————————————————————————————————

NULL Values

The SQL **NULL** is the term used to represent a missing value.
A NULL value in a table is a value in a field that appears to be blank.
A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Syntax:

The basic syntax of **NULL** while creating a table:

```
SQL> CREATE TABLE CUSTOMERS(
   ID   INT                NOT NULL,
   NAME VARCHAR (20)       NOT NULL,
   AGE  INT                NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.
A field with a NULL value is one that has been left blank during record creation.

Example:

The NULL value can cause problems when selecting data, however, because when comparing an unknown value to any other value, the result is always unknown and not included in the final results.
You must use the **IS NULL** or **IS NOT NULL** operators in order to check for a NULL value.

————————————————————————————————

GROUP BY / HAVING

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax:

The following is the position of the HAVING clause in a query:

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used. The following is the syntax of the SELECT statement, including the HAVING clause:

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

————————————————————————————————————

A **VIEW** is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query. A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

The WITH CHECK OPTION

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all

UPDATE and INSERTs satisfy the condition(s) in the view definition.
——————————————————————————————

The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.
——————————————————————————————

SQL Join Types:

There are different types of joins available in SQL:

- INNER JOIN: returns rows when there is a match in both tables.
- LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN: returns rows when there is a match in one of the tables.
- SELF JOIN: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined tables.

——————————————————————————————

The SQL **UNION** clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.
To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order, but they do not have to be the same length.

Syntax:

The basic syntax of **UNION** is as follows:

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

```
UNION

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here given condition could be any given expression based on your requirement.

———————————————————————————————

סיכום מתומצת של מה שכדאי לזכור

What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.
SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.
Also, they are using different dialects, such as:

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format), etc

Why SQL?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views

What is RDBMS?

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

What is table ?

The data in RDBMS is stored in database objects called **tables**. The table is a collection of related data entries and it consists of columns and rows.
Remember, a table is the most common and simplest form of data storage in a relational database.

What is field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.
A field is a column in a table that is designed to maintain specific information about every record in the table.

What is record or row?

A record, also called a row of data, is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table.
A record is a horizontal entity in a table.

What is column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

What is NULL value?

A NULL value in a table is a value in a field that appears to be blank which means A field with a NULL value is a field with no value.
It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

SQL Constraints:

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.
Constraints could be column level or table level. Column level constraints are applied only to one column where as table level constraints are applied to the whole table.

SQL Syntax:

SQL is followed by unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax:
All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).
Important point to be noted is that SQL is **case insensitive** which means SELECT and select have same meaning in SQL statements but MySQL make difference in table names. So if you are working with MySQL then you need to give table names as they exist in the database.

SQL SELECT Statement:

```
SELECT column1, column2....columnN
FROM   table_name;
```

SQL DISTINCT Clause:

```
SELECT DISTINCT column1, column2....columnN
FROM   table_name;
```

SQL WHERE Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION;
```

SQL AND/OR Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION-1 {AND|OR} CONDITION-2;
```

SQL IN Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name IN (val-1, val-2,...val-N);
```

SQL BETWEEN Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name BETWEEN val-1 AND val-2;
```

SQL Like Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name LIKE { PATTERN };
```

SQL ORDER BY Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION
ORDER BY column_name {ASC|DESC};
```

SQL GROUP BY Clause:

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name;
```

SQL COUNT Clause:

```
SELECT COUNT(column_name)
FROM   table_name
WHERE  CONDITION;
```

SQL HAVING Clause:

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (arithematic function condition);
```

SQL CREATE TABLE Statement:

```
CREATE TABLE table_name(
column1 datatype,
column2 datatype,
column3 datatype,
.....
columnN datatype,
PRIMARY KEY( one or more columns )
);
```

SQL DROP TABLE Statement:

```
DROP TABLE table_name;
```

SQL CREATE INDEX Statement :

```
CREATE UNIQUE INDEX index_name
ON table_name ( column1, column2,...columnN);
```

SQL DROP INDEX Statement :

```
ALTER TABLE table_name
DROP INDEX index_name;
```

SQL DESC Statement :

```
DESC table_name;
```

SQL TRUNCATE TABLE Statement:

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement:

```
ALTER TABLE table_name {ADD|DROP|MODIFY} column_name
{data_ype};
```

SQL ALTER TABLE Statement (Rename) :

```
ALTER TABLE table_name RENAME TO new_table_name;
```

SQL INSERT INTO Statement:

```
INSERT INTO table_name( column1, column2....columnN)
VALUES ( value1, value2....valueN);
```

SQL UPDATE Statement:

```
UPDATE table_name
SET column1 = value1, column2 = value2....columnN=valueN
[ WHERE  CONDITION ];
```

SQL DELETE Statement:

```
DELETE FROM table_name
WHERE  {CONDITION};
```

SQL CREATE DATABASE Statement:

```
CREATE DATABASE database_name;
```

SQL DROP DATABASE Statement:

```
DROP DATABASE database_name;
```

SQL USE Statement:

```
USE DATABASE database_name;
```

SQL COMMIT Statement:

```
COMMIT;
```

SQL ROLLBACK Statement:

```
ROLLBACK;
```

SQL - Operators:

SQL Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20, then:
Show Examples

Operator	**Description**	**Example**
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0

SQL Comparison Operators:

Assume variable a holds 10 and variable b holds 20, then:
Show Examples

Operator	**Description**	**Example**
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.

>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Logical Operators:

Here is a list of all the logical operators available in SQL.

Show Examples

Operator	**Description**
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

EXISTS
The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN
The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE
The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT
The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc.
This is a negate operator.
OR
The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL
The NULL operator is used to compare a value with a NULL value.
UNIQUE
The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL - Useful Functions:

SQL has many built-in functions for performing processing on string or numeric data. Following is the list of all useful SQL built-in functions:

- SQL COUNT Function - The SQL COUNT aggregate function is used to count the number of rows in a database table.
- SQL MAX Function - The SQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.
- SQL MIN Function - The SQL MIN aggregate function allows us to select the lowest (minimum) value for a certain column.
- SQL AVG Function - The SQL AVG aggregate function selects the average value for certain table column.
- SQL SUM Function - The SQL SUM aggregate function allows selecting the total for a numeric column.
- SQL SQRT Functions - This is used to generate a square root of a given number.

- SQL RAND Function - This is used to generate a random number using SQL command.
- SQL CONCAT Function - This is used to concatenate any string inside any SQL command.
- SQL Numeric Functions - Complete list of SQL functions required to manipulate numbers in SQL.
- SQL String Functions - Complete list of SQL functions required to manipulate strings in SQL.