**CPSC 50900 Database Systems Project**

**Initial Proposal**

The Idea here is to generate the policy ID and then determine which insurance policy is applicable for which person. This is basically showcasing the necessities of insurance terms and policies with the help of sql and python.

**Description Data**

The data will be divided into four columns namely Customer, Payment, Policy, Claims and Driver. These will be having fields of its own.

The Customer table will have a policy id which is the primary key, customer name, the date of birth, the address and the payment type.

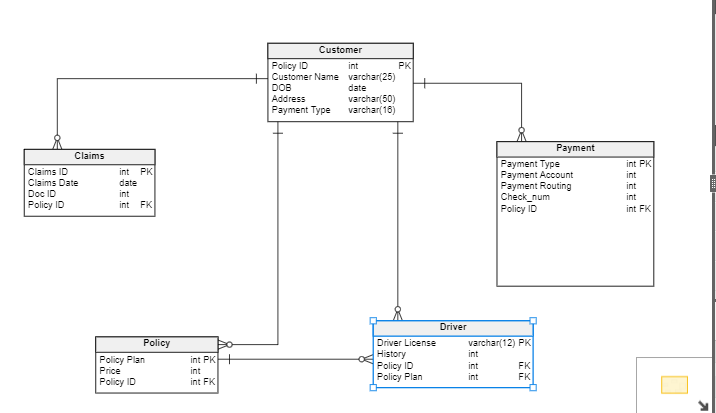
The Payment table has the policy id which is the foreign key, they payment type which is the primary key, the payment account, routing and checknum.

The Policy table will contain the policy id as the foreign key, the policy plan as primary key and the policy price.

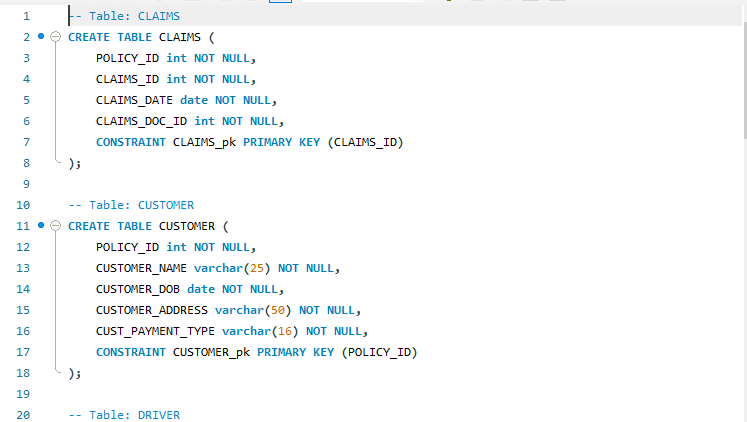
The Claims table has the Policy ID which is a foreign key, the Claims ID is the Primary Key, the Claims Date and the DOC\_ID are the other fields here

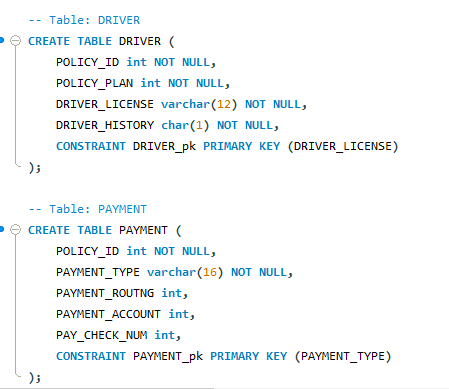
The last one is the Drivers table containing the Policy ID, Policy Plan, Drivers Licenses and History

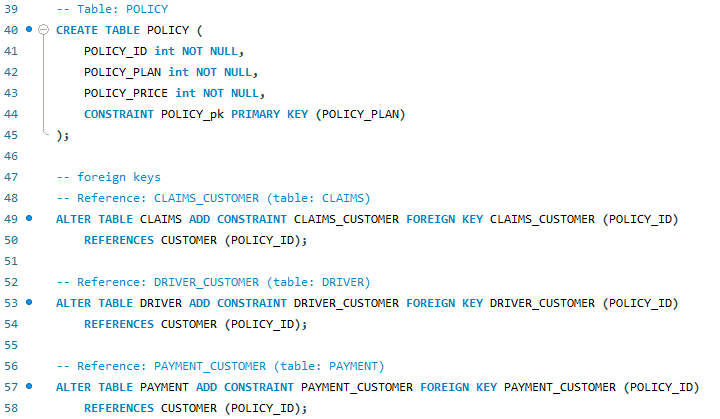
**Relational Database Design**

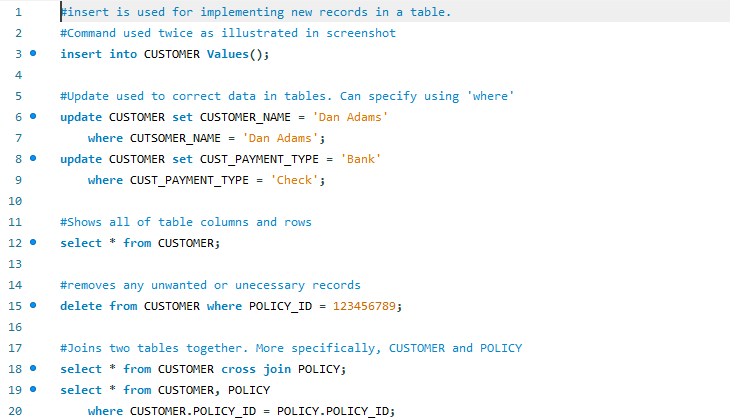


**Screenshots**

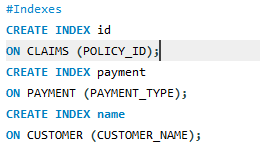






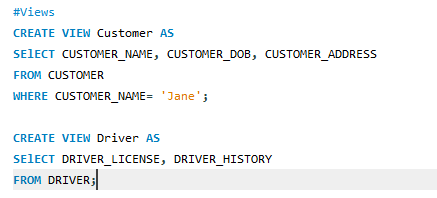


**Indexes**



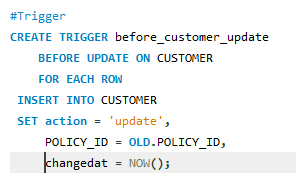
Three indexes from three different tables were chosen. The first index was used to keep a track on the policy id as policy id is the most essential factor to recognize the kind of policy type. The second index is to keep tabs on the payment and how it is done through cash or cheque. The last index is to keep tabs on the customer name so that the name of the customer could be easily mapped to the policy the customer opted for and the transaction type which was executed.

**Views**



We created two views the first one to list all details of customer for names starting with Jane and the second one to display all driver details.

Triggers



This is an example of trigger which will update the values of policy ID in the customer table accordingly. Policy ID is the most important field in the entire document to be mapped across the policies. Hence, if there is any change in the id it needs to be updated and reflected in the database first which is made possible by the trigger

**Transactions**

In a database, each transaction represents a single piece of work done. In many cases, it is made up of several steps. ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties of database transactions intended to guarantee validity even in the event of system crashes, power failures, and other errors.

Transactions are important because:

* They ensure absolute data integrity and safety
* Simplified Concurrency Control
* Future Proofing Database Needs

**Security**

Use the GRANT statement to give privileges to a specific user or role, or to all users, to perform actions on database objects. You can also use the GRANT statement to grant a role to a user, to PUBLIC, or to another role. In this case full privileges will be granted to the developer and the director while the employees will have the accessibility of inserting, updating and deleting data under supervision.

**Locking**

Locking is a mechanism to ensure data integrity while allowing maximum concurrent access to data. It is used to implement concurrency control when multiple users access table to manipulate its data at the same time.

1. Avoids deadlock conditions
2. Avoids clashes in capturing the resources

Types of locks:

1. Read Operations: Select
2. Write Operations: Insert, Update and Delete

Locking protect table when several users are accessing the same table. Locking is a concurrency control technique in oracle. It helps in data integrity while allowing maximum concurrency access to data. Oracle offers automatic locking whenever situation requires. This is called implicit locking.

**Backup**

The full backup, as it sounds, is a complete backup of a database. The full backup contains all the data in a database and can be used to do a complete restore of the database to the point-in-time that the full backup completed, less the uncommitted transaction in flight at that time. To remove the uncommitted transactions, the full backup also contains some transaction log information. The transaction log information in the backup is used, during the restore process, to roll back and remove any uncommitted transactions. By rolling backup uncommitted transactions, the database is left in a consistent state once the restore process completes.

When a full backup is created, it is written to an operating system file known as a media set. Discussion of media sets is outside the scope of this article. To create a full backup of a database, a DBA can use SQL Server Management Studio (SSMS) or can write a TSQL script. SSMS is typically used to take ad hoc backups of databases, whereas TSQL scripts are more often used to automate the backup process. Using TSQL scripts provides a repeatable backup process, that doesn’t require any other action besides submitting the script.

A differential back is a backup that copies only the data that has changed since the last full backup, also known as the delta changes. The amount of time it takes to perform a database backup is directly proportional to the amount of data a backup is required to write to the backup file. On a large database, it might take hours to back it up using a full backup. Whereas the time to take a differential backup is directly proportional to the amount of data that has changed since the last full backup. If your backup window is short, and the amount of data changed in a database since the last full backup is small then taking a differential backup will optimize the runtime and still provide a recovery point for your database.

In order to take a differential backup, you must first take a full backup. The full backup is the base for any follow-on differential backup. Each time a differential backup that is taken the delta changes (data changed since the full last backup) are written to the backup file. The amount of data and the time to take to perform a differential backup is directly proportional to the number of updates that have occurred since the last full backup. Therefore, over time a differential backup will get bigger and bigger the more a database is updated and will take longer and longer to run.

Because the differential backups grow in size over time, it is recommended that periodically a new base full backup be taken. Taking a new full backup will reset the number of delta changes back to zero, and thus help keep subsequent differential backups from being really large. A good timing for full and differential backups might be to take a full back up once a week, like on a Sunday, and take differential backups once a day the rest of a week.

One thing to keep in mind when using differential backups is that the restore process is more complicated. In order to restore to a differential backup recovery point, you first have to restore the base full back up before you restore the differential backup. This means you need to run two different restore processes, one for the full backup, and one for the differential backup.

**Policy**

