1. Book Table and Operations

Code:

SQL

```sql
-- Create Book table
CREATE TABLE Book (
  S_No INT PRIMARY KEY,
  B_Name VARCHAR(255),
  Author VARCHAR(255),
  Price DECIMAL(10,2),
  Publisher VARCHAR(255)
);

-- Insert sample data
INSERT INTO Book (S_No, B_Name, Author, Price, Publisher)
VALUES (1, 'DBMS', 'Seema Kedhar', 250, 'Charulatha'),
       (2, 'TOC', 'John Martin', 400, 'Tata McGraw Hill'),
       (3, 'C Programming', 'Balagurusamy', 300, 'Technical');

-- Display all books
SELECT * FROM Book;

-- Display books priced above 300
SELECT * FROM Book WHERE Price > 300;

-- Add a new book
INSERT INTO Book (S_No, B_Name, Author, Price, Publisher)
VALUES (4, 'Data Structures', 'Cormen et al.', 500, 'MIT Press');

-- Drop Author column (not recommended, consider altering data type
instead)
-- ALTER TABLE Book DROP COLUMN Author;
```

Analogy:

Imagine a library. The code creates a table named Book that acts like a catalog for the books in the library. Each book has details like a serial number (S_No), name (B_Name), author, price, and publisher.

- **Creating the Table:** It's like creating a new section in the library catalog to hold book information.
- **Inserting Data:** It's like adding new books to the library and recording their details in the

catalog.

- **Selecting Books:** It's like searching for books in the catalog based on criteria like price or title.
- **Adding a New Book:** It's like adding a new book to the library and updating the catalog with its details.
- **Dropping a Column (Not Recommended):** This is like removing the "Author" section from the catalog entirely. It's generally not recommended as it can cause data integrity issues. Consider altering the data type of the Author column instead if needed.

2. REPEAT Loop Function

Code:

SQL

```
DELIMITER // -- Define delimiter

CREATE FUNCTION check_income(income INT)
RETURNS INT
BEGIN
  DECLARE current_income INT DEFAULT income;
  REPEAT
    SET current_income = current_income + 100;
  UNTIL current_income >= 4000;
  END REPEAT;
  RETURN current_income;
END // -- End delimiter

DELIMITER ;  -- Reset delimiter

SELECT check_income(2000);  -- Example usage
```

Analogy:

Imagine saving money to reach a goal of $4000. This function acts like a loop that simulates saving $100 repeatedly until your income (current_income) reaches or exceeds the goal.

- **Setting Up:** You define a function named check_income that takes your starting income as input.
- **The Loop:** The function uses a REPEAT loop that continues as long as current_income is less than 4000.
- **Saving (Adding Income):** Inside the loop, you add $100 (simulating saving) to current_income in each iteration.

- **Checking the Goal:** The loop keeps repeating until current_income reaches or surpasses $4000.
- **Reaching the Goal:** Once the goal is met, the function returns the final current_income value.

3. CASE Statement Function

Code:

SQL

```
DELIMITER // -- Define delimiter

CREATE FUNCTION income_level(monthly_value INT)
RETURNS VARCHAR(255)
BEGIN
  DECLARE income_level VARCHAR(255);

  CASE
    WHEN monthly_value <= 4000 THEN SET income_level = 'Low Income';
    WHEN monthly_value <= 5000 THEN SET income_level = 'Avg Income';
    ELSE SET income_level = 'High Income';
  END CASE;

  RETURN income_level;
END // -- End delimiter

DELIMITER ;  -- Reset delimiter

SELECT income_level(3500);  -- Example usage
```

Analogy:

Imagine a system that categorizes income levels based on monthly income. This function acts like a decision-making tool based on set criteria.

- **Setting Up:** You define a function named income_level that takes your monthly income as input

Sources

1. https://github.com/subi15github/DBMS-Experiments

Absolutely! Here's the code with an analogy for listing employees working more than 20 years in a company:

Code:

```SQL
DELIMITER // -- Define delimiter

CREATE PROCEDURE find_long_term_employees()
BEGIN
  DECLARE hire_date_threshold DATE;
  SET hire_date_threshold = CURDATE() - INTERVAL 20 YEAR;

  SELECT Name, Designation, HireDate
  FROM Employee
  WHERE HireDate < hire_date_threshold;
END // -- End delimiter

DELIMITER ;  -- Reset delimiter

CALL find_long_term_employees();  -- Call the procedure
```

Analogy:

Imagine a company celebrating its employees' anniversaries. They want to recognize employees who have been with the company for more than 20 years.

The Code as a Special Process:

1. **DELIMITER // -- Define delimiter (Setting Up):** This line acts like defining a special instruction format for a specific task. Instead of regular SQL statements, you're creating a stored procedure with a `//` symbol as a marker to differentiate it.

2. **CREATE PROCEDURE find_long_term_employees() (Defining the Task):** This line is like naming the task and its purpose. It defines a stored procedure named `find_long_term_employees` that specifically retrieves information about employees who have been with the company for more than 20 years.

3. **BEGIN (Starting the Task):** This marks the beginning of the actual steps involved in the task.

4. **DECLARE hire_date_threshold DATE; SET hire_date_threshold = CURDATE() - INTERVAL 20 YEAR; (Calculating the Anniversary Date):** These lines are like calculating the eligibility date for the 20-year anniversary.

    - A variable `hire_date_threshold` is declared to hold the date.
    - It uses `CURDATE()` to get the current date and subtracts an interval of 20 years to determine the date that marks 20 years back from today.

5. **SELECT Name, Designation, HireDate FROM Employee WHERE HireDate < hire_date_threshold; (Finding Eligible Employees):** This line is like the core instruction of the task. It performs a selection operation on the `Employee` table, retrieving specific columns (Name, Designation, HireDate) for employees whose hire dates are before the calculated `hire_date_threshold`. This effectively identifies employees who joined before the 20-year anniversary date.

6. **END // -- End delimiter (Finishing Up):** This signifies the end of the specific task instructions defined with the `//` delimiter.

7. **DELIMITER ; -- Reset delimiter (Back to Normal):** This resets the delimiter back to the standard semicolon (`;`) for regular SQL statements outside the stored procedure.

8. **CALL find_long_term_employees(); -- Call the procedure (Running the Task):** This line is like executing the defined task. It calls the `find_long_term_employees` stored procedure, which retrieves and displays information about employees who have been with the company for more than 20 years based on the calculated anniversary date.

Overall, this code creates a reusable stored procedure that simplifies finding long-term employees without manually calculating the anniversary date or writing the selection statement every time. You can call this procedure anytime to get an updated list of employees celebrating their 20th anniversary or more with the company.