Patient Clinic Management Enhanced

A. System modification

1. Updated outpatient clinic database

<u>Requirement</u>:

Configure a system that logs all inserts, changes, and deletes to a series of database "history" tables. It is possible to ascertain how, precisely, the data in the database are changed over time by recording all modifications to the underlying data, as well as who made the changes and when. Database audit logs are particularly helpful in situations where the accuracy of the data is critical or when there are many users all working on the same underlying data.

<u>Solution</u>:

- **Create an additional history table**

As the table named "billing_detail" need to track all the changes of data

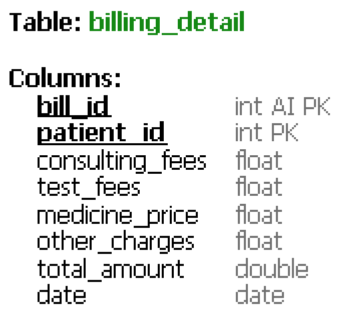


Table: billing_detail

Columns:

bill_id	int AI PK
patient_id	int PK
consulting_fees	float
test_fees	float
medicine_price	float
other_charges	float
total_amount	double
date	date

Adding an additional historical data table as the below

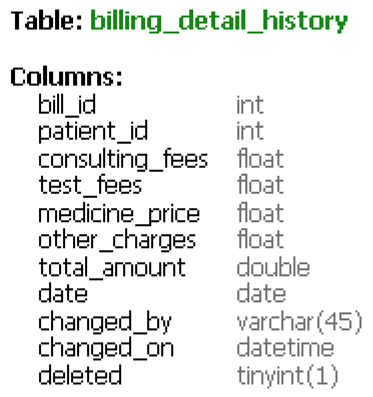


Table: billing_detail_history

Columns:

bill_id	int
patient_id	int
consulting_fees	float
test_fees	float
medicine_price	float
other_charges	float
total_amount	double
date	date
changed_by	varchar(45)
changed_on	datetime
deleted	tinyint(1)

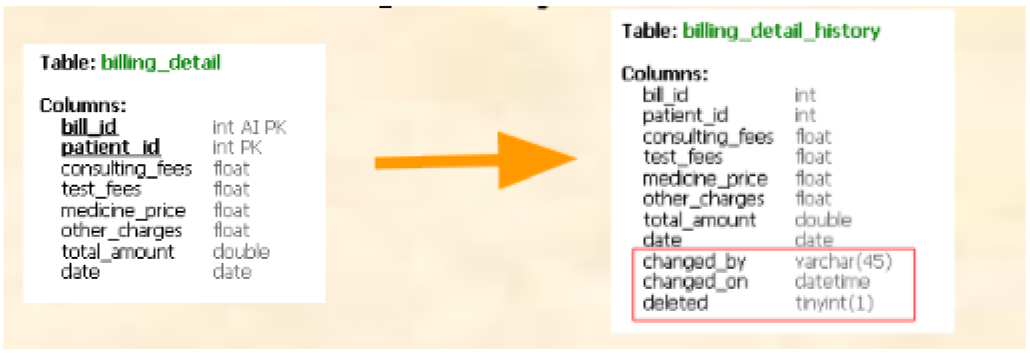


Table: billing_detail

Columns:

bill_id	int AI PK
patient_id	int PK
consulting_fees	float
test_fees	float
medicine_price	float
other_charges	float
total_amount	double
date	date

Table: billing_detail_history

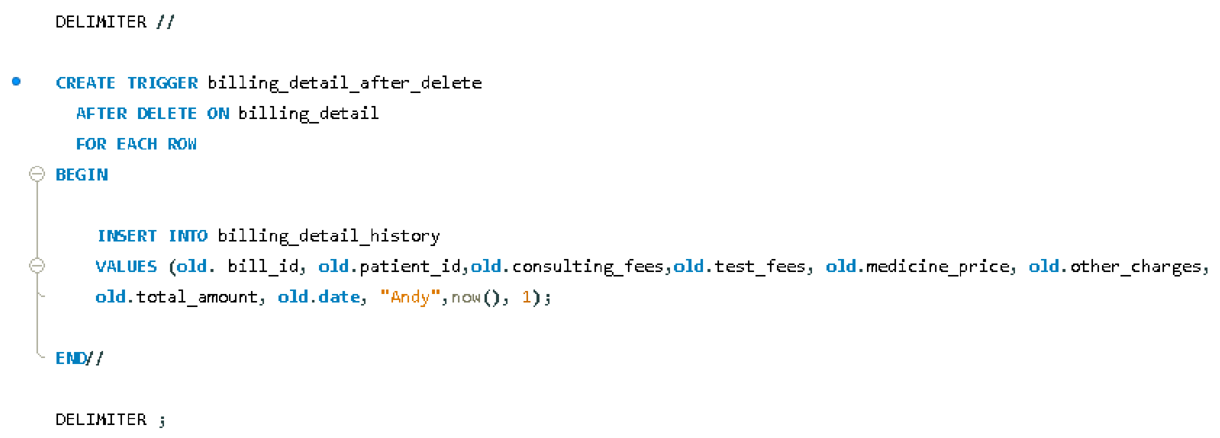
Columns:

bill_id	int
patient_id	int
consulting_fees	float
test_fees	float
medicine_price	float
other_charges	float
total_amount	double
date	date
changed_by	varchar(45)
changed_on	datetime
deleted	tinyint(1)

three additional columns might be named ChangedBy, ChangedOn, and Deleted.
We need to build an extra "history" table for each table whose history of changes needs to be documented. This "history" table will have the same schema as the original table, with three extra columns to record who made the change, why it happened, and whether or not the change resulted in the deletion of the record. These three new columns may be called ChangedBy, ChangedOn, and Deleted.

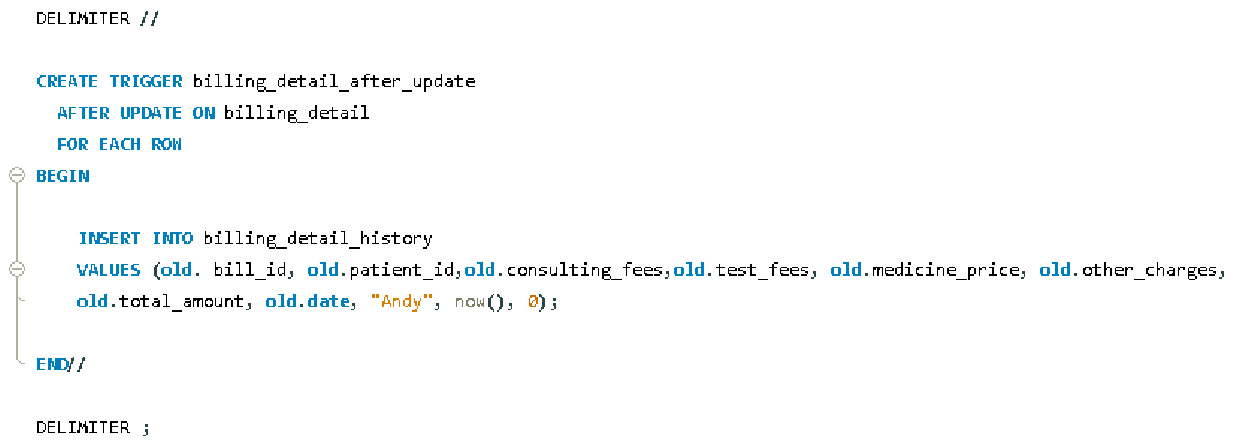
- **Using the trigger to fire the event handler for tracking the changes of tables**

Using the trigger to fire the event handler for tracking the changes of tables - After delete a record. This statement is the coding of the trigger to fire the event handler after deleting a record in the table named billing detail.



```
DELIMITER //

CREATE TRIGGER billing_detail_after_delete
  AFTER DELETE ON billing_detail
  FOR EACH ROW
BEGIN

    INSERT INTO billing_detail_history
    VALUES (old. bill_id, old.patient_id,old.consulting_fees,old.test_fees, old.medicine_price, old.other_charges,
    old.total_amount, old.date, "Andy",now(), 1);

END//

DELIMITER ;
```

Using the trigger to fire the event handler for tracking the changes of tables - After updating an record. This statement is the coding of the trigger to fire the event handler after updating a record in the table named billing detail.



```
DELIMITER //

CREATE TRIGGER billing_detail_after_update
  AFTER UPDATE ON billing_detail
  FOR EACH ROW
BEGIN

    INSERT INTO billing_detail_history
    VALUES (old. bill_id, old.patient_id,old.consulting_fees,old.test_fees, old.medicine_price, old.other_charges,
    old.total_amount, old.date, "Andy", now(), 0);

END//

DELIMITER ;
```

<u>Advantages and disadvantages of this solution</u>

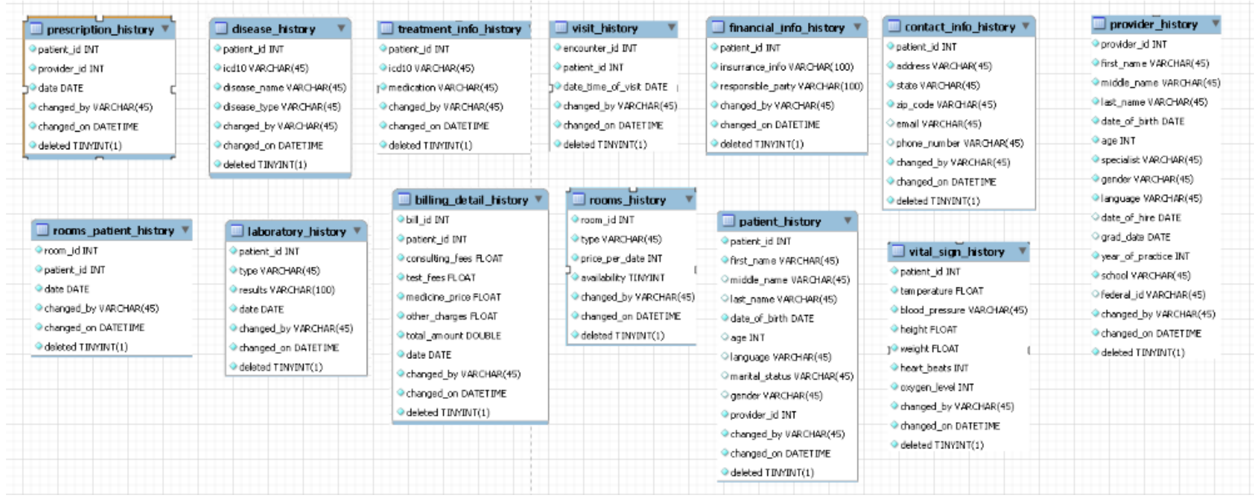
Advantages of this solution is that it makes searching the audit log very easy since there the "history" table's schema is a superset of the original's. For example, if the user want to see all changes that were made to a particular billing detail, user can just run a query like the following

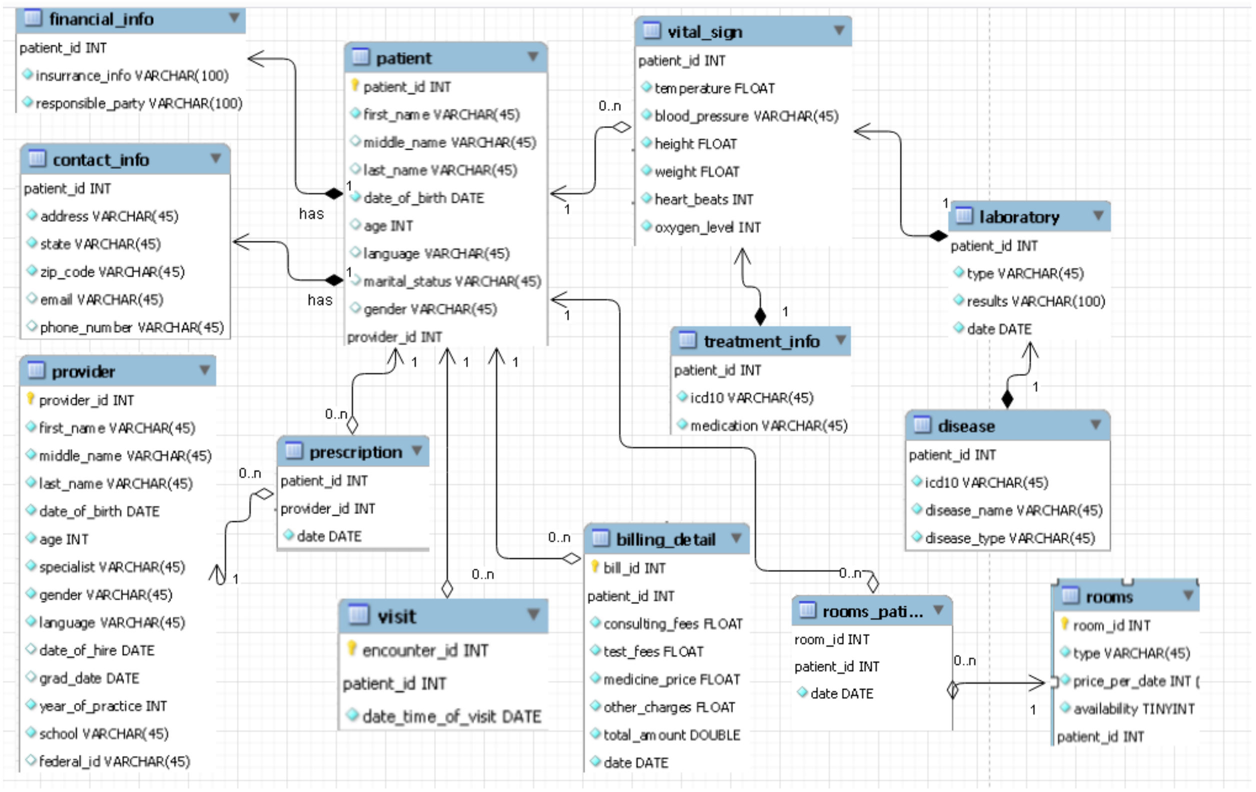
```
-- makes searching the audit log very easy. For example,
SELECT * FROM Billing_Detail_History
WHERE patient_id = patient_id
-- see all changes made to the system by user named "Andy" between May 15 and 20th
SELECT * FROM Billing_Detail_History
WHERE Changed_By = "Andy" AND Changed_On BETWEEN '2020-05-15' AND
'2020-05-20'
```

The downside of this strategy is that since the "history" table's schema is closely linked to the original's, all changes to the original table's schema would also be made to the "history" table's. Furthermore, since this method provides one "metadata" table for each table whose modifications must be recorded, it has the potential to double the number of tables in your database.
This is the method I used for monitoring database changes in this project. I've discovered that it fits well for mature data structures where schema updates to the original tables are infrequent. Each table need to create an additional history table for tracking data changes

2. UML Data Model

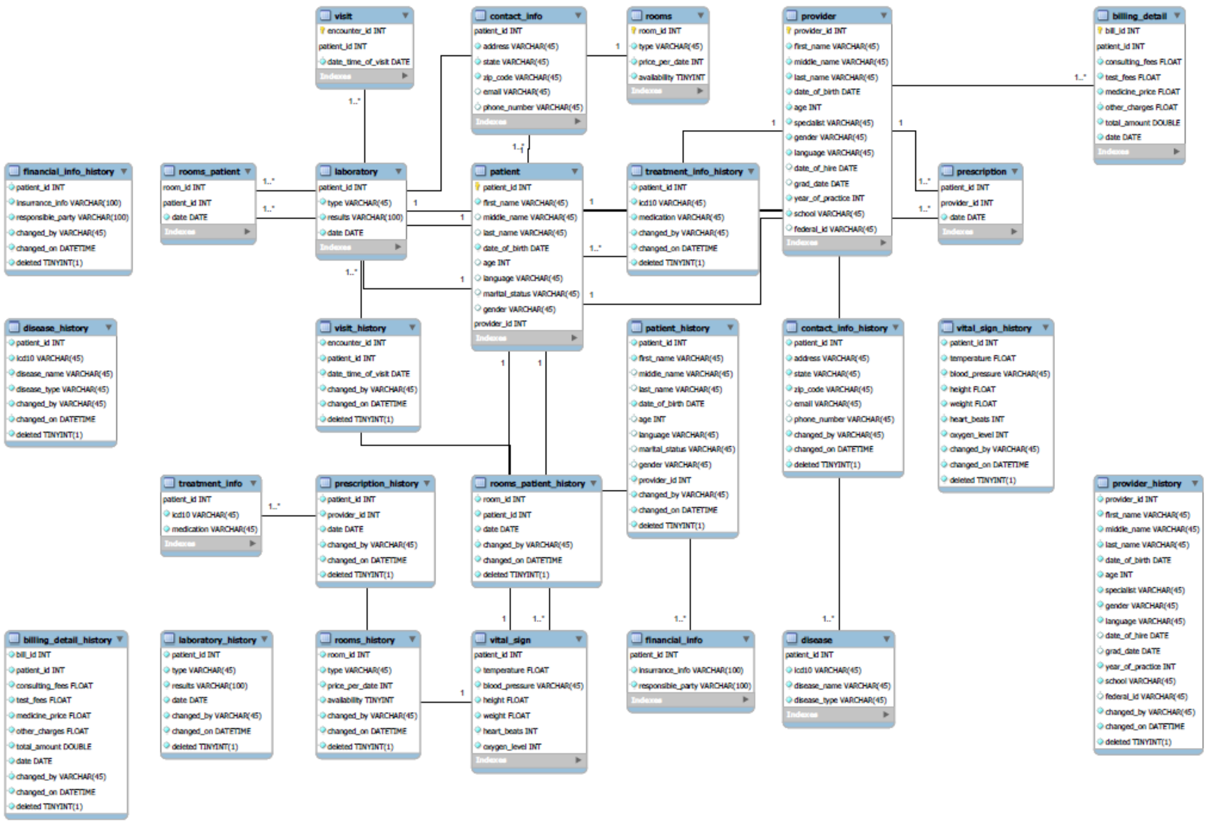
The list of history tables for tracking the changes of data.





3. **ER Diagram**

The only change is adding some historical data table for each table that needs to track the changes of data.



B. List my API for the required features:

1. Role-based access control

You can designate whether the user is an administrator, a professional user, or an end-user, and you can match responsibilities and access permissions to your employees' organizational positions. Permissions are granted only as much access as is needed for workers to perform their work.When a user is added to a task list, the user has access to all of the positions in that group. When they are disabled, access is blocked. Users can also be distributed to different classes if they need temporary access to certain data or services and then deleted until the project is over.

There are some kinds of roles in this system.

Administrative role: access for users that perform administrative tasks,

Technical role: assigned to users that perform technical tasks, such as developing some new features based on the the requirements, maintenance the database system(backup database)

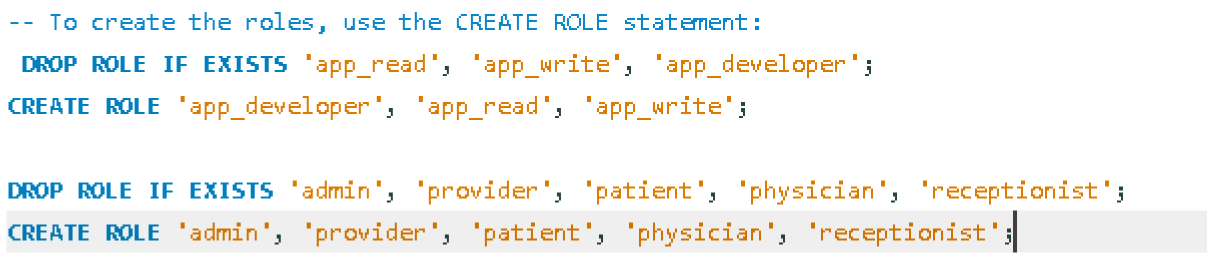
Patient role: access for users to perform the patient tasks, such as see the private theirs information, set up an visit order with physician, update the their information

Physician role: access for user to perform the physician tasks, such as see the private theirs information, update their information, cancel an prescription order

Receptionist role: access for users to perform the receptionist tasks, such as see all patient information, see all information of room (available or occupied), set up a visit order with physician for the patient if the patient requires by telephone or website via internet.
Billing role – access for one end-user to the billing account.

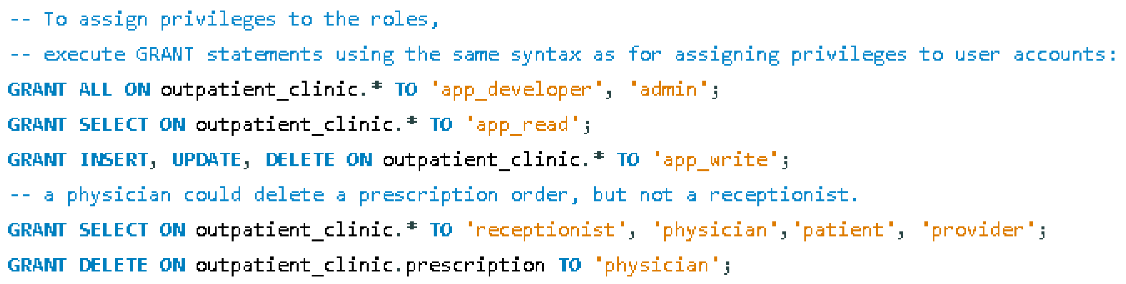
Consider this scenario:
A database called outpatient clinic system is used for an application. Accounts for developers who build and manage the program, as well as people who communicate with it, may be associated with it. Developers need complete database access. Some users need only read access, while others need both read and write access.
Establish positions as names for the appropriate privilege sets to prevent assigning privileges individually to potentially multiple user accounts. By assigning the requisite responsibilities, this makes it simple to assign the required rights to user accounts.

Creating the roles



```
-- To create the roles, use the CREATE ROLE statement:
 DROP ROLE IF EXISTS 'app_read', 'app_write', 'app_developer';
CREATE ROLE 'app_developer', 'app_read', 'app_write';

DROP ROLE IF EXISTS 'admin', 'provider', 'patient', 'physician', 'receptionist';
CREATE ROLE 'admin', 'provider', 'patient', 'physician', 'receptionist';
```

Assign privileges to the roles



```
-- To assign privileges to the roles,
-- execute GRANT statements using the same syntax as for assigning privileges to user accounts:
GRANT ALL ON outpatient_clinic.* TO 'app_developer', 'admin';
GRANT SELECT ON outpatient_clinic.* TO 'app_read';
GRANT INSERT, UPDATE, DELETE ON outpatient_clinic.* TO 'app_write';
-- a physician could delete a prescription order, but not a receptionist.
GRANT SELECT ON outpatient_clinic.* TO 'receptionist', 'physician','patient', 'provider';
GRANT DELETE ON outpatient_clinic.prescription TO 'physician';
```

CREATE ROLE and DROP ROLE create and remove roles.
GRANT and REVOKE assign privileges to revoke privileges from user accounts and roles.

2. **User authentication**
MySQL identities are stored in the user table of the mysql system database.
An account is identified by a user name and the client host(s) from which the user may connect to the server. Authentication credentials, such as a pin, can also be associated

with an account. The account verification plugin handles the passwords. MySQL offers a variety of security plugins.

Account UserNames and Passwords

User names, as used by MySQL for authentication purposes,

Passwords stored in the user table are encrypted using plugin-specific algorithms.

Creating Accounts and Granting Privileges

To create the user account, use the CREATE USER statement as the below

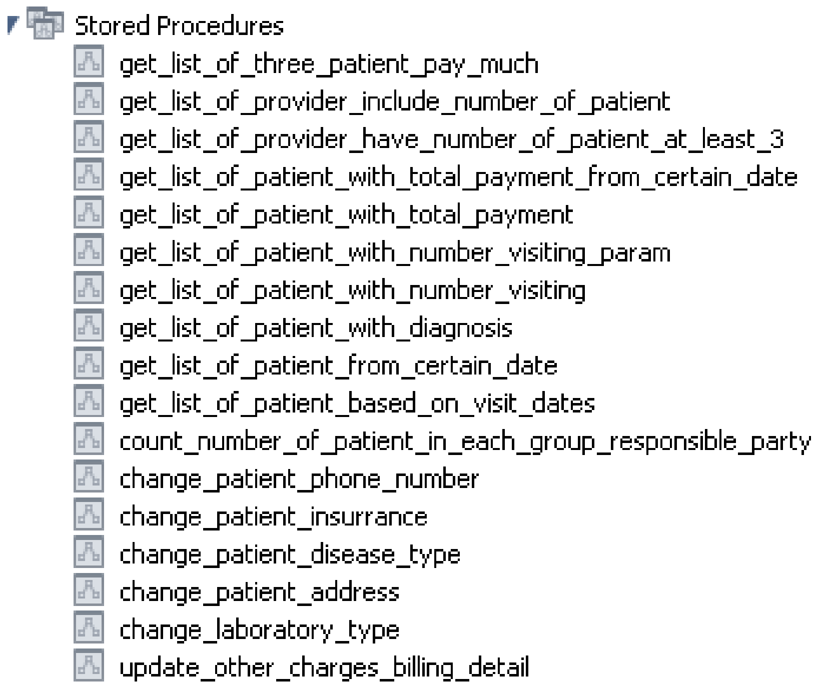
```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'adminpass';
CREATE USER 'receptionist1'@'localhost' IDENTIFIED BY 'receptionist1pass';
CREATE USER 'physician1'@'localhost' IDENTIFIED BY 'physician1pass';
CREATE USER 'patient1'@'localhost' IDENTIFIED BY 'patient1pass';
CREATE USER 'provider1'@'localhost' IDENTIFIED BY 'provider1pass';
```

Assign each user account its required privileges

```
GRANT 'admin' TO 'admin'@'localhost';
GRANT 'receptionist' TO 'receptionist1'@'localhost';
GRANT 'physician' TO 'physician1'@'localhost';
GRANT 'patient' TO 'patient1'@'localhost';
GRANT 'provider' TO 'provider1'@'localhost';
```

3. **Stored procedures**

   This database supports some functionalities, such as allowing the user to edit the wrong information, for example patient address or insurrance.It supports searching the patient record based on the name, or other information such as visit dates.It also supports the reporting functions. For example, it could list all patients who satisfy certain selection conditions, such as who visited on certain days, or who have been seen by a certain doctor or provider, and so on.



Stored Procedures
get_list_of_three_patient_pay_much
get_list_of_provider_include_number_of_patient
get_list_of_provider_have_number_of_patient_at_least_3
get_list_of_patient_with_total_payment_from_certain_date
get_list_of_patient_with_total_payment
get_list_of_patient_with_number_visiting_param
get_list_of_patient_with_number_visiting
get_list_of_patient_with_diagnosis
get_list_of_patient_from_certain_date
get_list_of_patient_based_on_visit_dates
count_number_of_patient_in_each_group_responsible_party
change_patient_phone_number
change_patient_insurrance
change_patient_disease_type
change_patient_address
change_laboratory_type
update_other_charges_billing_detail

There are two kind of stored procedure in this database system

- The simple stored procedures
    - ❏ change of patient address
    - ❏ change the patient disease type
    - ❏ change the patient insurance
    - ❏ change laboratory type
    - ❏ update the other charges in billing detail

- The complex stored procedures
    - ❏ Get the list of three patient pay much in clinic
    - ❏ Get the list of provider include number of patient
    - ❏ Get the list of provider have number of patient at least 3 or certain number as parameter
    - ❏ Get the list of patient with total payment from certain date to certain date
    - ❏ Get the list of patient with total payment of each person
    - ❏ Get the list of patient with number visiting the clinic as parameter
    - ❏ Get the list of patient with diagnosis as parameter
    - ❏ Get the list of patient visiting the clinic from certain date to certain date
    - ❏ Cont number of patient in each group responsible party

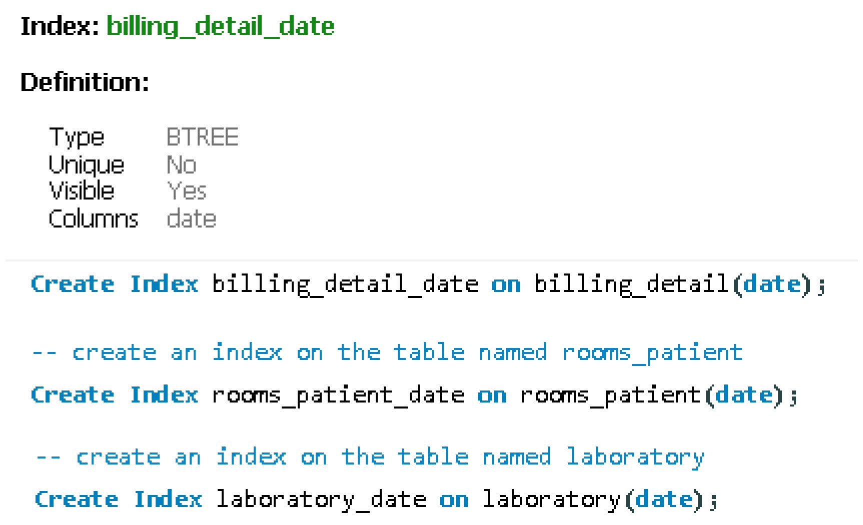
4. **Indexes**

   Create some indexes on the tables

   Use the CREATE INDEX statement to create the indexes

   For example:

   Create index named billing_detail_date on the table named billing_detail



Index: billing_detail_date

Definition:

```
Type      BTREE
Unique    No
Visible   Yes
Columns   date
```

```
Create Index billing_detail_date on billing_detail(date);

-- create an index on the table named rooms_patient
Create Index rooms_patient_date on rooms_patient(date);

-- create an index on the table named laboratory
Create Index laboratory_date on laboratory(date);
```

   Creating some indexes on some tables, such as patient, provider, rooms, vital_sign, disease...

   Materialization

   Using materialization to speed up the query execution by producing a subquery result as a temporary table.

   Select first_name, last_name from patient p where p.patient_id not in
   (select patient_id from rooms_patient where date >= ‘2020-08-01’)

   Materialization is used by the optimizer to allow more efficient subquery processing. Materialization accelerates query execution by producing a subquery result as a temporary table, which is typically stored in memory. When MySQL needs the subquery result for the first time, it materializes it into a temporary table. When the result is needed again, MySQL refers to the temporary table. The optimizer can use a hash index to index the table, making lookups quick and cheap. To remove duplicates and make the table smaller, the index includes unique values.

5. **Views**

   This system is implementing the list of views. For examples,

   - ❏ View for searching all the list of room
   - ❏ View for list of all the patients

- ❑ View for searching the first five patients records
- ❑ View for searching the patients has the first name or middle name contain parameter character
- ❑ View for list of the provider has the year of practice at least 6 years
- ❑ View for list of the provider male has the year of practice at least 6 years
- ❑ View for list of the provider graduated from the certain school such as Harvard
- ❑ View for searching of patient records based on the visit dates
- ❑ View for the listing of all patients who satisfy certain selection criteria, such as those who have been given a certain diagnosis
- ❑ View for the listing of all patients who satisfy certain selection criteria, such as who have been seen by a certain doctor has first name contain "jack"
- ❑ View for the listing of all patients who satisfy certain selection criteria, such as who visited on certain days

6. Triggers

This system is implementing some triggers on the tables.
Using the trigger to fire the event handler for tracking the changes of tables
- After updating an record and
- After deleting an record

Trigger to enforce data consistency
This trigger to make sure the total amount in table named "billing_detail" is equal of sum of other fees

Total_amount = consulting_fees + test_fees+medicine_price+other_charges

Reference sources

https://www.mysqltutorial.org/stored-procedures-parameters.aspx
https://www.4guysfromrolla.com/webtech/041807-1.shtml
http://www.4guysfromrolla.com/webtech/042507-1.shtml
https://www.4guysfromrolla.com/webtech/080305-1.shtml