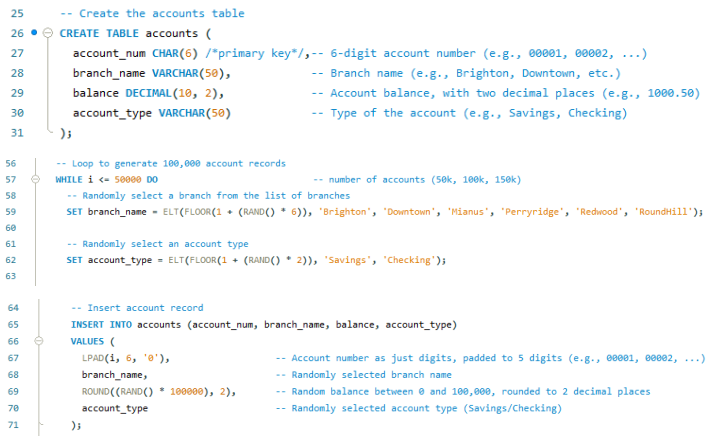
Title: Database Assignment 6
Your Name: Jack Saunders
Date: 12/10/2024

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

```
133     -- ********************************************************************************
134     -- This type of index will speed up queries that filter or search by the branch_name column.
135     -- ********************************************************************************
136 •   CREATE INDEX idx_branch_name ON accounts (branch_name);

144     -- ********************************************************************************
145     -- If you frequently run queries that filter or sort by both branch_name and account_type,
146     -- creating a composite index on these two columns can improve performance.
147     -- ********************************************************************************
148 •   CREATE INDEX idx_branch_account_type ON accounts (branch_name, 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:

```
177         -- Step 2: Run the query you want to measure (swapped manually)
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;*/
```

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

This was changed manually at line 57

Answer:

```
57      WHILE i <= 50000 DO
```

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.

The 4 queries from part 4 above were run with varying number of accounts and varying indecies. The results were copied to an Excel sheet. The average and sum is computed in the SQL code and in Excel. There is also a conditional formatting applied to the averages in Excel to show which query had a longer average

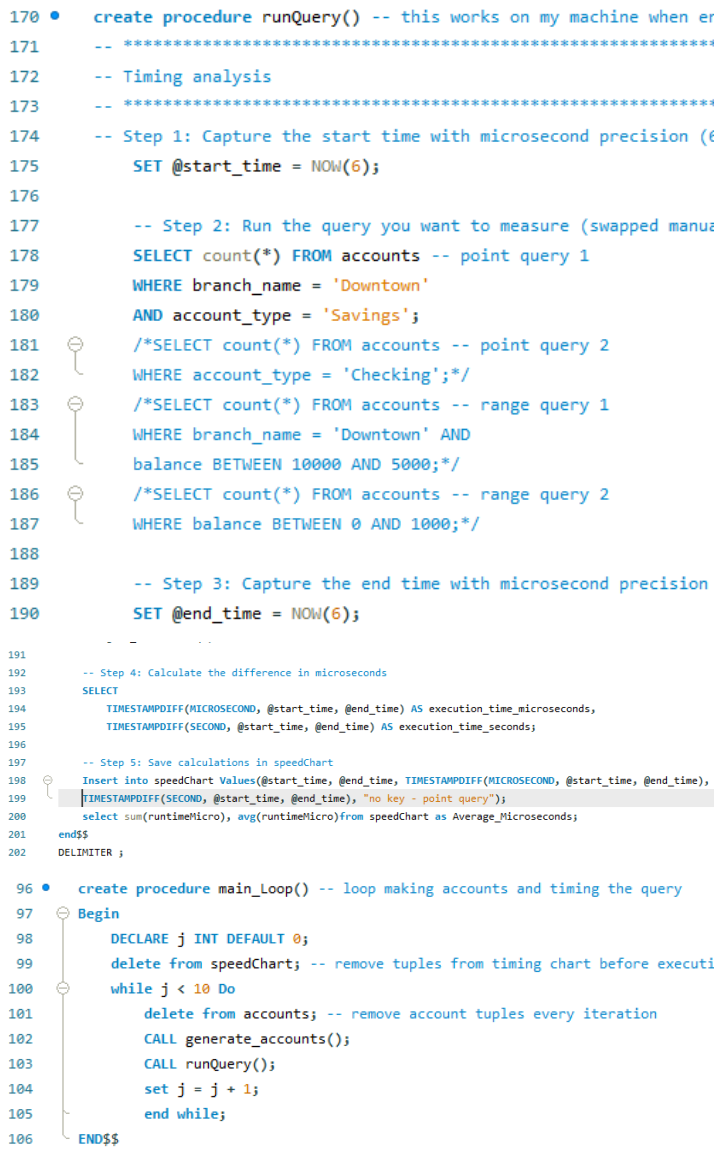
Answer:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Point 1 - no key			Point 2 - no key			Point 1 - primary key			Point 2 - primary key			Point 1 - pri & 2 index			Point 2 - pri & 2 index	
2	Trial 1	117,178		Trial 1	96,352		Trial 1	109,109		Trial 1	99,146		Trial 1	73,887		Trial 1	106,703
3	Trial 2	114,567		Trial 2	119,935		Trial 2	134,548		Trial 2	121,896		Trial 2	75,508		Trial 2	156,536
4	Trial 3	156,370		Trial 3	122,160		Trial 3	129,247		Trial 3	107,577		Trial 3	80,177		Trial 3	149,193
5	Trial 4	117,509		Trial 4	111,581		Trial 4	122,118		Trial 4	142,562		Trial 4	73,023		Trial 4	141,251
6	Trial 5	119,508		Trial 5	119,483		Trial 5	138,363		Trial 5	138,955		Trial 5	73,913		Trial 5	144,164
7	Trial 6	123,996		Trial 6	116,729		Trial 6	124,658		Trial 6	118,831		Trial 6	65,787		Trial 6	155,387
8	Trial 7	120,560		Trial 7	114,217		Trial 7	166,368		Trial 7	122,274		Trial 7	75,755		Trial 7	158,503
9	Trial 8	125,656		Trial 8	126,174		Trial 8	129,091		Trial 8	132,480		Trial 8	58,540		Trial 8	143,920
10	Trial 9	119,695		Trial 9	122,126		Trial 9	126,649		Trial 9	112,230		Trial 9	47,071		Trial 9	136,358
11	Trial 10	108,018		Trial 10	109,604		Trial 10	131,457		Trial 10	125,027		Trial 10	50,393		Trial 10	147,125
12	Average	122,306		Average	115,836		Average	131,161		Average	122,098		Average	67,405		Average	143,914
13	Sum	1,228,185		Sum	1,158,361		Sum	1,311,608		Sum	1,220,978		Sum	674,054		Sum	1,439,140
14		50k accounts						100k accounts						150k accounts			
15	Range 1 - no key			Range 2 - no key			Range 1 - primary key			Range 2 - primary key			Range 1 - pri & 2 index			Range 2 - pri & 2 index	
16	Trial 1	91,157		Trial 1	95,648		Trial 1	94,421		Trial 1	118,944		Trial 1	196,205		Trial 1	105,995

etc.

time in red and a shorter average time in green - using a gradient of color.	

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

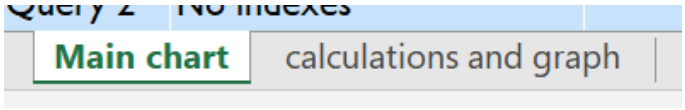
The stored procedure runQuery() records execution times of the queries. Line 175 starts a timer. One of the 4 queries are run between line 178 and 187, which are manually commented out each trial. Line 190 stops the timer. Line 193 calculates and displays the execution time based on the start and stop 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.	**Answer:** 

```
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

In general, using less accounts is faster. Also, indexing seems to be slower on average than not using indices. This is likely because using indices or multiple indices is more effective only with a large number of accounts like the 150k trials. Using no index at all is the slowest option for any number of accounts.
There is this standard chart and my own formatted charts.



Answer:

	A	B	C	D	E
1	Query Type	Description	Dataset Size	Index Type	(Microseconds) Average Execution Time
2	Point Query 1	Baseline	50,000	Without Indexes	122,306
3	Point Query 2	Baseline	50,000	Without Indexes	115,836
4	Range Query 1	Baseline	50,000	Without Indexes	117,999
5	Range Query 2	Baseline	50,000	Without Indexes	118,918
6	Point Query 1	Primary Key	100,000	With Indexes	131,161
7	Point Query 2	Primary Key	100,000	With Indexes	122,098
8	Range Query 1	Primary Key	100,000	With Indexes	136,272
9	Range Query 2	Primary Key	100,000	With Indexes	130,898
10	Point Query 1	Primary Key & 2 indexes	150,000	With Indexes	67,405
11	Point Query 2	Primary Key & 2 indexes	150,000	With Indexes	143,914
12	Range Query 1	Primary Key & 2 indexes	150,000	With Indexes	147,723
13	Range Query 2	Primary Key & 2 indexes	150,000	With Indexes	134,281
14	Point Query 1	No indexes	100,000	Without Indexes	168,935
15	Point Query 2	No indexes	100,000	Without Indexes	159,667
16	Range Query 1	No indexes	100,000	Without Indexes	187,272
17	Range Query 2	No indexes	100,000	Without Indexes	160,721
18	Point Query 1	No indexes	150,000	Without Indexes	209,807
19	Point Query 2	No indexes	150,000	Without Indexes	200,183
20	Range Query 1	No indexes	150,000	Without Indexes	208,495
21	Range Query 2	No indexes	150,000	Without Indexes	192,516
22	Point Query 1	1 index	50,000	With Indexes	82,563
23	Point Query 2	1 index	50,000	With Indexes	121,306

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.

The graph shows the execution times in bar chart format for all of the queries. More combinations of indices with sizes can be done, but this is decent. There is also a line chart overlaid on top of the bar char to show the average runtimes of each grouping of number of accounts (50k, 100k, 150k).

Answer: