1. Table Creation, Alteration, Description, Copying, Deletion, Dropping (Analogy Included)

Code:

SQL

```sql
CREATE TABLE Workers (
  S_No INT PRIMARY KEY,
  Name VARCHAR(255),
  Designation VARCHAR(255),
  Branch VARCHAR(255)
);

-- Insert sample data
INSERT INTO Workers (S_No, Name, Designation, Branch)
VALUES (1, 'Kannan', 'Manager', 'Chennai'),
       (2, 'Mahesh', 'Supervisor', 'Madurai'),
       (3, 'Maruthi', 'Assistant', 'Trichy');

-- Alter table to add Salary column
ALTER TABLE Workers ADD COLUMN Salary INT;

-- Alter table to modify Name column (**Caution: Existing data might be truncated!**)
ALTER TABLE Workers MODIFY Name VARCHAR(100);

-- Describe the table structure
DESCRIBE Workers;

-- Copy the Workers table to a new table named emp
CREATE TABLE emp AS SELECT * FROM Workers;

-- Delete the second row from the Workers table
DELETE FROM Workers WHERE S_No = 2;

-- Drop the Workers table (**Caution: This action is permanent!**)
DROP TABLE Workers;
```

Analogy:

Imagine a company employee database. You're creating a table (Workers) to store information like employee ID (S_No), name, designation, and branch.

- **Creating the Table:** This is like designing a form to capture employee details.

- **Altering the Table:** This is like modifying the employee form by adding a new section for salary (adding a column) and potentially adjusting the name field size (modifying a column).
- **Describing the Table:** This is like examining the employee form structure to understand what information it captures.
- **Copying the Table:** This is like creating a duplicate copy of the employee data (Workers table) into a new table (emp).
- **Deleting a Row:** This is like removing a specific employee's record (second row) from the database.
- **Dropping the Table:** This is like discarding the entire employee form (Workers table) permanently. **Caution:** Use this with care!

2. ITERATE Loop Function (Analogy Included)

Code:

SQL

```sql
CREATE FUNCTION calculate_savings(initial_amount INT)
RETURNS INT
BEGIN
  DECLARE total_saved INT DEFAULT initial_amount;
  DECLARE income INT DEFAULT 100;

  ITERATE saving_loop
  DO
    SET total_saved = total_saved + income;
  WHILE total_saved < 4000;
  END LOOP saving_loop;

  RETURN total_saved;
END;

-- Example usage: Call the function with an initial amount of 2000
SELECT calculate_savings(2000);
```

Analogy:

Imagine you're saving money to reach a goal (4000). This function simulates saving in fixed increments (income) until you reach your target.

- **initial_amount (parameter):** This represents your starting amount.
- **total_saved (variable):** This tracks the accumulated savings after each loop iteration.
- **income (variable):** This defines the amount saved each time (e.g., monthly savings).

- **ITERATE loop:** This loop keeps adding the income to the total saved amount as long as the total is less than the target (4000).
- **Function return:** This returns the final accumulated savings after the loop terminates.

3. Tables with Primary and Foreign Keys (Analogy Included)

Code:

SQL

```sql
CREATE TABLE Employee (
  S_No INT PRIMARY KEY,
  Name VARCHAR(255),
  Designation VARCHAR(255),
  Branch VARCHAR(255)
);

CREATE TABLE Workers (
  S_No INT PRIMARY KEY,
  Name VARCHAR(255),
  Designation VARCHAR(255),
  Branch VARCHAR(255),
  FOREIGN KEY (S_No) REFERENCES Employee(S_No)
);
```

Analogy:

Imagine a company with two departments: HR and Payroll.

- **Employee Table:** This is like the HR department's table that stores core employee information (S_No, Name, Designation, Branch) with a unique identifier (S_No) as the primary key.
- **Workers Table:** This is like the Payroll department's table that also stores employee information but references the HR table's S_No as a foreign key. This ensures data consistency and avoids duplicate employee entries in Payroll.

4. Nested Queries with 10 Records and Examples (Analogy

1. Table with 10 Records and Analogy

Code:

```
SQL
```

```sql
CREATE TABLE Products (
  Product_ID INT PRIMARY KEY AUTO_INCREMENT,
  Product_Name VARCHAR(255),
  Category VARCHAR(100),
  Price DECIMAL(10,2),
  Stock INT
);

-- Insert 10 sample records
INSERT INTO Products (Product_Name, Category, Price, Stock)
VALUES
  ('Laptop', 'Electronics', 800.00, 10),
  ('Headphones', 'Electronics', 50.00, 20),
  ('Shirt', 'Clothing', 25.00, 30),
  ('Jeans', 'Clothing', 40.00, 15),
  ('Book', 'Stationery', 15.00, 50),
  ('Pen', 'Stationery', 2.00, 100),
  ('Coffee Maker', 'Kitchen', 75.00, 5),
  ('Mug', 'Kitchen', 10.00, 25),
  ('Desk Lamp', 'Office', 30.00, 8),
  ('Stapler', 'Office', 5.00, 40);
```

Analogy:

Imagine a store with a product database. You're creating a table named `Products` to keep track of information like product ID (automatically generated unique identifier), name, category, price, and stock availability.

2. Nested Query Examples with Analogy

Example 1: Find Products with Low Stock (Analogy: Finding Scarce Items)

Code:

SQL

```
SELECT Product_Name, Category
FROM Products
WHERE Stock < (
  SELECT AVG(Stock)
  FROM Products
);
```

Analogy:

Imagine you want to find products that are running low in stock. This nested query is like a two-step process:

1. **Inner query:** This calculates the average stock level across all products, acting like checking the overall stock situation in the store.
2. **Outer query:** This selects products from the main `Products` table where their individual stock is lower than the average stock calculated in the inner query. This is like identifying products that fall below the average stock level, indicating scarcity.

Example 2: Find Products in Specific Category with Price Above Average (Analogy: Finding Premium Items in a Category)

Code:

SQL

```
SELECT Product_Name, Price
FROM Products
WHERE Category = 'Electronics'
  AND Price > (
```

```
    SELECT AVG(Price)
    FROM Products
    WHERE Category = 'Electronics'
  );
```

Analogy:

Imagine you want to find premium-priced electronics in the store. This nested query is like a focused search:

1. **Inner query:** This calculates the average price of electronic products, acting like finding out the typical price range for electronics.
2. **Outer query:** This selects products from the `Products` table where the category is 'Electronics' and the individual price is higher than the average price calculated for electronics in the inner query. This is like identifying electronics that are priced above the average price point for that category.

In both examples, the nested query acts like a sub-query within the main query, providing additional filtering criteria based on calculations or comparisons within the database.