Working with Oracle SQL

Chapter 3:

SQL Joins

Chapter Objectives

In this chapter, we will discuss:

- The use of `JOIN`s to retrieve data from multiple tables
- The difference between `INNER`, `OUTER`, and `CROSS JOIN`
- Joining a table to itself
- Cartesian Join
- SQL-89 vs SQL-92 syntax

Chapter Concepts

The Need for Joins

Inner Joins

Outer Joins

Self Joins

Cartesian Joins and SQL-89 Syntax

Chapter Summary

The Need for Joins

- A well designed Oracle database is normalized to 3rd Normal Form
  - 3NF means that the columns in each table describe:
    - The primary key
    - The WHOLE primary key
    - And NOTHING BUT the primary key
- The reason behind this concept is to ensure that data belongs with the other data it is associated with and dependent upon
  - This leads to having only one copy of each data element
  - This minimizes *update anomalies*
    - The application having to maintain multiple copies of the data element in more than one table

Three Normal Forms Are Most Often Practiced

- The physical implementation of most databases implement three normal forms to ensure data integrity
  - 1st Normal Form: declare a primary key and remove repeating groups
  - 2nd Normal Form: for composite keys, ensure functional dependency
  - 3rd Normal Form: functional dependency on only the primary key
- Additional Normal Forms:
  - 4th Normal Form: minimize the fields involved in a composite key
  - 5th Normal Form: removes all redundancies

Denormalized Data

- Employees work for a department
  - Departments have names
- Since requests for information about employees frequently want the department name on the listing, it is tempting to make the name a column in the employee table
- On the average, each department has around nine employees assigned to them
  - The implication is that the department name would have to be carried redundantly
    - Consuming more storage
    - Creating an *update anomaly*
      - What must we do if the department changes its name?

Normalized Data

- With 107 employees, this may not be an issue
  - Increasing table sizes compounds the problem
  - Having many denormalized data elements begins to exponentiate the problem
  - Eventually, an application person will forget (or not know) that the other copies of the data need to be maintained in synch
- This leads to a data integrity issue
  - Potentially, the most expensive problem
  - What is the impact on the business of a decision based upon inaccurate information?
- The solution is to design the database to 3NF
- In our case, this means that the SQL statement must *join* data from two or more tables to satisfy the request

Chapter Concepts

The Need for Joins

Inner Joins

Outer Joins

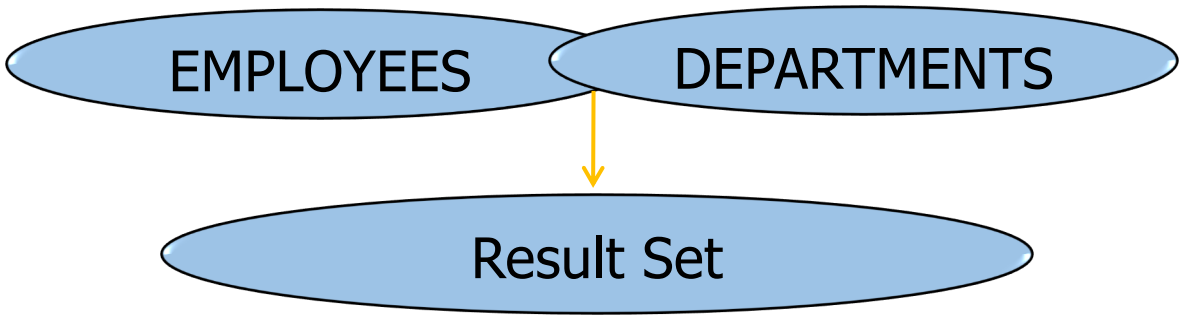
Self Joins

Cartesian Joins and SQL-89 Syntax

Chapter Summary

Preferred Types of Attribute Joins

- In order of preference, there are multiple ways to do an attribute join
  - A pre-defined Foreign Key to Primary Key relationship
  - An indexed attribute in one table that links to an indexed attribute in another table
    - An index is created by a database administrator
    - Results in faster performance of the `SELECT` clause
  - Columns without keys or indices, but that match in terms of data type and data value
    - Retrieval will be slower on these types of joins



Join Example

- Consider the `locations` and `departments` tables

LOCATIONS

LOCATION_ID	CITY
1700	Seattle
2500	Oxford
2700	Munich

DEPARTMENTS

LOCATION_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1700	30	Purchasing
1700	90	Executive
1700	110	Accounting
1700	120	Treasury
2500	80	Sales
2700	70	Public Relations

- Each department belongs to a particular location
  - We would like to list the departments and include the names of their locations

LOCATION_ID	CITY	DEPARTMENT_ID	DEPARTMENT_NAME
2700	Munich	70	Public Relations
1700	Seattle	90	Executive
1700	Seattle	110	Accounting
1700	Seattle	120	Treasury
2500	Oxford	80	Sales
1700	Seattle	30	Purchasing

…

Join Example (continued)

- The following query will produce the result on the previous slide:

```
SELECT locations.location_id
     , locations.city
     , departments.department_id
     , departments.department_name
FROM   departments
JOIN   locations
ON     departments.location_id = locations.location_id;
```

- Explanation:
  - A join returns columns from more than one table
    - In this case we need columns from two tables
  - The `ON` condition specifies how the rows in the tables relate to one another
  - The column names have prefixes to specify the table in which each column is located
    - Prefixes will be discussed in more detail later

Another Join Example

- For each employee, list the employee id, first name, last name, job id, and job title
  - `job_title` is a column in the `jobs` table
  - The other columns are located in the `employees` table

```
SELECT employees.employee_id
     , employees.first_name
     , employees.last_name
     , employees.job_id
     , jobs.job_title
FROM   employees
JOIN   jobs
ON     employees.job_id = jobs.job_id;

EMPLOYEE_ID FIRST_NAME   LAST_NAME    JOB_ID     JOB_TITLE
----------- ------------ ------------ ---------- ---------------
        100 Steven       King         AD_PRES    President
        111 Ismael       Sciarra      FI_ACCOUNT Accountant
        109 Daniel       Faviet       FI_ACCOUNT Accountant
        108 Nancy        Greenberg    FI_MGR     Finance Manager
…
```

Column Names

- A column name that occurs in more than one of the tables must have the table name as a prefix so as not to be ambiguous
- Prefixing all column names is strongly recommended, even when it is not strictly necessary
  - Improves readability
  - No risk of becoming ambiguous if:
    - New tables are added to the query
    - New columns are added to tables

FROM Clause and Table Alias

- FROM and JOIN clauses specify the tables being used in the statement
- A table can have an alias name
  - Also called *range variable* or *correlation name*
  - Like column aliases, the key word AS is specified in the standard
    - Most products do not require it
    - Oracle does *not allow* the key word AS with table alias names
    - For this reason, we will *omit* the AS with table alias names in the course examples
- The alias *replaces* the real table name *within* the query
  - The alias must be used as a prefix instead of the real table name
  - A table alias is useful to reduce typing and improve readability

Tables Alias Examples

```
SELECT last_name FROM employees;

LAST_NAME
----------------------
Abel
Ande
Atkinson
Austin
```

```
SELECT employees.last_name FROM employees;

LAST_NAME
----------------------
Abel
Ande
Atkinson
```

```
SELECT e.last_name FROM employees e;

LAST_NAME
----------------------
Abel
Ande
Atkinson
Austin
```

```
SELECT employees.last_name FROM employees e;
       *
ERROR at line 1:
ORA-00904: "EMPLOYEES"."LAST_NAME": invalid identifier
```

Re-write the Join with Table Aliases

- Using aliases in the previous example:
  - From:

```
SELECT employees.employee_id
     , employees.first_name
     , employees.last_name
     , employees.job_id
     , jobs.job_title
FROM   employees
JOIN   jobs
ON     employees.job_id = jobs.job_id;
```

  - To:

```
SELECT e.employee_id
     , e.first_name
     , e.last_name
     , e.job_id
     , j.job_title
FROM   employees e
JOIN   jobs j
ON     e.job_id = j.job_id;
```

JOIN and WHERE

- It is possible to combine a WHERE condition with a JOIN
  - Example: Restrict the previous example to include only the job id of FI_ACCOUNT

```
SELECT e.employee_id
     , e.first_name
     , e.last_name
     , e.job_id
     , j.job_title
FROM   employees e
JOIN   jobs j
ON     e.job_id = j.job_id
WHERE  e.job_id = 'FI_ACCOUNT';

EMPLOYEE_ID FIRST_NAME   LAST_NAME    JOB_ID     JOB_TITLE
----------- ------------ ------------ ---------- ---------------
        111 Ismael       Sciarra      FI_ACCOUNT Accountant
        109 Daniel       Faviet       FI_ACCOUNT Accountant
....
```

JOIN and Sorting

- It is impossible to predict the sort order of a JOIN result
  - Unless ORDER BY is specified
  - Example: Sort the result of the previous query by job title and employee ID

```
SELECT e.employee_id
     , e.first_name
     , e.last_name
     , e.job_id
     , j.job_title
FROM   employees e
JOIN   jobs j
ON     e.job_id = j.job_id
WHERE  e.job_id = 'FI_ACCOUNT'
ORDER BY job_title, employee_id;

EMPLOYEE_ID FIRST_NAME   LAST_NAME    JOB_ID     JOB_TITLE
----------- ------------ ------------ ---------- ---------------
        109 Daniel       Faviet       FI_ACCOUNT Accountant
        111 Ismael       Sciarra      FI_ACCOUNT Accountant
.....
```

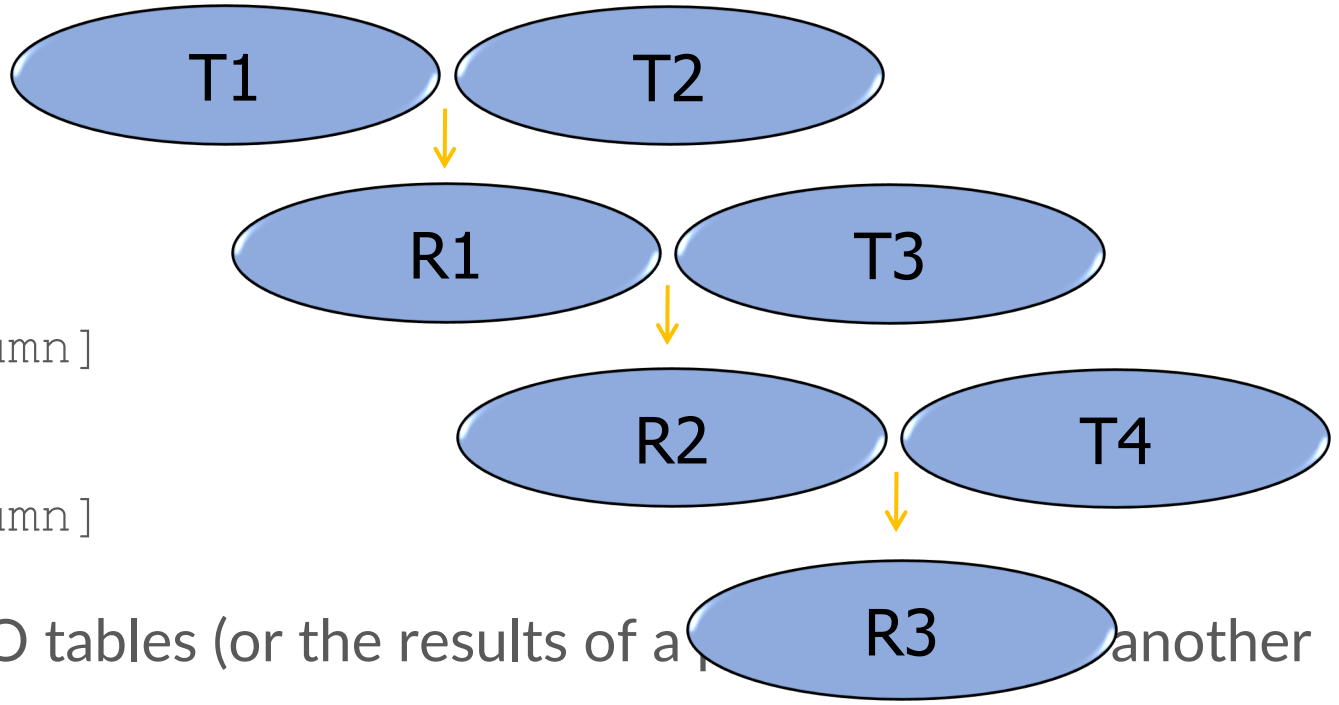
More than Two Tables

- There may be more than two tables in a JOIN
  - All tables must have a JOIN condition
  - The keywords JOIN and ON are repeated for each additional table
- Example: Include department name
  - Department name is a column in the departments table
  - The relationship between employees and departments is department ID

```
SELECT e.employee_id
     , e.first_name
     , e.last_name
     , e.job_id
     , j.job_title
     , d.department_name
FROM   employees e
JOIN   jobs j
ON     e.job_id = j.job_id
JOIN   departments d
ON     e.department_id = d.department_id
ORDER BY job_title, employee_id;
```

Visualizing Multiple Table JOINs

```
SELECT [columns]
FROM TABLE1 T1
JOIN TABLE2 T2
      ON T1.[column]=T2.[column]
JOIN TABLE3 T3
      ON table.[column]=T3.[column]
JOIN TABLE4 T4
      ON table.[column]=T4.[column]
```



- *Note*: each JOIN statement joins TWO tables (or the results of a previous JOIN with another table)

More than Two Tables (continued)

- Partial result of the query on the previous slide:

```
EMPLOYEE_ID FIRST_NAME   LAST_NAME    JOB_ID     JOB_TITLE       DEPARTMENT_NAME
----------- ------------ ------------ ---------- --------------- --------------------
        167 Amit         Banda        SA_REP     Sales Rep       Sales
        168 Lisa         Ozer         SA_REP     Sales Rep       Sales
        197 Kevin        Feeney       SH_CLERK   Shipping Clerk  Shipping
        198 Donald       OConnell     SH_CLERK   Shipping Clerk  Shipping
        199 Douglas      Grant        SH_CLERK   Shipping Clerk  Shipping
....
```

Other Join Conditions

- All the joins we have seen use the equals operator
  - Known as equijoins
- Joins may use any of the SQL conditional operators
  - Watch for multiple rows returned
  - These are usually less efficient than equijoins

```
SELECT e.first_name, e.last_name, e.salary, j.job_title, j.max_salary
FROM employees e
JOIN jobs j
ON e.salary BETWEEN j.min_salary AND j.max_salary
AND e.job_id != j.job_id
WHERE employee_id = 103;

FIRST_NAME LAST_NAME    SALARY JOB_TITLE                          MAX_SALARY
---------- ---------- ------- ------------------------------ ----------
Alexander  Hunold         8000 Accountant                           9000
Alexander  Hunold         8000 Public Accountant                    9000
…
```

Exercise 3.1: Working with `INNER JOIN`s

60 min

- Please complete this exercise in your Exercise Manual

Chapter Concepts

The Need for Joins

Inner Joins

Outer Joins

Self Joins

Cartesian Joins and SQL-89 Syntax

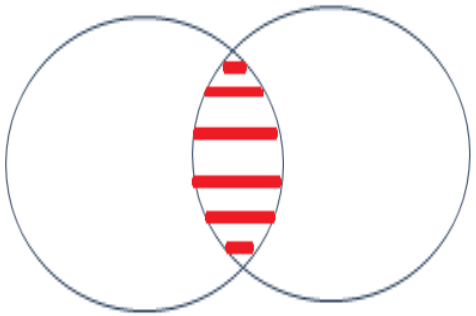
Chapter Summary

Inner and Outer Join

- The joins we have seen so far have all been *inner* joins
  - The result excludes rows that do *not* satisfy the join condition; i.e., have *no* matching rows in the other table
- Inner joins can be specified with the keyword `INNER JOIN`
  - This is the default, so the word `INNER` may be omitted, as we have seen
- It may be desirable to include rows from one table that have no matching row in the other table
  - This is called an *outer join*
  - Columns will contain `NULL` when there are no matching rows
- Common situations where there are no matching rows:
  - A primary key value with no corresponding foreign key values
    - There should never be foreign key values with no corresponding primary key value, because this would violate the referential integrity rule
  - A foreign key with `NULL` value

Inner Join Example

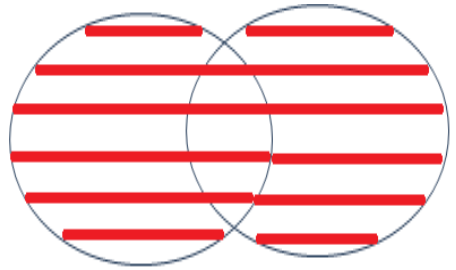
- Inner joins return rows where there is a match between two tables
  - This is the default join type
- Answers business questions like:
  - "I need an employee list with each employee's department"
- **Try It Now!**



```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id
```

  - How many records were returned?
  - Does this seem correct?

Full Outer Join Example



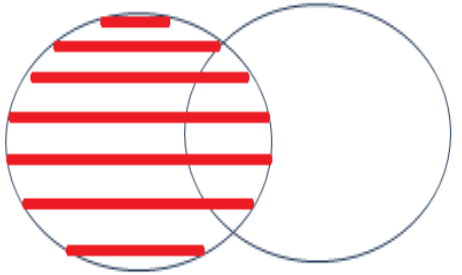
- Full outer joins return all rows from both tables, whether there is a match in the join column or not
  - NULL values are added to columns when there is no match
- Answers business questions like:
  - "I need a list of all the employees in the system and the department they work in. Be sure to include any employees that do not have a department, and any departments without employees"
- **Try It Now! and examine the results**

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
FULL OUTER JOIN departments d
ON e.department_id = d.department_id
```

Full Outer Join Example (continued)

- How many records were returned for the full outer join?
  - Were any records found with NULL in the first/last name?
  - What do you think these records represent?
  - Were there any records that NULL in the Department fields?
  - If present, what would these records represent?

Left Outer Join Example

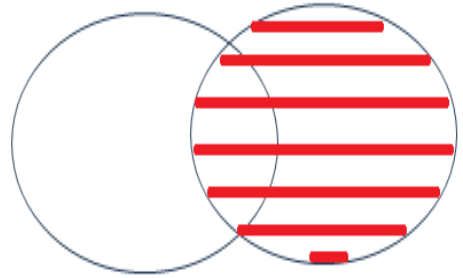


- Left outer join returns all rows from the table on the left plus any matching rows in the table on the right
  - NULL values are added to columns when there is no match
  - To determine which table is "Left", look at the `FROM` clause
    - The first table in the `FROM` clause is the left table
- Answers business questions like:
  - "I need the names of all the employees, and the department name (if available) associated with that ID. Include employees that are not assigned to a department."

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON e.department_id = d.department_id
```

Right Outer Join Example

- Right outer join returns all rows from the table on the right plus any matching rows in the table on the left
  - NULL values are added to columns when there is no match
  - To determine which table is “Right”, look at the `FROM` clause
    - The second table in the `FROM` clause is right table



```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON e.department_id = d.department_id
```

- This example can also be written as a Left Outer Join by changing the left and right tables in the join statement

Difference: `LEFT OUTER JOIN` and `RIGHT OUTER JOIN`?

```
SELECT e.first_name, e.last_name, d.department_name
FROM departments d
LEFT OUTER JOIN employees e
ON e.department_id = d.department_id
```

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON e.department_id = d.department_id
```

Outer Join with Non-Join Condition

- Be careful when combining an outer join and a non-join condition
  - This query only returns results where the department name matches the condition
    - There is no difference between this and the inner join

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON e.department_id = d.department_id
WHERE d.department_name LIKE 'P%';
```

  - This query returns all employees, but only matches departments that pass the condition
    - All others are `NULL` as you would expect in an outer join

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON e.department_id = d.department_id
AND d.department_name LIKE 'P%';
```

Exercise 3.2: Using `OUTER JOIN`s

45 min

- Please complete this exercise in your Exercise Manual

Chapter Concepts

The Need for Joins

Inner Joins

Outer Joins

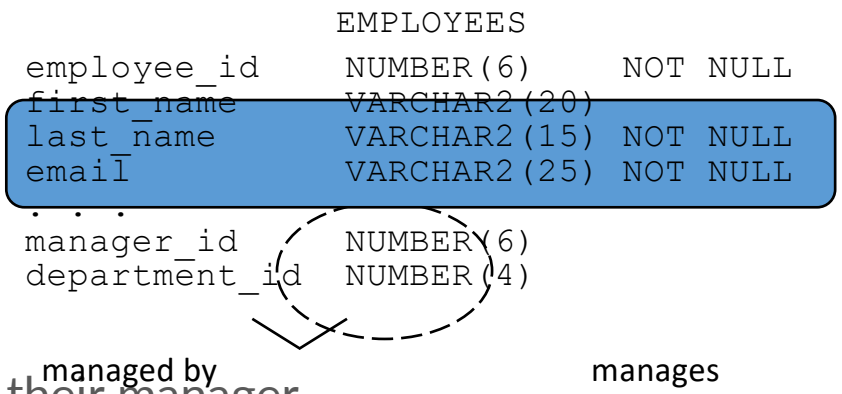
Self Joins

Cartesian Joins and SQL-89 Syntax

Chapter Summary

Self JOIN

- A table can be joined to itself
  - The query "sees" two identical copies of the table
  - Table aliases must be used to distinguish between them



- Example of self JOIN:
  - For each employee, list the name of the employee and the name of their manager
  - manager_id is a foreign key that references the manager's employee_id
  - OUTER JOIN is required in order to include employees who have no manager (i.e., NULL in the manager_id column)

```
SELECT e.employee_id, e.last_name, e.manager_id, m.last_name AS manager
FROM employees e
LEFT OUTER JOIN employees m
ON e.manager_id = m.employee_id
ORDER BY e.employee_id;
```

Chapter Concepts

The Need for Joins

Inner Joins

Outer Joins

Self Joins

Cartesian Joins and SQL-89 Syntax

Chapter Summary

SQL-89 Join Syntax

- The syntax we have used so far was introduced in SQL-92
- Consider this join:

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id
```

- In the older syntax, it would look like this:

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id
```

- There was no standard syntax for outer joins prior to SQL-92
  - The Oracle syntax involved putting `(+)` next to columns of the optional table in the `WHERE` clause

SQL-89 Cartesian Join

- What happens if you miss out the join condition?

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e, departments d
```

- **Try it now!**
  - How many rows are returned?
- Also try these queries

```
SELECT first_name, last_name FROM employees
```

```
SELECT department_name FROM departments
```

- Can you work out what has happened?

Cartesian Join

- In the previous example, a Cartesian Join (or Cartesian Product) was returned
  - We did not define the relationship between the two tables, so all records from the first table were combined with all records from the second table
- A Cartesian Join returns the columns for each table
  - Columns are returned side-by-side
    - If Table 1 has 10 columns, and Table 2 has 5 columns, the result set has 15 columns
  - This query returns a row for each combination between the two tables
    - If one table has 10 rows and the second table has 200 rows, then 2,000 rows would be returned
- There are limited uses for a Cartesian Join
  - Ask your instructor if he/she has ever had to generate one!

Cartesian Join in SQL-92 Syntax

- If you really need a Cartesian Join, you can produce it using the `CROSS JOIN`

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
CROSS JOIN departments d
```

- Prefer the SQL-92 syntax in all situations
  - It is more expressive
  - It is harder to inadvertently create Cartesian Joins in complex queries

Chapter Concepts

The Need for Joins

Inner Joins

Outer Joins

Self Joins

Cartesian Joins and SQL-89 Syntax

Chapter Summary

Chapter Summary

In this chapter, we have discussed:

- The use of `JOIN`s to retrieve data from multiple tables
- The difference between `INNER`, `OUTER`, and `CROSS JOIN`
- Joining a table to itself
- Cartesian Join
- SQL-89 vs SQL-92 syntax