St. Louis Gateway/400 Group

SQL: Join and Aggregate Value Techniques

- Speaker: Thibault Dambrine
- Time: Thursday January 12th 2017 meeting
- Web address: www.tylogix.com

SQL on iSeries:

Join and Aggregate Value Techniques

Higher Productivity iSeries Programming Using SQL

By Thibault Dambrine

SQL JOIN PROGRAMMING TECHNIQUES

- SQL joins
- SQL update and deletes
- Key (index) considerations
- Aggregate values
- Performance considerations

THE CASE FOR JOINS!

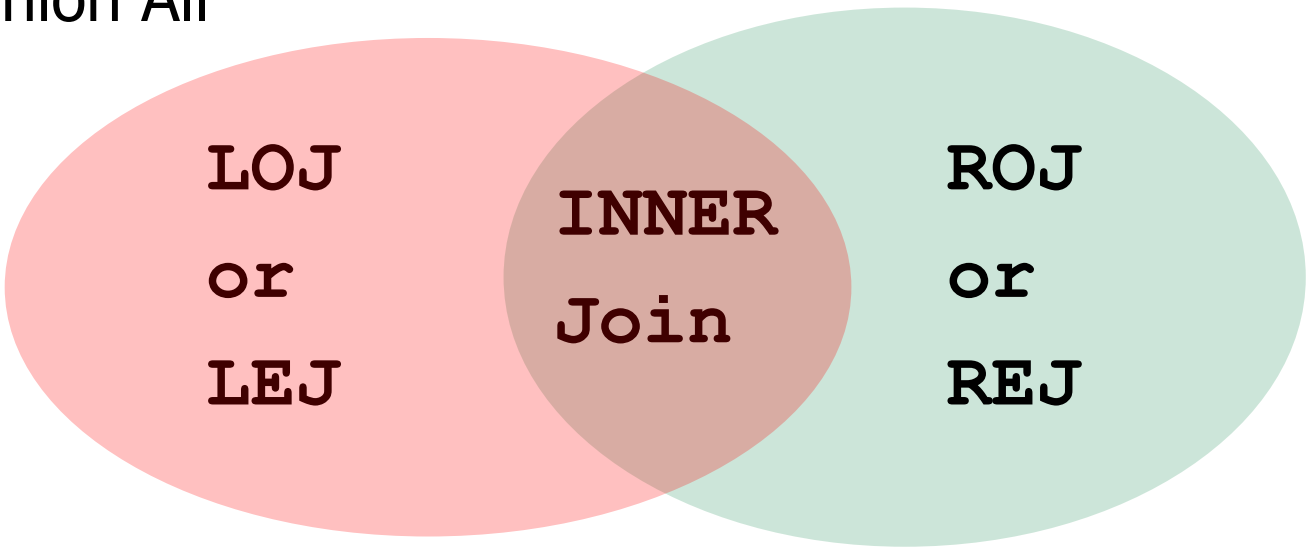
Most powerful way to join two or more tables

Powerful: Efficient, Clearer Code

Faster than Correlated Sub-Queries

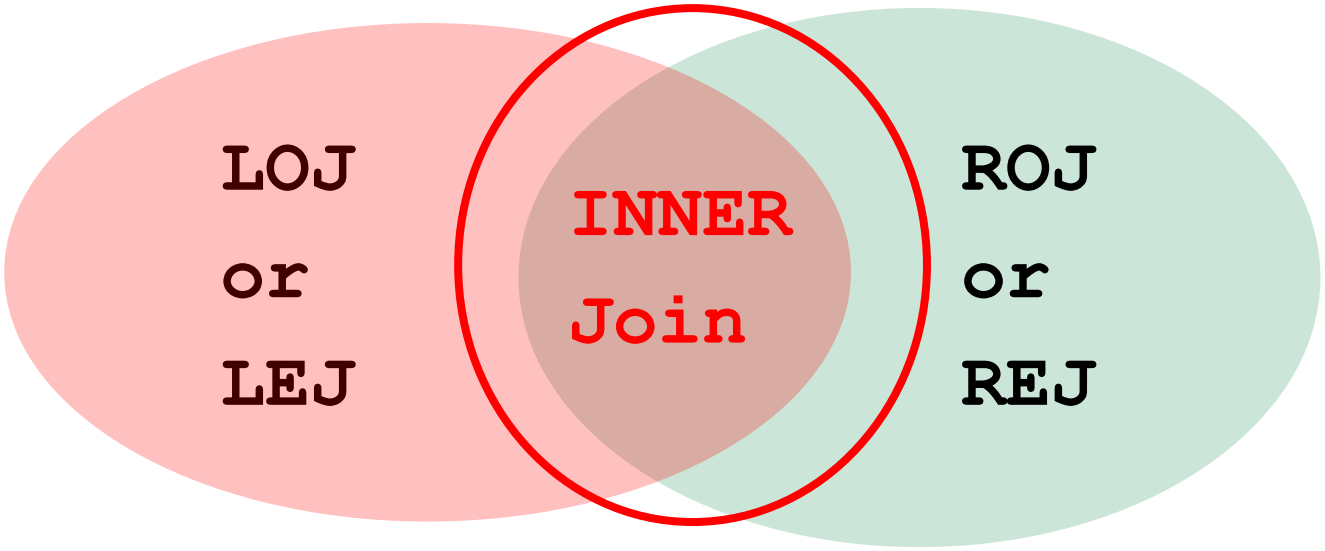
SQL Joins

- Join or Inner Join
- Left or Right Join or Left/Right Outer Join
- Left or Right Exception Join
- Union or Union All
- Cross Join
- Self-Join



JOIN or INNER JOIN

- Most commonly used join
- Returns as many rows as there are matches, no more, no less
- Returns values for all columns



Two Base Tables for this Presentation

- Employee Table:

EMP_NBR	EMP_NAME	BEN_NBR
121	Steven Lee	111
852	Brian Evans	111
1234	John Smith	222
4567	Garth Robson	0

- Benefits Table:

BEN_NBR	EMP_BEN_DESC
111	TOP DENTAL
222	BOTTOM DENTAL
333	NEW DENTAL

INNER Join Example: Getting only the exact key matches

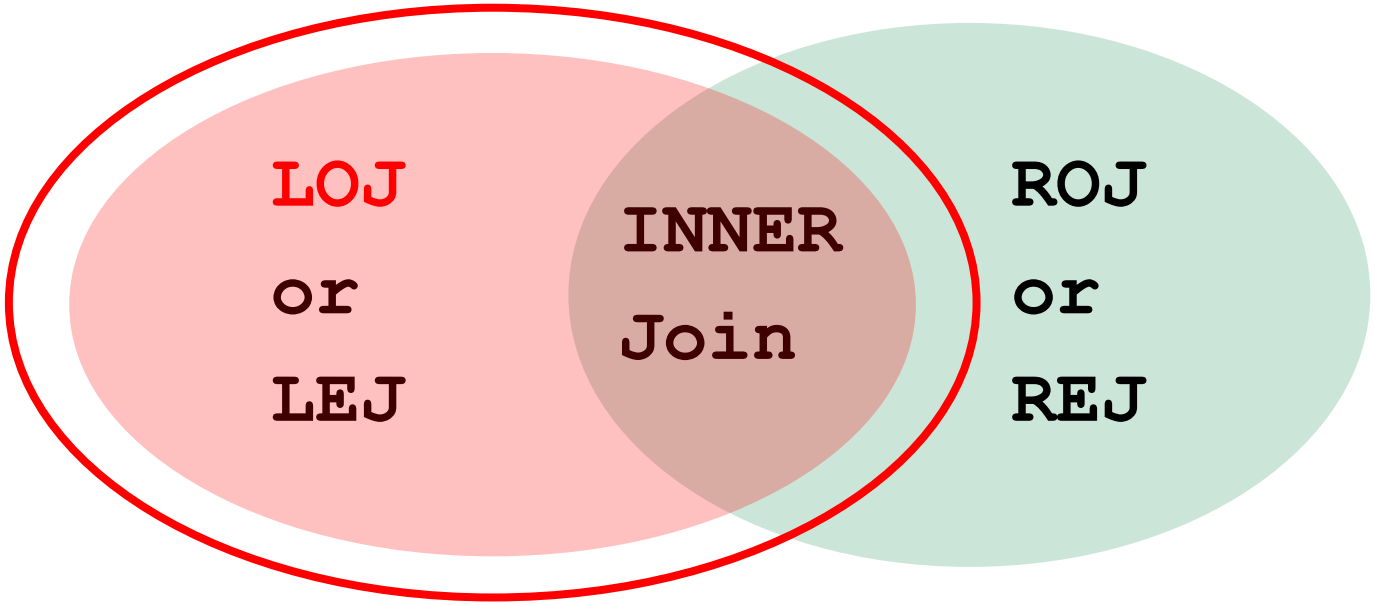
```
SELECT
EM.EMP_NBR,
EM.EMP_NAME,
EM.BEN_NBR
BM.EMP_BEN_DESC

FROM EMPLOYEE_MASTER EM
INNER JOIN BENEFITS_MASTER BM
ON EM.BEN_NBR = BM.BEN_NBR
```

EM.EMP_NBR	EM.EMP_NAME	EM.BEN_NBR	BM.EMP_BEN_DESC
121	Steven Lee	111	TOP DENTAL
852	Brian Evans	111	TOP DENTAL
1234	John Smith	222	BOTTOM DENTAL

LEFT JOIN or LEFT OUTER JOIN

- Second Most commonly used join
- Useful when you need to see ALL from the LEFT table and what ever can be found on the right side
- The "Not Found" data on the right is padded with NULL or DEFAULT Values



LOJ Example: Getting the matches, the data from the left table and defaults from the right table if no values found

```
SELECT
EM.EMP_NBR,
EM.EMP_NAME,
EM.BEN_NBR,
IFNULL(BM.EMP_BEN_DESC,
'Benefits not yet allocated')
FROM EMPLOYEE_MASTER EM
LEFT OUTER JOIN BENEFITS_MASTER BM
ON EM.BEN_NBR = BM.BEN_NBR
```

Note the use of IFNULL, which can replace un-found values with a pre-determined default (as opposed to a NULL)

LEFT JOIN or LEFT OUTER JOIN

LOJ Results **WITH** IFNULL default override

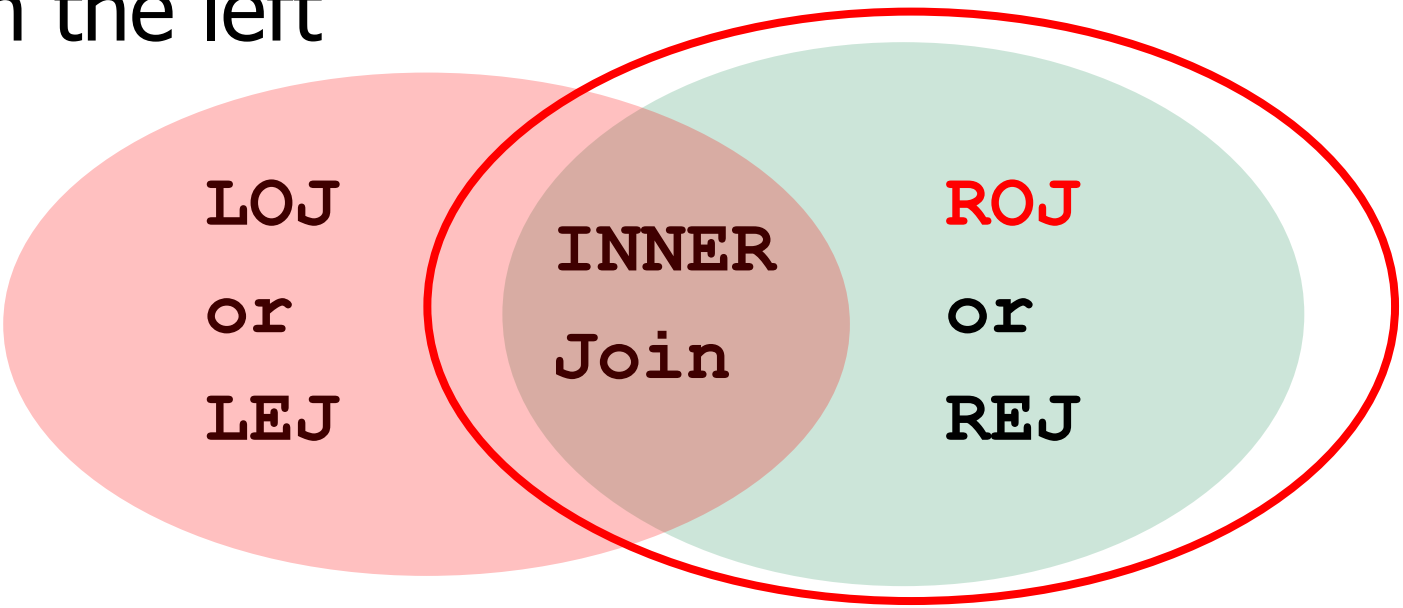
EMP_NBR	EMP_NAME	BEN_NBR	EM.EMP_BEN_DESC
121	Steven Lee	111	TOP DENTAL
852	Brian Evans	111	TOP DENTAL
1234	John Smith	222	BOTTOM DENTAL
4567	Garth Robson	0	**Benefits Not Yet Allocated**

LOJ Results **WITHOUT** IFNULL default override

EMP_NBR	EMP_NAME	BEN_NBR	EM.EMP_BEN_DESC
121	Steven Lee	111	TOP DENTAL
852	Brian Evans	111	TOP DENTAL
1234	John Smith	222	BOTTOM DENTAL
4567	Garth Robson	0	-

RIGHT JOIN or RIGHT OUTER JOIN

- Seldom used join
- Mirror image of LOJ, same rules: Bring ALL data from the right table, whatever can be found on the left



ROJ Example: Getting the matches, the data from the right table and defaults from the left table if no values found

```
SELECT
EM.EMP_NBR,
EM.EMP_NAME,
EM.BEN_NBR,
IFNULL(BM.EMP_BEN_DESC,
'Benefits not yet allocated')
FROM EMPLOYEE_MASTER EM
RIGHT OUTER JOIN BENEFITS_MASTER BM
ON EM.BEN_NBR = BM.BEN_NBR
```

Note:
Right Outer Join
is the only change
from previous
example

RIGHT JOIN or RIGHT OUTER JOIN Result

ROJ Results has **NO** IFNULL default overrides on the EMPLOYEE table, only on the BENEFITS TABLE
The NULLS WILL SHOW.

EMP_NBR	EMP_NAME	BEN_NBR	EM.EMP_BEN_DESC
121	Steven Lee	111	TOP DENTAL
852	Brian Evans	111	TOP DENTAL
1234	John Smith	222	BOTTOM DENTAL
-	-	-	NEW DENTAL

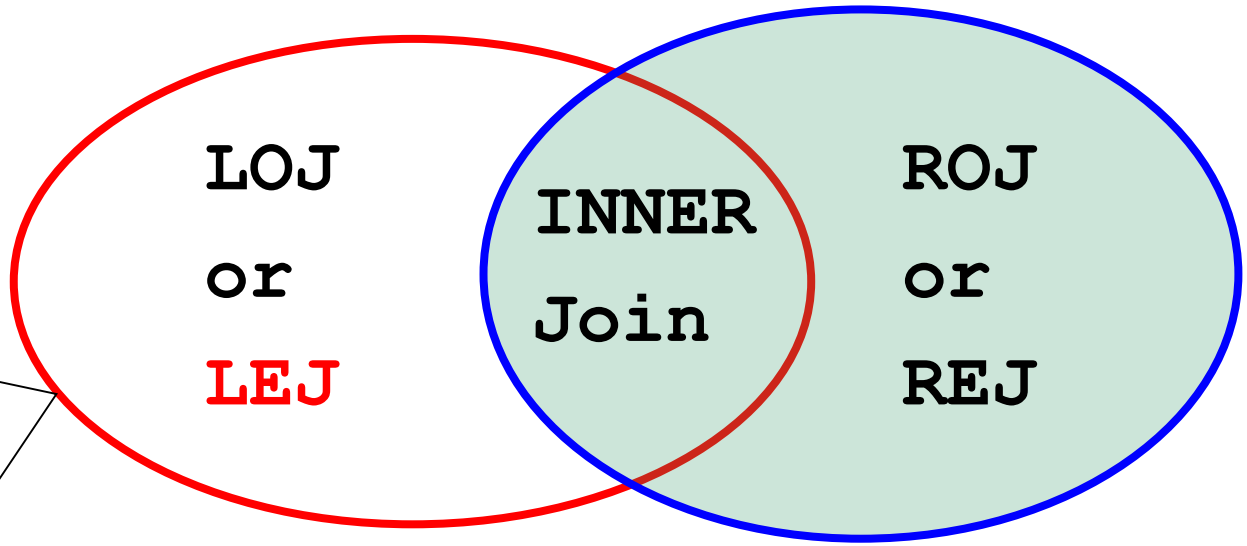
Multi LEFT OUTER JOIN Method

- One-to-many Left Outer Join can be a strong performer – AS LONG AS ALL TABLES ARE INDEXED!

```
INSERT INTO PRODUCT_BIG_PICTURE
SELECT PRD.*, INV.*, SLS.*
FROM PRODUCT_MASTER PRD
LEFT OUTER JOIN INVENTORY INV
                ON PRD.PRD# = INV.PRD#
LEFT OUTER JOIN SALES SLS
                ON PRD.PRD# = SLS.PRD#
```

LEFT EXCEPTION JOIN

LEJ
Returns data from the left table, minus any keys connecting to the right



- Returns only the rows from the left table that do not have a match in the right table
- Much more powerful than using "NOT IN" or "NOT EXISTS"

Two Base Tables for this Presentation

- An Employee Table:

EMP_NBR	EMP_NAME	BEN_NBR
121	Steven Lee	111
852	Brian Evans	111
1234	John Smith	222
4567	Garth Robson	0

- A Benefits Table:

BEN_NBR	EMP_BEN_DESC
111	TOP DENTAL
222	BOTTOM DENTAL
333	NEW DENTAL

LEFT EXCEPTION JOIN

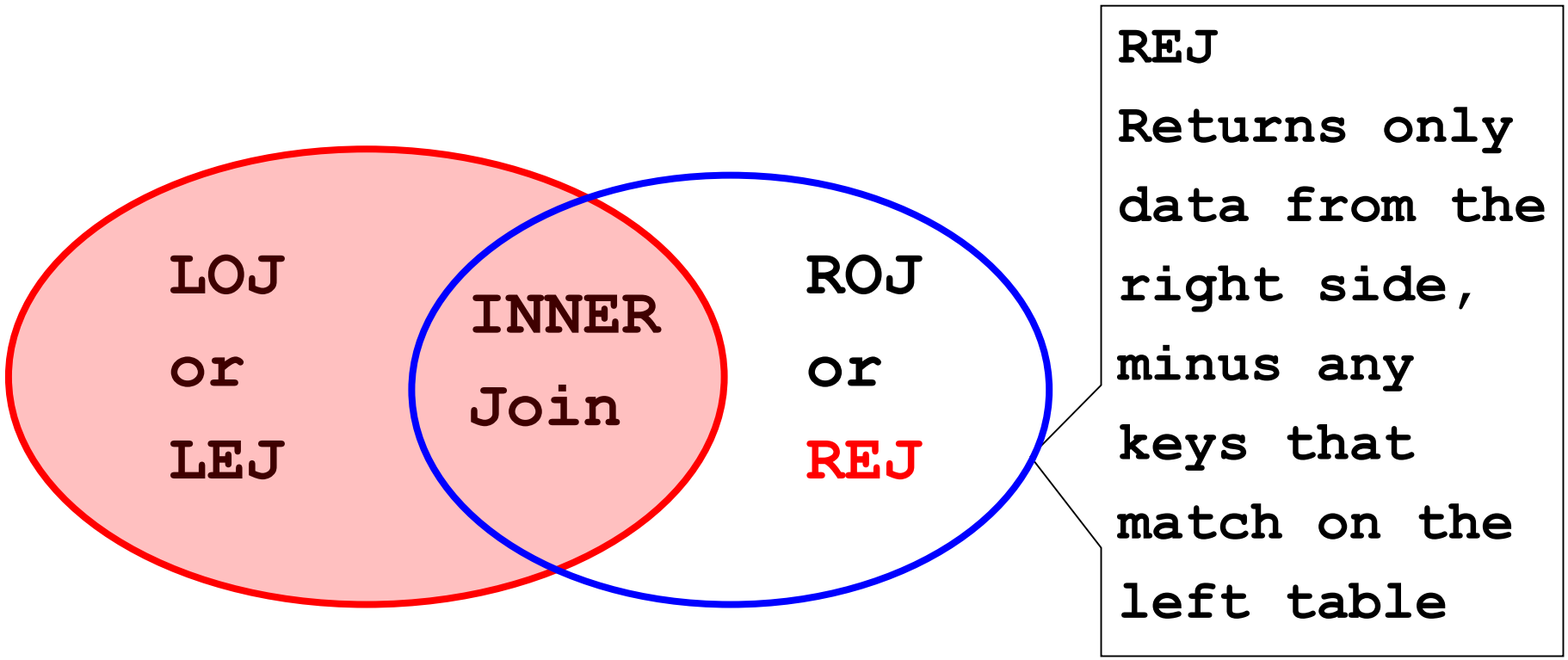
Returns only the rows from the left table that do not have a match in the right table

Example: What Employees DO NOT have Benefit Plans?

```
SELECT EM.EMP_NBR, EM.EMP_NAME,
    EM.BEN_NBR FROM EMPLOYEE EM
LEFT EXCEPTION JOIN BENEFITS BN
ON EM.BEN_NBR = BN.BEN_NBR
```

EMP_NBR	EMP_NAME	BEN_NBR
4567	Garth Robson	0

RIGHT EXCEPTION JOIN



- Returns only the rows from the RIGHT table that do not have a match in the left table
- Much more powerful than using "NOT IN" or "NOT EXISTS"

Two Base Tables for this Presentation

- An Employee Table:

EMP_NBR	EMP_NAME	BEN_NBR
121	Steven Lee	111
852	Brian Evans	111
1234	John Smith	222
4567	Garth Robson	0

- A Benefits Table:

BEN_NBR	EMP_BEN_DESC
111	TOP DENTAL
222	BOTTOM DENTAL
333	NEW DENTAL

RIGHT EXCEPTION JOIN

Returns only the rows from the right table that do not have a match in the left table

Example: What Benefits plans ARE NOT USED by employees?

```
SELECT
EM.EMP_NBR, EM.EMP_NAME,
BN.BEN_NBR, BN.EMP_BEN_DESC
FROM EMPLOYEE EM RIGHT EXCEPTION JOIN
BENEFITS BN ON EM.BEN_NBR = BN.BEN_NBR
```

EMP_NBR	EMP_NAME	BEN_NBR	EMP_BEN_DESC
-	-	333	NEW DENTAL

Exception Join Practical Use: Spotting KEY Differences

If two data sets – TABLE_A and TABLE_B are **IDENTICAL:**

- TABLE_A LEFT EXCEPTION JOIN TABLE_B
  will yield NO RESULTS

AND

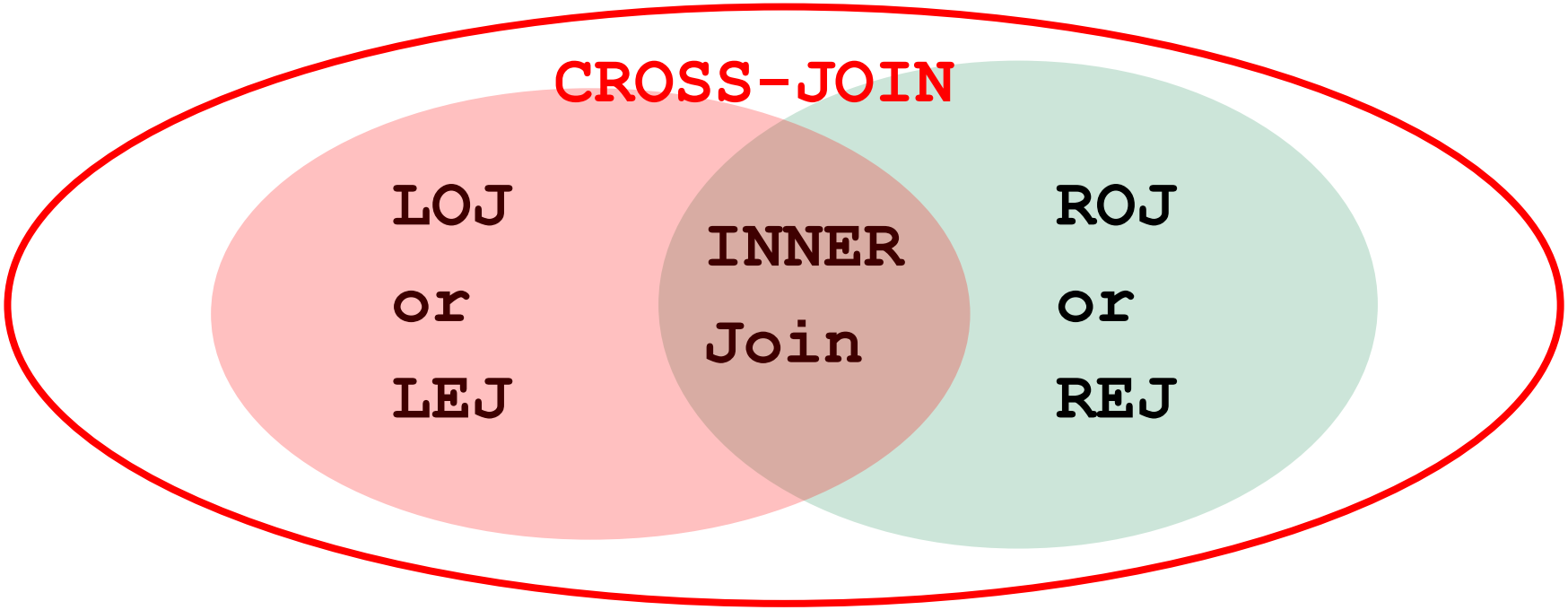
- TABLE_A RIGHT EXCEPTION JOIN TABLE_B
  will yield NO RESULTS

Any differences in keys will show up in these two statements

- Useful for integrities using summarized data
- Will tell you if "there is a difference"
- Will also tell you exactly "what the differences are "

CROSS JOIN OR "CARTESIAN PRODUCT"

- No Criteria or "Cross Join" Criteria
- Retrieves every possible combination of the two row sets



CROSS JOIN SYNTAX

- Also known as "CARTESIAN PRODUCT"
- Can be specified with the CROSS JOIN syntax or by listing two tables without a WHERE clause
- Returns a row in the result table for each combination of rows from the tables being joined

Syntax:

```
SELECT * FROM FILEA CROSS JOIN FILEB
```

same as:

```
SELECT * FROM FILEA, FILEB
```

CROSS JOIN or "CARTESIAN PRODUCT"

- Happens when there are no Join Criteria
- Returns every possible combination of two tables's contents combined

For TABLE_X with X Rows and
TABLE_Y with Y Rows

- The Cross Join will return X * Y Rows

CROSS JOIN EXAMPLE

EM.EMP_NBR	EM.EMP_NAME
121	Steve McPhearson
852	Brian Evans
1234	John Smith
4567	Garth Robson

BEN_NBR	EM.EMP_BEN_DESC
111	TOP DENTAL
222	BOTTOM DENTAL

CROSS JOIN Results

EM.EMP_NBR	EM.EMP_NAME	BEN_NBR	EM.EMP_BEN_DESC
121	Steve McPhearson	111	TOP DENTAL
121	Steve McPhearson	222	BOTTOM DENTAL
852	Brian Evans	111	TOP DENTAL
852	Brian Evans	222	BOTTOM DENTAL
1234	John Smith	111	TOP DENTAL
1234	John Smith	222	BOTTOM DENTAL
4567	Garth Robson	111	TOP DENTAL
4567	Garth Robson	222	BOTTOM DENTAL

CROSS JOIN or "CARTESIAN PRODUCT" MOST OFTEN CONSIDERED BAD

- Is there use for a CROSS-JOIN?

Exploring Sales Data with CROSS-JOIN

```
SELECT SLS.STORE_NBR, SLS.PRODUCT_NBR,
SUM(SLS.QTY_SOLD) FROM SALES SLS
GROUP BY SLS.STORE_NBR, SLS.PRODUCT_NBR
```

Query Above Will return nothing for zero $ products
The query below will show product with zero sales

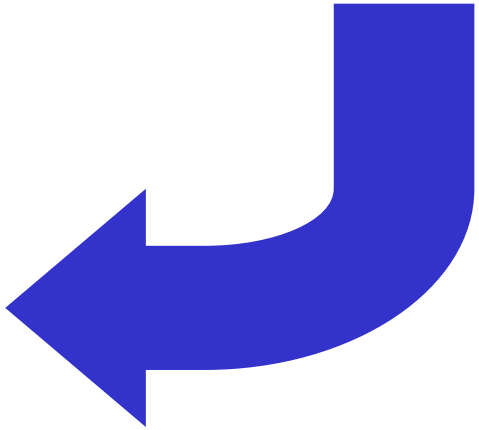
```
SELECT STR.STORE_NBR, PRD.PRODUCT_NBR,
SUM(IFNULL(SLS.QTY_SOLD,0)) AS TOTALSALES
FROM STORES STR CROSS JOIN PRODUCTS PRD
LEFT OUTER  JOIN SALES SLS
ON    SLS.STORE_NBR = STR.STORE_NBR
AND   SLS.PRODUCT_NBR = PRD.PRODUCT_NBR
GROUP BY STR.STORE_NBR,  PRD.PRODUCT_NBR
```

Get All Stores and All Products

Outer Join to Sales

Account ID	Year	Month 01 Amount	Month 02 Amount	Month 03 Amount	Month 04 Amount	Month 05 Amount	Month 06 Amount	Month 07 Amount	Month 08 Amount	Month 09 Amount	Month 10 Amount	Month 11 Amount	Month 12 Amount
000001	2005	16.66	27.22	38.33	49.44	60.55	71.66	82.77	93.88	104.99	15.1	16.11	17.12

Account ID	Year	Month	Month Amount
000001	2005	1	16.66
000001	2005	2	27.22
000001	2005	3	38.33
000001	2005	4	49.44
000001	2005	5	60.55
000001	2005	6	71.66
000001	2005	7	82.77
000001	2005	8	93.88
000001	2005	9	104.99
000001	2005	10	5.1
000001	2005	11	16.11
000001	2005	12	17.12



Pivoting a Table From Horizontal to Vertical Using SQL **CROSS JOIN**

H to V Table Pivot using **Cross-Join**

Pivot a 12-month table From HORIZONTAL To VERTICAL by

Using a CROSS JOIN To a 12-ROW table containing numbers 1 to 12 [here named]

' MONTH_NUMERIC '

- Use the CASE statement to pick the right value depending on the month processed

```
INSERT INTO VERTICAL
(       YEAR                ,
        ACCOUNT_ID          ,
           MONTH                ,
        NET_POSTING     )
```

```
SELECT
    YEAR                ,
    MONTH               ,
    ACCOUNT_ID          ,
  CASE MONTH_VALUE
         WHEN 1   THEN NET_01
         WHEN 2   THEN NET_02
         WHEN 3   THEN NET_03
         WHEN 4   THEN NET_04
         WHEN 5   THEN NET_05
         WHEN 6   THEN NET_06
         WHEN 7   THEN NET_07
         WHEN 8   THEN NET_08
         WHEN 9   THEN NET_09
         WHEN 10 THEN NET_10
         WHEN 11 THEN NET_11
         WHEN 12 THEN NET_12
    END
 FROM    HORIZONTAL
   CROSS JOIN  MONTH_NUMERIC ;
```

Passing a "Lazy" parameter using CROSS JOIN

- Adds flexibility when using Interpreted SQL
- Uses Dan Riehl's EXCSQL
- Uses a 1-ROW Parameter table to cross join (no multiplying effect)

CL Program:

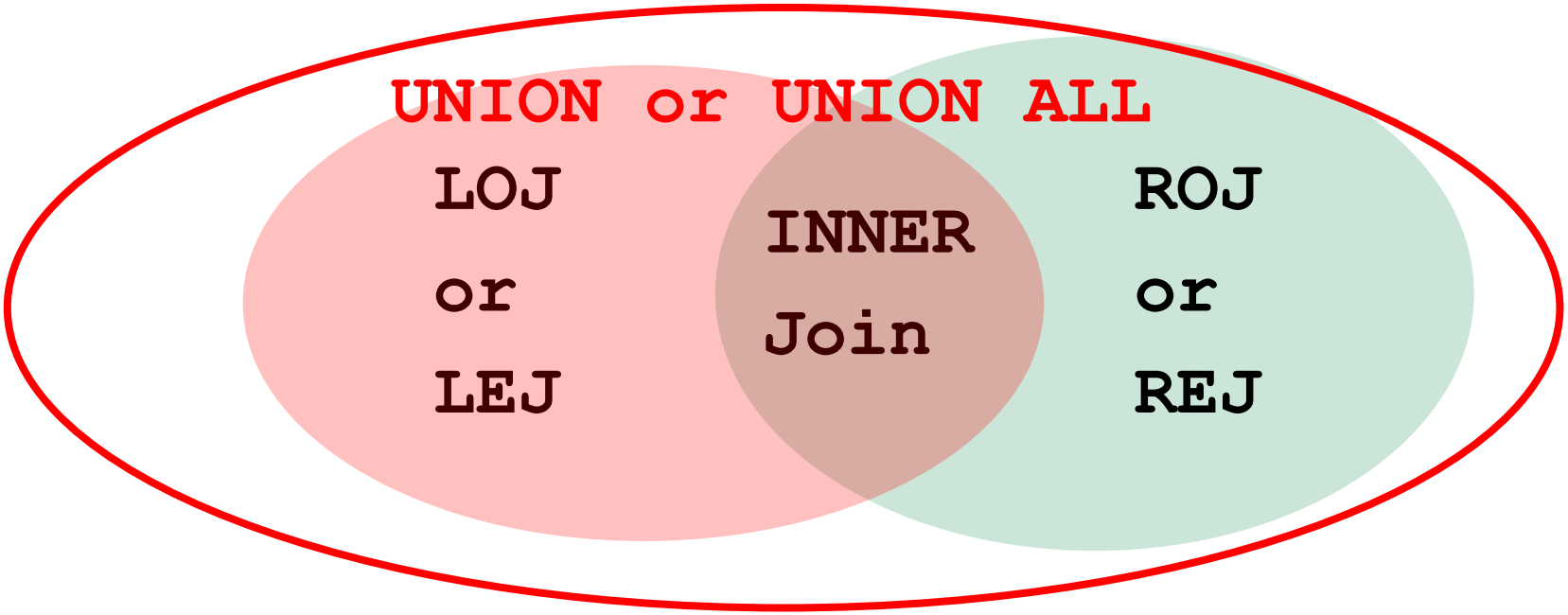
```
EXCSQL REQUEST( UPDATE PARM_TABLE
                SET PARM ='ALBERTA' )
RUNSQLSTM SRCFILE(SQLSRC) SRCMBR(EMPSQL)
```

SQL Join:

```
SELECT EMPLOYEE_NUMBER, FIRST_NAME, LAST_NAME
    FROM EMPLOYEE_TABLE CROSS JOIN PARM_TABLE
WHERE EMPLOYEE_PROVINCE = PARM_TABLE.PARM
```

UNION or UNION ALL

- Returns data from two identical sets of data



UNION

- Returns data from two sets of data
- The data from both SELECTS must be of the same format
- NOTE:

  UNION **RETURNS DISTINCT VALUES ONLY**

```
SELECT EMPLOYEE_NUMBER, FIRST_NAME, LAST_NAME
    FROM ALBERTA/EMPLOYEE_TABLE
 UNION
SELECT EMPLOYEE_NUMBER, FIRST_NAME, LAST_NAME
    FROM NOVASOTIA/EMPLOYEE_TABLE
```

UNION ALL

- Returns data from two sets of data
- The data from both SELECTS must be of the same format
- - NOTE: UNION ALL **RETURNS ALL VALUES, REGARDLESS OF DUPLICATES**

```
SELECT EMPLOYEE_NUMBER, FIRST_NAME, LAST_NAME
    FROM ALBERTA/EMPLOYEE_TABLE
 UNION ALL
SELECT EMPLOYEE_NUMBER, FIRST_NAME, LAST_NAME
    FROM NOVASOTIA/EMPLOYEE_TABLE
```

Using UNION with Multi-Member (conventional iSeries) FILES with SQL

SQL allows the targetting of individual members with the use of an ALIAS

```
CREATE ALIAS LIBRARY1/SALES_HIST_1999
    FOR LIBRARY1/SALESHIST(MBR_HST_99)

CREATE ALIAS LIBRARY1/SALES_HIST_2000
    FOR LIBRARY1/SALESHIST(MBR_HST_00)
```

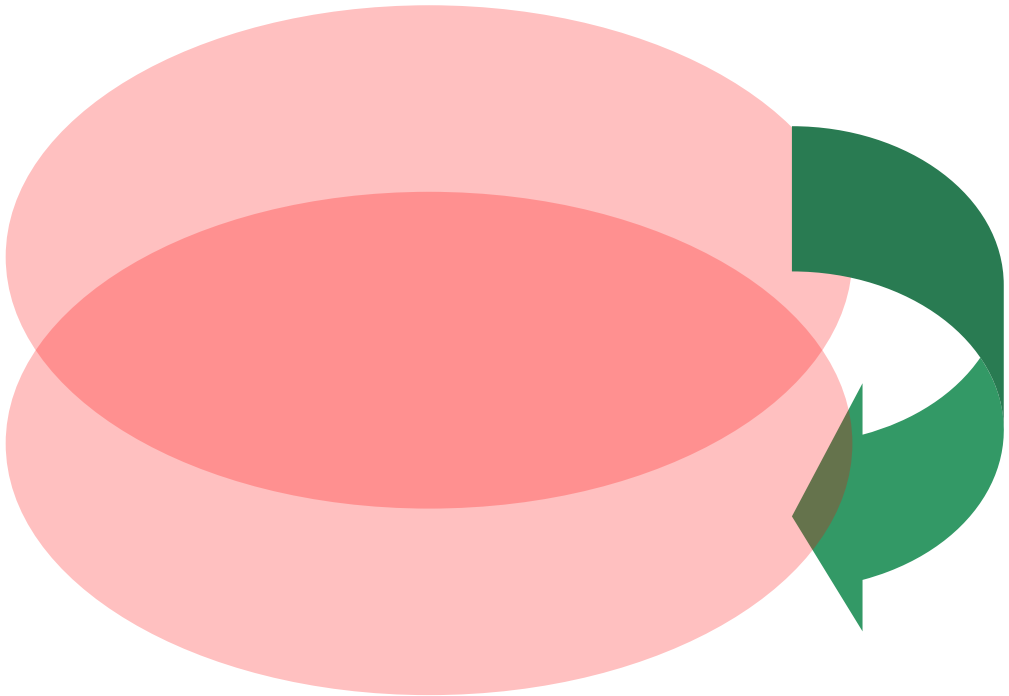
Using UNION to retrieve all members data

```
SELECT * FROM LIBRARY1/SALES_HIST_1999
UNION ALL
SELECT * FROM LIBRARY1/SALES_HIST_2000
ORDER BY SALES_DATE
```

SELF-JOIN

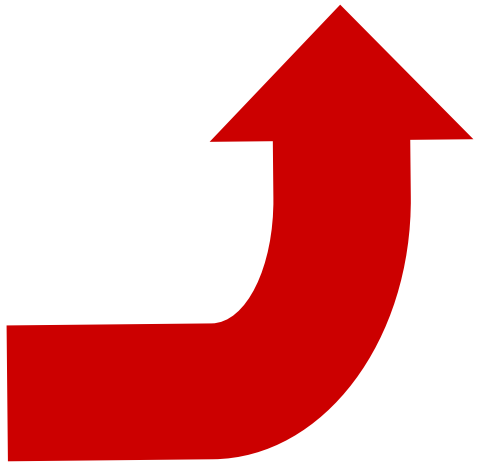
Useful

- For pivoting
- For finding and deleting duplicates



Account ID	Year	Month 01 Amount	Month 02 Amount	Month 03 Amount	Month 04 Amount	Month 05 Amount	Month 06 Amount	Month 07 Amount	Month 08 Amount	Month 09 Amount	Month 10 Amount	Month 11 Amount	Month 12 Amount
000001	2005	16.66	27.22	38.33	49.44	60.55	71.66	82.77	93.88	104.99	15.1	16.11	17.12

Account ID	Year	Month	Month Amount
000001	2005	1	16.66
000001	2005	2	27.22
000001	2005	3	38.33
000001	2005	4	49.44
000001	2005	5	60.55
000001	2005	6	71.66
000001	2005	7	82.77
000001	2005	8	93.88
000001	2005	9	104.99
000001	2005	10	5.1
000001	2005	11	16.11
000001	2005	12	17.12



Pivoting a Table From Vertical to Horizontal Using SQL by **JOINING A FILE TO ITSELF**

V to H Table Pivot using **Self-Join** (1 of 2)

- Principle: Join the table to itself 12 times to spread the data sideways for 12 months

```
INSERT INTO HORIZONTAL
(
   YEAR       ,
      ACCOUNT_ID,
      NET_01     ,
      NET_02     ,
      NET_03     ,
      NET_04     ,
      NET_05     ,
      NET_06     ,
      NET_07     ,
      NET_08     ,
      NET_09     ,
      NET_10     ,
      NET_11     ,
      NET_12     ,
)
```

```
SELECT
 V01.YEAR                          ,
 V01.ACCOUNT_ID                    ,
 V01.NET_POSTING                   ,
 IFNULL(V02.NET_POSTING, 0)   ,
 IFNULL(V03.NET_POSTING, 0) ,
 IFNULL(V04.NET_POSTING, 0) ,
 IFNULL(V05.NET_POSTING, 0) ,
 IFNULL(V06.NET_POSTING, 0) ,
 IFNULL(V07.NET_POSTING, 0) ,
 IFNULL(V08.NET_POSTING, 0) ,
 IFNULL(V09.NET_POSTING, 0) ,
 IFNULL(V10.NET_POSTING, 0) ,
 IFNULL(V11.NET_POSTING, 0) ,
 IFNULL(V12.NET_POSTING, 0)
```

V to H Table Pivot using **Self-Join** (2 of 2)

```
FROM VERTICAL V01

LEFT OUTER JOIN VERTICAL V02
on   V01. YEAR        = V02. YEAR
and  V01. ACCOUNT_ID  = V02. ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V03
on   V01. YEAR        = V03. YEAR
and  V01. ACCOUNT_ID  = V03. ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V04
on   V01. YEAR        = V04. YEAR
and  V01. ACCOUNT_ID  = V04. ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V05
on   V01. YEAR        = V05. YEAR
and  V01. ACCOUNT_ID  = V05. ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V06
on   V01. YEAR        = V06. YEAR
and  V01. ACCOUNT_ID  = V06. ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V07
on   V01. YEAR        = V07. YEAR
and  V01. ACCOUNT_ID  = V07. ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V08
on   V01. YEAR        = V08. YEAR
and  V01. ACCOUNT_ID  = V08. ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V09
on   V01. YEAR        = V09.YEAR
and  V01. ACCOUNT_ID  = V09.ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V10
on   V01. YEAR        = V10.YEAR
and  V01. ACCOUNT_ID  = V10.ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V11
on   V01. YEAR        = V11. YEAR
and  V01. ACCOUNT_ID  = V11. ACCOUNT_ID

LEFT OUTER JOIN VERTICAL V12
on   V01. YEAR        = V12. YEAR
and  V01. ACCOUNT_ID  = V12. ACCOUNT_ID


WHERE    V01.MONTH_VALUE = 01
     and V02.MONTH_VALUE = 02
     and V03.MONTH_VALUE = 03
     and V04.MONTH_VALUE = 04
     and V05.MONTH_VALUE = 05
     and V06.MONTH_VALUE = 06
     and V02.MONTH_VALUE = 07
     and V03.MONTH_VALUE = 08
     and V04.MONTH_VALUE = 09
     and V05.MONTH_VALUE = 10
     and V06.MONTH_VALUE = 11
     and V06.MONTH_VALUE = 12   ;
```

Delete / Update With SQL

- Use of SQL for UPDATE or DELETE

Deleting Data in a Table Using a Correlated Sub-Select (only method currently available on iSeries)

```
DELETE FROM EMPLOYEE_TABLE EM
WHERE EXISTS
(SELECT * FROM UPDATE_TABLE UPDT WHERE
   UPDT.ID = EM.ID);
```

- Note the use of TWO WHERE clauses

Removing Duplicate Rows In A Table using a self Correlated Sub-Select

```
DELETE FROM POS_TABLE POS1
WHERE RRN(POS1) <
 (
    SELECT MAX( RRN(POS2) )
    FROM POS_TABLE POS2
    WHERE
     POS1.TRANSACTION_# = POS2.TRANSACTION_#)
 )
```

- Note the use of the MAX clause
- Note the use of Correlation Names **POS1** and **POS2** - attacking the same table twice with two different correlated names

Updating Data in a Table Using a Correlated Query (update with join not possible for now)

```
UPDATE EMPLOYEE EM
SET (EM.PAY_SCALE, EM.SALARY) =
(SELECT NPAY.PAY_SCALE, NPAY.SALARY FROM
    NEW_PAY NPAY )
WHERE EXISTS
    (SELECT *
   FROM NEW_NEWPAY NPAY WHERE NPAY.ID = EM.ID
    )
```

- Note the use of TWO WHERE clauses
- **WARNING:** Will crash if the second select yields more than one row!

Join Summary

- Inner Join
- Left or Right Outer Join
- Left or Right Exception Join
- Union or Union All
- Cross Join and Self Join
- Update and Delete

Value added Techniques: Joins and Aggregates

- Joining Tips and Techniques
  - Using CAST
  - Using CASE
  - Using Unreliable Keys
  - Using Dates

- Aggregate Values in SQL
  - Aggregates and joins

Transform Data on the fly by using CAST: Two types of Syntax

INT to CHAR using the "CAST" operand:

```
SELECT
CAST(ZIP_NUMBER AS CHAR(5)) CHAR_ZIP
FROM FILEB
```

CHAR to INT using the "CAST" operand:

```
SELECT
INT(SUBSTRING(TELEPHONE, 1,3)
     || SUBSTRING(TELEPHONE, 5,4) )
    INT_TEL_NO
FROM FILEA
```

Joining Tables With Incompatible Keys using CAST

- Joining with Cast Values

```
SELECT
LT.FIRST_NAME,
LT.LAST_NAME,
LT.TELEPHONE
FROM  LOCAL_NAMES_TABLE
LT  INNER  JOIN  COMPARE_TABLE  CT
ON  INT(SUBSTRING(LT.TELEPHONE, 1,3)
          ||  SUBSTRING(LT.TELEPHONE, 5,4))
       =  CT.TELEPHONE#
```

Caveat!
Beware of performance hit with CAST JOINS

Using CASE

- Evaluated in the order listed
- Note: Will yield a NULL if no ELSE default is specified

```
SELECT ET.EMPLOYEE_NO, ET.FIRST_NAME,
  ET.LASTNAME,
CASE
      WHEN ET.YEARS_OF_SERVICE > 25
           THEN 'ELIGIBLE FOR RETIREMENT'
      WHEN ET.YEARS_OF_SERVICE > 15
           THEN '15 YEARS OR LESS TO GO!'
      ELSE 'TAKE A DEEP BREATH!'
END
FROM EMPLOYEE_TABLE ET
```

Joining Tables With **Unreliable Numeric Data** Stored in a Character Column

- Storing Numeric Data in an Character Column is makes for UNRELIABLE JOINS
- Sometimes, you just have no choice

```
SELECT * FROM FILEAA AA
     LEFT OUTER JOIN FILEBB BB
     ON
                  -- the case statement will determine if the
     CASE WHEN    -- values within the column are purely numeric
                ( -- or not
             LOCATE(SUBSTR(AA.CHAR_PO_NUMBER,1, 1),'0123456789') = 0
          OR LOCATE(SUBSTR(AA.CHAR_PO_NUMBER,2, 1),'0123456789') = 0
          OR LOCATE(SUBSTR(AA.CHAR_PO_NUMBER,3, 1),'0123456789') = 0
          OR LOCATE(SUBSTR(AA.CHAR_PO_NUMBER,4, 1),'0123456789') = 0
          OR LOCATE(SUBSTR(AA.CHAR_PO_NUMBER,5, 1),'0123456789') = 0
                )
     THEN 0      -- default value (there is non-numeric data)
     ELSE INT(AA.CHAR_PO_NUMBER)  -- valid numeric value
     END

     = BB.NUMERIC_PO_NUMBER
```

Casting Characters and Digits: **Char vs. Digits**

- CHAR strips leading zeros, DIGITS does not

- If you have a NUMVAR1 declared as NUMERIC(5,0) number equal to 00888:
  - **CHAR** (NUMVAR1) will yield '888'
  - **DIGITS**(NUMVAR1) will yield '00888'

Time & Date Data Types

- The TIMESTAMP value on iSeries records time to ONE MILLIONTH of a SECOND
- DO NOT USE DEC or INT to record dates or time
- DATE, TIME and TIMESTAMP columns are good candidates for joins – watch for the most appropriate granularity for your application

Query Join Condition Performance: Base Principles

- Be aware of the instances where DB2 will not use an index
  - Avoid data type conversions, casts
  - Avoid formulas
- Avoid the use of like patterns

Correlated Sub-Selects vs. Joins

This correlated sub-query will look for product ID's that DID have sales

```
SELECT PRD.ID FROM PRODUCT_TBL PRD
WHERE EXISTS
( SELECT SLS.ID FROM SALES_TBL SLS
  WHERE SLS.ID = PRD.ID )
```

WHERE EXISTS

Can be re-written as a INNER JOIN:

```
SELECT PRD.ID FROM PRODUCT_TBL PRD
INNER JOIN SALES_TBL SLS
ON SLS.ID = PRD.ID
```

Correlated Sub-Selects vs. Joins

This correlated sub-query will look for product ID's that did NOT have sales

```
SELECT PRD.ID FROM PRODUCT_TBL PRD
WHERE NOT EXISTS
( SELECT SLS.ID FROM SALES_TBL SLS
  WHERE SLS.ID = PRD.ID )
```

WHERE NOT EXISTS

Can be re-written as a LEFT EXECPTION JOIN:

```
SELECT PRD.ID FROM PRODUCT_TBL PRD
LEFT EXCEPTION JOIN SALES_TBL SLS
ON SLS.ID = PRD.ID
```

Joins vs. Sub-Queries

Rule of thumb:

- Joins are more efficient than sub-queries

Exception:

- When the sub-query contains one or more aggregates and it is not correlated

Joins vs. Sub-Selects – Aggregated

A non-correlated Sub-Select can be the best way to get the desired results

Example: Find above average sales performance:

```
SELECT TS1.SALESMAN, TS1.SALES,
FROM TOTAL_SALES TS1
WHERE TS1.SALES >
(SELECT AVG(TS2.SALES)FROM TOTAL_SALES TS2)
```

The AVERAGE aggregate function is performed ONLY ONCE for the entire query

Aggregating Data with GROUP BY

- Get aggregated values, for a specified group
- Note the "Select" and the "Group by" parameters are identical

```
SELECT CITY_NAME,
    COUNT(*) ORDERS_COUNT,
     SUM(ORDER_VALUE) ORDERS_VALUE,
     AVG(ORDER_VALUE) AVERAGE,
     MIN(ORDER_VALUE) MIN_ORDER,
     MAX(ORDER_VALUE) MAX_ORDER FROM ORDERS
     GROUP BY CITY_NAME ORDER BY AVERAGE
```

CITY_NAME	ORDERS_ COUNT	ORDERS_VALUE	AVERAGE	MIN_ORDER	MAX_ORDER
Edmonton	2324.00	45646546.00	19641.37	123.00	852.00
Red Deer	3434.00	544696445.00	158618.65	1822.00	5236.00
Calgary	4553.00	834098534.00	183197.56	268.00	7411.00
Banff	2.00	554556.00	277278.00	965.00	1258.00

WHERE and HAVING Clauses

- Use **WHERE** to compare individual row values
- Use **HAVING** to compare aggregated values

```
SELECT STORE_NAME, STORE_PROV,
SUM(SALES) STORE_SALES
FROM STORE_INFORMATION
WHERE STORE_PROV = 'AB'
GROUP BY STORE_NAME, STORE_PROV
    HAVING SUM(SALES) > 1500
```

STORE_NAME	STORE_PROV	STORE_SALES
Calgary Store	AB	3434
Red Deer Store	AB	4553
Edmonton Store	AB	8522

Finding Duplicate Data in a Table

```
SELECT TRANSACTION_NUMBER, COUNT(*)
    FROM POS_TABLE
    GROUP BY TRANSACTION_NUMBER
    HAVING COUNT(*) > 1
```

- Very Common SQL Example
- Note the use of the GROUP BY clause
- Unique Keys still best to keep duplicates out when possible!
- Useful to clean up raw data

Beware of Cascaded Joins: Break it up! (**Slow Join** Problem!)

- Proverbial "Forever Processing" Join:
- Files BB, CC, DD are all intertwined!

```
INSERT INTO FILEA
SELECT BB.* FROM FILEB BB
INNER  JOIN FILEC CC
   ON BB.KEYB = CC.KEYC
LEFT EXCEPTION JOIN FILED DD
   ON  CC.KEY2 = DD.WORK_KEY

 WHERE DD.WORK_KEY NOT LIKE '%DW%'
   AND DD.DW_ROW_TYPE NOT IN ('R','P')
```

Join from FILEB to FILEC
from FILEC to FILED
VERY EXPENSIVE!

LIKE and
NOT IN
Operations
EXPENSIVE!

Performance Considerations: Break it up! (Solution Part 1)

- Use an INDEXED WORKFILE to split the load into manageable chunks
- First, minimize the negative effect of "LIKE" and "NOT IN"

```
INSERT INTO DDWORKFILE
SELECT DD.* FROM FILED DD
WHERE DD.WORK_KEY NOT LIKE '%DW%'
AND DD.DW_ROW_TYPE NOT IN ('R','P')
```

Performance Considerations: Break it up! (Solution Part 2)

- Again, Use an INDEXED WORKFILE to split the load into manageable chunks
- Second, Create a new intermediate work file for the other join

```
INSERT INTO CCWORKFILE
SELECT CC.* FROM FILEC CC
LEFT EXCEPTION JOIN DDWORKFILE DD
    ON  CC.KEY2 = DD.WORK_KEY
```

Performance Considerations: Break it up! (Solution Part 3)

- The new join will use only keys, NO OTHER SELECTION CRITERIA

```
INSERT INTO FILEA
SELECT BB.* FROM FILEB BB
INNER  JOIN CCWORKFILE CC
   ON BB.KEYB = CC.KEYC
```

- 3 Simple joins are
  - - more efficient
  - - quicker to execute
- Than one complicated SQL statement

Performance Checklist/Recap

- Are CAST operations used in joins?
- Are LIKE or NOT IN operations used in joins?
- Are there Formulas in WHERE clauses?
- Is the technique optimum?
  - Is Join vs. Correlated sub-query used?
  - Are there cascaded joins?
- Could the process be broken up in smaller pieces?
- Are there proper INDEXES?
- Did you test with life-size samples?

Recap For This Presentation:

- Joins
  - Inner, Outer, Exception, Union, Union All
  - Updates and Deletes exception to the JOIN rule
- Casting & Case
  - Casts and Case can be used in joins (beware of performance!)
- Performance
  - Joins vs correlated sub-selects
  - Pay attention to keys, multi-join statements
  - Pay attention to cascaded joins

I hear and forget.

I see and remember.

I do and I understand.

Kung Futse

551 B.C.

Questions

Email: dambrine@tylogix.com

See the SQL Section in
www.tylogix.com