<https://www.tutorialspoint.com/sql/sql-drop-table.html>

**Types of SQL Databases:**

Relational databases are used to store and manage the data objects that are related to one another (structured way) (using tables). A system used to manage these relational databases is known as Relational Database Management System (RDBMS).

There are many popular RDBMS available:

MySQL MS SQL Server ORACLE MS ACCESS PostgreSQL SQLite

CREATE DATABASE / SCHEMA database\_name;

CREATE DATABASE sampled >> USE database\_name; USE sampleDB;

DROP DATABASE database\_name; DROP DATABASE sampleDB;

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) );

**for** description table or table info >> DESC CUSTOMERS;

To insert data

INSERT INTO CUSTOMERS VALUES

(1, 'Ramesh', 32, 'Ahmedabad', 2000.00 ),

(2, 'Khilan', 25, 'Delhi', 1500), (3, 'kaushik', 23, 'Kota', 2000),

(4, 'Chaitali',25, 'Mumbai', 6500),(5,'Hardik',27, 'Bhopal', 8500),

(6, 'Komal',22, 'Hyderabad', 4500),(7,'Muffy', 24, 'Indore', 10000);

Update data in table:/ UPDATE VIEWS:

A view is a database object that can contain rows (all or selected) from an existing table. It can be created from one or many tables which depends on the provided SQL query to create a view.

UPDATE CUSTOMERS SET ADDRESS = 'Pune' WHERE ID = 6;

UPDATE CUSTOMERS SET AGE=AGE+5, SALARY=SALARY+3000;

This query affected age column of all rows

MEANS PREVIOUS AGE IS 22 THEN ADD 5 MEANS NEW AGE IS 27.

UPDATE CUSTOMERS\_VIEW SET NAME=’ABC’, AGE=25 WHERE ID=3;

UPDATE CUSTOMERS\_VIEW SET AGE=25+6;

IN operator with update

Update customers set age=30 in (25,27);

CASE STATEMENT WITH UPDATE:

UPDATE CUSROMERS SET SALARY= CASE AGE WHEN 25 THEN 2500

WHEN 32 THEN 5500 ELSE 2000 END;

CASE STATEMENT WITH INSERT:

INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY) VALUES(10, ‘VIREN’, ’NASHIK’,

CASE WHEN AGE>=25 THEN 25000 ELSE 14000 END);

Change table name column name with datatype

ALTER TABLE table\_name

{ADD|DROP|MODIFY} column\_name {data\_type};

All records delete but table structure as it is

TRUNCATE TABLE CUSTOMERS;

Changes table column\_name, modify and drop

ALTER TABLE CUSTOMERS

ADD|DROP|MODIFY

Add city varchar(10);

change table name:

ALTER TABLE table\_name RENAME TO new\_table\_name;

SQL types of operators:

Arithmetic operators Comparison operators

Logical operators Operators used to negate conditions

Arithmetic operators

SQL Arithmetic Operators are used to perform mathematical operations on the numerical values.

+,-,/,\*, and % modules 5%2=1

ALL:- TRUE if all of a set of comparisons are TRUE.

AND:- TRUE if all the conditions separated by AND are TRUE.

ANY: TRUE if all of a set of comparisons are TRUE.

BETWEEN: TRUE if the operand lies within the range of comparisons.

EXISTS: TRUE if the subquery returns one or more records

IN: TRUE if the operand is equal to one of a list of expressions.

LIKE: TRUE if the operand matches a pattern specially with wildcard.

NOT: Reverses the value of any other Boolean operator.

OR: TRUE if any of the conditions separated by OR is TRUE

IS NULL: TRUE if the expression value is NULL.

SOME: TRUE if some of a set of comparisons are TRUE.

UNIQUE: The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

What is SQL Expression?

An SQL expression is a combination of one or more values, operators and SQL functions that are all evaluated to a value. These SQL EXPRESSION are like formulae and they are written in query language.

Expressions are used in WHERE clause of an SQL query.

Boolean, Numeric & Date and time Expressions.

Boolean Expressions that check for equality of two values using SQL comparison operators.

select count(\*) from customers;

select count(\*)+1 as total\_members from customers;

select sum(id) as total from customers;

SELECT CURRENT\_TIMESTAMP;

In sql can we connect one schema to another schema

Yes, in SQL, it's possible to communicate or reference objects (tables, views, procedures, etc.) from one schema to another schema within the same database or even across databases, depending on the database management system (DBMS) we're using.

Here are some common ways to reference objects between schemas:

Within the Same Database:

Using Fully Qualified Names:

Objects can be referenced using their fully qualified names, including the schema name.

For example: sql

SELECT \* FROM schema\_name.table\_name;

Changing the Default Schema:

Some databases allow changing the **default** schema **for** a user's session. This way, when an object name is used without a schema qualifier, the database system uses the default schema.

Syntax may vary across different database systems. For instance:

PostgreSQL: SET search\_path TO schema\_name;

SQL Server: ALTER USER username SET DEFAULT\_SCHEMA = schema\_name;

Across Different Databases:

Using Linked Servers (in some DBMS like SQL Server):

Linked Servers allow querying and modifying data from another database or instance within the same query.

Example in SQL Server:

SELECT \* FROM linked\_server\_name.schema\_name.table\_name;

Using Database Links (in Oracle):

Oracle Database Links enable communication between two databases.

Example in Oracle:

SELECT \* FROM schema\_name.table\_name@database\_link\_name;

When referencing objects between schemas or databases, ensure proper permissions and access rights are granted to the user executing these operations. The syntax and methods can vary between different database systems, so refer to the specific documentation **for** your database management system **for** detailed instructions and specific functionalities.

RENAME DATABASE OldDatabaseName TO NewDatabaseName;

ALTER TABLE BUYERS RENAME TO CUSTOMERS;

TRUNCATE a table completely in one go instead of deleting table records one by one And table structure as it is.

The SQL **TRUNCATE TABLE** command is used to empty a table.

TRUNCATE TABLE table\_name;

DROP TABLE command to delete a table but it will remove the complete table structure from the database

|  |  |
| --- | --- |
| **DELETE** | **TRUNCATE** |
| The DELETE command in SQL removes one or more rows from a table based on the conditions specified in a WHERE Clause. | SQL's TRUNCATE command is used to remove all of the rows from a table, regardless of whether or not any conditions are met. |
| It is a DML(Data Manipulation Language) command. | It is a DDL(Data Definition Language) command. |
| There is a need to make a manual COMMIT after making changes to the DELETE command, for the modifications to be committed. | When you use the TRUNCATE command, the modifications made to the table are committed automatically. |
| All rows are locked after deletion. | TRUNCATE utilizes a table lock, which locks the pages so they cannot be deleted. |
| When it comes to large databases, it is much slower. | It is much faster. |

SQL **Cloning Operation** allows to create the exact copy of an existing table along with its definition. There are three types of cloning

Simple Cloning Shallow Cloning Deep Cloning

Simple cloning operation creates a new replica table from the existing table and copies all the records in newly created table.

CREATE TABLE new\_table SELECT \* FROM original\_table;

Shallow cloning operation creates a new replica table from the existing table but does not copy any data records into newly created table, so only new but empty table is created.

CREATE TABLE NEW\_TABLE LIKE ORIGINAL\_TABLE;

Deep cloning operation is a combination of simple cloning and shallow cloning. It not only copies the structure of the existing table but also its data into the newly created table.

CREATE TABLE NEW\_TABLE LIKE ORIGINAL\_TABLE;

INSERT INTO NEW\_TABLE SELECT \* FROM ORIGINAL\_TABLE;

Temporary tables are pretty much what their name describes: they are the tables which are created in a database to store temporary data. We can perform SQL operations similar to the operations on permanent tables like CREATE, UPDATE, DELETE, INSERT, JOIN, etc. But these tables will be automatically deleted once the current client session is terminated.

CREATE TEMPORARY TABLE CUSTOMERS (

ID INT NOT NULL, NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL, PRIMARY KEY(ID) );

Though all the temporary tables are deleted by MySQL when your database connection gets terminated

DROP TEMPORARY TABLE CUSTOMERS;

The SQL **ALTER TABLE** command is a part of Data Definition Language (DDL) and modifies the structure of a table. The ALTER TABLE command can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself.

add a new column to a table:

ALTER TABLE TABLE\_NAME ADD COLUMN\_NAME datatypes;

Drop column:

ALTER TABLE TABLE\_NAME DROP COLUMN COLUMN\_NAME;

ADD PRIMARY KEY:

ALTER TABLE TABLE\_NAME

ADD CONSTRAINT constraint\_name

PRIMARY KEY (COLUMN1, COLUMN2…);

DROP PRIMARY KEY:

ALTER TABLE TABLE\_NAME DROP PRIMARY\_KEY;

ADD CONSTRAINTS KEY:

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name

UNIQUE (COLUMN1, COLUMN2…);

DROP CONSTRAINTS KEY:

ALTER TABLE table\_name DROP CONSTRAINT constraint\_name;

RENAME COLUMN NAME:

ALTER TABLE table\_name RENAME COLUMN column\_name TO new\_column\_name;

MODIFY DATATYPES

ALTER TABLE table\_name MODIFY COLUMN column\_name DATATYPE;

The SQL **DROP TABLE** statement is a Data Definition Language (DDL) command that is used to remove a table's definition, and its data, indexes, triggers, constraints and permission specifications (if any).

once a table is DROP then all the information available in that table will also be lost forever.

To drop a table in a database, one must require ALTER permission on the said table and CONTROL permissions on the table schema.

Even though it is a data definition language command, it is different from TRUNCATE TABLE statement as the DROP statement completely frees the table from the memory.

DROP TABLE causes an implicit commit, except when used with the TEMPORARY keyword.

IF EXISTS CLAUSE

DROP TABLE IF EXISTS TABLE\_NAME;

The **SQL DELETE** is a command of Data Manipulation Language (DML), so it does not delete or modify the table structure but it delete only the data contained within the table.

The **SQL DELETE TABLE** statement is used to delete the existing records from a table in a database.

DELETE FROM table\_name WHERE condition;

We can use the **SQL DELETE TABLE** statement without a WHERE clause to delete all records in a table in SQL. This statement will remove all the rows from the specified table, effectively resetting the table to its original state

DELETE FROM table\_name;

SQL Constraints are the rules applied to a data columns or the complete table to limit the type of data that can go into a table. When you try to perform any INSERT, UPDATE, or DELETE operation on the table, RDBMS will check whether that data violates any existing constraints and if there is any violation between the defined constraint and the data action, it aborts the operation and returns an error.

SQL CREATE Constraints means create a table.

Not null Constraints

CREATE TABLE table\_name (Column1 datatype constraint (NOT NULL),

Column2 datatype constraint, Column3 datatype constraint,

Column4 datatype constraint, PRIMARY KEY (Column1) );

Unique key constraints define unique key

Column1 datatype constraint UNIQUE KEY,

Default value constraint key

Column2 datatype DEFAULT ‘NOT AVAILABLE’,

PRIMARY Key constraint

When applied to a column, PRIMARY Key constraint ensure that a column accepts only UNIQUE value and there can be a single PRIMARY Key on a table but multiple columns can constitute a PRIMARY Key.

FOREIGN Key constraint maps with a column in another table and uniquely identifies a row/record in that table.

CREATE TABLE ORDERS ( ID INT NOT NULL, DATE DATETIME,

CUSTOMER\_ID INT FOREIGN KEUY REFERENCES CUSTOMER(ID),

AMOUNT DECIMAL, PRIMARY KEY(ID) );

When applied to a column, CHECK Value constraint works like a validation and it is used to check the validity of the data entered into the particular column of the table. table and uniquely identifies a row/record in that table.

CREATE TABLE CUSTOMERS (ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,AGE INT NOT NULL CHECK(AGE>=18),

ADDRESS CHAR (25), SALARY DECIMAL (18,2), PRIMARY KEY(ID) );

INDEX constraints:

The INDEX constraints are created to speed up the data retrieval from the database. An Index can be created by using a single or group of columns in a table. A table can have a single PRIMARY Key but can have multiple INDEXES. An Index can be Unique or Non-Unique based on requirements.

CREATE INDEX idx AGE ON CUSTOMERS (AGE);///?

DROPPING SQL CONSTRAINTS

ALTER TABLE CUSTOMERS DROP CONSTRAINT PRIMARY KEY;

The SQL **SELECT** Statement is used to fetch the data from a database table which returns this data in the form of a table.

If we want to fetch all / retrieve all fields then query of SELECT statement with an **Asterisk (\*)** instead of the column names.

DELETE MULTIPLE ROWS

DELETE FROM CUSTOMERS WHERE AGE>25;

Delete a rows based on conditions

DELETE FROM CUSTOMERS;

If we are using this query then delete all records from existing table (truncate it)

DELETE CUSTOMERS, ORDERS FROM CUSTOMERS C

INNER JOIN ORDERS O ON O.CUSTOMER\_ID =C.ID

WHERE C.SALARY >2000;

Following SQL query deletes the records of the customers (from the tables CUSTOMERS and ORDERS) who earn more than 2000 and have placed orders

SORTING RESULTS:

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. By default, some databases sort the query results in an ascending order.

In addition to that, ORDER BY clause can also sort the data in a database table in a preferred order. This case may not sort the records of a table in any standard order (like alphabetical or lexicographical), but, they could be sorted based on any external condition. For instance, in an ORDERS table containing the list of orders made by various customers of an organization, the details of orders placed can be sorted based on the dates on which those orders are made. This need not be alphabetically sorted, instead, it is based on "first come first serve".

SELECT \* FROM CUSTOMERS ORDER BY NAME;

BY DECENDING ORDER

SELECT \* FROM CUSTOMERS ORDER BY NAME DESC;

To fetch the rows with their own preferred order:

SELECT \* FROM CUSTOMERS ORDER BY (CASE ADDRESS

WHEN ‘DELHI’ THEN 1

WHEN ‘BHOPAL’ THEN 2

WHEN ‘KOTA’ THEN 3

WHEN ‘NASHIK’ THEN 4

WHEN ‘MUMBAI’ THEN 5

ELSE 100 END) ASC,ADDRESS DESC;

SQL VIEWS:

CREARE VIEWS:

To create a view in a database, you can use the SQL CREATE

CREATE VIEW CUSTOMERS\_VIEW AS SELECT \* FROM CUSTOMERS;

THEN CHECK SELECT \* FROM CUSTOMERS\_VIEW;

ALL FIELED TAKE IN CUSTOMERS\_VIEW TABLE FROM CUSTOMERS

IF WE WANT SPECIFIC FILED IN NEW TABLES THEN

CREATE VIEW SPECIFIC\_VIEW AS SELECT NAME, AGE FROM CUSTOMERS WHERE AGE>=25 WITH CHECK OPTION;

THEN CHECK SELECT \* FROM SPECIFIC\_VIEW; ITS SHOW ONLY NAME & AGE

DROP VIEWS:

SQL allows you to drop an exiting view and delete records from a view in a database. SQL uses DROP statement to delete all the records from the view along with its definition and using the DELETE statement, only the records are deleted while the view definition of the view remains unchanged.

DROP VIEW CUSOMERS\_VIEW;

DROP TABLE\_NAME;

DROP COLUMN:

ALTER TABLE table\_name

DROP COLUMN column\_name;

RENAME VIEWS:

TO CHANGE TABLE NAME

RENAME TABLE OLD\_NAME TO NEW\_NAME;

TO CHANGE COLUMN NAME

ALTER TABLE table\_name RENAME COLUMN old\_column\_name TO new\_column\_name;

SQL OPERATION AND CLAUSES:

WHERE CLAUSE:

The SQL **WHERE** clause is used to filter the results obtained by the DML statements such as SELECT, UPDATE and DELETE etc. We can retrieve the data from a single table or multiple tables(after join operation) using the WHERE clause.

We can specify a condition using the [comparison or logical operators](https://www.tutorialspoint.com/sql/sql-operators.htm) such as, >, <, =, **LIKE, NOT**, etc.

WHERE CLAUSE WITH IN OPERATOR:

SELECT \* from CUSTOMERS

WHERE NAME IN ('Khilan', 'Hardik', 'Muffy');

WHERE CLAUSE WITH NOT IN OPERATOR:

SELECT \* FROM CUSTOMERS WHERE AGE NOT IN(25,23,22);

WHERE WE CAN USED WITH LIKE, AND, OR OPERATORS.

TOP CLAUSE;

SELECT TOP 4 FROM CUSTOMERS; // THIS IS SQL SERVER QUERY

SELECT TOP 4 FROM CUSTOMERS ORDER BY SALARY DESC;

SELECT TOP 40 PERCENT FROM CUSTOMERS ORDER BY SALARY DESC;

SELECT TOP 4 \* FROM CUSTOMERS WHERE SALARY >4000;

DISTINCT CLAUSE:

The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to fetch unique records from a table.

SELECT DISTINCT ID, NAME FROM CUSTOMERS;

SHOW ONLY ID AND NAME

SELECT SALARY FROM CUSTOMERS ORDER BY SALARY;

SHOW ONLY SALARY IN ASCENDING ORDER.

SELECT COUNT (DISTINCT AGE) as UniqueAge FROM CUSTOMERS;

SHOW unique age result to count age and remove duplicate.

ORDER BY CLAUSE:

The SQL **ORDER BY** clause is used to sort the data in either ascending or descending order, based on one or more columns. This clause can sort data by a single column or by multiple columns. Sorting by multiple columns can be helpful when you need to sort data hierarchically, such as sorting by state, city, and then by the person's name.

ORDER BY is used with the SQL SELECT statement and is usually specified after the WHERE, HAVING, and GROUP BY clauses.

ORDER BY CLAUSE WITH ASC/DESC:

SELECT \* FROM CUSTOMERS ORDER BY NAME ASC/DESC

ORDER BY CLAUSE WITH MULTIPLE COLUMNS:

SELECT \* FROM CUSTOMERS ORDER BY AGE ASC, SALARY DESC;

IN THIS CASE 1ST ORDER BY AGE IF AGE IS DUPLICATE/SAME THEN DESCENDING ORDER BASED ON SALARY.

ORDER BY WITH WHERE CLAUSE:

SELECT \* FROM CUSTOMERS WHERE AGE=25 ORDER BY NAME DESC;

ORDER BY LIMIT CLAUSE:

SELECT SALARY FROM CUSTOMERS ORDER BY NAME LIMIT 4;

ITS SHOWING SALARY COLUMN 4 ROWS.

GROUP BY CLAUSE:

The SQL **GROUP BY** clause is used in conjunction with the SELECT statement to arrange identical data into groups. This clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY and HAVING clauses (if they exist).

The main purpose of grouping the records of a table based on particular columns is to perform calculations on these groups. Therefore, The GROUP BY clause is typically used with aggregate functions such as SUM(), COUNT(), AVG(), MAX(), or MIN() etc.

GROUP BY CLAUSE WITH AGGREGATE FUNCTIONS

The SQL **GROUP BY** clause is used in conjunction with the SELECT statement to arrange identical data into groups. This clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY and HAVING clauses (if they exist).

The main purpose of grouping the records of a table based on particular columns is to perform calculations on these groups. Therefore, The GROUP BY clause is typically used with aggregate functions such as SUM(), COUNT(), AVG(), MAX(), or MIN() etc.

SELECT AGE, COUNT(NAME) FROM CUSTOMERS GROUP BY AGE;

WE CAN COUNT(NAME) NAME AS \* OR ‘ANY\_NAME’

SELECT ADDRESS ,AGE, COUNT('TOTAL') FROM CUSTOMERS GROUP BY AGE;

SELECT ADDRESS,AGE, COUNT(\*) AS TOTAL FROM CUSTOMERS GROUP BY AGE;

GROUP BY WITH ORDER BY:

SELECT AGE, MIN(SALARY) AS MIN\_SALARY FROM CUSTOMERS GROUP BY AGE ORDER BY MIN\_SALARY DESC;

GROUP BY WITH HAVING CLAUSE:

SELECT ADDRESS, AGE,MIN(SALARY) AS MIN\_SUM FROM CUSTOMERS;

HAVING CLAUSE:

The SQL **HAVING clause** is similar to the WHERE clause; both are used to filter rows in a table based on specified criteria. However, the HAVING clause is used to filter grouped rows instead of single rows. These rows are grouped together by the GROUP BY clause, so, the HAVING clause must always be followed by the GROUP BY clause.

Moreover, the HAVING clause can be used with aggregate functions such as COUNT(), SUM(), AVG(), etc., whereas the WHERE clause cannot be used with them.

SELECT ADDRESS, AGE, MIN(SALARY) AS MIN\_SUM FROM CUSTOMERS GROUP BY ADDRESS, AGE HAVING AGE > 25;

HAVING WITH ORDER BY CLAUSE:

SELECT ADDRESS, AGE, SUM(SALARY) AS TOTAL\_SALARY FROM CUSTOMERS GROUP BY ADDRESS, AGE HAVING TOTAL\_SALARY >=5000 ORDER BY TOTAL\_SALARY DESC;

SELECT AGE, COUNT(AGE) FROM CUSTOMERS GROUP BY AGE HAVING COUNT (age) >= 1;

This query give the result showing age & count age from customers table those similar age is greater than or equal to one.

AND & OR:

And operator condition is satisfied then return true otherwise false, we can also used multiple and operator. We can used for update statement.

Or operator at least one condition true.

We can also used or operator with delete statement

BOOLEAN (BIT) OPERATOR:

A **Boolean** is a universal data type which stores true or false values.

When we are create a table that’s time need to add variable(isAvailable) with Boolean datatype in sql server datatypes as BIT (1,0)

LIKE OPERATOR:

A **Boolean** is a universal data type which stores true or false values.

It is used along with the WHERE clause of the UPDATE, DELETE and SELECT statements, to filter the rows based on the given pattern. These patterns are specified using **Wildcards**.

**%** The percent sign represents zero, one or multiple characters.

**\_** The underscore represents a single number or character.

**WHERE SALARY LIKE '200%'**

Finds any values that start with 200.

**WHERE SALARY LIKE '%200%'**

Finds any values that have 200 in any position.

**WHERE SALARY LIKE '\_00%'**

Finds any values that have 00 in the second and third positions.

**WHERE SALARY LIKE '2\_%\_%'**

Finds any values that start with 2 and are at least 3 characters in length.

**WHERE SALARY LIKE '%2'**

Finds any values that end with 2.

**WHERE SALARY LIKE '\_2%3'**

Finds any values that have a 2 in the second position and end with a 3.

**WHERE SALARY LIKE '2\_\_\_3'**

Finds any values in a five-digit number that start with 2 and end with 3.

IN OPERATOR:

The SQL **IN Operator** is used to specify multiple values or sub query in the **WHERE** clause. It returns all rows in which the specified column matches one of the values in the list. The list of values or sub query must be specified in the parenthesis e.g. IN **(select query)** or IN **(Value1, Value2, Value3, ...)**.

In some scenarios we may use multiple OR statements to include multiple conditions in SELECT, DELETE, UPDATE, or INSERT statements. Alternatively, we can use the IN operator instead of multiples OR statements.

Select \* from customers where age not in (25,27);

Select \* from customers where 2000 in (salary);

Subquery with in operator:

We can use the subquery with the IN operator that is used to return records from the single column. This means that more than one column in the SELECT column list cannot be included in the subquery.

**Subquery** − This is the SELECT statement that has a result set to be tested against the expression. The IN condition evaluates to true if any of these values match the expression.

Select \* from customers where name in (select name from customers where salary > 2000);

In operator with not

ANY, ALL OPERATOR:

The **SQL ANY and ALL** operators are used to perform a comparison between a single value and a range of values returned by the subquery.

*The ANY and ALL operators must be preceded by a standard comparison operator i.e. >, >=, <, <=, =, <>, != and followed by a subquery. The main difference between ANY and ALL is that ANY returns true if any of the subquery values meet the condition whereas ALL returns true if all of the subquery values meet the condition.*

Select \* from customers where salary >any (select salary from customers where age=32);

SELECT DISTINCT AGE FROM CUSTOMERS WHERE SALARY < ANY (SELECT AVG(SALARY) FROM CUSTOMERS);

Calculate avg salary all customers then which customers salary is less than avg salary and distinct age.

SELECT \* FROM CUSTOMERS WHERE AGE = ANY (SELECT AGE FROM CUSTOMERS WHERE NAME LIKE 'K%');

1st execute subquery it find name and age like start with ‘k’ then after main query all ages match with subquery age showing result.

SELECT NAME, AGE, ADDRESS, SALARY FROM CUSTOMERS GROUP BY AGE, SALARY HAVING SALARY > ALL (SELECT AVG(SALARY) FROM CUSTOMERS);

Main query show all fields there are mention column name and its group by age and salary as unique (if age and salary in two rows same then show one . first row age 30 and salary 6000 then another’s rows same value then grouping by base on age and salary otherwise unique result ) then having salary is greater than avg salary.

EXISTS OPERATOR:

CASE:

The **SQL CASE** statement is a conditional statement that helps us to make decisions based on a set of conditions. It evaluates the set of conditions and returns the respective values when a condition is satisfied.

The CASE statement works like a simplified IF-THEN-ELSE statement and allows for multiple conditions to be tested.

SELECT NAME, AGE,

CASE WHEN AGE > 30 THEN ‘GEN X’

WHEN AGE > 25 THEN ‘GEN Y’

WHEN AGE > 22 THEN ‘GEN Z’

ELSE ‘Gen Alpha’ END AS Generation from customer;

SELECT \*, CASE WHEN SALARY<4500 THEN (SALARY +SALARY \*25/100)

END AS INCREAMENT FROM CUSTOMERS;

CASE STATEMENT WITH ORDER BY CLAUSE:

We can use CASE statement with ORDER BY clause. The ORDER BY clause in SQL sorts the result in ascending (default) or descending order.

select \* from customers order by CASE(

when name LIKE ‘k%’ then name else address

END);

In this query sorting based on name and address, name is not start from k then sort by on address and if name is present start from k then sort the bases on name. all row executes.

CASE STATEMENT WITH WHERE CLAUSE:

We can use the CASE statement with the WHERE clause as well. The WHERE clause is used to filter the rows in a table based on a specified condition.

SELECT NAME, ADDRESS, CASE WHEN AGE<25 THEN ‘intern’

WHEN AGE>=25 AND AGE<=27 THEN ‘engineer’

ELSE ‘senior developer’

END as Designation from customers where salary>=2500;

NOT OPERATOR:

Not operator basically used for boolean type like as true or fasle and not operator also used with where clause, between.

Select \* from customers where not exists( select customers\_id from orders where orders.customer\_id=customers.id);

NOT EQUAL:

The SQL **NOT EQUAL** operator is used to compare two values and return true if they are not equal. It is represented by **"<>"** and **"!="**. The difference between these two is that <> follows the ISO standard, but != doesn't. So, it is recommended to use the <> operator.

We can use the NOT EQUAL operator in WHERE clause to filter records based on a specific condition and in GROUP BY clause to group the results.

select \* from customers where name <> 'ramesh';

# this query does not show ramesh from the customers table column name.

select \* from customers where name <> 'ramesh' | 'khilan'; # query execute but not showing result.

NOT EQUAL WITH GROUP BY:

SELECT COUNT(ID), AGE FROM CUSTOMERS

WHERE AGE <> '22' GROUP BY AGE;

NOT EQUAL WITH MULTIPLE CONDITIONS:

SELECT \* FROM CUSTOMERS

WHERE ADDRESS <> 'Bhopal' AND (SALARY>'2000' OR SALARY='2000');

IS NULL;

SELECT \* FROM CUSTOMERS WHERE ADDRESS IS NULL;

SELECT \* FROM CUSTOMERS WHERE salary IS NULL;

SELECT COUNT(\*) FROM CUSTOMERS WHERE salary IS NULL;

DELETE FROM CUSTOMERS WHERE SALARY IS NULL;

UPDATE CUSTOMERS SET address = 'sangli' WHERE address IS NULL;

IS NOT NULL;

NOT NULL;

In a table, columns can typically accept NULL values by default. However, if you want to ensure that a particular column does not contain NULL values, you need to add the NOT NULL constraint/condition on that column.

BETWEEN OPERATOR:

The **BETWEEN** operator is a logical operator in SQL, that is used to retrieve the data within a specified range. The retrieved values can be integers, characters, or dates

UNION OPERATOR:

The SQL UNION operator is used to combine data from multiple tables by eliminating duplicate rows (if any).

The same number of columns selected with the same datatype.

These columns must also be in the same order.

They need not have same number of rows.

*When using UNION on a single field, the column names in the result set will be determined by the column name in the first SELECT statement. Therefore, you may need to use an alias in the SELECT statement to ensure that the column name is meaningful for the final result set.*

SELECT SALARY FROM CUSTOMERS1 UNION SELECT AMOUNT FROM ORDERS1;

select id from customers1 union select customer\_id from orders1;

Union on multiple fields:

1)SELECT ID, NAME, AMOUNT, DATE, CUSTOMER\_ID FROM CUSTOMERS LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

UNION

2)SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

1) its showing only those records match on condition of the left side table records otherwise records null of the right side table.

2) its showing only those records match on condition of the right side table records otherwise records null of the right side table.

Union with where clause:

We can use the WHERE clause with UNION operator to filter the results of each SELECT statement before combining them.

SELECT ID, SALARY FROM CUSTOMERS WHERE ID > 5

UNION

SELECT CUSTOMER\_ID, AMOUNT FROM ORDERS WHERE CUSTOMER\_ID > 2;

In the following query, we are retrieving the id's of the customers where id is greater than 5 and 2 from the 'CUSTOMERS' and 'ORDERS' tables respectively

#UNION with ORDER BY Clause:

SELECT ID, SALARY FROM CUSTOMERS WHERE ID > 5 UNION

SELECT CUSTOMER\_ID, AMOUNT FROM ORDERS WHERE CUSTOMER\_ID > 2

ORDER BY SALARY;

SQL INTERSECT Operator − This is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.

SQL EXCEPT Operator − This combines two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

UNION VS UNION ALL:

Both of them are used to retrieve the rows from multiple tables and return them as one single table. The difference between these two operators is that UNION only returns distinct rows while UNION ALL returns all the rows present in the tables.

The tables to be combined must have the same number of columns with the same datatype.

The number of rows need not be the same.

UNION is a type of operator/clause in SQL, that works similar to the union operator in relational algebra. It just combines the information from multiple tables that are union compatible.

UNION ALL is also an operator/clause in SQL, that is used to combine multiple tables into one table. However, this operator also preserves the duplicate rows in the resultant tables.

INTERSECT OPERATOR:

The **INTERSECT** operator in SQL is used to retrieve the records that are identical/common between the result sets of two or more tables.

EXCEPT OPERATOR:

The **EXCEPT** operator in SQL is used to retrieve all the unique records from the left operand (query), except the records that are present in the result set of the right operand (query).

reference link:

https://www.tutorialspoint.com/sql/sql-except-clause.htm

SQL JOINS:

The SQL **Join** clause is used to combine data from two or more tables in a database. When the related data is stored across multiple tables, joins help you to retrieve records combining the fields from these tables using their foreign keys.

The part of the Join clause that specifies the columns on which records from two or more tables are joined is known as **join-predicate**. This predicate is usually specified along with the ON clause and uses various comparison operators such as, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT etc. We can also connect multiple join predicates with logical operators AND, OR, and NOT.

TYPES OF JOIN:

INNER JOIN & OUTER JOIN

INNER JOINS:

An [INNER JOIN](https://www.tutorialspoint.com/sql/sql-inner-joins.htm) is the default join which retrieves the intersection of two tables. It compares each row of the first table with each row of the second table. If the pairs of these rows satisfy the join-predicate, they are joined together.

SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS INNER JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

SELECT OID, DATE, AMOUNT, EMPLOYEE\_NAME FROM CUSTOMERS

INNER JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

INNER JOIN EMPLOYEE ON ORDERS.OID = EMPLOYEE.EID;

SELECT ID, NAME, DATE, AMOUNT FROM CUSTOMERS

INNER JOIN ORDERS

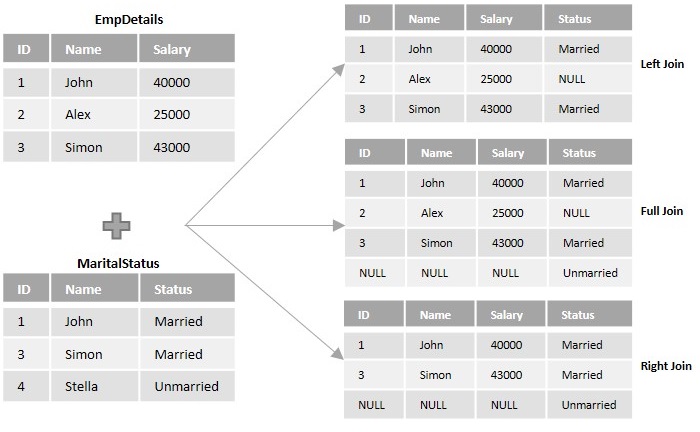
ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

WHERE ORDERS.AMOUNT > 2000.00;

OUTER JOINS:

Outer Join is used to join multiple database tables into a combined result-set, that includes all the records, even if they don't satisfy the join condition. NULL values are displayed against these records where the join condition is not met.

Outer join is further divided into three subtypes - Left Join, Right Join and Full Join.



LEFT JOINS:

Retrieves all the records from the left table(1st table), Matching records from the right table and NULL values in the unmatched rows.

SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

SELECT CUSTOMERS.ID, CUSTOMERS.NAME, ORDERS.DATE, EMPLOYEE.EMPLOYEE\_NAME FROM CUSTOMERS

LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

LEFT JOIN EMPLOYEE ON ORDERS.OID = EMPLOYEE.EID;

SELECT ID, NAME, DATE, AMOUNT FROM CUSTOMERS LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID WHERE ORDERS.AMOUNT > 2000.00;

RIGHT JOINS:

Retrieves all the records from the second table, Matching records from the first table and NULL values in the unmatched rows.

FULL JOINS:

Retrieves records from both the tables and fills the unmatched values with NULL.

OTHER JOINS: CROSS JOINS AND SELF JOINS

CROSS JOINS:

returns the Cartesian product of the sets of records from the two or more joined tables.

SELF JOINS:

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.

DELETE JOINS:

DELETE A FROM CUSTOMERS AS A INNER JOIN ORDERS AS B

ON A.ID=B.CUSTOMER\_ID;

WHERE A.SALARY>5000;

UPDATE JOINS:

UPDARE A FROM CUSTOMERS AS A INNER JOIN ORDERS AS B

ON A.ID=B.CUSTOMER\_ID SET A.SALARY=A.SALARY+1000,

B.AMOUNT =B.AMOUNT+500;

query increments the salary of customers by 1000 with respect to the inflation of their order amount by 500

LEFT VS RIGHT JOINS:

UNION VS JOIN;

SQL KEYS:

UNIQUE KEY:

PRIMARY KEY:

FOREIGN KEY:

COMPOSITE KEY:

ALTERNATE KEY:

SQL INDEXES:

CREATE INDEXES:

DROP INDEXES:

SHOW INDEXES:

UNIQUE INDEXES:

CLUSTRED INDEXES:

NON- CLUSTRED INDEXES:

ADVANCED SQL:

WILDCARDS:

COMMENTS:

INJECTION:

HOSTING:

MIN AND MAX:

NULL FUNCTIONS:

CHECK CONSTRAINTS:

DEFAULT CONSTRAINTS:

STORED PROCUDERS:

NULL VALUES:

TRANSACTIONS:

SUB QUERIES:

HANDLING DUPLICATES:

USING SEQUENCES:

AUTO INCREMENT:

DATE & TIME:

CURSORS:

COMMMON TABLE EXPRESSIONS:

GROP BY VS ORDER BY:

IN VS EXISTS:

DATABASE TUNNING:

SQL FUNCTION REFERENCES:

DATE FUNCTIONS:

STRING FUNCTIONS:

AGGREGATE FUNCTIONS:

NUMERIC FUNCTIONS:

TEXT & IMAGE FUNCTIONS:

STATICAL FUNCTIONS:

LOGICAL FUNCTIONS:

CURSOR FUNCTIONS:

JSON FUNCTIONS:

CONVERSATION FUNCTIONS:

DATATYPES FUNCTIONS:

SQL USEFUL RESOURCES:

select \* from customers;

SELECT DISTINCT salary

FROM customers

ORDER BY salary DESC

LIMIT 1 OFFSET 2;

SELECT MAX(salary) AS third\_highest\_salary

FROM customers

WHERE salary < (SELECT MAX(salary) FROM customers WHERE salary < (SELECT MAX(salary) FROM customers));

SELECT MAX(salary)

FROM customers

WHERE salary < (SELECT MAX(salary) FROM customers WHERE salary < (SELECT MAX(salary) FROM customers));

SELECT MAX(salary) AS third\_highest\_salary

FROM customers

WHERE salary NOT IN (SELECT salary FROM customers ORDER BY salary DESC LIMIT 2);

SELECT salary FROM customers ORDER BY salary DESC LIMIT 2;

SELECT DISTINCT e1.salary

FROM customers e1

JOIN customers e2 ON e1.salary <= e2.salary

GROUP BY e1.salary

HAVING COUNT(DISTINCT e2.salary) = 3;

SELECT salary

FROM (

SELECT salary, DENSE\_RANK() OVER (ORDER BY salary DESC) AS rnk

FROM customers

) AS ranked

WHERE rnk = 3;