DATABASE MANAGEMENT SYSTEM
(IT - 214)

PRODUCTION MANAGEMENT SYSTEM

SECTION: 2
GROUP: 11

GROUP MEMBERS:

Nandani Tank	202001201
Jaimin Satani	202001202

Table of Content:

Questionnaire Survey:
https://docs.google.com/forms/d/e/1FAIpQLSc3kBxxiCbaJG3As2-amDBu_VX1mC7jeVGqAq0-bqcx8kkUDg/viewform?usp=sf_link

ER Diagram: 2.11_ER
Query Link: Final_SQL.sql
DDL Script Link: DDL.sql
CSV Files: Final_Table_Big.zip
Website Link: https://github.com/1JDSOP/DBMS_Project/

Software **R**equirements **S**pecification document (**SRS** document)

PRODUCTION MANAGEMENT SYSTEM

1. Introduction:

A product management system(PMS) is a system that manages the records of employees and their work details, machines, and their details, produced goods and their details, different department details, sales details, and expenses details. A Production management system is designed to help Factories, including the home industry and large-level businesses.

1.1. Purpose:

Defining a production management System that manages the records of products which factory produces, details of factory employees details, departments of the production as per they produce a product, buyers of that product, sale of a particular product.

There are custom search capabilities to aid in finding factory employee information and working on employee records. This system shows the profit and loss after selling each product of the factory. This can make the system easier to use, maximizing the effectiveness of time and other details of the factory.

With the help of this system, the production management authority of some companies can easily manage data of all these things that were mentioned above and provide the functionality which is mentioned in the description.

1.2. Intended Audience and Reading Suggestions:

I. Intended Audience:

The main audience of this product management system is for management authority of the factory, which includes access to factory production growth and for their employees. Another major audience is factory employees. They use this system for their salary and for personal details. Factory buyers use this system for the purpose of checking the available stock of the goods and raw materials-related information for the factory. The general buyers of the goods use this system to check the availability of the product they want.

– Here, the following types of audience are intended.

1. Manager of Production management authority
2. Workers
3. Developers
4. Testers

II. Reading Suggestions:

1. https://www.studocu.com/in/document/st-josephs-college-autonomous/bachelors-of-commerce/student-management-system-srs/17638223

   Some details about the purpose of the system and what is the scope of the system.

2. https://www.cse.msu.edu/~chengb/RE-491/Papers/SRS-BECS-2007.pdf

   We are taking references for the functionality of the system and its description.

1.3. Scope:

The scope of the project includes the following:

- Any company which produces physical goods can use this system as it is not client-centric.
- Application Support & Maintenance after deployment to production
- The employee details module is reusable in software development companies and infrastructure development companies.
- The machine details module is reusable in infrastructure development companies.

- The sales details module is reusable in software development companies.

1.4. Description:

This section includes details about what is and is not expected of the product management system, in addition to which cases are intentionally unsupported and assumptions that will be used in the creation of the production management system.

The Production Management System will provide a number of functionalities. We can provide neat and easy to use database by production management system to the factories and and production companies. By this system management authority can track growth of factory, details about employees, machines, departments, etc. Also we can provide track of Sell and Order details, and by use of that details we can track growth of factory, profit & loss in interval of time.

The Product Management System will provide a number of functionalities. Each is listed below:

- **Employees**

  – As we know that employees have data about their employee IDs, password, employee personal details, working department details, salary, working product details, working hours, working machine details, shift(morning, noon), first day of joining date, Date of Birth etc.

  – Employees work on particular machine and in particular department.

  – Also Employees can edit some of the details like email address, mobile numbers.

  – Here each employee should have unique ID and password. So, that every employee can see their details on factory portal.

  – Also they can edit their details.

  – We are adding two functionalities:

    1. Auto - assign Employee ID to every employee
    2. Once any employee left the job His/Her data will be added to past employee details. So, that we can have details of all employees which had worked in company.

- **Products**

  – Products will be produced by departments by use of machines.
  – So, machines also produce products.
  – It shows product details like the name of the product, product ID, weight, available stock of a product, department of the product, rate of the product per packet, produce flag, and production capacity per day.
  – If produce flag is on then production of that product will be allowed
  – Here we are auto assigning product ID
  – Also for calculate cost and profit by products we need to store date wise production of all products.
  – Also products are connected with raw materials. Because by use of Raw materials products were produced.
  – It shows product details like the name, ID, weight, available stock of a product, department of the product, rate of the product per packet, and production capacity per day.
  – Many employees will be working on ne product and by use of many machines in which products will be produced.

- **Machines**

  – Here by observation employees will work on machines and machines will produce products.
  – Also Department will use machines to produce products.
  – Here one machine will produce only one product but products will be produced by many machines.
  – Here many employees will work on one machine.
  – This data contains machine data like machine id, rpm(round per minute) of the machine, working condition of the machine (Description), department of the machine, service provided by the machine, price of the machine, maintenance cost per month, employee working on machine, per day power consumption and date when the machine was planted.
  – Auto assigns machine ID to machines

- **Departments**

  – Here as mentioned above Departments will use use machines to produce products.
  – Also employees will work in departments.
  – One Department will have many employees and many machines.
  – Department details contain department name, id, address, machine capacity, employee capacity.
  – Here we are Auto assigning department ID to departments.

- **Raw Materials**

  – Here as all know products will use raw materials.
  – Raw materials will be supplied by suppliers.
  – Raw material details contain data about raw material id, name, stock, price.
  – IDs will be auto assigned.

- **Sell details**

  – Sales details will be used by management authority to calculate profit and track growth.
  – This contains data about the quantity of product sold by the factory who bought the product, the rate of the product buyer bought, date & time of the product when sold.

- **Buyer**

  – Contain buyer ID, password, buyer name, contact details, and email.
  – Buyers can buy products from the factory.

- **Supplier**

  – Contain supplier ID, supplier name, contact details, and email.
  – Suppliers can supply Raw materials to the factory.

- **Raw material order**

  – contain data about material ID, material name, supplier ID, the quantity of product bought from a supplier, price per kg, date and time of bought raw material.
  – Raw material order can be updated by suppliers.

- **Product - raw material connection**

  – As product and raw material have many to many relationships, we need to define one relationship that contains data about the relationship of raw material of the product and raw material.

  – It relates product ID with particular raw material ID and also contains data about for one packet of product how much raw material is needed(in kg).

– Keeping these suggestions in mind, we can design an efficient and user-friendly database that makes the process easier for all.

2. Requirements Collection/ Fact-Finding Phase

2.1 Background Readings:

- **Data collected from Wikipedia:**

  – In the early days production management was very difficult for companies that have a huge amount of production.
  – People need to write about sales and supplies in books.
  – And in the calculation of profit/loss, finding particular data about workers was very difficult.
  – But now as new software has arrived like Absolute ERP, AVEVA, Tally ERP, and many more, people are able to fully fill all these difficulties while handling and manipulating data.
  – So, we are also intended to build such a kind of software.

- **Data collected from tulip.co :**

  Intuitive user interface: Additionally, the straightforward features allow supervisors and managers to easily schedule activities, eliminating the need for frequent manual checking throughout the day.

  Detailed customization: Manufacturers can modify a production management system to fit their unique needs.

  Comprehensive analytics: Managing production requires different kinds of data, giving managers the necessary information to make decisions. But with modern factories, the sheer amount of data generated needs significant analysis.

  Shared access: Modern digital production management tools do away with the traditional methods that were highly siloed. Instead, the new iteration is cloud-based, ensuring that relevant entities can access data from anywhere.

  This allows for better collaboration among the various departments, enabling better planning and management. A production

management system is equipped with powerful analytics that provides detailed insights into the production situation, allowing managers to make informed decisions.

Reference reading:

- https://tulip.co/blog/production-management-in-the-modern-factory/
- https://en.wikipedia.org/wiki/Product_management

2.2 Interviews:

Interview Plan 1:

Roleplay Interview Plan

System: Production management Database
Interviewee: Ankur Vejapara
Designation: Owner of Maya Flour Mills

Interviewer:

1. Nandani Tank (202001201)
2. Jaimin Satani (202001202)

Date: 29/09/2022
Time: 14:30
Duration: 30 Minutes
Place: Virtual (Zoom video call)

Purpose of Interview:

– Preliminary meeting to know about how he manages the factory.

Agenda:

– To know the current situation of production management.
– Knowing about the current system that he is using for production management.
– Improvements that can be done in the current production management system.

Documents to be brought to the interview :

– Documents related to the data about current progress of the company based on their production.
– Noticed production difficulties by outsiders about the company.

Interview summary 1:

System: Production management Database
Interviewee: Ankur Vejapara
Designation: Owner of Maya Flour Mills

Interviewer:

1. Nandani Tank (202001201)
2. Jaimin Satani (202001202)

Date: 29/09/2022
Time: 14:30
Duration: 30 Minutes
Place: Virtual (Zoom video call)

Outcomes of the interview:

– They don’t have any particular system for product management, Instead they are using menial Excel files for storing, deleting, updating data about sales, workers etc.
– Workers are not able to see their working details because of the file system. They need to remember it or write it somewhere.
– Management authorities also face problems while operating with data.

Suggestions given:

– Build such a system that provides easeness for insertion, deletion & updating data efficiently.
– Provide interfaces that have authorization like, workers should have only reading permission.
– Easy to use interface that can be used by authorities.

Interview Plan 2:

Roleplay Interview Plan

System: Production management Database
Interviewee: Vivek Bavishi
Designation: Manager at Maya Flour Mills

Interviewer:

1. Nandani Tank (202001201)
2. Jaimin Satani (202001202)

Date: 29/09/2022
Time: 15:30
Duration: 30 Minutes
Place: Virtual (Zoom video call)

Purpose of Interview:

– Preliminary meeting to know about how he manages the workers and supplies of raw materials.

Agenda:

– To know the current situation of production management.
– Knowing about the current system that he is using for production management.
– Improvements that can be done in the current production management system.

Documents to be brought to the interview :

– Documents related to the data about current progress of the company based on their production.
– Noticed production difficulties by outsiders about the company.

Interview Summary 2:

System: Production management Database
Interviewee: Vivek Bavishi
Designation: Manager at Maya Flour Mills

Interviewer:

1. Nandani Tank (202001201)
2. Jaimin Satani (202001202)

Date: 29/09/2022
Time: 15:30
Duration: 30 Minutes
Place: Virtual (Zoom video call)

Outcomes of the interview:

– They don't have any particular system for product management, Instead they are using menial Excel files for storing, deleting, updating data about sales, workers etc.
– He faces difficulties while calculating the overall profit of the company because of the file system.
– While calculating the total expenses of the company for some interval he needs to work for many hours.

Suggestions given:

– Provide one interface that can be used for getting total expenses and earnings for a time interval.

Interview Plan 3 :

System :Production management database

Interviewee : Employee at Factory

Designation: The person who work at factory

Interviewer:

1. Nandani Tank (202001201)
2. Jaimin Satani (202001202)

Date: 29/09/2022
Time: 16:30
Duration: 30 Minutes
Place: At factory place

Purpose of Interview :

Preliminary meeting to know about how the factory manages and workplace efficiency.

Agenda :

– Thought on current management of the factory.
– The system is used for producing major products.
– Difficulties faced in the factory.
– Thought on how to improve the factory management system.

Documents to be brought to the interview :

– Magazines that talk about factory production and their workspace.

Interview summary 3 :

System : Production management database

Interviewee : Employee at Factory

Designation: The person who work at factory

Interviewer :

1. Nandani Tank (202001201)
2. Jaimin Satani (202001202)

Date : 29/09/2022
Time : 16:30
Duration : 20 Minutes
Place : At factory

Outcomes of the interview :

– To know about how factories work and produce items.
– They face difficulty when products need Raw materials but there is no available stock for Raw materials.
– Workers are not able to access their details about their salaries, working days details, about the shift(day-night).
-if there is any situation of machine failures happening there is no backup for that.

Suggestions given:

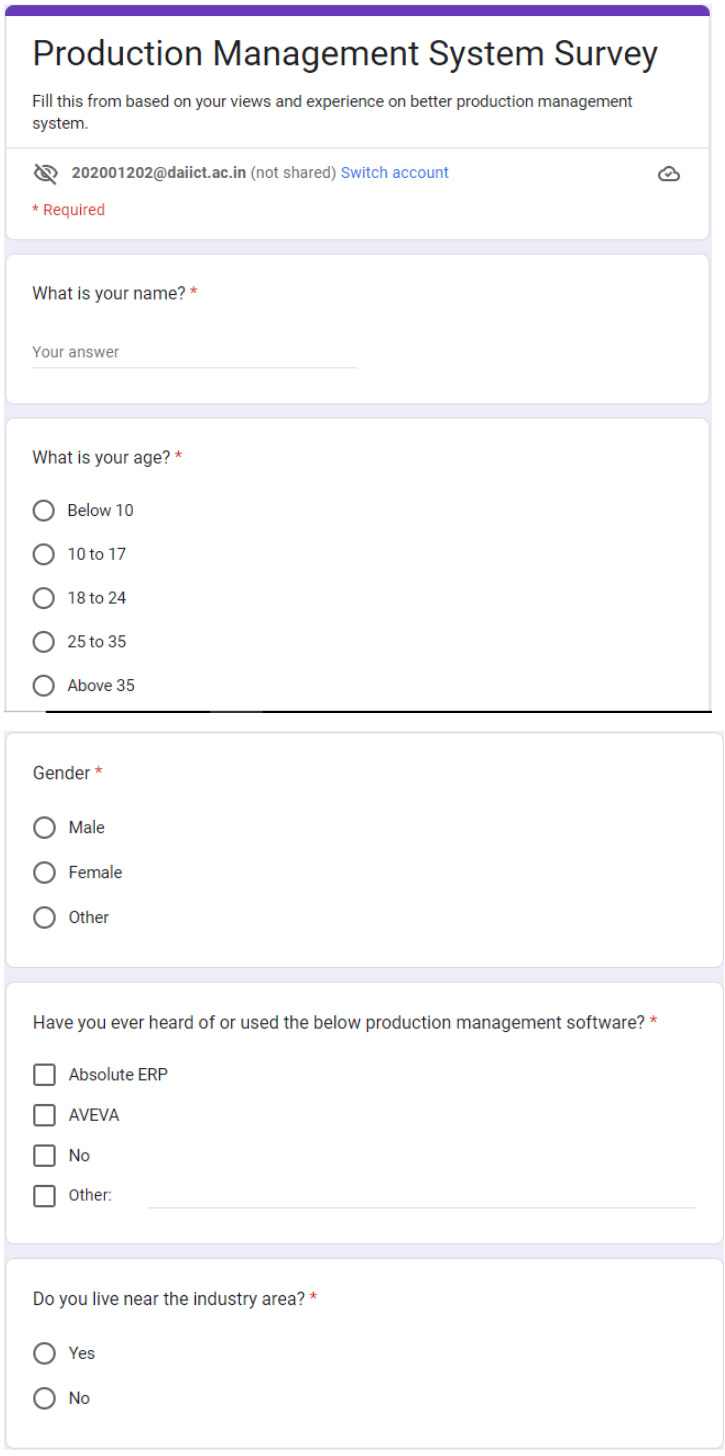
– Given suggestions about improving the system for access details whenever they want.
-Factory has details about availability of raw materials stocks.

2.2.1 List of the combined Requirements from Interviews:

– From the above interviews, we get to know that there is no such proper management system for factories which can solve the problems which are faced by management authorities and employees.
– Also, it is necessary to manage and update the available stock of products and Raw materials which are needed for producing products.
– Maintaining the records of employee details, sell details, buyers details.supplier details and record details about Raw materials which fcatory needed.
– Provide ease in analyzing production of the product and calculating the growth of the factory day by day.

2.3 Questionnaire Survey

Questions:



Production Management System Survey

Fill this from based on your views and experience on better production management system.

202001202@daiict.ac.in (not shared) Switch account

* Required

What is your name? *

Your answer

What is your age? *

- Below 10
- 10 to 17
- 18 to 24
- 25 to 35
- Above 35

Gender *

- Male
- Female
- Other

Have you ever heard of or used the below production management software? *

- Absolute ERP
- AVEVA
- No
- Other:

Do you live near the industry area? *

- Yes
- No

Did you ever visit any Factory related to Daily life or Home appliances products? *

- ◯ Yes
- ◯ No

If yes, then write then which type of product they produce.

Your answer

Have you ever work as _____ in production Department. *

- ◯ Sales person
- ◯ Management authority
- ◯ Employee
- ◯ Distributer
- ◯ No
- ◯ Other:

Have you ever bought products direct from Factory? *

	1	2	3	4	5	
Never	◯	◯	◯	◯	◯	Frequently

From your perspective, Who can be a significant consumer of Factory products? *

- ☐ Distributers
- ☐ Shop Owners
- ☐ Small Business dependent on factory product
- ☐ Other:

According to you which are main components of the Factory? *

- ☐ Buyers
- ☐ Suppliers of the products
- ☐ Management Authority
- ☐ Employees
- ☐ Supplier of the Raw Materials
- ☐ Other:

As per your perspective, Factory growth is dependent on Which Factors? *

- [ ] Consumer preference in market
- [ ] Quality of product
- [ ] Production capacity
- [ ] Mental and physical strength of employees
- [ ] Other: _____

What type of data should production management authorities have? *
(If you have any other suggestions, do consider writing in another field.)

- [ ] Employees data
- [ ] Buyers and Raw material supplier data
- [ ] Sales data
- [ ] Department data
- [ ] Machines data
- [ ] Other: _____

As per your Observation What are the limitations you seen in Factory management system? *

- [ ] Machine Failures
- [ ] Working efficiency of workers
- [ ] Power Failures
- [ ] Other: _____

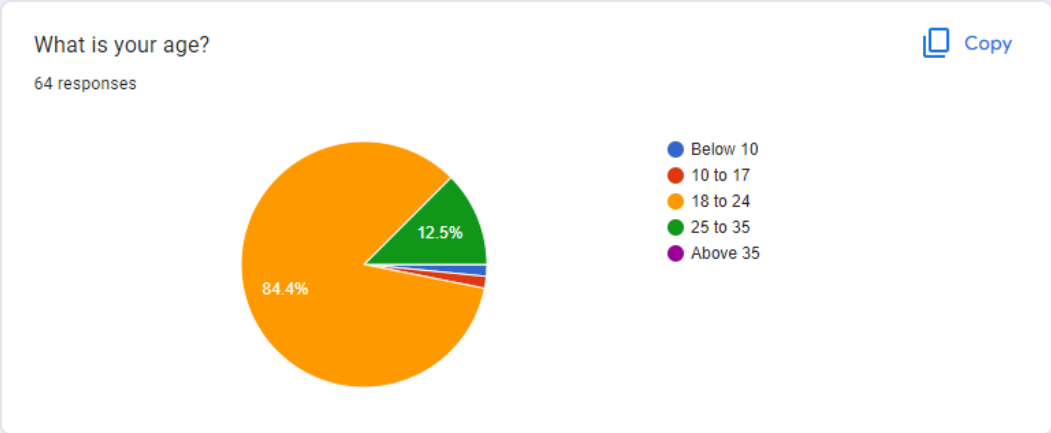
What can be included in factory daily cost? *

- [ ] Employees salary
- [ ] Raw material
- [ ] Power consumption
- [ ] Machines maintenance
- [ ] Other: _____

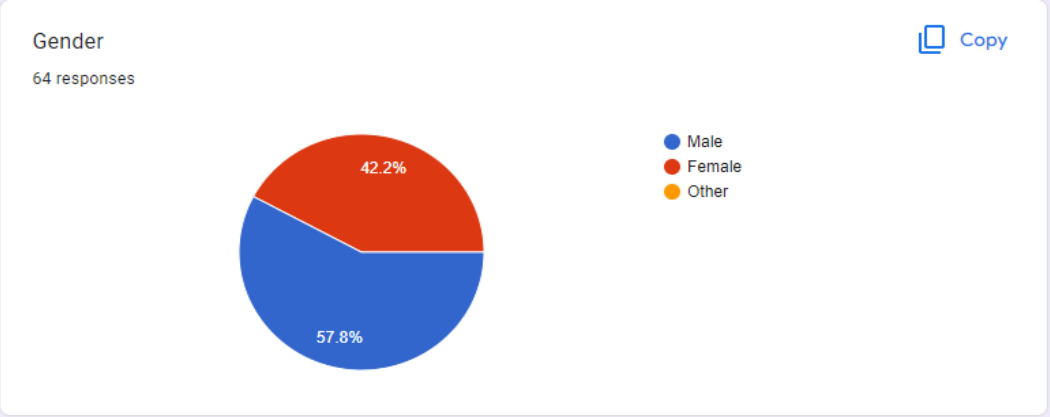
Suggestions(if any) for Production Management System

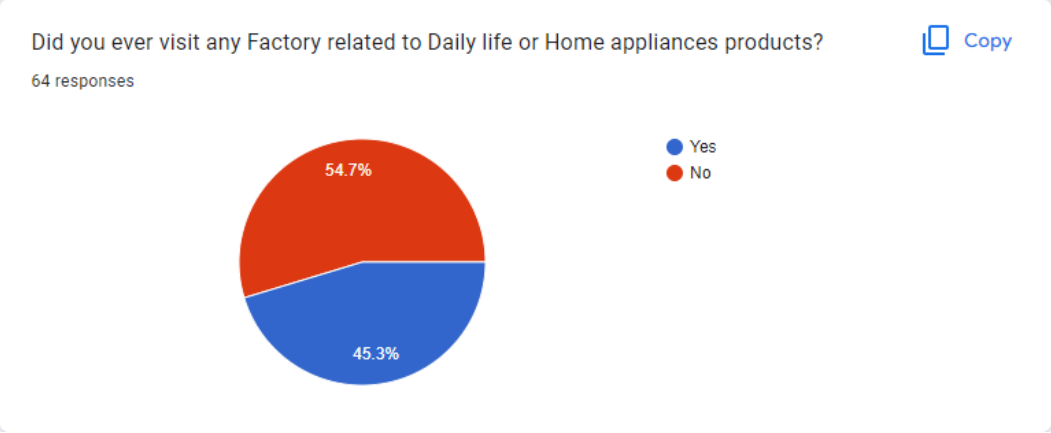
Your answer

Responses we get:

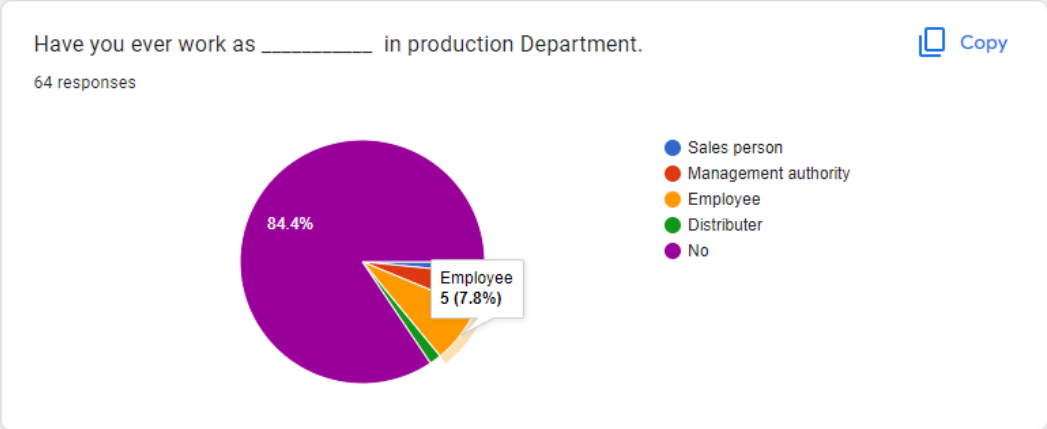


– Major age group is 18-24.

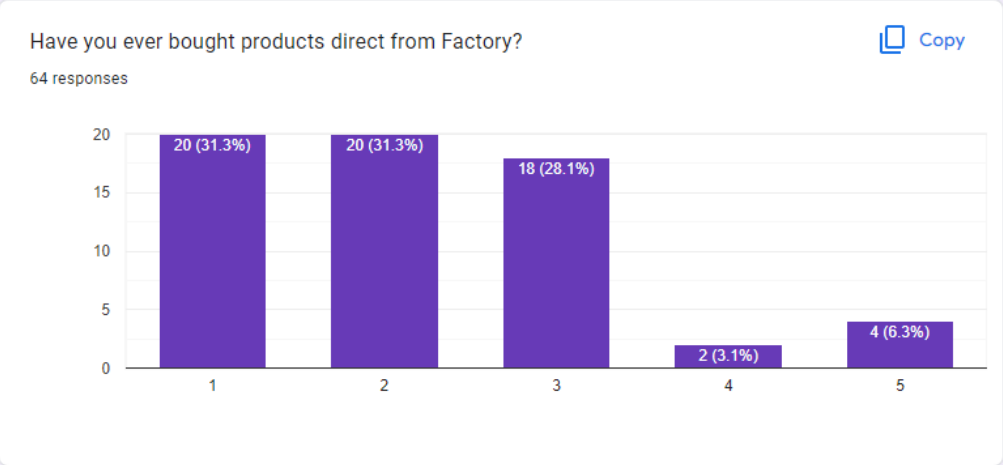




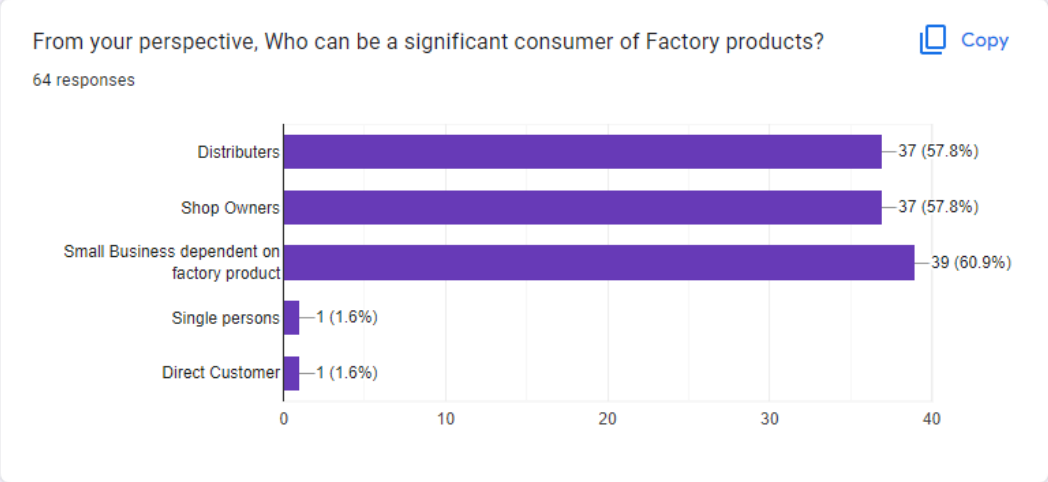
– As per survey people visit factories one time.



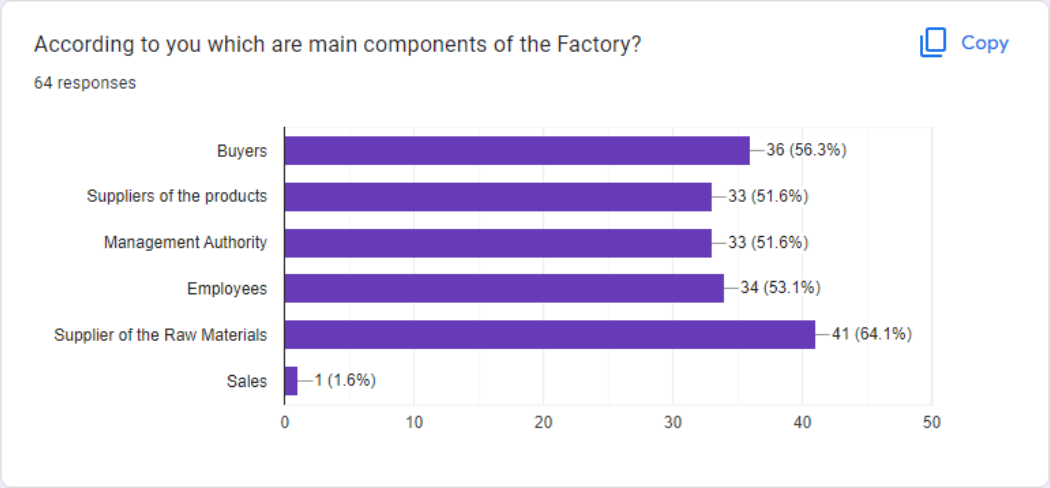
– Some people work as employees at production department.



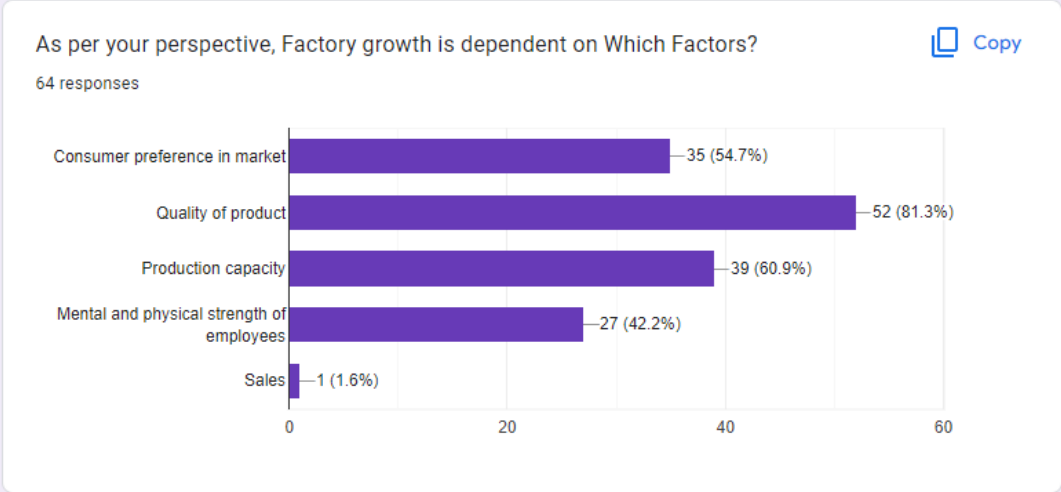
– As per survey, some people buy products from direct factories.



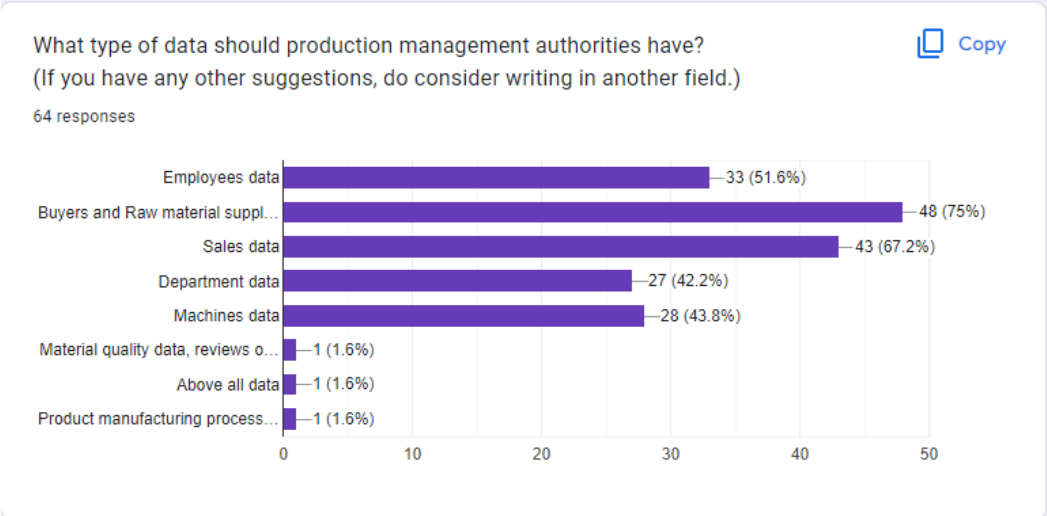
– As per the responses received major components are distributors, shop owners, small business etc.



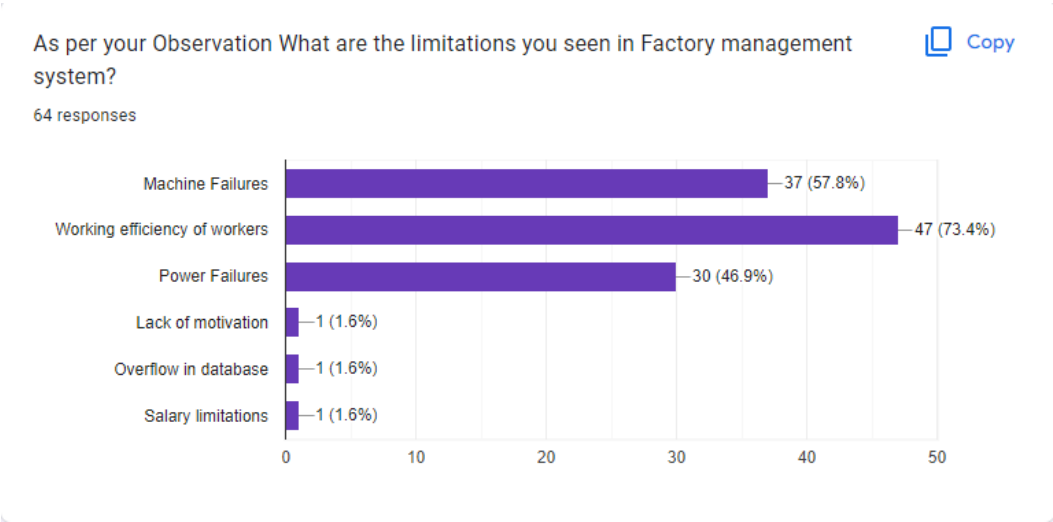
– Factory has main components like buyers,suppliers, management authorities,employees etc.



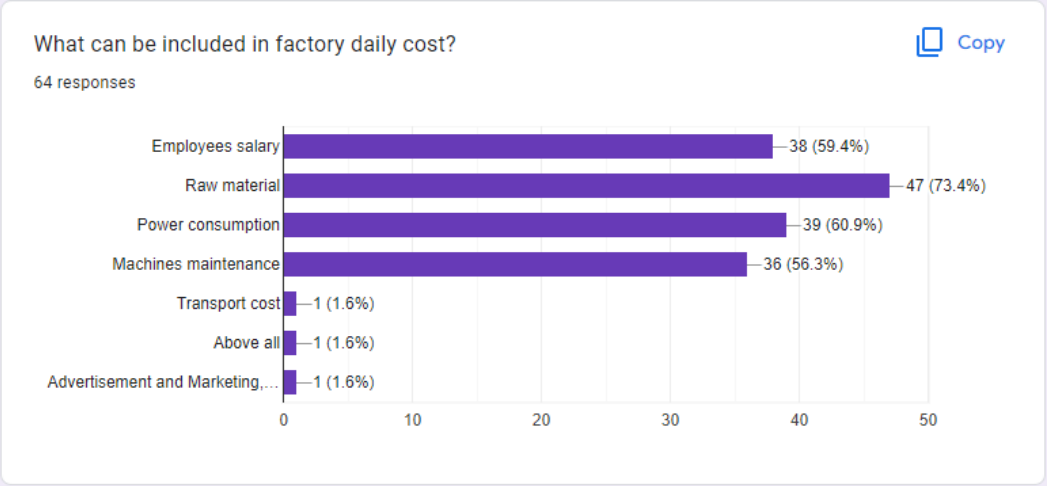
– As we can see in the above image all factors are dependent on factory growth.



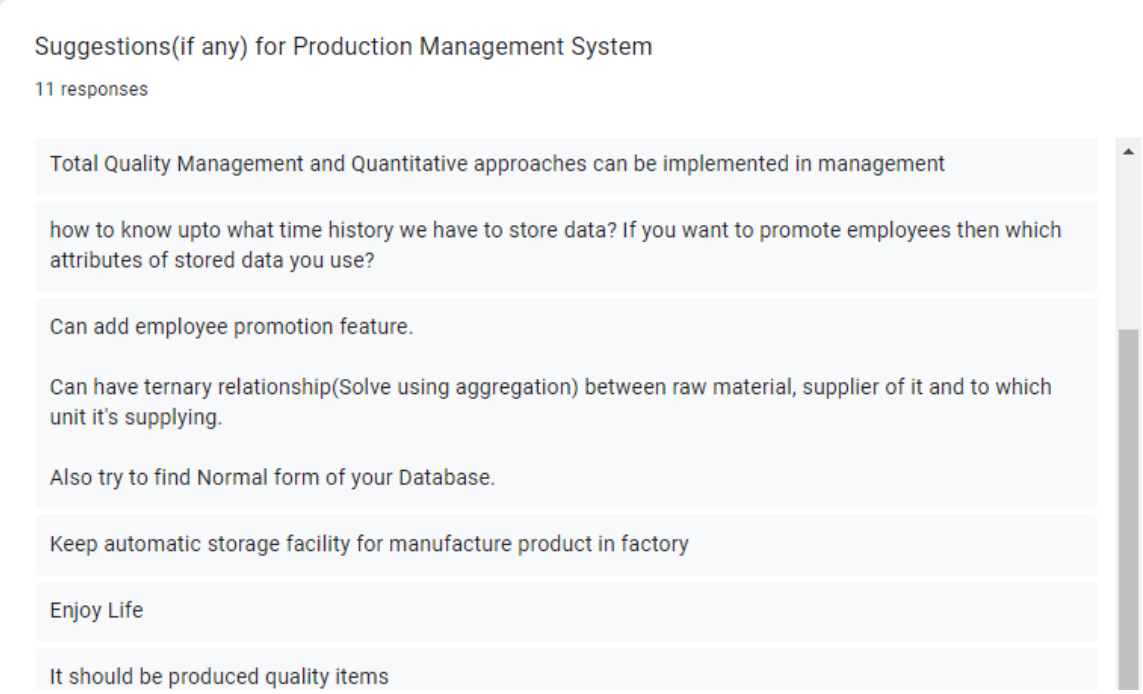
– As per above, the production management system authority has all the data about factories.



– There are many limitations in factory management system as per the responses to machine failures,power failures,working efficiency of workers.



– All of the above details are included in the daily cost of the factory.



– We are getting from some responses suggestions about how we can improve the production management system.

2.4 Observations

System : Production Management System

Observations by :

1. Nandani Tank (202001201)
2. Jaimin Satani (202001202)

Date : 29/09/2022
Time : 9:00
Duration : 45 minutes
Place : Maya Flour Mills

2.4.1 Observations Summary:

By visiting Maya Flour Mills and observing production management noticed by observers. Observers can see the actual work during visits or personal experiences to add/suggest more requirements which might have been missed by other techniques like background reading, interview, and questionnaires.

- As per the observation we can see that factories do not have proper database about employee details where employees can access their work details rather than asking an authority person for access details.
- Database can make work easy for management authority because it is easy to handle the data about employees,products etc.
- Factory does not have proper details about how much quantity stock of raw materials is available for producing the product.
- Database can make work easier because for each product we mention the stock of raw materials.
- Details about previous suppliers, buyers details are not available at their factory.using database we can get the details easily.

- For new employees and employees who leave the factory this update details are easy through the database.
- Any new security measures must not offend workers as well as employees.
- Should have restrictions on reading and manipulating data for different persons.
- Need to have data about daily production, their stocks, raw material stocks, suppliers, buyers, etc.
- Factory management system must have features about how much profit the factory gets in a given time interval and the expenses of the company.

2.4.2 List Of the Combined Requirements from Questionnaires Survey.

- A good quality of the product produced by the factory.
- The database should have records of each and every given details about the factory.
- Database should contain the details of growth of the factory in the current and previous year.
-Also, it is necessary to manage and update the available stock of products and Raw materials which are needed for producing products.

3. Fact Finding Chart

Objective	Technique	Subject(s)	Time Commitment
To get the background of the production factory & advertising industry	Background Readings	Company Reports, Tread journals	1 Day
To know Owner role in Production	Interview	Owner	0.5 hour
To know management authority role and their work	Interview	Manager	0.5 hour
To know the role of workers and difficulties they face	Interview	Workers at factory	1 hour
Storing data about sales and supplies, workers, machines, departments	Observation	Data present about sales and supplies at factory	0.25 hour
To know about current management system	Observation	People working for sales and supplies	0.5 day
Should have a proper system which is able to implement authorization	Interview	Owner and Manager	0.5 hour
Should give total value of profit and expenses for a time interval	Observation	Self (While preparing questions for interviews)	3 hour

4. List of Requirements

– Employee details and their working details and salary information which are easy to access for workers. (**By observation, Interview**)
– Management authority and employees should have separate roles to protect people's information. The database should be made in such a way that there will be roles assigned for every person which can assure the security of the database. (**Occurs at description, background reading**)
– Details of raw materials stock that are available at the factory for production. (**By observation, interview**)
– Factory needs the details about the day-to-day profit. (**By interview**)
– The production management system should have arrangements for 24x7 internet and power connectivity. (**Occurs at observation**)
– Each and every update in the database should be in real-time so that data queries can give real-time and efficient results. (**Background reading, Observation**)
– Sales records databases should not be modified by any person or ensure transparency in the process. (**Occurs at background readings, interviews, questionnaires**)
– Integrity constraint checking should be done in real-time of any modification to assure the integrity of the whole database. (**Occurs at interview, background reading**)
– There should be no redundant information in the database. (**Occurs at observations**)
– The database should be made in such a way that every query runs in the minimum possible time. (**Occurs at observations**).

5. User Categories and Privileges

- **List of the user categories:**
  1. Employee
  2. Management authority
  3. Buyers
  4. Suppliers

Privileges:

1. Employees

- Insertion
  – Not allowed
- Deletion
  – Not allowed
- Update
  – Update Mobile Number
  – Update email addredd
- Reading
  – View employee profile

2. Management Authority

- Insertion
  – New employee registration
  – Add new product
  – Add new machine
  – Add new department
  – Add new raw material
  – Add new supplier
  – Add new buyer
- Deletion
  – Omit old employee
  – Omit old product
  – Omit old machine
  – Omit old department
  – Omit new raw material
  – Omit new supplier

  - Omit new buyer
- Update
  - Update employee details
  - Update product details
  - Update product details
  - Update machine details
  - Update department details
  - Update raw material details
  - Update supplier details
  - Update buyer details
- Reading
  - employee details
  - product details
  - machine details
  - department details
  - raw material details
  - supplier details
  - buyer details
  - Sell details
  - Raw material buying details

3. Buyers

- Insertion
  - Can order products
- Deletion
  - Can delete order details
- Update
  - Can update order details
- Reading
  - Product details

4. Suppliers

- Insertion
  - Can order raw materials
- Deletion
  - Can cancel ordered raw materials
- Update
  - Can update ordered raw materials

- Reading
  – Product details
  – Raw materials details

6. Assumption

– All employees have their unique Employee ID and All employees should have passwords that are encrypted and cannot be hacked.
– In this database system we assume that all people are filling accurate information. And we also assume that This database provides an efficient solution to problems like system failure or power failure.
– All employees have their unique Employee ID and All employees should have passwords that are encrypted and cannot be hacked.
– All products have their unique Product ID.
– All Buyers have their unique Buyer ID.
– All Suppliers have their unique Supplier ID.
– All Departments have their unique Department ID.
– All raw materials have their unique Raw Material ID.
– Production of products doesn’t stop for any reason like power failure, machine failure, etc.
– There is enough memory available to store and process the database.
– Everyone has enough resources to access the database in their own respective roles.
– Applications of this database are completely handled by the respective Factory. i.e., website handling or mobile application handling in which this database is used.
– There are no unofficial employees working in Factory.
– There are no transport charges.
– Power consumption is fix for every machine.
– There will be no holiday in factory.

7. Business Constraints

– Limitless production of the product is not allowed.
– All requirements for production will always be fulfilled.
– There are no branches of the Factory.
– If any problem appears in the Factory then it is the responsibility of the management authority to handle the situation.
– There will be proper service for transport and transport fees will be paid by buyers.

8. Final Description:

This section includes details about what is and is not expected of the product management system, in addition to which cases are intentionally unsupported and assumptions that will be used in the creation of the production management system.

The Production Management System will provide a number of functionalities. We can provide neat and easy to use database by production management system to the factories and and production companies. By this system management, authority can track the growth of factory, details about employees, machines, departments, etc. Also we can provide track of Sell and Order details, and by use of that details, we can track the growth of factory, profit & loss in interval of time.

The Product Management System will provide a number of functionalities. Each is listed below:

- **Employees**
  – As we know that employees have data about their employee IDs, password, employee personal details, working department details, salary, working product details, working hours, working machine details, shift(morning, noon), the first day of joining date, Date of Birth, etc.
  – Employees work on particular machine and in particular department.
  – Also, Employees can edit some of the records like email address, and mobile numbers.
  – Here each employee should have a unique ID and password. So, that every employee can see their details on the factory portal.
  – Also, they can edit their details.
  – We are adding two functionalities:
    1. Auto-assign Employee ID to every employee
    2. Once any employee left the job His/Her data will be added to past employee details. So, that we can have details of all employees which had worked in a company.

- **Products**

  – Products will be produced by departments by use of machines.
  – So, machines also produce products.
  – It shows product details like the name of the product, product ID, weight, available stock of a product, department of the product, rate of the product per packet, produce flag, and production capacity per day.
  – If produce flag is on then production of that product will be allowed
  – Here we are auto assigning product ID
  – Also for calculate cost and profit by products we need to store date wise production of all products.
  – Also products are connected with raw materials. Because by use of Raw materials products were produced.
  – It shows product details like the name, ID, weight, available stock of a product, department of the product, rate of the product per packet, and production capacity per day.
  – Many employees will be working on ne product and by use of many machines in which products will be produced.

- **Machines**

  – Here, by observation, employees will work on machines and machines will produce products.
  – Also Department will use machines to produce products.
  – Here one machine will produce only one product but products will be produced by many machines.
  – Here many employees will work on one machine.
  – This data contains machine data like machine id, rpm(round per minute) of the machine, working condition of the machine (Description), department of the machine, service provided by the machine, price of the machine, maintenance cost per month, employee working on the machine, per day power consumption and date when the machine was planted.
  – Auto assigns machine ID to machines

- **Departments**

  – Here as mentioned above Departments will use machines to produce products.
  – Also employees will work in departments.

– One Department will have many employees and many machines.
– Department details contain department name, id, address, vacancies for jobs, machine capacity, machine vacancy, employee capacity.
– Here, we are Auto assigning department ID to departments.

- **Raw Materials**
  – Here as all know products will use raw materials.
  – Raw materials will be supplied by suppliers.
  – Raw material details contain data about raw material id, name, stock, price.
  – IDs will be auto-assigned.

- **Sell details**
  – Sell details will be used by management authority to calculate profit and track growth.
  – This contains data about the quantity of product sold by the factory who bought the product, the rate of the product buyer bought, date of the product when sold.

- **Buyer**
  – Contain buyer ID, password, buyer name, contact details, and email.
  – Buyers can buy products from the factory.

- **Supplier**
  – Contain supplier ID, supplier name, contact details, and email.
  – Suppliers can supply Raw materials to the factory.

- **Raw material order**
  – contain data about material ID, material name, supplier ID, the quantity of product bought from a supplier, price per kg, date and time of bought raw material.
  – Raw material order can be updated by suppliers.

- **Product - raw material connection**

  – As product and raw material have many to many relationships, we need to define one relationship that contains data about the relationship of raw material of the product and raw material.

  – It relates product ID with particular raw material ID and also contains data about for one packet of product how much raw material is needed(in kg).

– Keeping these suggestions in mind, we can design an efficient and user-friendly database that makes the process easier for all.

Assumptions:

– In this database system we assume that all people are filling accurate information. And we also assume that This database provides an efficient solution to problems like system failure or power failure.
– All employees have their unique Employee ID and All employees should have passwords that are encrypted and cannot be hacked.
– All products have their unique Product ID.
– All Buyers have their unique Buyer ID.
– All Suppliers have their unique Supplier ID.
– All Departments have their unique Department ID.
– All raw materials have their unique Raw Material ID.
– Production of products doesn't stop for any reason like power failure, machine failure, etc.
– There is enough memory available to store and process the database.
– Everyone has enough resources to access the database in their own respective roles.
– Applications of this database are completely handled by the respective Factory. i.e., website handling or mobile application handling in which this database is used.
– There are no unofficial employees working in Factory.
– There are no transport charges.
– Power consumption is fix for every machine.
– There will be no holiday in factory.

9. Noun & Verb Analysis

9.1 Extracted Nouns & Verbs from Problem Description

Noun	**Verb**
Section	**Include**
Product	Manage
System	Will
Functionalities	Provide
Employee	**Make**
Database	Use
Growth	Track
Employee	**Work**
Factory	Management
Authority	Can
Employee ID	Assign
Employee	Edit
System	Provide
Employee	Produce
We	Know
Employee	Have
Department	Working
Sell	Track
Employee	Have
Product	Working

Date	Joining
Machine	Working
Employee	Work
Date	Birth
Auto	Assign
Employee	Left
Data	Add
Product	Will
Department	Produce
Machine	**Use**
Cost	Calculate
Product	Need
Machine	Produce
Date	Store
Stock	Available
Day-Wise	Production
Product	Connect
Raw Material	Use
Product	**Produce**
Department	Use
Data	Contain
Data	Like
Condition	Working
Service	Provide
Power	Consumption

Machine ID	Assign
Department	Will
ID	Will
Department	Have
Sell Details	Use
Authority	Calculate
Department	Contain
Product	Sold
Product	Bought
Department ID	Assign
Buyer	**Buy**
Supplier	**Supply**
Buyer	Contain
Product Quantity	need
Raw Material	Bought
Supplier	Contain
Order	Update
Raw material	Have
Relationship	Define
Product	Relate
Raw Material	Need
Suggestions	Keep
Database	Design
Process	Easier
Information	Fill

Database	Provide
Problems	Like
Power	Failure
Raw Materials	Give
Password	Encrypted
Reason	Like
Machine	Failure
Database	Store
Product	Insufficient
Contact	Record
Product	**Track**
Database	Access
Sell_ID	Include
Factory	Produce
Weight	
Price	
Production_limt	
Production_flag	
Machine_capacity	
Address	
Salary	
Email	
Quantity	
RPM	
Maintenenece Cost	

Description	
Time	
Password	
Employee_shift	
Stock	
Hours	
Mobile number	
Time Shift	

9.2 Accepted Nouns & Verbs

Candidate entity set	Candidate attribute set	Candidate relationship set
Employee	• Employee_ID • Employee_password • Employee_Fname • Employee_Lname • Employee_Salary • Employee_hours • Employee_shift • Employee_joining_Date • Employee_mobile_no • Employee_address • Employee_Email • Employee_Role	• Make
Product	• Product_ID • Product_name • Product_weight • Product_stock • Product_price • Product_Production_limit • Product_Production_flag	• Use • Make • Produce • Include • Track
Production detail	• Production_date • Production_quantity • Production_cost	• Track
Department	• Department_ID • Department_name • Department_Address • Department_capacity_Employee • Department_capacity_Machine • Department_power_consumption	• Produce
Machine	• Machine_ID • Machine_name • Machine_RPM • Machine_Maintenance_cost	• Produce

	• Machine_Discription • Machine_Price • Machine_Buying_Date	
Raw Materials	• Material_ID • Material_name • Material_Stock • Material_Price	• Use • Give
Sell	• Sell_ID • Sell_Quantity • Sell_Price • Sell_Date • Sell_time	• Include • Buy
Orde(Raw materials Buying Details)	• Order_ID • Order_Quantity • Order_price • Order_date	• Supply • Give
Supplier	• Supplier_ID • Supplier_name • Supplier_contact • Supplier_Email	• Supply
Buyer	• Buyer_ID • Buyer_Password • Buyer_name • Buyer_contact • Buyer_Email	• Buy

9.3 Rejected Nouns & Verbs List

Noun	Rejected Reason
Section	General
Product_ID	Attribute
System	General
Functionalities	Vague
Database	General
Growth	General
Factory	General
Authority	General
Employee ID	Attributes
We	Irrelevant
Sell	Attributes
Date	Attributes
Auto	Vague
Data	General
Cost	Attributes
Date	Attributes
Stock	Attributes
Day-Wise	Attributes
Data	General
Condition	Irrelevant
Service	Vague

Power	Attributes
Machine ID	Attributes
ID	General
Weight	Attributes
Price	Attributes
Production_limt	Attributes
Production_flag	Attributes
Machine_capacity	Attributes
Address	Attributes
Salary	Attributes
Email	Attributes
Quantity	Attributes
RPM	Attributes
Maintenenece Cost	Attributes
Description	Attributes
Time	Attributes
Password	Attributes
Employee_shift	Attributes
Stock	Attributes
Hours	Attributes
Mobile number	Attributes
Time Shift	Attributes
Authority	General
Department ID	Attributes
Product Quantity	Attributes

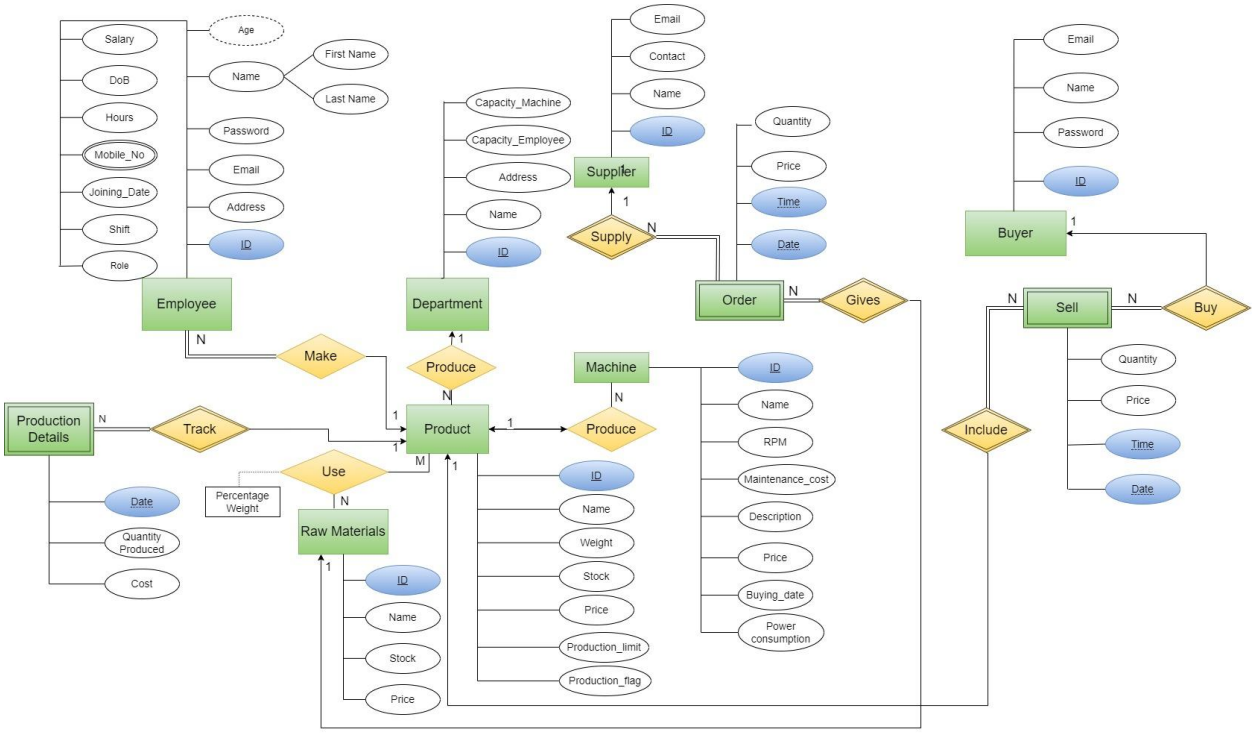
Relationship	General
Suggestions	Vague
Database	General
Process	Irrelevant
Information	General
Database	General
Problems	General
Power	Attributes
Password	Attributes
Reason	General
Contact	Attributes
Sell_ID	Attributes
Factory	General

Verb	**Reject Reason**
Manage	Irrelevant
Will	General
Provide	General
Management	vague
Can	General
Assign	Irrelevant
Edit	General
Know	Vague
Have	General
Joining	General

Should	General
Birth	Irrelevant
Left	Vague
Add	General
Will	General
Calculate	Irrelevant
Need	General
Store	Vague
Available	Vague
Production	General
Connect	Irrelevant
Contain	General
Like	Vague
Provide	General
Consumption	General
Will	General
Have	vague
need	General
Update	General
Have	General
Define	General
Relate	General
Keep	Vague
Design	General
Easier	vague

Fill	Irrelevant
Like	Vague
Store	General
Insufficient	General
Record	General
Access	General
work	General
Work on	duplicate

10. Entity Relationship Model



11. Mapping E-R Model to Relational Model:

1. **Departments (**
   Departement_ID,
   Department_Name,
   Department_Address,
   Department_Capacity_Employee,
   Department_Capacity_Machine
)

- In this relation Department_ID is PK.
- Department_ID is not null.

2. **Products** (
   Product_ID,
   ***Product_Department_ID*,**
   Product_Name,
   Product_Weight,
   Product_Stock,
   Product_Price,
   Product_Production_limit,
   Product_Production_flag
)

- In this relation Product_ID is PK.
- Product_ID and Department_ID are not null.

3. **Raw_Materials** (
   Raw_Material_ID,
   Raw_Material_Name,
   Raw_Material_Stock,
   Raw_Material_price
)

- In this relation Raw_Material_ID is PK.
- Raw_Material_ID is not null.

4. **Production_Details** (

Production_Details_Product_ID,

Production_Details_Date,

Production_Details_Quantity,

Production_Details_Cost

)

- IN this relation Production_Details_Product_ID and Production_Details_Date is PK.
- Production_Details_Product_ID and Production_Details_Date are not null.

5. **Used (**

Used_Product_ID,

Used_Raw_Material_ID,

Used_Percentage_weight

)

- In this relation Used_product_ID and Used_Raw_material_ID is composite PK.
- Used_product_ID is FK which references Product_ID in product table
- Used_Raw_Material_ID is FK which references the Raw_Materials table by Raw_Material_ID.
- We need this table because the Raw material to Product relationship is many to many relationships.

6. Machines (

Machine_ID,

Machine_Product_ID,

Machine_name,

Machine_rpm,

Machine_Maintenance_Cost,

Machine_Description,

Machine_Price,

Machine_Buying_Date,

Machine_Power_Consumption

)

- Machine_ID is PK.

- Machine_Product_IS is FK which references Product_ID in product table

7. Employees (

Employee_ID,
Employee_Product_ID,
Employee_password,
Employee_Fname,
Employee_Lname,
Employee_Salary,
Employee_hours,
Employee_shift,
Employee_joining_Date,
Employee_address,
Employee_Email
Employee_Role

)

- Employee_ID is PK.
- Employee_Product_ID is FK which references to Product_ID in Product table

8. Employee_mobile_no (

Employee_ID,
Employee_mobile_number

)

- We need this table for employee mobile number because employee mobile number is multivalued attribute

9. Suppliers(

Supplier_ID,
Supplier_name,
Supplier_contact,
Supplier_Email

)

- Supplier_ID is PK.

10. Order(

***Order_Raw_material_ID*,**
***Order_Supplier_ID*,**
Order_Date,
Order_Time,
Order_Price,
Order_Quantity

)

- Order_Raw_material_ID, Order_Supplier_ID, Order_Date and Order_Time, is composite PK
- Order_Raw_material_ID is FK references to Raw_Material_ID in Raw_Material table
- Order_Supplier_ID is FK references to Raw_Supplier_ID in Supplier table

11. **Buyers**(

Buyer_ID,
Buyer_Password,
Buyer_name,
Buyer_contact,
Buyer_Email

)

- Buyer_ID is PK.

12. **Sells**(

***Sell_Product_ID*,**
***Sell_Buyer_ID*,**
Sell_Date,
Sell_time,
Sell_Quantity,
Sell_Price

);

- Sell_product_ID, Sell_Buyer_ID, Sell_Date, Sell_time is composite PK.
- Sell_Product_ID is FK references to Product_ID in product table.
- Sell_Buyer_ID is FK references to Buyer_ID in Buyer table.
- Sell_product_ID, Sell_Buyer_ID,Sell_Date,Sell_time is not null.

→ All primary keys have not null & unique constraints.

***** Attribute -> Primary Key**
****** Attribute -> Primary Key and Foregin Key***
***** *Attribute* -> Foregin Key**

12. DDL Script:

```
CREATE TABLE "PMS"."Department"
(
    "Department_ID" INTEGER NOT NULL,
    "Department_Name" character varying(20),
    "Department_Address" character varying(100),
    "Department_Capacity_Employee" INTEGER,
    "Department_Capacity_Machine" INTEGER,
    PRIMARY KEY ("Department_ID"),
    CONSTRAINT "Department_ID" UNIQUE ("Department_ID"),
    CONSTRAINT "Department_ID_Check" CHECK
("Department_ID">100000 and "Department_ID"<999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Department"
    OWNER to postgres;

CREATE TABLE "PMS"."Product"
(
    "Product_ID" integer NOT NULL,
    "Product_Department_ID" integer,
    "Product_Name" character varying(20),
    "Product_Weight" integer,
    "Product_Stock" integer,
    "Product_Price" integer,
    "Product_Production_limit" integer,
    "Product_Production_flag" boolean,
    PRIMARY KEY ("Product_ID"),
    UNIQUE ("Product_ID"),
    FOREIGN KEY ("Product_Department_ID")
        REFERENCES "PMS"."Department" ("Department_ID") MATCH
SIMPLE
        ON UPDATE CASCADE
        ON DELETE NO ACTION
        NOT VALID,
    CHECK ("Product_ID">100000 and "Product_ID"<999999) NOT
VALID,
```

```
     CHECK ("Product_Department_ID">100000 and
"Product_Department_ID"<999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Product"
    OWNER to postgres;


CREATE TABLE "PMS"."Raw_Material"
(
    "Raw_Material_ID" integer NOT NULL,
    "Raw_Material_Name" character varying(20),
    "Raw_Material_Stock" integer,
    "Raw_Material_price" integer,
    PRIMARY KEY ("Raw_Material_ID"),
    UNIQUE ("Raw_Material_ID"),
    CHECK ("Raw_Material_ID">100000 and
"Raw_Material_ID"<999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Raw_Material"
    OWNER to postgres;


CREATE TABLE "PMS"."Used"
(
    "Used_Product_ID" integer NOT NULL,
    "Used_Raw_Material_ID" integer NOT NULL,
    "Used_Percentage_weight" integer,
    PRIMARY KEY ("Used_Product_ID", "Used_Raw_Material_ID"),
    UNIQUE ("Used_Product_ID", "Used_Raw_Material_ID"),
    FOREIGN KEY ("Used_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT
        NOT VALID,
    FOREIGN KEY ("Used_Raw_Material_ID")
        REFERENCES "PMS"."Raw_Material" ("Raw_Material_ID")
MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT
```

```
        NOT VALID,
    CHECK ("Used_Product_ID">100000 and
"Used_Product_ID"<999999) NOT VALID,
    CHECK ("Used_Raw_Material_ID">100000 and
"Used_Raw_Material_ID"<999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Used"
    OWNER to postgres;


CREATE TABLE "PMS"."Production_Detail"
(
    "Production_Details_Product_ID" integer NOT NULL,
    "Production_Details_Date" date NOT NULL,
    "Production_Details_Quantity" integer,
    "Production_Details_Cost" integer,
    PRIMARY KEY ("Production_Details_Product_ID",
"Production_Details_Date"),
    UNIQUE ("Production_Details_Product_ID",
"Production_Details_Date"),
    FOREIGN KEY ("Production_Details_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CHECK ("Production_Details_Product_ID">100000 and
"Production_Details_Product_ID"<999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Production_Detail"
    OWNER to postgres;


CREATE TABLE "PMS"."Machine"
(
    "Machine_ID" integer NOT NULL,
    "Machine_Product_ID" integer,
    "Machine_name" character varying(20),
    "Machine_rpm" integer,
    "Machine_Maintenance_Cost" integer,
```

```
    "Machine_Description" character varying(200),
    "Machine_Price" integer,
    "Machine_Buying_Date" date,
    "Machine_Power_Consumption" integer,
    PRIMARY KEY ("Machine_ID"),
    UNIQUE ("Machine_ID"),
    FOREIGN KEY ("Machine_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT
        NOT VALID,
    CHECK ("Machine_ID">100000 and "Machine_ID"<999999) NOT
VALID,
     CHECK ("Machine_Product_ID">100000 and
"Machine_Product_ID"<999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Machine"
    OWNER to postgres;


CREATE TABLE "PMS"."Employees"
(
    "Employee_ID" integer NOT NULL,
    "Employee_Product_ID" integer,
    "Employee_password" character varying(20),
    "Employee_Fname" character varying(20),
    "Employee_Lname" character varying(20),
    "Employee_Salary" integer,
    "Employee_hours" integer,
    "Employee_shift" boolean,
    "Employee_joining_Date" date,
    "Employee_address" character varying(100),
    "Employee_Email" character varying(30),
    "Employee_Role" character varying(20),
    PRIMARY KEY ("Employee_ID"),
    UNIQUE ("Employee_ID"),
    FOREIGN KEY ("Employee_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT
```

```
        NOT VALID,
    CHECK ("Employee_ID">100000 and "Employee_ID"<999999) NOT
VALID,
    CHECK ("Employee_Product_ID">100000 and
"Employee_Product_ID"<999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Employees"
    OWNER to postgres;


CREATE TABLE "PMS"."Employee_mobile_no"
(
    "Employee_ID" integer NOT NULL,
    "Employee_mobile_number" integer NOT NULL,
    FOREIGN KEY ("Employee_ID")REFERENCES "PMS"."Employees"
("Employee_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT
        NOT VALID,
    CHECK ("Employee_ID">100000 and "Employee_ID"<999999) NOT
VALID,
    CHECK ("Employee_mobile_number">1000000000 and
"Employee_mobile_number"<9999999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Employee_mobile_no"
    OWNER to postgres;


CREATE TABLE "PMS"."Supplier"
(
    "Supplier_ID" integer NOT NULL,
    "Supplier_name" character varying(20),
    "Supplier_contact" integer,
    "Supplier_Email" character varying(30),
    PRIMARY KEY ("Supplier_ID"),
    UNIQUE ("Supplier_ID"),
    CHECK ("Supplier_ID">100000 and "Supplier_ID"<999999) NOT
VALID,
```

```
    CHECK ("Supplier_contact">1000000000 and
"Supplier_contact"<9999999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Supplier"
    OWNER to postgres;


CREATE TABLE "PMS"."Order"
(
    "Order_Raw_material_ID" integer NOT NULL,
    "Order_Supplier_ID" integer NOT NULL,
    "Order_Date" date NOT NULL,
    "Order_Time" time with time zone NOT NULL,
    "Order_Price" integer,
    "Order_Quantity" integer,
    PRIMARY KEY ("Order_Raw_material_ID", "Order_Supplier_ID",
"Order_Date", "Order_Time"),
    UNIQUE ("Order_Raw_material_ID", "Order_Date", "Order_Time",
"Order_Supplier_ID"),
    FOREIGN KEY ("Order_Raw_material_ID")
        REFERENCES "PMS"."Raw_Material" ("Raw_Material_ID")
MATCH SIMPLE
        ON UPDATE RESTRICT
        ON DELETE RESTRICT
        NOT VALID,
    FOREIGN KEY ("Order_Supplier_ID")
        REFERENCES "PMS"."Supplier" ("Supplier_ID") MATCH SIMPLE
        ON UPDATE RESTRICT
        ON DELETE RESTRICT
        NOT VALID,
    CHECK ("Order_Raw_material_ID">100000 and
"Order_Raw_material_ID"<999999) NOT VALID,
    CHECK ("Order_Supplier_ID">100000 and
"Order_Supplier_ID"<999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Order"
    OWNER to postgres;
```

```
CREATE TABLE "PMS"."Buyer"
(
    "Buyer_ID" integer NOT NULL,
    "Buyer_Password" integer,
    "Buyer_name" character varying(20),
    "Buyer_contact" integer,
    "Buyer_Email" character(30),
    PRIMARY KEY ("Buyer_ID"),
    UNIQUE ("Buyer_ID"),
    CHECK ("Buyer_ID">100000 and "Buyer_ID"<999999) NOT VALID,
    CHECK ("Buyer_contact">1000000000 and
"Buyer_contact"<9999999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Buyer"
    OWNER to postgres;


CREATE TABLE "PMS"."Sell"
(
    "Sell_Product_ID" integer NOT NULL,
    "Sell_Buyer_ID" integer NOT NULL,
    "Sell_Date" date NOT NULL,
    "Sell_time" time with time zone NOT NULL,
    "Sell_Quantity" integer,
    "Sell_Price" integer,
    PRIMARY KEY ("Sell_Product_ID", "Sell_Buyer_ID",
"Sell_Date", "Sell_time"),
    UNIQUE ("Sell_Product_ID", "Sell_Date", "Sell_time",
"Sell_Buyer_ID"),
    FOREIGN KEY ("Sell_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT
        NOT VALID,
    FOREIGN KEY ("Sell_Buyer_ID")
        REFERENCES "PMS"."Buyer" ("Buyer_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT
        NOT VALID,
```

```
    CHECK ("Sell_Buyer_ID">100000 and "Sell_Buyer_ID"<999999)
NOT VALID,
    CHECK ("Sell_Product_ID">100000 and
"Sell_Product_ID"<999999) NOT VALID
);

ALTER TABLE IF EXISTS "PMS"."Sell"
    OWNER to postgres;
```

13. Normalization and Schema Refinement

***** Attribute → Primary Key**
***** *Attribute* → Primary Key and Foregin Key**
***** *Attribute* → Foregin Key**

1. Departments (
    Departement_ID,
    Department_Name,
    Department_Address,
    Department_Capacity_Employee,
    Department_Capacity_Machine
)

- Primary Key: Department_ID
- Foreign Key: No
- Redundancy: No
- Anomaly:
    - Update:
      – If we change the Department_ID of a particular department then we also need to change the DepartMent_ID of products associated with that department in the Product table.
    - Delete:
      – If we delete a particular department then products associated with that department should be deleted first.
    - Insert:
      – No Insertion Anomaly
- Functional Dependencies:
  – {Department_ID} → {Department_Name, Department_Address, Department_Capacity_Employee, Department_Capacity_Machine}
- 1NF: No change because all the attributes are atomic.
- 2NF: No change there is only one primary attribute. All the other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So, there is no requirement for decomposition.

2. Products (

Product_ID,
Product_Department_ID,
Product_Name,
Product_Weight,
Product_Stock,
Product_Price,
Product_Production_limit,
Product_Production_flag

)

- Primary Key: Product_ID
- Foreign Key: Product_Department_ID (Department)
- Redundancy: No
- Anomaly:
    - Update:
      – If we change the Product_ID of a particular product then we also need to change the Product_ID of machines, Product_ID of employees, Product_ID of raw material relations, Product_ID in the Production Details table, and Product_ID in the Sell table.
    - Delete:
      – If we delete a particular product then employees, machines, and Raw Material relations associated with that product should be deleted first.
      – We are not deleting from Sell and Production Details tables because even if the product is deleted we need to calculate the profit and loss out of that data.
    - Insert:
      – If we want to insert a new product then the department associated with that product should be present in the department table.
- Functional Dependencies:
  – {Product_ID} → {Product_Department_ID, Product_Name, Product_Weight, Product_Stock, Product_Price, Product_Production_limit, Product_Production_flag}
- 1NF: No change because all the primary attributes are atomic.
- 2NF: No change there is only one primary attribute. All other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So, there is no requirement for decomposition.

3. Raw_Materials (
Raw_Material_ID,
Raw_Material_Name,
Raw_Material_Stock,
Raw_Material_price
)

- Primary Key: Raw Material_ID
- Foreign Key: No
- Redundancy: No
- Anomaly:
    - Update:
      – If we change the Raw Material_ID of a particular product then we also need to change the Raw_Material_ID of related products, and Raw_Material_ID in the Order table.
    - Delete:
      – If we delete a particular Raw material then relations of those Products associated with that raw material should be deleted first.
      – We are not deleting from the Order table because even if the raw material is deleted we need to calculate the profit and loss out of that data.
    - Insert:
      – No Insertion Anomaly
- Functional Dependencies:
  – {Raw_Material_ID} → {Raw_Material_Name, Raw_Material_Stock, Raw_Material_price}
- 1NF: No change because all the primary attributes are atomic.
- 2NF: No change there is only one primary attribute. All other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So, there is no requirement for decomposition.

4. Production_Details (

Production_Details_Product_ID,
Production_Details_Date,
Production_Details_Quantity,
Production_Details_Cost

)

- Primary Key: {Production_Details_Product_ID, Production_Details_Date}
- Foreign Key: Production_Details_Product_ID (Product)
- Redundancy: No
- Anomaly:
    - Update:
      – No updates are allowed in this table.
    - Delete:
      – No deletions are allowed in this table.
    - Insert:
      – This table is used to track daily production of products.
      – So, if we want to insert data for a new day then all products should be present in the Product table.
- Functional Dependencies:
  – {Production_Details_Product_ID, Production_Details_Date} → {Production_Details_Quantity, Production_Details_Cost}
- 1NF: No change because all the primary attributes are atomic.
- 2NF: No change there is only one primary attribute. All other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So, there is no requirement for decomposition.

5. Used (

***Used_Product_ID*,**

***Used_Raw_Material_ID*,**

Used_Percentage_weight

)

- Primary Key: {Used_Product_ID, Used_Raw_Material_ID}
- Foreign Key: Used_Product_ID (Product), Used_Raw_Material_ID (Raw Material)
- Redundancy: No
- Anomaly:
    - Update:
      – All updates in this table will be done automatically as Product – Raw Material Relation is many to many.
    - Delete:
      – All deletions in this table will be done automatically as Product – Raw Material Relation is many to many.
    - Insert:
      – No Insertion Anomaly
- Functional Dependencies:
  – {Used_Product_ID, Used_Raw_Material_ID} → {Used_Percentage_weight}
- 1NF: No change because all the primary attributes are atomic.
- 2NF: No change there is only one primary attribute. All other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So, there is no requirement for decomposition.

*** This table defines many to many relationship between Raw Materials and Products.

6. Machines (

Machine_ID,
Machine_Product_ID,
Machine_name,
Machine_rpm,
Machine_Maintenance_Cost,
Machine_Description,
Machine_Price,
Machine_Buying_Date,
Machine_Power_Consumption

)

- Primary Key: Machine_ID
- Foreign Key: Machine_Product_ID (Product)
- Redundancy: No
- Anomaly:
    - Update:
      – No Update Anomaly
    - Delete:
      – No Deletion Anomaly
    - Insert:
      – If we want to insert a new machine then the product associated with that machine should be present in the Product table.

- Functional Dependencies:
  – {Machine_ID} → {*Machine_Product_ID*, Machine_name, Machine_rpm, Machine_Maintenance_Cost, Machine_Description, Machine_Price, Machine_Buying_Date, Machine_Power_Consumption}
- 1NF: No change because all the attributes are atomic.
- 2NF: No change there is only one primary attribute. All the other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So, there is no requirement for decomposition.

7. **Employees** (

Employee_ID,
Employee_Product_ID,
Employee_password,
Employee_Fname,
Employee_Lname,
Employee_Salary,
Employee_hours,
Employee_shift,
Employee_joining_Date,
Employee_address,
Employee_Email
Employee_Role

)

- Primary Key: {Employee_ID}
- Foreign Key: Employee_Product_ID (Product)
- Redundancy: No
- Anomaly:
    - Update:
      – If we update Employee_ID then in Employee_Mobile_no table Employee_ID of mobile numbers associated with that Employee_ID should be updated first.
    - Delete:
      – If we delete Employee_ID then in Employee_Mobile_no table Employee_ID of mobile numbers associated with that Employee_ID should be deleted first.
    - Insert:
      – If we want to insert a new employee then the product associated with that employee should be present in the Product table.
- Functional Dependencies:
  – {Employee_ID} → {Employee_Product_ID, Employee_password, Employee_Fname, Employee_Lname, Employee_Salary, Employee_hours, Employee_shift, Employee_joining_Date, Employee_address, Employee_Email}
- 1NF: No change because all the attributes are atomic.
- 2NF: No change there is only one primary attribute. All the other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.

- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So, there is no requirement for decomposition.

8. Employee_mobile_no (

***Employee_ID*,**

Employee_mobile_number

)

- Primary Key: {Employee_ID, Employee_mobile_number}
- Foreign Key: Employee_ID (Employee)
- Redundancy: No
- Anomaly:
    - Update:
      – If we update Employee_ID then in Employee_Mobile_no table Employee_ID of mobile numbers associated with that Employee_ID should be updated first.
    - Delete:
      – If we delete Employee_ID then in Employee_Mobile_no table Employee_ID of mobile numbers associated with that Employee_ID should be deleted first.
    - Insert:
      – If we want to insert a new mobile number then the employee associated with that mobile number should be present in the Employee table.
- Functional Dependencies:
  – No
- 1NF: No change because all the attributes are atomic.
- 2NF: No change there is only one primary attribute. All the other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change there is no non-primary attributes. So, there is no requirement for decomposition.

9. Suppliers (

Supplier_ID,

Supplier_name,

Supplier_contact,

Supplier_Email

)

- Primary Key: {Supplier_ID}
- Foreign Key: No
- Redundancy: No
- Anomaly:
    - Update:

      – If we change the Supplier_ID of a particular product then we also need to change the Order_ID of related products.
    - Delete:

      – No Deletion Anomaly
    - Insert:

      – No Insertion Anomaly
- Functional Dependencies:

  {Supplier_ID} → {Supplier_name,Supplier_contact,Supplier_Email}
- 1NF: No change because all the attributes are atomic.
- 2NF: No change there is only one primary attribute. All the other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So, there is no requirement for decomposition.

10. Order (

***Order_Raw_material_ID*,**
***Order_Supplier_ID*,**
Order_Date,
Order_Time,
Order_Price,
Order_Quantity

)

- Primary Key: {Order_Raw_material_ID, Order_Supplier_ID, Order_Date, Order_Time}
- Foreign Key: Order_Raw_material_ID (Raw_Material), Order_Supplier_ID(Supplier)
- Redundancy: No
- Anomaly:
    - Update:
      – No updates are allowed in this table.
    - Delete:
      – No deletions are allowed in this table.
    - Insert:
      – If we want to insert a new order then the Raw material and Supplier associated with that order should be present in the Raw Material and Supplier tables.
- Functional Dependencies:
  {Order_Raw_material_ID, Order_Supplier_ID,Order_Date,Order_Time} → {Order_Price, Order_Quantity}
- 1NF: No change because all the primary attributes are atomic.
- 2NF: No change there is only one primary attribute. All other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So there is no requirement for decomposition.

11. Buyers (

Buyer_ID,
Buyer_Password,
Buyer_name,
Buyer_contact,
Buyer_Email

)

- Primary Key: {Buyer_ID}
- Foreign Key: No
- Redundancy: No
- Anomaly:
    - Update:
      – If we change the Buyer_ID of a particular product then we also need to change the Sell of related products.
    - Delete:
      – No Deletion Anomaly
    - Insert:
      – No Insertion Anomaly
- Functional Dependencies:
  {Buyer_ID}-->{Buyer_Password,Buyer_name,Buyer_contact,Buyer_Email}
- 1NF: No change because all the primary attributes are atomic.
- 2NF: No change there is only one primary attribute. All other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So there is no requirement for decomposition.

12. Sells (

***Sell_Product_ID*,**
***Sell_Buyer_ID*,**
Sell_Date,
Sell_time,
Sell_Quantity,
Sell_Price

)

- Primary Key: {Sell_Product_ID, Sell_Buyer_ID, Sell_Date, Sell_Time}
- Foreign Key: Order_Raw_material_ID(Raw_Material), Order_Supplier_ID(Supplier)
- Redundancy: No
- Anomaly:
    - Update:
      – No updates are allowed in this table.
    - Delete:
      – No deletions are allowed in this table.
    - Insert:
      – If we want to insert a new Sell then the Product and Buyer associated with that sell should be present in the Product and Buyer tables.
- Functional Dependencies:
  {Sell_Product_ID,Sell_Buyer_ID,Sell_Date,Sell_time} → {Sell_Quantity,Sell_Price}
- 1NF: No change because all the primary attributes are atomic.
- 2NF: No change there is only one primary attribute. All other attributes are dependent on the primary attribute.
- 3NF: No change there is no transitive dependency.
- BCNF: No change because all the non-primary attributes are dependent on primary attributes. So there is no requirement for decomposition.

14. Final Schema:

1. Departments (
    Departement_ID,
    Department_Name,
    Department_Address,
    Department_Capacity_Employee,
    Department_Capacity_Machine
)

2. Products (
    Product_ID,
    Product_Department_ID,
    Product_Name,
    Product_Weight,
    Product_Stock,
    Product_Price,
    Product_Production_limit,
    Product_Production_flag
)

3. Raw_Materials (
    Raw_Material_ID,
    Raw_Material_Name,
    Raw_Material_Stock,
    Raw_Material_price
)

4. Production_Details (
    Production_Details_Product_ID,
    Production_Details_Date,
    Production_Details_Quantity,
    Production_Details_Cost
)

5. Used (

***Used_Product_ID*,**

***Used_Raw_Material_ID*,**

Used_Percentage_weight

)

6. Machines (

Machine_ID,

Machine_Product_ID,

Machine_name,

Machine_rpm,

Machine_Maintenance_Cost,

Machine_Description,

Machine_Price,

Machine_Buying_Date,

Machine_Power_Consumption

)

7. Employees (

Employee_ID,

Employee_Product_ID,

Employee_password,

Employee_Fname,

Employee_Lname,

Employee_Salary,

Employee_hours,

Employee_shift,

Employee_joining_Date,

Employee_address,

Employee_Email

Employee_Role

)

8. Employee_mobile_no (

***Employee_ID*,**

Employee_mobile_number

)

9. Suppliers (

Supplier_ID,
Supplier_name,
Supplier_contact,
Supplier_Email

)

10. Order (

Order_Raw_material_ID,
Order_Supplier_ID,
Order_Date,
Order_Time,
Order_Price,
Order_Quantity

)

11. Buyers (

Buyer_ID,
Buyer_Password,
Buyer_name,
Buyer_contact,
Buyer_Email

)

12. Sells (

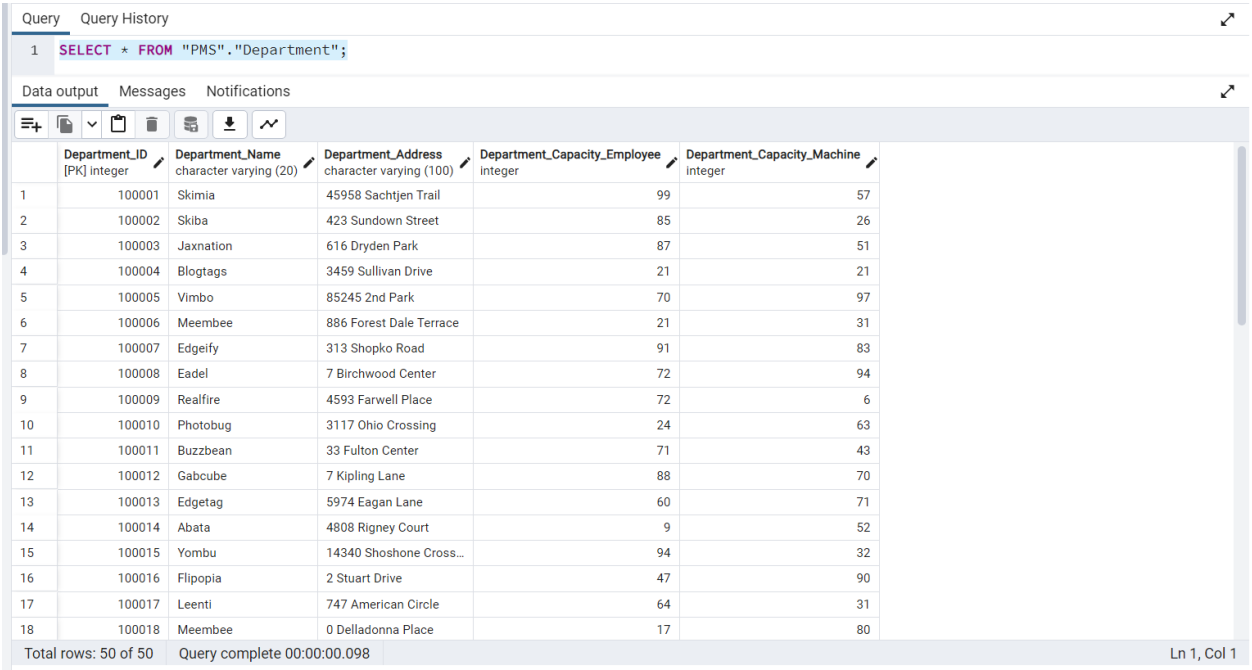
Sell_Product_ID,
Sell_Buyer_ID,
Sell_Date,
Sell_time,
Sell_Quantity,
Sell_Price

)

15.DDL Script with table snapshots:

1. Department:

```
-- Table: PMS.Department

-- DROP TABLE IF EXISTS "PMS"."Department";

CREATE TABLE IF NOT EXISTS "PMS"."Department"
(
    "Department_ID" integer NOT NULL,
    "Department_Name" character varying(20) COLLATE pg_catalog."default",
    "Department_Address" character varying(100) COLLATE pg_catalog."default",
    "Department_Capacity_Employee" integer,
    "Department_Capacity_Machine" integer,
    CONSTRAINT "Department_ID" PRIMARY KEY ("Department_ID"),
    CONSTRAINT "Department_ID_Check" CHECK ("Department_ID" > 100000 AND "Department_ID" <
999999)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Department"
    OWNER to postgres;
COPY "PMS"."Department" FROM 'E:\Table\Department.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Department";
```



Query | Query History

```
1  SELECT * FROM "PMS"."Department";
```

Data output | Messages | Notifications

	Department_ID [PK] integer	Department_Name character varying (20)	Department_Address character varying (100)	Department_Capacity_Employee integer	Department_Capacity_Machine integer
1	100001	Skimia	45958 Sachtjen Trail	99	57
2	100002	Skiba	423 Sundown Street	85	26
3	100003	Jaxnation	616 Dryden Park	87	51
4	100004	Blogtags	3459 Sullivan Drive	21	21
5	100005	Vimbo	85245 2nd Park	70	97
6	100006	Meembee	886 Forest Dale Terrace	21	31
7	100007	Edgeify	313 Shopko Road	91	83
8	100008	Eadel	7 Birchwood Center	72	94
9	100009	Realfire	4593 Farwell Place	72	6
10	100010	Photobug	3117 Ohio Crossing	24	63
11	100011	Buzzbean	33 Fulton Center	71	43
12	100012	Gabcube	7 Kipling Lane	88	70
13	100013	Edgetag	5974 Eagan Lane	60	71
14	100014	Abata	4808 Rigney Court	9	52
15	100015	Yombu	14340 Shoshone Cross...	94	32
16	100016	Flipopia	2 Stuart Drive	47	90
17	100017	Leenti	747 American Circle	64	31
18	100018	Meembee	0 Delladonna Place	17	80

Total rows: 50 of 50 | Query complete 00:00:00.098 | Ln 1, Col 1

2. Product:

```
-- Table: PMS.Product
CREATE TABLE IF NOT EXISTS "PMS"."Product"
(
    "Product_ID" integer NOT NULL,
    "Product_Department_ID" integer,
    "Product_Name" character varying(50) COLLATE pg_catalog."default",
    "Product_Weight" integer,
    "Product_Stock" integer,
    "Product_Price" integer,
    "Product_Production_limit" integer,
    "Product_Production_flag" boolean,
    CONSTRAINT "Product_pkey" PRIMARY KEY ("Product_ID"),
    CONSTRAINT "Product_Product_Department_ID_fkey" FOREIGN KEY ("Product_Department_ID")
        REFERENCES "PMS"."Department" ("Department_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT "Product_Product_ID_check" CHECK ("Product_ID" > 100000 AND "Product_ID" <
999999),
    CONSTRAINT "Product_Product_Department_ID_check" CHECK ("Product_Department_ID" > 100000 AND
"Product_Department_ID" < 999999)
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Product"
    OWNER to postgres;
COPY "PMS"."Product" FROM 'E:\Table\Product.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Product";
```

Query | Query History

```
1  SELECT * FROM "PMS"."Product";
```

Data output | Messages | Notifications

	Product_ID [PK] integer	Product_Department_ID integer	Product_Name character varying (50)	Product_Weight integer	Product_Stock integer	Product_Price integer	Product_Production_limit integer	Product_Production_flag boolean
1	100001	100013	Sausage - Chorizo	276	3300	898	57	true
2	100002	100031	Cheese - Ricotta	590	7970	955	37	true
3	100003	100047	Rice - Sushi	292	1311	11	16	true
4	100004	100018	Longos - Grilled Salm...	615	9963	652	32	true
5	100005	100038	Cheese - Parmigiano ...	450	9504	653	30	true
6	100006	100035	Nut - Macadamia	957	4556	523	32	true
7	100007	100012	Sour Puss Sour Apple	958	5088	189	73	true
8	100008	100001	Longos - Assorted Sa...	448	8530	682	6	true
9	100009	100049	Ecolab Silver Fusion	607	4075	304	1	true
10	100010	100022	Napkin White - Starch...	123	4159	840	62	true
11	100011	100048	Cinnamon Rolls	738	1764	318	15	true
12	100012	100022	Flour - Corn, Fine	854	1995	973	66	true
13	100013	100007	Soup - French Can Pea	354	4362	529	22	true
14	100014	100010	Beef - Rouladin, Sliced	716	8546	572	81	true
15	100015	100032	Mushroom - Morels, D...	583	9751	808	76	true
16	100016	100042	Sauce - Cranberry	288	6990	162	90	true
17	100017	100039	Juice - Apple, 1.36l	195	4225	998	13	true
18	100018	100009	Onions - Pearl	502	6914	999	99	true

Total rows: 50 of 50 | Query complete 00:00:00.101 | Ln 1, Col 1

3. Machine:

```
-- Table: PMS.Machine

-- DROP TABLE IF EXISTS "PMS"."Machine";
CREATE TABLE IF NOT EXISTS "PMS"."Machine"
(
    "Machine_ID" integer NOT NULL,
    "Machine_Product_ID" integer,
    "Machine_name" character varying(20) COLLATE pg_catalog."default",
    "Machine_rpm" integer,
    "Machine_Maintenance_Cost" integer,
    "Machine_Description" character varying(300) COLLATE pg_catalog."default",
    "Machine_Price" integer,
    "Machine_Buying_Date" date,
    "Machine_Power_Consumption" integer,
    CONSTRAINT "Machine_pkey" PRIMARY KEY ("Machine_ID"),
    CONSTRAINT "Machine_Machine_Product_ID_fkey" FOREIGN KEY ("Machine_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT "Machine_Machine_ID_check" CHECK ("Machine_ID" > 100000 AND "Machine_ID" <
999999),
    CONSTRAINT "Machine_Machine_Product_ID_check" CHECK ("Machine_Product_ID" > 100000 AND
"Machine_Product_ID" < 999999)
)
TABLESPACE pg_default;
ALTER TABLE IF EXISTS "PMS"."Machine"
    OWNER to postgres;
COPY "PMS"."Machine" FROM 'E:\Table\Machine.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Machine";
```

Query | Query History

```
1  SELECT * FROM "PMS"."Machine";
```

Data output | Messages | Notifications

	Machine_ID [PK] integer	Machine_Product_ID integer	Machine_name character varying (20)	Machine_rpm integer	Machine_Maintenance_Cost integer	Machine_Description character varying (300)	Machine_Price integer	Machine_Buying_Date date	Machine_Powe integer
1	100001	100034	Skid-Steer	53429	3830	Maecenas ut massa q...	385462	2022-11-06	
2	100002	100046	Dragline	48667	4808	Nullam sit amet turpis ...	940427	2022-10-07	
3	100003	100018	Bulldozer	49385	5058	Maecenas ut massa q...	94763	2022-11-24	
4	100004	100043	Dump Truck	51118	8367	Duis consequat dui ne...	42044	2022-11-12	
5	100005	100039	Bulldozer	21357	3246	In quis justo. Maecena...	29796	2022-11-24	
6	100006	100021	Scraper	10217	1373	Quisque id justo sit am...	2895	2022-10-27	
7	100007	100005	Backhoe	16977	2104	Morbi non lectus. Aliqu...	303092	2022-11-17	
8	100008	100017	Skid-Steer	35457	6415	Sed ante. Vivamus tort...	871744	2022-11-11	
9	100009	100026	Dragline	64316	1135	Phasellus in felis. Don...	730779	2022-10-03	
10	100010	100033	Trencher	9027	4539	Nullam porttitor lacus ...	871239	2022-10-01	
11	100011	100007	Skid-Steer	39190	8237	Phasellus sit amet erat...	813930	2022-11-08	
12	100012	100020	Bulldozer	46775	3595	Phasellus sit amet erat...	329547	2022-10-27	
13	100013	100045	Scraper	30059	4701	Curabitur gravida nisi a...	421488	2022-10-01	
14	100014	100047	Skid-Steer	35763	4019	Mauris enim leo, rhonc...	897576	2022-11-14	
15	100015	100035	Dump Truck	94616	7386	Integer tincidunt ante v...	302519	2022-10-22	
16	100016	100005	Compactor	82694	6383	Morbi non lectus. Aliqu...	877303	2022-11-01	
17	100017	100033	Dragline	82121	9554	Quisque porta volutpat...	345583	2022-10-10	

Total rows: 50 of 50 | Query complete 00:00:00.117 | Ln 1, Col 1

4. Employee:

```
-- Table: PMS.Employees
CREATE TABLE IF NOT EXISTS "PMS"."Employees"
(    "Employee_ID" integer NOT NULL,
    "Employee_Product_ID" integer,
    "Employee_password" character varying(20) COLLATE pg_catalog."default",
    "Employee_Fname" character varying(20) COLLATE pg_catalog."default",
    "Employee_Lname" character varying(20) COLLATE pg_catalog."default",
    "Employee_Salary" integer,
    "Employee_hours" integer,
    "Employee_shift" boolean,
    "Employee_joining_Date" date,
    "Employee_address" character varying(100) COLLATE pg_catalog."default",
    "Employee_Email" character varying(30) COLLATE pg_catalog."default",
    "Employee_Role" character varying(20) COLLATE pg_catalog."default",
    CONSTRAINT "Employees_pkey" PRIMARY KEY ("Employee_ID"),
    CONSTRAINT "Employees_Employee_Product_ID_fkey" FOREIGN KEY ("Employee_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT "Employees_Employee_ID_check" CHECK ("Employee_ID" > 100000 AND "Employee_ID" <
999999),
    CONSTRAINT "Employees_Employee_Product_ID_check" CHECK ("Employee_Product_ID" > 100000 AND
"Employee_Product_ID" < 999999)
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Employees"
    OWNER to postgres;
COPY "PMS"."Employee" FROM 'E:\Table\Employee.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Employee";
```

Query | Query History

```
1  SELECT * FROM "PMS"."Employees";
```

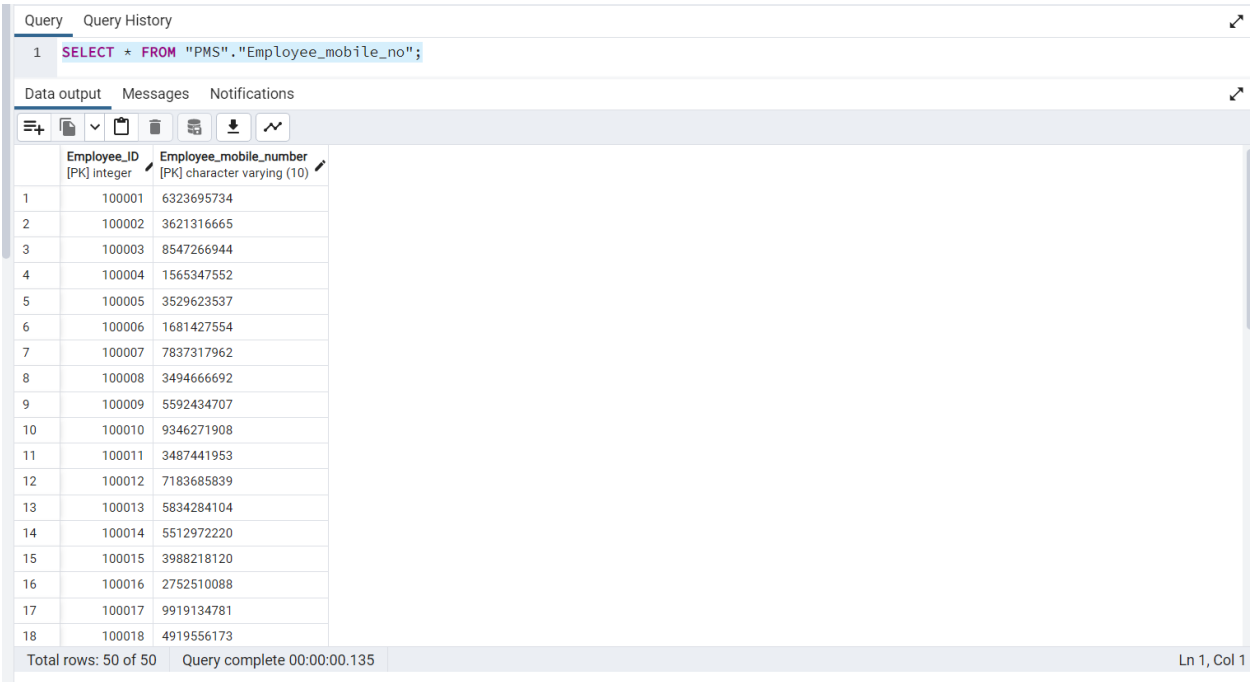
Data output | Messages | Notifications

	Employee_ID [PK] integer	Employee_Product_ID integer	Employee_password character varying (20)	Employee_Fname character varying (20)	Employee_Lname character varying (20)	Employee_Salary integer	Employee_hours integer	Employee_shift boolean	Employee_joining_Date date
1	100001	100009	eayndr	Currie	Daunay	52724	6	true	2022-10-26
2	100002	100013	lbLsBXU4Yh	Vanya	Bilt	85296	6	true	2022-11-25
3	100003	100016	B8eUfl	Godart	Burril	94238	6	true	2022-11-09
4	100004	100045	gThtjitFrRq	Gordon	Kitteringham	30018	6	true	2022-10-07
5	100005	100034	3IXyzm	Kinny	Pitcher	92157	6	true	2022-10-28
6	100006	100006	7NnheL	Faber	Ruseworth	43888	6	true	2022-11-18
7	100007	100013	0GchUn0iDA8F	Erastus	Catenot	48420	6	false	2022-11-14
8	100008	100042	KvqAW6oKZm	Raynard	Puvia	61073	6	true	2022-11-19
9	100009	100030	N5bQVR0y	Shepard	Overil	25448	6	false	2022-11-29
10	100010	100019	oUtRC4jbZ	Reynard	Paulack	55976	6	true	2022-10-31
11	100011	100028	PCxWjzGiZ5Ah	Roderich	MacKeeg	74761	6	false	2022-11-26
12	100012	100034	LT0rsF	Timothee	Rymmer	49860	6	false	2022-10-10
13	100013	100012	M9OjnNo	Butch	Atchly	33625	6	false	2022-10-25
14	100014	100037	yFL5xew1NFmP	Mateo	Philbin	52910	6	false	2022-10-03
15	100015	100010	k86r5iR	Dario	Nassie	87619	6	true	2022-10-10
16	100016	100006	XXBksMa	Tobit	Simcock	24352	6	true	2022-11-20
17	100017	100043	jidDl8c	Gregor	Schollick	91981	6	false	2022-10-21

Total rows: 50 of 50 | Query complete 00:00:00.113 | Ln 1, Col 1

5. Employee_Mobile_no:

```
-- Table: PMS.Employee_mobile_no

-- DROP TABLE IF EXISTS "PMS"."Employee_mobile_no";

CREATE TABLE IF NOT EXISTS "PMS"."Employee_mobile_no"
(
    "Employee_ID" integer NOT NULL,
    "Employee_mobile_number" character varying(10) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Employee_mobile_no_pkey" PRIMARY KEY ("Employee_ID", "Employee_mobile_number"),
    CONSTRAINT "Employee_mobile_no_Employee_ID_fkey" FOREIGN KEY ("Employee_ID")
        REFERENCES "PMS"."Employees" ("Employee_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT "Employee_mobile_no_Employee_ID_check" CHECK ("Employee_ID" > 100000 AND
"Employee_ID" < 999999),
    CONSTRAINT "Employee_mobile_no_Employee_mobile_number_check" CHECK
("Employee_mobile_number"::text > '1000000000'::text AND "Employee_mobile_number"::text <
'9999999999'::text)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Employee_mobile_no"
    OWNER to postgres;

COPY "PMS"."Employee_mobile_no" FROM 'E:\Table\Employee_mobile_no.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Employee_mobile_no";
```



Query | Query History

```
1  SELECT * FROM "PMS"."Employee_mobile_no";
```

Data output | Messages | Notifications

	Employee_ID [PK] integer	Employee_mobile_number [PK] character varying (10)
1	100001	6323695734
2	100002	3621316665
3	100003	8547266944
4	100004	1565347552
5	100005	3529623537
6	100006	1681427554
7	100007	7837317962
8	100008	3494666692
9	100009	5592434707
10	100010	9346271908
11	100011	3487441953
12	100012	7183685839
13	100013	5834284104
14	100014	5512972220
15	100015	3988218120
16	100016	2752510088
17	100017	9919134781
18	100018	4919556173

Total rows: 50 of 50 | Query complete 00:00:00.135 | Ln 1, Col 1

6. Production_detail:

```
-- Table: PMS.Production_Detail

-- DROP TABLE IF EXISTS "PMS"."Production_Detail";

CREATE TABLE IF NOT EXISTS "PMS"."Production_Detail"
(
    "Production_Details_Product_ID" integer NOT NULL,
    "Production_Details_Date" date NOT NULL,
    "Production_Details_Quantity" integer,
    "Production_Details_Cost" integer,
    CONSTRAINT "Production_Detail_pkey" PRIMARY KEY ("Production_Details_Product_ID",
"Production_Details_Date"),
    CONSTRAINT "Production_Detail_Production_Details_Product_ID_fkey" FOREIGN KEY
("Production_Details_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    CONSTRAINT "Production_Detail_Production_Details_Product_ID_check" CHECK
("Production_Details_Product_ID" > 100000 AND "Production_Details_Product_ID" < 999999)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Production_Detail"
    OWNER to postgres;

COPY "PMS"."Production_Detail" FROM 'E:\Table\Production_Detail.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Production_Detail";
```

Query | Query History

```
1  SELECT * FROM "PMS"."Production_Detail";
```

Data output | Messages | Notifications

	Production_Details_Product_ID [PK] integer	Production_Details_Date [PK] date	Production_Details_Quantity integer	Production_Details_Cost integer
1	100009	2022-10-02	2689	949
2	100001	2022-10-29	6127	660
3	100050	2022-11-27	2336	94
4	100042	2022-10-27	7716	474
5	100006	2022-11-20	6404	512
6	100008	2022-10-15	7825	294
7	100018	2022-10-06	2580	944
8	100015	2022-10-08	6819	318
9	100034	2022-10-17	2167	945
10	100011	2022-10-21	478	897
11	100045	2022-10-28	9396	430
12	100049	2022-11-15	3230	650
13	100009	2022-10-04	6309	954
14	100050	2022-10-10	27	971
15	100046	2022-11-12	3291	220
16	100026	2022-11-12	9361	830
17	100008	2022-11-23	376	972
18	100011	2022-11-05	8306	940

Total rows: 50 of 50 | Query complete 00:00:00.115 | Ln 1, Col 1

7. Raw_Material:

```
-- Table: PMS.Raw_Material

-- DROP TABLE IF EXISTS "PMS"."Raw_Material";

CREATE TABLE IF NOT EXISTS "PMS"."Raw_Material"
(
    "Raw_Material_ID" integer NOT NULL,
    "Raw_Material_Name" character varying(50) COLLATE pg_catalog."default",
    "Raw_Material_Stock" integer,
    "Raw_Material_price" integer,
    CONSTRAINT "Raw_Material_pkey" PRIMARY KEY ("Raw_Material_ID"),
    CONSTRAINT "Raw_Material_Raw_Material_ID_check" CHECK ("Raw_Material_ID" > 100000 AND
"Raw_Material_ID" < 999999)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Raw_Material"
    OWNER to postgres;

COPY "PMS"."Raw_Material" FROM 'E:\Table\Raw_Material.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Raw_Material";
```

Query | Query History

```
1  SELECT * FROM "PMS"."Raw_Material";
```

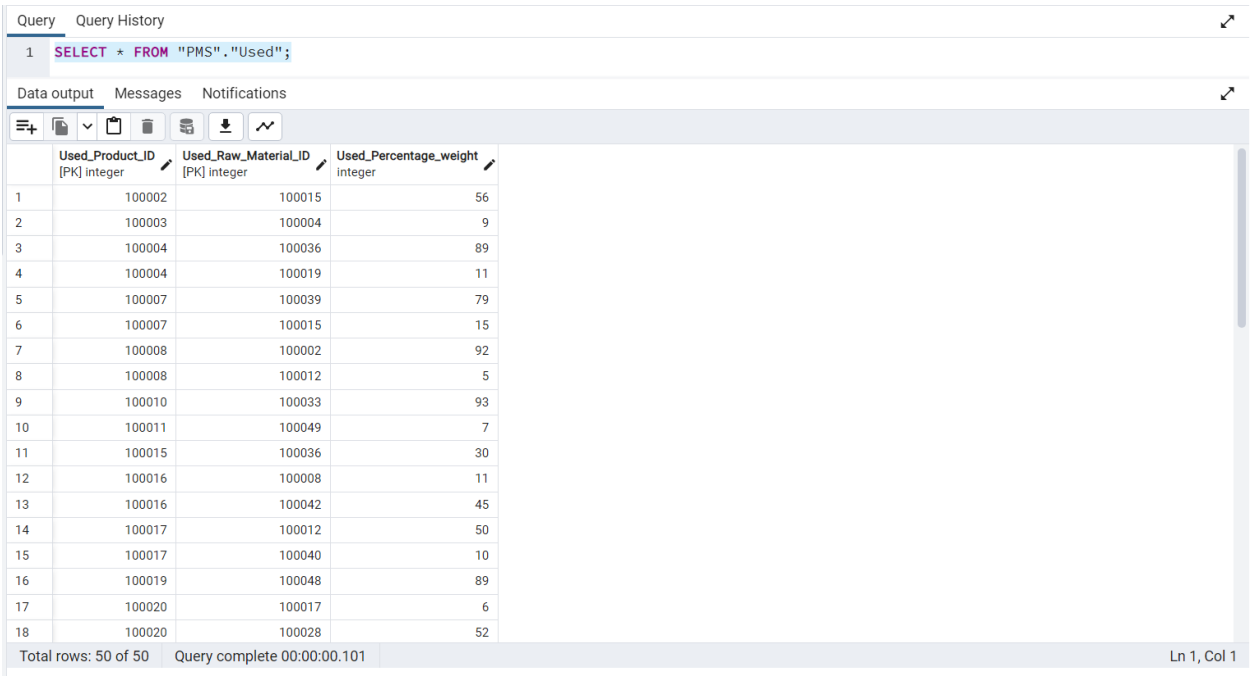
Data output | Messages | Notifications

	Raw_Material_ID [PK] integer	Raw_Material_Name character varying (50)	Raw_Material_Stock integer	Raw_Material_price integer
1	100001	Chocolate - Semi Swe…	39566	765
2	100002	Crab Brie In Phyllo	62144	681
3	100003	Cookies - Englishbay …	71745	289
4	100004	Sausage - Chorizo	90012	855
5	100005	Tea - Decaf 1 Cup	41159	326
6	100006	Coffee - Cafe Moreno	31649	934
7	100007	Beef - Bones, Cut - Up	16351	620
8	100008	Squash - Acorn	81512	742
9	100009	Langers - Cranberry C…	86966	936
10	100010	Cheese - Le Cru Du Cl…	90102	863
11	100011	Sauce - Hp	74043	591
12	100012	Black Currants	33341	571
13	100013	Mikes Hard Lemonade	39524	530
14	100014	Cheese - Brie,danish	59113	898
15	100015	Ecolab Crystal Fusion	81618	787
16	100016	Dome Lid Clear P920…	68817	567
17	100017	Beef - Sushi Flat Iron …	34993	257
18	100018	Pie Shell - 5	21816	492

Total rows: 50 of 50 | Query complete 00:00:00.177 | Ln 1, Col 1

8. Used:

```
-- Table: PMS.Used

CREATE TABLE IF NOT EXISTS "PMS"."Used"
(
    "Used_Product_ID" integer NOT NULL,
    "Used_Raw_Material_ID" integer NOT NULL,
    "Used_Percentage_weight" integer,
    CONSTRAINT "Used_pkey" PRIMARY KEY ("Used_Product_ID", "Used_Raw_Material_ID"),
    CONSTRAINT "Used_Used_Product_ID_fkey" FOREIGN KEY ("Used_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT "Used_Used_Raw_Material_ID_fkey" FOREIGN KEY ("Used_Raw_Material_ID")
        REFERENCES "PMS"."Raw_Material" ("Raw_Material_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT "Used_Used_Product_ID_check" CHECK ("Used_Product_ID" > 100000 AND
"Used_Product_ID" < 999999),
    CONSTRAINT "Used_Used_Raw_Material_ID_check" CHECK ("Used_Raw_Material_ID" > 100000 AND
"Used_Raw_Material_ID" < 999999)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Used"
    OWNER to postgres;

COPY "PMS"."Used" FROM 'E:\Table\Used.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Used";
```



9. Supplier:

```
-- Table: PMS.Supplier

-- DROP TABLE IF EXISTS "PMS"."Supplier";

CREATE TABLE IF NOT EXISTS "PMS"."Supplier"
(
    "Supplier_ID" integer NOT NULL,
    "Supplier_name" character varying(20) COLLATE pg_catalog."default",
    "Supplier_contact" character varying(10) COLLATE pg_catalog."default",
    "Supplier_Email" character varying(40) COLLATE pg_catalog."default",
    CONSTRAINT "Supplier_pkey" PRIMARY KEY ("Supplier_ID"),
    CONSTRAINT "Supplier_Supplier_ID_check" CHECK ("Supplier_ID" > 100000 AND "Supplier_ID" <
999999),
    CONSTRAINT "Supplier_Supplier_contact_check" CHECK ("Supplier_contact"::text >
'1000000000'::text AND "Supplier_contact"::text < '9999999999'::text)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Supplier"
    OWNER to postgres;

COPY "PMS"."Supplier" FROM 'E:\Table\Supplier.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Supplier";
```

Query | Query History

```
1  SELECT * FROM "PMS"."Supplier";
```

Data output | Messages | Notifications

	Supplier_ID [PK] integer	Supplier_name character varying (20)	Supplier_contact character varying (10)	Supplier_Email character varying (40)
1	100001	Franzen	2868581352	fikringill0@jalbum.net
2	100002	Chase	9432922883	cstapells1@naver.com
3	100003	Constantino	6023004525	cguittet2@cnn.com
4	100004	Faye	2082959632	fkisbee3@nationalge...
5	100005	Nelia	7345509982	nbriston4@altervista....
6	100006	Arlena	5921675101	alanghorn5@opensou...
7	100007	Aldous	7108483504	akitman6@usa.gov
8	100008	Moyra	6454550773	mharsum7@msu.edu
9	100009	Stanislaus	2967062321	scallinan8@youtube.c...
10	100010	Giacobo	2766737053	gbront9@mozilla.org
11	100011	Adriena	5883129015	abrigshawa@tumblr.c...
12	100012	Deirdre	1225111255	dgentileb@smh.com....
13	100013	Sephira	4167995831	sgodboltc@sogou.com
14	100014	Joly	8970853699	jbradockd@prweb.com
15	100015	Tommie	8851474206	tsedgeworthe@chica...
16	100016	Basil	2077177244	bpressmanf@home.pl
17	100017	Burton	1200858208	bmaveng@theglobea...
18	100018	Ruthi	9142166269	rlippitth@dot.gov

Total rows: 50 of 50 | Query complete 00:00:00.103 | Ln 1, Col 1

10. Order:

```
-- Table: PMS.Order

-- DROP TABLE IF EXISTS "PMS"."Order";

CREATE TABLE IF NOT EXISTS "PMS"."Order"
(
    "Order_Raw_material_ID" integer NOT NULL,
    "Order_Supplier_ID" integer NOT NULL,
    "Order_Date" date NOT NULL,
    "Order_Time" time with time zone NOT NULL,
    "Order_Price" integer,
    "Order_Quantity" integer,
    CONSTRAINT "Order_pkey" PRIMARY KEY ("Order_Raw_material_ID", "Order_Supplier_ID",
"Order_Date", "Order_Time"),
    CONSTRAINT "Order_Order_Raw_material_ID_Order_Date_Order_Time_Order_Sup_key" UNIQUE
("Order_Raw_material_ID", "Order_Date", "Order_Time", "Order_Supplier_ID"),
    CONSTRAINT "Order_Order_Raw_material_ID_fkey" FOREIGN KEY ("Order_Raw_material_ID")
        REFERENCES "PMS"."Raw_Material" ("Raw_Material_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    CONSTRAINT "Order_Order_Supplier_ID_fkey" FOREIGN KEY ("Order_Supplier_ID")
        REFERENCES "PMS"."Supplier" ("Supplier_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    CONSTRAINT "Order_Order_Raw_material_ID_check" CHECK ("Order_Raw_material_ID" > 100000 AND
"Order_Raw_material_ID" < 999999),
    CONSTRAINT "Order_Order_Supplier_ID_check" CHECK ("Order_Supplier_ID" > 100000 AND
"Order_Supplier_ID" < 999999)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Order"
    OWNER to postgres;

COPY "PMS"."Order" FROM 'E:\Table\Order.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Order";
```

Query | Query History

```
1  SELECT * FROM "PMS"."Order";
```

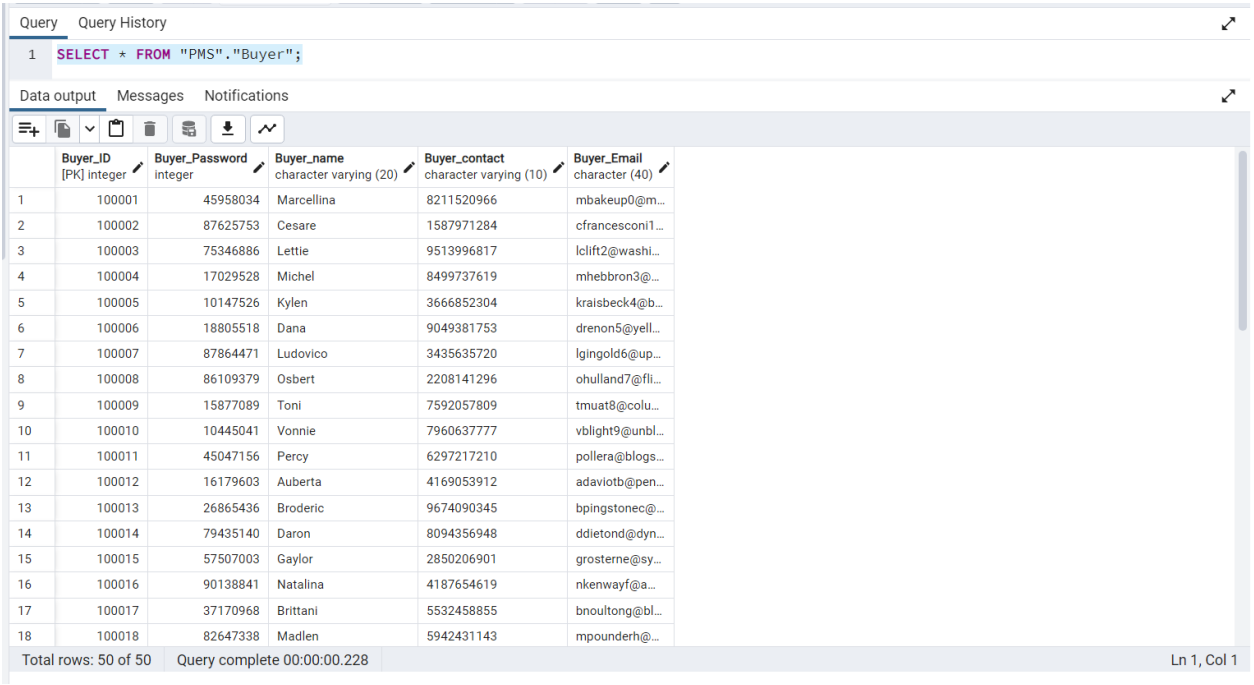
Data output | Messages | Notifications

	Order_Raw_material_ID [PK] integer	Order_Supplier_ID [PK] integer	Order_Date [PK] date	Order_Time [PK] time with time zone	Order_Price integer	Order_Quantity integer
1	100025	100045	2022-10-31	01:02:00+05:30	14477	83
2	100003	100024	2022-10-12	22:42:00+05:30	55580	248
3	100028	100043	2022-11-12	02:53:00+05:30	5763	46
4	100014	100018	2022-10-21	05:45:00+05:30	52786	191
5	100044	100028	2022-11-22	00:22:00+05:30	94651	258
6	100027	100036	2022-11-04	03:30:00+05:30	74370	378
7	100030	100041	2022-11-23	08:36:00+05:30	13375	22
8	100049	100050	2022-11-05	06:22:00+05:30	34403	197
9	100046	100049	2022-11-06	17:44:00+05:30	29632	389
10	100011	100020	2022-10-31	07:00:00+05:30	78778	47
11	100019	100041	2022-10-10	10:50:00+05:30	89815	88
12	100024	100035	2022-10-13	19:55:00+05:30	26124	479
13	100022	100034	2022-11-18	06:20:00+05:30	40769	269
14	100010	100018	2022-10-28	03:35:00+05:30	30286	289
15	100041	100011	2022-11-09	02:07:00+05:30	87618	411
16	100034	100041	2022-10-28	09:09:00+05:30	30526	118
17	100008	100007	2022-11-05	09:55:00+05:30	61398	20
18	100010	100048	2022-10-27	13:00:00+05:30	61637	135

Total rows: 50 of 50 Query complete 00:00:00.109 Ln 1, Col 1

11. Buyer:

```
-- Table: PMS.Buyer

-- DROP TABLE IF EXISTS "PMS"."Buyer";
CREATE TABLE IF NOT EXISTS "PMS"."Buyer"
(
    "Buyer_ID" integer NOT NULL,
    "Buyer_Password" integer,
    "Buyer_name" character varying(20) COLLATE pg_catalog."default",
    "Buyer_contact" character varying(10) COLLATE pg_catalog."default",
    "Buyer_Email" character(40) COLLATE pg_catalog."default",
    CONSTRAINT "Buyer_pkey" PRIMARY KEY ("Buyer_ID"),
    CONSTRAINT "Buyer_Buyer_ID_check" CHECK ("Buyer_ID" > 100000 AND "Buyer_ID" < 999999),
    CONSTRAINT "Buyer_Buyer_contact_check" CHECK ("Buyer_contact"::text > '1000000000'::text AND
"Buyer_contact"::text < '9999999999'::text)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "PMS"."Buyer"
    OWNER to postgres;

COPY "PMS"."Buyer" FROM 'E:\Table\Buyer.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Buyer";
```



Query | Query History

```
1  SELECT * FROM "PMS"."Buyer";
```

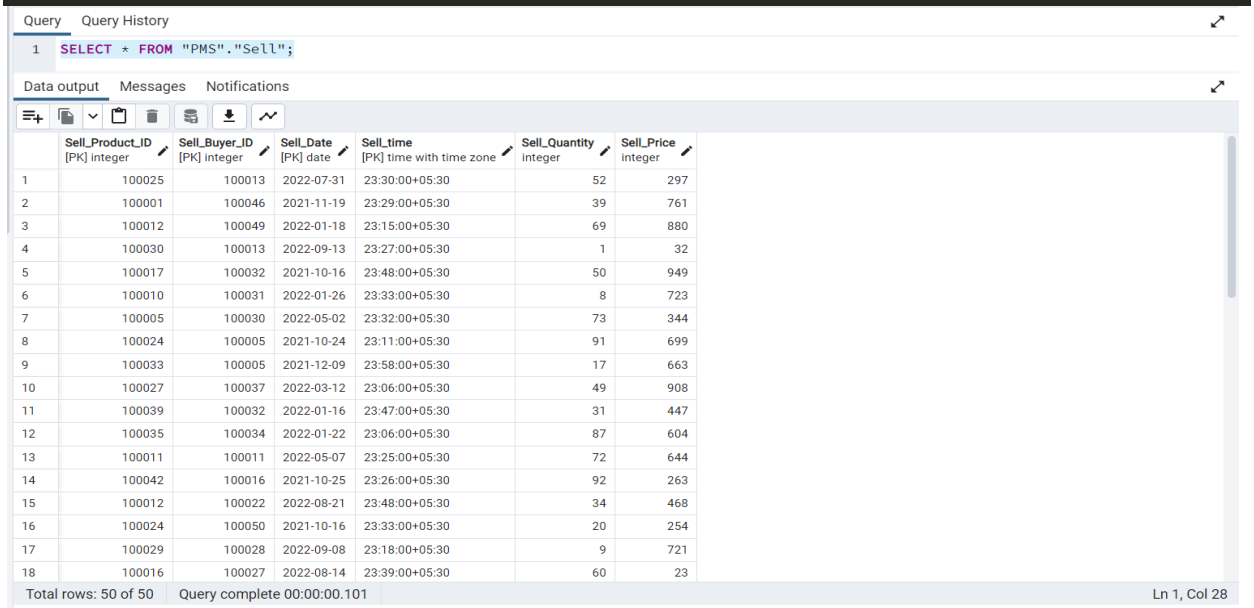
Data output | Messages | Notifications

	Buyer_ID [PK] integer	Buyer_Password integer	Buyer_name character varying (20)	Buyer_contact character varying (10)	Buyer_Email character (40)
1	100001	45958034	Marcellina	8211520966	mbakeup0@m...
2	100002	87625753	Cesare	1587971284	cfrancesconi1...
3	100003	75346886	Lettie	9513996817	lclift2@washi...
4	100004	17029528	Michel	8499737619	mhebbron3@...
5	100005	10147526	Kylen	3666852304	kraisbeck4@b...
6	100006	18805518	Dana	9049381753	drenon5@yell...
7	100007	87864471	Ludovico	3435635720	lgingold6@up...
8	100008	86109379	Osbert	2208141296	ohulland7@fli...
9	100009	15877089	Toni	7592057809	tmuat8@colu...
10	100010	10445041	Vonnie	7960637777	vblight9@unbl...
11	100011	45047156	Percy	6297217210	pollera@blogs...
12	100012	16179603	Auberta	4169053912	adaviotb@pen...
13	100013	26865436	Broderic	9674090345	bpingstonec@...
14	100014	79435140	Daron	8094356948	ddietond@dyn...
15	100015	57507003	Gaylor	2850206901	grosterne@sy...
16	100016	90138841	Natalina	4187654619	nkenwayf@a...
17	100017	37170968	Brittani	5532458855	bnoultong@bl...
18	100018	82647338	Madlen	5942431143	mpounderh@...

Total rows: 50 of 50 | Query complete 00:00:00.228 | Ln 1, Col 1

12. Sell:

```
-- Table: PMS.Sell
CREATE TABLE IF NOT EXISTS "PMS"."Sell"
("Sell_Product_ID" integer NOT NULL,
    "Sell_Buyer_ID" integer NOT NULL,
    "Sell_Date" date NOT NULL,
    "Sell_time" time with time zone NOT NULL,
    "Sell_Quantity" integer,
    "Sell_Price" integer,
    CONSTRAINT "Sell_pkey" PRIMARY KEY ("Sell_Product_ID", "Sell_Buyer_ID", "Sell_Date",
"Sell_time"),
    CONSTRAINT "Sell_Sell_Product_ID_Sell_Date_Sell_time_Sell_Buyer_ID_key" UNIQUE
("Sell_Product_ID", "Sell_Date", "Sell_time", "Sell_Buyer_ID"),
    CONSTRAINT "Sell_Sell_Buyer_ID_fkey" FOREIGN KEY ("Sell_Buyer_ID")
        REFERENCES "PMS"."Buyer" ("Buyer_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    CONSTRAINT "Sell_Sell_Product_ID_fkey" FOREIGN KEY ("Sell_Product_ID")
        REFERENCES "PMS"."Product" ("Product_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    CONSTRAINT "Sell_Sell_Buyer_ID_check" CHECK ("Sell_Buyer_ID" > 100000 AND "Sell_Buyer_ID" <
999999),
    CONSTRAINT "Sell_Sell_Product_ID_check" CHECK ("Sell_Product_ID" > 100000 AND
"Sell_Product_ID" < 999999)
)
TABLESPACE pg_default;
ALTER TABLE IF EXISTS "PMS"."Sell"
    OWNER to postgres;

COPY "PMS"."Sell" FROM 'E:\Table\Sell.csv' DELIMITER ',' CSV HEADER;
SELECT * FROM "PMS"."Sell";
```



Query | Query History

```
1  SELECT * FROM "PMS"."Sell";
```

Data output | Messages | Notifications

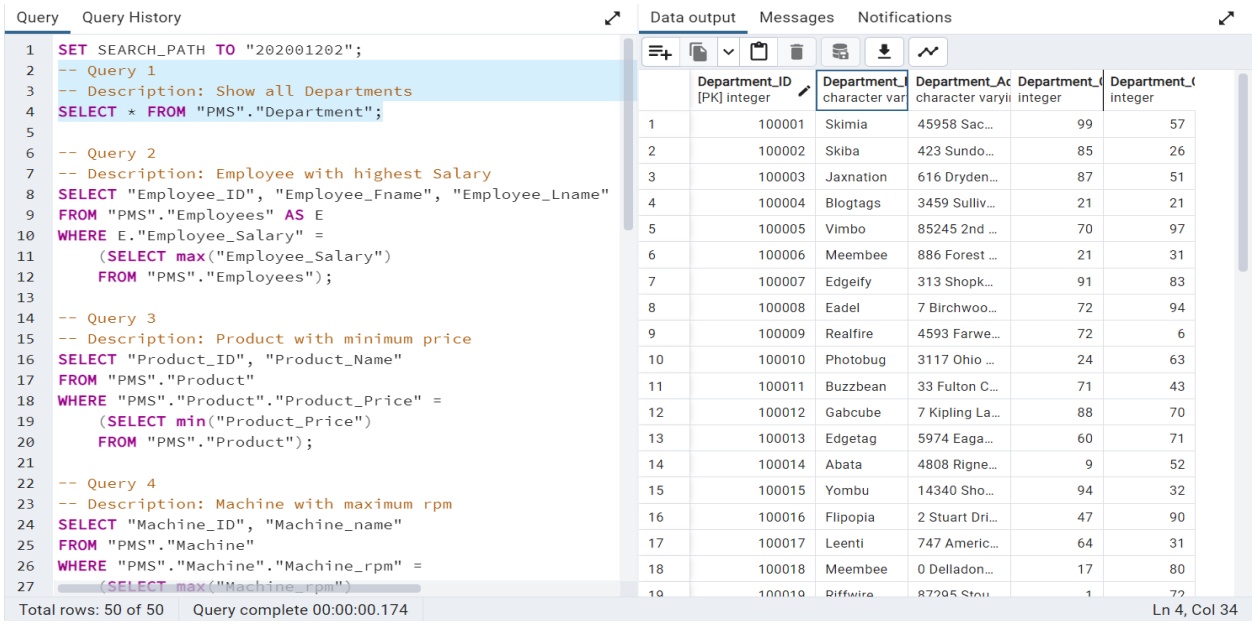
	Sell_Product_ID [PK] integer	Sell_Buyer_ID [PK] integer	Sell_Date [PK] date	Sell_time [PK] time with time zone	Sell_Quantity integer	Sell_Price integer
1	100025	100013	2022-07-31	23:30:00+05:30	52	297
2	100001	100046	2021-11-19	23:29:00+05:30	39	761
3	100012	100049	2022-01-18	23:15:00+05:30	69	880
4	100030	100013	2022-09-13	23:27:00+05:30	1	32
5	100017	100032	2021-10-16	23:48:00+05:30	50	949
6	100010	100031	2022-01-26	23:33:00+05:30	8	723
7	100005	100030	2022-05-02	23:32:00+05:30	73	344
8	100024	100005	2021-10-24	23:11:00+05:30	91	699
9	100033	100005	2021-12-09	23:58:00+05:30	17	663
10	100027	100037	2022-03-12	23:06:00+05:30	49	908
11	100039	100032	2022-01-16	23:47:00+05:30	31	447
12	100035	100034	2022-01-22	23:06:00+05:30	87	604
13	100011	100011	2022-05-07	23:25:00+05:30	72	644
14	100042	100016	2021-10-25	23:26:00+05:30	92	263
15	100012	100022	2022-08-21	23:48:00+05:30	34	468
16	100024	100050	2021-10-16	23:33:00+05:30	20	254
17	100029	100028	2022-09-08	23:18:00+05:30	9	721
18	100016	100027	2022-08-14	23:39:00+05:30	60	23

Total rows: 50 of 50 | Query complete 00:00:00.101 | Ln 1, Col 28

16. SQL Query:

16.1 SQL Simple Query:

1. Show all Departments

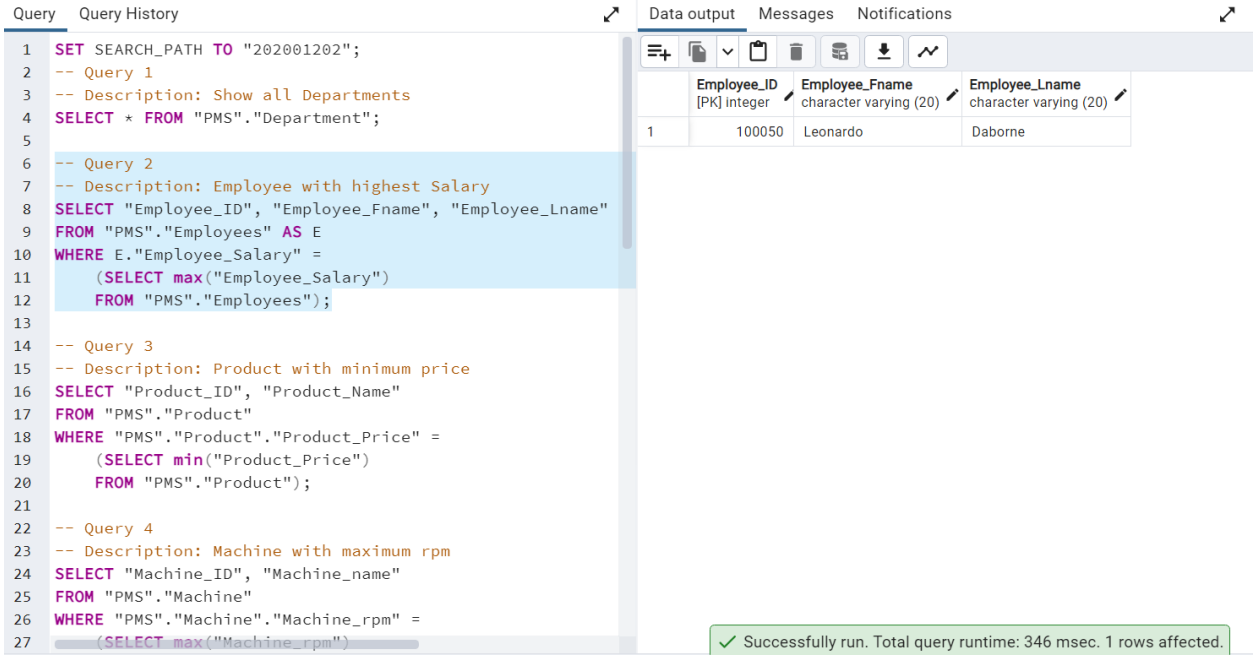


```
SET SEARCH_PATH TO "202001202";
-- Query 1
-- Description: Show all Departments
SELECT * FROM "PMS"."Department";

-- Query 2
-- Description: Employee with highest Salary
SELECT "Employee_ID", "Employee_Fname", "Employee_Lname"
FROM "PMS"."Employees" AS E
WHERE E."Employee_Salary" =
    (SELECT max("Employee_Salary")
    FROM "PMS"."Employees");

-- Query 3
-- Description: Product with minimum price
SELECT "Product_ID", "Product_Name"
FROM "PMS"."Product"
WHERE "PMS"."Product"."Product_Price" =
    (SELECT min("Product_Price")
    FROM "PMS"."Product");

-- Query 4
-- Description: Machine with maximum rpm
SELECT "Machine_ID", "Machine_name"
FROM "PMS"."Machine"
WHERE "PMS"."Machine"."Machine_rpm" =
    (SELECT max("Machine_rpm")
```

	Department_ID [PK] integer	Department_ character var	Department_Ac character varyi	Department_(integer	Department_(integer
1	100001	Skimia	45958 Sac...	99	57
2	100002	Skiba	423 Sundo...	85	26
3	100003	Jaxnation	616 Dryden...	87	51
4	100004	Blogtags	3459 Sulliv...	21	21
5	100005	Vimbo	85245 2nd ...	70	97
6	100006	Meembee	886 Forest ...	21	31
7	100007	Edgeify	313 Shopk...	91	83
8	100008	Eadel	7 Birchwoo...	72	94
9	100009	Realfire	4593 Farwe...	72	6
10	100010	Photobug	3117 Ohio ...	24	63
11	100011	Buzzbean	33 Fulton C...	71	43
12	100012	Gabcube	7 Kipling La...	88	70
13	100013	Edgetag	5974 Eaga...	60	71
14	100014	Abata	4808 Rigne...	9	52
15	100015	Yombu	14340 Sho...	94	32
16	100016	Flipopia	2 Stuart Dri...	47	90
17	100017	Leenti	747 Americ...	64	31
18	100018	Meembee	0 Delladon...	17	80
19	100019	Piffwire	87295 Stou	1	72

Total rows: 50 of 50 | Query complete 00:00:00.174 | Ln 4, Col 34

2. Show Employee with Highest Employee



```
SET SEARCH_PATH TO "202001202";
-- Query 1
-- Description: Show all Departments
SELECT * FROM "PMS"."Department";

-- Query 2
-- Description: Employee with highest Salary
SELECT "Employee_ID", "Employee_Fname", "Employee_Lname"
FROM "PMS"."Employees" AS E
WHERE E."Employee_Salary" =
    (SELECT max("Employee_Salary")
    FROM "PMS"."Employees");

-- Query 3
-- Description: Product with minimum price
SELECT "Product_ID", "Product_Name"
FROM "PMS"."Product"
WHERE "PMS"."Product"."Product_Price" =
    (SELECT min("Product_Price")
    FROM "PMS"."Product");

-- Query 4
-- Description: Machine with maximum rpm
SELECT "Machine_ID", "Machine_name"
FROM "PMS"."Machine"
WHERE "PMS"."Machine"."Machine_rpm" =
    (SELECT max("Machine_rpm")
```

	Employee_ID [PK] integer	Employee_Fname character varying (20)	Employee_Lname character varying (20)
1	100050	Leonardo	Daborne

Successfully run. Total query runtime: 346 msec. 1 rows affected.

3. Show Product with Lowest Price

Query | Query History

```
SET SEARCH_PATH TO "202001202";
-- Query 1
-- Description: Show all Departments
SELECT * FROM "PMS"."Department";

-- Query 2
-- Description: Employee with highest Salary
SELECT "Employee_ID", "Employee_Fname", "Employee_Lname"
FROM "PMS"."Employees" AS E
WHERE E."Employee_Salary" =
    (SELECT max("Employee_Salary")
    FROM "PMS"."Employees");

-- Query 3
-- Description: Product with minimum price
SELECT "Product_ID", "Product_Name"
FROM "PMS"."Product"
WHERE "PMS"."Product"."Product_Price" =
    (SELECT min("Product_Price")
    FROM "PMS"."Product");

-- Query 4
-- Description: Machine with maximum rpm
SELECT "Machine_ID", "Machine_name"
FROM "PMS"."Machine"
WHERE "PMS"."Machine"."Machine_rpm" =
    (SELECT max("Machine_rpm")
```

Data output | Messages | Notifications

	Product_ID [PK] integer	Product_Name character varying (50)
1	100003	Rice - Sushi

Total rows: 1 of 1 | Query complete 00:00:00.126 | Ln 20, Col 24

4. Show machine with maximum rpm

Query | Query History

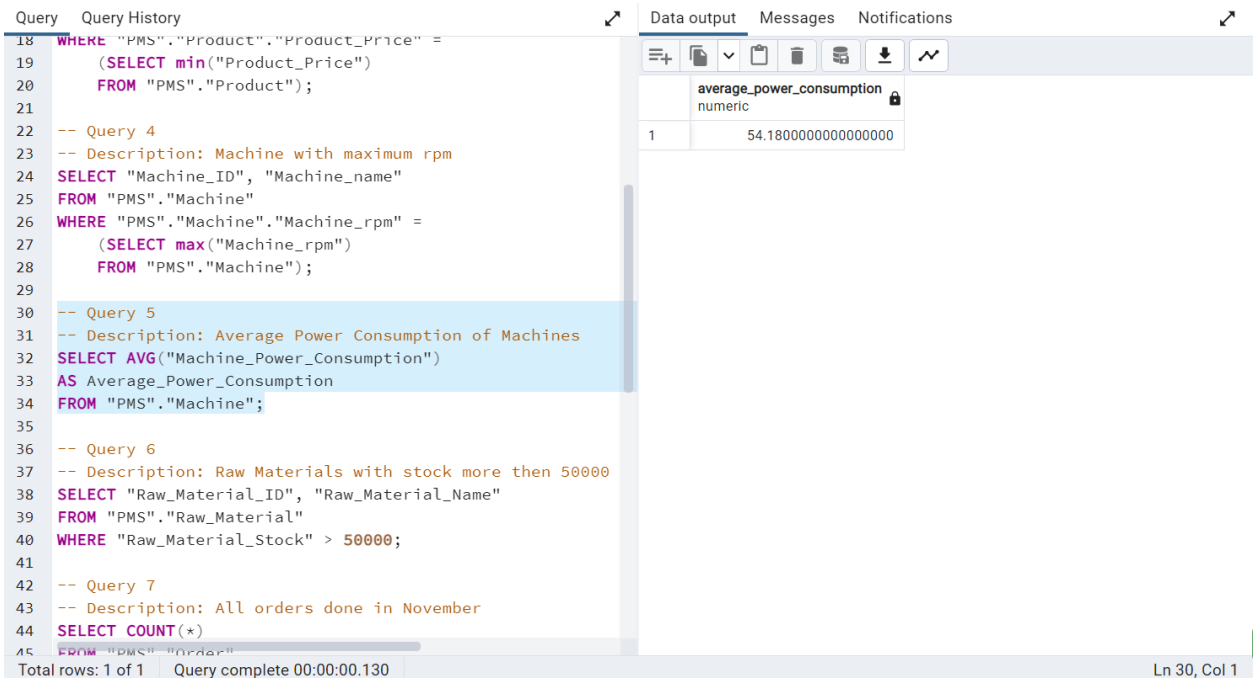
```
    FROM "PMS"."Employees");

-- Query 3
-- Description: Product with minimum price
SELECT "Product_ID", "Product_Name"
FROM "PMS"."Product"
WHERE "PMS"."Product"."Product_Price" =
    (SELECT min("Product_Price")
    FROM "PMS"."Product");

-- Query 4
-- Description: Machine with maximum rpm
SELECT "Machine_ID", "Machine_name"
FROM "PMS"."Machine"
WHERE "PMS"."Machine"."Machine_rpm" =
    (SELECT max("Machine_rpm")
    FROM "PMS"."Machine");

-- Query 5
-- Description: Average Power Consumption of Machines
SELECT AVG("Machine_Power_Consumption")
FROM "PMS"."Machine";

-- Query 6
-- Description: Raw Materials with stock more then 50000
SELECT "Raw_Material_ID", "Raw_Material_Name"
FROM "PMS"."Raw_Material"
```

Data output | Messages | Notifications

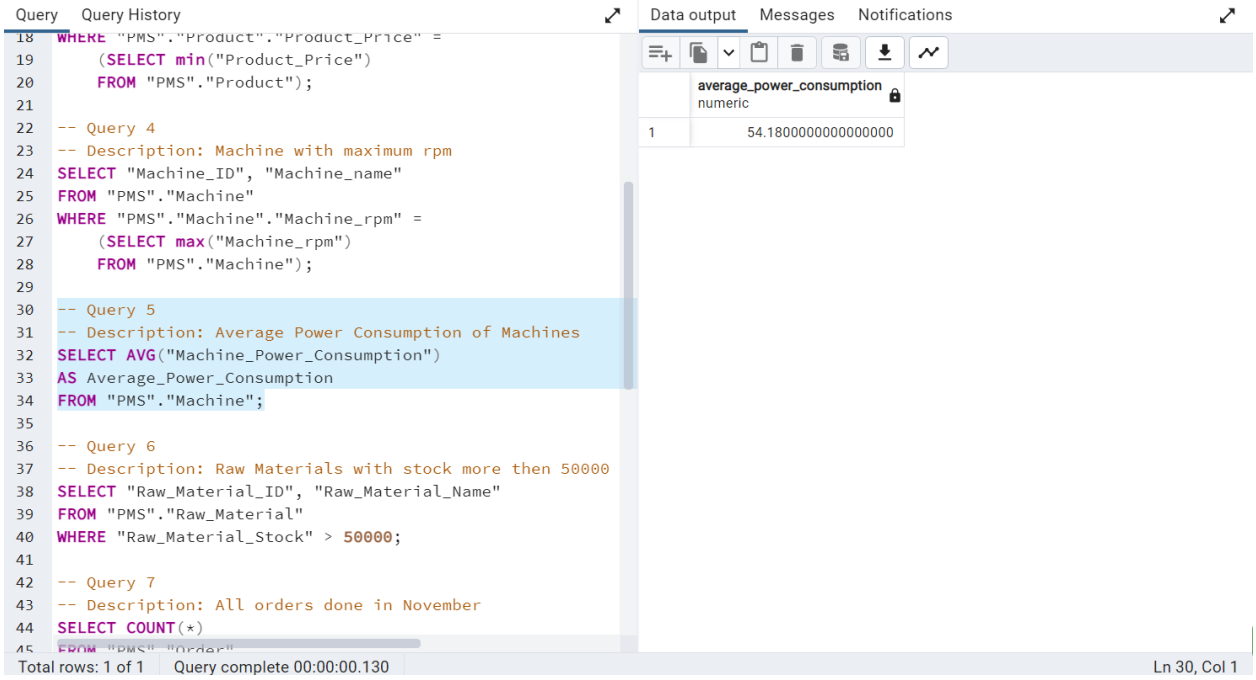
	Machine_ID [PK] integer	Machine_name character varying (20)
1	100015	Dump Truck

Total rows: 1 of 1 | Query complete 00:00:00.309 | Ln 28, Col 24

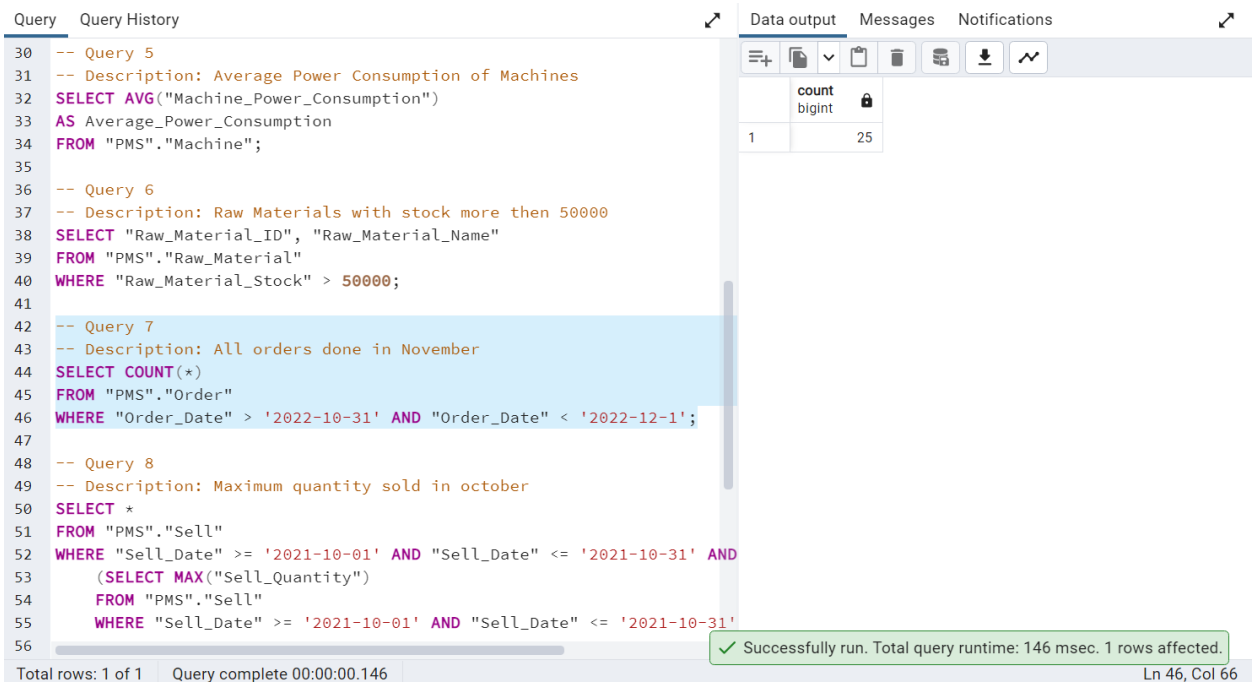
5. Average Power Consumption of Machines



6. Raw Material with the stock of more then 50000



7. Count Orders on November



```
-- Query 5
-- Description: Average Power Consumption of Machines
SELECT AVG("Machine_Power_Consumption")
AS Average_Power_Consumption
FROM "PMS"."Machine";

-- Query 6
-- Description: Raw Materials with stock more then 50000
SELECT "Raw_Material_ID", "Raw_Material_Name"
FROM "PMS"."Raw_Material"
WHERE "Raw_Material_Stock" > 50000;

-- Query 7
-- Description: All orders done in November
SELECT COUNT(*)
FROM "PMS"."Order"
WHERE "Order_Date" > '2022-10-31' AND "Order_Date" < '2022-12-1';

-- Query 8
-- Description: Maximum quantity sold in october
SELECT *
FROM "PMS"."Sell"
WHERE "Sell_Date" >= '2021-10-01' AND "Sell_Date" <= '2021-10-31' AND
    (SELECT MAX("Sell_Quantity")
    FROM "PMS"."Sell"
    WHERE "Sell_Date" >= '2021-10-01' AND "Sell_Date" <= '2021-10-31'
```

Query | Query History

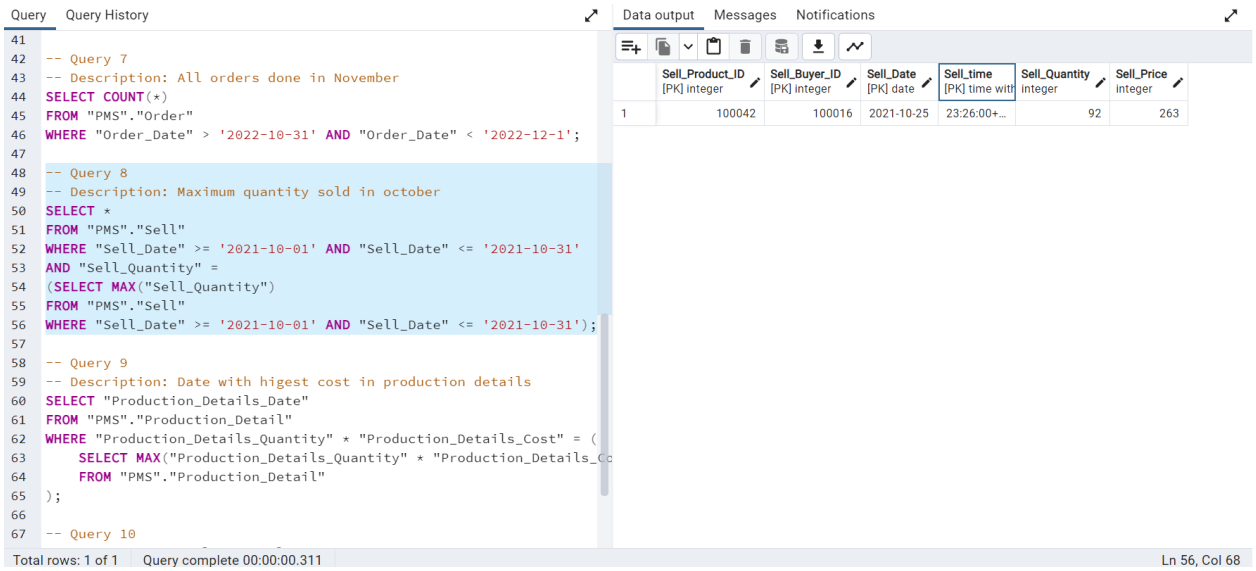
Data output | Messages | Notifications

	count bigint
1	25

Successfully run. Total query runtime: 146 msec. 1 rows affected.

Total rows: 1 of 1 | Query complete 00:00:00.146 | Ln 46, Col 66

8. Maximum Quantity sold in November



```
-- Query 7
-- Description: All orders done in November
SELECT COUNT(*)
FROM "PMS"."Order"
WHERE "Order_Date" > '2022-10-31' AND "Order_Date" < '2022-12-1';

-- Query 8
-- Description: Maximum quantity sold in october
SELECT *
FROM "PMS"."Sell"
WHERE "Sell_Date" >= '2021-10-01' AND "Sell_Date" <= '2021-10-31'
AND "Sell_Quantity" =
(SELECT MAX("Sell_Quantity")
FROM "PMS"."Sell"
WHERE "Sell_Date" >= '2021-10-01' AND "Sell_Date" <= '2021-10-31');

-- Query 9
-- Description: Date with higest cost in production details
SELECT "Production_Details_Date"
FROM "PMS"."Production_Detail"
WHERE "Production_Details_Quantity" * "Production_Details_Cost" = (
    SELECT MAX("Production_Details_Quantity" * "Production_Details_Co
    FROM "PMS"."Production_Detail"
);

-- Query 10
```

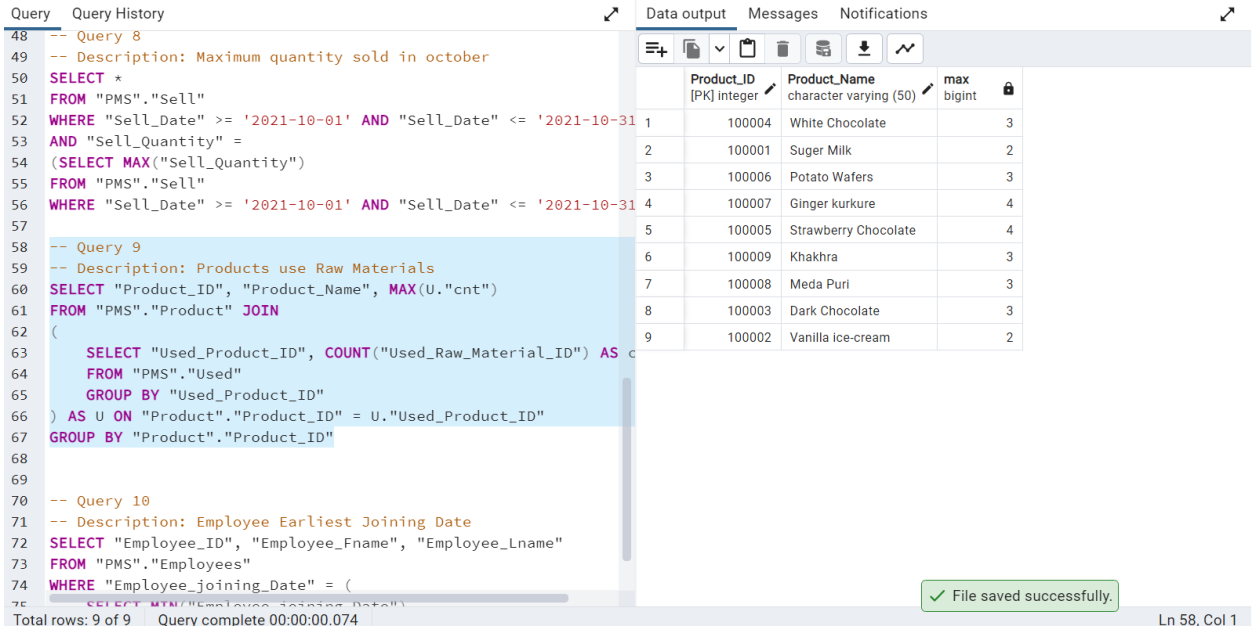
Query | Query History

Data output | Messages | Notifications

	Sell_Product_ID [PK] integer	Sell_Buyer_ID [PK] integer	Sell_Date [PK] date	Sell_time [PK] time with	Sell_Quantity integer	Sell_Price integer
1	100042	100016	2021-10-25	23:26:00+...	92	263

Total rows: 1 of 1 | Query complete 00:00:00.311 | Ln 56, Col 68

9. Products use Raw Materials:



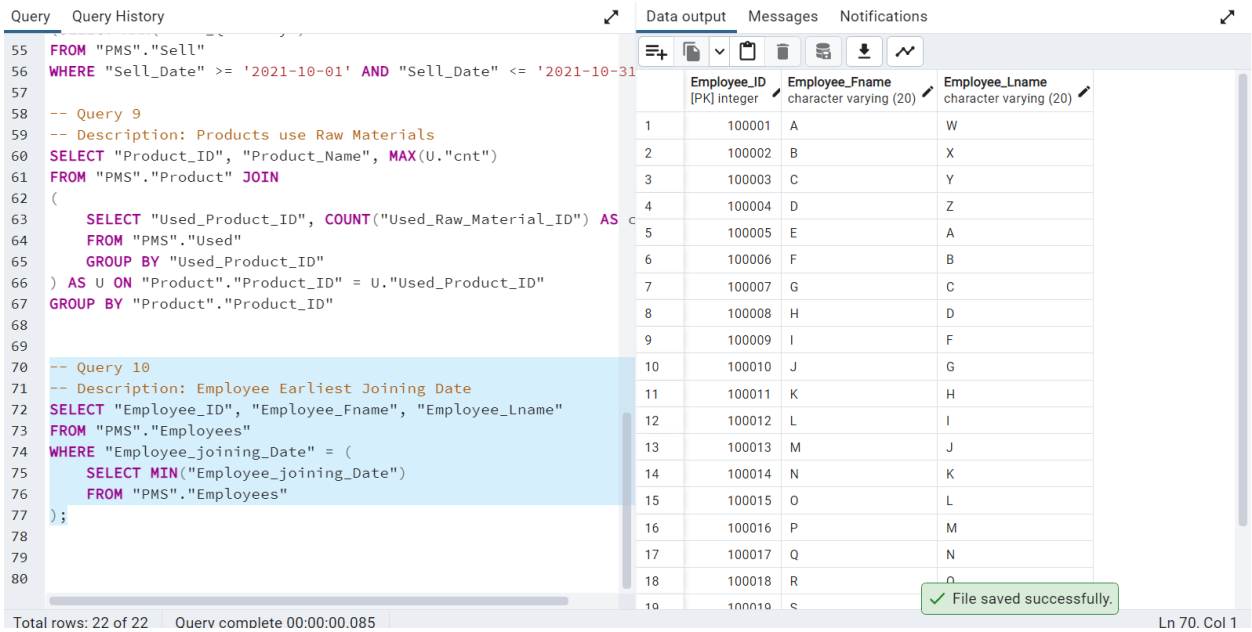
```
-- Query 8
-- Description: Maximum quantity sold in october
SELECT *
FROM "PMS"."Sell"
WHERE "Sell_Date" >= '2021-10-01' AND "Sell_Date" <= '2021-10-31
AND "Sell_Quantity" =
(SELECT MAX("Sell_Quantity")
FROM "PMS"."Sell"
WHERE "Sell_Date" >= '2021-10-01' AND "Sell_Date" <= '2021-10-31

-- Query 9
-- Description: Products use Raw Materials
SELECT "Product_ID", "Product_Name", MAX(U."cnt")
FROM "PMS"."Product" JOIN
(
    SELECT "Used_Product_ID", COUNT("Used_Raw_Material_ID") AS c
    FROM "PMS"."Used"
    GROUP BY "Used_Product_ID"
) AS U ON "Product"."Product_ID" = U."Used_Product_ID"
GROUP BY "Product"."Product_ID"


-- Query 10
-- Description: Employee Earliest Joining Date
SELECT "Employee_ID", "Employee_Fname", "Employee_Lname"
FROM "PMS"."Employees"
WHERE "Employee_joining_Date" = (
```

	Product_ID [PK] integer	Product_Name character varying (50)	max bigint
1	100004	White Chocolate	3
2	100001	Suger Milk	2
3	100006	Potato Wafers	3
4	100007	Ginger kurkure	4
5	100005	Strawberry Chocolate	4
6	100009	Khakhra	3
7	100008	Meda Puri	3
8	100003	Dark Chocolate	3
9	100002	Vanilla ice-cream	2

File saved successfully.

Total rows: 9 of 9 Query complete 00:00:00.074 Ln 58, Col 1

10. An employee with earliest joining date



```
FROM "PMS"."Sell"
WHERE "Sell_Date" >= '2021-10-01' AND "Sell_Date" <= '2021-10-31

-- Query 9
-- Description: Products use Raw Materials
SELECT "Product_ID", "Product_Name", MAX(U."cnt")
FROM "PMS"."Product" JOIN
(
    SELECT "Used_Product_ID", COUNT("Used_Raw_Material_ID") AS c
    FROM "PMS"."Used"
    GROUP BY "Used_Product_ID"
) AS U ON "Product"."Product_ID" = U."Used_Product_ID"
GROUP BY "Product"."Product_ID"


-- Query 10
-- Description: Employee Earliest Joining Date
SELECT "Employee_ID", "Employee_Fname", "Employee_Lname"
FROM "PMS"."Employees"
WHERE "Employee_joining_Date" = (
    SELECT MIN("Employee_joining_Date")
    FROM "PMS"."Employees"
);
```

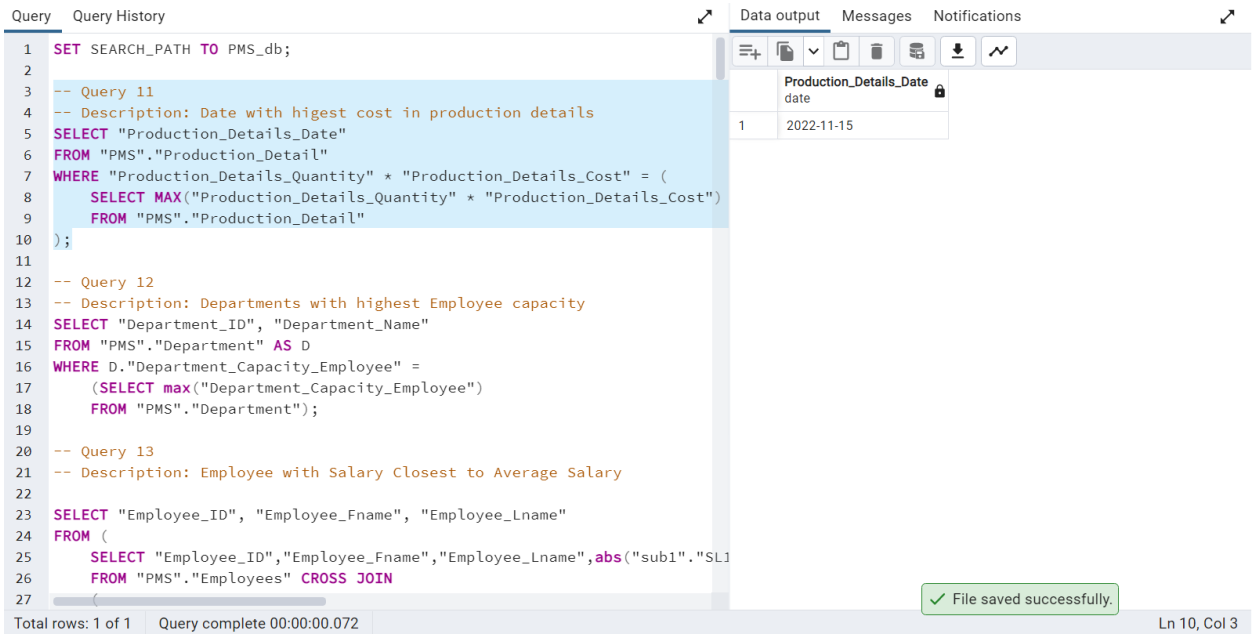
	Employee_ID [PK] integer	Employee_Fname character varying (20)	Employee_Lname character varying (20)
1	100001	A	W
2	100002	B	X
3	100003	C	Y
4	100004	D	Z
5	100005	E	A
6	100006	F	B
7	100007	G	C
8	100008	H	D
9	100009	I	F
10	100010	J	G
11	100011	K	H
12	100012	L	I
13	100013	M	J
14	100014	N	K
15	100015	O	L
16	100016	P	M
17	100017	Q	N
18	100018	R	O

File saved successfully.

Total rows: 22 of 22 Query complete 00:00:00.085 Ln 70, Col 1

16.2 SQL Complex Queries:

Query 11: Date with the Highest cost in production details



```
SET SEARCH_PATH TO PMS_db;

-- Query 11
-- Description: Date with higest cost in production details
SELECT "Production_Details_Date"
FROM "PMS"."Production_Detail"
WHERE "Production_Details_Quantity" * "Production_Details_Cost" = (
    SELECT MAX("Production_Details_Quantity" * "Production_Details_Cost")
    FROM "PMS"."Production_Detail"
);

-- Query 12
-- Description: Departments with highest Employee capacity
SELECT "Department_ID", "Department_Name"
FROM "PMS"."Department" AS D
WHERE D."Department_Capacity_Employee" =
    (SELECT max("Department_Capacity_Employee")
    FROM "PMS"."Department");

-- Query 13
-- Description: Employee with Salary Closest to Average Salary

SELECT "Employee_ID", "Employee_Fname", "Employee_Lname"
FROM (
    SELECT "Employee_ID","Employee_Fname","Employee_Lname",abs("sub1"."SL1
    FROM "PMS"."Employees" CROSS JOIN
    (
```

Query | Query History

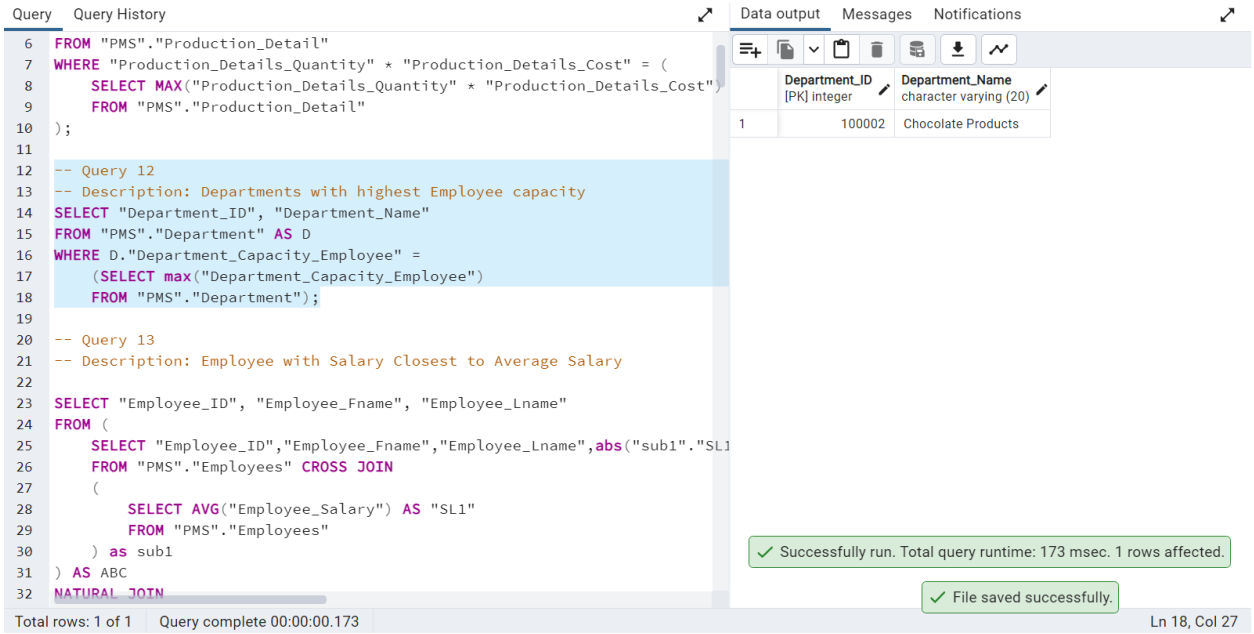
Data output | Messages | Notifications

	Production_Details_Date date
1	2022-11-15

File saved successfully.

Total rows: 1 of 1 | Query complete 00:00:00.072 | Ln 10, Col 3

Query 12: Department with Highest Employee Capacity



```
FROM "PMS"."Production_Detail"
WHERE "Production_Details_Quantity" * "Production_Details_Cost" = (
    SELECT MAX("Production_Details_Quantity" * "Production_Details_Cost")
    FROM "PMS"."Production_Detail"
);

-- Query 12
-- Description: Departments with highest Employee capacity
SELECT "Department_ID", "Department_Name"
FROM "PMS"."Department" AS D
WHERE D."Department_Capacity_Employee" =
    (SELECT max("Department_Capacity_Employee")
    FROM "PMS"."Department");

-- Query 13
-- Description: Employee with Salary Closest to Average Salary

SELECT "Employee_ID", "Employee_Fname", "Employee_Lname"
FROM (
    SELECT "Employee_ID","Employee_Fname","Employee_Lname",abs("sub1"."SL1
    FROM "PMS"."Employees" CROSS JOIN
    (
        SELECT AVG("Employee_Salary") AS "SL1"
        FROM "PMS"."Employees"
    ) as sub1
) AS ABC
NATURAL JOIN
```

Query | Query History

Data output | Messages | Notifications

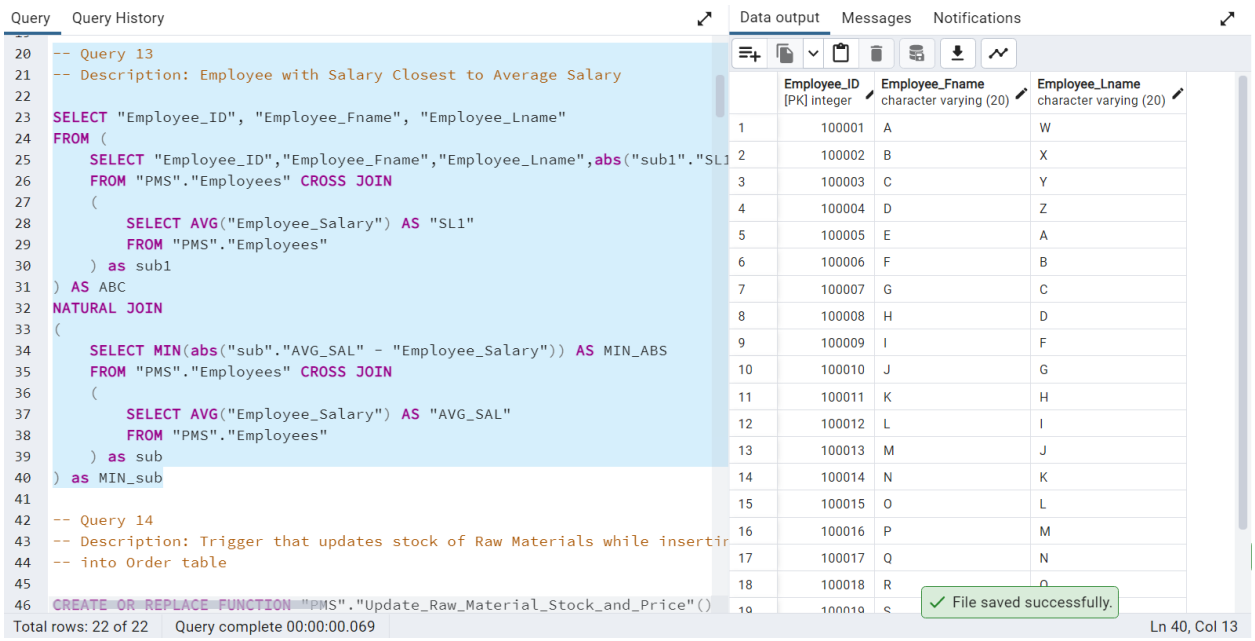
	Department_ID [PK] integer	Department_Name character varying (20)
1	100002	Chocolate Products

Successfully run. Total query runtime: 173 msec. 1 rows affected.

File saved successfully.

Total rows: 1 of 1 | Query complete 00:00:00.173 | Ln 18, Col 27

Query 13:find employee with a salary closest to the Average Salary.



```
-- Query 13
-- Description: Employee with Salary Closest to Average Salary

SELECT "Employee_ID", "Employee_Fname", "Employee_Lname"
FROM (
    SELECT "Employee_ID","Employee_Fname","Employee_Lname",abs("sub1"."SL1
    FROM "PMS"."Employees" CROSS JOIN
    (
        SELECT AVG("Employee_Salary") AS "SL1"
        FROM "PMS"."Employees"
    ) as sub1
) AS ABC
NATURAL JOIN
(
    SELECT MIN(abs("sub"."AVG_SAL" - "Employee_Salary")) AS MIN_ABS
    FROM "PMS"."Employees" CROSS JOIN
    (
        SELECT AVG("Employee_Salary") AS "AVG_SAL"
        FROM "PMS"."Employees"
    ) as sub
) as MIN_sub

-- Query 14
-- Description: Trigger that updates stock of Raw Materials while insertir
-- into Order table

CREATE OR REPLACE FUNCTION "PMS"."Update_Raw_Material_Stock_and_Price"()
```

Query | Query History

Data output | Messages | Notifications

	Employee_ID [PK] integer	Employee_Fname character varying (20)	Employee_Lname character varying (20)
1	100001	A	W
2	100002	B	X
3	100003	C	Y
4	100004	D	Z
5	100005	E	A
6	100006	F	B
7	100007	G	C
8	100008	H	D
9	100009	I	F
10	100010	J	G
11	100011	K	H
12	100012	L	I
13	100013	M	J
14	100014	N	K
15	100015	O	L
16	100016	P	M
17	100017	Q	N
18	100018	R	O
19	100019	S	

File saved successfully.

Total rows: 22 of 22 | Query complete 00:00:00.069 | Ln 40, Col 13

Query 14:Trigger that updates stock of Raw Materials while inserting into Order table

```
-- Query 14
-- Description: Trigger that updates stock of Raw Materials while
inserting into Order table

CREATE OR REPLACE FUNCTION "PMS"."Update_Raw_Material_Stock_and_Price"()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    UPDATE "PMS"."Raw_Material"
    SET "Raw_Material_price" = (("Raw_Material_Stock" *
"Raw_Material_price" + NEW."Order_Quantity" * NEW."Order_Price") /
("Raw_Material_Stock" + NEW."Order_Quantity")),
    "Raw_Material_Stock" = ("Raw_Material_Stock" + NEW."Order_Quantity")
    WHERE "Raw_Material_ID" = NEW."Order_Raw_material_ID";
RETURN NULL;
END
$BODY$;

CREATE OR REPLACE TRIGGER "Update_Raw_Material"
    AFTER INSERT
    ON "PMS"."Order"
    FOR EACH ROW
    EXECUTE PROCEDURE "PMS"."Update_Raw_Material_Stock_and_Price"();


UPDATE "PMS"."Raw_Material"
SET "Raw_Material_Stock" = 1000, "Raw_Material_price" = 150
WHERE "Raw_Material_ID" = 100006;
DELETE FROM "PMS"."Order";
INSERT INTO "PMS"."Order"
VALUES(100006,100002,'2022-11-14','19:29',130,1000);
```

Output:

- **Before Insert into Order table:**

Data output | Messages | Notifications

	Raw_Material_ID [PK] integer	Raw_Material_Name character varying (50)	Raw_Material_Stock integer	Raw_Material_price integer
3	100003	ChocolateD	20000	200
4	100004	ChocolateW	10000	200
5	100005	Butter	4000	400
6	100006	Strawberry	1000	150
7	100007	Wheat	50000	50

- **Insertion in order table:**

Data output | Messages | Notifications

	Order_Raw_material_ID [PK] integer	Order_Supplier_ID [PK] integer	Order_Date [PK] date	Order_Time [PK] time with time zone	Order_Price integer	Order_Quantity integer
1	100006	100002	2022-11-14	19:29:00+05:30	130	1000

- **After Insert into Order table:**

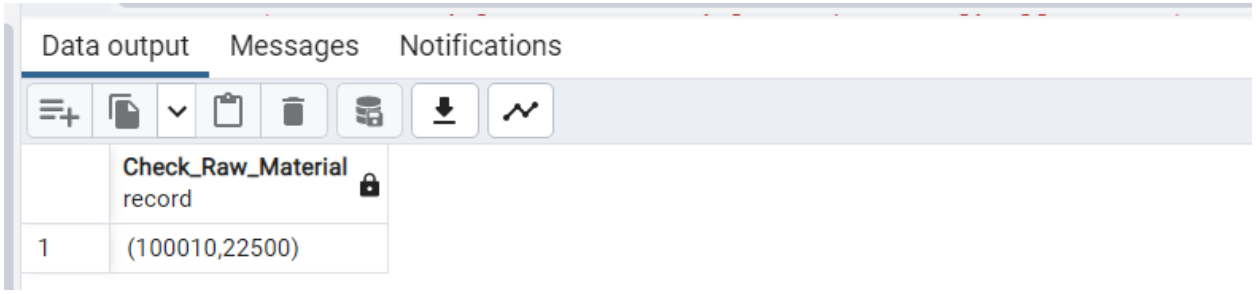
Data output | Messages | Notifications

	Raw_Material_ID [PK] integer	Raw_Material_Name character varying (50)	Raw_Material_Stock integer	Raw_Material_price integer
1	100001	Milk	50000	35
2	100002	Suger	30000	50
3	100003	ChocolateD	20000	200
4	100004	ChocolateW	10000	200
5	100005	Butter	4000	400
6	100006	Strawberry	2000	140
7	100007	Wheat	50000	50

Query 15: Function that returns insufficient raw material for next day production.

```
-- Query 15
-- Description: Function that return insufficient raw material for next
day production

CREATE OR REPLACE FUNCTION "PMS"."Check_Raw_Material"()
    RETURNS TABLE("Order_Raw_Material_ID" integer, use bigint)
    LANGUAGE 'plpgsql'


AS $BODY$
BEGIN
RETURN QUERY EXECUTE FORMAT(
    'SELECT "Raw_Material"."Raw_Material_ID", "finall"."use"
    FROM(
        SELECT "Raw_Material_ID",
sum("Used_Percentage_weight"*"Product_Weight"*"Product_Production_limit"/1
00) as use
        FROM(
            SELECT *
            FROM(
                (
                    "PMS"."Raw_Material" JOIN "PMS"."Used"
                    ON "Used"."Used_Raw_Material_ID" =
"Raw_Material"."Raw_Material_ID"
                ) AS rmu JOIN "PMS"."Product"
                ON "Product"."Product_ID"="rmu"."Used_Product_ID"
                AND "Product"."Product_Production_flag" = TRUE
            )
            AS rmup
            )AS rr
        GROUP BY "rr"."Raw_Material_ID"
    ) as finall JOIN "PMS"."Raw_Material"
    ON ("Raw_Material"."Raw_Material_ID" = "finall"."Raw_Material_ID")
    AND ("Raw_Material"."Raw_Material_Stock" <= "finall"."use")'
);
END;
```

```
$BODY$;
SELECT "PMS"."Check_Raw_Material"();
```

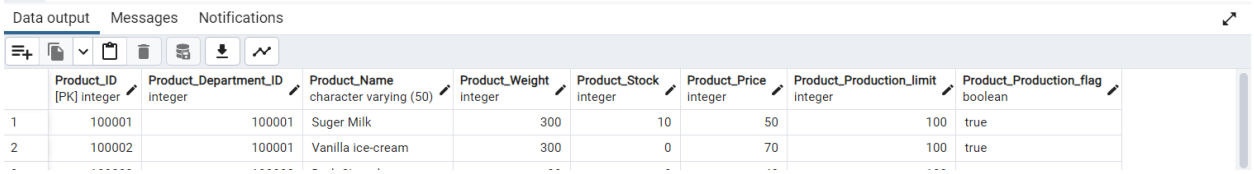


Query 16 : Validate sell i.e., if there is insufficient product stock then raise notice otherwise insert into Sell table and update product stock

```
-- Query 16
-- Description: Validate sell i.e., if there is insufficient product stock
then raise
-- notice otherwise insert into sell table and update product stock

CREATE OR REPLACE FUNCTION "PMS"."Update_Product_Stock"()
    RETURNS trigger


  LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE OLD_STOCK INTEGER;
BEGIN
    OLD_STOCK = (SELECT "Product_Stock" FROM "PMS"."Product" WHERE
"Product_ID" = NEW."Sell_Product_ID");
    NEW."Sell_Price" = (SELECT "Product_Price" FROM "PMS"."Product" WHERE
"Product_ID" = NEW."Sell_Product_ID");
    IF (OLD_STOCK < NEW."Sell_Quantity") THEN
        RAISE NOTICE 'Insufficient Stock!!';
        RAISE NOTICE 'Present Stock = %', OLD_STOCK;
        RETURN OLD;
    ELSE
        UPDATE "PMS"."Product"
        SET "Product_Stock" = "Product_Stock" - NEW."Sell_Quantity"
        WHERE "Product_ID" = NEW."Sell_Product_ID";
    END IF;
RETURN NEW;
END
$BODY$;

CREATE TRIGGER "Update_Product_Stock"
    BEFORE INSERT
    ON "PMS"."Sell"
```

```
    FOR EACH ROW
    EXECUTE PROCEDURE "PMS"."Update_Product_Stock"();

UPDATE "PMS"."Product"
SET "Product_Stock" = 10
WHERE "Product_ID" = 100001;
DELETE FROM "PMS"."Sell";
INSERT INTO "PMS"."Sell"
VALUES(100001,100001,'2022-11-15','14:35',1,NULL);
```

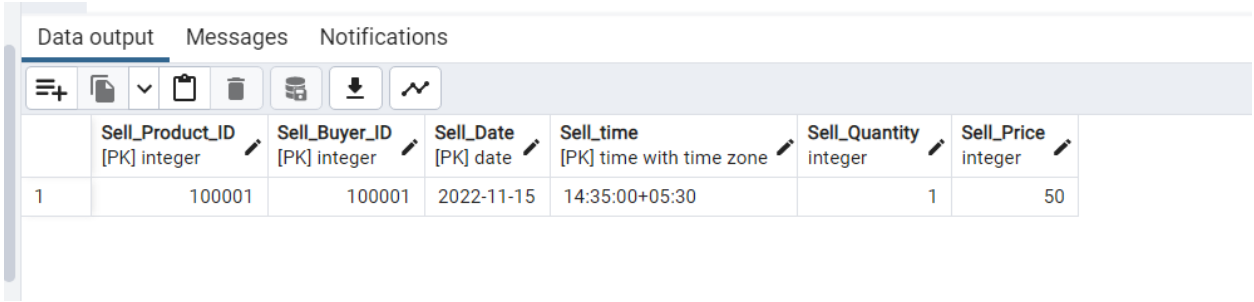
- **Before insert into sell table:**



Data output | Messages | Notifications

	Product_ID [PK] integer	Product_Department_ID integer	Product_Name character varying (50)	Product_Weight integer	Product_Stock integer	Product_Price integer	Product_Production_limit integer	Product_Production_flag boolean
1	100001	100001	Suger Milk	300	10	50	100	true
2	100002	100001	Vanilla ice-cream	300	0	70	100	true

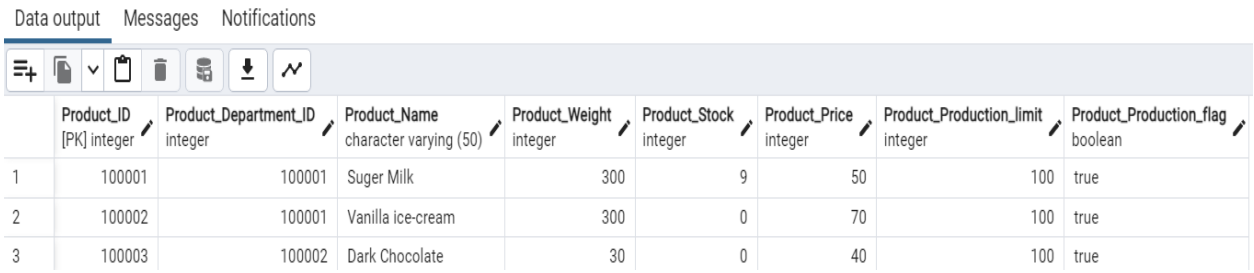
- **Insertion in sell table:**



Data output | Messages | Notifications

	Sell_Product_ID [PK] integer	Sell_Buyer_ID [PK] integer	Sell_Date [PK] date	Sell_time [PK] time with time zone	Sell_Quantity integer	Sell_Price integer
1	100001	100001	2022-11-15	14:35:00+05:30	1	50

- **After Insert into Sell table:**



Data output | Messages | Notifications

	Product_ID [PK] integer	Product_Department_ID integer	Product_Name character varying (50)	Product_Weight integer	Product_Stock integer	Product_Price integer	Product_Production_limit integer	Product_Production_flag boolean
1	100001	100001	Suger Milk	300	9	50	100	true
2	100002	100001	Vanilla ice-cream	300	0	70	100	true
3	100003	100002	Dark Chocolate	30	0	40	100	true

Query 17 :Trigger that calculate the production cost before inserting into Production_Detail table and update product stock in Product table

```
-- Query 17
-- Trigger that calculate the production cost before inserting into
-- Production_Detail table and update product stock in Product table
CREATE OR REPLACE FUNCTION "PMS"."Add_Production_Details"()
RETURNS trigger
LANGUAGE 'plpgsql'
AS $BODY$
DECLARE Employee_Cost INTEGER;
DECLARE Machine_Cost INTEGER;
DECLARE Raw_Material_Cost INTEGER;
DECLARE Produce_Flag BOOLEAN;
BEGIN

    Produce_Flag = (
        SELECT "Product_Production_flag"
        FROM "PMS"."Product"
        WHERE "Product_ID" = NEW."Production_Details_Product_ID"
    );
    IF (Produce_Flag = FALSE)
    THEN
        RAISE NOTICE 'Production Not Allowed !!';
        RETURN OLD;
    ELSE
        Employee_Cost = (
            SELECT SUM("Employee_Salary")
            FROM (
                "PMS"."Employees" JOIN "PMS"."Product"
                ON "Product"."Product_ID" =
"Employees"."Employee_Product_ID"
                AND NEW."Production_Details_Product_ID" =
"Employees"."Employee_Product_ID"
            ) as EP
```

```
            GROUP BY "Product_ID"
        );
        Machine_Cost = (
            SELECT SUM("Machine_Maintenance_Cost" +
"Machine_Power_Consumption")
            FROM (
                "PMS"."Machine" JOIN "PMS"."Product"
                ON "Product"."Product_ID" = "Machine"."Machine_Product_ID"
                AND NEW."Production_Details_Product_ID" =
"Machine"."Machine_Product_ID"
            ) as EP
            GROUP BY "Product_ID"
        );
        Raw_Material_Cost = (
            SELECT SUM(NEW."Production_Details_Quantity" *
"Used_Percentage_weight" * "Product_Weight" * "Raw_Material_price" /
100000)
            FROM(
                SELECT *
                FROM(
                    (
                        "PMS"."Raw_Material" JOIN "PMS"."Used"
                        ON "Used"."Used_Raw_Material_ID" =
"Raw_Material"."Raw_Material_ID"
                    ) AS rmu JOIN "PMS"."Product"
                    ON "Product"."Product_ID"="rmu"."Used_Product_ID"
                    AND NEW."Production_Details_Product_ID" =
"Product"."Product_ID"
                )
                AS rmup
            ) AS rmc
            GROUP BY "Product_ID"
        );
        NEW."Production_Details_Cost" = Raw_Material_Cost + Machine_Cost +
Employee_Cost;

        UPDATE "PMS"."Product"
        SET "Product_Stock" = "Product_Stock" +
NEW."Production_Details_Quantity"
        WHERE "Product_ID" = NEW."Production_Details_Product_ID";
```

```
    END IF;
    RETURN NEW;
END $BODY$;

CREATE OR REPLACE TRIGGER Add_Production_Detail
    BEFORE INSERT
    ON "PMS"."Production_Detail"
    FOR EACH ROW
    EXECUTE PROCEDURE "PMS"."Add_Production_Details"();

DELETE FROM "PMS"."Production_Detail";
UPDATE "PMS"."Product"
SET "Product_Stock" = 5
WHERE "Product_ID" = 100001;
INSERT INTO "PMS"."Production_Detail" VALUES(100001,'2022-11-15',50,NULL);
```

- **Before insert into Production_Detail table:**

Data output | Messages | Notifications

	Product_ID [PK] integer	Product_Department_ID integer	Product_Name character varying (50)	Product_Weight integer	Product_Stock integer	Product_Price integer	Product_Production_limit integer	Product_Production_flag boolean
1	100001	100001	Suger Milk	300	5	50	100	true
2	100002	100001	Vanilla ice-cream	300	0	70	100	true
3	100003	100002	Dark Chocolate	30	0	40	100	true

- **Insertion in Production_detail table:**

Data output | Messages | Notifications

	Production_Details_Product_ID [PK] integer	Production_Details_Date [PK] date	Production_Details_Quantity integer	Production_Details_Cost integer
1	100001	2022-11-15	50	728

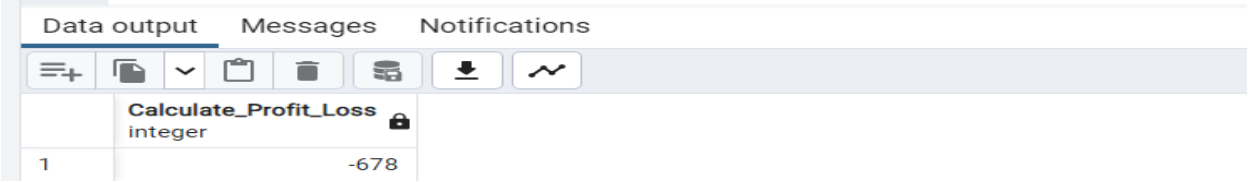
- **After Insert into Production_Detail table:**

Data output | Messages | Notifications

	Product_ID [PK] integer	Product_Department_ID integer	Product_Name character varying (50)	Product_Weight integer	Product_Stock integer	Product_Price integer	Product_Production_limit integer	Product_Production_flag boolean
1	100001	100001	Suger Milk	300	55	50	100	true
2	100002	100001	Vanilla ice-cream	300	0	70	100	true
3	100003	100002	Dark Chocolate	30	0	40	100	true

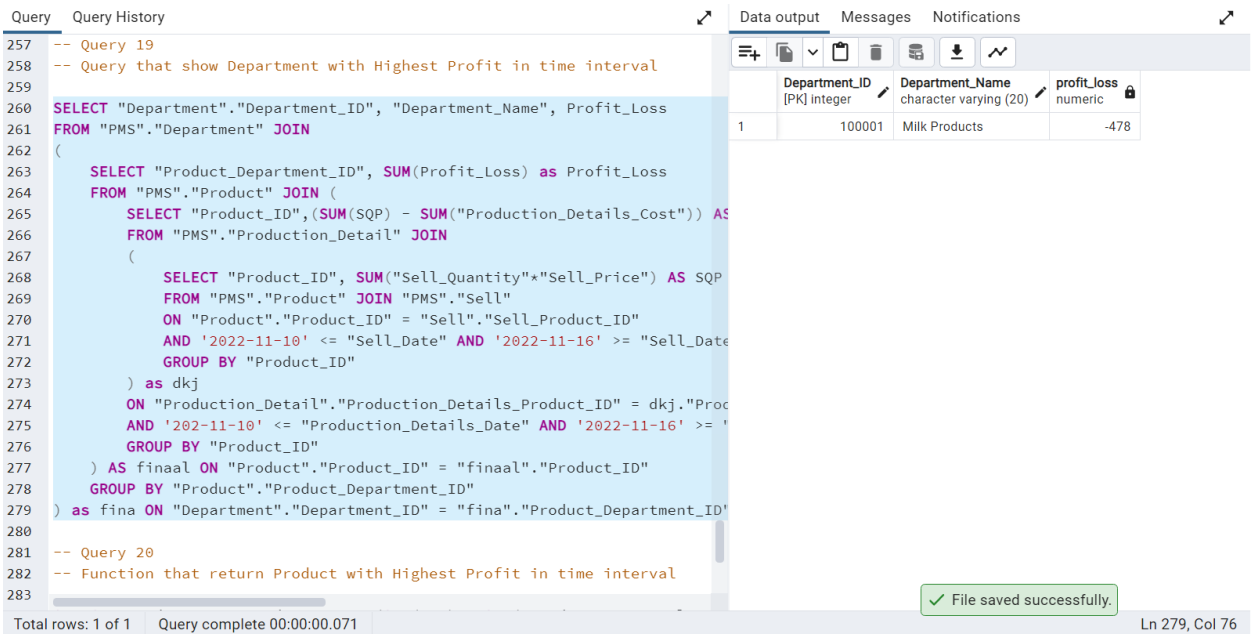
Query 18: Function that return Total Profit/Loss in time interval

```
-- Query 18
-- Function that return Total Profit/Loss in time interval


CREATE OR REPLACE FUNCTION "PMS"."Calculate_Profit_Loss"(Start_Date DATE,
End_Date DATE)
    RETURNS INTEGER
    LANGUAGE 'plpgsql'

AS $BODY$
DECLARE TOTAL_SELL_VALUE INTEGER;
DECLARE COST_VALUE INTEGER;
BEGIN
    TOTAL_SELL_VALUE = (
        SELECT SUM("Sell_Quantity" * "Sell_Price")
        FROM "PMS"."Sell"
        WHERE "Sell_Date" >= Start_Date AND "Sell_Date" <= End_Date
        GROUP BY "Sell_Product_ID"
    );
    COST_VALUE = (
        SELECT SUM("Production_Details_Cost")
        FROM "PMS"."Production_Detail"
        WHERE "Production_Details_Date" >= Start_Date AND
"Production_Details_Date" <= End_Date
        GROUP BY "Production_Details_Product_ID"
    );
    RETURN TOTAL_SELL_VALUE - COST_VALUE;
END;
$BODY$;
SELECT "PMS"."Calculate_Profit_Loss"('2022,11,10','2022-11-16');
```



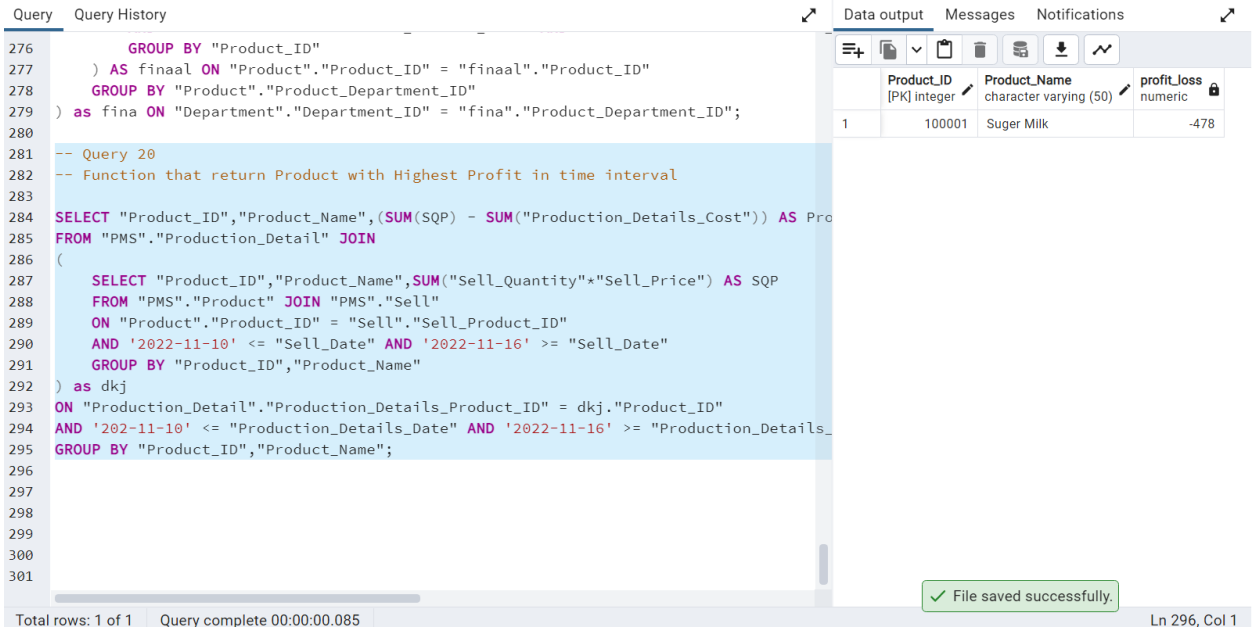
Query 19 :Query that show Department with Profit in time interval

```
-- Query 19
-- Query that show Department with Highest Profit in time interval

SELECT "Department"."Department_ID", "Department_Name", Profit_Loss
FROM "PMS"."Department" JOIN
(
    SELECT "Product_Department_ID", SUM(Profit_Loss) as Profit_Loss
    FROM "PMS"."Product" JOIN (
        SELECT "Product_ID",(SUM(SQP) - SUM("Production_Details_Cost")) AS
Profit_Loss
        FROM "PMS"."Production_Detail" JOIN
        (
            SELECT "Product_ID", SUM("Sell_Quantity"*"Sell_Price") AS SQP
            FROM "PMS"."Product" JOIN "PMS"."Sell"
            ON "Product"."Product_ID" = "Sell"."Sell_Product_ID"
            AND '2022-11-10' <= "Sell_Date" AND '2022-11-16' >=
"Sell_Date"
            GROUP BY "Product_ID"
        ) as dkj
        ON "Production_Detail"."Production_Details_Product_ID" =
dkj."Product_ID"
        AND '202-11-10' <= "Production_Details_Date" AND '2022-11-16' >=
"Production_Details_Date"
        GROUP BY "Product_ID"
    ) AS finaal ON "Product"."Product_ID" = "finaal"."Product_ID"
    GROUP BY "Product"."Product_Department_ID"
) as fina ON "Department"."Department_ID" =
"fina"."Product_Department_ID";
```


Query
Query History
Data output
Messages
Notifications
-- Query 19
-- Query that show Department with Highest Profit in time interval
SELECT "Department"."Department_ID", "Department_Name", Profit_Loss
FROM "PMS"."Department" JOIN
(
SELECT "Product_Department_ID", SUM(Profit_Loss) as Profit_Loss
FROM "PMS"."Product" JOIN (
SELECT "Product_ID",(SUM(SQP) - SUM("Production_Details_Cost")) AS
FROM "PMS"."Production_Detail" JOIN
(
SELECT "Product_ID", SUM("Sell_Quantity"*"Sell_Price") AS SQP
FROM "PMS"."Product" JOIN "PMS"."Sell"
ON "Product"."Product_ID" = "Sell"."Sell_Product_ID"
AND '2022-11-10' <= "Sell_Date" AND '2022-11-16' >= "Sell_Date
GROUP BY "Product_ID"
) as dkj
ON "Production_Detail"."Production_Details_Product_ID" = dkj."Proc
AND '202-11-10' <= "Production_Details_Date" AND '2022-11-16' >= '
GROUP BY "Product_ID"
) AS finaal ON "Product"."Product_ID" = "finaal"."Product_ID"
GROUP BY "Product"."Product_Department_ID"
) as fina ON "Department"."Department_ID" = "fina"."Product_Department_ID"
-- Query 20
-- Function that return Product with Highest Profit in time interval
Department_ID
[PK] integer
Department_Name
character varying (20)
profit_loss
numeric
1
100001
Milk Products
-478
File saved successfully.
Total rows: 1 of 1
Query complete 00:00:00.071
Ln 279, Col 76

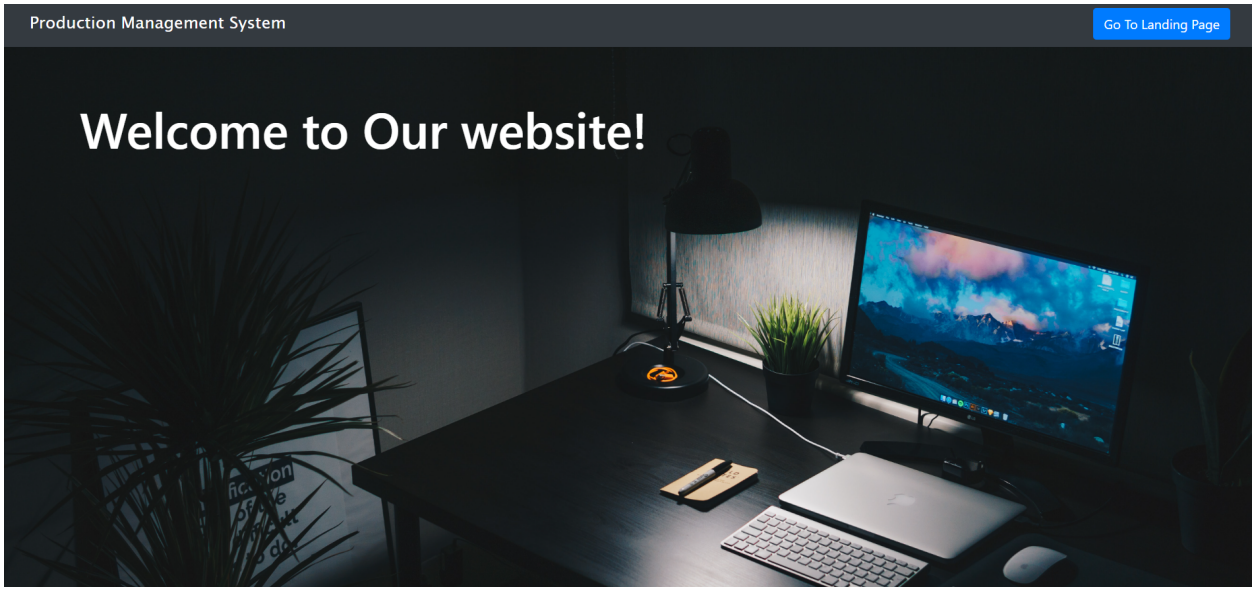
Query 20: Query that return Product with Highest Profit in time interval

```
-- Query 20
-- Query that return Product with Highest Profit in time interval
SELECT "Product_ID","Product_Name",(SUM(SQP) -
SUM("Production_Details_Cost")) AS Profit_Loss
FROM "PMS"."Production_Detail" JOIN
(
    SELECT "Product_ID","Product_Name",SUM("Sell_Quantity"*"Sell_Price")
AS SQP
    FROM "PMS"."Product" JOIN "PMS"."Sell"
    ON "Product"."Product_ID" = "Sell"."Sell_Product_ID"
    AND '2022-11-10' <= "Sell_Date" AND '2022-11-16' >= "Sell_Date"
    GROUP BY "Product_ID","Product_Name"
) as dkj
ON "Production_Detail"."Production_Details_Product_ID" = dkj."Product_ID"
AND '202-11-10' <= "Production_Details_Date" AND '2022-11-16' >=
"Production_Details_Date"
GROUP BY "Product_ID","Product_Name";
```



17. Interface Representation of Production Management System

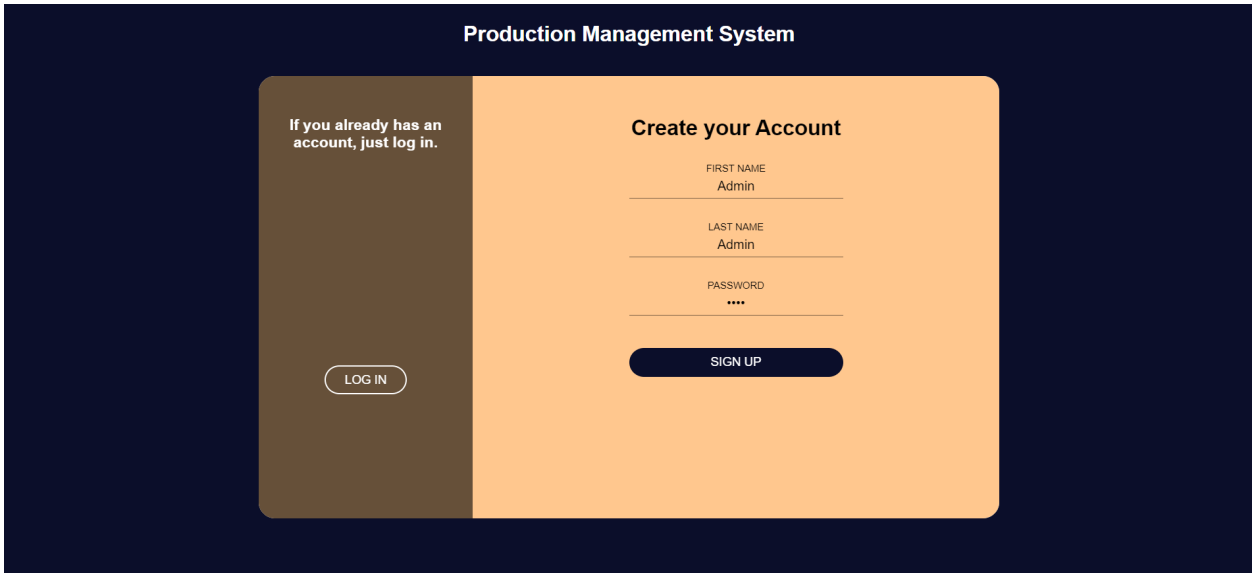
Home Page:



Press ‘Go to Landing page’ to go login Page

Sign Up:

→ Administrators can Sign Up by creating a new account.



Before Sign Up:

	Employee_ID [PK] integer	Employee_Product_ID integer	Employee_password character varying (20)	Employee_Fname character varying (20)	Employee_Lname character varying (20)	Employee_Salary integer	Employee_hours integer	Employee_shift boolean
5	100005	100003	100005	E	A	30	6	false
6	100006	100003	100006	F	B	30	6	true
7	100007	100004	100007	G	C	30	6	false
8	100008	100004	100008	H	D	30	6	true
9	100009	100005	100009	I	F	30	6	false
10	100010	100005	100010	J	G	30	6	true
11	100011	100006	100011	K	H	30	6	false
12	100012	100006	100012	L	I	30	6	true
13	100013	100007	100013	M	J	30	6	false
14	100014	100007	100014	N	K	30	6	true
15	100015	100008	100015	O	L	30	6	false
16	100016	100008	100016	P	M	30	6	true
17	100017	100008	100017	Q	N	30	6	false
18	100018	100008	100018	R	O	30	6	true
19	100019	100009	100019	S	P	30	6	false
20	100020	100009	100020	T	Q	30	6	true
21	100021	100009	100021	U	R	30	6	false
22	100022	100009	100022	V	S	30	6	true

After Sign Up:

	Employee_ID [PK] integer	Employee_Product_ID integer	Employee_password character varying (20)	Employee_Fname character varying (20)	Employee_Lname character varying (20)	Employee_Salary integer	Employee_hours integer	Employee_shift boolean	E d
7	100007	100004	100007	G	C	30	6	false	2
8	100008	100004	100008	H	D	30	6	true	2
9	100009	100005	100009	I	F	30	6	false	2
10	100010	100005	100010	J	G	30	6	true	2
11	100011	100006	100011	K	H	30	6	false	2
12	100012	100006	100012	L	I	30	6	true	2
13	100013	100007	100013	M	J	30	6	false	2
14	100014	100007	100014	N	K	30	6	true	2
15	100015	100008	100015	O	L	30	6	false	2
16	100016	100008	100016	P	M	30	6	true	2
17	100017	100008	100017	Q	N	30	6	false	2
18	100018	100008	100018	R	O	30	6	true	2
19	100019	100009	100019	S	P	30	6	false	2
20	100020	100009	100020	T	Q	30	6	true	2
21	100021	100009	100021	U	R	30	6	false	2
22	100022	100009	100022	V	S	30	6	true	2
23	100023	[null]	1234	Admin	Admin	[null]	[null]	[null]	[

→ After signing up using create account Admin data will be added to the Postgres database and shown in the Employee table.

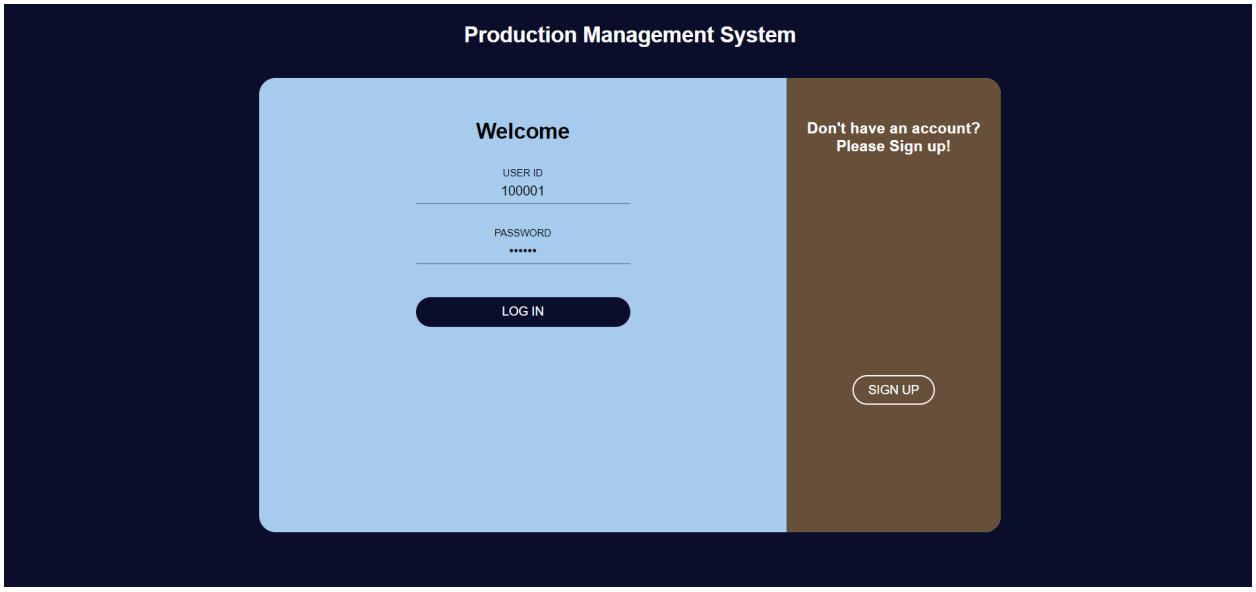
Login Page:

→ On the login Page we have to options for login

1) Login as Employee

2) Login as Administrator

Login As Employee:



→ Employee ID 100001 is able to login using their password and Employee will be able to see their details as shown in the below figures.

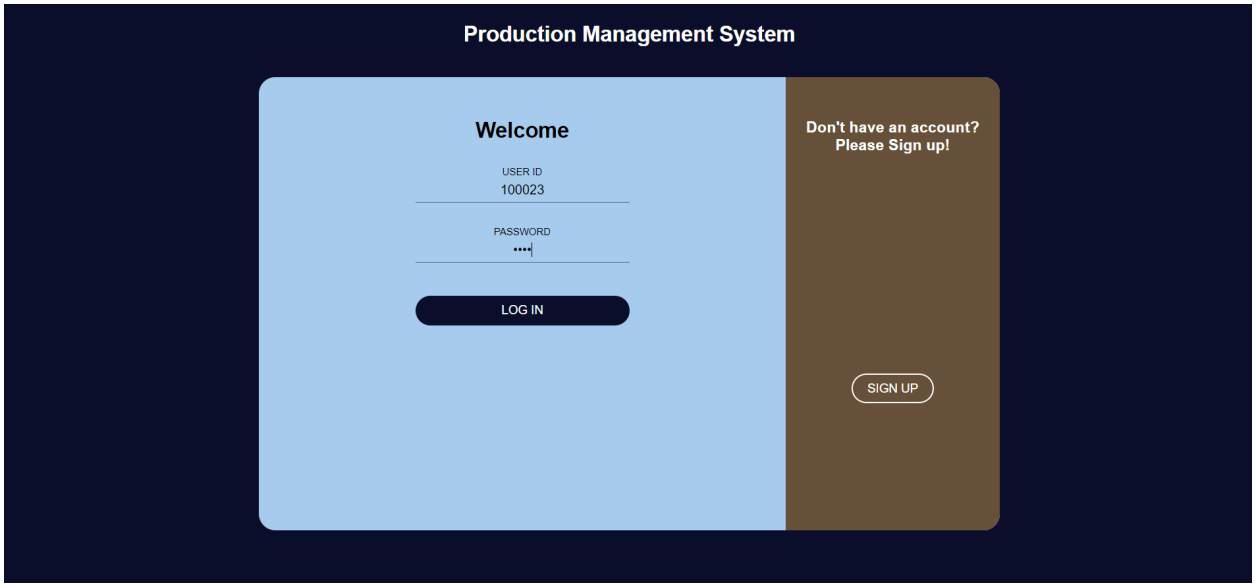
After Login As Employee:

→ Employee Can only see his/her Information

EMPLOYEES

Employee_ID	Employee_Product_ID	Employee_password	Employee_Fname	Employee_Lname	Employee_Salary	Employee_hours	Employee_shift	Employee_joining_Date	Employee_address
100001	100001	100001	A	W	30	6	false	Tue Nov 01 2022 00:00:00 GMT+0530 (India Standard Time)	Adrr 100001

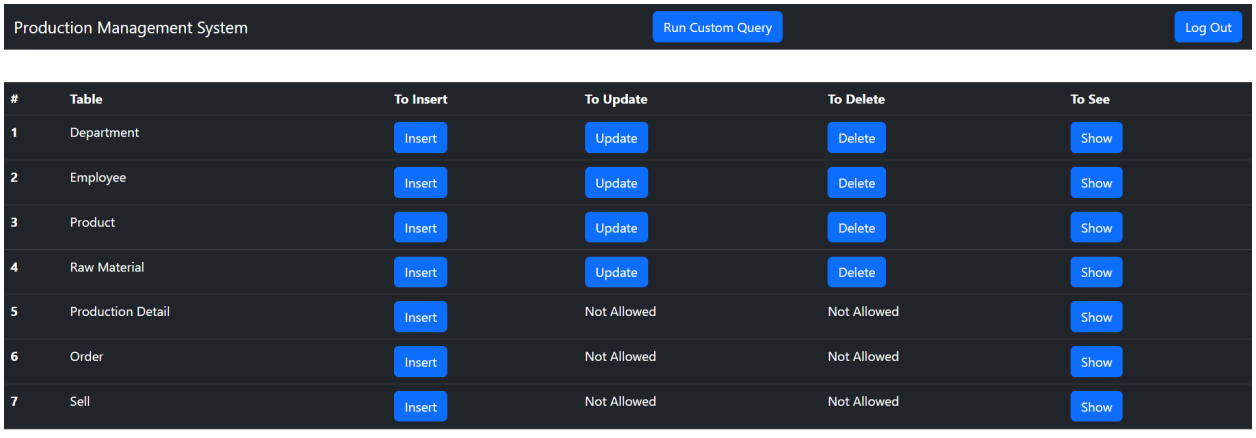
Login As Administrator:



→ Administrator with ID 100023 will able to login into the system using their password.

→ In the system there will be a possibility to have more than one Administrator.

After Login As Administrator:



→ Administrator Page Description:

1. Custom Query: Helps to write a query to execute the custom query in the Database and show results

2. Log Out: Log out as Administrator

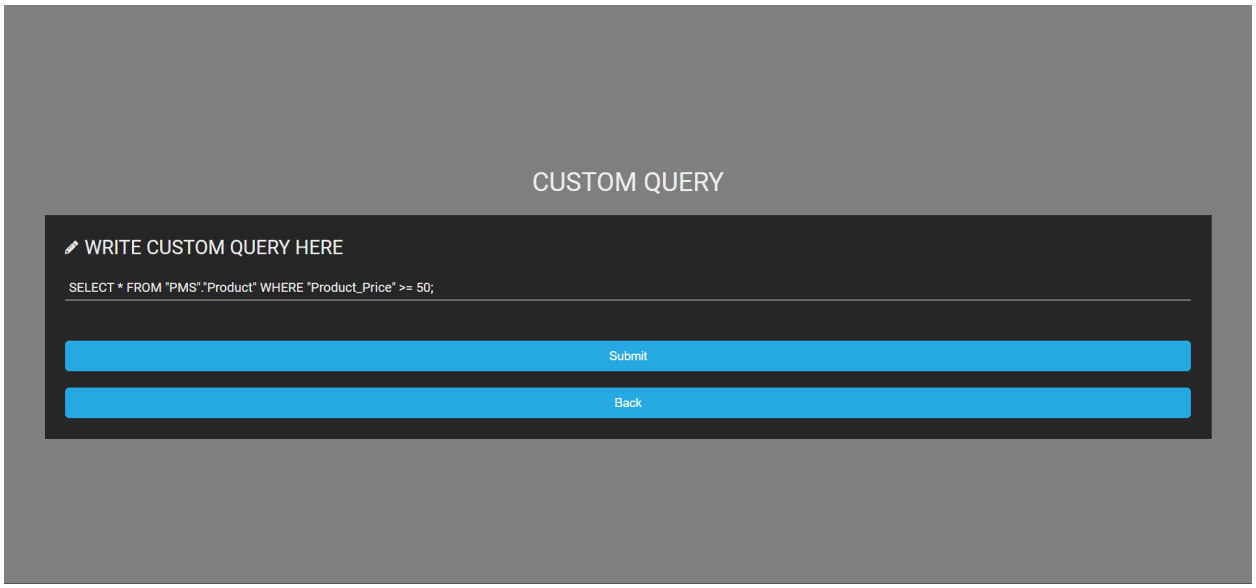
3. Insert: Provide a form and according to that information will be inserted into the Postgres database.

4. Update: Provide one Interface to write an Update query and execute it into the Postgres database.

5. Delete: Provide one Interface to write a Delete query and execute it into the Postgres database.

6. Show: Show data of the whole table.

1. RUN CUSTOM QUERY:



→After pressing the Run custom query button Administrator can write a custom query.

→ In above query we select product with price grater than 50 rupees.

→After Executing the Custom Query:

Product_ID	Product_Department_ID	Product_Name	Product_Weight	Product_Stock	Product_Price	Product_Production_limit	Product_Production_flag
100002	100001	Vanilla ice-cream	300	0	70	100	true
100005	100002	Strawberry Chocolate	40	0	60	100	true
100008	100003	Meda Puri	200	0	150	50	true
100009	100003	Khakhra	500	0	200	50	true
100001	100001	Suger Milk	300	55	50	100	true

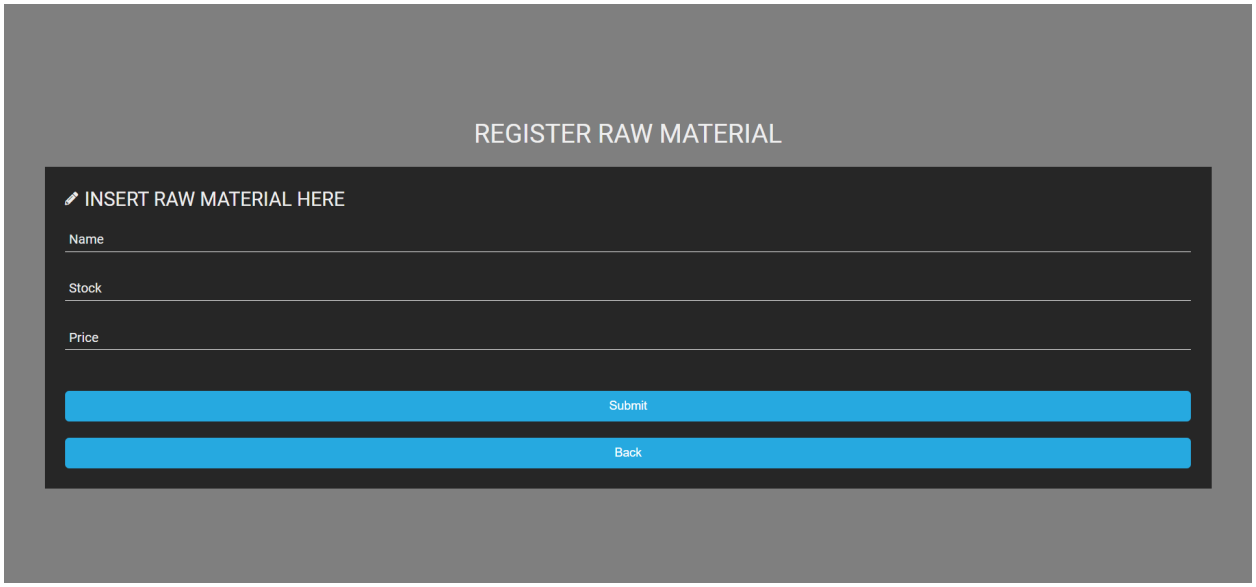
Back

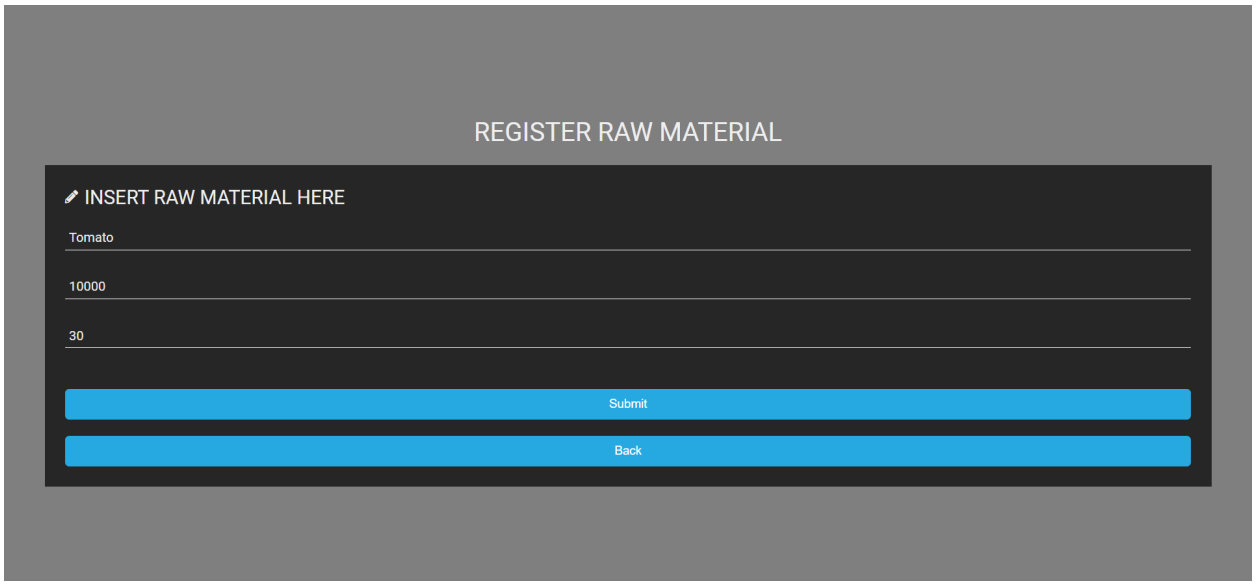
→ This will show the details of the product with price greater than 50 Rupees.

→ Here the Back button is for going back to the Administrator Page.

2. INSERT:

1) Insertion of Raw Material:





→After pressing the insert button Administrator can insert data in form and add the item which is selected.

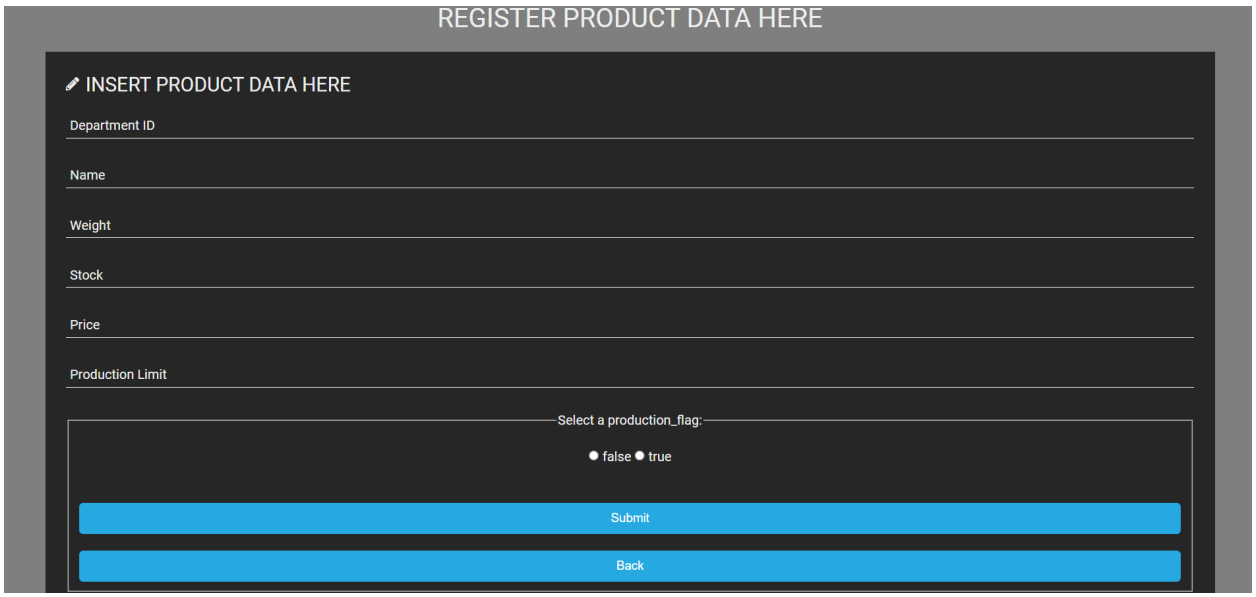
→ In above form we will be adding Raw material with their details.

After Adding Raw Material:

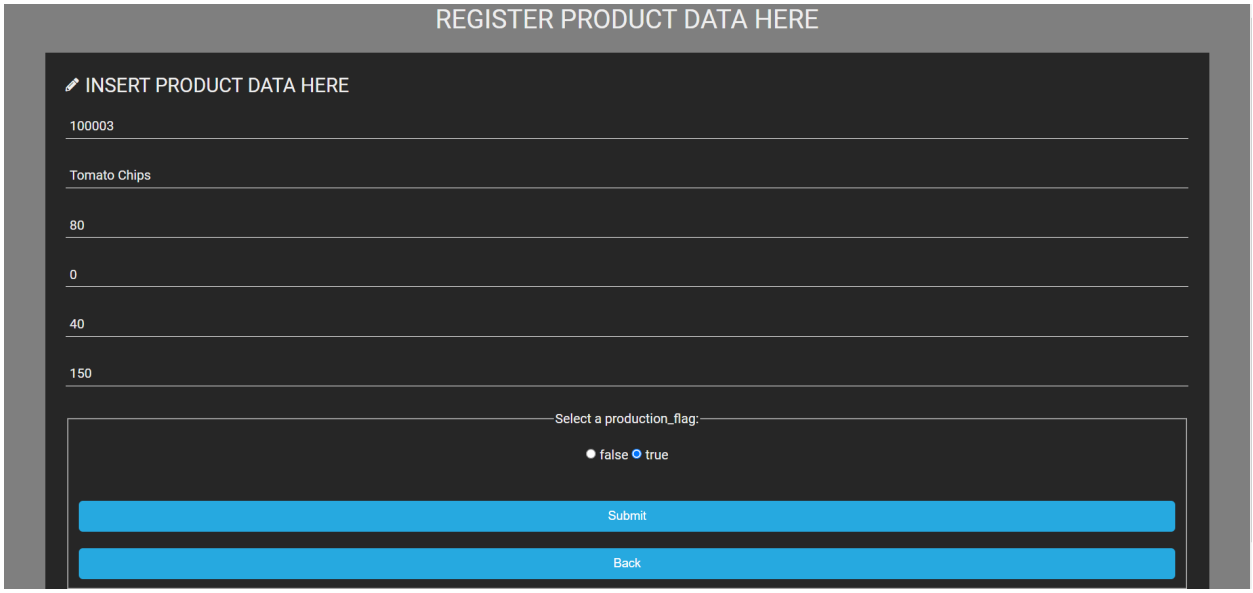
	Raw_Material_ID [PK] integer	Raw_Material_Name character varying (50)	Raw_Material_Stock integer	Raw_Material_price integer
1	100001	Milk	50000	35
2	100002	Suger	30000	50
3	100003	ChocolateD	20000	200
4	100004	ChocolateW	10000	200
5	100005	Butter	4000	400
6	100006	Strawberry	2000	140
7	100007	Wheat	50000	50
8	100008	Solt	10000	10
9	100009	Chilli	20000	300
10	100010	Meda	20000	30
11	100011	Potato	20000	30
12	100012	Ginger	2000	100
13	100013	Tomato	10000	30

→ After pressing the submit button Raw material with the name Tomato will be added to the Postgres database and shown in the Raw Material table.
→ ID is auto-assigned with the use of a trigger.

2) Insertion of Product:



→the form will be shown as above for insert product.

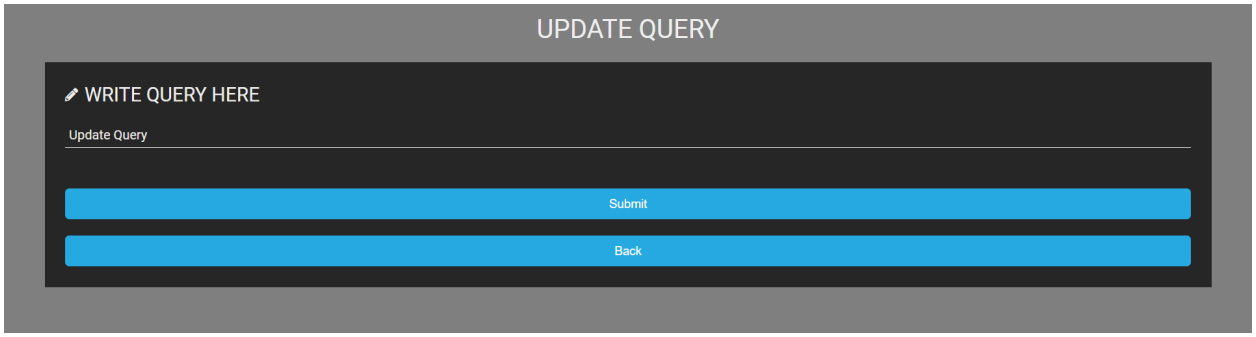


→After pressing submit button Product with the name Tomato chips will be added to the Postgres database and shown in the Product table.
→ In above form we will be adding Product with their details.

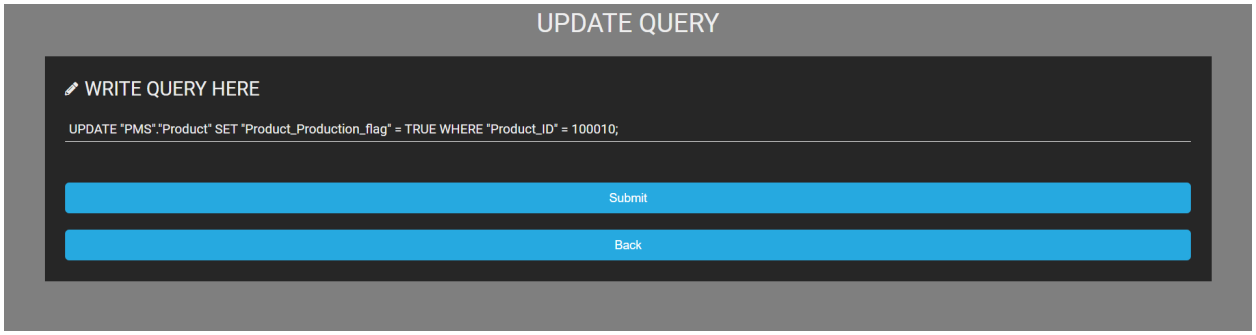
After Adding Product:

	Product_ID [PK] integer	Product_Department_ID integer	Product_Name character varying (50)	Product_Weight integer	Product_Stock integer	Product_Price integer	Product_Production_limit integer	Product_Production_flag boolean
1	100001	100001	Suger Milk	300	55	50	100	true
2	100002	100001	Vanilla ice-cream	300	0	70	100	true
3	100003	100002	Dark Chocolate	30	0	40	100	true
4	100004	100002	White Chocolate	30	0	30	100	true
5	100005	100002	Strawberry Chocolate	40	0	60	100	true
6	100006	100003	Potato Wafers	80	0	20	100	true
7	100007	100003	Ginger kurkure	80	0	25	100	true
8	100008	100003	Meda Puri	200	0	150	50	true
9	100009	100003	Khakhra	500	0	200	50	true
10	100010	100003	Tomato Chips	80	0	40	150	false

3. UPDATE:



→ Here administrator can write query for update

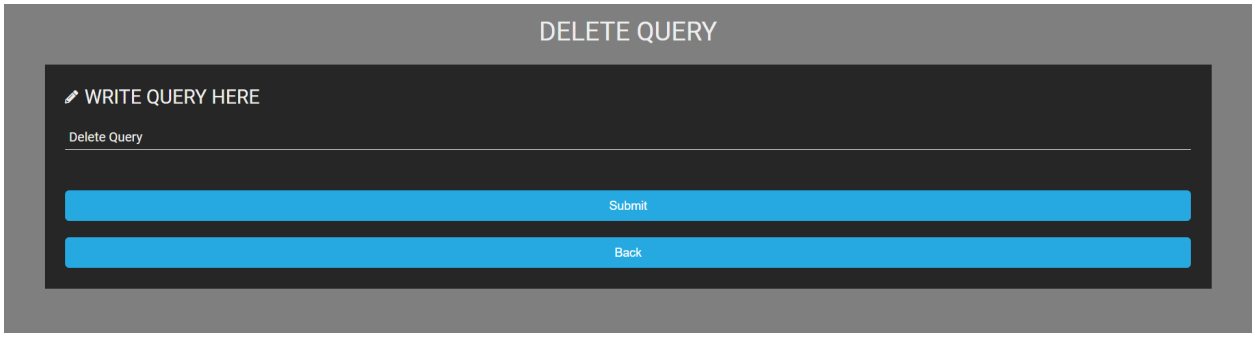


→ update product production flag TRUE with given product ID.
→ after pressing submit button update of the product will be updated to the Postgres database and shown in the Product table

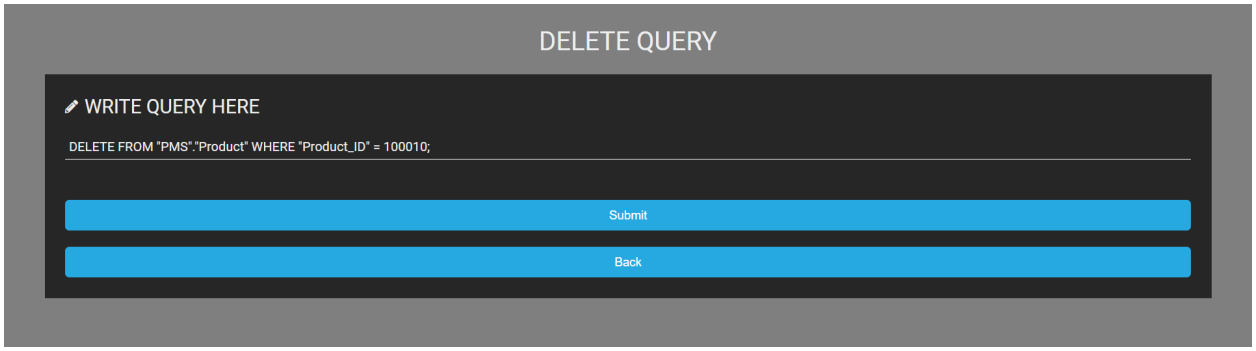
Update in the product table:

	Product_ID [PK] integer	Product_Department_ID integer	Product_Name character varying (50)	Product_Weight integer	Product_Stock integer	Product_Price integer	Product_Production_limit integer	Product_Production_flag boolean
1	100001	100001	Suger Milk	300	55	50	100	true
2	100002	100001	Vanilla ice-cream	300	0	70	100	true
3	100003	100002	Dark Chocolate	30	0	40	100	true
4	100004	100002	White Chocolate	30	0	30	100	true
5	100005	100002	Strawberry Chocolate	40	0	60	100	true
6	100006	100003	Potato Wafers	80	0	20	100	true
7	100007	100003	Ginger kurkure	80	0	25	100	true
8	100008	100003	Meda Puri	200	0	150	50	true
9	100009	100003	Khakhra	500	0	200	50	true
10	100010	100003	Tomato Chips	80	0	40	150	true

4. DELETE:



→ Here administrator can write a query for deletion.



→ delete product with product id 100010.

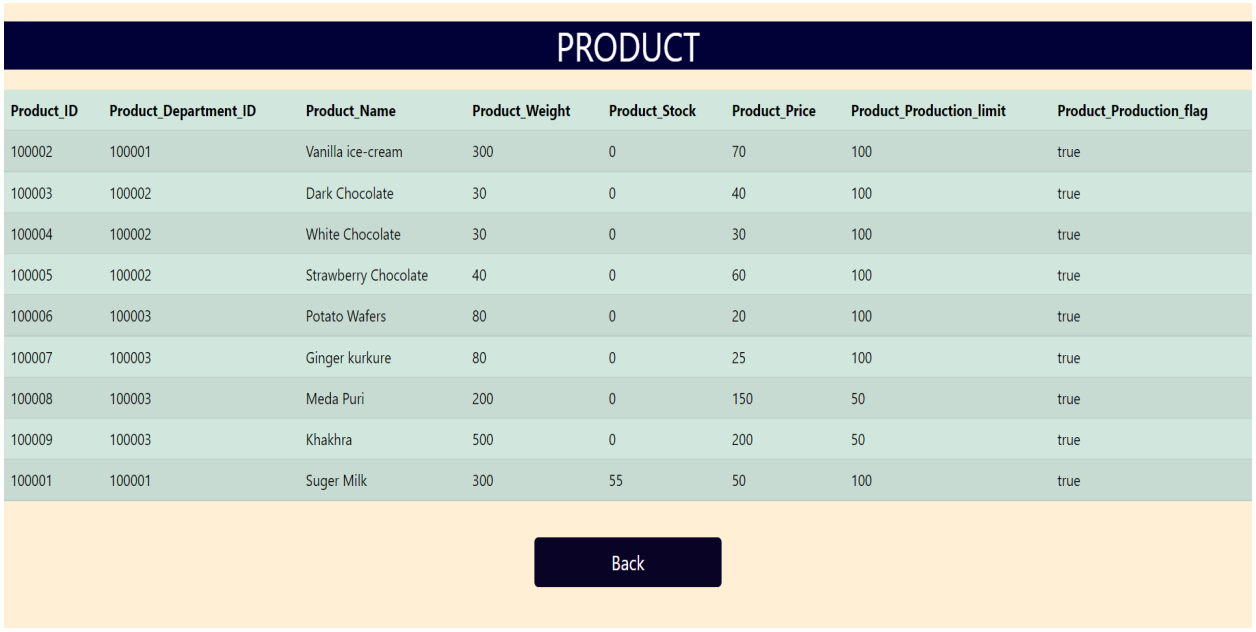
→ after pressing submit button deletion of the product will be deleted from the Postgres database and shown in the Product table.

Delete in product table:

	Product_ID [PK] integer	Product_Department_ID integer	Product_Name character varying (50)	Product_Weight integer	Product_Stock integer	Product_Price integer	Product_Production_limit integer	Product_Production_flag boolean
1	100001	100001	Suger Milk	300	55	50	100	true
2	100002	100001	Vanilla ice-cream	300	0	70	100	true
3	100003	100002	Dark Chocolate	30	0	40	100	true
4	100004	100002	White Chocolate	30	0	30	100	true
5	100005	100002	Strawberry Chocolate	40	0	60	100	true
6	100006	100003	Potato Wafers	80	0	20	100	true
7	100007	100003	Ginger kurkure	80	0	25	100	true
8	100008	100003	Meda Puri	200	0	150	50	true
9	100009	100003	Khakhra	500	0	200	50	true

5. SHOW:

→This option is available on the administrator page. Using the show button administrator is able to see the data of any table.
→After pressing the show button of the product table we are able to see the table as per below.



PRODUCT

Product_ID	Product_Department_ID	Product_Name	Product_Weight	Product_Stock	Product_Price	Product_Production_limit	Product_Production_flag
100002	100001	Vanilla ice-cream	300	0	70	100	true
100003	100002	Dark Chocolate	30	0	40	100	true
100004	100002	White Chocolate	30	0	30	100	true
100005	100002	Strawberry Chocolate	40	0	60	100	true
100006	100003	Potato Wafers	80	0	20	100	true
100007	100003	Ginger kurkure	80	0	25	100	true
100008	100003	Meda Puri	200	0	150	50	true
100009	100003	Khakhra	500	0	200	50	true
100001	100001	Suger Milk	300	55	50	100	true

Back

Show product table:

	Product_ID [PK] integer	Product_Department_ID integer	Product_Name character varying (50)	Product_Weight integer	Product_Stock integer	Product_Price integer	Product_Production_limit integer	Product_Production_flag boolean
1	100001	100001	Suger Milk	300	55	50	100	true
2	100002	100001	Vanilla ice-cream	300	0	70	100	true
3	100003	100002	Dark Chocolate	30	0	40	100	true
4	100004	100002	White Chocolate	30	0	30	100	true
5	100005	100002	Strawberry Chocolate	40	0	60	100	true
6	100006	100003	Potato Wafers	80	0	20	100	true
7	100007	100003	Ginger kurkure	80	0	25	100	true
8	100008	100003	Meda Puri	200	0	150	50	true
9	100009	100003	Khakhra	500	0	200	50	true

Thank You