Rene Descartes
1596-1650

SQL:
Overview of ANSI Joins

- Cross joins
- Equi-Join
- Natural Joins
- Condition Joins
- Column name Joins
- Inner Joins
- Outer Joins

You may not use all of these but you should be familiar with the terms.

Special Note

- Just about all the information you need for writing **INNER JOINS** is in the first set of slides on joins.
- These slides are presented here so that you will get the background on all the "Flavors" of joins.
- We will also be discussing the important concept of **CROSS JOINS** and introducing **OUTER JOINS**

In Third Grade…
You may have done a Cross Join

How many possible outfits can you make from 3 pair of pants and 3 shirts?

A Cross Join is the Cartesian Product

Table1

Name	Address	phone
Jones	111 Maple	555-1111
Smith	222 Elm	555-2222
Walters	333 Birch	555-3333

Table 2

Name	Salary
Jones	30000
Smith	20000
Walters	40000

SELECT table1.*, table2.*
FROM table1, table2;

Name	Address	Phone	Name	Salary
Jones	111 Maple	555-1111	Jones	30000
Jones	111 Maple	555-1111	Smith	20000
Jones	111 Maple	555-1111	Walters	40000
Smith	222 Elm	555-2222	Jones	30000
Smith	222 Elm	555-2222	Smith	20000
Smith	222 Elm	555-2222	Walters	40000
Walters	333 Birch	555-3333	Jones	30000
Walters	333 Birch	555-3333	Smith	20000
Walters	333 Birch	555-3333	Walters	40000

Result = table1 x table2

(the results of a cross join are sometimes meaningless!)

Cross Join continued

- Another standard syntax is:

  SELECT *

  FROM table 1 CROSS JOIN table 2;
- Both forms of the join have **no JOIN CONDITION.** (via ON or WHERE)
- The result is always a **Cartesian product**.
- The number of rows in the result will be the product of the rows in the base tables.

  e.g. 3 rows X 3 rows = 9 rows.

The results of this cross join are more meaningful!

Cross Join – All possibilities

Colors:

Color
Red
Blue
Silver

x

Engines:

Engine
Hybrid
Gas

x

Options:

Option_pkg
Standard
Deluxe

```
SELECT color,
engine, option_pkg
FROM
Colors
   CROSS JOIN
Engines
   CROSS JOIN
Options;
```

Result

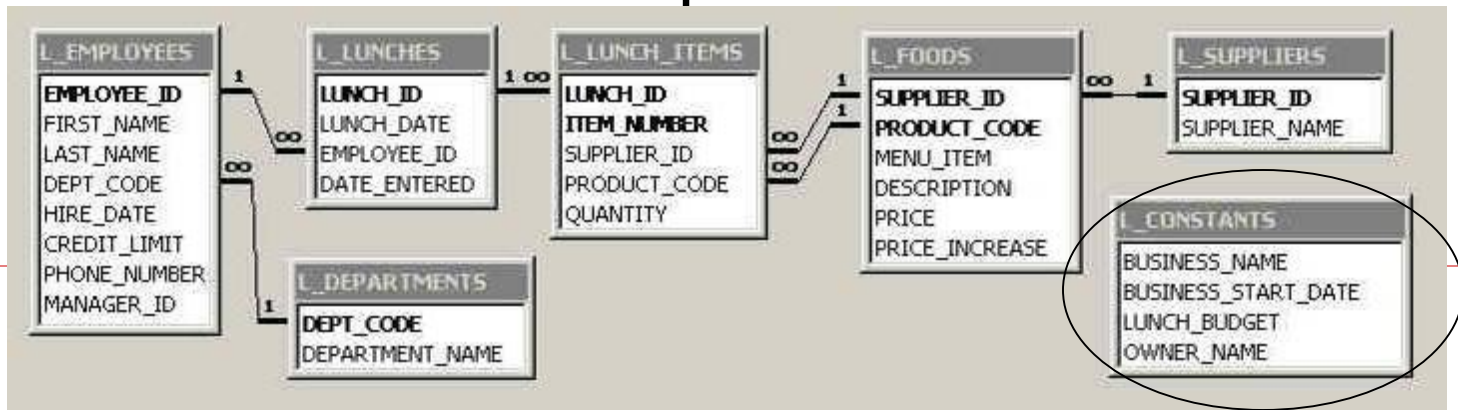
Color	Engine	Option_pkg
Red	Hybrid	Standard
Red	Hybrid	Deluxe
Red	Gas	Standard
Red	Gas	Deluxe
Blue	Hybrid	Standard
Blue	Hybrid	Deluxe
Blue	Gas	Standard
Blue	Gas	Deluxe
Silver	Hybrid	Standard
Silver	Hybrid	Deluxe
Silver	Gas	Standard
Silver	Gas	Deluxe

Another Example of Cross Join – Table of Constants

- Constants can be used to store same value in every row.
- Don't want to store this data in a base table.
- Options:
  - Place a literal value in SELECT clause

(not good practice or flexible)

  - Use a table of constants
- Best to see an example…..



Constants- literal value in SELECT

- We can create virtual column(s) and insert a literal as a constant.

```
SELECT employee_id, last_name,
'15-DEC-2006' AS eval_date
'15-DEC-2007' AS next_eval
FROM l_employees;
```

- But this may not be the best coding practice.

Using a Table of Constants

- Better to use a table of constants
- Benefits:
  - Flexibility in SQL, easy to change constants
  - Guarantees consistency
- Example, base table and table of constants can be combined:

```
SELECT employee_id, last_name, eval_date, next_eval
FROM l_employees CROSS JOIN sec0306_constants
ORDER BY 1;
```

Using the Table of Constants

- This is actually a join with no WHERE condition. So, it's a cross-product.

L_employees

201	Susan	Brown
202	Jim	Kern
203	Martha	Woods

Sec0306_constants

Eval_date	Next_eval
15-dec-2011	15-dec-2012

Results

ID	First_name	Last_name	Eval_date	Next_eval
201	Susan	Brown	15_dec-2011	15-dec-2012
202	Jim	Kern	15-dec-2011	15-dec-2012
203	Martha	Woods	15-dec-2011	15-dec-2012

```
SELECT employee_id, last_name, eval_date, next_eval
FROM l_employees CROSS JOIN sec0306_constants ORDER BY 1;
```

Cross Join - Summary

- Use CROSS JOIN to see all possible combinations of rows from two or more tables.
- If you really intend to do a cross join, be explicit and spell it out. Don’t just list two tables without the WHERE clause.
- Normally, AVOID cross joins. They may be costly and meaningless!

Equi-Join contains a WHERE clause

```
SELECT t1.*, t2.*
FROM table1 t1, table2 t2
WHERE t1.name = t2.name;
```

From the cross join, we have eliminated all the rows that do not match the join condition (i.e. the grey rows).

T1.Name	T1.Address	T1.Phone	T2.Name	T2.Salary
Jones	111 Maple	555-1111	Jones	30000
Jones	111 Maple	555-1111	Smith	20000
Jones	111 Maple	555-1111	Walters	40000
Smith	222 Elm	555-2222	Jones	30000
Smith	222 Elm	555-2222	Smith	20000
Smith	222 Elm	555-2222	Walters	40000
Walters	333 Birch	555-3333	Jones	30000
Walters	333 Birch	555-3333	Smith	20000
Walters	333 Birch	555-3333	Walters	40000

"Equi" pertains to the = sign in the join condition.

Natural Join

- Example:

```
SELECT e.last_name, e.first_name, d.department_name
FROM l_employees e NATURAL JOIN l_departments d;
```

- No WHERE clause is specified, BUT the join is executed by matching values in columns with the same name.
- Result is same as equi-join
- Can be good in testing when you don't know what the join columns are OR for comparison of results.

Condition Join

- Here's a join where the column names are NOT the same.

Example:

```
SELECT *
FROM l_employees e JOIN vacation_days v
ON e.empid = substr(v.empid, 1,3);
```

- The Join condition can be any valid SQL expression.

This type of join might suffer from performance problems in a large database.

Column Name Join – "USING"

- Same as natural join, but column names are explicitly named

Example:

```
SELECT e.last_name, e.first_name,
    d.department_name
FROM l_employees e JOIN l_departments d
USING (dept_code);
```

- USING clause is valid when join columns have the same name.

Inner Join

- Natural Joins, Condition Joins, Column-Name Joins are all **INNER JOINS**
- Inner join syntax is:

  SELECT [column-list]

  FROM Table1 [INNER JOIN] Table2

  USING ([column-list]) | ON [column-conditions];
- Notice: Join condition is specified with "Using" or "ON", not both! "On" is MUCH more common.
- Whenever you specify "JOIN" alone, it's an inner join.

Outer Join

- Inner join throws out rows that don't match.
- Outer join puts back some or all of the unmatched rows:
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

Left Outer Join

- Preserves unmatched rows from the left table (the one before keyword join)
- Example:

```
SELECT  e.last_name, d.dept_code, d.department_name
FROM l_employees e LEFT OUTER JOIN l_departments d
ON  e.dept_code = d.dept_code;
```

- Translation: Display employees and their departments. Show all employees, even if they have not yet been assigned to a department.

Left Outer Join

```
SELECT  e.lname, e. dept_code, d.dept_name
FROM l_employees
     LEFT OUTER JOIN l_departments
ON  e.dept_code = d.dept_code;
```

L_employees

Lname	Dept_code
BROWN	MKT
WOODS	-

L_departments

Dept_code	Dept_name
MKT	MARKETING
SAL	SALES

Result

Lname	Dept_code	Dept_name
BROWN	MKT	MARKETING
WOODS	-	-

ALL rows from l_employees are included in result. When there is no match for dept_code, a null value is inserted in the result.

In an inner join, Woods would not appear in the result.

Right Outer Join

- All rows from the right table (the one after the join keyword) are preserved.
- Example:

```
SELECT  e.last_name, d.dept_code, d.department_name
FROM l_employees e RIGHT OUTER JOIN l_departments d
ON  e.dept_code = d.dept_code;
```

- Translation: Display employees and their departments. Include all departments, even if there is no one assigned to the dept.

Right Outer Join

```
SELECT  e.last_name, d.dept_code, d.dept_name
FROM l_employees e
      RIGHT OUTER JOIN l_departments d
ON  e.dept_code = d.dept_code;
```

L_employees

Lname	Dept_code
BROWN	MKT
WOODS	-

L_departments

Dept_code	Dept_name
MKT	MARKETING
SAL	SALES

Result

Lname	Dept_code	Dept_name
BROWN	MKT	MARKETING
-	SAL	SALES

Both rows from l_departments are included in result. Even though there are no employees in the SALES department, null values are inserted in the result.

Full Outer Join

- Combination of RIGHT and LEFT outer join.
- Example:

```
SELECT  e.last_name, d.dept_code,
    d.department_name
FROM l_employees e
        FULL OUTER JOIN l_departments d
ON  e.dept_code = d.dept_code;
```

- Translation: Show me all employees, even if they do not have department assignment AND show me all departments even if there are no employees in the department.

Full Outer Join

```
SELECT  e.last_name, d.dept_code, d.dept_name
FROM l_employees e
      FULL OUTER JOIN l_departments d
ON  e.dept_code = d.dept_code;
```

L_employees

Lname	Dept_code
BROWN	MKT
WOODS	-

L_departments

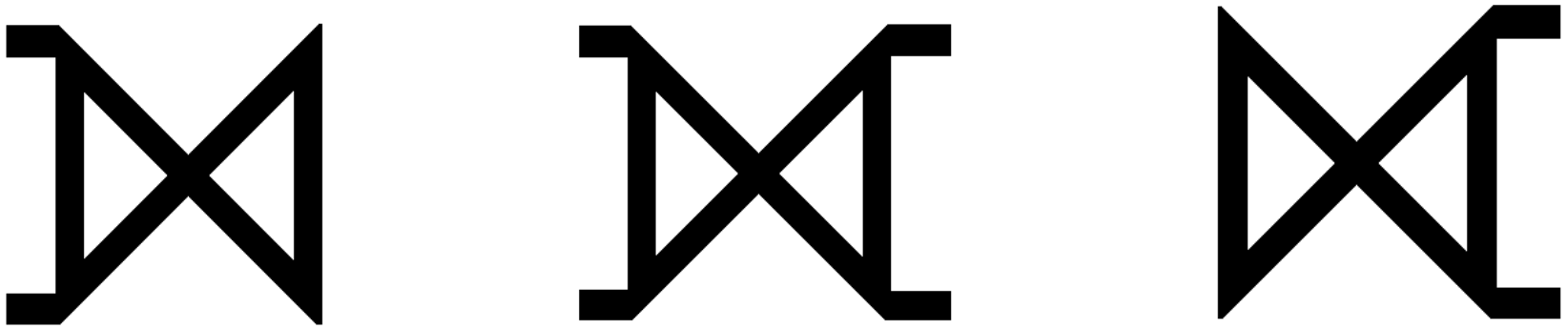
Dept_code	Dept_name
MKT	MARKETING
SAL	SALES

Result

Lname	Dept_code	Dept_name
BROWN	MKT	MARKETING
WOODS	-	-
-	SAL	SALES

Result set includes rows that satisfy join condition as well as those that don't.

Outer Joins – More Detail



Oracle Outer Joins – Older Syntax

- **Left** outer join example:

```
SELECT  e.last_name, d.dept_code, d.department_name
FROM l_employees e, l_departments d
ON  e.dept_code = d.dept_code (+);
```

- **Right** outer join example:

```
SELECT  e.last_name, d.dept_code, d.department_name
FROM l_employees e, l_departments d
ON  e.dept_code (+) = d.dept_code;
```

- Note: the plus sign in parens is not standard SQL. Plus sign appears on the side where nulls are added.

I would NOT recommend this syntax. It's here so that you will recognize it if you ever see it!

MySQL: Full Outer Join

- In a system that does not support full outer join, you can simulate it:

```
SELECT  e.last_name, d.dept_code, d.dept_name
FROM l_employees e LEFT OUTER JOIN l_departments d
ON  e.dept_code = d.dept_code
UNION
SELECT  e.last_name, d.dept_code, d.dept_name
FROM l_employees e RIGHT OUTER JOIN l_departments d
ON  e.dept_code = d.dept_code;
```

- Oracle, DB2, Microsoft support full outer join.
- **MySQL** does not support Full Outer Join; must simulate it.

Applications for Outer Joins - 1

- Find all employees and the number of lunches they will attend.
- Traditional (inner) join of l_employees and l_lunches will give us result set including employees who have attended 1 or more lunches.
- Must use an outer join to get those employees who have attended 0 lunches.
- This is the **“counting to zero” problem**.

The Code – Inner Join

```
SELECT e.employee_id, e.last_name,
    e.first_name, count (l.lunch_id)
FROM l_employees e
INNER JOIN l_lunches l
ON (e.employee_id = l.employee_id)
GROUP BY  e.employee_id,
    e.last_name, e.first_name ;
```

Results of Inner Join

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	COUNT(L.LUNCH_ID)
204	OWENS	ELLEN	2
205	PERKINS	HENRY	2
207	SMITH	DAN	2
203	WOODS	MARTHA	2
202	KERN	JIM	1
201	BROWN	SUSAN	3
208	CAMPBELL	FRED	2
210	HOFFMAN	NANCY	2

8 employees have attended at least one lunch.

The Code – *with* Outer Join

```
SELECT e.employee_id, e.last_name,
    e.first_name, count (l.lunch_id)
FROM l_employees e
LEFT OUTER JOIN l_lunches l
ON (e.employee_id = l.employee_id)
GROUP BY  e.employee_id,
    e.last_name, e.first_name ;
```

Results – Outer Join

Now we see all employees, even those who have not attended any lunches.

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	COUNT(L.LUNCH_ID)
206	ROSE	CAROL	0
204	OWENS	ELLEN	2
205	PERKINS	HENRY	2
209	JACOBS	PAULA	0
202	KERN	JIM	1
203	WOODS	MARTHA	2
207	SMITH	DAN	2
201	BROWN	SUSAN	3
208	CAMPBELL	FRED	2
210	HOFFMAN	NANCY	2

Applications for Outer Joins - 2

- Finding defects in a pattern.
- Create a table that contains the “perfect” pattern, i.e. some sequence of numbers.
- Then do an outer join to see if you have breaks in the pattern. The nulls will show you where the problems are.

Defects in a Pattern

num
1
2
3
4
5
6
7
8
9
10
11
12

perfect

num
1
2
3
3
5
6
7
8
9
9
10
11
11

test

```
SELECT p.num AS NUM, count(t.num) AS CNT
FROM perfect p LEFT OUTER JOIN  test t
        ON (p.num = t.num)
GROUP BY p.num
HAVING count(t.num) != 1
ORDER BY p.num;
```

NUM	CNT
3	2
4	0
9	2
11	2
12	0

Applications for outer join

- When do we want to include missing rows?
- It may be especially important when a certain action has not yet occurred.
- For example, a part exists in inventory, but there are **no** orders for that part yet. If we join parts and orders tables, the unordered parts do not show up. Must use an outer join to display these parts.

Outer Join – Rule of Thumb

- If you are joining two tables and using a summarization function, consider the need for an outer join.
- Do you need to see rows/groups with count or sum = 0?

Drawbacks to Full Outer Join

- Full outer join might cause problems when there are LOTS of rows in both tables that do not match.
- Save the full outer join for those cases when you suspect that just a few rows are not matching up or when the base tables are quite small.

References: Head First SQL

- Inner Joins: Chapter 8
- Outer Joins: Chapter 10
- How do I get to Head First SQL?
  - From vtc.edu, click on the library link.
  - Click on the tab for “Find Books & Videos”
  - Click on the link for Safari Tech eBooks.
  - In the search box, enter ‘SQL’ ; results should display ‘Head First SQL’.