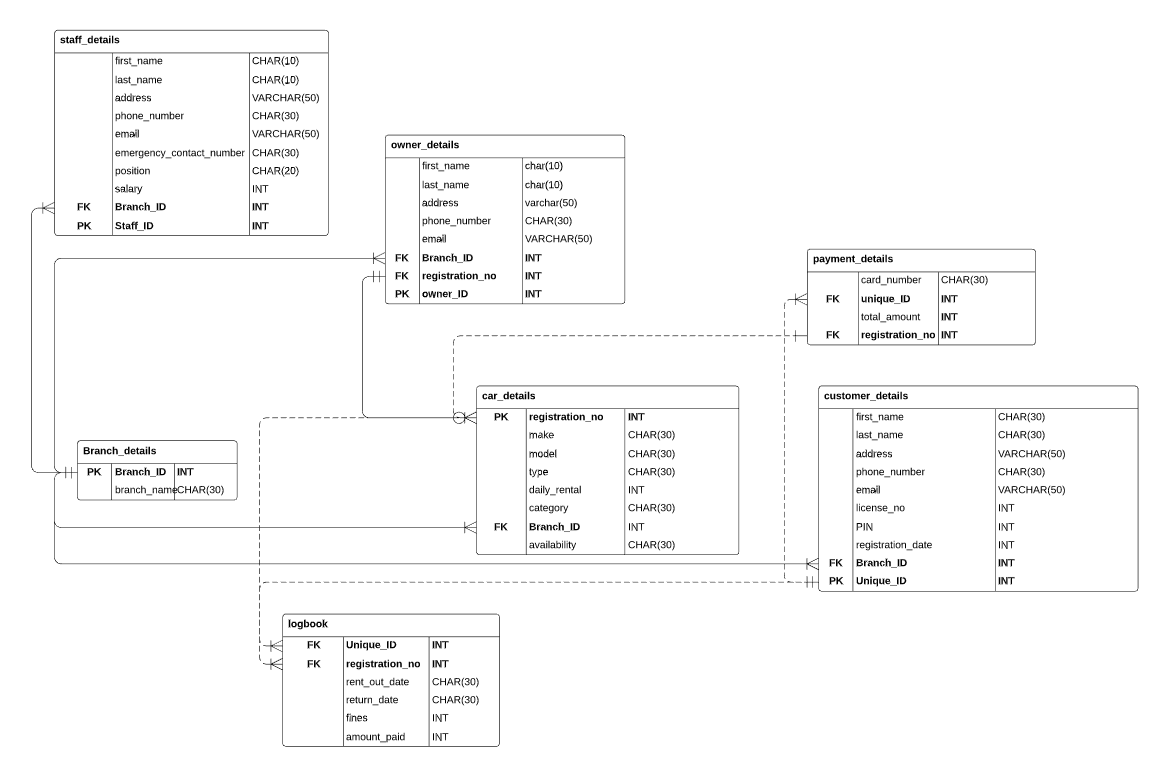
# SIT103- Data and Information Management

# Assessment Task 2- Project Documentation and DB

**Name: SATVIK SHARMA**

**Student ID – 218595095**

*Task A:*

**

*Task B:*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Table name | Columns | Data type | Character length | Structure | Required | PK/FK | references |
|  | Staff\_id | INT | 6 | NNNNNN | Y | PK |  |
| Staff\_Details | First\_name | CHAR | 10 | xxxxxxx | Y |  |  |
|  | Last\_name | CHAR | 10 | xxxxxxx | Y |  |  |
|  | Address | VARCHAR | 50 | xxxxxxx | Y |  |  |
|  | phone\_number | CHAR | 30 | +61xxx | Y |  |  |
|  | Email | VARCHAR | 50 | xxxxxxx | Y |  |  |
|  | Emergency\_contact\_number | CHAR | 30 | +61xxx | Y |  |  |
|  | Position | CHAR | 20 | xxxxxxx | Y |  |  |
|  | Salary | INT | 6 | NNNNNN | Y |  |  |
|  | Branch\_ID | INT | 6 | NNNNNN | Y | FK | Branch\_details(branch\_ID) |
| Branch\_details | Branch\_ID | INT | 6 | NNNNNN | Y | PK |  |
|  | Branch\_name | CHAR | 15 | xxxxxxx | Y |  |  |
|  | Owner\_ID | INT | 6 | NNNNNN | Y | PK |  |
| Owner\_details | First\_name | CHAR | 30 | xxxxxxx | Y |  |  |
|  | Last\_name | CHAR | 30 | xxxxxxx | Y |  |  |
|  | Address | VARCHAR | 50 | xxxxxxx | Y |  |  |
|  | Phone\_number | CHAR | 30 | +61xxx | Y |  |  |
|  | Email | VARCHAR | 50 | xxxxxxx | Y |  |  |
|  | Branch\_ID | INT | 6 | NNNNNN | Y | FK | Branch\_details(branch\_ID) |
|  | Registration\_no | INT | 6 | NNNNNN | Y | FK | Car\_details(registration\_no) |
| Car\_details | Registration\_no | INT | 6 | NNNNNN | Y | PK |  |
|  | Make | CHAR | 30 | xxxxxxx | Y |  |  |
|  | Model | CHAR | 30 | xxxxxxx | Y |  |  |
|  | Type | CHAR | 30 | Xxxxxxx | Y |  |  |
|  | Daily\_rental | INT | 6 | NNNNNN | Y |  |  |
|  | Category | CHAR | 30 | xxxxxxx | Y |  |  |
|  | Branch\_ID | INT | 6 | NNNNNN | Y | FK | Branch\_details(branch\_ID) |
|  | availabilty | CHAR | 30 | xxxxxxx | Y |  |  |
| Payment\_details | Card\_number | CHAR | 30 | xxxxxxx | Y |  |  |
|  | Unique\_ID | INT | 6 | NNNNNN | Y | FK | Customer\_details(unique\_ID) |
|  | Total\_amount | INT | 6 | NNNNNN | Y |  |  |
|  | Registration\_number | INT | 6 | NNNNNN | Y | FK |  |
| Logbook | unique\_ID | INT | 6 | NNNNNN | Y | FK | Customer\_details(unique\_ID) |
|  | Registration\_no | INT | 6 | NNNNNN | Y | FK | Car\_details(registration\_no) |
|  | Rent\_out\_date | CHAR | 30 | xxxxxxx | Y |  |  |
|  | Return\_date | CHAR | 30 | xxxxxxx | Y |  |  |
|  | Fines | INT | 6 | NNNNNN | Y |  |  |
|  | Amount\_paid | INT | 6 | NNNNNN | Y | PK |  |
| Customer\_details | First\_name | CHAR | 30 | xxxxxxxx | Y |  |  |
|  | Last\_name | CHAR | 30 | xxxxxxxx | Y |  |  |
|  | Address | VARCHAR | 50 | Xxxxxxxx | Y |  |  |
|  | Phone\_number | CHAR | 30 | +61xxxx | Y |  |  |
|  | Email | VARCHAR | 50 | xxxxxxxx | Y |  |  |
|  | License\_no | INT | 6 | NNNNNNN | Y |  |  |
|  | PIN | INT | 6 | NNNNNN | Y |  |  |
|  | Registration\_date | INT | 6 | NNNNNN | Y |  |  |
|  | Branch\_ID | INT | 6 | NNNNNN | Y | FK | Branch\_details(branch\_ID) |
|  | Unique\_ID | INT | 6 | NNNNNN | Y | PK |  |

*Task C:*

1. table 1 branch details

**

Table 2: staff details



Table 3: customer details



Table 4: car details



Table 5: logbook



Table 6: owner details

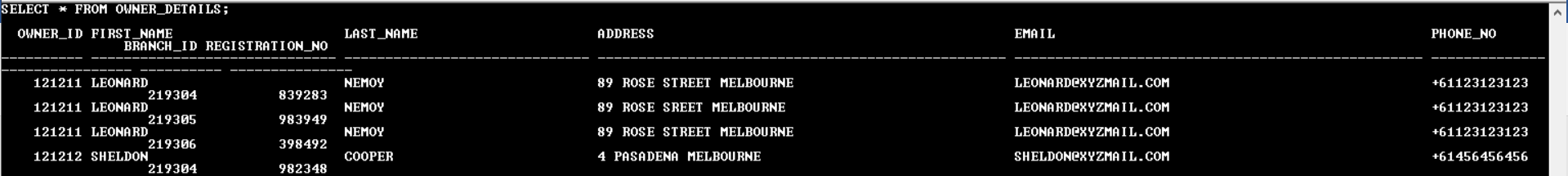
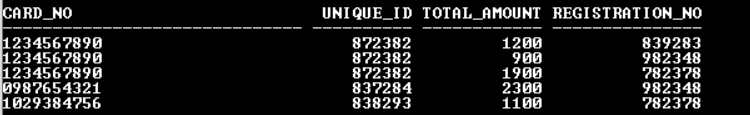
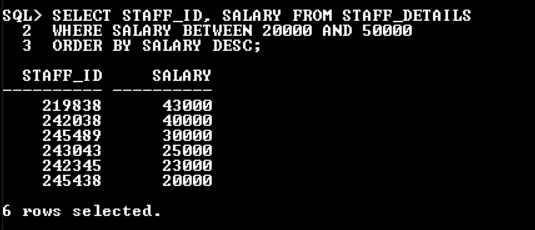


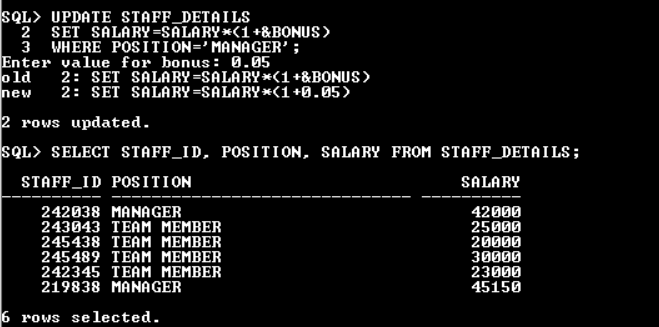
Table 7: payment details



1. sorting salary from highest to lowest in the staff table



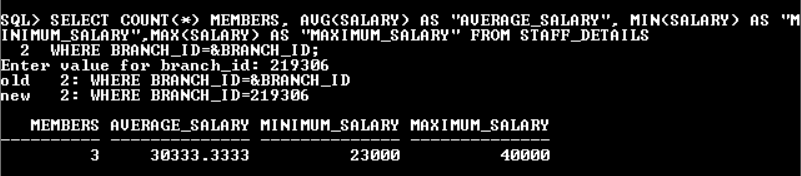
1. updating manager’s salary by 5%



1. displaying monthly salary by branch and sorting them



1. no of staff members from the branch id

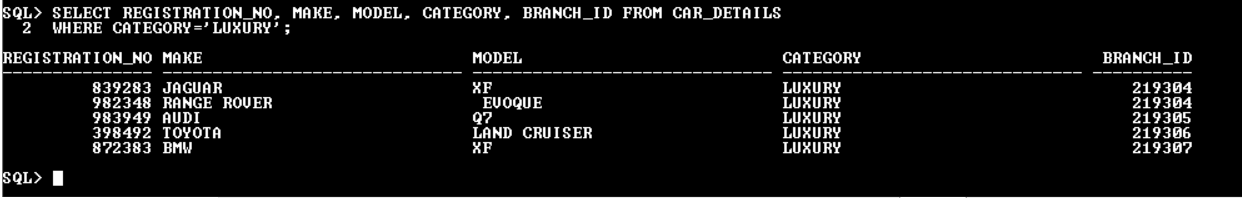


For this question, I added another row into the staff\_details table which might not correspond to the above screen shot given.

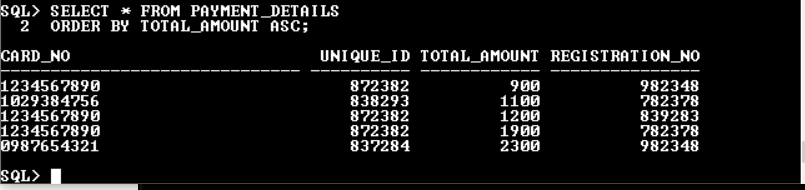
1. Sorting by last name



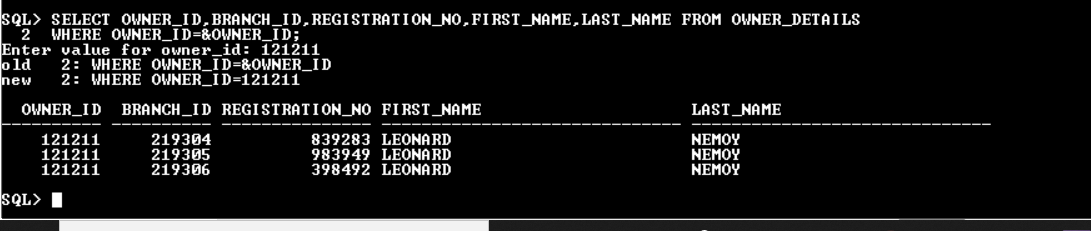
1. Luxury car data



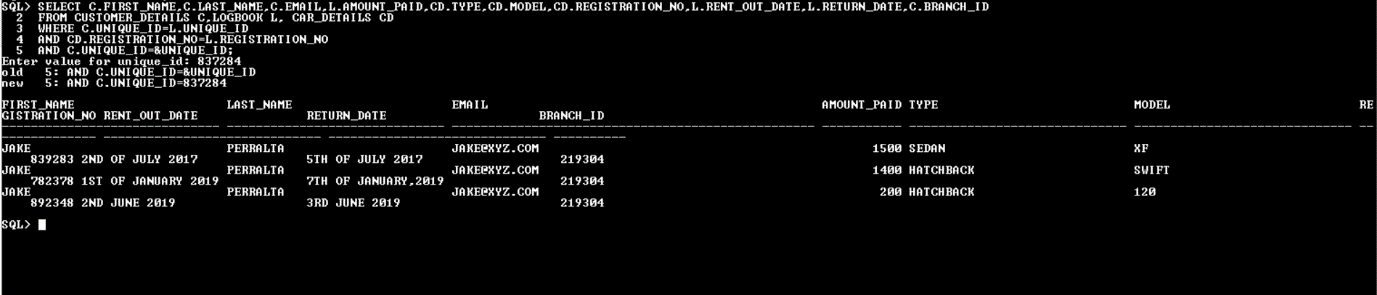
1. Ascending order of payment by customers



1. Cars owned in a particular area by a particular owner



1. Joining tables to declare all data



*Task D:*

1. Before understanding SQL injection, one should know about the injection attack. Injection attacks are special types of attacks, in which a different and non-trusted command is exercised and changes the output of the whole program. In SQL injection attack, the attacker uses dangerous SQL commands which control database sever behind a web application. With the use of SQL injection, the hackers can impersonate the hacked person, allow the access of unknown person in a server, alter databases and add new data. This attack is generally used on servers that use database such as MySQL, Oracle, SQL server or others. To use this type of attack, hacker generally finds vulnerable user inputs within a website and uses user input directly in an SQL query. The hacker creates and input content which is known as malicious payload. Malicious SQL commands are executed in database after the hacker sends this content. SQL is basically a query language that was created in order to manage the data stored in relational databases and to access, modify and delete data. The SQL injection looks like the command given below with an example from the above table’s payment\_details which is having unique\_ID and card\_details which might be used by the hacker to hack the bank details of the customer [1].[4]

U\_ID=request.POST[‘unique\_ID’]

CARD\_DETAILS=request.POST[‘card\_details’]

Sql=”SELECT ID FROM payment\_details WHERE unique\_ID=’”+U\_ID”’AND card\_details=’”+CARD\_DETAILS”’”

DATABASE.EXECUTE(Sal)

1. There are basically two ways to protect the server against an SQL injection. the parameterized queries in data-base-level code can be used. Sql supports binding parameters using colon with an index. The user integrates a solution into the program they are building. The first thing to look at is, if the data is mapped correctly or not. Then the input should be validated. By accepting the known good value. The developer must sanitize all input, not only web form inputs, like the sign-up forms. Moreover, potential malicious code elements such as single quotes must be removed. Visibility of the database errors on the production site must be turned off. Platform level security is also important to protect the database. User can deploy runtime protection techniques such as application level pulg ins without modifying the database to detact, prevent or mitigate sql injection. importantly, platform security is not a substitute for the insecure coding patterns that can allow the hackers to attack the database using SQL injections. A hardened network and application infrastructure combined with runtime monitoring and tuned prevention provide a formidable defence to thwart the sql injection.[2][3]

Bibliography:

[1] J. Clarke, *SQL injection attacks and defence*. Burlington, Mass.: Singers Pub., 2009.

[2] L. Shar and H. Tan, "Defeating SQL Injection", *Computer*, vol. 46, no. 3, pp. 69-77, 2013. Available: 10.1109/mc.2012.283.

[3] Y. Zhu and H. Liang, "The SQL Injection Vulnerability Detection of the Web Application", *Applied Mechanics and Materials*, vol. 198-199, pp. 1457-1461, 2012. Available: 10.4028/www.scientific.net/amm.198-199.1457.

[4] B. J. Santhosh Kumar and P. P. Anasarca, "Vulnerability detection and prevention of SQL injection", *International Journal of Engineering & Technology*, vol. 7, no. 231, p. 16, 2018. Available: 10.14419/ijet.v7i2.31.13388.