Car Inventory Management System: Object-Relational Database Implementation

CSE346 Advanced Database Systems - Spring 2025

May 8, 2025

Abstract

This technical report presents the design and implementation of an ORDBMS for car inventory management. The system leverages PostgreSQL's object-relational capabilities to model complex vehicle hierarchies while ensuring ACID compliance, implementing data fragmentation strategies, and incorporating security measures.

Contents

1 Relevance

This project directly applies the OODB concepts and advanced database features covered in CSE346 Advanced Database Systems.

2 Introduction

The Car Inventory Management System is designed to address the complex data management needs of automotive dealerships. Vehicle data exhibits inherent hierarchical relationships and specialized attributes across different vehicle types, making it an ideal candidate for object-relational database modeling. This implementation demonstrates how PostgreSQL's object-relational features can be leveraged to create a robust, scalable, and secure inventory management solution.

3 System Design

3.1 Domain Model

The system models vehicles using an inheritance hierarchy with specialized classes for different vehicle types:

- Base Vehicle class - Contains common attributes for all vehicles
- Car subclass - Extends Vehicle with car-specific attributes
- Specialized car subclasses:
  - Sedan - Passenger-focused cars with luxury classifications
  - SUV - Utility vehicles with cargo and seating capacities
  - Truck - Work-oriented vehicles with towing and payload specifications

3.2 Domain Constraints

Custom domains were created to enforce data integrity:

- `email_address` - Validates email format
- `phone_number` - Ensures phone number format
- `positive_decimal` - Ensures non-negative decimal values
- `positive_integer` - Ensures non-negative integer values
- `year_domain` - Validates year values within acceptable range

3.3 Custom Types

The following custom types enhance data organization:

- Enumerated types for categorical values:
  - `vehicle_status_type` - Tracks inventory status (in_stock, sold, maintenance, reserved)
  - `fuel_type` - Defines fuel system types (gasoline, diesel, electric, hybrid, plugin_hybrid)
  - `transmission_type` - Classifies transmission systems (automatic, manual, cvt, dct)
  - `payment_method_type` - Tracks sales payment methods (cash, finance, lease, credit)
  - `luxury_level_type` - Categorizes vehicle luxury tiers (standard, luxury, premium, ultra)
  - `cab_type` - Defines truck cab configurations (regular, extended, crew)
  - `user_role_type` - Assigns system access roles (admin, sales, inventory, readonly)
- Composite type for standardized address storage:
  - `address_type` - Contains structured address components

4 Database Implementation

4.1 Table Structure and Inheritance

4.1.1 Base Class

```sql
CREATE TABLE vehicles (
    vehicle_id VARCHAR(50) PRIMARY KEY,
    make VARCHAR(100) NOT NULL,
    model VARCHAR(100) NOT NULL,
    year year_domain,
    vin VARCHAR(17) UNIQUE NOT NULL,
    purchase_price positive_decimal NOT NULL,
    price positive_decimal,
    date_acquired DATE NOT NULL DEFAULT CURRENT_DATE,
    status vehicle_status_type NOT NULL DEFAULT 'in_stock',
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Listing 1: Base Vehicle Class

4.1.2 Subclass Structure

```sql
CREATE TABLE cars (
    car_id VARCHAR(50),
    body_type VARCHAR(50) NOT NULL,
    fuel_type fuel_type NOT NULL,
    transmission transmission_type NOT NULL,
    mileage positive_integer NOT NULL DEFAULT 0,
    engine_size DECIMAL(3, 1),
    PRIMARY KEY (vehicle_id)
) INHERITS (vehicles);
```

Listing 2: Car Subclass

```sql
CREATE TABLE sedans (
    sedan_id VARCHAR(50),
    luxury_level luxury_level_type,
    PRIMARY KEY (vehicle_id)
) INHERITS (cars);
```

Listing 3: Sedan Subclass

```sql
CREATE TABLE suvs (
    suv_id VARCHAR(50),
    seating_capacity positive_integer CHECK (seating_capacity > 0),
    cargo_capacity DECIMAL(6, 2),
    ground_clearance DECIMAL(4, 1),
    awd_4wd BOOLEAN DEFAULT FALSE,
    PRIMARY KEY (vehicle_id)
) INHERITS (cars);
```

Listing 4: SUV Subclass

```sql
CREATE TABLE trucks (
    truck_id VARCHAR(50),
    bed_length DECIMAL(5, 2),
    towing_capacity positive_integer,
    payload_capacity positive_integer,
    cab_type cab_type,
    PRIMARY KEY (vehicle_id)
) INHERITS (cars);
```

Listing 5: Truck Subclass

4.2 Supporting Tables

```sql
CREATE TABLE parts (
    part_id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    category VARCHAR(50) NOT NULL,
    part_number VARCHAR(50) UNIQUE NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    quantity_in_stock positive_integer NOT NULL DEFAULT 0,
    reorder_threshold positive_integer NOT NULL DEFAULT 5,
    reorder_quantity positive_integer NOT NULL DEFAULT 10,
    supplier_id VARCHAR(50),
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Listing 6: Parts Table

```
CREATE TABLE vehicle_parts (
    vehicle_id VARCHAR(50) REFERENCES vehicles(vehicle_id) ON DELETE
   CASCADE,
    part_id VARCHAR(50) REFERENCES parts(part_id) ON DELETE RESTRICT,
    quantity positive_integer NOT NULL DEFAULT 1,
    installed_date DATE DEFAULT CURRENT_DATE,
    PRIMARY KEY (vehicle_id, part_id)
);
```

Listing 7: Vehicle Parts Relationship

```
CREATE TABLE customers (
    customer_id VARCHAR(50) PRIMARY KEY,
    username VARCHAR(50) UNIQUE,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email email_address UNIQUE,
    phone phone_number,
    password_hash VARCHAR(255) NOT NULL,
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Listing 8: Customers Table

```
CREATE TABLE sales (
    sale_id VARCHAR(50) PRIMARY KEY,
    vehicle_id VARCHAR(50) REFERENCES vehicles(vehicle_id) ON DELETE
   RESTRICT,
    customer_id VARCHAR(50) REFERENCES customers(customer_id) ON DELETE
    RESTRICT,
    sale_date DATE NOT NULL DEFAULT CURRENT_DATE,
    sale_price DECIMAL(12, 2) NOT NULL,
    payment_method payment_method_type NOT NULL,
    finance_term positive_integer,
    notes TEXT,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Listing 9: Sales Table

5 Advanced Database Features

5.1 ACID Compliance

5.1.1 Atomicity

The system ensures that operations either complete fully or not at all, demonstrated in the vehicle status update transaction:

```sql
-- Atomicity example: Moving a vehicle from in_stock to sold status
BEGIN;

DO $$
    DECLARE
        vehicle_record RECORD;
    BEGIN
        -- Retrieve the vehicle record from vehicles_in_stock
        SELECT * INTO vehicle_record FROM vehicles_in_stock WHERE vehicle_id = 'your-vehicle-id';

        -- Insert the record into vehicles_sold with the updated status
        INSERT INTO vehicles_sold (vehicle_id, make, model, year, vin, purchase_price, price,
                                  date_acquired, status, created_at, updated_at)
        VALUES (vehicle_record.vehicle_id, vehicle_record.make, vehicle_record.model,
                vehicle_record.year, vehicle_record.vin, vehicle_record.purchase_price,
                vehicle_record.price, vehicle_record.date_acquired, 'sold',
                vehicle_record.created_at, CURRENT_TIMESTAMP);

        -- Delete the record from vehicles_in_stock
        DELETE FROM vehicles_in_stock WHERE vehicle_id = 'your-vehicle-id';
    END $$;

COMMIT;
```

Listing 10: Atomicity Example - Vehicle Status Change

5.1.2 Consistency

Consistency is enforced through domain constraints, unique constraints, and check constraints:

```sql
-- Insert with ROLLBACK on Error to maintain consistency
DO $$
    BEGIN
        -- Start of the transaction block
        BEGIN
            -- Insert a new vehicle into the vehicles table
            INSERT INTO vehicles (make, model, year, vin, purchase_price, price, status)
            VALUES ('Toyota', 'Camry', 2023, '1HGCM82633A004352', 25000.00, 28000.00, 'in_stock');

            -- Simulate an error by attempting to insert a duplicate VIN
            INSERT INTO vehicles (make, model, year, vin, purchase_price, price, status)
            VALUES ('Toyota', 'Camry', 2023, '1HGCM82633A004352', 25000.00, 28000.00, 'in_stock');

        EXCEPTION
            WHEN unique_violation THEN
```

```
                RAISE NOTICE 'Transaction rolled back due to unique constraint violation';
                RETURN;
            WHEN others THEN
                RAISE NOTICE 'Transaction rolled back due to an error: %', SQLERRM;
                RETURN;
        END;
    END $$;
```

Listing 11: Consistency Example - ROLLBACK on Error

5.1.3 Isolation

PostgreSQL's transaction isolation ensures that concurrent transactions do not interfere with each other. The system uses the default READ COMMITTED isolation level, preventing dirty reads while allowing non-repeatable reads and phantom reads.

5.1.4 Durability

Durability is assured through PostgreSQL's write-ahead logging (WAL) system. All committed transactions are recorded in WAL files before being applied to data files, ensuring persistence across system failures.

5.2 Data Fragmentation

5.2.1 Horizontal Fragmentation

Vehicles are horizontally fragmented based on their status, improving query performance by separating frequently accessed in-stock vehicles from sold and maintenance vehicles:

```
-- Horizontal Fragmentation
CREATE TABLE vehicles_in_stock (
    CHECK (status = 'in_stock')
) INHERITS (vehicles);

CREATE TABLE vehicles_sold (
    CHECK (status = 'sold')
) INHERITS (vehicles);

CREATE TABLE vehicles_maintenance (
    CHECK (status = 'maintenance')
) INHERITS (vehicles);

-- Rules to route inserts to the appropriate fragment
CREATE RULE vehicles_insert_in_stock AS
    ON INSERT TO vehicles
    WHERE (NEW.status = 'in_stock')
    DO INSTEAD
    INSERT INTO vehicles_in_stock VALUES (NEW.*);

CREATE RULE vehicles_insert_sold AS
    ON INSERT TO vehicles
    WHERE (NEW.status = 'sold')
    DO INSTEAD
    INSERT INTO vehicles_sold VALUES (NEW.*);
```

```sql

CREATE RULE vehicles_insert_maintenance AS
    ON INSERT TO vehicles
    WHERE (NEW.status = 'maintenance')
    DO INSTEAD
    INSERT INTO vehicles_maintenance VALUES (NEW.*);
```

Listing 12: Horizontal Fragmentation of Vehicles

5.2.2 Vertical Fragmentation

Service records are vertically fragmented to separate basic information, cost data, and technician details, allowing access control at a granular level:

```sql
-- Vertical Fragmentation of Service Records
CREATE TABLE service_basic_info (
    service_id varchar(50) PRIMARY KEY,
    vehicle_id varchar(50) REFERENCES vehicles(vehicle_id),
    service_date DATE NOT NULL,
    description TEXT NOT NULL,
    created_at TIMESTAMP NOT NULL,
    updated_at TIMESTAMP NOT NULL
);

CREATE TABLE service_cost_info (
    service_id varchar(50) PRIMARY KEY REFERENCES service_basic_info(
   service_id),
    cost DECIMAL(10, 2) NOT NULL
);

CREATE TABLE service_technician_info (
    service_id varchar(50) PRIMARY KEY REFERENCES service_basic_info(
   service_id),
    technician_name VARCHAR(100)
);
```

Listing 13: Vertical Fragmentation of Service Records

5.3 Security Implementation

5.3.1 Role-Based Access Control

The system implements role-based access control with predefined roles and permissions:

```sql
-- Create roles for security
CREATE ROLE car_admin WITH LOGIN PASSWORD 'admin_pass';
CREATE ROLE car_sales WITH LOGIN PASSWORD 'sales_pass';
CREATE ROLE car_inventory WITH LOGIN PASSWORD 'inventory_pass';
CREATE ROLE car_readonly WITH LOGIN PASSWORD 'readonly_pass';

-- Create users and assign roles
CREATE USER Omar WITH PASSWORD 'baherjr';
CREATE USER salesperson WITH PASSWORD 'sales';
CREATE USER inventory_manager WITH PASSWORD 'inv';
CREATE USER reporting_and_audit WITH PASSWORD 'audit';

GRANT car_admin TO Omar;
GRANT car_sales TO salesperson;
GRANT car_inventory TO inventory_manager;
GRANT car_readonly TO reporting_and_audit;
```

Listing 14: Role Creation

5.3.2 Row-Level Security

Row-level security policies restrict access based on user roles:

```sql
-- Enable Row Level Security on vehicles
ALTER TABLE vehicles ENABLE ROW LEVEL SECURITY;
ALTER TABLE cars ENABLE ROW LEVEL SECURITY;
ALTER TABLE sedans ENABLE ROW LEVEL SECURITY;
ALTER TABLE suvs ENABLE ROW LEVEL SECURITY;
ALTER TABLE trucks ENABLE ROW LEVEL SECURITY;

-- Admin can see and modify all
CREATE POLICY admin_vehicles_policy ON vehicles
    TO car_admin
    USING (TRUE)
    WITH CHECK (TRUE);

-- Sales can see all but only modify non-sold vehicles
CREATE POLICY sales_vehicles_view_policy ON vehicles
    TO car_sales
    USING (TRUE);

CREATE POLICY sales_vehicles_modify_policy ON vehicles
    TO car_sales
    USING (status != 'sold')
    WITH CHECK (status != 'sold');

-- Inventory staff can see all and modify inventory-related fields
CREATE POLICY inventory_vehicles_policy ON vehicles
    TO car_inventory
    USING (TRUE)
    WITH CHECK (TRUE);

-- Readonly role can only view data
CREATE POLICY readonly_vehicles_policy ON vehicles
    TO car_readonly
    USING (TRUE);
```

Listing 15: Row-Level Security Configuration

5.3.3 Permission Grants

Object-level permissions are assigned to roles:

```sql
-- Grant permissions
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO car_admin;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO car_admin;

-- Sales Role Permissions (Customer and Sales focused)
GRANT SELECT ON ALL TABLES IN SCHEMA public TO car_sales;
GRANT INSERT, UPDATE ON
    vehicles, cars, sedans, suvs, trucks,
    customers, sales
    TO car_sales;

-- Inventory Role Permissions (Vehicle and Parts focused)
GRANT SELECT ON ALL TABLES IN SCHEMA public TO car_inventory;
GRANT INSERT, UPDATE ON
    vehicles, cars, sedans, suvs, trucks,
    parts, vehicle_parts TO car_inventory;

-- Read-only Role Permissions
GRANT SELECT ON ALL TABLES IN SCHEMA public TO car_readonly;

-- Revoke critical operations from non-admin roles
REVOKE DELETE ON vehicles, cars, sedans, suvs, trucks, sales, customers
    FROM car_sales;
REVOKE INSERT, UPDATE, DELETE ON customers, sales FROM car_inventory;
```

Listing 16: Permission Grants

6 Performance Optimization

6.1 Indexing Strategy

Indexes are created on frequently queried columns to improve retrieval performance:

```sql
-- Create indexes for performance
CREATE INDEX idx_vehicles_status ON vehicles(status);
CREATE INDEX idx_cars_make_model ON cars(make, model);
CREATE INDEX idx_parts_category ON parts(category);
CREATE INDEX idx_sales_date ON sales(sale_date);
CREATE INDEX idx_sedans_sedan_id ON sedans(sedan_id);
CREATE INDEX idx_suvs_suv_id ON suvs(suv_id);
CREATE INDEX idx_trucks_truck_id ON trucks(truck_id);

-- Add unique constraints for subclass IDs
ALTER TABLE sedans ADD CONSTRAINT sedans_sedan_id_key UNIQUE (sedan_id)
    ;
ALTER TABLE suvs ADD CONSTRAINT suvs_suv_id_key UNIQUE (suv_id);
ALTER TABLE trucks ADD CONSTRAINT trucks_truck_id_key UNIQUE (truck_id)
    ;
```

Listing 17: Indexing Strategy

7 Advanced Queries

7.1 Query 1: Available SUVs with Minimum Seating Capacity

This query finds all available SUVs with at least 6 seats, ordered by price:

```sql
SELECT s.vehicle_id, s.make, s.model, s.year, s.price, s.
    seating_capacity,
       s.cargo_capacity, s.ground_clearance,
       CASE WHEN s.awd_4wd THEN 'Yes' ELSE 'No' END AS all_wheel_drive
FROM suvs s
WHERE s.status = 'in_stock'
```

```
  AND s.seating_capacity >= 6
ORDER BY s.price ASC;
```

Listing 18: Query 1

7.2 Query 2: Top Customers by Sales Amount

This query identifies the top 10 customers by total amount spent:

```
SELECT
    c.customer_id,
    c.first_name || ' ' || c.last_name AS customer_name,
    COUNT(s.sale_id) AS number_of_purchases,
    SUM(s.sale_price) AS total_spent
FROM customers c
JOIN sales s ON c.customer_id = s.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name
ORDER BY total_spent DESC
LIMIT 10;
```

Listing 19: Query 2

7.3 Query 3: Inventory Status Breakdown by Vehicle Type

This query provides a summary of vehicle inventory status across vehicle types:

```
SELECT
    'Sedan' AS vehicle_type,
    status,
    COUNT(*) AS count
FROM sedans
GROUP BY status
UNION ALL
SELECT
    'SUV' AS vehicle_type,
    status,
    COUNT(*) AS count
FROM suvs
GROUP BY status
UNION ALL
SELECT
    'Truck' AS vehicle_type,
    status,
    COUNT(*) AS count
FROM trucks
GROUP BY status
ORDER BY vehicle_type, status;
```

Listing 20: Query 3

7.4 Query 4: Vehicle Parts Analysis

This query identifies vehicles with the most replacement parts installed:

```
SELECT
    v.vehicle_id,
    v.make,
```

```sql
    v.model,
    v.year,
    COUNT(vp.part_id) AS total_parts,
    SUM(p.price * vp.quantity) AS total_parts_value
FROM vehicles v
JOIN vehicle_parts vp ON v.vehicle_id = vp.vehicle_id
JOIN parts p ON vp.part_id = p.part_id
GROUP BY v.vehicle_id, v.make, v.model, v.year
ORDER BY total_parts DESC, total_parts_value DESC
LIMIT 15;
```

Listing 21: Query 4

7.5 Query 5: Sales Performance Analysis

This query analyzes sales performance by vehicle type and payment method:

```sql
WITH vehicle_types AS (
    SELECT sale_id, 'Sedan' AS vehicle_type
    FROM sales s
    JOIN sedans sd ON s.vehicle_id = sd.vehicle_id
    UNION ALL
    SELECT sale_id, 'SUV' AS vehicle_type
    FROM sales s
    JOIN suvs sv ON s.vehicle_id = sv.vehicle_id
    UNION ALL
    SELECT sale_id, 'Truck' AS vehicle_type
    FROM sales s
    JOIN trucks t ON s.vehicle_id = t.vehicle_id
)
SELECT
    vt.vehicle_type,
    s.payment_method,
    COUNT(*) AS sales_count,
    AVG(s.sale_price)::NUMERIC(10,2) AS avg_sale_price,
    SUM(s.sale_price) AS total_revenue
FROM sales s
JOIN vehicle_types vt ON s.sale_id = vt.sale_id
GROUP BY vt.vehicle_type, s.payment_method
ORDER BY vt.vehicle_type, total_revenue DESC;
```

Listing 22: Query 5

8 Results and Analysis

8.1 OODB Advantages for Vehicle Management

The object-relational database implementation for the car inventory system demonstrated several key advantages:

- **Natural modeling of vehicle hierarchy**: The inheritance structure (Vehicle → Car → Sedan/SUV/Truck) closely mirrors the real-world classification of vehicles, making the database schema intuitive.

- **Type safety**: Custom domains and enumerated types enforce data integrity at the database level, reducing the need for application-level validation.
- **Polymorphic queries**: The ability to query the base vehicle table while retrieving subclass-specific attributes simplifies data access.
- **Data fragmentation benefits**: Horizontal fragmentation by vehicle status significantly improved query performance for common operations (listing available vehicles), while vertical fragmentation of service records enhanced data security by allowing fine-grained access control.

8.2 Security Model Effectiveness

The implemented security model provided robust protection through multiple layers:

- **Role-based access control**: Clearly defined roles (admin, sales, inventory, read-only) align database permissions with organizational responsibilities.
- **Row-level security**: Dynamic filtering based on user roles ensures users can only access and modify appropriate data records.
- **Permission restrictions**: Granular permission grants limit potential damage from compromised accounts.

8.3 ACID Compliance Analysis

The implementation successfully maintained ACID properties:

- **Atomicity**: Multi-step operations (like transferring vehicles between status categories) are completed as a unit or not at all.
- **Consistency**: Domain constraints, referential integrity, and check constraints ensure the database remains in a valid state.
- **Isolation**: PostgreSQL's transaction isolation prevents interference between concurrent operations.
- **Durability**: All committed transactions are persistently stored and recoverable after system failures.

9 WebApp Implementation

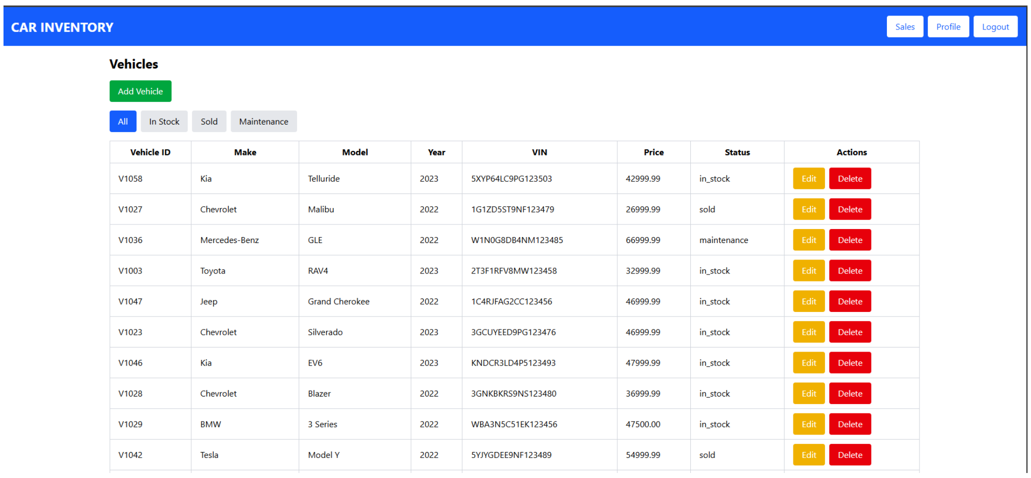
The Car Inventory Management System includes a web-based user interface that provides access to the database functionality through an intuitive dashboard. The WebApp implements role-based access with different views and capabilities based on user permissions.

9.1 User Interface Design

The WebApp follows a responsive design approach, ensuring accessibility across desktop and mobile devices. Key interface components include:

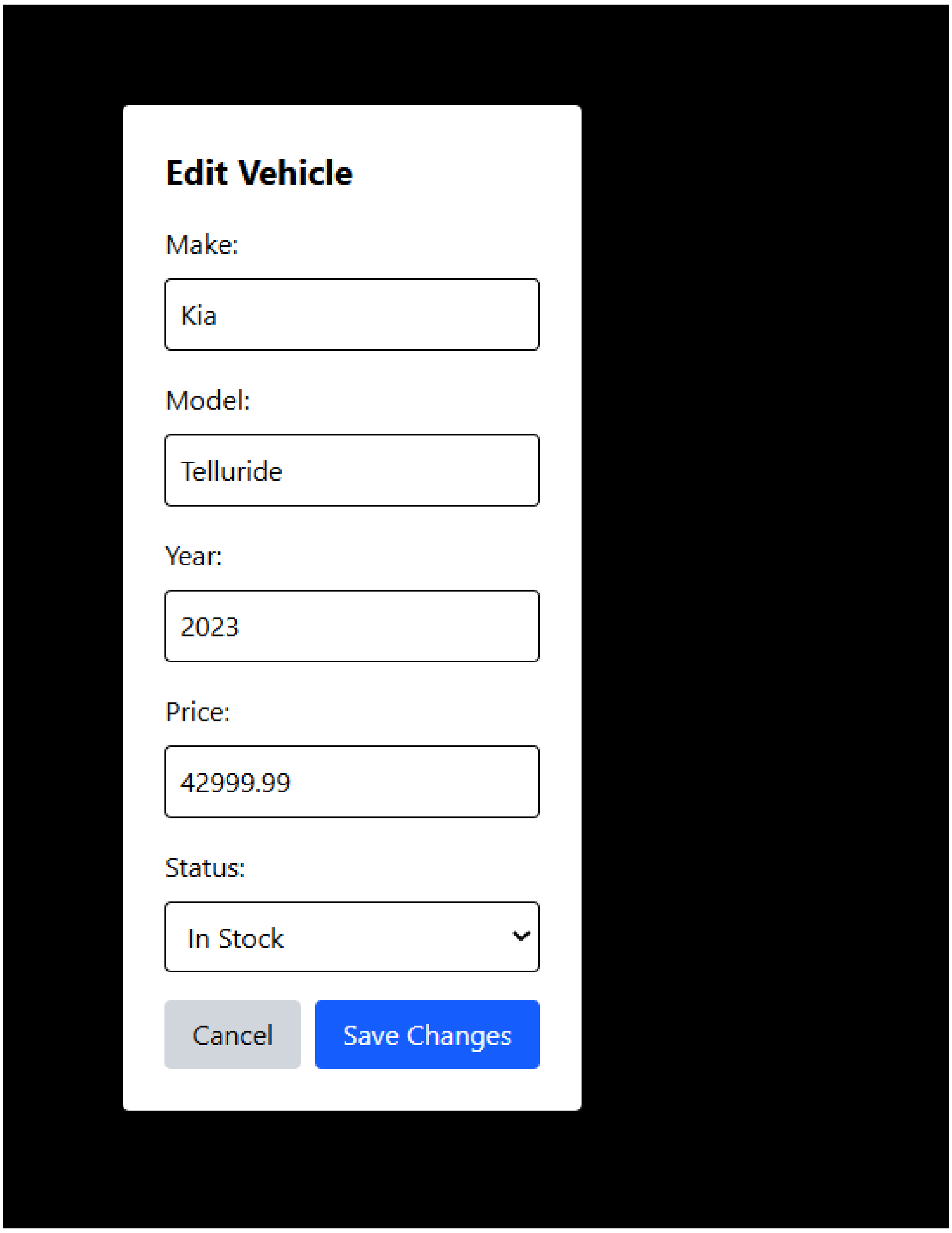
- Dashboard with real-time inventory metrics
- Vehicle browsing with advanced filtering options
- Detailed vehicle information pages
- Inventory management tools for authorized staff
- Sales tracking and reporting interface
- Administrative control panel

9.2 Key Screenshots

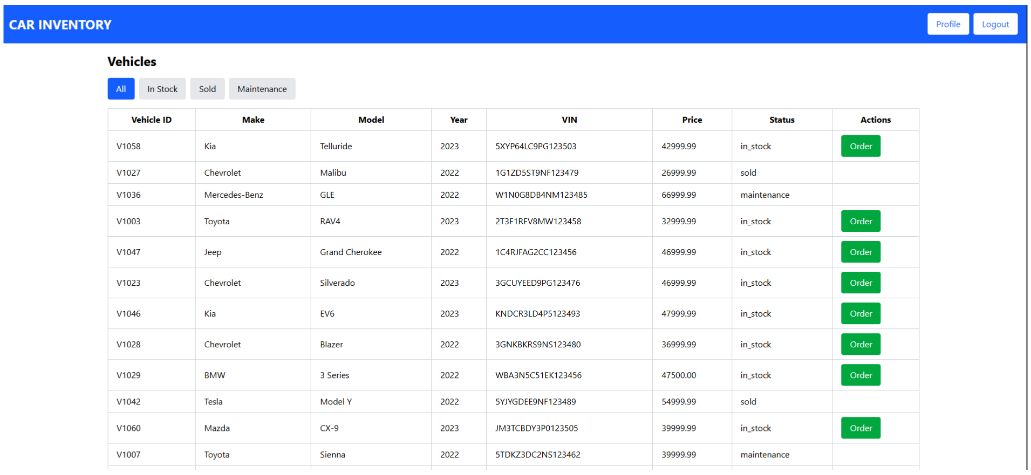


Vehicles

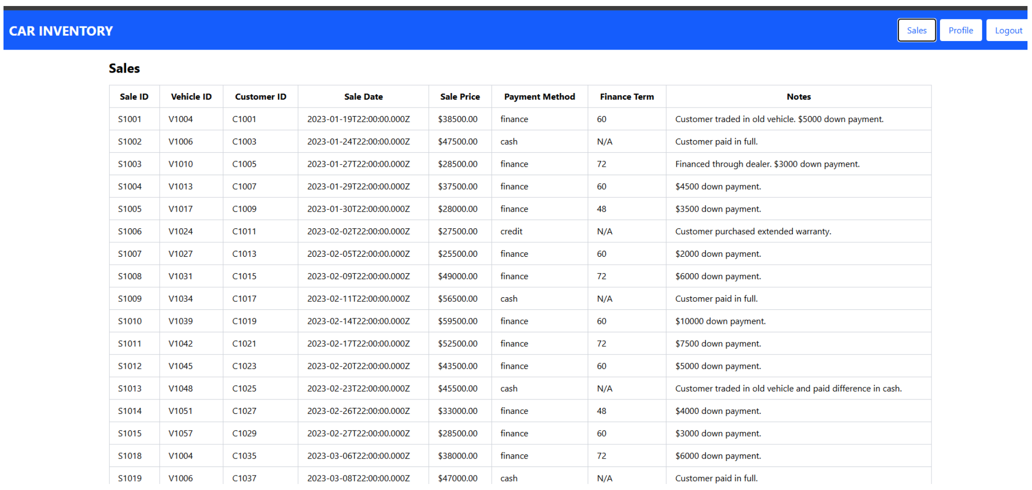
Vehicle ID	Make	Model	Year	VIN	Price	Status	Actions
V1058	Kia	Telluride	2023	5XYP64LC9PG123503	42999.99	in_stock	Edit Delete
V1027	Chevrolet	Malibu	2022	1G1ZD5ST9NF123479	26999.99	sold	Edit Delete
V1036	Mercedes-Benz	GLE	2022	W1N0G8DB4NM123485	66999.99	maintenance	Edit Delete
V1003	Toyota	RAV4	2023	2T3F1RFV8MW123458	32999.99	in_stock	Edit Delete
V1047	Jeep	Grand Cherokee	2022	1C4RJFAG2CC123456	46999.99	in_stock	Edit Delete
V1023	Chevrolet	Silverado	2023	3GCUYEED9PG123476	46999.99	in_stock	Edit Delete
V1046	Kia	EV6	2023	KNDCR3LD4P5123493	47999.99	in_stock	Edit Delete
V1028	Chevrolet	Blazer	2022	3GNKBKRS9NS123480	36999.99	in_stock	Edit Delete
V1029	BMW	3 Series	2022	WBA3N5C51EK123456	47500.00	in_stock	Edit Delete
V1042	Tesla	Model Y	2022	5YJYGDEE9NF123489	54999.99	sold	Edit Delete



Make:	Kia
Model:	Telluride
Year:	2023
Price:	42999.99
Status:	In Stock



Vehicle ID	Make	Model	Year	VIN	Price	Status	Actions
V1058	Kia	Telluride	2023	5XYP64LC9PG123503	42999.99	in_stock	Order
V1027	Chevrolet	Malibu	2022	1G1ZD5ST9NF123479	26999.99	sold	
V1036	Mercedes-Benz	GLE	2022	W1N0G8DB4NM123485	66999.99	maintenance	
V1003	Toyota	RAV4	2023	2T3F1RFV8MW123458	32999.99	in_stock	Order
V1047	Jeep	Grand Cherokee	2022	1C4RJFAG2CC123456	46999.99	in_stock	Order
V1023	Chevrolet	Silverado	2023	3GCUYEED9PG123476	46999.99	in_stock	Order
V1046	Kia	EV6	2023	KNDCR3LD4P5123493	47999.99	in_stock	Order
V1028	Chevrolet	Blazer	2022	3GNKBKRS9NS123480	36999.99	in_stock	Order
V1029	BMW	3 Series	2022	WBA3N5C51EK123456	47500.00	in_stock	Order
V1042	Tesla	Model Y	2022	5YJYGDEE9NF123489	54999.99	sold	
V1060	Mazda	CX-9	2023	JM3TCBDY3P0123505	39999.99	in_stock	Order
V1007	Toyota	Sienna	2022	5TDKZ3DC2NS123462	39999.99	maintenance	



Sales

Sale ID	Vehicle ID	Customer ID	Sale Date	Sale Price	Payment Method	Finance Term	Notes
S1001	V1004	C1001	2023-01-19T22:00:00.000Z	$38500.00	finance	60	Customer traded in old vehicle. $5000 down payment.
S1002	V1006	C1003	2023-01-24T22:00:00.000Z	$47500.00	cash	N/A	Customer paid in full.
S1003	V1010	C1005	2023-01-27T22:00:00.000Z	$28500.00	finance	72	Financed through dealer. $3000 down payment.
S1004	V1013	C1007	2023-01-29T22:00:00.000Z	$37500.00	finance	60	$4500 down payment.
S1005	V1017	C1009	2023-01-30T22:00:00.000Z	$28000.00	finance	48	$3500 down payment.
S1006	V1024	C1011	2023-02-02T22:00:00.000Z	$27500.00	credit	N/A	Customer purchased extended warranty.
S1007	V1027	C1013	2023-02-05T22:00:00.000Z	$25500.00	finance	60	$2000 down payment.
S1008	V1031	C1015	2023-02-09T22:00:00.000Z	$49000.00	finance	72	$6000 down payment.
S1009	V1034	C1017	2023-02-11T22:00:00.000Z	$56500.00	cash	N/A	Customer paid in full.
S1010	V1039	C1019	2023-02-14T22:00:00.000Z	$59500.00	finance	60	$10000 down payment.
S1011	V1042	C1021	2023-02-17T22:00:00.000Z	$52500.00	finance	72	$7500 down payment.
S1012	V1045	C1023	2023-02-20T22:00:00.000Z	$43500.00	finance	60	$5000 down payment.
S1013	V1048	C1025	2023-02-23T22:00:00.000Z	$45500.00	cash	N/A	Customer traded in old vehicle and paid difference in cash.
S1014	V1051	C1027	2023-02-26T22:00:00.000Z	$33000.00	finance	48	$4000 down payment.
S1015	V1057	C1029	2023-02-27T22:00:00.000Z	$28500.00	finance	60	$3000 down payment.
S1018	V1004	C1035	2023-03-06T22:00:00.000Z	$38000.00	finance	72	$6000 down payment.
S1019	V1006	C1037	2023-03-08T22:00:00.000Z	$47000.00	cash	N/A	Customer paid in full.

10 Conclusion

The object-relational database implementation for car inventory management demonstrates the power of PostgreSQL's advanced features in modeling complex domain relationships. The inheritance-based vehicle hierarchy, combined with custom types and domains, creates a robust foundation for the application. Performance optimizations through data fragmentation and indexing ensure the system can handle real-world workloads efficiently, while the comprehensive security model protects sensitive data through role-based access control and row-level security.

This implementation successfully fulfills the project requirements by showcasing ACID compliance, data fragmentation techniques, comprehensive security implementation, and complex query capabilities. The system provides a flexible and extensible foundation that could be further enhanced with additional object-relational features such as table functions and procedural extensions.

11 References

1. PostgreSQL Documentation. "Inheritance." https://www.postgresql.org/docs/current/ddl-inherit.html

2. PostgreSQL Documentation. "Row Security Policies." https://www.postgresql.org/docs/current/rowsecurity.html

3. PostgreSQL Documentation. "Transaction Isolation." https://www.postgresql.org/docs/current/iso.html

4. PostgreSQL Documentation. "Domain Types." https://www.postgresql.org/docs/current/domain

5. PostgreSQL Documentation. "User-defined Types." https://www.postgresql.org/docs/current/xt