Complete notes on

SQL

Written By:-
Dipti Gaydhane
(Software Engineer)

INDEX

SQL Tutorial

ocean group

PAGE:
DATE: / /

11) SQL Update Statement

12) SQL Delete Statement

13) First or Rownum clause

14) SQL Top percent example

15) Add the Order by Keyword

16) SQL min() and max() Functions

17) SQL Count Function

18) SQL Like Operator

19) SQL Joins

20) SQL Statements

21) SQL Comments

22) SQL Operators

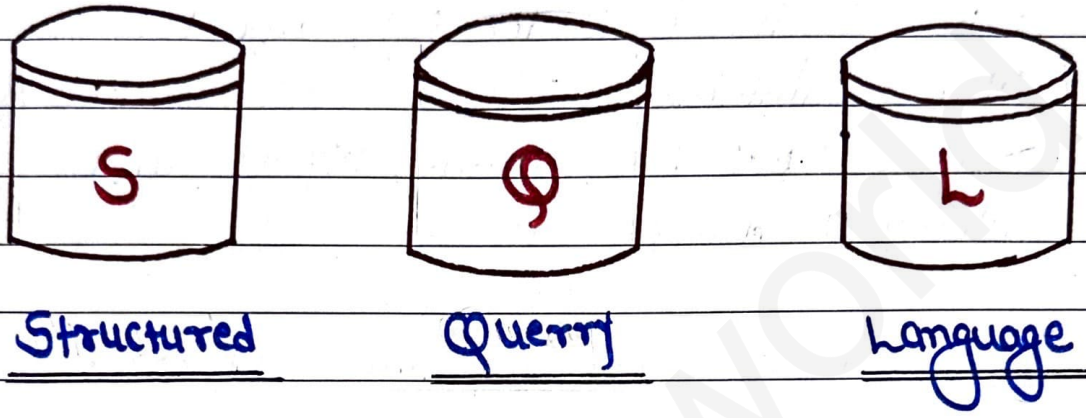
23) SQL Datatype

PAGE:
DATE: / /

24) SQL Constraints

25) SQL Injection

26) SQL Hosting



Introduction to SQL :

What is SQL ?

- SQL stands for Structured Querry Language
- SQL lets you access and manipulate database
- SQL became a Standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization For Standardization (ISO) in 1987

RDBMS

RDBMS Stands for Relational Database Management

System.

RDBMS is the basis for SQL, and for all Modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

Some Of The Most Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - delete data from a database
- INSERT INTO - insert new data into a database.
- CREATE DATABASE - creates a new database
- ALTER DATABASE - Modifies a database
- CREATE TABLE - Creates a new table
- ALTER TABLE - Modifies table
- DROP TABLE - deletes a table
- CREATE INDEX - Creates an index
- DROP INDEX - deletes an index

SQL SELECT Statement :

The SELECT Statement is used to select data from a database.

Example ↓

```
SELECT CustomerName, City FROM
Customers;
```

Select ALL Columns

Example ↓

```
SELECT * FROM Customers;
```

SQL SELECT DISTINCT Statement :

The SELECT DISTINCT statement is used to return only distinct (different) values.

Example ↓

```
SELECT DISTINCT Country FROM
Customers;
```

The SQL WHERE Clause

The WHERE clause is used to filter records.

PAGE:
DATE: / /

It is used to extract only those records that fulfill a specified condition.

Example ↓

```
SELECT * FROM customers
WHERE country = 'Mexico';
```

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note → The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc!

Operators in The WHERE clause

Example ↓

Select all customers with a CustomerID greater than 80:

```
SELECT * FROM customers
WHERE customerID > 80;
```

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal
Between	Between a certain range.
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

The SQL ORDER By :

The ORDER By keyword is used to sort the

result-set in ascending or descending order.

Example ↓

```
SELECT * FROM products
ORDER By price;
```

Syntax

```
SELECT column1, column2, ....
FROM table_name
ORDER By column1, column2, ----
ASC/DESC;
```

DESC

The ORDER By keyword sorts the records in ascending order by default. TO sort the records in descending order, use the DESC keyword.

Example ↓

```
SELECT * FROM products
ORDER By price DESC;
```

Order Alphabetically

For String values the ORDER By keyword will order alphabetically

Example↓

Sort the products alphabetically by productName:

```
SELECT * FROM products
ORDER By productName;
```

Using Both ASC and DESC

The following SQL statement selects all customers from the "Customers" table, sorted ascending by "Country" and descending by the "CustomerName" column:

Example↓

```
SELECT * FROM Customers
ORDER By Country ASC, CustomerName
DESC;
```

The SQL AND Operator :

The WHERE clause can contain one or many AND operators.

The AND operator is used to filter records based on More than one condition, like if you want to return all customers from Spain that starts with the letter 'G':

Example ↓

```
SELECT *
FROM customers
WHERE country = 'Spain' AND
customerName LIKE 'G%';
```

Syntax

```
SELECT column1, column2, ---
FROM table_name
WHERE condition1 AND condition2 AND
condition3 ---;
```

SQL OR Operator :

The WHERE clause can contain one or more OR operators.

The OR operator is used to filter records based on more than one condition, like if you want to return all customers from Germany but also those from Spain.

Example ↓

```
SELECT *
FROM Customers
WHERE Country = 'Germany' OR Country
 = 'Spain';
```

Syntax

```
SELECT column1, column2, ---
FROM table_name
WHERE condition1 OR condition2 OR
condition3 ---;
```

OR vs AND

The OR operator displays a record if any of the conditions are TRUE.

The AND operator displays a record if all the conditions are TRUE.

SQL NOT Operator :

The NOT operator

The NOT operator is used in combination with other operators to give the opposite result, also called the negative result.

In the Select statement below we want to return all customers that are NOT from Spain.

Example

```
SELECT * FROM Customers
WHERE NOT Country = 'Spain';
```

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

NOT BETWEEN

Example:

```
SELECT * FROM Customers
WHERE CustomerID NOT BETWEEN 10 AND
60;
```

NOT Greater Than

Example:

```
SELECT * FROM Customers
WHERE NOT CustomerID > 50;
```

NOT Less Than

Example →

```
SELECT * FROM Customers
WHERE NOT CustomerID < 50;
```

INSERT INTO Syntax :

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows.

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

SQL NULL Values :

What is a NULL value?

A field with NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

How to Test for NULL values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
```

```
FROM table_name
WHERE column_name IS NOT NULL;
```

SQL UPDATE Statement :

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2,
....
WHERE
```

Example ↓

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt',
City = 'Frankfurt'
WHERE CustomerID = 1 ;
```

SQL DELETE Statement :

The DELETE statement is used to delete existing records in a table.

Syntax

```
DELETE FROM table_name WHERE condition;
```

Example ↓

```
DELETE FROM customers WHERE
CustomerName = 'Alfreds Futterkiste';
```

Delete All Records

Example → `DELETE FROM customers;`

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and deleting the intact.

```
DELETE FROM table_name;
```

Delete a Table

To delete the table completely, use the DROP TABLE statement.

Example ↓

```
DROP TABLE customers;
```

FIRST or ROWNUM clause :

The SQL SELECT TOP clause :

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Example ↓

```
SELECT TOP 3 * FROM customers;
```

LIMIT

Example ↓

```
SELECT * FROM customers
LIMIT 3;
```

FETCH FIRST

Example ↓

```
SELECT * FROM customers
FETCH FIRST 3 ROWS ONLY;
```

SQL TOP PERCENT Example :

Example ↓

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

ADD a WHERE CLAUSE

Example →

```
SELECT TOP 3 * FROM customers
WHERE = 'Germany';
```

ADD the ORDER By Keyword :

Add the ORDER By keyword when you want to sort the result, and return the first 3 records of the sorted result:

Example 1

```
SELECT TOP 3 * FROM customers
ORDER By customerName DESC;
```

SQL Aggregate Functions

Aggregate functions are often used with the GROUP By clause of the SELECT statement.

The GROUP By clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are :

- MIN() - returns the smallest value within the

selected column.

- MAX() - Returns the largest value within the selected column.
- COUNT() - Returns the number of rows in a set.
- SUM() - Returns the total sum of a numerical column.
- AVG() - Returns the average value of a numerical column.

Aggregate functions ignore null values (except for COUNT()).

SQL MIN() and MAX() functions:

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN Example:

```
SELECT MIN(price)
FROM products;
```

MAX Example:

```
SELECT MAX(price)
FROM products;
```

SQL COUNT() Function :

The COUNT() function returns the number of rows that matches a specified criterion.

Example:

```
SELECT COUNT(*)
FROM products;
```

Syntax

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

Use COUNT() With GROUP By :

Here use the COUNT() function and the GROUP By clause, to return the number of records for each category in the products table:

Example ↓

```
SELECT COUNT (*) AS [Number of
records], CategoryID
FROM products
GROUP By CategoryID;
```

SQL SUM() Fimction :

The SUM() function returns the total sum of a numeric column.

Example ↓

```
SELECT SUM( Quantity)
FROM orderDetails ;
```

Syntax

```
SELECT SUM (column_name)
FROM table_name
WHERE condition;
```

The SQL AVG() Function :

The AVG() function returns the average value of a numeric column.

Example↓

```
SELECT AVG(price)
FROM products;
```

Syntax

```
SELECT AVG (column_name)
FROM table_name
WHERE condition;
```

SQL LIKE Operator :

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two Wildcards often used in conjuction with the LIKE Operator.

- The percent sign % represents zero, one, or Multiple characters.
- The underscore sign _ represents one, Single character

Example:

```
SELECT * FROM customers
WHERE customerName LIKE 'a%';
```

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

SQL Wildcard Characters

A Wildcard character is used to substitute One or More characters in a string

ocean group

PAGE:
DATE: / /

Wildcard characters

Symbol	Description
%	Represents zero or more characters
_	Represents a single character
[]	Represents any single character within the brackets *
^	Represents any character not in the brackets *
-	Represents any single character within the specified range *
{}	Represents any escaped character *

Using the % Wildcard

Example:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%es';
```

Using the _ Wildcard

Example:

Return all customers with a City starting with any character, followed by "ondon".

```
SELECT * FROM Customers
WHERE City LIKE '_ondon';
```

Using the [] Wildcard

Example:

Return all customers starting with either "b", "s", or "p".

```
SELECT * FROM Customers
WHERE CustomerName LIKE '[bsp]%';
```

Using the - Wildcard

Example↓

Return all customers starting with "a", "b", "c", "d", "e", or "f".

```
SELECT * FROM customers
WHERE customerName LIKE '[a-f]%';
```

Combine Wildcards

Example↓

Return all customers that starts with "a" and are at least 3 characters in length.

```
SELECT * FROM customers
WHERE customerName LIKE 'a--%';
```

SQL IN Operator :

The IN operator allows you to specify Multiple values in a WHERE clause.

Example:

Return all customers from 'Germany', 'France', or 'UK'.

```
SELECT * FROM customers
WHERE country IN ('Germany', 'France',
'UK');
```

SQL BETWEEN Operator:

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

Example:

Selects all products with a price between 10 and 20

```
SELECT * FROM products
WHERE price BETWEEN 10 AND 20;
```

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND
value2;
```

SQL Aliases

SQL alises are used to give a table, or a column in table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists or the duration of that query.

An alias is created with the AS keyword.

Example ↓

```
SELECT CustomerID AS ID
FROM Customers;
```

Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

SQL Joins :

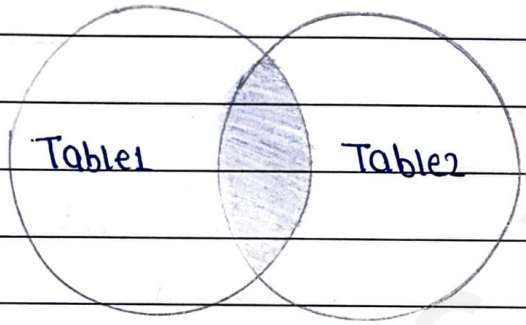
A JOIN clause is used to combine rows From two or More tables, based on a related column between them.

Different Types of SQL JOINS

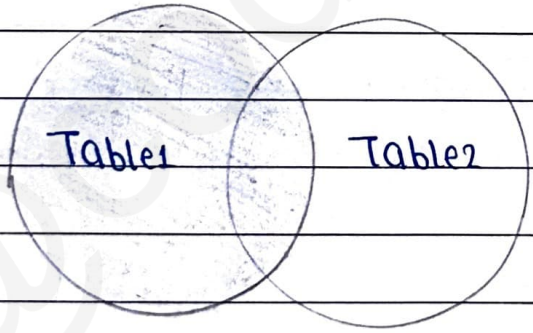
- (INNER) JOIN - Returns records that have Matching values in both tables.
- LEFT (OUTER) JOIN - Returns all records from the left table, and the Matched records From the right table.
- RIGHT (OUTER) JOIN - Returns all record from the right table, and the Matched records from the left table.
- FULL (OUTER) JOIN - Returns all records when

thre is a Match in either left or right table.

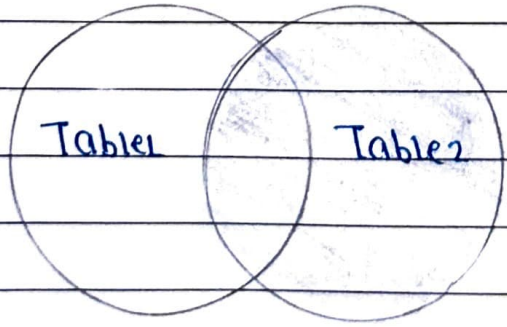
INNER JOIN



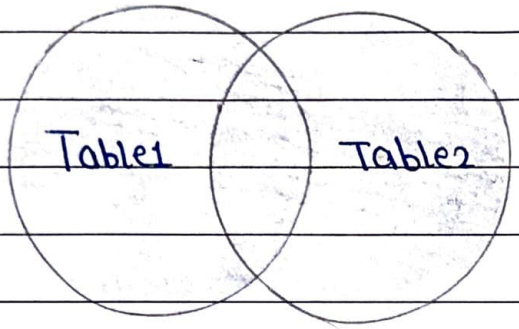
LEFT JOIN



RIGHT JOIN



FULL OUTER JOIN



SQL INNER JOIN :

The INNER JOIN keyword selects record that have matching values in both tables.

Examples ↓

Join products and categories with the INNER JOIN keyword.

```
SELECT productID, productName,
categoryName
FROM products
INNER JOIN categories ON
products.categoryID =
categories.categoryID;
```

SQL LEFT JOIN KEYWORD :

The LEFT JOIN keyword returns all records from the left table (table1) and the matching records from the right (table2). The result is 0 records from the right side, if there is no match.

Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name =
table2.column_name;
```

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name =
table2.column_name;
```

SQL FULL OUTER JOIN Keyword :

The FULL OUTER JOIN Keyword returns all records when there is a Match in left (table1) or right (table2) table records.

Tip: FULL OUTER JOIN and FULL JOIN are the same.

Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name =
table2.column_name
WHERE condition;
```

SQL Self Join

A self join is a regular Join, but the table is Joined with itself.

Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

SQL UNION OPERATOR :

The UNION operator is used to combine the result-set of two or More SELECT Statements.

- Every SELECT statement within UNION must have the same number of columns.
- The columns Must also have similar data types.
- The columns in every SELECT statement must also be in the same order.

Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION ALL Syntax

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

The SQL GROUP By Statement :

The GROUP By Statement groups rows that have the same values into Summary rows, like "Find the number of customers in each country".

Syntax →

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP By column_name(s)
ORDER By column_name(s);
```

SQL HAVING Clause :

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records

Syntax →

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name
WHERE condition);
```

The SQL ANY Operator

The ANY operator

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
    (SELECT column_name
FROM table_name
WHERE condition);
```

SQL SELECT INTO Statement :

The SELECT INTO Statement copies data FROM one table into a new table.

Copy all columns into a new table

Syntax

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

The SQL INSERT INTO SELECT Statement :

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in source and target tables match.

Syntax

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

The SQL CASE Expression

The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

Example ↓

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN
```

```
'The quantity is greater than 30'
    WHEN Quantity = 30 THEN
'The quantity is 30'
    Else 'The quantity is under 30'
AND AS QuantityText
FROM orderDetails;
```

SQL Comments :

Single Line Comments

Single line comments starts with --

Any text between -- and the end of the line will be ignored (will not be executed)

Example ↓

```
-- Select all:
SELECT * FROM Customers;
```

Multi-line Comments

Multi-line comments start with /* and end with */ .
Any text between /* and */ will be ignored.

Example ↓

```
/* Select all the columns
of all the records
in the customers table : */
SELECT * FROM customers ;
```

SQL Operators :

Arithmetic Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

Comparison Operators

Operator	Description
=	Equal to
>	GReater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL Compound Operators

Operator	Description
+=	Add equals
-=	subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

SQL Logical Operators

Operator	Description
ALL	TRUE if all the subquery values

	Meet the Condition
AND	TRUE if all the conditions seperated by AND is TRUE
ANY	TRUE if any of the subquerry values meet the condition
BETWEEN	True if the operand is within the range of comparisons
EXISIS	TRUE if the Subquery returns one ore More records
IN	TRUE if the operand is equal to one of the list of expressions.

LIKE	: TRUE if the operand Matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions Seperated by OR is TRUE
SOME	TRUE if any of the Subquery values meet the condition

SQL CREATE DATABASE Statement :

The CREATE DATABASE Statement is used to Create a new SQL database

Syntax

```
CREATE DATABASE databasename;
```

Example → CREATE DATABASE testDB;

SQL DROP DATABASE Statement :

The DROP DATABASE statement is used to drop an existing database.

Syntax

```
DROP DATABASE databasename;
```

Example ↓

```
DROP DATABASE testDB;
```

SQL BACKUP DATABASE for SQL Server :

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax

```
BACKUP DATABASE databasename
TO DISK = 'filepath';
```

SQL CREATE TABLE Statement :

The CREATE TABLE Statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name
(column1 datatype,
 column2 datatype,
 column3 datatype,
 ....
);
```

Example :

```
CREATE TABLE persons
(personID int,
 LastName varchar (255),
 FirstName varchar (255),
 Address varchar (255),
 city varchar (255),

);
```

SQL DROP TABLE Statement :

The DROP TABLE Statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

Example↓

```
DROP TABLE shippers;
```

SQL ALTER TABLE Statement :

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE Statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax.

Syntax

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example ↓

```
ALTER TABLE Customers
ADD Email varchar(225);
```

ALTER TABLE - DROP Column

To delete a column in a table, use the following syntax

Syntax

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example ↓

```
ALTER TABLE Customers
DROP COLUMN Email;
```

ALTER TABLE - RENAME Column

Syntax

```
ALTER TABLE table_name
RENAME COLUMN old_name to new_name;
```

ALTER TABLE - ALTER / MODIFY DATATYPE :

Syntax

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

SQL Constraints :

SQL constraints are used to specify rules for the data in a table.

Constraints 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 table. If there is any violation between the constraint and the data action, the action is aborted.

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value.
- UNIQUE - Ensures that all values in a column are different.
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
- FOREIGN KEY - prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies specific condition.

- DEFAULT - Set a default value for a column if no value is specified.

- CREATE INDEX - Used to create and retrieve data from the database very quickly.

SQL NOT NULL on CREATE TABLE :

Example ↓

```
CREATE TABLE persons (
    ID int NOT NULL,
    LastName varchar (255) NOT NULL,
    FirstName varchar (255) NOT NULL,
    Age int
);
```

SQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY

Constraints provide a guarantee For uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar (255) NOT NULL,
    Firstname varchar (255),
    Age int
);
```

SQL PRIMARY KEY Constraint :

The PRIMARY KEY constraint uniquely identifies each record in a table.

primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

Example↓

```
PRIMARY KEY (ID);
```

SQL PRIMARY KEY ON ALTER TABLE

```
ALTER TABLE persons
ADD PRIMARY KEY (ID);
```

SQL FOREIGN KEY Constraint :

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Example ↓

```
FOREING KEY (personID)
REFERENCES persons (personID)
```

SQL CHECK Constraint :

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

Example ↓

```
Age int,
CHECK (Age >= 18)
```

SQL DEFAULT Constraint :

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new

records, if no other value is specified.

Example:

```
City varchar (255) DEFAULT
'Sandnes'
```

SQL CREATE INDEX Statement :

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Syntax

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

SQL AUTO INCREMENT Field :

Auto-increment allows a unique number to be generated automatically when a new record

is inserted into a table.

Syntax

```
personid int NOT NULL
AUTO_INCREMENT,
```

SQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database.

- DATE - format yyyy-MM-DD
- DATETIME - format: yyyy-MM-DD HH:MI:SS
- TIMESTAMP - format: yyyy-MM-DD HH:MI:SS
- YEAR - format yyyy or yy

SQL Server comes with the following data types for storing a date or a date/time value in the database.

- DATE - format yyyy-MM-DD
- DATETIME - format : yyyy-MM-DD HH:MI:SS
- SMALLDATETIME - format : yyyy-MM-DD HH:MI:SS
- TIMESTAMP - format : a unique number

Statement ↴

```
SELECT * FROM Orders WHERE
OrderDate ='2008-11-11'
```

SQL CREATE VIEW Statement :

In SQL, a view is virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL Statements and Functions to a View and present the data as if the data were coming from one single table.

A View is created with the CREATE VIEW statement.

Syntax

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

Example ↓

The following SQL creates a view that shows all customers from Brazil.

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

We can query the view above as follows.

Example ↓

```
SELECT * FROM [Brazil Customers];
```

SQL Iniection :

SQL iniection is a code injection technique that might destroy your database.

SQL iniection is one of the most common Web hacking techniques.

SQL injection is the placement of Malicious code in SQL statements, via Web Page input.

Example ↓

```
txtUserId =
getRequestString ("UserId");
txtSQL = "SELECT * FROM Users WHERE
UserId = " + txtUserId;
```

SQL Hosting :

If you Want your web site to be able to store and retrieve data from a database, your web server should have access to a database System that uses the SQL Language.

If your web server is hosted by an Internet Service provider (ISP), you will have to look for SQL hosting plans.

The most common SQL hosting databases are MS SQL Server, Oracle, MySQL, and MS Access.