Table of Contents

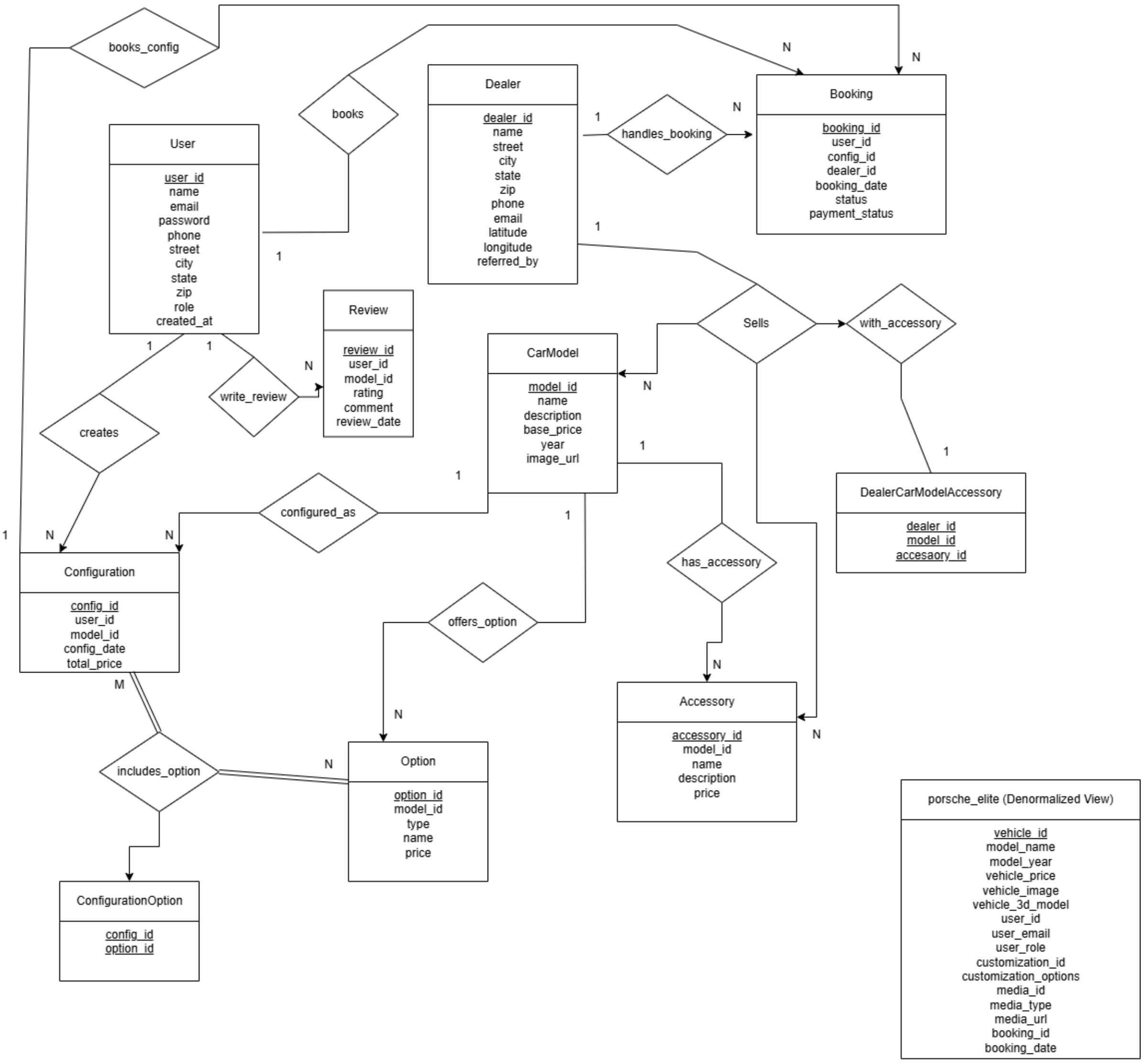
Project Abstract

The Porsche Elite Webspace project is a comprehensive web-based platform designed to provide a visually rich, interactive, and user-friendly interface for showcasing Porsche vehicles. Developed using modern web technologies such as React, TypeScript, Tailwind CSS, and Bootstrap, this application enables prospective customers to seamlessly browse, explore, and interact with various Porsche models. Key features include dynamic 3D model interaction, theme toggling, embedded promotional videos, vehicle customization, and a responsive layout optimized for both desktop and mobile devices.

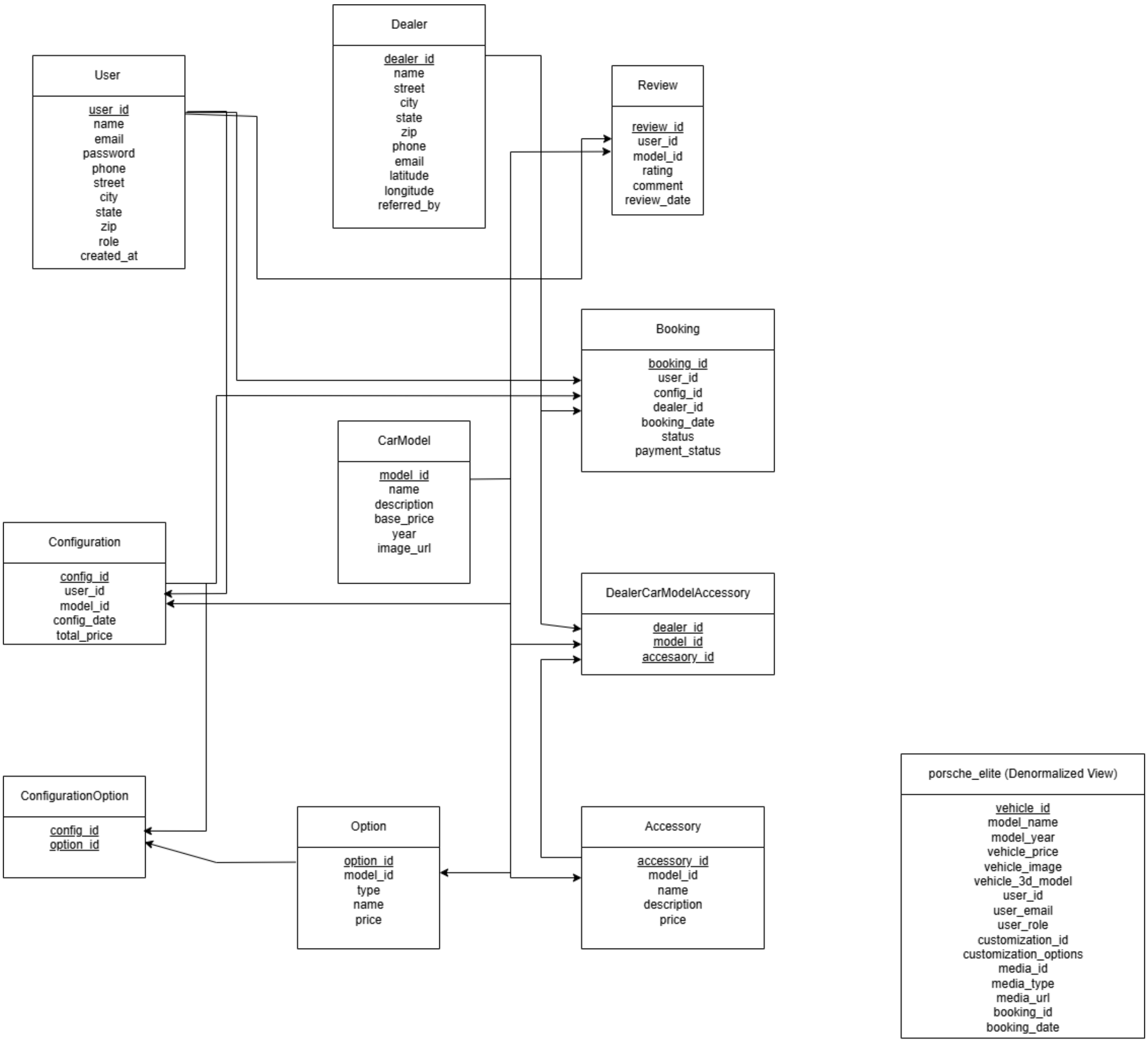
The system serves two primary user roles: customers and administrators. Customers can access detailed specifications, explore car galleries, watch design and performance videos, and book vehicles with personalized options. Administrators can manage website content, vehicle databases, user accounts, and access analytics through a dedicated admin interface. The backend infrastructure is supported by Node.js and PostgreSQL, ensuring scalability and data integrity.

Overall, the project aims to replicate a premium showroom experience in a digital environment, enhancing customer engagement and streamlining vehicle information management for the brand.

Entity-Relation Diagram



Schema Diagram



Relational Schema

User(user_id, name, email, password, phone, street, city, state, zip, role, created_at)

CarModel(model_id, name, description, base_price, year, image_url)

Dealer(dealer_id, name, street, city, state, zip, phone, email, latitude, longitude, referred_by)

Option(option_id, model_id, type, name, price)

Accessory(accessory_id, model_id, name, description, price)

Configuration(config_id, user_id, model_id, config_date, total_price)

ConfigurationOption(config_id, option_id)

Booking(booking_id, user_id, config_id, dealer_id, booking_date, status, payment_status)

Review(review_id, user_id, model_id, rating, comment, review_date)

DealerCarModelAccessory(dealer_id, model_id, accessory_id)

Normalization Forms

Normalization Forms

Normalization is the process of organizing a database to reduce redundancy and improve data integrity.

Step 1:Universal Relation

porsche_elite (vehicle_id, model_name, model_year, vehicle_price, vehicle_image, vehicle_3d_model, user_id, user_email, user_role, customization_id, customization_options, media_id, media_type, media_url, booking_id, booking_date)

Functional Dependancies:

1. vehicle_id → model_name, model_year, vehicle_price, vehicle_image, vehicle_3d_model
2. user_id → user_email, user_role
3. customization_id → customization_options
4. media_id → media_type, media_url
5. booking_id → user_id, vehicle_id, booking_date
6. (user_id, vehicle_id) → customization_id
7. (vehicle_id, media_id) → no additional attributes
8. (user_id, vehicle_id) → no additional attributes

Step 2: First Normal Form (1NF)

1NF Rule: Eliminate multi-valued attributes and ensure each attribute contains atomic values.

Since all the attributes in the universal relation are **not atomic**, it is **not in 1NF**.

Example:

- customization_options may store multiple values (like ["spoiler", "sunroof"]).
- A vehicle may have multiple media files (images, videos).

- A vehicle may be booked multiple times.

These repeating or multi-valued attributes violate the atomicity rule.

Updated Schema After Removing Multivalued Attributes:

1NF:

Porsche_Elite (1NF):
(vehicle_id, model_name, model_year, vehicle_price, vehicle_image, vehicle_3d_model, user_id, user_email, user_role, customization_id, customization_option, media_id, media_type, media_url, booking_id, booking_date)

Note:

- customization_option is now atomic (only one option per row).
- One row per media item per vehicle.
- One row per booking entry.

Step 3: Second Normal Form (2NF)

2NF Rule: Eliminate partial dependencies, where non-prime attributes depend only on part of a composite key. This rule applies to tables with composite primary keys.

Actions:

1. Ensure all non-prime attributes depend on the **entire primary key**.

Updated Schema After Removing Partial Dependencies:

2NF:

Users:
(user_id, user_email, user_role)

Vehicles:
(vehicle_id, model_name, model_year, vehicle_price, vehicle_image, vehicle_3d_model)

Customizations:
(customization_id, vehicle_id, customization_option)

Media:
(media_id, vehicle_id, media_type, media_url)

Bookings:
(booking_id, vehicle_id, user_id, booking_date)

Step 4: Third Normal Form (3NF)

3NF Rule: Eliminate transitive dependencies, where non-prime attributes depend on other non-prime attributes.

Actions:

- Separate attributes that transitively depend on the primary key.

Example:

In the Users table, attributes like user_email and user_role depend directly on user_id.
In Vehicles, all non-key attributes depend directly on vehicle_id.
Hence, all transitive dependencies are eliminated.

The schema is already in **3NF**.

Step 5: Boyce-Codd Normal Form (BCNF)

BCNF Rule: A table is in BCNF if **every determinant is a candidate key**.

Actions:

- Ensure there are no non-trivial functional dependencies where a **non-candidate key** determines a candidate key.

Example:

- In the Vehicles table, vehicle_id is the only determinant and a candidate key.

- Similarly, all other tables use primary keys that uniquely determine other attributes.

  All the tables meet the **BCNF** criteria.

Final Schema (BCNF):

Users
(user_id, user_email, user_role)

Vehicles
(vehicle_id, model_name, model_year, vehicle_price, vehicle_image, vehicle_3d_model)

Customizations
(customization_id, vehicle_id, customization_option)

Media
(media_id, vehicle_id, media_type, media_url)

Bookings
(booking_id, vehicle_id, user_id, booking_date)

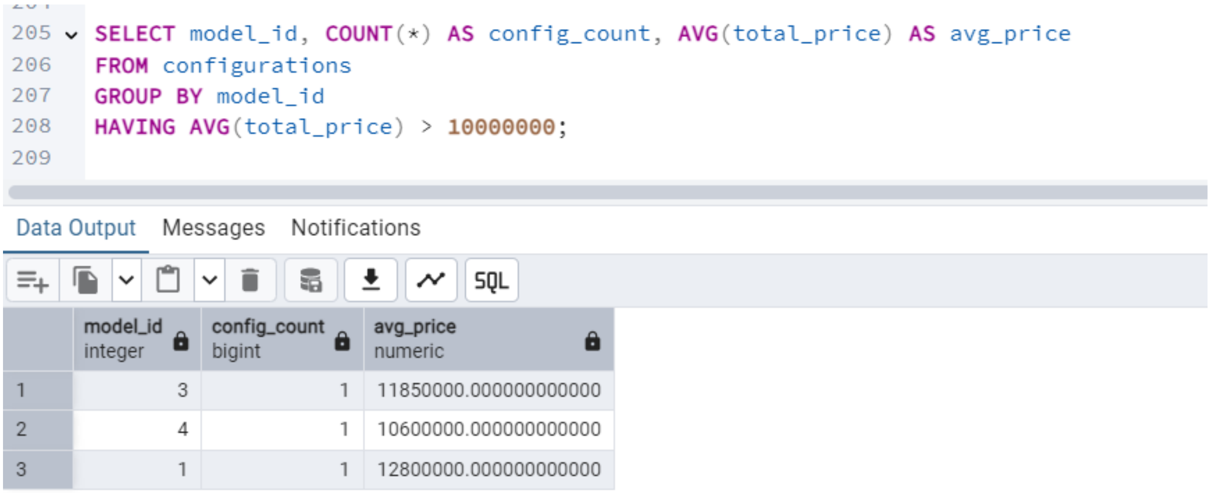
Data Definition Language - Database

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    phone VARCHAR(20),
    address_street VARCHAR(255),
    address_city VARCHAR(100),
    address_state VARCHAR(100),
    address_zip VARCHAR(20),
    role VARCHAR(50) CHECK (role IN ('customer', 'admin')) DEFAULT 'customer',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE car_models (
    model_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    base_price NUMERIC(10, 2) NOT NULL,
    year INT NOT NULL,
    image_url VARCHAR(255)
);

CREATE TABLE dealers (
    dealer_id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,

    address_street VARCHAR(255),

    address_city VARCHAR(100),

    address_state VARCHAR(100),

    address_zip VARCHAR(20),

    phone VARCHAR(20),

    email VARCHAR(255),

    latitude NUMERIC(10, 8),

    longitude NUMERIC(11, 8),

    referred_by INT,

    FOREIGN KEY (referred_by) REFERENCES dealers(dealer_id)

);


CREATE TABLE options (

    option_id SERIAL PRIMARY KEY,

    model_id INT NOT NULL,

    type VARCHAR(50) NOT NULL,

    name VARCHAR(100) NOT NULL,

    price NUMERIC(10, 2) NOT NULL,

    FOREIGN KEY (model_id) REFERENCES car_models(model_id)

);


CREATE TABLE accessories (

    accessory_id SERIAL PRIMARY KEY,

    model_id INT NOT NULL,

    name VARCHAR(100) NOT NULL,

    description TEXT,

    price NUMERIC(10, 2) NOT NULL,
    FOREIGN KEY (model_id) REFERENCES car_models(model_id)
);

CREATE TABLE configurations (
    config_id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    model_id INT NOT NULL,
    config_json JSONB NOT NULL,
    total_price NUMERIC(10, 2) NOT NULL,
    config_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (model_id) REFERENCES car_models(model_id)
);

-- Configuration-Option junction table (many-to-many)
CREATE TABLE configuration_options (
    config_id INT NOT NULL,
    option_id INT NOT NULL,
    PRIMARY KEY (config_id, option_id),
    FOREIGN KEY (config_id) REFERENCES configurations(config_id),
    FOREIGN KEY (option_id) REFERENCES options(option_id)
);

CREATE TABLE bookings (
    booking_id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    config_id INT NOT NULL,
    dealer_id INT NOT NULL,
    booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(50) CHECK (status IN ('pending', 'confirmed', 'cancelled')) DEFAULT 'pending',
    payment_status VARCHAR(50) CHECK (payment_status IN ('pending', 'completed', 'failed')) DEFAULT 'pending',
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (config_id) REFERENCES configurations(config_id),
    FOREIGN KEY (dealer_id) REFERENCES dealers(dealer_id)
);

-- Reviews table (weak entity)
CREATE TABLE reviews (
    review_id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    model_id INT NOT NULL,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    comment TEXT,
    review_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (model_id) REFERENCES car_models(model_id)
);

-- Dealer-CarModel-Accessory junction table (ternary relationship)
CREATE TABLE dealer_car_model_accessory (
    dealer_id INT NOT NULL,
    model_id INT NOT NULL,
    accessory_id INT NOT NULL,
    PRIMARY KEY (dealer_id, model_id, accessory_id),
    FOREIGN KEY (dealer_id) REFERENCES dealers(dealer_id),
    FOREIGN KEY (model_id) REFERENCES car_models(model_id),
    FOREIGN KEY (accessory_id) REFERENCES accessories(accessory_id)
);
```

Data Manipulation Language

i) Aggregate functions, Group by…having

Get the number of configurations and average total price per car model where the average price exceeds ₹1 crore.

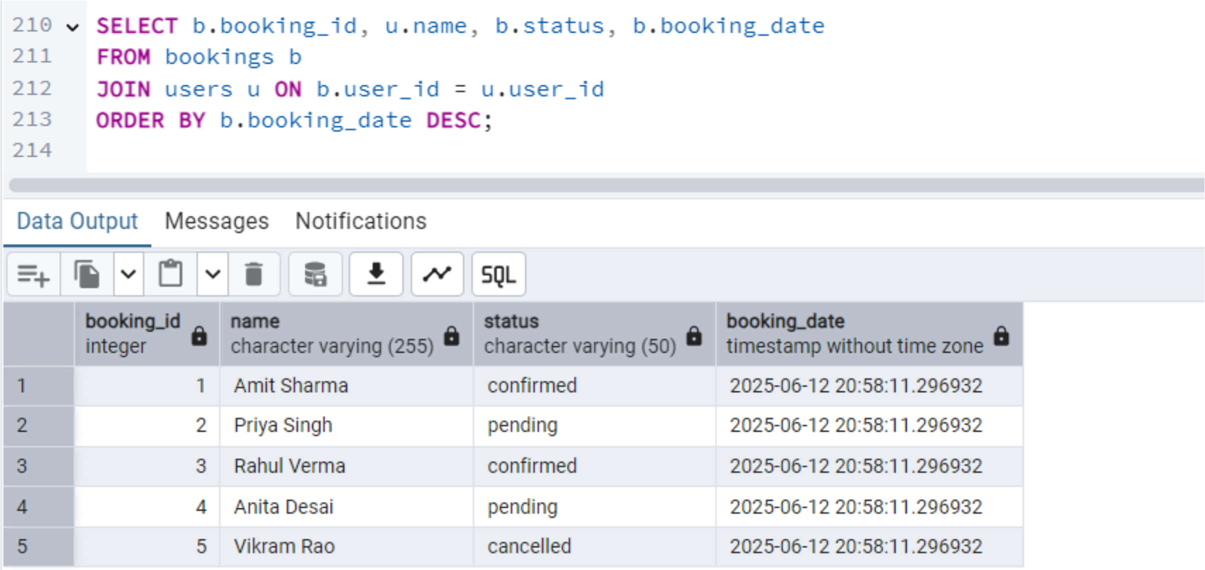


```sql
SELECT model_id, COUNT(*) AS config_count, AVG(total_price) AS avg_price
FROM configurations
GROUP BY model_id
HAVING AVG(total_price) > 10000000;
```

	model_id integer	config_count bigint	avg_price numeric
1	3	1	11850000.000000000000
2	4	1	10600000.000000000000
3	1	1	12800000.000000000000

ii) Order by

List all bookings ordered by booking_date descending, showing user name and booking status.

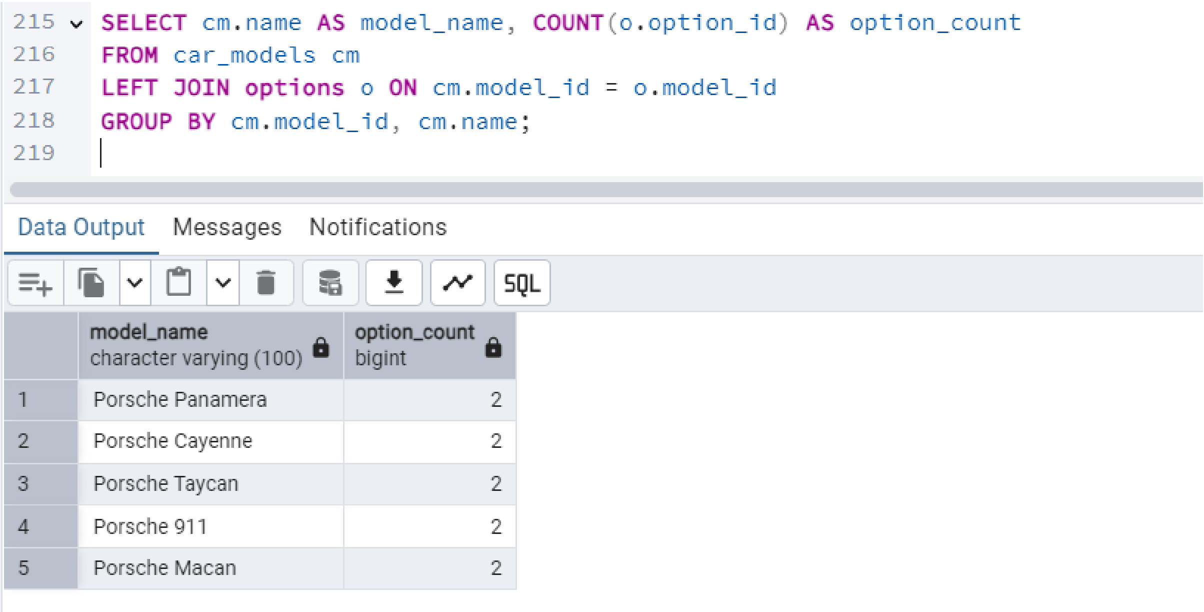


```sql
SELECT b.booking_id, u.name, b.status, b.booking_date
FROM bookings b
JOIN users u ON b.user_id = u.user_id
ORDER BY b.booking_date DESC;
```

	booking_id integer	name character varying (255)	status character varying (50)	booking_date timestamp without time zone
1	1	Amit Sharma	confirmed	2025-06-12 20:58:11.296932
2	2	Priya Singh	pending	2025-06-12 20:58:11.296932
3	3	Rahul Verma	confirmed	2025-06-12 20:58:11.296932
4	4	Anita Desai	pending	2025-06-12 20:58:11.296932
5	5	Vikram Rao	cancelled	2025-06-12 20:58:11.296932

iii) Join, Outer Join

List all car models and show the number of options they have (including models with zero options).

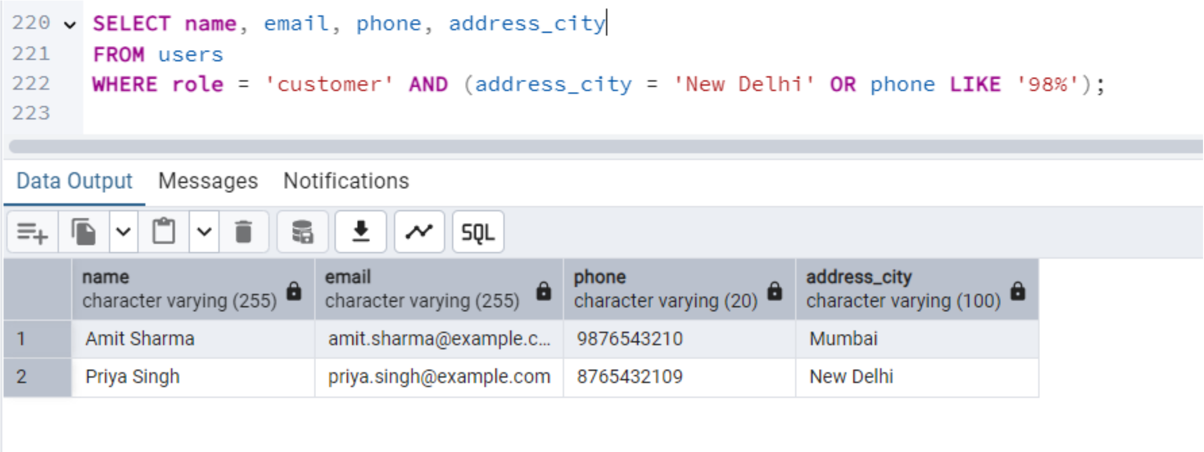


```sql
SELECT cm.name AS model_name, COUNT(o.option_id) AS option_count
FROM car_models cm
LEFT JOIN options o ON cm.model_id = o.model_id
GROUP BY cm.model_id, cm.name;

```

	model_name character varying (100)	option_count bigint
1	Porsche Panamera	2
2	Porsche Cayenne	2
3	Porsche Taycan	2
4	Porsche 911	2
5	Porsche Macan	2

iv) Query having Boolean operators

Get users who are customers and either from Delhi OR whose phone starts with '98'.

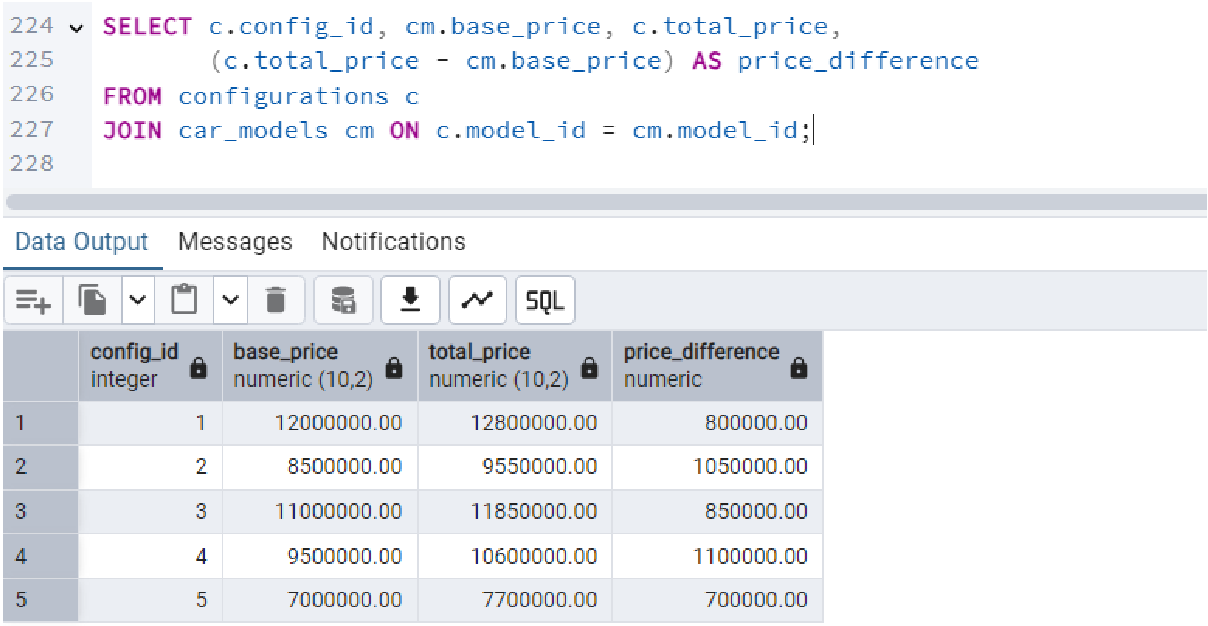


```sql
SELECT name, email, phone, address_city
FROM users
WHERE role = 'customer' AND (address_city = 'New Delhi' OR phone LIKE '98%');

```

	name character varying (255)	email character varying (255)	phone character varying (20)	address_city character varying (100)
1	Amit Sharma	amit.sharma@example.c...	9876543210	Mumbai
2	Priya Singh	priya.singh@example.com	8765432109	New Delhi

v) Query having arithmetic operators

Show configuration ID, base price, total price, and price difference for each configuration.

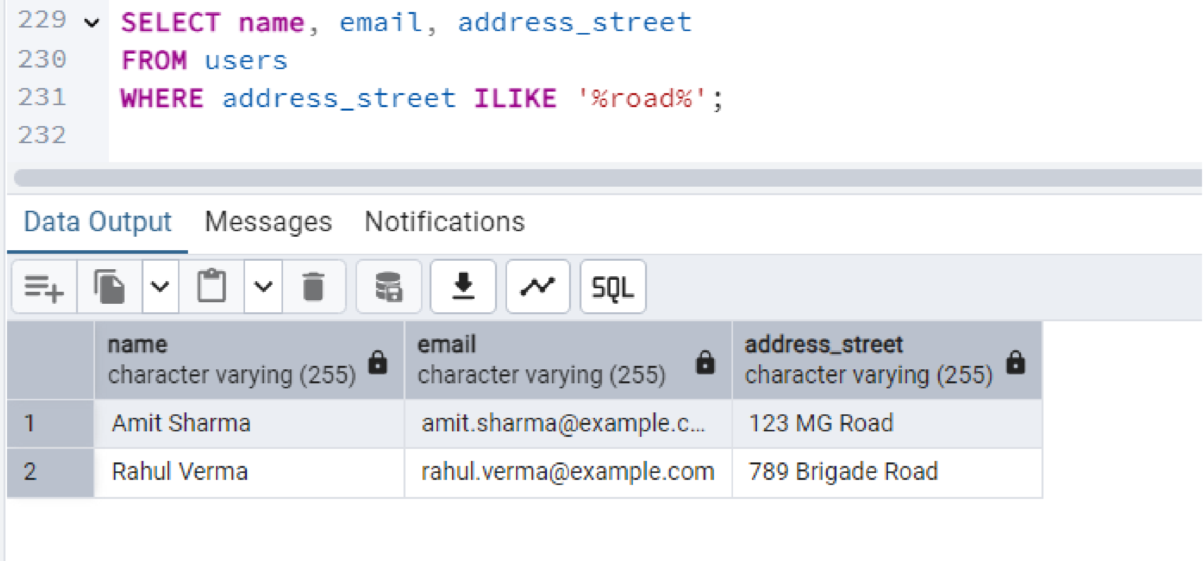


```sql
SELECT c.config_id, cm.base_price, c.total_price,
       (c.total_price - cm.base_price) AS price_difference
FROM configurations c
JOIN car_models cm ON c.model_id = cm.model_id;
```

	config_id integer	base_price numeric (10,2)	total_price numeric (10,2)	price_difference numeric
1	1	12000000.00	12800000.00	800000.00
2	2	8500000.00	9550000.00	1050000.00
3	3	11000000.00	11850000.00	850000.00
4	4	9500000.00	10600000.00	1100000.00
5	5	7000000.00	7700000.00	700000.00

vi) A search query using string operators

Find users whose address contains the word 'Road' (case-insensitive).

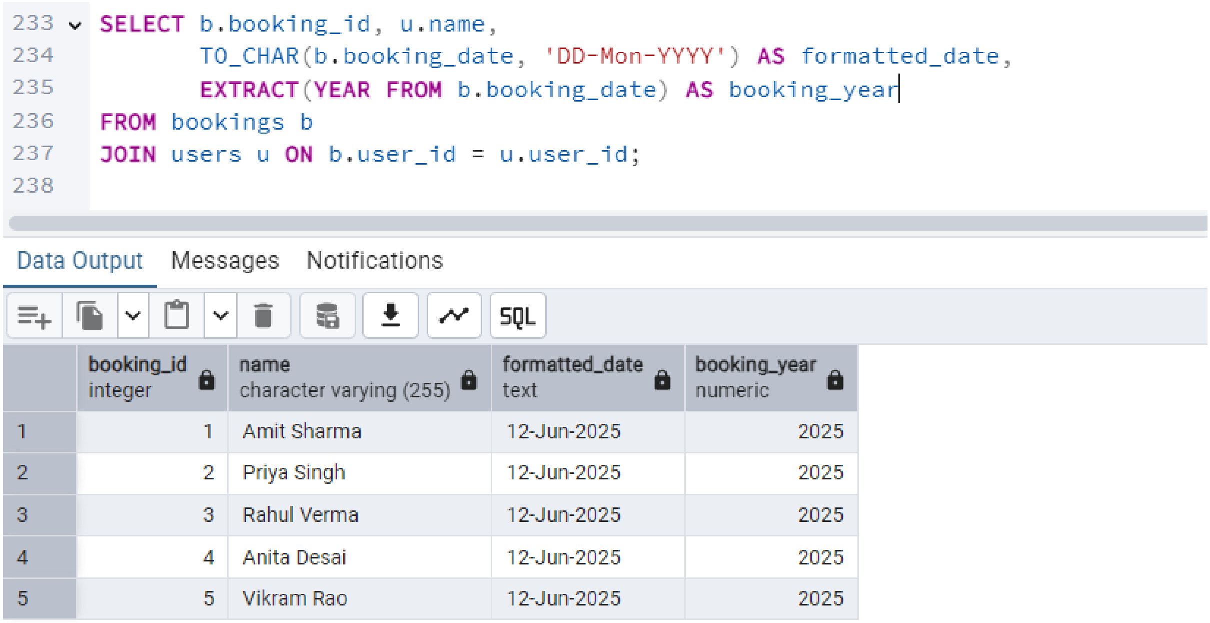


```sql
SELECT name, email, address_street
FROM users
WHERE address_street ILIKE '%road%';
```

	name character varying (255)	email character varying (255)	address_street character varying (255)
1	Amit Sharma	amit.sharma@example.c...	123 MG Road
2	Rahul Verma	rahul.verma@example.com	789 Brigade Road

vii) Usage of to_char, extract

Show booking ID, user, and booking date formatted as 'DD-Mon-YYYY', also extracting year.

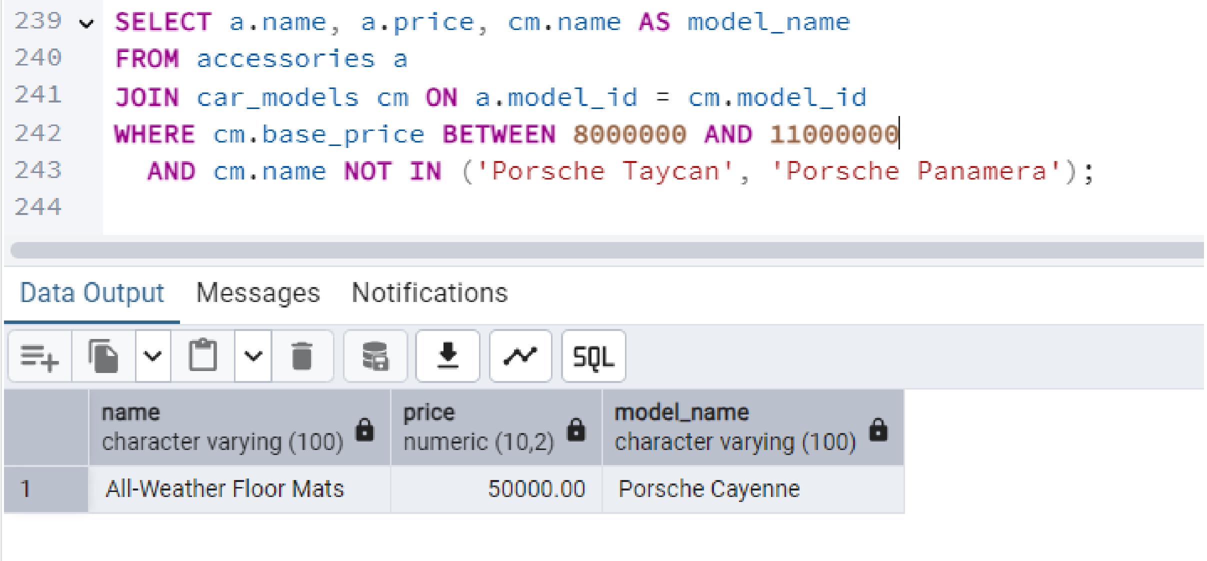


```
SELECT b.booking_id, u.name,
       TO_CHAR(b.booking_date, 'DD-Mon-YYYY') AS formatted_date,
       EXTRACT(YEAR FROM b.booking_date) AS booking_year
FROM bookings b
JOIN users u ON b.user_id = u.user_id;
```

	booking_id integer	name character varying (255)	formatted_date text	booking_year numeric
1	1	Amit Sharma	12-Jun-2025	2025
2	2	Priya Singh	12-Jun-2025	2025
3	3	Rahul Verma	12-Jun-2025	2025
4	4	Anita Desai	12-Jun-2025	2025
5	5	Vikram Rao	12-Jun-2025	2025

viii) Between, IN, Not between, Not in

List accessories for models whose base price is between 8M and 11M, but not for Taycan or Panamera.

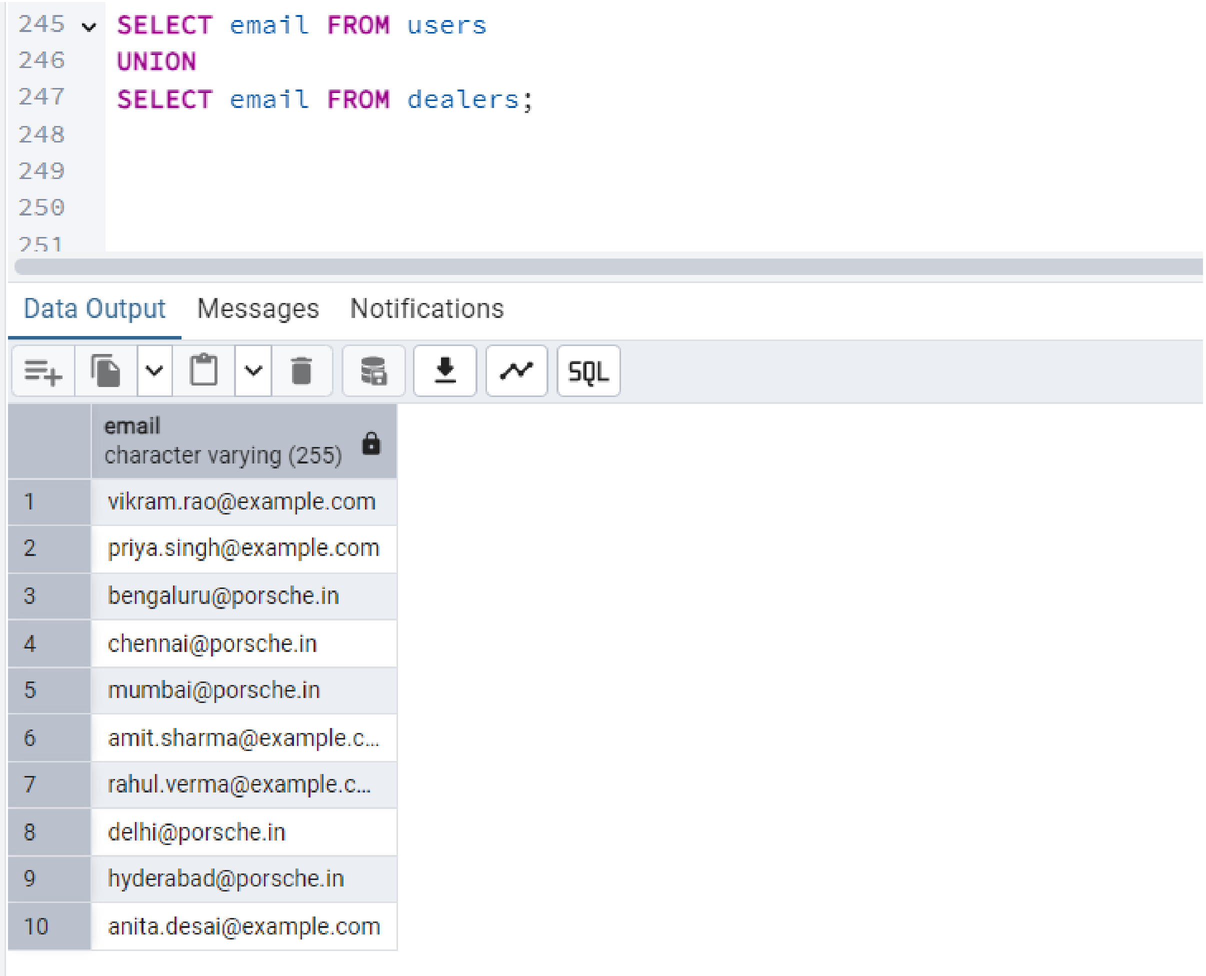


```
SELECT a.name, a.price, cm.name AS model_name
FROM accessories a
JOIN car_models cm ON a.model_id = cm.model_id
WHERE cm.base_price BETWEEN 8000000 AND 11000000
  AND cm.name NOT IN ('Porsche Taycan', 'Porsche Panamera');
```

	name character varying (100)	price numeric (10,2)	model_name character varying (100)
1	All-Weather Floor Mats	50000.00	Porsche Cayenne

ix) Set operations

List all distinct email addresses of users and dealers.

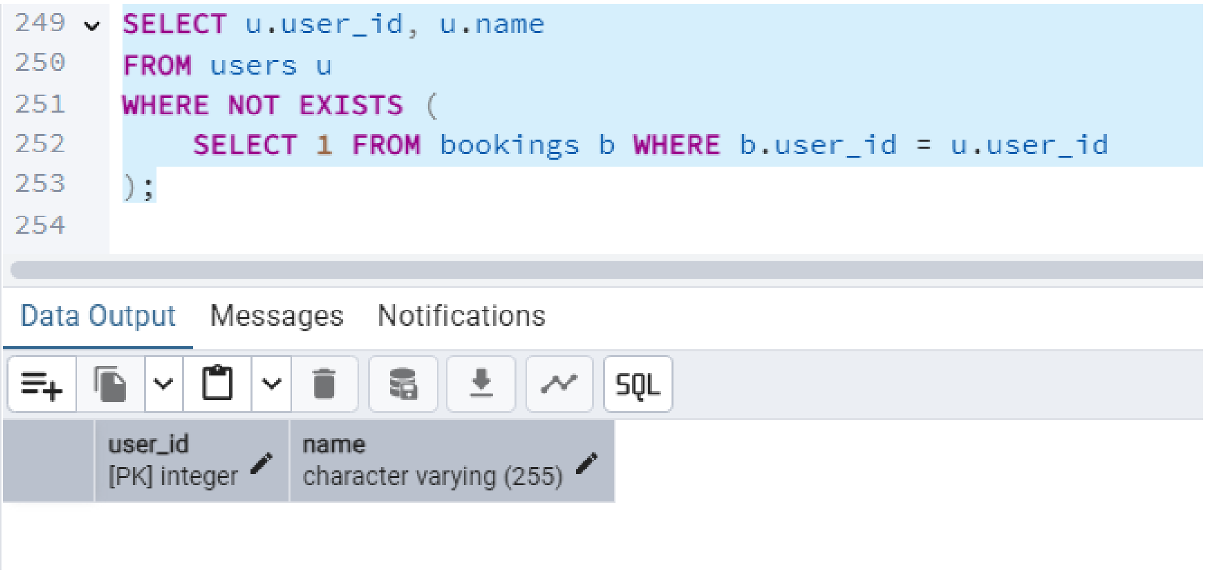


```sql
SELECT email FROM users
UNION
SELECT email FROM dealers;
```

	email character varying (255)
1	vikram.rao@example.com
2	priya.singh@example.com
3	bengaluru@porsche.in
4	chennai@porsche.in
5	mumbai@porsche.in
6	amit.sharma@example.c...
7	rahul.verma@example.c...
8	delhi@porsche.in
9	hyderabad@porsche.in
10	anita.desai@example.com

x) Subquery using EXISTS/NOT EXISTS, ANY, ALL

Find users who have **not** made any bookings.



```sql
SELECT u.user_id, u.name
FROM users u
WHERE NOT EXISTS (
    SELECT 1 FROM bookings b WHERE b.user_id = u.user_id
);
```

	user_id [PK] integer	name character varying (255)

Conclusion

The Porsche Elite Webspace project successfully demonstrates the integration of modern web technologies with a structured, relational database to deliver an interactive and immersive vehicle browsing and booking platform. By focusing on user experience, data integrity, and system scalability, the application allows both customers and administrators to seamlessly interact with vehicle data, configurations, and dealership information. Through well-defined modules, intuitive UI, and robust backend support, the system achieves its goal of replicating a premium automotive showroom experience online. This project not only showcases practical implementation of full-stack development but also reflects a solid understanding of database design, user interface design, and software engineering principles.