CHAPTER IV : QL (Structured Query Language)

1. INTRODUCTION

SQL, which stands for Structured Query Language, is a programming language used for managing and manipulating relational databases. It was created in the 1970s by IBM and became a de facto standard for data management in many applications and systems.

The history of SQL dates back to the 1970s when researchers at IBM, notably Donald D. Chamberlin and Raymond F. Boyce, began working on a query language for relational databases. This work led to the development of SEQUEL (Structured English Query Language) in 1974, which was later renamed SQL due to trademark issues.

- **1974**: Donald D. Chamberlin and Raymond F. Boyce at IBM developed SEQUEL (Structured English Query Language), a query language for relational databases.
- **1979**: Oracle Corporation (then known as Software Development Laboratories) developed its own version of SQL for its Oracle database management system.
- **1986**: The American National Standards Institute (ANSI) adopted SQL as the official standard for relational databases, and this standardized version is often referred to as SQL-86.
- **1989**: The International Organization for Standardization (ISO) also adopted SQL as an international standard, known as SQL-89 or SQL1.
- **1992**: SQL-92, also known as SQL2, became the first major revision of the SQL standard, introducing new features such as triggers, views, distributed transactions, etc.
- **1999**: SQL-99, also known as SQL3, was a major update to the SQL standard, introducing features like regular expressions, stored procedures, and the addition of data types such as geometric types and XML types.

  Since then, several revisions of the SQL standard have been released, each adding new features and improving the capabilities of the language. SQL implementations have proliferated with the development of different database management systems (DBMS), with the most popular being MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, SQLite, and others.

  Today, SQL remains the standard language for interacting with relational databases, and its importance in data management continues to grow with the rise of web, mobile, and enterprise applications.

2. CONSTITUENTS OF SQL LANGUAGE

SQL is a declarative language, meaning it is not strictly a programming language but rather a standard interface for accessing databases. It consists of four sublanguages:

- **Data Definition Language (DDL)**: Used to create and delete objects in the database (tables, integrity constraints, views, etc.).
  Example commands: CREATE, DROP, ALTER
- **Data Control Language (DCL)**: Used to manage access rights on database objects (creating users and assigning their rights).
  Example commands: GRANT, REVOKE
- **Data Manipulation Language (DML)**: Used to query, insert, update, and delete data. DML is based on relational operators, with added functions for aggregate calculations

and commands for performing insert, update, and delete operations.
Example commands: INSERT, UPDATE, DELETE, SELECT

- **Transaction Control Language (TCL)**: Used for transaction management (committing or rolling back changes to data in the database).
Example commands: COMMIT, ROLLBACK
In this section, we focus on the Data Manipulation Language (DML) of SQL, which is indeed the most widely used part in practice.

2.1 SQL SELECT

The most common use of SQL is to read data from a database. This is done using the SELECT command, which returns records in a result set. This clause allows users to specify the specific columns they wish to retrieve in a query, as well as criteria to filter the data.
The basic syntax for the SELECT clause is as follows:

```
SELECT column_name
FROM table_name;
```

Example:

Imagine a database called "Student" that contains information about students at a university.
Table "Student":

ID	FirstName	LastName	BirthDate	City
1	Nadir	Benslimane	21/03/2012	Alger
2	Lyes	Hammam	05/05/2010	Bouira
3	Abir	Saoucha	13/08/2011	Msila
4	Sofiane	Bakhti	02/01/2010	Msila
5	Hind	Boubakri	25/11/2012	Setif

If we want to get a list of all the cities of the students, we simply perform the following query:

```
SELECT City
FROM Student;
```

This will return the following result:

City
Alger
Bouira
Msila
Msila
Setif

- It is possible to read multiple columns at once by simply writing the names of the desired columns after SELECT and separating them with commas. For example, to get the names and last names of the students, we should execute the following query:

```
SELECT FirstName, LastName

FROM Student;
```

It is also possible to return all columns from a table without listing their names by using the "*" *character (asterisk). The* "*" character in a SELECT clause in SQL is a shortcut that means "all columns."

Example:

```
SELECT * FROM Student;
```

2.2 SQL DISTINCT

In SQL, using the SELECT command allows you to retrieve data from one or more columns in a table. However, this command may potentially display duplicate rows, making the results less readable. To avoid these redundancies, we can simply add the DISTINCT keyword after SELECT. This eliminates duplicate rows from the result and only presents unique values. The basic syntax to use DISTINCT is as follows:

```
SELECT DISTINCT column1, column2, ...

FROM table_name;
```

Example:

In the previous example, displaying the cities of the students gives duplicate city names. To avoid this redundancy, we simply add the DISTINCT keyword after SELECT as follows:

```
SELECT DISTINCT City

FROM Student;
```

This will give the following result:

City
Alger
Bouira
Msila
Setif

2.3 SQL AS (Alias)

In SQL, an alias is an alternative name that we can assign to a table or column in a query to make the results more readable or to shorten the code. Aliases are often used to simplify long or complex names and temporarily rename columns or tables.

```
SELECT FirstName AS FN, LastName AS LN

FROM Student;
```

This syntax can also be written as:

```
SELECT FirstName FN, LastName LN

FROM Student;
```

However, it is preferable to use the **AS** command for clarity (it is easier to read than just a space).

Alias on a table:

```
SELECT *
FROM Student AS S;


SELECT S.FirstName, S.LastName
FROM Student AS S;
```

2.4 SQL WHERE

In SQL, the WHERE command is used to filter the results of a query by specifying a ondition. This condition is typically a logical expression that is evaluated for each row in the table, and only the rows where the condition is true are included in the final result.

The WHERE command is used in conjunction with a SELECT query as follows:

```
SELECT column_names
FROM table_name
WHERE condition;
```

Example:

To display only the students from the city of M'sila, write:

```
SELECT *
FROM Student
WHERE City = 'Msila';
```

This will give the following result:

ID	FirstName	LastName	BirthDate	City
3	Abir	Saoucha	13/08/2011	Msila
4	Sofiane	Bakhti	02/01/2010	Msila

Comparison operators

In the SQL WHERE command, various comparison operators can be used to specify conditions that must be met by the data. These operators allow you to filter rows based on specific criteria. Here are the main comparison operators that can be used with WHERE:

- **Equality Comparison**:
  - =: Equal to
  - **!=** or <>: Not equal to

- **Numeric Comparison**:
    - <: Less than
    - >: Greater than
    - <=: Less than or equal to
    - >=: Greater than or equal to
- **Text Comparison**:
    - **LIKE**: Matches a pattern (typically used with wildcard characters like % to represent multiple characters or _ for a single character)
    - **NOT LIKE**: Does not match a pattern
- **Range Comparison**:
    - **BETWEEN**: Matches a range of values
    - **NOT BETWEEN**: Does not match a range of values
- **List Comparison**:
    - **IN**: Matches a list of values
    - **NOT IN**: Does not match a list of values
- **Null Value Comparison**:
    - **IS NULL**: Checks if a value is null
    - **IS NOT NULL**: Checks if a value is not null

2.5. SQL AND & OR

In SQL, the logical operators **AND** and **OR** are used to combine multiple conditions in a **WHERE** clause. They allow specifying more complex conditions to filter data from a table.

- **AND** ensures that both condition1 **and** condition2 are true:

```
SELECT column_names
FROM table_name
WHERE condition1 AND condition2;
```

- **OR** checks if either condition1 or condition2 is true:

```
SELECT column_names
FROM table_name
WHERE condition1 OR condition2;
```

These operators can be combined endlessly and mixed. The example below filters the results of the table table_name if either condition1 **and** condition2 **or** condition3 is true:

```
SELECT column_names
FROM table_name
WHERE condition1 AND (condition2 OR condition3);
```

Note: It is important to use parentheses when needed to avoid errors and to improve query readability.

Example:

Consider the following stock table:

Id	Designation	Category	Price	Quantity
1	Armoire	Meuble	12500	14
2	Chaise	Meuble	6000	35
3	Table	Meuble	9000	12
4	Porte Manteau	Meuble	7500	25
5	PC	Informatique	56000	20
6	Imprimante	Informatique	31000	5

```
SELECT *
FROM produit
WHERE Categorie = 'Meuble' AND Quantite < 20;
```

This will display the following:

Id	Designation	Category	Price	Quantity
1	Armoire	Meuble	12500	14
3	Table	Meuble	9000	12

```
SELECT *
FROM produit
WHERE Categorie = 'Informatique' OR Quantite >= 20;
```

This will display the following:

Id	Designation	Category	Price	Quantity
2	Chaise	Meuble	6000	35
4	Porte Manteau	Meuble	7500	25
5	PC	Informatique	56000	20
6	Imprimante	Informatique	31000	5

```
SELECT *
FROM produit
WHERE (Categorie = 'Informatique' AND Quantite < 20)
OR (Categorie = 'Meuble' AND Quantite < 15);
```

This will display the following:

Id	Designation	Category	Price	Quantity
1	Armoire	Meuble	12500	14
3	Table	Meuble	9000	12
6	Imprimante	Informatique	31000	5

2.6. SQL IN

In SQL, the `IN` command is used to specify a condition where the value of a column must match **one of the values** provided in a list, following the syntax:

```
SELECT column_name
FROM table
WHERE column_name IN (value1, value2, value3, ...);
```

Example :

```
SELECT *
FROM Student
WHERE City_St IN ('Alger', 'Msila');
```

Returns the following result:

ID	Last_Name	First_Name	Birth_Date	City_St
1	Benslimane	Nadir	21/03/2012	Alger
3	Saoucha	Abir	13/08/2011	Msila
4	Bakhti	Sofiane	02/01/2010	Msila

Notes:

- To negate the IN command in SQL, we can use NOT IN. This allows us to `select` records that do not match any value in the specified list.

```
SELECT *

FROM Student

WHERE City_St NOT IN ('Alger', 'Msila');
```

Returns the following result:

ID	Last_Name	First_Name	Birth_Date	City_St
2	Hammam	Lyes	05/05/2010	Bouira
5	Boubakri	Hind	25/11/2012	Setif

The syntax used with the IN operator is simpler than using multiple OR operators. To illustrate this clearly, here are two queries that return the same result — one using IN, and the other using multiple OR conditions:

```
SELECT *

FROM Student

WHERE City_St = 'Alger' OR City_St = 'Setif' OR City_St = 'Msila';

-----------------------

SELECT *

FROM Student

WHERE City_St IN ('Alger', 'Setif', 'Msila');
```

2.7. SQL BETWEEN

The BETWEEN operator is used in an SQL query to select a range of data in a query using WHERE. The range can consist of strings, numbers, or dates. The "Between" command is executed as follows:

```
SELECT *
FROM table
WHERE column_name BETWEEN 'value1' AND 'value2'
```

Example:

```
SELECT *
FROM STUDENT
WHERE Date_N_Et BETWEEN # 2011-01-01# AND #2012-12-31#;
```

Results:

ID	Last_Name	First_Name	Date_Of_Birth	City
1	Benslimane	Nadir	21/03/2012	Algiers
3	Saoucha	Abir	13/08/2011	Msila
5	Boubakri	Hind	25/11/2012	Setif

Note: The NOT BETWEEN command in SQL is used to select values outside a specified range. It is used when we want to exclude values that are within a given range.

Example:

```
SELECT *
FROM STUDENT
WHERE Date_N_Et NOT BETWEEN  # 2011-01-01# AND #2012-12-31#;
```

Results:

ID	Last_Name	First_Name	Date_Of_Birth	City
2	Hammam	Lyes	05/05/2010	Bouira
4	Bakhti	Sofiane	02/01/2010	Msila

2.8. SQL LIKE

The LIKE operator is used in the WHERE clause of SQL queries. It allows performing searches based on specific patterns. For example, it is possible to search for records where the value of a column starts with a particular letter. The search patterns can vary.

The syntax to use the LIKE operator is as follows:

```
SELECT *
FROM table
WHERE column LIKE pattern
```

The pattern usually looks like one of the following examples:

- `'A%'` matches any string starting with "A". The `%` character is a wildcard that replaces any other characters.
- `'%A'` matches any string ending with "A".
- `'%A%'` matches any string containing "A" anywhere.
- `'A_B'` matches strings starting with "A" and having one character in the middle, followed by "B".

Example:

```
SELECT *
FROM STUDENT
WHERE Last_Name LIKE 'B%'
```

Results:

ID	Last_Name	First_Name	Date_Of_Birth	City
1	Benslimane	Nadir	21/03/2012	Algiers
4	Bakhti	Sofiane	02/01/2010	Msila
5	Boubakri	Hind	25/11/2012	Setif

```
SELECT *
FROM STUDENT
WHERE Last_Name LIKE '%m%'
```

Results:

ID	Last_Name	First_Name	Date_Of_Birth	City
1	Benslimane	Nadir	21/03/2012	Algiers
2	Hammam	Lyes	05/05/2010	Bouira

2.9. SQL IS NULL / IS NOT NULL

"IS NULL" and "IS NOT NULL" are operators used in SQL to check if a value in a database column is NULL or not NULL.

- "IS NULL" is used to filter rows where a column's value is NULL.
- "IS NOT NULL" is used to filter rows where a column's value is not NULL.

To select records where the column values are NULL, use the following syntax:

```
SELECT *
FROM table
WHERE column_name IS NULL
```

To select records that are not NULL, use the following syntax:

```
SELECT *
FROM table
WHERE column_name IS NOT NULL
```

Example: Consider the following Contacts table:

ID	Last_Name	First_Name	Email
1	Cheride	Islam	Chride.Is@gmail.com
2	Kouissi	Ahlam	NULL
3	Madoui	Karim	NULL
4	Charif	Halim	Charif.Halim@gmail.com

The query:

```
SELECT * FROM Contacts WHERE Email IS NULL
```

Results:

ID	Last_Name	First_Name	Email
2	Kouissi	Ahlam	NULL
3	Madoui	Karim	NULL

The query:

```
SELECT * FROM Contacts WHERE Email IS NOT NULL
```

Results:

ID	Last_Name	First_Name	Email
1	Cheride	Islam	Chride.Is@gmail.com
4	Charif	Halim	Charif.Halim@gmail.com

2.10. SQL UNION

The UNION operator in SQL is used to combine the results of two or more SELECT queries into a single result set, including only distinct values. Each SELECT query in the UNION must have the same number of columns in the result with compatible data types.

The syntax to combine results from 2 tables (2 queries) without displaying duplicates is as follows:

```
SELECT * FROM table1
UNION
SELECT * FROM table2
```

If we want to include all occurrences of each selection, including duplicates, we can use UNION ALL as follows:

```
SELECT * FROM table1
UNION ALL
SELECT * FROM table2
```

Example: Consider the two tables from different departments:

EmpServ1

Id_Emp	Name_Emp	Last_name_Emp
1	Cheride	Islam
2	Kouissi	Ahlam
3	Madoui	Karim
4	Charif	Halim

EmpServ2

Id_Emp	Name_Emp	Last_name_Emp
1	Cheride	Islam
2	Kouissi	Ahlam
5	Abdelli	Nassim
6	Mazari	Yasser

```
SELECT * FROM EmpServ1
UNION
SELECT * FROM EmpServ2
```

Results:

ID_Emp	Last_Name	First_Name
1	Cheride	Islam
2	Kouissi	Ahlam
3	Madoui	Karim
4	Charif	Halim
5	Abdelli	Nassim
6	Mazari	Yasser

Whereas the query:

```
SELECT * FROM EmpServ1
UNION ALL
SELECT * FROM EmpServ2
```

Results (including duplicates):

ID_Emp	Last_Name	First_Name
1	Cheride	Islam
2	Kouissi	Ahlam
3	Madoui	Karim
4	Charif	Halim
1	Cheride	Islam
2	Kouissi	Ahlam
5	Abdelli	Nassim
6	Mazari	Yasser

2.11. SQL GROUP BY

The GROUP BY operation in SQL is used to group records that share the same value in a specified column, then apply an aggregate function to these groups.

Generally, the GROUP BY command is used as follows:

```
SELECT column1, function(column2)
FROM table
GROUP BY column1
```

Example: Consider the following Invoices table:

Id_Fac	Id_Client	Total_Fac	Date
1	3	356000	12/05/2021
2	2	452000	05/11/2021
3	3	600000	15/03/2022
4	1	752500	05/06/2022
5	2	157000	11/08/2022

The query:

```
SELECT ID_Client, SUM(Total_Fac)
FROM Facture
GROUP BY ID_Client
```

Results:

ID_Client	Total_Fac
1	752500
2	409000
3	656

2.12. Statistical Functions in SQL

In SQL, we can use various statistical functions to analyze our data. These are statistical aggregation functions, and the main ones are as follows:

1. **COUNT()**: Counts the number of rows in a result set.
   Example: `SELECT COUNT(*) FROM Table1;`
2. **SUM()**: Calculates the sum of the values in a column.
   Example: `SELECT SUM(Quantite) FROM Ventes;`
3. **AVG()**: Calculates the average of the values in a column.
   Example: `SELECT AVG(Prix) FROM Produits;`
4. **MIN()**: Returns the minimum value of a column.
   Example: `SELECT MIN(Prix) FROM Produits;`
5. **MAX()**: Returns the maximum value of a column.
   Example: `SELECT MAX(Prix) FROM Produits;`
6. **STDDEV()/STDEV()**: Calculates the standard deviation of the values in a column.
   Example: `SELECT STDDEV(Prix) FROM Produits;`
7. **VARIANCE()/VAR()**: Calculates the variance of the values in a column.
   Example: `SELECT VARIANCE(Prix) FROM Produits;`

These statistical functions can be used to analyze data in a relational database and obtain useful insights into data distribution, central tendency, and dispersion.

2.13. SQL HAVING

The HAVING operator in SQL is used to filter results obtained by a group after applying the GROUP BY operator. Unlike the WHERE operator, which filters rows before the data is grouped, HAVING allows you to filter data after the grouping has been done. This means you can use HAVING to apply conditions on aggregate functions such as SUM(), AVG(), MAX(), MIN(), and COUNT(), which cannot be used with WHERE.

The HAVING operator is used as follows:

```
SELECT column1, SUM (column2)
FROM table_name
GROUP BY colomn1
HAVING fonction (colomc2) operator value
```

Example
Let's imagine a table called "purchase" that contains purchases made by different customers:

Id_Achat	Name_Cl	Last_name_Cl	Purchase_Amount	Purchase_date
1	Cheride	Islam	5600	21/02/2024
2	Kouissi	Ahlam	2100	01/03/2024
3	Cheride	Islam	800	14/03/2024
4	Charif	Halim	3200	25/03/2024
5	Kouissi	Ahlam	1000	01/04/2024
6	Kouissi	Ahlam	4800	09/04/2024

To find the customers who made purchases totaling more than 6000 DA, we use the following query:

```
SELECT Name_Cl, Last_name_Cl, SUM (purchase_amount)
FROM purchase
GROUP BY Name_Cl
HAVING SUM (purchase_amount) > 6000
```

Which gives the following result:

Name_Cl	Last_name_Cl	SUM(purchase_amount)
Cheride	Islam	6400
Kouissi	Ahlam	7900

2.14. SQL ORDER BY

The SQL ORDER BY clause is used to sort the results of a query by one or more columns in a specified order. This clause allows us to sort results in ascending order (by default) or descending order based on our query needs.

The syntax of this clause is:

```
SELECT column1, column2
FROM table
ORDER BY column1
```

By default, the results are sorted in ascending order. However, it is possible to reverse the order by using the `DESC` suffix after the column name. Furthermore, it is possible to sort by multiple columns by separating them with a comma. A more elaborate query would look like this:

```
SELECT column1, column2, column3
FROM table
ORDER BY column1 DESC, column2 ASC
```

Example
From the previous "achat" table, the following query:

```
SELECT *
FROM purchase
ORDER BY purchase_amount
```

Displays the purchase table in ascending order based on the purchase amount:

Id_Achat	Name_Cl	_Last_name_Cl	Purchase_amount	Purchase_date
3	Cheride	Islam	800	14/03/2024
5	Kouissi	Ahlam	1000	01/04/2024
2	Kouissi	Ahlam	2100	01/03/2024
4	Charif	Halim	3200	25/03/2024
6	Kouissi	Ahlam	4800	09/04/2024
1	Cheride	Islam	5600	21/02/2024

On the other hand, the following query:

```
SELECT *
FROM achat
ORDER BY Name_Cl, Purchase_date DESC
```

Displays the purchase table as follows:

Id_Achat	Name_Cl	_Last_name_Cl	Purchase_amount	Purchase_date
4	Charif	Halim	3200	25/03/2024
3	Cheride	Islam	800	14/03/2024
1	Cheride	Islam	5600	21/02/2024
6	Kouissi	Ahlam	4800	09/04/2024
5	Kouissi	Ahlam	1000	01/04/2024
2	Kouissi	Ahlam	2100	01/03/2024

2.15. SQL TOP

The SQL TOP clause is used to limit the number of rows returned by a SELECT query. This is particularly useful when we only want to display a certain number of rows from the beginning of the results, such as the top 10 rows, or when a query could potentially return a large number of results and we want to display only part of them. The basic syntax of the TOP clause is as follows:

```
SELECT TOP number_of_rows [PERCENT] column1, column2, ...
FROM table
[WHERE condition]
[ORDER BY column1, column2, ...];
```

Example

```
SELECT TOP 10 *
FROM Client;
```

This query retrieves the first 10 clients from the Client table.

On the other hand:

```
SELECT TOP 50 PERCENT *
FROM Client;
```

Retrieves the top 50% of the clients in the Client table (half of the total rows).

Note: In many other DBMS, the `TOP` clause is replaced by `LIMIT` in the following way:

2.16. SUB Requests in SQL

A "SUB request" in SQL typically refers to a **subquery** - a query nested inside another SQL query. Subqueries are powerful tools that allow us to perform complex operations in a single statement. A sub request can have many formats like:

- **WHERE Clause Subqueries**

```
SELECT product_name, Unit_price
FROM product
WHERE Unit_price > (SELECT AVG (Unit_price) FROM product);
```

```
SELECT *
FROM client
WHERE CL_Id IN  (SELECT Client_ID FROM Orders);
```

- **SELECT Clause Subqueries (Scalar Subqueries)**

```
SELECT Cl_ID, NomC,
      (SELECT MAX(Order_Date)
       FROM Orders
       WHERE Orders.Client_ID = Client.Cl_ID) AS Latest_Order_Date
FROM Client;
```

- **FROM Clause Subqueries (Derived Tables)**

```
SELECT C.NomC, OrderCount.TotalOrders
FROM Client AS C
INNER JOIN (
   SELECT Client_ID, COUNT(*) AS TotalOrders
   FROM Orders
   GROUP BY Client_ID
) AS OrderCount
ON C.Cl_ID = OrderCount.Client_ID;
```

2.17. SQL INNER JOIN

In SQL, the INNER JOIN command, also known as EQUIJOIN, is a very common type of join that allows combining records from two tables using a matching condition, and returns only the rows with matches in both tables.
In other words, it returns only the rows where there is a match between the specified columns in both tables.

To use this type of join, the SQL query follows this syntax:

```
SELECT *
FROM table1
INNER JOIN table2 ON table1.id = table2.fk_id
```

Example
Given the two tables: City and Country

CountryID	CountryName
1	France
2	Spain
3	Germany

CityID	CityName	CountryID
1	Paris	1
2	Lyon	1
3	Madrid	2
4	Barcelona	2

The following query:

```
SELECT City.CityName, Country.CountryName
FROM City
INNER JOIN Country ON City.CountryID = Country.CountryID;
```

Returns the following result:

CityName	CountryName
Paris	France
Lyon	France
Madrid	Spain
Barcelona	Spain

Tip:
We can obtain the same result using a conditional Cartesian product like this:

```
SELECT *
FROM table1, table2
WHERE table1.id = table2.fk_id
```

2.18. SQL LEFT JOIN

In SQL, the LEFT JOIN command (also known as LEFT OUTER JOIN) is a type of join between two tables. It allows listing **all results from the left table**, even if there are no matching records in the second table.

This join uses the following syntax:

```
SELECT *
FROM table1
LEFT JOIN table2 ON table1.id = table2.fk_id
```

2.19. SQL RIGHT JOIN

In SQL, the RIGHT JOIN command (also known as RIGHT OUTER JOIN) is a type of join that returns **all records from the right table**, even if there are no matches in the left table. If a record in the right table has no matching record in the left table, the columns from the left table will return NULL.

This join uses the following syntax:

```
SELECT *
FROM table1
RIGHT JOIN table2 ON table1.id = table2.fk_id
```

2.20. Full Outer Join

We can simulate a FULL OUTER JOIN by combining a LEFT JOIN and a RIGHT JOIN, using UNION to merge the results.
In MS Access, this can be done like this:

```
SELECT *
FROM TableA LEFT JOIN TableB ON TableA.ID = TableB.ID

UNION

SELECT *
FROM TableB LEFT JOIN TableA ON TableB.ID = TableA.ID
```

2.21. Filtering on NULL Values

Be careful: NULL is not a string value. To filter on NULL values, you must use the IS NULL command.
Filtering on NULL or non-NULL values enables specific joins like:

- **LEFT-SEMI JOIN**

```
SELECT table1.*
FROM table1 LEFT JOIN table2 ON table1.id = table2ur_id
WHERE table2_id IS NOT NULL
```

- **RIGHT-SEMI JOIN**

```
SELECT table2.*
FROM table1 RIGHT JOIN table2 ON table1.id = table2ur_id
WHERE table1_id IS NOT NULL
```

- **ANTI JOIN**

```
SELECT table1. *
FROM table1 LEFT JOIN table2 ON table1.id = table2ur_id
WHERE table2_id IS NULL
```