|  |  |
| --- | --- |
|  | **Big Data & Analytics – Final Project**  **Part One: Creating Relational Data Base**  **Case Study: Used Car Agency**  **Bachelor of Science in “Digital Business & Data Science”**  **Student: C\*\*\*\* A\*\*\***  **Born in Italy on 7th November 19\*\*** |

**Introduction**

A Manager of used care agency would like to create a system that can be used to facilitate his work in the agency and all his employees and agents. Currently he is using spreadsheet for collecting and saving all the data about the cars that he had in his agency. The idea is to create an online system to help him to search for a car, by various different values and inputs. Your role ist o design and implement the relational database for his agency (The implementation of the software is not required).

**Data**

At first, data is to be explored. We dispose of an excel file with all data to be managed by the system. Let’s have a look at this file.

The file presents itself like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **name** | **year** | **selling\_price** | **km\_driven** | **fuel** | **seller\_type** | **transmission** |
| Maruti Swift Dzire VDI | 2014 | 450000 | 145500 | Diesel | Individual | Manual |
| Skoda Rapid 1.5 TDI Ambition | 2014 | 370000 | 120000 | Diesel | Individual | Manual |
| Honda City 2017-2020 EXi | 2006 | 158000 | 140000 | Petrol | Individual | Manual |
| Hyundai i20 Sportz Diesel | 2010 | 225000 | 127000 | Diesel | Individual | Manual |
| Maruti Swift VXI BSIII | 2007 | 130000 | 120000 | Petrol | Individual | Manual |
| Hyundai Xcent 1.2 VTVT E Plus | 2017 | 440000 | 45000 | Petrol | Individual | Manual |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **owner** | **mileage** | **engine** | **max\_power** | **torque** | **seats** | **Dealer** |
| First Owner | 23.4 kmpl | 1248 CC | 74 bhp | 190Nm@ 2000rpm | 5 | David |
| Second Owner | 21.14 kmpl | 1498 CC | 103.52 bhp | 250Nm@ 1500-2500rpm | 5 |  |
| Third Owner | 17.7 kmpl | 1497 CC | 78 bhp | 12.7@ 2,700(kgm@ rpm) | 5 | Henry |
| First Owner | 23.0 kmpl | 1396 CC | 90 bhp | 22.4 kgm at 1750-2750rpm | 5 |  |
| First Owner | 16.1 kmpl | 1298 CC | 88.2 bhp | 11.5@ 4,500(kgm@ rpm) | 5 | Henry |
| First Owner | 20.14 kmpl | 1197 CC | 81.86 bhp | 113.75nm@ 4000rpm | 5 | Anny |

At a firts look, it is possible to notice that some data is neither perfectly organized nor cleaned enough to be inserted into a performing relational database. This would require some transformation activities. Further explanation is coming in next paragraphs. Threfore, in addition to the fact that there are thousands of records, a python script program is created to perform the job. In order to access this data from python, it is necessary to take a csv file out of the excell file. In order to do this, it is possible to use a built-in fuction of the Microsoft application.

Graphical user interface, application

Description automatically generated

Through the export function, we create a CSV file with the character ‘;’ as a delimiter between the columns. The first line of the file is made up of the column names.

The new csv file “Car\_DataSet.CVS” looks like this:

Diagram

Description automatically generated with medium confidence

We can now upload and visualize our data, to be better explored and eventually processed, in the python environment. As for commodity, due to the semplicity of the task and the possibility to fast run/visualize the code/results in their single parts: we use a jupyter notebook.

At first, we import and display on screen our target data.

Table

Description automatically generated with low confidence

These are the fields we have:

* *Name*

This column represents the „name of the car“, comprehensive of the car Brand, Model and general set up. The combination of these elements give a clear understanding of what type of car the object of the record is. These features mix is unique and relevant per each car model, and it is a clear despription of the item. Therefore, the field appears to be a good column to be kept into consideration and no cleaning/transformation actions are needed.

The attribute is a string data type. There are no empty values in any record.

* *Year*

This column represents the year of production of the car. In a certain way, it can be useful to make assumptions about the production years and calculate the car’s age. This data is properly filled in in the occidental calendar year format with 4 digits. The data type is “year”.

No action is required. There are no empty values in any record.

* *Selling price*

This column represents the selling price of the car by the owning agency. All data seems to be in a unsigned int format, but let’s define it as a decimal value to not exclude future possible changes. According to the business market, prices appear not to be neither euros nor dollars, otherwise the amount would be excessive. Apart from understanding the current currency, which anyway looks to be coherent among the records, there is no action to be taken. There are no empty values in any record, so no action is to be performed (example, estimating a proce with k-nearest-neighbour technique or others).

* *Km\_driven*

This column represents the distant usage status of the car, expressed in Kms. As there are neither null values not other issues, no action is required. The dataType is an unsigned int.

* *Fuel*

This column represents the fuel type of the car’s engine. It is a string value. It is possible to notice that there are a few repeating strings that are the possible values for the field. These values ensemble, we call it “enum” from now on. Further explanation will be provided later. There are no empty values in any record. At the moment, no action is required.

* *Seller\_type*

This column represents the seller type of the car’s deal. It is a string value. It is possible to notice that there are a few repeating strings that are the possible values for the field. These values ensemble, we call it “enum” from now on. Further explanation will be provided later. There are no empty values in any record. At the moment, no action is required.

* *Transmission*

This column represents the transmission type of the car’s engine. It is a string value. It is possible to notice that there are a few repeating strings that are the possible values for the field. These values ensemble, we call it “enum” from now on. Further explanation will be provided later. There are no empty values in any record. At the moment, no action is required.

* *Owner*

This column represents the owner type of the car’s status. It is a string value. It is possible to notice that there are a few repeating strings that are the possible values for the field. These values ensemble, we call it “enum” from now on. Further explanation will be provided later. There are no empty values in any record. At the moment, no action is required.

* *Mileage*

This column represents the mileage fuel consumption rate of the car’s engine. It is always expressed in Kmpl measure. The string ‘Kmpl’ is present in most of the records. We want the value to be a decimal. Thus, we apply some data transformation to the column. Moreover, the records present many empty values. Nevertheless, it is possible to notice that, even from sipmle excell filters, that if grouping by Name, all records have the same mileage value except for the ones with nulls. Thus, when car data will be inserted into DB, it will possible to ignore the null values not inserting a new car type with new (null) features, but rather using the values already defined for the same car model in other records.

Table

Description automatically generated

* *Engine*

This column represents the power of the car’s engine. It is always expressed in CC measure. The string ‘CC’ is present in most of the records. We want the value to be a integer. Thus, we apply some data transformation to the column. Moreover, the records present many empty values. Nevertheless, it is possible to notice that, even from sipmle excell filters, that if grouping by Name, all records have the same engine value except for the ones with nulls. Thus, when car data will be inserted into DB, it will possible to ignore the null values not inserting a new car type with new (null) features, but rather using the values already defined for the same car model in other records.

Text

Description automatically generated with medium confidence

* *Max\_power*

This column represents the maximum power of the car’s engine. It is always expressed in Bhp measure. The string ‘bhp’ is present in most of the records. We want the value to be a decimal. Thus, we apply some data transformation to the column. Moreover, the records present many empty values. Nevertheless, it is possible to notice that, even from sipmle excell filters, that if grouping by Name, all records have the same max\_power value except for the ones with nulls. Thus, when car data will be inserted into DB, it will possible to ignore the null values not inserting a new car type with new (null) features, but rather using the values already defined for the same car model in other records.

A picture containing table

Description automatically generated

* *Torque*

This column represents the torque features of the car’s engine. It is expressed in different measures. There are two main values:

* first one, in Kgm or Nm. We choose the Nm as final measure
* the second is a single value or a range, always in Rpm measure.

we split it into 3 columns:

* First, ToqueNm in Nm measure
* Second, TorqueRpmMin in Rpm measure
* Third, TorqueRpmMax in Rpm measure

Thus, we apply some data transformation to the column. Moreover, the records present many empty values. Nevertheless, it is possible to notice that, even from sipmle excell filters, that if grouping by Name, all records have the same torque value except for the ones with nulls. Thus, when car data will be inserted into DB, it will possible to ignore the null values not inserting a new car type with new (null) features, but rather using the values already defined for the same car model in other records.

Text, letter

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

* *Seats*

This column represents the number of seats set up on the car’s interior. It is always expressed with an unsigned integer. We do not need to apply any data transformation to the column. Hoewver, the records present many empty values. Nevertheless, it is possible to notice that, even from sipmle excell filters, that if grouping by Name, all records have the same Seats value except for the ones with nulls. Thus, when car data will be inserted into DB, it will possible to ignore the null values not inserting a new car type with new (null) features, but rather using the values already defined for the same car model in other records.

* *Dealer*

This column represents the agent of the car selling deal. It is a String with the name of the agent. There are some empty values, but that is acceptable, as not every deal is made through an Agent, or the Agent can be assigned in a second moment. No action Is required.

We know that the agents have a surname, too and that we can appel to them through a tile like “Mr.” or “Ms.”. We will add this further features to our DB.

**Data Base Design**

Now we know everything about our data: what they are, what their structure look like and what nulls and coherence issues there are. Let’s design a relational data base.

At first, Let’s define our entities.

As previously said, some “enums” have been identified. As for commodity of any software system that might operate with the DataBase, we create an entity and so a table per each of these possible “enums”. By this way, any software that migh need to know what are the possible options for these fields, can merely select the available values and use them dinamically as they are easily changed by the system administrator.

Then, we have the following entities:

|  |  |
| --- | --- |
| **TransmissionTypes** |  |
| Id | Unsigned int autoincrement |
| Description | VARCHAR(45) |

|  |  |
| --- | --- |
| **FuelTypes** |  |
| Id | Unsigned int autoincrement |
| Description | VARCHAR(45) |

|  |  |
| --- | --- |
| **OwnerTypes** |  |
| Id | Unsigned int autoincrement |
| Description | VARCHAR(45) |

|  |  |
| --- | --- |
| **SellerTypes** |  |
| Id | Unsigned int autoincrement |
| Description | VARCHAR(45) |

These 4 entities are related to the main 2 target topics of our data: the car description and the deal of the specific car sale to be manages. So, now, we create our entities for the car features, so that we have tables able to describe a specific car model with specific features, and the different deals that are on action for the agency comprehensive of dealers and specific car status.

A car is made of its brand/model description name, its interiors features, and its engine features.

Therefore:

|  |  |
| --- | --- |
| **Engines** |  |
| Id | Unsigned int autoincrement |
| TransmissionType | Unsigned INT |
| PowerCC | Unsigned INT |
| MaxPowerBhp | DECIMAL(10,2) |
| TorqueNm | Unsigned INT |
| TorqueRpmMin | Unsigned INT |
| TorqueRpmMax | Unsigned INT |
| FuelType | Unsigned INT |
| MileageKmpl | DECIMAL(10,2) |

|  |  |
| --- | --- |
| **Interiors** |  |
| Id | Unsigned int autoincrement |
| SeatsNumber | Unsigned INT |

|  |  |
| --- | --- |
| **Car** |  |
| Id | Unsigned int autoincrement |
| Name | Varchar(255) |

A car is made of its own features like the price, joined to its sales features, target car model, and the specific car status for the specific car instance to be sold in that business transaction.

Therefore:

|  |  |
| --- | --- |
| **Dealers** |  |
| Id | Unsigned int autoincrement |
| Appellation | Varchar(5) |
| Name | Varchar(45) |
| Surname | Varchar(45) |

The appellation can be multiple and vary, and extremely variable. Example “Mr.”, “Ms.”, “Dr.”, “Dr. med.”, “Herr.”, “Eng. Mr.”, … therefore, we do not create an “enum”, but we leave the field to be open to usage.

|  |  |
| --- | --- |
| **CarStatus** |  |
| Id | Unsigned int autoincrement |
| OwnerType | Unsigned INT |
| KmsDriven | Unsigned INT |
| Year | Year |

|  |  |
| --- | --- |
| **Deal** |  |
| Id | Unsigned int autoincrement |
| Price | DECIMAL(10,2) |
| KmsDriven | Unsigned INT |
| Year | Year |

So here we have all our entities that cover all the data present in our original file, ready to be extended to new future features, normalized per logical business topic.

Now, it’s the moment to link these entities to create the relations.

At first, we can easily link the “enum” entities to the table entries that reflects them. These are N:1 association: more entities will refer to the same enum record. For example: N engines will refer to a single TransmissionType.

So… here they are

Engine – N:1 – TransmissionTypes

Engine – N:1 – FuelTypes

CarStatus – N:1 – OwnerTypes

Deals – N:1 – SellerTypes

As for this reason, we attach a foreign key on the N table referencing the Id of the 1 table.

Afterward, we need to link the other entities according to the previously defined business logic.

Deals – N:1 – Dealers //a deal refers to a dealer

Deals – N:1 – CarStatus //a deals refer to a specific car instance with a specific car status

As for this reason, we attach a foreign key on the N table referencing the Id of the 1 table.

Cars – 1:1 – Interios //any car model has its own specific features

Cars – 1:1 – Engines //any car model has its own engine features

As for this reason, we attach a foreign key on the Cars table referencing the Id of the linked table.

Diagram

Description automatically generated

**Data Base Creation**

After the ER diagram design, it’s time to generate our SQL statements necessary to create the Data Base through SQL DDL. Please, refer to the attached SQL script file to view the SQL code.

It is enough to run the script as it is to create the whole database structure, with tables, keys and indexes defined, without any errors.

**Data Import**

In order to manage flexibly our data import process, a python script has been created. This program can connect to the MySql DBMS and execute the queries necessary to complete the import process. The program is continued in the same jupyter notebook used in the previous sections.

Let’s get started with data we know is fixed data. Enums and dealers.

So, at first, let’s create a structure able to tell us what all these possible values are.

Text

Description automatically generated

Then, let’s create a function able to generate the insert of these values into our tables, considering that the table structure is always the same: Id + Description.

In the whole project, string escapes are not managed: data doesn’t look like having these issues.

Text

Description automatically generated

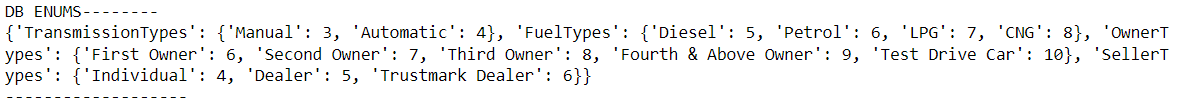
In the function, a structure DBenums is created, too. By this way, on our program we have somewhere to save the Ids on DB auto generated by the auto\_increment option during the insert.

Text

Description automatically generated

The structure has been populated with the function result. Each function run generates a DBenum that is inserted on the dictionary DBenums containing all our enums.

If we print DBenums, here’s the result:



Now we have our data on the DB. Let’s have a look as an example.

The query “select \* from fueltypes” results as follows:

Graphical user interface, table

Description automatically generated

The Ids and descriptions are exactly coherent with the structure printed by the python program.

Now, the other “fixed” data is concerning the dealers. Let’s have a look at which dealers there are in our data:

Text

Description automatically generated with medium confidence

Perfect! They are exactly the three dealers indexed in the assignment paper. Let’s create a function able to make the insert to DB.

Text

Description automatically generated

And let’s call the function to actually insert the rows on the table.

A picture containing calendar

Description automatically generated

A data structure where to save the dealers Ids is created so that we know during the import process what Id has been generated by the auto\_increment option of out MySql query run.

During the record insert, we will use this data to link the dealers to the other entities.

In order to check the result, we can run on our DB the query “select \* from dealers”, that results as follows:

Table

Description automatically generated

The Ids are coherent with the ones saved by the python program.

Now it’s time to create the import process of our records. Let’s take into account that we have to insert the cars with their features, only once per Car Model, while managing the null issues previously discussed. Moreover, each record has to generate a deal with the specific data about the business transaction.

In order to do this, let’s create a class Record, able to manage a record.

Graphical user interface, text

Description automatically generated

The class accepts as an input to the constructor all values present in a record. Each value is copied to a local public variable, accessible from the external. Moreover, the constructor initializes the protected values concerning the Ids of each table to be managed by the record. Finally, the DB connection objects are initialized.

Graphical user interface, text, application, chat or text message

Description automatically generated

The class disposes of a public method Insert() whose purpose is the one of generating the insert process of the record, starting from the properties of the class that are supposed to be filled in. There is no check on the nullability of the properties as this code is not supposed to be used otherwise outside this little controlled project.

Graphical user interface, text, application

Description automatically generated

The process is quite simple.

* If there is no open connection, the MySql connection is initialized and a transaction is started.
* A query is made on DB, in order to understand whether a car with the same Brand/Model name already exists.
* If the car does not exist, it means it is the first record insert trial with that car Name.
  + In this case, car details are inserted into the DB
    - Engine details are inserted on “Engines” table, and generated Id is saved
    - Interior details are inserted on “Interiors” table, and generated Id is saved
    - Car details are inserted on “Cars” table, and generated Id is saved

Please note that these operations are in a specific order: both engines and interiors must be generated before Car, as they have to provide their Ids as Car foreign keys

* + In case the Car presence is already found on the DB, it means this is not the first Insert call with this car Brand/Model name. Therefore, due to the NULLs issue, we try update the record. If the record currently on DB has NULLs values and the current record has the correct values, then the values are overwritten with an update. In the contrary case, in which the current record has NULLs values and the DB has the correct values, then the current db-persisted values are the preferred one. The Update functions are not optimized because the “Update” sql query is executed any time the function is called, despite the record is updated with the same values it already owns. This is due to comfortability of the program creation. As this will not affect in an annoying way the performances of the import process, let it be like this.
* Once the car is inserted, it is possible to create a deal referring to that car.
  + CarStatus details are inserted on “CarStatus” table, and generated Id is saved
  + Deal details are inserted on “Deals” table, and generated Id is saved

Please notice the order of the operations: CarStatus has o be created before, so that the Deal can reference it by foreign key. The Deal can be created only after the car record creation, and then it references it by foreign key. Actually, the saving of the deal is not necessary for the import process. However, the obtained Id is the transaction Id.

The process is under transaction. Begin/Commit/Rollback transaction are managed under the Try/Ecept/Finally block set, with connection close in any case, and commit for success rather than rollback for insuccess.

Once the class is properly defined, it is time to run the code:

Text, letter

Description automatically generated

The produced output will look as follows:

Text

Description automatically generated

In order to view the implementation details of the class methods, please open the attached jupyter notebook (attached both as jupyter extension and HTML view version).

It is possible to open the jupyter notebook, run each cell in the proper order from the top to the bottom of the notebook, and the code will be executed smoothly without any error, as long as the csv file is uploaded, and the database is both created and reachable according to the set ip/userpassword/name parameters. Please notice: MySql connector is not supported on python 3.9, thus, in order to run the code it is necessary to downgrade the python version to 3.8 and install the MySqlConnector Package. Due to the usage of DataFrames, Pandas and NumPy installations are required, too.

In order to have a look at the results, we can launch some simple queries in our DB and view what it follows:

* Select \* from deals

Table

Description automatically generated

* select \* from carstatus

Table

Description automatically generated

* select \* from cars

Table

Description automatically generated

* select \* from engines

Table

Description automatically generated

* select \* from interios

Graphical user interface, table

Description automatically generated

**SQL Questions**

Find out which car model are more sold.

Graphical user interface

Description automatically generated with low confidence

Which dealer sells more cars, and his total sales.

Graphical user interface, text, application

Description automatically generated

What is the average price of each car model?

Graphical user interface, text, application

Description automatically generated

What is the newest car and the oldest car?

Graphical user interface, text, application

Description automatically generated

Which is the best car related to price, and low mileage?

Graphical user interface, text, application

Description automatically generated

What is the total revenue from Individual selling?

Graphical user interface, text, application

Description automatically generated

What is the famous car type for each agent?

Graphical user interface, text, application, email

Description automatically generated

Student: C\*\*\*\* A\*\*\*

Born in Italy on 7th November 19\*\*

Bachelor of Science in “Digital Business & Data Science”

2nd semester.