1. Basic **SQL Queries (Essential for Beginners)**
   These queries focus on fundamental operations like selecting, inserting, updating, and deleting data.
   - **SELECT**: Retrieve data from a database.
     SELECT column1, column2 FROM table_name;

     Example: SELECT name, email FROM users;

   - **COUNT()**: Count the total number or rows in a table or those that meet a specific condition.
     COUNT(*): Counts all the rows, including those with NULL values.
     COUNT(column_name): Counts only non-NULL values in the specified column.
     SELECT COUNT(condition) FROM table_name;
     Example: SELECT COUNT(*) FROM users;
     Example: SELECT COUNT(*) FROM users WHERE age > 18;
     Example: SELECT COUNT(DISTINCT city) FROM users;
     DISTINCT in the above selects only distinct values of city.
     Example: SELECT city, COUNT(*) FROM users GROUP BY city;
     The above will return the number of users for each city.
   - **WHERE**: Filter results based on conditions.
     SELECT column1 FROM table_name WHERE condition;

     Example: SELECT column1 FROM users WHERE age > 18;

   - INSERT INTO: Insert data into a table.
     INSERT INTO table_name (column1, column2) VALUES (value1, value2);

     Example: INSERT INTO users (name, email) VALUES ('John Doe', 'john@doe.com');

   - **UPDATE**: Modify existing data in a table.
     UPDATE table_name SET column1 = value1 WHERE condition;
     Example: UPDATE users SET email = 'newemail@example.com' WHERE id = 1;
   - **DELETE**: Remove data from a table
     DELETE FROM table_name WHERE condition;
     Example: DELETE FROM users WHERE id = 1;
   - **ORDER BY**: Sort the result set in ascending or descending order.
     SELECT column1 FROM table_name ORDER BY column1 ASC;
     Example: SELECT name FROM users ORDER BY age DESC;
   - **LIMIT**: Limit the number of results returned.
     SELECT column1 FROM table_name LIMIT 10;
     Example: SELECT name FROM users LIMIT 5;
2. **Intermediate SQL Queries (For Practical Use)**
   These queries involve more advanced data manipulation and query optimization techniques.
   - **JOIN**: Combine rows from two or more tables based on a related column.
     Example:

SELECT users.name, COUNT(*) AS order_count
FROM users
JOIN orders ON users.id = orders.user_id
GROUP BY users.name;
The above will return the count of orders for each user.

- **INNER JOIN**: Returns records with matching values in both tables.
  SELECT users.name, orders.order_date FROM users
  INNER JOIN orders ON users.id = orders.user_id;
- **LEFT JOIN**: Returns all records from the left table and matched records from the right table.
  SELECT users.name, orders.order_date FROM users
  LEFT JOIN orders ON users.id = orders.user_id;
- **RIGHT JOIN**: Returns all records from the right table and matched record from the left table.
  SELECT users.name, orders.order_date FROM users
  RIGHT JOIN orders ON users.id = orders.user_id;

- **GROUP BY**: Aggregate data based on a column.
  SELECT column, COUNT(*) FROM table_name GROUP BY column;
  Example: SELECT city, COUNT(*) FROM users GROUP BY city;
- **HAVING**: Filter aggregated data (used after GROUP BY).
  SELECT city, COUNT(*) FROM users
  GROUP BY city HAVING COUNT(*) > 5;
- **ALIAS**: Create temporary names for columns or tables.
  SELECT column1 AS alias_name FROM table_name;
  Example: SELECT name AS user_name FROM users;
- **UNION**: Combine results of two or more SELECT statements.
  SELECT column1 FROM table1
  UNION
  SELECT column1 FROM table2;
  Example:
  SELECT name FROM users
  UNION
  SELECT name FROM admins;
- **IN**: Filter results based on a list of values.
  SELECT * FROM users WHERE city IN ('New York', 'Los Angeles', 'Chicago');
- **BETWEEN**: Filter results within a range.
  SELECT * FROM users WHERE age BETWEEN 18 AND 30;
- **LIKE**: Search for patterns in columns.
  SELECT * FROM users WHERE name LIKE 'J%'; – Names that start with 'J'

3. **Advanced SQL Queries (For Expert-Level Use)**
   These queries delve into more complex operations, such as subqueries, stored procedures, and advanced joins.
   - **Subqueries**: A query inside another query (nested query).

SELECT * FROM users WHERE id IN (SELECT user_id FROM orders WHERE amount > 100);

- **EXISTS**: Check for the existence of records.
  SELECT * FROM users WHERE EXISTS (SELECT 1 FROM orders WHERE users.id = orders.user_id);
- **CASE Statements:** Conditional logic within queries.
  ```
  SELECT name,
  CASE
    WHEN age < 18 THEN 'Minor'
    WHEN age BETWEEN 18 AND 60 THEN 'Adult'
    ELSE 'Senior'
  END AS age_group
  FROM users;
  ```
- **Stored Procedures**: A set of SQL statements that can be saved and reused.
  ```
  CREATE PROCEDURE GetUserOrders
  @UserID INT
  AS
  SELECT * FROM orders WHERE user_id = @UserID;
  ```
- **Triggers** SQL code that is automatically executed when an event occurs in the database.
  ```
  CREATE TRIGGER user_update
  AFTER UPDATE ON users
  FOR EACH ROW
  BEGIN
    INSERT INTO audit_log (user_id, action) VALUES (NEW.id, 'updated');
  END;
  ```
- **Indexes**: Optimize database performance by creating indexes on frequently queried columns.
  CREATE INDEX idx_username ON users (username);
- **Views**: Virtual tables created based on a query.
  ```
  CREATE VIEW user_orders AS
  SELECT users.name, orders.order_date FROM users
  JOIN orders ON users.id = orders.user_id;
  ```
- **WITH (Common Table Expressions - CTE)**: Temporary result set used within a SELECT, INSERT, UPDATE, or DELETE.
  ```
  WITH OrderCount AS (
    SELECT user_id, COUNT(*) AS total_orders
    FROM orders
    GROUP BY user_id
  )
  SELECT users.name, OrderCount.total_orders
  FROM users
  JOIN OrderCount ON users.id = OrderCount.user_id;
  ```

4.