PES University, Bangalore

B.TECH. IV SEMESTER UE18MA252-

Database Management Systems

PROJECT REPORT ON

PHARMACY MANAGEMENT SYSTEM

Submitted by ISHA ARORA

SRN: PES22018000130

Under the Guidance of Prof. Rashma BM

SUMMARY

Pharmacy management system is a user-friendly application for Pharmacist which reduces the burden and helps to manage all sections of Pharmacy like Medicine management and Billing etc., which improve the processing efficiency. It deals with the automating tasks of maintaining of Bills. In Pharmacy, Billing management is the key process. In addition, Pharmacy management system will be able to process drug prescription with ease. This project constitutes entities named pharmacist, invoice, invoice_details, admin, paymenttypes, receipts, stocks and prescription_details. In total the no. of transactions applied are six *viz.* Invoice_detail has customer to customer_id in invoice, pharmacist has first_name to served_by in invoice, receipt has payType to id in paymenttypes and customer_id to customer in invoice_details, prescription_detail has drug to stock_id in stock and id to customer_id in invoice. After analysing the Relational schema, it was observed that all the entities in this project are strong entities except prescription_details as it depends on the id of the customer relation to exist and for every entity a corresponding relation is created. None of the three normal forms are violated while doing this project. This, project included the use of triggers on different entities along with the aggregate, correlated-nested and join queries as well. Therefore, our system should provide quick access to the records maintained in the pharmacy

INTRODUCTION

The main aim of the project is the management of the database of the pharmaceutical shop. This project gives insight into the design and implementation of a Pharmacy Management System. The primary aim of pharmacy management system is to improve accuracy and enhance safety and efficiency in the pharmaceutical store. Pharmacy management system is useful to maintain correct database by providing an option to update the drugs in stock. It is the user friendly application for Pharmacist which reduces the burden and helps to manage all sections of Pharmacy like Medicine management and Billing, which improve the processing efficiency.

The **MINI WORLD** of this project constitutes entities named pharmacist, invoice, invoice_details, admin, paymenttypes, receipts, stocks, prescription_details.

1. **Pharmacist:** This entity consists of all the details of the cashiers working in the store. The attributes of the cashier entity are Pharmacist_id, first_name, last_name, postal_address, email and phone number.
2. **Invoice:** The invoice entity shows us the customer information in customer_id and customer_name attributes along with the name of the cashier who served a particular customer and the status and date of purchase. The attributes of this entity are customer_id, customer_name, served_by, status and date
3. **Invoice_details:** This entity shows a detail account of the drug purchased, quantity of it, cost and the customer who bought that particular drug along with a unique invoice_no. The attributes of the invoice_details are invoice_no, customer, quantity, cost and drug.
4. **Paymenttypes:** This table shows us the different methods of transaction that a customer can opt while making a payment. There is an id assigned to each payment method like cash, credit card etc, therefore the attributes are id and name of the payment method.
5. **Prescription_details:** This entity tells us the company's name, quantity, strength of the drug and dose prescribed to a particular customer. The attributes of this table are id, drug, quantity, dose and strength.

6. **Receipts:** The receipt entity tells us all the details about the customer who bought a particular medicine, its quantity, total cost, the payment type he opted to make the payment and the cashier who served the customer. The attributes are receiptNo, customer_id, total, payType and served_by.
7. **Stocks:** This table contains all the relevant information about the name of the drug an identifier drug id corresponding to each drug, quantity in stock, the company that manufactured the particular drug, date of expiry and the status of availability of each medicine in stock.

TRANSACTIONS: Database transactions represent real-world events of any management system. In our database, the tables are linked or related to each other in a certain way.

In the pharmacist table, the first_name attribute is related to the served_by attribute in the invoice table which gives us the name and status of all customers that a particular cashier served.

In the invoice_detail entity the customer attribute is related to the customer_id attribute of the invoice table, which in turn shows the name of the customer against a particular invoice number and the status of his invoice.

The drug attribute in this entity is related to the stock_id attribute of the stocks table that contains all the information about a specific drug such as name, quantity and status of the medicine in the stock.

The prescription_detail entity tells us the drug's name, quantity, strength of the drug and dose prescribed to a particular customer. In this the drug attribute is related to the stock_id attribute of the stock table.

The id column is related to the customer_id column of the invoice table

The receipts entity has the attribute customer_id that links its values to the customer_id attribute of the invoice table, that in turn tells the information about each customer.

The payType attribute in the receipts table is related to the id attribute of the paymenttypes entity. This gives an insight about the mode of payment opted by a particular customer as the id in paymenttypes table has a specific mode of payment assigned to it.

DATA MODEL

A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities.

ER DIAGRAM AND SCHEMA:

An entity–relationship model describes interrelated things of interest in a specific domain of knowledge. The term "**schema**" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases)

The following pictures shows all the tables or entities that are present in our pharmacy database. There are in total 8 tables named admin, invoice, invoice_details, paymenttypes, pharmacist, prescription_details, receipts and stocks.

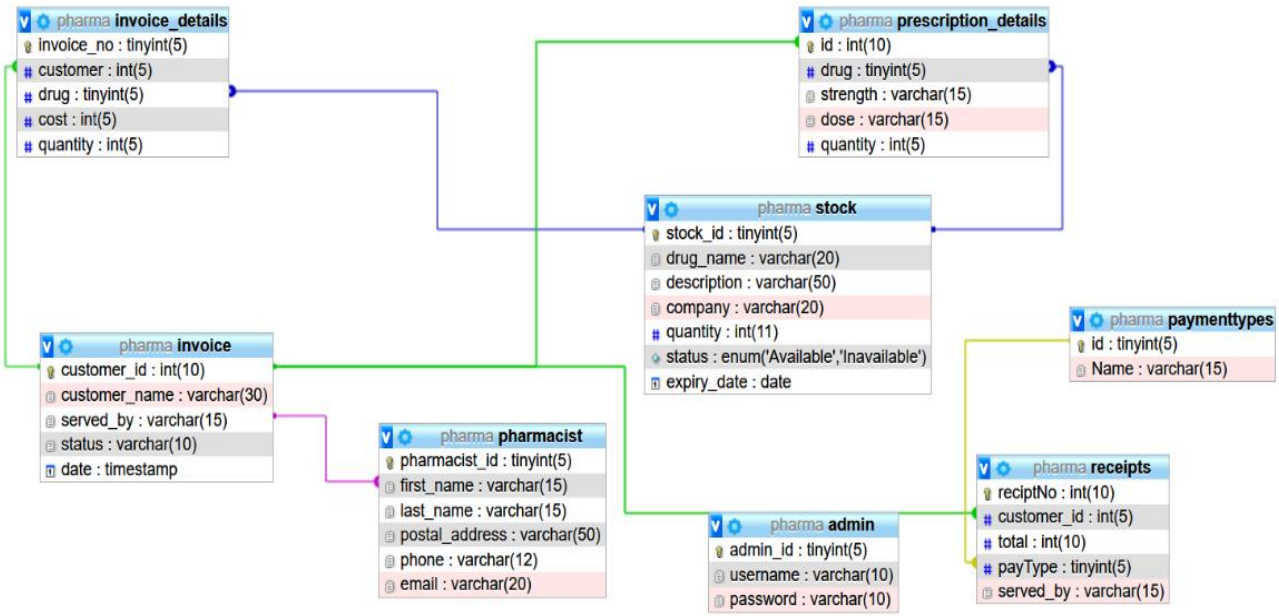
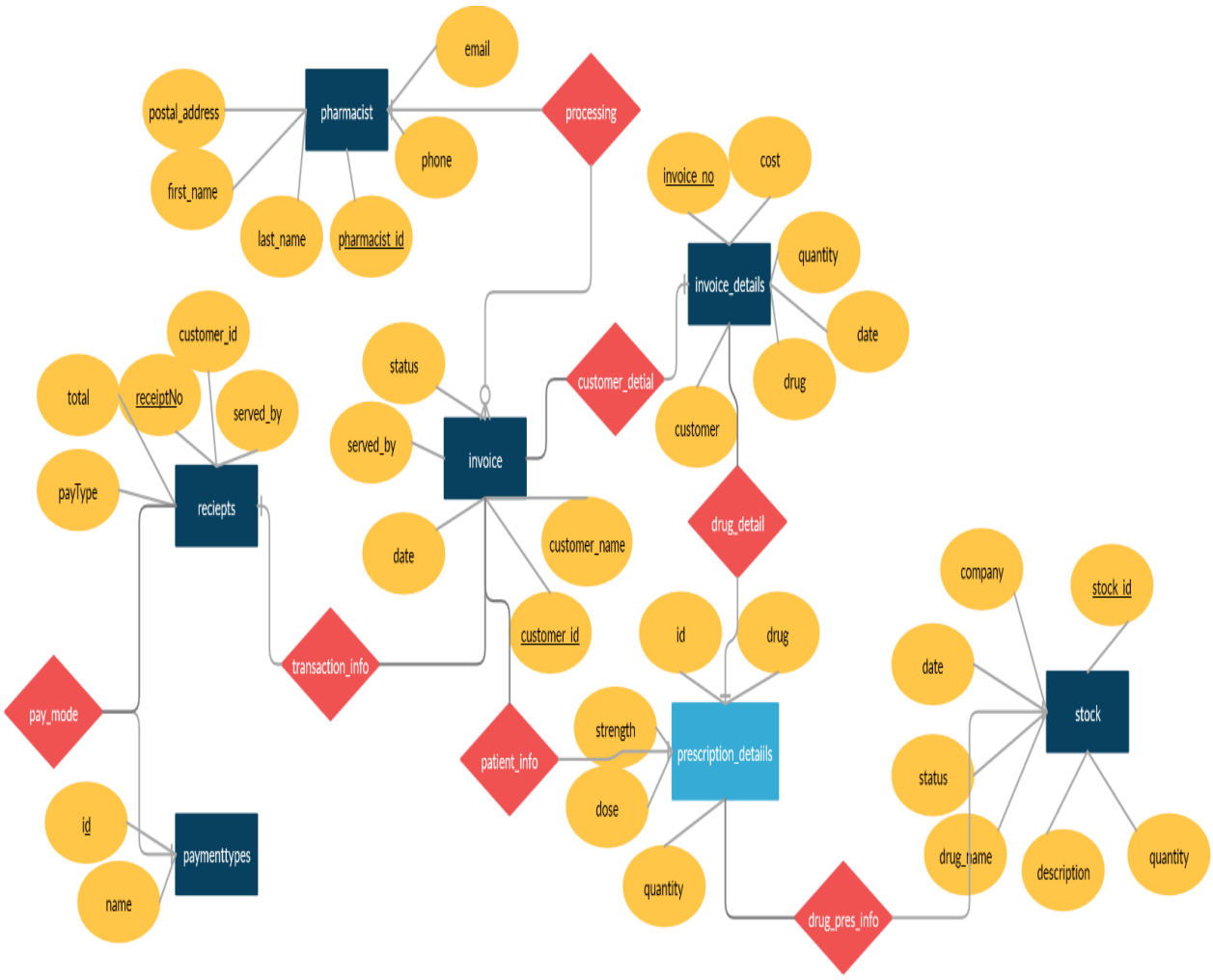


	Table	Action	Rows	Type	Collation	Size	Overhead
☐	admin	Browse Structure Search Insert Empty Drop	1	InnoDB	latin1_swedish_ci	16.0 KiB	-
☐	invoice	Browse Structure Search Insert Empty Drop	10	InnoDB	latin1_swedish_ci	16.0 KiB	-
☐	invoice_details	Browse Structure Search Insert Empty Drop	10	InnoDB	latin1_swedish_ci	48.0 KiB	-
☐	paymenttypes	Browse Structure Search Insert Empty Drop	5	InnoDB	latin1_swedish_ci	16.0 KiB	-
☐	pharmacist	Browse Structure Search Insert Empty Drop	2	InnoDB	latin1_swedish_ci	16.0 KiB	-
☐	prescription_details	Browse Structure Search Insert Empty Drop	10	InnoDB	latin1_swedish_ci	32.0 KiB	-
☐	receipts	Browse Structure Search Insert Empty Drop	10	InnoDB	latin1_swedish_ci	16.0 KiB	-
☐	stock	Browse Structure Search Insert Empty Drop	10	InnoDB	latin1_swedish_ci	16.0 KiB	-
	8 tables	Sum	58	MyISAM	utf8mb4_0900_ai_ci	176.0 KiB	0 B



After analysing the above figure I have come up with the following conclusions for the Relational schema :

- ❖ All the entities in this project are strong entities except prescription_details as it depends on the id of the customer relation to exist.
- ❖ Each relation is mapped using an appropriate foreign key keeping in mind the cardinality and participation constraints
- ❖ The weak entity prescription_details is also mapped to a relation with an appropriate foreign key.
- ❖ For every entity a corresponding relation is created.

TABLE WISE DESCRIPTION

INVOICE TABLE

+ Options

	customer_id	customer_name	served_by	status	date
Edit Copy Delete	10	Sagar	ajay	complete	2020-04-10 16:49:42
Edit Copy Delete	11	Jalas	ajay	complete	2020-04-10 16:58:59
Edit Copy Delete	12	Abdul	bhairav	complete	2020-04-10 17:49:02
Edit Copy Delete	13	Andre	ajay	complete	2020-04-11 17:55:19
Edit Copy Delete	14	William	bhairav	complete	2020-04-13 17:59:38
Edit Copy Delete	15	Sita	bhairav	complete	2020-04-13 18:09:51
Edit Copy Delete	16	Sameeksha	bhairav	complete	2020-04-15 18:19:45
Edit Copy Delete	17	Peter	ajay	complete	2020-04-15 18:21:48
Edit Copy Delete	18	mohit	ajay	complete	2020-04-17 00:50:44
Edit Copy Delete	19	Jay	ajay	complete	2020-04-18 02:04:51

PHARMACIST TABLE

	pharmacist_id	first_name	last_name	postal_address	phone	email
Edit Copy Delete	1	ajay	kumar	Bangalore	9111111101	ajay@pharmacy.com
Edit Copy Delete	2	bhairav	Sharma	Bangalore	9222222202	bhairav@pharmacy.com

INVOICE_DETAILS TABLE

	invoice_no	customer	drug	cost	quantity
Edit Copy Delete	101	10	10	200	10
Edit Copy Delete	102	11	10	250	5
Edit Copy Delete	103	18	10	50	5
Edit Copy Delete	104	15	9	100	2
Edit Copy Delete	105	10	6	120	12
Edit Copy Delete	106	16	8	50	5
Edit Copy Delete	107	12	6	120	10
Edit Copy Delete	108	13	6	120	10
Edit Copy Delete	109	17	7	250	10
Edit Copy Delete	110	19	8	150	15

PAYMENTTYPES TABLE

	id	Name
Edit Copy Delete	1	Cash
Edit Copy Delete	2	Credit card
Edit Copy Delete	3	Mobile Money
Edit Copy Delete	4	Cheque
Edit Copy Delete	5	net banking

PRESCRIPTION_DETAILS

	id	drug	strength	dose	quantity
Edit Copy Delete	10	8	15 mg	1x3	15
Edit Copy Delete	11	9	10 mg	2x3	20
Edit Copy Delete	12	6	150 mg	2x4	12
Edit Copy Delete	13	10	50 mgms	2x3	15
Edit Copy Delete	14	6	150 mg	2x3	10
Edit Copy Delete	15	7	50 mg	2x3	20
Edit Copy Delete	16	8	40 mg	2x3	16
Edit Copy Delete	17	9	15 gms	2x3	20
Edit Copy Delete	18	7	50 gms	1X3	15
Edit Copy Delete	19	10	50 gms	1x2	5

RECEIPTS

	reciptNo	customer_id	total	payType	served_by
Edit Copy Delete	1	12	120	2	bhairav
Edit Copy Delete	2	11	250	1	ajay
Edit Copy Delete	3	13	120	2	ajay
Edit Copy Delete	4	14	105	3	bhairav
Edit Copy Delete	5	15	100	1	bhairav
Edit Copy Delete	6	16	50	4	bhairav
Edit Copy Delete	7	17	250	3	ajay
Edit Copy Delete	8	18	50	2	ajay
Edit Copy Delete	9	19	150	1	ajay
Edit Copy Delete	10	10	150	5	ajay

STOCKS

+ Options

	stock_id	drug_name	Company	quantity	Cost	expiry_date	status
Edit Copy Delete	1	pentop	Ranbaxy	500	2000	2021-03-27	Available
Edit Copy Delete	2	crocin	clinix	1000	5000	2021-04-27	Available
Edit Copy Delete	3	Albandazole	Pfizer	100	700	2021-04-23	Available
Edit Copy Delete	4	levocetrizine	Ranbaxy	300	1200	2021-04-15	Available
Edit Copy Delete	5	Bactrium-DS	clinix	150	900	2021-04-27	Available
Edit Copy Delete	6	Dual Cotexin	Cipla	150	4500	2021-01-05	Available
Edit Copy Delete	7	Naprosyn	Mankind	250	750	2021-03-16	Available
Edit Copy Delete	8	Flagi	Ranbaxy	65	400	2021-03-31	Available
Edit Copy Delete	9	Actal	Cipla	1000	3000	2021-02-17	Available
Edit Copy Delete	10	Piriton	Sunpharma	1000	2500	2021-02-14	Available
Edit Copy Delete	11	dolo	ranbaxy	100	1000	0000-00-00	Inavailable

FUNCTIONAL DEPENDENCIES AND NORMALISATION

1. FUNCTIONAL DEPENEDENCIES:

RELATION	FUNCTIONAL DEPENDENCIES	PRIMARY KEY	CANDIDATE KEY
Invoice	customer_id→ customer_name	Customer_id	Customer_name
Invoice_details	invoice_no→ customer,drug, cost, quantity	Invoice_ no	Invoice_no
Paymenttypes	Id→ name	id	id
Pharmacist	pharmacist_id→ first_name, last_name, postal_address, email, phone	Pharmacist_id	Pharmacist_id, phone, email
admin	Admin_id→admin, password	Admin_id	Admin_id, admin
Prescription_ details	id→ drug, strength, dose, quantity	id	id
Receipts	receiptNo→ customer_id, total, payType, served_by	ReceiptNo	receiptNo
Stocks	stock_id→ drug_name, description, company, quantity, status, expiry_date	Stock_id	Stock_id, drug_name

2. NORMALISATION

First Normal Form:

ER-to-Relational Mapping Algorithm ensures that all the multivalued attributes will be converted into a new set of independent tables, thus the domain of every attribute in every relation contains only atomic values. Every relation (table) has a unique set of columns with all the values in that particular column belonging to the same domain. Thus I can infer that all my relations are in First Normal Form (1NF)

Second Normal Form:

There are no partial dependencies in any of the relations. Every relation has functional dependencies from prime attribute to nonprime attribute only. Thus I can infer that all my relations are in Second Normal Form (2NF).

Third Normal Form:

There is no functional or transitional dependency from a non-prime attribute to a prime attribute in any of the relations, hence there are no transitive dependencies. Thus I can infer that all my relations are in Third Normal Form (3NF).

Discussion of violation of the normal-forms :

Consider the table stocks, in this there is stock_id and drug_name, now assuming if there is a column named description of the medicine as well. Suppose one of the values in the table was Crocin corresponding to fever, now if I would've changed this value to Levocetrizine which is an anti-allergic, the corresponding data in the description column would've been false and therefore 3NF would've failed. As this is a case of transitional dependency and hence violates 3NF.

LOSELESS JOIN:

The following conditions must be satisfied to obtain a lossless join:

- Att(R1) ∪ Att (R2) =Att(R)
- Att(R1) ∩ Att(R2) ≠ Φ
- Att(R1) ∩ Att(R2) = Att(R1) or Att(R1) ∩ Att(R2) = Att(R2)

If we take the tables, Invoice and Receipt we can find that they lose no information after performing a join and later decomposing them, therefore it satisfies lossless join

DDL

1. Table structure for table `pharmacist`

```
CREATE TABLE IF NOT EXISTS `pharmacist` (
  `pharmacist_id` tinyint(5) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(15) NOT NULL,
  `last_name` varchar(15) NOT NULL,
  `postal_address` varchar(50) NOT NULL,
  `phone` varchar(12) NOT NULL,
  `email` varchar(20) NOT NULL,
  PRIMARY KEY (`pharmacist_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;
```

Constraints for table `pharmacist`

```
ALTER TABLE `pharmacist` ADD FOREIGN KEY (`first_name`) REFERENCES
`invoice`(`served_by`) ON DELETE RESTRICT ON UPDATE RESTRICT;
```

2. Table structure for table `invoice`

```
CREATE TABLE IF NOT EXISTS `invoice` (
  `customer_id` int(10) NOT NULL,
  `customer_name` varchar(30) NOT NULL,
  `served_by` varchar(15) NOT NULL,
  `status` varchar(10) NOT NULL DEFAULT 'Unpaid',
  `date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`customer_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
ALTER TABLE `invoice` ADD INDEX(`served_by`);
```

3. Table structure for table `invoice_details`

```
CREATE TABLE IF NOT EXISTS `invoice_details` (
  `invoice_no` tinyint(5) NOT NULL AUTO_INCREMENT,
  `customer` int(5) NOT NULL,
  `drug` tinyint(5) NOT NULL,
  `cost` int(5) DEFAULT NULL,
  `quantity` int(5) NOT NULL,
  PRIMARY KEY (`invoice_no`),
  KEY `stocks` (`drug`),
  KEY `invoices` (`customer`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=31 ;
```

Constraints for table `invoice_details`

```
ALTER TABLE `invoice_details`
  ADD CONSTRAINT `invoices` FOREIGN KEY (`customer`) REFERENCES `invoice` (`customer_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `stocks` FOREIGN KEY (`drug`) REFERENCES `stock` (`stock_id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

4. Table structure for table `paymenttypes`

```
CREATE TABLE IF NOT EXISTS `paymenttypes` (
  `id` tinyint(5) NOT NULL AUTO_INCREMENT,
  `Name` varchar(15) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;
```

5. Table structure for table `admin`

```
CREATE TABLE IF NOT EXISTS `admin` (
  `admin_id` tinyint(5) NOT NULL AUTO_INCREMENT,
  `username` varchar(10) NOT NULL,
  `password` varchar(10) NOT NULL,
  PRIMARY KEY (`admin_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;
```

6. Table structure for table `prescription_details`

```
CREATE TABLE IF NOT EXISTS `prescription_details` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `drug` tinyint(5) NOT NULL,
  `strength` varchar(15) NOT NULL,
  `dose` varchar(15) NOT NULL,
  `quantity` int(5) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `dsfd` (`drug`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=31 ;
```

Constraints for table `prescription_details`

```
ALTER TABLE`prescription_details` ADD FOREIGN KEY (`id`) REFERENCES
`invoice`(`customer_id`) ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE `prescription_details` ADD FOREIGN KEY (`drug`) REFERENCES
`stock`(`stock_id`) ON DELETE RESTRICT ON UPDATE RESTRICT;
```

7. Table structure for table `receipts`

```
CREATE TABLE IF NOT EXISTS `receipts` (
  `reciptNo` int(10) NOT NULL,
  `customer_id` int(5) NOT NULL,
  `total` int(10) NOT NULL,
  `payType` tinyint(5) NOT NULL,
  `served_by` varchar(15) NOT NULL,
   PRIMARY KEY (`reciptNo`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Constraints for table `receipts`

```
ALTER TABLE`receipts` ADD FOREIGN KEY (`customer_id`) REFERENCES
`invoice`(`customer_id`) ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE`receipts` ADD FOREIGN KEY (`payType`) REFERENCES `paymenttypes`(`id`) ON
DELETE RESTRICT ON UPDATE RESTRICT;
```

8. Table structure for table `stock`

```
CREATE TABLE IF NOT EXISTS `stock` (
  `stock_id` tinyint(5) NOT NULL AUTO_INCREMENT,
  `drug_name` varchar(20) NOT NULL,
  `description` varchar(50) NOT NULL,
  `company` varchar(20) NOT NULL,
   `quantity` int(11) NOT NULL,
    `status` enum('Available','Inavailable') NOT NULL,
  `expiry_date` date NOT NULL,
  PRIMARY KEY (`stock_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=10 ;
```

TRIGGERS

In DBMS , a *trigger* is a SQL procedure that initiates an action (i.e., *fires* an action) when an event (INSERT, DELETE or UPDATE) occurs. A trigger cannot be called or executed; the DBMS automatically fires the trigger as a result of a data modification to the associated table. Triggers are used to maintain the referential integrity of data by changing the data in a systematic fashion.

1. **In table receipts we created a trigger, if the total in the receipts table is NULL, it shows a message saying "please enter total" in invoice_details table. The query written for it is:**

```
CREATE TRIGGER null_total
AFTER INSERT
ON receipts FOR EACH ROW
 BEGIN
 IF NEW.total IS NULL THEN
        INSERT INTO
invoice_details(invoice_no,customer,drug,cost,quantity,Date)
      VALUES(113,18,10,'please enter total',25,2020-06-03);
   END IF;
END
```

Details

Trigger name	null_total
Table	receipts
Time	BEFORE
Event	INSERT

Definition	BEGIN
	IF NEW.total IS NULL THEN
	INSERT INTO invoice_details(invoice_no,customer,drug,cost,quantity,Date)
	VALUES(113,18,10,'please enter total',25,2020-06-03);
	END IF;
	END

Definer	root@localhost

☐	Edit Copy Delete	12	15	NULL	2	ajay	2020-05-28 17:48:16
☐	Edit Copy Delete	112	15	8	please enter total	12	2020-05-28

2. **This trigger updates the date when a row is inserted into the table receipts and a similar trigger exists in table invoice.**

```
CREATE TRIGGER receipt_tri
AFTER INSERT
BEGIN
   SET @date=CURRENT_TIMESTAMP();
END
```

Details

Trigger name	receipt_tri
Table	receipts
Time	AFTER
Event	INSERT
Definition	BEGIN SET @date=CURRENT_TIMESTAMP(); END

Details

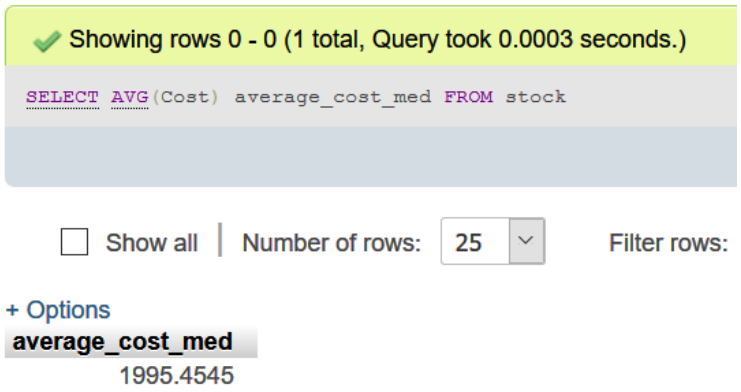
Trigger name	invoice_tri
Table	invoice
Time	AFTER
Event	INSERT
Definition	BEGIN SET @date=CURRENT_TIMESTAMP(); END

SQL QUERIES

AGGREGATE QUERIES:

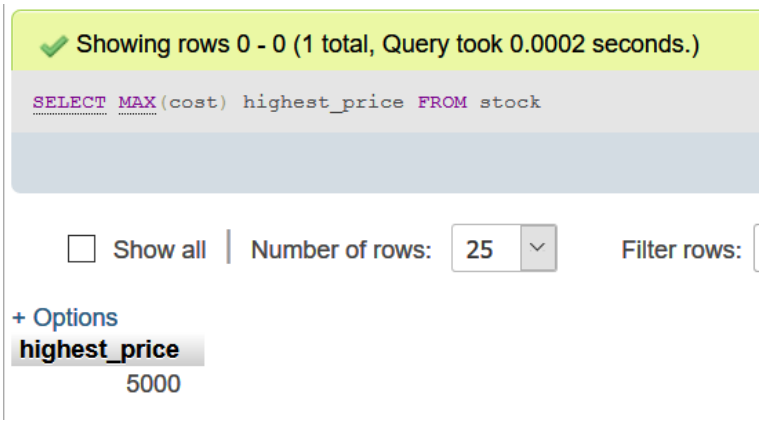
1. **We are finding average cost of the medicines that are there in our stock. It tells us the lump-sum amount we pay for a particular medicine in the stock.**

```
SELECT AVG(Cost) average_cost_med FROM stock
```



2. **Query to find out the most expensive medicine stock wise.**

SELECT MAX(cost) highest_price FROM stock

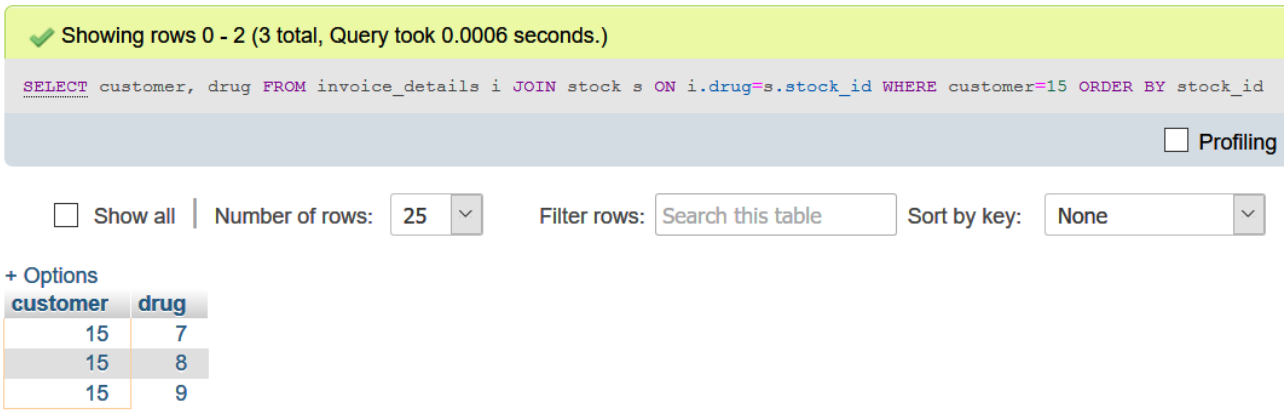


JOIN QUERIES:

The join clause joins two tables based on a condition which is known as a join predicate.

3. **The given query finds out the drug purchased by a particular customer by taking join of invoice_details table and stock table**

```
SELECT customer, drug FROM invoice_details i JOIN stock s ON
i.drug=s.stock_id WHERE customer=15 ORDER BY stock_id
```



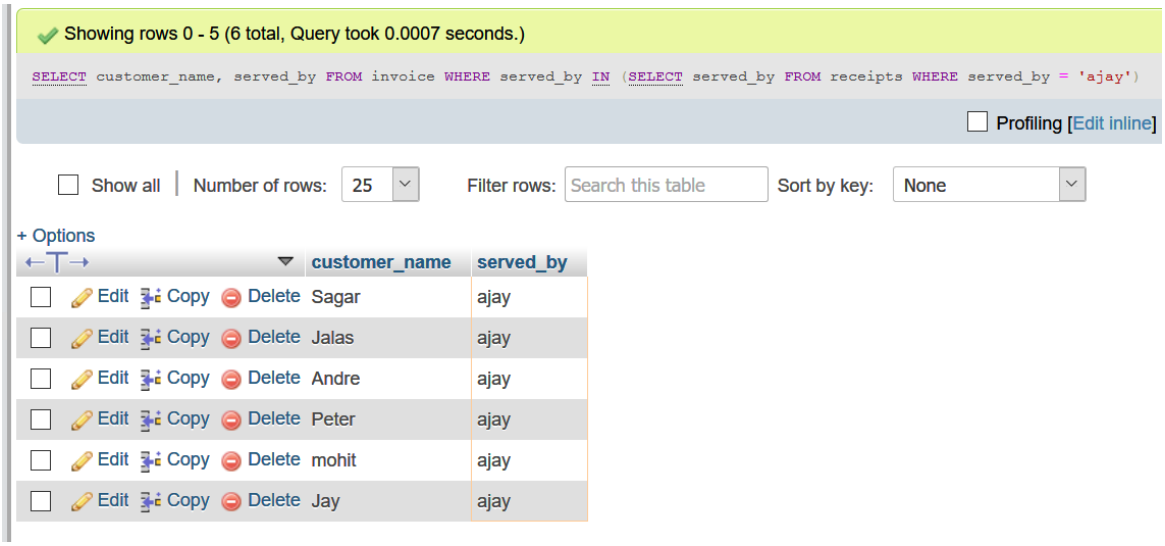
4. **This question is an example of JOIN where it takes two tables namely invoice_details and stock and shows customer id, drug id and drug name**

```
SELECT customer, drug, drug_name FROM invoice_details i JOIN stock s ON
i.drug=s.stock_id WHERE customer=15 ORDER BY stock_id
```

NESTED QUERIES:

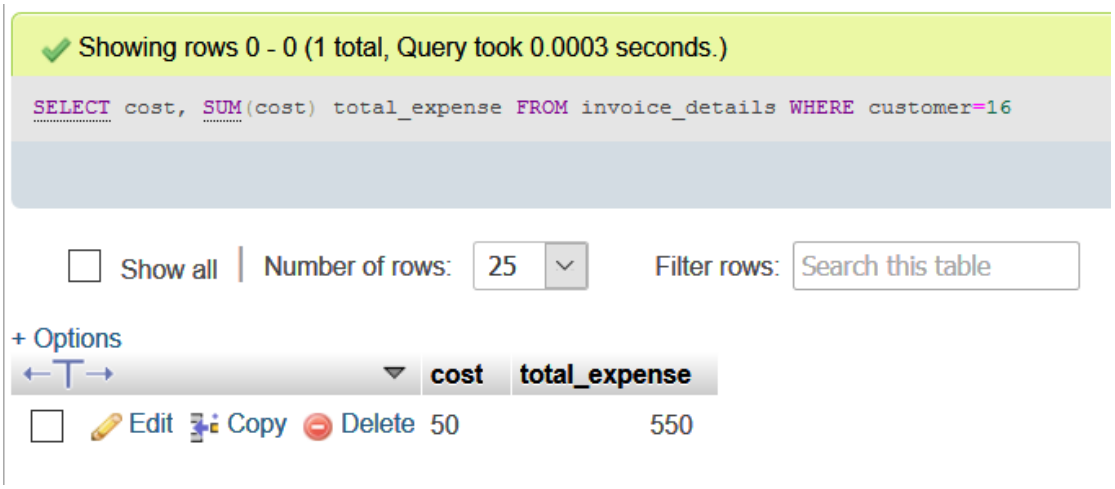
5. **The given nested query tells us all the customer served by the pharmacist "ajay"**

```
SELECT customer_name, served_by FROM invoice WHERE served_by IN (SELECT
served_by FROM receipts WHERE served_by = 'ajay')
```



6. **The given query shows the total expense of the customer with customer id 16. This query is a combination of aggregate and select statement.**

```
SELECT cost, SUM(cost) total_expense FROM invoice_details WHERE customer=16
```



CONCLUSION AND FUTURE SCOPE

Pharmacy management system is a management system that is designed to improve accuracy and to enhance safety and efficiency in the pharmaceutical store. This program can be used in any pharmaceutical shops having a database to maintain. It is a computer-based system which helps the Pharmacist to improve inventory management, cost etc. At present, manual system is being utilized in the pharmacy. It requires the pharmacist to manually monitor each drug that is available in the pharmacy. In order to overcome this problem, the automated pharmacy management system was built. The database is created using MySQL. It keeps the record of all the customers, invoices, pharmacist details, prescription details and stocked medicine along with their cost to provide a hassle-free processing, and easy customer dealing. Therefore, our system provides quick access to the records maintained in the pharmacy.

Since the local stores don’t have the resources to set up a monitor / desktop, the future scope of the project extends to creating a user-friendly mobile app for the same.