Database

The database contains most of the information used by the web application. A database is a collection of related data in the form of schemas, tables, views, stored procedures, triggers, indexes, and other objects.

Data Interface

The data interface is the point of contact between the application data access layer and the database. Similar to the software package interfaces you already know, the data interface provides a pre-defined set of functions that will serve as the gateway to the data from the application’s data access layer.
Universe of Discourse

A database represents some aspect of the real world called the mini-world or the universe of discourse. You will design a database based on the description of the mini-world contained in this section. The subject matter of this project was selected to increase the likelihood of students being familiar with its main concepts and make it easier to find relevant information online to guide the design decisions. However, when explicit requirements given in this description contradict with the information you find from other sources regarding this subject matter, your solution must satisfy the requirements given here.

You must explicitly state and justify all assumptions that are not part of this description.

Your task consists of the design of the database and data interface of a hospital system. As such, you will deal with patients, doctors, nurses, receptionists, accountants, diagnoses, drugs, laboratory, radiology, bills, payments, suppliers, insurance companies, and so on.

You will need to store information on the full name, gender, and date of birth of your patients and one of the addresses to be used for mailing the paper-based bill. Each patient may have one or more credit cards on file, each one associated with a billing address. Each patient may have several addresses and several credit cards, but each credit card must only have one billing address. You must allow patients to specify their preferred credit card associated with the billing address for each payment. In order to communicate with patients and send a billing reminder, you must store their e-mail address and have the option of adding their phone number.

You will control access to patients’ accounts with credentials consisting of a username and a password. The username will be the patient’s e-mail address and the password will be stored encrypted in the database. You must allow patients to change their e-mail address and password at any time. At the same time, you may also want to store the patient’s insurance policy ID as a reference to facilitate your project development.

A patient will first be registered by a receptionist. The receptionist will keep a record of the patient and create or locate the patient’s insurance information for billing purposes. The receptionist is responsible for recording the date of the patient’s first visit, checking the doctors’ availability, making an appointment with a doctor on a particular date, and informing the patient to come on that date. To simplify this process, we assume the availability of doctors is measured as the date based. Also, we won’t ask you to simulate scenarios such as an appointment with a preferred doctor. You can assume that if for a particular day there’s a doctor available, then the patient will be assigned to that doctor. If there are many doctors available, the patient will be randomly assigned to one doctor. You need to keep a list of receptionists’ names and contact information such as address and phone number.
Before the patient sees the doctor, a nurse will record the check-in time and measure some vital information such as height, weight, blood pressure, and body temperature. The nurse is also responsible for room-related issues such as assigning rooms to patients, recording the usage of rooms, recording patients’ admission date and date of discharge. You need to keep a list of nurses’ names, gender, and contact information such as address and phone number.

The doctors will have access to the patient’s medical record, including the patient’s vital information recorded by the nurses. The doctors will determine a diagnosis, write a prescription, and arrange laboratory or radiology tests as necessary. The doctor’s name, specialty, gender, and contact information such as address and phone number should be stored in your system.

You need to store the prescription medicine information. The medicine must have a name, description, picture, and price. Each medicine must be associated with one category. You will design a way of storing trees of categories. Even though you should think of the category hierarchy as trees, you will store them as a table in the database. Populate the table with a set of categories of your choice. Your chosen category hierarchy must be at least three levels deep. Medicines may be associated with any category, not just with the leaves of your trees. The following are two very basic trees:

The rooms designated by the nurses as well as the patients’ laboratory and radiology tests should all be kept on record for billing purposes, together with the date. The charges for a room are by date and the charges for radiology and laboratory tests are by usage count.

All expenses incurred by a patient (prescription drugs, lab tests, X-rays, consultation from doctors or nurses, room usage etc.) will be combined into a final bill. Each payment created from the bill must be associated with exactly one credit card. You may assume that cards are valid and have enough funds to cover the transaction if your insurance policy will not cover all the expenses. The patient’s system balance will be automatically applied to payments when a payment is created; only the outstanding balance after using up the system balance is charged to the patient’s credit card. Since patients can update or delete their credit cards and addresses on their accounts, you must store a copy of the
payment information for each patient per visit so that the system can generate invoices in the future (you will not model invoices, though). It must be possible to determine the amount of system balance, insurance coverage, and credit card charge that are paid within one payment.

When a patient completes the purchase of the medicine, the hospital’s inventory is updated accordingly. Since medicine pricing may change at any time, you must store the price of each medicine at the time the bill is generated so that you know what price to charge. Each medicine will have a low inventory threshold. When a payment brings the count of one medicine below the threshold, a re-stocking reminder must be generated by the database.

The hospital does not manufacture any medicines. Instead, it purchases all medicines from suppliers. In order to expedite orders, the hospital maintains a stock of all the medicines it offers. You need to keep a list of suppliers and store the company name, business address, sales representative’s contact information such as name, address, email address, phone number, etc., and discount percentage, i.e., a fixed percent discount applied to all purchases from this supplier. Each medicine must have at least one supplier. Each supplier in the database must supply at least one medicine. Each supplier may supply medicines at different prices.

Re-stocking reminders are used by the hospital to place orders with medicine suppliers. You will not model the orders to the suppliers beyond the details of re-stocking reminders. A re-stocking reminder must have information about the medicine name, category, and the best supplier to order from. The system will determine the best supplier at the time of the creation of the reminder by comparing the final cost of the medicine to the hospital taking into account the supplier’s price for the medicine and the arranged discount, if any.

An accountant will be responsible for coordinating payments with the patient and patient’s insurance company at the same time. If a patient doesn’t pay his or her share with a credit card on the due date, a billing reminder must be generated by the database. To implement the billing reminder, you need to store information on the current date and compute the distance between the current date and the date the bill was generated. For consistency, the first due date is 3 months after the bill is generated. You need to keep a list of accountants’ name, gender and contact information such as address and phone number.

Regarding a patient’s insurance company, you must keep a record of the patient’s insurance policy number, insurance status, coverage amount, coverage percentage per visit, etc. To simplify this process, you may assume that each patient will get a constant coverage amount in total and per visit. The coverage percentage is a fixed constant regardless of the services the patient gets billed by the hospital.

For each visit to the doctor, the insurance will cover a certain percentage of the cost and the rest must be paid entirely by the patient with a credit card on file before the due date in order to avoid a billing reminder. When the insurance amount is nearly used up, for that patient’s visit the insurance coverage will be possibly less than the percentage it should cover. Obviously, for those patients with an invalid
insurance status, the insurance company will not cover any of the expenses they incur. This case also applies at the time when the coverage amount reduces to 0.

Data definition language

Data definition language (DDL) statements let you perform these tasks:
- Create, Alter, and Drop schema objects in your database;
- Grant and revoke privileges and roles;
- Analyze information on a table.

We will discuss the DDL statement components required by Project 2 in more detail:

A. CREATE TABLE statement

A commonly used CREATE command is the CREATE TABLE command. The typical usage is:

```
CREATE TABLE [table name] ([column definitions]) [table parameters]
```

An example statement to create a table named `employees` with a few columns is:

```sql
CREATE TABLE employees (  
id INTEGER PRIMARY KEY,  
first_name VARCHAR(50) not null,  
last_name VARCHAR(75) not null,  
fname VARCHAR(50) not null,  
dateofbirth DATE not null
);
```

B. ALTER TABLE statement

The ALTER statement modifies an existing database object. An ALTER statement in SQL changes the properties of an object. The typical usage is:

```
ALTER TABLE [table name] [table parameters]
```

For example, the command to add (then remove) a column named `bubbles` for an existing table named `sink` is:

```
ALTER TABLE sink ADD bubbles INTEGER;  
ALTER TABLE sink DROP COLUMN bubbles;
```

C. DROP TABLE statement

The DROP statement destroys an existing database object. A DROP statement in SQL removes a component from a relational database management system. The typical usage is simply:

```
DROP TABLE [table name]
```

For example, the command to drop a table named `employees` is:

```
DROP TABLE employees;
```
D. CREATE SEQUENCE statement

The CREATE SEQUENCE statement creates a sequence, which is a database object from which multiple users may generate unique integers. You can use sequences to automatically generate primary key values. The typical usage is:

```
CREATE SEQUENCE name [ INCREMENT [ BY ] increment ]
[ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
[ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]
```

If a schema name is given then the sequence is created in the specified schema. Otherwise, it is created in the current schema. The sequence name must be distinct from the name of any other sequence, table, index, or view in the same schema.

Parameters:
1. name: The name of the sequence to be created.
2. increment: The optional clause INCREMENT BY increment specifies which value is added to the current sequence value to create a new value. A positive value will make an ascending sequence, a negative one a descending sequence. The default value is 1.
3. minvalue: NO MINVALUE. The optional clause MINVALUE minvalue determines the minimum value a sequence can generate. If this clause is not supplied or NO MINVALUE is specified, then defaults will be used.
4. maxvalue: NO MAXVALUE. The optional clause MAXVALUE maxvalue determines the maximum value for the sequence. If this clause is not supplied or NO MAXVALUE is specified, then default values will be used.
5. start: The optional clause START WITH start allows the sequence to begin anywhere. The default starting value is minvalue for ascending sequences and maxvalue for descending ones.
6. cache: The optional clause CACHE cache specifies how many sequence numbers are to be pre-allocated and stored in memory for faster access. The minimum value is 1 (only one value can be generated at a time, i.e., no cache), and this is also the default.
7. CYCLE or NO CYCLE: The CYCLE option allows the sequence to wrap around when the maxvalue or minvalue has been reached by an ascending or descending sequence respectively. If the limit is reached, the next number generated will be the minvalue or maxvalue, respectively. If NO CYCLE is specified, any calls to nextval after the sequence has reached its maximum value will return an error. If neither CYCLE nor NO CYCLE are specified, NO CYCLE is the default.

The following example creates the sequence Cust_Seq. This sequence could be used to provide customer ID values when rows are added to the customers table.

```
CREATE SEQUENCE "project"."Cust_Seq"
    INCREMENT 1
    MINVALUE 0
    MAXVALUE 9223372036854775807
    START 0
    CACHE 1
    NOCYCLE;
```

E. DROP SEQUENCE statement

DROP SEQUENCE removes sequence number generators. A sequence can only be dropped by its owner or a superuser. The typical usage is simply:

```
DROP SEQUENCE [ IF EXISTS ] name [...] [ CASCADE | RESTRICT ]
```
Parameters:
1. IF EXISTS: Do not throw an error if the sequence does not exist. A notice is issued in this case.
2. name: The name of a sequence.
3. CASCADE or RESTRICT: CASCADE: Automatically drop objects that depend on the sequence.
   RESTRICT: Refuse to drop the sequence if any objects depend on it. This is the default.

The following statement drops the sequence `Cust_Seq`.
```
DROP SEQUENCE IF EXISTS "project"."Cust_Seq";
```

Data manipulation language

Data manipulation language (DML) comprises the SQL data change statements, which modify stored data but not the schema or database objects. In Deliverable 2, students are only required to turn in DML statements related to the INSERT statement. Students are encouraged to practice the DELETE and UPDATE statements by themselves. The syntax and usage of DELETE and UPDATE statements are covered in the tutorial slides.

We will discuss the requirement in more detail:

A. INSERT statement

INSERT statement adds one or more records to any single table in a relational database. Insert statements have the following form:
```
INSERT INTO table (column1, column2, column3 ... ) VALUES (value1, value2, value3 ... )
```

Example:
```
INSERT INTO phone_book (name, number) VALUES ('John Doe', '555-1212');
```

Data Access Interface

For each one of the entities in the miniworld, you must create all the appropriate maintenance functions depending on the allowable actions over the given entity. For instance, patients may be created, modified, and deleted; therefore, you must create the following functions: create_patient(...), update_patient(...), and delete_patient(...). Identifying the appropriate arguments to each of these functions is part of your design job. Every entity must have an ID. All IDs must be automatically drawn from a sequence, which you must create explicitly. All create_entity(...) functions must return the ID of the entity that was created.

We will cover the function implementation syntax in the tutorial slides. Here we just give an example of a function of the type create_entity(...).

For instance, the following function create_card will insert a row of values in the table `Card`:
CREATE OR REPLACE FUNCTION project.create_card(card_no character varying, validate_date date, sec_code character, ful_name character varying, def character varying, cust_id character varying, add_id character varying)
RETURNS character varying
LANGUAGE plpgsql
AS $function$
begin
  VALUES (Card_NO, Validate_Date, Sec_Code, Ful_Name, Def, Cust_ID, Add_ID);
  return Card_NO || ',' || Cust_ID;
end;
$function$

Similarly, you also need to implement the function update_card to update one particular attribute such as updated_card_number in the table Card given the arguments declared in the function signature such as old_card_number and customer_id. You will also implement the function delete_card to delete one row in the table Card.

In Deliverable 2, each entity should have a set of appropriate maintenance functions -- create_entity(...), update_entity(...), and delete_entity(...). Students should turn in an SQL file of maintenance functions for all entities based on their relational data model design submitted in Project 1.

Deliverables

The deliverable for Project 2 contains 3 files -- DDL, DML, and maintenance functions

1. Turn in a single SQL file containing all the data definition language (DDL) statements that when executed will create a new database with all the tables, views, and sequences in your design. For the DDL part, students are only required to turn in DDL statements related to CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE SEQUENCE, and DROP SEQUENCE.
   A. For CREATE TABLE, students should specify the following:
      • Each attribute should be defined with an adequate data type.
      • Constraints for attributes should be specified, e.g., a price should be nonnegative.
      • The primary key and foreign key(s) of each table should be specified.
      In the end, for each table, you should submit one statement.
   B. For ALTER TABLE, students should be able to meet the following requirements:
      • Add or remove one attribute in the existing table.
      • Alter the table name.
      • Update an attribute's data type.
      In the end, for each table, you should submit three statements. The attribute changed in each table is arbitrary and you only need to change the data type of one attribute per table.
   C. For DROP TABLE, the requirement is the following:
      • Drop an existing table from the database.
      In the end, for each table, you should submit one statement.
D. For CREATE SEQUENCE, the requirement is the following:
   - Create a sequence automatically.
   - Use the sequence generated as the primary key for each table.
In the end, for each table, you should submit one statement.
E. For DROP SEQUENCE, the requirement is the following:
   - Drop an existing sequence from each table.
In the end, for each table, you should submit one statement.

2. Include data manipulation language (DML) statements to populate the tables. In Deliverable 2, students are only required to turn in DML statements related to the INSERT statement.
   The requirement for the insert statement is the following:
   - Each table should have at least five records.
   - The insertion must not violate any constraints associated with the table the values are being inserted into.

3. At the same time, turn in an SQL file of maintenance functions: create_entity(...), update_entity(...), and delete_entity(...).
   The requirements for the maintenance functions are the following:
   - Each entity should have a corresponding create, update, delete functions.
   - Create_entity function inserts one row of values in the target table specified by the arguments.
   - Update_entity function updates a target attribute in the target table specified by the arguments.
   - delete_entity function deletes one target row of values in the target table specified by the arguments.
   - Create_entity function must return the newly inserted sequence value.

Submit a Zip file through the Moodle system containing three SQL files: DDL, DML, and maintenance functions.

Due date: October 13, 2016

Important Note

As deliverables are related, you will be given the opportunity to improve your previous work so that your future work does not carry over mistakes. In this regard, when a deliverable is due, you will be given the opportunity to turn in the previous deliverable if you revised it based on the feedback you received when it was graded. The next due deliverable will then be graded considering its consistency with the revised deliverable, not with the original one. Note that this is optional and that it always applies to the previous deliverable only, not to all previous deliverables. Additionally, once a deliverable is graded, the score will not be changed even if you improve it afterwards.
Submission

Submit your PDF / SQL file with the format Deliverable#_YourName_Fall2016 as the filename and upload it to Moodle.

For example, for the second deliverable, you should name your file Deliverable2_LebronJames_Fall2016.zip