Session-4
Displaying data from multiple tables

- To extract data from two or more than two tables *joining* between tables is needed.

  - Types of Joins:
    - Cross joins
    - Natural joins
    - USING clause
    - Full outer join
    - Left outer join
    - Right outer join

Cross Joins

- SELECT * FROM employees, departments;
- SELECT *

  FROM employees, departments

  WHERE departments.department_id = 10

  AND employees.salary > 10000;
- SELECT *

  FROM employees e, departments d

  WHERE d.department_id = 10

  AND e.salary > 10000;

Natural Joins

- The *Natural Join* clause is based on ALL columns in the two tables that have the same name.

- It selects rows from the two tables that have equal values in all matched columns.

- If the columns having the same names have different data types, an error is returned.

Natural Joins

- SELECT department_id, department_name, location_id, city
  FROM departments
  NATURAL JOIN locations;

- SELECT department_id, department_name, location_id, city
  FROM departments
  NATURAL JOIN locations
  WHERE department_id IN (20, 50);

USING clause

- If several columns have the same names but different data types than *USING* clause be used to specify the columns that should be used for an equijoin.
- Use the *USING* clause to match the only one column when more than one column matches.
- Do not use a table name or alias in the referenced columns.

```
SELECT location_id, street_address, postal_code, country_name
  FROM locations
  JOIN countries
  USING (country_id);
```

USING clause

- SELECT location_id, street_address, country_name
  FROM locations JOIN countries USING (country_id)
  WHERE country_id='CA';

- Couldn't use a table name or alias in the referenced columns. Below query will got error:

- SELECT l.location_id, l.street_address, c.country_name
  FROM locations l JOIN countries c USING (country_id)
  WHERE l.country_id='CA';

Table alias

- Use table aliases to simplify queries.

- Use table aliases to improve performance.

  - SELECT l.location_id, l.street_address, c.country_name
    FROM locations **l** JOIN countries **c** USING (country_id);

ON clause

- Use the *ON* clause to specify the columns to join.
- The join condition is separated from others search conditions.
- The ON clause makes code easy to understand.

- SELECT e.employee_id, e.last_name, e.department_id,
  d.department_id, d.location_id
  FROM employees e JOIN departments d
  ON (e.department_id = d.department_id);

- SELECT e.employee_id, e.last_name, e.department_id,
  d.department_id, d.location_id
  FROM employees e JOIN departments d
  ON (e.department_id = d.department_id)
  AND e.manager_id=149;

Three-way joins with the ON clause

- SELECT employee_id, city, department_name
  FROM employees e
  JOIN departments d
  ON d.department_id = e.department_id
  JOIN locations l
  ON(d.location_id = l.location_id);

OUTER Joins

- The join of two tables returning only matched rows is called an inner join.
- Joining tables with NATURAL join, USING or ON clauses results in an inner join.
- A join between two tables that returns the results of the inner join as well as the unmatched rows from the left(or right) table is called a left(or right) outer join.
- A join between two tables that returns the results of an inner join as well as the results of a left and right join is a Full outer join.

Three Types of Outer Join:

1. LEFT OUTER JOIN
2. RIGHT OUTER JOIN
3. FULL OUTER JOIN

LEFT OUTER Joins

- Inner Join:

```
SELECT employee_id, last_name, department_name
FROM employees e JOIN departments d
ON d.department_id = e.department_id;
```

***Returns 106 rows , missing data from both employees and departments table.

- Left Outer Join:

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

***Returns 107 rows , ALL the available data from employees table.

RIGHT OUTER Joins

- Inner Join:

```
SELECT employee_id, last_name, department_name
FROM employees e JOIN departments d
ON d.department_id = e.department_id;
```

***Returns 106 rows , missing data from both employees and departments table.

- Right Outer Join:

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

***Returns 122 rows , ALL the available data from departments table.

FULL OUTER Joins

- Inner Join:

```
SELECT employee_id, last_name, department_name
FROM employees e JOIN departments d
ON d.department_id = e.department_id;
```

***Returns 106 rows , missing data from both employees and departments table.

- Right Outer Join:

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

***Returns 123 rows , ALL the available data from both employees and departments table.

Cartesian product

- A Cartesian product is formed when:
  - A join condition is omitted.
  - A join condition is invalid.
  - All rows in the first table is joined to all rows in the second table.

    - SELECT * FROM employees, departments;

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

    - SELECT last_name , department_name
      FROM employees CROSS JOIN departments.

- To avoid Cartesian product, always include a valid join condition.