**SQL Learning Roadmap**

1. Basics of SQL:

Introduction to Databases: Understand the importance of databases and their role in storing and managing data.

Basic SQL syntax: Learn the fundamental SQL commands such as DDL, DML, DCL, TCL, and DQL.

Data Types: Familiarize yourself with common data types like VARCHAR, INT, DATE, BIGINT, etc.

1. Database Design:

Normalization & Denormalization: Grasp the concepts of database normalization and when denormalization is appropriate.

Primary & Foreign Keys: Understand how to establish relationships between tables using keys.

Indexes: Learn about creating indexes for better query performance.

1. Advanced SQL Queries:

Joins: Master various types of joins INNER, LEFT, RIGHT, OUTER, FULL to combine data from multiple tables.

Subqueries: Explore the power of subqueries to create more complex queries.

Aggregation Functions: Use functions like COUNT, SUM, AVG, MIN and MAX to analyse data.

1. Data Manipulation:

Transactions: Understand the concepts of transactions and how to ensure data consistency.

Views: Create and use views for a virtual representation of data.

Stored Procedures & Functions: Learn to write reusable code for complex tasks.

1. Performance Optimization:

Query Optimization: Dive into techniques for optimizing SQL queries.

Indexing Strategies: Explore advanced indexing strategies for large datasets.

1. Database Administration:

Backup & Recovery: Understand the importance of regular backups and recovery procedures.

Security: Learn about SQL injection prevention and secure database management.

1. NoSQL Databases:

Introduction to NoSQL: Explore the differences between SQL and NoSQL databases.

Popular NoSQL Databases: Learn the basics of databases like MongoDB, Cassandra or Redis, Hadoop.

**Introduction to Databases:**

* Data is the driving force of any business. It helps you understand your customers, track your inventory, and make informed decisions. But without proper database management, your data can quickly become disorganized and unusable.
* Database management is the process of organizing and storing data in a way that makes it easy to find, use, and analyse.
* To manage these databases, Database Management System are used.
* Types of DBMS:

1. Relational DBMS: In this DBMS, data stored in table format.

EX: MySQL, Oracle

1. Non-relational DBMS: In this DBMS, data is stored in key-value pair.

EX: MongoDB, Redis

* SQL: Stands for Structured Query Language. It is used to update, delete, insert data in a table or relational database.

**Basic SQL commands:**

* DDL (Data Definition Language):
* DDL changes the structure of the table creating, deleting, altering a table.
* All the commands of DDL are auto-committed, that means it permanently save all the changes in database.
* Here are the commands that come under DDL:

1. **CREATE:** It is used to create a new table in the database.

**Syntax:** CREATE TABLE table\_name (column1 datatype, column2 datatype…);

**Ex:** CREATE TABLE EMPLOYEE (Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

1. **DROP:** It is used to delete both the structure and record stored in the table.

**Syntax:** DROP TABLE table\_name;

**Ex:** DROP TABLE EMPLOYEE;

1. **ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

**Syntax:** ALTER TABLE table\_name ADD column\_name datatype;

**Ex:** ALTER TABLE EMPLOYEE ADD (ADDRESS VARCHAR2(20));

1. **TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.

**Syntax:** TRUNCATE TABLE *table\_name*;

**Ex:** TRUNCATE TABLE EMPLOYEE;

* DML (Data Manipulation Language):
* DML commands are used to modify the database. It is responsible for all form of changes.
* The command of DML is not auto-committed that means it can’t permanently save all the changes in the database. They can be rollback.
* Here are the commands that come under DML:

1. **INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

**Syntax:** INSERT INTO *table\_name* (col1, col2, col3,.... col N) *values* (value1, value2, value3, .... valueN);

**Ex:** INSERT INTO EMPLOYEE(Name, Email)values(“Admin”, “admin@gmail.com”);

1. **UPDATE:** This command is used to update or modify the value of a column in the table.

**Syntax:** UPDATE *table\_name* SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]

**Ex:** UPDATE EMPLOYEE SET name = “Administrator” WHERE Id = '1'

1. **DELETE:** It is used to remove one or more row from a table.

**Syntax:** DELETE FROM *table\_name* [WHERE condition];

**Ex:** DELETE FROM EMPLOYEE WHERE id=1;

* DCL (Data Control Language):
* DCL commands are used to grant and take back authority from any database user.
* Here are the commands that come under DCL:

1. **Grant:** It is used to give user access privileges to a database.

**Ex:** GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

1. **Revoke:** It is used to take back permissions from the user.

**Ex:** REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

* TCL (Transaction Control Language):
* TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.
* These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.
* Here are the commands that come under TCL:

1. **Commit:** Commit command is used to save all the transactions to the database.

**Syntax:** COMMIT;

**Ex**: DELETE FROM CUSTOMERS WHERE AGE = 25;

COMMIT;

1. **Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

**Syntax:** ROLLBACK;

**Ex**: DELETE FROM CUSTOMERS WHERE AGE = 25;

ROLLBACK;

1. **SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:** SAVEPOINT SAVEPOINT\_NAME;

* DQL (Data Query Language):
* DQL is used to fetch the data from the database.
* It uses only one command:

1. **SELECT:** This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

**Syntax:** SELECT column1, column2, ... FROM *table\_name*;

**Ex:** SELECT name FROM employee WHERE age > 20;

**Database Design**

* Primary & Foreign Keys: To establish relationships between tables using keys.
* **One-to-One relationship:** A one-to-one relationship exists when each record in one table corresponds to exactly one record in another table. E**xample: I**f you have a table of employees and a table of employee profiles, each employee can have only one profile, and each profile can belong to only one employee. To create a one-to-one relationship, you need to use a primary key and a foreign key. A primary key is a column or a combination of columns that uniquely identifies each row in a table. A foreign key is a column or a combination of columns that references a primary key in another table. For example, you can use the employee ID as the primary key in the employees table, and as the foreign key in the employee profiles table.
* **One-to-Many relationship:** A one-to-many relationship exists when each record in one table corresponds to zero, one, or more records in another table. E**xample:** If you have a table of customers and a table of orders, each customer can have zero or more orders, but each order can belong to only one customer. To create a one-to-many relationship, you need to use a primary key and a foreign key as well. The primary key is usually in the table that represents the "one" side of the relationship, and the foreign key is usually in the table that represents the "many" sides of the relationship. For example, you can use the customer ID as the primary key in the customers table, and as the foreign key in the orders table.
* **Many-to-Many relationship:** A many-to-many relationship exists when each record in one table corresponds to zero, one, or more records in another table, and vice versa. E**xample:** If you have a table of products and a table of categories, each product can belong to zero or more categories, and each category can contain zero or more products. To create a many-to-many relationship, you need to use a third table, called a junction or associative table, that stores the combinations of primary keys from both tables. For example, you can use a product ID and a category ID as the primary keys in the products and categories tables, respectively, and as the foreign keys in the junction table.

**Advanced SQL Queries:**

* **Joins:** In SQL, JOIN means "to combine two or more tables". JOIN clause is used to combine the records from two or more tables in a database.

**Types of Joins:**

* **Inner Join** selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

**Syntax:** SELECT *column\_name(s)* FROM *table1* INNER JOIN *table2* ON *table1.column\_name = table2.column\_name*;

* **Left Join** returns all the values from left table and the matching values from the right table. If there is no matching in both tables, it will return NULL.

**Syntax:** SELECT *column\_name(s)* FROM *table1* LEFT JOIN *table2* ON *table1.column\_name = table2.column\_name*;

* **Right Join** returns all the values from right table and the matching values from the left table. If there is no matching in both tables, it will return NULL.

**Syntax:** SELECT *column\_name(s)* FROM *table1* RIGHT JOIN table2 ON *table1.column\_name = table2.column\_name*;

* **Full Join** returns all values from both left and right tables. It puts NULL on the place of matches not found.

**Syntax:** SELECT *column\_name(s)* FROM *table1* FULL JOIN *table2* ON *table1.column\_name* = *table2.column\_name* WHERE *condition*;

* **Self Join** is a regular join, but the table joined itself.

**Syntax:** SELECT *column\_name(s)* FROM *table1 T1, table1 T2*  
WHERE *condition*;

* **Subqueries:** A Subquery is a query within another SQL query and embedded within the WHERE clause.
* A subquery can be placed in a no. of SQL clauses like WHERE, FROM & HAVING.
* We can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
* A subquery is a query within another query. The outer query is known as main query and inner query is known as sub query.
* Subqueries are on the right side of the comparison operator.
* A subquery is enclosed in parenthesis.
* In the subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.
* Subqueries with **SELECT** statement are most frequently used.

**Syntax:** SELECT *column\_name* FROM *table\_name* WHERE *column\_name* expression operator (SELECT *column\_name* from *table\_name* WHERE condition);

* Subqueries with **INSERT** statement – In the insert statement, data returned from the subquery is used to insert into another table. In the subquery, the selected data can be modified with any of the char, date functions.

**Syntax:** INSERT INTO *table\_name* (column1, column2, column3....) SELECT \* FROM *table\_name* WHERE value operator

* Subqueries with **UPDATE** statement – can be used in conjuction with the update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

**Syntax:** UPDATE *table* SET *column\_name = new\_value* WHERE value operator (SELECT *column\_name* FROM *table\_name WHERE* condition);

* Subqueries with **DELETE** statement – can be used in conjunction with the Delete statement just like any other statements.

**Syntax:** DELETE FROM *table\_name* WHERE value operator (SELECT *column\_name* FROM *table\_name* WHERE condition);

* **Aggregate Functions:** SQL aggregate functions are used to perform the calculations on multiple rows of a single column of a table. It returns a single value. It is also used to summarize the data.

**Types of Functions:**

* **COUNT function** is used to count the no. of rows in a table. It can work on both numeric and non-numeric data types. COUNT function uses COUNT (\*) that returns the count of all rows in a specified table. It considers duplicate and null.

**Syntax:** SELECT COUNT (*column\_name*) FROM *table\_name* WHERE condition;

* **SUM function** is used to calculate the sum of all selected columns. It works on numeric fields only.

**Syntax:** SELECT SUM (*column\_name*) FROM *table\_name* WHERE condition;

* **AVG function** is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

**Syntax:** SELECT AVG (*column\_name*) FROM *table\_name* WHERE condition;

* **MAX function** is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

**Syntax:** SELECT MAX (*column\_name*) FROM *table\_name* WHERE condition;

* **MIN function** is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

**Syntax:** SELECT MIN (*column\_name*) FROM *table\_name* WHERE condition;

**Data Manipulation:**

* **Stored Procedures:** A stored procedure is a function in SQL that can be stored for later execution and then used many times hence, saving time. It is a group of SQL statements that performs the task. The stored procedure can be invoked explicitly whenever required. It may accept some inputs in the form of parameters, these may be one parameter or multiple parameters.

**When to use Stored Procedures:**

* Stored Procedures are generally used for reporting needs. They can usually retrieve the data faster and in a way that the report can just spit out directly instead of having to do any kind of calculation or similar.
* They are used for queries which are used very often.

**When not to use Stored Procedures:**

* Stored procedures are not to be used for queries that do not run very often. As stored procedures are compiled and saved in memory, using for queries that do not run often is an overload on our application.

**Syntax:** CREATE PROCEDURE *procedure\_name* AS *sql\_statement* GO;

**Ex:** The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

CREATE PROCEDURE SelectAllCustomers  
AS  
SELECT \* FROM Customers  
GO;

EXEC SelectAllCustomers;

**Execute Stored Procedure:** Exec *procedure\_name;*

* **Stored Procedure with one parameter:** The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

**Ex:**

CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)  
AS  
SELECT \* FROM Customers WHERE City = @City  
GO;

EXEC SelectAllCustomers @City = 'cityname';

* **Stored Procedure with Multiple parameters:** Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.
* The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

**Ex:**

CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)  
AS  
SELECT \* FROM Customers WHERE City = @City AND PostalCode = @PostalCode  
GO;

EXEC SelectAllCustomers @City = 'cityname', @PostalCode = 'postalcode';

* **Views in SQL:**
* Views in SQL are considered as a virtual table. A view also contains rows & columns.
* To create the view, we can select the fields from one or more tables present in the database.
* A view can either have specific rows based on certain conditions or all the rows of a table.

1. **Creating View:** A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

**Syntax:**

CREATE VIEW *view\_name* AS SELECT *column1, column2* FROM *table\_name* WHERE *condition*;

1. **Deleting View:** A view can be deleted using the **DROP VIEW** statement.

**Syntax:**

DROP VIEW *view\_name;*

* **Transactions in SQL:** A transaction is a unit or sequence of work that is performed on a database. Transactions are accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

**Transactional Control Commands:** Transactional control commands are only used with the DML Commands such as - INSERT, UPDATE and DELETE. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

* **COMMIT:** The COMMIT command is the transactional command used to save changes invoked by a transaction. It saves all the transactions occurred on the database since the last COMMIT or ROLLBACK.

**Syntax:** COMMIT;

* **ROLLBACK:** The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only undo transactions since the last COMMIT or ROLLBACK.

**Syntax:** ROLLBACK;