Member_1: **Somin Gandhi** ID: **202001138**

Member_2: **Parth Patel** ID: **202001147**

Group No: **03** Section: **09**

Date: Nov 17, 2022 Team_id: **9**

Case Study: <u>Price History</u>

This database system helps to keep track of prices throughout a specific period of time, so that users can decide whether it's the right time to buy or not. This can be applied to various e-commerce sites like Amazon, Flipkart and Myntra. Users can search with given product links and it will output with price records. Admin can monitor and delete products. By searching specific product links, the system should also suggest a similar product list with platforms, for quick access. It should extract the name from the obtained product from the given link and search it on other platforms to compare prices. Users can login into the system and set alerts for particular products, if price goes below given criteria. Users should be able to see the history of products that were searched by him/her in the past.

Index

Section 1: Final version of SRS

1.1 Introduction

1.1.1 Purpose

→ Nowadays, most e-commerce companies frequently modify the prices of their items, making it difficult for buyers to find the most excellent bargain within their price range. To make it easier for customers to acquire the most excellent bargain, we may implement a price history tracker system that helps consumers determine if it is the right moment to buy or not by keeping track of prices over a predetermined period of time.

→ Along with this function, it aids the user in determining when and where to make purchases. Also, it keeps track of the numerous e-commerce sites that clients often like to use based on their prior purchases/searches.

→ Additionally, it provides clients with options for comparable goods that adhere to their filtering preferences, such as choices based on brands, models, and product specifications.

→ Hence, Technical tools like price trackers make it easier for manufacturers, distributors, and (online) retailers to monitor what their competitors and dealers are charging. As it provides information for pricing decisions, making the process of tracking prices simpler and less cumbersome.

1.1.2 Intended Audience and Reading Suggestions

→ Price History tracker system not only intended to ease the task of **customers** but also useful for other audiences like **manufacturers** to increase/decrease the supply of products based on past pricing scenarios, **retailers** to maintain the stock of those products whose prices are falling/rising, **stocks-analyst** to study the trend of e-commerce companies, to ease the product pricing study of **marketing staff,** etc.

→ One can learn more about the price history tracker system by going through various third-party tools that gather pricing information from different stores/e-commerce companies like https://pricehistoryapp.com, Naaptol, price panda, Keepa by amazon, etc. One can also read newspaper/site articles based on price history like link1, link2, link3, etc.

→ The above sources are also useful for developers, project managers, testers and document writer who works/is interested in the above case study.

1.1.3 Product Scope

→ Improves the visibility of pricing trends of various products sold on e-commerce websites.

→ A deal-making app can be created using this concept if customized to include a pricing history tracker and a product review tracker. Generally, every deal depends on the cost of a good and its performance in terms of ratings or results.

→ The idea of a price history tracker can also be used in domains like those whose price changes regularly, e.g., stocks, fuels, construction materials, etc.

→ This idea can also be extended to the availability of seats in the reservation of Flights, railways, buses, etc. As its availability also keeps on changing daily.

1.1.4 Description

- **Requirements:**
  → Our system keeps track of the prices of all the products that are registered on various e-commerce websites. For this, we need to store the prices of all the registered products for one year. Also, to keep track of customer search/purchase history, we will be creating a database of user detail that are registered on the site. As our price history tracker searches for the lowest price for a given product, we also store data of various discount offers like card offers, coupon codes, gift vouchers, etc.

- **The relations that we are planning to implement for this system:**

  1. **Platforms**(platform_id, platform_name)
     → Used to register the e-commerce websites name and id whose price history needs to be tracked.
     E.g., Amazon, Flipkart, Myntra, etc.

  2. **Departmen**t(pro_dept_id, dept_name)
     → Used to categorize similar products.
     E.g., Footwear, electronics, furniture, etc.

  3. **Product**(product_id, product_name, pro_dept_id, platform_id)
     → Used to register products on website which is uniquely identified by Its product_id, product_name, pro_dept_id and platform_id.
     E.g., iPhone, Apple, Smartphone, Amazon.

  4. **Offer**(Sr_id, company_name, Type_name, dis_price)
     → Used to register different types of offers and its discount rate.

E.g., 1, HDFC, Credit card, 15%.

5. **Offer_On**(Sr_id, pro_dept_id, platform_id)

→ Used to keep track of e-commerce websites that allow particular offers.
E.g. 1, electronics, Amazon.

6. **Yearly_price**(date,product_id, product_name, pro_dept_id, platform_id, price)

→ Used to keep track of prices of all registered products for duration of One year.
E.g., 22/09/22, iPhone, Apple, Smartphone, Amazon, 1,00,000.

→ Above relation will be used to plot graph/charts of price variations of user-given product for different platforms.

→ Above relation will also be used to compare prices of users given products on different platforms.

→ Above relation on mapping with offer relation would be used to provide the lowest price of the product that gives the best deal to user.

7. **User_detail**(user_id, user_name, user_pass)

→ Used to register users who visit the price history tracker site.
E.g. 138, SG18, ******.

8. **User_hist**(user_id, product_id, product_name, platform_id, date, is_purchased)

→ Used to keep track of user search and purchase history of products.
E.g., 138, iPhone, Apple, Amazon, 22/09/22, yes.

- **The functions that we are planning to implement for this system**:

1. **Add_platform**
2. **Add_product_department**
3. **Add_product**
4. **Add_new_Offer**
5. **Register_new_user**
6. **Add_user_history**
7. **Drop_platform**
8. **Drop_product_department**
9. **Drop_product**
10. **Drop_Offer**

11. **Remove_user**
12. **Drop_user_history**
13. **Update_Offer**
14. **Update_Offer_On**
15. **Update/Add_current_price**
16. **Remove_previous_data**
17. **Update_product**
18. **Update_password**
19. **Set_drop_alert**
20. **Notify_drop_alert**
21. **Modify_authority**

- **Real-World Work Flow of Price History Tracker:**
  - An existing customer, who has already registered, may log in using credentials to the system and can start using the services provided by the price history tracker.
  - If the customer is new, then he/she will be able to register and will be assigned a user_ID.
  - As the user add filters on his/her choice, he/she will be shown a graph/chart of the price variation of a given product on various site over a specific period, which could also be zoomed in/out.
  - Along with charts, users will also be able to see a list of current prices of a given product on various sites and best deals along with applicable offers.
  - The user can set the desired price for a given product so that the app will notify/alert the user when the price touches or goes below the desired price.
  - Also, the user will be fed various options of similar products that he/she has searched/purchased.

1.2 Fact-Finding Phase

1.2.1 Background Reading/s

1. **Amazon Price History Tracker, also known as Keepa link and Multi-platform Price History Tracker, Price History link.**
   From these two sites, we referred to the functionalities and services that we need to provide and the type of data that we need to maintain for the working of price tracker sites. In short, we got an idea about the basic structure of the database that we need to implement.

2. **The Best Amazon Price History Tracker, an article posted by Elizabeth Harper on the Techlicious website.**
   From this article, we got the idea that price history trackers tend to collect a lot of data as they show the history of price trends for a longer duration of time, like six months or one year. This gave us the points to be considered for the smooth working of the system as well as keeping the customers happy by focusing on their needs.

3. **Database System Concepts, by Abraham Silberschatz, Henry F. Korth, S. Sudarshan**
   From this article, we understood the DBMS concepts like E-R Model, Database designing, etc.

- **Combined Requirements gathered from the readings:**
  - A well-functioning Database management system is required to maintain and update all the information about the price of every product available on various e-commerce sites.
  - A user interface is required so that the customers can easily access the data required without having to know much about the actual implementation of the system.
  - System structure and functions should be designed in such a way that it ensures the fast performance of the system as price tracker sites deals with a huge amount of data for showing graphical statistics of price trends.
  - The interface should be clean, and filters should work efficiently without showing any redundant data, i.e., it must be user-specific.
  - The functioning of price drop alerts should be implemented appropriately to notify customers on time.

1.2.2 Interview/s

1. **Interview Plan**

System: All in One Price History Tracker

Project Reference: SF/SJ/2022/09

Interviewee: Vedant Patel (Role Play)

Designation: Chief Marketing Officer (CMO) at Flipkart

Interviewer:

1. Somin Gandhi **Designation:** Database Developer
2. Parth Patel **Designation:** Business Development Executive

Date: 28/09/2022 **Time:** 12:00
Duration: 45 minutes **Place:** Skype

Purpose of Interview: Preliminary meeting to identify key factors and requirements necessary for building an efficient Price History Tracker.

Agenda:

- Introduction and idea of the current pricing process
- Factors that affect the price of a product over some time
- An idea about the product management within the firm
- How firms encourage people to shop online

Documents to be brought to the interview:

- Rough plan of building
- Any documents relating to current Price History Trackers and the needs of the customers

- **Results of the interview:**
  - Other than product category, price and offers, pricing process also depends on the location of the customers, so for this, we can also add filters that show product automatically based on the customer’s location.
  - Other than factors, demand for the product, quantity, and brand, the price of the product is also affected by competition and availability of products

on different sites, so this problem will be handled by the comparison filter which we will be using.

- Other than everyday sales and offers, various platforms encourage people to shop online by giving customers special offers based on their purchase history, special subscriptions, gift vouchers/coupons, and fast or free deliveries. So all requirements can be fulfilled by keeping data of timely offers, subscription codes, and purchase history of the customers.

2. Interview Plan

System: All in One Price History Tracker

Project Reference: SF/SJ/2022/09

Interviewee: Gaurang Parmar (Role Play) **Designation:** Retailer

Interviewer:

1. Somin Gandhi **Designation:** Database Developer
2. Parth Patel **Designation:** Business Development Executive

Date: 28/09/2022 **Time:** 15:30
Duration: 45 minutes **Place:** Skype

Purpose of Interview: To understand and study stock keeping of products by local stores based on pricing trends of products. Also, their requirements that ease their analysis of product pricing/market scenario.

Agenda:

- Introduction and their strategy to maintain stock of goods
- Their requirements/expectations with Price History Tracker
- Things or features that are dissatisfactory or need improvement in the current Price History Tracker

Documents to be brought to the interview:

- Rough plan of building
- Any documents relating to current Price History Trackers and feedback regarding the current system

- **Results of the interview:**
  - To get a regular idea about increasing and decreasing trend of the price of user selected product, we will be maintaining it by showing a streak of days along with percentage increase or decrease in price.
  - Also, we will try to alert customers about the special offers that are given by the various platforms like removal of old stocks, discounts in replacement of old products, festival offers, etc.

3. Interview Plan

System: All in One Price History Tracker

Project Reference: SF/SJ/2022/09

Interviewee: Het Patel (Role Play) **Designation:** Customer

Interviewer:

1. Somin Gandhi **Designation:** Database Developer
2. Parth Patel **Designation:** Business Development Executive

Date: 28/09/2022 **Time:** 16:30
Duration: 45 minutes **Place:** Skype

Purpose of Interview: To get an idea about how an ideal customer thinks of price scenarios before purchasing products. Also, how to create a user-friendly price history tracker to make the shopping of customers easier/beneficial.

Agenda:

- Introduction and their strategy to buy products
- Things or features that were more likable to the customer while shopping online
- Their requirements/expectations with Price History Tracker

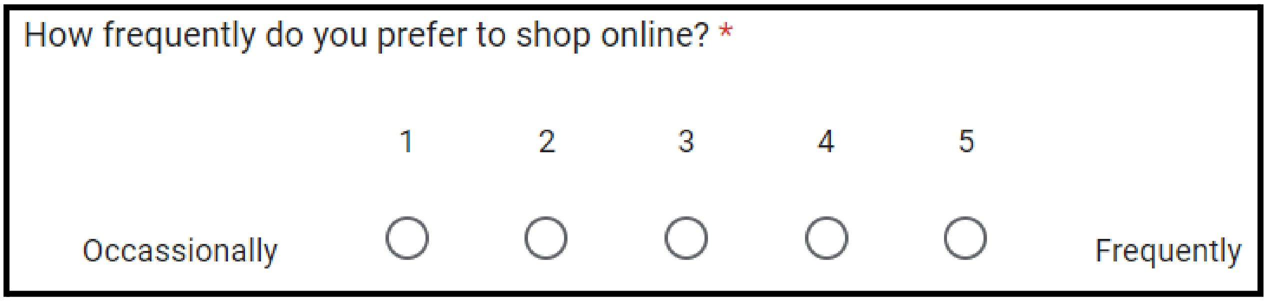
Documents to be brought to the interview: Rough plan of building

- **Results of the interview:**
  - Other than offers and low prices customers also looks for reliability/services of platform and product. So we can implement it by showing the rating of products based on their review on platforms.

  - Whereas for the reliability of platforms, we can show the products in a sorted manner of their platform reliability.

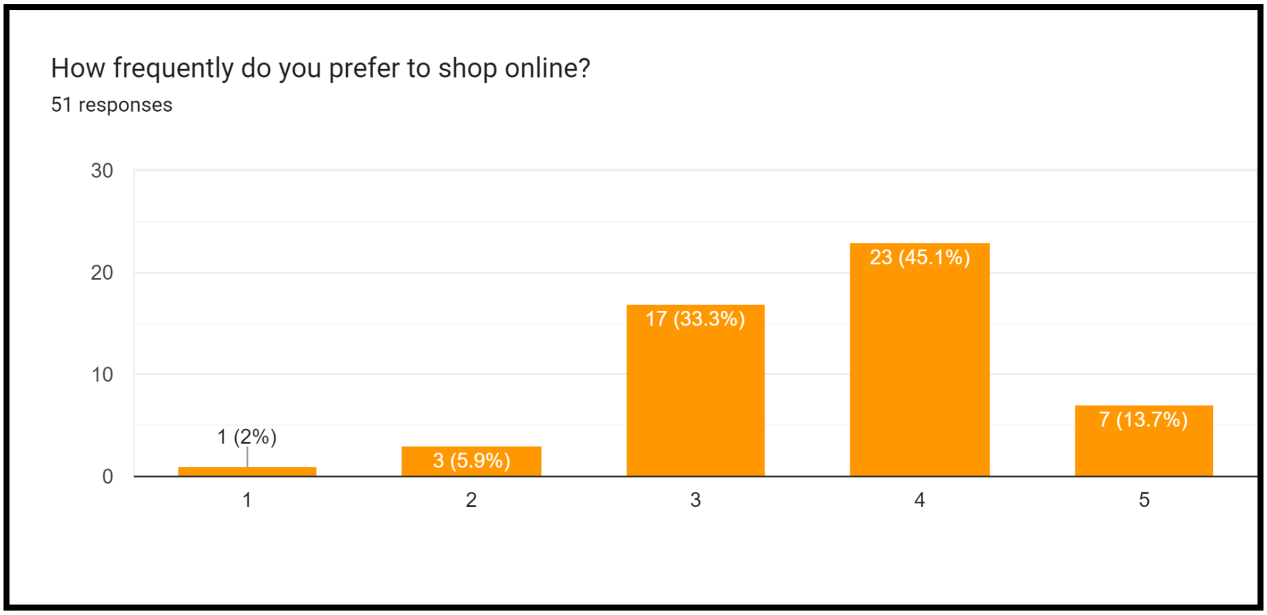
- **Combined Requirements gathered from the Interviews:**
  - We need to take sufficient details of the customers to implement the offers based on their purchased history, offers based on the location of the customers, and based on his/her interest.
  - Other than price charts & lists, we also need to maintain data on increasing/decreasing price streak data to provide a better idea of inflation.
  - Other than price add/drop alerts, we also need to maintain alerts regarding the timely offers given by the platforms.
  - We need to summarise the reviews regarding each product on various sites and present it in user-friendly and easy-to-understand stats, as many customers also look for reliability and review of the product before buying.

1.2.3 Questionnaire/s



How frequently do you prefer to shop online? *

	1	2	3	4	5	
Occassionally	○	○	○	○	○	Frequently

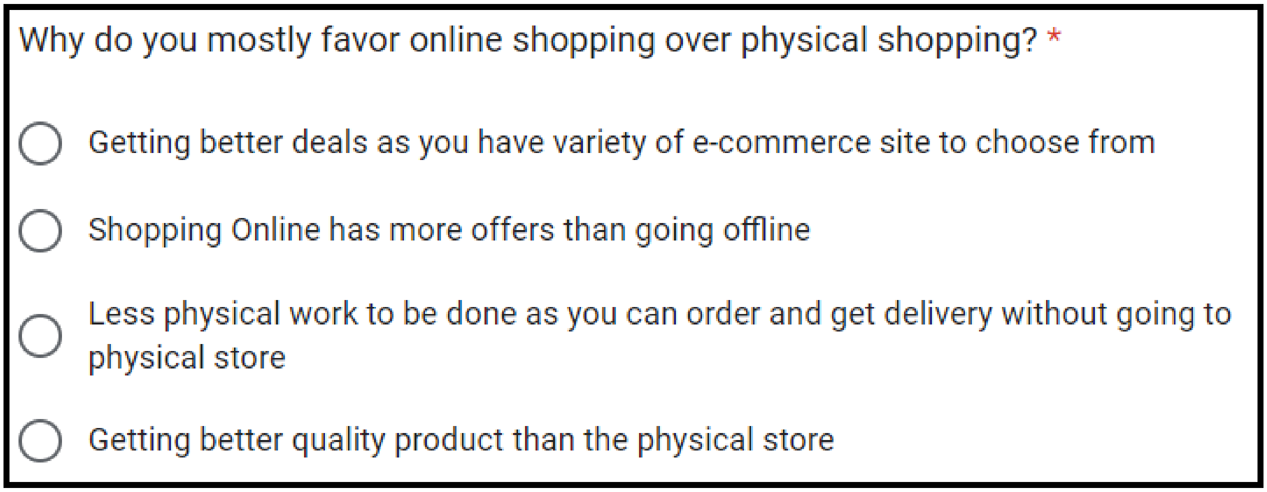


How frequently do you prefer to shop online?

51 responses

- **The intent of this Question:**
  To get to know how much people shop online and not just only check prices. Also, we get an idea of how to select server size for maintaining a large amount of data and smooth functioning of the site to prevent network traffic.

- **Observations from the Responses:**
  The majority of the people shop online very often, and very few are one who just checks prices. So due to this, the site must need an efficient database and large servers for smooth functioning of the site even when millions of people are using the site at the same time.



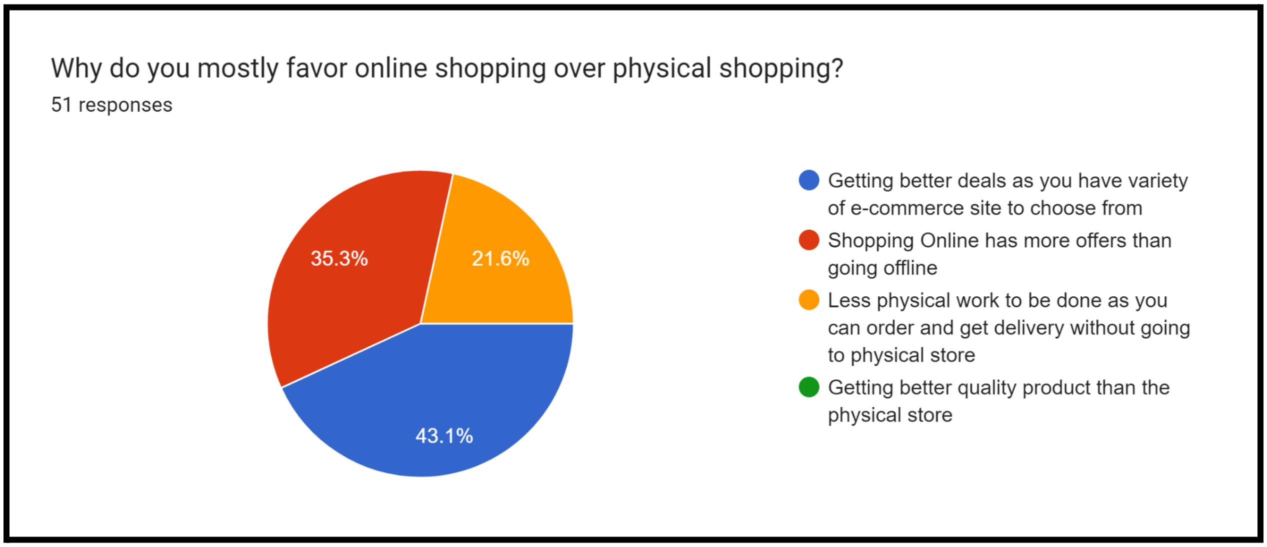
Why do you mostly favor online shopping over physical shopping? *

○ Getting better deals as you have variety of e-commerce site to choose from

○ Shopping Online has more offers than going offline

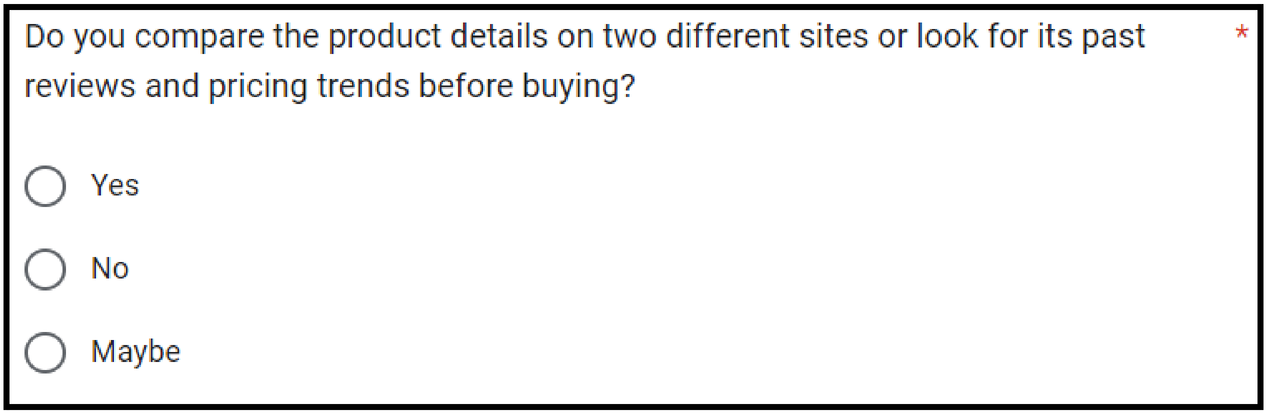
○ Less physical work to be done as you can order and get delivery without going to physical store

○ Getting better quality product than the physical store



- **The intent of this Question:**
  To get an idea behind people's interest in online shopping so that we can add new/improve current functionalities accordingly.

- **Observations from the Responses:**
  As most people shop online due to the large varieties and offers available on various products, so the customer must be notified whenever he/she gets the best deal as per his/her requirements. Also, we need to implement precise comparison and contrast between the products on various platforms by applying all possible types of offers on it to get a satisfactory deal for the customer.

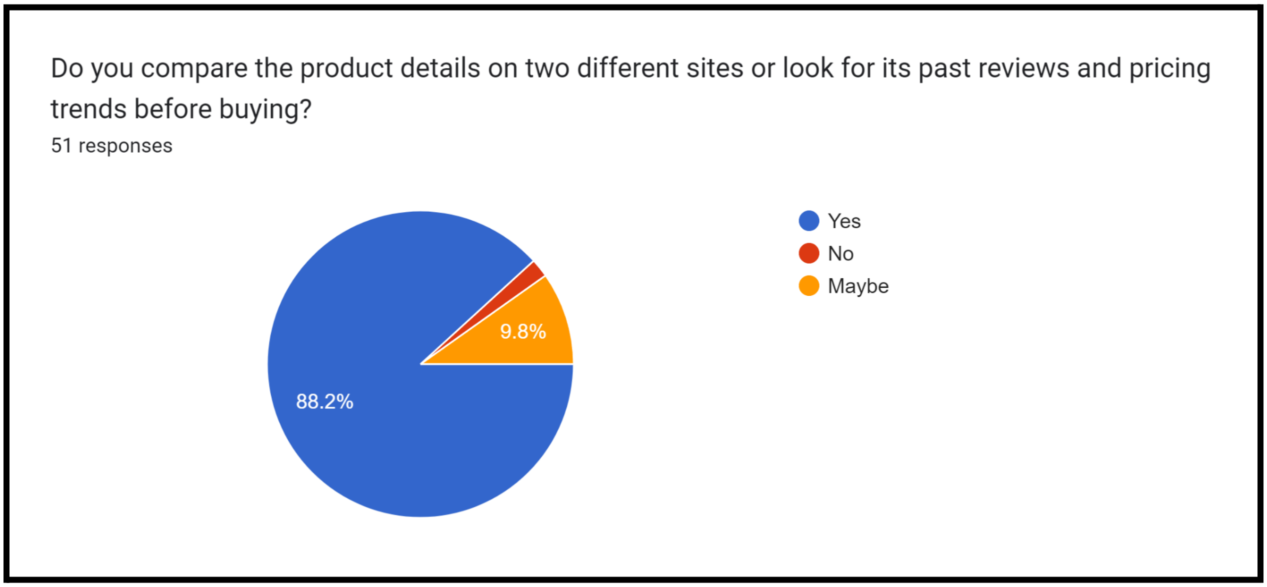


Do you compare the product details on two different sites or look for its past reviews and pricing trends before buying? *

○ Yes

○ No

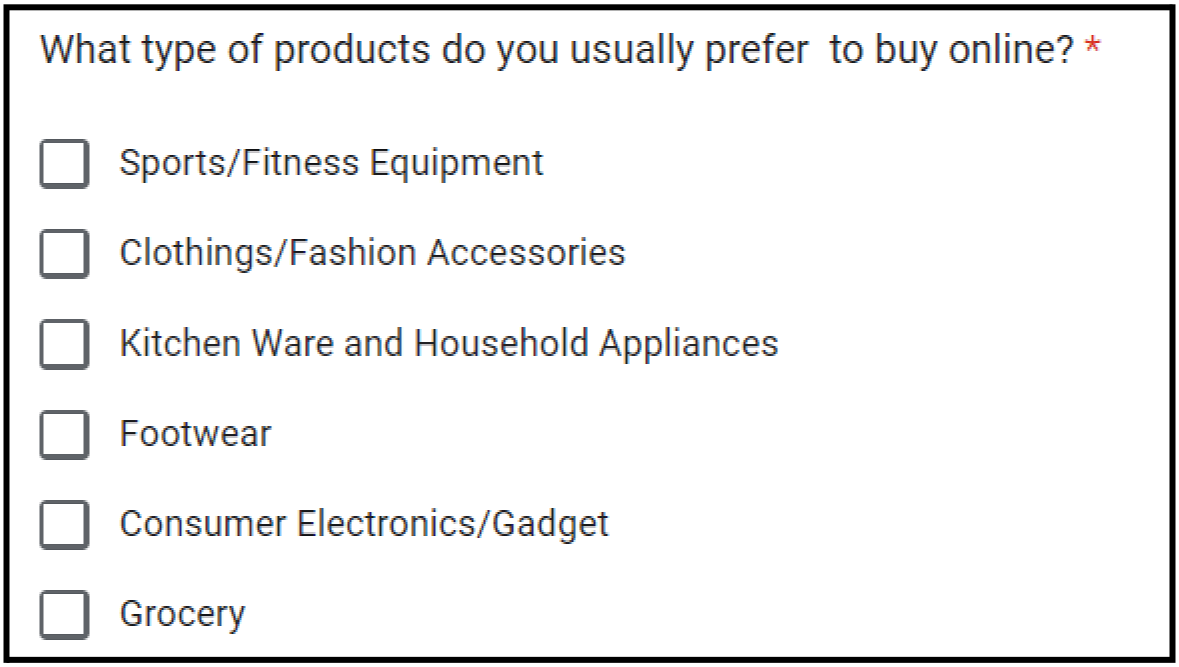
○ Maybe



- **The intent of this Question:**
  To get to know how recent statistical data we need to use for creating price charts/lists

- **Observations from the Responses:**
  From this, we get the idea that most online shoppers do an analysis of a product in a variety of manner so to make their study more precise towards their desired deal, so we need a sufficiently recent data regarding a product, i.e. up to

six months or a year.



What type of products do you usually prefer to buy online? *

□ Sports/Fitness Equipment

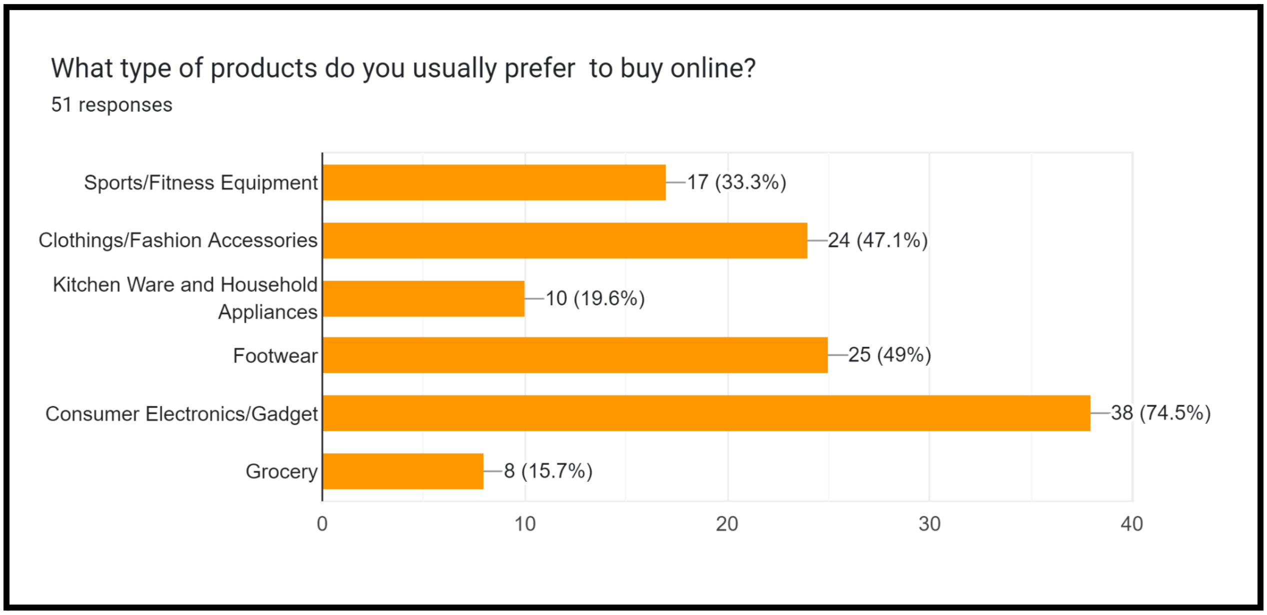
□ Clothings/Fashion Accessories

□ Kitchen Ware and Household Appliances

□ Footwear

□ Consumer Electronics/Gadget

□ Grocery

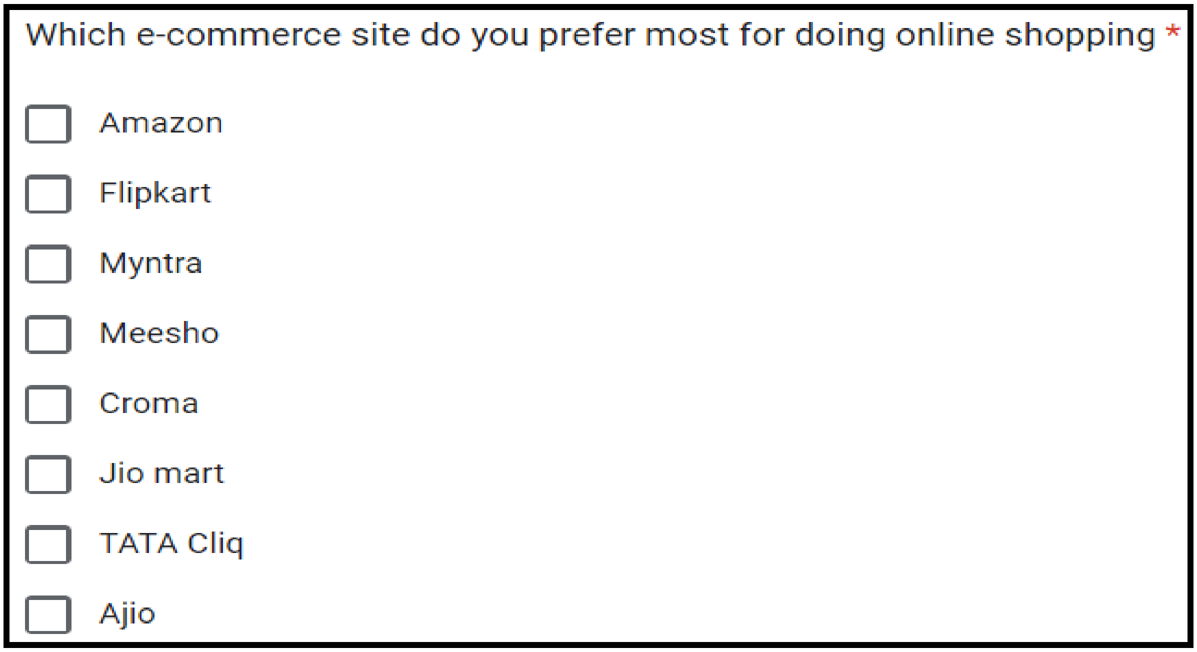


What type of products do you usually prefer to buy online?

51 responses

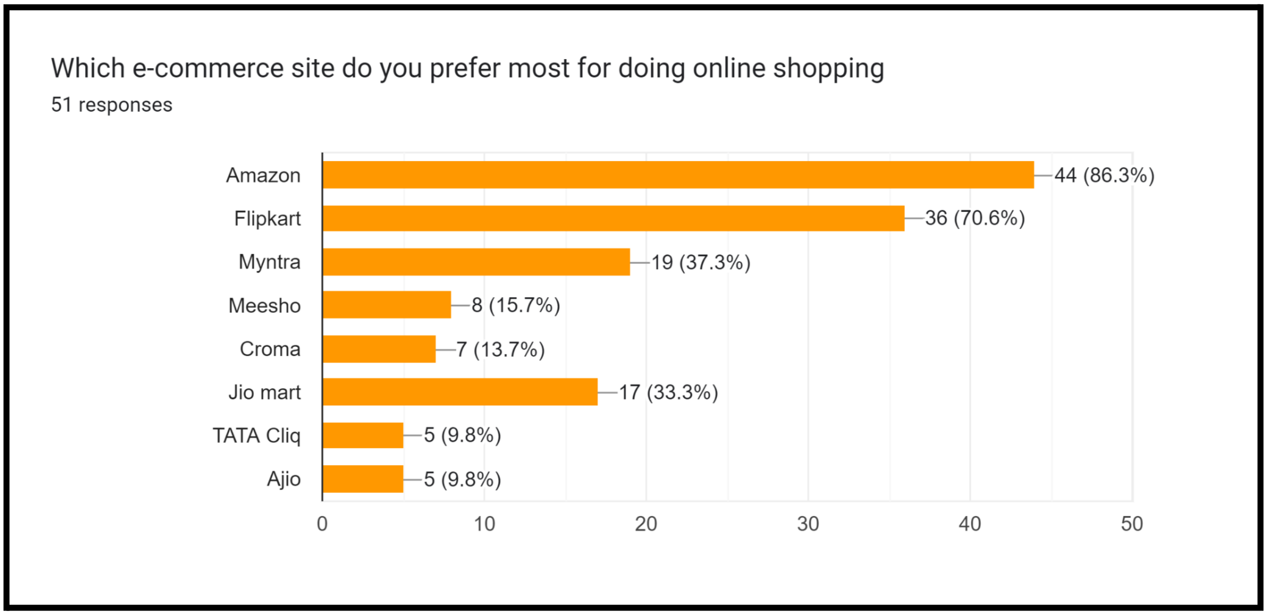
- **The intent of this Question:**
  To get an idea about product types for which we need to do a detailed price variation analysis to adjust the price chart zoom functionalities

- **Observations from the Responses:**
  From this, we get the idea of the product types that we need to study more on various platforms and provide enough details about a product that falls within this category.



Which e-commerce site do you prefer most for doing online shopping *

- □ Amazon
- □ Flipkart
- □ Myntra
- □ Meesho
- □ Croma
- □ Jio mart
- □ TATA Cliq
- □ Ajio



Which e-commerce site do you prefer most for doing online shopping

51 responses

- **The intent of this Question:**
  To get an idea of platforms from whom we need to get the past price details the most and their product study so that we can cover the needs/wishes of most of the customers

- **Observations from the Responses:**
  From this, we get the idea about the platforms that are usually preferred the most by the online shoppers so that we can now study about the products/prices/offers that are available on these platforms to cover the needs of most of the customers.

- **Combined Requirements gathered from the Questionnaire:**
  - Need for efficient database management for storing large amounts of data to carry out various filtering operations quickly, so that customer shopping is done smoothly without any loading/buffering time.
  - Using user-friendly ways to represent the statistical data regarding pricing trends as most online shoppers shop after performing several price analyses and comparisons.
  - Providing regular price drop alerts for the consumers and similar search options along with offers to help customers get the best deal.
  - Providing detailed pricing trends of well-known e-commerce sites and products that are mostly purchased online to cover the interest of many customers.

1.2.4 Observation/s

System: All in One Price History Tracker

Project Reference: SF/SJ/2022/09

Observations by: Somin Gandhi, Parth Patel

Date: 28/09/2022 **Time:** 20:00

Duration: 45 minutes **Place:** Camelcamelcamel site

Observations:

- Gives price drop alerts, and the price history of products to help the buyer makes a better buying decision.
- Keep track of the prices of items you're most interested in and feed you with a similar types of items to provide a variety of options to the user.
- Create synchronization of the contents of your Amazon Wish lists for new price monitoring and set price drop alerts based on it.
- Examine Product Price History Charts/lists to determine trends in the pricing of products.

1.3 Fact Finding Chart

Objective	Technique	Subject	Time commitment
To get background knowledge on the working and requirements for Price History Tracker.	Background readings	Similar sites and articles related to project requirements	1 day
To identify critical factors and requirements necessary for building an efficient Price History Tracker based on the pricing strategies of e-commerce platforms.	Interview	CMO at Flipkart	45 minutes
To understand and study stock keeping of products by local stores based on pricing trends of products. Also, their requirements that ease their analysis of product pricing/market scenario.	Interview	Retailer	45 minutes
To get customer feedback on the current sites and other problems they face while shopping in choosing a better bargain.	Interview	Customer	45 minutes
To get knowledge of every functionality of current Price History Tracker sites.	Observations	Similar Sites	1 hour
To understand the user's perspective on online shopping.	Questionnaire	General Public	1 day

1.4 List of Requirements

1. The database for the Price History tracker should be designed in such a manner that accessing data can be implemented quickly, as it should filter out customers' choices from a vast pool of product details.
   (**got at Background Reading, Questionnaire**)
2. The database should also contain sufficient data regarding the user's details, as most of the offers will be implemented based on the location, search/purchase history, and subscription details.
   (**got at Interview, Questionnaire**)
3. The Price History Tracker should implement statistical data like charts, graphs, and lists in clear, easy-to-understand, and adequately defined so that it gives precise and correct information regarding the price trend and offers/deals.
   (**got at Background Reading, Observation**)
4. The Price History Tracker should also implement price drop alerts and offer notifications in a well-synchronized manner as set by the customer or obtained by its search/purchase history.
   (**got at Background Reading, Questionnaire, Interview, Observations**)
5. We also need to maintain a database of timely applicable offers/deals and offers that are available on subscriptions and regular bases so that we can filter out a better bargain for the customers.
   (**got at Questionnaire, Interview**)
6. As we need to implement the zoom in/out of price trends in price charts/graphs, for that, we need to maintain data of all the products that are available on various platforms for almost one year to implement it in a better way.
   (**got at Background Reading, Observations**)
7. Also, to cover most of the customer's requirements, we need to collect detailed data on those products that fall under the highly preferred category and those platforms that are mostly preferred by online shoppers.
   (**got at Questionnaire**)
8. Also, millions of users will be using the site at the same time, so we need to take care of site traffic and the loading/buffering time. For this, we need a large server and proper site maintenance so user analysis and shopping go smoothly.
   (**got at Background Reading, Questionnaire, Interview**)

1.5 User Categories and Privileges

1.5.1 List of User Classes with their Characteristics:

- **Admin:** They have all the rights and can also manipulate the entire database if required. Its work is to just look after the task of the Database Managers and developers.

- **Database Managers:** They are the ones that will be working on the data obtained from various e-commerce sites. They can manipulate the database to keep it updated and glitch-free.

- **Developers:** They are responsible for the development and maintenance of the site. In case of technical difficulties, they will be responsible for solving them.

- **Customer:** They can only view the information related to them or filtered by them, such as price charts/lists, offers notifications, wish lists, etc. Other than this, they can also update their details and, most importantly, can set price drop alerts and offer opening/closing alerts.

1.5.2 List Functions/Privileges corresponding to User Classes:

1. **Admin:**
    a. Add/Remove_platform
    b. Add/Remove_department
    c. Can share/amend database rights of developers, customers, and Database Managers.
    d. All those functions that are accessed by the customer, developer, and database manager.

2. **Database Managers:**
    a. Add/Remove_product
    b. Add/Remove_Offers
    c. Update/Modify_database
    d. Change/Modify_Functions/Triggers

3. **Developers:**
    a. Permission_to_access_database (for testing and maintenance)

4. **Customer:**
    a. Register_itself
    b. Update_details
    c. Search_product
    d. Filter_product_details
    e. Set_price_drop_alert
    f. Set_offer_close_alert

1.6 Assumptions

- It is assumed that all the registered e-commerce sites provide every detail about the products and offers available on it regularly to admin and are updated timely as it is on our sites so that there is no misleading information from our side.
- No customers/sellers are given any specific priority unless or until he/she registers under a specific category.
- It is assumed that the site works correctly and smoothly without any network traffic/glitches, i.e., site and server maintenance is done by the hardware department of the company.
- It is assumed that the GUI-based interface of the site is taken care of by the development engineers and not by database designers
- It is assumed that our site only helps customers to get a better deal but does not provide purchase or any delivery-based facilities.

1.7 Business Constraints

- Every time user needs to log in on the website to carry out analysis by setting filters. If the user is new to the system, it must sign in using an id and fill in details, and set its username and password.
- The details about the products, their availability, and offers applicable to it should be constantly updated to ensure that the user gets the best choice in which its interest lies.
- The price drop alerts and offers notifications should be adequately implemented without any delay so that customers do not get fall short in limited-time offers/deals in which he/she is interested.
- The price charts/lists should be updated from time to time so that customers can analyze his/her product anytime and anywhere.

- Also, user searches/purchases should be appropriately tracked, and their details should be stored to feed or notify users about the best deals related to it in the future.
- The reviews of the customers regarding various products and platforms should be taken regularly and updated also based on the ratings available at various e-commerce sites.

Section 2: Noun Analysis

2.1 Noun & Verb Analysis.

Sr. No.	Nouns	Verbs
1	product	keep
2	brand	register
3	price	need
4	customer	track
5	user	should
6	card	offers
7	coupon _code	used
8	gift_voucher	categorize
9	user_detail	create
10	site	identified
11	platform	Applicable
12	Sr_id	be
13	yeraly_price	is
14	track	compare
15	Amazon	provide
16	Flipkart	will
17	alert	gives
18	search	searched
19	is_purchase	purchased
20	user_class	drop
21	footwear	add

22	electronics	remove
23	service	notify
24	product_history	existing
25	discont_rate	has
26	credit_card	start
27	offer	can
28	type	assigned
29	price_variation	zoomed
30	mapping	alert
31	date	log
32	department	fed
33	credentials	designed
34	graph	accessing
35	list	implemented
36	chart	contained
37	deal	reading
38	filter	obtained
39	choice	sets
40	period	based
41	app	maintain
42	option	got
43	data	belong
44	background	collect
45	questionnaire	loading
46	location	buffering

47	subscription	take
48	information	help
49	User_detail	examine
50	trend	determine
51	notification	make
52	observation	analyze
53	way	carry
54	buyer	represent
55	decision	perform
56	price_monitoring	known
57	variety	refer
58	wishlist	posted
59	amount	considered
60	operation	focusing
61	shopping	understood
62	shoppers	working
63	consumer	available
64	streak	Sold
65	review	
66	stats	
67	reliability	
68	interest	
69	inflation	
70	management	
71	product_id	

72	platform_id	
73	user_id	
74	pro_dept_id	
75	product_name	
76	platform_name	
77	company_name	
78	dept_name	
79	user_name	
80	type_name	
81	dis_price	
82	user_pass	

2.2 Entity-Attribute.

Sr. No.	Candidate Entity Set	Candidate Attribute Set	Candidate Relationship Set
1	Platform	Plat_id, Plat_name	Applicable_on_P, Available_on
2	Product	Pro_id, Pro_name	Available_on, Belongs_to, Sold_by, Price_hist
3	Department	Dept_id, Dept_name	Applicable_on_D, Belongs_to
4	Brand	Brand_id, Brand_name	Applicable_on_B, Sold_by
5	Offer	Sr_id, Firm_name, Type, Start_date, End_date, Disc_rate	Applicable_on_P, Applicable_on_D, Applicable_on_B
6	Track (weak)	Date, Plat_id, Price	Price_hist
7	Customer	Cust_id, Cust_name, Cust_pass, DOB, Age(), Pin_code	Sets, Searched
8	Price_alerts	A_id, Pro_id, Brand_id, Price_drop	Sets
9	Pro_hist (weak)	Pro_name, Brand_name, date	Searched
10	User_class	U_id, U_name, U_pass, U_class, {Contact_no}, {Access}, Last_status	-
11	Sold_by	Pro_id, Brand_id, Price	-
12	Available_On	Plat_id, Pro_id, Curr_stock, Disc_rate, Ratings	-

2.3 Rejected Nouns & Verbs.

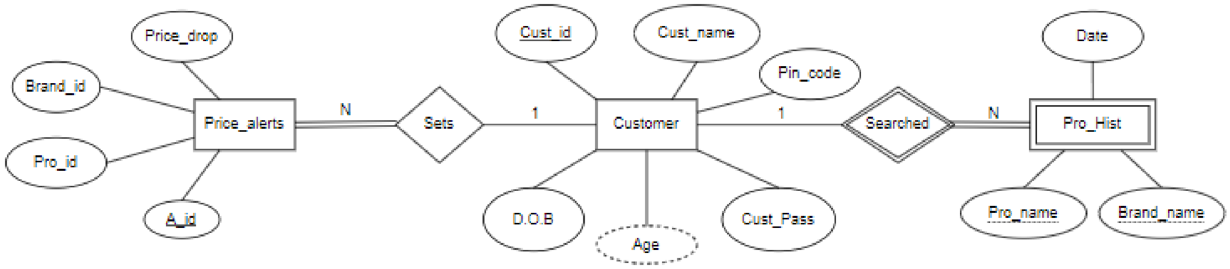
Nouns	Reject Reason
user	General
coupon _code	Duplicate
gift_voucher	Duplicate
user_detail	Attribute
site	Duplicate
yeraly_price	Duplicate
search	Association
is_purchase	Vague
service	Irrelevant
credit_card	Duplicate
price_variation	Duplicate
mapping	Irrelevant
credentials	Attribute
graph	Duplicate
list	Duplicate
chart	Duplicate
deal	Duplicate
filter	Irrelevant
choice	Duplicate
period	Vague
option	Duplicate
data	General
background	Vague

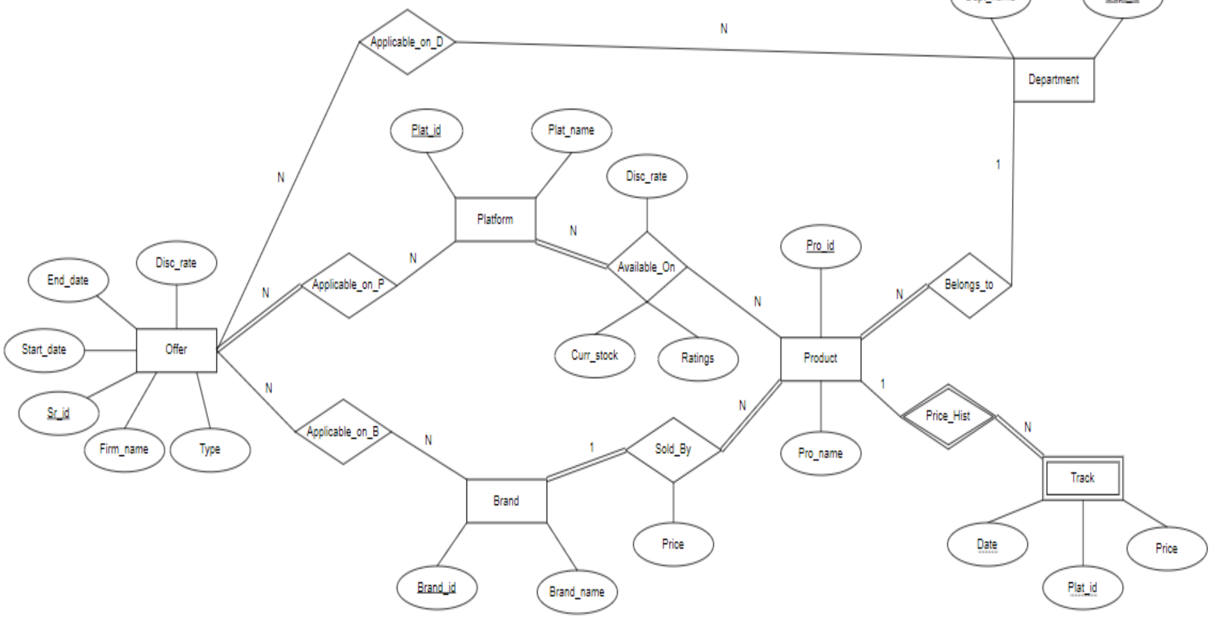
questionnaire	Vague
location	Attribute
subscription	Duplicate
information	Vague
trend	Duplicate
notification	Association
observation	Vague
way	Vague
buyer	Duplicate
decision	Irrelevant
price_monitoring	Association
variety	Duplicate
wishlist	Irrelevant
amount	Attribute
operation	Irrelevant
consumer	Duplicate
streak	Irrelevant
review	Attribute
stats	Duplicate
reliability	Vague
interest	Duplicate
inflation	Irrelevant
management	Irrelevant

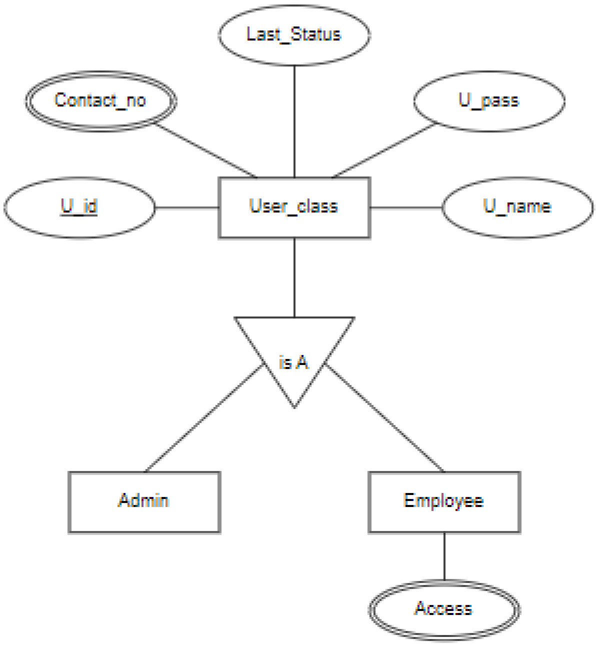
Verbs	Reject Reason
keep	General
register	Irrelevant
need	Vague
should	General
used	General
categorize	Irrelevant
create	Vague
identified	Irrelevant
be	General
is	General
compare	Vague
provide	Vague
will	General
gives	General
purchased	Irrelevant
add	General
remove	General
zoomed	Vague
existing	Vague
has	General
can	General
assigned	Irrelevant
notify	Duplicate
alert	Duplicate

log	Irrelevant
fed	Irrelevant
designed	Duplicate
implemented	Duplicate
reading	Irrelevant
contained	Duplicate
obtained	Duplicate
based	General
maintain	Irrelevant
got	General
collect	Vague
loading	Duplicate
buffering	Duplicate
take	Vague
help	Vague
examine	Duplicate
determine	Duplicate
make	General
analyze	Irrelevant
carry	Vague
represent	Vague
perform	Vague
known	General
refer	Vague
posted	Vague

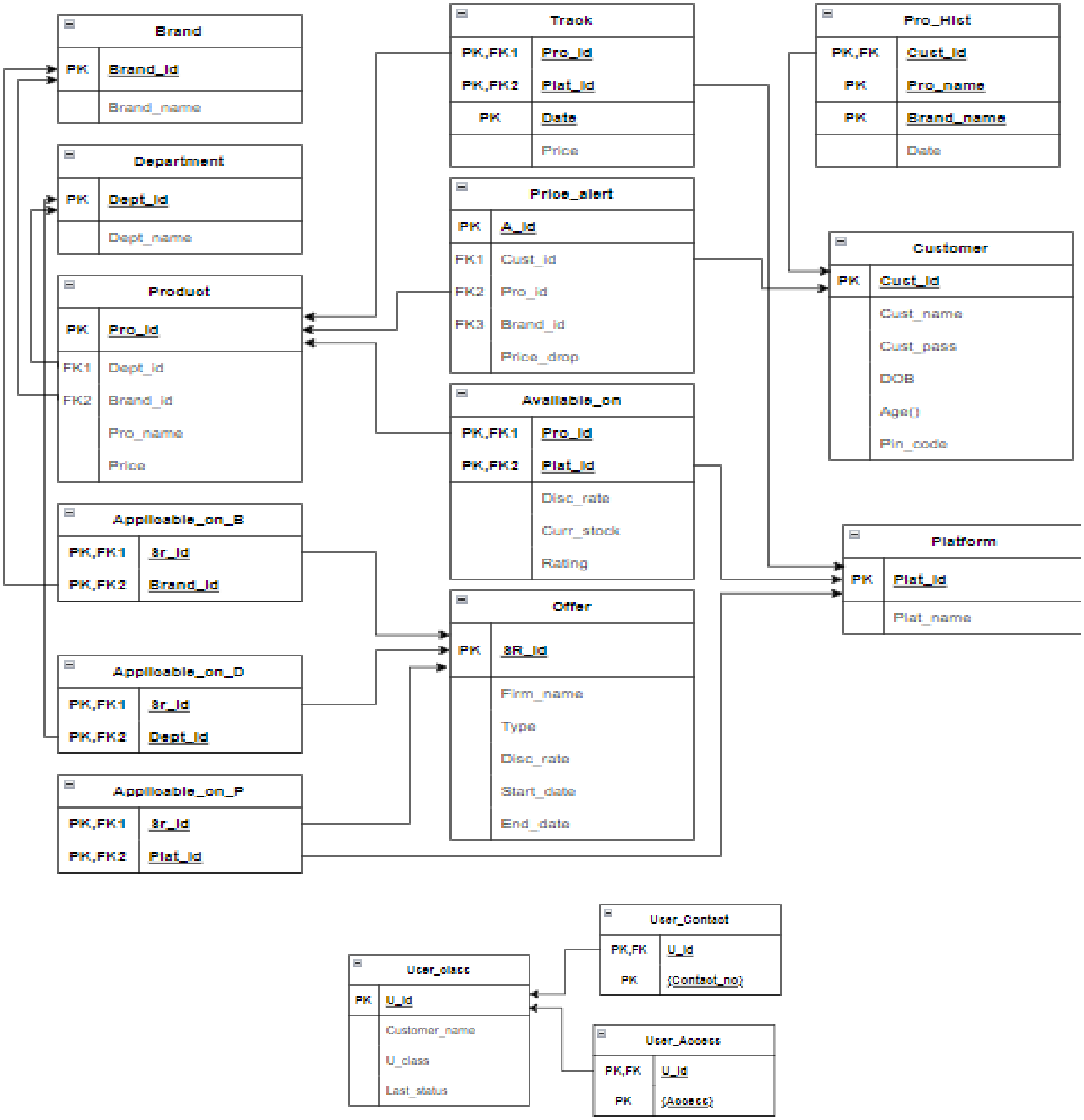
Section 3: Final ER Diagram







Section 4: Final Relational Model



Final Relations

- **Product**(Pro_id, Dept_id, Brand_id, Pro_name, Price)
  - FK Dept_id references to **Department**
  - FK Brand_id references to **Brand**
- **Platform**(Plat_id, Plat_name)
- **Department**(Dept_id, Dept_name)
- **Brand**(Brand_id, Brand_name)
- **Offer**(Sr_id, Frim_name, Type, Disc_rate, Start_date, End_date)
- **Applicable_on_P**(Sr_id, Plat_id)
  - FK Sr_id references to **Offer**
  - Fk Plat_id references to **Platform**
- **Applicable_on_D**(Sr_id, Dept_id)
  - FK Sr_id references to **Offer**
  - FK Dept_id references to **Department**
- **Applicable_on_B**(Sr_id, Brand_id)
  - FK Sr_id references to **Offer**
  - FK Brand_id references to **Brand**
- **Track**(Pro_id, Plat_id, Date, Price)
  - FK Pro_id references to **Product**
  - FK Plat_id references to **Platform**
- **Customer**(Cust_id, Cust_name, Cust_pass, DOB, Age(), Pin_code)
- **Price_alert**(A_id, Cust_id, Pro_id, Brand_id, Price_drop)
  - FK Cust_id references to **Customer**
  - FK Pro_id references to **Product**
  - FK Brand_id references to **Brand**
- **Pro_Hist**(Cust_id, Pro_name, Brand_name, Date)
  - FK Cust_id references to **Customer**
- **User_class**(U_id, customer_name, U_class, Last_status)
  - Note : As Contact_no and Access are multivalued attribute so below two separate tables are created.
- **User_contact**(U_id, { Contact_no })
  - FK U_id references to **User_class**
- **User_access**(U_id, { Access })
  - FK U_id references to **User_class**
- **Available_on**(Pro_id, Plat_id, Disc_rate, Curr_stock, Ratings)
  - FK Pro_id references to **Product**
  - FK Plat_id references to **Platform**

Section 5: Normalization and schema refinement

5.1 Functional Dependency

- **Product**(Pro_id → Dept_id, Pro_id → Brand_id, Pro_id → Pro_name, Pro_id → Price)

- **Platform**(Plat_id → Plat_name)

- **Department**(Dept_id → Dept_name)

- **Brand**(Brand_id → Brand_name)

- **Offer**(Sr_id → Firm_name, Sr_Id → Type, Sr_id → Disc_rate, Sr_id → Start_date, Sr_id → End_date)

- **Applicable_on_P**: No Dependencies

- **Applicable_on_D**: No Dependencies

- **Applicable_on_B**: No Dependencies

- **Track** : No Dependencies

- **Customer**(Cust_id → Cust_name, Cust_id → Cust_pass, Cust_id → DOB, Cust_id → Age, Cust_id → Pin_code, DOB → Age)

- **Price_alert**(A_id → Cust_id, A_id → Pro_id, A_Id → Brand_id, A_id → Price_drop)

- **Pro_Hist** : No Dependencies

- **User_class**(U_id → customer_name, U_id → U_class, U_id → Last_status)

- **User_contact** : No Dependencies

- **User_access** : No Dependencies

- **Available_on**((Pro_id, Plat_id) → Disc_rate, (Pro_id, Plat_id) → Curr_stock, (Pro_id, Plat_id) → Ratings)

5.2 Redundancy and Analysis

- **Product**(Pro_id, Dept_id, Brand_id, Pro_name, Price)
    - FK Dept_id references to **Department**
    - FK Brand_id references to **Brand**
    - **Redundancy:** There can be multiple pro_id with the same brand and department.
    - There are no transitive dependencies.
    - There are no anomalies in this entity.
- **Platform**(Plat_id, Plat_name)
    - There are no transitive dependencies.
    - There are no anomalies in this entity.
- **Department**(Dept_id, Dept_name)
    - There are no transitive dependencies.
    - There are no anomalies in this entity.
- **Brand**(Brand_id, Brand_name)
    - There are no transitive dependencies.
    - There are no anomalies in this entity.
- **Offer**(Sr_id, Firm_name, Type, Disc_rate, Start_date, End_date)
    - **Redundancy** : Different offers can be of the same Type, provide the same Disc_rate and can have the same validity.
    - There are no transitive dependencies.
    - There are no anomalies in this entity.
- **Applicable_on_P**(Sr_id, Plat_id)
    - FK Sr_id references to **Offer**
    - Fk Plat_id references to **Platform**
    - There are no transitive dependencies.
    - There are no anomalies in this entity.
- **Applicable_on_D**(Sr_id, Dept_id)
    - FK Sr_id references to **Offer**
    - FK Dept_id references to **Department**
    - There are no transitive dependencies.
    - There are no anomalies in this entity.
- **Applicable_on_B**(Sr_id, Brand_id)
    - FK Sr_id references to **Offer**
    - FK Brand_id references to **Brand**
    - There are no transitive dependencies.
    - There are no anomalies in this entity.
- **Track**(Pro_id, Plat_id, Date, Price)
    - **Redundancy:** Multiple (Pro_id, Plat_id, Date) pairs can have the same value of price.
    - FK Pro_id references to **Product**
    - FK Plat_id references to **Platform**

- **Customer**(Cust_id, Cust_name, Cust_pass, DOB, Age(), Pin_code)
  - **Redundancy** : Here we have transitive redundancy because of (Cust_id → DOB, DOB → Age implies Cust_id → Age).
  - **We removed the attribute Age. So, now there are no transitive dependencies.**
- **Price_alert**(A_id, Cust_id, Pro_id, Brand_id, Price_drop)
  - **Redundancy:** There can be multiple alerts by the same customer and multiple alerts can be of the same product by different customers.
  - FK Cust_id references to **Customer**
  - FK Pro_id references to **Product**
  - FK Brand_id references to **Brand**
- **Pro_Hist**(Cust_id, Pro_name, Brand_name, Date)
  - **Redundancy:** different customers can have searched for the same product on the same date.
  - FK Cust_id references to **Customer**
- **User_class**(U_id, customer_name, U_class, Last_status)
  - **Note:** As Contact_no and Access are multivalued attributes so below two separate tables are created.
- **User_contact**(U_id, { Contact_no })
  - FK U_id references to **User_class**
- **User_access**(U_id, { Access })
  - FK U_id references to **User_class**
- **Available_on**(Pro_id, Plat_id, Disc_rate, Curr_stock, Ratings)
  - **Redundancy:** Multiple (Pro_id, Plat_id) pairs can have the same value for all other three attributes.
  - FK Pro_id references to **Product**
  - FK Plat_id references to **Platform**

5.3 Normalization up to 3NF/BCNF

- **Product**(Pro_id, Dept_id, Brand_id, Pro_name, Price)
    - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
    - There is only one attribute (Pro_id) in the candidate key and only one candidate key and no FDs with proper subset of CK . There are no partial dependencies. Hence, it is in 2NF form.
    - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Platform**(Plat_id, Plat_name)
    - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
    - There is only one attribute (Plat_id) in the candidate key and only one candidate key and no FDs with proper subset of CK. There are no partial dependencies. Hence, it is in 2NF form.
    - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Department**(Dept_id, Dept_name)
    - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
    - There is only one attribute (Dept_id) in the candidate key and only one candidate key and no FDs with proper subset of CK. There are no partial dependencies. Hence, it is in 2NF form.
    - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Brand**(Brand_id, Brand_name)
    - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
    - There is only one attribute (Brand_id) in the candidate key and only one candidate key and no FDs with proper subset of CK. There are no partial dependencies. Hence, it is in 2NF form.
    - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Offer**(Sr_id, Frim_name, Type, Disc_rate, Start_date, End_date)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
  - There is only one attribute (Sr_id) in the candidate key and only one candidate key and no FDs with proper subset of CK. There are no partial dependencies. Hence, it is in 2NF form.
  - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Applicable_on_P**(Sr_id, Plat_id)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
  - There is only one candidate key (Sr_id, Plat_id) in the candidate key. There are no non prime attributes.Hence, it is in 2NF form.
  - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Applicable_on_D**(Sr_id, Dept_id)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
  - There is only one candidate key (Sr_id, Dept_id) in the candidate key. There are no non prime attributes.Hence, it is in 2NF form.
  - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Applicable_on_B**(Sr_id, Brand_id)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
  - There is only one candidate key (Sr_id, Brand_id) in the candidate key. There are no non prime attributes.Hence, it is in 2NF form.
  - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Track**(Pro_id, Plat_id, Date, Price)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
  - There is only one candidate key (Pro_id, Plat_id, Date) and no FDs with proper subset of CK and only one non prime attribute Price . Hence, it is in 2NF form.
  - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Customer**(Cust_id, Cust_name, Cust_pass, DOB, Pin_code)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.

  - There is only one attribute (Cust_id) in the candidate key and only one candidate key and no FDs with proper subset of CK. There are no partial dependencies. Hence, it is in 2NF form.
  - As we have removed the attribute Age. Hence there are no transitive dependencies in this entity. So, it is in 3NF form.

- **Price_alert**(A_id, Cust_id, Pro_id, Brand_id, Price_drop)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
  - There is only one attribute (A_id) in the candidate key and only one candidate key and no FDs with proper subset of CK. There are no partial dependencies. Hence, it is in 2NF form.
  - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Pro_Hist**(Cust_id, Pro_name, Brand_name, Date)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
  - There is only one candidate key (Cust_id, Pro_name, Brand_name), and no FDs with proper subset of CK and only one non prime attribute date. Hence, it is in 2NF form.
  - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **Available_on**(Pro_id, Plat_id, Disc_rate, Curr_stock, Ratings)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
  - There is only one candidate key (Pro_id, Plat_id) and no non prime attributes and no FDs with proper subset of CK.Hence, it is in 2NF form
  - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **User_class**(U_id, customer_name, U_class, Last_status)
  - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
  - There is only one attribute (U_id) in the candidate key and only one candidate key and no FDs with proper subset of CK. There are no partial dependencies. Hence, it is in 2NF form.
  - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

Note: As per ER Model we had two multivalued attributes so to convert it in 1NF we made two separate relations which are as follows.

- **User_contact**(U_id, { Contact_no })
    - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
    - There is only one candidate key (U_id, Contact_no) and no non prime attributes.Hence, it is in 2NF form.
    - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

- **User_access**(U_id, { Access })
    - This schema does not have any composite or multivalued attribute. It satisfies atomicity. Hence, it is already in 1NF form.
    - There is only one candidate key (U_id, Access) and no non prime attributes.Hence, it is in 2NF form
    - All the functional dependencies have a candidate key on the left side. So, it is in BCNF form which implies it is 3NF form.

5.4 Final Relational Schema

- **Product**(Pro_id, Dept_id, Brand_id, Pro_name, Price)
    - FK Dept_id references to **Department**
    - FK Brand_id references to **Brand**
- **Platform**(Plat_id, Plat_name)
- **Department**(Dept_id, Dept_name)
- **Brand**(Brand_id, Brand_name)
- **Offer**(Sr_id, Frim_name, Type, Disc_rate, Start_date, End_date)
- **Applicable_on_P**(Sr_id, Plat_id)
    - FK Sr_id references to **Offer**
    - Fk Plat_id references to **Platform**
- **Applicable_on_D**(Sr_id, Dept_id)
    - FK Sr_id references to **Offer**
    - FK Dept_id references to **Department**
- **Applicable_on_B**(Sr_id, Brand_id)
    - FK Sr_id references to **Offer**
    - FK Brand_id references to **Brand**
- **Track**(Pro_id, Plat_id, Date, Price)
    - FK Pro_id references to **Product**
    - FK Plat_id references to **Platform**
- **Customer**(Cust_id, Cust_name, Cust_pass, DOB, Pin_code)
- **Price_alert**(A_id, Cust_id, Pro_id, Brand_id, Price_drop)
    - FK Cust_id references to **Customer**
    - FK Pro_id references to **Product**
    - FK Brand_id references to **Brand**

- **Pro_Hist**(Cust_id, Pro_name, Brand_name, Date)
  - FK Cust_id references to **Customer**
- **User_class**(U_id, customer_name, U_class, Last_status)
  - Note : As Contact_no and Access are multivalued attribute so below two separate tables are created.
- **User_contact**(U_id, { Contact_no })
  - FK U_id references to **User_class**
- **User_access**(U_id, { Access })
  - FK U_id references to **User_class**
- **Available_on**(Pro_id, Plat_id, Disc_rate, Curr_stock, Ratings)
  - FK Pro_id references to **Product**
  - FK Plat_id references to **Platform**

Section 6: Final DDL Script

6.1 Create Table Script:

1. **Customer:**

```
CREATE TABLE IF NOT EXISTS price_history.customer
(
    cust_id bigint NOT NULL,
    cust_name character varying(30) COLLATE pg_catalog."default" NOT NULL,
    cust_pass character varying(8) COLLATE pg_catalog."default" NOT NULL,
    dob date,
    pin_code integer,
    email character varying(255) COLLATE pg_catalog."default",
    phone_num bigint,
    CONSTRAINT customer_pkey PRIMARY KEY (cust_id),
    CONSTRAINT customer_phone_num_check CHECK (phone_num > 999999999 AND
phone_num < '10000000000'::bigint),
    CONSTRAINT customer_dob_check2 CHECK (dob <= '2003-12-31'::date)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.customer
    OWNER to postgres;
```

2. **Platform:**

```
CREATE TABLE IF NOT EXISTS price_history.platform
(
    plat_id bigint NOT NULL,
    plat_name character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT platform_pkey PRIMARY KEY (plat_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.platform
    OWNER to postgres;
```

3. Brand:

```
CREATE TABLE IF NOT EXISTS price_history.brand
(
    brand_id bigint NOT NULL,
    brand_name character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT brand_pkey PRIMARY KEY (brand_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.brand
    OWNER to postgres;
```

4. Department:

```
CREATE TABLE IF NOT EXISTS price_history.department
(
    dept_id bigint NOT NULL,
    dept_name character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT department_pkey PRIMARY KEY (dept_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.department
    OWNER to postgres;
```

5. Product:

```
CREATE TABLE IF NOT EXISTS price_history.product
(
    pro_id bigint NOT NULL,
    pro_name character varying COLLATE pg_catalog."default" NOT NULL,
    price integer NOT NULL,
    dept_id bigint,
    brand_id bigint,
    CONSTRAINT product_pkey PRIMARY KEY (pro_id),
    CONSTRAINT product_brand_id_fkey FOREIGN KEY (brand_id)
        REFERENCES price_history.brand (brand_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT product_dept_id_fkey FOREIGN KEY (dept_id)
        REFERENCES price_history.department (dept_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)
```

```
TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.product
    OWNER to postgres;
```

Trigger Associated to Product table:

```
create or replace function checkpricedrop()
returns trigger
language 'plpgsql'
as $body$
DECLARE alert record;
BEGIN
FOR alert in (select A_id, pro_id, brand_id, price_drop from price_alert where
pro_id=new.pro_id and brand_id=new.brand_id)
loop
    if new.price <= alert.price_drop THEN
        delete from price_alert where A_id = alert.A_id;
    end if;
end loop;
return new;
end;
$body$

create or replace trigger trig
before insert or UPDATE
on product
for each row
execute procedure checkpricedrop();
```

6. Price_alert:

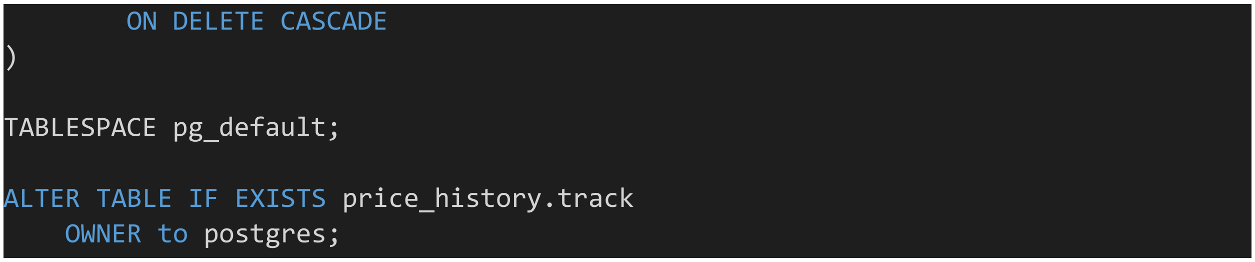
```
CREATE TABLE IF NOT EXISTS price_history.price_alert
(
    a_id bigint NOT NULL,
    pro_id bigint NOT NULL,
    brand_id bigint NOT NULL,
    cust_id bigint NOT NULL,
    price_drop integer,
    CONSTRAINT price_alert_pkey PRIMARY KEY (a_id),
    CONSTRAINT price_alert_brand_id_fkey FOREIGN KEY (brand_id)
        REFERENCES price_history.brand (brand_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT price_alert_cust_id_fkey FOREIGN KEY (cust_id)
        REFERENCES price_history.customer (cust_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT price_alert_pro_id_fkey FOREIGN KEY (pro_id)
        REFERENCES price_history.product (pro_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT price_alert_price_drop_check CHECK (price_drop >= 0)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.price_alert
    OWNER to postgres;
```

7. Pro_hist:

```
CREATE TABLE IF NOT EXISTS price_history.pro_hist
(
    cust_id bigint NOT NULL,
    pro_name character varying COLLATE pg_catalog."default" NOT NULL,
    brand_name character varying COLLATE pg_catalog."default" NOT NULL,
    last_date date,
    CONSTRAINT pro_hist_pkey PRIMARY KEY (cust_id, pro_name, brand_name),
    CONSTRAINT pro_hist_cust_id_fkey FOREIGN KEY (cust_id)
        REFERENCES price_history.customer (cust_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.pro_hist
    OWNER to postgres;
```

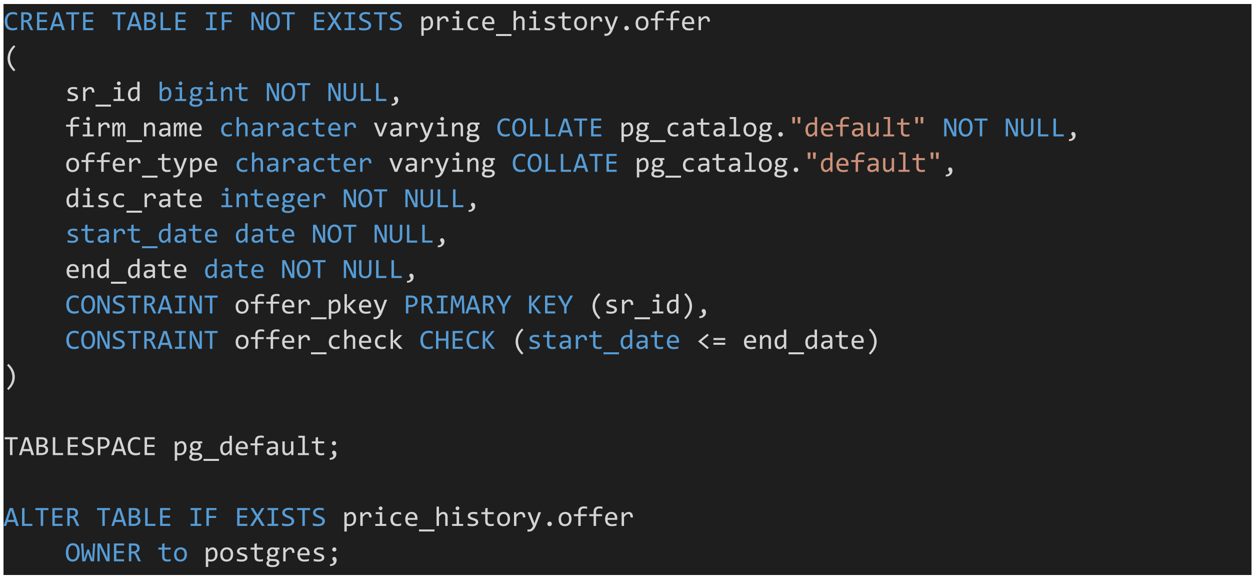
8. Track:

```
CREATE TABLE IF NOT EXISTS price_history.track
(
    pro_id bigint NOT NULL,
    plat_id bigint NOT NULL,
    curr_date date NOT NULL,
    price integer NOT NULL,
    CONSTRAINT track_pkey PRIMARY KEY (pro_id, plat_id, curr_date),
    CONSTRAINT track_plat_id_fkey FOREIGN KEY (plat_id)
        REFERENCES price_history.platform (plat_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT track_pro_id_fkey FOREIGN KEY (pro_id)
        REFERENCES price_history.product (pro_id) MATCH SIMPLE
        ON UPDATE CASCADE
```



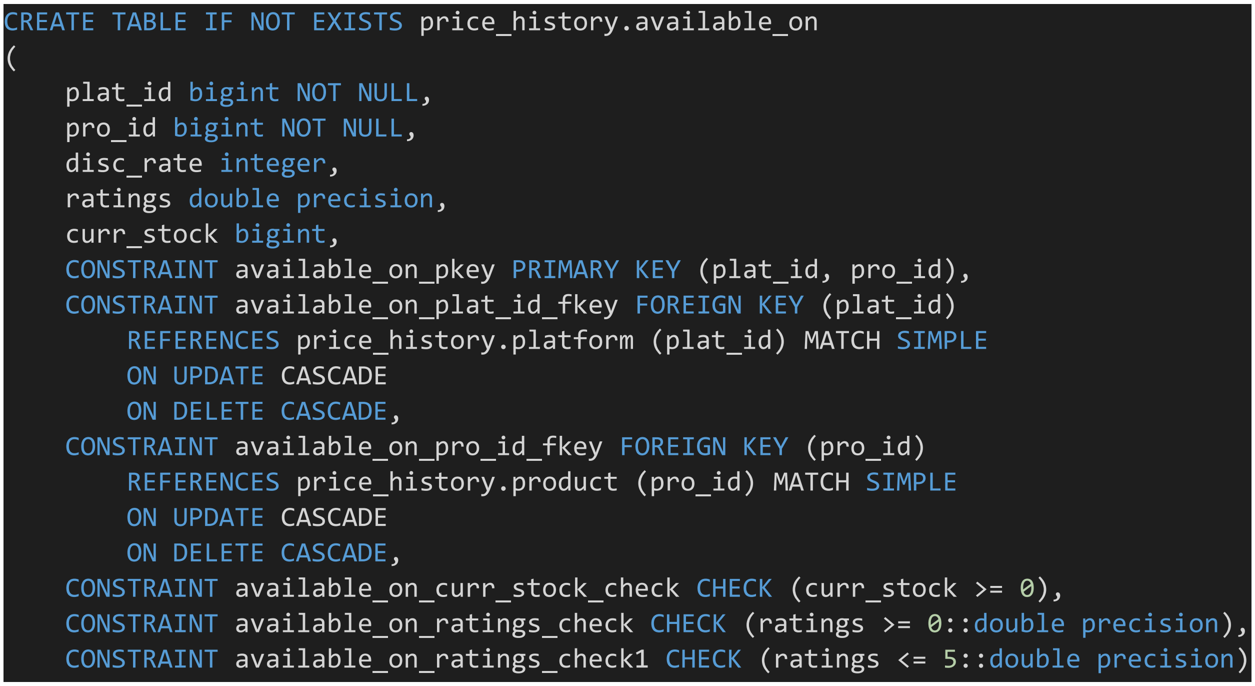
```
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.track
    OWNER to postgres;
```

9. Offer:

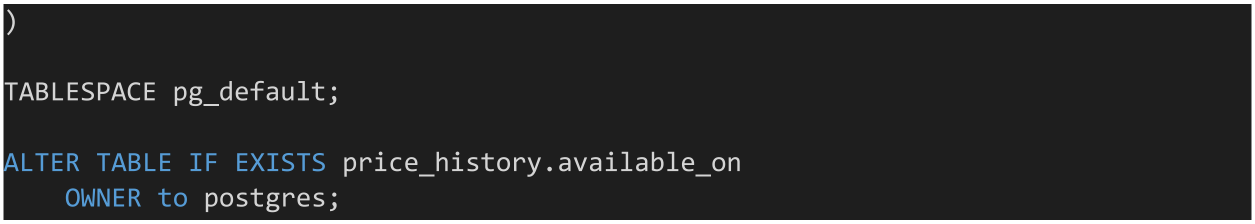


```
CREATE TABLE IF NOT EXISTS price_history.offer
(
    sr_id bigint NOT NULL,
    firm_name character varying COLLATE pg_catalog."default" NOT NULL,
    offer_type character varying COLLATE pg_catalog."default",
    disc_rate integer NOT NULL,
    start_date date NOT NULL,
    end_date date NOT NULL,
    CONSTRAINT offer_pkey PRIMARY KEY (sr_id),
    CONSTRAINT offer_check CHECK (start_date <= end_date)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.offer
    OWNER to postgres;
```

10. available_on:

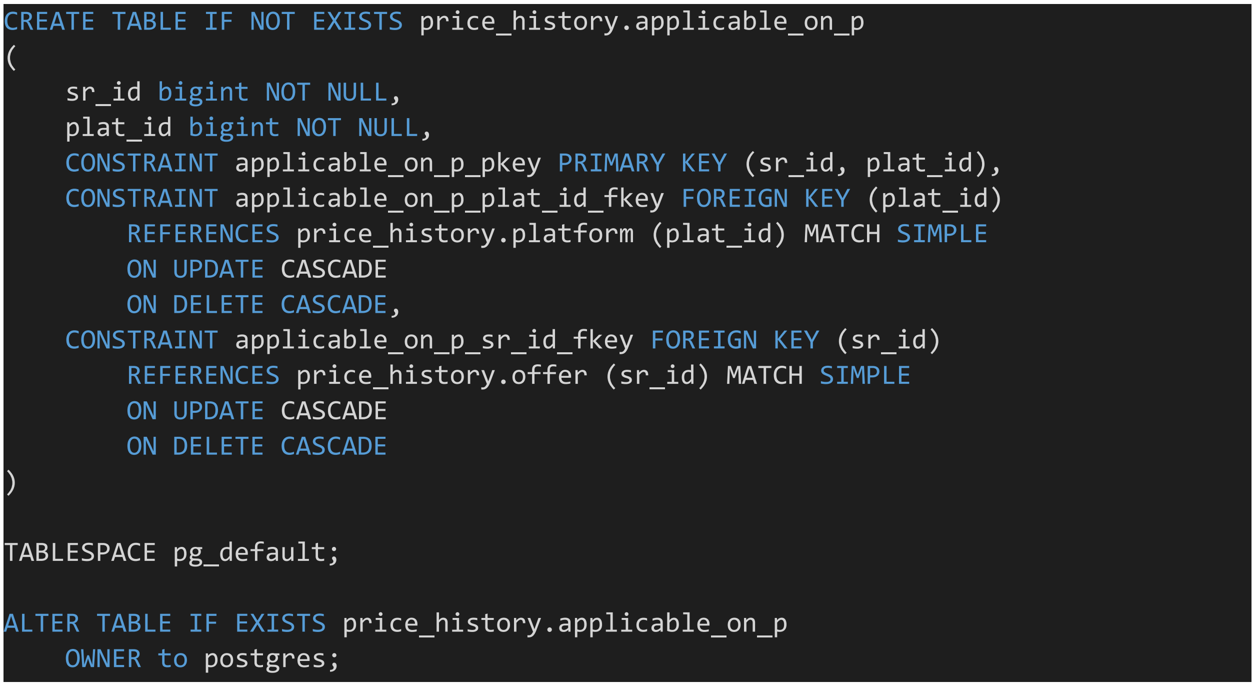


```
CREATE TABLE IF NOT EXISTS price_history.available_on
(
    plat_id bigint NOT NULL,
    pro_id bigint NOT NULL,
    disc_rate integer,
    ratings double precision,
    curr_stock bigint,
    CONSTRAINT available_on_pkey PRIMARY KEY (plat_id, pro_id),
    CONSTRAINT available_on_plat_id_fkey FOREIGN KEY (plat_id)
        REFERENCES price_history.platform (plat_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT available_on_pro_id_fkey FOREIGN KEY (pro_id)
        REFERENCES price_history.product (pro_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT available_on_curr_stock_check CHECK (curr_stock >= 0),
    CONSTRAINT available_on_ratings_check CHECK (ratings >= 0::double precision),
    CONSTRAINT available_on_ratings_check1 CHECK (ratings <= 5::double precision)
```



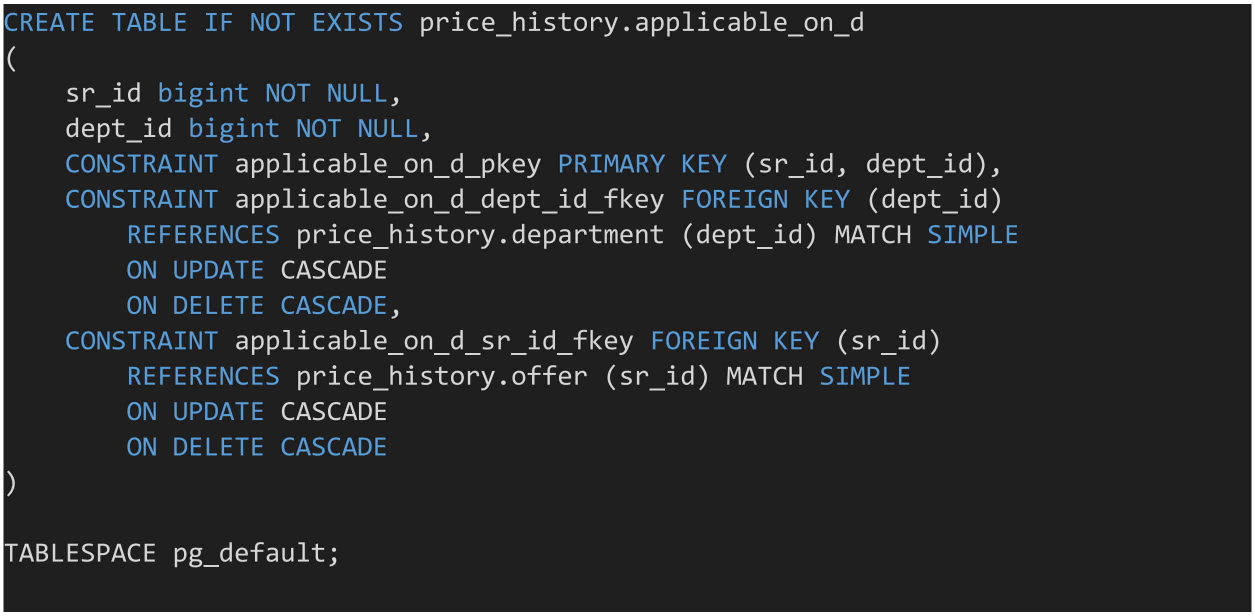
```
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.available_on
    OWNER to postgres;
```

11. applicable_on_p:



```
CREATE TABLE IF NOT EXISTS price_history.applicable_on_p
(
    sr_id bigint NOT NULL,
    plat_id bigint NOT NULL,
    CONSTRAINT applicable_on_p_pkey PRIMARY KEY (sr_id, plat_id),
    CONSTRAINT applicable_on_p_plat_id_fkey FOREIGN KEY (plat_id)
        REFERENCES price_history.platform (plat_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT applicable_on_p_sr_id_fkey FOREIGN KEY (sr_id)
        REFERENCES price_history.offer (sr_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.applicable_on_p
    OWNER to postgres;
```

12. applicable_on_d:



```
CREATE TABLE IF NOT EXISTS price_history.applicable_on_d
(
    sr_id bigint NOT NULL,
    dept_id bigint NOT NULL,
    CONSTRAINT applicable_on_d_pkey PRIMARY KEY (sr_id, dept_id),
    CONSTRAINT applicable_on_d_dept_id_fkey FOREIGN KEY (dept_id)
        REFERENCES price_history.department (dept_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT applicable_on_d_sr_id_fkey FOREIGN KEY (sr_id)
        REFERENCES price_history.offer (sr_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;
```

```
ALTER TABLE IF EXISTS price_history.applicable_on_d
    OWNER to postgres;
```

13. applicable_on_b:

```
CREATE TABLE IF NOT EXISTS price_history.applicable_on_b
(
    sr_id bigint NOT NULL,
    brand_id bigint NOT NULL,
    CONSTRAINT applicable_on_b_pkey PRIMARY KEY (sr_id, brand_id),
    CONSTRAINT applicable_on_b_brand_id_fkey FOREIGN KEY (brand_id)
        REFERENCES price_history.brand (brand_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT applicable_on_b_sr_id_fkey FOREIGN KEY (sr_id)
        REFERENCES price_history.offer (sr_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.applicable_on_b
    OWNER to postgres;
```

14. user_class:

```
CREATE TABLE IF NOT EXISTS price_history.user_class
(
    u_id bigint NOT NULL,
    u_name character varying(30) COLLATE pg_catalog."default" NOT NULL,
    u_pass character varying(8) COLLATE pg_catalog."default" NOT NULL,
    u_class character varying(10) COLLATE pg_catalog."default" NOT NULL,
    last_status date NOT NULL,
    CONSTRAINT user_class_pkey PRIMARY KEY (u_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.user_class
    OWNER to postgres;
```

15. user_contact:

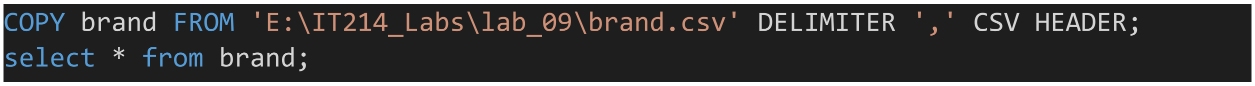
```
CREATE TABLE IF NOT EXISTS price_history.user_contact
(
    u_id bigint NOT NULL,
    contact_no bigint NOT NULL,
    CONSTRAINT user_contact_u_id_fkey FOREIGN KEY (u_id)
        REFERENCES price_history.user_class (u_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.user_contact
    OWNER to postgres;
```

16. user_access:

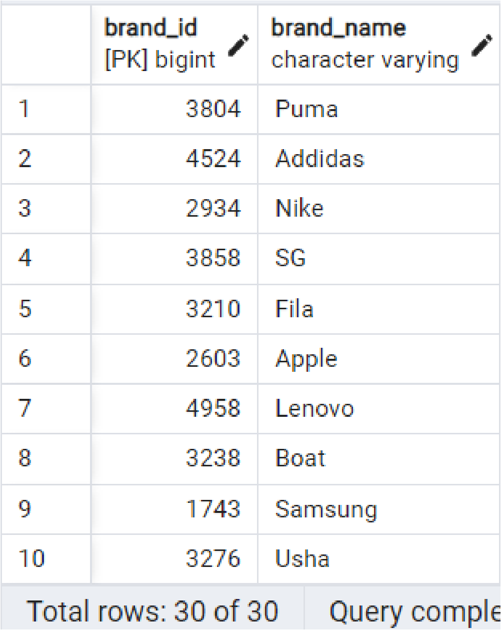
```
CREATE TABLE IF NOT EXISTS price_history.user_access
(
    u_id bigint NOT NULL,
    access_rel character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT user_access_pkey PRIMARY KEY (u_id, access_rel),
    CONSTRAINT user_access_u_id_fkey FOREIGN KEY (u_id)
        REFERENCES price_history.user_class (u_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS price_history.user_access
    OWNER to postgres;
```

6.2 Snapshots of Tables:

1. brand:



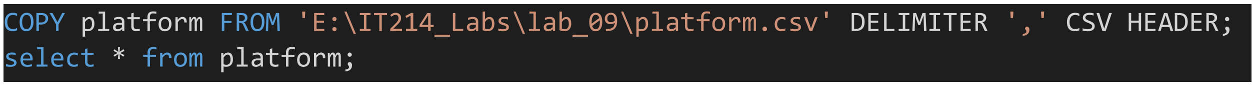
```
COPY brand FROM 'E:\IT214_Labs\lab_09\brand.csv' DELIMITER ',' CSV HEADER;
select * from brand;
```



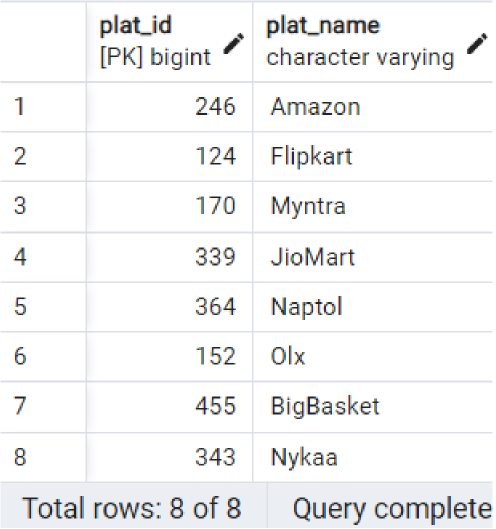
	brand_id [PK] bigint	brand_name character varying
1	3804	Puma
2	4524	Addidas
3	2934	Nike
4	3858	SG
5	3210	Fila
6	2603	Apple
7	4958	Lenovo
8	3238	Boat
9	1743	Samsung
10	3276	Usha

Total rows: 30 of 30 Query comple

2. platform:



```
COPY platform FROM 'E:\IT214_Labs\lab_09\platform.csv' DELIMITER ',' CSV HEADER;
select * from platform;
```

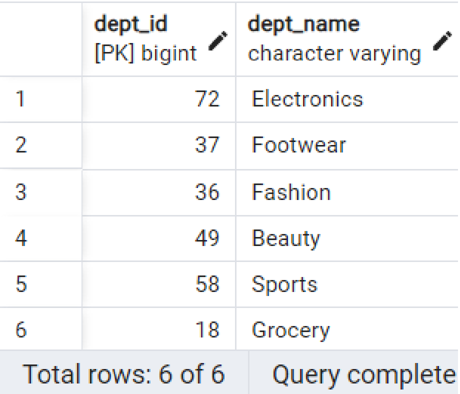


	plat_id [PK] bigint	plat_name character varying
1	246	Amazon
2	124	Flipkart
3	170	Myntra
4	339	JioMart
5	364	Naptol
6	152	Olx
7	455	BigBasket
8	343	Nykaa

Total rows: 8 of 8 Query complete

3. Department:

```
COPY department FROM 'E:\IT214_Labs\lab_09\department.csv' DELIMITER ',' CSV
HEADER;
select * from department;
```

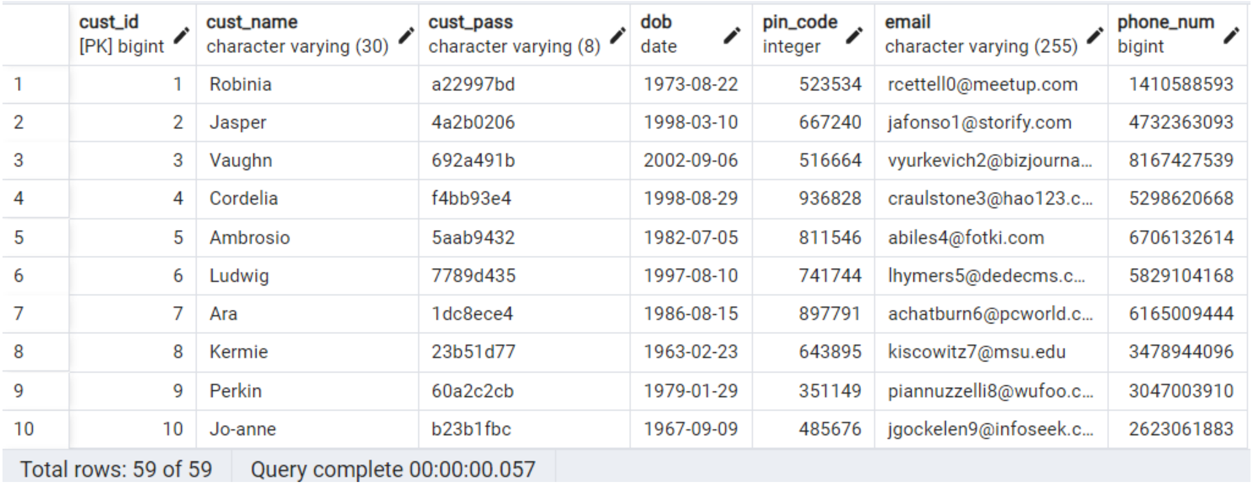


	dept_id [PK] bigint	dept_name character varying
1	72	Electronics
2	37	Footwear
3	36	Fashion
4	49	Beauty
5	58	Sports
6	18	Grocery

Total rows: 6 of 6 Query complete

4. Customer:

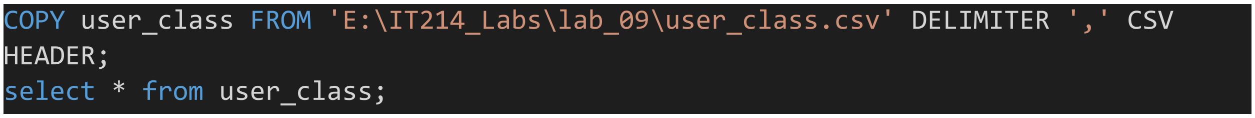
```
COPY customer FROM 'E:\IT214_Labs\lab_09\customer.csv' DELIMITER ',' CSV HEADER;
select * from customer;
```



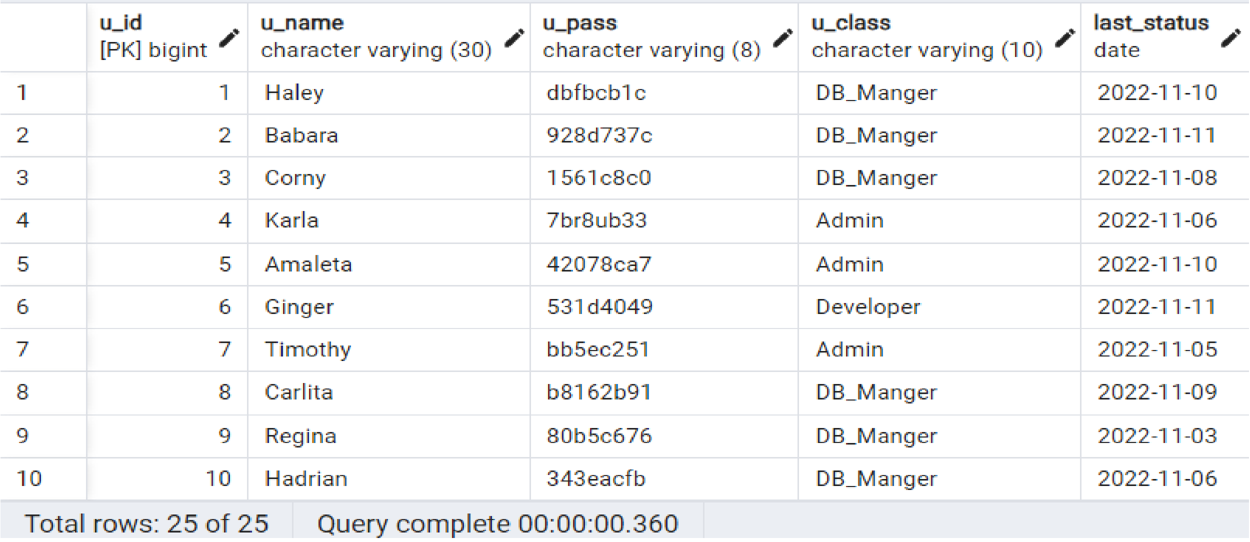
	cust_id [PK] bigint	cust_name character varying (30)	cust_pass character varying (8)	dob date	pin_code integer	email character varying (255)	phone_num bigint
1	1	Robinia	a22997bd	1973-08-22	523534	rcettell0@meetup.com	1410588593
2	2	Jasper	4a2b0206	1998-03-10	667240	jafonso1@storify.com	4732363093
3	3	Vaughn	692a491b	2002-09-06	516664	vyurkevich2@bizjourna...	8167427539
4	4	Cordelia	f4bb93e4	1998-08-29	936828	craulstone3@hao123.c...	5298620668
5	5	Ambrosio	5aab9432	1982-07-05	811546	abiles4@fotki.com	6706132614
6	6	Ludwig	7789d435	1997-08-10	741744	lhymers5@dedecms.c...	5829104168
7	7	Ara	1dc8ece4	1986-08-15	897791	achatburn6@pcworld.c...	6165009444
8	8	Kermie	23b51d77	1963-02-23	643895	kiscowitz7@msu.edu	3478944096
9	9	Perkin	60a2c2cb	1979-01-29	351149	piannuzzelli8@wufoo.c...	3047003910
10	10	Jo-anne	b23b1fbc	1967-09-09	485676	jgockelen9@infoseek.c...	2623061883

Total rows: 59 of 59 Query complete 00:00:00.057

5. User_class:



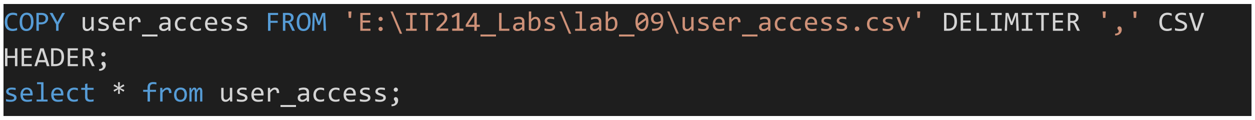
```
COPY user_class FROM 'E:\IT214_Labs\lab_09\user_class.csv' DELIMITER ',' CSV
HEADER;
select * from user_class;
```



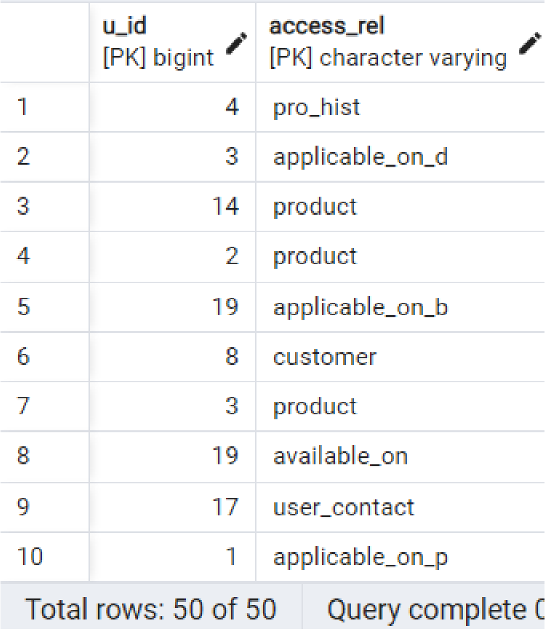
	u_id [PK] bigint	u_name character varying (30)	u_pass character varying (8)	u_class character varying (10)	last_status date
1	1	Haley	dbfbcb1c	DB_Manger	2022-11-10
2	2	Babara	928d737c	DB_Manger	2022-11-11
3	3	Corny	1561c8c0	DB_Manger	2022-11-08
4	4	Karla	7br8ub33	Admin	2022-11-06
5	5	Amaleta	42078ca7	Admin	2022-11-10
6	6	Ginger	531d4049	Developer	2022-11-11
7	7	Timothy	bb5ec251	Admin	2022-11-05
8	8	Carlita	b8162b91	DB_Manger	2022-11-09
9	9	Regina	80b5c676	DB_Manger	2022-11-03
10	10	Hadrian	343eacfb	DB_Manger	2022-11-06

Total rows: 25 of 25 Query complete 00:00:00.360

6. User_access:



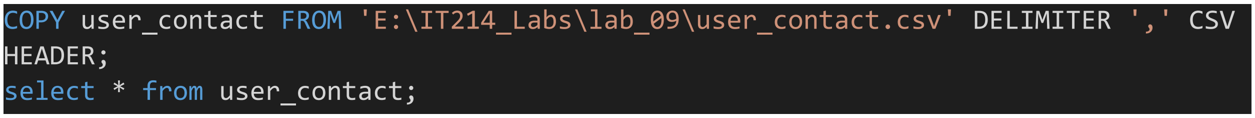
```
COPY user_access FROM 'E:\IT214_Labs\lab_09\user_access.csv' DELIMITER ',' CSV
HEADER;
select * from user_access;
```



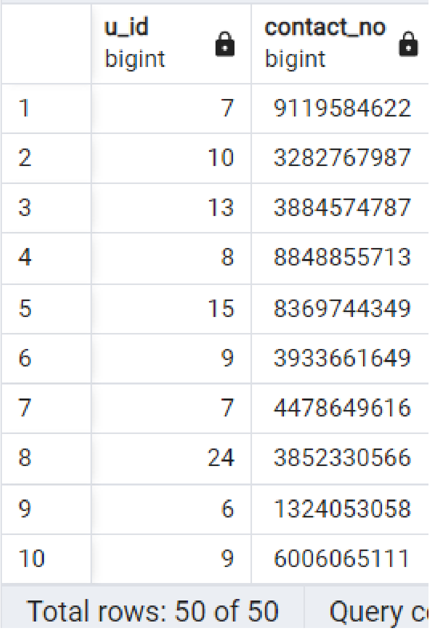
	u_id [PK] bigint	access_rel [PK] character varying
1	4	pro_hist
2	3	applicable_on_d
3	14	product
4	2	product
5	19	applicable_on_b
6	8	customer
7	3	product
8	19	available_on
9	17	user_contact
10	1	applicable_on_p

Total rows: 50 of 50 Query complete (

7. User_contact:



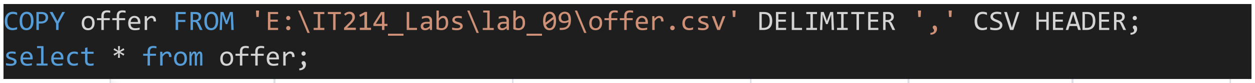
```
COPY user_contact FROM 'E:\IT214_Labs\lab_09\user_contact.csv' DELIMITER ',' CSV
HEADER;
select * from user_contact;
```



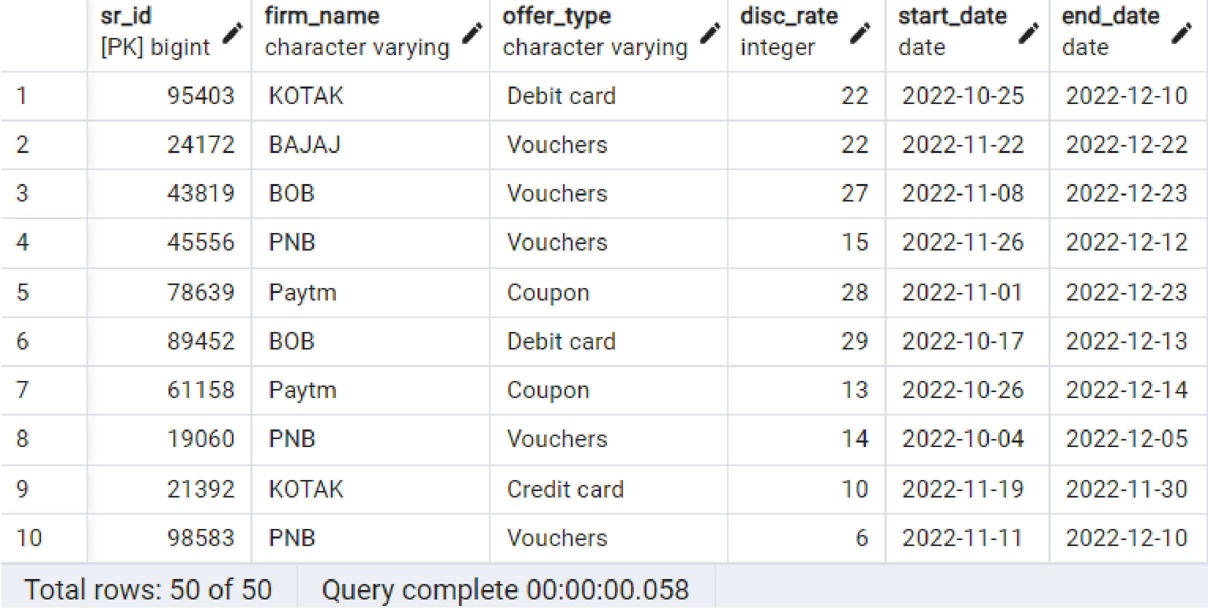
	u_id bigint	contact_no bigint
1	7	9119584622
2	10	3282767987
3	13	3884574787
4	8	8848855713
5	15	8369744349
6	9	3933661649
7	7	4478649616
8	24	3852330566
9	6	1324053058
10	9	6006065111

Total rows: 50 of 50 Query c

8. Offer:



```
COPY offer FROM 'E:\IT214_Labs\lab_09\offer.csv' DELIMITER ',' CSV HEADER;
select * from offer;
```

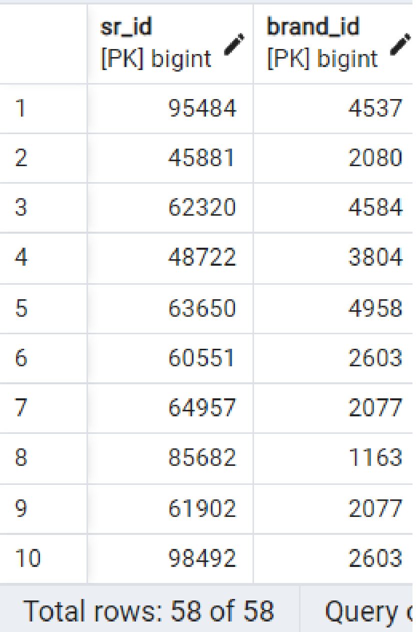


	sr_id [PK] bigint	firm_name character varying	offer_type character varying	disc_rate integer	start_date date	end_date date
1	95403	KOTAK	Debit card	22	2022-10-25	2022-12-10
2	24172	BAJAJ	Vouchers	22	2022-11-22	2022-12-22
3	43819	BOB	Vouchers	27	2022-11-08	2022-12-23
4	45556	PNB	Vouchers	15	2022-11-26	2022-12-12
5	78639	Paytm	Coupon	28	2022-11-01	2022-12-23
6	89452	BOB	Debit card	29	2022-10-17	2022-12-13
7	61158	Paytm	Coupon	13	2022-10-26	2022-12-14
8	19060	PNB	Vouchers	14	2022-10-04	2022-12-05
9	21392	KOTAK	Credit card	10	2022-11-19	2022-11-30
10	98583	PNB	Vouchers	6	2022-11-11	2022-12-10

Total rows: 50 of 50 Query complete 00:00:00.058

9. Applicable_on_b:

```
COPY applicable_on_b FROM 'E:\IT214_Labs\lab_09\applicable_on_b.csv' DELIMITER
',' CSV HEADER;
select * from applicable_on_b;
```

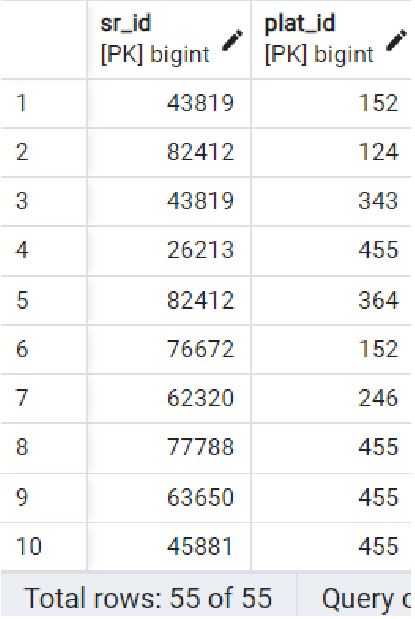


	sr_id [PK] bigint	brand_id [PK] bigint
1	95484	4537
2	45881	2080
3	62320	4584
4	48722	3804
5	63650	4958
6	60551	2603
7	64957	2077
8	85682	1163
9	61902	2077
10	98492	2603

Total rows: 58 of 58 Query c

10. Applicable_on_p:

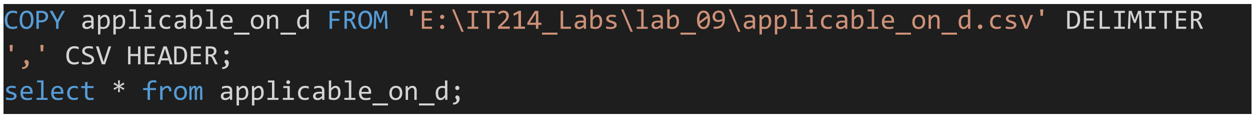
```
COPY applicable_on_p FROM 'E:\IT214_Labs\lab_09\applicable_on_p.csv' DELIMITER
',' CSV HEADER;
select * from applicable_on_p;
```



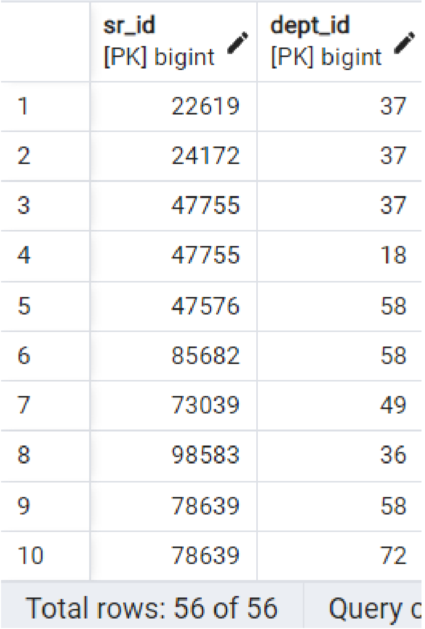
	sr_id [PK] bigint	plat_id [PK] bigint
1	43819	152
2	82412	124
3	43819	343
4	26213	455
5	82412	364
6	76672	152
7	62320	246
8	77788	455
9	63650	455
10	45881	455

Total rows: 55 of 55 Query c

11. Applicable_on_d:



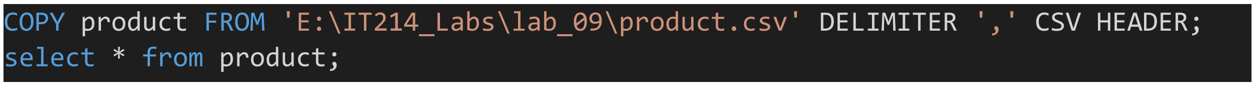
```
COPY applicable_on_d FROM 'E:\IT214_Labs\lab_09\applicable_on_d.csv' DELIMITER
',' CSV HEADER;
select * from applicable_on_d;
```



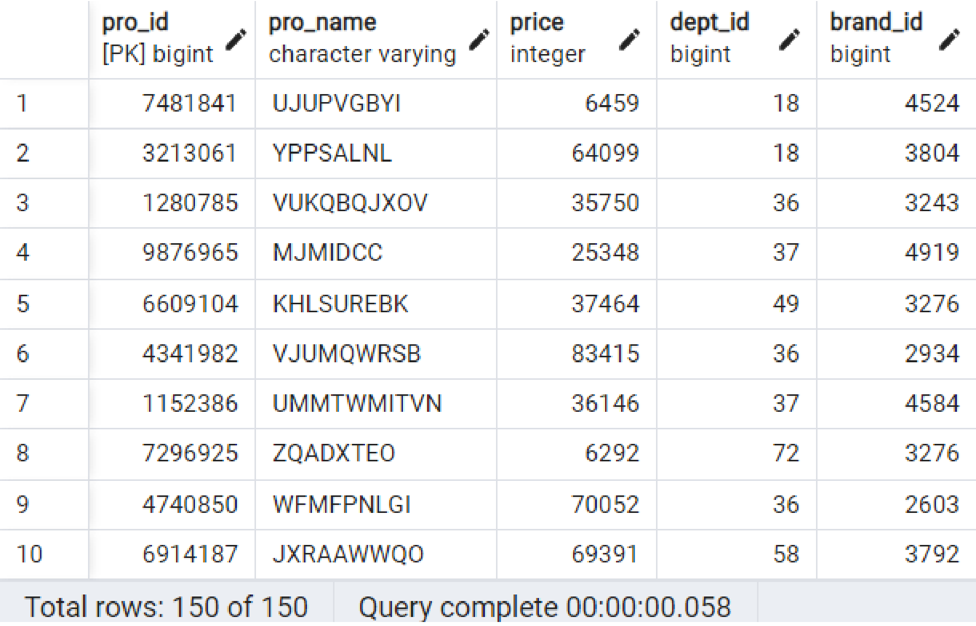
	sr_id [PK] bigint	dept_id [PK] bigint
1	22619	37
2	24172	37
3	47755	37
4	47755	18
5	47576	58
6	85682	58
7	73039	49
8	98583	36
9	78639	58
10	78639	72

Total rows: 56 of 56 Query c

12. Product:



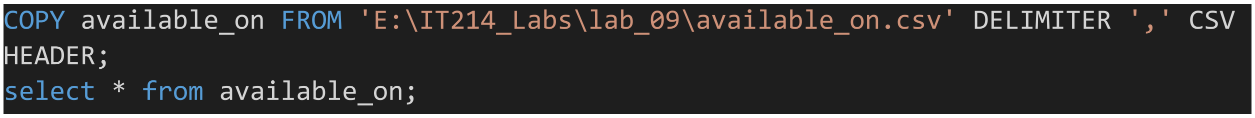
```
COPY product FROM 'E:\IT214_Labs\lab_09\product.csv' DELIMITER ',' CSV HEADER;
select * from product;
```



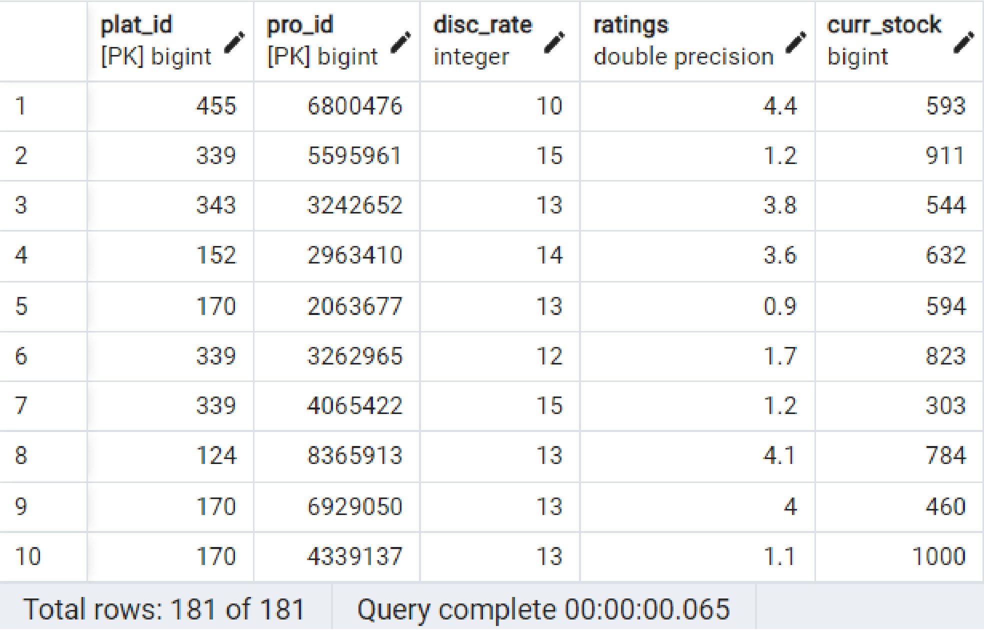
	pro_id [PK] bigint	pro_name character varying	price integer	dept_id bigint	brand_id bigint
1	7481841	UJUPVGBYI	6459	18	4524
2	3213061	YPPSALNL	64099	18	3804
3	1280785	VUKQBQJXOV	35750	36	3243
4	9876965	MJMIDCC	25348	37	4919
5	6609104	KHLSUREBK	37464	49	3276
6	4341982	VJUMQWRSB	83415	36	2934
7	1152386	UMMTWMITVN	36146	37	4584
8	7296925	ZQADXTEO	6292	72	3276
9	4740850	WFMFPNLGI	70052	36	2603
10	6914187	JXRAAWWQO	69391	58	3792

Total rows: 150 of 150 Query complete 00:00:00.058

13. Available_on:



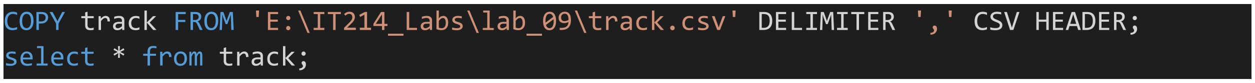
```
COPY available_on FROM 'E:\IT214_Labs\lab_09\available_on.csv' DELIMITER ',' CSV
HEADER;
select * from available_on;
```



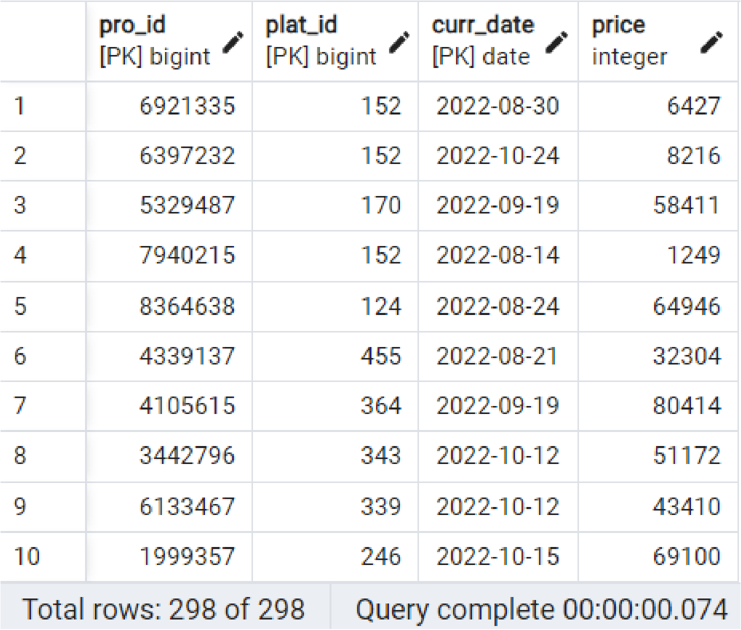
	plat_id [PK] bigint	pro_id [PK] bigint	disc_rate integer	ratings double precision	curr_stock bigint
1	455	6800476	10	4.4	593
2	339	5595961	15	1.2	911
3	343	3242652	13	3.8	544
4	152	2963410	14	3.6	632
5	170	2063677	13	0.9	594
6	339	3262965	12	1.7	823
7	339	4065422	15	1.2	303
8	124	8365913	13	4.1	784
9	170	6929050	13	4	460
10	170	4339137	13	1.1	1000

Total rows: 181 of 181 Query complete 00:00:00.065

14. Track:



```
COPY track FROM 'E:\IT214_Labs\lab_09\track.csv' DELIMITER ',' CSV HEADER;
select * from track;
```

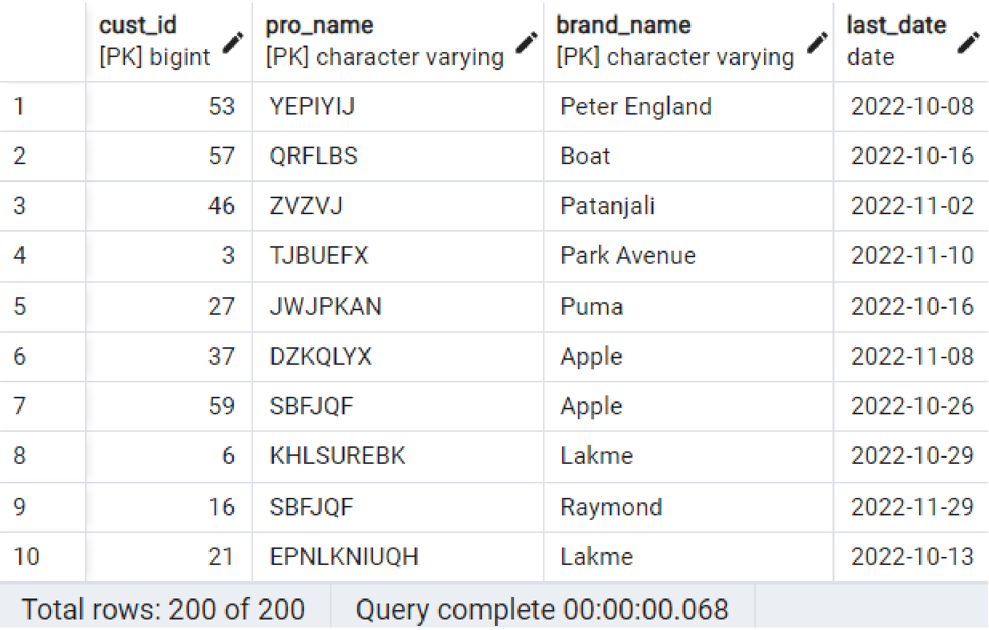


	pro_id [PK] bigint	plat_id [PK] bigint	curr_date [PK] date	price integer
1	6921335	152	2022-08-30	6427
2	6397232	152	2022-10-24	8216
3	5329487	170	2022-09-19	58411
4	7940215	152	2022-08-14	1249
5	8364638	124	2022-08-24	64946
6	4339137	455	2022-08-21	32304
7	4105615	364	2022-09-19	80414
8	3442796	343	2022-10-12	51172
9	6133467	339	2022-10-12	43410
10	1999357	246	2022-10-15	69100

Total rows: 298 of 298 Query complete 00:00:00.074

15. Pro_hist:

```
COPY pro_hist FROM 'E:\IT214_Labs\lab_09\pro_hist.csv' DELIMITER ',' CSV HEADER;
select * from pro_hist;
```

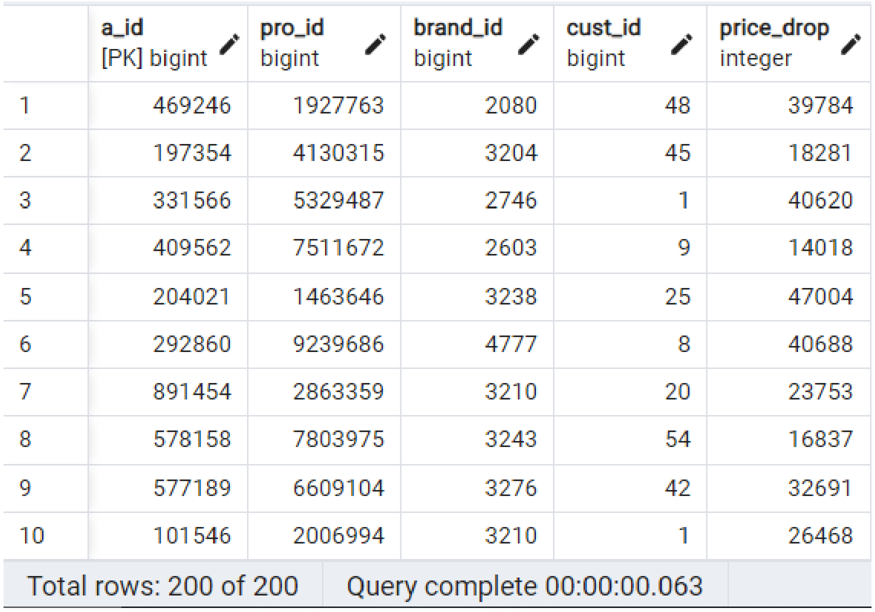


	cust_id [PK] bigint	pro_name [PK] character varying	brand_name [PK] character varying	last_date date
1	53	YEPIYIJ	Peter England	2022-10-08
2	57	QRFLBS	Boat	2022-10-16
3	46	ZVZVJ	Patanjali	2022-11-02
4	3	TJBUEFX	Park Avenue	2022-11-10
5	27	JWJPKAN	Puma	2022-10-16
6	37	DZKQLYX	Apple	2022-11-08
7	59	SBFJQF	Apple	2022-10-26
8	6	KHLSUREBK	Lakme	2022-10-29
9	16	SBFJQF	Raymond	2022-11-29
10	21	EPNLKNIUQH	Lakme	2022-10-13

Total rows: 200 of 200 Query complete 00:00:00.068

16. Price_alert:

```
COPY price_alert FROM 'E:\IT214_Labs\lab_09\price_alert.csv' DELIMITER ',' CSV
HEADER;
select * from price_alert;
```



	a_id [PK] bigint	pro_id bigint	brand_id bigint	cust_id bigint	price_drop integer
1	469246	1927763	2080	48	39784
2	197354	4130315	3204	45	18281
3	331566	5329487	2746	1	40620
4	409562	7511672	2603	9	14018
5	204021	1463646	3238	25	47004
6	292860	9239686	4777	8	40688
7	891454	2863359	3210	20	23753
8	578158	7803975	3243	54	16837
9	577189	6609104	3276	42	32691
10	101546	2006994	3210	1	26468

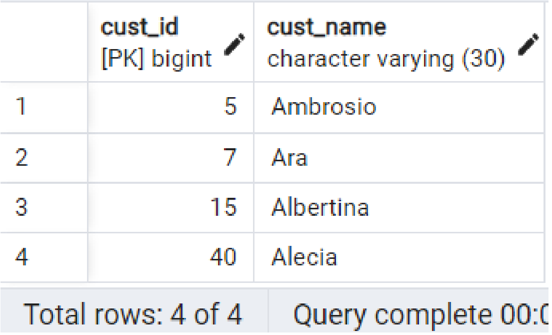
Total rows: 200 of 200 Query complete 00:00:00.063

Section 7: 20 SQL Queries

7.1 Queries:

1. **Select all customers' names and customer IDs whose names begin with the letter 'A'.**

```sql
select cust_id, cust_name
from customer
where cust_name like 'A%'
```

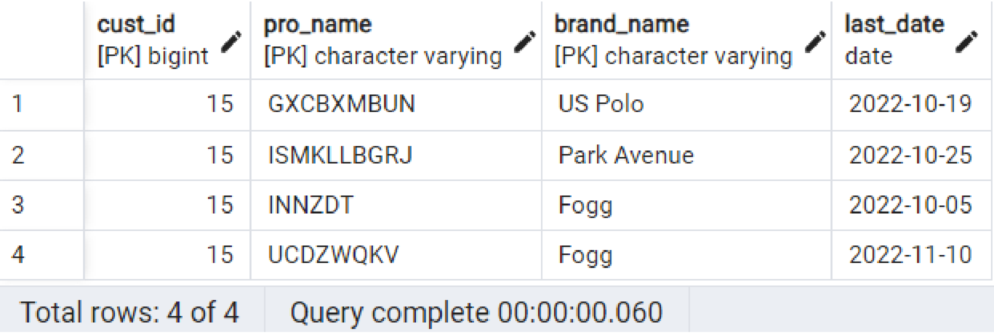


	cust_id [PK] bigint	cust_name character varying (30)
1	5	Ambrosio
2	7	Ara
3	15	Albertina
4	40	Alecia

Total rows: 4 of 4 Query complete 00:(

2. **Get the search history of the customer whose id = 15.**

```sql
select *
from pro_hist
where cust_id = 15
```

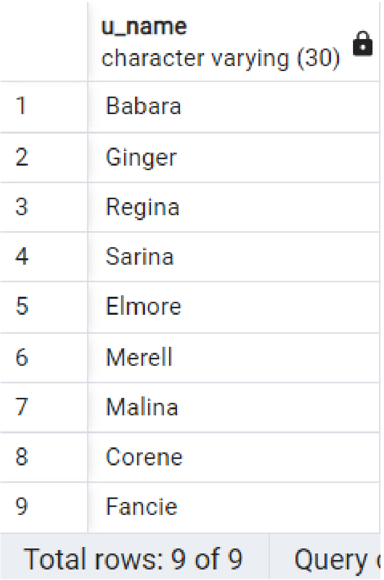


	cust_id [PK] bigint	pro_name [PK] character varying	brand_name [PK] character varying	last_date date
1	15	GXCBXMBUN	US Polo	2022-10-19
2	15	ISMKLLBGRJ	Park Avenue	2022-10-25
3	15	INNZDT	Fogg	2022-10-05
4	15	UCDZWQKV	Fogg	2022-11-10

Total rows: 4 of 4 Query complete 00:00:00.060

3. **Display all employee names whose names consist of exactly 6 characters.**

```sql
select u_name
from user_class
where u_name like '______'
```

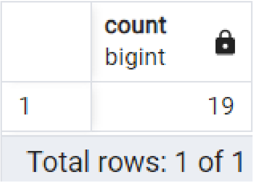


	u_name character varying (30)
1	Babara
2	Ginger
3	Regina
4	Sarina
5	Elmore
6	Merell
7	Malina
8	Corene
9	Fancie

Total rows: 9 of 9 Query

4. **Display the number of products in the electronics department (dept_id = 72).**

```
select count(pro_id)
from product
where dept_id = 72
```

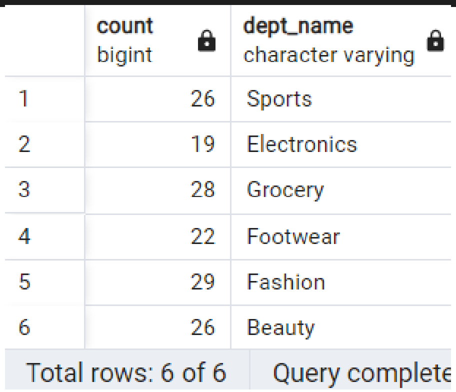


	count bigint
1	19

Total rows: 1 of 1

5. **Display the count of products in different departments.**

```
select count(pro_id), department.dept_name
from product
INNER JOIN department ON product.dept_id = department.dept_id
group by department.dept_name
```

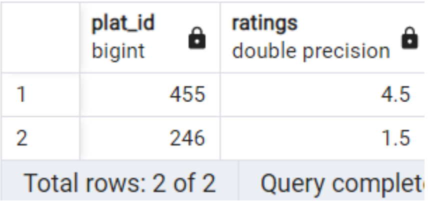


	count bigint	dept_name character varying
1	26	Sports
2	19	Electronics
3	28	Grocery
4	22	Footwear
5	29	Fashion
6	26	Beauty

Total rows: 6 of 6 Query complete

6. **Display ratings of a given product on different platforms.**

```
select plat_id, ratings
from available_on
where pro_id = 1829523
```

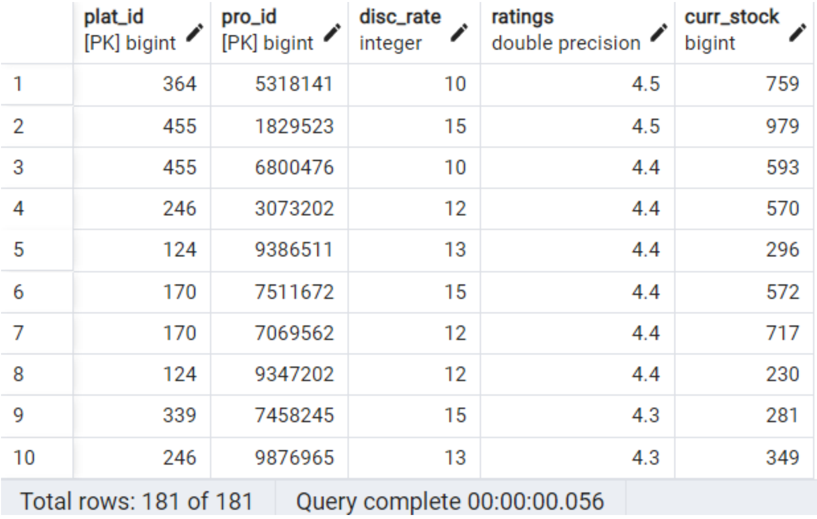


	plat_id bigint	ratings double precision
1	455	4.5
2	246	1.5

Total rows: 2 of 2 Query complet

7. **Display all products sorted in higher to low ratings.**

```
select *
from available_on
order by ratings desc
```

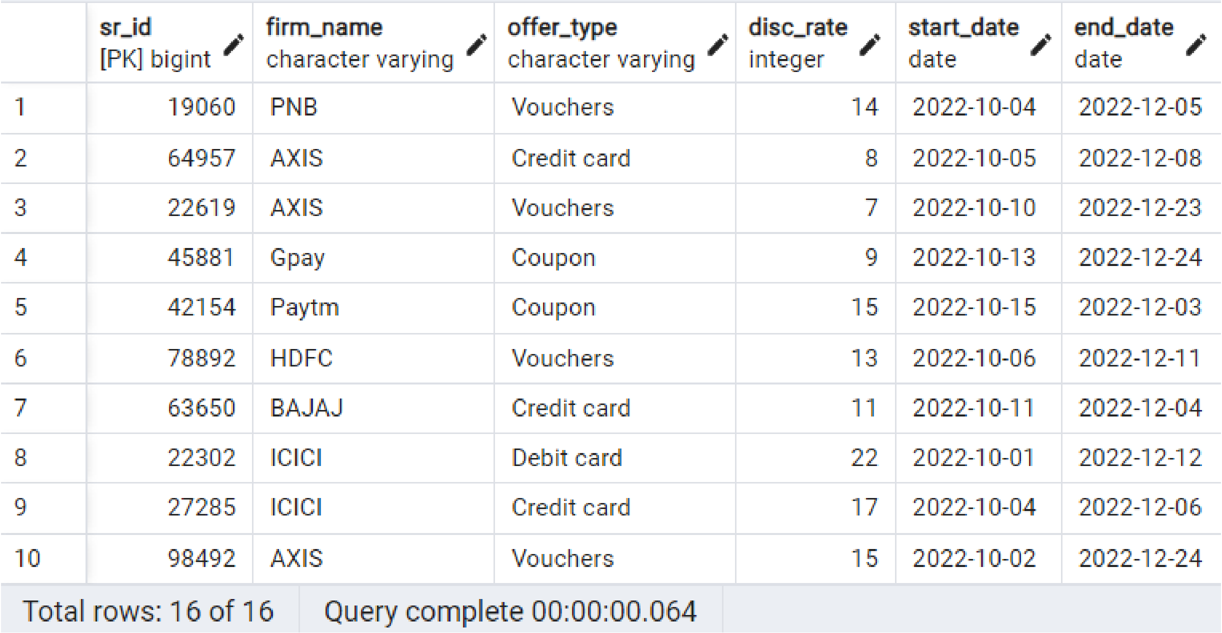


	plat_id [PK] bigint	pro_id [PK] bigint	disc_rate integer	ratings double precision	curr_stock bigint
1	364	5318141	10	4.5	759
2	455	1829523	15	4.5	979
3	455	6800476	10	4.4	593
4	246	3073202	12	4.4	570
5	124	9386511	13	4.4	296
6	170	7511672	15	4.4	572
7	170	7069562	12	4.4	717
8	124	9347202	12	4.4	230
9	339	7458245	15	4.3	281
10	246	9876965	13	4.3	349

Total rows: 181 of 181 Query complete 00:00:00.056

8. **Display all offers between the 15th of October and the 15th of November.**

```
select *
from offer
where start_date<='2022-10-15' and end_date>='2022-11-15'
```

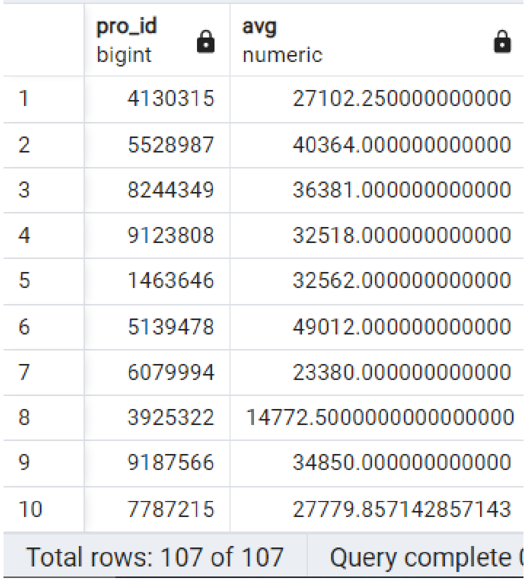


	sr_id [PK] bigint	firm_name character varying	offer_type character varying	disc_rate integer	start_date date	end_date date
1	19060	PNB	Vouchers	14	2022-10-04	2022-12-05
2	64957	AXIS	Credit card	8	2022-10-05	2022-12-08
3	22619	AXIS	Vouchers	7	2022-10-10	2022-12-23
4	45881	Gpay	Coupon	9	2022-10-13	2022-12-24
5	42154	Paytm	Coupon	15	2022-10-15	2022-12-03
6	78892	HDFC	Vouchers	13	2022-10-06	2022-12-11
7	63650	BAJAJ	Credit card	11	2022-10-11	2022-12-04
8	22302	ICICI	Debit card	22	2022-10-01	2022-12-12
9	27285	ICICI	Credit card	17	2022-10-04	2022-12-06
10	98492	AXIS	Vouchers	15	2022-10-02	2022-12-24

Total rows: 16 of 16 Query complete 00:00:00.064

9. **Display the average price drop expected by customers for every product.**

```
select pro_id, avg(price_drop)
from price_alert
group by pro_id
```

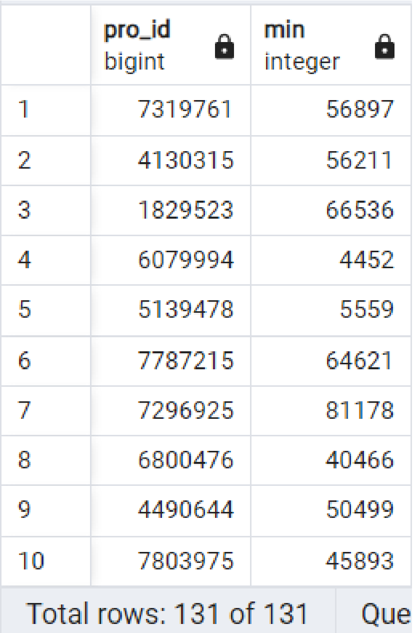


	pro_id bigint	avg numeric
1	4130315	27102.250000000000
2	5528987	40364.000000000000
3	8244349	36381.000000000000
4	9123808	32518.000000000000
5	1463646	32562.000000000000
6	5139478	49012.000000000000
7	6079994	23380.000000000000
8	3925322	14772.5000000000000000
9	9187566	34850.000000000000
10	7787215	27779.857142857143

Total rows: 107 of 107 Query complete (

10. **Display the minimum price of all products in the track history.**

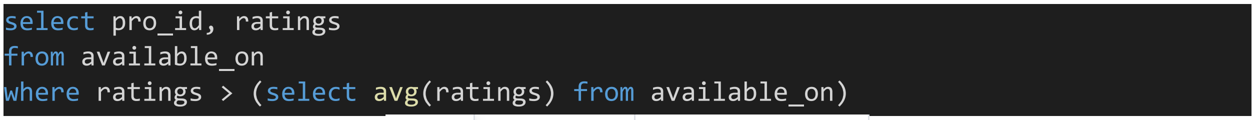
```
select pro_id, min(price)
from track
group by pro_id
```



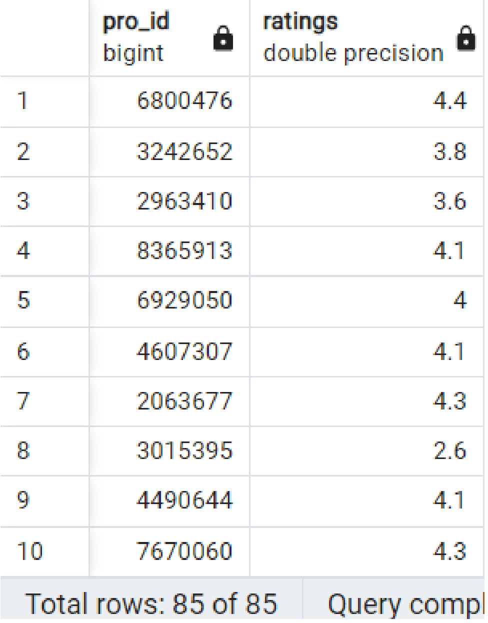
	pro_id bigint	min integer
1	7319761	56897
2	4130315	56211
3	1829523	66536
4	6079994	4452
5	5139478	5559
6	7787215	64621
7	7296925	81178
8	6800476	40466
9	4490644	50499
10	7803975	45893

Total rows: 131 of 131 Que

11. Display all the products whose ratings are higher than average rating.



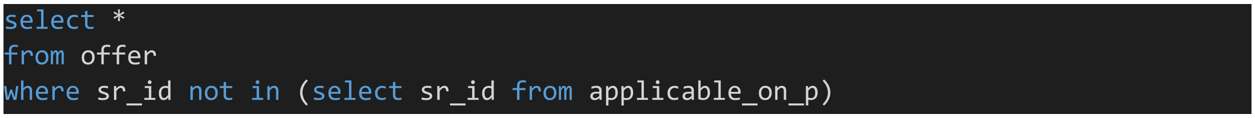
```
select pro_id, ratings
from available_on
where ratings > (select avg(ratings) from available_on)
```



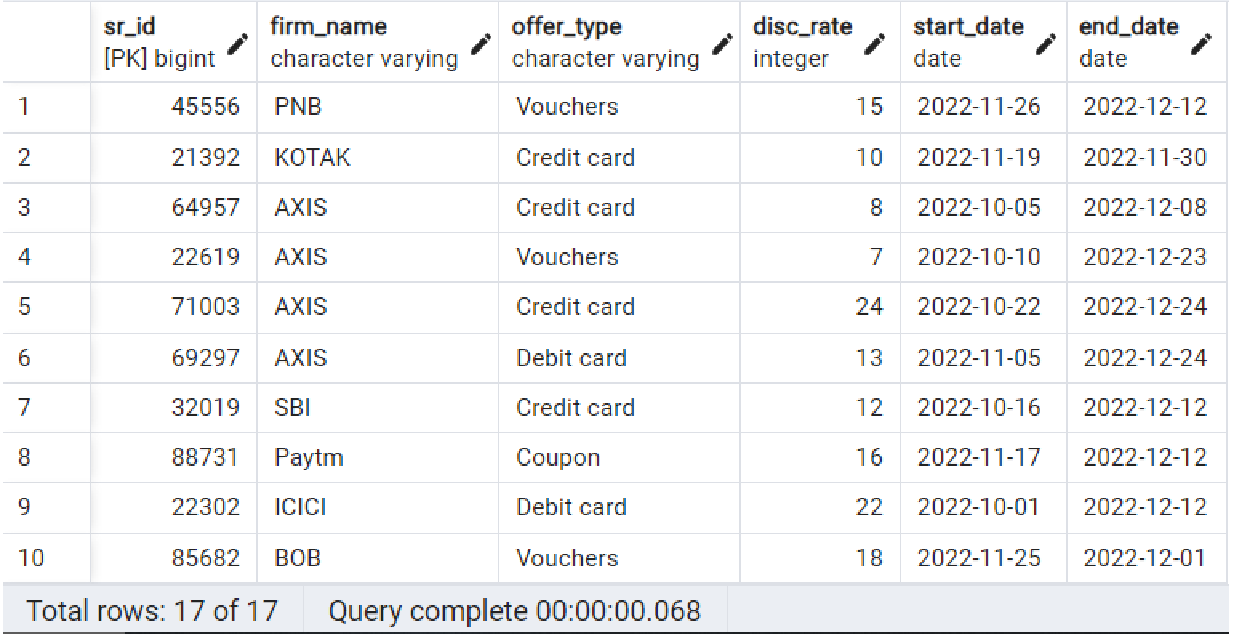
	pro_id bigint	ratings double precision
1	6800476	4.4
2	3242652	3.8
3	2963410	3.6
4	8365913	4.1
5	6929050	4
6	4607307	4.1
7	2063677	4.3
8	3015395	2.6
9	4490644	4.1
10	7670060	4.3

Total rows: 85 of 85 Query compl

12. Display all offers that are not on any platform.



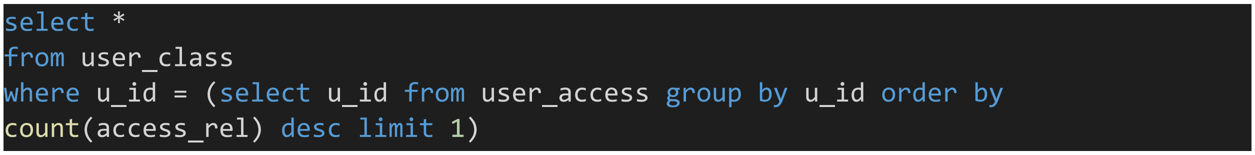
```
select *
from offer
where sr_id not in (select sr_id from applicable_on_p)
```



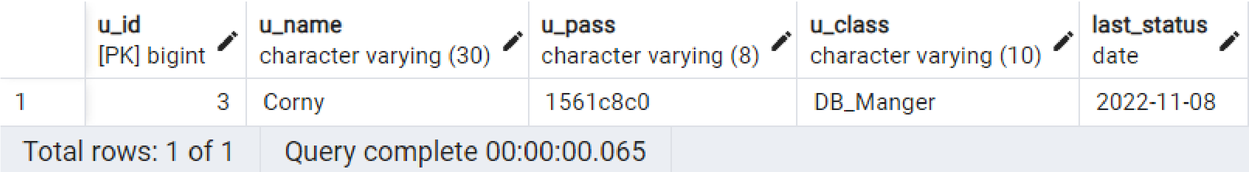
	sr_id [PK] bigint	firm_name character varying	offer_type character varying	disc_rate integer	start_date date	end_date date
1	45556	PNB	Vouchers	15	2022-11-26	2022-12-12
2	21392	KOTAK	Credit card	10	2022-11-19	2022-11-30
3	64957	AXIS	Credit card	8	2022-10-05	2022-12-08
4	22619	AXIS	Vouchers	7	2022-10-10	2022-12-23
5	71003	AXIS	Credit card	24	2022-10-22	2022-12-24
6	69297	AXIS	Debit card	13	2022-11-05	2022-12-24
7	32019	SBI	Credit card	12	2022-10-16	2022-12-12
8	88731	Paytm	Coupon	16	2022-11-17	2022-12-12
9	22302	ICICI	Debit card	22	2022-10-01	2022-12-12
10	85682	BOB	Vouchers	18	2022-11-25	2022-12-01

Total rows: 17 of 17 Query complete 00:00:00.068

13. Display employee detail with most of the accesses.



```
select *
from user_class
where u_id = (select u_id from user_access group by u_id order by
count(access_rel) desc limit 1)
```

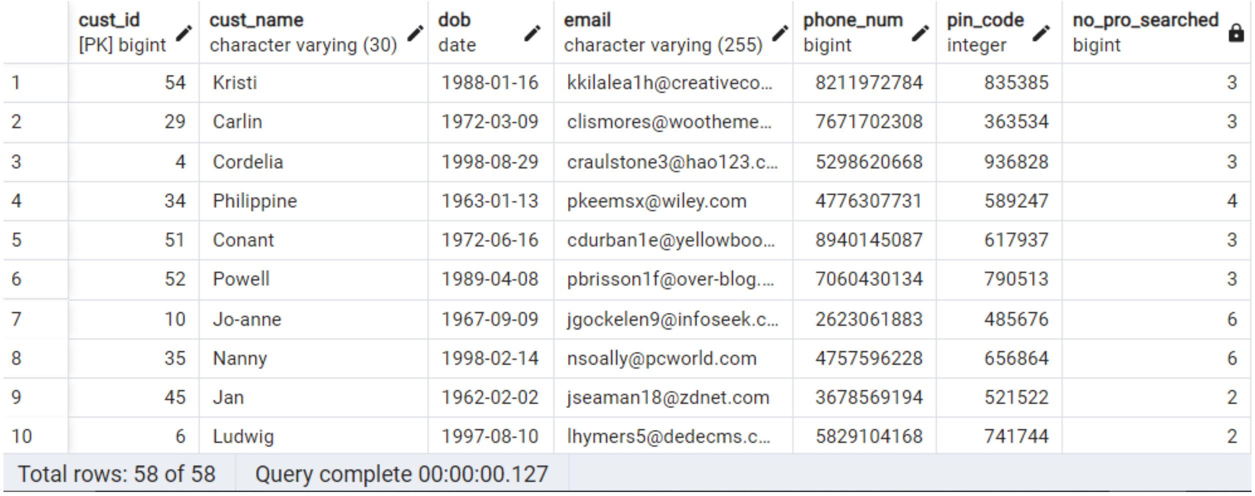


	u_id [PK] bigint	u_name character varying (30)	u_pass character varying (8)	u_class character varying (10)	last_status date
1	3	Corny	1561c8c0	DB_Manger	2022-11-08

Total rows: 1 of 1 Query complete 00:00:00.065

14. Display the detail of customers along with the number of distinct product searched by them.

```
select tmp.cust_id, tmp.cust_name, tmp.dob, tmp.email, tmp.phone_num,
tmp.pin_code, count(tmp.pro_name) as No_pro_searched
from (select * from customer natural join pro_hist where pro_hist.cust_id =
customer.cust_id) as tmp
group by tmp.cust_id, tmp.cust_name, tmp.dob, tmp.email, tmp.phone_num,
tmp.pin_code
```

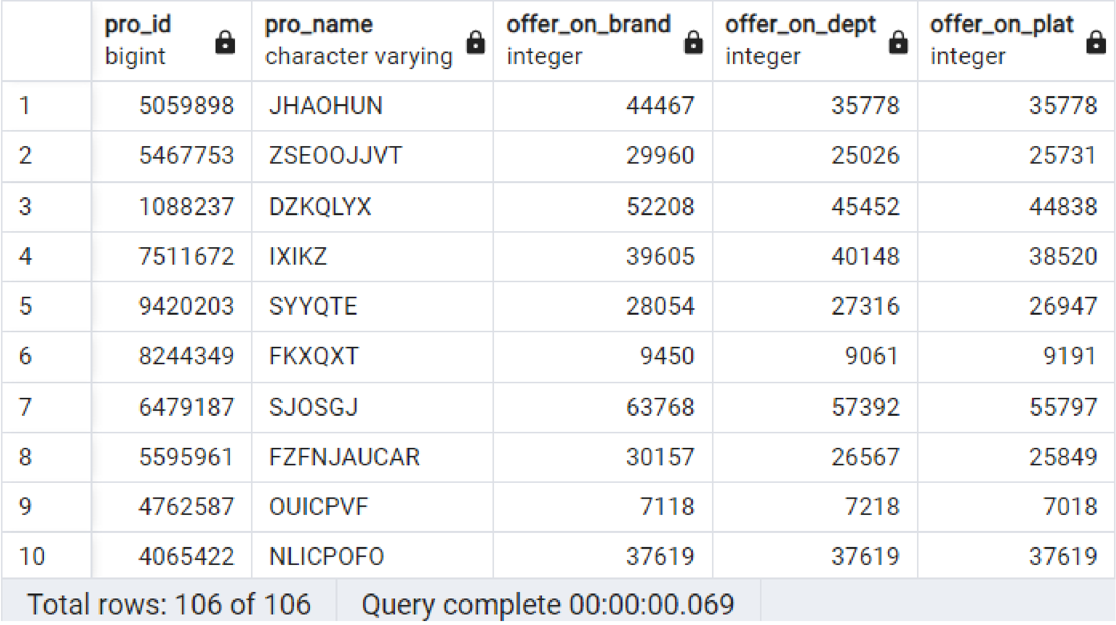


	cust_id [PK] bigint	cust_name character varying (30)	dob date	email character varying (255)	phone_num bigint	pin_code integer	no_pro_searched bigint
1	54	Kristi	1988-01-16	kkilalea1h@creativeco...	8211972784	835385	3
2	29	Carlin	1972-03-09	clismores@wootheme...	7671702308	363534	3
3	4	Cordelia	1998-08-29	craulstone3@hao123.c...	5298620668	936828	3
4	34	Philippine	1963-01-13	pkeemsx@wiley.com	4776307731	589247	4
5	51	Conant	1972-06-16	cdurban1e@yellowboo...	8940145087	617937	3
6	52	Powell	1989-04-08	pbrisson1f@over-blog....	7060430134	790513	3
7	10	Jo-anne	1967-09-09	jgockelen9@infoseek.c...	2623061883	485676	6
8	35	Nanny	1998-02-14	nsoally@pcworld.com	4757596228	656864	6
9	45	Jan	1962-02-02	jseaman18@zdnet.com	3678569194	521522	2
10	6	Ludwig	1997-08-10	lhymers5@dedecms.c...	5829104168	741744	2

Total rows: 58 of 58 Query complete 00:00:00.127

15. Display the minimum price for a product after applying the offer applicable to its brand, department, and platform.

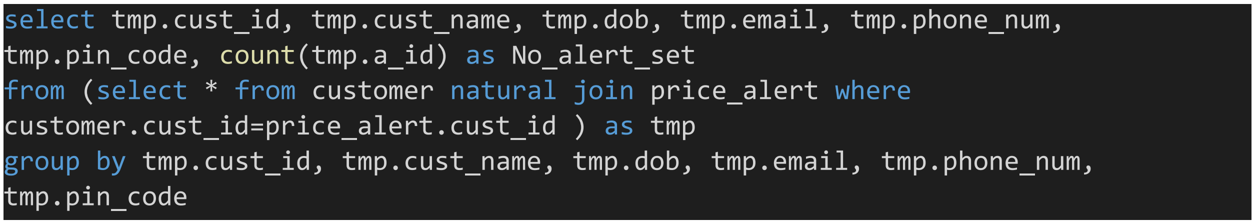
```
select tmp.pro_id, tmp.pro_name, min(tmp.disc_of_brand) as offer_on_brand,
min(tmp.disc_of_dept) as offer_on_dept, min(tmp.disc_of_plat) as offer_on_plat
from (select r1.pro_id, r1.pro_name,
      r1.brand_id, r2.sr_id as offer_on_brand,
r1.best_deal-r1.best_deal*r2.disc_rate/100 as disc_of_brand,
      r1.dept_id, r3.sr_id as offer_on_dept,
r1.best_deal-r1.best_deal*r3.disc_rate/100 as disc_of_dept,
      r1.plat_id, r4.sr_id as offer_on_plat,
r1.best_deal-r1.best_deal*r4.disc_rate/100 as disc_of_plat, r1.ratings
      from base_price as r1, applicable_on_brand as r2, applicable_on_dept as r3,
applicable_on_plat as r4
      where r1.brand_id = r2.brand_id and r1.dept_id = r3.dept_id and r1.plat_id
= r4.plat_id) as tmp
group by tmp.pro_id, tmp.pro_name
```



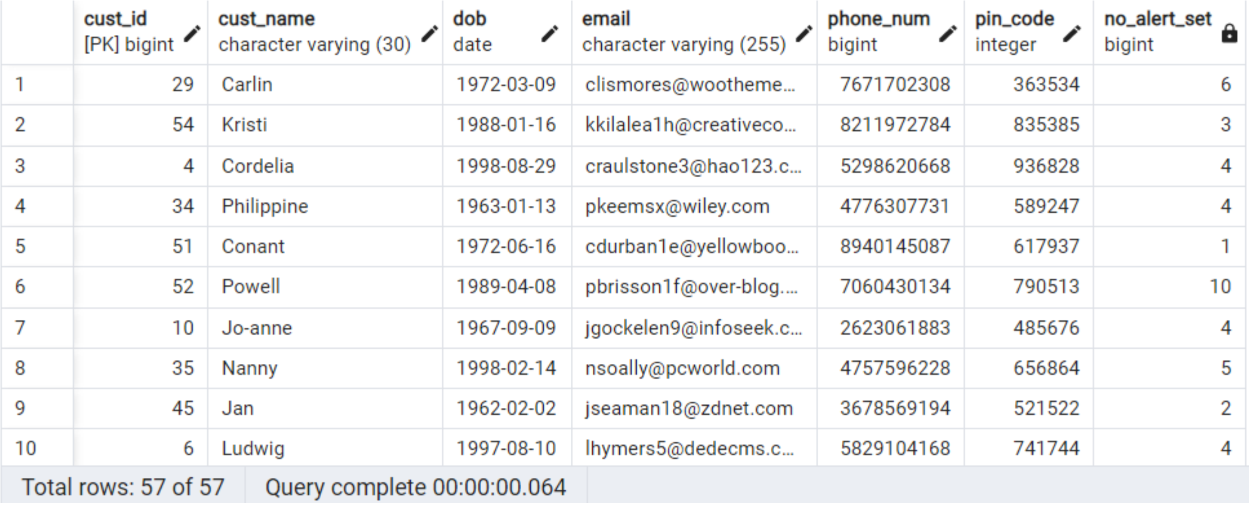
	pro_id bigint	pro_name character varying	offer_on_brand integer	offer_on_dept integer	offer_on_plat integer
1	5059898	JHAOHUN	44467	35778	35778
2	5467753	ZSEOOJJVT	29960	25026	25731
3	1088237	DZKQLYX	52208	45452	44838
4	7511672	IXIKZ	39605	40148	38520
5	9420203	SYYQTE	28054	27316	26947
6	8244349	FKXQXT	9450	9061	9191
7	6479187	SJOSGJ	63768	57392	55797
8	5595961	FZFNJAUCAR	30157	26567	25849
9	4762587	OUICPVF	7118	7218	7018
10	4065422	NLICPOFO	37619	37619	37619

Total rows: 106 of 106 Query complete 00:00:00.069

16. Display the detail of customers along with the number of alerts set by them.



```
select tmp.cust_id, tmp.cust_name, tmp.dob, tmp.email, tmp.phone_num,
tmp.pin_code, count(tmp.a_id) as No_alert_set
from (select * from customer natural join price_alert where
customer.cust_id=price_alert.cust_id ) as tmp
group by tmp.cust_id, tmp.cust_name, tmp.dob, tmp.email, tmp.phone_num,
tmp.pin_code
```

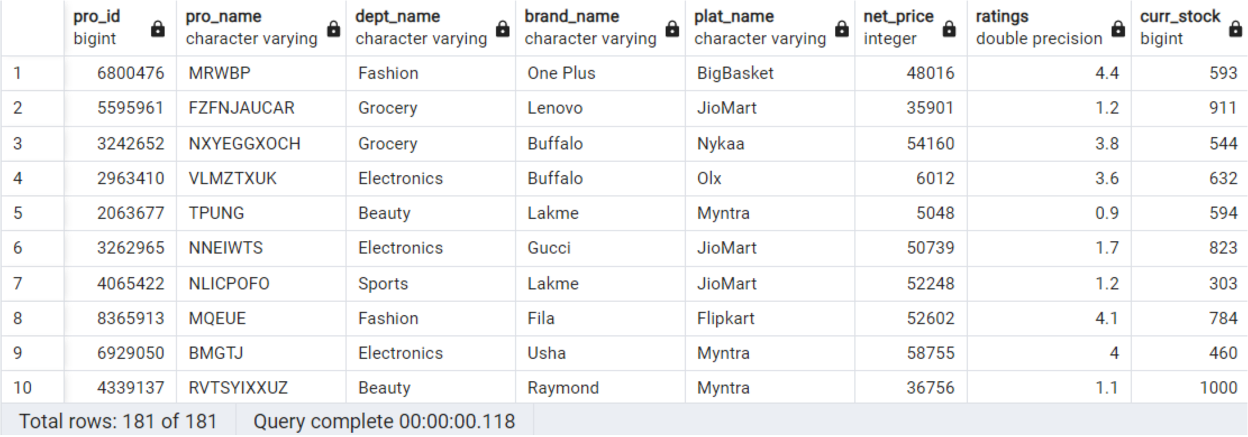


	cust_id [PK] bigint	cust_name character varying (30)	dob date	email character varying (255)	phone_num bigint	pin_code integer	no_alert_set bigint
1	29	Carlin	1972-03-09	clismores@wootheme...	7671702308	363534	6
2	54	Kristi	1988-01-16	kkilalea1h@creativeco...	8211972784	835385	3
3	4	Cordelia	1998-08-29	craulstone3@hao123.c...	5298620668	936828	4
4	34	Philippine	1963-01-13	pkeemsx@wiley.com	4776307731	589247	4
5	51	Conant	1972-06-16	cdurban1e@yellowboo...	8940145087	617937	1
6	52	Powell	1989-04-08	pbrisson1f@over-blog....	7060430134	790513	10
7	10	Jo-anne	1967-09-09	jgockelen9@infoseek.c...	2623061883	485676	4
8	35	Nanny	1998-02-14	nsoally@pcworld.com	4757596228	656864	5
9	45	Jan	1962-02-02	jseaman18@zdnet.com	3678569194	521522	2
10	6	Ludwig	1997-08-10	lhymers5@dedecms.c...	5829104168	741744	4

Total rows: 57 of 57 Query complete 00:00:00.064

17. Display all product details.

```
select r1.pro_id, r1.pro_name, r3.dept_name, r2.brand_name, r4.plat_name,
r1.best_deal as net_price, r1.ratings, r1.curr_stock
from base_price as r1, brand as r2, department as r3, platform as r4
where r1.dept_id = r3.dept_id and r1.brand_id = r2.brand_id and r1.plat_id =
r4.plat_id
```

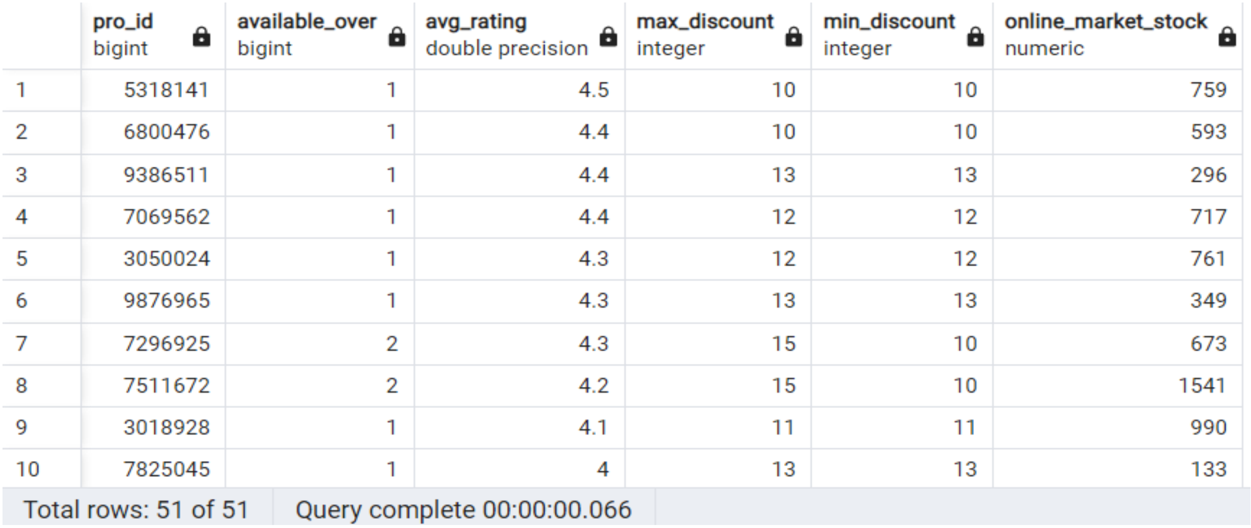


	pro_id bigint	pro_name character varying	dept_name character varying	brand_name character varying	plat_name character varying	net_price integer	ratings double precision	curr_stock bigint
1	6800476	MRWBP	Fashion	One Plus	BigBasket	48016	4.4	593
2	5595961	FZFNJAUCAR	Grocery	Lenovo	JioMart	35901	1.2	911
3	3242652	NXYEGGXOCH	Grocery	Buffalo	Nykaa	54160	3.8	544
4	2963410	VLMZTXUK	Electronics	Buffalo	Olx	6012	3.6	632
5	2063677	TPUNG	Beauty	Lakme	Myntra	5048	0.9	594
6	3262965	NNEIWTS	Electronics	Gucci	JioMart	50739	1.7	823
7	4065422	NLICPOFO	Sports	Lakme	JioMart	52248	1.2	303
8	8365913	MQEUE	Fashion	Fila	Flipkart	52602	4.1	784
9	6929050	BMGTJ	Electronics	Usha	Myntra	58755	4	460
10	4339137	RVTSYIXXUZ	Beauty	Raymond	Myntra	36756	1.1	1000

Total rows: 181 of 181 Query complete 00:00:00.118

18. display all the products and their availability in terms of current stock and on a number of platforms, it is available along with max and min discounts provided by platforms having ratings more than or equal to 2.5 and arranged in decreasing order.

```
select pro_id, count(plat_id) as available_over, avg(ratings) as avg_rating,
max(disc_rate) as max_discount, min(disc_rate) as min_discount, sum(curr_stock)
as online_market_stock
from available_on
group by pro_id
having avg(ratings) >= 2.5
order by avg(ratings) desc
```

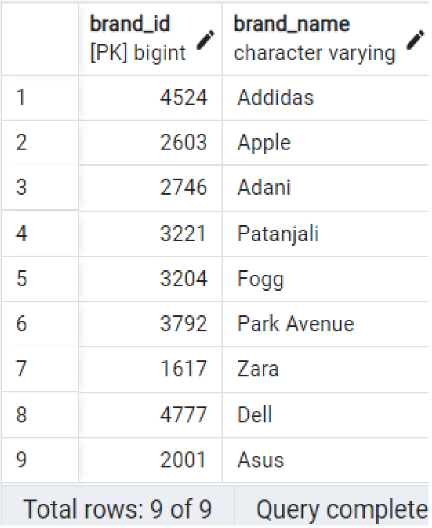


	pro_id bigint	available_over bigint	avg_rating double precision	max_discount integer	min_discount integer	online_market_stock numeric
1	5318141	1	4.5	10	10	759
2	6800476	1	4.4	10	10	593
3	9386511	1	4.4	13	13	296
4	7069562	1	4.4	12	12	717
5	3050024	1	4.3	12	12	761
6	9876965	1	4.3	13	13	349
7	7296925	2	4.3	15	10	673
8	7511672	2	4.2	15	10	1541
9	3018928	1	4.1	11	11	990
10	7825045	1	4	13	13	133

Total rows: 51 of 51 Query complete 00:00:00.066

19. Display all the brands searched by the customer with the most search history.

```
select brand_id, brand_name
from (select pro_hist.cust_id, brand.brand_id, brand.brand_name
      from pro_hist natural join brand
      where pro_hist.brand_name = brand.brand_name) as tmp
where tmp.cust_id = (select cust_id from pro_hist group by cust_id order by
count(last_date) desc limit 1)
```

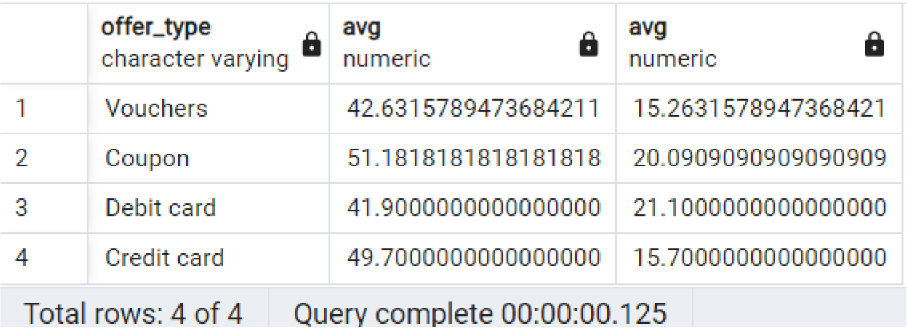


	brand_id [PK] bigint	brand_name character varying
1	4524	Addidas
2	2603	Apple
3	2746	Adani
4	3221	Patanjali
5	3204	Fogg
6	3792	Park Avenue
7	1617	Zara
8	4777	Dell
9	2001	Asus

Total rows: 9 of 9 Query complete

20. Display the average duration and discount rate of each offer type.

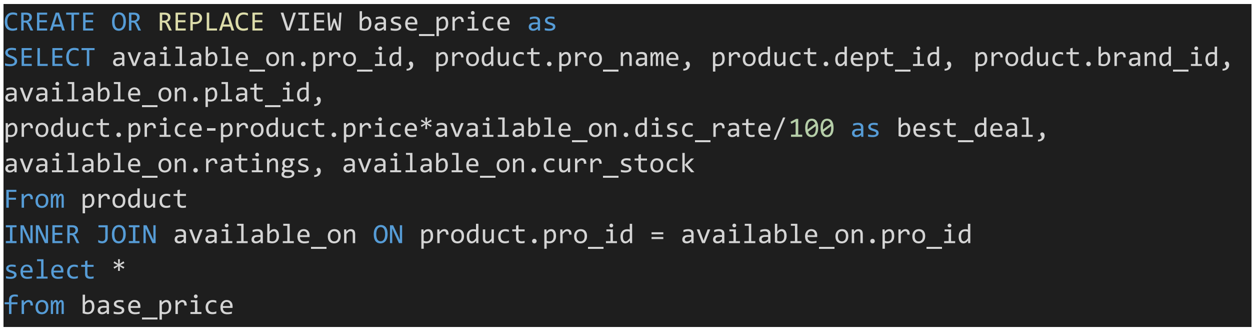
```
select offer_type, avg(end_date-start_date), avg(disc_rate)
from offer
group by offer_type
```



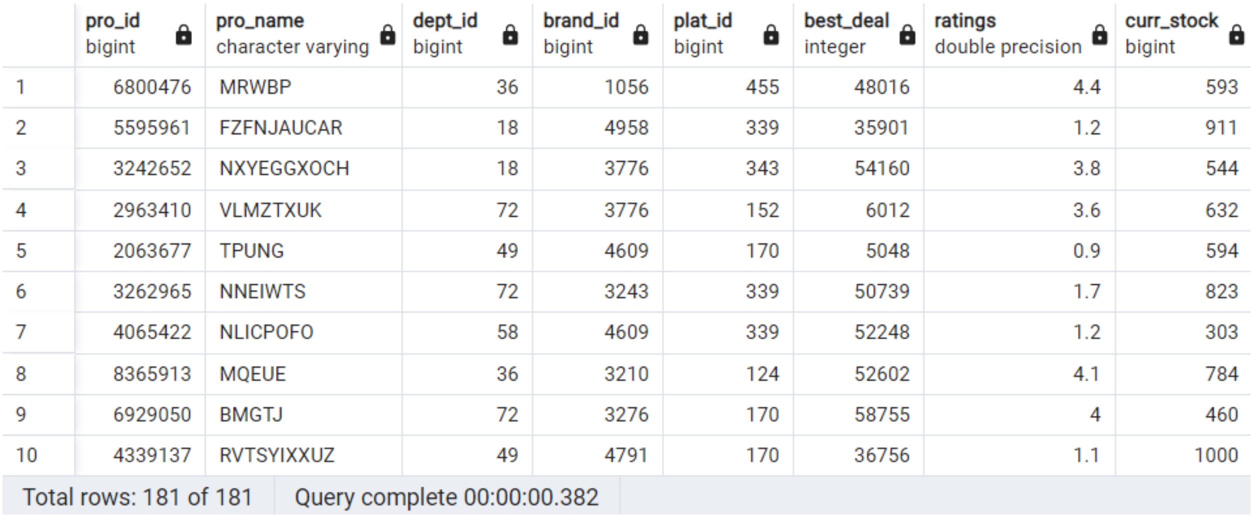
	offer_type character varying	avg numeric	avg numeric
1	Vouchers	42.6315789473684211	15.2631578947368421
2	Coupon	51.1818181818181818	20.0909090909090909
3	Debit card	41.9000000000000000	21.1000000000000000
4	Credit card	49.7000000000000000	15.7000000000000000

Total rows: 4 of 4 Query complete 00:00:00.125

7.2 Views used in the above queries are listed below:

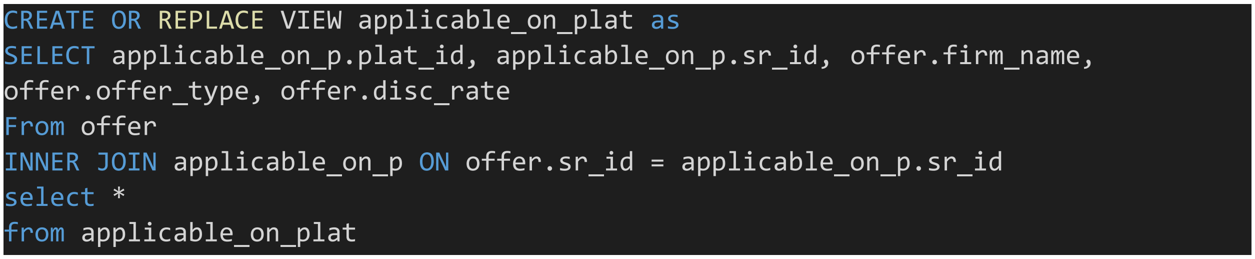


```
CREATE OR REPLACE VIEW base_price as
SELECT available_on.pro_id, product.pro_name, product.dept_id, product.brand_id,
available_on.plat_id,
product.price-product.price*available_on.disc_rate/100 as best_deal,
available_on.ratings, available_on.curr_stock
From product
INNER JOIN available_on ON product.pro_id = available_on.pro_id
select *
from base_price
```

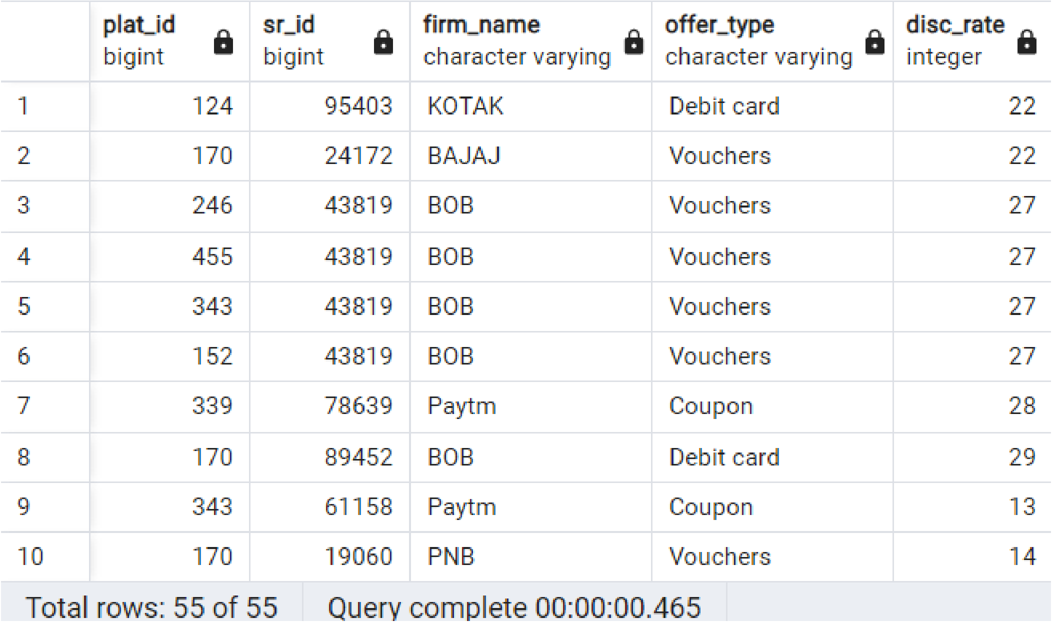


	pro_id bigint	pro_name character varying	dept_id bigint	brand_id bigint	plat_id bigint	best_deal integer	ratings double precision	curr_stock bigint
1	6800476	MRWBP	36	1056	455	48016	4.4	593
2	5595961	FZFNJAUCAR	18	4958	339	35901	1.2	911
3	3242652	NXYEGGXOCH	18	3776	343	54160	3.8	544
4	2963410	VLMZTXUK	72	3776	152	6012	3.6	632
5	2063677	TPUNG	49	4609	170	5048	0.9	594
6	3262965	NNEIWTS	72	3243	339	50739	1.7	823
7	4065422	NLICPOFO	58	4609	339	52248	1.2	303
8	8365913	MQEUE	36	3210	124	52602	4.1	784
9	6929050	BMGTJ	72	3276	170	58755	4	460
10	4339137	RVTSYIXXUZ	49	4791	170	36756	1.1	1000

Total rows: 181 of 181 Query complete 00:00:00.382

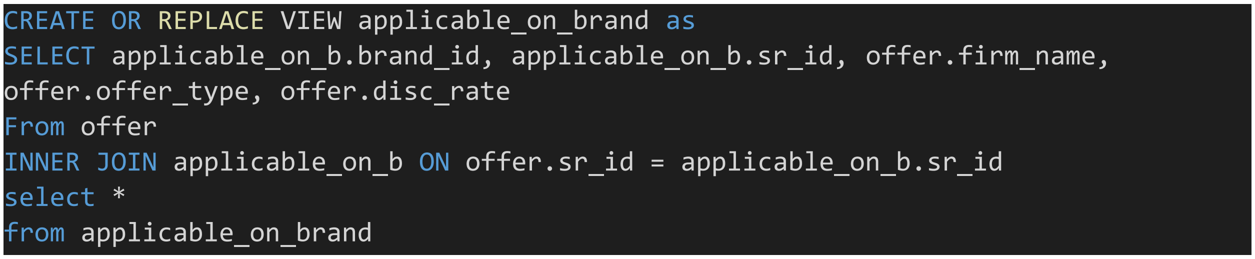


```
CREATE OR REPLACE VIEW applicable_on_plat as
SELECT applicable_on_p.plat_id, applicable_on_p.sr_id, offer.firm_name,
offer.offer_type, offer.disc_rate
From offer
INNER JOIN applicable_on_p ON offer.sr_id = applicable_on_p.sr_id
select *
from applicable_on_plat
```

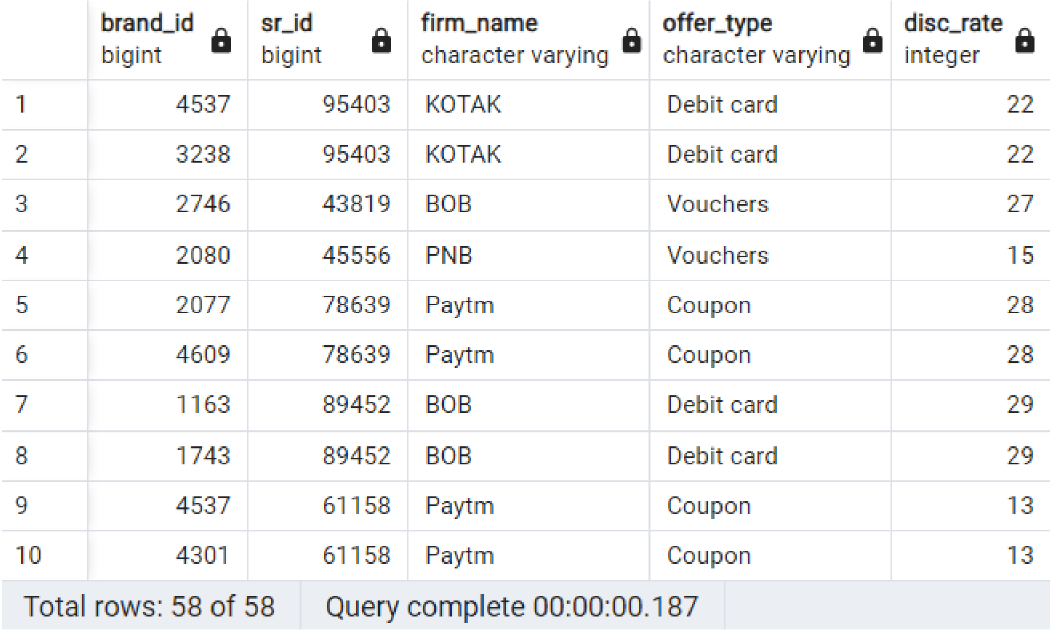


	plat_id bigint	sr_id bigint	firm_name character varying	offer_type character varying	disc_rate integer
1	124	95403	KOTAK	Debit card	22
2	170	24172	BAJAJ	Vouchers	22
3	246	43819	BOB	Vouchers	27
4	455	43819	BOB	Vouchers	27
5	343	43819	BOB	Vouchers	27
6	152	43819	BOB	Vouchers	27
7	339	78639	Paytm	Coupon	28
8	170	89452	BOB	Debit card	29
9	343	61158	Paytm	Coupon	13
10	170	19060	PNB	Vouchers	14

Total rows: 55 of 55 Query complete 00:00:00.465

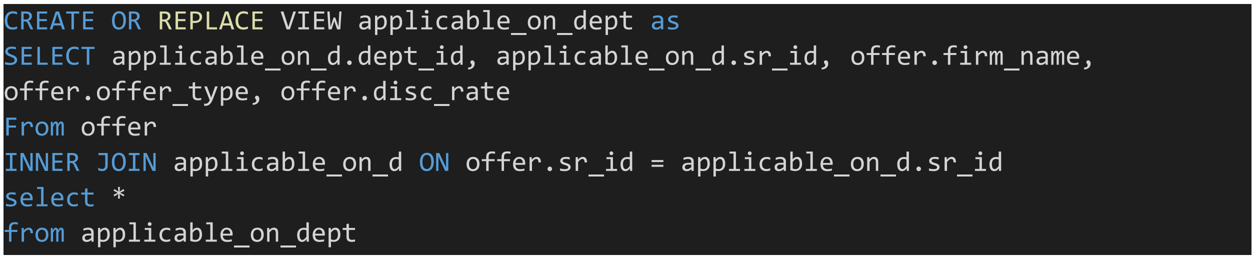


```
CREATE OR REPLACE VIEW applicable_on_brand as
SELECT applicable_on_b.brand_id, applicable_on_b.sr_id, offer.firm_name,
offer.offer_type, offer.disc_rate
From offer
INNER JOIN applicable_on_b ON offer.sr_id = applicable_on_b.sr_id
select *
from applicable_on_brand
```

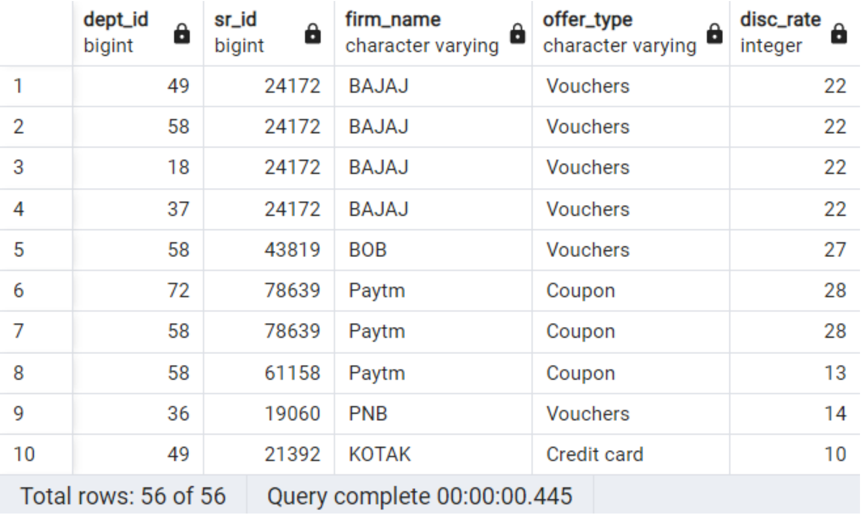


	brand_id bigint	sr_id bigint	firm_name character varying	offer_type character varying	disc_rate integer
1	4537	95403	KOTAK	Debit card	22
2	3238	95403	KOTAK	Debit card	22
3	2746	43819	BOB	Vouchers	27
4	2080	45556	PNB	Vouchers	15
5	2077	78639	Paytm	Coupon	28
6	4609	78639	Paytm	Coupon	28
7	1163	89452	BOB	Debit card	29
8	1743	89452	BOB	Debit card	29
9	4537	61158	Paytm	Coupon	13
10	4301	61158	Paytm	Coupon	13

Total rows: 58 of 58 Query complete 00:00:00.187



```
CREATE OR REPLACE VIEW applicable_on_dept as
SELECT applicable_on_d.dept_id, applicable_on_d.sr_id, offer.firm_name,
offer.offer_type, offer.disc_rate
From offer
INNER JOIN applicable_on_d ON offer.sr_id = applicable_on_d.sr_id
select *
from applicable_on_dept
```



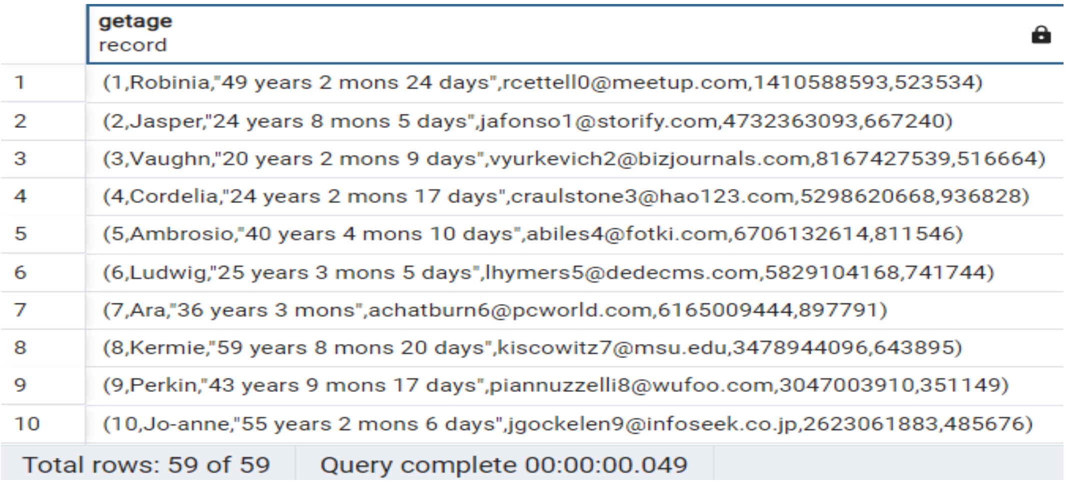
	dept_id bigint	sr_id bigint	firm_name character varying	offer_type character varying	disc_rate integer
1	49	24172	BAJAJ	Vouchers	22
2	58	24172	BAJAJ	Vouchers	22
3	18	24172	BAJAJ	Vouchers	22
4	37	24172	BAJAJ	Vouchers	22
5	58	43819	BOB	Vouchers	27
6	72	78639	Paytm	Coupon	28
7	58	78639	Paytm	Coupon	28
8	58	61158	Paytm	Coupon	13
9	36	19060	PNB	Vouchers	14
10	49	21392	KOTAK	Credit card	10

Total rows: 56 of 56 Query complete 00:00:00.445

7.3 Functions:

Function to calculate age from DOB:

```
create or replace function getage()
RETURNS TABLE(cust_id bigint,cust_name character varying,age interval,email
character varying,phone_num bigint,pin_code integer)
LANGUAGE 'plpgsql'
AS $BODY$
     BEGIN
        return query execute format('select cust_id, cust_name, age(dob), email,
phone_num, pin_code from customer');
     END;
$BODY$;
select getage();
```

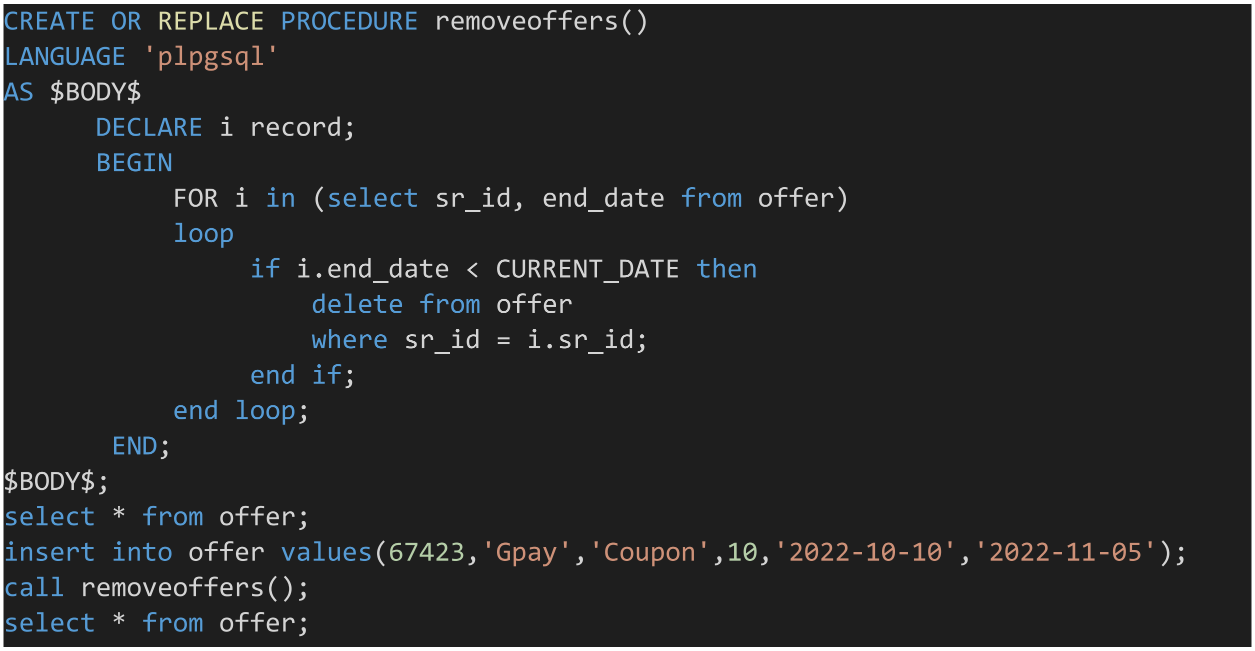


	getage record
1	(1,Robinia,"49 years 2 mons 24 days",rcettell0@meetup.com,1410588593,523534)
2	(2,Jasper,"24 years 8 mons 5 days",jafonso1@storify.com,4732363093,667240)
3	(3,Vaughn,"20 years 2 mons 9 days",vyurkevich2@bizjournals.com,8167427539,516664)
4	(4,Cordelia,"24 years 2 mons 17 days",craulstone3@hao123.com,5298620668,936828)
5	(5,Ambrosio,"40 years 4 mons 10 days",abiles4@fotki.com,6706132614,811546)
6	(6,Ludwig,"25 years 3 mons 5 days",lhymers5@dedecms.com,5829104168,741744)
7	(7,Ara,"36 years 3 mons",achatburn6@pcworld.com,6165009444,897791)
8	(8,Kermie,"59 years 8 mons 20 days",kiscowitz7@msu.edu,3478944096,643895)
9	(9,Perkin,"43 years 9 mons 17 days",piannuzzelli8@wufoo.com,3047003910,351149)
10	(10,Jo-anne,"55 years 2 mons 6 days",jgockelen9@infoseek.co.jp,2623061883,485676)

Total rows: 59 of 59 Query complete 00:00:00.049

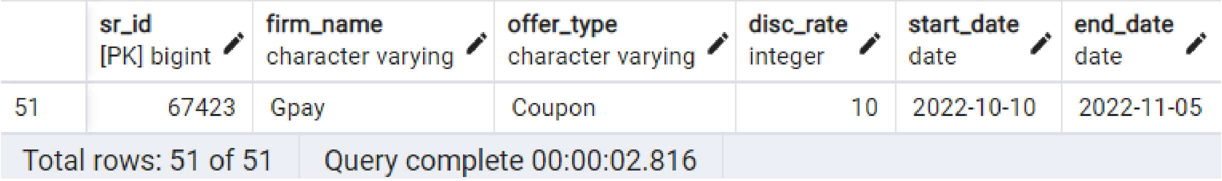
7.4 Procedures:

Procedure to remove completed offers:



```
CREATE OR REPLACE PROCEDURE removeoffers()
LANGUAGE 'plpgsql'
AS $BODY$
      DECLARE i record;
      BEGIN
           FOR i in (select sr_id, end_date from offer)
           loop
                if i.end_date < CURRENT_DATE then
                    delete from offer
                    where sr_id = i.sr_id;
                end if;
           end loop;
       END;
$BODY$;
select * from offer;
insert into offer values(67423,'Gpay','Coupon',10,'2022-10-10','2022-11-05');
call removeoffers();
select * from offer;
```

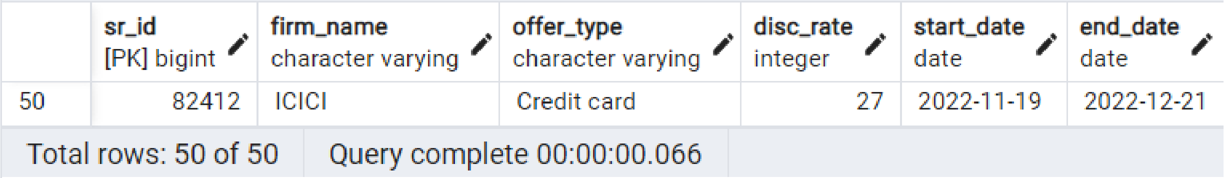
Before Call of procedure:



	sr_id [PK] bigint	firm_name character varying	offer_type character varying	disc_rate integer	start_date date	end_date date
51	67423	Gpay	Coupon	10	2022-10-10	2022-11-05

Total rows: 51 of 51 Query complete 00:00:02.816

After Call of procedure:



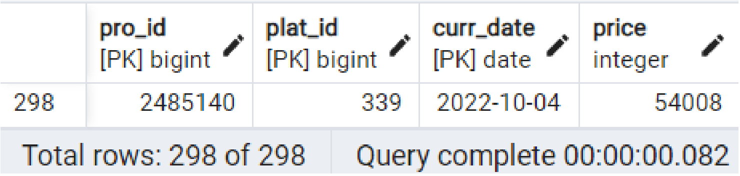
	sr_id [PK] bigint	firm_name character varying	offer_type character varying	disc_rate integer	start_date date	end_date date
50	82412	ICICI	Credit card	27	2022-11-19	2022-12-21

Total rows: 50 of 50 Query complete 00:00:00.066

Procedure to keep track of price history:

```
CREATE OR REPLACE PROCEDURE keeptrack()
LANGUAGE 'plpgsql'
AS $BODY$
      DECLARE i record;
      BEGIN
        FOR i in (select pro_id, plat_id, best_deal from base_price)
        loop
            insert into track values (i.pro_id, i.plat_id, CURRENT_DATE,
i.best_deal);
        end loop;
      END;
$BODY$;
select * from track;
select * from product;
select * from available_on;
select * from base_price;
insert into product values(9838428,'HJEOHE',7999,49,3243);
insert into available_on values(455,9838428,12,3.3,1000);
where pro_id=9838428
select * from product;
select * from available_on;
select * from base_price;
call keeptrack();
select * from track;
```

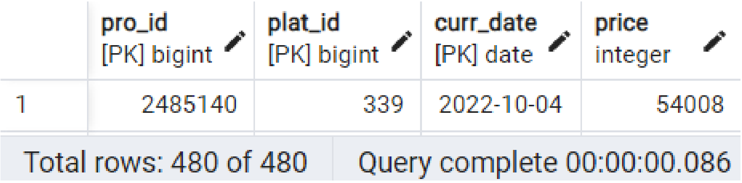
Before Call of procedure:



	pro_id [PK] bigint	plat_id [PK] bigint	curr_date [PK] date	price integer
298	2485140	339	2022-10-04	54008

Total rows: 298 of 298 Query complete 00:00:00.082

After Call of procedure:

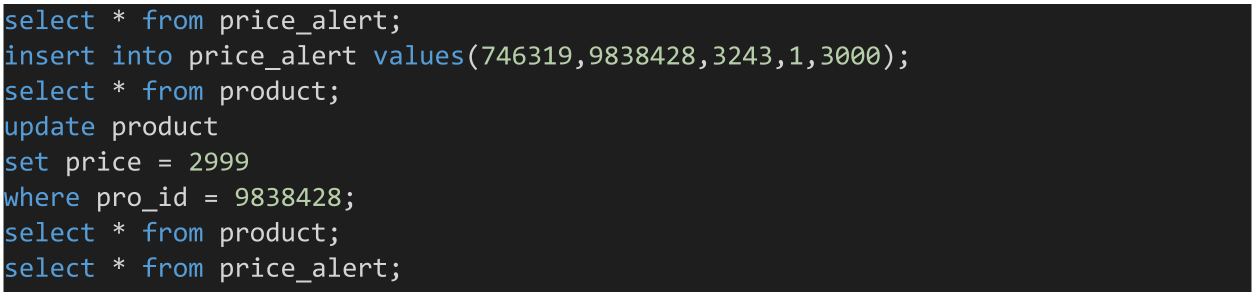


	pro_id [PK] bigint	plat_id [PK] bigint	curr_date [PK] date	price integer
1	2485140	339	2022-10-04	54008

Total rows: 480 of 480 Query complete 00:00:00.086

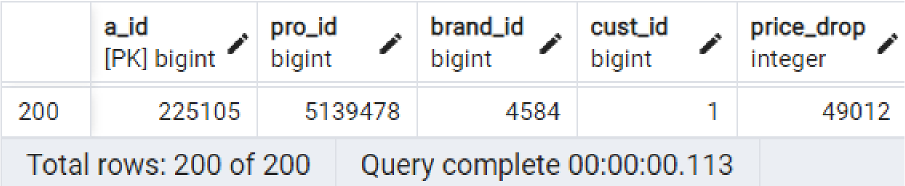
7.5 Triggers:

Trigger function to remove completed alerts:



```
select * from price_alert;
insert into price_alert values(746319,9838428,3243,1,3000);
select * from product;
update product
set price = 2999
where pro_id = 9838428;
select * from product;
select * from price_alert;
```

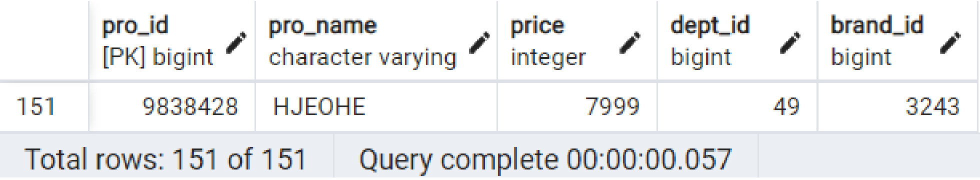
Before insertion in price_alert:



	a_id [PK] bigint	pro_id bigint	brand_id bigint	cust_id bigint	price_drop integer
200	225105	5139478	4584	1	49012

Total rows: 200 of 200 Query complete 00:00:00.113

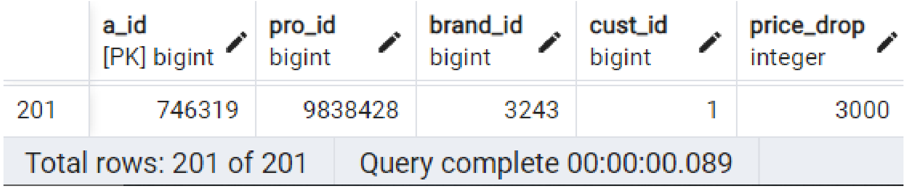
Before updation in product:



	pro_id [PK] bigint	pro_name character varying	price integer	dept_id bigint	brand_id bigint
151	9838428	HJEOHE	7999	49	3243

Total rows: 151 of 151 Query complete 00:00:00.057

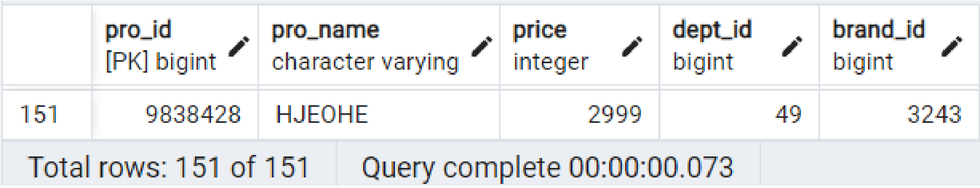
After insertion in price_alert:



	a_id [PK] bigint	pro_id bigint	brand_id bigint	cust_id bigint	price_drop integer
201	746319	9838428	3243	1	3000

Total rows: 201 of 201 Query complete 00:00:00.089

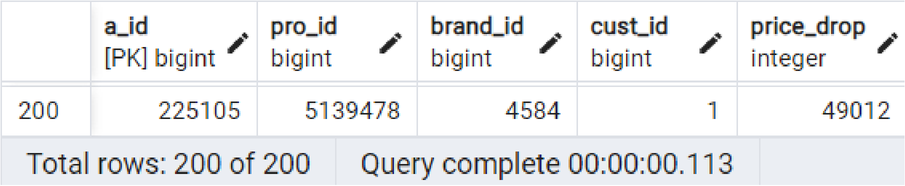
After updation in product:



	pro_id [PK] bigint	pro_name character varying	price integer	dept_id bigint	brand_id bigint
151	9838428	HJEOHE	2999	49	3243

Total rows: 151 of 151 Query complete 00:00:00.073

After running of trigger before insertion:



	a_id [PK] bigint	pro_id bigint	brand_id bigint	cust_id bigint	price_drop integer
200	225105	5139478	4584	1	49012

Total rows: 200 of 200 Query complete 00:00:00.113

Note: whenever any product is updated in product table then if it satisfy any of the alert set by any customer then all that alerts will be deleted indicating completion of alert.

Section 8: Project Code with Output Screenshot

8.1 Web Application Code:

8.1.1 Connects Postgres with Django:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'price_tracker',
        'USER': 'postgres',
        'PASSWORD': 'admin',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

8.1.2 Code for setting up the URLs:

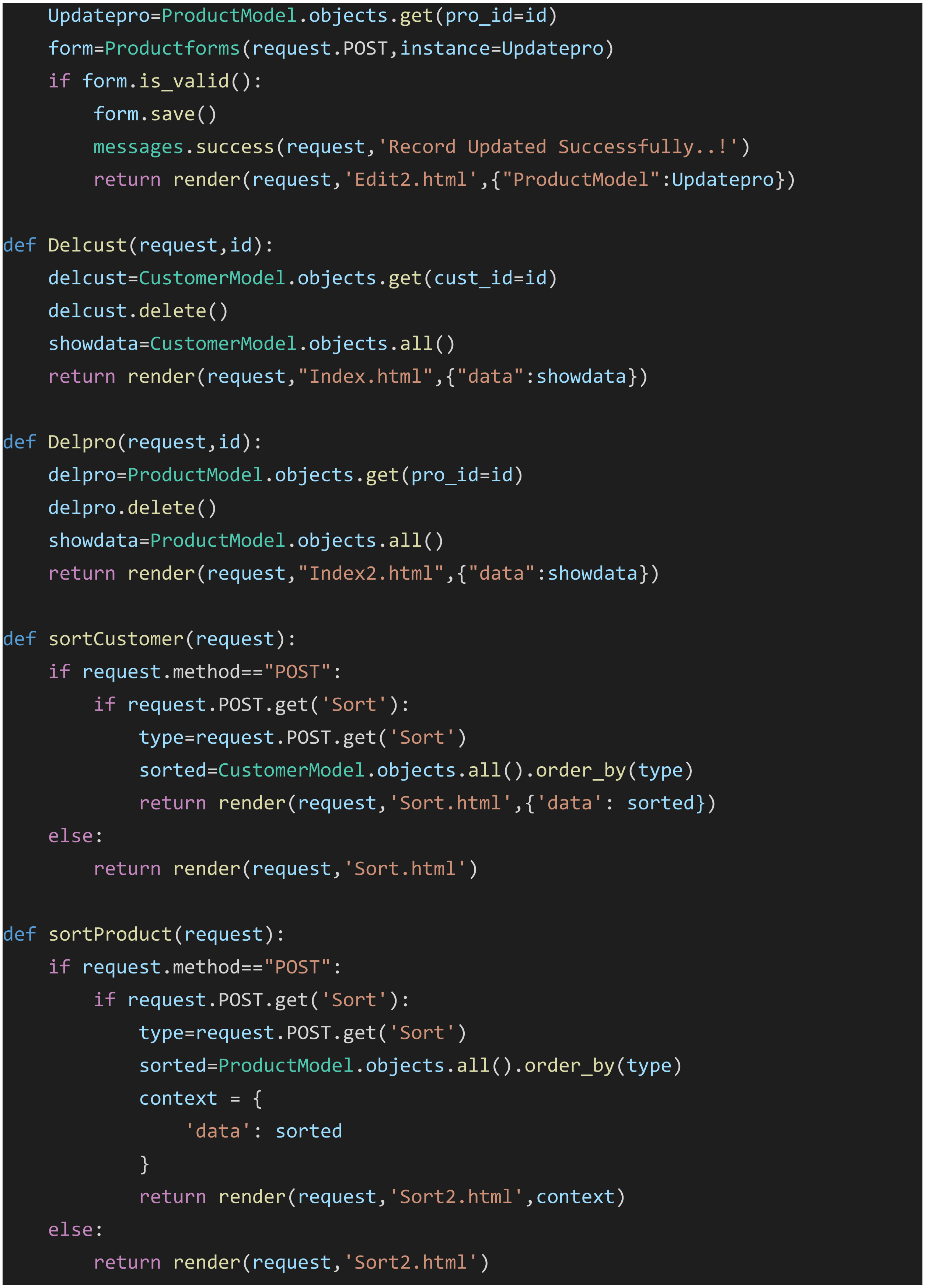
```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.HomePage,name="HomePage"),

    path('showCustomer',views.showcust,name="showcust"),
    path('Insertcust',views.Insertcust,name="Insertcust"),
    path('Edit/<int:id>',views.Editcust,name="Editcust"),
    path('Update/<int:id>',views.updatecust,name="updatecust"),
    path('Delete/<int:id>',views.Delcust,name="Delcust"),
    path('Sort',views.sortCustomer,name="sortCustomer"),

    path('showProduct',views.showpro,name="showpro"),
    path('Insertpro',views.Insertpro,name="Insertpro"),
    path('Edit2/<int:id>',views.Editpro,name="Editpro"),
    path('Update2/<int:id>',views.updatepro,name="updatepro"),
    path('Delete2/<int:id>',views.Delpro,name="Delpro"),
    path('Sort2',views.sortProduct,name="sortProduct"),
    path('Set/<int:id>',views.Setalert,name="Setalert"),
    path('Seealert/<int:id>',views.showcustalerts,name="showcustalerts"),
    path('Delete3/<int:id>',views.Delalert,name="Delalert"),
    path('runQuery',views.runQuery,name="runQuery"),
]
```

8.1.3 Code for setting the models used for creating views:

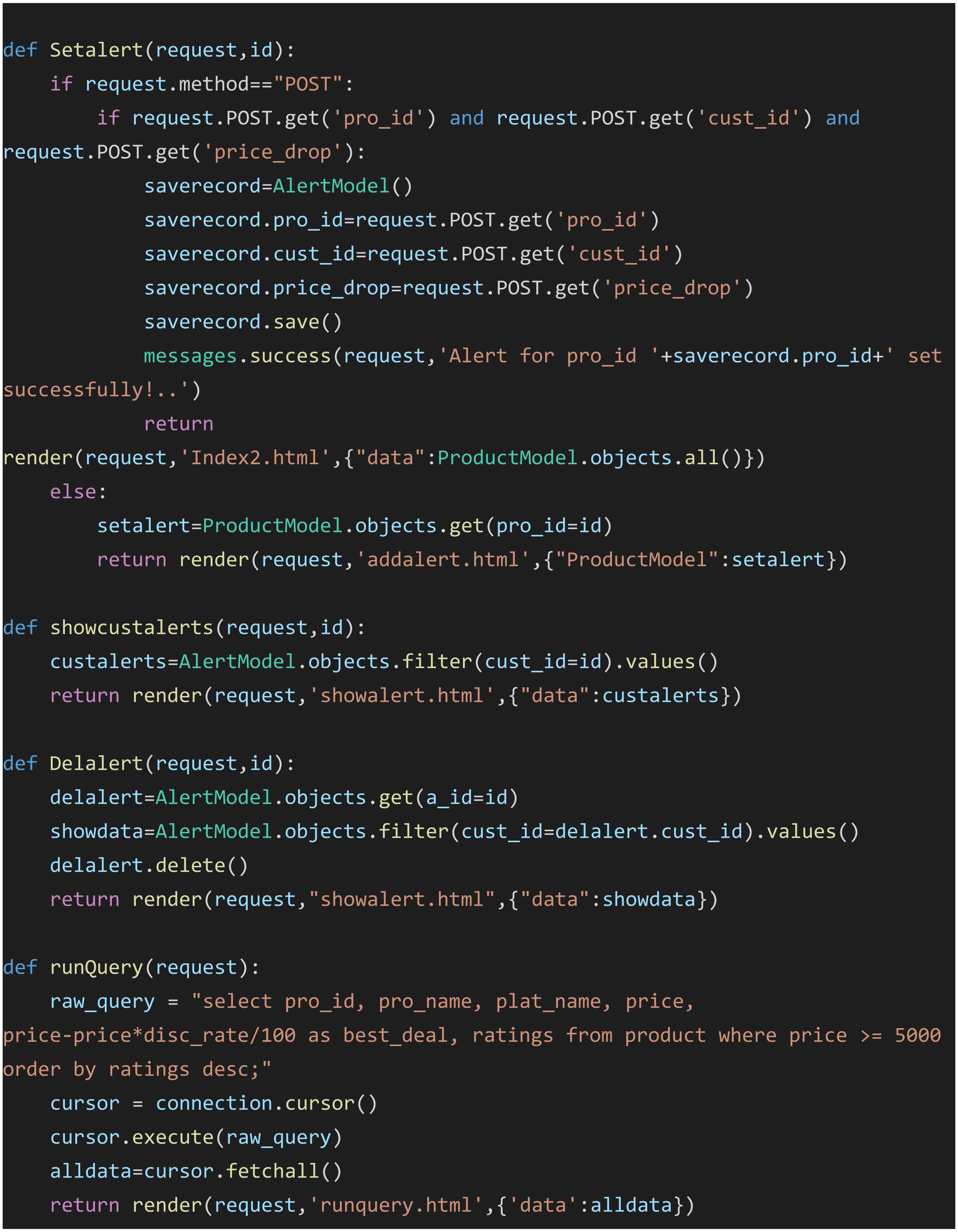
```
class CustomerModel(models.Model):
    cust_id=models.BigIntegerField(primary_key=True)
    cust_name=models.CharField(max_length=30)
    cust_pass=models.CharField(max_length=8,null=False)
    dob=models.DateField()
    pin_code=models.IntegerField()
    email=models.CharField(max_length=255)
    phone_num=models.BigIntegerField()
    class Meta:
        db_table="Customer"

class ProductModel(models.Model):
    pro_id=models.BigIntegerField(primary_key=True)
    pro_name=models.CharField(max_length=255,null=False)
    price=models.IntegerField(null=False)
    dept_name=models.CharField(max_length=255,null=False)
    brand_name=models.CharField(max_length=255,null=False)
    plat_name=models.CharField(max_length=255,null=False)
    disc_rate=models.IntegerField()
    ratings=models.DecimalField(max_digits=2,decimal_places=1)
    class Meta:
        db_table="product"

class AlertModel(models.Model):
    a_id=models.AutoField(primary_key=True)
    pro_id=models.BigIntegerField(null=False)
    cust_id=models.BigIntegerField(null=False)
    price_drop=models.IntegerField(null=False)
    class Meta:
        db_table="alerts"
```

8.1.4 Code for fetching, editing, deleting, sorting and querying in the database:

```
def HomePage(request):
    return render(request,'main.html')

def showcust(request):
    showall=CustomerModel.objects.all()
    return render(request,'Index.html',{"data":showall})

def showpro(request):
    showall=ProductModel.objects.all()
    return render(request,'Index2.html',{"data":showall})

def Insertcust(request):
    if request.method=="POST":
        if request.POST.get('cust_id') and request.POST.get('cust_name') and
request.POST.get('cust_pass') and request.POST.get('dob') and
request.POST.get('pin_code') and request.POST.get('email') and
request.POST.get('phone_num'):
            saverecord=CustomerModel()
            saverecord.cust_id=request.POST.get('cust_id')
            saverecord.cust_name=request.POST.get('cust_name')
            saverecord.cust_pass=request.POST.get('cust_pass')
            saverecord.dob=request.POST.get('dob')
            saverecord.pin_code=request.POST.get('pin_code')
            saverecord.email=request.POST.get('email')
            saverecord.phone_num=request.POST.get('phone_num')
            saverecord.save()
            messages.success(request,'Customer '+saverecord.cust_id+ ' is Saved
Successfully..!')
            return render(request,'Insert.html')
    else:
        return render(request,'Insert.html')



def Insertpro(request):
    if request.method=="POST":
        if request.POST.get('pro_id') and request.POST.get('pro_name') and
request.POST.get('price') and request.POST.get('dept_name') and
request.POST.get('brand_name') and request.POST.get('plat_name') and
request.POST.get('disc_rate') and request.POST.get('ratings'):
            saverecord=ProductModel()
            saverecord.pro_id=request.POST.get('pro_id')
            saverecord.pro_name=request.POST.get('pro_name')
            saverecord.price=request.POST.get('price')
            saverecord.dept_name=request.POST.get('dept_name')
            saverecord.brand_name=request.POST.get('brand_name')
            saverecord.plat_name=request.POST.get('plat_name')
            saverecord.disc_rate=request.POST.get('disc_rate')
            saverecord.ratings=request.POST.get('ratings')
            saverecord.save()
            messages.success(request,'Product '+saverecord.pro_id+ ' is Saved
Successfully..!')
            return render(request,'Insert2.html')
    else:
        return render(request,'Insert2.html')

def Editcust(request,id):
    editcustobj=CustomerModel.objects.get(cust_id=id)
    return render(request,'Edit.html',{"CustomerModel":editcustobj})

def updatecust(request,id):
    Updatecust=CustomerModel.objects.get(cust_id=id)
    form=Customerforms(request.POST,instance=Updatecust)
    if form.is_valid():
        form.save()
        messages.success(request,'Record Updated Successfully..!')
        return render(request,'Edit.html',{"CustomerModel":Updatecust})

def Editpro(request,id):
    editproobj=ProductModel.objects.get(pro_id=id)
    return render(request,'Edit2.html',{"ProductModel":editproobj})



def updatepro(request,id):
```



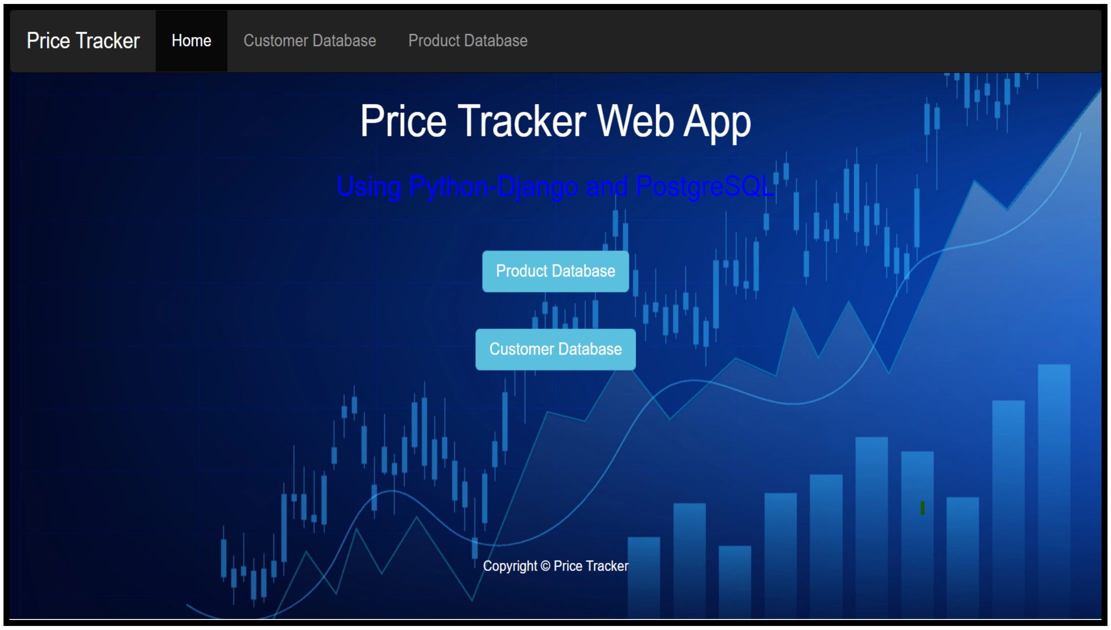
```
    Updatepro=ProductModel.objects.get(pro_id=id)
    form=Productforms(request.POST,instance=Updatepro)
    if form.is_valid():
        form.save()
        messages.success(request,'Record Updated Successfully..!')
        return render(request,'Edit2.html',{"ProductModel":Updatepro})

def Delcust(request,id):
    delcust=CustomerModel.objects.get(cust_id=id)
    delcust.delete()
    showdata=CustomerModel.objects.all()
    return render(request,"Index.html",{"data":showdata})

def Delpro(request,id):
    delpro=ProductModel.objects.get(pro_id=id)
    delpro.delete()
    showdata=ProductModel.objects.all()
    return render(request,"Index2.html",{"data":showdata})

def sortCustomer(request):
    if request.method=="POST":
        if request.POST.get('Sort'):
            type=request.POST.get('Sort')
            sorted=CustomerModel.objects.all().order_by(type)
            return render(request,'Sort.html',{'data': sorted})
    else:
        return render(request,'Sort.html')

def sortProduct(request):
    if request.method=="POST":
        if request.POST.get('Sort'):
            type=request.POST.get('Sort')
            sorted=ProductModel.objects.all().order_by(type)
            context = {
                'data': sorted
            }
            return render(request,'Sort2.html',context)
    else:
        return render(request,'Sort2.html')
```



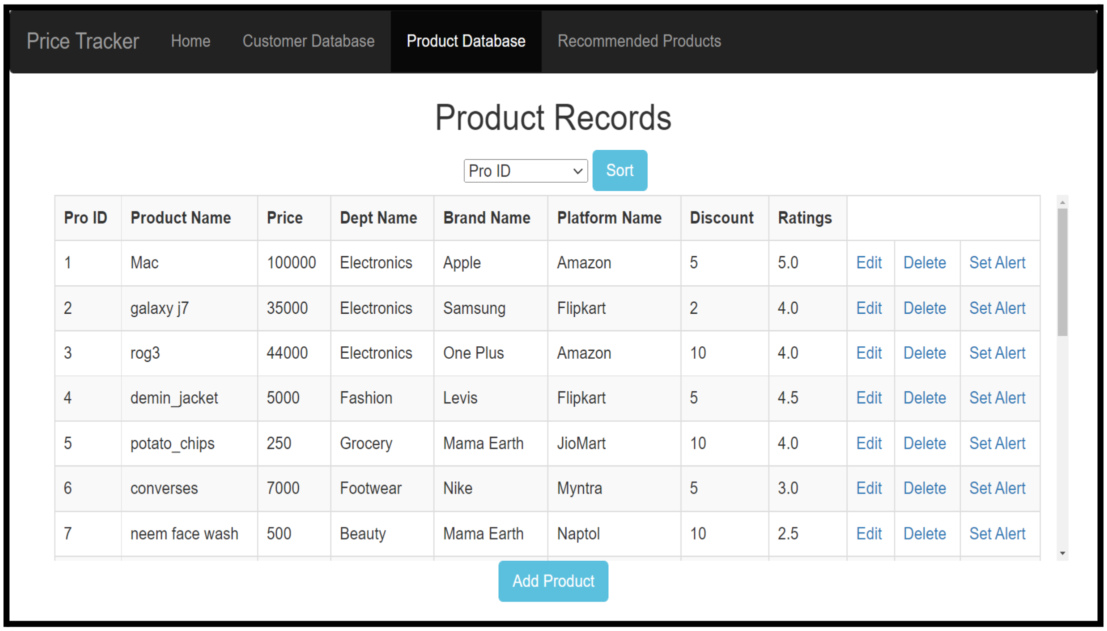
```
def Setalert(request,id):
    if request.method=="POST":
        if request.POST.get('pro_id') and request.POST.get('cust_id') and
request.POST.get('price_drop'):
            saverecord=AlertModel()
            saverecord.pro_id=request.POST.get('pro_id')
            saverecord.cust_id=request.POST.get('cust_id')
            saverecord.price_drop=request.POST.get('price_drop')
            saverecord.save()
            messages.success(request,'Alert for pro_id '+saverecord.pro_id+' set
successfully!..')
            return
render(request,'Index2.html',{"data":ProductModel.objects.all()})
    else:
        setalert=ProductModel.objects.get(pro_id=id)
        return render(request,'addalert.html',{"ProductModel":setalert})

def showcustalerts(request,id):
    custalerts=AlertModel.objects.filter(cust_id=id).values()
    return render(request,'showalert.html',{"data":custalerts})

def Delalert(request,id):
    delalert=AlertModel.objects.get(a_id=id)
    showdata=AlertModel.objects.filter(cust_id=delalert.cust_id).values()
    delalert.delete()
    return render(request,"showalert.html",{"data":showdata})

def runQuery(request):
    raw_query = "select pro_id, pro_name, plat_name, price,
price-price*disc_rate/100 as best_deal, ratings from product where price >= 5000
order by ratings desc;"
    cursor = connection.cursor()
    cursor.execute(raw_query)
    alldata=cursor.fetchall()
    return render(request,'runquery.html',{'data':alldata})
```

8.2 Snapshots of Web Application:

1. Home Page



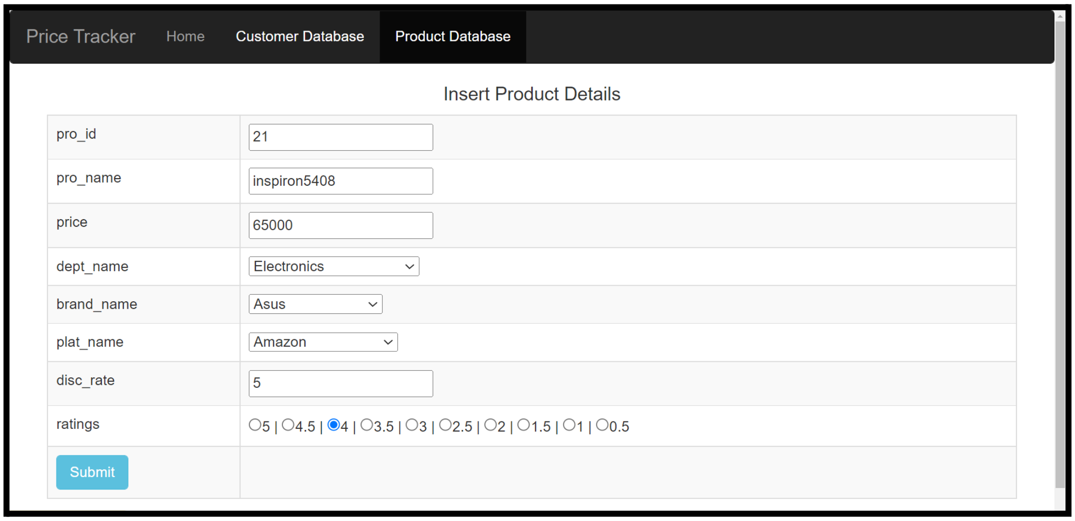
2. Product Database



Product Records

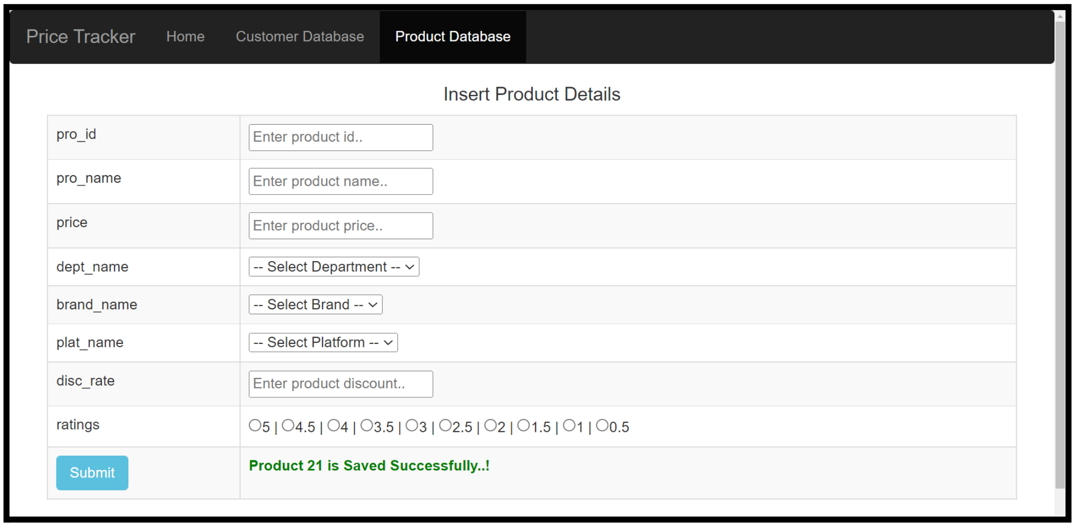
Pro ID	Product Name	Price	Dept Name	Brand Name	Platform Name	Discount	Ratings			
1	Mac	100000	Electronics	Apple	Amazon	5	5.0	Edit	Delete	Set Alert
2	galaxy j7	35000	Electronics	Samsung	Flipkart	2	4.0	Edit	Delete	Set Alert
3	rog3	44000	Electronics	One Plus	Amazon	10	4.0	Edit	Delete	Set Alert
4	demin_jacket	5000	Fashion	Levis	Flipkart	5	4.5	Edit	Delete	Set Alert
5	potato_chips	250	Grocery	Mama Earth	JioMart	10	4.0	Edit	Delete	Set Alert
6	converses	7000	Footwear	Nike	Myntra	5	3.0	Edit	Delete	Set Alert
7	neem face wash	500	Beauty	Mama Earth	Naptol	10	2.5	Edit	Delete	Set Alert

3. Insert Product Record



Insert Product Details

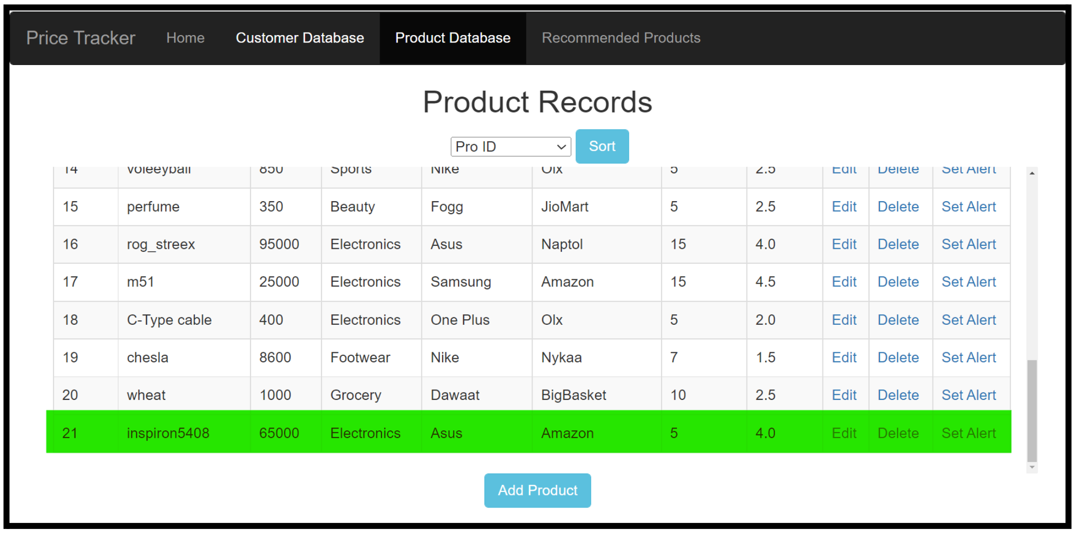
pro_id	21
pro_name	inspiron5408
price	65000
dept_name	Electronics
brand_name	Asus
plat_name	Amazon
disc_rate	5
ratings	○5 | ○4.5 | ●4 | ○3.5 | ○3 | ○2.5 | ○2 | ○1.5 | ○1 | ○0.5



Insert Product Details

pro_id	Enter product id..
pro_name	Enter product name..
price	Enter product price..
dept_name	-- Select Department --
brand_name	-- Select Brand --
plat_name	-- Select Platform --
disc_rate	Enter product discount..
ratings	○5 | ○4.5 | ○4 | ○3.5 | ○3 | ○2.5 | ○2 | ○1.5 | ○1 | ○0.5

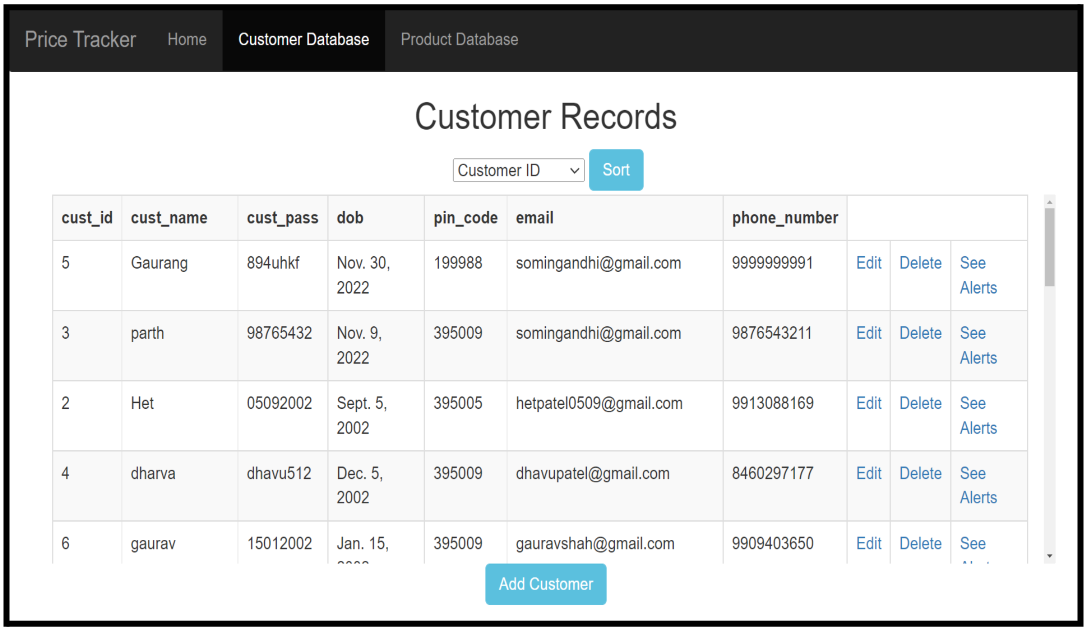
Product 21 is Saved Successfully..!



Product Records

14	volleyball	850	Sports	Nike	Olx	5	2.5	Edit	Delete	Set Alert
15	perfume	350	Beauty	Fogg	JioMart	5	2.5	Edit	Delete	Set Alert
16	rog_streex	95000	Electronics	Asus	Naptol	15	4.0	Edit	Delete	Set Alert
17	m51	25000	Electronics	Samsung	Amazon	15	4.5	Edit	Delete	Set Alert
18	C-Type cable	400	Electronics	One Plus	Olx	5	2.0	Edit	Delete	Set Alert
19	chesla	8600	Footwear	Nike	Nykaa	7	1.5	Edit	Delete	Set Alert
20	wheat	1000	Grocery	Dawaat	BigBasket	10	2.5	Edit	Delete	Set Alert
21	inspiron5408	65000	Electronics	Asus	Amazon	5	4.0	Edit	Delete	Set Alert

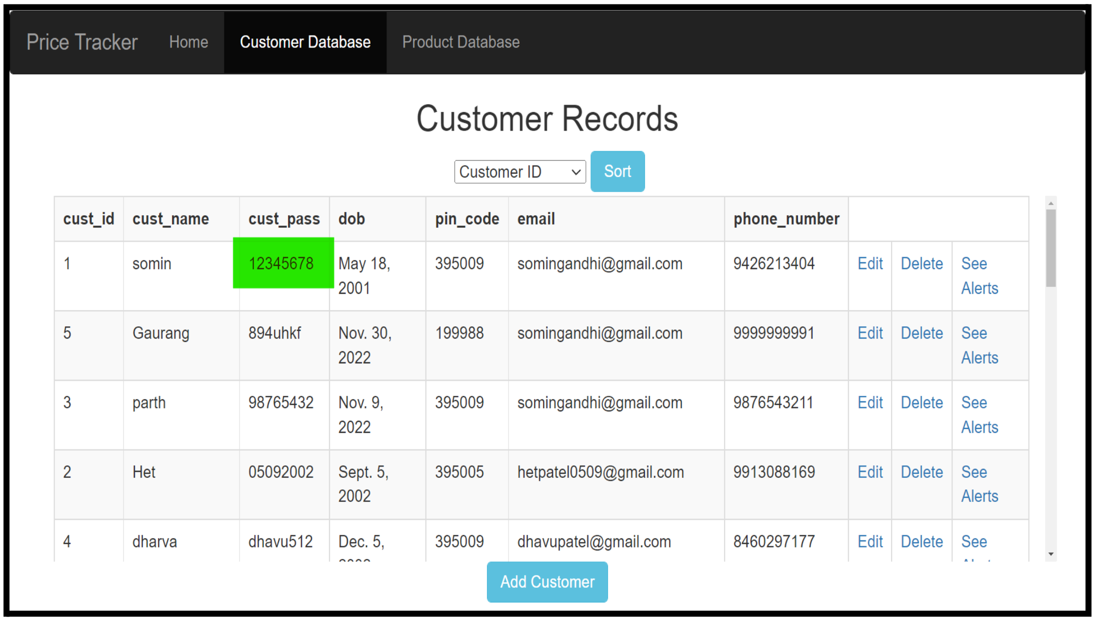
4. Customer Database



Customer Records

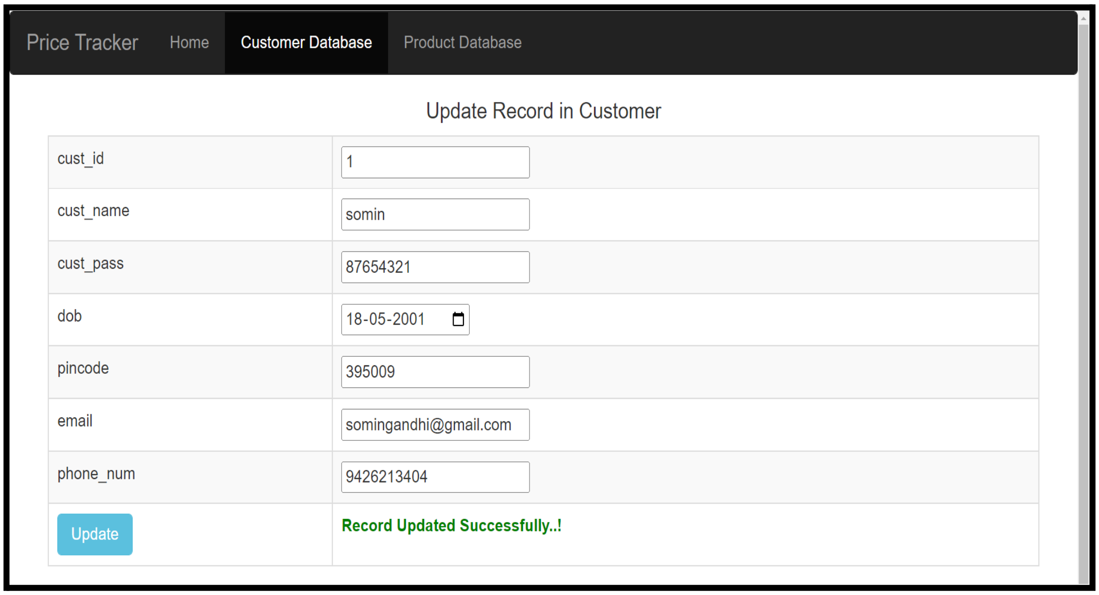
cust_id	cust_name	cust_pass	dob	pin_code	email	phone_number			
5	Gaurang	894uhkf	Nov. 30, 2022	199988	somingandhi@gmail.com	9999999991	Edit	Delete	See Alerts
3	parth	98765432	Nov. 9, 2022	395009	somingandhi@gmail.com	9876543211	Edit	Delete	See Alerts
2	Het	05092002	Sept. 5, 2002	395005	hetpatel0509@gmail.com	9913088169	Edit	Delete	See Alerts
4	dharva	dhavu512	Dec. 5, 2002	395009	dhavupatel@gmail.com	8460297177	Edit	Delete	See Alerts
6	gaurav	15012002	Jan. 15,	395009	gauravshah@gmail.com	9909403650	Edit	Delete	See

5. Edit Customer Database



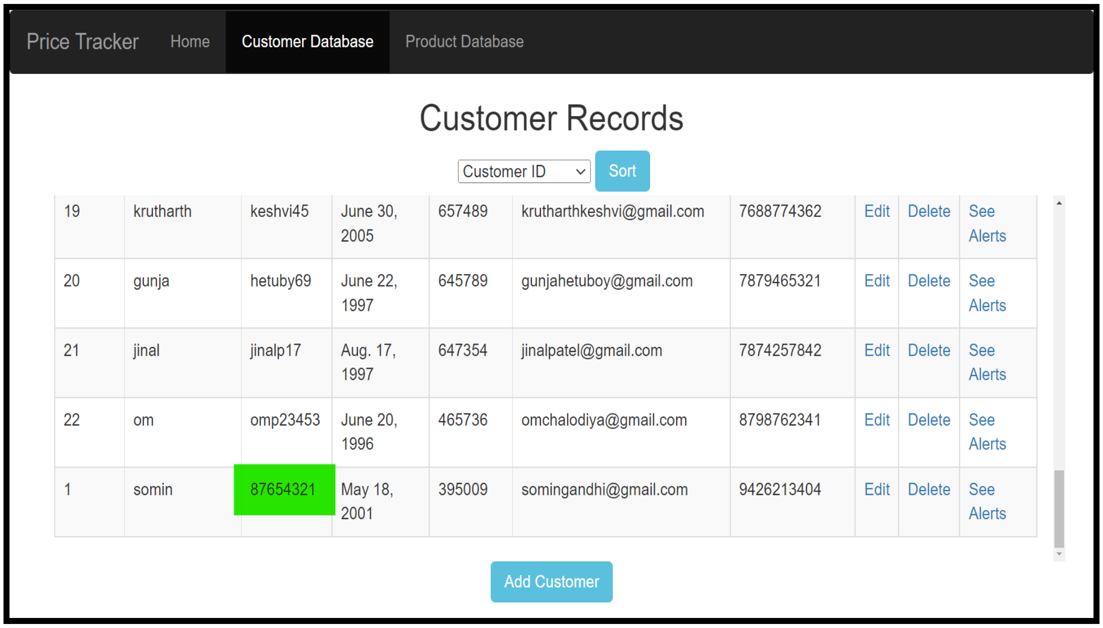
Customer Records

cust_id	cust_name	cust_pass	dob	pin_code	email	phone_number			
1	somin	12345678	May 18, 2001	395009	somingandhi@gmail.com	9426213404	Edit	Delete	See Alerts
5	Gaurang	894uhkf	Nov. 30, 2022	199988	somingandhi@gmail.com	9999999991	Edit	Delete	See Alerts
3	parth	98765432	Nov. 9, 2022	395009	somingandhi@gmail.com	9876543211	Edit	Delete	See Alerts
2	Het	05092002	Sept. 5, 2002	395005	hetpatel0509@gmail.com	9913088169	Edit	Delete	See Alerts
4	dharva	dhavu512	Dec. 5,	395009	dhavupatel@gmail.com	8460297177	Edit	Delete	See



Update Record in Customer

cust_id	1
cust_name	somin
cust_pass	87654321
dob	18-05-2001
pincode	395009
email	somingandhi@gmail.com
phone_num	9426213404
Update	**Record Updated Successfully..!**

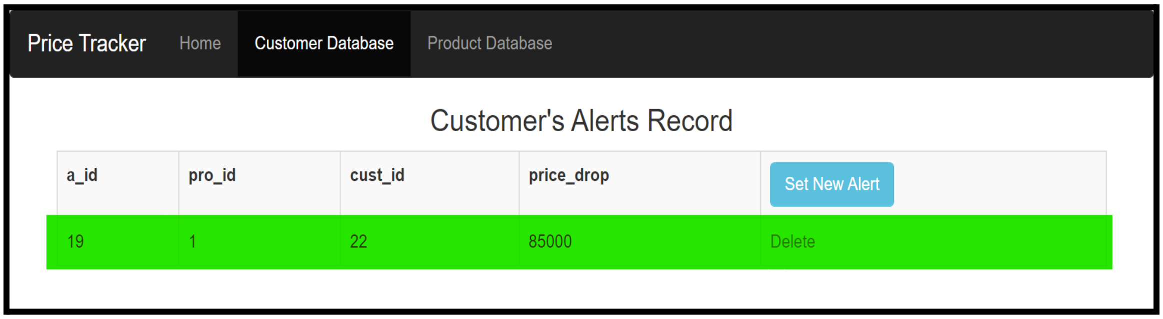


Customer Records

Customer ID

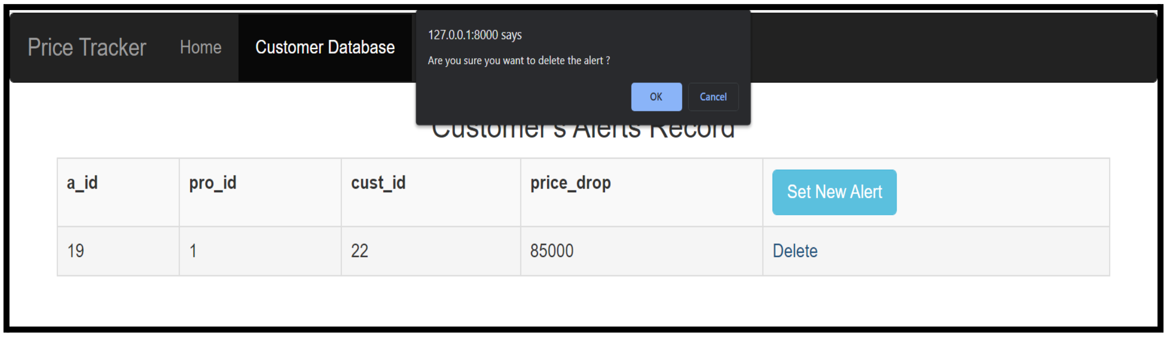
19	krutharth	keshvi45	June 30, 2005	657489	krutharthkeshvi@gmail.com	7688774362	Edit	Delete	See Alerts
20	gunja	hetuby69	June 22, 1997	645789	gunjahetuboy@gmail.com	7879465321	Edit	Delete	See Alerts
21	jinal	jinalp17	Aug. 17, 1997	647354	jinalpatel@gmail.com	7874257842	Edit	Delete	See Alerts
22	om	omp23453	June 20, 1996	465736	omchalodiya@gmail.com	8798762341	Edit	Delete	See Alerts
1	somin	87654321	May 18, 2001	395009	somingandhi@gmail.com	9426213404	Edit	Delete	See Alerts

6. Delete Customer's Alert



Customer's Alerts Record

a_id	pro_id	cust_id	price_drop	Set New Alert
19	1	22	85000	Delete



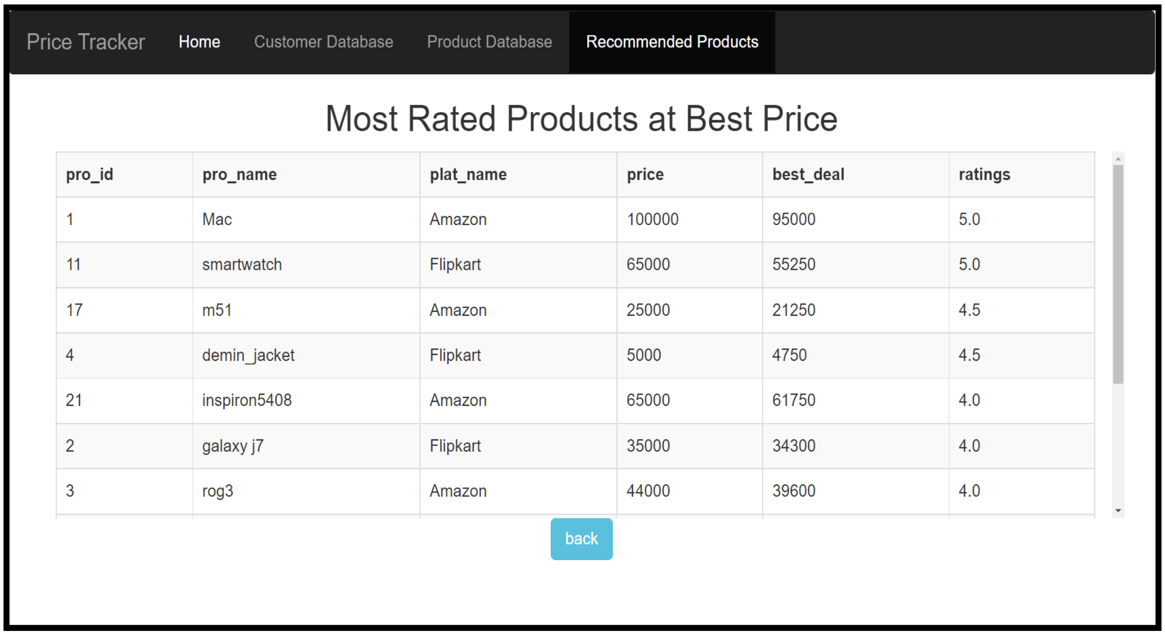
127.0.0.1:8000 says

Are you sure you want to delete the alert ?

Customer's Alerts Record

a_id	pro_id	cust_id	price_drop	Set New Alert
19	1	22	85000	Delete

7. Query : Display the most rated products with applicable discount



Most Rated Products at Best Price

pro_id	pro_name	plat_name	price	best_deal	ratings
1	Mac	Amazon	100000	95000	5.0
11	smartwatch	Flipkart	65000	55250	5.0
17	m51	Amazon	25000	21250	4.5
4	demin_jacket	Flipkart	5000	4750	4.5
21	inspiron5408	Amazon	65000	61750	4.0
2	galaxy j7	Flipkart	35000	34300	4.0
3	rog3	Amazon	44000	39600	4.0