WEB DEVELOPER

@CODERSS_WORLD

LEARN SQL

INTRODUCTION TO SQL:

SQL (Structured Query Language) is a standard programming language used for managing and manipulating relational databases. It enables users to retrieve, insert, update, and delete data in a structured format. SQL is essential for database management systems (DBMS) like MySQL, PostgreSQL, Oracle, and SQL Server.

KEYS IN SQL:

Keys are crucial in SQL for identifying records uniquely in a table and establishing relationships between tables.

1. Primary Key:

- Uniquely identifies each record in a table.
- Cannot have NULL values.
- **Example:** In a Students table, the StudentID can be a primary key to identify each student.

KEY COMPONENTS OF SQL:

Data Definition Language (DDL): Used to define database structures (tables, schemas, etc.). Examples: CREATE, ALTER, DROP.

Data Manipulation Language (DML): Used to manipulate data within tables. Examples: SELECT, INSERT, UPDATE, DELETE.

Data Control Language (DCL): Controls access to data. Examples: GRANT, REVOKE.

Transaction Control Language (TCL): Manages database transactions. Examples: COMMIT, ROLLBACK, SAVEPOINT.

2. Foreign Key:

- Establishes a link between two tables by referring to the primary key in another table.
- Helps maintain referential integrity between tables.
- Example: In a Courses table, StudentID can be a foreign key linking back to the Students table.

3. Unique Key:

- Ensures that all values in a column are unique but allows NULL values.
- Example: A PhoneNumber column may use a unique key to prevent duplicate phone numbers.

4. Composite Key:

- Combines two or more columns to create a unique identifier for rows.
- Example: A combination of CourseID and StudentID in an Enrollment table can serve as a composite key.

5. Candidate Key:

- A column or combination of columns that can qualify as a unique key.
- A table can have multiple candidate keys, but one will become the primary key.


3. RIGHT JOIN (or RIGHT OUTER JOIN):

- Returns all records from the right table and matching records from the left table. If no match exists, NULL is returned for left table columns.

- **Example:** Retrieve all courses and students, even if some courses don't have any students.

4. FULL JOIN (or FULL OUTER JOIN):

- Returns all records when there is a match in either table. If no match exists, NULL values are returned for the missing side.

- **Example:** Retrieve all students and all courses, even if they don't have any related data.

SQL JOINS:

1. INNER JOIN:

- Returns records with matching values in both tables.
- Example: If you join Students and Courses, only records with matching StudentID in both tables are retrieved.

2. LEFT JOIN (or LEFT OUTER JOIN):

- Returns all records from the left table and matching records from the right table. If no match exists, NULL is returned for right table columns.
- Example: Retrieve all students and their courses, even if some students aren't enrolled in any courses.

CROSS JOIN:

- Returns the Cartesian product of both tables, combining every row from the first table with every row from the second.

- **Example:** If Students has 10 rows and Courses has 5 rows, the result of a CROSS JOIN will be 50 rows.

These are the foundational concepts that make SQL essential for relational database management.

Vikram
@code._learning

100 SQL Commands

- **SELECT** - retrieves data from a database
- **INSERT** - inserts new data into a database
- **UPDATE** - updates existing data in a database
- **DELETE** - deletes data from a database
- **CREATE DATABASE** - creates a new database
- **CREATE TABLE** - creates a new table in a database
- **ALTER TABLE** - modifies an existing table structure

- **DROP TABLE** - deletes a table from a database
- **TRUNCATE TABLE** - removes all records from a table
- **CREATE INDEX** - creates an index on a table
- **DROP INDEX** - deletes an index from a table
- **JOIN** - combines rows from two or more tables based on a related column
- **INNER JOIN** - returns rows when there is a match in both tables
- **LEFT JOIN** - returns all rows from the left table, and the matched rows from the right table
- **RIGHT JOIN** - returns all rows from the right table, and the matched rows from the left table

- **FULL JOIN** - returns rows when there is a match in one of the tables
- **UNION** - combines the results of two or more SELECT statements
- **UNION ALL** - combines the results of two or more SELECT statements, including duplicates
- **GROUP BY** - groups rows that have the same values into summary rows
- **HAVING** - filters records based on a specified condition
- **ORDER BY** - sorts the result set in ascending or descending order
- **COUNT** - returns the number of rows that satisfy the condition
- **SUM** - calculates the sum of a set of values
- **AVG** - calculates the average of a set of values
- **MIN** - returns the smallest value in a set of values
- **MAX** - returns the largest value in a set of values
- **DISTINCT** - selects unique values from a column
- **WHERE** - filters records based on specified conditions

- **AND** - combines multiple conditions in a WHERE clause
- **OR** - specifies multiple alternative conditions in a WHERE clause
- **NOT** - negates a condition in a WHERE clause
- **BETWEEN** - selects values within a specified range
- **IN** - specifies multiple values for a column
- **LIKE** - selects rows that match a specified pattern
- **IS NULL** - checks for NULL values in a column
- **IS NOT NULL** - checks for non-NULL values in a column

- **EXISTS** - tests for the existence of any record in a subquery
- **CASE** - performs conditional logic in SQL statements
- **WHEN** - specifies conditions in a CASE statement
- **THEN** - specifies the result if a condition is true in a CASE statement
- **ELSE** - specifies the result if no condition is true in a CASE statement
- **END** - ends the CASE statement

- **RESTRICT** - restricts the deletion of a referenced record if there are related records
- **CASE WHEN** - conditional expression in SELECT statements
- **WITH** - defines a common table expression (CTE)
- **INTO** - specifies a target table for the result set of a SELECT statement
- **TOP** - limits the number of rows returned by a query
- **LIMIT** - limits the number of rows returned by a query (used in some SQL dialects)
- **OFFSET** - specifies the number of rows to skip before starting to return rows
- **FETCH** - retrieves rows from a result set one at a time
- **ROW_NUMBER()** - assigns a unique sequential integer to each row in a result set
- **RANK()** - assigns a unique rank to each row in a result set, with gaps in the ranking sequence possible
- **DENSE_RANK()** - assigns a unique rank to each row in a result set, with no gaps in the ranking sequence

- **NTILE()** - divides the result set into a specified number of equally sized groups
- **LEAD()** - retrieves the value from the next row in a result set
- **LAG()** - retrieves the value from the previous row in a result set
- **PARTITION BY** - divides the result set into partitions to which the window function is applied separately
- **ORDER BY** - specifies the order of rows within each partition for window functions
- **ROWS** - specifies the window frame for window functions

- **RANGE** - specifies the window frame based on values rather than rows for window functions
- **CURRENT_TIMESTAMP** - returns the current date and time
- **CURRENT_DATE** - returns the current date
- **CURRENT_TIME** - returns the current time
- **DATEADD** - adds a specified time interval to a date
- **DATEDIFF** - calculates the difference between two dates

- **UNPIVOT** - rotates a table-valued expression by turning multiple columns into unique rows in the output
- **COALESCE** - returns the first non-NULL expression in a list
- **NULLIF** - returns NULL if the two specified expressions are equal, otherwise returns the first expression
- **IIF** - returns one of two values based on a Boolean expression

- **CONCAT** - concatenates two or more strings
- **SUBSTRING** - extracts a substring from a string
- **CHARINDEX** - finds the position of a substring within a string
- **REPLACE** - replaces all occurrences of a specified substring within a string with another substring
- **LEN** - returns the length of a string
- **UPPER** - converts a string to uppercase
- **LOWER** - converts a string to lowercase
- **TRIM** - removes leading and trailing spaces from a string
- **ROUND** - rounds a numeric value to a specified number of decimal places

7

- **DATEPART** - extracts a specific part of a date
- **GETDATE** - returns the current date and time (similar to CURRENT_TIMESTAMP)
- **GROUPING SETS** - specifies multiple groupings for aggregation
- **CUBE** - generates all possible combinations of grouping sets for aggregation
- **ROLLUP** - generates subtotal values for a hierarchy of values
- **INTERSECT** - returns the intersection of two result sets
- **EXCEPT** - returns the difference between two result sets
- **MERGE** - performs insert, update, or delete operations on a target table based on the results of a join with a source table
- **CROSS APPLY** - performs a correlated subquery against each row of the outer table
- **OUTER APPLY** - similar to CROSS APPLY, but also returns rows from the outer table that have no matching rows in the inner table
- **PIVOT** - rotates a table-valued expression by turning the unique values from one column into multiple columns in the output

Field: ID, Department, email

Records: 1, Atul,IT,atulhx@gmail.com

What are Constraints in SQL?

Constraints are used to specify the rules concerning data in the table. It can be applied for single or multiple fields in an SQL table during the creation of the table or after creating using the ALTER TABLE command. The constraints are:

1. **NOT NULL** - Restricts NULL value from being inserted into a column.
2. **CHECK** - Verifies that all values in a field satisfy a condition.
3. **DEFAULT** - Automatically assigns a default value if no value has been specified for the field.
4. **UNIQUE** - Ensures unique values to be inserted into the field.
5. **INDEX** - Indexes a field providing faster retrieval of records.
6. **PRIMARY KEY** - Uniquely identifies each record in a table.
7. **FOREIGN KEY** - Ensures referential integrity for a record in another table.

What is a Primary Key?

The PRIMARY KEY constraint uniquely identifies each row in a table. It must contain UNIQUE values and has an implicit NOT NULL constraint

EXAMPLE:

```
CREATE TABLE Students (   /* Create table with a single field as primary key */
  ID INT NOT NULL
  Name VARCHAR(255)
  PRIMARY KEY (ID)
);

CREATE TABLE Students (   /* Create table with multiple fields as primary key */
  ID INT NOT NULL
  LastName VARCHAR(255)
  FirstName VARCHAR(255) NOT NULL,
  CONSTRAINT PK_Student
  PRIMARY KEY (ID, FirstName)
);
```

What are the different subsets of SQL?

Data Definition Language (DDL) - It allows you to perform various operations on the database such as CREATE, ALTER, and DELETE objects.

Data Manipulation Language(DML) - It allows you to access and manipulate data. It helps you to insert, update, delete and retrieve data from the database.

Data Control Language(DCL) - It allows you to control access to the database. Example - Grant, Revoke access permissions.

Transaction Control Language (TCL): It is used to deal with the transaction operations in the database. The commands in this category are COMMIT, ROLLBACK, SET TRANSACTION, SAVEPOINT, etc.

What are the set operators in SQL?

We use the set operators to merge data from one or more tables of the same kind. Although the set operators are like SQL joins, there is a significant distinction.

1. **UNION**: It combines two or more results from multiple SELECT queries into a single result set. It has a default feature to remove the duplicate rows from the tables. The following syntax illustrates the Union operator:

   SELECT columns FROM table1 UNION SELECT columns FROM table2;

2. **UNION ALL**: This operator is similar to the Union operator, but it does not remove the duplicate rows from the output of the SELECT statements. The following syntax illustrates the UNION ALL operator:

   SELECT columns FROM table1 UNION ALL SELECT columns FROM table2;

3. **INTERSECT**: This operator returns the common records from two or more SELECT statements. It always retrieves unique records and arranges them in ascending order by default. Here, the number of columns and data types should be the same. The following syntax illustrates the INTERSECT operator:

   SELECT columns FROM table1 INTERSECT SELECT columns FROM table2;

4. **MINUS**: This operator returns the records from the first query, which is not found in the second query. It does not return duplicate values. The following syntax illustrates the MINUS operator:

   SELECT columns FROM table1 MINUS SELECT columns FROM table2;

What are SQL comments?

Comments are explanations or annotations in SQL queries that are readable by programmers. It's used to make SQL statements easier to understand for humans. During the parsing of SQL code, it will be ignored. Comments can be written on a single line or across several lines.

- **Single Line Comments**: It starts with two consecutive hyphens (--).
- **Multi-line Comments**: It starts with /* and ends with */

What is a UNIQUE constraint?

A UNIQUE constraint ensures that all values in a column are different. This provides uniqueness for the column(s) and helps identify each row uniquely. Unlike primary key, there can be multiple unique constraints defined per table.

EXAMPLE:

```
CREATE TABLE Students (   /* Create table with a single field as unique */
  ID INT NOT NULL UNIQUE
  Name VARCHAR(255)
);

CREATE TABLE Students (   /* Create table with multiple fields as unique */
  ID INT NOT NULL
  LastName VARCHAR(255)
  FirstName VARCHAR(255) NOT NULL
  CONSTRAINT PK_Student
  UNIQUE (ID, FirstName)
);
```

What is a Foreign Key?

A FOREIGN KEY comprises of single or collection of fields in a table that essentially refers to the PRIMARY KEY in another table. Foreign key constraint ensures referential integrity in the relation between two tables.

EXAMPLE:

```
CREATE TABLE Students (   /* Create table with foreign key - Way 1 */
  ID INT NOT NULL
  Name VARCHAR(255)
  LibraryID INT
  PRIMARY KEY (ID)
  FOREIGN KEY (Library_ID) REFERENCES Library(LibraryID)
);
```

What are the syntax and use of the COALESCE function?

The COALESCE() function evaluates the arguments in sequence and returns the first NON-NULL value in a specified number of expressions. If it evaluates arguments as NULL or not found any NON-NULL value, it returns the NULL result.

The syntax of COALESCE function is given below:

```
COALESCE (exp1, exp2, .... expn)
```

What is the usage of the NVL() function?

The NVL() function is used to convert the NULL value to the other value. The function returns the value of the second parameter if the first parameter is NULL.

SQL Query for creating a database?

```
create database database_name;
```

SQL Query for creating a table?

```
use database_name;
create table table_name(col1_name data_type, col2_name data_type,.....)
```

EXAMPLE

```
create table emp (id int primary key auto_increment, name varchar(100), salary decimal(9,2));
```

SQL Query used to create a Table with same structure of another table?

```
create table emp_copy(select * from emp where 1=2);
```

here we write a false statement 1=2 because through this we just copy a structure of table not a data of table

SQL Query used to create a Table with same structure with data of another table?

```
create table emp select * from employee;
```

What is the SQL Query used to find the 2nd / 3rd /nth highest salary ?

1. **USING SUB-QUERY:**

```
select max(salary) from employee
where salary < (select max(salary) from employee
```

where salary < (select max(salary)) from employee));

2. **USING LIMIT:**

select salary from employee order by salary desc limit n-1,1;

EXAMPLE:

select salary from employee order by salary desc limit 2,1; - 3rd highest salary
select salary from employee order by salary desc limit 1,1; - 2nd highest salary
select salary from employee order by salary desc limit 0,1; - nth highest salary

3. **USING LIMIT OFFSET**

select salary from employee order by salary limit 1 offset n-1;

EXAMPLE:

select salary from employee order by salary limit 1 offset 2;

- 3rd highest salary

4. **USING DISTINCT:**

select salary from employee e1

where(n-1) = (

select count(distint(e2.salary)) fromemployee e2 where e2.salary > e1.salary);

nth highest salary we get here

What is the SELECT statement?

- ➔ SELECT operator are use to select data from database.
- ➔ The data return are stored in a result table called as result-set
- ➔ Select is Data manipulation language (DML) command

EXAMPLE:

select * from emp;

What are some common clauses used with SELECT query in SQL?

The following are some frequent SQL clauses used in conjunction with a SELECT query:

WHERE clause: In SQL, the WHERE clause is used to filter records that are required depending on certain criteria.

ORDER BY clause: The ORDER BY clause in SQL is used to sort data in ascending (ASC) or descending (DESC) order depending on specified field(s) (DESC).

GROUP BY clause: GROUP BY clause in SQL is used to group entries with identical data and may be used with aggregation methods to obtain summarised database results.

It can be used with SELECT, UPDATE and DELETE statements	It can be used with SELECT statement
It is implemented in row operation	It is implemented in column operation

What is an ALIAS command?

ALIAS command in SQL is the name that can be given to any table or a column. This alias name can be referred in WHERE clause to identify a particular table or a column.

EXAMPLE:

```
select id as empid from emp;
```

here empid - is temprory name given to a column name id

What is a View?

A view is a virtual table which consists of a subset of data contained in a table. Since views are not present, it takes less space to store. View can have data of one or more tables combined and it depends on the relationship.

What are Views used for?

A view refers to a logical snapshot based on a table or another view. It is used for the following reasons:

1. Restricting access to data.
2. Making complex queries simple.
3. Ensuring data independence.
4. Providing different views of same data.

What is the difference between DELETE and TRUNCATE statements?

DELETE	TRUNCATE
Delete command is used to delete a row in a table.	Truncate is used to delete all the rows from a table.
You can rollback data after using delete statement.	You cannot rollback data.
It is a DML command.	It is a DDL command.
It is slower than truncate statement.	It is faster.

Explain different types of index in SQL.

There are three types of index in SQL namely:

1. **Unique Index:** This index does not allow the field to have duplicate values if the column is unique indexed. If a primary key is defined, a unique index can be applied automatically.
2. **Clustered Index:** This index reorders the physical order of the table and searches based on the basis of key values. Each table can only have one clustered index.
3. **Non-Clustered Index:** Non-Clustered Index does not alter the physical order of the table and maintains a logical order of the data. Each table can have many nonclustered indexes.

What is the difference between DROP and TRUNCATE commands?

DROP command removes a table and it cannot be rolled back from the database whereas TRUNCATE command removes all the rows from the table.

How many Aggregate functions are available in SQL?

SQL provides seven (7) aggregate functions, which are given below:

1. **AVG():** returns the average value from specified columns.
2. **COUNT():** returns the number of table rows, including rows with null values.
3. **MAX():** returns the largest value among the group.
4. **MIN():** returns the smallest value among the group.
5. **SUM():** returns the total summed values(non-null) of the specified column.
6. **FIRST():** returns the first value of an expression.
7. **LAST():** returns the last value of an expression.

Explain String function in SQL?

1. **length()** -select length('Amit');
2. **upper()** - select upper('amit');
3. **lower()** - select lower('AmIT');
4. **replace()** - select replace('Amit Mahto', 'Mahto', 'Welcome'); ➔Output: Amit Welcome

What is the ACID property in a database?

ACID stands for Atomicity, Consistency, Isolation, Durability. It is used to ensure that the data transactions are processed reliably in a database system.

1. **Atomicity**: This property ensures that the transaction is completed in all-or-nothing way.
2. **Consistency**: This ensures that updates made to the database is valid and follows rules and restrictions.
3. **Isolation**: This property ensures integrity of transaction that are visible to all other transactions.
4. **Durability**: This property ensures that the committed

HAVING clause in SQL is used to filter records in combination with the GROUP BY clause. It is different from WHERE, since the WHERE clause cannot filter aggregated records.

What is a Join? List its different types.

The SQL Join clause is used to combine records (rows) from two or more tables in a SQL database based on a related column between the two.

There are four different types of JOINs in SQL:

1. **(INNER) JOIN:** Retrieves records that have matching values in both tables involved in the join. This is the widely used join for queries.
   SELECT *FROM Table_A INNER JOIN Table_B on Table_A.id = Table_B.id
   EXAMPLE:
   SELECT *FROM emp INNER JOIN student on emp.id = student.sid
2. **LEFT (OUTER) JOIN**: Retrieves all the records/rows from the left and the matched records/rows from the right table.
   EXAMPLE
   Select * from emp left outer join student on emp.id=student.sid;
3. **RIGHT (OUTER) JOIN:** Retrieves all the records/rows from the right and the matched records/rows from the left table.
   EXAMPLE:
   select * from emp right outer join student on emp.id=student.sid;
4. **FULL (OUTER) JOIN:** Retrieves all the records where there is a match in either the left or right table.

What is a Self-Join?

A self-join is a type of join that can be used to connect two tables. As a result, it is a unary relationship. Each row of the table is attached to itself and all other rows of the same table in a self-join.

What is a Cross-Join?

Cross join can be defined as a cartesian product of the two tables included in the join. The table after join contains the same number of rows as in the cross-product of the number of rows in the two tables. If a WHERE clause is used in cross join then the query will work like an INNER JOIN.

EXAMPLE:

```
SELECT stu.name, sub.subject FROM students AS stu CROSS JOIN subjects AS sub;
```

What is an Index?

An index refers to a performance tuning method of allowing faster retrieval of records from the table. An index creates an entry for each value and hence it will be faster to retrieve data.