1. Create Table

Theory: Creating a table involves defining the table structure, including columns and their data types. **Structure**:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
);
```

2. Insert Sample Data

Theory: Inserting data into a table involves specifying the values for each column. **Structure**:

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

3. Select All Data from Each Table

Theory: Retrieving all rows and columns from a table. **Structure**:

```
SELECT * FROM table_name;
```

4. Show All Columns and Their Details for Each Table

Theory: Retrieving metadata about the table structure. **Structure**:

```
DESCRIBE table_name;
```

5. Basic Data Retrieval

- **Select All Columns from a Table**:
- `SELECT * FROM table_name;`
- **Select Specific Columns from a Table**:
- `SELECT column1, column2 FROM table_name;`
- **Select Distinct Values**:
- `SELECT DISTINCT column_name FROM table_name;`

6. Filtering Data

- **Filter Rows Using the WHERE Clause**:
- `SELECT * FROM table_name WHERE condition;`
- **Filter Rows Using Multiple Conditions**:
- `SELECT * FROM table_name WHERE condition1 AND condition2;`
- **Filter Rows Using the IN Operator**:
- `SELECT * FROM table_name WHERE column_name IN (value1, value2, ...);`

7. Sorting Data

- **Order Results by a Column**:
- `SELECT * FROM table_name ORDER BY column_name;`
- **Order Results by Multiple Columns**:
- `SELECT * FROM table_name ORDER BY column1, column2;`

8. Aggregating Data

- **Count the Number of Rows**:
- `SELECT COUNT(*) FROM table_name;`
- **Calculate the Average Value**:
- `SELECT AVG(column_name) FROM table_name;`
- **Find the Minimum and Maximum Values**:
- `SELECT MIN(column_name), MAX(column_name) FROM table_name;`

9. Grouping Data

- **Group by a Column and Count**:
- `SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;`
- **Group by Multiple Columns**:
- `SELECT column1, column2, COUNT(*) FROM table_name GROUP BY column1, column2;`

10. Joining Tables

- **Inner Join**:
- `SELECT * FROM table1 INNER JOIN table2 ON table1.column = table2.column;`
- **Left Join**:
- `SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column;`

11. Subqueries

- **Subquery in the WHERE Clause**:
- `SELECT * FROM table_name WHERE column_name IN (SELECT column_name FROM another_table);`
- **Subquery in the SELECT Clause**:
- `SELECT column_name, (SELECT column_name FROM another_table WHERE condition) FROM table_name;`

12. Advanced Queries

- **Using CASE Statements**:
- `SELECT column_name,`
- `       CASE`
- `           WHEN condition1 THEN result1`

```
            WHEN condition2 THEN result2
            ELSE result3
        END
FROM table_name;
```

- **Using COALESCE to Handle NULL Values**:

```
SELECT COALESCE(column_name, default_value) FROM table_name;
```

- **Using UNION to Combine Results from Multiple Queries**:

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

13. Updating Data

- **Update a Single Column**:

```
UPDATE table_name SET column_name = new_value WHERE condition;
```

- **Update Multiple Columns**:

```
UPDATE table_name SET column1 = value1, column2 = value2 WHERE
condition;
```

14. Deleting Data

- **Delete Specific Rows**:

```
DELETE FROM table_name WHERE condition;
```

- **Delete All Rows from a Table**:

```
DELETE FROM table_name;
```

15. Merging Data

- **Merge (Upsert) Data**:

```
MERGE INTO target_table USING source_table
ON (condition)
WHEN MATCHED THEN
  UPDATE SET column1 = value1
WHEN NOT MATCHED THEN
  INSERT (column1, column2) VALUES (value1, value2);
```

16. Advanced Data Retrieval

- **Using LIKE for Pattern Matching**:

```
SELECT * FROM table_name WHERE column_name LIKE 'pattern';
```

- **Using BETWEEN for Range Filtering**:

```
SELECT * FROM table_name WHERE column_name BETWEEN value1 AND value2;
```

- **Using EXISTS to Check for the Existence of Rows**:

```
SELECT * FROM table_name WHERE EXISTS (SELECT 1 FROM another_table
WHERE condition);
```

17. Window Functions

- **Using ROW_NUMBER for Ranking**:
- `SELECT column_name, ROW_NUMBER() OVER (ORDER BY column_name) AS row_num FROM table_name;`
- **Using SUM with PARTITION BY**:
- `SELECT column_name, SUM(column_name) OVER (PARTITION BY column_name) FROM table_name;`

18. Advanced Data Retrieval

- **Using CTE (Common Table Expressions)**:
- `WITH cte_name AS (`
- `    SELECT * FROM table_name WHERE condition`
- `)`
- `SELECT * FROM cte_name;`
- **Using Recursive CTE**:
- `WITH RECURSIVE cte_name AS (`
- `    SELECT * FROM table_name WHERE condition`
- `    UNION ALL`
- `    SELECT * FROM cte_name WHERE condition`
- `)`
- `SELECT * FROM cte_name;`
- **Using PIVOT to Transform Data**:
- `SELECT * FROM`
- `(SELECT column1, column2 FROM table_name)`
- `PIVOT (`
- `    SUM(column2)`
- `    FOR column1 IN (value1, value2, ...)`
- `);`

19. Advanced Data Manipulation

- **Using TRIGGERS to Automate Actions**:
- `CREATE TRIGGER trigger_name`
- `BEFORE INSERT ON table_name`
- `FOR EACH ROW`
- `BEGIN`
- `    -- trigger logic`
- `END;`
- **Using STORED PROCEDURES for Reusable Code**:
- `CREATE PROCEDURE procedure_name (parameters)`
- `BEGIN`
- `    -- procedure logic`
- `END;`
- **Using TRANSACTIONS to Ensure Data Integrity**:
- `BEGIN TRANSACTION;`
- `-- SQL statements`
- `COMMIT;`

20. Performance Optimization

- **Creating INDEXES to Speed Up Queries**:
- `CREATE INDEX index_name ON table_name (column_name);`
- **Using EXPLAIN to Analyze Query Performance**:
- `EXPLAIN SELECT * FROM table_name WHERE condition;`

21. Security and Permissions

- **Granting and Revoking Permissions**:
- `GRANT SELECT ON table_name TO user_name;`
- `REVOKE SELECT ON table_name FROM user_name;`

22. Data Export and Import

- **Exporting Data to a CSV File**:
- `SELECT * FROM table_name INTO OUTFILE 'file_path' FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';`
- **Importing Data from a CSV File**:
- `LOAD DATA INFILE 'file_path' INTO TABLE table_name FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';`

These topics cover a wide range of SQL functionalities, from basic data retrieval to advanced data manipulation and performance optimization.

Section 1: Table Creation and Data Insertion

1. Creating a Table:

- **Theory:** Defines the structure of a dataset in a database, specifying column names, data types, and constraints.
- **Structure:**

SQL

```
CREATE TABLE table_name (
    column1 data_type,
    column2 data_type,
    ...
    columnN data_type,
    PRIMARY KEY (column1),
    FOREIGN KEY (column2) REFERENCES other_table(column)
);
```

Use code with caution.

- **Example:**

  SQL

  ```
  CREATE TABLE Books (
      BookID INT PRIMARY KEY,
      Title VARCHAR(255),
      AuthorID INT,
      PublicationYear INT
  );
  ```

  Use code with caution.

2. Inserting Sample Data:

- **Theory:** Populates a table with initial data.
- **Structure:**

  SQL

  ```
  INSERT INTO table_name (column1, column2, ...)
  VALUES (value1, value2, ...);
  ```

  Use code with caution.

- **Example:**

  SQL

  ```
  INSERT INTO Books (BookID, Title, AuthorID, PublicationYear)
  VALUES (1, 'The Lord of the Rings', 1, 1954);
  ```

  Use code with caution.

Section 2: Basic Data Retrieval

3. Selecting All Data:

- **Theory:** Retrieves all rows and columns from a table.
- **Structure:**

  SQL

  ```
  SELECT * FROM table_name;
  ```

  Use code with caution.

- **Example:**

SQL

```
SELECT * FROM Books;
```

Use code with caution.

4. Showing Column Details:

- **Theory:** Provides information about a table's columns, including data types, nullability, and default values.
- **Structure:**

  SQL

  ```
  DESCRIBE table_name;
  ```

  Use code with caution.

- **Example:**

  SQL

  ```
  DESCRIBE Books;
  ```

  Use code with caution.

5. Specific Data Retrieval:

- **Selecting All Columns:**

  SQL

  ```
  SELECT * FROM table_name;
  ```

  Use code with caution.

- **Selecting Specific Columns:**

  SQL

  ```
  SELECT column1, column2, ... FROM table_name;
  ```

  Use code with caution.

- **Selecting Distinct Values:**

  SQL

```
SELECT DISTINCT column_name FROM table_name;
```

Section 2: Filtering Data

6. Filtering Rows:

- **Using the WHERE Clause:**

  SQL

  ```
  SELECT * FROM table_name WHERE condition;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT * FROM Books WHERE PublicationYear > 2000;
    ```

    Use code with caution.

- **Using Multiple Conditions:**

  SQL

  ```
  SELECT * FROM table_name WHERE condition1 AND condition2 OR condition3;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT * FROM Books WHERE PublicationYear > 2000 AND AuthorID =
    1;
    ```

    Use code with caution.

- **Using the IN Operator:**

  SQL

  ```
  SELECT * FROM table_name WHERE column_name IN (value1, value2, ...);
  ```

Use code with caution.

- **Example:**

  SQL

  ```
  SELECT * FROM Books WHERE AuthorID IN (1, 2);
  ```

  Use code with caution.

Section 3: Sorting Data

7. Ordering Results:

- **By a Column:**

  SQL

  ```
  SELECT * FROM table_name ORDER BY column_name ASC|DESC;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT * FROM Books ORDER BY PublicationYear DESC;
    ```

    Use code with caution.

- **By Multiple Columns:**

  SQL

  ```
  SELECT * FROM table_name ORDER BY column1 ASC|DESC, column2 ASC|DESC;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT * FROM Books ORDER BY AuthorID ASC, PublicationYear DESC;
    ```

    Use code with caution.

Section 4: Aggregating Data

8. Calculating Aggregates:

- **Counting Rows:**

  SQL

  ```
  SELECT COUNT(*) FROM table_name;
  ```

  Use code with caution.

- **Calculating Average:**

  SQL

  ```
  SELECT AVG(column_name) FROM table_name;
  ```

  Use code with caution.

- **Finding Minimum and Maximum:**

  SQL

  ```
  SELECT MIN(column_name), MAX(column_name) FROM table_name;
  ```

  Use code with caution.

Section 5: Grouping Data

9. Grouping and Counting:

- **Grouping by a Column:**

  SQL

  ```
  SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT AuthorID, COUNT(*) FROM Books GROUP BY AuthorID;
    ```

Use code with caution.

- **Grouping by Multiple Columns:**

  SQL

  ```
  SELECT column1, column2, COUNT(*) FROM table_name GROUP BY column1,
  column2;
  ```

Section 6: Joining Tables

10. Joining Tables:

- **Inner Join:**

  SQL

  ```
  SELECT * FROM table1 INNER JOIN table2 ON table1.column =
  table2.column;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT Books.Title, Authors.Name
    FROM Books
    INNER JOIN Authors ON Books.AuthorID = Authors.AuthorID;
    ```

    Use code with caution.

- **Left Join:**

  SQL

  ```
  SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT Books.Title, Authors.Name
    FROM Books
    ```

```
LEFT JOIN Authors ON Books.AuthorID = Authors.AuthorID;
```

Use code with caution.

Section 7: Subqueries

11. Subqueries:

- **Subquery in the WHERE Clause:**

  SQL

  ```
  SELECT * FROM table1 WHERE column IN (SELECT column FROM table2 WHERE
  condition);
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT * FROM Books WHERE AuthorID IN (SELECT AuthorID FROM
    Authors WHERE City = 'New York');
    ```

    Use code with caution.

- **Subquery in the SELECT Clause:**

  SQL

  ```
  SELECT column, (SELECT column FROM table2 WHERE condition) AS alias
  FROM table1;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT Books.Title, (SELECT COUNT(*) FROM Reviews WHERE
    Reviews.BookID = Books.BookID) AS ReviewsCount
    FROM Books;
    ```

    Use code with caution.

Section 8: Advanced Queries

12. Advanced Features:

- **Using CASE Statements:**

  SQL

  ```
  SELECT CASE WHEN condition THEN value1 ELSE value2 END AS result FROM
  table_name;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT BookID, Title, CASE WHEN PublicationYear > 2010 THEN
    'Modern' ELSE 'Classic' END AS BookType
    FROM Books;
    ```

    Use code with caution.

- **Using COALESCE to Handle NULL Values:**

  SQL

  ```
  SELECT COALESCE(column1, column2, ...) AS result FROM table_name;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT BookID, Title, COALESCE(AuthorName, 'Unknown Author') AS
    AuthorName
    FROM Books;
    ```

    Use code with caution.

- **Using UNION to Combine Results:**

  SQL

  ```
  SELECT column1, column2 FROM table1
  UNION
  SELECT column1, column2 FROM table2;
  ```

Use code with caution.

- **Example:**

  SQL

  ```
  SELECT BookID, Title FROM Books
  UNION
  SELECT MagazineID, Title FROM Magazines;
  ```

  Use code with caution.

Section 9: Updating Data

13. Updating Data:

- **Updating a Single Column:**

  SQL

  ```
  UPDATE table_name SET column_name = new_value WHERE condition;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    UPDATE Books SET PublicationYear = 2005 WHERE BookID = 1;
    ```

    Use code with caution.

- **Updating Multiple Columns:**

  SQL

  ```
  UPDATE table_name SET column1 = new_value1, column2 = new_value2 WHERE
  condition;
  ```

  Use code with caution.

  - **Example:**

    SQL

```
UPDATE Books SET Title = 'New Title', PublicationYear = 2010
WHERE BookID = 2;
```

Use code with caution.

Section 10: Deleting Data

14. Deleting Rows:

- **Deleting Specific Rows:**

  SQL

  ```
  DELETE FROM table_name WHERE condition;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    DELETE FROM Books WHERE PublicationYear < 1990;
    ```

    Use code with caution.

- **Deleting All Rows:**

  SQL

  ```
  DELETE FROM table_name;
  ```

  Use code with caution.

Section 11: Merging Data

15. Merging Data (UPSERT):

- **Using MERGE (if supported by your database):**

  SQL

  ```
  MERGE INTO target_table t
  USING source_table s
  ON t.column = s.column
  WHEN MATCHED THEN UPDATE SET ...
  WHEN NOT MATCHED THEN INSERT (column1, column2, ...) VALUES (s.column1,
  s.column2, ...);
  ```

Use code with caution.

- **Example:**

  SQL

  ```
  MERGE INTO Books t
  USING new_books s
  ON t.BookID = s.BookID
  WHEN MATCHED THEN UPDATE SET t.Title = s.Title, t.AuthorID =
  s.AuthorID
  WHEN NOT MATCHED THEN INSERT (BookID, Title, AuthorID) VALUES
  (s.BookID, s.Title, s.AuthorID);
  ```

  Use code with caution.

Section 12: Advanced Data Retrieval

16. Advanced Filtering:

- **Using LIKE for Pattern Matching:**

  SQL

  ```
  SELECT * FROM table_name WHERE column_name LIKE pattern;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT * FROM Books WHERE Title LIKE '%Lord%';
    ```

    Use code with caution.

- **Using BETWEEN for Range Filtering:**

  SQL

  ```
  SELECT * FROM table_name WHERE column_name BETWEEN value1 AND value2;
  ```

  Use code with caution.

  - **Example:**

    SQL

```
SELECT * FROM Books WHERE PublicationYear BETWEEN 2000 AND 2010;
```

Use code with caution.

- **Using EXISTS to Check for the Existence of Rows:**

  SQL

  ```
  SELECT * FROM table1 WHERE EXISTS (SELECT 1 FROM table2 WHERE
  condition);
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT * FROM Authors WHERE EXISTS (SELECT 1 FROM Books WHERE
    Books.AuthorID = Authors.AuthorID);
    ```

    Use code with caution.

Section 13: Window Functions

17. Window Functions:

- **Using ROW_NUMBER for Ranking:**

  SQL

  ```
  SELECT ROW_NUMBER() OVER (ORDER BY column_name) AS rank, column1,
  column2 FROM table_name;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT ROW_NUMBER() OVER (ORDER BY PublicationYear DESC) AS rank,
    BookID, Title
    FROM Books;
    ```

    Use code with caution.

- **Using SUM with PARTITION BY:**

SQL

```
SELECT column1, SUM(column2) OVER (PARTITION BY column1) AS total FROM
table_name;
```

Use code with caution.

- **Example:**

  SQL

  ```
  SELECT AuthorID, SUM(PublicationYear) OVER (PARTITION BY
  AuthorID) AS total_publication_years
  FROM Books;
  ```

  Use code with caution.

Section 14: Advanced Data Retrieval

18. Advanced Techniques:

- **Using CTE (Common Table Expressions):**

  SQL

  ```
  WITH cte_name AS (
      SELECT ...
  )
  SELECT ... FROM cte_name;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    WITH popular_authors AS (
        SELECT AuthorID, COUNT(*) AS book_count
        FROM Books
        GROUP BY AuthorID
        HAVING COUNT(*) > 5
    )
    SELECT Books.Title, Authors.Name
    FROM Books
    INNER JOIN popular_authors ON Books.AuthorID =
    popular_authors.AuthorID;
    ```

    Use code with caution.

- **Using Recursive CTE:**

  SQL

  ```
  WITH RECURSIVE cte_name AS (
      SELECT ...
      UNION ALL
      SELECT ...
  )
  SELECT ... FROM cte_name;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    WITH RECURSIVE hierarchy AS (
        SELECT EmployeeID, ManagerID, Level
        FROM Employees
        WHERE ManagerID IS NULL
        UNION ALL
        SELECT e.EmployeeID, e.ManagerID, h.Level + 1
        FROM Employees e
        INNER
     JOIN hierarchy h ON e.ManagerID = h.EmployeeID
    )
    SELECT * FROM hierarchy;
    ```

    Use code with caution.

- **Using PIVOT to Transform Data:**

  SQL

  ```
  SELECT ...
  FROM table_name
  PIVOT (
      aggregate_function(column_to_pivot) FOR pivot_column IN
  (pivot_value1, pivot_value2, ...)
  ) AS pvt;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    SELECT BookID, [2010], [2011], [2012]
    FROM Books
    PIVOT (
    ```

```
    COUNT(*) FOR PublicationYear IN ([2010], [2011], [2012])
) AS pvt;
```

Use code with caution.

Section 15: Advanced Data Manipulation

19. Advanced Manipulation:

- **Using TRIGGERS to Automate Actions:**

  SQL

  ```
  CREATE TRIGGER trigger_name
  BEFORE|AFTER INSERT|UPDATE|DELETE
  ON table_name
  FOR EACH ROW
  BEGIN
      -- Trigger body
  END;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    CREATE TRIGGER update_book_count
    AFTER INSERT OR DELETE ON Books
    FOR EACH ROW
    BEGIN
        UPDATE Authors
        SET BookCount = BookCount + CASE WHEN NEW.BookID IS NOT NULL
    THEN 1 ELSE -1 END
        WHERE AuthorID = NEW.AuthorID;
    END;
    ```

    Use code with caution.

- **Using STORED PROCEDURES for Reusable Code:**

  SQL

  ```
  CREATE PROCEDURE procedure_name (parameter1, parameter2, ...)
  BEGIN
      -- Procedure body
  END;
  ```

  Use code with caution.

- **Example:**

  SQL

  ```
  CREATE PROCEDURE get_books_by_author (IN author_id INT)
  BEGIN
      SELECT * FROM Books WHERE AuthorID = author_id;
  END;
  ```

  Use code with caution.

- **Using TRANSACTIONS to Ensure Data Integrity:**

  SQL

  ```
  START TRANSACTION;
  -- Multiple DML statements
  COMMIT;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    START TRANSACTION;
    DELETE FROM Orders WHERE OrderID = 1;
    UPDATE Customers SET TotalOrders = TotalOrders - 1 WHERE
    CustomerID = (SELECT CustomerID FROM Orders WHERE OrderID = 1);
    COMMIT;
    ```

    Use code with caution.

Section 16: Performance Optimization

20. Performance Optimization:

- **Creating INDEXES to Speed Up Queries:**

  SQL

  ```
  CREATE INDEX index_name ON table_name (column1, column2, ...);
  ```

  Use code with caution.

  - **Example:**

    SQL

```
CREATE INDEX idx_books_author ON Books (AuthorID);
```

Use code with caution.

- **Using EXPLAIN to Analyze Query Performance:**

  SQL

  ```
  EXPLAIN SELECT ... FROM table_name;
  ```

  Use code with caution.

Section 17: Security and Permissions

21. Security and Permissions:

- **Granting and Revoking Permissions:**

  SQL

  ```
  GRANT privilege ON table_name TO user;
  REVOKE privilege ON table_name FROM user;
  ```

  Use code with caution.

  - **Example:**

    SQL

    ```
    GRANT SELECT, INSERT, UPDATE, DELETE ON Books TO 'user1';
    REVOKE INSERT ON Books FROM 'user1';
    ```

    Use code with caution.

Section 18: Data Export and Import

22. Data Export and Import:

- **Exporting Data to a CSV File:**

  SQL

  ```
  SELECT * FROM table_name INTO OUTFILE 'file.csv' FIELDS TERMINATED BY
  ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';
  ```

  Use code with caution.

- **Importing Data from a CSV File:**

  SQL

  ```
  LOAD DATA INFILE 'file.csv' INTO TABLE table_name FIELDS TERMINATED BY
  ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';
  ```

  Use code with caution.

Remember to replace placeholders like `table_name`, `column_name`, and `user` with your specific values.