWEEN 10 AND 30
      AND ID NOT BETWEEN 30 AND 10
      AND NOT ID NOT BETWEEN 10 AND 30
ORDER BY ID;
```

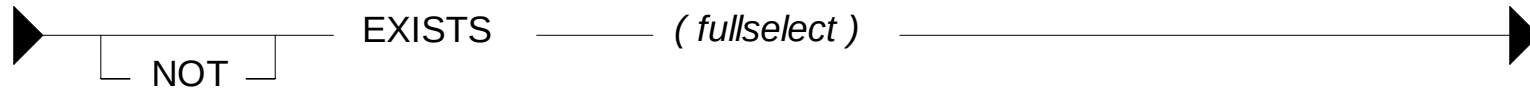
```
ANSWER
=====
ID    JOB
---  -
 10  Mgr
 20  Sales
 30  Mgr
```

Notes

- Always check between LOW and HIGH

EXISTS Predicate

Syntax



- **Sample SQL**

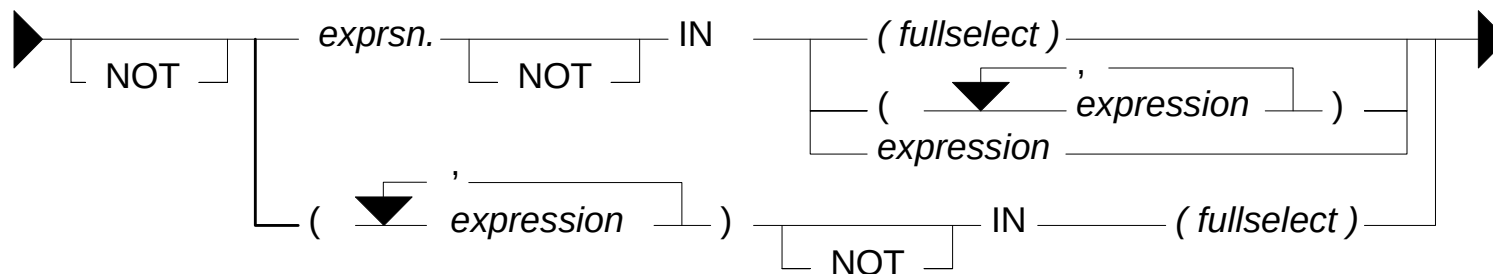
```
SELECT ID, JOB
FROM STAFF A
WHERE EXISTS
      (SELECT *
        FROM STAFF B
        WHERE B.ID = A.ID
              AND B.ID < 50)
ORDER BY ID;
```

```
ANSWER
=====
ID    JOB
----  -
 10  Mgr
 20  Sales
 30  Mgr
 40  Sales
```

Notes

- **See Sub-Query chapter for more details**

IN Predicate



Single Value

```

SELECT ID, JOB
FROM   STAFF A
WHERE  ID IN (10,20,30)
      AND ID IN (SELECT ID
                  FROM   STAFF)
      AND ID NOT IN 99
ORDER BY ID;
  
```

```

ANSWER
=====
ID  JOB
--- ---
10  Mgr
20  Sales
30  Mgr
  
```

Multiple Values

```

SELECT  EMPNO, LASTNAME
FROM    EMPLOYEE
WHERE   (EMPNO, 'AD3113') IN
        (SELECT EMPNO, PROJNO
         FROM    EMP_ACT
         WHERE   EMPTIME > 0.5)
ORDER BY 1;
  
```

```

ANSWER
=====
EMPNO  LASTNAME
-----
000260 JOHNSON
000270 PEREZ
  
```

LIKE Predicate (1 of 2)

Syntax



LIKE Usage

```

SELECT ID, NAME
FROM   STAFF
WHERE  NAME LIKE 'S%n'
      OR NAME LIKE '_a_a%'
      OR NAME LIKE '%r_%a'
ORDER BY ID;

```

```

ANSWER
=====
ID    NAME
---  -
130   Yamaguchi
200   Scoutten

```

LIKE and ESCAPE Usage

```

SELECT ID
FROM   STAFF
WHERE  ID = 10
      AND 'ABC' LIKE 'AB%'
      AND 'A%C' LIKE 'A/%C' ESCAPE '/'
      AND 'A_C' LIKE 'A\C_C' ESCAPE '\'
      AND 'A_$' LIKE 'A$__$' ESCAPE '$';

```

```

ANSWER
=====
      ID
      --
      10

```


NULL Predicate



Sample SQL

```
SELECT    ID, COMM
FROM      STAFF
WHERE     ID    < 100
        AND    ID    IS NOT NULL
        AND    COMM IS      NULL
        AND NOT COMM IS NOT NULL
ORDER BY  ID;
```

```
ANSWER
=====
ID    COMM
---  -
10    -
30    -
50    -
```

Notes

- Can't use "=" to check for nulls
- Use COALESCE function to convert null values to something else

Precedence Rules (1 of 2)

Basic Arithmetic

Example:

555	+	-22	/	(12 - 3)	*	66	ANSWER
^		^	^	^	^		=====
5th		2nd	3rd	1st	4th		423

Integer Example

```

SELECT          (12      - 3)          AS INT1          =>          9
                ,      -22 / (12      - 3)      AS INT2          =>          -2
                ,      -22 / (12      - 3) * 66  AS INT3          =>        -132
                , 555 + -22 / (12      - 3) * 66  AS INT4          =>        432
FROM            SYSIBM.SYSDUMMY1;

```

Decimal Example

```

SELECT          (12.0 - 3)          AS DEC1          =>          9.0
,              -22 / (12.0 - 3)      AS DEC2          =>         -2.4
,              -22 / (12.0 - 3) * 66 AS DEC3          =>       -161.3
, 555 + -22 / (12.0 - 3) * 66 AS DEC4          =>       393.6
FROM            SYSIBM.SYSDUMMY1;

```

Notes

- MULTIPLICATION and DIVISION before ADDITION and SUBTRACTION
- INTEGER arithmetic truncates fractional results

Precedence Rules (2 of 2)

And vs. OR

```
SELECT *
FROM   TABLE1
WHERE  COL1 = 'C'
      AND COL1 >= 'A'
      OR  COL2 >= 'AA';
```

ANSWER>>

COL1	COL2
A	AA
B	BB
C	CC

TABLE1	
COL1	COL2
A	AA
B	BB
C	CC

```
SELECT *
FROM   TABLE1
WHERE  COL1 = 'C'
      AND (COL1 >= 'A'
      OR  COL2 >= 'AA');
```

ANSWER>>

COL1	COL2
C	CC

Notes

- AND operations done before OR operations
- Parenthesis can be used to alter precedence

Temporary Tables (in Statement)

Types

- Use WITH to define Common Table Expression
- Define FULL-SELECT in FROM part of query
- Define FULL-SELECT in SELECT part of query

Temporary Tables - Identical Query (1 of 3)

Using Common Table Expression

```

WITH STAFF_DEPT AS
( SELECT      DEPT          AS DEPT#
  , MAX(SALARY) AS MAX_SAL
  FROM        STAFF
  WHERE       DEPT < 50
  GROUP BY   DEPT
)
SELECT      ID
  , DEPT
  , SALARY
  , MAX_SAL
FROM        STAFF
LEFT OUTER JOIN
  STAFF_DEPT
ON          DEPT      = DEPT#
WHERE       NAME LIKE 'S%'
ORDER BY   ID;

```

ANSWER

```

=====
ID  DEPT  SALARY  MAX_SAL
---  ---  -
10   20  18357.50  18357.50
190  20  14252.75  18357.50
200  42  11508.60  18352.80
220  51  17654.50   -

```

Temporary Tables - Identical Query (2 of 3)

Using full-select in FROM

```

SELECT      ID
            ,DEPT
            ,SALARY
            ,MAX_SAL
FROM        STAFF
LEFT OUTER JOIN
            (SELECT      DEPT          AS DEPT#
                      ,MAX(SALARY) AS MAX_SAL
            FROM        STAFF
            WHERE      DEPT < 50
            GROUP BY DEPT
            )AS STAFF_DEPT
ON          DEPT      = DEPT#
WHERE      NAME LIKE 'S%'
ORDER BY ID;

```

```

ANSWER
=====
ID  DEPT  SALARY  MAX_SAL
---  ---  -
10   20  18357.50  18357.50
190  20  14252.75  18357.50
200  42  11508.60  18352.80
220  51  17654.50   -

```

Temporary Tables - Identical Query (3 of 3)

Using full-select in SELECT

```

SELECT    ID
          , DEPT
          , SALARY
          , (SELECT    MAX(SALARY)
              FROM      STAFF S2
              WHERE     S1.DEPT = S2.DEPT
              AND       S2.DEPT < 50
              GROUP BY DEPT)
          AS MAX_SAL
FROM      STAFF S1
WHERE     NAME LIKE 'S%'
ORDER BY ID;

```

ANSWER

```

=====
ID  DEPT  SALARY  MAX_SAL
---  ----  -
10   20  18357.50  18357.50
190  20  14252.75  18357.50
200  42  11508.60  18352.80
220  51  17654.50   -

```

Common Table Expression (1 of 5)

Syntax



Using named fields

```

WITH TEMP1 AS
  (SELECT MAX(NAME) AS MAX_NAME
    ,MAX(DEPT) AS MAX_DEPT
   FROM   STAFF
  )
SELECT *
FROM   TEMP1;

```

```

ANSWER
=====
MAX_NAME  MAX_DEPT
-----
Yamaguchi      84

```

Using unnamed fields

```

WITH TEMP1 (MAX_NAME,MAX_DEPT) AS
  (SELECT MAX(NAME)
    ,MAX(DEPT)
   FROM   STAFF
  )
SELECT *
FROM   TEMP1;

```

```

ANSWER
=====
MAX_NAME  MAX_DEPT
-----
Yamaguchi      84

```

Common Table Expression (2 of 5)

Query with two common table expressions

```
WITH
TEMP1 AS
  (SELECT   DEPT
            ,AVG(SALARY) AS AVG_SAL
    FROM     STAFF
    GROUP BY DEPT),
TEMP2 AS
  (SELECT   MAX(AVG_SAL) AS MAX_AVG
    FROM     TEMP1)
SELECT *
FROM   TEMP2;
```

```
ANSWER
=====
MAX_AVG
-----
20865.8625
```

Same as above, but using nested table expressions

```
SELECT *
FROM   (SELECT MAX(AVG_SAL) AS MAX_AVG
        FROM   (SELECT DEPT
                  ,AVG(SALARY) AS AVG_SAL
                FROM   STAFF
                GROUP BY DEPT
              )AS TEMP1
      )AS TEMP2;
```

```
ANSWER
=====
MAX_AVG
-----
20865.8625
```

Common Table Expression (3 of 5) - Nested Usage

```

WITH TEMP1 AS
(SELECT   ID
         ,NAME
         ,DEPT
         ,SALARY
FROM     STAFF
WHERE    ID      < 300
AND      DEPT    <> 55
AND      NAME    LIKE 'S%'
AND      DEPT    NOT IN
        (SELECT DEPTNUMB
         FROM    ORG
         WHERE   DIVISION = 'SOUTHERN'
              OR   LOCATION = 'HARTFORD')
)
,TEMP2 AS
(SELECT   DEPT
         ,MAX(SALARY) AS MAX_SAL
FROM     TEMP1
GROUP BY DEPT
)
SELECT   T1.ID
         ,T1.DEPT
         ,T1.SALARY
         ,T2.MAX_SAL
FROM     TEMP1 T1
         ,TEMP2 T2
WHERE    T1.DEPT = T2.DEPT
ORDER BY T1.ID;

```

```

ANSWER
=====
ID  DEPT  SALARY  MAX_SAL
---  ---  -
10   20  18357.50  18357.50
190  20  14252.75  18357.50
200  42  11508.60  11508.60
220  51  17654.50  17654.50

```

Common Table Expression (4 of 5)

Insert Usage

```
INSERT INTO STAFF
WITH TEMP1 (MAX1) AS
  (SELECT MAX(ID) + 1
   FROM   STAFF
  )
SELECT MAX1, 'A', 1, 'B', 2, 3, 4
FROM   TEMP1;
```

Same as above, without Common Table Expression

```
INSERT INTO STAFF
SELECT MAX(ID) + 1
      , 'A', 1, 'B', 2, 3, 4
FROM   STAFF;
```

Common Table Expression (5 of 5)

Nested Column Function Usage

```

SELECT  DIVISION
        , DEC(AVG(DEPT_AVG), 7, 2) AS DIV_DEPT
        , COUNT(*)                AS #DPTS
        , SUM(#EMPS)              AS #EMPS
FROM    (SELECT  DIVISION
          , DEPT
          , AVG(SALARY) AS DEPT_AVG
          , COUNT(*)   AS #EMPS
        FROM    STAFF
          , ORG
        WHERE   DEPT = DEPTNUMB
        GROUP BY DIVISION
          , DEPT
        )AS XXX
GROUP BY DIVISION;

```

ANSWER

```

=====
DIVISION  DIV_DEPT  #DPTS  #EMPS
-----
Corporate 20865.86      1      4
Eastern   15670.32      3     13
Midwest   15905.21      2      9
Western   17946.40      2     11

```

Full-Select (1 of 6)

Nested full-selects

```

SELECT ID
FROM (SELECT *
      FROM (SELECT ID, YEARS, SALARY
            FROM (SELECT *
                  FROM (SELECT *
                        FROM STAFF
                        WHERE DEPT < 77
                      )AS T1
                     WHERE ID < 300
                    )AS T2
                 WHERE JOB LIKE 'C%'
                )AS T3
         WHERE SALARY < 18000
        )AS T4
WHERE YEARS < 5;

```

ANSWER

=====

```

ID
---
170
180
230

```

Notes

- Full-selects can be used in FROM or SELECT part of query
- If in FROM part of query, table must have a correlation name
- If in SELECT part of query, must one return only one value per row

Full-Select (2 of 6)

Join full-select to real table

```

SELECT    A.ID
          ,A.DEPT
          ,A.SALARY
          ,DEC(B.AVGSAL,7,2) AS AVG_DEPT
FROM      STAFF A
LEFT OUTER JOIN
          (SELECT    DEPT          AS DEPT
                  ,AVG(SALARY) AS AVGSAL
            FROM      STAFF
            GROUP BY DEPT
            HAVING    AVG(SALARY) > 16000
          )AS B
ON        A.DEPT = B.DEPT
WHERE     A.ID    < 40
ORDER BY A.ID;

```

ANSWER

```

=====
ID DEPT  SALARY  AVG_DEPT
--  ----  -
10  20  18357.50  16071.52
20  20  18171.25  16071.52
30  38  17506.75   -

```

Full-Select (3 of 6)

Full-Select with external table reference

```

SELECT   A.ID
         ,A.DEPT
         ,A.SALARY
         ,B.DEPTSAL
FROM     STAFF  A
         ,TABLE
         (SELECT   B.DEPT
              ,SUM(B.SALARY) AS DEPTSAL
            FROM     STAFF B
            WHERE    B.DEPT = A.DEPT
            GROUP BY B.DEPT
         )AS B
WHERE    A.ID    < 40
ORDER BY A.ID;

```

ANSWER

```

=====
ID DEPT  SALARY    DEPTSAL
-- ----  -
10 20    18357.50  64286.10
20 20    18171.25  64286.10
30 38    17506.75  77285.55

```

Full-Select without external table reference

```

SELECT   A.ID
         ,A.DEPT
         ,A.SALARY
         ,B.DEPTSAL
FROM     STAFF  A
         , (SELECT   B.DEPT
              ,SUM(B.SALARY) AS DEPTSAL
            FROM     STAFF B
            GROUP BY B.DEPT
         )AS B
WHERE    A.ID    < 40
        AND     B.DEPT = A.DEPT
ORDER BY A.ID;

```

ANSWER

```

=====
ID DEPT  SALARY    DEPTSAL
-- ----  -
10 20    18357.50  64286.10
20 20    18171.25  64286.10
30 38    17506.75  77285.55

```

Full-Select (4 of 6)

Use an uncorrelated Full-Select in a SELECT list

```
SELECT    ID
          , SALARY
          , (SELECT MAX(SALARY)
              FROM    STAFF
              ) AS MAXSAL
FROM      STAFF  A
WHERE     ID < 60
ORDER BY ID;
```

```
ANSWER
=====
ID  SALARY    MAXSAL
--  -
10  18357.50  22959.20
20  18171.25  22959.20
30  17506.75  22959.20
40  18006.00  22959.20
50  20659.80  22959.20
```

Use a correlated Full-Select in a SELECT list

```
SELECT    ID
          , SALARY
          , (SELECT MAX(SALARY)
              FROM    STAFF  B
              WHERE   A.DEPT = B.DEPT
              ) AS MAXSAL
FROM      STAFF  A
WHERE     ID < 60
ORDER BY ID;
```

```
ANSWER
=====
ID  SALARY    MAXSAL
--  -
10  18357.50  18357.50
20  18171.25  18357.50
30  17506.75  18006.00
40  18006.00  18006.00
50  20659.80  20659.80
```

Full-Select (5 of 6)

Use correlated and uncorrelated Full-Selects in a SELECT list

```
SELECT ID
      , DEPT
      , SALARY
      , (SELECT MAX(SALARY)
         FROM   STAFF B
         WHERE  B.DEPT = A.DEPT)
      , (SELECT MAX(SALARY)
         FROM   STAFF)
FROM   STAFF A
WHERE  ID < 60
ORDER BY ID;
```

```
ANSWER
=====
ID DEPT  SALARY  4          5
--  ----  -
10  20  18357.50  18357.50  22959.20
20  20  18171.25  18357.50  22959.20
30  38  17506.75  18006.00  22959.20
40  38  18006.00  18006.00  22959.20
50  15  20659.80  20659.80  22959.20
```

Full-select in INSERT

```
INSERT INTO STAFF
SELECT  ID + 1
      , (SELECT MIN(NAME)
         FROM   STAFF)
      , (SELECT DEPT
         FROM   STAFF S2
         WHERE  S2.ID = S1.ID - 100)
      , 'A', 1, 2, 3
FROM    STAFF S1
WHERE   ID =
        (SELECT MAX(ID)
         FROM   STAFF);
```

Full-Select (6 of 6)

Full-Select (uncorrelated), gives all workers AVG salary (+\$2000)

```
UPDATE STAFF A
SET    SALARY =
      (SELECT AVG(SALARY)+ 2000
       FROM   STAFF)
WHERE  ID < 60;
```

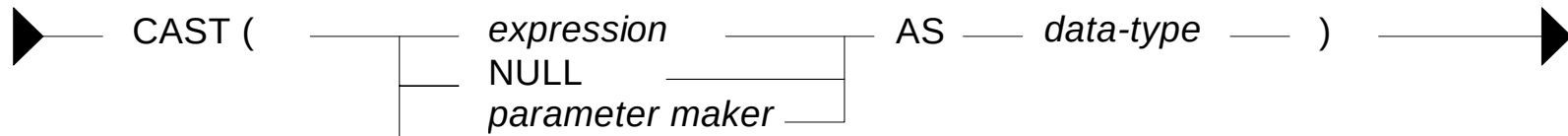
```
ANSWER:      SALARY
=====
ID DEPT BEFORE  AFTER
-- ---- -
10  20 18357.50 18675.64
20  20 18171.25 18675.64
30  38 17506.75 18675.64
40  38 18006.00 18675.64
50  15 20659.80 18675.64
```

Full-Select (correlated) gives all workers AVG salary per department (+\$2000)

```
UPDATE STAFF A
SET    SALARY =
      (SELECT AVG(SALARY) + 2000
       FROM   STAFF B
       WHERE  A.DEPT = B.DEPT )
WHERE  ID < 60;
```

```
ANSWER:      SALARY
=====
ID DEPT BEFORE  AFTER
-- ---- -
10  20 18357.50 18071.52
20  20 18171.25 18071.52
30  38 17506.75 17457.11
40  38 18006.00 17457.11
50  15 20659.80 17482.33
```

CAST Expression (1 of 2)



Notes

- Converts one data type to another
- Defines field type of temporary field

Convert Decimal to Integer

```
SELECT    ID
          , SALARY
          , CAST(SALARY AS INTEGER) AS SAL2
FROM      STAFF
WHERE     ID < 30
ORDER BY  ID;
```

```
ANSWER
=====
ID  SALARY    SAL2
--  -
10  18357.50  18357
20  18171.25  18171
```

CAST Expression (2 of 2)

Truncate Char field

```
SELECT    ID
          , JOB
          , CAST(JOB AS CHAR(3)) AS JOB2
FROM      STAFF
WHERE     ID < 30
ORDER BY  ID;
```

```
ANSWER
=====
ID JOB   JOB2
-- ----
10 Mgr   Mgr
20 Sales Sal
```

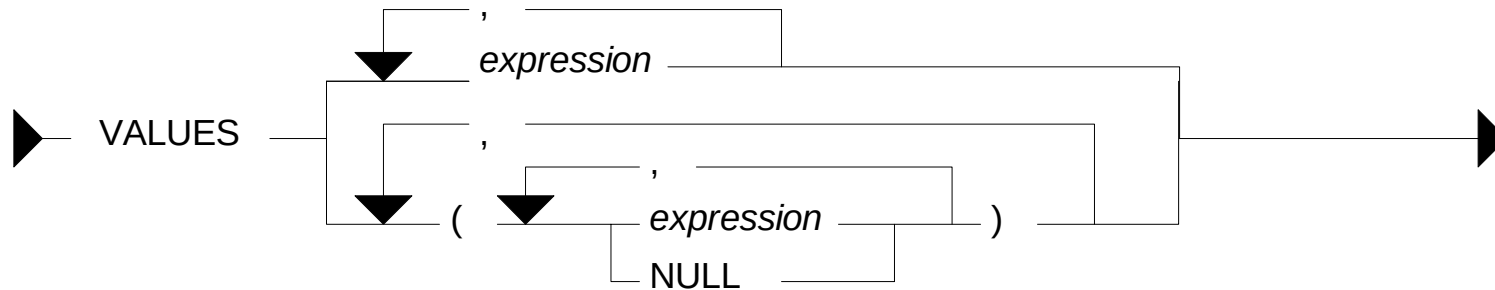
Define SMALLINT field with null values

```
SELECT    ID
          , CAST(NULL AS SMALLINT) AS JUNK
FROM      STAFF
WHERE     ID < 30
ORDER BY  ID;
```

```
ANSWER
=====
ID JUNK
-- ----
10     -
20     -
```

VALUES Expression (1 of 5)

Syntax



Notes

- Define one or more rows of temporary data

Examples

VALUES 6	<= 1 row, 1 column
VALUES (6)	<= 1 row, 1 column
VALUES 6, 7, 8	<= 1 row, 3 columns
VALUES (6), (7), (8)	<= 3 rows, 1 column
VALUES (6,66), (7,77), (8,NULL)	<= 3 rows, 2 columns

VALUES Expression (2 of 5)

Define a temporary table #1

```
WITH TEMP1 (COL1, COL2) AS
(VALUES      (      0, 'AA')
          , (      1, 'BB')
          , (      2, NULL)
        )
SELECT *
FROM   TEMP1;
```

```
ANSWER
=====
COL1 COL2
---- ----
      0 AA
      1 BB
      2 -
```

Define a temporary table #2

```
WITH TEMP1 (COL1, COL2) AS
(VALUES      (DECIMAL(0 ,3,1), 'AA')
          , (DECIMAL(1 ,3,1), 'BB')
          , (DECIMAL(2 ,3,1), NULL)
        )
SELECT *
FROM   TEMP1;
```

```
ANSWER
=====
COL1 COL2
---- ----
    0.0 AA
    1.0 BB
    2.0 -
```

VALUES Expression (3 of 5)

Define a temporary table #3

```
WITH TEMP1 (COL1, COL2) AS
(VALUES      (      0, CAST('AA' AS CHAR(1)))
      , (      1, CAST('BB' AS CHAR(1)))
      , (      2, CAST(NULL AS CHAR(1)))
)
SELECT *
FROM   TEMP1;
```

```
ANSWER
=====
COL1 COL2
---- ----
    0  A
    1  B
    2  -
```

Define a temporary table #4

```
WITH TEMP1 (COL1, COL2) AS
(VALUES      (      0, CHAR('AA',1))
      , (      1, CHAR('BB',1))
      , (      2, NULL)
)
SELECT *
FROM   TEMP1;
```

```
ANSWER
=====
COL1 COL2
---- ----
    0  A
    1  B
    2  -
```

VALUES Expression (4 of 5)

Derive one temporary table from another

```
WITH TEMP1 (COL1, COL2, COL3) AS
  (VALUES
    ( 0, 'AA', 0.00)
    , ( 1, 'BB', 1.11)
    , ( 2, 'CC', 2.22)
  )
, TEMP2 (COL1B, COLX) AS
  (SELECT COL1
    , COL1 + COL3
    FROM TEMP1
  )
SELECT *
FROM TEMP2;
```

```
ANSWER
=====
COL1B COLX
-----
      0 0.00
      1 2.11
      2 4.22
```

Define a view using a VALUES clause

```
CREATE VIEW SILLY (C1, C2, C3)
AS VALUES (11, 'AAA', SMALLINT(22))
    , (12, 'BBB', SMALLINT(33))
    , (13, 'CCC', NULL);
COMMIT;
```

VALUES Expression (5 of 5)

Seed a recursive SQL statement

```
WITH TEMP1 (COL1) AS
  (VALUES      0
   UNION ALL
   SELECT COL1 + 1
   FROM    TEMP1
   WHERE   COL1 + 1 < 100
  )
SELECT *
FROM   TEMP1;
```

ANSWER

=====

COL1

0

1

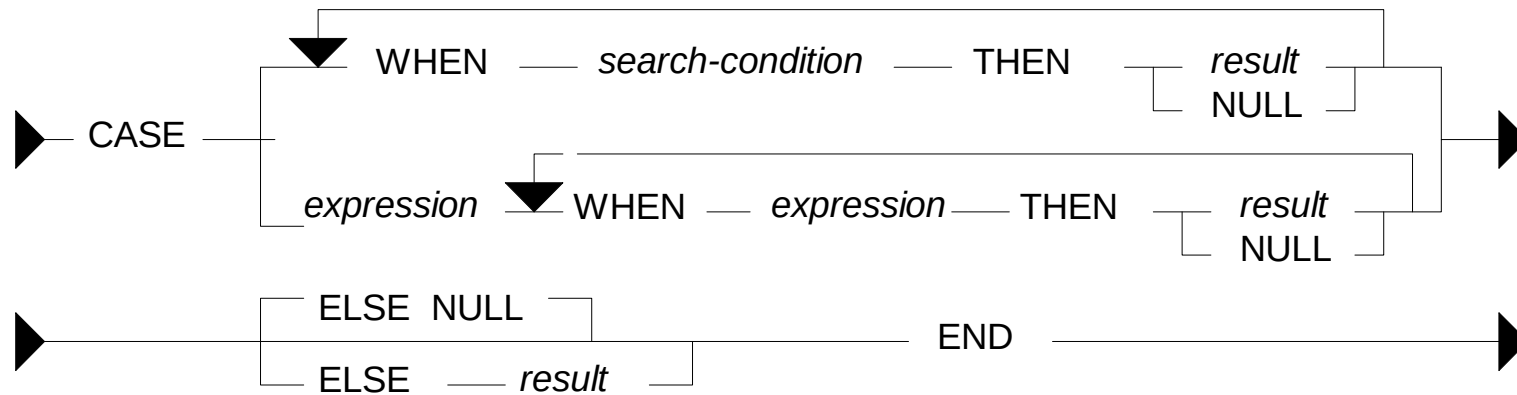
2

3

etc

CASE Expression (1 of 6)

Syntax



Notes

- Lets one have "if then else" logic in a SQL statement
- First CASE that matches is the one used
- If none matches, result is null (default)
- Two syntax styles - see next page

CASE Expression (2 of 6)

Use CASE (type 1) to expand a value

```
SELECT  LASTNAME
        ,SEX    AS SX
        ,CASE   SEX
            WHEN 'F' THEN 'FEMALE'
            WHEN 'M' THEN 'MALE'
            ELSE NULL
        END AS SEXX
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%'
ORDER BY 1;
```

```
ANSWER
=====
LASTNAME  SX  SEXX
-----  --  -----
JEFFERSON  M   MALE
JOHNSON    F   FEMALE
JONES      M   MALE
```

Use CASE (type 2) to expand a value

```
SELECT  LASTNAME
        ,SEX    AS SX
        ,CASE
            WHEN SEX = 'F' THEN 'FEMALE'
            WHEN SEX = 'M' THEN 'MALE'
            ELSE NULL
        END AS SEXX
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%'
ORDER BY 1;
```

```
ANSWER
=====
LASTNAME  SX  SEXX
-----  --  -----
JEFFERSON  M   MALE
JOHNSON    F   FEMALE
JONES      M   MALE
```

CASE Expression (3 of 6)

Use CASE to display the higher of two values

```

SELECT  LASTNAME
        ,MIDINIT AS MI
        ,SEX      AS SX
        ,CASE
            WHEN MIDINIT > SEX
            THEN MIDINIT
            ELSE SEX
        END AS MX
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%'
ORDER BY 1;

```

```

ANSWER
=====
LASTNAME  MI  SX  MX
-----
JEFFERSON  J   M   M
JOHNSON    P   F   P
JONES      T   M   T

```

Use CASE to get multiple counts in one pass

```

SELECT  COUNT(*) AS TOTAL
        ,SUM(CASE SEX WHEN 'F' THEN 1 ELSE 0 END) AS FEMALE
        ,SUM(CASE SEX WHEN 'M' THEN 1 ELSE 0 END) AS MALE
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%';

```

```

ANSWER
=====
TOT  #F  #M
---  --  --
    3   1   2

```

CASE Expression (4 of 6)

Use CASE in a predicate

```
SELECT  LASTNAME
        ,SEX
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%'
        AND CASE SEX
              WHEN 'F' THEN ''
              WHEN 'M' THEN ''
              ELSE NULL
        END IS NOT NULL
ORDER BY 1;
```

```
ANSWER
=====
LASTNAME  SEX
-----
JEFFERSON  M
JOHNSON    F
JONES      M
```

Use CASE inside a function

```
SELECT  LASTNAME
        ,LENGTH(RTRIM(LASTNAME)) AS LEN
        ,SUBSTR(LASTNAME,1,
        CASE
            WHEN LENGTH(RTRIM(LASTNAME))
                > 6 THEN 6
            ELSE LENGTH(RTRIM(LASTNAME))
        END ) AS LASTNM
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%'
ORDER BY 1;
```

```
ANSWER
=====
LASTNAME  LEN  LASTNM
-----
JEFFERSON    9  JEFFER
JOHNSON      7  JOHNSO
JONES        5  JONES
```

CASE Expression (5 of 6)

UPDATE statement with nested CASE expressions

```

UPDATE STAFF
SET   COMM = CASE DEPT
                WHEN 15 THEN COMM * 1.1
                WHEN 20 THEN COMM * 1.2
                WHEN 38 THEN
                    CASE
                        WHEN YEARS < 5 THEN COMM * 1.3
                        WHEN YEARS >= 5 THEN COMM * 1.4
                        ELSE NULL
                    END
                ELSE COMM
            END
WHERE COMM IS NOT NULL
      AND DEPT < 50;

```

Use CASE to avoid divide by zero

```

WITH TEMP1 (C1,C2) AS
(VALUES      (88,9),(44,3),(22,0),(0,1))
SELECT C1
      ,C2
      ,CASE C2
          WHEN 0 THEN NULL
          ELSE C1/C2
        END AS C3
FROM   TEMP1;

```

```

ANSWER
=====
C1  C2  C3
--  --  --
88   9   9
44   3  14
22   0   -
0    1   0

```

CASE Expression (6 of 6)

Use CASE to derive a value (correct)

```
SELECT  LASTNAME
        ,SEX
        ,CASE
            WHEN SEX >= 'M' THEN 'MAL'
            WHEN SEX >= 'F' THEN 'FEM'
        END AS SXX
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%'
ORDER BY 1;
```

```
ANSWER
=====
LASTNAME  SX  SXX
-----  --  ---
JEFFERSON  M  MAL
JOHNSON    F  FEM
JONES      M  MAL
```

Use CASE to derive a value (incorrect)

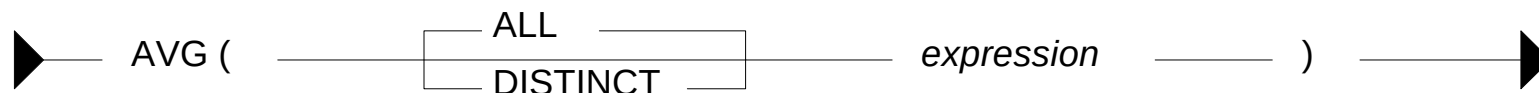
```
SELECT  LASTNAME
        ,SEX
        ,CASE
            WHEN SEX >= 'F' THEN 'FEM'
            WHEN SEX >= 'M' THEN 'MAL'
        END AS SXX
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%'
ORDER BY 1;
```

```
ANSWER
=====
LASTNAME  SX  SXX
-----  --  ---
JEFFERSON  M  FEM
JOHNSON    F  FEM
JONES      M  FEM
```

Column Functions

AVG Function (1 of 3)

Syntax



Notes

- Gets the average (mean) of a set of non-null rows
- If no rows match, null returned
- Beware of loss of precision (see A4 vs. A5 below)

Sample SQL

```
SELECT    AVG(DEPT)           AS A1
          ,AVG(ALL DEPT)      AS A2
          ,AVG(DISTINCT DEPT) AS A3
          ,AVG(DEPT/10)       AS A4
          ,AVG(DEPT)/10       AS A5
FROM      STAFF
HAVING    AVG(DEPT) > 40;
```

```
ANSWER
=====
A1 A2 A3 A4 A5
-- -- -- -- --
41 41 40 3  4
```

AVG Function (2 of 3)

Convert zero to null before doing AVG

```
SELECT AVG(SALARY) AS SALARY
      ,AVG(COMM)    AS COMM1
      ,AVG(CASE COMM
            WHEN 0 THEN NULL
            ELSE COMM
            END) AS COMM2
FROM   STAFF;
```

```
ANSWER
=====
SALARY  COMM1  COMM2
-----  -
16675.6 351.9  513.3
```

Convert null output (from AVG) to zero

```
SELECT  COUNT(*) AS C1
        ,AVG(SALARY) AS A1
        ,COALESCE(AVG(SALARY),0) AS A2
        ,CASE
            WHEN AVG(SALARY) IS NULL THEN 0
            ELSE AVG(SALARY)
          END AS A3
FROM    STAFF
WHERE   ID < 10;
```

```
ANSWER
=====
C1  A1  A2  A3
--  --  --  --
0   -   0   0
```

AVG Function (3 of 3)

AVG of date column

```
SELECT  AVG(DAYS(BIRTHDATE))
        , DATE(AVG(DAYS(BIRTHDATE)))
FROM    EMPLOYEE;
```

```
ANSWER
=====
1      2
-----
709113 06/27/1942
```

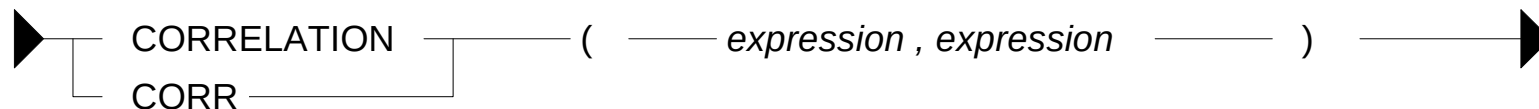
Select average of average

```
SELECT  AVG(AVG_SAL) AS AVG_AVG
FROM    (SELECT  DEPT
            ,AVG(SALARY) AS AVG_SAL
        FROM    STAFF
        GROUP BY DEPT
        )AS XXX;
```

```
ANSWER
=====
<Overflow error>
```

CORRELATION Function

Syntax


CORRELATION (*expression , expression*)

Sample SQL

```

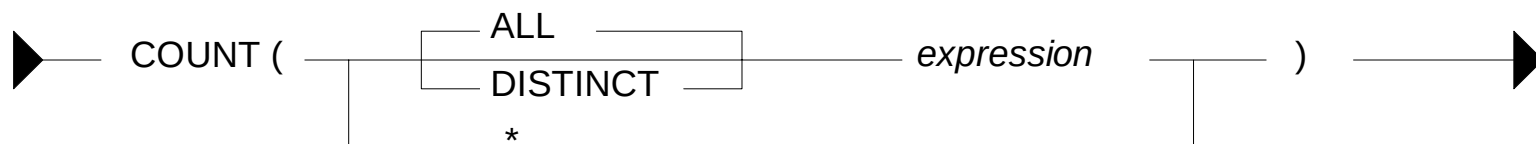
WITH TEMP1(COL1, COL2, COL3, COL4) AS
(VALUES (0
        ,0
        ,0
        ,RAND(1))
 UNION ALL
 SELECT COL1 + 1
        ,COL2 - 1
        ,RAND()
        ,RAND()
 FROM    TEMP1
 WHERE   COL1 <= 1000
 )
 SELECT DEC(CORRELATION(COL1,COL1),5,3) AS COR11
        ,DEC(CORRELATION(COL1,COL2),5,3) AS COR12
        ,DEC(CORRELATION(COL2,COL3),5,3) AS COR23
        ,DEC(CORRELATION(COL3,COL4),5,3) AS COR34
 FROM    TEMP1;
  
```

```

ANSWER
=====
COR11  COR12  COR23  COR34
-----
 1.000 -1.000 -0.017 -0.005
  
```

COUNT Function (1 of 2)

Syntax



Notes

- **COUNT(*)** counts matching rows
- **COUNT(expression)** counts rows with a non-null expression values
- **COUNT(ALL expression)** is the same as **COUNT(expression)** statement
- **COUNT(DISTINCT expression)** counts distinct non-null expression values

Sample SQL

SELECT COUNT(*)	AS C1	ANSWER
, COUNT(INT(COMM/10))	AS C2	=====
, COUNT(ALL INT(COMM/10))	AS C3	C1 C2 C3 C4 C5 C6
, COUNT(DISTINCT INT(COMM/10))	AS C4	-- -- -- -- --
, COUNT(DISTINCT INT(COMM))	AS C5	35 24 24 19 24 2
, COUNT(DISTINCT INT(COMM))/10	AS C6	
FROM STAFF;		

COUNT Function (2 of 2)

COUNT function with and without GROUP BY

```

SELECT      'NO GP-BY'    AS C1
            , COUNT(*)    AS C2
FROM        STAFF
WHERE       ID = -1
UNION
SELECT      'GROUP-BY'   AS C1
            , COUNT(*)    AS C2
FROM        STAFF
WHERE       ID = -1
GROUP BY    DEPT;

```

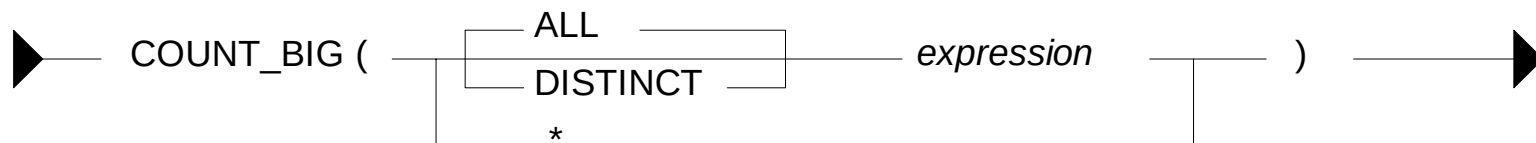
```

ANSWER
=====
C1          C2
----- --
NO GP-BY    0

```

COUNT_BIG Function

Syntax



Notes

- Same COUNT function, but returns BIGINT value

Sample SQL

```

SELECT    COUNT_BIG(*)           AS C1
          , COUNT_BIG(DEPT)       AS C2
          , COUNT_BIG(DISTINCT DEPT) AS C3
          , COUNT_BIG(DISTINCT DEPT/10) AS C4
          , COUNT_BIG(DISTINCT DEPT)/10 AS C5
FROM      STAFF;

```

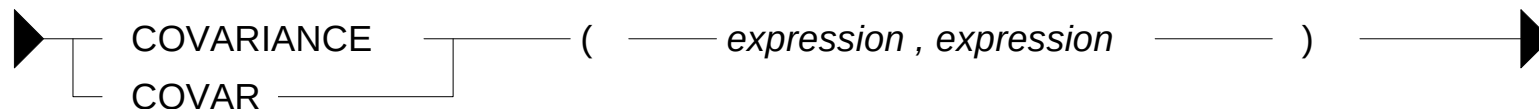
```

ANSWER
=====
C1  C2  C3  C4  C5
---  ---  ---  ---  ---
35. 35.  8.  7.  0.

```

COVARIANCE Function

Syntax



Sample SQL

```

WITH TEMP1(C1, C2, C3, C4) AS
(VALUES (0
        ,0
        ,0
        ,RAND(1))
 UNION ALL
 SELECT C1 + 1
        ,C2 - 1
        ,RAND()
        ,RAND()
 FROM   TEMP1
 WHERE  C1 <= 1000
 )
 SELECT DEC(COVARIANCE(C1,C1),6,0) AS COV11
        ,DEC(COVARIANCE(C1,C2),6,0) AS COV12
        ,DEC(COVARIANCE(C2,C3),6,4) AS COV23
        ,DEC(COVARIANCE(C3,C4),6,4) AS COV34
 FROM   TEMP1;

```

```

ANSWER
=====
COV11   COV12   COV23   COV34
-----
83666.  -83666.  -1.4689 -0.0004

```

GROUPING Function

► GROUPING (*expression*) ►

Notes

- Used in CUBE, ROLLUP, and GROUPING SETS statements
- Identifies what rows came from which grouping set

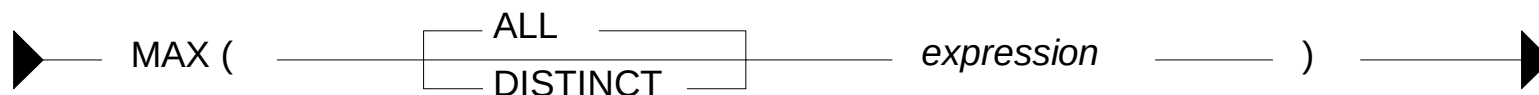
Sample SQL

```
SELECT    DEPT
          ,AVG(SALARY)      AS SALARY
          ,GROUPING(DEPT) AS DF
FROM      STAFF
GROUP BY  ROLLUP(DEPT)
ORDER BY  DEPT;
```

```
ANSWER
=====
DEPT  SALARY    DF
----  -
10    20865.86    0
15    15482.33    0
20    16071.52    0
38    15457.11    0
42    14592.26    0
51    17218.16    0
66    17215.24    0
84    16536.75    0
-     16675.64    1
```

MAX Function (1 of 2)

Syntax



Notes

- Gets max value from a set of rows
- The DISTINCT option has no affect
- If no rows match, null returned

Sample SQL

```
SELECT    MAX(DEPT)
          , MAX(ALL DEPT)
          , MAX(DISTINCT DEPT)
          , MAX(DISTINCT DEPT/10)
FROM      STAFF;
```

```
ANSWER
=====
  1    2    3    4
---  ---  ---  ---
 84   84   84    8
```

MAX Function (2 of 2)

MAX function with dates

```
SELECT MAX(HIREDATE)
       , CHAR(MAX(HIREDATE), USA)
       , MAX(CHAR(HIREDATE, USA))
FROM   EMPLOYEE;
```

```
ANSWER
=====
1           2           3
-----
09/30/1980 09/30/1980 12/15/1976
```

MAX function with numbers, 1 of 2

```
SELECT MAX(ID)           AS ID
       , MAX(CHAR(ID))   AS CHR
       , MAX(DIGITS(ID)) AS DIG
FROM   STAFF;
```

```
ANSWER
=====
ID      CHR      DIG
-----
      350 90      00350
```

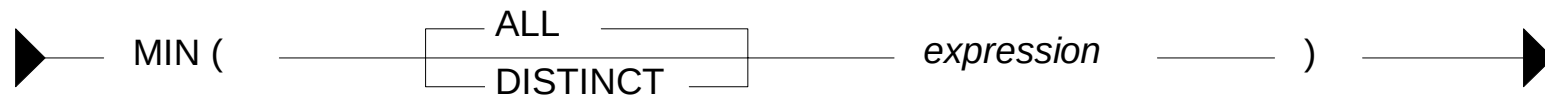
MAX function with numbers, 2 of 2

```
SELECT MAX(ID - 250)      AS ID
       , MAX(CHAR(ID - 250)) AS CHR
       , MAX(DIGITS(ID - 250)) AS DIG
FROM   STAFF;
```

```
ANSWER
=====
ID      CHR      DIG
-----
      100 90      0000000240
```

MIN Function

Syntax

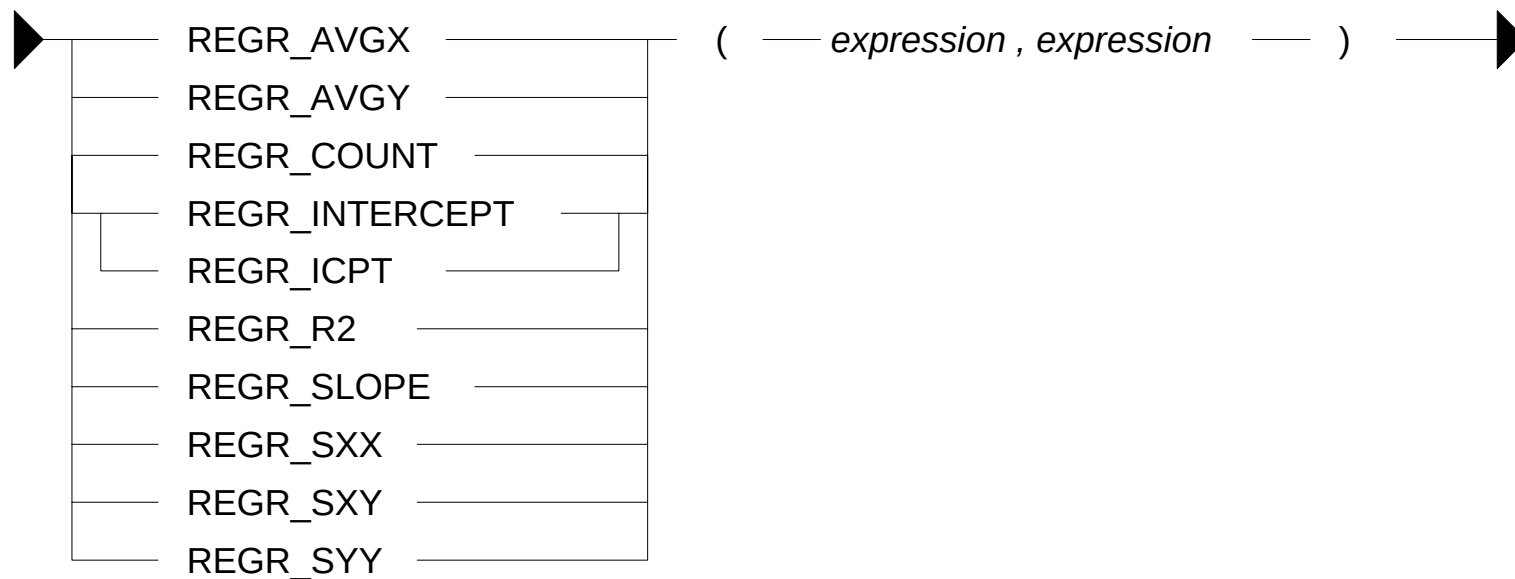


Sample SQL

```
SELECT    MIN(DEPT)
          , MIN(ALL DEPT)
          , MIN(DISTINCT DEPT)
          , MIN(DISTINCT DEPT/10)
FROM      STAFF;
```

```
ANSWER
=====
  1    2    3    4
---  ---  ---  ---
 10   10   10    1
```

REGRESSION Functions



Sample SQL

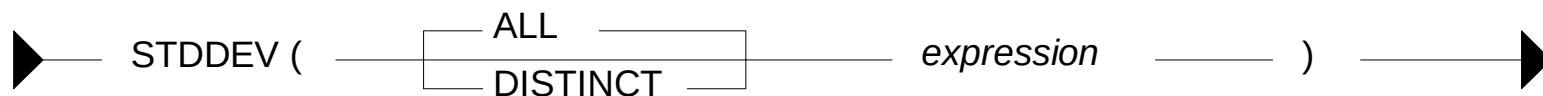
```

SELECT  REGR_SLOPE(BONUS, SALARY)      AS R_SLOPE
        , REGR_INTERCEPT(BONUS, SALARY) AS R_ICPT
        , REGR_COUNT(BONUS, SALARY)    AS R_COUNT
        , REGR_AVGX(BONUS, SALARY)     AS R_AVGX
        , REGR_AVGY(BONUS, SALARY)     AS R_AVGY
        , REGR_SXX(BONUS, SALARY)      AS R_SXX
        , REGR_SXY(BONUS, SALARY)      AS R_SXY
        , REGR_SYY(BONUS, SALARY)      AS R_SYY
FROM    EMPLOYEE
WHERE   WORKDEPT = 'A00';

```

STDDEV Function

Syntax



Sample SQL

```

SELECT AVG(DEPT) AS A1
      ,STDDEV(DEPT) AS S1
      ,DEC(STDDEV(DEPT),3,1) AS S2
      ,DEC(STDDEV(ALL DEPT),3,1) AS S3
      ,DEC(STDDEV(DISTINCT DEPT),3,1) AS S4
FROM   STAFF;

```

ANSWER

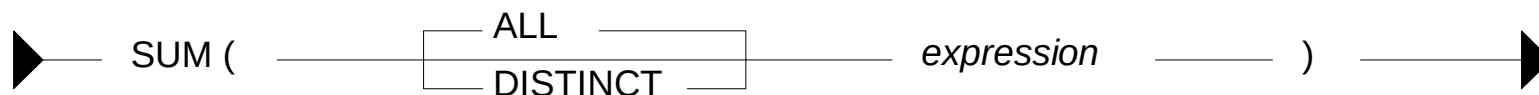
```

=====
A1  S1                S2   S3   S4
--  - - - - - - - - - - - - - - -
41  +2.3522355E+1    23.5  23.5  24.1

```

SUM Function

Syntax



Notes

- Gets the SUM of a set of numeric values (null if no rows match)
- If **DISTINCT** used, duplicate values are ignored
- Nulls values always ignored
- Beware loss of precision (see S4 vs. S5 below)

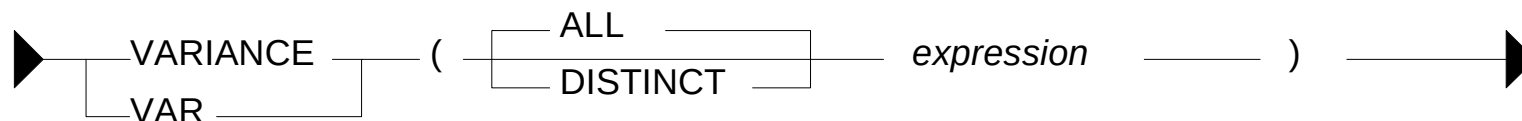
Sample SQL

```
SELECT    SUM(DEPT)           AS S1
          , SUM(ALL DEPT)     AS S2
          , SUM(DISTINCT DEPT) AS S3
          , SUM(DEPT/10)      AS S4
          , SUM(DEPT)/10      AS S5
FROM      STAFF;
```

```
ANSWER
=====
  S1    S2    S3    S4    S5
----
1459 1459   326   134   145
```

VARIANCE Function

Syntax



Sample SQL

```

SELECT AVG(DEPT) AS A1
      , VARIANCE(DEPT) AS S1
      , DEC(VARIANCE(DEPT), 4, 1) AS S2
      , DEC(VARIANCE(ALL DEPT), 4, 1) AS S3
      , DEC(VARIANCE(DISTINCT DEPT), 4, 1) AS S4
FROM   STAFF;
  
```

ANSWER

```

=====
A1  V1                                V2  V3  V4
--  - - - - - - - - - - - - - - - -
41  +5.533012244E+2  553  553  582
  
```

OLAP Functions

OLAP Functions

- RANK()**

Returns number of preceding rows + 1 (for given value)
 If >1 row has same value - all get same rank
 - next row counts all prior rows
 - can have gaps in sequence
- DENSE_RANK()**

Returns number of distinct preceding values + 1 (for value)
 If >1 row has same value - all get same rank
 - next row counts prior values
 - no gaps in sequence
- ROW_NUMBER()**

Returns row number
 Use ORDER BY to get row number within field(s)
- Col. function OVER()**

Uses standard DB2 column functions
 Gets cumulative totals, and/or running averages

Example

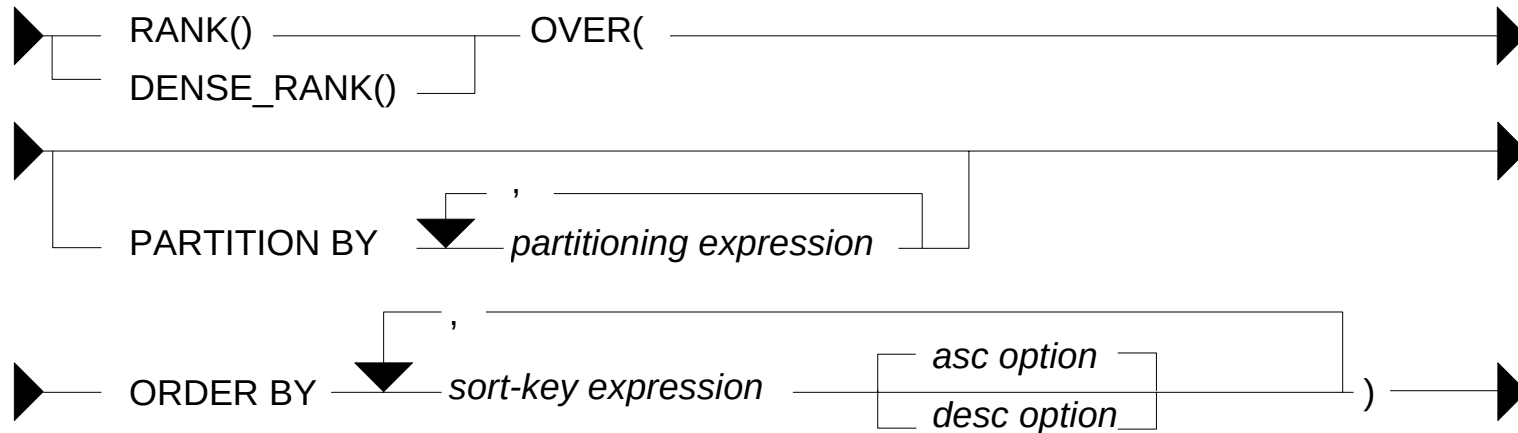
```

SELECT      S1.JOB, S1.ID, S1.SALARY
            ,SUM(SALARY) OVER(ORDER BY JOB, ID) AS SUMSAL
            ,ROW_NUMBER() OVER(ORDER BY JOB, ID) AS R
FROM        STAFF S1
WHERE       S1.NAME LIKE '%S%'
AND        S1.ID      <  90
ORDER BY   S1.JOB
            ,S1.ID;

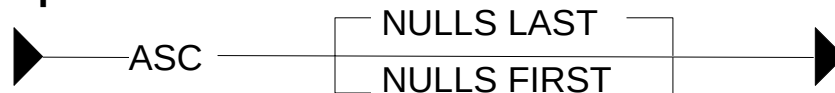
```

					ANSWER
=====					
JOB	ID	SALARY	SUMSAL	R	
----	--	-----	-----	-	
Clerk	80	13504.60	13504.60	1	
Mgr	10	18357.50	31862.10	2	
Mgr	50	20659.80	52521.90	3	

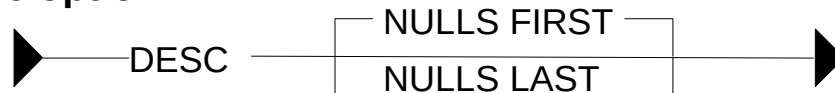
Ranking Functions Syntax



asc option



desc option



Note

- "OVER(... ORDER BY field(s))" is mandatory

Ranking Functions Example

```

SELECT    ID
          , YEARS
          , SALARY
          , RANK( )           OVER(ORDER BY YEARS) AS RANK#
          , DENSE_RANK( )    OVER(ORDER BY YEARS) AS DENSE#
          , ROW_NUMBER( )    OVER(ORDER BY YEARS) AS ROW#
FROM      STAFF
WHERE     ID      < 100
        AND     YEARS IS NOT NULL
ORDER BY  YEARS;

```

ANSWER

```

=====
ID YEARS  SALARY    RANK# DENSE# ROW#
--  - - - - -
30      5 17506.75      1      1      1
40      6 18006.00      2      2      2
90      6 18001.75      2      2      3
10      7 18357.50      4      3      4
70      7 16502.83      4      3      5
20      8 18171.25      6      4      6
50     10 20659.80      7      5      7

```

Notes

- **ORDER BY** is mandatory
- **ORDER BY** indicates ranking sequence
- **ORDER BY** in function has no relationship to **ORDER BY** in query

ORDER BY Usage

```

SELECT  JOB , YEARS , ID , NAME
        , SMALLINT(RANK() OVER(ORDER BY JOB  ASC)) AS ASC1
        , SMALLINT(RANK() OVER(ORDER BY JOB  ASC
                                     , YEARS ASC)) AS ASC2
        , SMALLINT(RANK() OVER(ORDER BY JOB  ASC
                                     , YEARS ASC
                                     , ID    ASC)) AS ASC3
        , SMALLINT(RANK() OVER(ORDER BY JOB  DESC)) AS DSC1
        , SMALLINT(RANK() OVER(ORDER BY JOB  DESC
                                     , YEARS DESC)) AS DSC2
        , SMALLINT(RANK() OVER(ORDER BY JOB  DESC
                                     , YEARS DESC
                                     , ID    DESC)) AS DSC3
        , SMALLINT(RANK() OVER(ORDER BY JOB  ASC
                                     , YEARS DESC
                                     , ID    ASC)) AS MIX1
        , SMALLINT(RANK() OVER(ORDER BY JOB  DESC
                                     , YEARS ASC
                                     , ID    DESC)) AS MIX2

```

```

FROM      STAFF
WHERE     ID      < 150
        AND     YEARS IN (6,7)
        AND     JOB  > 'L'

```

											ANSWER	
											=====	
ORDER BY JOB	YEARS	ID	NAME	ASC1	ASC2	ASC3	DSC1	DSC2	DSC3	MIX1	MIX2	

Mgr	6	140	Fraye	1	1	1	4	6	6	3	4	
Mgr	7	10	Sanders	1	2	2	4	4	5	1	6	
Mgr	7	100	Plotz	1	2	3	4	4	4	2	5	
Sales	6	40	O'Brien	4	4	4	1	2	3	5	2	
Sales	6	90	Koonitz	4	4	5	1	2	2	6	1	
Sales	7	70	Rothman	4	6	6	1	1	1	4	3	

Ordering NULL Values

```

SELECT      ID
           , YEARS                                AS YR
           , SALARY
           , DENSE_RANK( ) OVER(ORDER BY YEARS ASC)           AS A
           , DENSE_RANK( ) OVER(ORDER BY YEARS ASC NULLS FIRST) AS AF
           , DENSE_RANK( ) OVER(ORDER BY YEARS ASC NULLS LAST ) AS AL
           , DENSE_RANK( ) OVER(ORDER BY YEARS DESC)           AS D
           , DENSE_RANK( ) OVER(ORDER BY YEARS DESC NULLS FIRST) AS DF
           , DENSE_RANK( ) OVER(ORDER BY YEARS DESC NULLS LAST ) AS DL
FROM        STAFF
WHERE       ID      < 100
ORDER BY    YEARS
           , SALARY;

```

ANSWER

```

=====
ID YR SALARY      A  AF AL  D  DF DL
-- -- -
30  5 17506.75    1  2  1  6  6  5
90  6 18001.75    2  3  2  5  5  4
40  6 18006.00    2  3  2  5  5  4
70  7 16502.83    3  4  3  4  4  3
10  7 18357.50    3  4  3  4  4  3
20  8 18171.25    4  5  4  3  3  2
50 10 20659.80    5  6  5  2  2  1
80  - 13504.60    6  1  6  1  1  6
60  - 16808.30    6  1  6  1  1  6

```

Notes

- Can define if NULLS rank high or low
- By default - rank high for an ascending field, low for a descending field

Partition Usage

```

SELECT    ID
          , YEARS   AS YR
          , SALARY
          , RANK() OVER(PARTITION BY YEARS
                        ORDER   BY SALARY) AS R1
FROM      STAFF
WHERE     ID        < 80
        AND YEARS IS NOT NULL
ORDER BY YEARS
        , SALARY;

```

```

ANSWER
=====
ID YR  SALARY    R1
-- --  -
30  5  17506.75    1
40  6  18006.00    1
70  7  16502.83    1
10  7  18357.50    2
20  8  18171.25    1
50  0  20659.80    1

```

Notes

- Rank values by subset of matching rows
- In above example, rows ranked by salary within year

Subsequent Processing

```

SELECT    XXX.*
          ,RANK()OVER(ORDER BY ID) AS R2
FROM      (SELECT    ID
            ,NAME
            ,RANK() OVER(ORDER BY ID) AS R1
            FROM      STAFF
            WHERE      ID    < 100
            AND        YEARS IS NOT NULL
          )AS XXX
WHERE     ID > 30
ORDER BY ID;

```

```

ANSWER
=====
ID NAME      R1 R2
-- -
40 O'Brien   4  1
50 Hanes     5  2
70 Rothman   6  3
90 Koonitz   7  4

```

Notes

- RANK function gets rank, as of when the function is applied
- Subsequent processing can change the rank (in final output)
- In above example, rows are removed after getting R1 values

Ordering Rows by Rank

```

SELECT      ID
            ,RANK() OVER(PARTITION BY DEPT
                          ORDER BY SALARY DESC) AS R1
            ,SALARY
            ,DEPT AS DP
FROM        STAFF
WHERE       ID      < 80
           AND      YEARS IS NOT NULL
ORDER BY    R1      ASC
            ,SALARY DESC;

```

```

ANSWER
=====
ID R1  SALARY    DP
-- --  -
50  1  20659.80  15
10  1  18357.50  20
40  1  18006.00  38
20  2  18171.25  20
30  2  17506.75  38
70  2  16502.83  15

```

Notes

- Query orders output by rank of salary within department

Get Highest Salary in Each Department

```

SELECT    ID
          ,SALARY
          ,DEPT AS DP
FROM      (SELECT S1.*
           ,RANK() OVER(PARTITION BY DEPT
                        ORDER BY SALARY DESC) AS R1
           FROM    STAFF S1
           WHERE   ID    < 80
           AND     YEARS IS NOT NULL
           )AS XXX
WHERE     R1 = 1
ORDER BY DP;

```

```

ANSWER
=====
ID  SALARY    DP
--  -
50  20659.80  15
10  18357.50  20
40  18006.00  38

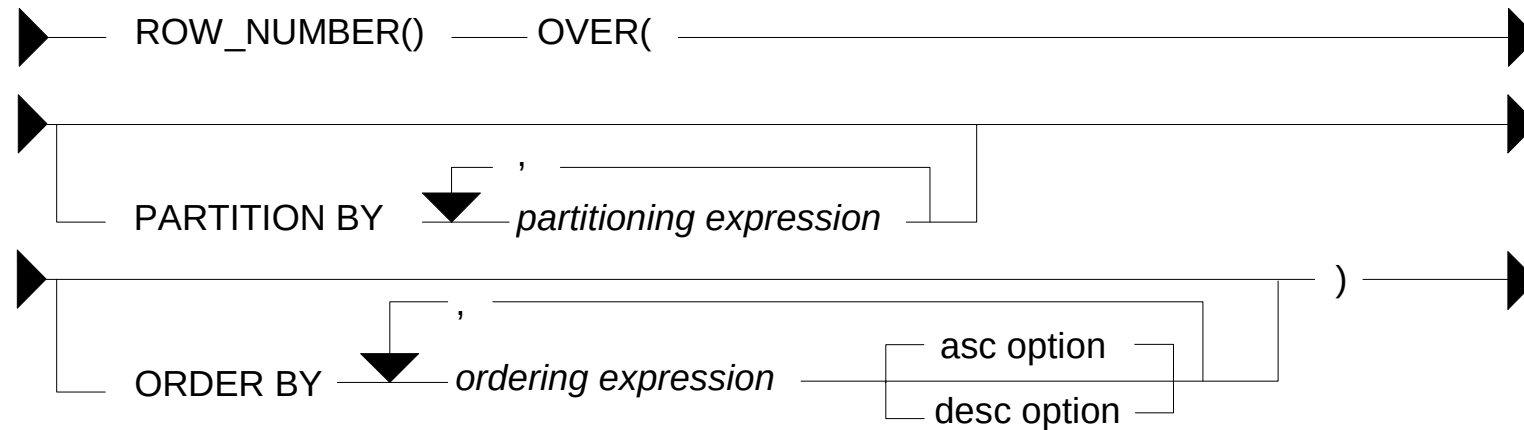
```

Notes

- In above example, rank is worked out, then subsequently queried

Row Number Function

Syntax



Notes

- Numbers rows returned
- Output type is BIGINT

ORDER BY Usage

SELECT	ID			ANSWER
	, NAME			=====
	, ROW_NUMBER() OVER()	AS R1		ID NAME R1 R2 R3
	, ROW_NUMBER() OVER(ORDER BY ID)	AS R2		-- ----- -- -- --
	, ROW_NUMBER() OVER(ORDER BY NAME)	AS R3		10 Sanders 1 1 4
FROM	STAFF			20 Pernal 2 2 3
WHERE	ID < 50			30 Marenghi 3 3 1
	AND YEARS IS NOT NULL			40 O'Brien 4 4 2
ORDER BY	ID;			

Notes

- **ORDER BY** in function is not related to **ORDER BY** in output
- **ORDER BY** is not required, but is wise to have
- If not supplied, result can be meaningless

PARTITION Usage

```

SELECT  JOB
        , YEARS
        , ID
        , NAME
        , ROW_NUMBER( ) OVER(PARTITION BY JOB
                               ORDER      BY YEARS) AS ROW#
        , RANK( )      OVER(PARTITION BY JOB
                               ORDER      BY YEARS) AS RN1#
        , DENSE_RANK( ) OVER(PARTITION BY JOB
                               ORDER      BY YEARS) AS RN2#
FROM    STAFF
WHERE   ID      < 150
       AND YEARS IN (6,7)
       AND JOB  > 'L'
ORDER BY JOB
        , YEARS;

```

ANSWER

```

=====
JOB   YEARS ID  NAME      ROW#  RN1#  RN2#
-----
Mgr    6  140 Fraye       1     1     1
Mgr    7   10 Sanders     2     2     2
Mgr    7  100 Plotz       3     2     2
Sales  6   40 O'Brien    1     1     1
Sales  6   90 Koonitz    2     1     1
Sales  7   70 Rothman    3     3     2

```

Notes

- Assign row numbers, by subset of matching rows
- In above example, rows numbers by year within job

Selecting "n" Rows (1 of 3)

Select first 3 rows - Use ROW_NUMBER Function

```

SELECT      *
FROM        (SELECT      ID
                  ,NAME
                  ,ROW_NUMBER() OVER(ORDER BY ID) AS R
              FROM        STAFF
              WHERE       ID      < 100
              AND         YEARS IS NOT NULL
            )AS XXX
WHERE       R <= 3
ORDER BY   ID;

```

```

ANSWER
=====
ID NAME      R
-- - - - - -
10 Sanders   1
20 Pernal    2
30 Marenghi  3

```

Select first 3 rows - Use FETCH FIRST Notation

```

SELECT      ID
            ,NAME
            ,ROW_NUMBER() OVER(ORDER BY ID) AS R
FROM        STAFF
WHERE       ID      < 100
            AND     YEARS IS NOT NULL
ORDER BY   ID
FETCH FIRST 3 ROWS ONLY;

```

```

ANSWER
=====
ID NAME      R
-- - - - - -
10 Sanders   1
20 Pernal    2
30 Marenghi  3

```

Selecting "n" Rows (2 of 3)

Select 3rd through 6th rows

```

SELECT      *
FROM        (SELECT      ID
                  ,NAME
                  ,ROW_NUMBER() OVER(ORDER BY ID) AS R
              FROM        STAFF
              WHERE       ID          < 200
                  AND     YEARS IS NOT NULL
              )AS XXX
WHERE       R BETWEEN 3 AND 6
ORDER BY ID;

```

```

ANSWER
=====
ID  NAME      R
--  - - - - - -
30  Marenghi  3
40  O'Brien   4
50  Hanes     5
70  Rothman   6

```

Select every 5th matching row

```

SELECT      *
FROM        (SELECT      ID
                  ,NAME
                  ,ROW_NUMBER() OVER(ORDER BY ID) AS R
              FROM        STAFF
              WHERE       ID          < 200
                  AND     YEARS IS NOT NULL
              )AS XXX
WHERE       (R - 1) = ((R - 1) / 5) * 5
ORDER BY ID;

```

```

ANSWER
=====
ID  NAME      R
--- - - - - - -
10  Sanders    1
70  Rothman    6
140 Fraye     11
190 Sneider   16

```

Selecting "n" Rows (2 of 3)

Select last two rows

```

SELECT      *
FROM        (SELECT      ID
                  ,NAME
                  ,ROW_NUMBER() OVER(ORDER BY ID DESC) AS R
              FROM        STAFF
              WHERE       ID          < 200
                  AND      YEARS IS NOT NULL
              )AS XXX
WHERE       R <= 2
ORDER BY   ID;

```

```

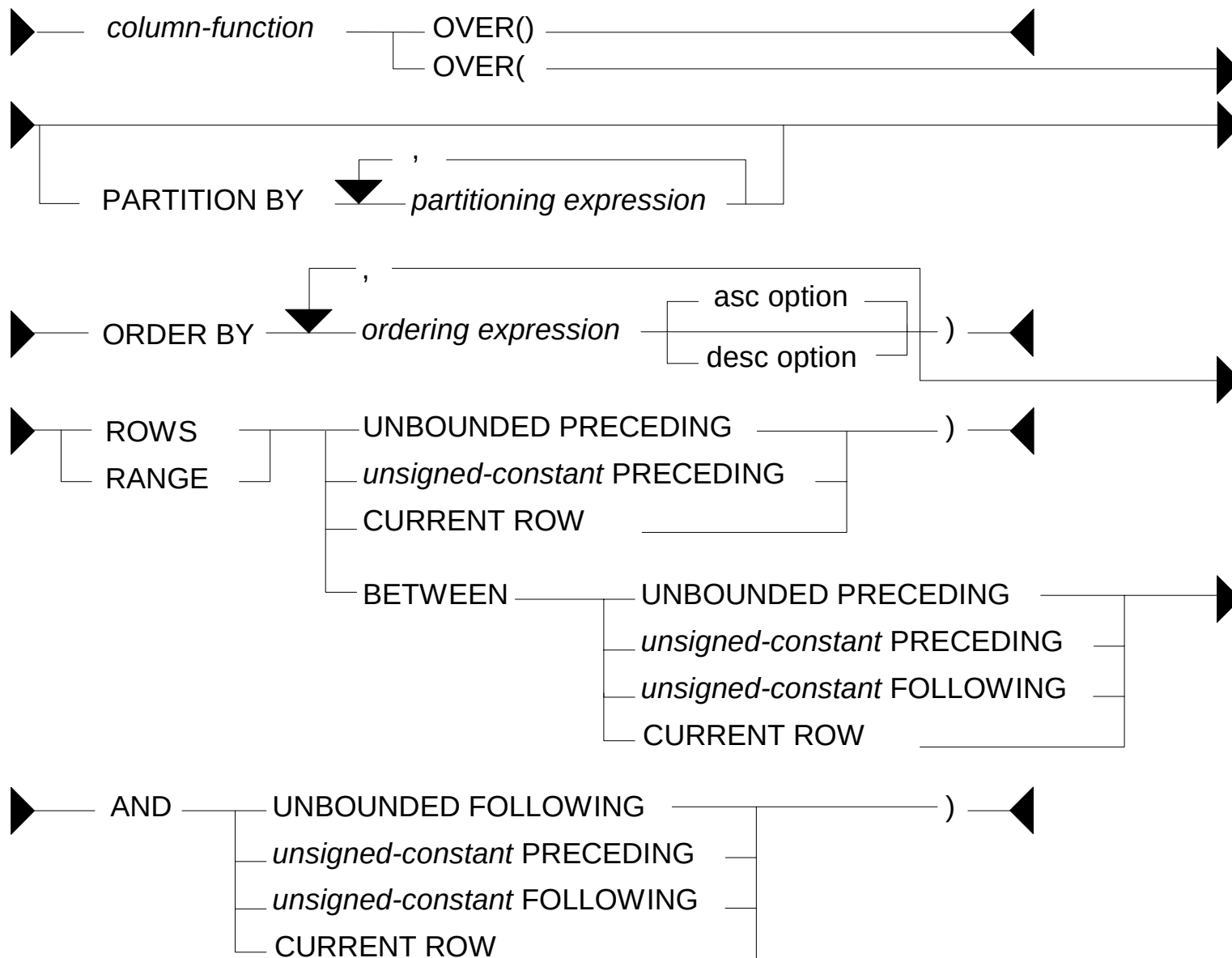
ANSWER
=====
ID   NAME      R
---  -
180  Abrahams    2
190  Sneider     1

```

Notes

- **ROW_NUMBER** is more useful than **FIRST FETCH** notation
- Beware selecting "n" rows if "n+1" row has the same value as "n"

Aggregation Function, Syntax



Aggregation Function, Notes

- Any column function (e.g. AVG) can use the aggregation function
- **OVER()** Aggregates all rows
- **PARTITION** Limits aggregation to subset of rows
- **ORDER BY** Provides values to aggregate on
Defines direction of aggregation
- **ROWS** Limits aggregation to a set of rows
- **RANGE** Limits aggregation to a set of values

Example

```

SELECT    ID
          , NAME
          , SALARY
          , SUM(SALARY) OVER() AS SUM_SAL
          , AVG(SALARY) OVER() AS AVG_SAL
          , MIN(SALARY) OVER() AS MIN_SAL
          , MAX(SALARY) OVER() AS MAX_SAL
          , COUNT(*)    OVER() AS #ROWS
FROM      STAFF
WHERE     ID < 60
ORDER BY  ID;

```

Example

```

SELECT    ID
          , NAME
          , SALARY
          , SUM(SALARY) OVER() AS SUM_SAL
          , AVG(SALARY) OVER() AS AVG_SAL
          , MIN(SALARY) OVER() AS MIN_SAL
          , MAX(SALARY) OVER() AS MAX_SAL
          , COUNT(*)      OVER() AS #ROWS
FROM      STAFF
WHERE     ID < 60
ORDER BY ID;

```

ANSWER

```

=====
ID  NAME      SALARY    SUM_SAL    AVG_SAL    MIN_SAL    MAX_SAL    #ROWS
--  -
10  Sanders    18357.50   92701.30   18540.26   17506.75   20659.80     5
20  Pernal     18171.25   92701.30   18540.26   17506.75   20659.80     5
30  Marenghi   17506.75   92701.30   18540.26   17506.75   20659.80     5
40  O'Brien    18006.00   92701.30   18540.26   17506.75   20659.80     5
50  Hanes      20659.80   92701.30   18540.26   17506.75   20659.80     5

```

Notes

- Above example selects detailed data, plus summary data - all on one line.

OVER() Usage

```

SELECT    ID
          , NAME
          , SALARY
          , SUM(SALARY) OVER( )                      AS SUM1
          , SUM(SALARY) OVER(ORDER BY ID * 0)        AS SUM2
          , SUM(SALARY) OVER(ORDER BY 'ABC' )        AS SUM3
          , SUM(SALARY) OVER(ORDER BY 'ABC'
                                RANGE BETWEEN UNBOUNDED PRECEDING
                                AND UNBOUNDED FOLLOWING) AS SUM4

FROM      STAFF
WHERE     ID < 60
ORDER BY ID;

```

ANSWER

```

=====
ID  NAME      SALARY    SUM1      SUM2      SUM3      SUM4
--  -
10  Sanders    18357.50  92701.30  92701.30  92701.30  92701.30
20  Pernal     18171.25  92701.30  92701.30  92701.30  92701.30
30  Marenghi   17506.75  92701.30  92701.30  92701.30  92701.30
40  O'Brien    18006.00  92701.30  92701.30  92701.30  92701.30
50  Hanes      20659.80  92701.30  92701.30  92701.30  92701.30

```

Notes

- **OVER()** is same as having **ORDER BY** of field that is same for all matching rows
- All of the above aggregations are equivalent

ORDER BY Usage

```

SELECT    DEPT    , NAME
          , SALARY
          , SUM(SALARY) OVER(ORDER BY DEPT)           AS SUM1
          , SUM(SALARY) OVER(ORDER BY DEPT DESC)      AS SUM2
          , SUM(SALARY) OVER(ORDER BY DEPT, NAME)     AS SUM3
          , SUM(SALARY) OVER(ORDER BY DEPT DESC, NAME DESC) AS SUM4
          , COUNT(*) OVER(ORDER BY DEPT)             AS ROW1
          , COUNT(*) OVER(ORDER BY DEPT, NAME)       AS ROW2
FROM      STAFF
WHERE     ID < 60
ORDER BY  DEPT
          , NAME;

```

ANSWER

```

=====
DEPT NAME      SALARY    SUM1      SUM2      SUM3      SUM4      ROW1 ROW2
-----
  15 Hanes      20659.80  20659.80  92701.30  20659.80  92701.30    1    1
  20 Pernal      18171.25   57188.55  72041.50  38831.05  72041.50    3    2
  20 Sanders     18357.50   57188.55  72041.50  57188.55  53870.25    3    3
  38 Marengchi   17506.75   92701.30  35512.75  74695.30  35512.75    5    4
  38 O'Brien    18006.00   92701.30  35512.75  92701.30  18006.00    5    5

```

Notes

- Provides set of values to aggregate on. Each distinct value gets a new result
- Gives a direction for aggregation (i.e. ASC or DESC)

ROWS Usage

```

SELECT  DEPT
        ,NAME
        ,YEARS
        ,SMALLINT(SUM(YEARS) OVER(ORDER BY DEPT))          AS D
        ,SMALLINT(SUM(YEARS) OVER(ORDER BY DEPT, NAME))     AS DN
        ,SMALLINT(SUM(YEARS) OVER(ORDER BY DEPT, NAME
                                ROWS      UNBOUNDED PRECEDING))AS DNU
        ,SMALLINT(SUM(YEARS) OVER(ORDER BY DEPT, NAME
                                ROWS      3 PRECEDING))      AS DN3
FROM      STAFF
WHERE     ID          < 100
AND       YEARS IS NOT NULL
ORDER BY  DEPT
        ,NAME;

```

ANSWER

```

=====
DEPT NAME      YEARS D  DN  DNU DN3
-----
    15 Hanes      10 17 10   10  10
    15 Rothman     7 17 17   17  17
    20 Pernal      8 32 25   25  25
    20 Sanders     7 32 32   32  32
    38 Marenghi    5 43 37   37  27
    38 O'Brien     6 43 43   43  26
    42 Koonitz     6 49 49   49  24

```

Notes

- **ROWS** phrase limits aggregation to subset of matching rows
- Options are: **UNBOUNDED PRECEDING**
 Integer PRECEDING
 CURRENT ROW
 BETWEEN

ROWS Usage - with BETWEEN and ORDER BY (1 of 2)

```

SELECT    ID
          ,NAME
          ,SMALLINT(SUM(ID) OVER(ORDER BY ID ASC
                                ROWS BETWEEN 1 PRECEDING
                                      AND CURRENT ROW)) AS APC
          ,SMALLINT(SUM(ID) OVER(ORDER BY ID ASC
                                ROWS BETWEEN CURRENT ROW
                                      AND 1 FOLLOWING)) AS ACF
          ,SMALLINT(SUM(ID) OVER(ORDER BY ID DESC
                                ROWS BETWEEN 1 PRECEDING
                                      AND CURRENT ROW)) AS DPC
          ,SMALLINT(SUM(ID) OVER(ORDER BY ID DESC
                                ROWS BETWEEN CURRENT ROW
                                      AND 1 FOLLOWING)) AS DCF

```

```

FROM      STAFF
WHERE     ID          < 50
        AND  YEARS IS NOT NULL
ORDER BY  ID;

```

ANSWER					
ID	NAME	APC	ACF	DPC	DCF
10	Sanders	10	30	30	10
20	Pernal	30	50	50	30
30	Marenghi	50	70	70	50
40	O'Brien	70	40	40	70

Notes

- **BETWEEN** answer depends on **ORDER BY** sequence

ROWS Usage - with BETWEEN and ORDER BY (2 of 2)

ASC ID (10,20,30,40)

READ ROWS, LEFT to RIGHT	1ST-ROW	2ND-ROW	3RD-ROW	4TH-ROW
=====	=====	=====	=====	=====
1 PRECEDING to CURRENT ROW	10=10	10+20=30	20+30=40	30+40=70
CURRENT ROW to 1 FOLLOWING	10+20=30	20+30=50	30+40=70	40 =40

DESC ID (40,30,20,10)

READ ROWS, RIGHT to LEFT	1ST-ROW	2ND-ROW	3RD-ROW	4TH-ROW
=====	=====	=====	=====	=====
1 PRECEDING to CURRENT ROW	20+10=30	30+20=50	40+30=70	40 =40
CURRENT ROW to 1 FOLLOWING	10 =10	20+10=30	30+20=50	40+30=70

Notes

- **Preceding row is always on LEFT of current row**
Following row is always on RIGHT of current row

```

SELECT  ID ,NAME
        ,SMALLINT(SUM(ID) OVER(ORDER BY ID ASC
                               ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)) AS APC
        ,SMALLINT(SUM(ID) OVER(ORDER BY ID ASC
                               ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING)) AS ACF
        ,SMALLINT(SUM(ID) OVER(ORDER BY ID DESC
                               ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)) AS DPC
        ,SMALLINT(SUM(ID) OVER(ORDER BY ID DESC
                               ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING)) AS DCF
FROM      STAFF
WHERE     ID          < 50
        AND     YEARS IS NOT NULL
ORDER BY ID;
```

PARTITION Usage

```

SELECT  DEPT , NAME , YEARS AS YR
        , SMALLINT(SUM(YEARS) OVER(ORDER      BY DEPT))           AS X
        , SMALLINT(SUM(YEARS) OVER(ORDER      BY DEPT
                                     ROWS BETWEEN 1 PRECEDING
                                     AND 1 FOLLOWING)) AS Y
        , SMALLINT(SUM(YEARS) OVER(PARTITION BY DEPT))           AS P
        , SMALLINT(SUM(YEARS) OVER(PARTITION BY DEPT
                                     ORDER      BY DEPT))           AS P0
        , SMALLINT(SUM(YEARS) OVER(PARTITION BY DEPT
                                     ORDER      BY DEPT
                                     ROWS 1 PRECEDING))           AS P1

```

```

FROM      STAFF
WHERE     ID BETWEEN 40 AND 120
        AND YEARS IS NOT NULL
ORDER BY  DEPT
        , NAME;

```

ANSWER

```

=====
DEPT   NAME      YR  X  Y  P  P0 P1
-----
    15  Hanes    10 22 15 22 22 10
    15  Ngan      5 22 22 22 22 15
    15  Rothman   7 22 18 22 22 12
    38  O'Brien   6 28 19  6  6  6
    42  Koonitz    6 41 19 13 13  6
    42  Plotz      7 41 13 13 13 13

```

Notes

- **PARTITION** phrase limits aggregation to set of common values

Scalar Functions

Sample View DDL - Scalar functions

```
CREATE VIEW SCALAR (D1,F1,S1,C1,V1,TS1,DT1,TM1,TC1) AS
WITH TEMP1 (N1, C1, T1) AS
(VALUES (-2.4, 'ABCDEF', '1996-04-22-23.58.58.123456')
      , (+0.0, 'ABCD ', '1996-08-15-15.15.15.151515')
      , (+1.8, 'AB ', '0001-01-01-00.00.00.000000'))
SELECT DECIMAL(N1,3,1)
      ,DOUBLE(N1)
      ,SMALLINT(N1)
      ,CHAR(C1,6)
      ,VARCHAR(RTRIM(C1),6)
      ,TIMESTAMP(T1)
      ,DATE(T1)
      ,TIME(T1)
      ,CHAR(T1)
FROM   TEMP1;
```

SCALAR view, contents (3 rows)

D1	F1	S1	C1	V1	TS1
-2.4	-2.4e+000	-2	ABCDEF	ABCDEF	1996-04-22-23.58.58.123456
0.0	0.0e+000	0	ABCD	ABCD	1996-08-15-15.15.15.151515
1.8	1.8e+000	1	AB	AB	0001-01-01-00.00.00.000000

DT1	TM1	TC1
04/22/1996	23:58:58	1996-04-22-23.58.58.123456
08/15/1996	15:15:15	1996-08-15-15.15.15.151515
01/01/0001	00:00:00	0001-01-01-00.00.00.000000

BIGINT Function

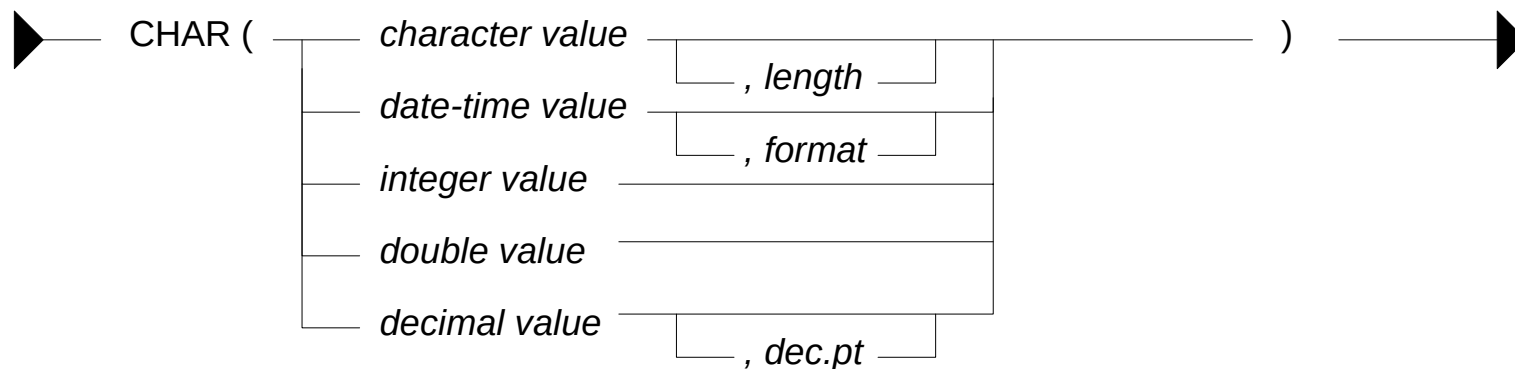
```
WITH TEMP (BIG) AS
(VALUES BIGINT(1)
 UNION ALL
 SELECT BIG * 256
 FROM   TEMP
 WHERE  BIG < 1E16
)
SELECT BIG
FROM   TEMP;
```

```
ANSWER
=====
BIG
-----
1
256
65536
16777216
4294967296
1099511627776
281474976710656
72057594037927936
```

Notes

- Converts number to big-integer

CHAR Function (1 of 2)



CHAR Function Examples

SELECT C1	ANSWER
, CHAR(C1, 3)	=====
, S1	C1 2 S1 4 D1 6 7
, CHAR(S1)	-----
, D1	ABCDEF ABC -2 -2 -2.4 -02.4 -02\$4
, CHAR(D1)	ABCD ABC 0 0 0.0 00.0 00\$0
, CHAR(D1, '\$')	AB AB 1 1 1.8 01.8 01\$8
FROM SCALAR;	

Notes

- Converts numbers to character
- Converts date-time values to character
- Converts character-expression to character (fixed length)

CHAR Function (2 of 2)

Date Input

```
SELECT CHAR(HIREDATE, ISO)
       , CHAR(HIREDATE, USA)
       , CHAR(HIREDATE, EUR)
FROM   EMPLOYEE
WHERE  LASTNAME < 'C'
ORDER BY 2;
```

ANSWER

```
=====
1           2           3
-----
1972-02-12  02/12/1972  12.02.1972
1966-03-03  03/03/1966  03.03.1966
```

DIGITS vs. CHAR

```
SELECT D1
       , CHAR(D1)      AS CD1
       , DIGITS(D1)    AS DD1
FROM   SCALAR;
```

ANSWER

```
=====
D1      CD1      DD1
-----
-2.4    -02.4    024
0.0     00.0     000
1.8     01.8     018
```

COALESCE Function

```
SELECT    ID
          , COMM
          , COALESCE(COMM, 0)
FROM      STAFF
WHERE     ID < 30
ORDER BY  ID;
```

```
ANSWER
=====
ID   COMM      3
--   -
10   -         0.00
20   612.45    612.45
```

NOT NULL field returning null value

```
SELECT COUNT(*)           AS #ROWS
       , MIN(ID)          AS MIN_ID
       , COALESCE(MIN(ID), -1) AS CCC_ID
FROM    STAFF
WHERE   ID < 5;
```

```
ANSWER
=====
#ROWS MIN_ID CCC_ID
-----
      0      -    -1
```

Notes

- Returns first non-null value in list
- Same as VALUE function
- Useful in outer joins

CONCAT Function (1 of 2)

```

SELECT      'A' || 'B'
            , 'A' CONCAT 'B'
            , CONCAT('A', 'B')
            , 'A' || 'B' || 'C'
            , CONCAT(CONCAT('A', 'B'), 'C')
FROM        STAFF
WHERE       ID = 10;

```

```

ANSWER
=====
  1    2    3    4    5
---  ---  ---  ---  ---
AB   AB   AB   ABC  ABC

```

Notes

- Joins two strings together

Syntax Styles

- "AB" || "CD"
- "AB" CONCAT "CD"
- CONCAT("AB","CD")

CONCAT Function (2 of 2)

CONCAT used with ORDER BY - wrong output sequence

```
WITH TEMP1 (COL1, COL2) AS
(VALUES      ('A' , 'YYY')
          , ('AE', '000')
          , ('AE', 'YYY')
)
SELECT      COL1
           , COL2
           , COL1 CONCAT COL2 AS COL3
FROM        TEMP1
ORDER BY    COL3;
```

ANSWER

```
=====
COL1 COL2 COL3
----
```

AE	000	AE000
AE	YYY	AEYYY
A	YYY	AYYY

CONCAT used with ORDER BY - correct output sequence

```
WITH TEMP1 (COL1, COL2) AS
(VALUES      ('A' , 'YYY')
          , ('AE', '000')
          , ('AE', 'YYY')
)
SELECT      COL1
           , COL2
           , CHAR(COL1,2) CONCAT
             CHAR(COL2,3) AS COL3
FROM        TEMP1
ORDER BY    COL3;
```

ANSWER

```
=====
COL1 COL2 COL3
----
```

A	YYY	A YYY
AE	000	AE000
AE	YYY	AEYYY

DAYOFWEEK_ISO Function

```

WITH
TEMP1 (N) AS
  (VALUES (0)
   UNION ALL
   SELECT N+1
   FROM   TEMP1
   WHERE  n < 9),
TEMP2 (DT1) AS
  (VALUES (DATE('1999-12-26'))
   , (DATE('2000-12-24'))),
TEMP3 (DT2) AS
  (SELECT DT1 + N DAYS
   FROM   TEMP1
   ,TEMP2)
SELECT   CHAR(DT2, ISO)           AS DATE
        , SUBSTR(DAYNAME(DT2), 1, 3) AS DAY
        , WEEK(DT2)                AS W
        , DAYOFWEEK(DT2)           AS D
        , WEEK_ISO(DT2)            AS WI
        , DAYOFWEEK_ISO(DT2)       AS I
FROM     TEMP3
ORDER BY 1;

```

```

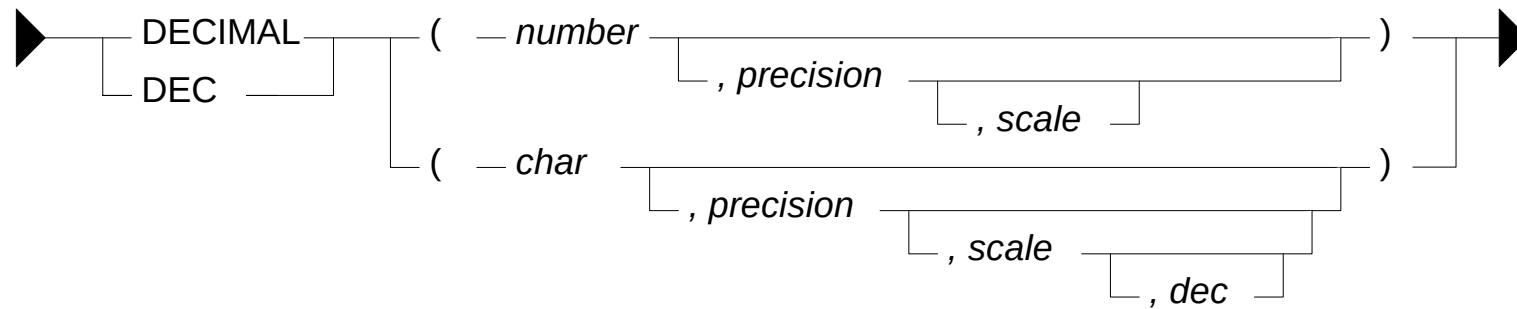
ANSWER
=====
DATE          DAY W  D WI I
-----
1999-12-28 Tue 53 3 52 2
1999-12-29 Wed 53 4 52 3
1999-12-30 Thu 53 5 52 4
1999-12-31 Fri 53 6 52 5
2000-01-01 Sat  1 7 52 6
2000-01-02 Sun  2 1 52 7
2000-01-03 Mon  2 2  1 1
2000-01-04 Tue  2 3  1 2
2000-12-24 Sun 53 1 51 7
2000-12-25 Mon 53 2 52 1
2000-12-26 Tue 53 3 52 2
2000-12-27 Wed 53 4 52 3
2000-12-28 Thu 53 5 52 4
2000-12-29 Fri 53 6 52 5
2000-12-30 Sat 53 7 52 6
2000-12-31 Sun 54 1 52 7
2001-01-01 Mon  1 2  1 1
2001-01-02 Tue  1 3  1 2

```

Warning

- Weeks are not fixed items - use with care

DECIMAL Function



Examples

```
WITH TEMP1(N1,N2,C1,C2) AS
(VALUES
    (123
    ,1E2
    , '123.4'
    , '567$8' ))
SELECT DEC(N1,3) AS DEC1
    ,DEC(N2,4,1) AS DEC2
    ,DEC(C1,4,1) AS DEC3
    ,DEC(C2,4,1, '$') AS DEC4
FROM TEMP1;
```

ANSWER			
=====			
DEC1	DEC2	DEC3	DEC4
-----	-----	-----	-----
123.	100.0	123.4	567.8

GENERATE_UNIQUE Function

```
SELECT    ID
          ,GENERATE_UNIQUE( )           AS UNIQUE_VAL#1
          ,DEC(HEX(GENERATE_UNIQUE( )),26) AS UNIQUE_VAL#2
FROM      STAFF
WHERE     ID < 50
ORDER BY  ID;
```

	ANSWER	
	=====	=====
	ID UNIQUE_VAL#1	UNIQUE_VAL#2
	-----	-----
NOTE: 2ND FIELD =>	10	20000901131648990521000000.
IS UNPRINTABLE. =>	20	20000901131648990615000000.
	30	20000901131648990642000000.
	40	20000901131648990669000000.

LOCATE Function

LOCATE (*find-string* , *look-in-string* [, *start-pos.*])

Examples

```

SELECT C1
      , LOCATE( 'D' , C1)
      , LOCATE( 'D' , C1, 2)
      , LOCATE( 'EF' , C1)
      , LOCATE( 'A' , C1, 2)
FROM   SCALAR;
  
```

ANSWER

C1	2	3	4	5
ABCDEF	4	4	5	0
ABCD	4	4	0	0
AB	0	0	0	0

Notes

- Returns location of one string within another
- Returns zero if no value found

Compared to POSSTR function

- Return same results for same input
- Input parameters are reversed
- POSSTR is SYSIBM function

RAISE_ERROR Function

► `RAISE_ERROR ( sqlstate , error-message )` ►

Example

```
SELECT S1
      , CASE
          WHEN S1 < 1 THEN S1
          ELSE RAISE_ERROR( '80001' , C1 )
        END AS S2
FROM   SCALAR;
```

```
ANSWER
=====
S1      S2
-----
      -2      -2
       0       0
SQLSTATE=80001
```

Notes

- Causes the SQL to stop and return a user-defined error
- See SQL Reference for (many) usage restrictions

RAND Function (1 of 4)

```
WITH TEMP (NUM, RAN) AS
(VALUES (INT(1)
        ,RAND(2))
 UNION ALL
 SELECT  NUM + 1
        ,RAND()
 FROM    TEMP
 WHERE   NUM < 100000
 )
```

SELECT	COUNT(*)	AS #ROWS	==>	100000
	,COUNT(DISTINCT RAN)	AS #VALUES	==>	31242
	,DEC(AVG(RAN),7,6)	AS AVG_RAN	==>	0.499838
	,DEC(STDDEV(RAN),7,6)	AS STD_DEV		0.288706
	,DEC(MIN(RAN),7,6)	AS MIN_RAN		0.000000
	,DEC(MAX(RAN),7,6)	AS MAX_RAN		1.000000
	,DEC(MAX(RAN),7,6) -			
	DEC(MIN(RAN),7,6)	AS RANGE		1.000000
	,DEC(VAR(RAN),7,6)	AS VARIANCE		0.083351
FROM	TEMP;			

Warnings

- Beware using RAND in predicates
- Returns different results in different flavors of DB2

RAND Function (2 of 4)

Make reproducible random numbers (use seed)

```
SELECT  DEPTNO  AS DNO
        ,RAND(0) AS RAN
FROM    DEPARTMENT
WHERE   DEPTNO < 'E'
ORDER BY 1;
```

```
ANSWER
=====
DNO  RAN
---  -
A00  +1.15970336008789E-003
B01  +2.35572374645222E-001
C01  +6.48152104251228E-001
D01  +7.43736075930052E-002
D11  +2.70241401409955E-001
D21  +3.60026856288339E-001
```

Notes

- Returns floating-point pseudo-random values in range zero to one
- Seed can be provided to get reproducible values
- Maximum of 32K distinct values generated

RAND Function (3 of 4)

Use RAND to make sample data

```
WITH TEMP1 (COL1, COL2, COL3) AS
(VALUES (0
        , SMALLINT(RAND(2)*35)*10
        , DECIMAL(RAND()*10000,7,2))
UNION ALL
SELECT  COL1 + 1
        , SMALLINT(RAND()*35)*10
        , DECIMAL(RAND()*10000,7,2)
FROM    TEMP1
WHERE   COL1 + 1 < 10
)
SELECT *
FROM    TEMP1;
```

ANSWER

```
=====
COL1   COL2   COL3
----
0      0      9342.32
1      250     8916.28
2      310     5430.76
3      150     5996.88
4      110     8066.34
5      50      5589.77
6      130     8602.86
7      340     184.94
8      310     5441.14
9      70      9267.55
```

RAND Function (4 of 4)

Randomly select 10% of matching rows

```
SELECT    ID
          , NAME
FROM      STAFF
WHERE     RAND() < 0.1
ORDER BY ID;
```

```
ANSWER
=====
ID  NAME
---
140 Fraye
190 Sneider
290 Quill
```

Select five random rows

```
SELECT    ID
          , NAME
FROM      (SELECT S.*
              ,ROW_NUMBER() OVER(ORDER BY RAND()) AS R
            FROM    STAFF S
            )AS XXX
WHERE     R <= 5
ORDER BY ID;
```

```
ANSWER
=====
ID  NAME
---
10  Sanders
30  Marenghi
190 Sneider
270 Lea
280 Wilson
```

Warning

- Beware using RAND in predicates

SUBSTR Function (1 of 2)

► SUBSTR (*string* , *start* , *length*) ►

Example, gets error when LENGTH too long

```
WITH TEMP1 (LEN, DAT1) AS
(VALUES      ( 6, '123456789' )
             , ( 4, '12345'   )
             , (16, '123'     )
)
SELECT      LEN
           , DAT1
           , LENGTH(DAT1) AS LDAT
           , SUBSTR(DAT1,1,LEN) AS SUBDAT
FROM        TEMP1;
```

```
ANSWER
=====
LEN DAT1          LDAT SUBDAT
--- -
  6 123456789      9 123456
  4 12345          5 1234
<error>
```

Notes

- Returns part of a string
- If no length, output is from START to end of string
- If LENGTH longer than string, error results

SUBSTR Function (2 of 2)

Avoid LENGTH error using CASE

```

WITH TEMP1 (LEN, DAT1) AS
(VALUES      ( 6, '123456789')
      , ( 4, '12345' )
      , ( 16, '123' )
)
SELECT      LEN
           , DAT1
           , LENGTH(DAT1) AS LDAT
           , SUBSTR(DAT1, 1, CASE
                           WHEN LEN < LENGTH(DAT1) THEN LEN
                           ELSE LENGTH(DAT1)
                           END ) AS SUBDAT
FROM        TEMP1;

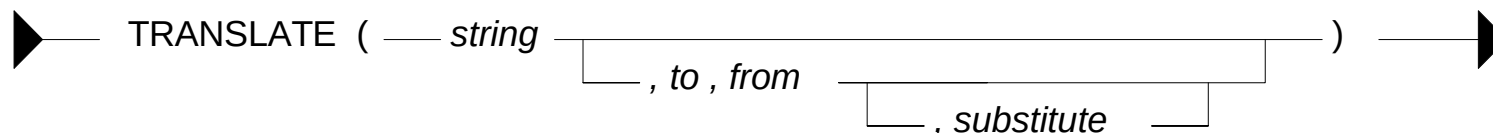
```

```

ANSWER
=====
LEN  DAT1          LDAT  SUBDAT
---  -
  6  123456789      9    123456
  4  12345          5    1234
 16  123            3    123

```

TRANSLATE Function



Examples

```
SELECT 'abcd'
      ,TRANSLATE('abcd')
      ,TRANSLATE('abcd',' ','a')
      ,TRANSLATE('abcd','A','A')
      ,TRANSLATE('abcd','A','a')
      ,TRANSLATE('abcd','A','ab')
      ,TRANSLATE('abcd','A','ab',' ')
      ,TRANSLATE('abcd','A','ab','z')
FROM   STAFF
WHERE  ID = 10;
```

```
SELECT DT1
      ,YEAR(DT1) AS YR
      ,WEEK(DT1) AS WK
FROM   SCALAR;
```

ANS. NOTES

```
==== =====
==> abcd No change
==> ABCD Make upper case
==> bcd 'a'=>' '
    abcd 'A'=>'A'
    Abcd 'a'=>'A'
    A cd 'a'=>'A','b'=>' '
    A cd 'a'=>'A','b'=>' '
    Azcd 'a'=>'A','b'=>'z'
```

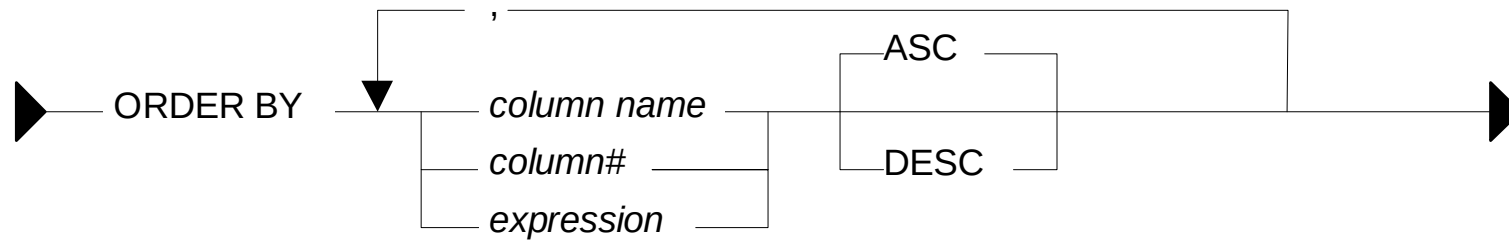
ANSWER

```
=====
DT1          YR      WK
-----
04/22/1996   1996    17
08/15/1996   1996    33
01/01/0001    1      1
```

Order By, Group By, and Having

ORDER BY

Syntax



Simple ORDER BY

```

SELECT  COL1
        , COL2
FROM    SEQ_DATA
ORDER BY COL1 ASC
        , COL2;
  
```

```

ANSWER
=====
COL1 COL2
---- ----
ab    xy
ac    XY
Ab    12
AB    xy
AB    XY
  
```

Sample View DDL

```

CREATE VIEW SEQ_DATA(COL1,COL2) AS VALUES
('ab','xy'),('AB','xy'),('ac','XY'),('AB','XY'),('Ab','12');
  
```

ORDER BY Examples (1 of 2)

Case Insensitive

```
SELECT    COL1
          ,COL2
FROM      SEQ_DATA
ORDER BY  TRANSLATE(COL1) ASC
          ,TRANSLATE(COL2) ASC
```

```
ANSWER
=====
COL1  COL2
----  ----
Ab    12
ab    xy
AB    XY
AB    xy
ac    XY
```

On non-displayed Column

```
SELECT    COL2
FROM      SEQ_DATA
ORDER BY  COL1
          ,COL2;
```

```
ANSWER
=====
      COL2
      ----
      xy
      XY
      12
      xy
      XY
```

ORDER BY Examples (2 of 2)

On second byte of first column

```
SELECT    COL1
          , COL2
FROM      SEQ_DATA
ORDER BY  SUBSTR(COL1,2) DESC
          , COL2
          , 1;
```

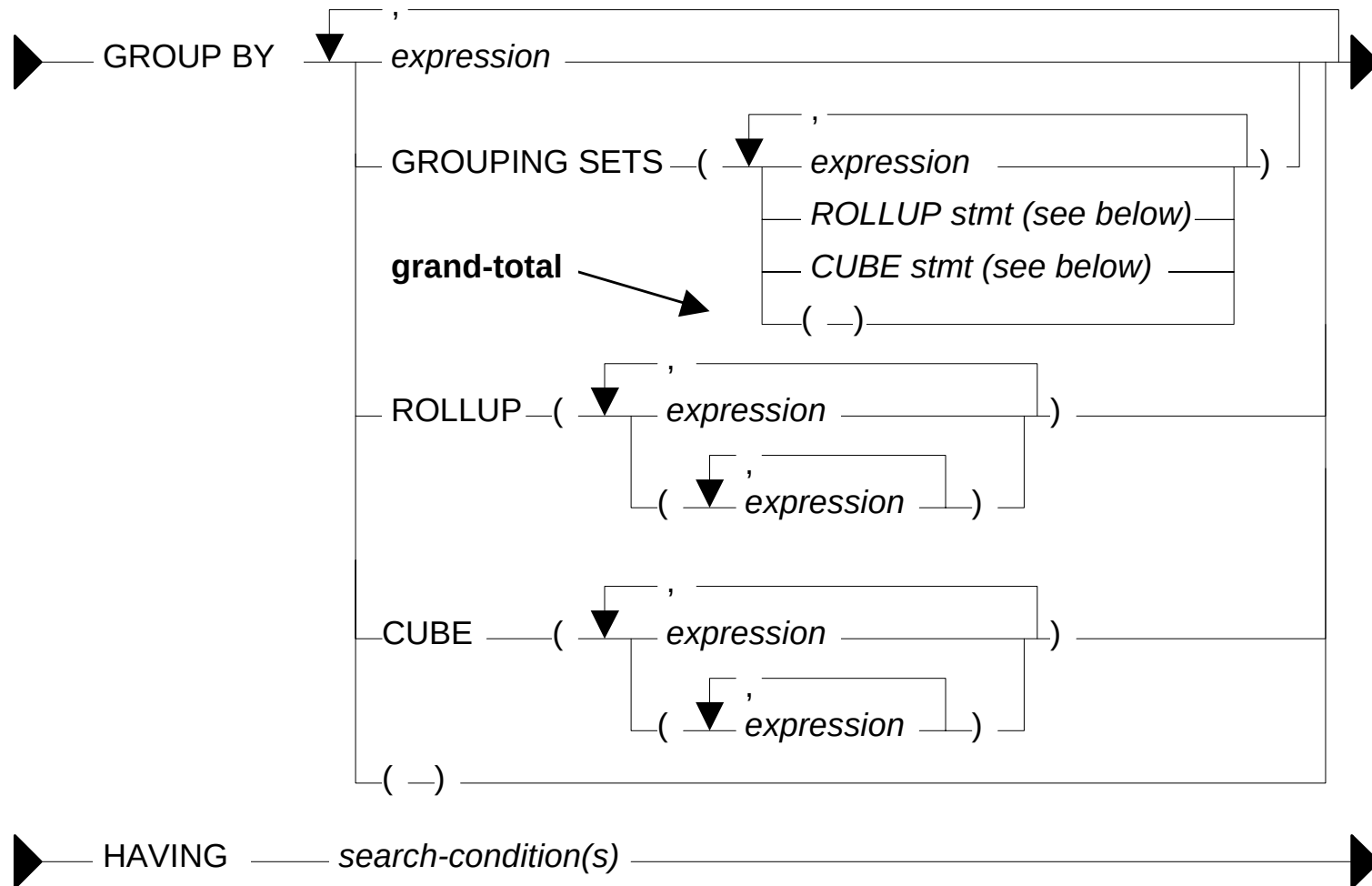
```
ANSWER
=====
COL1  COL2
----  ----
ac    XY
AB    xy
AB    XY
Ab    12
ab    xy
```

In bit-data sequence

```
SELECT    COL1
          , HEX(COL1) AS HEX1
          , COL2
          , HEX(COL2) AS HEX2
FROM      SEQ_DATA
ORDER BY  HEX(COL1)
          , HEX(COL2)
```

```
ANSWER
=====
COL1  HEX1  COL2  HEX2
----  ----  ----  ----
AB    4142  XY    5859
AB    4142  xy    7879
Ab    4162  12    3132
ab    6162  xy    7879
ac    6163  XY    5859
```

GROUP BY



Notes

- **GROUP BY** rolls up multiple rows into one

GROUP BY Sample Data

```
CREATE VIEW EMPLOYEE_VIEW AS
SELECT    SUBSTR(WORKDEPT,1,1) AS D1
          ,WORKDEPT           AS DEPT
          ,SEX                 AS SEX
          ,INTEGER(SALARY)     AS SALARY
FROM      EMPLOYEE
WHERE     WORKDEPT < 'D20';
COMMIT;
```

```
SELECT    *
FROM      EMPLOYEE_VIEW
ORDER BY  1,2,3,4;
```

```
ANSWER
=====
D1 DEPT SEX SALARY
-- ---- - - - -
A  A00  F   52750
A  A00  M   29250
A  A00  M   46500
B  B01  M   41250
C  C01  F   23800
C  C01  F   28420
C  C01  F   38250
D  D11  F   21340
D  D11  F   22250
D  D11  F   29840
D  D11  M   18270
D  D11  M   20450
D  D11  M   24680
D  D11  M   25280
D  D11  M   27740
D  D11  M   32250
```

Notes

- Observe, first three fields are non-unique

GROUP BY, Basic

```

SELECT    D1, DEPT, SEX
          ,SUM(SALARY)           AS SALARY
          ,SMALLINT(COUNT(*)) AS #ROWS
FROM      EMPLOYEE_VIEW
WHERE     DEPT <> 'ABC'
GROUP BY  D1, DEPT, SEX
HAVING    DEPT > 'A0'
          AND (SUM(SALARY) > 100
              OR  MIN(SALARY) > 10
              OR  COUNT(*) <> 22)
ORDER BY  D1, DEPT, SEX;

```

```

ANSWER
=====
D1 DEPT SEX SALARY #ROWS
-- ---- --- -
A  A00  F   52750    1
A  A00  M   75750    2
B  B01  M   41250    1
C  C01  F   90470    3
D  D11  F   73430    3
D  D11  M  148670    6

```

Notes

- Get one row per set of values in GROUP BY field list
- WHERE predicates applied before GROUP BY
- HAVING predicates applied during GROUP BY

GROUP BY, on non-displayed fields

```
SELECT    SEX
          ,SUM(SALARY)          AS SALARY
          ,SMALLINT(COUNT(*)) AS #ROWS
FROM      EMPLOYEE_VIEW
WHERE     SEX IN ('F','M')
GROUP BY  DEPT
          ,SEX
ORDER BY  SEX;
```

```
ANSWER
=====
SEX SALARY #ROWS
--- -
```

F	52750	1
F	90470	3
F	73430	3
M	75750	2
M	41250	1
M	148670	6

On derived field, not shown

```
SELECT    SUM(SALARY)          AS SALARY
          ,SMALLINT(COUNT(*)) AS #ROWS
FROM      EMPLOYEE_VIEW
WHERE     D1 <> 'X'
GROUP BY  SUBSTR(DEPT,3,1)
HAVING    COUNT(*) <> 99;
```

```
ANSWER
=====
SALARY #ROWS
-----
```

128500	3
353820	13

On derived field, shown

```
SELECT    SUBSTR(DEPT,3,1) AS WPART
          ,SUM(SALARY)     AS SALARY
          ,SMALLINT(COUNT(*)) AS #ROWS
FROM      EMPLOYEE_VIEW
GROUP BY  SUBSTR(DEPT,3,1)
ORDER BY  WPART DESC;
```

```
ANSWER
=====
WPART SALARY #ROWS
-----
```

1	353820	13
0	128500	3

GROUPING SETS

Parenthesis Usage

`GROUP BY GROUPING SETS ((A,B,C))` *is equivalent to* `GROUP BY A
 ,B
 ,C`

GROUP BY GROUPING SETS (A,B,C)	<i>is equivalent to</i>	GROUP BY A UNION ALL GROUP BY B UNION ALL GROUP BY C
--------------------------------	-------------------------	--

`GROUP BY GROUPING SETS (A, (B,C))` *is equivalent to* `GROUP BY A
UNION ALL
GROUP BY B
 , BY C`

Notes

- **GROUP BY GROUPING SETS** is used to get multiple groupings in one stmt
- **Parenthesis rules:**
 - Nested list of columns works as simple **GROUP BY**
 - Non-nested list of columns works separate **GROUP BY** stmts, separated by a **UNION**

Multiple GROUPING SETS

GROUP BY GROUPING SETS (A)	<i>is equivalent to</i>	GROUP BY A
, GROUPING SETS (B)		, B
, GROUPING SETS (C)		, C

GROUP BY GROUPING SETS (A)	<i>is equivalent to</i>	GROUP BY A
, GROUPING SETS ((B,C))		, B
		, C

GROUP BY GROUPING SETS (A)	<i>is equivalent to</i>	GROUP BY A
, GROUPING SETS (B,C)		, B
		UNION ALL
		GROUP BY A
		, C

Simple GROUP BY expression and GROUPING SETS combined

GROUP BY A	<i>is equivalent to</i>	GROUP BY A
, GROUPING SETS ((B,C))		, B
		, C

Notes

- Multiple GROUPING SETS in the same GROUP BY are combined as if they were single fields in a GROUP BY list

ROLLUP

```
GROUP BY GROUPING SETS ((A,B,C)
                        , (A,B)
                        , (A)
                        , ( ))
```

is equivalent to

```
GROUP BY A
        ,B
        ,C
UNION ALL
GROUP BY A
        ,B
UNION ALL
GROUP BY A
UNION ALL
grand-totl
```

is equivalent to

```
ROLLUP(A,B,C)
```

Notes

- **ROLLUP of "n" fields**

**Does GROUP BY on all fields listed, plus
GROUP BY subsets of the fields (from left), plus
Grand total of all fields**

Example

```
SELECT    DEPT
          ,SUM(SALARY)           AS SALARY
          ,SMALLINT(COUNT(*)) AS #ROWS
          ,GROUPING(DEPT)      AS FD
FROM      EMPLOYEE_VIEW
GROUP BY  ROLLUP(DEPT)
ORDER BY  DEPT;
```

```
ANSWER
=====
DEPT  SALARY  #ROWS  FD
----  -
A00   128500    3    0
B01    41250    1    0
C01    90470    3    0
D11   222100    9    0
-     482320   16    1
```

CUBE

```
GROUP BY GROUPING SETS ((A,B,C)
                        , (A,B)
                        , (A,C)
                        , (B,C)
                        , (A)
                        , (B)
                        , (C)
                        , ( ))
```

is equivalent to

```
GROUP BY A
        ,B
        ,C
UNION ALL
GROUP BY A
        ,B
UNION ALL
GROUP BY A
        ,C
UNION ALL
GROUP BY B
        ,C
UNION ALL
GROUP BY A
UNION ALL
GROUP BY B
UNION ALL
GROUP BY C
UNION ALL
grand-totl
```

is equivalent to

```
CUBE(A,B,C)
```

Notes

- CUBE of "n" fields

**Does GROUP BY on all fields listed, plus
GROUP BY on <all> subsets of the fields, plus
Grand total of all fields**

Cute Trick (1 of 2)

Basic GROUP BY

```

SELECT      D1                      AS D1
            ,DEPT                  AS DPT
            ,SEX                   AS SX
            ,INT(SUM(SALARY))      AS SAL
            ,SMALLINT(COUNT(*))   AS R
FROM        EMPLOYEE_VIEW
GROUP BY D1
            ,DEPT
            ,SEX
ORDER BY 1,2,3;

```

```

ANSWER
=====
D1 DPT SX   SAL   R
-- --- -- -
A  A00 F   52750  1
A  A00 M   75750  2
B  B01 M   41250  1
C  C01 F   90470  3
D  D11 F   73430  3
D  D11 M  148670  6

```

Sub-totals we want to get

```

DESIRED SUB-TOTALS
=====
D1, DEPT, and SEX.
D1 and DEPT.
D1 and SEX.
D1.
SEX.
Grand total.

```

```

EQUIVILENT TO
=====
GROUP BY GROUPING SETS ((D1,DEPT,SEX)
                        , (D1,DEPT)
                        , (D1,SEX)
                        , (D1)
                        , (SEX)
                        , ())

```

```

GROUP BY ROLLUP(D1,DEPT)
         ,ROLLUP(SEX)

```

Cute Trick (2 of 2)

```

SELECT      *
FROM        (SELECT      D1
                    ,DEPT
                    ,SEX
                    ,INT(SUM(SALARY))
                    ,SMALLINT(COUNT( * ))
                    ,SMALLINT(GROUPING(D1))
                    ,SMALLINT(GROUPING(DEPT))
                    ,SMALLINT(GROUPING(SEX))
FROM        EMPLOYEE_VIEW
GROUP BY    CUBE(D1,DEPT,SEX)
)AS XXX
WHERE      (G1,GD,GS) = (0,0,0)
OR        (G1,GD,GS) = (0,0,1)
OR        (G1,GD,GS) = (0,1,0)
OR        (G1,GD,GS) = (0,1,1)
OR        (G1,GD,GS) = (1,1,0)
OR        (G1,GD,GS) = (1,1,1)
ORDER BY  1,2,3;

```

Predicates used - Explanation

```

=====
(G1,GD,GS) = (0,0,0)    <==  D1, DEPT, SEX
(G1,GD,GS) = (0,0,1)    <==  D1, DEPT
(G1,GD,GS) = (0,1,0)    <==  D1, SEX
(G1,GD,GS) = (0,1,1)    <==  D1,
(G1,GD,GS) = (1,1,0)    <==  SEX,
(G1,GD,GS) = (1,1,1)    <==  grand total

```

```

=====
ANSWER
=====
D1 DPT SX SAL #R G1 GD GS
-- -- - - - - -
A A00 F 52750 1 0 0 0
A A00 M 75750 2 0 0 0
A A00 - 128500 3 0 0 1
A - F 52750 1 0 1 0
A - M 75750 2 0 1 0
A - - 128500 3 0 1 1
B B01 M 41250 1 0 0 0
B B01 - 41250 1 0 0 1
B - M 41250 1 0 1 0
B - - 41250 1 0 1 1
C C01 F 90470 3 0 0 0
C C01 - 90470 3 0 0 1
C - F 90470 3 0 1 0
C - - 90470 3 0 1 1
D D11 F 73430 3 0 0 0
D D11 M 148670 6 0 0 0
D D11 - 222100 9 0 0 1
D - F 73430 3 0 1 0
D - M 148670 6 0 1 0
D - - 222100 9 0 1 1
- - F 216650 7 1 1 0
- - M 265670 9 1 1 0
- - - 482320 16 1 1 1

```

Joins

Sample Views, Join Examples

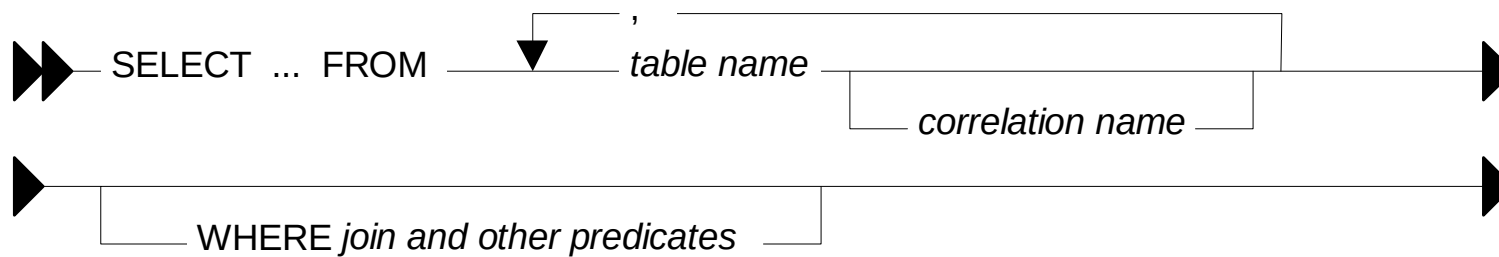
```
CREATE VIEW STAFF_V1 AS
SELECT ID, NAME
FROM   STAFF
WHERE  ID BEWTEEN 10 AND 30;
```

```
CREATE VIEW STAFF_V2 AS
SELECT ID, JOB
FROM   STAFF
WHERE  ID BETWEEN 20 AND 50;
```

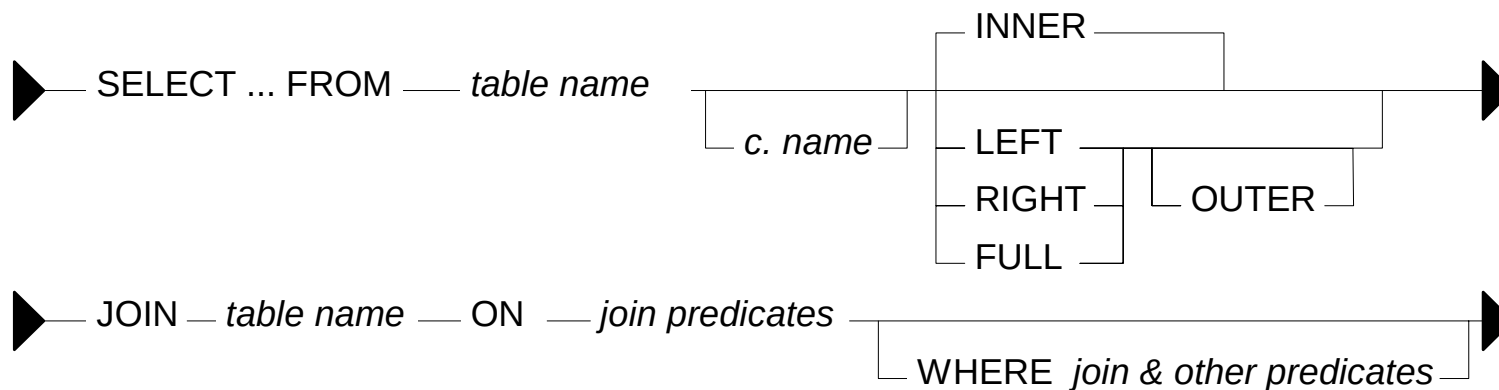
STAFF_V1		STAFF_V2	
ID	NAME	ID	JOB
10	Sanders	20	Sales
20	Pernal	30	Mgr
30	Marenghi	40	Sales
		50	Mgr

Join Syntax

Style #1



Style #2



Sample Joins, Style #1

Two Table

```
SELECT    V1.ID
          ,V1.NAME
          ,V2.JOB
FROM      STAFF_V1 V1
          ,STAFF_V2 V2
WHERE     V1.ID = V2.ID
ORDER BY V1.ID;
```

```
JOIN ANSWER
=====
ID NAME      JOB
-- - - - - -
20 Pernal    Sales
30 Marenghi  Mgr
```

Three Table

```
SELECT    V1.ID
          ,V2.JOB
          ,V3.NAME
FROM      STAFF_V1 V1
          ,STAFF_V2 V2
          ,STAFF_V1 V3
WHERE     V1.ID = V2.ID
        AND V2.ID = V3.ID
        AND V3.NAME LIKE 'M%'
ORDER BY V1.NAME;
```

```
JOIN ANSWER
=====
ID JOB NAME
-- - - - - -
30 Mgr Marenghi
```

Sample Joins, Style #2

Two Table

```
SELECT    V1.ID
          ,V1.NAME
          ,V2.JOB
FROM      STAFF_V1 V1
INNER JOIN
          STAFF_V2 V2
ON        V1.ID = V2.ID
ORDER BY V1.ID;
```

```
JOIN ANSWER
=====
ID NAME      JOB
-- - - - - -
20 Pernal    Sales
30 Marenghi  Mgr
```

Three Table

```
SELECT    V1.ID
          ,V2.JOB
          ,V3.NAME
FROM      STAFF_V1 V1
JOIN
          STAFF_V2 V2
ON        V1.ID = V2.ID
JOIN
          STAFF_V3 V3
ON        V2.ID = V3.ID
WHERE     V3.NAME LIKE 'M%'
ORDER BY V1.NAME;
```

```
JOIN ANSWER
=====
ID JOB NAME
-- - - - - -
30 Mgr Marenghi
```

Inner Join

STAFF_V1 +-----+ ID NAME -- ----- 10 Sanders 20 Pernal 30 Marenghi +-----+	STAFF_V2 +-----+ ID JOB -- ----- 20 Sales 30 Mgr 40 Sales 50 Mgr +-----+	=====>	INNER-JOIN ANSWER ===== ID NAME ID JOB -- ----- -- ----- 20 Pernal 20 Sales 30 Marenghi 30 Mgr
--	---	--------	--

SQL

```

SELECT      *
FROM        STAFF_V1 V1
            ,STAFF_V2 V2
WHERE       V1.ID = V2.ID
ORDER BY   V1.ID;
```

```

SELECT      *
FROM        STAFF_V1 V1
INNER JOIN
            STAFF_V2 V2
ON          V1.ID = V2.ID
ORDER BY   V1.ID;
```

Cartesian Product

STAFF_V1	
ID	NAME
10	Sanders
20	Pernal
30	Marenghi

STAFF_V2	
ID	JOB
20	Sales
30	Mgr
40	Sales
50	Mgr

=====>

CARTESIAN-PRODUCT			
ID	NAME	ID	JOB
10	Sanders	20	Sales
10	Sanders	30	Mgr
10	Sanders	40	Sales
10	Sanders	50	Mgr
20	Pernal	20	Sales
20	Pernal	30	Mgr
20	Pernal	40	Sales
20	Pernal	50	Mgr
30	Marenghi	20	Sales
30	Marenghi	30	Mgr
30	Marenghi	40	Sales
30	Marenghi	50	Mgr

```

SELECT      *
FROM        STAFF_V1 V1
INNER JOIN
           STAFF_V2 V2
ON          'A' <> 'B'
ORDER BY V1.ID
        ,V2.ID;

```

Partial Cartesian Product (1 of 2)

Simple Join

```

SELECT    V2A.ID
          ,V2A.JOB
          ,V2B.ID
FROM      STAFF_V2 V2A
          ,STAFF_V2 V2B
WHERE     V2A.JOB = V2B.JOB
        AND V2A.ID < 40
ORDER BY  V2A.ID
          ,V2B.ID;

```

```

ANSWER
=====
ID JOB   ID
-- ---- --
20 Sales 20
20 Sales 40
30 Mgr   30
30 Mgr   50

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Partial Cartesian Product (2 of 2)

Join with GROUP BY

```
SELECT    V2.JOB
          ,COUNT(*) AS #ROWS
FROM      STAFF_V1 V1
          ,STAFF_V2 V2
GROUP BY  V2.JOB
ORDER BY  #ROWS
          ,V2.JOB;
```

ANSWER

```
=====
JOB      #ROWS
-----
Mgr       6
Sales     6
```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Warning

- Beware of JOINS with GROUP BY or DISTINCT logic.
There may be a Partial Cartesian Product

Left Outer Join (1 of 2)

STAFF_V1	STAFF_V2		LEFT-OUTER-JOIN ANSWER
+-----+	+-----+		=====
ID NAME	ID JOB		ID NAME ID JOB
-- -----	-- -----	=====>	-- -----
10 Sanders	20 Sales		10 Sanders - -
20 Pernal	30 Mgr		20 Pernal 20 Sales
30 Marenghi	40 Sales		30 Marenghi 30 Mgr
+-----+	50 Mgr		
	+-----+		

SQL, Style #2

```

SELECT      *
FROM        STAFF_V1 V1
LEFT OUTER JOIN
            STAFF_V2 V2
ON          V1.ID = V2.ID
ORDER BY V1.ID;
```

In English

- Select rows in 1st table
- Join to matching rows in 2nd table
- If no matching row, return null field

Left Outer Join (2 of 2)

Style #2

```
SELECT      *
FROM        STAFF_V1 V1
LEFT OUTER JOIN
            STAFF_V2 V2
ON          V1.ID = V2.ID
ORDER BY V1.ID;
```

Style #1

```
SELECT      V1.*
            ,V2.*
FROM        STAFF_V1 V1
            ,STAFF_V2 V2
WHERE       V1.ID = V2.ID
UNION
SELECT      V1.*
            ,CAST(NULL AS SMALLINT) AS ID
            ,CAST(NULL AS CHAR(5))  AS JOB
FROM        STAFF_V1 V1
WHERE       V1.ID NOT IN
            (SELECT ID FROM STAFF_V2)
ORDER BY 1,3;
```

<== This join gets all rows in STAFF_V1 that match rows in STAFF_V2.

<== This query gets all the rows in STAFF_V1 with no matching rows in STAFF_V2.

Right Outer Join (1 of 2)

STAFF_V1	STAFF_V2		RIGHT-OUTER-JOIN ANSWER
+-----+	+-----+		=====
ID NAME	ID JOB		ID NAME ID JOB
-- -----	-- -----	=====>	-- ----- -- -----
10 Sanders	20 Sales		20 Pernal 20 Sales
20 Pernal	30 Mgr		30 Marenghi 30 Mgr
30 Marenghi	40 Sales		- - 40 Sales
+-----+	50 Mgr		- - 50 Mgr
	+-----+		

SQL, Style #2

```

SELECT      *
FROM        STAFF_V1 V1
RIGHT OUTER JOIN
            STAFF_V2 V2
ON          V1.ID = V2.ID
ORDER BY V2.ID;
```

In English

- Select rows in 2nd table
- Join to matching rows in 1st table
- If no matching row, return null field

Right Outer SQL (2 of 2)

Style #2

```
SELECT      *
FROM        STAFF_V1 V1
RIGHT OUTER JOIN
            STAFF_V2 V2
ON          V1.ID = V2.ID
ORDER BY V2.ID;
```

Style #1

```
SELECT      V1.*
            ,V2.*
FROM        STAFF_V1 V1
            ,STAFF_V2 V2
WHERE       V1.ID = V2.ID
UNION
SELECT      CAST(NULL AS SMALLINT) AS ID
            ,CAST(NULL AS VARCHAR(9)) AS NAME
            ,V2.*
FROM        STAFF_V2 V2
WHERE       V2.ID NOT IN
            (SELECT ID FROM STAFF_V1)
ORDER BY 1,3;
```

ANSWER

```
=====
ID  NAME          ID  JOB
--  -
20  Pernal         20  Sales
30  Marengghi      30  Mgr
-   -             40  Sales
-   -             50  Mgr
```

Full Outer Join (1 of 2)

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

=====>

FULL-OUTER-JOIN ANSWER			
=====			
ID	NAME	ID	JOB
--	-----	--	-----
10	Sanders	-	-
20	Pernal	20	Sales
30	Marenghi	30	Mgr
-	-	40	Sales
-	-	50	Mgr

SQL, Style #2

```

SELECT      *
FROM        STAFF_V1 V1
FULL OUTER JOIN
            STAFF_V2 V2
ON          V1.ID = V2.ID
ORDER BY V1.ID
          ,V2.ID;
```

In English

- Select rows from both tables
- Join rows on match field(s).

Full Outer Join (2 of 2)

```
SELECT    V1.*
          ,V2.*
FROM      STAFF_V1 V1
          ,STAFF_V2 V2
WHERE     V1.ID = V2.ID
UNION
SELECT    V1.*
          ,CAST(NULL AS SMALLINT) AS ID
          ,CAST(NULL AS CHAR(5))  AS JOB
FROM      STAFF_V1 V1
WHERE     V1.ID NOT IN
          (SELECT ID FROM STAFF_V2)
UNION
SELECT    CAST(NULL AS SMALLINT)  AS ID
          ,CAST(NULL AS VARCHAR(9)) AS NAME
          ,V2.*
FROM      STAFF_V2 V2
WHERE     V2.ID NOT IN
          (SELECT ID FROM STAFF_V1)
ORDER BY 1,3;
```

COALESCE Function in Outer Join

```

SELECT    COALESCE(V1.ID,V2.ID) AS ID
          ,COALESCE(V1.NAME,'?') AS NAME
          ,V2.JOB
FROM      STAFF_V1 V1
FULL OUTER JOIN
          STAFF_V2 V2
ON        V1.ID = V2.ID
ORDER BY ID;

```

```

ANSWER
=====
ID NAME      JOB
-- -
10 Sanders   -
20 Pernal    Sales
30 Marenghi  Mgr
40 ?         Sales
50 ?         Mgr

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Notes

- Coalesce function returns first non-null value.

Full Outer Join, Match on Keys

```

SELECT      *
FROM        STAFF_V1 V1
FULL OUTER JOIN
            STAFF_V2 V2
ON          V1.ID = V2.ID
ORDER BY   V1.ID
           ,V2.ID;

```

ANSWER

```

=====
ID NAME      ID JOB
-- -
10 Sanders   -  -
20 Pernal    20 Sales
30 Marenghi  30 Mgr
-  -         40 Sales
-  -         50 Mgr

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Notes

- Match on keys, all rows in both tables

Full Outer Join, Match on Keys > 20

```

SELECT      *
FROM        STAFF_V1 V1
FULL OUTER JOIN
            STAFF_V2 V2
ON          V1.ID = V2.ID
AND         V1.ID > 20
ORDER BY   V1.ID
           ,V2.ID;

```

```

ANSWER
=====
ID NAME      ID JOB
-- - - - - -
10 Sanders   -  -
20 Pernal    -  -
30 Marenghi  30 Mgr
-  -         20 Sales
-  -         40 Sales
-  -         50 Mgr

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Important

- Adding the above "ON" predicate has meant more rows returned!
- ON predicates NEVER eliminate rows, they define the join.

Full Outer Join, Match on Keys (no rows match)

```

SELECT      *
FROM        STAFF_V1 V1
FULL OUTER JOIN
            STAFF_V2 V2
ON          V1.ID = V2.ID
AND         +1 = -1
ORDER BY   V1.ID
           ,V2.ID;

```

```

ANSWER
=====
ID NAME      ID JOB
-- - - - - -
10 Sanders   -  -
20 Pernal    -  -
30 Marenghi  -  -
-  -         20 Sales
-  -         30 Mgr
-  -         40 Sales
-  -         50 Mgr

```

Sample Data

STAFF_V1		STAFF_V2	
ID	NAME	ID	JOB
10	Sanders	20	Sales
20	Pernal	30	Mgr
30	Marenghi	40	Sales
		50	Mgr

Notes

- No rows match 2nd "ON" check, so all rows deemed non-matching
- Important - no rows are rejected from answer

Full Outer Join, Don't Match on Keys (no rows match)

```

SELECT      *
FROM        STAFF_V1 V1
FULL OUTER JOIN
            STAFF_V2 V2
ON          +1 = -1
ORDER BY   V1.ID
           ,V2.ID;

```

```

ANSWER
=====
ID NAME      ID JOB
-- -
10 Sanders   -  -
20 Pernal    -  -
30 Marenghi  -  -
- -          20 Sales
- -          30 Mgr
- -          40 Sales
- -          50 Mgr

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Full Outer Join, Don't Match on Keys (all rows match)

```

SELECT      *
FROM        STAFF_V1 V1
FULL OUTER JOIN
           STAFF_V2 V2
ON          +1 <> -1
ORDER BY V1.ID
         ,V2.ID;

```

```

ANSWER
=====
ID  NAME      ID  JOB
--  -
10  Sanders    20  Sales
10  Sanders    30  Mgr
10  Sanders    40  Sales
10  Sanders    50  Mgr
20  Pernal     20  Sales
20  Pernal     30  Mgr
20  Pernal     40  Sales
20  Pernal     50  Mgr
30  Marengchi  20  Sales
30  Marengchi  30  Mgr
30  Marengchi  40  Sales
30  Marengchi  50  Mgr

```

STAFF_V1		STAFF_V2	
ID	NAME	ID	JOB
10	Sanders	20	Sales
20	Pernal	30	Mgr
30	Marengchi	40	Sales
		50	Mgr

Full Outer Join - Turned into Inner Join by WHERE

```

SELECT      *
FROM        STAFF_V1 V1
FULL JOIN
           STAFF_V2 V2
ON          V1.ID = V2.ID
WHERE       V1.ID = V2.ID
ORDER BY 1, 3;

```

```

ANSWER
=====
ID NAME      ID JOB
-- -
20 Pernal    20 Sales
30 Marenghi  30 Mgr

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Notes

- "Full Join" = "Full Outer Join"
- "Where" applied after join is processed

Outer Join V1.ID < 30, Sample Data

STAFF_V1	STAFF_V2		ANSWER
+-----+	+-----+	OUTER-JOIN CRITERIA	=====
ID NAME	ID JOB	=====>	???, DEPENDS
-- -----	-- -----	V1.ID = V2.ID	
10 Sanders	20 Sales	V1.ID < 30	
20 Pernal	30 Mgr		
30 Marengi	40 Sales		
+-----+	50 Mgr		
	+-----+		

Notes

- Join the two test tables together using the above predicates
- There answer returned will depend on what checks are applied when

Outer Join V1.ID < 30, Check Applied in WHERE (after join)

```

SELECT      *
FROM        STAFF_V1 V1
FULL JOIN   STAFF_V2 V2
ON          V1.ID = V2.ID
WHERE       V1.ID < 30
ORDER BY 1, 3;

```

```

ANSWER
=====
ID NAME      ID JOB
-- -
10 Sanders   -  -
20 Pernal    20 Sales

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marengi	40 Sales
+-----+	50 Mgr
	+-----+

Outer Join V1.ID < 30, Check Applied in ON (during join)

```

SELECT      *
FROM        STAFF_V1 V1
FULL JOIN   STAFF_V2 V2
ON          V1.ID = V2.ID
AND        V1.ID < 30
ORDER BY 1,3;

```

```

ANSWER
=====
ID NAME      ID JOB
-- -
10 Sanders   -  -
20 Pernal    20 Sales
30 Marenghi  -  -
-  -         30 Mgr
-  -         40 Sales
-  -         50 Mgr

```

Sample Data

STAFF_V1		STAFF_V2	
ID	NAME	ID	JOB
10	Sanders	20	Sales
20	Pernal	30	Mgr
30	Marenghi	40	Sales
		50	Mgr

Outer Join V1.ID < 30, Check Applied in WHERE (before join)

```

SELECT      *
FROM        (SELECT *
              FROM    STAFF_V1
              WHERE    ID < 30) AS V1
OUTER JOIN  STAFF_V2 V2
ON          V1.ID = V2.ID
ORDER BY 1,3;

```

```

ANSWER
=====
ID NAME      ID JOB
-- - - - - -
10 Sanders   -  -
20 Pernal    20 Sales
-  -         30 Mgr
-  -         40 Sales
-  -         50 Mgr

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- - - - - -	-- - - - - -
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marengi	40 Sales
+-----+	50 Mgr
	+-----+

Outer Join V1.ID < 30, (Gives Wrong Answer)

```

SELECT      *
FROM        STAFF_V1 V1
OUTER JOIN
           STAFF_V2 V2
ON          V1.ID = V2.ID
WHERE       V1.ID < 30
           OR V1.ID IS NULL
ORDER BY 1,3;

```

```

ANSWER
=====
ID NAME      ID JOB
-- -
10 Sanders   -  -
20 Pernal    20 Sales
-  -         40 Sales
-  -         50 Mgr

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Outer Join V1.ID < 30, (Gives Wrong Answer)

```

SELECT      *
FROM        STAFF_V1 V1
OUTER JOIN
           STAFF_V2 V2
ON          V1.ID = V2.ID
WHERE       V1.ID < 30
           OR V1.ID = V2.ID
           OR V1.ID IS NULL
ORDER BY 1,3;

```

```

ANSWER
=====
ID NAME      ID JOB
-- - - - - -
10 Sanders   -  -
20 Pernal    20 Sales
30 Marenghi  30 Sales
-  -         40 Sales
-  -         50 Mgr

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Outer Join, Get non-matching rows only, (1 of 4)

```
STAFF_V1
+-----+
|ID|NAME  |
|--|-----|
|10|Sanders |
|20|Pernal  |
|30|Marenghi|
+-----+
```

```
STAFF_V2
+-----+
|ID|JOB    |
|--|-----|
|20|Sales   |
|30|Mgr     |
|40|Sales   |
|50|Mgr     |
+-----+
```

```
NON-MATCHING
OUTER-JOIN
=====>
```

```
ANSWER
=====
ID NAME      ID JOB
-- -
10 Sanders -  -
-  -      40 Sales
-  -      50 Mgr
```

Outer Join, Get non-matching rows only, (2 of 4)

```

SELECT    V1.*
          ,CAST(NULL AS SMALLINT) AS ID
          ,CAST(NULL AS CHAR(5))  AS JOB
FROM      STAFF_V1 V1
WHERE     V1.ID NOT IN
          (SELECT ID FROM STAFF_V2)

UNION

SELECT    CAST(NULL AS SMALLINT)  AS ID
          ,CAST(NULL AS VARCHAR(9)) AS NAME
          ,V2.*
FROM      STAFF_V2 V2
WHERE     V2.ID NOT IN
          (SELECT ID FROM STAFF_V1)
ORDER BY 1,3;

```

<== Get all the rows
in STAFF_V1 that
have no matching
row in STAFF_V2.

<== Get all the rows
in STAFF_V2 that
have no matching
row in STAFF_V1.

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- - - - - -	-- - - - - -
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Outer Join, Get non-matching rows only, (3 of 4)

```

SELECT      *
FROM        (SELECT V1.*      , 'V1' AS FLAG      FROM STAFF_V1 V1) AS V1
FULL OUTER JOIN
            (SELECT V2.*      , 'V2' AS FLAG      FROM STAFF_V2 V2) AS V2
ON          V1.ID = V2.ID
WHERE       V1.FLAG IS NULL
           OR V2.FLAG IS NULL
ORDER BY   V1.ID
           ,V2.ID;

```

ANSWER

```

=====
ID NAME      FLAG ID JOB      FLAG
-- - - - - - - - - - - - - - - -
10 Sanders V1      - -      -
- -      -      40 Sales V2
- -      -      50 Mgr   V2

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Outer Join, Get non-matching rows only, (4 of 4)

Use Temp Table (works if matching field allows nulls)

```

WITH
  V1 AS (SELECT V1.*      , 'V1' AS FLAG   FROM STAFF_V1 V1)
, V2 AS (SELECT V2.*      , 'V2' AS FLAG   FROM STAFF_V2 V2)
SELECT *
FROM   V1 V1
FULL OUTER JOIN
      V2 V2
ON     V1.ID = V2.ID
WHERE  V1.FLAG IS NULL
      OR V2.FLAG IS NULL
ORDER BY V1.ID
        ,V2.ID;

```

ANSWER

```

=====
ID NAME      FLAG ID JOB      FLAG
-- -
10 Sanders V1      - -      -
- -      -      40 Sales V2
- -      -      50 Mgr   V2

```

Check for Null (only works if matching field not null)

```

SELECT *
FROM   STAFF_V1 V1
FULL OUTER JOIN
      STAFF_V2 V2
ON     V1.ID = V2.ID
WHERE  V1.ID IS NULL
      OR V2.ID IS NULL
ORDER BY V1.ID
        ,V2.ID;

```

```

STAFF_V1      STAFF_V2
+-----+ +-----+
|ID|NAME      | |ID|JOB      |
|--|-----| |--|-----|
|10|Sanders | |20|Sales |
|20|Pernal  | |30|Mgr   |
|30|Marenghi| |40|Sales |
+-----+ |50|Mgr   |
+-----+

```

Outer Join, Get one Column

Done in FROM phrase of SQL

```
SELECT    V1.ID
          ,V1.NAME
          ,V2.JOB
FROM      STAFF_V1 V1
LEFT OUTER JOIN
          STAFF_V2 V2
ON        V1.ID = V2.ID
ORDER BY V1.ID;
```

ANSWER

```
=====
ID  NAME      JOB
--  -
10  Sanders    -
20  Pernal     Sales
30  Marenghi   Mgr
```

Done in SELECT phrase of SQL

```
SELECT    V1.ID
          ,V1.NAME
          ,(SELECT  V2.JOB
             FROM    STAFF_V2 V2
             WHERE   V1.ID = V2.ID)
FROM      STAFF_V1 V1
ORDER BY V1.ID;
```

STAFF_V1		STAFF_V2	
ID	NAME	ID	JOB
10	Sanders	20	Sales
20	Pernal	30	Mgr
30	Marenghi	40	Sales
		50	Mgr

Outer Join, Get two Columns

Done in FROM phrase of SQL

```
SELECT    V1.ID
          ,V1.NAME
          ,V2.JOB
          ,LENGTH(V2.JOB) AS J2
FROM      STAFF_V1 V1
LEFT OUTER JOIN
          STAFF_V2 V2
ON        V1.ID = V2.ID
ORDER BY  V1.ID;
```

ANSWER

```
=====
ID NAME      JOB    J2
-- -
10 Sanders   -        -
20 Pernal    Sales    5
30 Marenghi Mgr      5
```

Done in SELECT phrase of SQL

```
SELECT    V1.ID
          ,V1.NAME
          ,(SELECT  V2.JOB
             FROM    STAFF_V2 V2
             WHERE   V1.ID = V2.ID)
          ,(SELECT  LENGTH(V2.JOB) AS J2
             FROM    STAFF_V2 V2
             WHERE   V1.ID = V2.ID)
FROM      STAFF_V1 V1
ORDER BY  V1.ID;
```

```
STAFF_V1      STAFF_V2
+-----+ +-----+
|ID|NAME      | |ID|JOB      |
|--|-----| |--|-----|
|10|Sanders   | |20|Sales    |
|20|Pernal    | |30|Mgr      |
|30|Marenghi  | |40|Sales    |
+-----+ |50|Mgr      |
+-----+
```

Outer Join, and Running Totals

Using JOIN in SELECT

```
SELECT    V1.ID
          ,V1.NAME
          ,(SELECT  SUM(X1.ID)
             FROM    STAFF_V1 X1
             WHERE   X1.ID <= V1.ID
            )AS SUM_ID
FROM      STAFF_V1 V1
ORDER BY  V1.ID;
```

ANSWER

```
=====
ID NAME      SUM_ID
-- - - - - - - - -
10 Sanders    10
20 Pernal     30
30 Marenghi   60
```

Using OLAP Function

```
SELECT    V1.ID
          ,V1.NAME
          ,SUM(ID) OVER(ORDER BY ID) AS SUM_ID
FROM      STAFF_V1 V1
ORDER BY  V1.ID;
```

ANSWER

```
=====
ID NAME      SUM_ID
-- - - - - - - - -
10 Sanders    10
20 Pernal     30
30 Marenghi   60
```

Outer Join, With WHERE Filter, Wrong

```
STAFF_V1
+-----+
|ID|NAME  |
|--|-----|
|10|Sanders |
|20|Pernal  |
|30|Marenghi|
+-----+
```

```
STAFF_V2
+-----+
|ID|JOB   |
|--|-----|
|20|Sales  |
|30|Mgr    |
|40|Sales  |
|50|Mgr    |
+-----+
```

```
OUTER-JOIN CRITERIA
=====>
V1.ID      = V2.ID
V2.JOB LIKE 'S%'
```

```
ANSWER
=====
ID NAME      JOB
-- - - - - -
10 Sanders   -
20 Pernal    Sales
30 Marenghi  -
```

Wrong

```
SELECT    V1.ID
          ,V1.NAME
          ,V2.JOB
FROM      STAFF_V1 V1
LEFT OUTER JOIN
          STAFF_V2 V2
ON        V1.ID      = V2.ID
WHERE     V2.JOB LIKE 'S%'
ORDER BY V1.ID;
```

```
ANSWER (WRONG)
=====
ID NAME      JOB
-- - - - - -
20 Pernal    Sales
```

Outer Join, WHERE Done Before, Correct

STAFF_V1		STAFF_V2		ANSWER	
ID	NAME	ID	JOB	=====	
10	Sanders	20	Sales	ID	NAME
20	Pernal	30	Mgr	10	Sanders
30	Marenghi	40	Sales	20	Pernal
		50	Mgr	30	Marenghi

OUTER-JOIN CRITERIA
=====>
V1.ID = V2.ID
V2.JOB LIKE 'S%'

Correct

```

SELECT    V1.ID
          ,V1.NAME
          ,V2.JOB
FROM      STAFF_V1 V1
LEFT OUTER JOIN
  (SELECT  *
   FROM    STAFF_V2
   WHERE   JOB LIKE 'S%'
  )AS V2
ON        V1.ID = V2.ID
ORDER BY V1.ID;

```

ANSWER		
ID	NAME	JOB
10	Sanders	-
20	Pernal	Sales
30	Marenghi	-

Outer Join, WHERE Done Independently - Correct

```

SELECT    V1.ID
          ,V1.NAME
          , (SELECT V2.JOB
              FROM   STAFF_V2 V2
              WHERE  V1.ID      = V2.ID
              AND    V2.JOB LIKE 'S%')
FROM      STAFF_V1 V1
ORDER BY V1.ID;

```

```

ANSWER
=====
ID  NAME      JOB
--  -
10  Sanders    -
20  Pernal     Sales
30  Marenghi   -

```

Sample Data

STAFF_V1	STAFF_V2
+-----+	+-----+
ID NAME	ID JOB
-- -----	-- -----
10 Sanders	20 Sales
20 Pernal	30 Mgr
30 Marenghi	40 Sales
+-----+	50 Mgr
	+-----+

Sub-Query

Sample Table DDL - Sub-Queries

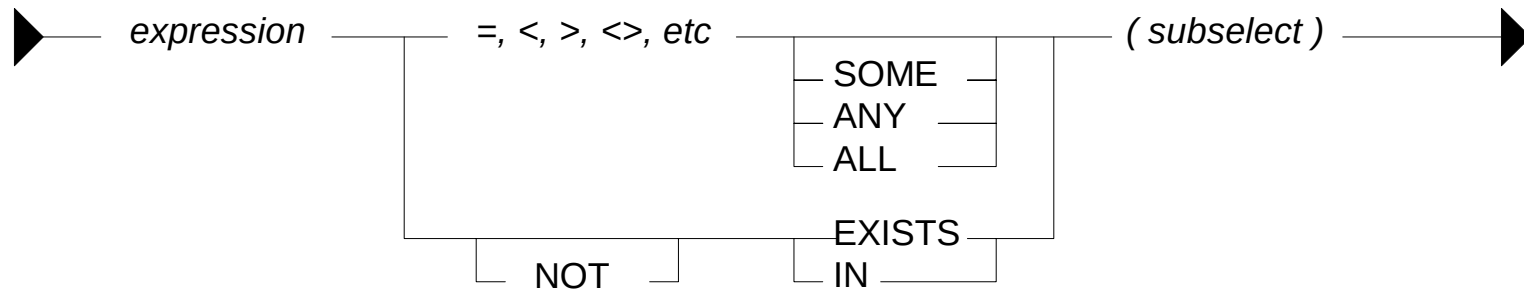
```
CREATE TABLE TABLE1
(T1A          CHAR(1)      NOT NULL
, T1B          CHAR(2)      NOT NULL
, PRIMARY KEY(T1A));
COMMIT;
```

```
CREATE TABLE TABLE2
(T2A          CHAR(1)      NOT NULL
, T2B          CHAR(1)      NOT NULL
, T2C          CHAR(1));
```

```
INSERT INTO TABLE1 VALUES ('A', 'AA'), ('B', 'BB'), ('C', 'CC');
INSERT INTO TABLE2 VALUES ('A', 'A', 'A'), ('B', 'A', NULL);
```

TABLE1		TABLE2		
+-----+		+-----+		
T1A T1B		T2A T2B T2C		
--- ---		--- --- ---		
A AA		A A A		
B BB		B A -		
C CC		+-----+		
+-----+		"-" = null		

Sub-Query Syntax



Example

```
SELECT *
FROM   TABLE1
WHERE  T1A > ANY
      (SELECT T2A
       FROM   TABLE2);
```

Warning

- Sub-queries return "true" or "false"
- Data can be "true", "false", or "unknown"
- Presence of null data and/or empty-sets greatly complicates sub-queries

No Keyword Sub-Query (1 of 2)

Works

```
SELECT *
FROM   TABLE1
WHERE  T1A =
      (SELECT T2A
       FROM   TABLE2
       WHERE  T2A = 'A');
```

```
ANSWER
=====
T1A T1B
--- --
A   AA
```

SUB-Q RESLT	TABLE1	TABLE2
+---+	+-----+	+-----+
T2A	T1A T1B	T2A T2B T2C
---	--- ---	--- --- ---
A	A AA	A A A
B	B BB	B A -
+---+	C CC	+-----+
	+-----+	"-" = null

Notes

- If no row in the sub-query result matches, the answer is false
- If one row in the sub-query result matches, the answer is true
- If more than one row in the sub-query result matches, you get a SQL error

No Keyword Sub-Query (2 of 2)

Fails

```
SELECT *
FROM   TABLE1
WHERE  T1A =
      (SELECT T2A
       FROM   TABLE2);
```

```
ANSWER
=====
<error>
```

SUB-Q	TABLE1	TABLE2
RESULT	+-----+	+-----+
+---+	T1A T1B	T2A T2B T2C
T2A	--- ---	--- --- ---
---	A AA	A A A
A	B BB	B A -
B	C CC	+-----+
+---+	+-----+	"-" = null

Notes

- If no row in the sub-query result matches, the answer is false
- If one row in the sub-query result matches, the answer is true
- If more than one row in the sub-query result matches, you get a SQL error

ANY Sub-Query

```
SELECT *
FROM   TABLE1
WHERE  T1A > ANY
      (SELECT T2A
       FROM   TABLE2);
```

ANSWER	SUB-Q	TABLE1	TABLE2
=====	RESULT	+-----+	+-----+
T1A T1B	+---+	T1A T1B	T2A T2B T2C
---	T2A	--- ---	--- --- ---
B BB	---	A AA	A A A
C CC	A	B BB	B A -
	B	C CC	+-----+
	+---+	+-----+	"-" = null

Notes

- If any row in the sub-query result matches, the answer is true
- If the sub-query result is empty, or all nulls, the answer is false
- If no value found in the sub-query result matches, the answer is also false

ANY and ALL vs. Column Functions

SUB-QUERY CHECK	EQUIVALENT COLUMN FUNCTION
=====	=====
> ANY(sub-query)	> MINIMUM(sub-query results)
< ANY(sub-query)	< MAXIMUM(sub-query results)
> ALL(sub-query)	> MAXIMUM(sub-query results)
< ALL(sub-query)	< MINIMUM(sub-query results)

ALL Sub-Query (1 of 3)

With non-Empty Sub-Query Result

```
SELECT *
FROM   TABLE1
WHERE  T1A = ALL
      (SELECT T2B
       FROM   TABLE2
       WHERE  T2B >= 'A');
```

ANSWER		SUB-Q
=====		RESLT
T1A	T1B	
---	--	T2B
A	AA	---
		A
		A
		+---+

TABLE1		TABLE2		
+-----+		+-----+		
T1A	T1B	T2A	T2B	T2C
---	---	---	---	---
A	AA	A	A	A
B	BB	B	A	-
C	CC	+-----+		
+-----+		"- " = null		

Notes

- If all rows in the sub-query result match, the answer is true
- If there are no rows in the sub-query result, the answer is also true
- If any row in the sub-query result does not match, or is null, the answer is false

ALL Sub-Query (2 of 3)

With Empty Sub-Query Result

```
SELECT *
FROM   TABLE1
WHERE  T1A = ALL
      (SELECT T2B
       FROM   TABLE2
       WHERE  T2B >= 'X');
```

ANSWER		SUB-Q
=====		RESLT
T1A	T1B	
---	--	T2B
A	AA	---
B	BB	+---+
C	CC	

TABLE1		TABLE2		
+-----+		+-----+		
T1A	T1B	T2A	T2B	T2C
---	---	---	---	---
A	AA	A	A	A
B	BB	B	A	-
C	CC	+-----+		
+-----+		"- " = null		

Notes

- If all rows in the sub-query result match, the answer is true
- If there are no rows in the sub-query result, the answer is also true
- If any row in the sub-query result does not match, or is null, the answer is false

ALL Sub-Query (2 of 3)

With Extra Check for Empty Set

```

SELECT *
FROM   TABLE1
WHERE  T1A = ALL
      (SELECT T2B
       FROM   TABLE2
       WHERE  T2B >= 'X')
AND    0 <>
      (SELECT COUNT(*)
       FROM   TABLE2
       WHERE  T2B >= 'X');

```

ANSWER
=====
0 rows

SQ-#1	SQ-#2	TABLE1	TABLE2
RESULT	RESULT	+-----+	+-----+
+---+	+---+	T1A T1B	T2A T2B T2C
T2B	(*)	--- ---	--- --- ---
---	---	A AA	A A A
+---+	0	B BB	B A -
	+---+	C CC	+-----+
		+-----+	"-" = null

Notes

- 1st sub-query: Check that all rows equal value
- 2nd sub-query: Check that there is more than zero values

EXISTS Sub-Query (1 of 2)

Always Returns a Match

```
SELECT *
FROM   TABLE1
WHERE  EXISTS
      (SELECT *
       FROM   TABLE2);
```

ANSWER	TABLE1	TABLE2
=====	+-----+	+-----+
T1A T1B	T1A T1B	T2A T2B T2C
--- --	--- ---	--- --- ---
A AA	A AA	A A A
B BB	B BB	B A -
C CC	C CC	+-----+
	+-----+	"-" = null

Always Returns a Non-Match

```
SELECT *
FROM   TABLE1
WHERE  EXISTS
      (SELECT *
       FROM   TABLE2
       WHERE  T2B >= 'X');
```

```
ANSWER
=====
0 rows
```

Notes

- If the sub-query matches on one or more rows, the result is true
- If the sub-query matches on no rows, the result is false

EXISTS Sub-Query (2 of 2)

Always returns a match (error)

```
SELECT *
FROM   TABLE1
WHERE  EXISTS
      (SELECT COUNT(*)
       FROM   TABLE2
       WHERE  T2B = 'X');
```

ANSWER	TABLE1	TABLE2
=====	+-----+	+-----+
T1A T1B	T1A T1B	T2A T2B T2C
--- --	--- ---	--- --- ---
A AA	A AA	A A A
B BB	B BB	B A -
C CC	C CC	+-----+
	+-----+	"-" = null

Notes

- Even when no data found in sub-query, COUNT(*) has value - zero
- Do not code this sub-query

NOT EXISTS vs. ALL (1 of 3)

Ignore Nulls, Find Match - Data Found

```
SELECT *
FROM   TABLE1
WHERE  NOT EXISTS
      (SELECT *
       FROM   TABLE2
        WHERE T2C >= 'A'
          AND T2C <> T1A);
```

ANSWERS		TABLE1	TABLE2
=====		+-----+	+-----+
T1A T1B		T1A T1B	T2A T2B T2C
--- ---		--- ---	--- --- ---
A AA		A AA	A A A
		B BB	B A -
		C CC	+-----+
		+-----+	"-" = null

```
SELECT *
FROM   TABLE1
WHERE  T1A = ALL
      (SELECT T2C
       FROM   TABLE2
        WHERE T2C >= 'A');
```

Notes

- Compare ALL and NOT EXISTS to return true when all matching rows equal
- In above examples, data found = return matching TABLE1 rows
- Both results equal

NOT EXISTS vs. ALL (2 of 3)

Ignore Nulls, No Match - No Data Found

```
SELECT *
FROM   TABLE1
WHERE  NOT EXISTS
      (SELECT *
       FROM   TABLE2
        WHERE T2C >= 'X'
          AND T2C <> T1A);
```

ANSWERS		TABLE1	TABLE2
=====	+-----+	+-----+	
T1A T1B	T1A T1B	T2A T2B T2C	
--- ---	--- ---	--- --- ---	
A AA	A AA	A A A	
B BB	B BB	B A -	
C CC	C CC	+-----+	
	+-----+	"-" = null	

```
SELECT *
FROM   TABLE1
WHERE  T1A = ALL
      (SELECT T2C
       FROM   TABLE2
        WHERE T2C >= 'X');
```

Notes

- In above examples, sub-query empty set = return all TABLE1 rows
- Both results equal

NOT EXISTS vs. ALL (3 of 3) - Process Nulls

```
SELECT *
FROM   TABLE1
WHERE  NOT EXISTS
      (SELECT *
       FROM   TABLE2
       WHERE  T2C <> T1A);
```

ANSWER		TABLE1	TABLE2
=====	+-----+	+-----+	
T1A T1B	T1A T1B	T2A T2B T2C	
--- ---	--- ---	--- --- ---	
A AA	A AA	A A A	
	B BB	B A -	
	C CC	+-----+	
	+-----+	"-" = null	

```
SELECT *
FROM   TABLE1
WHERE  T1A = ALL
      (SELECT T2C
       FROM   TABLE2);
```

```
ANSWER
=====
no rows
```

Notes

- NOT EXISTS sub-query does not see null - returns true
- ALL sub-query sees null - returns false

NOT EXISTS - same as ALL

```
SELECT *
FROM   TABLE1
WHERE  NOT EXISTS
      (SELECT *
       FROM   TABLE2
       WHERE  T2C <> T1A)
      OR  T2C IS NULL);
```

```
ANSWER
=====
no rows
```

IN Sub-Query

Two Matches

```
SELECT *
FROM   TABLE1
WHERE  T1A IN
      (SELECT T2A
       FROM   TABLE2);
```

ANSWER	TABLE1	TABLE2
=====	+-----+	+-----+
T1A T1B	T1A T1B	T2A T2B T2C
--- --	--- ---	--- --- ---
A AA	A AA	A A A
B BB	B BB	B A -
	C CC	
	+-----+	+-----+
		"-" = null

No Matches

```
SELECT *
FROM   TABLE1
WHERE  T1A IN
      (SELECT T2A
       FROM   TABLE2
       WHERE  T2A >= 'X');
```

```
ANSWER
=====
0 rows
```

Notes

- If any row in the sub-query result matches, the answer is true.
- If the sub-query result is empty, the answer is false.
- If no row in the sub-query result matches, the answer is also false.
- If all of the values in the sub-query result are null, the answer is false.

IN and = ANY Sub-Query

Nulls in "top" table

```
SELECT *
FROM   TABLE2
WHERE  T2C IN
      (SELECT T2C
       FROM   TABLE2);
```

```
SELECT *
FROM   TABLE2
WHERE  T2C = ANY
      (SELECT T2C
       FROM   TABLE2);
```

```
ANSWERS
=====
T2A T2B T2C
--- --- ---
A   A   A
```

```
TABLE2
+-----+
|T2A|T2B|T2C|
|---|---|---|
|A  |A  |A  |
|B  |A  |-  |
+-----+
"- " = null
```

Notes

- Nulls in "top" table results in no match (null does not equal null)

NOT IN Sub-Query

No Matches

```
SELECT *
FROM   TABLE1
WHERE  T1A NOT IN
      (SELECT T2C
       FROM   TABLE2);
```

ANSWER	TABLE1	TABLE2
=====	+-----+	+-----+
0 rows	T1A T1B	T2A T2B T2C
	--- ---	--- --- ---
	A AA	A A A
	B BB	B A -
	C CC	+-----+
	+-----+	"-" = null

Matches

```
SELECT *
FROM   TABLE1
WHERE  T1A NOT IN
      (SELECT T2C
       FROM   TABLE2
       WHERE  T2C IS NOT NULL);
```

ANSWER	
=====	
T1A	T1B
---	--
B	BB
C	CC

Notes

- True if no match found
- False if match found
- False if null found - to get true, eliminate from check

NOT EXISTS Sub-Query

Matches

```
SELECT *
FROM   TABLE1
WHERE  NOT EXISTS
      (SELECT *
       FROM   TABLE2
       WHERE  T1A = T2C);
```

ANSWER	TABLE1	TABLE2
=====	+-----+	+-----+
T1A T1B	T1A T1B	T2A T2B T2C
--- --	--- ---	--- --- ---
B BB	A AA	A A A
C CC	B BB	B A -
	C CC	+-----+
	+-----+	"-" = null

Notes

- Finds non-matching rows while ignoring nulls
- Compare to previous foil

Sub-Query Flavours

Uncorrelated Sub-Query

```
SELECT *
FROM   TABLE1
WHERE  T1A IN
      (SELECT T2A
       FROM   TABLE2);
```

ANSWER	TABLE1	TABLE2
=====	+-----+	+-----+
T1A T1B	T1A T1B	T2A T2B T2C
--- --	--- ---	--- --- ---
A AA	A AA	A A A
B BB	B BB	B A -
	C CC	+-----+
	+-----+	"-" = null

Correlated Sub-Query

```
SELECT *
FROM   TABLE1
WHERE  T1A IN
      (SELECT T2A
       FROM   TABLE2
       WHERE  T1A = T2A);
```

ANSWER
=====
T1A T1B
--- --
A AA
B BB

Notes

- **Uncorrelated:** Can resolve sub-query without reference to "top" table
- **Correlated:** Uses current value in "top" table resolve sub-query

Correlated Sub-Query, with Correlation Names

```
SELECT *
FROM   TABLE2 AA
WHERE  EXISTS
      (SELECT *
       FROM   TABLE2 BB
       WHERE  AA.T2A = BB.T2B);
```

```
ANSWER
=====
T2A T2B T2C
--- --- ---
A   A   A
```

```
TABLE2
+-----+
|T2A|T2B|T2C|
|---|---|---|
|A  |A  |A  |
|B  |A  | - |
+-----+
"- " = null
```

Notes

- Need to use correlation names when field names not unique
- Can make the SQL easier to read
- Does not affect access path

Multi-field Sub-Queries

Equal Checks

```
SELECT *
FROM   TABLE1
WHERE  (T1A,T1B) IN
        (SELECT T2A, T2B
         FROM   TABLE2);
```

ANSWER	TABLE1	TABLE2
=====	+-----+	+-----+
0 rows	T1A T1B	T2A T2B T2C
	--- ---	--- --- ---
	A AA	A A A
	B BB	B A -
	C CC	+-----+
	+-----+	"-" = null

```
SELECT *
FROM   TABLE1
WHERE  EXISTS
        (SELECT *
         FROM   TABLE2
         WHERE  T1A = T2A
         AND    T1B = T2B);
```

```
ANSWER
=====
0 rows
```

Non-Equal Checks

```
SELECT *
FROM   TABLE1
WHERE  EXISTS
        (SELECT *
         FROM   TABLE2
         WHERE  T1A = T2A
         AND    T1B >= T2B);
```

```
ANSWER
=====
T1A T1B
--- --
A   AA
B   BB
```

Nested Sub-Queries

```

SELECT EMPNO
       , LASTNAME
       , SALARY
FROM   EMPLOYEE
WHERE  SALARY >
      (SELECT MAX(SALARY)
       FROM   EMPLOYEE
       WHERE  EMPNO NOT IN
            (SELECT EMPNO
             FROM   EMP_ACT
             WHERE  PROJNO LIKE 'MA%'))
ORDER BY 1;

```

```

ANSWER
=====
EMPNO  LASTNAME  SALARY
-----
000010 HAAS        52750.00
000110 LUCCHESSI  46500.00

```

Sub-Queries, True if NONE Match

```
SELECT *
FROM   TABLE1 T1
WHERE  0 =
      (SELECT COUNT(*)
       FROM   TABLE2 T2
       WHERE  T1.T1A = T2.T2C);
```

```
SELECT *
FROM   TABLE1 T1
WHERE  NOT EXISTS
      (SELECT *
       FROM   TABLE2 T2
       WHERE  T1.T1A = T2.T2C);
```

```
SELECT *
FROM   TABLE1
WHERE  T1A NOT IN
      (SELECT T2C
       FROM   TABLE2
       WHERE  T2C IS NOT NULL);
```

TABLE1		TABLE2		
T1A	T1B	T2A	T2B	T2C
A	AA	A	A	A
B	BB	B	A	-
C	CC			

"-" = null

```
ANSWER
=====
T1A T1B
--- ---
B   BB
C   CC
```

Outer Join, True if NONE Match

```
SELECT T1.*
FROM   TABLE1 T1
LEFT OUTER JOIN
      TABLE2 T2
ON     T1.T1A = T2.T2C
WHERE  T2.T2C IS NULL;
```

```
ANSWER
=====
T1A T1B
--- ---
B   BB
C   CC
```

Sub-Queries, True if ANY Match

```
SELECT *
FROM   TABLE1 T1
WHERE  EXISTS
      (SELECT *
       FROM   TABLE2 T2
       WHERE  T1.T1A = T2.T2C);
```

```
SELECT *
FROM   TABLE1 T1
WHERE  1 <=
      (SELECT COUNT(*)
       FROM   TABLE2 T2
       WHERE  T1.T1A = T2.T2C);
```

```
SELECT *
FROM   TABLE1
WHERE  T1A = ANY
      (SELECT T2C
       FROM   TABLE2);
```

```
SELECT *
FROM   TABLE1
WHERE  T1A = SOME
      (SELECT T2C
       FROM   TABLE2);
```

```
SELECT *
FROM   TABLE1
WHERE  T1A IN
      (SELECT T2C
       FROM   TABLE2);
```

TABLE1	TABLE2
+-----+	+-----+
T1A T1B	T2A T2B T2C
--- ---	--- --- ---
A AA	A A -
B BB	A - -
C CC	+-----+
+-----+	null value

```
ANSWER
=====
T1A T1B
--- ---
A   AA
```

Joins, True if ANY Match

```
WITH T2 AS
  (SELECT DISTINCT T2C
   FROM   TABLE2
  )
SELECT T1.*
FROM   TABLE1 T1
      , T2
WHERE  T1.T1A = T2.T2C;
```

```
SELECT T1.*
FROM   TABLE1 T1
      , (SELECT DISTINCT T2C
        FROM   TABLE2
       ) AS T2
WHERE  T1.T1A = T2.T2C;
```

```
SELECT T1.*
FROM   TABLE1 T1
INNER JOIN
      (SELECT DISTINCT T2C
       FROM   TABLE2
      ) AS T2
ON     T1.T1A = T2.T2C;
```

TABLE1		TABLE2		
+-----+		+-----+		
T1A T1B		T2A T2B T2C		
--- ---		--- --- ---		
A AA		A A A		
B BB		B A -		
C CC		+-----+		
+-----+		"- " = null		

```
ANSWER
=====
T1A T1B
--- ---
A   AA
```

Sub-Queries, True if Ten Match

```
SELECT *
FROM   TABLE1 T1
WHERE  10 =
      (SELECT COUNT(*)
       FROM   TABLE2 T2
       WHERE  T1.T1A = T2.T2B);
```

```
SELECT *
FROM   TABLE1
WHERE  EXISTS
      (SELECT T2B
       FROM   TABLE2
       WHERE  T1A = T2B
       GROUP BY T2B
       HAVING COUNT(*) = 10);
```

```
SELECT *
FROM   TABLE1
WHERE  T1A IN
      (SELECT T2B
       FROM   TABLE2
       GROUP BY T2B
       HAVING COUNT(*) = 10);
```

```
SELECT *
FROM   TABLE1
WHERE  (T1A,10) IN
      (SELECT T2B, COUNT(*)
       FROM   TABLE2
       GROUP BY T2B);
```

TABLE1	TABLE2
+-----+	+-----+
T1A T1B	T2A T2B T2C
--- ---	--- --- ---
A AA	A A A
B BB	B A -
C CC	+-----+
+-----+	"-" = null

ANSWER
=====
0 rows

Joins, True if TEN Match

```
WITH T2 AS
  (SELECT    T2B
   FROM      TABLE2
   GROUP BY  T2B
   HAVING    COUNT(*) = 10
  )
SELECT T1.*
FROM   TABLE1 T1
      ,T2
WHERE  T1.T1A = T2.T2B;
```

```
SELECT T1.*
FROM   TABLE1 T1
      , (SELECT    T2B
        FROM      TABLE2
        GROUP BY  T2B
        HAVING    COUNT(*) = 10
      ) AS T2
WHERE  T1.T1A = T2.T2B;
```

```
SELECT T1.*
FROM   TABLE1 T1
INNER JOIN
      (SELECT    T2B
        FROM      TABLE2
        GROUP BY  T2B
        HAVING    COUNT(*) = 10
      ) AS T2
ON     T1.T1A = T2.T2B;
```

TABLE1	TABLE2
+-----+	+-----+
T1A T1B	T2A T2B T2C
--- ---	--- --- ---
A AA	A A A
B BB	B A -
C CC	+-----+
+-----+	"-" = null

ANSWER
=====
0 rows

Sub-Queries, True if ALL Match (1 of 3)

Sub-Query Finds Rows

```
SELECT *
FROM   TABLE1
WHERE  T1A = ALL
      (SELECT T2B
       FROM   TABLE2);
```

```
SELECT *
FROM   TABLE1
WHERE  NOT EXISTS
      (SELECT *
       FROM   TABLE2
        WHERE T1A <> T2B);
```

TABLE1		TABLE2		
T1A	T1B	T2A	T2B	T2C
A	AA	A	A	A
B	BB	B	A	-
C	CC			

"-" = null

ANSWER	
T1A	T1B
A	AA

Before You Begin - Business Issues

- What do we do if we encounter nulls?
- Should we exclude nulls?
- What do we do if we get an empty set (in sub-query)?

Sub-Queries, True if ALL Match (2 of 3)

Sub-Query is Empty Set

```
SELECT *
FROM   TABLE1
WHERE  T1A = ALL
      (SELECT T2B
       FROM   TABLE2
       WHERE  T2B >= 'X');
```

```
SELECT *
FROM   TABLE1
WHERE  NOT EXISTS
      (SELECT *
       FROM   TABLE2
       WHERE  T1A <> T2B
            AND T2B >= 'X');
```

TABLE1		TABLE2		
T1A	T1B	T2A	T2B	T2C
A	AA	A	A	A
B	BB	B	A	-
C	CC			

"-" = null

ANSWER	
T1A	T1B
A	AA
B	BB
C	CC

Notes

- Above examples - empty set means sub-query returns true

Sub-Queries, True if ALL Match (3 of 3)

Sub-Query is Empty Set

```
SELECT *
FROM   TABLE1
WHERE  T1A = ALL
      (SELECT T2B
       FROM   TABLE2
       WHERE  T2B >= 'X')
AND    0 <>
      (SELECT COUNT(*)
       FROM   TABLE2
       WHERE  T2B >= 'X');
```

```
SELECT *
FROM   TABLE1
WHERE  T1A IN
      (SELECT MAX(T2B)
       FROM   TABLE2
       WHERE  T2B >= 'X'
       HAVING COUNT(DISTINCT T2B) = 1);
```

TABLE1		TABLE2		
T1A	T1B	T2A	T2B	T2C
A	AA	A	A	A
B	BB	B	A	-
C	CC			

"-" = null

ANSWER
=====
0 rows

Notes

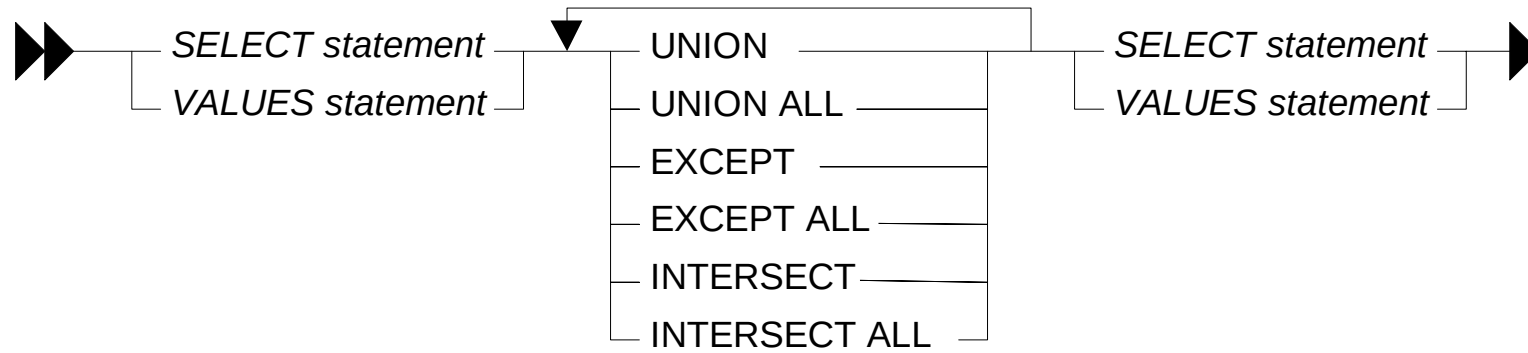
- Above examples - empty set means sub-query returns false

Union, Intersect, and Except

Examples of Union, Except, and Intersect

R1	R2	R1 UNION R2	R1 UNION ALL R2	R1 INTERSECT R2	R1 INTERSECT ALL R2	R1 EXCEPT R2	R1 EXCEPT ALL R2
--	--	-----	-----	-----	-----	-----	-----
A	A	A	A	A	A	E	A
A	A	B	A	B	A		C
A	B	C	A	C	B		C
B	B	D	A		B		E
B	B	E	A		C		
C	C		B				
C	D		B				
C			B				
E			B				
			B				
			B				
			C				
			C				
			C				
			C				
			D				
			E				

Union, Except, and Intersect Syntax



Sample Views

```
CREATE VIEW R1 (R1)
  AS VALUES ('A'), ('A'), ('A'), ('B'), ('B'), ('C'), ('C'), ('C'), ('E');
CREATE VIEW R2 (R2)
  AS VALUES ('A'), ('A'), ('B'), ('B'), ('B'), ('C'), ('D');
```

```
SELECT  R1
FROM    R1
ORDER BY R1;
```

```
SELECT  R2
FROM    R2
ORDER BY R2;
```

ANSWER

=====

R1 R2

-- --

A A

A A

A B

B B

B B

C C

C D

C

E

Union and Union All SQL

```
SELECT  R1
FROM    R1
UNION
SELECT  R2
FROM    R2
ORDER BY 1;
```

```
SELECT  R1
FROM    R1
UNION ALL
SELECT  R2
FROM    R2
ORDER BY 1;
```

R1	R2	UNION	UNION ALL
--	--	=====	=====
A	A	A	A
A	A	B	A
A	B	C	A
B	B	D	A
B	B	E	A
C	C		B
C	D		B
C			B
E			B
			C
			C
			C
			C
			D
			E

Notes

- **UNION** returns all values
- **UNION ALL** returns all rows

Intersect and Intersect All SQL

```
SELECT  R1
FROM    R1
INTERSECT
SELECT  R2
FROM    R2
ORDER BY 1;
```

```
SELECT  R1
FROM    R1
INTERSECT ALL
SELECT  R2
FROM    R2
ORDER BY 1;
```

R1	R2	INTERSECT	INTERSECT ALL
--	--	=====	=====
A	A	A	A
A	A	B	A
A	B	C	B
B	B		B
B	B		C
C	C		
C	D		
C			
E			

Notes

- **INTERSECT** returns intersecting values
- **INTERSECT ALL** returns all intersecting rows

Except and Except All SQL (R1 on top)

```
SELECT  R1
FROM    R1
EXCEPT
SELECT  R2
FROM    R2
ORDER BY 1;
```

```
SELECT  R1
FROM    R1
EXCEPT ALL
SELECT  R2
FROM    R2
ORDER BY 1;
```

R1	R2	R1 EXCEPT R2	R1 EXCEPT ALL R2
--	--	=====	=====
A	A	E	A
A	A		C
A	B		C
B	B		E
B	B		
C	C		
C	D		
C			
E			

Notes

- **EXCEPT** returns non-intersecting values
- **EXCEPT ALL** returns all non-intersecting rows

Except and Except All SQL (R2 on top)

```
SELECT  R2
FROM    R2
EXCEPT
SELECT  R1
FROM    R1
ORDER BY 1;
```

```
SELECT  R2
FROM    R2
EXCEPT ALL
SELECT  R1
FROM    R1
ORDER BY 1;
```

R1	R2	R2 EXCEPT R1	R2 EXCEPT ALL R1
--	--	=====	=====
A	A	D	B
A	A		D
A	B		
B	B		
B	B		
C	C		
C	D		
C			
E			

Notes

- **EXCEPT** and **EXCEPT ALL** return different result - depending upon the sequence in which tables are listed

Use of Parenthesis in Union

```
SELECT  R1
FROM    R1
UNION
SELECT  R2
FROM    R2
EXCEPT
SELECT  R2
FROM    R2
ORDER BY 1;
```

```
ANSWER
=====
E
```

```
(SELECT  R1
FROM    R1
UNION
SELECT  R2
FROM    R2
)EXCEPT
SELECT  R2
FROM    R2
ORDER BY 1;
```

```
ANSWER
=====
E
```

```
SELECT  R1
FROM    R1
UNION
(SELECT  R2
FROM    R2
EXCEPT
SELECT  R2
FROM    R2
)ORDER BY 1;
```

```
ANSWER
=====
A
B
C
E
```

R1	R2
--	--
A	A
A	A
A	B
B	B
B	B
C	C
C	D
C	
E	

Precedence Rules

- Operations in parenthesis done first
- INTERSECT operations done before either UNION or EXCEPT
- Operations of equal worth done from top to bottom

View Containing Union All

```
CREATE VIEW VVVV (COL1, COL2) AS
SELECT COL1, COL2 FROM TABLE1
UNION ALL
SELECT COL1, COL3 FROM TABLE2
EXCEPT
SELECT COL1, COL1 FROM TABLE1;
```

Select from View

```
SELECT *
FROM   VVVV
WHERE  COL2 > 'BB';
```

Identity Column

Introduction

Notes

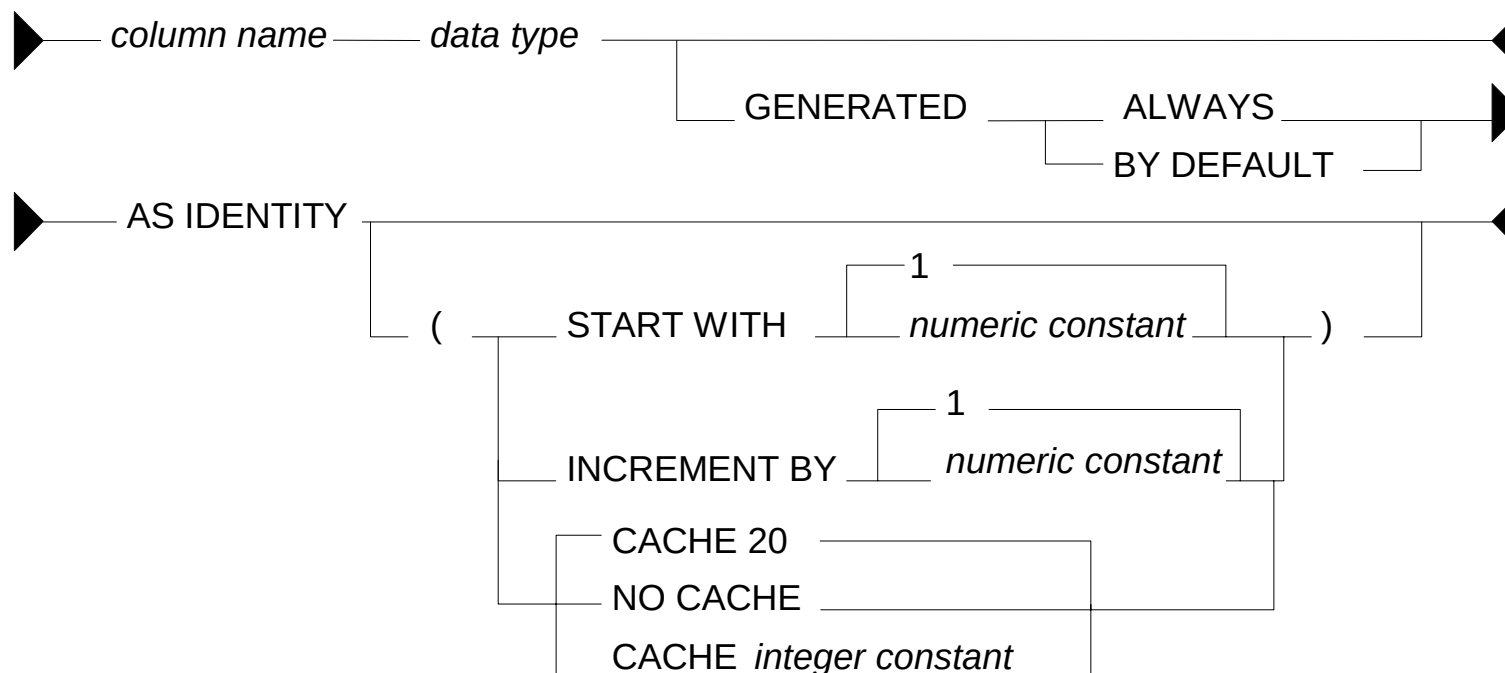
- Used to automatically assign unique values to table keys
- Can be automatically generated, or generated by default
- Current value can be reset if needed

Sample Table Definition

```
CREATE TABLE INVOICE_DATA
(INVOICE#          INTEGER                      NOT NULL
      GENERATED ALWAYS AS IDENTITY
      (START WITH 100
      , INCREMENT BY 1
      , NO CACHE)

, SALE_DATE        DATE                        NOT NULL
, CUSTOMER_ID      CHAR(20)                   NOT NULL
, PRODUCT_ID       INTEGER                     NOT NULL
, QUANTITY         INTEGER                     NOT NULL
, PRICE            DECIMAL(18,2)              NOT NULL
, PRIMARY KEY      (INVOICE#));
```

Identity Column Syntax



Sample Table Definition

```

CREATE TABLE INVOICE_DATA
(INVOICE#          INTEGER                NOT NULL
    GENERATED ALWAYS AS IDENTITY
        (START WITH 100
        , INCREMENT BY 1
        , NO CACHE)
, SALE_DATE        DATE                  NOT NULL
, CUSTOMER_ID      CHAR(20)              NOT NULL
, PRODUCT_ID       INTEGER               NOT NULL
, PRIMARY KEY      (INVOICE#));
  
```

Rules and Restrictions

Come in two flavors

- Always generated by DB2
- Generated by DB2 only if user does not provide value

Rules and Restrictions

- Only one per table
- Field cannot be updated if "generated always"
- Column type must be numeric - and not allow fractional values
- Identity column value generated before any BEFORE triggers applied
- Unique index is not required, but is a good idea
- Unlike triggers, is used in a LOAD
- Beware "gaps in sequence"

Insert Usage

```
CREATE TABLE INVOICE_DATA
(INVOICE#          INTEGER              NOT NULL
      GENERATED ALWAYS AS IDENTITY
      (START WITH 100
      , INCREMENT BY 1
      , NO CACHE)
, SALE_DATE        DATE                  NOT NULL
, CUSTOMER_ID      CHAR(20)              NOT NULL
, PRODUCT_ID       INTEGER               NOT NULL
, QUANTITY         INTEGER               NOT NULL
, PRICE            DECIMAL(18,2)         NOT NULL
, PRIMARY KEY      (INVOICE#));
```

Insert Rows

```
INSERT INTO INVOICE_DATA
VALUES (DEFAULT, '2001-11-22', 'ABC', 123, 100, 10);
```

```
INSERT INTO INVOICE_DATA
(SALE_DATE, CUSTOMER_ID, PRODUCT_ID, QUANTITY, PRICE)
VALUES ('2001-11-23', 'DEF', 123, 100, 10);
```

After Inserts

INVOICE#	SALE_DATE	CUSTOMER_ID	PRODUCT_ID	QUANTITY	PRICE
-----	-----	-----	-----	-----	-----
100	11/22/2001	ABC	123	100	10.00
101	11/23/2001	DEF	123	100	10.00

Restart Identity Column Value

Restart Value

```
ALTER TABLE INVOICE_DATA
ALTER COLUMN INVOICE# RESTART WITH 1000;
```

More sample inserts

```
INSERT INTO INVOICE_DATA
VALUES (DEFAULT, '2001-11-24', 'XXX', 123, 100, 10)
      , (DEFAULT, '2001-11-25', 'YYY', 123, 100, 10);
```

After second inserts

INVOICE#	SALE_DATE	CUSTOMER_ID	PRODUCT_ID	QUANTITY	PRICE
-----	-----	-----	-----	-----	-----
100	11/22/2001	ABC	123	100	10.00
101	11/23/2001	DEF	123	100	10.00
1000	11/24/2001	XXX	123	100	10.00
1001	11/25/2001	YYY	123	100	10.00

Usage Notes

- Can only alter the start number - not the direction or increment value
- Resetting the start number to a lower number can cause duplicate values

Overriding the default identity value

```
CREATE TABLE CUSTOMERS
(CUST#          INTEGER                NOT NULL
      GENERATED BY DEFAULT AS IDENTITY (NO CACHE)
, CNAME         CHAR(10)              NOT NULL
, CTYPE         CHAR(03)              NOT NULL
, PRIMARY KEY   (CUST#));
COMMIT;
```

```
INSERT INTO CUSTOMERS
VALUES (DEFAULT, 'FRED', 'XXX');
```

```
SELECT      *
FROM        CUSTOMERS
ORDER BY 1;
```

```
ROLLBACK;
```

```
<<< ANSWER
=====
CUST#  CNAME  CTYPE
-----
      1  FRED   XXX
```

```
INSERT INTO CUSTOMERS
VALUES (1, 'FRED', 'XXX');
```

```
SELECT      *
FROM        CUSTOMERS
ORDER BY 1;
```

```
COMMIT;
```

```
<<< ANSWER
=====
CUST#  CNAME  CTYPE
-----
      1  FRED   XXX
```

IDENTITY_VAL_LOCAL Function (1 of 2)

```
INSERT INTO INVOICE_TABLE
VALUES (DEFAULT, '2000-11-22', 'ABC', 123, 100, 10);
```

```
WITH TEMP (ID) AS
(VVALUES (IDENTITY_VAL_LOCAL()))
SELECT *
FROM TEMP;
```

```
COMMIT;
```

```
WITH TEMP (ID) AS
(VVALUES (IDENTITY_VAL_LOCAL()))
SELECT *
FROM TEMP;
```

```
<<< ANSWER
=====
      ID
      ---
      1
```

```
<<< ANSWER
=====
      ID
      ---
      1
```

Notes

- Gets most recent Identity Column value
- Returns null if no insert done in thread (or Unit of Work?)
- If inserts done to multiple tables, returns value from last insert done
- Cannot be called from a trigger - use transition variable for column instead
- Value returned is unpredictable if prior insert failed.

IDENTITY_VAL_LOCAL Function (2 of 2)

```
INSERT INTO INVOICE_TABLE
VALUES (DEFAULT, '2000-11-23', 'ABC', 123, 100, 10);
```

```
INSERT INTO INVOICE_TABLE
VALUES (DEFAULT, '2000-11-24', 'ABC', 123, 100, 10)
      , (DEFAULT, '2000-11-25', 'ABC', 123, 100, 10);
```

```
SELECT  INVOICE#           AS INV#
        , SALE_DATE
        , IDENTITY_VAL_LOCAL() AS ID
FROM    INVOICE_TABLE
ORDER BY 1;
COMMIT;
```

ANSWER

```
=====
INV#  SALE_DATE  ID
----  -
1 11/22/2000 2
2 11/23/2000 2
3 11/24/2000 2
4 11/25/2000 2
```

Usage in SET statement

```
SELECT INVOICE#           AS INV#
        , SALE_DATE
        , IDENTITY_VAL_LOCAL() AS ID
FROM    INVOICE_TABLE
WHERE   ID = IDENTITY_VAL_LOCAL();
```

ANSWER

```
=====
INV#  SALE_DATE  ID
----  -
2 11/23/2000 2
```

Recursive SQL

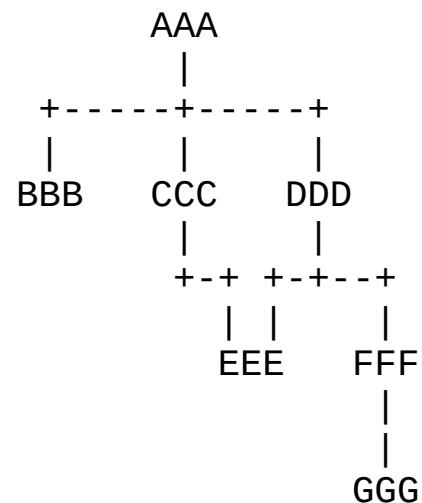
Sample Table - Recursion

```
CREATE TABLE HIERARCHY
(PKEY      CHAR(03)      NOT NULL
,CKEY      CHAR(03)      NOT NULL
,NUM       SMALLINT      NOT NULL
,PRIMARY KEY(PKEY, CKEY)
,CONSTRAINT DT1 CHECK (PKEY <> CKEY)
,CONSTRAINT DT2 CHECK (NUM  > 0));
COMMIT;
```

```
CREATE UNIQUE INDEX HIER_X1 ON HIERARCHY
(CKEY, PKEY);
COMMIT;
```

HIERARCHY

PKEY	CKEY	NUM
AAA	BBB	1
AAA	CCC	5
AAA	DDD	20
CCC	EEE	33
DDD	EEE	44
DDD	FFF	5
FFF	GGG	5



Recursive SQL Processing

```

WITH PARENT (PKEY, CKEY) AS
(
  SELECT PKEY, CKEY
  FROM   HIERARCHY
  WHERE  PKEY = 'AAA'
  UNION ALL
  SELECT C.PKEY, C.CKEY
  FROM   HIERARCHY C
        , PARENT P
  WHERE  P.CKEY = C.PKEY
)
SELECT PKEY, CKEY
FROM   PARENT;

```

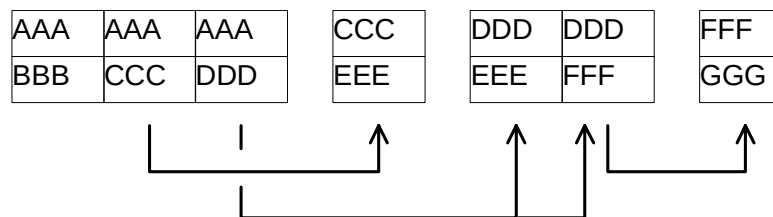
ANSWER		PROCESSING SEQUENCE
=====	=====	
PKEY	CKEY	
----	----	
AAA	BBB	< 1st pass
AAA	CCC	" "
AAA	DDD	" "
CCC	EEE	< 2nd pass
DDD	EEE	< 3rd pass
DDD	FFF	" "
FFF	GGG	< 4th pass

Processing Logic

HIERARCHY		
-----	-----	-----
PKEY	CKEY	NUM
----	----	----
AAA	BBB	1
AAA	CCC	5
AAA	DDD	20
CCC	EEE	33
DDD	EEE	44
DDD	FFF	5
FFF	GGG	5
-----	-----	-----

PKEY >

CKEY >



List Children of AAA (but not AAA)

```
WITH PARENT (CKEY) AS
(SELECT CKEY
 FROM    HIERARCHY
 WHERE   PKEY = 'AAA'
 UNION ALL
 SELECT C.CKEY
 FROM    HIERARCHY C
         ,PARENT      P
 WHERE   P.CKEY = C.PKEY
 )
SELECT CKEY
FROM    PARENT;
```

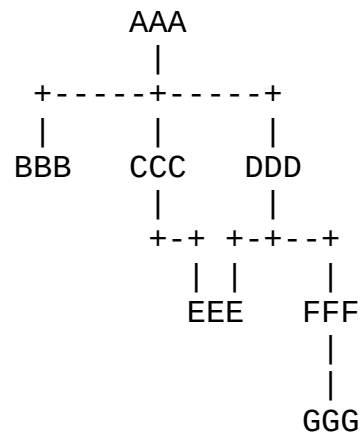
ANSWER
=====

CKEY

BBB
CCC
DDD
EEE
EEE
FFF
GGG

Sample Data

HIERARCHY		
+-----+		
PKEY	CKEY	NUM
----	----	---
AAA	BBB	1
AAA	CCC	5
AAA	DDD	20
CCC	EEE	33
DDD	EEE	44
DDD	FFF	5
FFF	GGG	5
+-----+		



List Children of AAA (Include AAA)

```
WITH PARENT (CKEY) AS
(
  SELECT DISTINCT PKEY
  FROM   HIERARCHY
  WHERE  PKEY = 'AAA'
  UNION ALL
  SELECT C.CKEY
  FROM   HIERARCHY C
         , PARENT P
  WHERE  P.CKEY = C.PKEY
)
SELECT CKEY
FROM   PARENT;
```

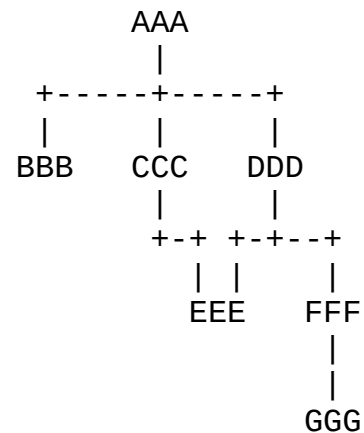
ANSWER
=====

CKEY

AAA
BBB
CCC
DDD
EEE
EEE
FFF
GGG

Sample Data

HIERARCHY		
+	-	-
+	-	-
PKEY	CKEY	NUM
----	----	---
AAA	BBB	1
AAA	CCC	5
AAA	DDD	20
CCC	EEE	33
DDD	EEE	44
DDD	FFF	5
FFF	GGG	5
+	-	-
+	-	-



List DISTINCT Children of AAA (1 of 2)

```
WITH PARENT (CKEY) AS
(SELECT DISTINCT PKEY
 FROM    HIERARCHY
 WHERE   PKEY = 'AAA'
 UNION ALL
 SELECT  C.CKEY
 FROM    HIERARCHY C
        ,PARENT      P
 WHERE   P.CKEY = C.PKEY
 )
SELECT DISTINCT CKEY
FROM    PARENT;
```

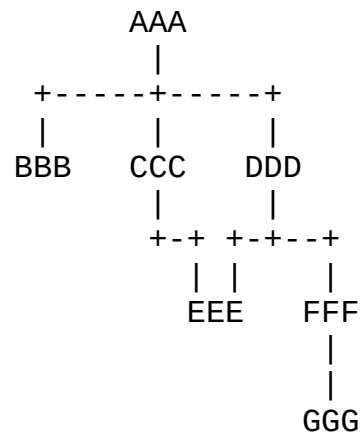
ANSWER
=====

CKEY

AAA
BBB
CCC
DDD
EEE
FFF
GGG

Sample Data

HIERARCHY		
+	-----	+
PKEY	CKEY	NUM
----	-----	---
AAA	BBB	1
AAA	CCC	5
AAA	DDD	20
CCC	EEE	33
DDD	EEE	44
DDD	FFF	5
FFF	GGG	5
+	-----	+



List DISTINCT Children of AAA (2 of 2)

```

WITH PARENT (CKEY) AS
(SELECT DISTINCT PKEY
 FROM   HIERARCHY
 WHERE  PKEY = 'AAA'
 UNION ALL
 SELECT C.CKEY
 FROM   HIERARCHY C
        , PARENT      P
 WHERE  P.CKEY = C.PKEY
 )
 ,DISTINCT_PARENT (CKEY) AS
 (SELECT DISTINCT CKEY
  FROM   PARENT
 )
 SELECT CKEY
 FROM   DISTINCT_PARENT;

```

ANSWER

=====

CKEY

AAA

BBB

CCC

DDD

EEE

FFF

GGG

HIERARCHY

+-----+

| PKEY | CKEY | NUM |

|-----|-----|---

| AAA | BBB | 1 |

| AAA | CCC | 5 |

| AAA | DDD | 20 |

| CCC | EEE | 33 |

| DDD | EEE | 44 |

| DDD | FFF | 5 |

| FFF | GGG | 5 |

+-----+

Compared to Prior SQL

- Makes intermediate DISTINCT_PARENT table
- This table can be then be joined to other tables

Show Item Level in Hierarchy

```
WITH PARENT (CKEY, LVL) AS
(SELECT DISTINCT PKEY, 0
 FROM   HIERARCHY
 WHERE  PKEY = 'AAA'
 UNION ALL
 SELECT C.CKEY, P.LVL +1
 FROM   HIERARCHY C
        , PARENT P
 WHERE  P.CKEY = C.PKEY
 )
SELECT CKEY, LVL
FROM   PARENT;
```

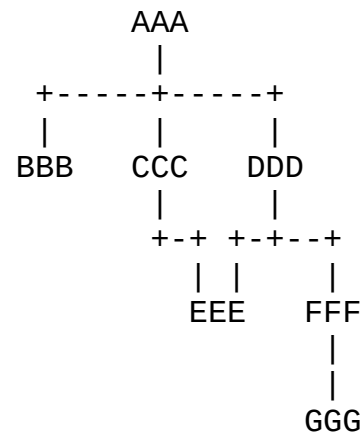
```

ANSWER
=====
CKEY  LVL
-----
AAA      0
BBB      1
CCC      1
DDD      1
EEE      2
EEE      2
FFF      2
GGG      3

```

Sample Data

HIERARCHY		
+	-	-
+	-	-
PKEY	CKEY	NUM
----	----	----
AAA	BBB	1
AAA	CCC	5
AAA	DDD	20
CCC	EEE	33
DDD	EEE	44
DDD	FFF	5
FFF	GGG	5
+	-	-
+	-	-



Select Certain Levels (1 of 2)

```

WITH PARENT (CKEY, LVL) AS
(SELECT DISTINCT PKEY, 0
 FROM   HIERARCHY
 WHERE  PKEY = 'AAA'
 UNION ALL
 SELECT C.CKEY, P.LVL +1
 FROM   HIERARCHY C
        ,PARENT     P
 WHERE  P.CKEY = C.PKEY
 )
SELECT CKEY, LVL
FROM   PARENT
WHERE  LVL < 3;

```

ANSWER		HIERARCHY		
=====		+-----+-----+-----+		
CKEY	LVL	PKEY	CKEY	NUM
----	---	-----	-----	----
AAA	0	AAA	BBB	1
BBB	1	AAA	CCC	5
CCC	1	AAA	DDD	20
DDD	1	CCC	EEE	33
EEE	2	DDD	EEE	44
EEE	2	DDD	FFF	5
FFF	2	FFF	GGG	5
		+-----+-----+-----+		

Warning

- Example selects rows where level < 3
- Statement may run forever if hierarchy has loop
- Inefficient - Resolves whole hierarchy, then discards rows

Select Certain Levels (2 of 2)

```

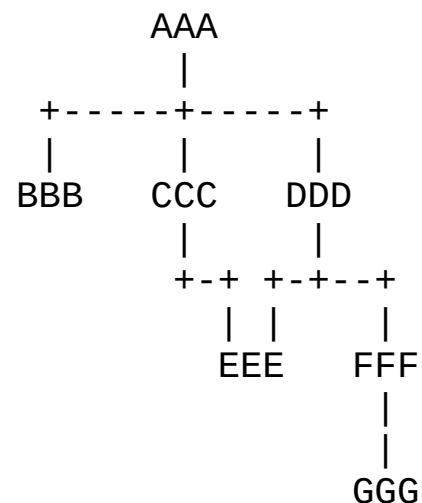
WITH PARENT (CKEY, LVL) AS
(SELECT DISTINCT PKEY, 0
 FROM   HIERARCHY
 WHERE  PKEY = 'AAA'
 UNION ALL
 SELECT C.CKEY, P.LVL +1
 FROM   HIERARCHY C
        ,PARENT P
 WHERE  P.CKEY = C.PKEY
        AND P.LVL+1 < 3
 )
SELECT CKEY, LVL
FROM   PARENT;

```

```

ANSWER
=====
CKEY  LVL
----  ---
AAA    0
BBB    1
CCC    1
DDD    1
EEE    2
EEE    2
FFF    2

```



Better Design

- Example selects rows where level < 3
- Statement doesn't run forever if loop in hierarchy
- Efficient - Stops processing when "n" levels reached

Select Rows Where LEVEL = 2

```
WITH PARENT (CKEY, LVL) AS
(SELECT DISTINCT PKEY, 0
 FROM   HIERARCHY
 WHERE  PKEY = 'AAA'
 UNION ALL
 SELECT C.CKEY, P.LVL +1
 FROM   HIERARCHY C
        ,PARENT     P
 WHERE  P.CKEY = C.PKEY
        AND P.LVL+1 < 3
 )
SELECT CKEY, LVL
FROM   PARENT
WHERE  LVL = 2;
```

ANSWER		HIERARCHY		
=====		+-----+-----+-----+		
CKEY	LVL	PKEY	CKEY	NUM
----	---	-----	-----	----
EEE	2	AAA	BBB	1
EEE	2	AAA	CCC	5
FFF	2	AAA	DDD	20
		CCC	EEE	33
		DDD	EEE	44
		DDD	FFF	5
		FFF	GGG	5
		+-----+-----+-----+		

Processing Logic

- Get start row
- Use recursion to get all rows, LEVEL <= 2
- In final query, select only LEVEL = 2

Find all Values in Path, Where Path is at least 4 Levels Deep

```

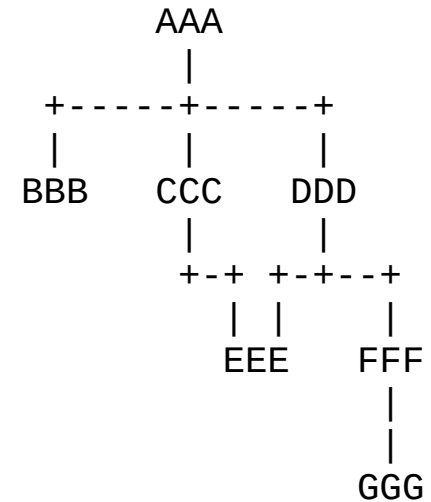
WITH TEMP1 (CKEY, LVL) AS
(SELECT DISTINCT PKEY, 1
 FROM   HIERARCHY
 WHERE  PKEY = 'AAA'
 UNION ALL
 SELECT C.CKEY, P.LVL +1
 FROM   HIERARCHY C
        ,TEMP1      P
 WHERE  P.CKEY = C.PKEY
        AND  P.LVL  < 4
 )
,TEMP2 (CKEY, LVL) AS
(SELECT CKEY, LVL
 FROM   TEMP1
 WHERE  LVL = 4
 UNION ALL
 SELECT C.PKEY, D.LVL -1
 FROM   HIERARCHY C
        ,TEMP2      D
 WHERE  D.CKEY = C.CKEY
 )
SELECT *
FROM   TEMP2;

```

```

ANSWER
=====
CKEY  LVL
----  ---
GGG    4
FFF    3
DDD    2
AAA    1

```



Recursion - Warning Messages

With Warning

```
WITH TEMP1 (N1) AS
(SELECT ID
 FROM   STAFF
 WHERE  ID = 10
 UNION ALL
 SELECT N1 +10
 FROM   TEMP1
 WHERE  N1 < 50
 )
SELECT *
FROM   TEMP1;
```

```
ANSWER
=====
      N1
      --
      warn
      10
      20
      30
      40
      50
```

Without Warning

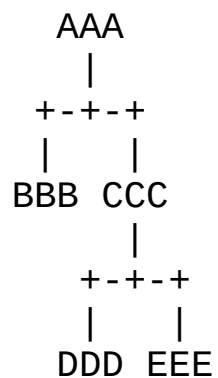
```
WITH TEMP1 (N1) AS
(SELECT INT(ID)
 FROM   STAFF
 WHERE  ID = 10
 UNION ALL
 SELECT N1 +10
 FROM   TEMP1
 WHERE  N1 < 50
 )
SELECT *
FROM   TEMP1;
```

```
ANSWER
=====
      N1
      --
      10
      20
      30
      40
      50
```

Hierarchy Flavours

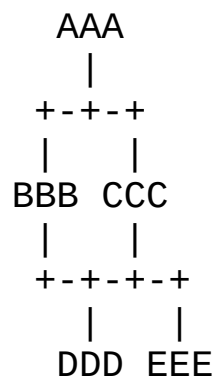
DIVERGENT

=====



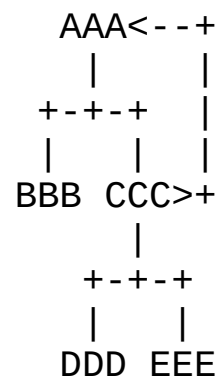
CONVERGENT

=====



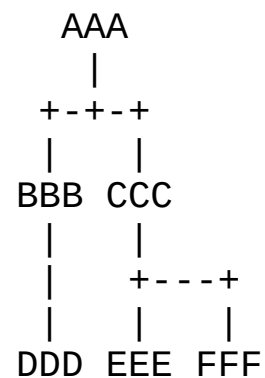
RECURSIVE

=====



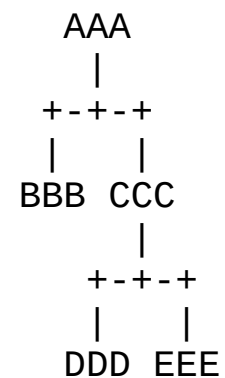
BALANCED

=====



UNBALANCED

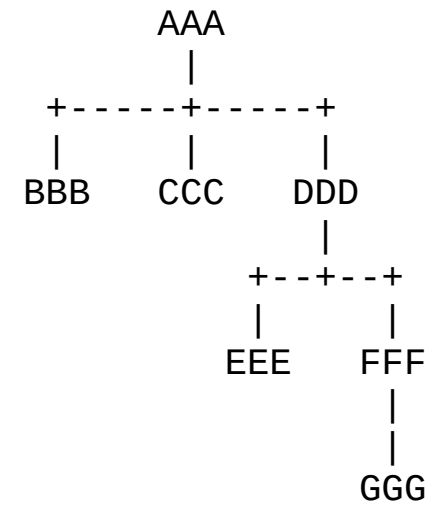
=====



Divergent Hierarchy - Table and Layout

OBJECTS_RELATES

KEYO	PKEY	NUM	PRICE
AAA			\$10
BBB	AAA	1	\$21
CCC	AAA	5	\$23
DDD	AAA	20	\$25
EEE	DDD	44	\$33
FFF	DDD	5	\$34
GGG	FFF	5	\$44



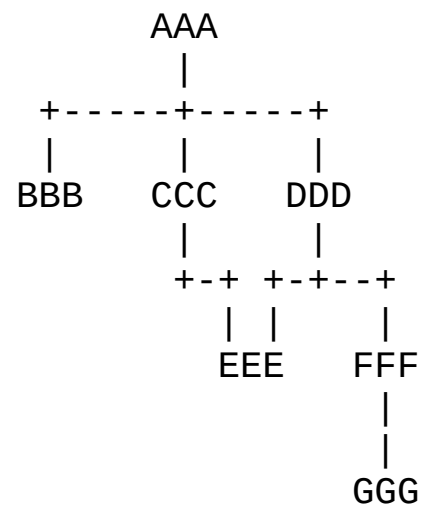
Convergent Hierarchy - Tables and Layout

OBJECTS

+-----+	
KEYO	PRICE
+-----+	
AAA	\$10
BBB	\$21
CCC	\$23
DDD	\$25
EEE	\$33
FFF	\$34
GGG	\$44
+-----+	

RELATIONSHIPS

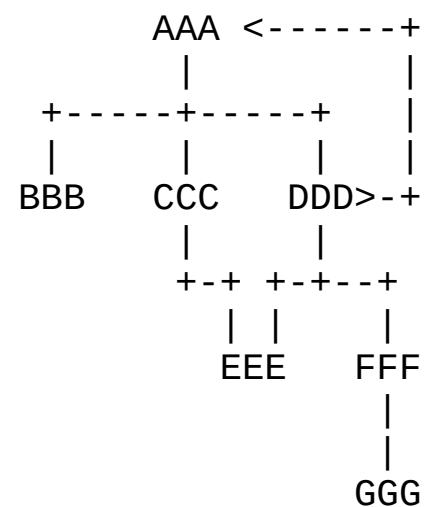
+-----+		
PKEY	CKEY	NUM
+-----+		
AAA	BBB	1
AAA	CCC	5
AAA	DDD	20
CCC	EEE	33
DDD	EEE	44
DDD	FFF	5
FFF	GGG	5
+-----+		



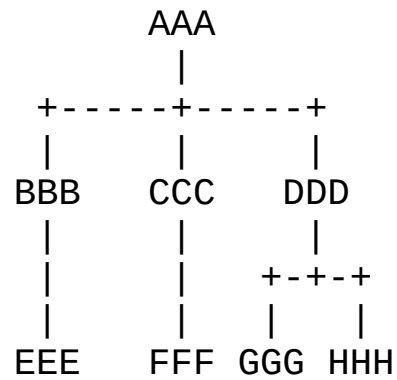
Recursive Hierarchy - Tables and Layout

OBJECTS	
KEYO	PRICE
AAA	\$10
BBB	\$21
CCC	\$23
DDD	\$25
EEE	\$33
FFF	\$34
GGG	\$44

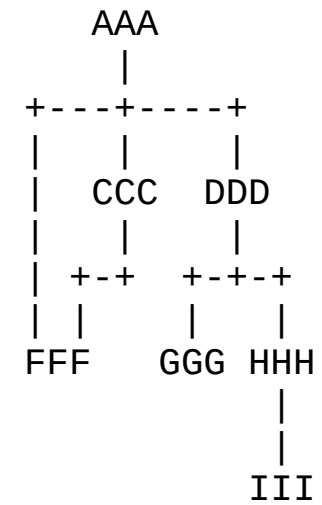
RELATIONSHIPS		
PKEY	CKEY	NUM
AAA	BBB	1
AAA	CCC	5
AAA	DDD	20
CCC	EEE	33
DDD	AAA	99
DDD	FFF	5
DDD	EEE	44
FFF	GGG	5



Balanced and Unbalanced Hierarchies



```
<< Balanced Hierarchy
      Unbalanced Hierarchy >>
```



Recursive Hierarchy - Sample Table and Layout

TROUBLE

```

+-----+
| PKEY | CKEY |
|-----|-----|
| AAA  | BBB  |
| AAA  | CCC  |
| AAA  | DDD  |
| CCC  | EEE  |
| DDD  | AAA  |
| DDD  | FFF  |
| DDD  | EEE  |
| FFF  | GGG  |
+-----+

```

<=== This row
points back to
the hierarchy
parent.

```

          AAA <-----+
          |             |
      +---+ +---+ +---+
      |   |   |   |   |
    BBB CCC DDD >--+
          |       |
      +--+ +--+ +--+
          |   |   |
          EEE FFF
              |
              GGG

```

Sample Table DDL

```

CREATE TABLE TROUBLE
(PKEY      CHAR(03)      NOT NULL
, CKEY      CHAR(03)      NOT NULL
, CONSTRAINT TBX1 PRIMARY KEY(PKEY, CKEY)
, CONSTRAINT TBC1 CHECK (PKEY <> CKEY)
, CONSTRAINT TBC2 CHECK (LOCATE('>', PKEY)=0)
, CONSTRAINT TBC3 CHECK (LOCATE('>', CKEY)=0));

```

Stop Recursion After "n" Levels

```

WITH PARENT (CKEY, LVL) AS
(
  SELECT DISTINCT PKEY, 0
    FROM   TROUBLE
   WHERE  PKEY = 'AAA'
  UNION ALL
  SELECT C.CKEY, P.LVL +1
    FROM   TROUBLE   C
          , PARENT   P
   WHERE  P.CKEY = C.PKEY
        AND P.LVL+1 < 4
)
SELECT CKEY, LVL
FROM   PARENT;

```

```

ANSWER
=====
CKEY  LVL
----  ---
AAA    0
BBB    1
CCC    1
DDD    1
EEE    2
AAA    2
EEE    2
FFF    2
BBB    3
CCC    3
DDD    3
GGG    3

```

```

TROUBLE
+-----+
| PKEY | CKEY |
|-----|-----|
| AAA  | BBB  |
| AAA  | CCC  |
| AAA  | DDD  |
| CCC  | EEE  |
| DDD  | AAA  |
| DDD  | FFF  |
| DDD  | EEE  |
| FFF  | GGG  |
+-----+

```

Stop Recursion After "n" Levels, Remove duplicates

```

WITH PARENT (CKEY, LVL) AS
(SELECT DISTINCT PKEY, 0
 FROM   TROUBLE
 WHERE  PKEY = 'AAA'
 UNION ALL
 SELECT C.CKEY, P.LVL +1
 FROM   TROUBLE   C
        ,PARENT   P
 WHERE  P.CKEY = C.PKEY
        AND P.LVL+1 < 4
 )
,NO_DUPS (CKEY, LVL, NUM) AS
(SELECT   CKEY, MIN(LVL), COUNT(*)
 FROM     PARENT
 GROUP BY CKEY
 )
SELECT CKEY, LVL, NUM
FROM   NO_DUPS;

```

```

ANSWER
=====
CKEY  LVL  NUM
----  ---  ---
AAA    0    2
BBB    1    2
CCC    1    2
DDD    1    2
EEE    2    2
AAA    2    1
GGG    3    1

```

```

TROUBLE
+-----+
| PKEY | CKEY |
| ---- | ---- |
| AAA  | BBB  |
| AAA  | CCC  |
| AAA  | DDD  |
| CCC  | EEE  |
| DDD  | AAA  |
| DDD  | FFF  |
| DDD  | EEE  |
| FFF  | GGG  |
+-----+

```

This row ==>
points back to
the hierarchy
parent.

Stop Recursion After "n" Levels, Show Data Paths

```

WITH PARENT (CKEY, LVL, PATH, LOC) AS
(
  SELECT DISTINCT PKEY, 0
    , VARCHAR(PKEY, 20)
    , 0
  FROM   TROUBLE
  WHERE  PKEY = 'AAA'
  UNION ALL
  SELECT C.CKEY, P.LVL + 1
    , P.PATH || '>' || C.CKEY
    , LOCATE(C.CKEY || '>', P.PATH)
  FROM   TROUBLE C
    , PARENT P
  WHERE  P.CKEY = C.PKEY
    AND  P.LVL + 1 < 4
)
SELECT CKEY, LVL, PATH, LOC
FROM   PARENT;

```

```

ANSWER
=====
CKEY  LVL  PATH                      LOC
-----
AAA    0  AAA                      0
BBB    1  AAA>BBB                      0
CCC    1  AAA>CCC                      0
DDD    1  AAA>DDD                      0
EEE    2  AAA>CCC>EEE                  0
AAA    2  AAA>DDD>AAA                  1
EEE    2  AAA>DDD>EEE                  0
FFF    2  AAA>DDD>FFF                  0
BBB    3  AAA>DDD>AAA>BBB            0
CCC    3  AAA>DDD>AAA>CCC            0
DDD    3  AAA>DDD>AAA>DDD            5
GGG    3  AAA>DDD>FFF>GGG            0

```

Recursive SQL, Ignore Data Loops (don't retrieve)

```

WITH PARENT (CKEY, LVL, PATH) AS
(
  SELECT DISTINCT PKEY, 0
    , VARCHAR(PKEY, 20)
  FROM   TROUBLE
  WHERE  PKEY = 'AAA'
  UNION ALL
  SELECT C.CKEY, P.LVL + 1
    , P.PATH || '|' > ' || C.CKEY
  FROM   TROUBLE    C
    , PARENT        P
  WHERE  P.CKEY = C.PKEY
    AND   LOCATE(C.CKEY || '|' > ' , P.PATH)=0
)
SELECT CKEY, LVL, PATH
FROM   PARENT;

```

```

ANSWER
=====
CKEY  LVL  PATH
-----
AAA    0  AAA
BBB    1  AAA>BBB
CCC    1  AAA>CCC
DDD    1  AAA>DDD
EEE    2  AAA>CCC>EEE
EEE    2  AAA>DDD>EEE
FFF    2  AAA>DDD>FFF
GGG    3  AAA>DDD>FFF>GGG

```

This row ==>
points back to
the hierarchy
parent.

```

TROUBLE
+-----+
| PKEY | CKEY |
| ---- | ---- |
| AAA  | BBB  |
| AAA  | CCC  |
| AAA  | DDD  |
| CCC  | EEE  |
| DDD  | AAA  |
| DDD  | FFF  |
| DDD  | EEE  |
| FFF  | GGG  |
+-----+

```

Recursive SQL, Mark Data Loops

```

WITH PARENT (CKEY, LVL, PATH) AS
(
  SELECT DISTINCT PKEY, 0
    , VARCHAR(PKEY, 20)
  FROM   TROUBLE
  WHERE  PKEY = 'AAA'
  UNION ALL
  SELECT CASE
    WHEN LOCATE(C.CKEY || '>', P.PATH) > 0
      THEN CHAR('>>>', 3)
    ELSE C.CKEY
      END
    , P.LVL + 1
    , P.PATH || '>' || C.CKEY
  FROM   TROUBLE    C
    , PARENT        P
  WHERE  P.CKEY = C.PKEY
)
SELECT CKEY, LVL, PATH
FROM   PARENT;

```

```

ANSWER
=====
CKEY  LVL  PATH
-----
AAA    0  AAA
BBB    1  AAA>BBB
CCC    1  AAA>CCC
DDD    1  AAA>DDD
EEE    2  AAA>CCC>EEE
>>>    2  AAA>DDD>AAA
EEE    2  AAA>DDD>EEE
FFF    2  AAA>DDD>FFF
GGG    3  AAA>DDD>FFF>GGG

```

Recursive SQL, List Data Loops

```

WITH PARENT (CKEY, PATH) AS
(
  SELECT DISTINCT PKEY
    , VARCHAR(PKEY, 20)
  FROM   TROUBLE
 WHERE  PKEY = 'AAA'
 UNION ALL
  SELECT CASE
    WHEN LOCATE(C.CKEY || '>', P.PATH) > 0
      THEN CHAR('>>>', 3)
    ELSE C.CKEY
    END
    , P.PATH || '>' || C.CKEY
  FROM   TROUBLE    C
    , PARENT        P
 WHERE  P.CKEY = C.PKEY
)
SELECT PATH
FROM   PARENT
WHERE  CKEY = '>>>';

```

```

ANSWER
=====
PATH
-----
AAA>DDD>AAA

```

```

TROUBLE
+-----+
| PKEY | CKEY |
+-----+-----+
| AAA  | BBB  |
| AAA  | CCC  |
| AAA  | DDD  |
| CCC  | EEE  |
| DDD  | AAA  |
| DDD  | FFF  |
| DDD  | EEE  |
| FFF  | GGG  |
+-----+

```

Recursive SQL, Set Error if Data Loop Found

```

WITH PARENT
(CKEY, LVL, PATH) AS
(SELECT DISTINCT
    PKEY
    ,0
    ,VARCHAR(PKEY,20)
FROM    TROUBLE
WHERE   PKEY = 'AAA'
UNION ALL
SELECT CASE
    WHEN LOCATE(C.CKEY|| '>',P.PATH) > 0
        THEN RAISE_ERROR('70001',
            'ERROR: LOOP IN DATABASE FOUND')
    ELSE C.CKEY
    END
    ,P.LVL +1
    ,P.PATH|| '>' || C.CKEY
FROM    TROUBLE    C
    ,PARENT        P
WHERE   P.CKEY = C.PKEY
)
SELECT CKEY, LVL, PATH
FROM    PARENT;

```

ANSWER	TROUBLE	
=====	+-----+	
CKEY...	PKEY	CKEY
----	----	----
<error>	AAA	BBB
	AAA	CCC
	AAA	DDD
	CCC	EEE
	DDD	AAA
	DDD	FFF
	DDD	EEE
	FFF	GGG
	+-----+	

Fun with SQL

Creating Made-up Data

Create One Row

```
WITH TEMP1 (COL1) AS
(VALUES      0
)
SELECT *
FROM   TEMP1;
```

```
ANSWER
=====
COL1
----
0
```

Create Multiple Rows

```
WITH TEMP1 (COL1, COL2, COL3) AS
(VALUES      (    0, 'AA', 0.00)
              , (    1, 'BB', 1.11)
              , (    2, 'CC', 2.22)
)
SELECT *
FROM   TEMP1;
```

```
ANSWER
=====
COL1 COL2 COL3
---- -
0 AA    0.00
1 BB    1.11
2 CC    2.22
```

Use Recursion to Make Array (1 of 2)

```

WITH TEMP1 (C0,C1,C2,C3,C4,C5,C6,C7,C8,C9) AS
(VALUES      ( 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
  UNION ALL
  SELECT C0+10, C1+10, C2+10, C3+10, C4+10
        ,C5+10, C6+10, C7+10, C8+10, C9+10
  FROM    TEMP1
  WHERE   C0+10 < 100
)
SELECT *
FROM    TEMP1;

```

Answer

C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Use Recursion to Make Array (2 of 2)

```

WITH TEMP1 (C0) AS
  (VALUES      ( 0)
   UNION ALL
   SELECT C0+10
   FROM   TEMP1
   WHERE  C0+10 < 100
  )
SELECT C0
      ,C0+1 AS C1, C0+2 AS C2, C0+3 AS C3, C0+4 AS C4, C0+5 AS C5
      ,C0+6 AS C6, C0+7 AS C7, C0+8 AS C8, C0+9 AS C9
FROM   TEMP1;

```

Answer

C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
---	---	---	---	---	---	---	---	---	---
0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Use RAND to Create Pseudo-Random Numbers

```

WITH TEMP1 (S1, R1) AS
  (VALUES (0, RAND(1))
   UNION ALL
   SELECT S1+1, RAND()
   FROM   TEMP1
   WHERE  S1+1 < 5
  )
SELECT SMALLINT(S1)      AS SEQ#
       ,DECIMAL(R1,5,3) AS RAN1
FROM   TEMP1;

```

```

ANSWER
=====
SEQ#    RAN1
-----
0      0.001
1      0.563
2      0.193
3      0.808
4      0.585

```

Note

- If initial RAND seeded, gets reproducible results

Make Differing Ranges of Random Numbers

```

WITH TEMP1 (S1, R1) AS
(VVALUES (0, RAND(2))
 UNION ALL
 SELECT  S1+1, RAND()
 FROM    TEMP1
 WHERE   S1+1 < 5
)
SELECT SMALLINT(S1) AS SEQ#
      ,SMALLINT(R1*10000) AS RAN2
      ,DECIMAL(R1,6,4) AS RAN1
      ,SMALLINT(R1*10) AS RAN3
FROM    TEMP1;

```

```

ANSWER
=====
SEQ#  RAN2  RAN1    RAN3
----  ----  -
0      13  0.0013    0
1    8916  0.8916    8
2    7384  0.7384    7
3    5430  0.5430    5
4    8998  0.8998    8

```

Converting RAND Output, Number to Character

```

WITH TEMP1 (S1, R1) AS
(VVALUES (0, RAND(2))
 UNION ALL
 SELECT  S1+1, RAND()
 FROM    TEMP1
 WHERE   S1+1 < 5
)
SELECT SMALLINT(S1) AS SEQ#
      , SMALLINT(R1*26+65) AS RAN2
      , CHR(SMALLINT(R1*26+65)) AS RAN3
      , CHAR(SMALLINT(R1*26)+65) AS RAN4
FROM    TEMP1;

```

```

ANSWER
=====
SEQ#  RAN2  RAN3  RAN4
----  -
0     65  A     65
1     88  X     88
2     84  T     84
3     79  O     79
4     88  X     88

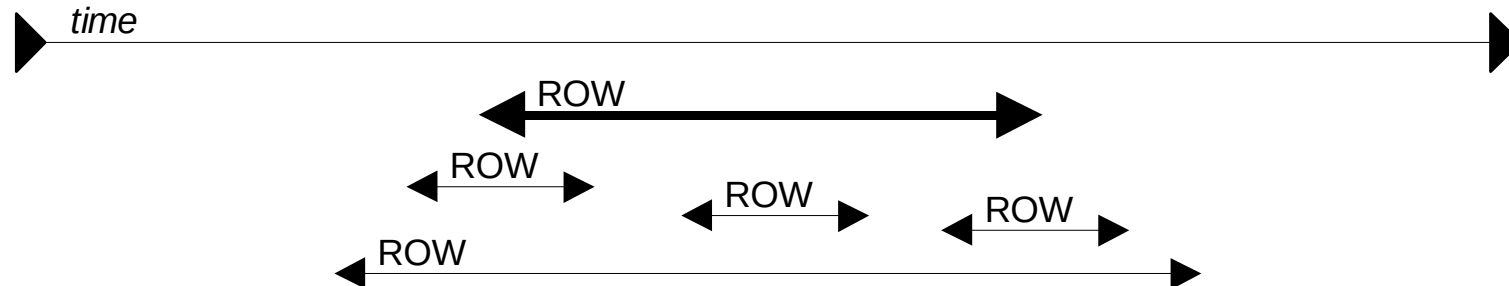
```

Time Series, Sample Table

```
CREATE TABLE TIME_SERIES
(KYY          CHAR(03)          NOT NULL
,BGN_DT       DATE              NOT NULL
,END_DT       DATE              NOT NULL
,CONSTRAINT TSX1 PRIMARY KEY(KYY,BGN_DT)
,CONSTRAINT TSC1 CHECK (KYY <> '')
,CONSTRAINT TSC2 CHECK (BGN_DT <= END_DT));
COMMIT;
```

```
INSERT INTO TIME_SERIES VALUES
('AAA', '1995-10-01', '1995-10-04'),
('AAA', '1995-10-06', '1995-10-06'),
('AAA', '1995-10-07', '1995-10-07'),
('AAA', '1995-10-15', '1995-10-19'),
('BBB', '1995-10-01', '1995-10-01'),
('BBB', '1995-10-03', '1995-10-03');
```

Overlapping Time-Series rows - Definition



Find Overlapping Rows in Time-Series

```

SELECT KYK
      ,BGN_DT
      ,END_DT
FROM   TIME_SERIES A
WHERE  EXISTS
      (SELECT *
      FROM   TIME_SERIES B
      WHERE  A.KYY      = B.KYY
            AND  A.BGN_DT <> B.BGN_DT
            AND  (A.BGN_DT BETWEEN B.BGN_DT AND B.END_DT
            OR  B.BGN_DT BETWEEN A.BGN_DT AND A.END_DT))
ORDER BY 1,2;

```

ANSWER
 =====
 <no rows>

TIME_SERIES

```

+-----+
| KYK | BGN_DT      | END_DT      |
| --- | - - - - - | - - - - - |
| AAA | 1995-10-01 | 1995-10-04 |
| AAA | 1995-10-06 | 1995-10-06 |
| AAA | 1995-10-07 | 1995-10-07 |
| AAA | 1995-10-15 | 1995-10-19 |
| BBB | 1995-10-01 | 1995-10-01 |
| BBB | 1995-10-03 | 1995-10-03 |
+-----+

```

Find Gap in Time-Series

```

SELECT  A.KYY
        ,A.BGN_DT
        ,A.END_DT
        ,B.BGN_DT
        ,B.END_DT
        ,DAYS(B.BGN_DT) -
        DAYS(A.END_DT)
          AS DIFF
FROM    TIME_SERIES A
        ,TIME_SERIES B
WHERE   A.KYY      = B.KYY
        AND  A.END_DT < B.BGN_DT - 1 DAY
        AND  NOT EXISTS
          (SELECT *
           FROM   TIME_SERIES Z
           WHERE  Z.KYY      = A.KYY
                 AND  Z.KYY      = B.KYY
                 AND  Z.BGN_DT > A.BGN_DT
                 AND  Z.BGN_DT < B.BGN_DT)
ORDER BY 1,2;

```

```

TIME_SERIES
+-----+
| KYY | BGN_DT      | END_DT      |
| --- | - - - - - | - - - - - |
| AAA | 1995-10-01 | 1995-10-04 |
| AAA | 1995-10-06 | 1995-10-06 |
| AAA | 1995-10-07 | 1995-10-07 |
| AAA | 1995-10-15 | 1995-10-19 |
| BBB | 1995-10-01 | 1995-10-01 |
| BBB | 1995-10-03 | 1995-10-03 |
+-----+

```

KEYCOL	BGN_DT	END_DT	BGN_DT	END_DT	DIFF
-----	-----	-----	-----	-----	----
AAA	10/01/1995	10/04/1995	10/06/1995	10/06/1995	2
AAA	10/07/1995	10/07/1995	10/15/1995	10/19/1995	8
BBB	10/01/1995	10/01/1995	10/03/1995	10/03/1995	2

Find Gap in Time-Series

```

SELECT  A.KYY
        ,A.END_DT + 1 DAY
          AS BGN_GAP
        ,B.BGN_DT - 1 DAY
          AS END_GAP
        ,(DAYS(B.BGN_DT) -
          DAYS(A.END_DT) - 1)
          AS GAP_SIZE
FROM    TIME_SERIES A
        ,TIME_SERIES B
WHERE   A.KYY      = B.KYY
        AND  A.END_DT < B.BGN_DT - 1 DAY
        AND  NOT EXISTS
          (SELECT *
           FROM   TIME_SERIES Z
           WHERE  Z.KYY      = A.KYY
                 AND  Z.KYY      = B.KYY
                 AND  Z.BGN_DT > A.BGN_DT
                 AND  Z.BGN_DT < B.BGN_DT)
ORDER BY 1,2;

```

```

TIME_SERIES
+-----+
| KYY | BGN_DT      | END_DT      |
| --- | - - - - - | - - - - - |
| AAA | 1995-10-01 | 1995-10-04 |
| AAA | 1995-10-06 | 1995-10-06 |
| AAA | 1995-10-07 | 1995-10-07 |
| AAA | 1995-10-15 | 1995-10-19 |
| BBB | 1995-10-01 | 1995-10-01 |
| BBB | 1995-10-03 | 1995-10-03 |
+-----+

```

KEYCOL	BGN_GAP	END_GAP	GAP_SIZE
AAA	10/05/1995	10/05/1995	1
AAA	10/08/1995	10/14/1995	7
BBB	10/02/1995	10/02/1995	1

Show Each Day in Time-Series Gap

```

WITH TEMP
(KYY, GAP_DT, GSIZE) AS
(SELECT A.KYY
      ,A.END_DT + 1 DAY
      ,(DAYS(B.BGN_DT) -
        DAYS(A.END_DT) - 1)
FROM   TIME_SERIES A
      ,TIME_SERIES B
WHERE  A.KYY      = B.KYY
      AND A.END_DT < B.BGN_DT - 1 DAY
      AND NOT EXISTS
        (SELECT *
         FROM   TIME_SERIES Z
         WHERE  Z.KYY      = A.KYY
              AND Z.KYY      = B.KYY
              AND Z.BGN_DT > A.BGN_DT
              AND Z.BGN_DT < B.BGN_DT)
UNION ALL
SELECT KYY
      ,GAP_DT + 1 DAY
      ,GSIZE - 1
FROM   TEMP
WHERE  GSIZE > 1
)
SELECT *
FROM   TEMP
ORDER BY 1,2;

```

```

TIME_SERIES
+-----+
| KYY | BGN_DT      | END_DT      |
| --- | - - - - - | - - - - - |
| AAA | 1995-10-01 | 1995-10-04 |
| AAA | 1995-10-06 | 1995-10-06 |
| AAA | 1995-10-07 | 1995-10-07 |
| AAA | 1995-10-15 | 1995-10-19 |
| BBB | 1995-10-01 | 1995-10-01 |
| BBB | 1995-10-03 | 1995-10-03 |
+-----+

```

```

ANSWER
=====
KEYCOL GAP_DT      GSIZE
-----
AAA    10/05/1995    1
AAA    10/08/1995    7
AAA    10/09/1995    6
AAA    10/10/1995    5
AAA    10/11/1995    4
AAA    10/12/1995    3
AAA    10/13/1995    2
AAA    10/14/1995    1
BBB    10/02/1995    1

```

Covert Character to Numeric

```
WITH TEMP1 (C1) AS
(VALUES '123 ' ',' 345 ' ',' 567')
SELECT C1
      ,DOUBLE(C1)      AS DBL
      ,DECIMAL(C1,3)   AS DEC
      ,SMALLINT(C1)    AS SML
      ,INTEGER(C1)     AS INT
FROM   TEMP1;
```

```
ANSWER (numbers shortened)
=====
C1      DBL              DEC    SML    INT
-----
123      +1.2300E+2    123.    123    123
345      +3.4500E+2    345.    345    345
567      +5.6700E+2    567.    567    567
```

Valid Input Types

INPUT STRING	COMPATIBLE FUNCTIONS
=====	=====
" 1234"	DOUBLE, DECIMAL, INTEGER, SMALLINT, BIGINT
" 12.4"	DOUBLE, DECIMAL
" 12E4"	DOUBLE

Covert Character to Numeric, Check Input

```

WITH TEMP1 (C1) AS (VALUES ' 123', '456 ', ' 1 2', ' 33%', NULL)
SELECT C1
      , CHAR(RTRIM(LTRIM(C1)), 4)
      , CHAR(TRANSLATE(RTRIM(LTRIM(C1)), '$', ' 0123456789'), 4)
      , CASE
          WHEN TRANSLATE(RTRIM(LTRIM(C1)), '$', ' 0123456789') = ''
          THEN SMALLINT(C1)
          ELSE SMALLINT(0)
        END
FROM   TEMP1;

```

Answer

C1	2	3	4
----	-----	-----	-----
123	123		123
456	456		456
1 2	1 2	\$	0
33%	33%	%	0
-	-	-	0

Make Chart Using SQL

```
WITH TEMP1 (COL1) AS (VALUES 12, 22, 33, 16, 0, 44, 15, 15)
SELECT COL1
       ,SUBSTR(TRANSLATE(CHAR(' ',50),'*', ' '),1,COL1)
       AS PRETTY_CHART
FROM   TEMP1;
```

Answer

COL1	PRETTY_CHART
12	*****
22	*****
33	*****
16	*****
0	
44	*****
15	*****
15	*****

Multiple Counts in One Pass

Group By

```
SELECT    SEX
          ,COUNT(*) AS NUM
FROM      STATS
GROUP BY  SEX
ORDER BY  SEX;
```

ANSWER >>

```
SEX NUM
--- ---
F 595
M 405
```

Common Table Expression

```
WITH F (F) AS (SELECT COUNT(*) FROM STATS WHERE SEX = 'F')
     ,M (M) AS (SELECT COUNT(*) FROM STATS WHERE SEX = 'M')
SELECT  F, M
FROM    F, M;
```

ANSWER

```
=====
F      M
--- ---
595 405
```

CASE and SUM

```
SELECT    COUNT(*)
          ,SUM(CASE SEX WHEN 'F' THEN 1 ELSE 0 END) AS TOTAL
          ,SUM(CASE SEX WHEN 'M' THEN 1 ELSE 0 END) AS FEMALE
          ,SUM(CASE SEX WHEN 'M' THEN 1 ELSE 0 END) AS MALE
FROM      STATS;
```

Multiple Counts From Same Row, SQL

```

WITH CATEGORY (CAT,SUBCAT,DEPT) AS
(VVALUES ('1ST','ROWS IN TABLE ',''))
      ,('2ND','SALARY > $20K ',''))
      ,('3RD','NAME LIKE ABC%',''))
      ,('4TH','NUMBER MALES ',''))
UNION
SELECT '5TH',DEPTNAME,DEPTNO
FROM DEPARTMENT
)
SELECT   XXX.CAT           AS "CATEGORY"
        ,XXX.SUBCAT       AS "SUBCATEGORY/DEPT"
        ,SUM(XXX.FOUND) AS "#ROWS"
FROM     (SELECT   CAT.CAT
            ,CAT.SUBCAT
            ,CASE
                WHEN EMP.EMPNO IS NULL THEN 0
                ELSE 1
            END AS FOUND
        FROM     CATEGORY CAT
            LEFT OUTER JOIN
            EMPLOYEE EMP
            ON     CAT.SUBCAT = 'ROWS IN TABLE'
            OR     (CAT.SUBCAT = 'NUMBER MALES'
                AND EMP.SEX = 'M')
            OR     (CAT.SUBCAT = 'SALARY > $20K'
                AND EMP.SALARY > 20000)
            OR     (CAT.SUBCAT = 'NAME LIKE ABC%'
                AND EMP.FIRSTNME LIKE 'ABC%')
            OR     (CAT.DEPT <> ''
                AND CAT.DEPT = EMP.WORKDEPT)
        )AS XXX
GROUP BY XXX.CAT
        ,XXX.SUBCAT
ORDER BY 1,2;

```

Multiple Counts From Same Row, Answer

CATEGORY	SUBCATEGORY/DEPT	#ROWS
-----	-----	-----
1ST	ROWS IN TABLE	32
2ND	SALARY > \$20K	25
3RD	NAME LIKE ABC%	0
4TH	NUMBER MALES	19
5TH	ADMINISTRATION SYSTEMS	6
5TH	DEVELOPMENT CENTER	0
5TH	INFORMATION CENTER	3
5TH	MANUFACTURING SYSTEMS	9
5TH	OPERATIONS	5
5TH	PLANNING	1
5TH	SOFTWARE SUPPORT	4
5TH	SPIFFY COMPUTER SERVICE DIV.	3
5TH	SUPPORT SERVICES	1

Find Missing Rows / Count all Values (1 of 3)

Count staff joined per year

```
SELECT    YEARS
          , COUNT(*) AS #STAFF
FROM      STAFF
WHERE     UCASE(NAME) LIKE '%E%'
AND       YEARS          <= 5
GROUP BY YEARS;
```

```
ANSWER
=====
YEARS  #STAFF
-----
      1      1
      4      2
      5      3
```

Count staff joined per year, all years - use VALUES

```
WITH LIST_YEARS (YEAR#) AS
  (VALUES (0), (1), (2), (3), (4), (5)
  )
SELECT    YEAR#                AS YEARS
          , COALESCE(#STFF,0) AS #STAFF
FROM      LIST_YEARS
LEFT OUTER JOIN
  (SELECT    YEARS
            , COUNT(*) AS #STFF
  FROM      STAFF
  WHERE     UCASE(NAME) LIKE '%E%'
  AND       YEARS          <= 5
  GROUP BY YEARS
  )AS XXX
ON        YEAR# = YEARS
ORDER BY 1;
```

```
ANSWER
=====
YEARS  #STAFF
-----
      0      0
      1      1
      2      0
      3      0
      4      2
      5      3
```

Find Missing Rows / Count all Values (2 of 3)

Count staff joined per year, all years - use Recursion

```
WITH LIST_YEARS (YEAR#) AS
  (VALUES  SMALLINT(0)
   UNION  ALL
   SELECT  YEAR# + 1
   FROM    LIST_YEARS
   WHERE   YEAR# < 5)
SELECT    YEAR#                                AS YEARS
          , COALESCE(#STFF, 0) AS #STAFF
FROM      LIST_YEARS
LEFT OUTER JOIN
  (SELECT  YEARS
          , COUNT(*) AS #STFF
   FROM    STAFF
   WHERE   UCASE(NAME) LIKE '%E%'
   AND     YEARS          <= 5
   GROUP BY YEARS
  ) AS XXX
ON        YEAR# = YEARS
ORDER BY 1;
```

```
ANSWER
=====
YEARS #STAFF
-----
      0      0
      1      1
      2      0
      3      0
      4      2
      5      3
```

Find Missing Rows / Count all Values (3 of 3)

List years when no staff joined

```

WITH LIST_YEARS (YEAR#) AS
  (VALUES  SMALLINT(0)
   UNION  ALL
   SELECT  YEAR# + 1
   FROM    LIST_YEARS
   WHERE   YEAR# < 5)
SELECT    YEAR#
FROM      LIST_YEARS Y
WHERE     NOT EXISTS
  (SELECT  *
   FROM    STAFF S
   WHERE   UCASE(S.NAME) LIKE '%E%'
   AND     S.YEARS           =  Y.YEAR#)
ORDER BY 1;

```

ANSWER

=====

YEAR#

0

2

3

Reverse Character Field Using Recursion

```

WITH TEMP (OLD_NAME, NEW_NAME, REST) AS
(
  SELECT      NAME
              , SUBSTR(NAME, LENGTH(NAME))
              , SUBSTR(NAME, 1, LENGTH(NAME) - 1)
    FROM      STAFF
   WHERE      ID < 100
   UNION
  SELECT      OLD_NAME
              , NEW_NAME CONCAT
              , SUBSTR(REST, LENGTH(REST))
              , SUBSTR(REST, 1, LENGTH(REST) - 1)
    FROM      TEMP
   WHERE      LENGTH(REST) > 0
)
SELECT      OLD_NAME
           , NEW_NAME
  FROM      TEMP
 WHERE      REST = ''
 ORDER BY  OLD_NAME;

```

```

ANSWER
=====
OLD_NAME  NEW_NAME
-----
Hanes     senaH
James     semaJ
Koonitz   ztinook
Marenghi  ihgneraM
O'Brien  neirB'O
Pernal    lanreP
Quigley   yelgiuQ
Rothman   namhtoR
Sanders   srednaS

```

Notes

- **Start:** Put last byte in NAME in NEW_NAME, rest of name in REST
- **Recursively:** Add last byte in REST to NEW_NAME, put rest in REST
- **Finish:** Select those rows where REST is empty

Quirks in SQL

Timestamp Comparison

Incorrect

```
WITH TEMP1 (C1,T1,T2) AS (VALUES
    ('A'
    ,TIMESTAMP('1996-05-01-24.00.00.000000')
    ,TIMESTAMP('1996-05-02-00.00.00.000000') ))
SELECT C1
FROM   TEMP1
WHERE  T1 = T2;
```

ANSWER
 =====
 <no rows>

Correct

```
WITH TEMP1 (C1,T1,T2) AS (VALUES
    ('A'
    ,TIMESTAMP('1996-05-01-24.00.00.000000')
    ,TIMESTAMP('1996-05-02-00.00.00.000000') ))
SELECT C1
FROM   TEMP1
WHERE  T1 + 0 MICROSECOND = T2 + 0 MICROSECOND;
```

ANSWER
 =====
 C1
 --
 A

No Rows Match - Various queries and their answers (1 of 2)

SELECT	CREATOR	ANSWER
FROM	SYSIBM.SYSTABLES	=====
WHERE	CREATOR = 'ZZZ';	<no row>

SELECT	MAX(CREATOR)	ANSWER
FROM	SYSIBM.SYSTABLES	=====
WHERE	CREATOR = 'ZZZ';	<null>

SELECT	MAX(CREATOR)	ANSWER
FROM	SYSIBM.SYSTABLES	=====
WHERE	CREATOR = 'ZZZ'	<no row>
HAVING	MAX(CREATOR) IS NOT NULL;	

SELECT	MAX(CREATOR)	ANSWER
FROM	SYSIBM.SYSTABLES	=====
WHERE	CREATOR = 'ZZZ'	<no row>
HAVING	MAX(CREATOR) = 'ZZZ';	

Notes

- If no column function, row returned on if one matches
- If column function (and no HAVING or GROUP BY), always get row
- If column function and HAVING, row returned only if HAVING predicate true
- If column function and GROUP BY row returned on if one matches

No Rows Match - Various queries and their answers (2 of 2)

SELECT	MAX(CREATOR)	ANSWER
FROM	SYSIBM.SYSTABLES	=====
WHERE	CREATOR = 'ZZZ'	<no row>
GROUP BY	CREATOR;	

SELECT	CREATOR	ANSWER
FROM	SYSIBM.SYSTABLES	=====
WHERE	CREATOR = 'ZZZ'	<no row>
GROUP BY	CREATOR;	

SELECT	COUNT(*)	ANSWER
FROM	SYSIBM.SYSTABLES	=====
WHERE	CREATOR = 'ZZZ'	<no row>
GROUP BY	CREATOR;	

SELECT	COUNT(*)	ANSWER
FROM	SYSIBM.SYSTABLES	=====
WHERE	CREATOR = 'ZZZ';	0

Notes

- If no column function, row returned on if one matches
- If column function (and no HAVING or GROUP BY), always get row
- If column function and HAVING, row returned only if HAVING predicate true
- If column function and GROUP BY row returned on if one matches

Get Random Rows (1 of 3)

Incorrect

```
SELECT    ID
          ,NAME
FROM      STAFF
WHERE     ID  <= 100
        AND ID  = (INT(RAND()* 10) * 10) + 10
ORDER BY ID;
```

```
ANSWER
=====
ID NAME
-- -----
30 Marenghi
60 Quigley
```

Explanation

```
WITH TEMP AS
(
  SELECT    ID
            ,NAME
            ,(INT(RAND(0)* 10) * 10) + 10 AS RAN
  FROM      STAFF
  WHERE     ID <= 100
)
SELECT     T.*
          ,CASE ID
            WHEN RAN THEN 'Y'
            ELSE          ' '
          END AS EQL
FROM       TEMP T
ORDER BY  ID;
```

```
ANSWER
=====
ID  NAME      RAN EQL
---  -
10 Sanders    10 Y
20 Pernal     30
30 Marenghi   70
40 O'Brien    10
50 Hanes      30
60 Quigley    40
70 Rothman    30
80 James     100
90 Koonitz    40
100 Plotz    100 Y
```

Get Random Rows (2 of 3)

Gets "n" random rows - not necessarily distinct

```
WITH
STAFF_NUMBERED AS
  (SELECT  S.*
           ,ROW_NUMBER() OVER() AS ROW#
    FROM    STAFF S
   WHERE   ID <= 100
  ),
COUNT_ROWS AS
  (SELECT  MAX(ROW#) AS #ROWS
    FROM    STAFF_NUMBERED
  ),
RANDOM_VALUES (RAN#) AS
  (VALUES (RAND())
         , (RAND())
         , (RAND())
  ),
ROWS_TO_GET AS
  (SELECT  INT(RAN# * #ROWS) + 1 AS GET_ROW
    FROM    RANDOM_VALUES
           ,COUNT_ROWS
  )
SELECT    ID
         ,NAME
  FROM      STAFF_NUMBERED
         ,ROWS_TO_GET
 WHERE     ROW# = GET_ROW
 ORDER BY  ID;
```

```
ANSWER
=====
ID   NAME
---  -
 10 Sanders
 20 Pernal
 90 Koonitz
```

Get Random Rows (3 of 3)

Gets "n" random rows - distinct

```
SELECT    ID
          ,NAME
FROM      (SELECT S.*
           ,ROW_NUMBER() OVER(ORDER BY RAND()) AS R
           FROM    STAFF S
           WHERE   ID <= 100
           )AS XXX
WHERE     R <= 3
ORDER BY ID;
```

```
ANSWER
=====
ID NAME
-- -----
10 Sanders
40 O'Brien
60 Quigley
```

Degrees of Randomness

- Random number of rows returned
- Fixed number of random rows, but possibly the same row returned twice
- Fixed number of random rows, each row distinct

Date/Time Manipulation (1 of 2)

Wrong

```
WITH TEMP1 (BGN_TSTAMP, ELP_SEC) AS
(VVALUES (TIMESTAMP('2001-01-15-01.02.03.000000'), 1.234)
        , (TIMESTAMP('2001-01-15-01.02.03.123456'), 1.234)
)
SELECT    BGN_TSTAMP
        , ELP_SEC
        , BGN_TSTAMP + ELP_SEC SECONDS AS END_TSTAMP
FROM      TEMP1;
```

ANSWER

=====

BGN_TSTAMP	ELP_SEC	END_TSTAMP
-----	-----	-----
2001-01-15-01.02.03.000000	1.234	2001-01-15-01.02.04.000000
2001-01-15-01.02.03.123456	1.234	2001-01-15-01.02.04.123456

Notes

- DB2 ignores fractional values in date/time calculations
- To get correct answer, integer value of small time unit

Date/Time Manipulation (2 of 2)

Right

```
WITH TEMP1 (BGN_TSTAMP, ELP_SEC) AS
(VVALUES (TIMESTAMP('2001-01-15-01.02.03.000000'), 1.234)
        , (TIMESTAMP('2001-01-15-01.02.03.123456'), 1.234)
)
SELECT   BGN_TSTAMP
        , ELP_SEC
        , BGN_TSTAMP + (ELP_SEC * 1E6) MICROSECONDS AS END_TSTAMP
FROM     TEMP1;
```

ANSWER

=====

BGN_TSTAMP	ELP_SEC	END_TSTAMP
-----	-----	-----
2001-01-15-01.02.03.000000	1.234	2001-01-15-01.02.04.234000
2001-01-15-01.02.03.123456	1.234	2001-01-15-01.02.04.357456

Notes

- Previous query Used "n.nnn" seconds
- Above query Uses "nnnn" microseconds

Use LIKE Predicate

On CHAR Field

```
WITH TEMP1 (C0,C1,V1) AS (VALUES
    ('A',CHAR(' ',1),VARCHAR(' ',1)),
    ('B',CHAR(' ',1),VARCHAR(' ',1)))
SELECT C0
FROM   TEMP1
WHERE  C1 = V1
      AND C1 LIKE ' ';
```

```
ANSWER
=====
      C0
      --
      A
      B
```

On VARCHAR Field

```
WITH TEMP1 (C0,C1,V1) AS (VALUES
    ('A',CHAR(' ',1),VARCHAR(' ',1)),
    ('B',CHAR(' ',1),VARCHAR(' ',1)))
SELECT C0
FROM   TEMP1
WHERE  C1 = V1
      AND V1 LIKE ' ';
```

```
ANSWER
=====
      C0
      --
      A
```

Use RTRIM to remove trailing blanks

```
WITH TEMP1 (C0,C1,V1) AS (VALUES
    ('A',CHAR(' ',1),VARCHAR(' ',1)),
    ('B',CHAR(' ',1),VARCHAR(' ',1)))
SELECT C0
FROM   TEMP1
WHERE  C1 = V1
      AND RTRIM(V1) LIKE ' ';
```

```
ANSWER
=====
      C0
      --
      A
      B
```

Comparing Weeks

Compare week 33 over 10 years

```

WITH TEMP1 (YYMMDD) AS
(VALUES DATE( '2000-01-01' )
 UNION ALL
 SELECT YYMMDD + 1 DAY
 FROM TEMP1
 WHERE YYMMDD < '2010-12-31'
 )
SELECT      YY                                AS YEAR
           , CHAR(MIN(YYMMDD), ISO) AS MIN_DT
           , CHAR(MAX(YYMMDD), ISO) AS MAX_DT
FROM        (SELECT YYMMDD
              , YEAR(YYMMDD) YY
              , WEEK(YYMMDD) WK
              FROM TEMP1
              WHERE WEEK(YYMMDD) = 33
            )AS XXX
GROUP BY YY
         ,WK;

```

ANSWER

```

=====
YEAR MIN_DT      MAX_DT
---- -
2000 2000-08-06 2000-08-12
2001 2001-08-12 2001-08-18
2002 2002-08-11 2002-08-17
2003 2003-08-10 2003-08-16
2004 2004-08-08 2004-08-14
2005 2005-08-07 2005-08-13
2006 2006-08-13 2006-08-19
2007 2007-08-12 2007-08-18
2008 2008-08-10 2008-08-16
2009 2009-08-09 2009-08-15
2010 2010-08-08 2010-08-14

```

DB2 Truncation vs. Rounding

Truncation

```
SELECT  SUM(INTEGER(SALARY)) AS S1
        , INTEGER(SUM(SALARY)) AS S2
FROM    STAFF;
```

```
ANSWER
=====
S1      S2
-----
583633  583647
```

Rounding

```
SELECT  SUM(INTEGER(ROUND(SALARY, -1))) AS S1
        , INTEGER(SUM(SALARY)) AS S2
FROM    STAFF;
```

```
ANSWER
=====
S1      S2
-----
583640  583647
```

Case WHEN Processing

Incorrect

```
SELECT  LASTNAME
        ,SEX
        ,CASE
            WHEN SEX >= 'F' THEN 'FEM'
            WHEN SEX >= 'M' THEN 'MAL'
        END AS SXX
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%'
ORDER BY 1;
```

ANSWER

```
=====
LASTNAME  SX  SXX
-----  --  ---
JEFFERSON  M  FEM
JOHNSON    F  FEM
JONES      M  FEM
```

Correct

```
SELECT  LASTNAME
        ,SEX
        ,CASE
            WHEN SEX >= 'M' THEN 'MAL'
            WHEN SEX >= 'F' THEN 'FEM'
        END AS SXX
FROM    EMPLOYEE
WHERE   LASTNAME LIKE 'J%'
ORDER BY 1;
```

ANSWER

```
=====
LASTNAME  SX  SXX
-----  --  ---
JEFFERSON  M  MAL
JOHNSON    F  FEM
JONES      M  MAL
```

Division and Average

```
SELECT  AVG(SALARY) / AVG(COMM) AS A1
        ,AVG(SALARY / COMM)      AS A2
FROM    STAFF;
```

```
ANSWER >>>  A1  A2
              --  ---
              32  61.98
```

1st Field

- Does division **AFTER** average
- Is probably what the user wants

2nd Field

- Does division **BEFORE** average
- Does **NOT** weight rows for absolute size of values
- Is probably incorrect

DATE Output Order

Year, Month, Day

```
SELECT    HIREDATE
FROM      EMPLOYEE
WHERE     HIREDATE < '1960-01-01'
ORDER BY 1;
```

```
ANSWER
=====
05/05/1947
08/17/1949
05/16/1958
```

Month, Day, Year

```
SELECT    CHAR(HIREDATE, USA)
FROM      EMPLOYEE
WHERE     HIREDATE < '1960-01-01'
ORDER BY 1;
```

```
ANSWER
=====
05/05/1947
05/16/1958
08/17/1949
```

Note

- To get rows in DATE sequence, order on the DATE field
- Do NOT order on CHARACTER equivalent of DATE field

Multiply Floating-point Number by Ten

```

WITH TEMP (F1) AS
(VVALUES FLOAT(1.23456789)
 UNION ALL
 SELECT F1 * 10
 FROM    TEMP
 WHERE   F1 < 1E18)
SELECT F1          AS FLOAT1
      ,DEC(F1,19) AS DECIMAL1
      ,BIGINT(F1) AS BIGINT1
FROM    TEMP;

```

FLOAT1	DECIMAL1	BIGINT1
-----	-----	-----
+1.234567890000000E+000	1.	1
+1.234567890000000E+001	12.	12
+1.234567890000000E+002	123.	123
+1.234567890000000E+003	1234.	1234
+1.234567890000000E+004	12345.	12345
+1.234567890000000E+005	123456.	123456
+1.234567890000000E+006	1234567.	1234567
+1.234567890000000E+007	12345678.	12345678
+1.234567890000000E+008	123456789.	123456788
+1.234567890000000E+009	1234567890.	1234567889
+1.234567890000000E+010	12345678900.	12345678899
+1.234567890000000E+011	123456789000.	123456788999
+1.234567890000000E+012	1234567890000.	1234567889999
+1.234567890000000E+013	12345678900000.	12345678899999
+1.234567890000000E+014	123456789000000.	123456788999999
+1.234567890000000E+015	1234567890000000.	1234567889999999
+1.234567890000000E+016	12345678900000000.	12345678899999998
+1.234567890000000E+017	123456789000000000.	1234567889999999984
+1.234567890000000E+018	1234567890000000000.	12345678899999999744

Two Numbers That Look Equal, But Aren't Equal

Number Representation

```
WITH TEMP (F1,F2) AS
(VALUES (FLOAT(1.23456789E1 * 10 * 10 * 10 * 10 * 10 * 10 * 10)
        ,FLOAT(1.23456789E8)))
SELECT F1
       ,F2
FROM   TEMP
WHERE  F1 <> F2;
```

ANSWER

```
=====
F1                                F2
-----
+1.234567890000000E+008 +1.234567890000000E+008
```

HEX Representation

```
WITH TEMP (F1,F2) AS
(VALUES (FLOAT(1.23456789E1 * 10 * 10 * 10 * 10 * 10 * 10 * 10)
        ,FLOAT(1.23456789E8)))
SELECT HEX(F1) AS HEX_F1
       ,HEX(F2) AS HEX_F2
FROM   TEMP
WHERE  F1 <> F2;
```

ANSWER

```
=====
HEX_F1                                HEX_F2
-----
FFFFFFFF53346F9D41 000000054346F9D41
```

Comparing Float and Decimal Division

WITH

```
TEMP1 (DEC1, DBL1) AS
  (VALUES (DECIMAL(1),DOUBLE(1)))
,TEMP2 (DEC1, DEC2, DBL1, DBL2) AS
  (SELECT DEC1
        ,DEC1 / 3 AS DEC2
        ,DBL1
        ,DBL1 / 3 AS DBL2
   FROM   TEMP1)
SELECT *
FROM     TEMP2
WHERE    DBL2 <> DEC2;
```

ANSWER (1 row returned)

```
=====
DEC1 = 1.0
DEC2 = 0.33333333333333333333
DBL1 = +1.0000000000000000E+000
DBL2 = +3.333333333333333E-001
```

Comparing Float and Decimal Multiplication

```

WITH TEMP (F1, D1) AS
(VALUES (FLOAT(1.23456789)
        ,DEC(1.23456789,20,10))
 UNION ALL
 SELECT F1 * 10
        ,D1 * 10
 FROM    TEMP
 WHERE   F1 < 1E9
 )
 SELECT F1
        ,D1
        ,CASE
            WHEN D1 = F1 THEN 'SAME'
            ELSE             'DIFF'
        END AS COMPARE
 FROM    TEMP;

```

F1	D1	COMPARE
-----	-----	-----
+1.234567890000000E+000	1.2345678900	SAME
+1.234567890000000E+001	12.3456789000	SAME
+1.234567890000000E+002	123.4567890000	DIFF
+1.234567890000000E+003	1234.5678900000	DIFF
+1.234567890000000E+004	12345.6789000000	DIFF
+1.234567890000000E+005	123456.7890000000	DIFF
+1.234567890000000E+006	1234567.8900000000	SAME
+1.234567890000000E+007	12345678.9000000000	DIFF
+1.234567890000000E+008	123456789.0000000000	DIFF
+1.234567890000000E+009	1234567890.0000000000	DIFF

Internal Representation of "one tenth" in Floating-Point

```
WITH TEMP (F1) AS
(VVALUES FLOAT(0.1))
SELECT F1
      , HEX(F1) AS HEX_F1
FROM   TEMP;
```

```
ANSWER
=====
F1                                HEX_F1
-----
+1.0000000000000000E-001  9A9999999999B93F
```