FUNDAMENTALS OF DATABASE
PROJECT PART-3
HETERO PHARMACEUTICALS DATABASE

Group – 3
Team Members:

- Sreekar Thanda
- Shravan Kumar Nuka
- Manideep Renikindi
- Varsha Umannagari
- Rama Venkat Sumith Thota

Hetero Pharmaceuticals is a service-oriented company for medicinal drugs. Hetero provides pharma services across metro cities in INDIA. The Headquarters is located at Hyderabad. Each pharmacy has details of patients, prescriptions, doctors, day-to-day transactions, Inventory of Medicines, Insurance details. Additionally, its database holds information about the nearby clinics, which serve as entities in the whole database and have their respective attributes.
There are several entities used in this project and those are listed below,

Entity:

Hetero has enormous database as its spread across several locations in India. These entities comprise of strong and weak entities which are as listed below,

Constraints:

- We are using various constraints, they are primary key, foreign key, check and unique in this database.
- Primary keys are Employee id, Medicine code, Drug id etc.
- Foreign keys are Patient id, doctor name etc.
- Check constraint is used to check the inventory stock.

1. **HETERO**

Hetero is an Indian pharmacy around which our project revolves and here are the important aspects that are to be considered with Hetero. Each pharmacy has a set of attributes.
Attributes: ph_code, **Address, Contact, Zip Code.**
Primary key: Pharmacy code (ph_code). Here, each pharmacy store has a unique code to identify them among the listed pharmacies of that company.

Relationship: Hetero Pharmacy Store is in One-Many relationship with prescription as every pharmacy can receive multiple prescriptions from patients, One-Many relationship with employee since there are multiple employees in a single pharmacy and Many-Many relationship with inventory because there are several Medicines/Supplies in an inventory and store needs, several of such supplies.

2. **Employees data:** Employees are individuals working across several stores and they include cashiers, store managers etc... Each employee has a set of attributes.

Attributes: Employee Name, Designation, Employee Contact, Salary, Gender, Pharmacy Code.

Primary key: Employee Identity Number (Employee_ID). Each employee would have unique Employee_ID when compared to other employees in the store.

Relationship: This Employees entity shares Many to One relationship with Pharmacy store as there are several employees working in a single store.
shares Many-One relationship with Pharmacy Locations. Employees data is considered as strong entity as this an independent to pharmacy.

3. **Inventory:** Inventory is the excess stock/surplus supplies present in warehouse or storage unit for every pharmacy company from where the supplies are sent out to every store depending upon the requirement. Inventory stores the data of the quantity of medicines and related information with respect to medicines, vaccines etc. Each inventory possess a specific set of attributes.

Attributes: Item identity Number, Item name, Quantity, Availaility.
Primary Key: Item identity Number (Item_Id).Each medicine is identified by specific identity number for identification as several medicines from several companies have same or similar compositions and this id is used to distinguish them easily.
Relationship: This entity shares Many-Many relationship with Pharmacy since the inventory holds data for several medicines and the pharmacy possess several medicines and requires similar medicines.

4. **Prescription:** Prescription is a list of medicine that a patient need to use and that varies with person to person and disease to disease as doctor recommends to the patient.

Attributes: Prescription Identification Number (Prescription_Id), Date, Pharmacy Code.
Primary Key: Prescription Identification Number (Prescription_Id).
Relationship: It shares Many-One relationship with Pharmacy since there is a single is a single prescription and there can be many pharmacy stores to purchase that medicine. Prescription shares Ternary Relationship with Doctor and Patient and Prescription shares ternary relationship with Billing and patient.

5. **Patients**: Patients are those that suffer from any ailments and they the crucial for any pharmacy because patients give complete analysis to most of the entities in this E-R representation.

Attributes: **Patient Identification Number(Patient_ Id), Patient Name, Date of Birth, Disease, Gender, Contact,AGE**
With the Date Of Birth we could derive the age, Hence age could be a derived attribute.
Note: AGE is a derived Attribute here.
Primary key: **Patient Identification Number (Patient_ Id).**
Relationship: Patient has ternary relationship with Prescription and doctor and patient possess ternary relationship with billing and prescription.

6. **Doctor:** Doctor may or might not be present in the pharmacy, but he is the only authorized person to prescribe medication to patient. The following are the attributes.

Attributes: Doctor Registration Identity Number, Specialization,Contact.
Primary Key: Doctor Registration Identity Number (Doctor Reg_No).
Relationship: Doctor has only one Ternary Relationship with prescription and patient.

7. **Insurance:** Every patient has medical insurance that covers partial or full payment for the patient or their dependents. Each patient can have one or many insurances depending upon their necessity and preference. The attributes are as follows.

Attributes: Insurance Company, Amount, Insurance Number.
Primary Key: Insurance Number.
Relationship: Insurance has many to one relation with billing and Insurance possess Many-Many relationship with Patient.

8. **Billing:** This Entity describes about the transactions made by patient at pharmacy during purchase. Each medicine has a price depending on the drug and dosage and so this entity stores the cost price of the medicines and stores the information of day-to-day payments related to buying and selling the medicines.

Attributes: Date of Billing, Billing Amount, Transaction Identity Number, Transaction Number.
Primary Key: Transaction Identity Number (Transaction_Id).
Relationship: Billing entity has many to one relationship with insurance and billing possess ternary relationship with prescription and patient.

RELATIONSHIP TABLES

TERNARY RELATIONS.

1.TREATMENT: This is ternary relationship table for patient, doctor, and prescription.
2. PURCHASE: This is ternary relationship for patient, billing & Prescription.

MANY TO MANY RELATIONS.

1. INSURANCE CLAIM:

2. STOCK:

CONSTRAINTS:
The set of rules, ensures that when an authorized user modifies the database, they do not disturb the data consistency, A constraint is a rule that is used for optimization purposes.

Primary key: A primary key constraint is a column or combination of columns that has the same properties as a unique constraint. You can use primary key and foreign key constraints to define relationships between tables.
Foreign key: ***(*referential constraint *or a* referential integrity constraint*)*** is a logical rule about values in one or more columns in one or more tables. For example, a set of tables shares

information about a corporation's suppliers. Occasionally, a supplier's name changes. You can define a referential constraint that states the ID of the supplier in a table must match a supplier ID in the supplier information. This constraint prevents insert, update, or delete operations that would otherwise result in missing supplier information.

Unique ***(*****unique key constraint*****):*** This is a rule that forbids duplicate values in one or more columns within a table. Unique and primary keys are the supported unique constraints. For example, a unique constraint can be defined on the supplier identifier in the supplier table to ensure that the same supplier identifier is not given to two suppliers.

Check: A check constraint (also referred to as a **table check constraint**) is a database rule that specifies the values allowed in one or more columns of every row of a table. Specifying check constraints is done through a restricted form of a search condition.

Not null: A NOT NULL constraint is a rule that prevents null values from being entered into one or more columns within a table.

NEW ASSUMPTIONS: Project Update

RELATIONSHIPS:

TERNARY RELATIONSHIPS:

1. Doctor-Patient-Prescription
2. Patient-Prescription-Billing

ONE-MANY or MANY-MANY RELATIONSHIPS:

1. Hetero Pharmacy to Employee_data
2. Prescription to Hetero Pharmacy
3. Bills to Insurance

MANY-MANY RELATIONSHIPS:

1. Inventory to Hetero Pharmacy
2. Patient to Insurance

ONE-ONE RELATIONSHIP:

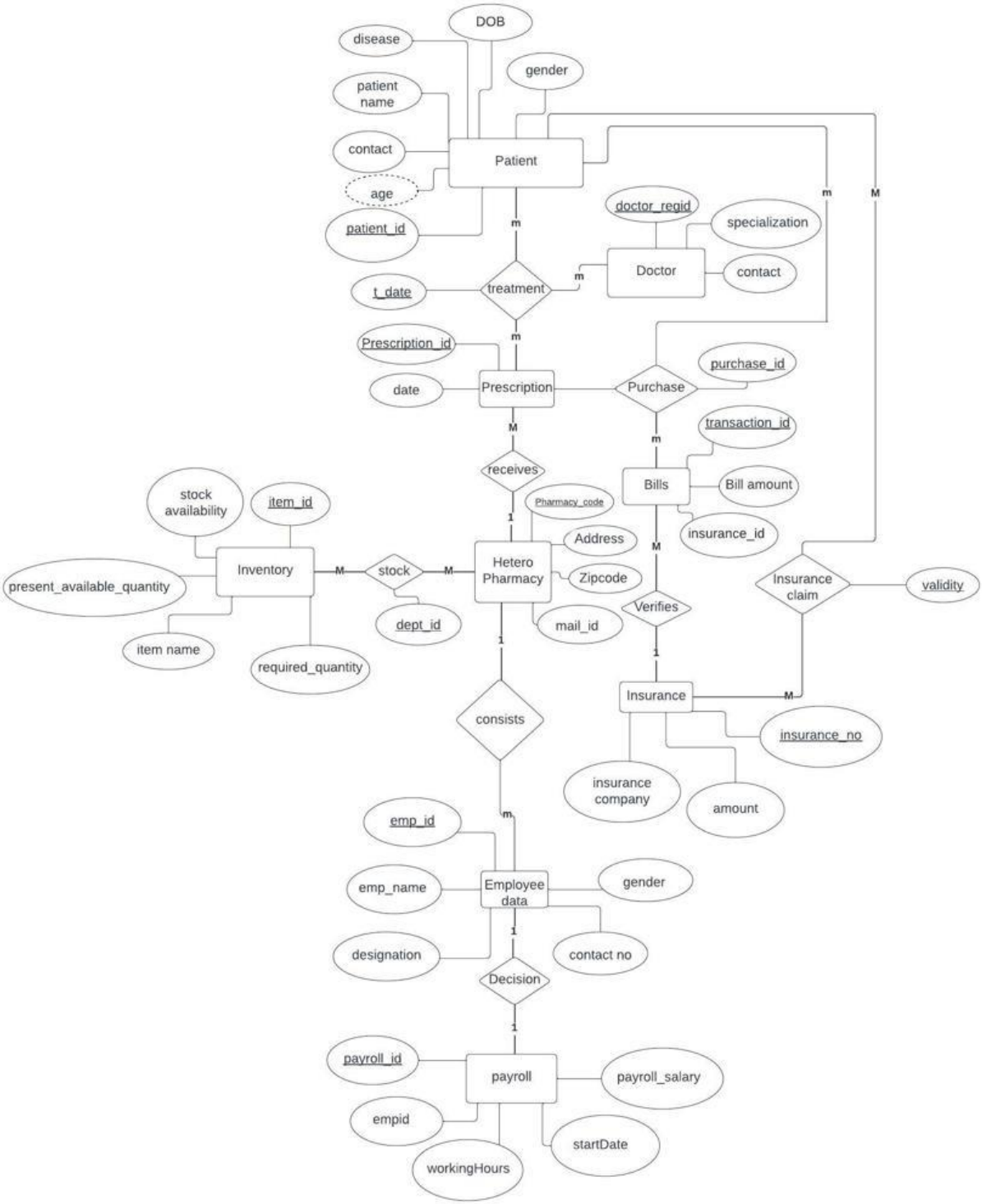
1. Employee_data to Payroll

Note: We have created a payroll table and inserted data into such that to ensure the data required to solve the given 7 questions. Also altered the prescription table by adding the column "doctor_id".

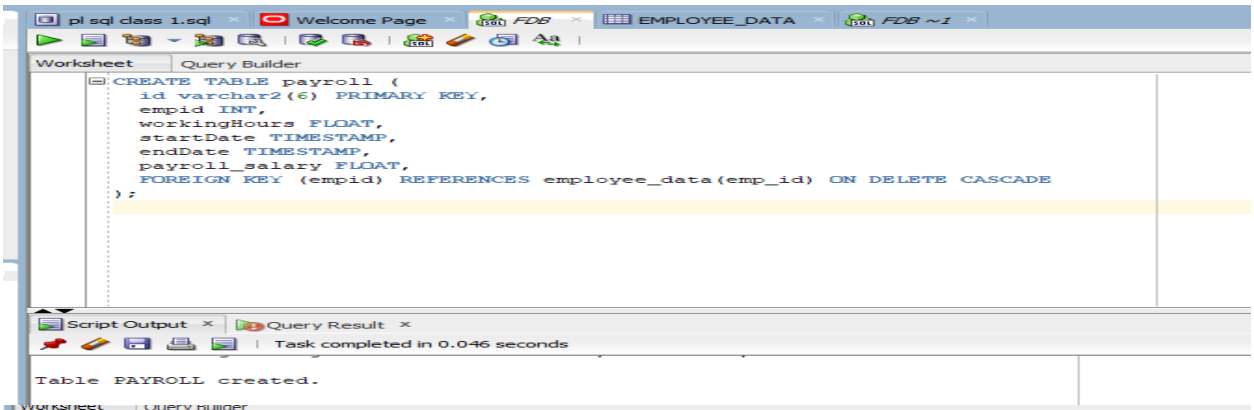
- ## PAYROLL TABLE

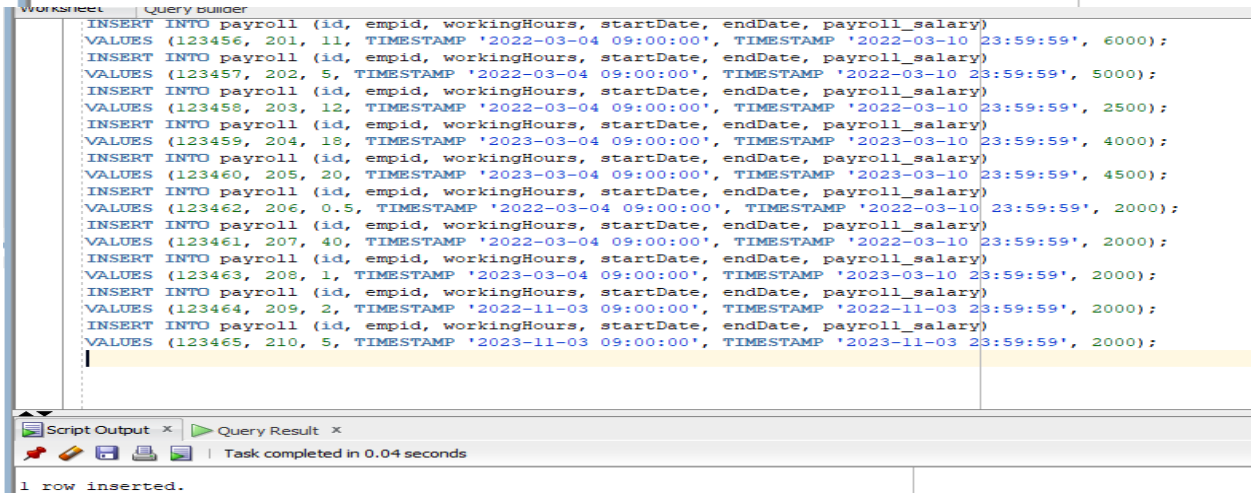
Every Employee in a pharmacy is paid with salaries based on their working hours and other factors.
Attributes: working hours, id, empid, start date, end date, payroll_salary
Primary Key: id
Foreign Key: empid from Employee_data table
Relationship: Every Employee is getting salaries, This data is stored in the form of payroll.

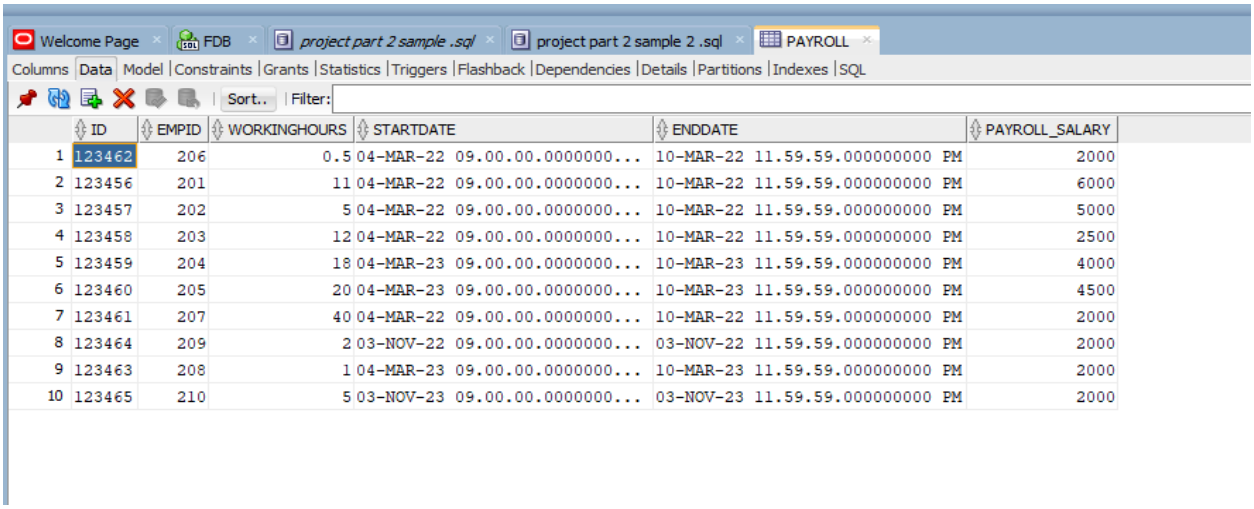
The updated E-R Diagram is shown below:



Creation and Insertion of **PAYROLL** Table:

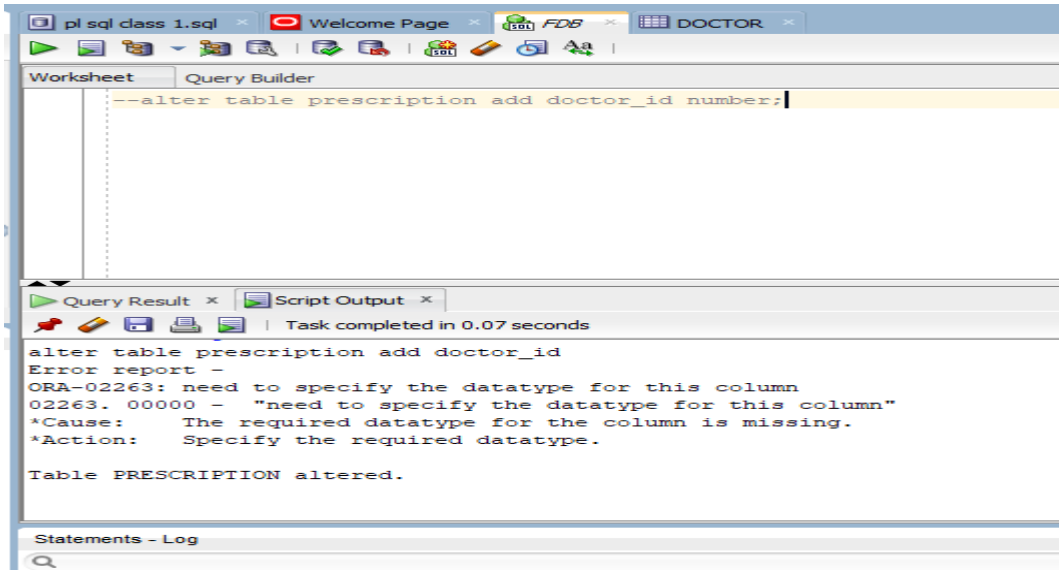






	ID	EMPID	WORKINGHOURS	STARTDATE	ENDDATE	PAYROLL_SALARY
1	123462	206	0.5	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	2000
2	123456	201	11	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	6000
3	123457	202	5	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	5000
4	123458	203	12	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	2500
5	123459	204	18	04-MAR-23 09.00.00.0000000...	10-MAR-23 11.59.59.000000000 PM	4000
6	123460	205	20	04-MAR-23 09.00.00.0000000...	10-MAR-23 11.59.59.000000000 PM	4500
7	123461	207	40	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	2000
8	123464	209	2	03-NOV-22 09.00.00.0000000...	03-NOV-22 11.59.59.000000000 PM	2000
9	123463	208	1	04-MAR-23 09.00.00.0000000...	10-MAR-23 11.59.59.000000000 PM	2000
10	123465	210	5	03-NOV-23 09.00.00.0000000...	03-NOV-23 11.59.59.000000000 PM	2000

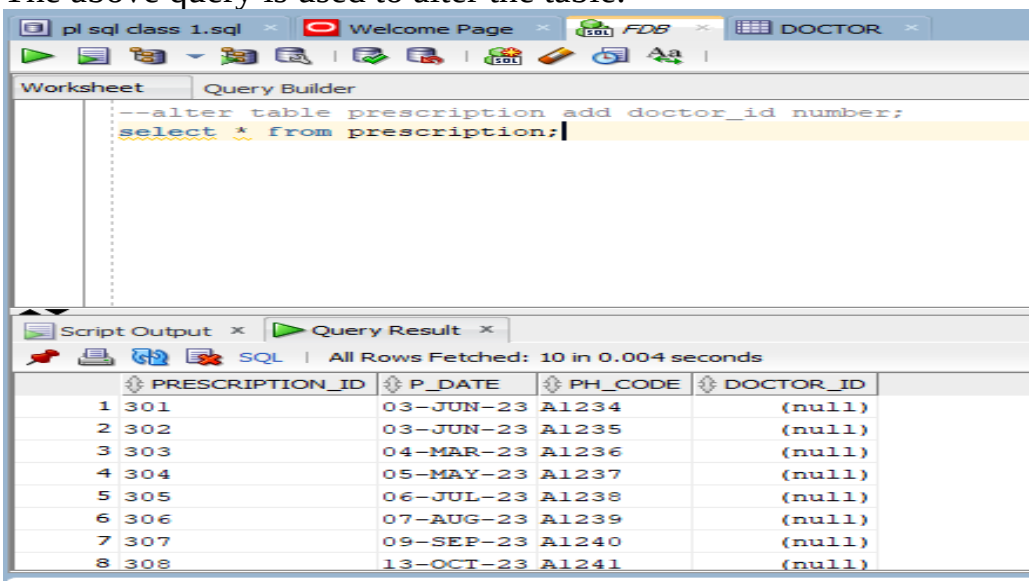
- Added a new column **doctor_id,** in the prescription table .



Altered the Prescription Table

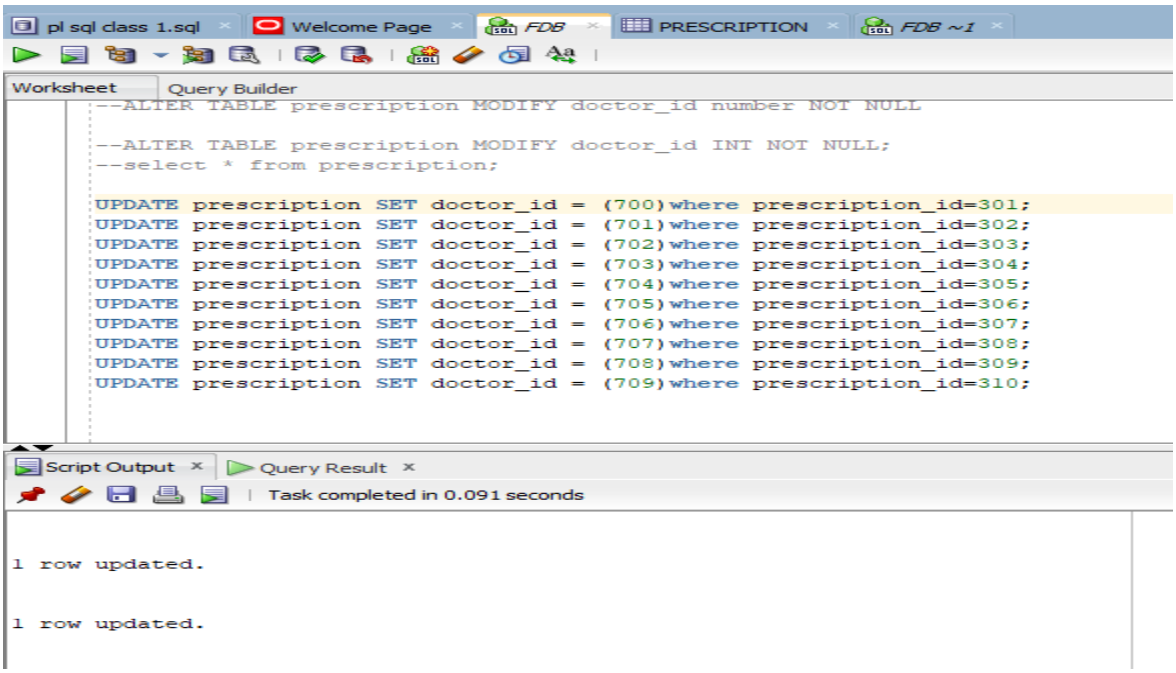
alter table prescription
add constraint fk_doctor_id
foreign key (doctor_id) references doctor(doctor_id) on delete cascade;

The above query is used to alter the table.

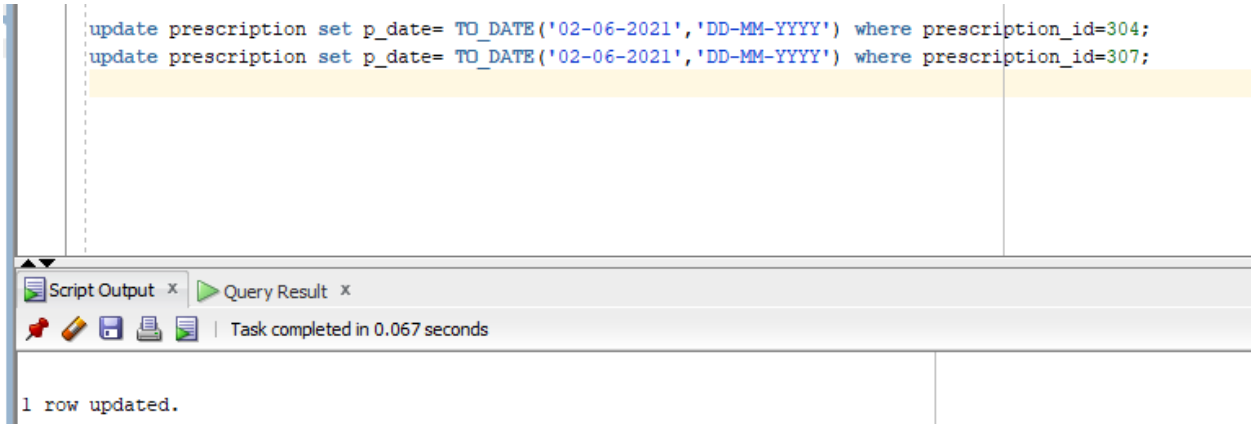


	PRESCRIPTION_ID	P_DATE	PH_CODE	DOCTOR_ID
1	301	03-JUN-23	A1234	(null)
2	302	03-JUN-23	A1235	(null)
3	303	04-MAR-23	A1236	(null)
4	304	05-MAY-23	A1237	(null)
5	305	06-JUL-23	A1238	(null)
6	306	07-AUG-23	A1239	(null)
7	307	09-SEP-23	A1240	(null)
8	308	13-OCT-23	A1241	(null)

Updated the values in the Prescription Table:

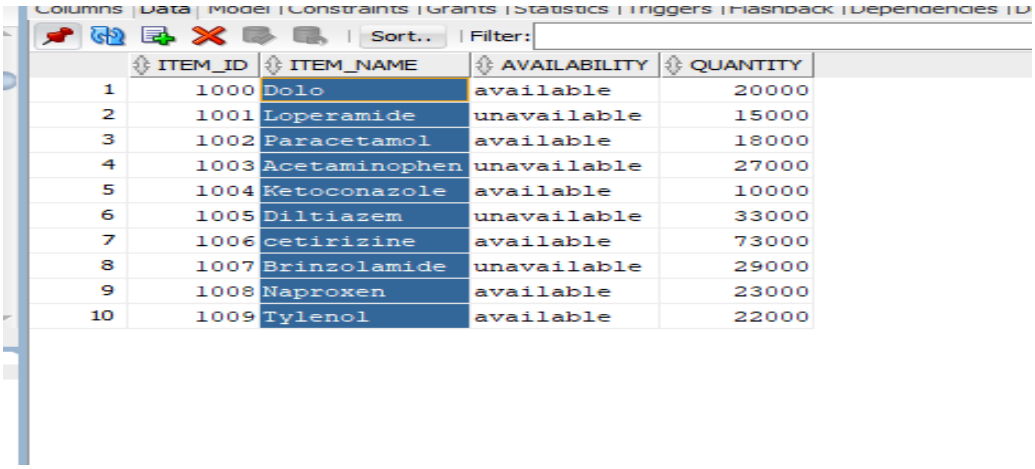


Upon Creation of the table and adding values to it, we updated 2 rows in the prescription table to adjust the data for required queries as shown below.



- **Altered the inventory table:**

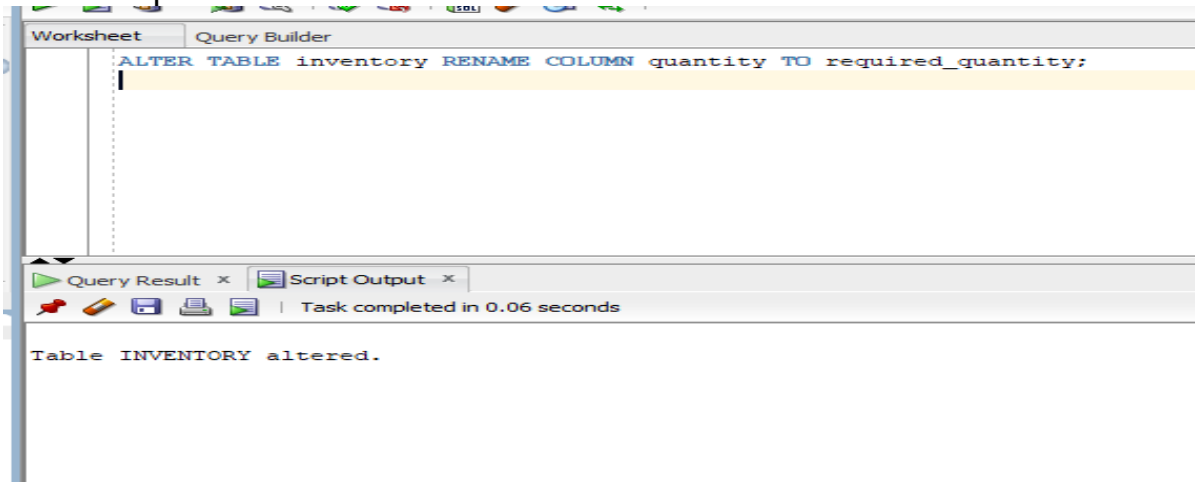
  Actual inventory table:



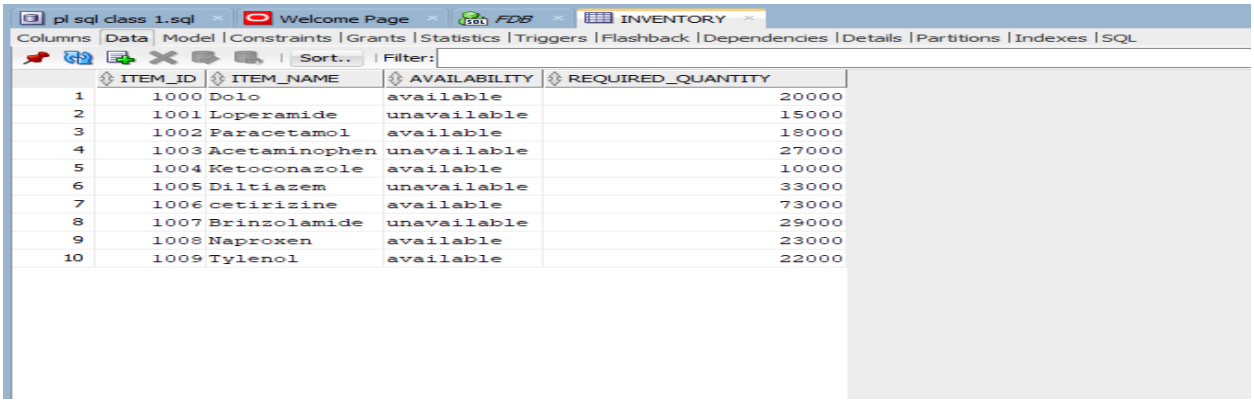
	ITEM_ID	ITEM_NAME	AVAILABILITY	QUANTITY
1	1000	Dolo	available	20000
2	1001	Loperamide	unavailable	15000
3	1002	Paracetamol	available	18000
4	1003	Acetaminophen	unavailable	27000
5	1004	Ketoconazole	available	10000
6	1005	Diltiazem	unavailable	33000
7	1006	cetirizine	available	73000
8	1007	Brinzolamide	unavailable	29000
9	1008	Naproxen	available	23000
10	1009	Tylenol	available	22000

Changes Incurred in the process.

Here we have renamed the column name "**Quantity**" in Inventory table and we have in corporated it in the column name quantity as **"Required_quantity**" and below are the implemented screenshots attached to it.

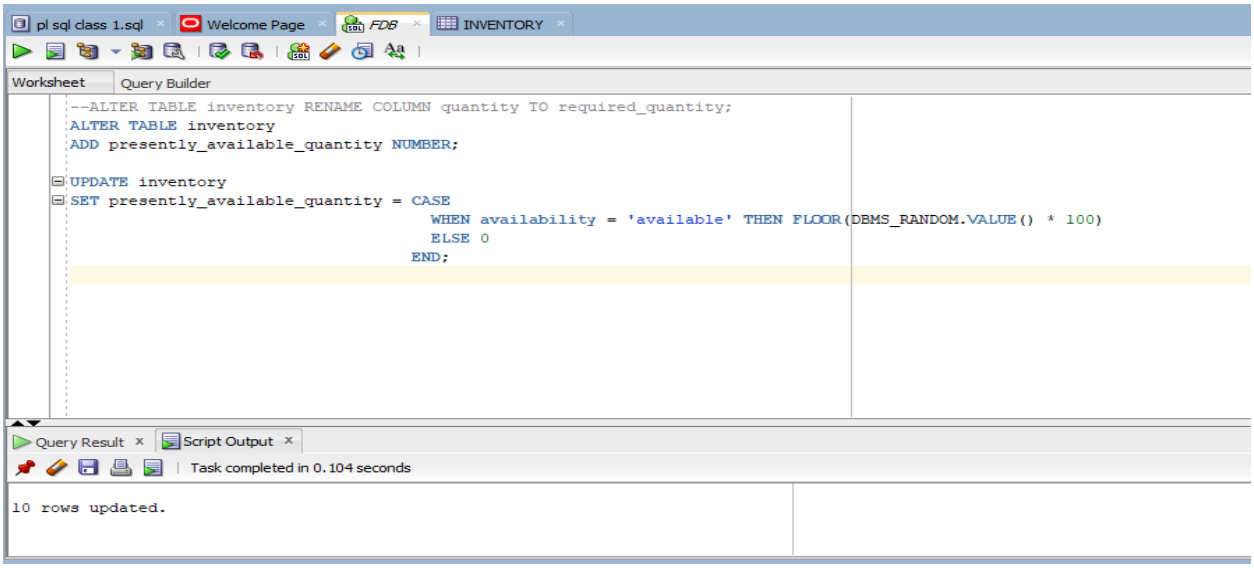


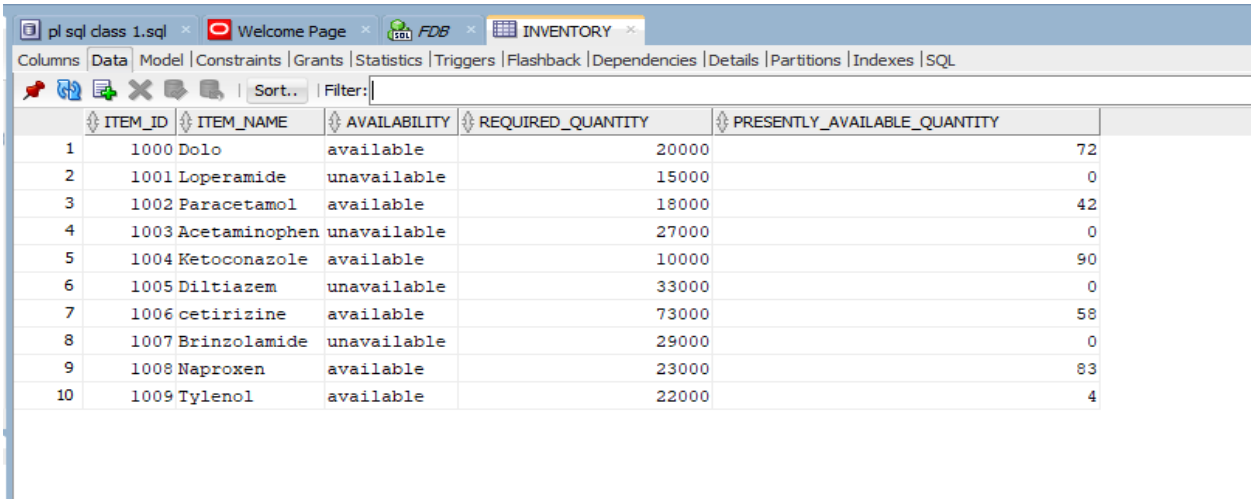
Altered Table is shown here.



	ITEM_ID	ITEM_NAME	AVAILABILITY	REQUIRED_QUANTITY
1	1000	Dolo	available	20000
2	1001	Loperamide	unavailable	15000
3	1002	Paracetamol	available	18000
4	1003	Acetaminophen	unavailable	27000
5	1004	Ketoconazole	available	10000
6	1005	Diltiazem	unavailable	33000
7	1006	cetirizine	available	73000
8	1007	Brinzolamide	unavailable	29000
9	1008	Naproxen	available	23000
10	1009	Tylenol	available	22000

Created a **new column** to the inventory table and named it as **presently_available_quantity** and this is successfully implemented.





	ITEM_ID	ITEM_NAME	AVAILABILITY	REQUIRED_QUANTITY	PRESENTLY_AVAILABLE_QUANTITY
1	1000	Dolo	available	20000	72
2	1001	Loperamide	unavailable	15000	0
3	1002	Paracetamol	available	18000	42
4	1003	Acetaminophen	unavailable	27000	0
5	1004	Ketoconazole	available	10000	90
6	1005	Diltiazem	unavailable	33000	0
7	1006	cetirizine	available	73000	58
8	1007	Brinzolamide	unavailable	29000	0
9	1008	Naproxen	available	23000	83
10	1009	Tylenol	available	22000	4

Values are inserted in the table and it’s clear that all the values are visible.

The Given 7 Questions:

1. **List the total number of prescriptions grouped by doctors and pharmacy location issued on June 2nd, 2021.**

```
SELECT d.doctor_name, h.address AS hetero_address, COUNT(*) AS total_prescriptions
FROM prescription p
JOIN doctor d ON p.doctor_id = d.doctor_id
JOIN hetero h ON p.ph_code= h.ph_code
WHERE p.p_date =TO_DATE('2021-06-02','YYYY-MM-DD')
GROUP BY d.doctor_name, h.address;
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.001 seconds

	DOCTOR_NAME	HETERO_ADDRESS	TOTAL_PRESCRIPTIONS
1	Bill Hunter	1500 oak st	2
2	Olivia	6550 stella st	1

Here in this query, we have joined prescription, doctor and hetero tables and filtered out that table for data containing issue date(p_date) as "2nd June 2021".

Considering the issue data as "02-06-2021" the total number of prescriptions grouped by doctor and pharmacy location is 2, 1 for doctor "Bill Hunter" at location/address "1500 oak st", doctor "Olivia" at address "6500 stella st" respectively.

2. **Find locations with inventories that list at least one missing product (a product that has quantity of zero in the inventory).**

```
SELECT h.address, i.item_name
FROM hetero h
JOIN stock s ON h.ph_code = s.ph_code
JOIN inventory i ON s.item_id = i.item_id
WHERE i.presently_available_quantity= 0;
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0.011 seconds

	ADDRESS	ITEM_NAME
1	3500N Bonniebrae st	Acetaminophen
2	1500 oak st	Loperamide
3	1230 north elem st	Brinzolamide
4	4000 southloop st	Diltiazem

In this query, we have displayed address from hetero table and item_name from inventory by using Join Mechanism. We have joined Hetero on Stock and inventory tables.

The following are the pharmacy locations that do not possess the following listed inventory Items at the locations are:
Acetaminophen, loperamide, brinzolamide, diltiazem at Bonniebrae st, Oak st, north elem st, Southloop st respectively.

3. **Find the name of the employee(s) that had worked the most hours on November 3, 2022**

```
SELECT employee_data.emp_name, payroll.workingHours
FROM payroll
INNER JOIN employee_data ON payroll.empid = employee_data.emp_id
WHERE payroll.startDate >= TO_TIMESTAMP('2022-11-03 00:00:00', 'YYYY-MM-DD HH24:MI:SS')
  AND payroll.endDate <= TO_TIMESTAMP('2022-11-03 23:59:59', 'YYYY-MM-DD HH24:MI:SS')
ORDER BY payroll.workingHours DESC
FETCH FIRST 1 ROW ONLY;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.004 seconds

	EMP_NAME	WORKINGHOURS
1	Varsha	2

We have displayed employee names along with their salares who worked more hours on November 3rd, 2022. For this, we have joined payroll table on employee_data table and sorted the resulted data to descending order withrespect to working Hours.

The employee who worked for most hours on November 3, 2022 is Varsha.

4. **List the items that currently have the least quantity on inventory.**

```
SELECT item_name, presently_available_quantity
FROM inventory
WHERE presently_available_quantity = (SELECT MIN(presently_available_quantity) FROM inventory);
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0.003 seconds

	ITEM_NAME	PRESENTLY_AVAILABLE_QUANTITY
1	Loperamide	0
2	Acetaminophen	0
3	Diltiazem	0
4	Brinzolamide	0

In this query we have used sub-query mechanism to find the required data. Select min(presently_avaialable_quantity) from inventory will display the minimum value of

presently_available_quantity column and the outer query will display only the item name and values presently_available_quantity.
The following are the medicines that have zero (0) stock in the inventory are Loperamide, Acetaminophen, diltiazem, and Brinzolamide respectively.

5. **Print the payroll from March 4, 2022 to March 10, 2022 displaying employee name, hours worked and total salary for all employees.**

```
SELECT employee_data.emp_name, payroll.workingHours, payroll.payroll_salary
FROM payroll, employee_data where payroll.empid = employee_data.emp_id
AND payroll.startDate >= TIMESTAMP '2022-03-04 00:00:00'
  AND payroll.endDate <= TIMESTAMP '2022-04-10 23:59:59';
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0.003 seconds

	EMP_NAME	WORKINGHOURS	PAYROLL_SALARY
1	Vignesh	0.5	2000
2	Arun	11	6000
3	Tarun	5	5000
4	Varun	12	2500
5	Charitha	40	2000

The list of employees who worked from march 4,2022 to march 10, 2022 are :
vignesh worked for .5 hours and his salary is 2000,
arun worked for 11 hours and his salary is 6000,
tarun worked for 5 hours and his salary is 5000,
varun worked for 12 hours and his salary is 2500,
charitha worked for 40 hours and his salary is 2000.

6. **Design a delete statement to delete employees working less than 5 hours from March 4, 2023 to March 10, 2023.**

Employee from the payroll table whose working hours are less than 5 from March 4,2023 to March 10,2023 is **employee id : 208 with 1 hour.**

```
--question 6;
select * FROM payroll
WHERE empid IN (
  SELECT emp_id
  FROM employee_data
  WHERE emp_id IN (
    SELECT empid
    FROM payroll
    WHERE startDate >= TO_TIMESTAMP('2023-03-04 00:00:00', 'YYYY-MM-DD HH24:MI:SS')
    AND endDate <= TO_TIMESTAMP('2023-03-10 23:59:59', 'YYYY-MM-DD HH24:MI:SS')
    AND workingHours < 5
  )
);
```

Query Result | SQL | All Rows Fetched: 1 in 0.001 seconds

	ID	EMPID	WORKINGHOURS	STARTDATE	ENDDATE	PAYROLL_SALARY
1	123463	208	1	04-MAR-23 09.00.00.000000000 AM	10-MAR-23 11.59.59.000000000 PM	2000

We have used subquery mechanism to find the employees who is working less than 5hours from March 4th to March 10th 2023.
Inner query find the empid who who is working less than 5hours from March 4th to March 10th 2023. Outer query matches the resulted empid with the emp_id in employee_data table.
Now, deleting the **Employees working less than 5 hours from March 4, 2023 to March 10, 2023**.

```
DELETE FROM payroll
WHERE empid IN (
  SELECT emp_id
  FROM employee_data
  WHERE emp_id IN (
    SELECT empid
    FROM payroll
    WHERE startDate >= TO_TIMESTAMP('2023-03-04 00:00:00', 'YYYY-MM-DD HH24:MI:SS')
    AND endDate <= TO_TIMESTAMP('2023-03-10 23:59:59', 'YYYY-MM-DD HH24:MI:SS')
    AND workingHours < 5
  )
);
```

Query Result | Script Output | Task completed in 0.082 seconds

```
1 row deleted.
```

Here the **emp_id 208** employee details are deleted from the payroll table. When checked the table data **after executing delete query**, it's confirmed **that the row is deleted.**

Welcome Page | FDB | project part 2 sample 2 .sql | PAYROLL | project part 2 sample .sql

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	ID	EMPID	WORKINGHOURS	STARTDATE	ENDDATE	PAYROLL_SALARY
1	123462	206	0.5	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	2000
2	123456	201	11	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	6000
3	123457	202	5	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	5000
4	123458	203	12	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	2500
5	123459	204	18	04-MAR-23 09.00.00.0000000...	10-MAR-23 11.59.59.000000000 PM	4000
6	123460	205	20	04-MAR-23 09.00.00.0000000...	10-MAR-23 11.59.59.000000000 PM	4500
7	123461	207	40	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	2000
8	123464	209	2	03-NOV-22 09.00.00.0000000...	03-NOV-22 11.59.59.000000000 PM	2000
9	123465	210	5	03-NOV-23 09.00.00.0000000...	03-NOV-23 11.59.59.000000000 PM	2000

7. **Design an update statement to give a 23% salary raise to employees working more than 5 hours from March 4, 2023 to March 10, 2023.**

```
--question 7:

SELECT e.emp_name,p.empid, p.workingHours, p.payroll_salary
FROM employee_data e
JOIN payroll p ON e.emp_id = p.empid
WHERE p.startDate >= TO_TIMESTAMP('2023-03-04 00:00:00', 'YYYY-MM-DD HH24:MI:SS')
  AND p.endDate <= TO_TIMESTAMP('2023-03-10 23:59:59', 'YYYY-MM-DD HH24:MI:SS')
  AND p.workingHours > 5
GROUP BY e.emp_name,p.empid, p.workingHours, p.payroll_salary
HAVING COUNT(*) = 1;
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.005 seconds

	EMP_NAME	EMPID	WORKINGHOURS	PAYROLL_SALARY
1	Varsini	204	18	4000
2	Fahien	205	20	4500

We have joined Employee_data with Payroll to that match the empid of both tables to extract the required data.

The list of employees that receive 23% hike for working for more than 5 hours from march 4,2023 to march 10, 2023 are Varshini and Fahien.

Now **updating their salaries to 23% hike.**

```
UPDATE payroll
SET payroll_salary = payroll_salary * 1.23
WHERE empid IN (SELECT emp_id FROM employee_data)
AND workingHours > 5
AND startDate >= TO_TIMESTAMP('2023-03-04 00:00:00', 'YYYY-MM-DD HH24:MI:SS')
AND endDate <= TO_TIMESTAMP('2023-03-10 23:59:59', 'YYYY-MM-DD HH24:MI:SS');
```

Script Output | Query Result

Task completed in 0.025 seconds

```
2 rows updated.
```

To update their salaries we used Update statements with subquery.

Employees having employee id = 204,205 salaries have been hiked to 4920,5535 respectively.

Welcome Page | FDB | *project part 2 sample 2 .sql* | project part 2 sample .sql | PAYROLL

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	ID	EMPID	WORKINGHOURS	STARTDATE	ENDDATE	PAYROLL_SALARY
1	123462	206	0.5	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	2000
2	123456	201	11	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	6000
3	123457	202	5	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	5000
4	123458	203	12	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	2500
5	123459	204	18	04-MAR-23 09.00.00.0000000...	10-MAR-23 11.59.59.000000000 PM	4920
6	123460	205	20	04-MAR-23 09.00.00.0000000...	10-MAR-23 11.59.59.000000000 PM	5535
7	123461	207	40	04-MAR-22 09.00.00.0000000...	10-MAR-22 11.59.59.000000000 PM	2000
8	123464	209	2	03-NOV-22 09.00.00.0000000...	03-NOV-22 11.59.59.000000000 PM	2000
9	123465	210	5	03-NOV-23 09.00.00.0000000...	03-NOV-23 11.59.59.000000000 PM	2000

10 SQL queries

1. **Display pharmacy is located either in 76247 or 67890.**

```
SELECT *
FROM hetero
WHERE zipcode =76247 OR zipcode = 67890;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.004 seconds

	PH_CODE	PH_CONTACT	ADDRESS	ZIPCODE
1	A1237	839-283-2822	1500 oak st	76247

Displayed the pharmacy address which is in 76247 or 67890 zipcode.

2. **Display male employees whose salary is less than 5000 and in ascending order wr.t their salaries.**

```
SELECT * FROM employee_data
WHERE salary < 5000 AND gender = 'male'
order by salary;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.01 seconds

	EMP_ID	EMP_NAME	DESIGNATION	EMP_CONTACT	SALARY	GENDER	PH_CODE
1	206	Vignesh	Sales_Man-1	123-346-3411	2000	male	A1239
2	203	Varun	Receiptionist	123-643-1234	2500	male	A1236
3	205	Fahien	Sales Manager	975-222-2348	4500	male	A1238

Displayed male employees whose salary is 5000 and are ordered in ascending order with respect to salaries.

3. **Display doctor details whose specialization is Dentist or ENT and has contact number 978 or 666.**

```
SELECT doctor_name, specialization, contact
FROM Doctor
WHERE (specialization = 'dentist' OR specialization = 'ENT') AND (contact LIKE '978%' OR contact LIKE '%666');
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.006 seconds

	DOCTOR_NAME	SPECIALIZATION	CONTACT
1	Olivia	ENT	9787666666
2	Elizabeth	dentist	9087666666

LIKE operator is used to compare a contact numbers with the pattern 978 and 666
Displayed the details of doctor whose specialization is either dentist or ENT and they are having contact number 978 or 666

4. **SQL Query Displaying the List of Pharmacies whose present Stock Availability is zero.**

```
--4)
SELECT h.ph_code, h.ph_contact, h.address, h.zipcode
FROM hetero h
JOIN stock s ON h.ph_code = s.ph_code
JOIN inventory i ON s.item_id=i.item_id
WHERE i.presently_available_quantity = 0;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.004 seconds

	PH_CODE	PH_CONTACT	ADDRESS	ZIPCODE
1	A1237	839-283-2822	1500 oak st	76247

The join mechanism is used and we have joined hetero with stock tables matching with Pharmacy codes and the resulted table is joined with inventory matching item_id.

The Ph_code: 'A1237' has the stock, present available quantity is zero.

5. **SQL Query to Display the employee name and their salary whose salary is greater than the average of total employees salary.**

```
--5)
SELECT emp_name, salary
FROM employee_data
WHERE salary > (
  SELECT AVG(salary)
  FROM employee_data
);
--
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.002 seconds

	EMP_NAME	SALARY
1	Arun	6600
2	Tarun	5000
3	Fahien	4500

We have used subquery mechanism in above query. Inner query find the average of all salaries in the employee_data table and outer query finds the employee_name and their salary of whose salary is greater than the average salaries.
Employees Arun, Tarun, Fahien has salaries greater than the average of salaries.

6. **SQL query retrieves the total bill amounts of each insurance company whose total bill amounts are greater than or equal to 500, sorted in descending order of the total bill amounts.**

```
SELECT i.insurance_company, SUM(b.bill_amount) AS total_bill_amount
FROM bills b, insurance i
WHERE b.insurance_id = i.insurance_id
GROUP BY i.insurance_company
HAVING SUM(b.bill_amount) >=500
ORDER BY total_bill_amount DESC;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.017 seconds

	INSURANCE_COMPANY	TOTAL_BILL_AMOUNT
1	hdfc credila	1710
2	point32health	932
3	upmc health care	765

We have used group by, having , order by in above queries.
Group by is used to group together the same insurance providers. Having finds if the sum of bill amount is greater than or equal to 500 and order by is used to sort the resulted data in descending order with respect to total_bill_amount.

Insurance providers HDFC Credilla, oint32health, upsc health care are provided totally 1710,932,765 to their patients.

7. SQL Query to Display the no. of Patients suffering from respective disease.

```
insert into patient values ('Sravs','2023','chronic kidney disease',TO_DATE('21-11-1998','DD-MM-YYYY'),9332129909,'male');
SELECT disease, COUNT(*) as patient_count
FROM patient
WHERE disease IS NOT NULL
GROUP BY disease
HAVING COUNT(*) >=1
ORDER BY patient_count DESC;
```

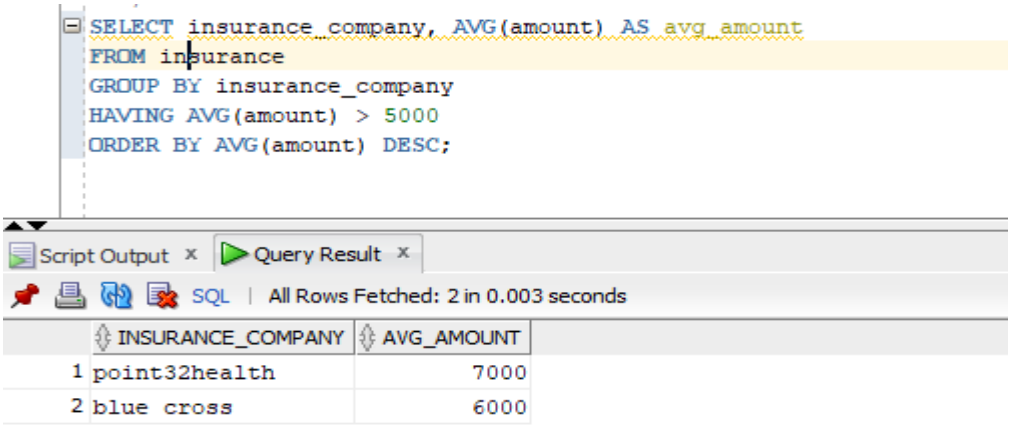
Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.001 seconds

	DISEASE	PATIENT_COUNT
1	chronic kidney disease	2
2	skin cancer	1
3	chronic respiratory disease	1
4	alzheimers disease	1
5	[illegible]	1

In the above query, we have displayed no. of Patients suffering from respective diseases. We have used group by to group together patients having same disease and having is used to verify if the count > 1 and order by is used to sort the resulted data in descending order of patient_count.
So, there 2 patients suffering from chronic kidney disease, 2 are suffering from skin cancer, 1 from chronic respiratory disease etc.

8. SQL query is to obtain the insurance companies which provide insurance greater than 5000.



```
SELECT insurance_company, AVG(amount) AS avg_amount
FROM insurance
GROUP BY insurance_company
HAVING AVG(amount) > 5000
ORDER BY AVG(amount) DESC;
```

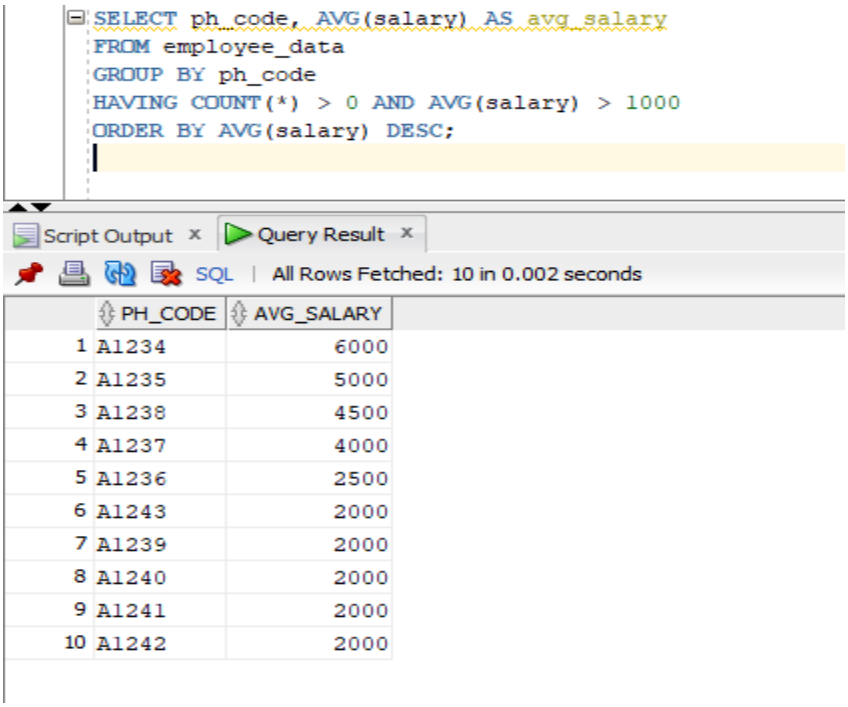
Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.003 seconds

	INSURANCE_COMPANY	AVG_AMOUNT
1	point32health	7000
2	blue cross	6000

In the above screenshot, we have used group by to group together insurance_company and having to check if group data is greater than the average of amount and order by is used to sort the resulted data in descending order of average amount.
Point32health and blue cross are top insurance providers who are providing average amount 7000,6000 respectively..

9. **SQL query to select the pharmacy codes and their average employee salary, but only for pharmacies that have at least one employee and whose average salary is greater than 1000. The results are ordered by average salary in descending order.**



```
SELECT ph_code, AVG(salary) AS avg_salary
FROM employee_data
GROUP BY ph_code
HAVING COUNT(*) > 0 AND AVG(salary) > 1000
ORDER BY AVG(salary) DESC;
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.002 seconds

	PH_CODE	AVG_SALARY
1	A1234	6000
2	A1235	5000
3	A1238	4500
4	A1237	4000
5	A1236	2500
6	A1243	2000
7	A1239	2000
8	A1240	2000
9	A1241	2000
10	A1242	2000

We have used groupby pharmacy codes and used having to find if the count of employees is greater than zero and average of salaries is greater than 1000, order by is used to sort the resulted data to average of salaries to descending order.

Listed the pharmacy codes with their respective employee's average salary expenditure.

10. Display total number of bills and their insurance provider names coming from same insurance.

```
SELECT COUNT(*) AS total_bills, insurance_company
FROM bills
JOIN insurance ON bills.insurance_id = insurance.insurance_id
GROUP BY insurance_company
HAVING COUNT(DISTINCT insurance_company) > 0;
```

Script Output | Query Result | SQL | All Rows Fetched: 7 in 0.005 seconds

	TOTAL_BILLS	INSURANCE_COMPANY
1	4	hdfc credila
2	1	care source
3	2	point32health
4	1	united health
5	1	blue cross
6	1	upmc health care
7	1	Axis Health

We have joined the insurance table with bills table matching the insuranceid in both tables and grouped together with insurance company, having is used to count(number of distinct insurance providers names) is greater than zero.
Hence, The HDFC credilla is associated to 4 bill payments, care source with 1 bill, point32health with 2, united health with 1, blue cross with 1, upmc health care with 1, axis health 1.

Update Statements:

1. Updating the Doctor table by setting doctor name = 'Sophia' for doctor_id = 706

```
select * from doctor where specialization = 'cardiologist';

UPDATE Doctor
SET doctor_name = 'Sophia', specialization = 'cardiologist'
WHERE doctor_id = 706;
--
select * from doctor where specialization = 'cardiologist';
```

Script Output | Query Result 1 | Task completed in 0.377 seconds

```
1 row updated.

>>Query Run In:Query Result 1
```

```
select * from doctor where specialization = 'cardiologist';

UPDATE Doctor
SET doctor_name = 'Sophia', specialization = 'cardiologist'
WHERE doctor_id = 706;
--
select * from doctor where specialization = 'cardiologist';
```

Script Output | Query Result 1

SQL | All Rows Fetched: 2 in 0.001 seconds

	DOCTOR_NAME	SPECIALIZATION	DOCTOR_ID	CONTACT
1	John	cardiologist	700	2837489655
2	Sophia	cardiologist	706	9787666666

Updated the doctor's name as Sophia from Olivia who's specialization is cardiologist

2. Updating doctor table by considering the doctors with required doctor id who has multiple designations. For example, surgeon is being added to the required doctors.

```
);

select * from doctor;
```

Script Output | Query Result 4 | Query Result 5

SQL | All Rows Fetched: 10 in 0.001 seconds

	DOCTOR_NAME	SPECIALIZATION	DOCTOR_ID	CONTACT
1	John	cardiologist	700	2837489655
2	Peter	ENT	701	8786431467
3	Cinderella	oncologist	702	9387328964
4	Bill Hunter	dentist	703	7859766658
5	Alex	neurologist	704	2787666439
6	Joe	dentist	705	5387668736
7	Sophia	cardiologist	706	9787666666
8	Mark Henry	neurologist	707	6855437466
9	Elizabeth	dentist	708	9087666666
10	Ashley	ENT	709	4238366346

```
select * from doctor;
UPDATE Doctor
SET doctor_name = doctor_name || ' (Surgeon) '
WHERE doctor_id IN (
  SELECT MAX(doctor_id)
  FROM Doctor
  GROUP BY specialization
  HAVING COUNT(*) > 1
);

select * from doctor;
```

Script Output | Query Result 4 | Query Result 5

Task completed in 0.364 seconds

```
>>Query Run In:Query Result 3

Table DOCTOR altered.


4 rows updated.

>>Query Run In:Query Result 5
```

Inner query is used with group by and having to find the doctors with respect to specialization where specialization is greater than 2 and outer query updates the doctors name to surgeon.

```
select * from doctor;
```

Script Output | Query Result 4 | Query Result 5

SQL | All Rows Fetched: 10 in 0.002 seconds

	DOCTOR_NAME	SPECIALIZATION	DOCTOR_ID	CONTACT
1	John	cardiologist	700	2837489655
2	Peter	ENT	701	8786431467
3	Cinderella	oncologist	702	9387328964
4	Bill Hunter	dentist	703	7859766658
5	Alex	neurologist	704	2787666439
6	Joe	dentist	705	5387668736
7	Sophia (Surgeon)	cardiologist	706	9787666666
8	Mark Henry (Surgeon)	neurologist	707	6855437466
9	Elizabeth (Surgeon)	dentist	708	9087666666
10	Ashley (Surgeon)	ENT	709	4238366346

The doctor names who are having same specialization are assigned to surgeon and this designation is added to their names.

3. Updating the bill amount to 1.1 times if the total insurance amount to be claimed by a person exceeds 5000.

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 10 in 0.007 seconds

	TRANS_ID	BILL_AMOUNT	INSURANCE_ID
1	234567	234	605
2	234578	500	601
3	234589	489	607
4	234512	122	602
5	234523	523	601
6	234534	100	603
7	234545	443	607
8	234389	235	608
9	234553	765	609
10	234999	565	602

```
select * from bills;
UPDATE bills
  SET bill_amount = bill_amount * 1.1
  WHERE insurance_id IN (
       SELECT insurance_id
       FROM insurance
       GROUP BY insurance_id
       HAVING SUM(amount) >= 5000
  );
select * from bills;
```

Script Output | Query Result | Query Result 1

Task completed in 0.385 seconds

```
4 rows updated.

>>Query Run In:Query Result 1
```

In the above query, We have used in inner query to find insurance_id, grouped by insurance_id and whose sum of the amount is greater than equal to 5000. The outer queries to update that bill amounts to multiple with 1.1 times of existing original values.

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 10 in 0.001 seconds

	TRANS_ID	BILL_AMOUNT	INSURANCE_ID
1	234567	234	605
2	234578	500	601
3	234589	537.9	607
4	234512	122	602
5	234523	523	601
6	234534	100	603
7	234545	487.3	607
8	234389	258.5	608
9	234553	841.5	609
10	234999	565	602

4. Update the employee_data table by changing the salary of a manager to 1.1 times.

```
select * from employee_data;
--UPDATE employee_data
-- SET salary = salary * 1.1
-- WHERE emp_id IN (
--   SELECT emp_id
--   FROM employee_data
--   WHERE designation = 'Manager'
-- );
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.001 seconds

	EMP_ID	EMP_NAME	DESIGNATION	EMP_CONTACT	SALARY	GENDER	PH_CODE
1	201	Arun	Manager	9222922222	6000	male	A1234
2	202	Tarun	Assistant Manager	234-234-6464	5000	male	A1235
3	203	Varun	Receiptionist	123-643-1234	2500	male	A1236
4	204	Varsini	Stock Manager	342-282-9382	4000	female	A1237
5	205	Fahien	Sales Manager	975-222-2348	4500	male	A1238
6	206	Vignesh	Sales_Man-1	123-346-3411	2000	male	A1239
7	207	Charitha	Sales_Man-2	321-067-3933	2000	female	A1240
8	208	Vinisha	Sales_Man-3	985-363-2933	2000	female	A1241
9	209	Varsha	Sales_Man-4	322-229-2223	2000	female	A1242
10	210	Auish	Sales_Man-5	763-272-2721	2000	TS	A1243

```
--select * from employee_data;
UPDATE employee_data
 SET salary = salary * 1.1
 WHERE emp_id IN (
   SELECT emp_id
   FROM employee_data
   WHERE designation = 'Manager'
 );
select * from employee_data;
```

Script Output | Query Result | Task completed in 0.482 seconds

```
4 rows updated.

>>Query Run In:Query Result 5
```

We have used subquery, where inner query is used to find the emp_id of employees who are Managers and multiplying their slaries with 1.1 times of existing salaries.

Script Output | Query Result | SQL | All Rows Fetched: 10 in 0.001 seconds

	EMP_ID	EMP_NAME	DESIGNATION	EMP_CONTACT	SALARY	GENDER	PH_CODE
1	201	Arun	Manager	9222922222	6600	male	A1234
2	202	Tarun	Assistant Manager	234-234-6464	5000	male	A1235
3	203	Varun	Receiptionist	123-643-1234	2500	male	A1236
4	204	Varsini	Stock Manager	342-282-9382	4000	female	A1237
5	205	Fahien	Sales Manager	975-222-2348	4500	male	A1238
6	206	Vignesh	Sales_Man-1	123-346-3411	2000	male	A1239
7	207	Charitha	Sales_Man-2	321-067-3933	2000	female	A1240
8	208	Vinisha	Sales_Man-3	985-363-2933	2000	female	A1241
9	209	Varsha	Sales_Man-4	322-229-2223	2000	female	A1242
10	210	Auish	Sales_Man-5	763-272-2721	2000	TS	A1243

5. Update the patient table by changing the disease from valve disease to flu.

```
select * from patient where disease = 'valve disease';
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.002 seconds

	PATIENT_NAME	PATIENT_ID	DISEASE	DATE_OF_BIRTH	CONTACT	GENDER
1	williamson	2022	valve disease	12-JUL-97	9493939393	male

```
UPDATE patient
SET disease = 'Flu'
WHERE patient_id IN (
  SELECT patient_id
  FROM patient
  WHERE disease LIKE 'valve disease'
);
```

Script Output | Task completed in 0.039 seconds

```
1 row updated.
```

In this query, we have used subquery where inner query is used to find the patient_id who's disease is like valve disease. And outer query is set the value to 'Flu'.

```
select * from patient where disease = 'Flu';
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.007 seconds

	PATIENT_NAME	PATIENT_ID	DISEASE	DATE_OF_BIRTH	CONTACT	GENDER
1	williamson	2022	Flu	12-JUL-97	9493939393	male

6. Update the inventory table if the inventory is filled with 15000 with respect to the item and pharmacy codes starts with A12.

```
select * from inventory;
```

Script Output | Query Result | SQL | All Rows Fetched: 10 in 0.002 seconds

	ITEM_ID	ITEM_NAME	AVAILABILITY	REQUIRED_QUANTITY	PRESENTLY_AVAILABLE_QUANTITY
1	1000	Dolo	available	20000	72
2	1001	Loperamide	unavailable	15000	0
3	1002	Paracetamol	available	18000	42
4	1003	Acetaminophen	unavailable	27000	0
5	1004	Ketoconazole	available	10000	90
6	1005	Diltiazem	unavailable	33000	0
7	1006	cetirizine	available	73000	58
8	1007	Brinzolamide	unavailable	29000	0
9	1008	Naproxen	available	23000	83
10	1009	Tylenol	available	22000	4

```
UPDATE inventory
SET presently_available_quantity = 15000
WHERE item_id IN (
  SELECT item_id
  FROM stock
  WHERE ph_code LIKE 'A12%'
  GROUP BY item_id
  HAVING SUM(required_quantity) BETWEEN 20000 AND 50000
);
```

Script Output | Task completed in 0.03 seconds

```
6 rows updated.
```

Inner query is used to find the item_id in stock whose ph_code flows pattern ‘A12’ and groupby with item_id and those having sum of required_quantity is between 20000 and 50000.

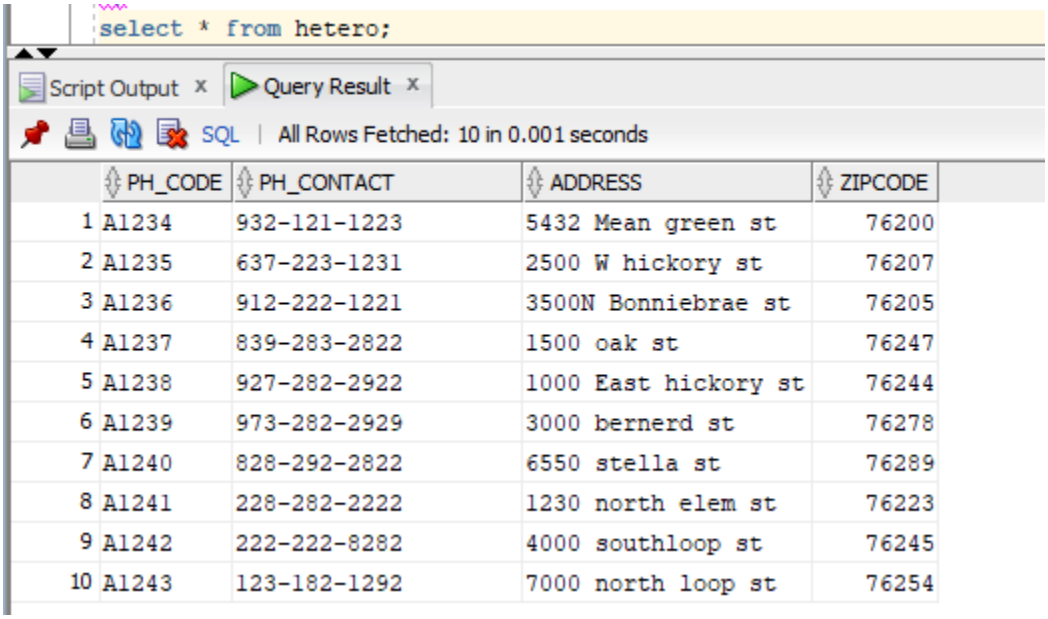
```
select * from inventory;
```

Script Output | Query Result | SQL | All Rows Fetched: 10 in 0.002 seconds

	ITEM_ID	ITEM_NAME	AVAILABILITY	REQUIRED_QUANTITY	PRESENTLY_AVAILABLE_QUANTITY
1	1000	Dolo	available	20000	15000
2	1001	Loperamide	unavailable	15000	0
3	1002	Paracetamol	available	18000	42
4	1003	Acetaminophen	unavailable	27000	15000
5	1004	Ketoconazole	available	10000	90
6	1005	Diltiazem	unavailable	33000	15000
7	1006	cetirizine	available	73000	58
8	1007	Brinzolamide	unavailable	29000	15000
9	1008	Naproxen	available	23000	15000
10	1009	Tylenol	available	22000	15000

Delete Statements:

1. Deleting the pharmacy branch/ pharmacy row which has the available quantity as zero. Initial hetero table is below.

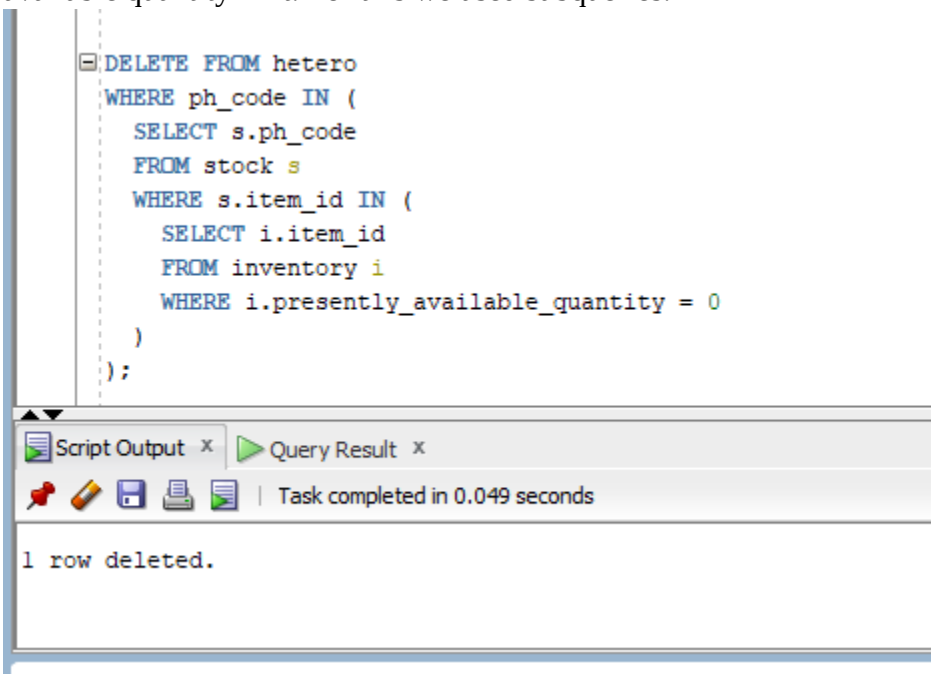


```
select * from hetero;
```

Script Output | Query Result | SQL | All Rows Fetched: 10 in 0.001 seconds

	PH_CODE	PH_CONTACT	ADDRESS	ZIPCODE
1	A1234	932-121-1223	5432 Mean green st	76200
2	A1235	637-223-1231	2500 W hickory st	76207
3	A1236	912-222-1221	3500N Bonniebrae st	76205
4	A1237	839-283-2822	1500 oak st	76247
5	A1238	927-282-2922	1000 East hickory st	76244
6	A1239	973-282-2929	3000 bernerd st	76278
7	A1240	828-292-2822	6550 stella st	76289
8	A1241	228-282-2222	1230 north elem st	76223
9	A1242	222-222-8282	4000 southloop st	76245
10	A1243	123-182-1292	7000 north loop st	76254

The below query is used to delete the hetero pharmacy row which has no medicines or available quantity In it. For this we used subqueries.



```
DELETE FROM hetero
WHERE ph_code IN (
  SELECT s.ph_code
  FROM stock s
  WHERE s.item_id IN (
    SELECT i.item_id
    FROM inventory i
    WHERE i.presently_available_quantity = 0
  )
);
```

Script Output | Query Result | Task completed in 0.049 seconds

```
1 row deleted.
```

In the above query, there are 2 subqueries where, inner query is used to find the item_id from inventory table whose value in available quantity is zero and outer query is used to find the pharmacy codes from hetero table and delete those rows.
Then Hetero pharmacy which has Ph_code = 'A1237' Is deleted.

```
select * from hetero;
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 9 in 0.001 seconds

	PH_CODE	PH_CONTACT	ADDRESS	ZIPCODE
1	A1234	932-121-1223	5432 Mean green st	76200
2	A1235	637-223-1231	2500 W hickory st	76207
3	A1236	912-222-1221	3500N Bonniebrae st	76205
4	A1238	927-282-2922	1000 East hickory st	76244
5	A1239	973-282-2929	3000 bernerd st	76278
6	A1240	828-292-2822	6550 stella st	76289
7	A1241	228-282-2222	1230 north elem st	76223
8	A1242	222-222-8282	4000 southloop st	76245
9	A1243	123-182-1292	7000 north loop st	76254

2. Deleting the female employees whose salary is less than 5000;

```
--2)
select * from employee_data;
--DELETE FROM employee_data
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.005 seconds

	EMP_ID	EMP_NAME	DESIGNATION	EMP_CONTACT	SALARY	GENDER	PH_CODE
1	201	Arun	Manager	9222922222	6600	male	A1234
2	202	Tarun	Assistant Manager	234-234-6464	5000	male	A1235
3	203	Varun	Receiptionist	123-643-1234	2500	male	A1236
4	204	Varsini	Stock Manager	342-282-9382	4000	female	A1237
5	205	Fahien	Sales Manager	975-222-2348	4500	male	A1238
6	206	Vignesh	Sales_Man-1	123-346-3411	2000	male	A1239
7	207	Charitha	Sales_Man-2	321-067-3933	2000	female	A1240
8	208	Vinisha	Sales_Man-3	985-363-2933	2000	female	A1241
9	209	Varsha	Sales_Man-4	322-229-2223	2000	female	A1242
10	210	Auish	Sales_Man-5	763-272-2721	2000	TS	A1243

```
DELETE FROM employee_data
WHERE salary < 5000 AND gender = 'female';
```

Script Output | Query Result | Task completed in 0.039 seconds

```
4 rows deleted.
```

Deleting the employee_data data whose salary is greater than 5000 and gender is female.

```
--2)
--select * from employee_data;
--DELETE FROM employee_data
--WHERE salary < 5000 AND gender = 'female';
select * from employee_data;
```

Script Output | Query Result | SQL | All Rows Fetched: 6 in 0.002 seconds

	EMP_ID	EMP_NAME	DESIGNATION	EMP_CONTACT	SALARY	GENDER	PH_CODE
1	201	Arun	Manager	9222922222	6600	male	A1234
2	202	Tarun	Assistant Manager	234-234-6464	5000	male	A1235
3	203	Varun	Receiptionist	123-643-1234	2500	male	A1236
4	205	Fahien	Sales Manager	975-222-2348	4500	male	A1238
5	206	Vignesh	Sales_Man-1	123-346-3411	2000	male	A1239
6	210	Auish	Sales_Man-5	763-272-2721	2000	TS	A1243

3. Deleting the data in payroll who's salary is repeated more than once in given duration;

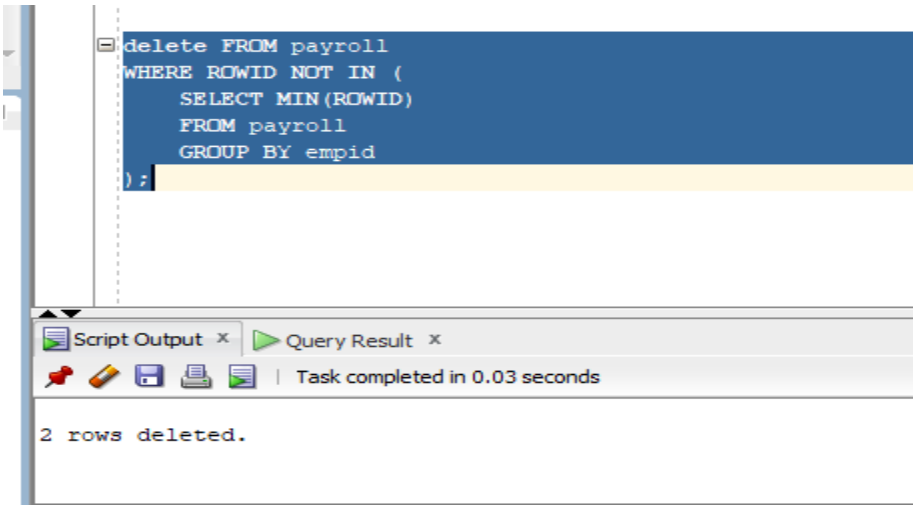
```
select * FROM payroll
WHERE empid IN (
    SELECT empid
    FROM payroll
    GROUP BY empid
    HAVING COUNT(*) > 1
);
```

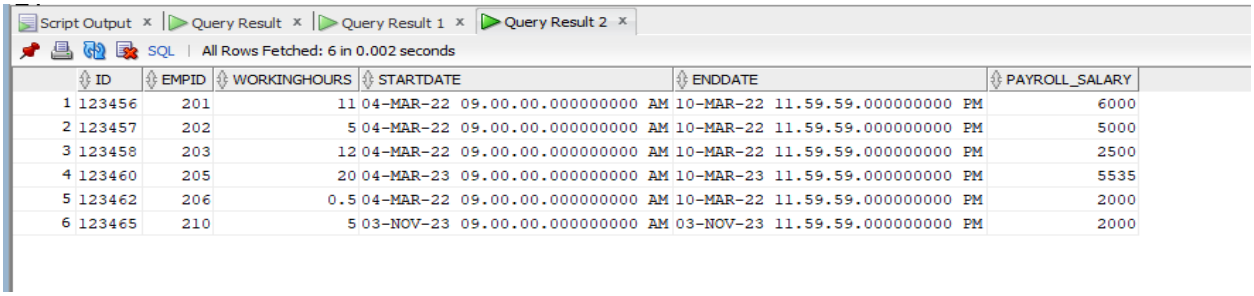
Script Output | Query Result | SQL | All Rows Fetched: 4 in 0.002 seconds

	ID	EMPID	WORKINGHOURS	STARTDATE	ENDDATE	PAYROLL_SALARY
1	123456	201	11	04-MAR-22 09.00.00.000000000 AM	10-MAR-22 11.59.59.000000000 PM	6000
2	123457	202	5	04-MAR-22 09.00.00.000000000 AM	10-MAR-22 11.59.59.000000000 PM	5000
3	123988	201	11	04-MAR-22 09.00.00.000000000 AM	10-MAR-22 11.59.59.000000000 PM	6000
4	123221	202	11	04-MAR-22 09.00.00.000000000 AM	10-MAR-22 11.59.59.000000000 PM	6000

The inner query finds the empids in payroll table grouping together with empid and where the count of empid is greater than 1. Outer query find the details in payroll table from the resulted inner query data matching with resulted empids.

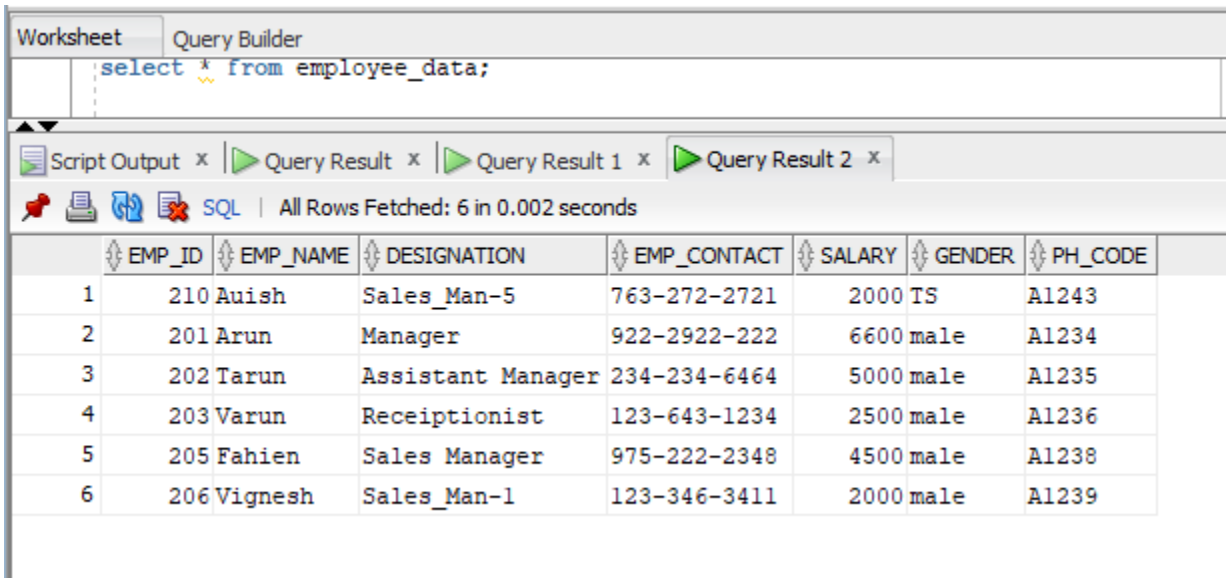
Empid 201 and 202 has 2 rows in payroll which means the employees are paid twice, Hence deleting single row of each employee of such kind.





	ID	EMPID	WORKINGHOURS	STARTDATE	ENDDATE	PAYROLL_SALARY
1	123456	201	11	04-MAR-22 09.00.00.000000000 AM	10-MAR-22 11.59.59.000000000 PM	6000
2	123457	202	5	04-MAR-22 09.00.00.000000000 AM	10-MAR-22 11.59.59.000000000 PM	5000
3	123458	203	12	04-MAR-22 09.00.00.000000000 AM	10-MAR-22 11.59.59.000000000 PM	2500
4	123460	205	20	04-MAR-23 09.00.00.000000000 AM	10-MAR-23 11.59.59.000000000 PM	5535
5	123462	206	0.5	04-MAR-22 09.00.00.000000000 AM	10-MAR-22 11.59.59.000000000 PM	2000
6	123465	210	5	03-NOV-23 09.00.00.000000000 AM	03-NOV-23 11.59.59.000000000 PM	2000

4. Deleting an employee who worked before and endDate is more than 1 year from now.



	EMP_ID	EMP_NAME	DESIGNATION	EMP_CONTACT	SALARY	GENDER	PH_CODE
1	210	Auish	Sales_Man-5	763-272-2721	2000	TS	A1243
2	201	Arun	Manager	922-2922-222	6600	male	A1234
3	202	Tarun	Assistant Manager	234-234-6464	5000	male	A1235
4	203	Varun	Receiptionist	123-643-1234	2500	male	A1236
5	205	Fahien	Sales Manager	975-222-2348	4500	male	A1238
6	206	Vignesh	Sales_Man-1	123-346-3411	2000	male	A1239

```
SELECT * FROM employee_data e WHERE e.emp_id NOT IN (SELECT EMPID FROM (
    SELECT P.EMPID, MAX(P.ENDDATE)
     FROM payroll p
     WHERE enddate BETWEEN add_months(trunc(current_date), -24) and current_date GROUP BY P.EMPID
));
```

Query Result | Script Output | Query Result 1

SQL | All Rows Fetched: 1 in 0.006 seconds

	EMP_ID	EMP_NAME	DESIGNATION	EMP_CONTACT	SALARY	GENDER	PH_CODE
1	210	Auish	Sales_Man-5	763-272-2721	2000	TS	A1243

The inner query finds the empid and maximum data of end date from payroll whose end date is greater than 12 months. Trunc(current_date,-12) is used to convert the current data which is in YYYY-MM-DD format data into months grouped together to empid and outer query is used find the employee data which are not matching with resulted subquery.

The resulted data is deleted using below query.

```
DELETE FROM EMPLOYEE_DATA e WHERE e.emp_id NOT IN (SELECT EMPID FROM (
    SELECT P.EMPID, MAX(P.ENDDATE)
     FROM payroll p
     WHERE enddate BETWEEN add_months(trunc(current_date), -24) and current_date GROUP BY P.EMPID
));
```

Query Result | Script Output | Query Result 1

Task completed in 0.028 seconds

```
1 row deleted.
```

Auish worked in pharmacy and exited from pharmacy 1 year ago, Hence the data of Auish is deleted from employee_data table.

```
select * from employee_data;
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 5 in 0.001 seconds

	EMP_ID	EMP_NAME	DESIGNATION	EMP_CONTACT	SALARY	GENDER	PH_CODE
1	201	Arun	Manager	922-2922-222	6600	male	A1234
2	202	Tarun	Assistant Manager	234-234-6464	5000	male	A1235
3	203	Varun	Receiptionist	123-643-1234	2500	male	A1236
4	205	Fahien	Sales Manager	975-222-2348	4500	male	A1238
5	206	Vignesh	Sales_Man-1	123-346-3411	2000	male	A1239

5. Delete doctor who did not treat patients in last 1 year.

project part 2 sample 2 .sql | Welcome Page | FDB | FDB ~2 | PRESCRIPTION

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Index

Sort.. | Filter:

	PRESCRIPTION_ID	P_DATE	PH_CODE	DOCTOR_ID
1	301	03-JUN-23	A1234	700
2	302	03-JUN-23	A1235	701
3	303	04-MAR-23	A1236	702
4	305	06-JUL-23	A1238	704
5	306	07-AUG-23	A1239	705
6	307	02-JUN-21	A1240	706
7	308	13-OCT-23	A1241	707

Inner query finds the doctored, maximum of prescription issued data from prescriptions grouping together with doctor_id and whose maximum prescription issued date is less than 12months, outer query finds the doctor details from the resulted inner query.

```
--6) delete doctor who did not treet patients in last 1 year
select * from doctor where doctor_id in (select doctor_id from(
    select doctor_id, max(p_date) as pt_date from prescription group by doctor_id
    ) where pt_date < add_months(trunc(current_date), -12));
```

Query Result | Script Output | Query Result 1

SQL | All Rows Fetched: 1 in 0.001 seconds

	DOCTOR_NAME	SPECIALIZATION	DOCTOR_ID	CONTACT
1	Sophia (Surgeon)	cardiologist	706	9787666666

Doctor name Sophia didn't treat any patient for the last 1 year. Hence deleting the data of Sophia.

```
delete from doctor where doctor_id in (select doctor_id from(
    select doctor_id, max(p_date) as pt_date from prescription group by doctor_id
    ) where pt_date < add_months(trunc(current_date), -12));
```

Query Result | Script Output | Query Result 1

Task completed in 0.051 seconds

```
1 row deleted.
```

After deleting the doctor details who didn't treat patients for more than 1 year, could see that doctor_id=706 is deleted.

project part 2 sample 2 .sql | Welcome Page | FDB | FDB ~2 | PRESCRIPTION

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	PRESCRIPTION_ID	P_DATE	PH_CODE	DOCTOR_ID
1	301	03-JUN-23	A1234	700
2	302	03-JUN-23	A1235	701
3	303	04-MAR-23	A1236	702
4	305	06-JUL-23	A1238	704
5	306	07-AUG-23	A1239	705
6	308	13-OCT-23	A1241	707
7	309	23-NOV-22	A1242	708
8	310	30-DEC-23	A1243	709

6. Deleting the patient who's bill amount is minimum.

project part 2 sample 2 .sql | Welcome Page | FDB | FDB ~2 | PATIENT

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	PATIENT_NAME	PATIENT_ID	DISEASE	DATE_OF_BIRTH	CONTACT	GENDER
1	John	2012	heart disease	01-JAN-98	9329222222	male
2	Taylor	2013	skin cancer	02-MAR-87	8327372722	female
3	pearson	2014	chronic respiratory disease	04-MAY-96	962123452	male
4	olivea	2015	alzheimers disease	02-AUG-97	9372827282	female
5	Jordan	2016	chronic kidney disease	05-SEP-99	9332129432	male
6	bill hunt	2017	liver disease	07-MAR-97	937283383	male
7	Jesika	2019	cardiovascular disease	14-FEB-99	9272293829	female
8	watson	2020	cerebrovasculsar disease	17-JUN-98	9383928392	male
9	robinson	2021	scleroderma	15-SEP-99	938293222	male
10	williamson	2022	Flu	12-JUL-97	9493939393	male

```
SELECT p.patient_name, p.patient_id, i.insurance_company, MIN(b.bill_amount) AS min_bill_amount
FROM patient p
INNER JOIN insurance_claim ic ON p.patient_id = ic.patient_id
INNER JOIN insurance i ON ic.insurance_id = i.insurance_id
INNER JOIN bills b ON ic.insurance_id = b.insurance_id
GROUP BY p.patient_name, p.patient_id, i.insurance_company
HAVING MIN(b.bill_amount) = (SELECT MIN(bill_amount) FROM bills)
ORDER BY min_bill_amount;
```

Query Result | Script Output | Query Result 1

SQL | All Rows Fetched: 1 in 0.009 seconds

	PATIENT_NAME	PATIENT_ID	INSURANCE_COMPANY	MIN_BILL_AMOUNT
1	olivea	2015	united health	100

In this query, We have used inner joins, insurance claim on patient matching patient ids, insurance with insuranceids and bills with insurance ids, grouping together with respect to patient, name, patientid, insurance company which are having min of bill amount = minimum of bill amount from bills and sorted by ascending order with respect to min_bill_amount.

Patient Name Olivea has the minimum amount (100). Hence deleting the data of olivea.

```
delete from patient pt where pt.patient_id in (
select patient_id from (SELECT p.patient_name, p.patient_id, i.insurance_company, MIN(b.bill_amount) AS min_bill_amount
FROM patient p
INNER JOIN insurance_claim ic ON p.patient_id = ic.patient_id
INNER JOIN insurance i ON ic.insurance_id = i.insurance_id
INNER JOIN bills b ON ic.insurance_id = b.insurance_id
```

Query Result | Script Output | Query Result 1

Task completed in 0.022 seconds

```
1 row deleted.
```

	PATIENT_NAME	PATIENT_ID	DISEASE	DATE_OF_BIRTH	CONTACT	GENDER
1	John	2012	heart disease	01-JAN-98	9329222222	male
2	Taylor	2013	skin cancer	02-MAR-87	8327372722	female
3	pearson	2014	chronic respiratory disease	04-MAY-96	962123452	male
4	Jordan	2016	chronic kidney disease	05-SEP-99	9332129432	male
5	bill hunt	2017	liver disease	07-MAR-97	937283383	male
6	Jesika	2019	cardiovascular disease	14-FEB-99	9272293829	female
7	watson	2020	cerebrovasculsar disease	17-JUN-98	9383928392	male
8	robinson	2021	scleroderma	15-SEP-99	938293222	male
9	williamson	2022	Flu	12-JUL-97	9493939393	male

we can see the data of olivea is deleted.

Individual Contributions:

- I have added few assumptions to part 3 of project.
- I have altered the prescription table and added doctor_id column to it and updates the values into the prescription table.
- I have created payroll table and inserted values into it.
- I have the tables and its data in order to solve the given questions by professor.
- I have solved 7 Questions given by professor cross verified the SQL queries.
- I have distributed the work to teammates, monitored and guided them with their works.
- I have cross verified my teammates works and their queries.
- As a team, We have collaborated with each other to complete this project Part 3.