

-- Task 3: SQL for Data Analysis
-- Dataset: Ecommerce_SQL_Database

/* Q1: Difference between WHERE and HAVING */
-- WHERE filters before aggregation

The screenshot shows the DB Browser for SQLite interface. The SQL editor contains the following query:

```
1 SELECT * FROM orders WHERE order_amount > 500;
2
```

The results pane displays a single row of data:

order_id	customer_id	order_amount	order_date	
1	105	4	800	2025-07-15

The status bar at the bottom indicates "Execution finished without errors. Result: 1 rows returned in 15ms. At line 1: SELECT * FROM orders WHERE order_amount > 500;".

-- HAVING filters after aggregation

The screenshot shows the DB Browser for SQLite interface. The SQL editor contains the following query:

```
1 SELECT customer_id, SUM(order_amount) AS total_spent
2 FROM orders
3 GROUP BY customer_id
4 HAVING total_spent > 500;
```

The results pane displays two rows of data:

customer_id	total_spent	
1	650	
2	4	800

The status bar at the bottom indicates "Execution finished without errors. Result: 2 rows returned in 14ms. At line 1: SELECT customer_id, SUM(order_amount) AS total_spent FROM orders GROUP BY customer_id HAVING total_spent > 500;".

/* Q2: Types of Joins */

-- INNER JOIN

The screenshot shows the DB Browser for SQLite interface. The SQL editor contains the following query:

```
1 SELECT c.name, o.order_id
2 FROM customers c
3 INNER JOIN orders o ON c.customer_id = o.customer_id;
```

The results pane displays the following data:

	name	order_id
1	Alice	101
2	Bob	102
3	Alice	103
4	Charlie	104
5	David	105
6	Bob	106
7	Eva	107

The execution log shows: "Execution finished without errors. Result: 7 rows returned in 31ms. At line 1: SELECT c.name, o.order_id FROM customers c INNER JOIN orders o ON c.customer_id = o.customer_id;".

-- LEFT JOIN

The screenshot shows the DB Browser for SQLite interface. The SQL editor contains the following query:

```
1 SELECT c.name, o.order_id
2 FROM customers c
3 LEFT JOIN orders o ON c.customer_id = o.customer_id;
```

The results pane displays the following data:

	name	order_id
1	Alice	101
2	Alice	103
3	Bob	102
4	Bob	106
5	Charlie	104
6	David	105
7	Eva	107

The execution log shows: "Execution finished without errors. Result: 7 rows returned in 18ms. At line 1: SELECT c.name, o.order_id FROM customers c LEFT JOIN orders o ON c.customer_id = o.customer_id;".

-- RIGHT JOIN

DB Browser for SQLite - C:\Users\DelI\Downloads\task4_08-08-2025.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

SQL 1*

```
1 SELECT c.name, o.order_id
2 FROM customers c
3 RIGHT JOIN orders o ON c.customer_id = o.customer_id;
```

	name	order_id
1	Alice	101
2	Alice	103
3	Bob	102
4	Bob	106
5	Charlie	104
6	David	105
7	Eva	107

Execution finished without errors.
Result: 7 rows returned in 15ms
At line 1:
SELECT c.name, o.order_id
FROM customers c
RIGHT JOIN orders o ON c.customer_id = o.customer_id;

Edit Database Cell

Mode: Text

NULL

No cell active.
Type: NULL; Size: 0 bytes

Remote

Identity Select an identity to connect

DBHub.io Local Current Database

Name Last modified Size

SQL Log Plot DB Schema Remote

UTF-8

/* Q3: Average Revenue Per User */

DB Browser for SQLite - C:\Users\DelI\Downloads\task4_08-08-2025.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

SQL 1*

```
1 SELECT AVG(total_spent) AS avg_revenue_per_user
2 FROM (
3     SELECT customer_id, SUM(order_amount) AS total_spent
4     FROM orders
5     GROUP BY customer_id
6 ) AS revenue_table;
```

	avg_revenue_per_user
1	504.0

Execution finished without errors.
Result: 1 rows returned in 12ms
At line 1:
SELECT AVG(total_spent) AS avg_revenue_per_user
FROM (
 SELECT customer_id, SUM(order_amount) AS total_spent
 FROM orders
)

Edit Database Cell

Mode: Text

NULL

No cell active.
Type: NULL; Size: 0 bytes

Remote

Identity Select an identity to connect

DBHub.io Local Current Database

Name Last modified Size

SQL Log Plot DB Schema Remote

UTF-8

/* Q4: Subqueries */

The screenshot shows the DB Browser for SQLite interface. The SQL editor contains the following query:

```
2 FROM customers
3 WHERE customer_id IN (
4   SELECT customer_id
5   FROM orders
6   GROUP BY customer_id
7   HAVING SUM(order_amount) > (
8     SELECT AVG(order_amount) FROM orders
9   )
10 );
```

The results pane displays a table with one column, 'name', and four rows:

	name
1	Alice
2	Bob
3	David
4	Eva

The execution log shows: "Execution finished without errors. Result: 4 rows returned in 20ms. At line 1: SELECT name FROM customers WHERE customer_id IN (SELECT customer_id".

/* Q5: Create View */

The screenshot shows the DB Browser for SQLite interface. The SQL editor contains the following query:

```
1 CREATE VIEW customer_spending AS
2 SELECT customer_id, SUM(order_amount) AS total_spent
3 FROM orders
4 GROUP BY customer_id;
5
6 SELECT * FROM customer_spending WHERE total_spent > 500;
```

The results pane displays a table with two columns, 'customer_id' and 'total_spent', and two rows:

	customer_id	total_spent
1	1	650
2	4	800

The execution log shows: "Execution finished without errors. Result: 2 rows returned in 23ms. At line 6: SELECT * FROM customer_spending WHERE total_spent > 500;"

/* Q6: Handling NULL Values */

DB Browser for SQLite - C:\Users\De\Downloads\task4_08-08-2025.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

SQL 1*

```
1 SELECT COALESCE(city, 'Unknown') AS customer_city
2 FROM customers;
```

customer_city
1 New York
2 Los Angeles
3 Unknown
4 Chicago
5 Houston

Execution finished without errors.
Result: 5 rows returned in 12ms
At line 1:
SELECT COALESCE(city, 'Unknown') AS customer_city
FROM customers;

Edit Database Cell

Mode: Text

NULL

No cell active.
Type: NULL; Size: 0 bytes

Remote

Identity Select an identity to connect

DBHub.io Local Current Database

Name Last modified Size

SQL Log Plot DB Schema Remote

UTF-8

/*Q7.Create an Index on order_date */

DB Browser for SQLite - C:\Users\De\Downloads\task4_08-08-2025.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

SQL 1*

```
1 CREATE INDEX idx_order_date ON orders(order_date);
2
```

Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE INDEX idx_order_date ON orders(order_date);

Edit Database Cell

Mode: Text

NULL

No cell active.
Type: NULL; Size: 0 bytes

Remote

Identity Select an identity to connect

DBHub.io Local Current Database

Name Last modified Size

SQL Log Plot DB Schema Remote

UTF-8

Create a Composite Index

The screenshot shows the DB Browser for SQLite interface. The main editor contains the following SQL command:

```
1 CREATE INDEX idx_customer_date ON orders(customer_id, order_date);
2
```

The execution log at the bottom shows:

```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE INDEX idx_customer_date ON orders(customer_id, order_date);
```

The right-hand pane shows the 'Edit Database Cell' window with 'Mode: Text' and 'Type: NULL; Size: 0 bytes'. The 'Remote' tab is active, showing a table with columns 'Name', 'Last modified', and 'Size'.

Check Existing Indexes

The screenshot shows the DB Browser for SQLite interface. The main editor contains the following SQL commands:

```
1 PRAGMA index_list('orders');
2 PRAGMA index_list('customers');
3
```

The execution log at the bottom shows:

```
Execution finished without errors.
Result: 1 rows returned in 21ms
At line 2:
PRAGMA index_list('customers');
```

The right-hand pane shows the 'Edit Database Cell' window with 'Mode: Text' and 'Type: NULL; Size: 0 bytes'. The 'Remote' tab is active, showing a table with columns 'Name', 'Last modified', and 'Size'.

Drop an Unused Index

DB Browser for SQLite - C:\Users\DeIi\Downloads\task4_08-08-2025.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

SQL 1*

```
1 DROP INDEX IF EXISTS idx_order_date;
2
```

Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
DROP INDEX IF EXISTS idx_order_date;

Edit Database Cell

Mode: Text

NULL

No cell active.
Type: NULL; Size: 0 bytes

Apply

Remote

Identity Select an identity to connect

Upload

DBHub.io Local Current Database

Name	Last modified	Size
------	---------------	------

SQL Log Plot DB Schema Remote

UTF-8