Car Sales Database Management System

Ajay Vijayakumar, Lokesh Velmurugan Sujatha, Shivaramakrishnan Rajendran
vijayak3@buffalo.edu, lvelmuru@buffalao.edu, rajendr6@buffalo.edu

***Abstract*—As every industry is moving towards digital era, the auto sales sector is still stuck in the past with old-fashioned methods. There are several difficulties because of this dependency on manual procedures. Dealerships must drop their outdated ledgers and adopt modern technological solutions as the world moves toward a connected, data-driven future.**

I. Introduction

Data Disasters: Imagine youre trying to ask a question or ask for a request online, and the form disappears post you submit it. This is the sad reality of data disasters caused by outdated technology. Missing files and misplaced information are common, resulting in lost sales opportunities and reducing overall efficiency. Important information gets lost, affecting everything from consumer interactions to financial reporting. **Process PitFalls**: that are old fashioned. Compensating for technological deficiencies and dealing with data inaccuracies wastes valuable time and resources. Consider a sales professional attempting to address a customer's question about car availability, and what if we need to get the data from many spreadsheets or handwritten notes. This not only slows down the sales process, but also results in missed opportunities and consumer unhappiness.

Blindfolded Business: Without real-time insights and up to date information, dealerships work without vision. Inventory management becomes a guessing game, with the danger of stock shortages or excesses. Consider a dealership buying more sedans if SUVs are the current fashion and most selling, resulting in unsold stock and reduced business and profits. A lack of visibility into customer trends obstructs strategic decision-making, leaving dealerships unable to respond to market shifts and leverage on the new opportunities.

II. MILESTONE 1

III. Database Design

A. Why do we need a database and how does this solve the problem?

- Structured Data Organization: Imagine having a separate shelf for toys, one for clothes, one for stationaries and so on. This is similar to what a database does. Databases allow structured data storage with separate tables for vehicles, customers, sales, and pricing history. This prevents redundancy and ensures efficient querying.
- Data Relationships and Integrity: In database we can establish relationships between tables which helps us in maintaining the data consistency. For instance, connecting sale to the car and the customer who bought it.
- Granular Permissions and Security: Databases offer fine-grained access control, enhancing security. Depending upon the users authorization and accesses they could just see or edit the data. Real-Time Insights: Tracking sales, inventory, and customer trends in real time empowers managers to make well informed decisions.
- Performance and Efficiency: Databases load faster, consume less storage space, and focus on relational data, laying the foundation for automation.

B. Target User

Consider a streamlined car dealership where information flows smoothly, increasing sales and customer satisfaction. This becomes a reality with the suggested database system, which serves as a central hub for important inventory, sales, and customer history data. Let's look at the important user groups that will unlock its potential:

- **Salespeople**: With the proposed system, employees dont have to rush to find information about the cars or potential buyers. This system provides them with rapid access to car specifications, customer preferences, and previous interactions, helping them to personalize pitches, close deals more quickly, and establish long-term connections.
- **Inventory Managers**: With real-time information on what vehicles are available now, how often a particular car is being sold, and the demand from consumers, they can optimize inventory levels, avoid sales being lost, and make educated purchasing decisions.
- **Marketing Teams**: The proposed database system stores information about the cars which are popular in different regions and customers preferred cars, which helps in targeting the right customers with the accurate offers. They may create targeted campaigns, give personalized promotions, and enhance marketing ROI by leveraging extensive customer data and sales analytics.
- **Management**: From the Management perspective it is easier for them to fix the price for the cars based on the demand it has. They get a complete overview of what the dealership looks like, which helps in making informed decisions about whether to add more employees for making more inventories and to provide promotional offers to the targeted customers.

C. ER Diagram (Before Normalization)

Fig. 1. ER Diagram

D. TABLE

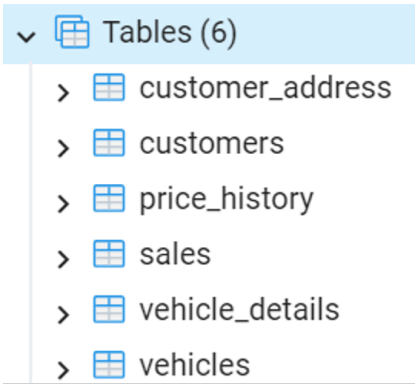


Fig. 2. Tables

Field	Description
Customer_id	Unique identifier for each customers
Name	Customer's name
email	Customer's mail ID
phone_number	Customer's phone number

TABLE I
CUSTOMER TABLE SCHEMA

Field	Description
Customer_id	Unique identifier for each customers
address	address of the customer
city	City where the customer is from
state	state where the customer resides
zipcode	The postal code of the customer's address

TABLE II
CUSTOMER_ADDRESS TABLE SCHEMA

Field	Description
vehicle_id	Unique identifier for each customers
type	vehicle type
make	make of the vehicle
model	model of the vehicle
year	year when vehicle is manufactured
vin	Vehicle Identification Number
availability	availability of the vehicle
customer_id	customer_id of the person owns the vehicle

TABLE III
VEHICLE TABLE SCHEMA

Field	Description
sale_id	saleid of the transaction
vehicle_id	ID of the vehicle purchased
sale_date	Date when the transaction happened
sale_price	price of the vehicle

TABLE IV
SALES TABLE SCHEMA

Field	Description
vehicle_id	ID of the vehicle
price_recorded_date	Date when the price change happened
price	price it changed to

TABLE V
PRICE_HISTORY TABLE SCHEMA

vehicle_id	ID of the vehicle
miles_used	miles that the vehicle has travelled
color	color of the vehicle
engine_type	engine_type of the vehicle
transmission_type	transmission_type of the vehicle
fuel_type	fuel_type of the vehicle
number_of_seats	number_of_seats of the vehicle

TABLE VI
VEHICLE_DETAILS TABLE SCHEMA

IV. MILESTONE 2

In milestone 2, we focused on improving the database's performance and storage efficiency. We achieved this by refining the initial schema through a process called decomposition. This involved breaking down tables to achieve a specific design standard called Boyce_Codd Normal Form (BCNF).

Specific improvements: The **customer_address** table was split into two separate tables: **customer_address** and **zipcodes** The **vehicles** table was divided into **vehicles** and

vehicle_models. The **vehicle_details** table was decomposed into **vehicle_details** and **engines**.

V. NORMALIZATION

Database normalization ensures efficient data organization. It resembles organizing belongings for easy retrieval, eliminating duplicates. By dividing data into smaller, logically connected parts, information remains organized, preventing inconsistencies and facilitating updates. This process creates a well-structured filing system for the database. We decompose the table to be in BCNF form.

VI. CONDITIONS FOR BCNF

For a table to meet BCNF, it must pass two tests.

- First, any attribute that determines the value of another must uniquely identify each row. We call these determinants "candidate keys."
- Secondly, if an attribute determines another attribute in a non-trivial manner, it must itself be unique and capable of identifying a row independently. So, when we're looking at each table, we're basically checking if these two rules are satisfied. If they are, then the table is BCNFcompliant. If not, we might need to make some adjustments to ensure everything fits the criteria.

So, when we're looking at each table, we're basically checking if these two rules are satisfied. If they are, then the table is BCNFcompliant. If not, we might need to make some adjustments to ensure everything fits the criteria. To ensure the database adheres to BoyceCodd Normal Form (BCNF), we will begin by identifying non-trivial functional dependencies within each of the six tables. These dependencies will guide us in decomposing tables that violate BCNF principles.
Now Let's look at the current relations and the functional dependencies of those.

A. Table Customers

```
customer_id -> name,email,phone_number
email -> name
phone_number -> name
```

The customers table contains information about customers, with a primary key customer_id. Each customer has attributes such as name, email, and phone_number, all of which are fully functionally dependent on the primary key. There are no partial dependencies, so the table is in BCNF.

B. Table Customer Address

```
customer_id -> address,city,state,zipcode
address,city,state -> zipcode (Not in BCNF)
zipcode -> state (Not in BCNF)
```

The customer_address table contains the information such as customer ID, address, city, state, and zipcode. However, it's not in Boyce-Codd Normal Form (BCNF) since it does not satisfy the two conditions that we mentioned above. To adhere to BCNF, we decompose the table into two relations. We can figure out that the state just by knowing the zipcode, which means some data is redundant. Hence, we split the table into two.

- In the first table, we just have zipcode and state. Each zipcode corresponds to one state, so there's no redundancy there. This relation is named zipcodes.
- In the second table, we keep the rest of the information: zipcode, address, city, and customer ID. Since each customer ID is unique and every combination of address and city is unique (two different customers can't have the same address in the same city), this arrangement avoids redundancy too. This FD confirms to BCNF and there are no more FD that is derived from the parent relation. This relation is named customer_address. For this relation, we will have the zipcode as the foreign key.

C. Table Vehicles

```
vehicle_id -> type,make,model,year,vin,
    availability,customer_id
vin -> type,make,model,year
model -> make(Not in BCNF)
```

The vehicles table, containing attributes like vehicle ID, type, make, model, year, VIN, availability, and customer ID, exhibits functional dependencies which is not in BoyceCodd Normal Form (BCNF). To adhere to BCNF, decomposition is necessary.One relation is formed with attributes

```
{vehicle_id, type, model, year, vin,
availability, customer_id}
```

where vehicle ID uniquely determines these attributes and the model is a foreign key. Another relation includes model, make is created to resolve the dependency between model and make. This decomposition ensures each relation adheres to BCNF, eliminating redundancy and maintaining data integrity within the vehicles dataset.

D. Table Vehicle Details

```
vehicle_id -> miles_used, color ,
    engine_type,transmission_type,
    fuel_type,number_of_seats
engine_type -> fuel_type
```

E. Table Sales

```
sale_id -> vehicle_id,customer_id,
    sale_date,sale_price
vehicle_id -> customer_id,sale_date,
    sale_price
```

The table vehicle_details serves as a repository for essential information pertaining to vehicles, encompassing attributes such as vehicle ID, miles used, color, engine type, transmission type, fuel type, and number of seats. As we delved into the dataset, we discerned a significant concern regarding the relationship between engine type and fuel type. Considering this observation, we recognized the necessity of reorganizing the dataset to address this issue. Our aim was to enhance the structure and coherence of the data, ensuring that it aligns with established standards for data organization and normalization. To achieve this objective, we do decomposition, aimed at restructuring the ”vehicle_details” table into more streamlined and logically coherent components. We split the table into two relations: The first one has the attributes (engine_type, fuel_type) and the second one with all the other attributes. These relations are in BCNF, and no further decomposition is required. The first relation is named engines, and second relation is named vehicle_details in which the engine_type is a foreign key.

F. Table Price_History

`vehicle_id,price_recorded_date -> price`

The ”price_history” table keeps track of how prices for vehicles change over time. Each entry in the table includes the vehicle ID, the date when the price was recorded, and the price itself. This setup follows a rule called Boyce-Codd Normal Form (BCNF), which means it’s organized in a way that makes sense. Each combination of vehicle ID and recording date uniquely determines the price at that time, without any unnecessary connections or complications. So, the ”price_history” table is already structured in a way that fits the BCNF rules.

VII. Database Design - ER Diagram (After Normalization)

Fig. 3. ER Diagaram after normalization

VIII. Tables (After Normalization)

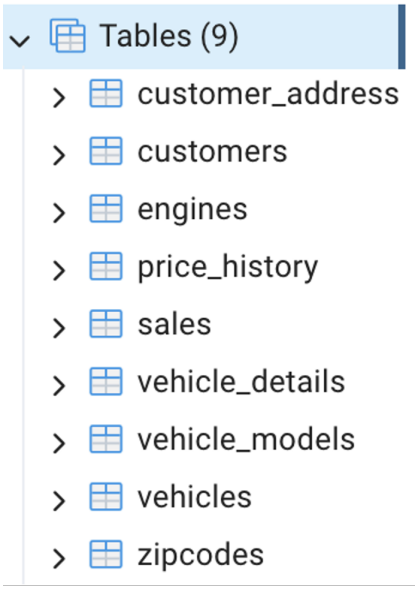


Fig. 4. TABLES After Normalization

Field	Description
Customer_id	Unique identifier for each customers
Name	Customer’s name
email	Customer’s mail ID
phone_number	Customer’s phone number

TABLE VII
CUSTOMER TABLE SCHEMA

Field	Description
Customer_id	Unique identifier for each customers
address	address of the customer
city	City where the customer is from
zipcode	The postal code of the customer’s address

TABLE VIII
CUSTOMER_ADDRESS TABLE SCHEMA

Field	Description
zipcode	The postal code of the customer’s address
state	state where the customer resides

TABLE IX
ZIPCODES TABLE SCHEMA

Field	Description
vehicle_id	Unique identifier for each customers
type	vehicle type
model	model of the vehicle
year	year when vehicle is manufactured
vin	Vehicle Identification Number
availability	availability of the vehicle
customer_id	customer_id of the person owns the vehicle

TABLE X
VEHICLE TABLE SCHEMA

Field	Description
model	model of the vehicle
make	make of the vehicle

TABLE XI
MODEL TABLE SCHEMA

Field	Description
saleid	saleid of the transaction
vehicle_id	ID of the vehicle purchased
sale_date	Date when the transaction happened
sale_price	price of the vehicle

TABLE XII
SALES TABLE SCHEMA

Field	Description
vehicle_id	ID of the vehicle
price_recorded_date	Date when the price change happened
price	price it changed to

TABLE XIII
PRICE_HISTORY TABLE SCHEMA

vehicle_id	ID of the vehicle
miles_used	miles that the vehicle has travelled
color	color of the vehicle
engine_type	engine_type of the vehicle
transmission_type	transmission_type of the vehicle
number_of_seats	number_of_seats of the vehicle

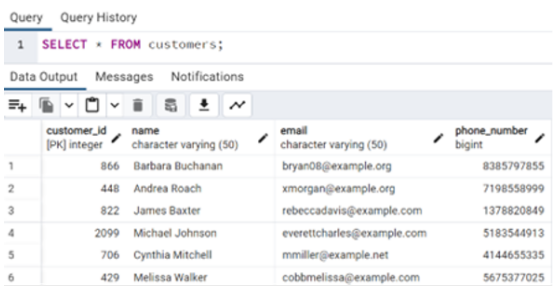
TABLE XIV
VEHICLE_DETAILS TABLE SCHEMA

Field	Description
engine_type	engine_type of the vehicle
fuel_type	fuel_type of the vehicle

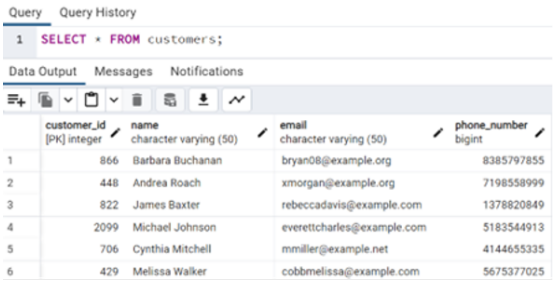
TABLE XV
PRICE_HISTORY TABLE SCHEMA

IX. DATABASE LOOKUP

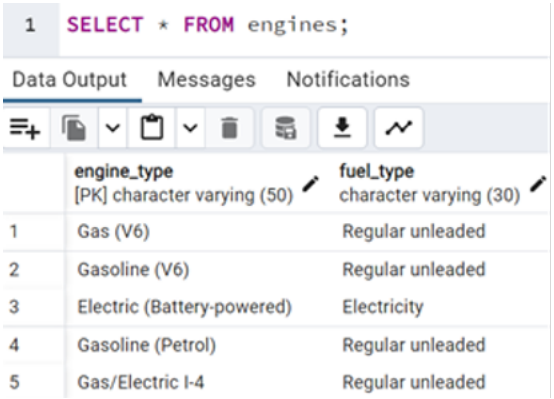
A. Customer Table:



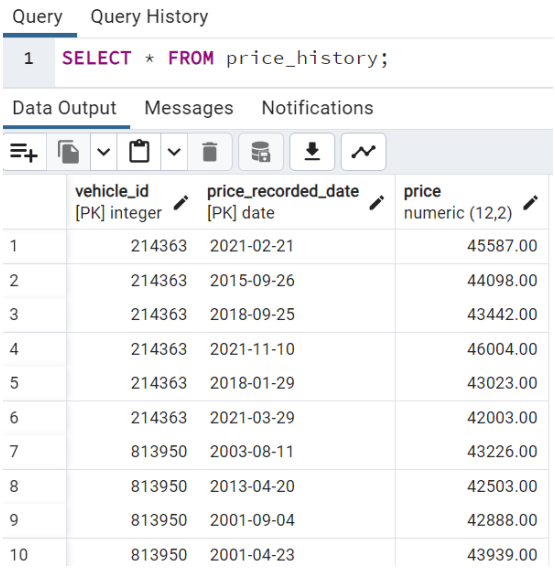
B. Customer Address Table:



C. Engines Table:



D. Price History Table:

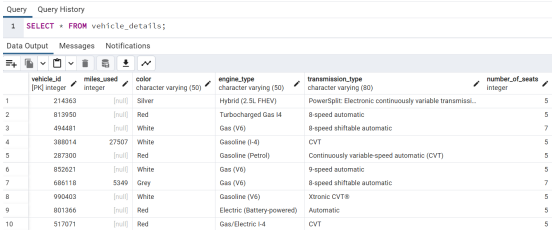


E. Sales Table:

```
SELECT * FROM sales;
```

	sale_id [PK] integer	vehicle_id integer	customer_id integer	sale_date date	sale_price numeric (12,2)
1	956807	214363	866	2023-08-06	23414.00
2	289381	813950	448	2013-08-22	30958.00
3	784217	494481	822	2019-09-25	38387.00
4	633004	388014	2099	2014-11-01	24826.00
5	932420	287300	706	2015-09-29	25223.00
6	377444	852621	429	2009-09-09	41590.00
7	317663	686118	2297	2023-08-28	37528.00
8	76327	990403	3638	2023-06-25	25861.00
9	809545	801366	2350	2023-05-14	56349.00
10	907456	517071	1156	2023-06-27	34589.00

F. Vehicle Details Table:

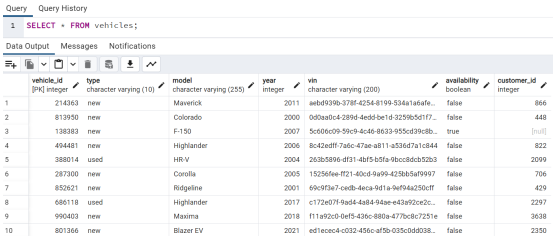


G. Vehicle Models Table:

```
SELECT * FROM vehicle_models;
```

	model [PK] character varying (255)	make character varying (255)
1	RAV4 Hybrid	Toyota
2	Bronco Sport	Ford
3	Edge	Ford
4	Equinox	Chevrolet
5	Mustang Mach-E	Ford
6	Blazer EV	Chevrolet
7	Highlander	Toyota
8	Tundra	Toyota
9	F-150	Ford
10	Rouge	Nissan

H. Vehicles Table:



I. Zipcodes Table:

```
SELECT * FROM zipcodes;
```

	zipcode [PK] character	state character varying (30)
1	00601	Puerto Rico
2	00624	Puerto Rico
3	00662	Puerto Rico
4	00676	Puerto Rico
5	00685	Puerto Rico
6	00690	Puerto Rico

X. QUERY EXECUTION

A. Find the distribution of the vehicle sales among different car models.

```
--#1 The distribution of vehicle sales among different car models

SELECT model,
       count(vehicle_id) AS vehicles_sold
FROM vehicles
WHERE availability = FALSE
GROUP BY model
ORDER BY vehicles_sold DESC;
```

	model character varying (255)	vehicles_sold bigint
1	Corolla	328
2	Maverick	324
3	Passport	303
4	Highlander	284
5	Trax	275
6	RAV4 Hybrid	256
7	F-150	239
8	Silverado	239
9	CR-V Hybrid	227
10	Tundra	222

B. Find all the customers who purchased more than two vehicles.

```sql
--#2 Customers who purchased more than two vehicles

WITH vehicles_per_cust AS
  (SELECT customer_id,
          count(vehicle_id) AS vehicles_count
   FROM sales
   GROUP BY customer_id
   HAVING count(vehicle_id) > 2)
SELECT name,
       email,
       vehicles_count
FROM vehicles_per_cust
JOIN customers cust ON cust.customer_id = vehicles_per_cust.customer_id;
```

Data Output | Messages | Notifications

	name character varying (50)	email character varying (50)	vehicles_count bigint
1	Theresa Woods	wgordon@example.com	3
2	Alex Phillips	vjenkins@example.org	3
3	Steven Munoz	duncanjudith@example.org	3
4	Shannon Frost	davilakristen@example.net	3
5	Elizabeth Johnson	kellyortiz@example.net	3
6	Nicole Cohen	ygonzalez@example.com	3
7	Jennifer Peterson	wvaughan@example.com	3
8	Gregory Moore	katelyn35@example.com	3
9	Miguel Henry	mgraham@example.org	3
10	Tiffany Walker	cmason@example.net	3

C. Find the average number of days between consecutive vehicle purchases for customers who have bought more than one vehicle.

```sql
--#3 Average number of days between consecutive vehicle purchases for
-- customers who have bought more than one vehicle?

WITH vehicles_per_cust AS
  (SELECT customer_id,
          count(vehicle_id) AS vehicles_count
   FROM sales
   GROUP BY customer_id
   HAVING count(vehicle_id) > 1),
     days_bwn_purchase AS
  (SELECT vehicles_per_cust.customer_id,
          sale_date,
          lag(sale_date) OVER (PARTITION BY vehicles_per_cust.customer_id
                               ORDER BY sale_date) AS previous_purchase_date
   FROM vehicles_per_cust
   JOIN sales ON vehicles_per_cust.customer_id = sales.customer_id)
SELECT round(avg(sale_date - previous_purchase_date)) AS avg_days_between_purchase
FROM days_bwn_purchase
WHERE previous_purchase_date IS NOT NULL;
```

Data Output | Messages | Notifications

	avg_days_between_purchase numeric
1	1860

D. Find the top 5 cars with the highest sales volume.

```sql
--#4 Top 5 cars with the highest sales volume
WITH sales_volume AS
  (SELECT vehicle_models.model,
          vehicle_models.make,
          COUNT(*) AS sales_volume
   FROM vehicles
   JOIN vehicle_models ON vehicles.model = vehicle_models.model
   JOIN sales ON vehicles.vehicle_id = sales.vehicle_id
   GROUP BY vehicle_models.model,
            vehicle_models.make
   ORDER BY sales_volume DESC)
SELECT model,
       make
FROM sales_volume
LIMIT 5;
```

Data Output | Messages | Notifications

	model [PK] character varying (255)	make character varying (255)
1	Corolla	Toyota
2	Maverick	Ford
3	Passport	Honda
4	Highlander	Toyota
5	Trax	Chevrolet

E. Find the models with highest sales in each state.

```sql
--#5 Model with the highest sales in each state
WITH sales_volume_by_state AS
  (SELECT model,
          state,
          COUNT(model) AS model_count,
          RANK() OVER (PARTITION BY state
                       ORDER BY COUNT(model) DESC) AS sales_rank
   FROM sales
   JOIN customer_address ON sales.customer_id = customer_address.customer_id
   JOIN zipcodes ON customer_address.zipcode = zipcodes.zipcode
   JOIN customers ON customer_address.customer_id = customers.customer_id
   JOIN vehicles ON customers.customer_id = vehicles.customer_id
   WHERE availability = FALSE
   GROUP BY model,
            state)
SELECT state,
       model
FROM sales_volume_by_state
WHERE sales_rank = 1;
```

Data Output | Messages | Notifications

	state character varying (30)	model character varying (255)
1	Alabama	Trax
2	Alaska	Highlander
3	Alaska	Trax
4	Arizona	Maxima
5	Arkansas	Equinox
6	California	Silverado
7	Colorado	Maxima
8	Connecticut	Maverick
9	Delaware	Sienna
10	Delaware	Equinox

INSERTION

F. Inserting Customer who hasnt made any vehicle purchases.

```sql
--#6 Add a customer who hasn't made any vehicle purchases
INSERT INTO customers(customer_id, name, email, phone_number)
VALUES (4000,'Elijah Evans','elijahfevans@example.net',9177523920),
       (4001,'James Case','jamesacase@example.com',5024097921);
```

Data Output | Messages | Notifications

```
INSERT 0 2

Query returned successfully in 56 msec.
```

UPDATION

G. Updating the Customers address.

```sql
--#7 Update customer address
UPDATE customer_address
SET address = '155 Pitts Lodge'
WHERE customer_id = 2582;
```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 37 msec.
```

DELETION

H. Deleting the incorrect price from price history.

Query Query History

```
1  --#8 Delete incorrect price from price history
2  DELETE
3  FROM price_history
4  WHERE vehicle_id = 7511
5    AND price_recorded_date='2009-04-25';
```

Data Output Messages Notifications

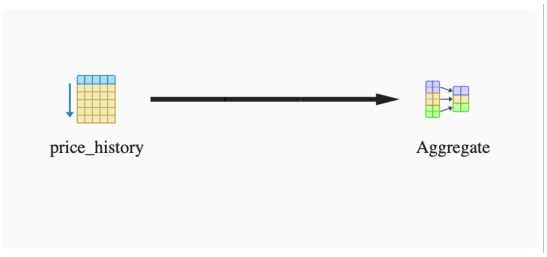
```
DELETE 1

Query returned successfully in 52 msec.
```

XI. INDEXING

While creating the database and running the SQL queries we found that a few of the queries were running for a long time. Upon investigating this we found that the queries were doing a full table scan before joining them. This can be rectified by creating and using indexes on tables.
Indexes enable faster access to data in a large table. Indexes enhance performance, especially in large data sets, when columns involved in querying large data sets are indexed. For example, if the primary key or filtering/ordering columns are often used to retrieve data from the database, the DBMS can quickly find the rows relevant to the query, without having to read every row of the table. Queries with filtering, sorting, and joining data based on the indexed columns can have significant performance improvement. The DBMS will be able to find the relevant rows quickly using the index rather than full-table scans for querying, which would increase the time for query execution.

We have created five indexes namely: An index named **vehicles_availability_idx** is used to enhance the retrieval of the vehicle availability status thereby, when needed, it is possible to check whether a vehicle is available or not in a matter of seconds. To support faster querying of vehicle models, because of the frequent insertion into WHERE clauses, **vehicles_model_idx** is designed. An index named **vehicles_type_idx** is designed to retrieve a user preference for either new or pre-owned vehicles quickly. An index **price_history_price_idx** is created for querying the price. An index named **price_history_price_idx** is established to optimize querying of price ranges within WHERE conditions, recognizing its frequent usage. e range in where condition, this is one of the most used conditions.

A. Without-Indexing (Price between range)



Query Query History

```
1  --#9 Price between range
2  SELECT min(price) AS lowest,
3         max(price) AS highest
4  FROM price_history
5  WHERE price BETWEEN 41650 AND 41750;
```

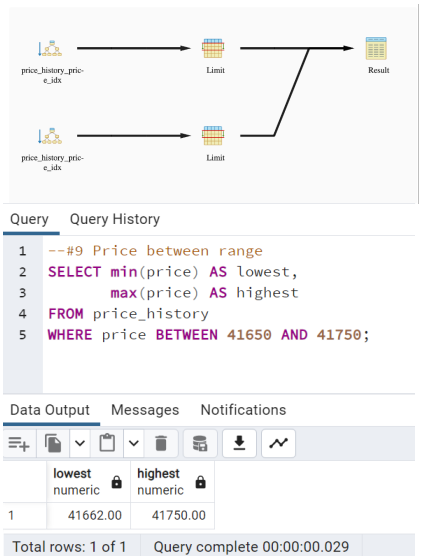
Data Output Messages Explain Notifications

	lowest numeric	highest numeric
1	41662.00	41750.00

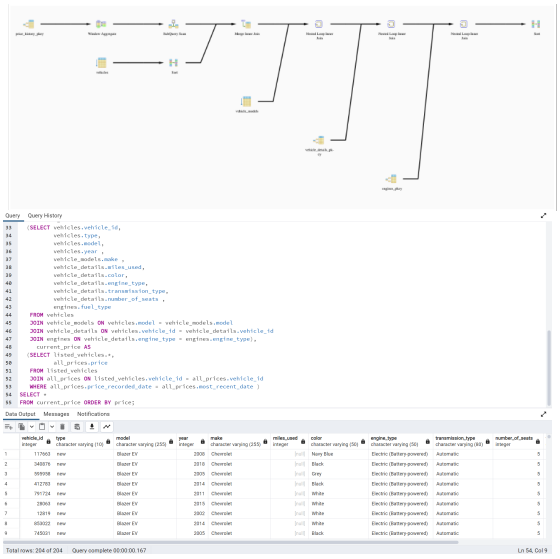
Total rows: 1 of 1 Query complete 00:00:00.059

We have used B+ indexing for finding the price between ranges. B+ indexing facilitates efficient retrieval of data within a given price range. The prices are sorted within a balanced tree so that the prices of a particular category are organized within a balanced tree. Each node within the tree points to child nodes or to data records, allowing quick navigation through subsets of data that meet a condition given. For example, a query requesting a range of prices is efficiently done through the B+ tree since it identifies the leaf nodes that contain price records in the specified range. It does this through its sorted order and balanced height properties in identifying the proper data that must be accessed to ensure that the query execution time is minimal, yet the performance is enhanced for the price range conditions.

B. With-Indexing (Price between range):



C. Without-Indexing:

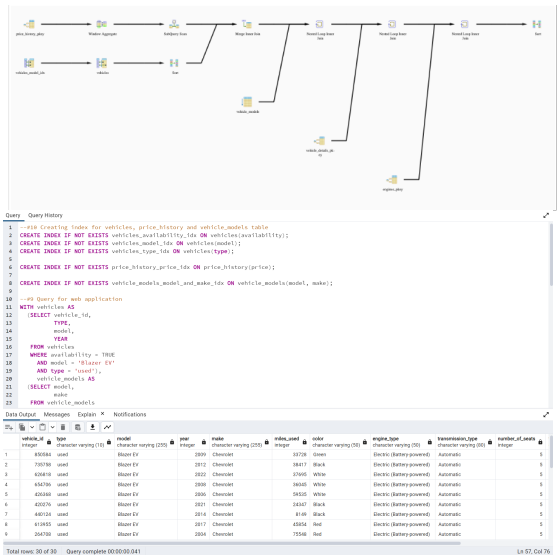


This index is very important in optimizing model-related queries. This index will allow the database system to directly move to the specific model being queried without having to scan the entire vehicles table. Therefore, the index makes it possible for the system to directly fetch the rows that are associated with the model being queried without performing a costly full-table scan.

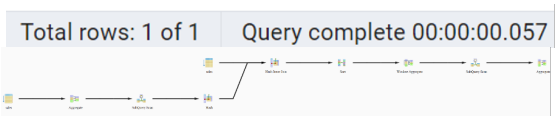
Observations indicate that indexing significantly improves performance, the performance increased around 4 times as compared to the one without indexing as evidenced by notably reduced retrieval times compared to operations conducted without indexing.

Similarly for the query where we found the average number of days between consecutive vehicle purchases for customers who have bought more than one vehicle, we have indexed the sales table to improve the query performance.

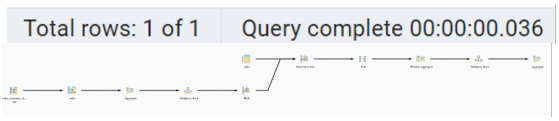
D. With Indexing:



E. Without-Indexing:



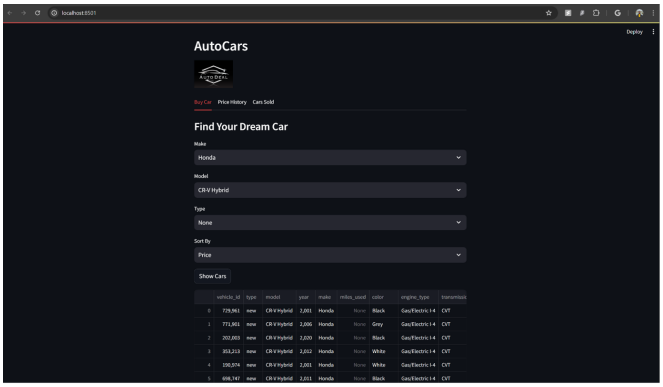
F. With Indexing:



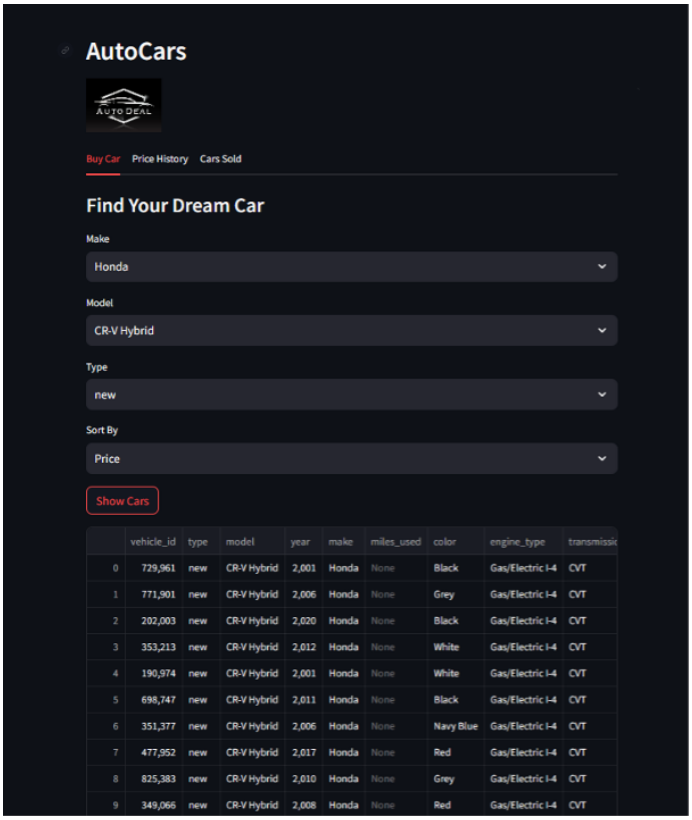
XII. WEBPAGE

The website is hosted in streamlit community servers, and the database is hosted in AWS RDS instances. The website is accessible through the following link:
https://cse-560-dmql-proj-rangers.streamlit.app/

This page contains the page for buying cars, price history of the car model, and the cars that are sold. In the page buy car it contains the inputs that is got from the user such as Make, Model, Type, Sort By (Price).



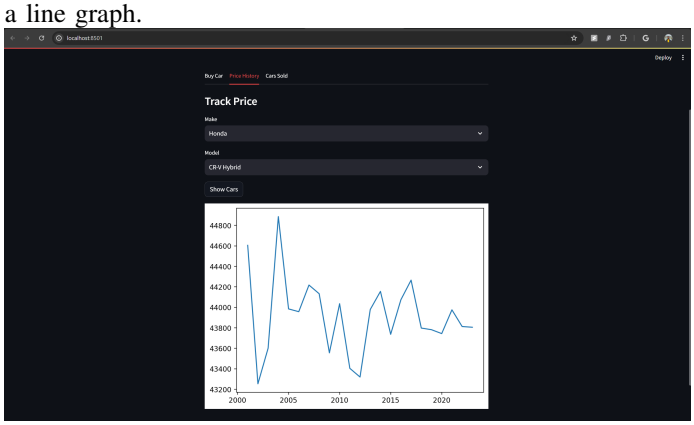
If a customer is looking to purchase a new Honda CR-V Hybrid and want to ensure youre getting a competitive price, the Autocars website offers a streamlined process to assist you. By selecting Honda as the make and CR-V Hybrid as the model, and specifying the type as new, you can sort the available vehicles by price. This feature allows you to view a list of new Honda CR-V Hybrids in ascending order of price, making it easier to find the best deal within your budget. Each listing provides comprehensive details such as the vehicle ID, model year, color, engine type, and transmission, ensuring you have all the information needed to make an informed decision.



	vehicle_id	type	model	year	make	miles_used	color	engine_type	transmissic
0	729,961	new	CR-V Hybrid	2,001	Honda	None	Black	Gas/Electric I-4	CVT
1	771,901	new	CR-V Hybrid	2,006	Honda	None	Grey	Gas/Electric I-4	CVT
2	202,003	new	CR-V Hybrid	2,020	Honda	None	Black	Gas/Electric I-4	CVT
3	353,213	new	CR-V Hybrid	2,012	Honda	None	White	Gas/Electric I-4	CVT
4	190,974	new	CR-V Hybrid	2,001	Honda	None	White	Gas/Electric I-4	CVT
5	698,747	new	CR-V Hybrid	2,011	Honda	None	Black	Gas/Electric I-4	CVT
6	351,377	new	CR-V Hybrid	2,006	Honda	None	Navy Blue	Gas/Electric I-4	CVT
7	477,952	new	CR-V Hybrid	2,017	Honda	None	Red	Gas/Electric I-4	CVT
8	825,383	new	CR-V Hybrid	2,010	Honda	None	Grey	Gas/Electric I-4	CVT
9	349,066	new	CR-V Hybrid	2,008	Honda	None	Red	Gas/Electric I-4	CVT

In the second page of the website, it shows the graph of the price history for a make and model of the car selected by the user. For instance in the above graph it shows price history of the Nissan Altima spanning several years. When users select ”Nissan” as the make and ”Altima” as the model within the ”Price History” tab, the chart indicates the way in which the price fluctuates from year to year. Notice the distinct cyclical trend: the price increases and decreases on an alternating basis. This is a visual insight that allows users to understand market value fluctuations of the Nissan Altima.

The price history page gives us the information of the previous price of the make, model and the outputs is given as a line graph.



The third page consist of the map that shows the cars that are sold in a region. By observing the concentration of blue dots, users can infer which car models are popular in different parts of the region. Areas with a high density of sales likely correspond to popular car choices among local buyers. Users can use this map to identify trends and understand which car models are in demand. If certain areas consistently show higher sales density for specific car brands or models, those vehicles are likely popular in that region.