University El Arbi ben M'Hidi Oum El Bouaghi
Level : LCS2
Dr Bouneb Messaouda
Chapter 06: SQL Language (Structed Query Language)

Chapter 06: SQL Language (Structed Query Language)

1. *Introduction :*

The relational DBMS offers several query languages, the most notable being SQL (Structured Query Language). It was first introduced in 1973 by a team of researchers from IBM and was quickly adopted as a standard.
The first standard was published in 1986 (SQL86).
SQL is available on all types of hardware, from microcomputers to the largest mainframes.

☺ ***Remark :*** *In all the examples in sections 2 and 3, we consider a database for managing the purchase and sale of products:*

- *A client is described by their client number, last name, first name, address, and locality, which specifies the city where their project is located. The "Category" field indicates whether the client is a frequent customer or a new customer. Specifically, the number of orders placed by each client is calculated, and clients are classified accordingly. Thus, a client is characterized by their account.*
- *Each product is described by its product number, description, quantity in stock, and price.*
- *A client can place orders for products at any time. An order is described by the order number, client number, order date, and the quantity of product ordered by the client.*

This database is described by the following relational schema:

CLIENT*(Client_N, LName, FName, Address, Locality, Category, Account)*
PRODUCT*(Prod_N, Description, Stock_Quantity, Price)*
ORDER*(Order_N, Client_N, Prod_N, Ordered_Quantity, Order_Date)*

2. *Definition of the Database Schema:*

SQL provides commands for defining and modifying structures. These commands allow you to define or create a table, delete an existing table, and add a column to an existing table.

2.1 Creating a Table:

*Creating a table is done using the **CREATE** command. This command is accompanied by various parameters specifying, among other things, access authorization conditions.*

The table creation operation generates a table it is a description of its columns (or attributes). For each column, its name and the type of its values are specified.

Example 01:
Creating a table for the **CLIENT** schema:

CLIENT(Client_N, LName, FName, Address, Locality, Category, Account)

```
create table CLIENT (
   Client-N  CHAR(6),
   LName   CHAR(12),
   FName  CHAR(12),
   Address  CHAR(50),
   Locality CHAR(20),
   Category  CHAR(12),
   Account  INTEGER(6)
);
```

2.2 Definition of a Key:

☞ *To define a primary key, the **primary key** command is used.*

Example 02:
*Defining a primary key for the **CLIENT** schema:*

```
create table CLIENT (
   Client-N CHAR(6),
   LName CHAR(12),
   FName CHAR(12),
   Address CHAR(50),
   Locality CHAR(20),
   Category CHAR(12),
   Account  INTEGER(6),
   primary key (Client-N)
);
```

Example 03:

☞ To define a foreign key, the foreign key command is used.

ORDER*(Order_N, Client_N, Prod_N, Ordered_Quantity, Order_Date)*

```
create table  order (
    Order_N CHAR(6),
    Client_N CHAR(6),
   Prod_N  CHAR(4),
    Ordered_Quantity  INTEGER(10),
    Order_Date INTEGER(8),
    primary key (Order_N),
    foreign key (Client_N) REFERENCES CLIENT(Client_N),
    foreign key (Prod_N) REFERENCES PRODUCT(Prod_N)
);
```

2.3 Deleting a Table:

*Any table can be deleted; it then becomes unknown, and its contents are lost. To delete a table, the **drop table** command is used.*

Example 04:

```
Drop table Order;
```

*This command deletes the **Order** table and all of its data permanently.*

2.4 Adding a Column:

*To add a column to an existing table, the **ALTER** and **ADD COLUMN** commands are used.*

Example 05:

Alter table PRODUCT

ADD column Weight DECIMAL(10,2);

3. Querying and Extracting Data:

The basic SQL instruction is ***Select****. Numerous variants of this instruction allow data to be extracted from one or more tables and presented, either to the user at a terminal or to the application program that requested its execution.*

Executing a ***Select*** *query produces a result that is a table. A simple query generally contains three main parts:*

- ***Select clause****: Specifies the values (column names or derived values) that make up each row in the result.*
- ***From clause****: Indicates the table(s) from which the result pulls its values.*
- ***Where clause****: Provides the selection condition that must be met by the rows providing the result.*

☺ ***Remark :*** *The selection condition introduced by the* ***Where*** *clause can consist of a boolean expression, which is either a simple condition formed by operators such as (=, <, >, <>) or a composite condition formed by combining simple conditions using logical operators (AND, OR, NOT).*

3.1 Data Extraction:

3.1.1 Simple Extraction:

This is the simplest query, which retrieves the values of specific columns from a table.

SQL Syntax:

Select x1, x2 */* where x1, x2 are the column names of table R */*

From R;

Example 3.1:

Retrieve the client number, name, first name, and locality:

Select Client-N, LName, FName, Locality

From CLIENT;

*This query extracts the client number, last name, first name, and locality from the **CLIENT** table.*

3.1.2 Retrieve All Information from a Table:

*To retrieve all the information from a table, the Select * command is used.*

Example 3.2:
Retrieve all the information about clients:

```
Select *     /* the asterisk * means all columns of the table */
From CLIENT;
```

*This query extracts all columns from the **CLIENT** table.*

3.1.3 Extracting Selected Rows:

*To extract specific rows based on conditions, the **Where** clause is used to filter the results.*

Example 3.3:
Retrieve the client numbers, last names, and first names of clients who live in the locality of Constantine:

```
Select Client-N, LName, FName
From CLIENT
Where Locality = 'constantine';
```

Example 3.4:
Retrieve the last name, first name, address, and account of clients whose locality is Constantine and category is C1:

```
Select LName, FName, Address, Account
From CLIENT
Where Locality = 'constantine' AND Category = 'C1';
```

Example 3.5:
Retrieve the names and first names of clients who belong to category C1, C2, or C3:

```
Select  LName, FName
```

From CLIENT

Where Category = 'C1' OR Category = 'C2' OR Category = 'C3';

3.1.4 Duplicate Rows in the Result:

In principle, the result of a query contains as many rows as there are in the starting table that satisfy the selection condition.

Example 3.6:
To perform a statistical study, we want to know the localities of clients in category C1.
Select Locality
From CLIENT
Where Category = 'C1';
The result of this query will contain duplicate rows. To eliminate redundancy, ***Distinct*** *is used:*
Select distinct Locality
From CLIENT
Where Category = 'C1';

☺ ***Remark :***

- *The existence of duplicate rows in a table can cause numerous issues.*
- *If the* ***Select*** *clause references a subset of columns and one of them is the table's identifier, the uniqueness of the result rows is guaranteed. In such cases, it is unnecessary to include* ***distinct****.*

3.2 Aggregate Functions:

SQL also includes predefined functions that provide an aggregated value calculated for selected rows. These functions are used to perform operations like counting, averaging, summing, and finding the maximum or minimum values.

Syntax:

- *Name-column* ***in (x1, x2, …, xn)****: The column value is one of x1, x2, ..., xn.*
- *Name-column* ***between valeur1 AND valeur2****: The column value is between valeur1 and valeur2.*
- ***Count (****Name-column****)****: Counts the number of rows.*

- ***AVG(****Name-column****)****: Calculates the average of a column.*
- ***Sum(****Name-column* ***)****: Calculates the sum of the values in a column.*
- ***Max(****Name-column* ***)****: Finds the maximum value in a column.*
- ***Min (****Name-column****)****: Finds the minimum value in a column.*

Example 3.7:
Retrieve the number of clients who have placed at least one order. (Note: Pay attention to duplicate rows.)
Select Count (distinct Client-Ni)
From ORDER;
This query counts the distinct number of client numbers from the ***ORDER*** *table, effectively ensuring that duplicate rows are not counted.*

3.3 Grouping Results, the Group by Clause:

The ***group by*** *clause allows you to partition the result set according to the values of one or more attributes. This logically divides the result into sub-relations containing tuples that have the same values for the specified grouping attributes.*

You can also use the ***having*** *clause: When present, it acts similarly to the* ***where*** *clause but is applied to the results after partitioning.*

Example 3.8:
Retrieve the sum of the quantities ordered for each product ordered:
Select Prod-N, SUM(Ordered_Quantity)
From ORDER
Group by Prod-N;
This query groups the ***ORDER*** *table by* ***Prod-N*** *(product number) and calculates the total ordered quantity for each product.*
Example 3.9:
Retrieve the products that have been ordered at least twice:
Select Prod-N
From ORDER
Group by Prod-N

Having count(*) > 2;

*This query groups the **ORDER** table by P**rod-N** and uses the **having** clause to filter products ordered more than twice. The **having** clause is used here because we are applying an aggregate function (**count(*)**) to the grouped results.*

3.4 Ordering Rows in a Result:

*The order of rows in the result of a query is arbitrary. It is possible to impose a specific order of presentation using the **Order by** clause.*

Syntax:

***Order by** type name-column /* where type is ASC or DESC */*

- ***ASC**: Ascending order (default).*
- ***DESC**: Descending order.*

Example 3.10:

Retrieve all data from the ORDER table, ordered by Order_Date in ascending order:

```
Select *
From  ORDER
Order by Order_Date ASC;
```

*This query retrieves all columns from the ORDER table and sorts the results by the Order_Date (order date) column in ascending order. If **ASC** is omitted, it is the default sorting order.*

3.5 Selection Using Multiple Tables:

*You can use **nested queries** (subqueries) to answer questions about a database. In short, subqueries allow you to select data based on conditions involving multiple tables.*

Example:

Retrieve the products that have been ordered by at least one client from Constantine.

```
Select *
From PRODUCT
Where Prod-N in (
            Select Prod-N
            From ORDER
           Where Client-N in (
                              Select Client-N
```

```
                                    From CLIENT
                                   Where Locality = 'constantine'
                                    )
                                 );
```

3.6 Multiple References to the Same Table:

You can make multiple references to the same table in a query to answer questions about the database. This is useful when comparing or retrieving data from the same table based on certain conditions.

Example:

Retrieve the clients who live in the same locality as the client with ***Ncli = 100****:*

```
Select Client-N, LNama, FName
From CLIENT
Where Locality in (
   Select Locality
   From CLIENT
   Where Client-N = 100
);
```

4. Set Operations:

In this section, we consider the management of a shopping mall with multiple stores.

4.1 Union:

The ***Union*** *command in SQL allows you to concatenate the results of multiple queries that use the* ***Select*** *statement. It is used to combine the results of two or more queries (tables). To use* ***UNION****, each query that is being concatenated must return the same number of columns, with the same data types, and in the same order.*

☺ ***Remark:***

- *By default, identical records will not be repeated in the results.*

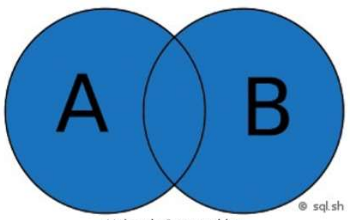
- *To include duplicate rows in the result, use the **Union ALL** command instead of **Union**.*

Syntax:

The syntax for combining the results of two tables without showing duplicates is as follows:

```
Select  *
From  table1
Union
Select *
From  table2;
```

Explanatory Schema:



Union de 2 ensembles

Example:

Imagine a shopping mall with several stores, and each store has a table listing its clients.

- *The table for store 1 is called **store1_client** and contains the following data:*

ClientID	*Name*	*Age*
1	*Alice*	*30*
2	*Bob*	*25*
3	*Carol*	*40*

- *The table for store 2 is called **store2_client** and contains the following data:*

ClientID	*Name*	*Age*
2	*Bob*	*25*
3	*Carol*	*40*
4	*Dave*	*35*

*Some clients appear in both tables. To avoid returning duplicate records, you can use the **UNION** query. The SQL query will look like this:*

```
SELECT *
FROM magasin1_client
UNION
SELECT *
FROM magasin2_client;
```

Result:

The result of this query will combine the rows from both tables but exclude duplicates:

ClientID	Name	Age
1	Alice	30
2	Bob	25
3	Carol	40
4	Dave	35

*The **UNION** operation ensures that duplicate records (such as Bob and Carol) are only shown once, merging the data from both tables without repetition.*

4.2 *UNION ALL:*

*The **UNION ALL** command in SQL is very similar to the **Union** command. It allows you to concatenate records from multiple queries, with the key difference being that **Union ALL** includes all records, even duplicates. This means that if the same record appears in the results of both concatenated queries, **Union ALL** will return this record twice.*

☺ ***Remark:***

*Like the **Union** command, both queries must return the same number of columns, with the same data types, and in the same order.*

Syntax:

*The SQL syntax to unite the results of two tables with **UNION ALL** is as follows:*

```
Select *
From table1
Union ALL
Select *
From table2;
```

Example:

Imagine a shopping mall that has databases for each of its stores. Each store has a table listing its clients with some information and the total purchases in the shopping mall.

- *The table **store1_client** corresponds to the first store.*

ClientID	Name	Total_Purchases
1	Alice	200
2	Bob	150

- *The table **store2_client** corresponds to the second store.*

ClientID	Name	Total_Purchases
2	Bob	100
3	Carol	250

*To concatenate all records from these tables, you can use a single query with the **Union ALL** command, as shown in the example below:*

```
Select *
From store1_client
Union ALL
Select *
From store2_client;
```

Result:

ClientID	Name	Total_Purchases
1	Alice	200
2	Bob	150
2	Bob	100
3	Carol	250

*The result table includes duplicates, showing that **Bob** is a customer in both stores. You might use a query like this to track loyal customers who shop at both stores, helping the mall reward them with incentives, such as gifts.*

4.3 *Intersection:*

*The **Intersect** command in SQL allows you to obtain the intersection of the results from two queries. This command retrieves the common records between two queries. It can be useful when you need to find if there are any matching data between two distinct tables.*

☺ ***Remark :***

*For the **Intersect** command to work properly, both queries must return the same number of columns, with the same types, and in the same order.*

Syntax:

*The syntax to use the **Intersect** command is as follows:*

*Select **

From table1

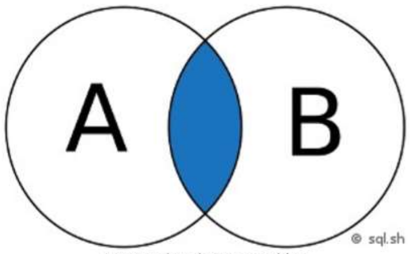
Intersect

*Select **

From table2;

*In this example, both tables must be similar (i.e., having the same columns, data types, and order). The result will correspond to the records that exist in both **table1** and **table2**.*

Explanatory Diagram:



Intersection de 2 ensembles

Example:

Imagine two stores in a shopping mall, and we want to find the common customers between the two stores.

- *The table **store1_client** corresponds to the first store.*

ClientID	***Name***	***Total_Purchases***
1	*Alice*	*200*
2	*Bob*	*100*

- *The table **store2_client** corresponds to the second store.*

ClientID	Name	Total_Purchases
2	Bob	100
3	Carol	250

*To obtain the list of customers who appear identically in both tables, you can use the **Intersect** command as follows:*

```
Select *
From store1_client
Intersect
Select *
From store2_client;
```

Result:

ClientID	Name	Total_Purchases
2	Bob	100

*The result presents the records of clients who are found in both the **store1_client** table and the **store2_client** table. In this case, **Bob** is the common client in both stores.*

4.4 *Difference:*

*In SQL, the **Minus** command is used between two tables to retrieve the records from the first table while excluding the records that are also present in the second table. If the same record appears in both tables, it will not appear in the final result.*

☺ ***Remark :***

*The name of this command may vary depending on the Database Management System (DBMS), and it can be called either **Minus** or **Except**.*

Syntax:

*The syntax for the **Minus** command is simple:*

```
Select *
From table1
```

Minus

*Select **

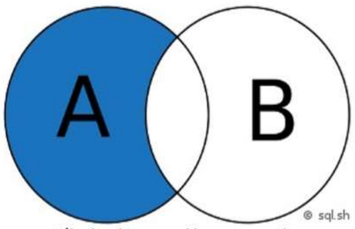
From table2;

*This query lists the records from **table1** without including those records that are also present in **table2**.*

Important:

The columns of the first query must be similar to the second query (i.e., the same number of columns, data types, and column order).

Explanatory Diagram:



Sélection d'un ensemble avec exception

Example:

Imagine a system used for managing a shopping mall. This system has two tables containing lists of customers:

- ***Table "clients_registered"**: Contains the names, surnames, and registration dates of frequent customers.*
- ***Table "clients_refus_email"**: Contains the information of customers who do not wish to be contacted by email to receive advertisements for new products from stores.*

The goal of this example is to select customers to send an advertising email. Therefore, customers from the second table (those who have refused emails) should not appear in the results.

- ***Table "clients_registered"**:*

LastName	*FirstName*	*date_registration*
Alice	*Dupont*	*2023-01-15*
Bob	*Martin*	*2023-02-20*
Emma	*Durand*	*2023-03-12*

- ***Table "clients_refus_email"****:*

LastName	*FirstName*
Bob	*Martin*
Lucas	*Petit*

To select only the names and surnames of users who have accepted to receive informational emails, the following SQL query is used:

Select LastName, FirstName

From clients_registered

Minus

Select LastName, FirstName

From clients_refus_email;

Result:

LastName	*FirstName*
Alice	*Dupont*
Emma	*Durand*

*This result shows customers who are in the **clients_registered** table but not in the **clients_refus_email** table. Therefore, the customers from the second table (those who refuse emails) are excluded from the result.*

5. Specific Operations
5.1 Projection:

Select list of attributes FROM table;

5.2 Join:

*The **JOIN** operation in SQL is used to combine rows from two or more tables based on a related column between them. Here's the basic syntax and types of joins:*

Basic Syntax:

```
SELECT column1, column2, ...
FROM table1
JOIN table2
ON table1.common_column = table2.common_column;
```

Types of Joins:

- ***INNER JOIN**:*

Returns records that have matching values in both tables.

***Syntax**:*

```
Select column1, column2, ...
From table1
INNER JOIN table2
ON table1.common_column = table2.common_column;
```

- ***LEFT JOIN (or LEFT OUTER JOIN)**:*

Returns all records from the left table and the matched records from the right table. If no match is found, NULL values are returned for columns from the right table.

***Syntax**:*

```
Select column1, column2, ...
From table1
LEFT JOIN table2
ON table1.common_column = table2.common_column;
```

✓ ***RIGHT JOIN (or RIGHT OUTER JOIN)***:

Returns all records from the right table and the matched records from the left table. If no match is found, NULL values are returned for columns from the left table.

Syntax:

```
Select column1, column2, ...
From table1
RIGHT JOIN table2
ON table1.common_column = table2.common_column;
```

✓ ***FULL JOIN (or FULL OUTER JOIN)***:

Returns records when there is a match in either the left or right table. It returns NULL for unmatched rows from both tables.

Syntax:

```
Select column1, column2, ...
From table1
FULL JOIN table2
ON table1.common_column = table2.common_column;
```

✓ ***CROSS JOIN***:

Returns the Cartesian product of both tables. Every row from the first table is combined with every row from the second table.

Syntax:

```
Select column1, column2, ...
From table1
CROSS JOIN table2;
```

Example:

If you have two tables, customers and orders, and you want to get a list of all customers with their orders, you would use an INNER JOIN like this:

```
Select customers.name, orders.order_date
From customers
INNER JOIN orders
ON customers.customer_id = orders.customer_id;
```

This will return only customers who have placed orders, matching the customer_id between the two tables.

5.3 Division:

Currently, there is no direct equivalent to division in SQL. However, it is still possible to find an alternative solution, particularly through the use of aggregation and grouping operations.

6. Example of database management with SQL:

Consider the following relational database schema:

ARTICLES (<u>NART</u>, LABEL, STOCK, PRICE_S)
SUPPLIER (<u>NS</u>, NAME_S, ADR_S, CITY_S)
PURCHASED (<u>#NS, #NART</u>, PRICE_P, DEADLINE)

***Question 1 / 8**: Numbers and labels of the items whose stock is less than 10?*

```
SELECT NART, LABEL
FROM ARTICLES
WHERE STOCK < 10;
```

***Question 2 / 8**: List of items whose sale price is between 100 and 300?*

```
SELECT *
FROM ARTICLES
WHERE PRICE_S BETWEEN 100 AND 300;
```

***Question 3 / 8**: List of suppliers whose address is unknown?*

```
SELECT *
FROM SUPPLIER
WHERE ADR_S IS NULL;
```

Question 4 / 8*: List of suppliers whose name starts with "STE"?*

```
SELECT *
FROM SUPPLIER
WHERE NAME_S  LIKE 'STE%';
```

Question 5 / 8*: Names and addresses of suppliers who provide items for which the supply lead time is greater than 20 days?*

```
SELECT NAME_S, ADR_S
FROM SUPPLIERS
WHERE NS IN (SELECT NS
             FROM PURCHASED
             WHERE DEADLINE > 20);
```

Question 6 / 8*: Numbers and labels of items sorted in descending order of stock?*

```
SELECT NART, LABEL
FROM ARTICLES
ORDER BY STOCK DESC;
```

Question 7 / 8*: Which suppliers supply both items 100 and 106, and at what price?*

```
SELECT NS, NART, PRICE_P
FROM PURCHASED
WHERE NART = 100 OR NART = 106;
```

Question 8 / 8*: Which supplier(s) supply the most products?*

```
SELECT NS
FROM PURCHASED
GROUP BY NS
HAVING COUNT(NART) = (SELECT MAX(COUNT(NART))
                  FROM PURCHASED
                  GROUP BY NS);
```