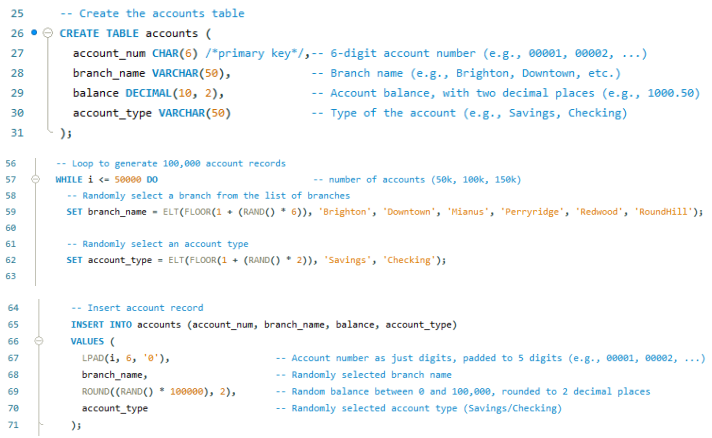
Title: Database Assignment 6
Your Name: Jack Saunders
Date: 12/10/2024

1. Similar to what we did in class, use a stored procedure to create the accounts table

Lines 26-30 define the accounts table that will be used for queries. There is a 6-digit account_num because the number may go up to 150k thus requiring 6 digits. Line 59 sets the branch_name to be one of 6 options. Line 62 sets account_type to be one of 2 options. Line 69 randomly assigns a balance up to 100k. Lines 65-70 insert these randomly chosen values as a tuple during the generate_accounts() stored procedure.	**Answer:** 

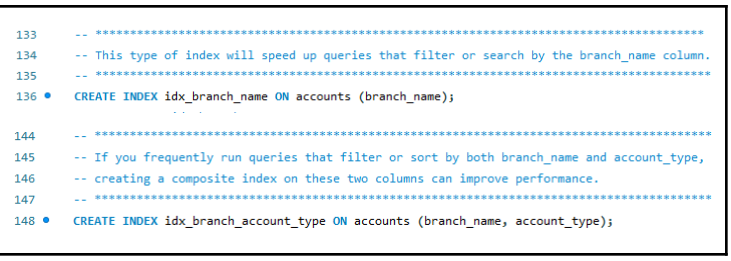
```
25    -- Create the accounts table
26    CREATE TABLE accounts (
27      account_num CHAR(6) /*primary key*/,-- 6-digit account number (e.g., 00001, 00002, ...)
28      branch_name VARCHAR(50),            -- Branch name (e.g., Brighton, Downtown, etc.)
29      balance DECIMAL(10, 2),             -- Account balance, with two decimal places (e.g., 1000.50)
30      account_type VARCHAR(50)            -- Type of the account (e.g., Savings, Checking)
31    );

56    -- Loop to generate 100,000 account records
57    WHILE i <= 50000 DO                   -- number of accounts (50k, 100k, 150k)
58      -- Randomly select a branch from the list of branches
59      SET branch_name = ELT(FLOOR(1 + (RAND() * 6)), 'Brighton', 'Downtown', 'Mianus', 'Perryridge', 'Redwood', 'RoundHill');
60
61      -- Randomly select an account type
62      SET account_type = ELT(FLOOR(1 + (RAND() * 2)), 'Savings', 'Checking');
63

64      -- Insert account record
65      INSERT INTO accounts (account_num, branch_name, balance, account_type)
66      VALUES (
67        LPAD(i, 6, '0'),                  -- Account number as just digits, padded to 5 digits (e.g., 00001, 00002, ...)
68        branch_name,                      -- Randomly selected branch name
69        ROUND((RAND() * 100000), 2),      -- Random balance between 0 and 100,000, rounded to 2 decimal places
70        account_type                      -- Randomly selected account type (Savings/Checking)
71      );
```

2. For timing analysis, you will need to populate the table with 50,000, 100,000, and 150,000 records.

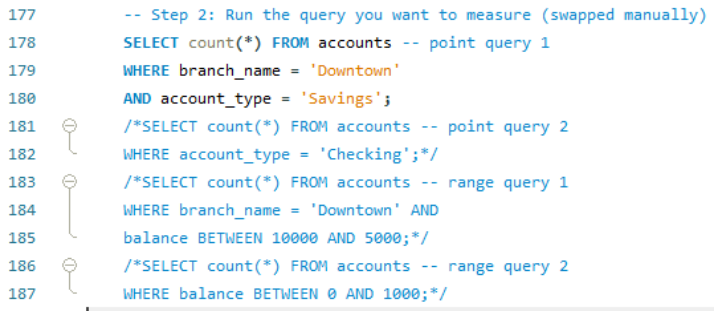
The limit for i in the while loop of the generate_accounts stored procedure is set manually to 50k, 100k, or 150k. This determines the number of accounts.	**Answer:**
	`WHILE i <= 50000 DO`
	`-- number of accounts (50k, 100k, 150k)`

3. Create **indexes** on the branch_name and account_type columns to optimize query performance.

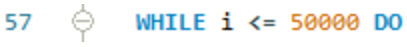
Line 136 creates an index on branch_name	**Answer:**

Line 148 creates an index on both branch_name and account_type	

4. You will compare **point queries** and **range queries**

There are 4 different queries: Point query 1 on line 178 looks for saving accounts in Downtown. Point query 2 on line 181 looks for all checking accounts. Range query 1 on line 183 looks for balances between 5k and 10k in Downtown. Range query 2 on line 186 looks for balances up to 1k.	**Answer:** 

5. Experiment with the following dataset sizes: 50K, 100K, 150K

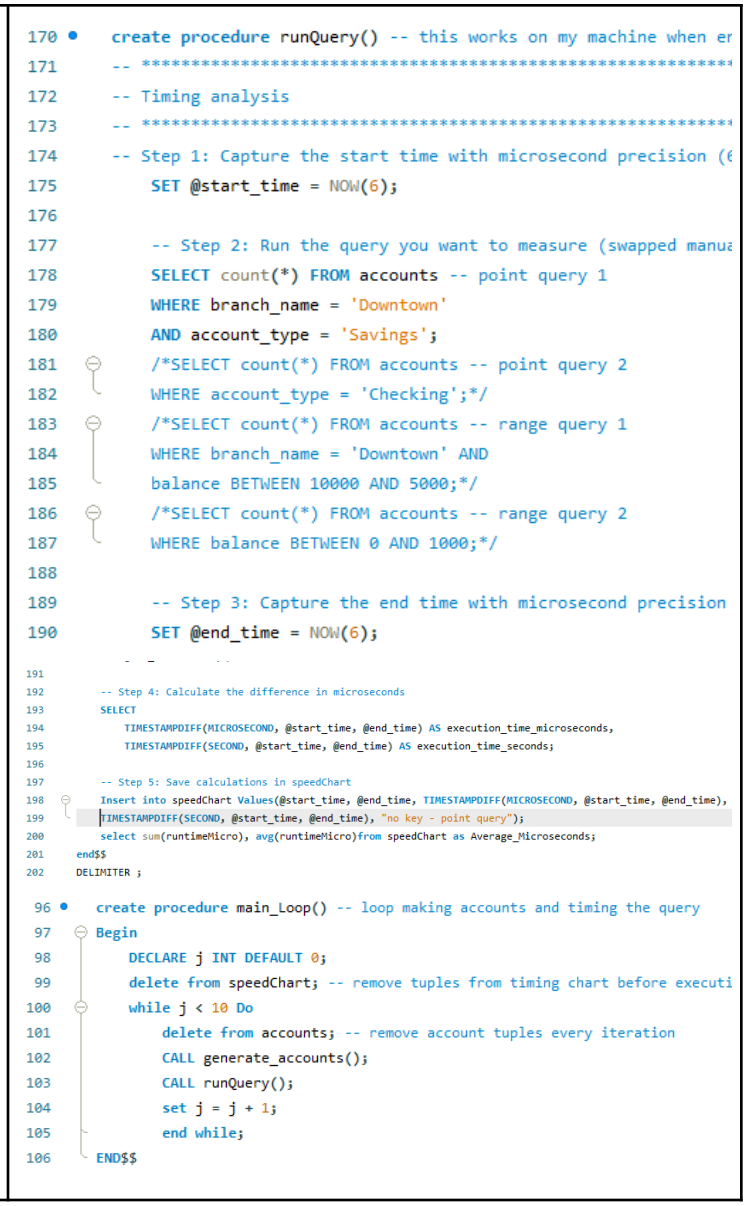
This was changed manually at line 57	**Answer:** 

6. For each dataset size, execute both **point** queries and **range** queries (2 times each) 10 times and record the execution time for each run.

T	**Answer:**

7. Create a **stored procedure** to **measure average execution times**

The stored procedure runQuery() records execution times of the queries. Line 175 starts a timer. One	**Answer:**

of the 4 queries are run between line 178 and 187, which are manually commented out each trial. Line 190 stops the timer. Line 193 calculates and displays the execution time based on the start and stop times. Line 198 inserts the value into the timing chart called speedChart. Line 200 calculates the total run time and average run time. Line 100 loops the 10 iterations of doing the timing sequence for line 103, which measures the execution time.	

```
170  create procedure runQuery() -- this works on my machine when er
171  -- ************************************************************
172  -- Timing analysis
173  -- ************************************************************
174  -- Step 1: Capture the start time with microsecond precision (6
175      SET @start_time = NOW(6);
176
177      -- Step 2: Run the query you want to measure (swapped manua
178      SELECT count(*) FROM accounts -- point query 1
179      WHERE branch_name = 'Downtown'
180      AND account_type = 'Savings';
181      /*SELECT count(*) FROM accounts -- point query 2
182      WHERE account_type = 'Checking';*/
183      /*SELECT count(*) FROM accounts -- range query 1
184      WHERE branch_name = 'Downtown' AND
185      balance BETWEEN 10000 AND 5000;*/
186      /*SELECT count(*) FROM accounts -- range query 2
187      WHERE balance BETWEEN 0 AND 1000;*/
188
189      -- Step 3: Capture the end time with microsecond precision
190      SET @end_time = NOW(6);
191
192  -- Step 4: Calculate the difference in microseconds
193  SELECT
194      TIMESTAMPDIFF(MICROSECOND, @start_time, @end_time) AS execution_time_microseconds,
195      TIMESTAMPDIFF(SECOND, @start_time, @end_time) AS execution_time_seconds;
196
197  -- Step 5: Save calculations in speedChart
198  Insert into speedChart Values(@start_time, @end_time, TIMESTAMPDIFF(MICROSECOND, @start_time, @end_time),
199  TIMESTAMPDIFF(SECOND, @start_time, @end_time), "no key - point query");
200  select sum(runtimeMicro), avg(runtimeMicro)from speedChart as Average_Microseconds;
201  end$$
202  DELIMITER ;

96   create procedure main_Loop() -- loop making accounts and timing the query
97   Begin
98       DECLARE j INT DEFAULT 0;
99       delete from speedChart; -- remove tuples from timing chart before executi
100      while j < 10 Do
101          delete from accounts; -- remove account tuples every iteration
102          CALL generate_accounts();
103          CALL runQuery();
104          set j = j + 1;
105          end while;
106  END$$
```

8. Summarize the results of the timing experiments

T	**Answer:**

9. Extra credit: Plot the timing results for each query. Represent execution times (index vs. no index) on the y-axis and num. of records on the x-axis.

L	**Answer:**