

- **Title:** DB Assignment 6
- **Husam Alanazi**
- **Dec 10**

```
CREATE PROCEDURE generate_accounts()
BEGIN
    CREATE TABLE IF NOT EXISTS accounts (
        account_num INT PRIMARY KEY AUTO_INCREMENT,
        branch_name VARCHAR(50),
        balance DECIMAL(10, 2),
        account_type ENUM('Checking', 'Savings')
    );
END $$
```

This query creates a table called **accounts**. Here's what it does:

- **account\_num:** This is a unique number for each account. It increases automatically for every new account.
- **branch\_name:** This is where the account is located, and it can hold up to 50 letters.
- **balance:** This stores how much money is in the account. It can handle numbers with up to 10 digits, 2 of which can be after the decimal (like 1234.56).
- **account\_type:** This tells if the account is a "Checking" or "Savings" account.

[illegible]

```

DELIMITER $$

DROP PROCEDURE IF EXISTS populate_accounts $$

CREATE PROCEDURE populate_accounts(IN num_records INT)
BEGIN
    DECLARE i INT DEFAULT 0;
    WHILE i < num_records DO
        INSERT INTO accounts (branch_name, balance, account_type)
        VALUES (
            CASE
                WHEN RAND() < 0.2 THEN 'Downtown'
                WHEN RAND() < 0.4 THEN 'Uptown'
                WHEN RAND() < 0.6 THEN 'Suburb'
                WHEN RAND() < 0.8 THEN 'Eastside'
                ELSE 'Westside'
            END,
            ROUND(RAND() * 100000, 2),
            CASE WHEN RAND() < 0.5 THEN 'Checking' ELSE 'Savings' END
        );
        SET i = i + 1;
    END WHILE;
END $$

DELIMITER ;

```

This procedure, named `populate_accounts`, is designed to fill the `accounts` table with random data. It takes an input, `num_records`, which specifies how many rows of data to insert into the table. Inside the procedure, there is a loop (`WHILE`) that runs until the desired number of rows is inserted. For each iteration, a new record is created with three values

```
CALL populate_accounts(5);
SELECT COUNT(*) FROM accounts;
SELECT * FROM accounts LIMIT 10;
```

Result Grid

Filter Rows:

Q

Search

Export:

C...

50...

100%

33:60

Result Grid

Filter Rows:

Q

Search

Edit:

Export/Import:

Fetch rows:

branch_name	balance	account_ty...	account_num
Uptown	33640.00	Savings	1
Downtown	5583.00	Savings	2
Downtown	94137.00	Savings	3
Uptown	95206.00	Savings	4
Uptown	69769.00	Checking	5
Downtown	41280.00	Savings	6
Uptown	20782.00	Checking	7
Downtown	66433.00	Savings	8
Uptown	53476.00	Checking	9
Uptown	22860.00	Savings	10
NULL	NULL	NULL	NULL

```

DELIMITER $$

CREATE PROCEDURE create_indexes()
BEGIN
    -- Create index on branch_name
    CREATE INDEX idx_branch_name ON accounts(branch_name);

    -- Create index on account_type
    CREATE INDEX idx_account_type ON accounts(account_type);

    -- Create index on balance
    CREATE INDEX idx_balance ON accounts(balance);

END $$

DELIMITER ;

```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
accounts	0	PRIMARY	1	account_num	A	149734				BTREE			YES	
accounts	1	idx_branch_name	1	branch_name	A	1			YES	BTREE			YES	
accounts	1	idx_account_type	1	account_type	A	1			YES	BTREE			YES	
accounts	1	idx_balance	1	balance	A	74856			YES	BTREE			YES	

The **create\_indexes()** procedure creates three indexes on the accounts table to improve the performance of queries involving the specified columns. Here's a breakdown

```
CALL measure_query_time('SELECT count(*) FROM accounts WHERE branch_name = "Downtown" AND balance = 50000');
```

### Explanation:

- This query counts the number of records in the accounts table where the branch\_name is "Downtown" and the balance is exactly 50,000. Since there are no indexes on the table, the query has to scan all the records to find matches, which takes longer.

avg_time_microseco...	
270047.2000	

```
CALL measure_query_time('SELECT count(*) FROM accounts WHERE branch_name = "Downtown" AND balance = 50000');
```

This query counts the number of records in the accounts table where the branch\_name is 'Downtown' and the balance is exactly 50,000. Since an index is now applied on both the branch\_name and balance columns, the query executes much faster compared to when there were no indexes. The database engine uses the indexes to locate the matching records directly, avoiding a full table scan.

Result Grid		Filter Rows:	Search	Export:
avg_time_microseco...				
756.9000				

```
CALL measure_query_time('SELECT count(*) FROM accounts WHERE branch_name = "Downtown" AND balance BETWEEN 10000 AND 50000');
```

This query counts the number of records in the accounts table where the branch\_name is "Downtown" and the balance is between 10,000 and 50,000. Since indexes are applied on branch\_name and balance, the query runs significantly faster as the database engine can use the indexes to locate matching records instead of scanning the entire table.

Result Grid		Filter Rows:	Search	Export:
avg_time_microseco...				
463635.4000				

```
CALL measure_query_time('SELECT count(*) FROM accounts WHERE branch_name = "Downtown" AND balance BETWEEN 10000 AND 50000');
```

Running the query without indexes causes the database engine to perform a full table scan, as it cannot quickly locate records matching the conditions. This significantly increases the query execution time."

avg_time_microseco...	
318880.6000	

Summery:

### Key Observations:

#### 1. Point Queries:

- Without indexes, the execution times were much higher because the database had to scan the whole table to find matches.
- With indexes, the queries ran significantly faster. For example, in Point Query 1, the time dropped from 273,361.6 to 531.7 microseconds.

#### 2. Range Queries:

- Without indexes, range queries were slower because they required scanning a large number of records.
- With indexes, performance improved but not as drastically as point queries, especially for Range Query 2, where the improvement was relatively small.

#### 3. Overall Impact of Indexes:

- Indexing reduced query execution time across all scenarios. However, the improvement was most noticeable for point queries

1	Query Type	Description	Dataset Size	Index Type	Time (microseconds)		
2	Point Quer	branch_nar	50000	Without In	273361.6		
3	Point Quer	branch_nar	50000	With Index	531.7		
4	Point Quer	account_ty	50000	Without In	996408		
5	Point Quer	account_ty	50000	With Index	599.8		
6	Range Quer	branch_nar	50000	Without In	517260.9		
7	Range Quer	branch_nar	50000	With Index	312720.6		
8	Range Quer	account_ty	50000	Without In	879348.3		
9	Range Quer	account_ty	50000	With Index	843158.5		
10							

Extra credit:

Query Type	Description	Dataset Size	Index Type	Time (microseconds)
Point Query	branch_name	50000	Without Index	273361.6
Point Query	branch_name	50000	With Index	531.7
Point Query	account_type	50000	Without Index	996408
Point Query	account_type	50000	With Index	599.8
Range Query	branch_name	50000	Without Index	517260.9
Range Query	branch_name	50000	With Index	312720.6
Range Query	account_type	50000	Without Index	879348.3
Range Query	account_type	50000	With Index	843158.5

