MULTI-TABLE QUERIES

UNIT 5. Part 1.

Aim

- Learn the different operations used for retrieving data from several tables.
  - UNION – EXCEPT – INTERSECT
  - Cartesian product – CROSS JOIN
  - INNER JOIN (Composición interna)
  - LEFT – RIGHT – FULL OUTER JOIN (Composición externa)
  - *Combine several operations*
  - *Derived tables*
  - *Reflexive queries*

Introduction

- In TRANSACT-SQL data can be retrieved from several tables using operations based on Relational Algebra.
- The result is always a logical table.
- Throughout this unit, the word *table* will be used to refer to:
  - a table
  - a temporary table
  - a query result
  - a derived table
  - a view

Introduction

- Operations are clasified in two groups:
  - Set operations: Same schema return same schema
    - UNION
    - INTERSECT
    - EXCEPT
  - JOINS: different schemas return concatenated schemas
    - Cartesian product – CROSS JOIN
    - INNER JOIN
    - OUTER JOIN
      - LEFT OUTER JOIN
      - RIGHT OUTER JOIN
      - FULL OUTER JOIN

Set operations

UNION

```
{< query >|(< query >)}
UNION [ALL]
{< query >|(< query >)}
[{UNION [ALL] {< query >|(< query >)}} [ ...n ] ]
[ORDER BY {column_expression|column_position
               [ASC|DESC]} [ ,...n ]]
```

- Returns the first query rows and below the second query rows.
- Querys have the SAME SCHEMA (Number of columns and compatible data types).
- The result has the same schema and inherits the first query headers.
- Querys cannot contain an ORDER BY clause.

UNION

```
SELECT office as OFI, city FROM Valencia
UNION ALL
SELECT office, city FROM Madrid;

SELECT office as OFI, city FROM Valencia
UNION ALL
SELECT office, city FROM Madrid
ORDER BY ofi;

SELECT office FROM Valencia
UNION
SELECT office FROM Madrid
UNION
SELECT office FROM Pamplona;
```

EXCEPT

```
{< query >|(< query >)}
EXCEPT
{< query >|(< query >)}
[{EXCEPT {< query >|(< query >)}} [ ...n ] ]
[ORDER BY {column_expression|column_position
              [ASC|DESC]} [ ,...n ]]
```

- Returns the first query rows that are not in the second query.
- Querys have the SAME SCHEMA (Number of columns and compatible data types).
- The result has the same schema and inherits the first query headers.
- Querys cannot contain an ORDER BY clause.

EXCEPT

```
SELECT  id FROM T1
EXCEPT
SELECT code FROM T2;

SELECT manid, productid FROM  products
EXCEPT
SELECT DISTINCT man, product FROM orders;
```

INTERSECT

{< query >|**(**< query >**)**}
INTERSECT
{< query >|**(**< query >**)**}
[{INTERSECT {< query >|**(**< query >**)**}} [...*n*]]
[ORDER BY {column_expression|column_position
[<u>ASC</u>|DESC]} [,...*n*]]

- Returns the first query rows that are also in the second query, the common rows.
- Querys have the SAME SCHEMA (Number of columns and compatible data types).
- The result has the same schema and inherits the first query headers.
- Querys cannot contain an ORDER BY clause.

INTERSECT

```
SELECT cod FROM T1
INTERSECT
SELECT codigo FROM T2;

SELECT manid, productid FROM  products
WHERE price> 20
INTERSECT
SELECT DISTINCT man, product FROM orders
WHERE amount >300;
```

JOINS

JOINS

- Tables can have any schema.
- The result schema is the first table schema concatenate with the second table schema.
- The operation is written in the FROM clause.
- Different kinds of composition:
  - Cartesian product CROSS JOIN (producto cartesiano)
  - INNER JOIN (composición interna)
  - OUTER JOIN (composition externa)
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN

CROSS JOIN

```
FROM {<source_table>} [ ,...n ]
    |<source_table> CROSS JOIN <source_table>
```

- <source_table> can be a table, a view or a derived table.
- Up to 256 <source_table> but also limited by resources.
- Each row from the first table is concatenated with each row of the second table.

CROSS JOIN

Por ejemplo:

```
SELECT *
FROM employees, offices;
```

Or

```
SELECT *
FROM employees CROSS JOIN offices;
```

CROSS JOIN

- It is not the most used composition. Usually we will concatenate an employee with **his/her** office.
- This could be:

```
SELECT *
FROM employees, offices
WHERE employees.office=offices.office;
```

But we have a more efficient operation.

INNER JOIN

- Express a cartesian product with a condition but it is more efficient.

```
FROM <source_table> INNER JOIN <source_table>
                                        ON <condition>
```

```
SELECT *
FROM employees INNER JOIN offices
        ON employees.office=offices.office;
```

```
SELECT *
FROM orders INNER JOIN products
  ON product = productid AND fab = manid;
```

INNER JOIN

- Usually used to join rows from related tables.
- Does not cover all cases:
- Employees with no office do not appear in the result, neither offices without employees.
- To solve this problem we have the OUTER JOIN.

OUTER JOIN LEFT/RIGHT/FULL JOIN

- In the result we have the INNER JOIN result rows plus the rows that are not combined. Which rows? It depends on the OUTER JOIN. The rows of the table that is on the LEFT/RIGHT side, or the rows of both tables (FULL)

FROM <source_table> {LEFT|RIGHT|FULL} [OUTER] JOIN <source_table> ON <condition>

OUTER is optional.

LEFT JOIN

```
SELECT numemp,nombre,employees.office, city
FROM employees LEFT JOIN offices
         ON employees.office=offices.office;
```

ALL the rows from the table that is on the left side.

Employees that have no office are combined with NULL values.

RIGHT JOIN

```
SELECT numemp,nombre,employees.office, city
FROM employees RIGHT JOIN offices
          ON employees.office=offices.office;
```

ALL the rows from the table that is on the right side.

Offices that have no employess are combined with NULL values.

FULL JOIN

```
SELECT numemp,nombre,employees.office, city
FROM employees FULL JOIN offices
         ON employees.office=offices.office;
```

All the rows

JOINS - Summary

- Combine each row1 with each row2 → CROSS JOIN.
- With a combining condition → INNER JOIN (at first)
- If there can be rows from table1 that do not comply with the condition AND they must appear in the result → LEFT JOIN.
- If there can be rows from table2 that do not comply with the condition AND they must appear in the result → RIGHT JOIN.
- If we have a LEFT and RIGHT → FULL JOIN.