Department of CSE
Jan – May 2020

CS252
Database Management Systems

Project Report

Pharmaceutical - Retail Management System

a.k.a Leo
00157

PROJECT SUMMARY

The title of this project is Pharmaceutical-Retail Management system where pharma stores have contracts with hospitals. The retail part of the pharma store including the Dealer supplying the medicines to the stores and the patient buying medicine from the stores are focused. The schema consists of 11 relations (6 strong entities + 1 weak entity + 4 relationships). The Data Model and the Relational Schema have been discussed below. The schema is normalized until the third normal form (3NF) and the process that led to normalizing until 3NF is given in the upcoming sections.

The weak entity modeled relation (QUANT) is auto filled using triggers upon insert action into *RETAIL* and *TRANSACTIONS* . Hence two triggers are used here. DDL commands have been put in the DDL sections, which were used to create the schema of our database. Query statements using aggregate functions and queries using join and correlated - nested queries have been used to obtain certain outputs.

Thus this database system is capable of performing most of the actions related to a Pharmaceutical-Retail system, where emphasis has been laid on both the Pharma and Retail sectors. Limitations and Future work have been discussed thoroughly in the conclusion section.

➢ Table Of Contents

➢ Introduction

In general Pharmaceutical Management/Information is a system that stores data and enables functionality that organizes and maintains the medication use process within pharmacies. These systems may be an independent technology for the pharmacy's use only, or in a hospital setting, pharmacies may be integrated within an inpatient hospital system. The mini-world I have chosen is a Pharmacy-Retail Management System in a hospital setting.

<u>Table Descriptions:</u>

1. MEDICINE
   Keep information about medicine link medicine id(med_id), name, composition, manufacture and expiry date(*mfg_date, exp_date*) and cost per tablet(*cost_per_tab*). Each store has certain available tablets in stock, supplied by one or more dealers.

2. DEALER
   Holds details about the Dealer id(*dealer_id*), name, address, phone.
   - Dealers supply medicines to the stores.

3. HOSPITAL
   Holds information about the hospital id(*hos_id*), name, address and phone.
   - Patients are treated at the hospitals.
   - Hospitals have contracts with one or many stores.

4. DOCTOR
   Contains information like doctor id(*hos_id*), hospital is(*hos_id*) and doctor name(*doc_name*).
   - hos_id indicates the hospital in which the doctor works.

5. PATIENT
   Keeps information like patient id(*pat_id*), name, address and phone.
   - Patient is strictly treated at a hospital.
   - Is part of the TRANSACTION, to buy medicines in stores.

6. TREATMENT
   Indicates the attributes of a relationship mainly, the patient involved(*pat_id*), the hospital in which he was treated(*hos_id*), the doctor, in that hospital, he was treated under(*doc_id*) and the treatment date.

7. TRANSACTIONS
   It is a process that involves the patient(*pat_id*) buying medicines(*med_id*) in few/bulk quantities(*quantity*) at a store(*store_id*) generating a bill(*bill_id*) and a total on a particular date(*pur_date*). This entire process, between the PATIENT and the STORES, has been modeled into this single relation.

8. RETAIL
   It is a relationship between the dealer(*dealer_id*) and the store(*store_id*). The medicines(*med_id*) and the batch number(*batch_no*) along with quantity_supplied is noted down.

9. CONTRACT
   It is a contract between the Hospital(*hos_id*) and the Store(*store_id*).

10. STORES
    Has information about the store(*store_id*), name, address, phone and store manager(*store_man*).

11. QUANT
    It is a table automatically filled by triggers on insertions into the RETAIL and TRANSACTIONS tables. It keeps a track of all the medicines in all the stores along with its quantity. This will be useful in notifying when to replenish the medicine stocks.

➢ Data Model

The Entity-Relationship Diagram (ERD) for the Pharmaceutical-Retail Management System is shown in Fig 1.

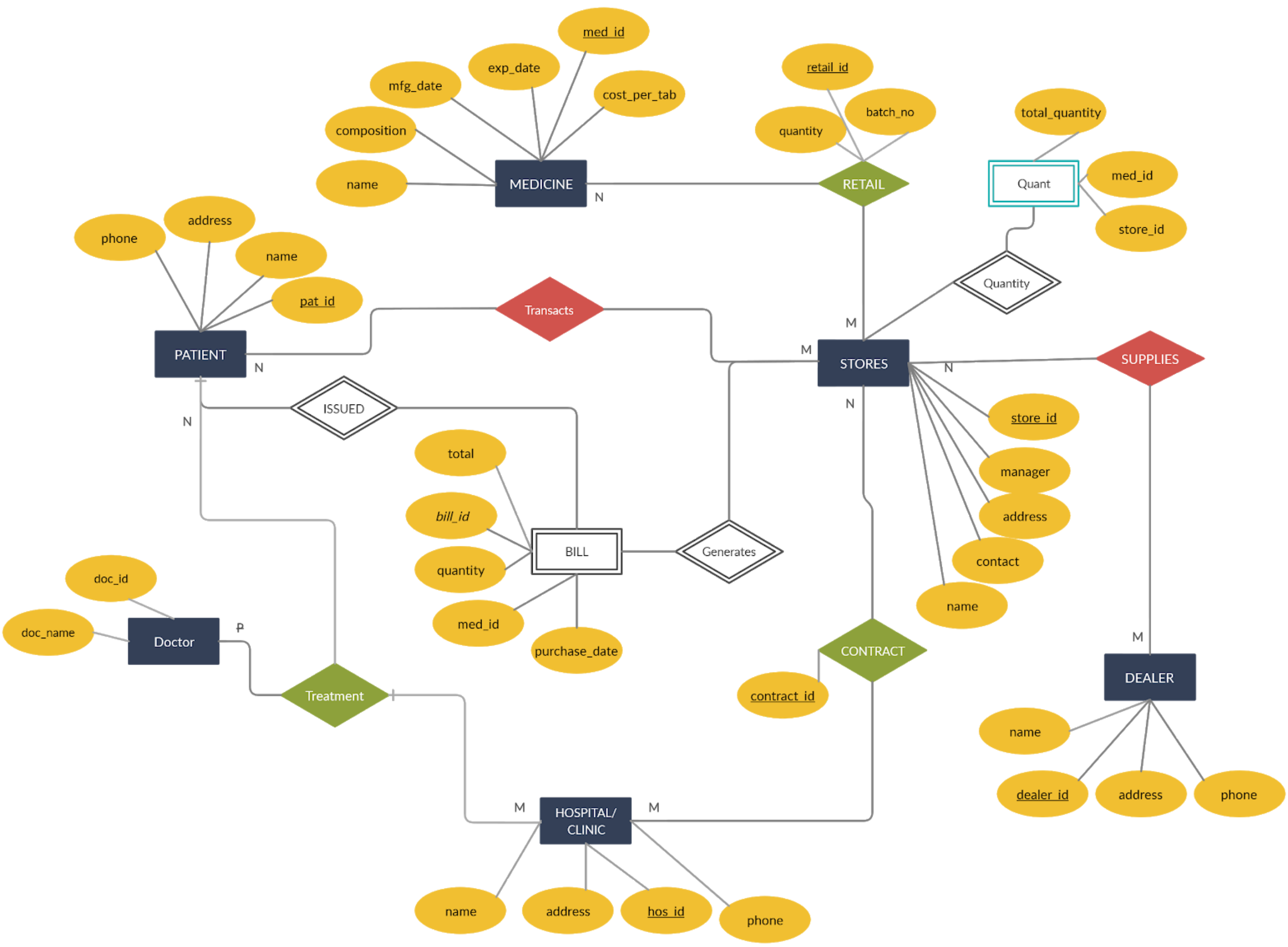

Fig 1: ER diagram of Pharmaceutical-Retail Database System

- All the strong(regular) entities, in Black, have a corresponding relation of their own - MEDICINE, STORES, DEALER, PATIENT, DOCTOR and HOSPITAL.

- The entire process of a PATIENT transacting with a STORE and the same STORE generating a bill which is ISSUED back to the PATIENT is modeled into a single relation called TRANSACTION. The ERD elaborates what is involved in a TRANSACTION. The transactions are stored such that each tuple represents only a single medicine being bought, and there will be 'n' rows signifying a patient buying 'n' medicines.

- RETAIL - It is a relationship between the dealer(dealer_id) and the store(*store_id*). The medicines(*med_id*) and the batch number(*batch_no*) along with quantity_supplied is noted down.

- CONTRACT - It is a contract between the HOSPITAL(*hos_id*) and the STORE(*store_id*). It had to be made into a separate table even though it has only two attributes, because the earlier design was not in 2 Normal Form (2NF).

- TREATMENT - here is modeled as a ternary relationship. For simplicity it can be viewed as a PATIENT consulting a DOCTOR and the DOCTOR providing treatment in a HOSPITAL.
- Each relation is mapped using an appropriate foreign key keeping in mind the cardinality and participation constraints.

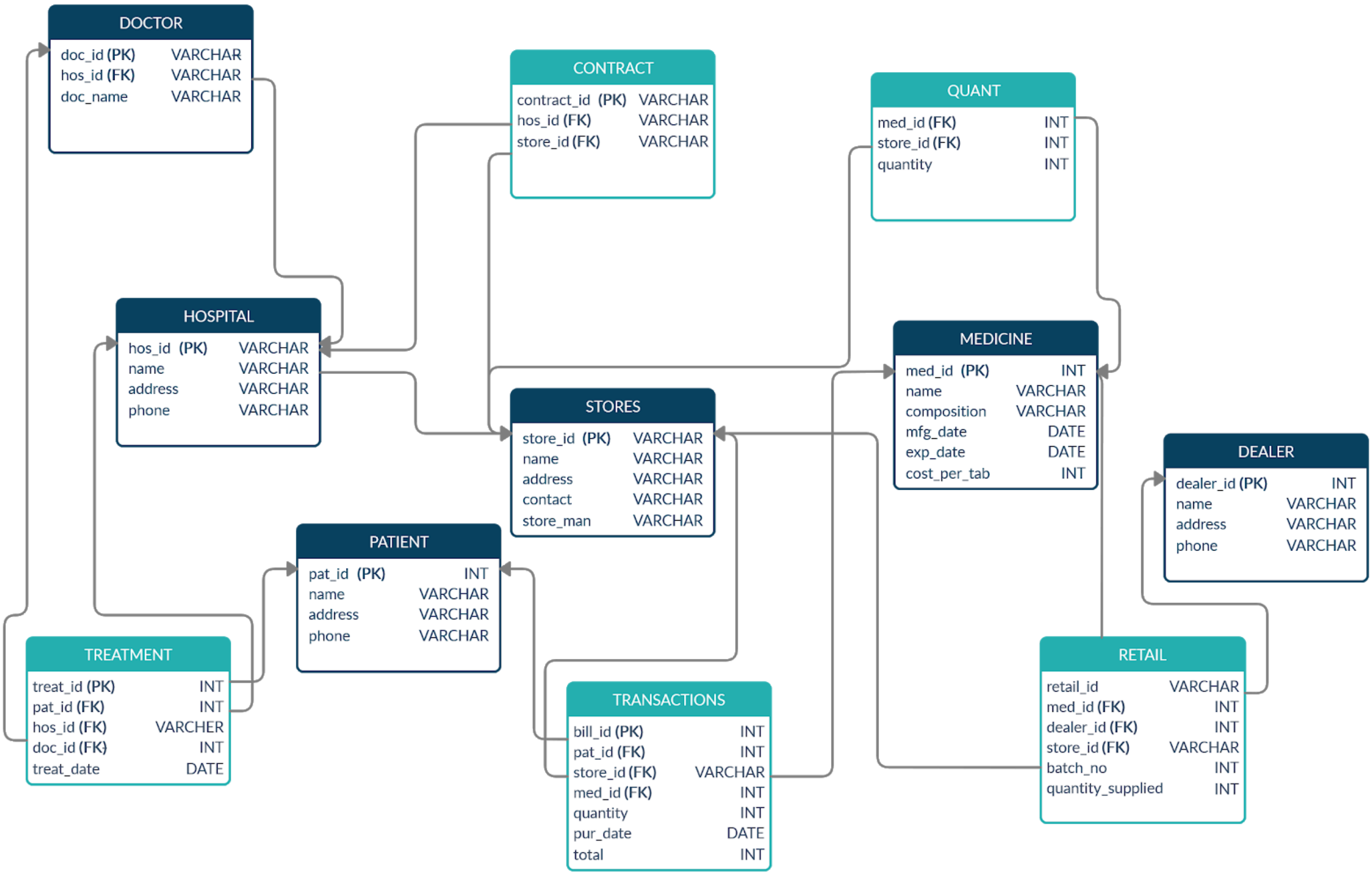

Fig 2: Relational Schema of Pharmaceutical-Retail Database System

The attributes along with its data types of all the relations can be seen in Fig 2. Some id's like store_id and hos_id are chosen to be of type VARCHAR while the other id's are INT. DATE type has been used in attributes whose values are dates of the calendar in "yyyy-mm-dd" format.

➢ Functional Dependencies and Normalisation

Relation	Functional Dependencies	Primary Key	Candidate Keys
MEDICINE	med_id → {name, composition, mfg_date, exp_date, cost_per_tab} (name, composition) → {cost_per_tab}	med_id	med_id
STORES	store_id → {name, address, contact, store_man} (name, address) →{store_id, contact, store_man}	store_id	store_id
DEALER	dealer_id → {name, address, phone} (name, address) → {dealer_id, phone}	dealer_id	dealer_id
PATIENT	pat_id → {name, address, phone} (name, address) → {pat_id, phone}	pat_id	pat_id
HOSPITAL	hos_id → {name, address, phone} (name, address) → {hos_id, phone}	hos_id	hos_id
DOCTOR	doc_id → { hos_id, doc_name} (hos_id, doc_name) → doc_id	doc_id	doc_id, hos_id
TRANSACTIONS	bill_id → {pat_id, store_id, med_id, quantity, pur_date, total} (bill_id, pat_id, store_id, med_id) → {quantity, pur_date, total}	bill_id	bill_id, pat_id, store_id, med_id
RETAIL	reatil_id → {med_id, store_id, dealer_id, batch_no, quantity_supplied}	retail_id	retail_id, med_id, store_id, dealer_id
TREATMENT	treat_id → {pat_id, hos_id, doc_id, treat_date} (treat_id, pat_id, hos_id, doc_id) → treat_date	treat_id	treat_id, hos_id, doc_id, pat_id
CONTRACT	contract_id → {hos_id, store_id}	contract_id	contract_id, hos_id, store_id
QUANT	(med_id, store_id) → {quantity}		med_id, store_id

First Normal Form (1NF):

While converting an ER diagram to a Relational Mapping it is ensured that if any multivalued attributes are present then they are given separate independent tables. Since all the relations(tables) have a unique set of columns, with all the values in a particular column being atomic and belonging to the same domain, the relations are in 1NF.

Second Normal Form (2NF):

Fig 3: HOSPITAL relation not in 2NF

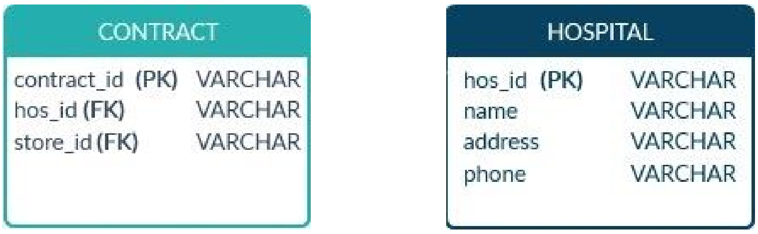

Fig 4: Solution to partial dependency in Fig 3

In the earlier design, given in Fig 3, the relation HOSPITAL had store_id as an attribute under it to indicate the tie-up/contract that the hospital(hos_id) had with the store(store_id). But here, it can be seen that following dependency holds :

- *hos_id → {store_id, name, address, phone}*
- *hos_id → {name, address, phone}*
- *(name, address) → {hos_id, phone}*

But in no way does store_id determine other attributes of HOSPITAL excluding hos_id i.e *store_id →* {name, address, phone} is false.
Thus *{name, address, phone}* depend only on a proper subset of prime attributes i.e only *hos_id*. Thus there exists a partial dependency here.

Fig 4: Shows how the partial dependency was taken care off by utilizing an extra table called CONTRACT. Now *hos_id → {name, address, phone}* and *(name, address) → {hos_id, phone}*. In CONTRACT table *contract_id → {hos_id, store_id}* dependency holds.

Thus by eliminating the partial dependency in the table HOSPITAL and since all the other relations are in 1NF, without any partial dependency in them, the overall schema is in 2NF.

Here, since hos_id is the candidate key in either table which is an unique, primary key itself, the broken down tables have a lossless decomposition.
attributes(CONTRACT) U *attributes(HOSPITAL) = attributes(HOSPITAL'')*
attributes(CONTRACT) ∩ *attributes(HOSPITAL)* ≠ Φ

Third Normal Form (3NF):

Earlier in the schema design , Fig 5, shows the relation TREATMENT. <Excluded '*type*' attribute later on>

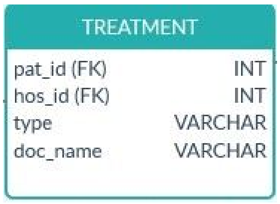

Fig 5: TREATMENT relation not in 3NF

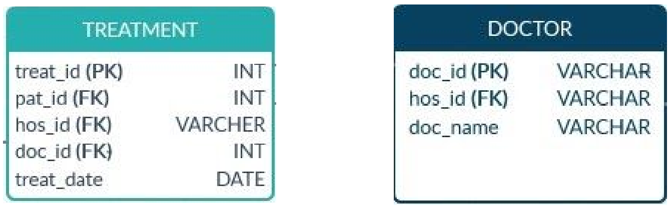

Fig 6: Solution to transitive dependency in Fig 5

The dependencies were *pat_id → {hos_id}* and *hos_id → {doc_name}*. This shows true transitive dependency between *pat_id* and *doc_name*, which is not true at all. Thus this table is not in 3NF. Hence, this was resolved using another table called DOCTOR to separately hold the contents of the doctor working in one or more hospitals/clinics. By doing so the TREATMENT table now is free from transitive dependency and since it is already in 1NF and 2NF, the relation is now in 3NF and the decomposition is lossless.

Giving another example, if the TREATMENT relation had another attribute called hos_name, indicating the name of the hospital where the treatment was carried out, this table would have transitive dependency because *treat_id → {hos_id}* and *hos_id → {hos_name}* would now come into existence and resulting in a 3NF violation.

Since all the relations are in 3NF, the schema relations are now normalised until the 3NF.

➢ Data Definition Language (DDL) commands

DDL commands are used to define the database schema. It deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. Commands include CREATE, DROP, ALTER, COMMENT etc.

```
CREATE TABLE MEDICINE
(
	med_id INT NOT NULL PRIMARY KEY ,
	name VARCHAR(25) NOT NULL,
	composition VARCHAR(70),
	mfg_date DATE NOT NULL,
	exp_date DATE NOT NULL,
	cost_per_tab INT,
	CHECK( exp_date <= "2030-01-01"),
	CHECK( mfg_date >= "2018-01-01")
);

CREATE TABLE STORES
(
	store_id VARCHAR(20) NOT NULL PRIMARY KEY,
	name VARCHAR(50) NOT NULL,
	address VARCHAR(70),
	contact VARCHAR(25) NOT NULL,
	store_man VARCHAR(25) NOT NULL
);

CREATE TABLE DEALER
(
	dealer_id INT NOT NULL PRIMARY KEY,
	name VARCHAR(25) NOT NULL,
	address VARCHAR(70),
	phone VARCHAR(25) NOT NULL
);

CREATE TABLE HOSPITAL
(
	hos_id VARCHAR(20) NOT NULL,
	name VARCHAR(25) NOT NULL,
	address VARCHAR(70),
	phone VARCHAR(25) NOT NULL,
	CONSTRAINT HOSPITAL_PK PRIMARY KEY(hos_id)
);

CREATE TABLE CONTRACT
(
	contract_id VARCHAR(20) NOT NULL,
	hos_id VARCHAR(20) NOT NULL,
	store_id VARCHAR(20) NOT NULL,
	CONSTRAINT CONTRACT_PK PRIMARY KEY(hos_id, store_id),
	CONSTRAINT CONTRACT_hos_id_FK FOREIGN KEY(hos_id) REFERENCES HOSPITAL(hos_id),
```

```
	CONSTRAINT CONTRACT_store_id_FK FOREIGN KEY(store_id) REFERENCES STORES(store_id)
);

CREATE TABLE DOCTOR
(
	doc_id INT NOT NULL,
	hos_id VARCHAR(20) NOT NULL,
	doc_name VARCHAR(25) NOT NULL,
	CONSTRAINT DOCTOR_PK PRIMARY KEY(doc_id, hos_id),
	CONSTRAINT DOCTOR_hos_id_FK FOREIGN KEY(hos_id) REFERENCES HOSPITAL(hos_id)
);

CREATE TABLE RETAIL
(
	retail_id VARCHAR(20) NOT NULL,
	med_id INT NOT NULL,
	store_id VARCHAR(20) NOT NULL,
	dealer_id INT NOT NULL,
	batchno INT NOT NULL,
	quantity_supplied INT NOT NULL,
	CONSTRAINT RETAIL_PK PRIMARY KEY(retail_id, med_id,store_id,dealer_id),
	CHECK(quantity_supplied <= 500)
);

CREATE TABLE PATIENT
(
	pat_id INT NOT NULL PRIMARY KEY,
	name VARCHAR(25) NOT NULL,
	address VARCHAR(70),
	phone VARCHAR(15) NOT NULL
);

CREATE TABLE TREATMENT
(
	treat_id INT NOT NULL UNIQUE,
	pat_id INT NOT NULL,
	hos_id VARCHAR(20) NOT NULL,
	doc_id INT NOT NULL,
	treat_date DATE NOT NULL,
	CONSTRAINT TREATEMENT_PK PRIMARY KEY(treat_id, pat_id, hos_id, doc_id),
	CONSTRAINT TREATEMENT_pat_id_FK FOREIGN KEY(pat_id) REFERENCES PATIENT(PAT_id),
	CONSTRAINT TREATEMENT_hos_id_FK FOREIGN KEY(hos_id) REFERENCES HOSPITAL(hos_id),
	CONSTRAINT TREATEMENT_doc_id_FK FOREIGN KEY(doc_id) REFERENCES DOCTOR(doc_id)
);

CREATE TABLE TRANSACTIONS
(
	bill_id INT NOT NULL,
	pat_id INT NOT NULL,
	store_id VARCHAR(20) NOT NULL,
	med_id INT NOT NULL,
	quantity INT NOT NULL,
```

```
        pur_date DATE NOT NULL,
        total INT ,
        CHECK(quantity <= 150),
        CHECK(pur_date < '2030-01-01' and pur_date > '2018-01-01'),
        CONSTRAINT TRANSACTIONS_PK PRIMARY KEY(bill_id,pat_id,med_id)
);

CREATE TABLE QUANT
(
        med_id INT ,
        store_id VARCHAR(20) ,
        quantity INT,
        CONSTRAINT QUANT_PK PRIMARY KEY(med_id,store_id)

);


ALTER TABLE RETAIL
ADD CONSTRAINT RETAIL_med_id_FK FOREIGN KEY(med_id) REFERENCES MEDICINE(med_id)
ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE RETAIL
ADD CONSTRAINT RETAIL_store_id_FK FOREIGN KEY(store_id) REFERENCES STORES(store_id)
ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE RETAIL
ADD CONSTRAINT RETAIL_dealer_id_FK FOREIGN KEY(dealer_id) REFERENCES DEALER(dealer_id)
ON DELETE CASCADE ON UPDATE CASCADE;


ALTER TABLE TRANSACTIONS
ADD CONSTRAINT TRANSACTIONS_pat_id_FK FOREIGN KEY(pat_id) REFERENCES PATIENT(pat_id)
ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE TRANSACTIONS
ADD CONSTRAINT TRANSACTIONS_med_id_FK FOREIGN KEY(med_id) REFERENCES MEDICINE(med_id)
ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE TRANSACTIONS
ADD CONSTRAINT TRANSACTIONS_store_id_FK FOREIGN KEY(store_id) REFERENCES STORES(store_id)
ON DELETE CASCADE ON UPDATE CASCADE;


ALTER TABLE QUANT
ADD CONSTRAINT QUANT_med_id_FK FOREIGN KEY(med_id) REFERENCES MEDICINE(med_id)
ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE QUANT
ADD CONSTRAINT QUANT_store_id_FK FOREIGN KEY(store_id) REFERENCES STORES(store_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

➢ Triggers

A trigger is a stored procedure in a database which automatically invokes whenever a special event in the database occurs. The following is a trigger used to populate the relation QUANT, which is a weak entity, whenever a row is inserted into the TRANSACTION and RETAIL tables. QUANT holds the total number of tablets of each MEDICINE available at each STORE.

```
delimiter //
CREATE TRIGGER  add_quant
AFTER INSERT ON RETAIL
FOR EACH ROW
BEGIN
INSERT INTO QUANT(med_id, store_id, quantity)
VALUES(NEW.med_id, NEW.store_id, NEW.quantity_supplied)
ON DUPLICATE KEY UPDATE
quantity = quantity + NEW.quantity_supplied;
END;
//
CREATE TRIGGER sub_quant
AFTER INSERT ON TRANSACTIONS
FOR EACH ROW
BEGIN
UPDATE QUANT
SET quantity = quantity - NEW.quantity
where med_id = NEW.med_id and store_id = NEW.store_id;
END;
//
delimiter ;
```

The first trigger, ***add_quant***, adds the corresponding quantity_supplied of a MEDICINE from the DEALER to a particular STORE, in the QUANT table. If it's an already existing medicine, then it is equivalent to saying ‘replenishing of stock’. If a new MEDICINE is being supplied then a new tuple under QUANT is added.

The second trigger, ***sub_quant***, subtracts the corresponding quantity of a MEDICINE from a STORE, upon a successful insert in the TRANSACTION table.

➢ SQL Queries

This Query ***<u>MUST BE RUN COMPULSORILY</u>*** in order to populate the transaction table completely. This should be done after inserting into the TRANSACTIONS table, before querying anything else.

```
UPDATE TRANSACTIONS, MEDICINE
SET TRANSACTIONS.total = TRANSACTIONS.quantity * MEDICINE.cost_per_tab
where TRANSACTIONS.med_id = MEDICINE.med_id;
```

1. Get the grand total of a bill issued to a patient

```
SELECT bill_id, ANY_VALUE(pat_id) AS pat_id, ANY_VALUE(store_id) AS store_id, SUM(total) AS
Grand_total
FROM TRANSACTIONS
GROUP BY bill_id;
```

bill_id	pat_id	store_id	Grand_total
21	50895	str10	3440
34	77485	str1	470
44	66778	str10	7740
78	11235	str8	495
89	52369	str4	40
123	60085	str2	450
233	33009	str3	90
677	56689	str1	275

8 rows in set (0.04 sec)

2. List patients who got treated today and bought medicines today.

```
SELECT DISTINCT(A.pat_id) , name, B.med_id
FROM PATIENT A, TRANSACTIONS B, TREATMENT C
WHERE A.pat_id = B.pat_id AND B.pur_date = CURRENT_date() AND B.pat_id IN
(
        SELECT C.pat_id
        FROM TREATMENT C
        WHERE C.treat_date = CURRENT_date()
);
```

pat_id	name	med_id
60085	Martin	6
60085	Martin	2
77485	John	5
77485	John	3

4 rows in set (0.00 sec)

3. List the necessary details, where a patient who has transacted in a STORE, has been treated in a hospital having a contract with the same store. (Useful for operations like giving discounts for tie ups etc).

```
SELECT DISTINCT(A.pat_id), A.name, A.phone, C.hos_id, E.name, D.store_id, F.name
FROM PATIENT A, TREATMENT B, CONTRACT C, TRANSACTIONS D, HOSPITAL E, STORES F
WHERE A.pat_id = D.pat_id and E.hos_id = C.hos_id and F.store_id = D.store_id and(D.pat_id,
C.hos_id, D.store_id) IN
(
        SELECT B.pat_id, B.hos_id, C.store_id
        FROM TREATMENT B, CONTRACT C
        WHERE B.hos_id = C.hos_id
);
```

pat_id	name	phone	hos_id	name	store_id	name
33009	James	9978665543	hos3	Ashwini ENT Clinic	str3	MSR Pharma
77485	John	7766345672	hos5	Roma Dental Clinic	str1	Frank Ross Pharma - Emami Group
66778	Ankit	5566789045	hos6	Manasa Pediatrics	str10	Life Line Pharma & General Store

3 rows in set (0.00 sec)

4. Find Total Sales in each store in non-increasing order.
<Newer version of MYSQL doesn't support FULL OUTER JOIN, but the same can be mimicked by using LEFT and RIGHT JOINS. >

```
SELECT ANY_VALUE(B.store_id) AS STORE_ID, ANY_VALUE(B.name) AS STORE_NAME, SUM(A.total)
AS TOTAL_SALES
FROM TRANSACTIONS A LEFT JOIN STORES B
USING(store_id) GROUP BY A.store_id
UNION
SELECT ANY_VALUE(B.store_id) AS STORE_ID, ANY_VALUE(B.name) AS STORE_NAME, SUM(A.total)
AS TOTAL_SALES
FROM TRANSACTIONS A RIGHT JOIN STORES B
USING(store_id) GROUP BY B.store_id
ORDER BY TOTAL_SALES DESC;
```

STORE_ID	STORE_NAME	TOTAL_SALES
str10	Life Line Pharma & General Store	11180
str1	Frank Ross Pharma - Emami Group	745
str8	Noble Pharma	495
str2	Sagar Pharma	450
str3	MSR Pharma	90
str4	Unitree Pharma & General Store	40
str5	Misba Pharma & General Stores	NULL
str6	Chetan Pharma	NULL
str7	Praveen Pharma	NULL
str9	Pragathi Pharma & General Stores	NULL

10 rows in set (0.00 sec)

5. The stores want to know what medicines are unsold in their stores, so that they can effectively save money by ordering less next time.

```
SELECT DISTINCT(A.store_id), A.name, D.med_id, D.quantity
FROM STORES A, RETAIL B, TRANSACTIONS C, QUANT D
WHERE D.store_id = A.store_id AND A.store_id=B.store_id AND B.store_id NOT IN
(
        SELECT DISTINCT(C.store_id)
        FROM TRANSACTIONS C
);
```

store_id	name	med_id	quantity
str7	Praveen Pharma	7	90
str6	Chetan Pharma	5	135
str6	Chetan Pharma	3	75
str5	Misba Pharma & General Stores	6	100

4 rows in set (0.00 sec)

6. Display retail and dealer information about all Registered dealers who have supplied a bulk order of >= 100 tablets.

```
SELECT B.dealer_id, A.name, B.retail_id, B.store_id,  B.med_id, B.quantity_supplied
FROM RETAIL B, DEALER A
WHERE A.dealer_id = B.dealer_id and EXISTS
(
        SELECT B.dealer_id
        FROM DEALER A
        WHERE A.dealer_id = B.dealer_id and B.quantity_supplied >= 100
)
ORDER BY A.dealer_id;
```

dealer_id	name	phone	retail_id	store_id	med_id	quantity_supplied
3	Morty	1122457896	ret4	str6	5	135
4	Kendrick	9985665231	ret13	str10	10	180
7	Tom	1100223645	ret10	str4	8	120
8	Tim	9877754236	ret5	str4	5	300
9	Suzy	5566369875	ret1	str1	2	200
9	Suzy	5566369875	ret2	str2	2	200
10	Cindy	5522369877	ret14	str10	9	345
10	Cindy	5522369877	ret3	str2	6	100
10	Cindy	5522369877	ret6	str5	6	100

9 rows in set (0.00 sec)

7. Give an output which shows the MEDICINE sold at a store, which was supplied by a dealer, to a patient along with the purchased date and expiry date.

```
SELECT  A.store_id, A.med_id, C.name, A.dealer_id, C.exp_date, B.pur_date, B.pat_id
FROM MEDICINE C JOIN TRANSACTIONS B JOIN RETAIL A
ON B.med_id = A.med_id and C.med_id = B.med_id AND
C.med_id = A.med_id and B.store_id = A.store_id
ORDER BY A.store_id;
```

store_id	med_id	name	dealer_id	exp_date	pur_date	pat_id
str1	2	Saridon	9	2022-06-10	2020-04-20	56689
str10	10	Otrivin	4	2022-08-11	2020-01-01	50895
str10	10	Otrivin	4	2022-08-11	2019-09-11	66778
str2	2	Saridon	9	2022-06-10	2020-05-30	60085
str2	6	Glycomet	10	2023-01-24	2020-05-30	60085
str3	1	Crocin	3	2021-05-24	2020-05-01	33009
str4	8	Volini	7	2022-08-13	2020-01-10	52369
str8	11	Ciplox	4	2020-09-20	2020-04-20	11235

8 rows in set (0.00 sec)

8. Display the daily transaction statistics of every store

```
SELECT ANY_VALUE(a.store_id) as store_id, ANY_VALUE(b.name) as store_name,
ANY_VALUE(COUNT(a.store_id)) as No_of_Trancts, ANY_VALUE(AVG(a.total)) as AVG_amt,
ANY_VALUE(MIN(a.total)) as MIN_amt, ANY_VALUE(MAX(a.total)) as MAX_amt,
ANY_VALUE(a.pur_date)as Purch_Date
FROM TRANSACTIONS a JOIN STORES b
ON a.store_id = b.store_id
GROUP BY(a.pur_date) ORDER BY a.pur_date DESC;
```

store_id	store_name	No_of_Trancts	AVG_amt	MIN_amt	MAX_amt	Purch_Date
str1	Frank Ross Pharma - Emami Group	4	230.0000	120	350	2020-05-30
str3	MSR Pharma	1	90.0000	90	90	2020-05-01
str8	Noble Pharma	2	385.0000	275	495	2020-04-20
str4	Unitree Pharma & General Store	1	40.0000	40	40	2020-01-10
str10	Life Line Pharma & General Store	1	3440.0000	3440	3440	2020-01-01
str10	Life Line Pharma & General Store	1	7740.0000	7740	7740	2019-09-11

6 rows in set (0.00 sec)

➢ Conclusion

This Pharmaceutical - Retail Management system works well when concentrated towards the retail sector of buying and selling. It also works well with the Hospital side of the Pharmacy which involves the whole process of the patient getting treated in a hospital by many doctors. This database system provides essential daily store transactions statistics. The record of medicine along its dealer selling to a particular store can also be kept. The contract between the store and hospital along with details of each patient, hospital, store, dealer and medicine are also stored.

When it comes to limitations if a *PATIENT* buys 2 or more different medicines in the same store in the same day, then the *bill_id, pat_id, store_id* has to be the same or else the output doesn't make sense (Eg: bill_id is same but store_id is different on the same purchase date!). *CONTRACTS* are assumed to be that 1 *STORE* can have contracts with many *HOSPITALS* and not the other way around, so that it focuses on the importance of each store. *treat_id, hosp_id, pat_id, store_id, contract_id, dealer_id and med_id* are assumed to be unique in their respective tables. It is assumed that a doctor can only work in 1 hospital and multiple doctors can work in 1 hospital. Before transacting, it has to be taken care that the corresponding quantity of tablets are available in the store.

Future enhancements include adding a *DRUG_MANUFACTURE* relation so that in case if, for example irregular quantities of paracetamol was mixed in tablets on a particular date. So now all the medicines containing Paracetamol manufactured by that Manufacturer should be tracked and its corresponding supplies, by the *DEALER*, to the *STORES* have to be tracked and canceled in the *RETAIL* table.

There is also a need for a table like *DEALER_COST* which has prices for each medicine the dealer charges for selling it to a store (more like a Cost Price and Selling Price concept). Using this additional table, we can also view the net profit or loss the pharma store makes upon each kind of medicine. This will further improve the Pharma stores to order medicine depending on its popularity and the stock in store. More functionalities could be added to QUANT like, alerting when a particular medicine is falling out of stock etc using stored procedures in the future.

There is a need for a simple web - app here due to the presence of a large number of relations(tables), hence this has to be implemented in the future for better usability.