

NAME – Shashank Mishra

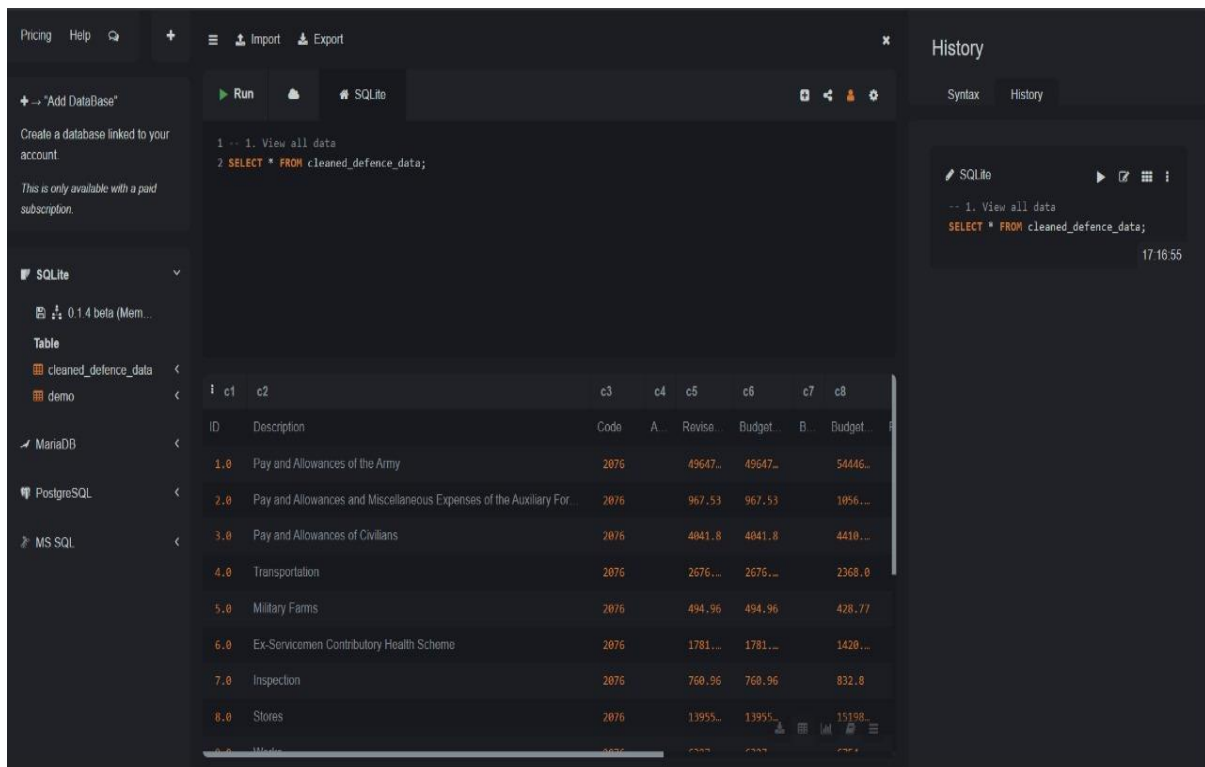
Task 3: SQL for Data Analysis

Objective: Use SQL queries to extract and analyze data from a database.

Tools: MySQL or PostgreSQL or SQLite

Deliverables: SQL queries in a SQL file + screenshots of output

1.



The screenshot shows a SQL client interface with a dark theme. On the left, there's a sidebar with a list of databases: SQLite, MariaDB, PostgreSQL, and MS SQL. The main area displays a SQL query and its results. The query is:

```
-- 1. View all data
SELECT * FROM cleaned_defence_data;
```

The results are shown in a table with 8 columns: ID, Description, Code, A..., Revise..., Budget..., B..., and Budget... The table contains 8 rows of data.

ID	Description	Code	A...	Revise...	Budget...	B...	Budget...
1.0	Pay and Allowances of the Army	2076		49647...	49647...		54446...
2.0	Pay and Allowances and Miscellaneous Expenses of the Auxiliary For...	2076		967.53	967.53		1056...
3.0	Pay and Allowances of Civilians	2076		4041.8	4041.8		4410...
4.0	Transportation	2076		2676...	2676...		2368.0
5.0	Military Farms	2076		494.96	494.96		428.77
6.0	Ex-Servicemen Contributory Health Scheme	2076		1781...	1781...		1420...
7.0	Inspection	2076		760.96	760.96		832.8
8.0	Stores	2076		13955...	13955...		15198...

On the right, there's a 'History' panel showing the same query and its execution time: 17:16:55.

2.

The screenshot shows the SQLite Studio application. On the left, the 'Database Explorer' pane shows a project named '0.1.4 beta (Mem...)' with a table named 'cleaned_defence_data'. The main editor displays a SQL query: `-- 2. View limited rows (top 5 rows)`
`SELECT * FROM cleaned_defence_data`
`LIMIT 5;` The 'Run' button is highlighted. The results pane shows a table with 8 columns: ID, Description, Code, A, Revisa, Budget, B, Budget. The first four rows of data are visible.

ID	Description	Code	A	Revisa	Budget	B	Budget
1.0	Pay and Allowances of the Army	2076		49647...	49647...		54446...
2.0	Pay and Allowances and Miscellaneous Expenses of the Auxiliary For...	2076		967.53	967.53		1056...
3.0	Pay and Allowances of Civilians	2076		4041.8	4041.8		4410...
4.0	Transportation	2076		2676...	2676...		2368.0

The History pane on the right shows the executed query and its timestamp: 17:21:34.

3.

The screenshot shows the SQLite Studio application. The main editor displays a SQL query: `-- 3. View specific columns`
`SELECT`
`c3 AS Scheme,`
`CAST(c6 AS REAL) AS Budget_2015_2016,`
`CAST(c14 AS REAL) AS Budget_2016_2017`
`FROM cleaned_defence_data;` The 'Run' button is highlighted. The results pane shows a table with 4 columns: Scheme, Budget_2015_2016, Budget_2016_2017. The first 10 rows of data are visible.

Scheme	Budget_2015_2016	Budget_2016_2017
Code	0	0
2076	49647.95	60548.18
2076	967.53	1172.2
2076	4041.8	4878.8
2076	2676.16	2828.1
2076	494.96	354.1
2076	1781.39	2639
2076	760.96	851.04
2076	13955.3	16697.84
2076	6327.95	8259
2076	4436.32	5453
2076	874.47	1016.39

The History pane on the right shows the executed query and its timestamp: 17:22:03.

4.

SQLite

0.1.4 beta (Mem...)

Table

- cleaned_defence_data
- demo

MariaDB

PostgreSQL

MS SQL

```

1 -- 4. List unique values (Distinct Descriptions)
2 SELECT DISTINCT c2 AS Unique_Descriptions
3 FROM cleaned_defence_data;

```

Unique_Descriptions
Description
Pay and Allowances of the Army
Pay and Allowances and Miscellaneous Expenses of the Auxiliary Forces
Pay and Allowances of Civilians
Transportation
Military Farms
Ex-Servicemen Contributory Health Scheme
Inspection
Stores

History

SQLite

```

-- 4. List unique values (Distinct Descriptions)
SELECT DISTINCT c2 AS Unique_Descriptions
FROM cleaned_defence_data;

```

19:59:59

SQLite

```

CREATE TABLE 'cleaned_defence_data' ('c2' TEXT);

```

19:59:56

SQLite

```

-- 4. List unique values (Distinct Descriptions)
SELECT DISTINCT c2 AS Unique_Descriptions
FROM cleaned_defence_data;

```

19:59:30

SQLite

```

-- 4. List unique values (Distinct Descriptions)
SELECT DISTINCT c2 AS Unique_Descriptions
FROM cleaned_defence_data;

```

2000

24-04-2025

5.

SQLite

0.1.4 beta (Mem...)

Table

- cleaned_defence_data
- demo

MariaDB

PostgreSQL

MS SQL

```

1 -- 5. Use WHERE to filter rows (where revised budget is greater than original)
2 SELECT
3   c3 AS Scheme,
4   CAST(c6 AS REAL) AS Budget_2015_2016,
5   CAST(c7 AS REAL) AS Revised_2015_2016
6 FROM cleaned_defence_data
7 WHERE CAST(c7 AS REAL) > CAST(c6 AS REAL);

```

Scheme	Budget_2015_2016	Revised_2015_2016
0076	-1964.72	0

History

SQLite

```

-- 5. Use WHERE to filter rows (where revised budget is greater than original)
SELECT
  c3 AS Scheme,
  CAST(c6 AS REAL) AS Budget_2015_2016,
  CAST(c7 AS REAL) AS Revised_2015_2016
FROM cleaned_defence_data
WHERE CAST(c7 AS REAL) > CAST(c6 AS REAL);

```

17:23:17

SQLite

```

-- 4. List unique values (Distinct Descriptions)
SELECT DISTINCT c2 AS Unique_Descriptions
FROM cleaned_defence_data;

```

17:22:45

SQLite

```

-- 3. View specific columns
SELECT
  c3 AS Scheme,
  CAST(c6 AS REAL) AS Budget_2015_2016,
  CAST(c7 AS REAL) AS Revised_2015_2016
FROM cleaned_defence_data;

```

17:22:03

Upcoming Earnings

24-04-2025

6.

The screenshot shows the SQLite IDE interface. The SQL query being executed is:

```
1 -- 6. Combine conditions with AND/OR
2 SELECT
3   c3 AS Scheme,
4   CAST(c6 AS REAL) AS Budget_2015_2016,
5   CAST(c14 AS REAL) AS Budget_2016_2017
6 FROM cleaned_defence_data
7 WHERE CAST(c6 AS REAL) > 5000 AND CAST(c14 AS REAL) > 10000;
```

The results table is as follows:

Scheme	Budget_2015_2016	Budget_2016_2017
2076	49647.95	60548.18
2076	13955.3	16697.84
	85785.82	104158.95

7.

The screenshot shows the SQLite IDE interface. The SQL query being executed is:

```
1 -- 7. Sort results in descending order (by budget increase)
2 SELECT
3   c3 AS Scheme,
4   CAST(c6 AS REAL) AS Budget_2015_2016,
5   CAST(c14 AS REAL) AS Budget_2016_2017,
6   (CAST(c14 AS REAL) - CAST(c6 AS REAL)) AS Increase
7 FROM cleaned_defence_data
8 ORDER BY Increase DESC
9 LIMIT 5;
```

The results table is as follows:

Scheme	Budget_2015_2016	Budget_2016_2017	Increase
	85785.82	104158.95	18373.129999999999
2076	49647.95	60548.18	10900.230000000003
2076	13955.3	16697.84	2742.5400000000001
2076	6327.95	8259	1931.0500000000002
2076	4436.32	5453	1016.6800000000003

8.

The screenshot shows the SQLite Studio interface. The main editor contains the following SQL query:

```
1 -- 8. Count total number of rows
2 SELECT COUNT(*) AS Total_Rows
3 FROM cleaned_defence_data;
```

The results pane on the right shows a single row with the value 15 for the column Total_Rows.

Total_Rows
15

9.

The screenshot shows the SQLite Studio interface. The main editor contains the following SQL query:

```
1 -- 9. Aggregate: Sum, Avg, Max, Min of 2016 Budget
2 SELECT
3   SUM(CAST(c14 AS REAL)) AS Total_2016,
4   AVG(CAST(c14 AS REAL)) AS Average_2016,
5   MAX(CAST(c14 AS REAL)) AS Max_2016,
6   MIN(CAST(c14 AS REAL)) AS Min_2016
7 FROM cleaned_defence_data;
```

The results pane on the right shows a single row with the following values:

Total_2016	Average_2016	Max_2016	Min_2016
288317.9	13887.859999999999	104158.95	-2836.88

10.

The screenshot shows the SQLite IDE interface. On the left, there's a sidebar with a list of databases: SQLite (0.14 beta), MariaDB, PostgreSQL, and MS SQL. The main window displays a SQL query and its results.

```

1 -- 10. Group and aggregate by Description
2 SELECT
3   c2 AS Description,
4   COUNT(*) AS Count_Entries,
5   SUM(CAST(c14 AS REAL)) AS Total_2016_Budget
6 FROM cleaned_defence_data
7 GROUP BY c2
8 ORDER BY Total_2016_Budget DESC;
  
```

Description	Count_Entries	Total_2016_Budget
Grand Total arate document.	1	104150.95
Pay and Allowances of the Army	1	60548.18
Stores	1	16697.84
Works	1	8259
Rashtriya Rifles	1	5453
Pay and Allowances of Civilians	1	4878.8
Transportation	1	2828.1
Ex-Servicemen Contributory Health Scheme	1	2639
Other Expenditure	1	2298.18
Pay and Allowances and Miscellaneous Ex.	1	1172.2
National Cadet Corps	1	1016.96

The right sidebar shows the history of queries executed, with timestamps ranging from 17:27:08 to 17:27:35.

11.

The screenshot shows the SQLite IDE interface. On the left, there's a sidebar with a list of databases: SQLite (0.14 beta), MariaDB, PostgreSQL, and MS SQL. The main window displays a SQL query and its results.

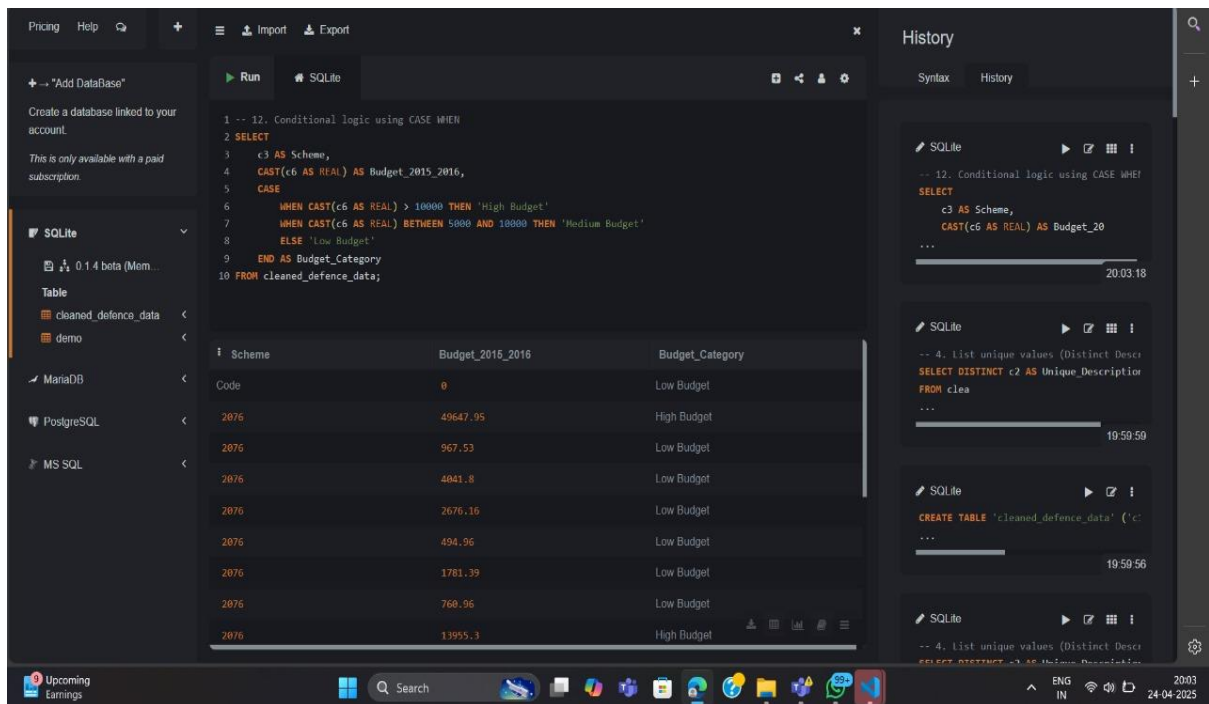
```

1 -- 11. Rename columns in result using AS
2 SELECT
3   c2 AS Description,
4   CAST(c14 AS REAL) AS Budget_2016_2017
5 FROM cleaned_defence_data
6 ORDER BY Budget_2016_2017 DESC
7 LIMIT 5;
8
  
```

Description	Budget_2016_2017
Grand Total arate document.	104150.95
Pay and Allowances of the Army	60548.18
Stores	16697.84
Works	8259
Rashtriya Rifles	5453

The right sidebar shows the history of queries executed, with timestamps ranging from 17:27:08 to 17:27:42.

12.



The screenshot shows the Elevate Labs SQL editor interface. The main query window displays a SQL query for conditional logic using CASE WHEN. The query is as follows:

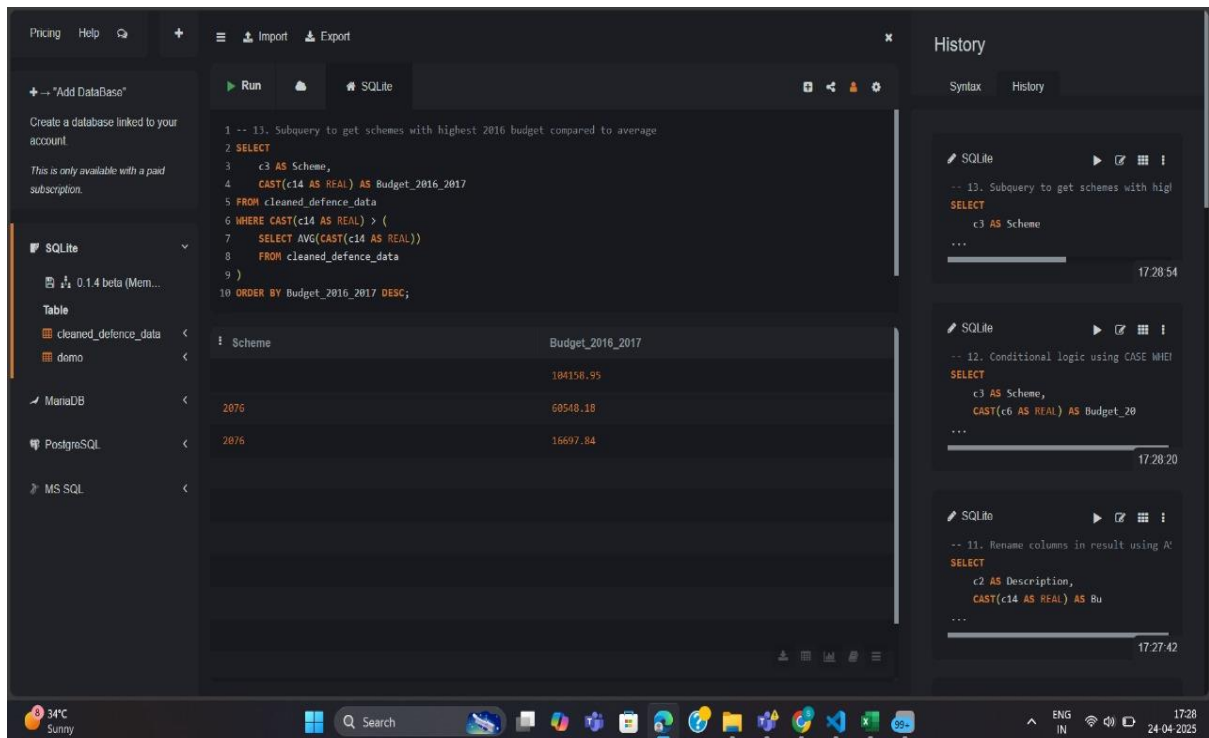
```
1 -- 12. Conditional logic using CASE WHEN
2 SELECT
3   c3 AS Scheme,
4   CAST(c6 AS REAL) AS Budget_2015_2016,
5   CASE
6     WHEN CAST(c6 AS REAL) > 10000 THEN 'High Budget'
7     WHEN CAST(c6 AS REAL) BETWEEN 5000 AND 10000 THEN 'Medium Budget'
8     ELSE 'Low Budget'
9   END AS Budget_Category
10 FROM cleaned_defence_data;
```

The results table shows the following data:

Scheme	Budget_2015_2016	Budget_Category
Code	0	Low Budget
2076	49647.95	High Budget
2076	967.53	Low Budget
2076	4841.8	Low Budget
2076	2676.16	Low Budget
2076	494.96	Low Budget
2076	1781.39	Low Budget
2076	768.96	Low Budget
2076	13955.3	High Budget

The History panel on the right shows the execution history of the query, including the SQL code and the time taken to execute.

13.



The screenshot shows the Elevate Labs SQL editor interface. The main query window displays a SQL query for a subquery to get schemes with highest 2016 budget compared to average. The query is as follows:

```
1 -- 13. Subquery to get schemes with highest 2016 budget compared to average
2 SELECT
3   c3 AS Scheme,
4   CAST(c14 AS REAL) AS Budget_2016_2017
5 FROM cleaned_defence_data
6 WHERE CAST(c14 AS REAL) > (
7   SELECT AVG(CAST(c14 AS REAL))
8   FROM cleaned_defence_data
9 )
10 ORDER BY Budget_2016_2017 DESC;
```

The results table shows the following data:

Scheme	Budget_2016_2017
	184158.95
2076	60548.18
2076	16697.84

The History panel on the right shows the execution history of the query, including the SQL code and the time taken to execute.