1. Workers Table Alterations, Description, Copy, Delete, and Drop (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 (same as provided)
INSERT INTO Workers (S_No, Name, Designation, Branch)
VALUES (1, 'Ram', 'Manager', 'Chennai'),
       (2, 'Santhosh', 'Supervisor', 'Madurai'),
       (3, 'Hari', 'Assistant', 'Trichy');

-- Add Salary column
ALTER TABLE Workers ADD COLUMN Salary INT;

-- Modifying the Name column (Caution):**
-- This modification is not recommended as it alters existing data.
Consider creating a new column with a different name instead.
-- (Example provided for demonstration purposes only)
-- ALTER TABLE Workers MODIFY Name VARCHAR(255) CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;

-- Describe table structure
DESCRIBE Workers;

-- Copy table Workers as emp (assuming emp doesn't exist)
CREATE TABLE emp (
  LIKE Workers
);

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

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

Analogy:

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

- **Adding Salary column:** This is like adding a new field (Salary) to the employee record, similar to including salary information for each employee in the database.
- **Modifying Name Column (Caution):** Modifying existing columns like Name can be risky as it might alter existing data. It's generally better to create a new column with a different name for additional data points. This analogy is similar to potentially corrupting existing employee information if you try to change the format or meaning of the "Name" field.
- **Describing the table:** This is like checking the table structure, similar to examining the different fields (columns) and their properties stored in the employee database.
- **Copying the Table:** This is like creating a duplicate of the employee database table with a new name (emp). Imagine making a backup copy of the employee data for reference or further processing.
- **Deleting a Row:** This is like removing a specific employee record from the database. In this case, you're deleting the second row (which corresponds to Santhosh).
- **Dropping the Table (Caution):** This is like permanently deleting the entire employee database table. Use this with caution as it cannot be undone! Imagine completely erasing the employee data.

2. CASE Statement Function for Income Level (Analogy Included)

Refer to explanation and code in previous response (section 2.2) for the CASE statement function.

Analogy:

Imagine a system that categorizes income levels based on monthly income. This function acts like a decision-making tool based on pre-defined thresholds.

3. Stored Procedure GetCustomerLevel (Analogy Included)

Code:

SQL

```
CREATE PROCEDURE GetCustomerLevel (
  IN p_customer_number INT,
  OUT p_customer_level VARCHAR(255)
)
BEGIN
  DECLARE credit_limit INT;

  SELECT credit_limit INTO credit_limit FROM Customers WHERE
customer_number = p_customer_number;

  CASE credit_limit
    WHEN credit_limit >= 10000 THEN SET p_customer_level = 'PLATINUM';
    WHEN credit_limit >= 5000 THEN SET p_customer_level = 'GOLD';
    ELSE SET p_customer_level = 'SILVER';
  END CASE;
END;
```

Analogy:

Imagine a customer loyalty program with different tiers based on credit limits. This stored procedure acts like an automated process to determine a customer's level based on their credit limit in the Customers table.

- **p_customer_number (IN parameter):** This represents the customer number used to identify the customer in the database.
- **p_customer_level (OUT parameter):** This variable stores the determined customer level (PLATINUM, GOLD, SILVER) after the procedure execution.
- **credit_limit (variable):** This variable temporarily holds the retrieved credit limit for the specified customer.
- **SELECT statement:** This retrieves the credit limit for the given customer number.
- **CASE statement:** This defines the criteria for assigning customer levels based on the credit limit.
  - **WHEN conditions:** These specify the

Here's the code and analogy for a procedure to list all records with a salary greater than

50000:

Code:

SQL

```
CREATE PROCEDURE ListHighEarners()
BEGIN
  DECLARE emp_name VARCHAR(255);
  DECLARE dept VARCHAR(255);
```

```
  DECLARE salary INT;

  DECLARE emp_cursor CURSOR FOR SELECT Name, Designation AS dept, Salary
FROM Workers WHERE Salary > 50000;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET DONE = TRUE;

  OPEN emp_cursor;
  SET DONE = FALSE;

  WHILE NOT DONE DO
    FETCH emp_cursor INTO emp_name, dept, salary;
    IF NOT DONE THEN
      SELECT CONCAT(emp_name, ', ', dept, ', ', salary) AS details;
    END IF;
  END WHILE;
  CLOSE emp_cursor;
END;

-- Call the procedure to execute
CALL ListHighEarners();
```

Analogy:

Imagine the Workers table as a filing cabinet with employee folders. You're creating a procedure like an automated process to:

1. **Declare variables:** These act like temporary storage spaces to hold retrieved data (employee name, department, salary).
2. **Declare cursor:** This acts like a pointer that will move through the qualified employee records (those with a salary greater than 50000).
3. **Open cursor:** This initiates the process of iterating through the qualified employee records.
4. **Declare DONE flag and handler:** This flag and handler act like a control mechanism to stop the loop when there are no more records to fetch.
5. **WHILE loop:** This loop keeps iterating as long as there are more records (DONE

is not set to TRUE).

6. **Fetch data:** This retrieves data from the current record pointed to by the cursor and stores it in the declared variables.
7. **Conditional display:** This checks if there's data retrieved (NOT DONE). If yes, it constructs a formatted string containing employee details (name, department, salary) and potentially displays it (the `SELECT` statement here would typically be used to display the data within the procedure or integrate it with calling code).
8. **Close cursor:** This releases resources associated with the cursor after the loop finishes iterating through all qualified records.

This procedure helps you efficiently identify and potentially display details of employees who earn more than 50000 in the Workers table.