**RDBMS: R**elational **D**atabase **M**anagement **S**ystem

Examples: E. F. Codd in 1970.

MySQL, MS SQL Server, ORACLE, MS ACCESS, PostgreSQL, SQLite

Data Integrity

The following categories of data integrity exist with each RDBMS −

* **Entity Integrity** − This ensures that there are no duplicate rows in a table.
* **Domain Integrity** − Enforces valid entries for a given column by restricting the type, the format, or the range of values.
* **Referential integrity** − Rows cannot be deleted, which are used by other records.
* **User-Defined Integrity** − Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process −

Eliminating redundant data, for example, storing the same data in more than one table.

Ensuring data dependencies make sense.

main types of SQL data types available in any RDBMS

* String
* Numeric
* Date and Time

#### **MySQL - String Data Types**

MySQL - String Data Types

**CHAR(size)**: Fixed length string (0-255 characters). Default is 1.

Example: CHAR(10)

**VARCHAR(size)**: Variable length string (0-65535 characters).

Example: VARCHAR(255)

**BINARY(size)**: Fixed length binary string. Default is 1.

Example: BINARY(10)

**VARBINARY(size)**: Variable length binary string.

Example: VARBINARY(255)

**TINYTEXT**: String up to 255 characters.

**TEXT(size)**: String up to 65,535 bytes.

**LONGTEXT**: String up to 4,294,967,295 characters.

**TINYBLOB**: Binary up to 255 bytes.

**BLOB(size)**: Binary up to 65,535 bytes.

**MEDIUMTEXT**: String up to 16,777,215 characters.

**MEDIUMBLOB**: Binary up to 16,777,215 bytes.

**LONGBLOB**: Binary up to 4,294,967,295 bytes.

**ENUM(val1, val2, ...)**: Single value from a list (up to 65535 values).

Example: ENUM('small', 'medium', 'large')

CREATE TABLE sizes (

size ENUM('small', 'medium', 'large') NOT NULL

);

INSERT INTO sizes (size) VALUES ('small'), ('medium'), ('large');

SELECT \* FROM sizes WHERE size = 'medium';

**SET(val1, val2, ...)**: Multiple values from a list (up to 64 values).

Example: SET('option1', 'option2', 'option3')

**MySQL - Numeric Data Types**

**INT**: Signed (-2147483648 to 2147483647) or unsigned (0 to 4294967295). Width up to 11 digits. Example: INT(11)

**TINYINT**: Signed (-128 to 127) or unsigned (0 to 255). Width up to 4 digits. Example: TINYINT(4)

**SMALLINT**: Signed (-32768 to 32767) or unsigned (0 to 65535). Width up to 5 digits.

Example: SMALLINT(5)

**MEDIUMINT**: Signed (-8388608 to 8388607) or unsigned (0 to 16777215). Width up to 9 digits.

Example: MEDIUMINT(9)

**BIGINT**: Signed (-9223372036854775808 to 9223372036854775807) or unsigned (0 to 18446744073709551615). Width up to 20 digits.

Example: BIGINT(20)

**FLOAT(M,D)**: Floating-point number (default 10,2). Precision up to 24 places.

Example: FLOAT(7,3)

**DOUBLE(M,D)**: Double precision floating-point number (default 16,4). Precision up to 53 places.

Example: DOUBLE(10,4)

**DECIMAL(M,D)**: Fixed-point number. Precision and scale required.

Example: DECIMAL(10,2)

**MySQL - Date and Time Data Types**

**DATE**: YYYY-MM-DD, from 1000-01-01 to 9999-12-31.

Example: 1973-12-30

**DATETIME**: YYYY-MM-DD HH:MM

, from 1000-01-01 00:00:00 to 9999-12-31 23:59:59.

Example: 1973-12-30 15:30:00

**TIMESTAMP**: YYYYMMDDHHMMSS, from 1970-01-01 to 2037.

Example: 19731230153000

**TIME**: HH:MM

format.

Example: 15:30:00

**YEAR(M)**: 2-digit (70-69 for 1970-2069) or 4-digit (1901-2155). Default is 4 digits.

Example: YEAR(2) -> 73, YEAR(4) -> 1973

-------------tutot

**SQL-Syntax**

CREATE DATABASE DatabaseName;

SHOW DATABASES;//SHOW DATABASES [LIKE 'pattern' | WHERE expr] ;

SHOW SCHEMAS LIKE 'test%';///// SCHEME=DB

DROP DATABASE DatabaseName; //DROP DATABASE IF EXISTS DatabaseName;//DROP DATABASE testDB3, testDB4;

SHOW DATABASES; //list down all the available databases:

**SQL Operators**

**What is SQL Operator?**

An SQL operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), they can be either a **unary** or **binary operator(two operands )**

[Arithmetic operators](https://www.tutorialspoint.com/sql/sql-arithmetic-operators.htm): +,-,\*,/,%

* [Comparison operators](https://www.tutorialspoint.com/sql/sql-comparison-operators.htm):=, !=,<>,<,>,<=,>=,!<,!>
* [Logical operators](https://www.tutorialspoint.com/sql/sql-logical-operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| ALL | TRUE if all comparisons are TRUE | SELECT \* FROM table WHERE column > ALL (subquery); |
| AND | TRUE if all conditions are TRUE | SELECT \* FROM table WHERE condition1 AND condition2; |
| ANY | TRUE if any one comparison is TRUE | SELECT \* FROM table WHERE column = ANY (subquery); |
| BETWEEN | TRUE if operand is within a range | SELECT \* FROM table WHERE column BETWEEN value1 AND value2; |
| EXISTS | TRUE if subquery returns one or more records | SELECT \* FROM table WHERE EXISTS (subquery); |
| IN | TRUE if operand equals any in a list | SELECT \* FROM table WHERE column IN (value1, value2, ...); |
| LIKE | TRUE if operand matches a pattern | SELECT \* FROM table WHERE column LIKE 'pattern%'; |
| NOT | Reverses the value of a Boolean operator | SELECT \* FROM table WHERE NOT condition; |
| OR | TRUE if any condition is TRUE | SELECT \* FROM table WHERE condition1 OR condition2; |
| IS NULL | TRUE if expression value is NULL | SELECT \* FROM table WHERE column IS NULL; |
| SOME | TRUE if some comparisons are TRUE | SELECT \* FROM table WHERE column = SOME (subquery); |
| UNIQUE | Ensures all values in a column are unique | SELECT UNIQUE column FROM table; |

* Operators used to negate conditions

|  |  |
| --- | --- |
| **Operator** | **Operation** |
| +, - | identity, negation |
| \*, / | multiplication, division |
| +, - | addition, subtraction |
| =, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN | Comparison |
| NOT | logical negation |
| AND | conjunction |
| OR | inclusion |

**SQL - Expressions**

An SQL expression is a combination of one or more values, operators and SQL functions that are all evaluated to a value. used in WHERE clause, WHERE clause specifies a condition that needs to be satisfied for the purpose of filtering records from a database table. This condition is comprised of either single or multiple expressions. These expressions are further classified into three types −

* Boolean Expressions
* Numeric Expressions
* Date and time Expressions

SELECT column1, column2, columnN

FROM table\_name

WHERE [CONDITION|EXPRESSION];

**SQL Boolean Expressions:**SQL Boolean Expressions are SQL expressions that return only Boolean Datatype as a result.

SELECT \* FROM CUSTOMERS WHERE SALARY = 10000;(mostly comparision operators are used)

**SQL Date Expressions**

SQL Date Expressions are used to compare date related values with current system date and time values.

SELECT column\_name(s)

FROM table\_name

WHERE DATE EXPRESSION ;

Ex:: SELECT CURRENT\_TIMESTAMP;

SELECT \* FROM ORDERS WHERE DATE < '2008/06/01';

**SQL Database**

A database is a structured collection of data that is stored in a computer system. They are used to store and retrieve the data efficiently. Databases can be created using different query languages, and SQL is one such language.

SQL DROP Database Statement

**Following are the important points to remember before you delete an existing database −**

Make sure you have taken proper backup of the database before you delete it.

Make sure no other application is connected and using this database.

Make sure you have the necessary privilege to delete the database. Usually an admin can delete the databaase.

CREATE DATABASE testDB1;

USE testDB;

The SQL **USE DATABASE** statement is used to select a database from a list of databases available in the system. Once a database is selected, we can perform various operations on it such as [creating tables](https://www.tutorialspoint.com/sql/sql-create-table.htm), [inserting data](https://www.tutorialspoint.com/sql/sql-insert-query.htm), updating data, and [deleting data](https://www.tutorialspoint.com/sql/sql-delete-query.htm).

DROP DATABASE database\_name; -> DROP DATABASE testDB1;

DROP DATABASE IF EXISTS DatabaseName;//[ DROP DATABASE testDB3, testDB4;]

SHOW DATABASES;

SELECT \* FROM sys.databases;

RENAME DATABASE OldDatabaseName TO NewDatabaseName;

ALTER DATABASE OldDatabaseName MODIFY NAME = NewDatabaseName;

**EG**:: ALTER DATABASE testDB MODIFY NAME = tutorialsDB ;

# SQL Backup Database

In SQL, there are three types of database backups available. These are:

* Full Backup
* Differential Backup
* Transaction Log (T-log) backup

**syntax of SQL Backup Database Statement −**

BACKUP DATABASE database\_name

TO DISK = 'filepath'

GO

Full Backup −

BACKUP DATABASE database\_name

TO medium = 'filepath'

GO

backup only the new changes by using the **WITH DIFFERENTIAL** command.

Syntax

Following is the syntax of Sql Differential Backup −

BACKUP DATABASE my\_db

TO medium = 'filepath'

WITH DIFFERENTIAL;

GO

**Transaction Log (T-log) backup**

A transaction log backup includes all the transactions since the last transaction log backup. BACKUP LOG comnmand is used to perfom the Transaction Log backup.

Syntax

Following is the syntax of Sql transaction log backup −

BACKUP LOG database\_name

TO medium = 'filepath';

GO

**Restore Database From Backup**

To restore a backup file in Database, we can use the RESTORE DATABASE command.

Syntax

Following is the syntax of Restore Database From Backup −

RESTORE DATABASE database\_name

FROM DISK = 'filepath';

GO

**Restore MySQL Database**

If we have a database dump then we can use the following two step process to restore our database. First step is to create our new database using **mysqladmin** prompt command as follo ws:

$ mysqladmin -u username -p"password" create tutorialsDB;

The next step is to import old database into new database shown below :

$ mysql -u username -p"password" tutorialsDB < testDB.sql;

syntax to list down all the tables in SQL using the **SYS.TABLES** view − Server **does not** provide SHOW TABLE command in an SQL Server. --sys.tables--information\_schema.tables--sysobjects

SELECT \* FROM SYS.TABLES;

Following is the output of the above query −

|  |  |  |  |
| --- | --- | --- | --- |
| name | object\_id | principal\_id | schema\_id |
| CUSTOMER | 4195065 | NULL | 1 |
| ORDERS | 68195293 | NULL | 1 |
| COMPANIES | 100195407 | NULL | 1 |
| SALARY | 2107154552 | NULL | 1 |

**SQL - CREATE Table**

**CREATE TABLE IF NOT EXISTS table\_name(**

**column1 datatype,**

**.....**

**columnN datatype,**

**PRIMARY KEY( one or more columns )**

**);**

 creating a table from another table is as follows −

CREATE TABLE NEW\_TABLE\_NAME AS

SELECT [column1, column2...columnN]

FROM EXISTING\_TABLE\_NAME

WHERE Condition;

**SQL Rename Table**

SHOW TABLES;

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

ALTER TABLE BUYERS RENAME TO CUSTOMERS;

TRUNCATE TABLE table\_name;

ALTER TABLE table\_name RENAME [TO|AS] new\_table\_name

Avoid renaming system tables: Renaming system tables can cause issues with the database system.  
• Update all references to the table: Stored procedures, views, triggers, or other database objects should be updated to use the new table name.  
• Test thoroughly: Before renaming in a production environment, ensure all references to the table have been updated correctly.  
• Use a consistent naming convention: Follow the same naming convention for tables and other database objects to simplify understanding and maintenance.  
• Backup the database: Create a backup of the database before renaming to have a restore point in case of any issues.

**SQL - TRUNCATE TABLE**

SQL provides command to TRUNCATE a table completely in one go instead of deleting table records one by one which will be very time consuming and cumbersome process.

TRUNCATE TABLE table\_name;

**TRUNCATE TABLE CUSTOMER** statement will remove all the records of the CUSTOMERS table −

TRUNCATE TABLE CUSTOMERS;

|  |  |
| --- | --- |
| **DELETE** | **TRUNCATE** |
| The [**DELETE command**](https://www.tutorialspoint.com/sql/sql-delete-table.htm) 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. |
| It deletes rows one at a time and applies same criteria to each deletion. | It removes all of the information in one go. |
| The WHERE clause serves as the condition in this case. | The WHERE Clause is not available. |
| All rows are locked after deletion. | TRUNCATE utilizes a table lock, which locks the pages so they cannot be deleted. |
| It makes a record of each and every transaction in the log file. | The only activity recorded is the deallocation of the pages on which the data is stored. |
| It consumes a greater amount of transaction space compared to TRUNCATE command. | It takes comparatively less amount of transaction space. |
| If there is an identity column, the table identity is not reset to the value it had when the table was created. | It returns the table identity to a value it was given as a seed. |
| It requires authorization to delete. | It requires table alter permission. |
| When it comes to large databases, it is much slower. | It is much faster. |

|  |  |
| --- | --- |
| **DROP** | **TRUNCATE** |
| The DROP command in SQL removes an entire table from a database including its definition, indexes, constraints, data etc. | The TRUNCATE command is used to remove all of the rows from a table, regardless of whether or not any conditions are met and resets the table definition. |
| It is a DDL(Data Definition Language) command. | It is also a DDL(Data Definition Language) command. |
| The table space is completely freed from the memory. | The table still exists in the memory. |
| All the integrity constraints are removed. | The integrity constraints still exist in the table. |
| Requires ALTER and CONTROL permissions on the table schema and table respectively, to be able to perform this command. | Only requires the ALTER permissions to truncate the table. |
| DROP command is much slower than TRUNCATE but faster than DELETE. | TRUNCATE command is faster than both DROP and DELETE commands. |

**CLONING**

SQL **Cloning Operation** allows to create the exact copy of an existing table along with its definition. There are three types of cloning possible using SQL in various RDBMS; they are listed below −

* Simple Cloning
* Shallow Cloning
* Deep Cloning

CREATE TABLE new\_table SELECT \* FROM original\_table; simple--full

CREATE TABLE new\_table LIKE original\_table;---shallow

CREATE TABLE new\_table LIKE original\_table;

INSERT INTO new\_table SELECT \* FROM original\_table;---deep cloning

 **Simple Cloning**: Copies table structure and data.

CREATE TABLE new\_table AS SELECT \* FROM existing\_table;

* Note: Does not copy indexes or AUTO\_INCREMENT.

 **Shallow Cloning**: Copies table structure only (no data).

CREATE TABLE new\_table LIKE existing\_table;

* Note: Copies indexes and AUTO\_INCREMENT.

 **Deep Cloning**: Copies table structure and data, including indexes and AUTO\_INCREMENT.

CREATE TABLE new\_table LIKE existing\_table;

INSERT INTO new\_table SELECT \* FROM existing\_table;

**TEMPRORY TABLE**

CREATE TEMPORARY 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)

);

INSERT INTO CUSTOMERS VALUES

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

DROP TEMPORARY TABLE table\_name;

**Temporary Tables in SQL Server**

**Types of Temporary Tables:**

1. **Local Temporary Tables**:
   * Accessible only within the session that created it.
   * Automatically deleted when the connection closes.
   * If created inside a stored procedure, it is dropped after the procedure execution completes.

CREATE TABLE #LocalTemp (ID INT, Name VARCHAR(50));

1. **Global Temporary Tables**:
   * Visible to all sessions and connections.
   * Dropped when the last connection referencing the table closes.

CREATE TABLE ##GlobalTemp (ID INT, Name VARCHAR(50));

**Alter table**

ALTER TABLE table\_name ADD column\_name datatype;

ALTER TABLE table\_name DROP COLUMN column\_name;

ALTER TABLE CUSTOMERS ADD INDEX name\_index (NAME);

ALTER TABLE table\_name DROP INDEX index\_name;

ALTER TABLE EMPLOYEES

ADD CONSTRAINT MyPrimaryKey

PRIMARY KEY(ID);

Different Operations Using ALTER in SQL

Add Column

ALTER TABLE table\_name ADD column\_name datatype;

ALTER TABLE Customers ADD Email VARCHAR(100);

2. Drop Column

ALTER TABLE table\_name DROP COLUMN column\_name;

ALTER TABLE Customers DROP COLUMN Email;

3. Modify Column

ALTER TABLE table\_name MODIFY column\_name new\_datatype;

ALTER TABLE Customers MODIFY Email VARCHAR(150);

4. Rename Column

Syntax (MySQL):

ALTER TABLE table\_name CHANGE old\_column\_name new\_column\_name datatype;

ALTER TABLE Customers CHANGE Email EmailAddress VARCHAR(150);

5. Rename Table

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

ALTER TABLE Customers RENAME TO Clients;

6. Add Primary Key

ALTER TABLE table\_name ADD PRIMARY KEY (column\_name);

ALTER TABLE Customers ADD PRIMARY KEY (ID);

7. Drop Primary Key

ALTER TABLE table\_name DROP PRIMARY KEY;

ALTER TABLE Customers DROP PRIMARY KEY;

8. Add Foreign Key

ALTER TABLE table\_name ADD CONSTRAINT fk\_name FOREIGN KEY (column\_name) REFERENCES parent\_table(parent\_column);

ALTER TABLE Orders ADD CONSTRAINT FK\_CustomerID FOREIGN KEY (CustomerID) REFERENCES Customers(ID);

9. Drop Foreign Key

ALTER TABLE table\_name DROP FOREIGN KEY fk\_name;

ALTER TABLE Orders DROP FOREIGN KEY FK\_CustomerID;

10. Add Unique Constraint

ALTER TABLE table\_name ADD CONSTRAINT constraint\_name UNIQUE (column\_name);

ALTER TABLE Customers ADD CONSTRAINT UQ\_Email UNIQUE (Email);

11. Drop Unique Constraint

ALTER TABLE table\_name DROP INDEX constraint\_name;

ALTER TABLE Customers DROP INDEX UQ\_Email;

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE,

Salary DECIMAL(10, 2),

DepartmentID INT,

CONSTRAINT CHK\_Salary CHECK (Salary >= 0),

FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)

);

ALTER TABLE Employee DROP CONSTRAINT PK\_EmployeeID;

ALTER TABLE Employee ALTER COLUMN FirstName VARCHAR(50) NULL;

ALTER TABLE Employee ALTER COLUMN LastName VARCHAR(50) NULL;

**Adding not null constraint**

ALTER TABLE Employee ALTER COLUMN FirstName VARCHAR(50) NOT NULL;

ALTER TABLE Employee ALTER COLUMN LastName VARCHAR(50) NOT NULL;

ALTER TABLE Employee ADD CONSTRAINT FK\_DepartmentID FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID);

**SQL - ORDER BY Clause**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| ORDER BY Single Column | How to sort the result by a single column? | SELECT name FROM employees ORDER BY name; |
| ORDER BY Multiple Columns | How to sort the result by multiple columns? | SELECT name, salary FROM employees ORDER BY department, salary; |
| ORDER BY Descending Order | How to sort the result in descending order? | SELECT name FROM employees ORDER BY name DESC; |
| ORDER BY Ascending and Descending | How to sort multiple columns in different orders? | SELECT name, salary FROM employees ORDER BY department ASC, salary DESC; |
| ORDER BY with LIMIT | How to sort and limit the result set? | SELECT name FROM employees ORDER BY salary DESC LIMIT 10; |

**SQL - GROUP BY Clause**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| GROUP BY Single Column | How to group the result by a single column? | SELECT department, COUNT(\*) FROM employees GROUP BY department; |
| GROUP BY Multiple Columns | How to group the result by multiple columns? | SELECT department, job\_title, COUNT(\*) FROM employees GROUP BY department, job\_title; |
| GROUP BY with Aggregate Function | How to use aggregate functions with GROUP BY? | SELECT department, AVG(salary) FROM employees GROUP BY department; |
| GROUP BY with HAVING Clause | How to filter grouped results using HAVING? | SELECT department, COUNT(\*) FROM employees GROUP BY department HAVING COUNT(\*) > 10; |
| GROUP BY with ORDER BY | How to sort grouped results? | SELECT department, COUNT(\*) FROM employees GROUP BY department ORDER BY COUNT(\*) DESC; |

**SQL - HAVING Clause**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| HAVING with Aggregate Function | How to use HAVING with an aggregate function? | SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000; |
| HAVING with Multiple Conditions | How to use multiple conditions in HAVING? | SELECT department, COUNT(\*) FROM employees GROUP BY department HAVING COUNT(\*) > 10 AND AVG(salary) > 50000; |
| HAVING with WHERE and GROUP BY | How to use HAVING with WHERE and GROUP BY? | SELECT department, COUNT(\*) FROM employees WHERE hire\_date > '2020-01-01' GROUP BY department HAVING COUNT(\*) > 5; |
| HAVING with Aliased Columns | How to use HAVING with aliased columns? | SELECT department, COUNT(\*) AS emp\_count FROM employees GROUP BY department HAVING emp\_count > 10; |
| HAVING with Subquery | How to use HAVING with a subquery? | SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > (SELECT AVG(salary) FROM employees); |

**SQL - AND & OR**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| AND Clause | How to use the AND operator in a WHERE clause? | SELECT \* FROM employees WHERE department = 'Sales' AND salary > 50000; |
| OR Clause | How to use the OR operator in a WHERE clause? | SELECT \* FROM employees WHERE department = 'Sales' OR department = 'HR'; |
| Combined AND & OR | How to combine AND and OR operators? | SELECT \* FROM employees WHERE (department = 'Sales' AND salary > 50000) OR department = 'HR'; |
| AND with Multiple Conditions | How to use multiple AND conditions? | SELECT \* FROM employees WHERE department = 'Sales' AND salary > 50000 AND hire\_date > '2020-01-01'; |
| OR with Multiple Conditions | How to use multiple OR conditions? | SELECT \* FROM employees WHERE department = 'Sales' OR department = 'HR' OR department = 'Finance'; |

**SQL - BOOLEAN (BIT) Operator**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| BIT as Boolean | How to use BIT data type for boolean values? | CREATE TABLE tasks (id INT, is\_completed BIT); |
| Insert Boolean True | How to insert a TRUE value in a BIT column? | INSERT INTO tasks (id, is\_completed) VALUES (1, 1); |
| Insert Boolean False | How to insert a FALSE value in a BIT column? | INSERT INTO tasks (id, is\_completed) VALUES (2, 0); |
| Query with Boolean | How to query records with a boolean condition? | SELECT \* FROM tasks WHERE is\_completed = 1; |
| Update Boolean | How to update a BIT column? | UPDATE tasks SET is\_completed = 1 WHERE id = 2; |

**SQL - LIKE Operator**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic LIKE | How to use the LIKE operator for pattern matching? | SELECT \* FROM employees WHERE name LIKE 'J%'; |
| LIKE with Wildcard (%) | How to use % as a wildcard? | SELECT \* FROM employees WHERE name LIKE '%Smith'; |
| LIKE with Wildcard (\_) | How to use \_ as a wildcard for a single character? | SELECT \* FROM employees WHERE name LIKE '\_a%'; |
| LIKE with NOT | How to use NOT with LIKE for pattern matching? | SELECT \* FROM employees WHERE name NOT LIKE 'J%'; |
| LIKE with Escape Character | How to use escape character in LIKE? | SELECT \* FROM employees WHERE name LIKE '50\%%' ESCAPE '\'; |

**SQL - IN Operator**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic IN | How to use the IN operator for a list of values? | SELECT \* FROM employees WHERE department IN ('Sales', 'HR', 'Finance'); |
| IN with Subquery | How to use IN with a subquery? | SELECT \* FROM employees WHERE department\_id IN (SELECT id FROM departments WHERE location = 'NY'); |

**SQL - ANY, ALL Operators**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| ANY with Comparison | How to use ANY with a comparison operator? | SELECT \* FROM employees WHERE salary > ANY (SELECT salary FROM employees WHERE department = 'HR'); |
| ALL with Comparison | How to use ALL with a comparison operator? | SELECT \* FROM employees WHERE salary > ALL (SELECT salary FROM employees WHERE department = 'HR'); |

**SQL - EXISTS Operator**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic EXISTS | How to use the EXISTS operator? | SELECT \* FROM employees WHERE EXISTS (SELECT 1 FROM departments WHERE employees.department\_id = departments.id); |
| EXISTS with Subquery | How to use EXISTS with a subquery? | SELECT \* FROM employees WHERE EXISTS (SELECT 1 FROM projects WHERE projects.manager\_id = employees.id); |

**QL - CASE**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Simple CASE | How to use the simple CASE expression? | SELECT employee\_id, salary, CASE department WHEN 'Sales' THEN salary \* 1.1 WHEN 'HR' THEN salary \* 1.2 ELSE salary END AS adjusted\_salary FROM employees; |
| Searched CASE | How to use the searched CASE expression? | SELECT employee\_id, salary, CASE WHEN department = 'Sales' THEN salary \* 1.1 WHEN department = 'HR' THEN salary \* 1.2 ELSE salary END AS adjusted\_salary FROM employees; |

**SQL - NOT Operator**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic NOT | How to use the NOT operator? | SELECT \* FROM employees WHERE NOT department = 'Sales'; |
| NOT with AND | How to use NOT with AND? | SELECT \* FROM employees WHERE NOT (department = 'Sales' AND salary > 50000); |
| NOT with OR | How to use NOT with OR? | SELECT \* FROM employees WHERE NOT (department = 'Sales' OR department = 'HR'); |

**SQL - NOT EQUAL**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic Not Equal | How to use the NOT EQUAL operator? | SELECT \* FROM employees WHERE department <> 'Sales'; |
| Not Equal with AND | How to use Not Equal with AND? | SELECT \* FROM employees WHERE department <> 'Sales' AND salary > 50000; |
| Not Equal with OR | How to use Not Equal with OR? | SELECT \* FROM employees WHERE department <> 'Sales' OR department <> 'HR'; |

DROP TABLE table\_name;

DSC Tablename; Describe table name

DESCRIBE CUSTOMERS;

ALTER TABLE EMPLOYEES DROP PRIMARY KEY;

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name

UNIQUE(column1, column2...);

ALTER TABLE CUSTOMERS RENAME COLUMN name to full\_name;

ALTER TABLE table\_name MODIFY COLUMN column\_name datatype;//datatype

**DROP Table**

DROP TABLE IF EXISTS CUSTOMERS; DROP TEMPORARY TABLE TEMP\_TABLE;

DELETE FROM table\_name

WHERE condition1 AND condition2 OR ... conditionN;

DELETE FROM CUSTOMERS;

XXXXX

**SQL - Delete Table**

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. Therefore, any constraints, indexes, or triggers defined in the table will still exist after you delete data from it.

DELETE FROM table\_name

WHERE condition;

DELETE FROM CUSTOMERS WHERE NAME='Hardik';

DELETE FROM table\_name

WHERE condition1 AND condition2 OR ... conditionN;

DELETE FROM CUSTOMERS;-🡪>>Deleting all the records in a table

SQL Constraints/////////////////

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 a specific condition

DEFAULT - Sets a default value for a column if no value is specified

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

|  |  |  |
| --- | --- | --- |
| Operation | Query Question | Answer |
| Add Primary Key | How to add a primary key constraint to a column in a table? | ALTER TABLE Employees ADD CONSTRAINT PK\_EmployeeID PRIMARY KEY (EmployeeID); |
| Add Foreign Key | How to add a foreign key constraint to a column in a table? | ALTER TABLE Orders ADD CONSTRAINT FK\_CustomerID FOREIGN KEY (CustomerID) REFERENCES Customers(ID); |
| Add Unique | How to add a unique constraint to a column in a table? | ALTER TABLE Employees ADD CONSTRAINT UQ\_Email UNIQUE (Email); |
| Add Check | How to add a check constraint to a column in a table? | ALTER TABLE Employees ADD CONSTRAINT CHK\_Age CHECK (Age >= 18); |
| Drop Primary Key | How to drop a primary key constraint from a table? | ALTER TABLE Employees DROP CONSTRAINT PK\_EmployeeID; |
| Drop Foreign Key | How to drop a foreign key constraint from a table? | ALTER TABLE Orders DROP CONSTRAINT FK\_CustomerID; |
| Drop Unique | How to drop a unique constraint from a table? | ALTER TABLE Employees DROP CONSTRAINT UQ\_Email; |
| Drop Check | How to drop a check constraint from a table? | ALTER TABLE Employees DROP CONSTRAINT CHK\_Age; |
| Add Default | How to add a default constraint to a column in a table? | ALTER TABLE Employees ADD CONSTRAINT DF\_DefaultSalary DEFAULT 50000 FOR Salary; |
| Drop Default | How to drop a default constraint from a table? | ALTER TABLE Employees DROP CONSTRAINT DF\_DefaultSalary; |

CREATE TABLE ORDERS (

ID INT NOT NULL,

DATE DATETIME,

CUSTOMER\_ID INT FOREIGN KEY REFERENCES CUSTOMERS(ID),

AMOUNT DECIMAL,

PRIMARY KEY (ID)

);

ALTER TABLE table\_name

MODIFY COLUMN column\_name datatype NULL;

[[ ALTER TABLE CUSTOMERS MODIFY COLUMN NAME VARCHAR(20) NULL;]]

**Adding a NOT NULL Constraint to the Existing Table**

ALTER TABLE CUSTOMERS MODIFY COLUMN ADDRESS CHAR(25) NOT NULL;

AGE INT NOT NULL CHECK(AGE>=18),

ALTER TABLE CUSTOMERS DROP CONSTRAINT PRIMARY KEY;

**SQL - INSERT Query**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| INSERT | How to insert values into a table? | INSERT INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...); |
| INSERT INTO SELECT | How to insert data from another table? | INSERT INTO table\_name (column1, column2, ...) SELECT column1, column2, ... FROM another\_table WHERE condition; |
| INSERT IGNORE | How to insert data ignoring duplicates? | INSERT IGNORE INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...); |
| REPLACE INTO | How to replace existing data or insert new? | REPLACE INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...); |

INSERT INTO BUYERS (ID, NAME, AGE, ADDRESS, SALARY)

SELECT \* FROM CUSTOMERS;

INSERT INTO CUSTOMERS

VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** | **Explanation** |
| Inserting Data into a Table Using Another | How to insert data from one table into another? | INSERT INTO table1 SELECT \* FROM table2 WHERE condition; | This operation inserts data from table2 into table1 based on a specified condition. It's useful for transferring data between tables or selecting subsets of data from one table into another. |
| Using INSERT... SELECT | How to insert selected data into a table? | INSERT INTO table\_name (column1, column2, ...) SELECT column1, column2, ... FROM another\_table WHERE condition; | This allows inserting data into specified columns of table\_name from a SELECT query that retrieves data from another\_table. It's effective for copying data or inserting filtered results into a table. |
| Using INSERT... TABLE | How to insert all data from one table into another? | INSERT INTO table1 SELECT \* FROM table2; | This operation inserts all rows and columns from table2 into table1. It's useful when you need to duplicate the entire structure and data of one table into another. |
| Insert Data Only in Specified Columns | How to insert data into specific columns of a table? | INSERT INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...); | This method explicitly specifies which columns of table\_name will receive new values. It's essential when you want to insert data into specific fields and omit others that may have default values or be auto-generated. |

Computing Using SELECT

SELECT mathematical\_expression;

SELECT 56\*65;

# SQL - Select Into Statement

Copying Data From Specific Columns

SELECT column1, column2, ..., columnN

INTO new\_table\_name

FROM existing\_table\_name;

Copying Data From Multiple Tables

SELECT column1, column2, ..., columnN

INTO new\_table\_name

FROM table1

JOIN table2 ON table1.column = table2.column

Copying Specific Records

SELECT \*

INTO new\_table\_name

FROM existing\_table\_name

WHERE condition;

SELECT column1, column2, ..., columnN

INTO new\_table\_name

FROM table1

JOIN table2 ON table1.column = table2.column

====================================

CREATE DATABASE testDB;

USE testDB;

ALTER DATABASE OldDatabaseName MODIFY NAME = NewDatabaseName;//RENAME DATABASE OldDatabaseName TO NewDatabaseName;

$ mysqladmin -u username -p"password" create tutorialsDB;

String

Numeric

Date and Time

CREATE TABLE comprehensive\_example (

id INT AUTO\_INCREMENT PRIMARY KEY, -- Integer type with auto-increment

name VARCHAR(100), -- Variable-length string

email VARCHAR(255), -- Variable-length string

profile\_fixed CHAR(10), -- Fixed-length string

description TEXT, -- Long text data

birthdate DATE, -- Date value

appointment\_time TIME, -- Time value

created\_at DATETIME, -- Date and time value

updated\_at TIMESTAMP, -- Timestamp value

membership\_year YEAR, -- Year value

total\_sales DECIMAL(10, 2), -- Exact fixed-point number

rating FLOAT, -- Floating-point number

large\_data BLOB, -- Binary large object

preferences JSON, -- JSON formatted data

status ENUM('active', 'inactive', 'pending'), -- Enum type

tags SET('new', 'featured', 'sale') -- Set type

);

id: An integer that auto-increments with each new row, serving as the primary key.

name: A variable-length string for storing names.

email: A variable-length string for storing email addresses.

profile\_fixed: A fixed-length string, suitable for short, fixed-width data.

description: A long text field for storing descriptions.

birthdate: A date field for storing birthdates in YYYY-MM-DD format.

appointment\_time: A time field for storing times in HH:MM:SS format.

created\_at: A datetime field for storing date and time values in YYYY-MM-DD HH:MM:SS format.

updated\_at: A timestamp field for storing automatic timestamps when rows are updated.

membership\_year: A year field for storing years in YYYY format.

total\_sales: A decimal field for storing exact fixed-point numbers with 10 digits in total and 2 digits after the decimal point.

rating: A floating-point field for storing ratings.

large\_data: A binary large object for storing large binary data like images or files.

preferences: A JSON field for storing JSON formatted data.

status: An enumeration for storing one of the predefined status values.

tags: A set for storing zero or more tags chosen from the predefined list.

INSERT INTO comprehensive\_example (

name, email, profile\_fixed, description, birthdate, appointment\_time, created\_at,

updated\_at, membership\_year, total\_sales, rating, large\_data, preferences, status, tags

) VALUES (

'John Doe', 'john.doe@example.com', 'JD12345', 'Sample description text.', '1990-05-15',

'13:45:00', '2024-06-20 10:30:00', '2024-06-20 10:30:00', 2024, 12345.67, 4.5,

LOAD\_FILE('/path/to/file'), '{"key": "value"}', 'active', 'new,featured'

);

query inserts the records of the customers whose name starts with the letter **k** from the CUSTOMERS table to the BUYERS table −

INSERT INTO NAMESTARTSWITH\_K

SELECT \* FROM CUSTOMERS

WHERE NAME LIKE 'k%';

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** | **Explanation** |
| Basic Insertion | How to insert a single row of data into a table? | INSERT INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...); | This is the basic form of INSERT INTO statement used to insert a single row of data into specified columns of table\_name. It's useful for adding new records to a table. |
| Inserting Multiple Rows | How to insert multiple rows of data into a table at once? | INSERT INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...), (value1, value2, ...), ...; | This operation allows inserting multiple rows of data into table\_name with a single INSERT INTO statement. It's efficient for bulk data insertion. |
| Inserting Data from Another Table | How to insert data into a table from another table? | INSERT INTO table1 SELECT column1, column2, ... FROM table2 WHERE condition; | This query inserts data into table1 from a SELECT query that retrieves data from table2. It's useful for copying data between tables or inserting filtered results. |
| Inserting Default Values | How to insert default values into a table? | INSERT INTO table\_name DEFAULT VALUES; | This operation inserts default values into all columns of table\_name. It's handy when inserting records where default values are defined for columns. |
| Conditional Insertion | How to insert data into a table based on a condition? | INSERT INTO table\_name (column1, column2, ...) SELECT value1, value2, ... FROM other\_table WHERE condition; | This allows inserting data into table\_name based on a condition specified in the SELECT query. It's useful for selective data insertion. |

# SQL - UPDATE Query

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

The ALTER command is used to modify the structure of a table or a database object, such as adding, deleting, or modifying columns, constraints, indexes, etc.

UPDATE table\_name

SET column1 = value1, column2 = value2,..., columnN = valueN

WHERE [condition];

| **Operation** | **Query Question** | **Query Answer Example** | **Explanation** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| Basic Update | How to update values in a table for specific rows? | UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition; | This basic form updates specified columns in table\_name for rows that match the WHERE condition. It's used for general updates based on criteria. |

|  |  |  |  |
| --- | --- | --- | --- |
| Update Using Subquery | How to update values in a table based on a subquery result? | UPDATE table\_name SET column1 = (SELECT value FROM other\_table WHERE condition) WHERE condition; | This operation updates column1 in table\_name using a subquery result from other\_table. It's useful for complex updates derived from another query. |

|  |  |  |  |
| --- | --- | --- | --- |
| Update with Joins | How to update values in a table using joins with another table? | UPDATE table1 SET table1.column = value FROM table1 JOIN table2 ON table1.key = table2.key WHERE condition; | This allows updating table1 based on conditions involving a join with table2. It's effective for updating based on related data in different tables. |

|  |  |  |  |
| --- | --- | --- | --- |
| Update with Case Statement | How to conditionally update values in a table using CASE statements? | UPDATE table\_name SET column1 = CASE WHEN condition1 THEN value1 WHEN condition2 THEN value2 ELSE default\_value END WHERE condition; | This operation updates column1 in table\_name based on specified conditions using CASE statements. It's used for conditional updates within a single query. |

|  |  |  |  |
| --- | --- | --- | --- |
| Updating Multiple Columns | How to update multiple columns in a table simultaneously? | UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition; | This updates multiple columns (column1, column2, etc.) in table\_name for rows matching the WHERE condition. It's efficient for updating several columns at once. |

**=============DELETE==========**

DELETE FROM CUSTOMERS WHERE AGE > 25;

DELETE FROM CUSTOMERS;//Deleting All The Records From a Table

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

DELETE CUSTOMERS, ORDERS FROM CUSTOMERS

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

WHERE CUSTOMERS.SALARY > 2000;

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic Delete | How to delete specific rows from a table? | DELETE FROM table\_name WHERE condition; |
| Delete All Rows | How to delete all rows from a table? | DELETE FROM table\_name; |
| Delete Using Subquery | How to delete rows based on a subquery result? | DELETE FROM table\_name WHERE column IN (SELECT column FROM other\_table WHERE condition); |
| Delete with Joins | How to delete rows using joins with another table? | DELETE table1 FROM table1 JOIN table2 ON table1.key = table2.key WHERE condition; |

DELETE TOP (N) FROM table\_name;

**=================Views====================**

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

UPDATE CUSTOMERS\_VIEW

SET AGE = 35 WHERE name = 'Ramesh';

DROP VIEW view\_name;

DROP VIEW IF EXISTS view\_name;;

RENAME TABLE old\_view\_name To new\_view\_name;

**SQL - SORTING Results**

SELECT column-list

FROM table\_name

[WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

**SQL - WHERE Clause**

The SQL WHERE clause is used to filter records that meet a certain condition. It is used in various SQL statements to specify criteria that must be fulfilled for the records to be selected or affected. Here are some operations with the WHERE clause:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Simple WHERE | How to filter records based on a condition? | SELECT \* FROM employees WHERE salary > 50000; Retrieves employees with salaries greater than 50,000. |
| WHERE with AND | How to filter records with multiple conditions? | SELECT \* FROM employees WHERE salary > 50000 AND department = 'IT'; Retrieves IT employees with high salaries. |
| WHERE with OR | How to filter records with any of multiple conditions? | SELECT \* FROM employees WHERE department = 'IT' OR department = 'HR'; Retrieves employees in IT or HR. |
| WHERE with NOT | How to filter records excluding a condition? | SELECT \* FROM employees WHERE NOT department = 'HR'; Retrieves employees not in HR. |
| WHERE with BETWEEN | How to filter records within a range? | SELECT \* FROM employees WHERE salary BETWEEN 30000 AND 50000; Retrieves employees with salaries in the range. |
| WHERE with IN | How to filter records with values in a list? | SELECT \* FROM employees WHERE department IN ('IT', 'HR', 'Finance'); Retrieves employees in specified departments. |
| WHERE with LIKE | How to filter records matching a pattern? | SELECT \* FROM employees WHERE name LIKE 'A%'; Retrieves employees whose names start with 'A'. |
| WHERE with IS NULL | How to filter records with NULL values? | SELECT \* FROM employees WHERE manager\_id IS NULL; Retrieves employees without a manager. |
| WHERE with IS NOT NULL | How to filter records without NULL values? | SELECT \* FROM employees WHERE manager\_id IS NOT NULL; Retrieves employees with a manager. |
| WHERE with EXISTS | How to filter records based on a subquery result? | SELECT \* FROM employees e WHERE EXISTS (SELECT 1 FROM projects p WHERE p.employee\_id = e.id); Checks if employees have projects. |
| WHERE with Subquery | How to filter records based on another query? | SELECT \* FROM employees WHERE department\_id = (SELECT id FROM departments WHERE name = 'IT'); Retrieves IT employees. |

methods:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| WHERE with SELECT | How to select records based on a condition? | SELECT \* FROM employees WHERE salary > 50000; Retrieves employees with salaries greater than 50,000. |
| WHERE with UPDATE | How to update records based on a condition? | UPDATE employees SET salary = salary \* 1.1 WHERE department = 'IT'; Increases salary by 10% for IT employees. |
| WHERE with DELETE | How to delete records based on a condition? | DELETE FROM employees WHERE department = 'HR'; Deletes employees in the HR department. |
| WHERE with INSERT...SELECT | How to insert records from another table based on a condition? | INSERT INTO senior\_employees SELECT \* FROM employees WHERE salary > 70000; Inserts high salary employees into another table. |
| WHERE with JOIN | How to filter records in a join based on a condition? | SELECT e.name, d.name FROM employees e JOIN departments d ON e.department\_id = d.id WHERE d.name = 'IT'; Joins employees and departments, filtering for IT. |
| WHERE with GROUP BY | How to group records and apply a condition? | SELECT department, AVG(salary) FROM employees WHERE salary > 30000 GROUP BY department; Groups by department and filters by salary. |
| WHERE with HAVING | How to apply a condition on aggregated data? | SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 40000; Filters groups by average salary. |
| WHERE with ORDER BY | How to order filtered records? | SELECT \* FROM employees WHERE department = 'IT' ORDER BY salary DESC; Orders IT employees by salary in descending order. |
| WHERE with DISTINCT | How to select distinct records based on a condition? | SELECT DISTINCT department FROM employees WHERE salary > 50000; Retrieves distinct departments with high salaries. |
| WHERE with LIMIT | How to limit the number of filtered records? | SELECT \* FROM employees WHERE department = 'IT' LIMIT 10; Retrieves the first 10 IT employees. |
| WHERE with COUNT | How to count records based on a condition? | SELECT COUNT(\*) FROM employees WHERE department = 'HR'; Counts the number of HR employees. |
| WHERE with SUM | How to sum a column based on a condition? | SELECT SUM(salary) FROM employees WHERE department = 'IT'; Sums the salaries of IT employees. |
| WHERE with AVG | How to average a column based on a condition? | SELECT AVG(salary) FROM employees WHERE department = 'Finance'; Averages the salaries of Finance employees. |
| WHERE with MIN | How to find the minimum value of a column based on a condition? | SELECT MIN(salary) FROM employees WHERE department = 'Marketing'; Finds the minimum salary in Marketing. |
| WHERE with MAX | How to find the maximum value of a column based on a condition? | SELECT MAX(salary) FROM employees WHERE department = 'Sales'; Finds the maximum salary in Sales. |
| WHERE with UNION | How to filter records in a UNION operation? | SELECT name FROM employees WHERE department = 'IT' UNION SELECT name FROM managers WHERE department = 'IT'; Combines IT employees and managers. |
| WHERE with INTERSECT | How to filter records in an INTERSECT operation? | SELECT name FROM employees WHERE department = 'IT' INTERSECT SELECT name FROM managers WHERE department = 'IT'; Finds common IT employees and managers. |
| WHERE with EXCEPT | How to filter records in an EXCEPT operation? | SELECT name FROM employees WHERE department = 'IT' EXCEPT SELECT name FROM managers WHERE department = 'IT'; Finds IT employees not managers. |
| WHERE with EXISTS | How to check existence of records based on a condition? | SELECT \* FROM employees e WHERE EXISTS (SELECT 1 FROM projects p WHERE p.employee\_id = e.id); Checks if employees have projects. |
| WHERE with SUBQUERY | How to filter records based on a subquery? | SELECT \* FROM employees WHERE department\_id = (SELECT id FROM departments WHERE name = 'IT'); Retrieves IT employees. |

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Ascending Order | How to sort data in ascending order? | SELECT \* FROM table\_name ORDER BY column\_name ASC; |
| Descending Order | How to sort data in descending order? | SELECT \* FROM table\_name ORDER BY column\_name DESC; |
| Multiple Columns | How to sort data using multiple columns? | SELECT \* FROM table\_name ORDER BY column1 ASC, column2 DESC; |
| Expression Ordering | How to sort data using expressions? | SELECT \* FROM table\_name ORDER BY LENGTH(column\_name) DESC; |
| NULLs Handling | How to handle NULL values during sorting? | SELECT \* FROM table\_name ORDER BY column\_name ASC NULLS LAST; |

**Explanation:**

* **Multiple Columns:** Allows sorting by multiple columns (column1 and column2), specifying the sort order (ASC for ascending and DESC for descending) for each column.
* **Expression Ordering:** Sorts data based on an expression (LENGTH(column\_name)), ordering results by the length of column\_name in descending order.
* **NULLs Handling:** Controls how NULL values are treated during sorting (NULLS LAST ensures NULLs appear after non-NULL values in ascending order; NULLS FIRST places them before non-NULL values).

| **Operation** | **Query Question** | **Query Answer Example** |
| --- | --- | --- |
| Basic View Creation | How to create a basic view? | CREATE VIEW view\_name AS SELECT column1, column2 FROM table\_name; |
| View with Aliases | How to create a view with column aliases? | CREATE VIEW view\_name AS SELECT column1 AS alias1, column2 AS alias2 FROM table\_name; |
| View with WHERE Clause | How to create a view with a WHERE clause? | CREATE VIEW view\_name AS SELECT \* FROM table\_name WHERE condition; |
| View with JOINs | How to create a view with JOIN operations? | CREATE VIEW view\_name AS SELECT t1.column1, t2.column2 FROM table1 t1 JOIN table2 t2 ON t1.id = t2.id; |
| View with Aggregation | How to create a view with aggregated data? | CREATE VIEW view\_name AS SELECT category, SUM(amount) AS total\_amount FROM sales GROUP BY category; |
| View with Subquery | How to create a view using a subquery? | CREATE VIEW view\_name AS SELECT \* FROM (SELECT column1, column2 FROM table\_name WHERE condition) AS subquery; |

The WITH CHECK OPTION Clause

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERT statements satisfy the condition(s) specified by the WHERE clause.

If they do not satisfy the condition(s), the UPDATE or INSERT statements return an error. The following example creates the view named BUYERS\_VIEW with the WITH CHECK OPTION clause.

CREATE VIEW MY\_VIEW AS

SELECT name, age

FROM CUSTOMERS

WHERE age >= 25

WITH CHECK OPTION;

The WITH CHECK OPTION in this case should deny the entry and updates of the of records whose age value is greater than or equal to 25.

Here is a list of different operations for the SQL - UNION Operator:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic UNION | How to combine results from two SELECT queries? | SELECT column1, column2 FROM table1 UNION SELECT column1, column2 FROM table2; Combines and removes duplicates from both queries. |
| UNION with ALL | How to combine results from two SELECT queries and keep duplicates? | SELECT column1, column2 FROM table1 UNION ALL SELECT column1, column2 FROM table2; Keeps duplicates in the combined results. |
| UNION with ORDER BY | How to sort the combined results of a UNION query? | SELECT column1 FROM table1 UNION SELECT column1 FROM table2 ORDER BY column1; Orders the combined results by column1. |
| UNION with different columns | How to use UNION with different columns and aliases? | SELECT column1 AS col FROM table1 UNION SELECT column2 AS col FROM table2; Combines results with columns having different names. |
| UNION with WHERE clause | How to apply a WHERE clause in UNION queries? | SELECT column1 FROM table1 WHERE condition UNION SELECT column1 FROM table2 WHERE condition; Applies conditions to both queries. |
| UNION with JOINs | How to use UNION with JOINs in SELECT queries? | SELECT a.column1, b.column2 FROM table1 a JOIN table2 b ON a.id = b.id UNION SELECT c.column1, d.column2 FROM table3 c JOIN table4 d ON c.id = d.id; Combines results of JOINs from different tables. |

Here are the lists of different operations for the SQL - INTERSECT and SQL - EXCEPT operators:

**SQL - INTERSECT Operator**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic INTERSECT | How to find common rows between two SELECT queries? | SELECT column1, column2 FROM table1 INTERSECT SELECT column1, column2 FROM table2; Finds rows common to both queries. |
| INTERSECT with WHERE clause | How to apply WHERE clauses in INTERSECT queries? | SELECT column1 FROM table1 WHERE condition1 INTERSECT SELECT column1 FROM table2 WHERE condition2; Applies conditions to both queries. |
| INTERSECT with JOINs | How to use INTERSECT with JOINs in SELECT queries? | SELECT a.column1, b.column2 FROM table1 a JOIN table2 b ON a.id = b.id INTERSECT SELECT c.column1, d.column2 FROM table3 c JOIN table4 d ON c.id = d.id; Finds common rows from JOINs. |
| INTERSECT with ORDER BY | How to sort the results of an INTERSECT query? | SELECT column1 FROM table1 INTERSECT SELECT column1 FROM table2 ORDER BY column1; Orders the common rows by column1. |
| INTERSECT with different columns | How to use INTERSECT with different columns and aliases? | SELECT column1 AS col FROM table1 INTERSECT SELECT column2 AS col FROM table2; Finds common rows with columns having different names. |

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| INTERSECT with SELECT | How to find common records in two SELECT queries? | SELECT name FROM employees WHERE department = 'IT' INTERSECT SELECT name FROM managers WHERE department = 'IT'; Finds common names in both employees and managers who are in the IT department. |
| INTERSECT with WHERE | How to apply a condition to both sets in INTERSECT? | SELECT id FROM employees WHERE salary > 50000 INTERSECT SELECT id FROM projects WHERE budget > 10000; Finds common IDs in employees with high salary and projects with high budget. |
| INTERSECT with ORDER BY | How to order results of an INTERSECT operation? | SELECT name FROM employees WHERE department = 'IT' INTERSECT SELECT name FROM managers WHERE department = 'IT' ORDER BY name; Finds and orders common names in IT employees and managers. |
| INTERSECT with GROUP BY | How to group results before performing an INTERSECT? | SELECT department, COUNT(\*) FROM employees GROUP BY department INTERSECT SELECT department, COUNT(\*) FROM managers GROUP BY department; Finds common departments and counts in employees and managers. |
| INTERSECT with HAVING | How to apply a HAVING clause in an INTERSECT operation? | SELECT department FROM employees GROUP BY department HAVING AVG(salary) > 50000 INTERSECT SELECT department FROM managers GROUP BY department HAVING AVG(salary) > 60000; Finds common departments with high average salary. |
| INTERSECT with JOIN | How to join tables before performing an INTERSECT? | SELECT e.name FROM employees e JOIN departments d ON e.department\_id = d.id WHERE d.name = 'IT' INTERSECT SELECT m.name FROM managers m JOIN departments d ON m.department\_id = d.id WHERE d.name = 'IT'; Finds common names in IT employees and managers using joins. |
| INTERSECT with DISTINCT | How to find distinct common records in two SELECT queries? | SELECT DISTINCT name FROM employees INTERSECT SELECT DISTINCT name FROM managers; Finds distinct common names in employees and managers. |
| INTERSECT with LIMIT | How to limit the number of results in an INTERSECT operation? | SELECT name FROM employees INTERSECT SELECT name FROM managers LIMIT 10; Finds common names in employees and managers, limited to 10 results. |
| INTERSECT with UNION | How to combine results using UNION and INTERSECT? | (SELECT name FROM employees UNION SELECT name FROM contractors) INTERSECT SELECT name FROM managers; Finds common names in the combined employees/contractors and managers. |
| INTERSECT with SUBQUERY | How to use a subquery in an INTERSECT operation? | SELECT name FROM employees WHERE id IN (SELECT employee\_id FROM projects) INTERSECT SELECT name FROM managers WHERE id IN (SELECT manager\_id FROM projects); Finds common names in employees and managers involved in projects. |
| INTERSECT with EXISTS | How to use EXISTS clause in an INTERSECT operation? | SELECT name FROM employees WHERE EXISTS (SELECT 1 FROM projects WHERE employee\_id = employees.id) INTERSECT SELECT name FROM managers WHERE EXISTS (SELECT 1 FROM projects WHERE manager\_id = managers.id); Finds common names in employees and managers who are in projects. |

Here is a comprehensive list of SQL INTERSECT operations combined with various other operators:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| INTERSECT with = | How to find common records where specific columns are equal? | SELECT name FROM employees WHERE department = 'IT' INTERSECT SELECT name FROM managers WHERE department = 'IT'; Finds common names in both employees and managers who are in the IT department. |
| INTERSECT with <> | How to find common records where specific columns are not equal? | SELECT id FROM employees WHERE salary <> 50000 INTERSECT SELECT id FROM projects WHERE budget <> 10000; Finds common IDs in employees with different salary and projects with different budget. |
| INTERSECT with > | How to find common records where specific columns are greater? | SELECT id FROM employees WHERE salary > 50000 INTERSECT SELECT id FROM projects WHERE budget > 10000; Finds common IDs in employees with high salary and projects with high budget. |
| INTERSECT with < | How to find common records where specific columns are less? | SELECT id FROM employees WHERE salary < 50000 INTERSECT SELECT id FROM projects WHERE budget < 10000; Finds common IDs in employees with low salary and projects with low budget. |
| INTERSECT with >= | How to find common records where specific columns are greater or equal? | SELECT id FROM employees WHERE salary >= 50000 INTERSECT SELECT id FROM projects WHERE budget >= 10000; Finds common IDs in employees with salary greater or equal and projects with budget greater or equal. |
| INTERSECT with <= | How to find common records where specific columns are less or equal? | SELECT id FROM employees WHERE salary <= 50000 INTERSECT SELECT id FROM projects WHERE budget <= 10000; Finds common IDs in employees with salary less or equal and projects with budget less or equal. |
| INTERSECT with BETWEEN | How to find common records within a range of values? | SELECT id FROM employees WHERE salary BETWEEN 30000 AND 50000 INTERSECT SELECT id FROM projects WHERE budget BETWEEN 10000 AND 30000; Finds common IDs in employees with salary in range and projects with budget in range. |
| INTERSECT with IN | How to find common records from a set of values? | SELECT id FROM employees WHERE department IN ('IT', 'HR') INTERSECT SELECT id FROM projects WHERE project\_code IN ('A', 'B'); Finds common IDs in employees from specific departments and projects with specific codes. |
| INTERSECT with LIKE | How to find common records matching a pattern? | SELECT name FROM employees WHERE name LIKE 'A%' INTERSECT SELECT name FROM managers WHERE name LIKE 'A%'; Finds common names in employees and managers starting with 'A'. |
| INTERSECT with IS NULL | How to find common records where a column is NULL? | SELECT id FROM employees WHERE manager\_id IS NULL INTERSECT SELECT id FROM projects WHERE lead\_id IS NULL; Finds common IDs in employees and projects where the respective IDs are NULL. |
| INTERSECT with IS NOT NULL | How to find common records where a column is not NULL? | SELECT id FROM employees WHERE manager\_id IS NOT NULL INTERSECT SELECT id FROM projects WHERE lead\_id IS NOT NULL; Finds common IDs in employees and projects where the respective IDs are not NULL. |
| INTERSECT with EXISTS | How to find common records where a subquery returns a result? | SELECT id FROM employees WHERE EXISTS (SELECT 1 FROM departments WHERE employees.department\_id = departments.id) INTERSECT SELECT id FROM projects WHERE EXISTS (SELECT 1 FROM departments WHERE projects.department\_id = departments.id); Finds common IDs in employees and projects with existing department records. |

**SQL - EXCEPT Operator**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic EXCEPT | How to find rows in the first query but not in the second? | SELECT column1, column2 FROM table1 EXCEPT SELECT column1, column2 FROM table2; Finds rows in table1 not in table2. |
| EXCEPT with WHERE clause | How to apply WHERE clauses in EXCEPT queries? | SELECT column1 FROM table1 WHERE condition1 EXCEPT SELECT column1 FROM table2 WHERE condition2; Applies conditions to both queries. |
| EXCEPT with JOINs | How to use EXCEPT with JOINs in SELECT queries? | SELECT a.column1, b.column2 FROM table1 a JOIN table2 b ON a.id = b.id EXCEPT SELECT c.column1, d.column2 FROM table3 c JOIN table4 d ON c.id = d.id; Finds rows from JOINs in the first query not in the second. |
| EXCEPT with ORDER BY | How to sort the results of an EXCEPT query? | SELECT column1 FROM table1 EXCEPT SELECT column1 FROM table2 ORDER BY column1; Orders the rows found only in table1 by column1. |
| EXCEPT with different columns | How to use EXCEPT with different columns and aliases? | SELECT column1 AS col FROM table1 EXCEPT SELECT column2 AS col FROM table2; Finds rows in table1 not in table2 with columns having different names. |

**SQL Indexes Operations**

**SQL - Indexes**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Basic Index | What is an index in SQL? | An index is a database object that improves the speed of data retrieval. |

**SQL - Create Index**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Create Basic Index | How to create an index on a table? | CREATE INDEX idx\_name ON table\_name (column1); Creates an index on column1 of table\_name. |
| Create Composite Index | How to create a composite index? | CREATE INDEX idx\_name ON table\_name (column1, column2); Creates an index on column1 and column2. |
| Create Unique Index | How to create a unique index? | CREATE UNIQUE INDEX idx\_name ON table\_name (column1); Creates a unique index ensuring all values are unique. |
| Create Clustered Index | How to create a clustered index? | CREATE CLUSTERED INDEX idx\_name ON table\_name (column1); Creates a clustered index on column1. |
| Create Non-Clustered Index | How to create a non-clustered index? | CREATE NONCLUSTERED INDEX idx\_name ON table\_name (column1); Creates a non-clustered index on column1. |

**SQL - Drop Index**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Drop Index | How to drop an index? | DROP INDEX idx\_name; Drops the index named idx\_name. |
| Drop Index from Table | How to drop an index from a table? | DROP INDEX idx\_name ON table\_name; Drops the index idx\_name from table\_name. |

**SQL - Show Indexes**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Show Indexes | How to show all indexes on a table? | SHOW INDEXES FROM table\_name; Shows all indexes for table\_name. |
| Show Index Details | How to get details of indexes? | SHOW INDEX FROM table\_name; Provides details like index name, column name, and type for table\_name. |

**SQL - Unique Index**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Create Unique Index | How to create a unique index? | CREATE UNIQUE INDEX idx\_name ON table\_name (column1); Ensures all values in column1 are unique. |

**SQL - Clustered Index**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Create Clustered Index | How to create a clustered index? | CREATE CLUSTERED INDEX idx\_name ON table\_name (column1); Orders table data by column1. |

**SQL - Non-Clustered Index**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Create Non-Clustered Index | How to create a non-clustered index? | CREATE NONCLUSTERED INDEX idx\_name ON table\_name (column1); Creates an index that doesn't alter table order. |

**. INNER JOIN**

Combines rows that have matching values in both tables.

**Example**

SELECT employees.name, departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.id;

**Explanation:** Selects employees and their respective department names where there is a match in the department\_id.

**2. LEFT (OUTER) JOIN**

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

**Example**

SELECT employees.name, departments.department\_name

FROM employees

LEFT JOIN departments ON employees.department\_id = departments.id;

**Explanation:** Selects all employees, with department names where available. If an employee doesn't belong to a department, NULL is returned for the department name.

**3. RIGHT (OUTER) JOIN**

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

**Example**

SELECT employees.name, departments.department\_name

FROM employees

RIGHT JOIN departments ON employees.department\_id = departments.id;

**Explanation:** Selects all departments, with employee names where available. If a department doesn't have any employees, NULL is returned for the employee name.

**4. FULL (OUTER) JOIN**

Returns rows when there is a match in one of the tables. It returns NULL values when there is no match.

**Example**

SELECT employees.name, departments.department\_name

FROM employees

FULL OUTER JOIN departments ON employees.department\_id = departments.id;

**Explanation:** Selects all employees and all departments. For unmatched rows, NULL values are returned.

**5. CROSS JOIN**

Returns the Cartesian product of the two tables. It combines all rows from the left table with all rows from the right table.

**Example**

SELECT employees.name, departments.department\_name

FROM employees

CROSS JOIN departments;

**Explanation:** Each employee is paired with each department, resulting in all possible combinations of employees and departments.

**6. SELF JOIN**

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

**Example**

SELECT A.name AS EmployeeName, B.name AS ManagerName

FROM employees A, employees B

WHERE A.manager\_id = B.id;

**Explanation:** Selects employees and their respective managers' names by joining the employees table with itself.

**7. NATURAL JOIN**

Automatically joins tables based on columns with the same name and compatible data types.

**Example**

SELECT employees.name, departments.department\_name

FROM employees

NATURAL JOIN departments;

**Explanation:** Joins employees and departments based on columns with the same name (e.g., department\_id), without explicitly specifying the condition.

**SQL DELETE... JOIN Clause**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| DELETE with INNER JOIN | How to delete rows using INNER JOIN? | DELETE t1 FROM table1 t1 INNER JOIN table2 t2 ON t1.id = t2.id WHERE t2.condition; Deletes rows in table1 based on a condition in table2. |
| DELETE with LEFT JOIN | How to delete rows using LEFT JOIN? | DELETE t1 FROM table1 t1 LEFT JOIN table2 t2 ON t1.id = t2.id WHERE t2.id IS NULL; Deletes rows in table1 that have no match in table2. |
| DELETE with RIGHT JOIN | How to delete rows using RIGHT JOIN? | DELETE t1 FROM table1 t1 RIGHT JOIN table2 t2 ON t1.id = t2.id WHERE t1.id IS NULL; Deletes rows in table2 that have no match in table1. |

|  |  |  |
| --- | --- | --- |
| **Join Type** | **Description** | **Example Query Snippet** |
| INNER JOIN | Combines rows with matching values in both tables. | INNER JOIN |
| LEFT (OUTER) JOIN | Returns all rows from the left table, matched rows from the right table, and NULLs for no match. | LEFT JOIN |
| RIGHT (OUTER) JOIN | Returns all rows from the right table, matched rows from the left table, and NULLs for no match. | RIGHT JOIN |
| FULL (OUTER) JOIN | Returns rows when there is a match in one of the tables, with NULLs for no match. | FULL OUTER JOIN |
| CROSS JOIN | Returns the Cartesian product of both tables. | CROSS JOIN |
| SELF JOIN | Joins the table with itself. | Self Join Example |
| NATURAL JOIN | Automatically joins tables based on columns with the same name. | NATURAL JOIN |

Example: Remove products that haven’t sold (using a LEFT JOIN):

DELETE P

FROM Product P

LEFT JOIN OrderItem I ON P.Id = I.ProductId

WHERE I.Id IS NULL;

UPDATE CUSTOMERS

JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

SET CUSTOMERS.SALARY = CUSTOMERS.SALARY + 1000,

ORDERS.AMOUNT = ORDERS.AMOUNT + 500;

**Union -longitudal**

SELECT \* FROM COURSES\_PICKED

UNION

SELECT \* FROM EXTRA\_COURSES\_PICKED;

Join-latitudal

**SQL Function Reference**

**SQL - Date Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| CURRENT\_DATE() | How to get the current date? | SELECT CURRENT\_DATE(); Retrieves the current date. |
| DATE\_ADD() | How to add days to a date? | SELECT DATE\_ADD('2024-01-01', INTERVAL 5 DAY); Adds 5 days to the date '2024-01-01'. |
| DATEDIFF() | How to calculate the difference between two dates? | SELECT DATEDIFF('2024-01-10', '2024-01-01'); Calculates the difference in days between two dates. |
| NOW() | How to get the current date and time? | SELECT NOW(); Retrieves the current date and time. |

**SQL - String Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| CONCAT() | How to concatenate strings? | SELECT CONCAT('Hello, ', 'world!'); Concatenates 'Hello, ' and 'world!'. |
| LENGTH() | How to get the length of a string? | SELECT LENGTH('Hello'); Returns the length of 'Hello'. |
| LOWER() | How to convert a string to lowercase? | SELECT LOWER('HELLO'); Converts 'HELLO' to 'hello'. |
| SUBSTRING() | How to extract a substring from a string? | SELECT SUBSTRING('Hello, world!', 1, 5); Extracts 'Hello' from 'Hello, world!'. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Syntax** | **Example** | **Output** | **Explanation** |
| CONCAT() | CONCAT(string1, string2, ...) | SELECT CONCAT('Hello ', 'World'); | Hello World | Concatenates two or more strings together. |
| SUBSTRING() | SUBSTRING(string FROM start [FOR length]) | SELECT SUBSTRING('Hello World' FROM 1 FOR 5); | Hello | Extracts a substring from a string. |
| LENGTH() | LENGTH(string) | SELECT LENGTH('Hello'); | 5 | Returns the length of the string. |
| TRIM() | TRIM([{BOTH | LEADING | TRAILING} [removal\_string] FROM] string) | SELECT TRIM(' Hello '); |
| UPPER() | UPPER(string) | SELECT UPPER('hello'); | HELLO | Converts a string to uppercase. |
| LOWER() | LOWER(string) | SELECT LOWER('HELLO'); | hello | Converts a string to lowercase. |
| REPLACE() | REPLACE(string, search\_string, replacement\_string) | SELECT REPLACE('Hello World', 'World', 'Everyone'); | Hello Everyone | Replaces occurrences of a substring in a string. |
| LEFT() | LEFT(string, length) | SELECT LEFT('Hello', 3); | Hel | Extracts a given number of characters from the left. |
| RIGHT() | RIGHT(string, length) | SELECT RIGHT('Hello', 3); | llo | Extracts a given number of characters from the right. |
| INSTR() | INSTR(string, substring) | SELECT INSTR('Hello World', 'World'); | 7 | Returns the position of the first occurrence of a substring. |
| LPAD() | LPAD(string, length, pad\_string) | SELECT LPAD('Hello', 10, '-'); | -----Hello | Pads a string on the left with a pad string up to a certain length. |
| RPAD() | RPAD(string, length, pad\_string) | SELECT RPAD('Hello', 10, '-'); | Hello----- | Pads a string on the right with a pad string up to a certain length. |

**Explanation Notes:**

* **CONCAT():** Combines multiple strings into one.

**SUBSTRING():** Extracts a portion of a string based on given start position and optionally a

**SQL - Aggregate Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| AVG() | How to calculate the average of a column? | SELECT AVG(salary) FROM employees; Calculates the average salary from the employees table. |
| COUNT() | How to count the number of rows? | SELECT COUNT(\*) FROM employees; Counts the number of rows in the employees table. |
| MAX() | How to find the maximum value in a column? | SELECT MAX(salary) FROM employees; Finds the maximum salary in the employees table. |
| SUM() | How to calculate the sum of a column? | SELECT SUM(salary) FROM employees; Calculates the sum of all salaries in the employees table. |

**SQL - Numeric Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| ABS() | How to get the absolute value of a number? | SELECT ABS(-10); Returns 10. |
| CEIL() | How to get the smallest integer greater than or equal to a number? | SELECT CEIL(4.2); Returns 5. |
| FLOOR() | How to get the largest integer less than or equal to a number? | SELECT FLOOR(4.7); Returns 4. |
| ROUND() | How to round a number to a specified number of decimal places? | SELECT ROUND(123.456, 2); Rounds 123.456 to 2 decimal places, returning 123.46. |

**SQL - Text & Image Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| CHARINDEX() | How to find the position of a substring in a string? | SELECT CHARINDEX('world', 'Hello, world!'); Returns 8. |
| PATINDEX() | How to find the position of a pattern in a string? | SELECT PATINDEX('%world%', 'Hello, world!'); Returns 8. |

**SQL - Statistical Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| VARIANCE() | How to calculate the variance of a set of values? | SELECT VARIANCE(salary) FROM employees; Calculates the variance of the salary column in the employees table. |
| STDDEV() | How to calculate the standard deviation of a set of values? | SELECT STDDEV(salary) FROM employees; Calculates the standard deviation of the salary column in the employees table. |

**SQL - Logical Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| COALESCE() | How to return the first non-null value in a list? | SELECT COALESCE(NULL, NULL, 'Hello'); Returns 'Hello'. |
| NULLIF() | How to return NULL if two expressions are equal? | SELECT NULLIF(10, 10); Returns NULL. |

**SQL - Cursor Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| CURSOR\_STATUS() | How to check the status of a cursor? | SELECT CURSOR\_STATUS('global', 'myCursor'); Returns the status of the cursor 'myCursor'. |

**SQL - JSON Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| JSON\_VALUE() | How to extract a value from a JSON string? | SELECT JSON\_VALUE('{"name": "John"}', '$.name'); Returns 'John'. |
| JSON\_QUERY() | How to extract an object or array from a JSON string? | SELECT JSON\_QUERY('{"name": "John", "info": {"age": 30}}', '$.info'); Returns '{"age": 30}'. |

**SQL - Conversion Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| CAST() | How to convert a value from one data type to another? | SELECT CAST('123' AS INT); Converts the string '123' to an integer. |
| CONVERT() | How to convert a value from one data type to another with formatting? | SELECT CONVERT(VARCHAR, GETDATE(), 103); Converts the current date to a string in DD/MM/YYYY format. |

**SQL - Datatype Functions**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| SQL\_VARIANT\_PROPERTY() | How to return the properties of a SQL variant value? | SELECT SQL\_VARIANT\_PROPERTY(myVariant, 'BaseType'); Returns the base data type of the SQL variant 'myVariant'. |

# **SQL - Wildcards**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Query Question** | **Query Answer Example** |
| Using % wildcard | How to match any sequence of characters? | SELECT \* FROM employees WHERE name LIKE 'J%'; |
| Using \_ wildcard | How to match a single character? | SELECT \* FROM employees WHERE name LIKE 'J\_n'; |
| Using [ ] wildcard | How to match any character in a set? | SELECT \* FROM employees WHERE name LIKE '[JM]ack'; |
| Using [^] wildcard | How to exclude characters in a set? | SELECT \* FROM employees WHERE name LIKE '[^JM]ack'; |
| Using % and \_ together | How to match a pattern with both wildcards? | SELECT \* FROM employees WHERE name LIKE 'J\_n%'; |
| Using escape character | How to search for wildcard characters? | SELECT \* FROM employees WHERE name LIKE '10\% off' ESCAPE '\'; |

# **Explanation**

# **% Wildcard**: Matches any sequence of characters (including zero characters).

# **\_ Wildcard**: Matches a single character.

# **[ ] Wildcard**: Matches any one of the characters within the brackets.

# **[^] Wildcard**: Matches any single character not within the brackets.

# **Escape Character**: Allows searching for wildcard characters by preceding them with an escape character.

-- This is a single-line comment

/\* This is a

multi-line

comment \*/

# SQL - UPDATE View== SQL - UPDATE Table

DROP VIEW view\_name;

========================================================================

---------------------TOP CLAUSE-------------

SELECT TOP 4 \* FROM CUSTOMERS ORDER BY SALARY DESC;

SELECT TOP 40 PERCENT \* FROM CUSTOMERS ORDER BY SALARY

---------------------DistinctE-------------

In the following query, we are retrieving a list of all unique combinations of customer's age and salary using the DISTINCT keyword −

SELECT DISTINCT AGE, SALARY FROM CUSTOMERS ORDER BY AGE;

-------------------------ORDER BY-----------------

SELECT column1, column2, ...

FROM table\_name

ORDER BY column\_name1 [ASC | DESC], column\_name2 [ASC | DESC], ...

LIMIT N;

In SQL, when there are NULL values in the column, DISTINCT treats them as unique values and includes them in the result set.

!< Not less than 4 !< 5 returns FALSE

!> Not greater than 4 !> 5 returns TRUE

UPDATE CUSTOMERS

SET SALARY = SALARY \* 0.25

WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP WHERE AGE >= 27 );

DELETE FROM CUSTOMERS

WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP WHERE AGE >= 27 );

SELECT ADDRESS, AGE, SUM(SALARY) AS TOTAL\_SALARY

FROM CUSTOMERS GROUP BY ADDRESS, AGE;

SELECT AGE, MIN(SALARY) AS MIN\_SALARY

FROM CUSTOMERS

GROUP BY AGE ORDER BY MIN\_SALARY DESC;

SELECT department, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department

HAVING AVG(salary) > 50000;

INSERT INTO CUSTOMERS\_BKP

SELECT \* FROM CUSTOMERS

WHERE ID IN (SELECT ID FROM CUSTOMERS);

CREATE TABLE NEW\_TABLE\_NAME AS

SELECT [column1, column2...columnN]

FROM EXISTING\_TABLE\_NAME

WHERE Condition;

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

CREATE TABLE NEW\_CUSTOMERS SELECT \* FROM CUSTOMERS;

REATE TABLE new\_table LIKE original\_table;

INSERT INTO new\_table SELECT \* FROM original\_table;

INSERT INTO BUYERS (ID, NAME, AGE)

SELECT ID, NAME, AGE FROM CUSTOMERS;

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

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

SELECT \* FROM Customers;

SELECT CustomerName, City FROM Customers;

SELECT DISTINCT Country FROM Customers;

SELECT COUNT(DISTINCT Country) FROM Customers;

SELECT \* FROM Customers

WHERE Country='Mexico';

SELECT \* FROM Customers

WHERE CustomerID > 80;

= Equal

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal. Note: In some versions of SQL this operator may be written as !=

BETWEEN Between a certain range

LIKE Search for a pattern

IN To specify multiple possible values for a column

SELECT \* FROM Products

ORDER BY Price;

SELECT \* FROM Products

ORDER BY ProductName DESC;

SELECT \* FROM Customers

ORDER BY Country, CustomerName;

SELECT \* FROM Customers

ORDER BY Country ASC, CustomerName DESC;

Select all customers from Spain that starts with the letter 'G':

SELECT \*

FROM Customers

WHERE Country = 'Spain' AND CustomerName LIKE 'G%';

selects all fields from Customers where Country is "Germany" AND City is "Berlin" AND PostalCode is higher than 12000:

Example

SELECT \* FROM Customers

WHERE Country = 'Germany'

AND City = 'Berlin'

AND PostalCode > 12000;

Select all Spanish customers that starts with either "G" or "R":

SELECT \* FROM Customers

WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');

Select all customers that either:

are from Spain and starts with either "G", or

starts with the letter "R":

SELECT \* FROM Customers

WHERE Country = 'Spain' AND CustomerName LIKE 'G%' OR CustomerName LIKE 'R%';

Select all customers from Germany or Spain:

SELECT \*

FROM Customers

WHERE Country = 'Germany' OR Country = 'Spain';

selects all fields from Customers where either City is "Berlin", CustomerName starts with the letter "G" or Country is "Norway":

Example

SELECT \* FROM Customers

WHERE City = 'Berlin' OR CustomerName LIKE 'G%' OR Country = 'Norway';

Select all Spanish customers that starts with either "G" or "R":

SELECT \* FROM Customers

WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');

Select only the customers that are NOT from Spain:

SELECT \* FROM Customers

WHERE NOT Country = 'Spain';

Select customers that does not start with the letter 'A':

SELECT \* FROM Customers

WHERE CustomerName NOT LIKE 'A%';

Select customers with a customerID not between 10 and 60:

SELECT \* FROM Customers

WHERE CustomerID NOT BETWEEN 10 AND 60;

Select customers that are not from Paris or London:

SELECT \* FROM Customers

WHERE City NOT IN ('Paris', 'London');

Select customers with a CustomerId not greater than 50:

SELECT \* FROM Customers

WHERE NOT CustomerID > 50;

The following SQL statement inserts a new record in the "Customers" table:

ExampleGet your own SQL Server

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

The IS NULL Operator

The IS NULL operator is used to test for empty values (NULL values).

The following SQL lists all customers with a NULL value in the "Address" field:

ExampleGet your own SQL Server

SELECT CustomerName, ContactName, Address

FROM Customers

WHERE Address IS NULL;

UPDATE Customers

SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'

WHERE CustomerID = 1;

Select only the first 3 records of the Customers table:

SELECT TOP 3 \* FROM Customers;

SELECT \* FROM Customers

LIMIT 3;

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

DELETE FROM Customers

ORDER BY some\_column

LIMIT 3;

SELECT \* FROM Customers

FETCH FIRST 3 ROWS ONLY;

SELECT TOP 50 PERCENT \* FROM Customers;

SELECT \* FROM Customers

FETCH FIRST 50 PERCENT ROWS ONLY;

SELECT TOP 3 \* FROM Customers

WHERE Country='Germany';

SELECT \* FROM Customers

WHERE Country='Germany'

LIMIT 3;

SELECT \* FROM Customers

ORDER BY CustomerName DESC

LIMIT 3;

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

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()).

SELECT SUM(Quantity)

FROM OrderDetails;

SELECT SUM(Quantity)

FROM OrderDetails

WHERE ProductId = 11;

SELECT SUM(Quantity \* 10)

FROM OrderDetails;

SELECT AVG(Price)

FROM Products

WHERE CategoryID = 1;

\_ wildcard represents a single character.

% wildcard represents any number of characters, even zero characters.

Return all customers that contains the phrase 'or'

SELECT \* FROM Customers

WHERE CustomerName LIKE '%or%';

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 \*

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

SELECT \* FROM Customers

WHERE CustomerName LIKE '[bsp]%';

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

SELECT \* FROM Customers

WHERE CustomerName LIKE '[a-f]%';

If no wildcard is specified, the phrase has to have an exact match to return a result.

Example

Return all customers from Spain:

SELECT \* FROM Customers

WHERE Country LIKE 'Spain';

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

SELECT \* FROM Customers

WHERE CustomerID NOT IN (SELECT CustomerID FROM Orders);

SELECT \* FROM Customers

WHERE Country NOT IN ('Germany', 'France', 'UK');

SELECT \* FROM Products

WHERE Price BETWEEN 10 AND 20;

SELECT \* FROM Products

WHERE Price BETWEEN 10 AND 20

AND CategoryID IN (1,2,3);

SELECT \* FROM Products

WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'

ORDER BY ProductName;

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

JOINS

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

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders

INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

ADD Email varchar(255);

ALTER TABLE Persons

ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

The ALL command returns true if all of the subquery values meet the condition

SELECT ProductName

FROM Products

WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

Some of The Most Important SQL Commands

SELECT - extracts data from a database

UPDATE - updates data in a database

DELETE - deletes data from a database

INSERT INTO - inserts 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 a table

DROP TABLE - deletes a table

CREATE INDEX - creates an index (search key)

DROP INDEX - deletes an index

The **NOT NULL** constraint in SQL is used to ensure that a column in a table doesn't contain NULL (empty) values, and prevent any attempts to insert or update rows with NULL values.

CREATE TABLE table\_name (

column1 datatype NOT NULL,

);