SQL Training Database

Objective

The objective of this project is to design and implement a fully functional SQL training database that simulates a real-world Australian drone retailer. The project demonstrates practical SQL knowledge gained through formal education, including database schema design, table relationships, and data seeding. It serves as a hands-on training resource for users at both beginner and advanced levels to practice querying, joining, and analysing realistic commercial data within PostgreSQL (pgAdmin 4).

Dataset

The dataset represents a comprehensive simulation of an Australian drone retail and service business. It includes four interrelated tables Drones, Customers, Payments, and Maintenance Logs. Designed to reflect realistic operational data for a commercial retailer.

The dataset contains over 300 fully synthesised records, featuring authentic Australian names, addresses, phone formats, and transaction histories in AUD. Drone models and manufacturers such as DJI, Autel Robotics, and Skydio are represented with realistic specifications and pricing. All foreign key relationships are validated to ensure relational integrity, making the database suitable for both structural and analytical SQL practice.

This dataset enables users to explore a wide range of learning outcomes, from basic SELECT queries and joins to more advanced concepts such as data aggregation, indexing, and schema optimisation within PostgreSQL.

Methodology

The project was developed using PostgreSQL within pgAdmin 4, following a structured approach to database design and data validation. The schema was first planned to ensure logical relationships between tables, establishing one-to-many connections between customers, payments, and drones, as well as linking maintenance activities to individual drones.

Each table was defined with appropriate data types, constraints, and indexes to enforce integrity and optimise performance. Key features include the use of foreign keys, CHECK constraints for payment methods and status fields, and indexed date columns to support efficient analytical queries.
A custom-generated dataset of over 300 ultra-realistic records was then seeded using an SQL file that includes both CREATE TABLE and INSERT statements. The data generation process incorporated Australian-specific naming, currency (AUD), and address conventions to enhance realism.

The database was tested with a range of queries, from beginner-level joins and filters to advanced aggregations and relationship checks to confirm full functionality and consistency across all tables.

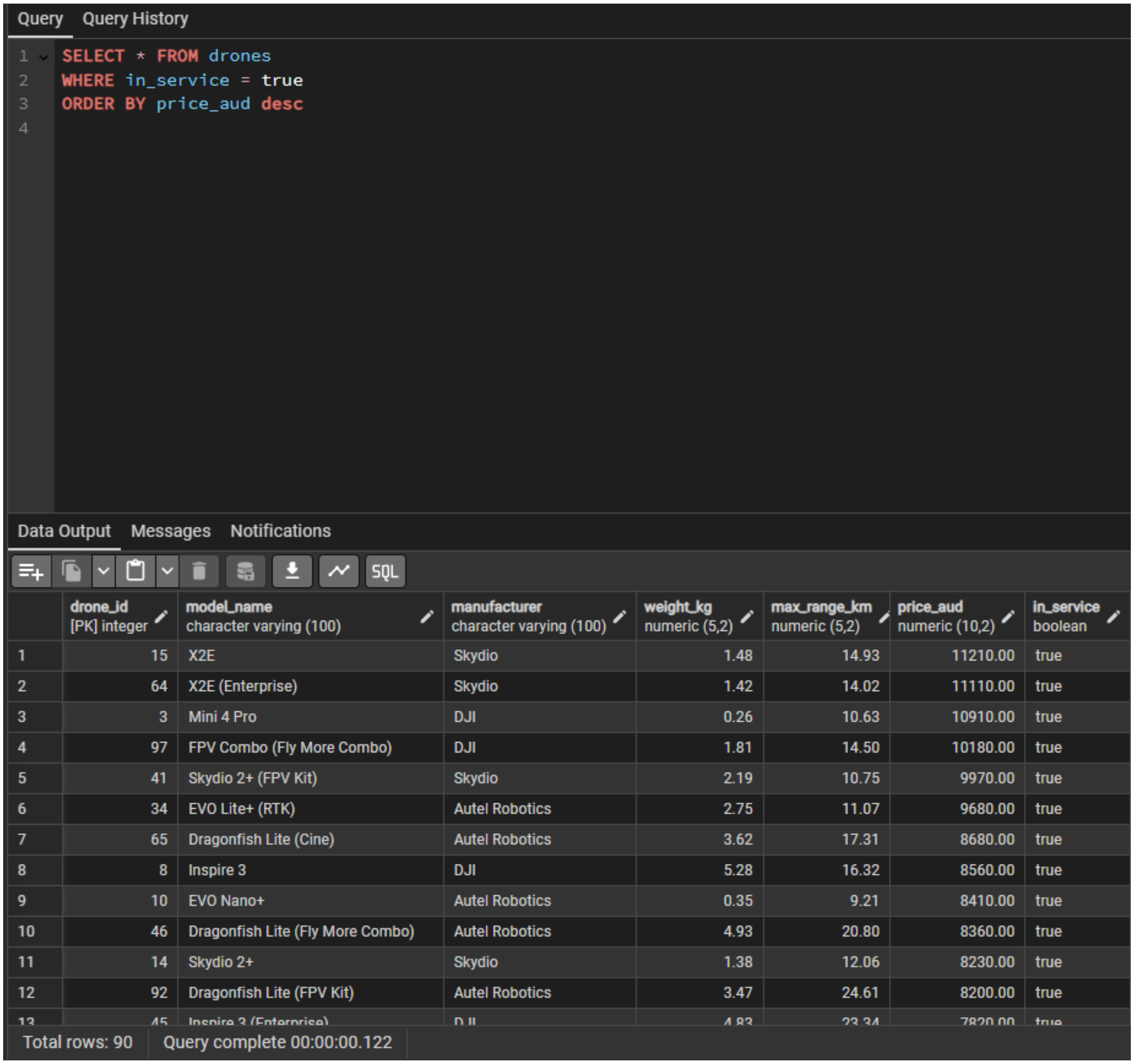
Functionality Test and Example Queries

Basic Queries:

- **Management would like to see a list of stocked drones from most expensive to least expensive.**

Syntax: SELECT * FROM drones
WHERE in_service = true
ORDER BY price_aud desc

Output: Query correctly calls on the 100 drones listed in descending order by price (- the 10 drones which arent currently in service).



```sql
SELECT * FROM drones
WHERE in_service = true
ORDER BY price_aud desc
```

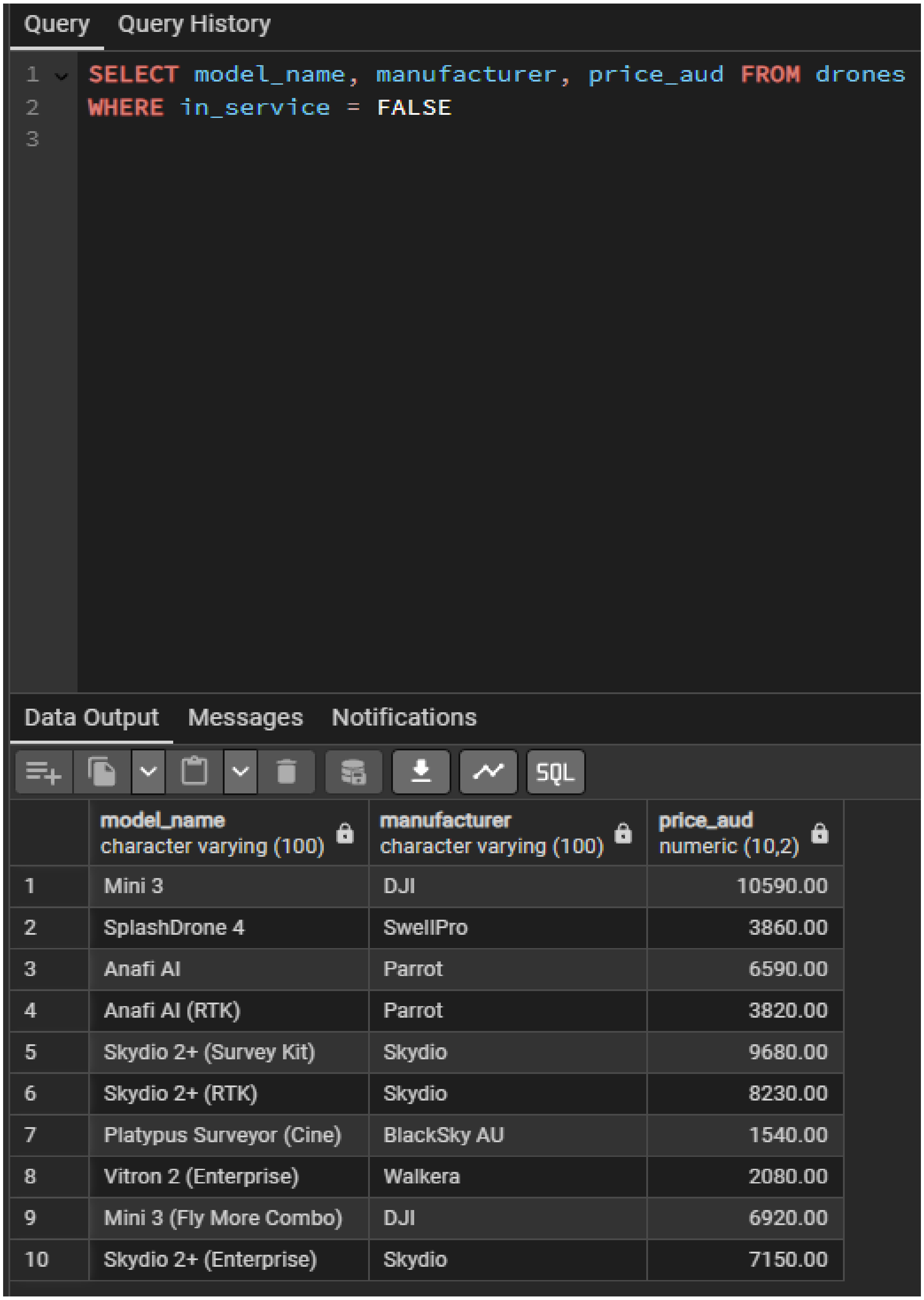
	drone_id [PK] integer	model_name character varying (100)	manufacturer character varying (100)	weight_kg numeric (5,2)	max_range_km numeric (5,2)	price_aud numeric (10,2)	in_service boolean
1	15	X2E	Skydio	1.48	14.93	11210.00	true
2	64	X2E (Enterprise)	Skydio	1.42	14.02	11110.00	true
3	3	Mini 4 Pro	DJI	0.26	10.63	10910.00	true
4	97	FPV Combo (Fly More Combo)	DJI	1.81	14.50	10180.00	true
5	41	Skydio 2+ (FPV Kit)	Skydio	2.19	10.75	9970.00	true
6	34	EVO Lite+ (RTK)	Autel Robotics	2.75	11.07	9680.00	true
7	65	Dragonfish Lite (Cine)	Autel Robotics	3.62	17.31	8680.00	true
8	8	Inspire 3	DJI	5.28	16.32	8560.00	true
9	10	EVO Nano+	Autel Robotics	0.35	9.21	8410.00	true
10	46	Dragonfish Lite (Fly More Combo)	Autel Robotics	4.93	20.80	8360.00	true
11	14	Skydio 2+	Skydio	1.38	12.06	8230.00	true
12	92	Dragonfish Lite (FPV Kit)	Autel Robotics	3.47	24.61	8200.00	true
13	45	Inspire 3 (Enterprise)	DJI	4.83	23.34	7820.00	true

Total rows: 90   Query complete 00:00:00.122

- **Management would like to know which drones are out of service but is only interested in the model, manufacturer and price of each drone**

Syntax: SELECT model_name, manufacturer, price_aud FROM drones
WHERE in_service = FALSE

Output: Query correctly calls on the 10 out of service drones with the required columns.



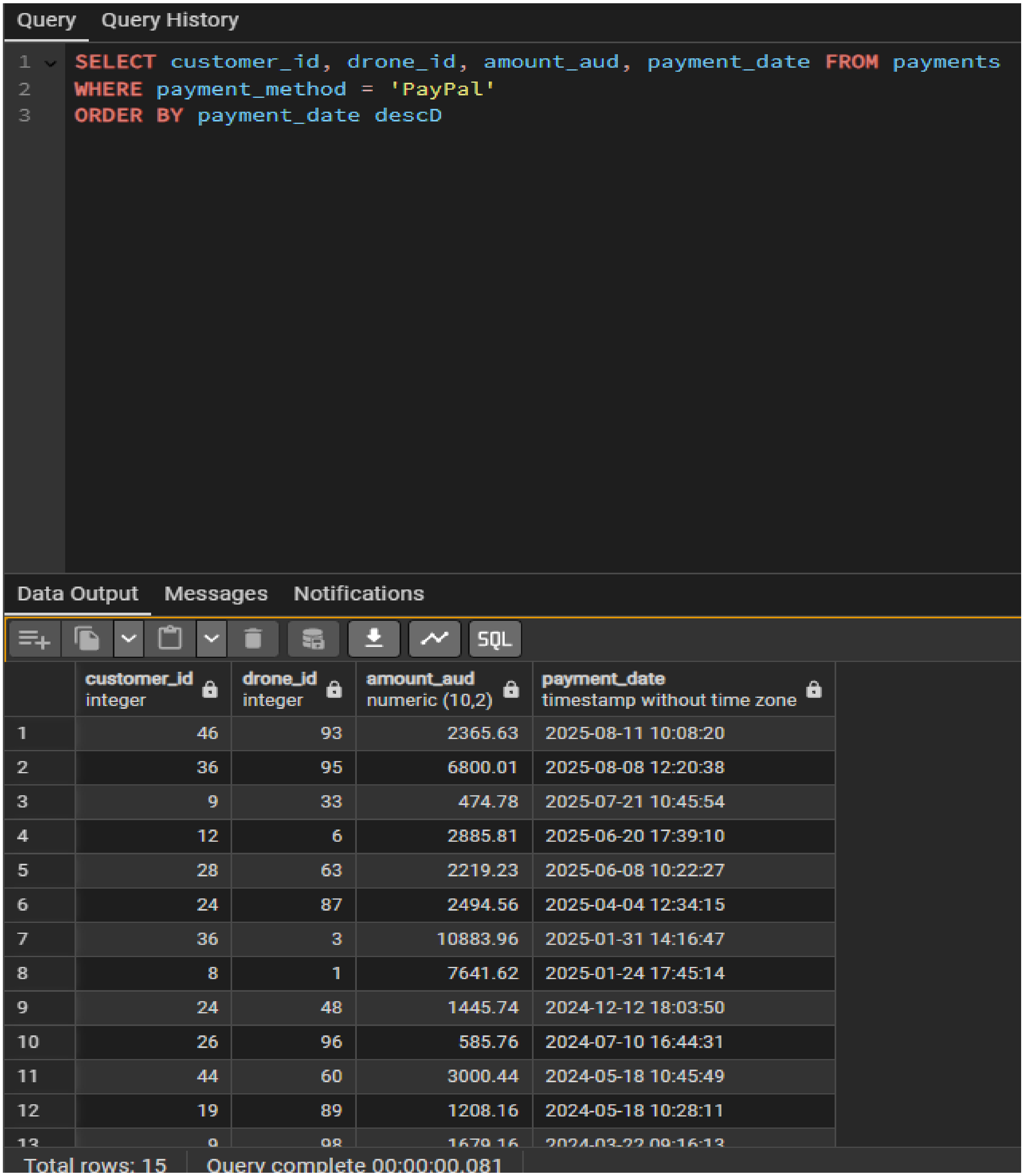
```
SELECT model_name, manufacturer, price_aud FROM drones
WHERE in_service = FALSE
```

	model_name character varying (100)	manufacturer character varying (100)	price_aud numeric (10,2)
1	Mini 3	DJI	10590.00
2	SplashDrone 4	SwellPro	3860.00
3	Anafi AI	Parrot	6590.00
4	Anafi AI (RTK)	Parrot	3820.00
5	Skydio 2+ (Survey Kit)	Skydio	9680.00
6	Skydio 2+ (RTK)	Skydio	8230.00
7	Platypus Surveyor (Cine)	BlackSky AU	1540.00
8	Vitron 2 (Enterprise)	Walkera	2080.00
9	Mini 3 (Fly More Combo)	DJI	6920.00
10	Skydio 2+ (Enterprise)	Skydio	7150.00

- **Management wants to see if a recent paypal promotion has been effective**

Syntax: SELECT customer_id, drone_id, amount_aud, payment_date FROM payments
WHERE payment_method = 'PayPal'
ORDER BY payment_date desc

Output: All paypal payments are correctly shown in descending order.



```sql
SELECT customer_id, drone_id, amount_aud, payment_date FROM payments
WHERE payment_method = 'PayPal'
ORDER BY payment_date descD
```

	customer_id integer	drone_id integer	amount_aud numeric (10,2)	payment_date timestamp without time zone
1	46	93	2365.63	2025-08-11 10:08:20
2	36	95	6800.01	2025-08-08 12:20:38
3	9	33	474.78	2025-07-21 10:45:54
4	12	6	2885.81	2025-06-20 17:39:10
5	28	63	2219.23	2025-06-08 10:22:27
6	24	87	2494.56	2025-04-04 12:34:15
7	36	3	10883.96	2025-01-31 14:16:47
8	8	1	7641.62	2025-01-24 17:45:14
9	24	48	1445.74	2024-12-12 18:03:50
10	26	96	585.76	2024-07-10 16:44:31
11	44	60	3000.44	2024-05-18 10:45:49
12	19	89	1208.16	2024-05-18 10:28:11
13	9	98	1679.16	2024-03-22 09:16:13

Total rows: 15 Query complete 00:00:00.081

Advanced Queries:

- **Management wants to see who made the most recent purchase, what they bought, how much was spent, how they paid, and when the purchase was made**

Syntax: SELECT customers.full_name, drones.manufacturer, drones.model_name, payments.amount_aud, payments.payment_method, payments.payment_date

FROM payments

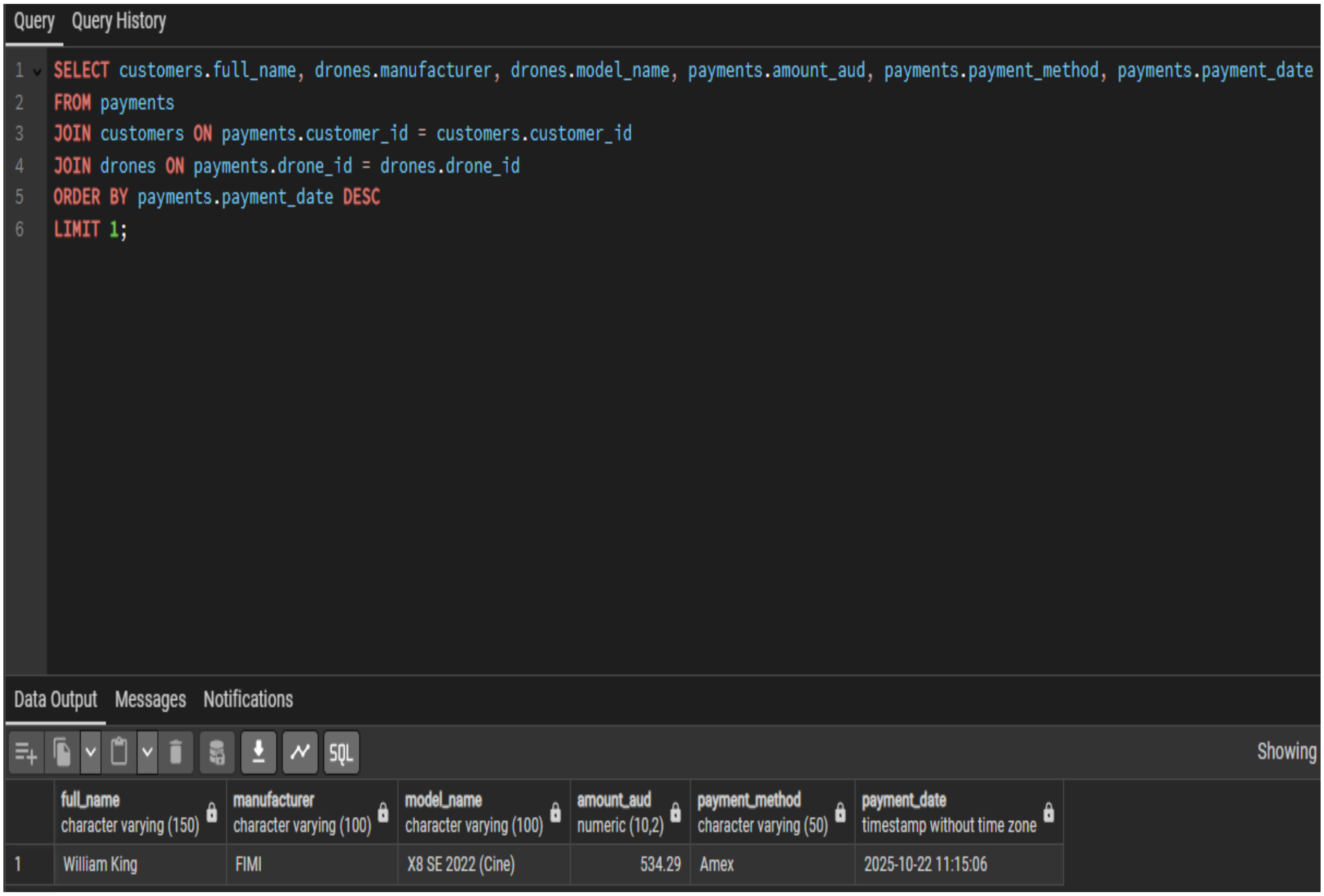
JOIN customers ON payments.customer_id = customers.customer_id

JOIN drones ON payments.drone_id = drones.drone_id

ORDER BY payments.payment_date DESC

LIMIT 1

Output: Correctly calls the most recent purchase with the requested details



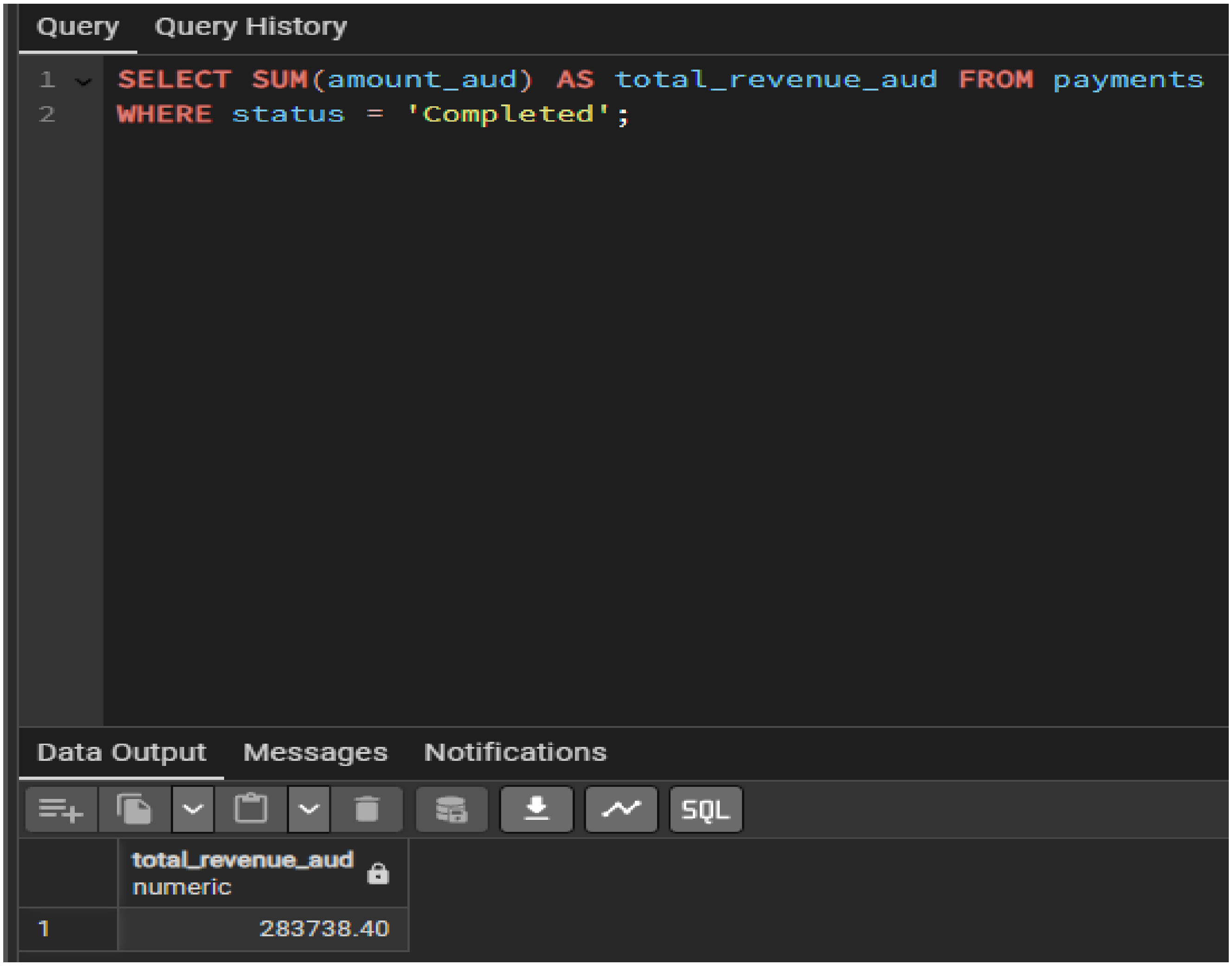
```sql
SELECT customers.full_name, drones.manufacturer, drones.model_name, payments.amount_aud, payments.payment_method, payments.payment_date
FROM payments
JOIN customers ON payments.customer_id = customers.customer_id
JOIN drones ON payments.drone_id = drones.drone_id
ORDER BY payments.payment_date DESC
LIMIT 1;
```

	full_name character varying (150)	manufacturer character varying (100)	model_name character varying (100)	amount_aud numeric (10,2)	payment_method character varying (50)	payment_date timestamp without time zone
1	William King	FIMI	X8 SE 2022 (Cine)	534.29	Amex	2025-10-22 11:15:06

- **Management would like to know the total generated revenue**

Syntax: SELECT SUM(amount_aud) AS total_revenue_aud FROM payments
WHERE status = 'Completed'

Output: The total payment in aud is correctly summed and shown as “total revenue aud” instead of payments



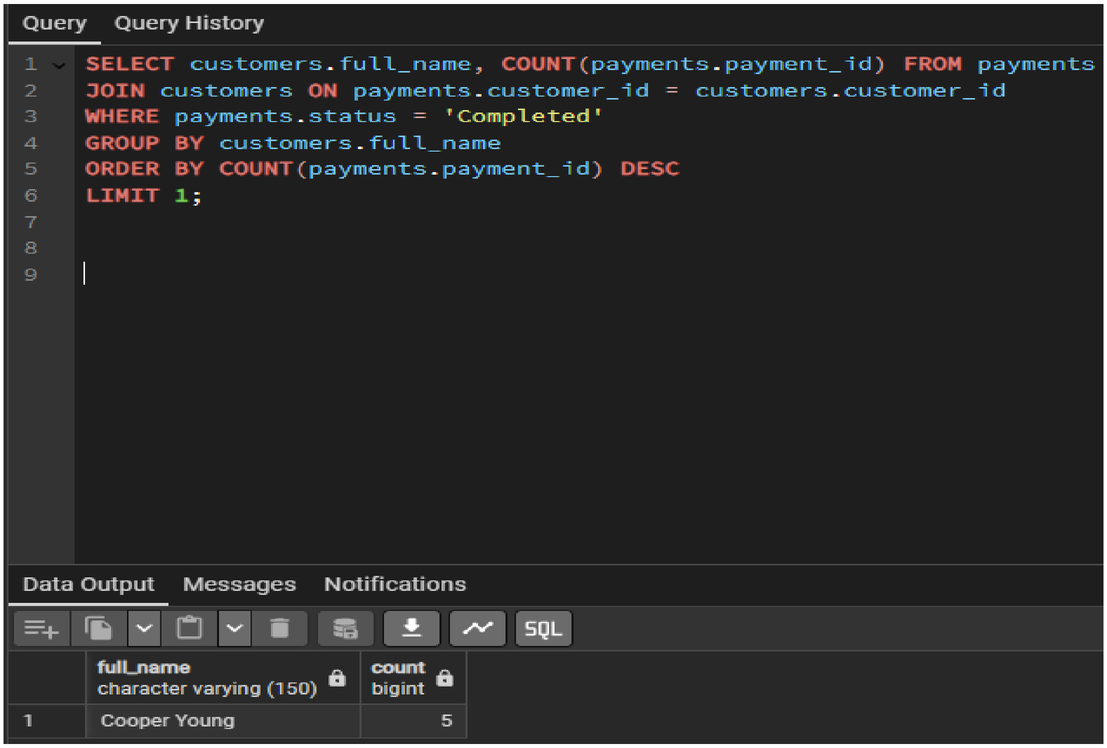
```
SELECT SUM(amount_aud) AS total_revenue_aud FROM payments
WHERE status = 'Completed';
```

	total_revenue_aud numeric
1	283738.40

- **Management wants to know their customer with the highest orders**

Syntax: SELECT customers.full_name, COUNT(payments.payment_id) FROM payments
JOIN customers ON payments.customer_id = customers.customer_id
WHERE payments.status = 'Completed'
GROUP BY customers.full_name
ORDER BY COUNT(payments.payment_id) DESC
LIMIT 1;

Output: The customer with the highest amount of total purchases is correctly called



```sql
SELECT customers.full_name, COUNT(payments.payment_id) FROM payments
JOIN customers ON payments.customer_id = customers.customer_id
WHERE payments.status = 'Completed'
GROUP BY customers.full_name
ORDER BY COUNT(payments.payment_id) DESC
LIMIT 1;
```

	full_name character varying (150)	count bigint
1	Cooper Young	5

Results

The SQL Training Database Project successfully demonstrates a complete, functioning relational database built for realistic training and analytical use. All four tables: Drones, Customers, Payments, and Maintenance Logs were verified to operate seamlessly within PostgreSQL using pgAdmin 4.

The database was tested through a range of queries, from simple data retrieval to complex multi-table joins. Basic queries validated the correctness of data seeding and table relationships, while advanced queries confirmed referential integrity and indexing efficiency. Examples include identifying the most recent purchase, calculating total revenue, and determining customers with the highest purchase frequency.

The dataset’s realism, integrity, and structure provide a strong foundation for SQL skill development. This allows users to explore SQL basicsin a controlled, business-oriented environment.

Conclusion

This project demonstrates the ability to design, implement, and validate a realistic SQL database from concept to execution. By developing a complete training environment based on an Australian drone retail business, it showcases practical database design principles, indexing, and query optimisation, applied in a hands-on context.

The dataset’s realism and structural integrity make it a valuable tool for workforce training and self-directed SQL learning, supporting a full range of exercises from beginner queries to advanced analytical operations. The successful creation and testing of this database highlight both technical proficiency in PostgreSQL and the capacity to translate theoretical knowledge into practical, business-relevant database solutions.