Report on Stored Procedure by Vaddanak Seng

The design incorporates stored functions/procedures that facilitates answering questions that requires multiple queries. The 4 stored functions are **getTier**, **getAllPolicies**, **getAllLicensePlates**, and **getPremium**. The function **getTier** is most useful. It takes two inputs: car-make and number of tiers. It returns the tier to which the passed-in car-make belongs based on its cost. The function employs 3 sql queries. The first sql determines the average cost for the given car-make.

```
SELECT AVG(insurance.cars.cost) INTO target_price
FROM insurance.cars
WHERE insurance.cars.make = car_make;
```

The second sql contains a subquery, and it determines the number of unique cost. The number of unique cost is used to divide the costs into some number of groups equal to the number of tiers, which is specified in the function argument. For example if there is 100 unique costs and the caller specifies 10 tiers, then each tier group contains 10 different costs that are in increasing order.

```
SELECT COUNT(grouped.cost) INTO tier_group_count
FROM
    (SELECT *
    FROM insurance.cars
    GROUP BY insurance.cars.cost) AS grouped;
```

The third query is used inside a while-loop. The number of iterations is equal to the number of tiers. The third query is also a nested query.

```
SELECT MIN(grouped.cost), MAX(grouped.cost) INTO min_price, max_price
FROM
    (SELECT *
     FROM insurance.cars
     GROUP BY insurance.cars.cost
     ORDER BY insurance.cars.cost LIMIT limit_whole) AS grouped
ORDER BY grouped.cost DESC
LIMIT limit_part;
```

In the first iteration, the nested query obtains a group of cost equalling to a limit of 1 x 10, ordered in ascending order. The relation contains the first 10 cost in increasing order. The outer query reverses the order to decreasing, obtains the first 10, and determines the min and max values. If the cost of the given car-make is within this range, then it belongs to the first tier and the function returns a 1. If it does not fall within this range, the next iteration represents the second tier. The nested query creates a new relation consisting of the first 2 x 10 unique costs in increasing order. The outer query reverses order so that the highest cost is first. It obtains the first 10 costs and determines the min and max values for this new relation. The cost of the given car-make is checked if it falls within the range. If it is, function returns 2. If not, the third iteration examines the third tier. The inner query creates a new relation from the first 3 x 10 unique costs in increasing order. The outer query reverses the order and obtains the first 10, which represents the 10 highest cost in this new relation. The min and max values are determined and cost of given car-make is checked. The iterations continue up to the number of specified tiers. The **getTier** function is the most sophisticated out of the 4. The **getAllPolicies** and **getAllLicensePlates** follow the same algorithm. This algorithm is used because MySQL does not have

a **record** type like PostGreSql that allows a variable to hold a collection of values. The **getPremium** uses a query that returns one record and uses other MySQL built-in functions.