

**Prabhdeep Singh**

Singp2@farmingdale.edu

BCS 360

Project Final Report

Professor: **David Gerstl**

11/30/2017

Table of Contents

1. [Getting Started 3](#_Toc219623644)

1.1 NYC TLC : [Organization 3](#_Toc219623645)

1.2 Street Hail Livery Vehicle (SHL) or Green Cab 3

1.3 [Finding the data 3](#_Toc219623646)

1.4 [Purpose of data 3](#_Toc219623647)

2. [Detail Information about data 4](#_Toc219623648)

2.1 [Data format 4](#_Toc219623649)

2.2 [Data fields 4](#_Toc219623649)

3. [[Tutorial 5](#_Toc219623648)](#_Toc219623657)

3.1 [Downloading the data 5](#_Toc219623658)

3.2 [Creating Database schema 5](#_Toc219623659)

3.3 [Loading data into shl\_orig database 7](#_Toc219623660)

3.4 [Checking imported data 8](#_Toc219623651)

4. [[SQL Files](#_Toc219623648)](#_Toc219623657) 9

4.1 [Create table 9](#_Toc219623658)

4.2 [Load data 9](#_Toc219623658)

5. [[Population of Normalized Tables](#_Toc219623648)](#_Toc219623657) 10

5.1 [Database normalization, its importance & Gerstl’s Golden Quote 10](#_Toc219623658)

5.2 [Recognizing functional dependencies 11](#_Toc219623658)

5.3 [Normalized Tables 11](#_Toc219623658)

5.4 [Creating populated, normalized tables 12-14](#_Toc219623658)

6. [[Constraining the Normalized Tables](#_Toc219623648)](#_Toc219623657) 15

6.1 [Reviewing the implied constraints 15](#_Toc219623658)

6.2 [Constraining Bases table 16-18](#_Toc219623658)

6.3 [Constraining Vehicles table 18-20](#_Toc219623658)

6.4 [Constraining Permit table 21-24](#_Toc219623658)

# 1. Getting Started

**1.1 Organization**

The New York City Taxi and Limousine Commission (NYC TLC) is an agency of the New York City government that regulates the medallion taxis and for-hire vehicle industries, including app-based companies. This includes medallion (yellow) taxicabs, green or Boro taxicabs, black cars (including both traditional and app-based services), community-based livery cars, commuter vans, paratransit vehicles (ambulettes), and some luxury limousines.

**1.2 Street Hail Livery Vehicle (SHL)**

In New York City, a Street Hail Livery Vehicle (SHL) is sometimes referred to as a Green Taxi or Outer Boro Taxi. These vehicles have a distinctive apple green color and can be seen roaming the streets of NYC, especially outside of Manhattan. So what exactly is a Green Taxi? Historically only yellow taxis had the right to pick up passengers off the street, but spent almost 95% of the time serving Manhattan customers below 96th street. As a result of the poor yellow taxi coverage in the outer boros (Queens, Bronx, Brooklyn, Staten Island), Governor Cuomo signed into law the Five Borough Taxi Plan in 2011.

**1.3 Data Source**

The data pertaining to all SHL permits and licensed vehicles is found on publicly available NYC Open Data web portal. The data source link is <https://data.cityofnewyork.us/Transportation/Street-Hail-Livery-SHL-Permits/yhuu-4pt3/data>

**1.4 Purpose**

The purpose of SHL database is as follows

* To keep record of all issues permits and permit related information such as permit owner, expiration date.
* To keep record of all actively licensed SHL vehicles corresponding the SHL permit such as vehicle VIN, license plate number.
* To keep record of all active SHL base stations and related information such as base name, telephone, address, affiliated SHL vehicles.

# 2. Detail Information

**2.1 Data Format**

The open data website hosts the data and displays the data in tabular form using html. The website also offers the functionality to obtain a complete export of the data in various formats including csv, json, rss, xml and rdf. For the purpose of this project, the data is downloaded in csv format.

**2.2 Data Fields**

The data is organized in 20 fields which are listed below along with a brief description of the data field. The data fields are color coded according to the entities they are relevant to and for organization.

Color Codes are used to identify and mark different entities in source data.

Green: Permit Info (These fields relate to a specific SHL Permit)

Blue: Vehicle Info (These fields relate to a specific vehicle licensed under SHL Permit).

Brown: Base Info (These fields relate to a SHL base dispatch station that a particular SHL licensed vehicle is affiliated with)

Red: Dataset Info (This data is imported but does not link to any entity and is of no use. We are looking at a snapshot of the dataset at a particular time and will not be taking into account the changes that are made to source data over time and the update date/time fields. Therefore, the downloaded data is treated as final).

|  |  |  |
| --- | --- | --- |
| Field Number | Field Name | Field Description |
| 1 | Active | YES/NO |
| 2 | Permit License Number | 5 Character field for permit license |
| 3 | Name | Name of permit owner |
| 4 | Expiration Date | Expiration date of permit |
| 5 | Vehicle License Number | License number given to vehicle |
| 6 | DMV License Plate Number | DMV plate number of licensed vehicle |
| 7 | Vehicle VIN Number | VIN number of vehicle |
| 8 | Vehicle Type | Type of vehicle – WAV accessible or not |
| 9 | Certification Date | Date permit was certified |
| 10 | Hack Up Date | Date vehicle was hacked up with permit |
| 11 | Vehicle Year | Model year of vehicle |
| 12 | Base Number | Unique ID of SHL base station |
| 13 | Base Name | Name of SHL base station |
| 14 | Base Telephone Number | Telephone number of SHL base station |
| 15 | Base Website | Website of SHL base station |
| 16 | Base Address | Address location of SHL base station |
| 17 | Reason | Code of permit status : G for active ; B &C etc. for inactive/suspended |
| 18 | Suspension Date | Date permit was suspended |
| 19 | Last Date Updated | Last date data was updated |
| 20 | Last Time Updated | Last time data was updated |

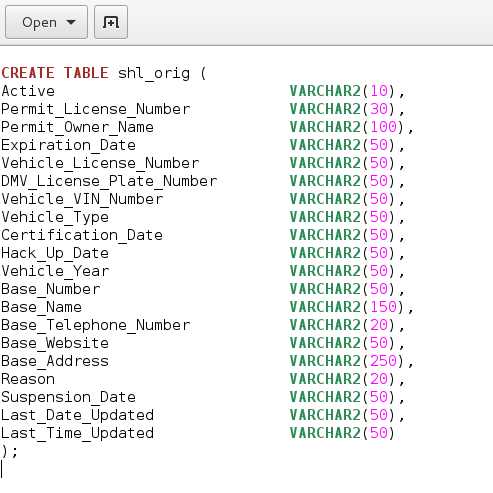
This color coding can be seen as making buckets to hold immediately relevant data together and figuring out relation between two colors coded buckets. Further into the project, these buckets with help us identify different database table entities and relation between these table entities. This will make easier to spot the transitive dependencies in each entity and assign primary keys to the tables.

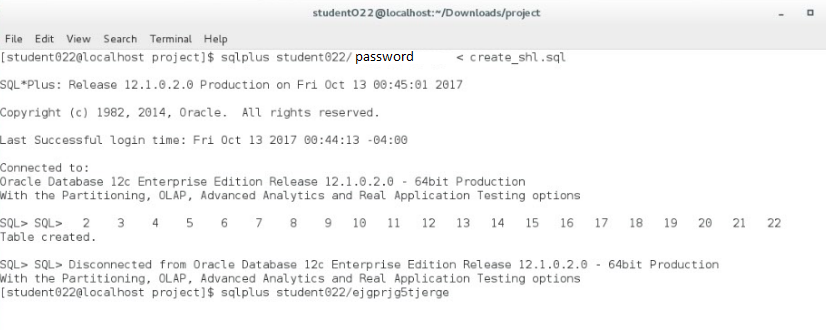
Currently, all the data fields are held in one single table in source data and will be imported into database as a single table also.

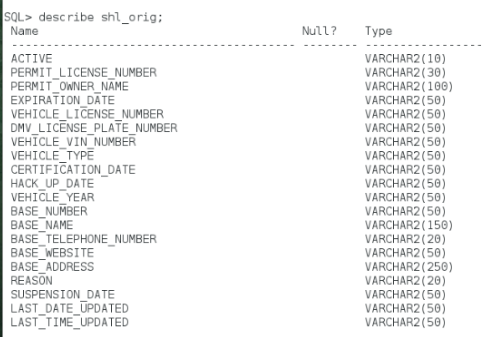
# 3. Tutorial for project

**3.1 Downloading the data**

* 1. Browse <https://data.cityofnewyork.us/Transportation/Street-Hail-Livery-SHL-Permits/yhuu-4pt3/data>.
  2. Click on Export button and select CSV format.
  3. The file will be provided for download by the website, save the file in project folder. If the folder is not created, create it first.
  4. Open the downloaded file and make sure you can see about 8300 rows of data. If you only see few rows, data might be corrupted, in which case you should delete the file and repeat steps a-d.
  5. **Creating Database schema**
  6. After you have downloaded, open the csv file and take a look at the data format and access the length of fields. (The name and description of data fields is provided earlier).
  7. Our primary focus at this step is to create a database schema which can hold the data we have in csv format. Currently, we are not focusing on getting the field data type completely accurate; rather we aim to load complete data in the database first. Therefore we will be importing complex data types such as dates as string data for now (as VARCHAR).
  8. Open gedit to create a sql query which will create a table named shl\_orig with field data type specified as in following screenshot. Save and close the sql file.



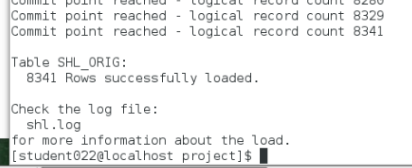
* 1. Open the folder where the file is saved, right click and open terminal.
  2. Log onto sqlplus with your credentials and directing the sql query into the database as *sqlplus username/password < create\_table\_query.sql .* Sample result is shown below.
  3. Running this command will create the table in the database. To confirm the database structure, run *sql describe shl\_orig;* . This command will display the created table with field and their corresponding data types. Sample result is shown below.



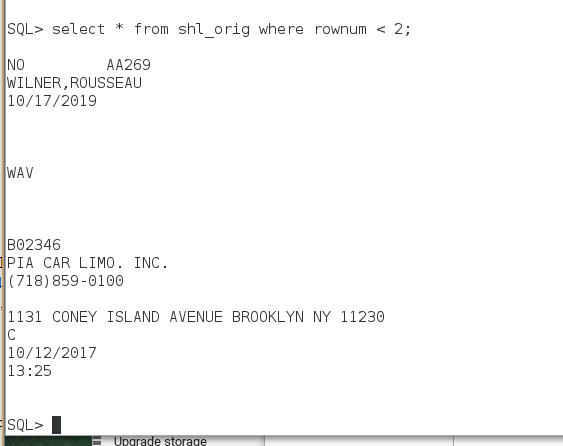
* 1. Once your database is created, move to next step.

**3.3 Loading data into shl\_orig database**: This step will include loading the downloaded cvs data into shl\_orig table using sqlldr tool. The load instructions will be supplied in shl.ctl control file.

* 1. Open gedit and create a control file with extension .ctl
  2. Provide the INFILE with file name (in this case shl.csv).
  3. Provide the field terminator, options (skip=1) since we do not want to import the table header row.
  4. Provide the ordered field names from the shl\_orig table ordered in the same order the corresponding fields appear in the csv file.
  5. The shl.ctl file should now look like below.
  6. Open terminal in same folder as the shl.ctl file is located.
  7. Open the command to load the data in using sqlldr. The command is in the format - *sqlldr username/password CONTROL=shl.ctl LOG=shl.log.*
  8. This command will use the sqlldr to load data in according to specifications in shl.ctl file and log the results in shl.log file.
  9. The command will load all the data in and display total number of successfully loaded rows. More specific details can be checked by opening the shl.log file. Sample result is shown below.



**3.4 Checking imported data:** This step will consist ofrunning a select statement to verify the accuracy of loaded data.

* 1. Log into terminal and sqlplus with your credentials.
  2. Execute a select statement which displays data in the shl\_orig table. Sample query is : *sql select \* from shl\_orig;*
  3. You can add a where clause after from to specify a number of rows you wish to see by specifying rownum < number, where number is the maximum rows you wish to display. The complete query will look like : *sql select \* from shl\_orig where rownum < 20;* 

# 4. SQL Files

* 1. Create table: The sql for create table is provided below for reference.

CREATE TABLE shl\_orig (

Active VARCHAR2(10),

Permit\_License\_Number VARCHAR2(30),

Permit\_Owner\_Name VARCHAR2(100),

Expiration\_Date VARCHAR2(50),

Vehicle\_License\_Number VARCHAR2(50),

DMV\_License\_Plate\_Number VARCHAR2(50),

Vehicle\_VIN\_Number VARCHAR2(50),

Vehicle\_Type VARCHAR2(50),

Certification\_Date VARCHAR2(50),

Hack\_Up\_Date VARCHAR2(50),

Vehicle\_Year VARCHAR2(50),

Base\_Number VARCHAR2(50),

Base\_Name VARCHAR2(150),

Base\_Telephone\_Number VARCHAR2(20),

Base\_Website VARCHAR2(50),

Base\_Address VARCHAR2(250),

Reason VARCHAR2(20),

Suspension\_Date VARCHAR2(50),

Last\_Date\_Updated VARCHAR2(50),

Last\_Time\_Updated VARCHAR2(50)

);

4.2 Load Data: The specification of the control file to load the data in using sqlldr is provided below for reference.

OPTIONS (skip=1)

LOAD DATA

INFILE shl.csv

INTO TABLE shl\_orig

REPLACE

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'

TRAILING NULLCOLS

(Active,Permit\_License\_Number,Permit\_Owner\_Name,Expiration\_Date,Vehicle\_License\_Number,DMV\_License\_Plate\_Number,Vehicle\_VIN\_Number,Vehicle\_Type,Certification\_Date,Hack\_Up\_Date,Vehicle\_Year,Base\_Number,Base\_Name,Base\_Telephone\_Number,Base\_Website,Base\_Address,Reason,Suspension\_Date,Last\_Date\_Updated,Last\_Time\_Updated)`

# 5. Population of Normalized Tables

**5.1 Database normalization**

**Database normalization** is the process of organizing data with common theme into tables. Data normalization is important step in implementing a database schema because it helps to reduce and even eliminate data redundancy. By properly normalizing a database schema, we can save storage space and ensure the consistency of data.

An important question we need to ask is how much normalization is needed for the required task ? . That is a difficult question since it varies greatly with the source data and the individual entities that come together to create the data.

**Gerstl’s Golden Quote**

Here’s an quote from professor **David Gerstl**.

*“The answer to any difficult question is ‘It depends.’ ”.*

That is true in this case as well. The level of normalization depends on what our aim is in terms of organizing and using the data. The normalization levels are represented by normal forms and the most commonly used normal forms are 1NF (First normal form) and similarly 2NF, 3NF and BCNF. Higher normalization forms do exist but they are not commonly used for data with low to average complexity.

**5.2 Recognizing functional dependencies**

A **Functional dependency** is a relationship that exists when one attribute uniquely determines another attribute. For example : If A is a relation with attributes X and Y, a functional dependency between the attributes is represented as X->Y, which specifies Y is functionally dependent on X.

The data has to be carefully reviewed to identify the functional dependencies and it becomes easier to see the themes in the table by analyzing functional dependencies. In fact, the functional dependencies can even aid the database implementer in confirming if the data fields belong in a chosen theme or not.

In *section 2.2 (Data Fields)*, we listed all the fields in raw import table and color coded them according to the relevance to entities. In that section, we actually recognized three high level themes and used them to mark three different entities, namely Permit Info, Vehicle Info and Base Info. In this step, we will represent these themes in terms of functional dependencies.

**Permit\_License\_Number** -> Active, Permit\_License\_Number , Permit\_Owner\_Name ,

Expiration\_Date , Vehicle\_License\_Number , Suspension\_Date , Reason

**Vehicle\_VIN\_Number** -> DMV\_License\_Plate\_Number, Vehicle\_Type, Certification\_Date, Hack\_Up\_Date, Vehicle\_Year

**Base\_Number** -> Base\_Name, Base\_Telephone\_Number, Base\_Website, Base\_Address

**5.3 Normalized Tables**

The functional dependencies in previous step are listed with determinants on the left and the dependent set on the right. We can analyze these dependencies and conclude that we will need three tables to hold the data represented by these themes. The determinants will form the primary keys of the normalized tables. The names and data fields of proposed tables are listed below.

1. **Permit** ( **Permit\_License\_Number** , Active, Permit\_License\_Number , Permit\_Owner\_Name , Expiration\_Date , Vehicle\_License\_Number , Suspension\_Date , Reason , *Vehicle\_VIN\_Number*)

Primary Key: Permit\_License\_Number

Foreign Key: Vehicle\_VIN\_Number - References **Vehicle (Vehicle\_VIN\_Number)**

1. **Vehicle** ( **Vehicle\_VIN\_Number** , DMV\_License\_Plate\_Number, Vehicle\_Type, Certification\_Date, Hack\_Up\_Date, Vehicle\_Year , Base\_Number)

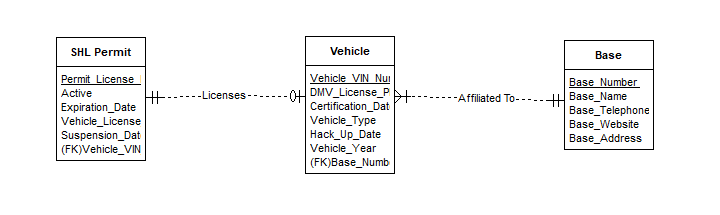
Primary Key: **Vehicle\_VIN\_Number**

Foreign Key: Base\_Number – References Base (Base\_Number)

1. **Base ( Base\_Number** -> Base\_Name , Base\_Telephone\_Number , Base\_Website , Base\_Address **)**

**Primary Key: Base\_Number**

**An Entity Relation Diagram of the proposed database schema is shown below. This diagram lists the three table with data fields and relationship between the tables. The diagram also includes marking for primary keys as underlined and foreign keys with keyword (FK).**



**5.4 Creating populated & normalized tables**

We currently have the raw data imported into table shl\_orig from section 4 and the skeleton of required tables from previous step. Therefore, we can now create the required tables.

Since we already have the raw data table, we are not going to use create table query to create the tables and then populate the tables with data, Instead we are going to use create table as select. We will also be using Distinct with Select statement to make sure we do not get duplicated records.

The create table as select statement selects the specified columns from source table and imports the corresponding data from one table into the specified new table. The syntax of a sample query is listed below.

CREATE TABLE new\_tbl AS (SELECT Distinct \* FROM orig\_tbl);

**Note 1**: We will not be creating constraints on these tables until the next section. Therefore, primary and foreign keys are omitted. However, the fields which will act as foreign keys have been included in permit and vehicles table. Explanation of constraints and constraining fields will be provided in next section as well. In table scheme and sql create table as query, we have highlighted primary keys as underlined and foreign keys as italic).

**Note 2:** We arenot includingLast\_Date\_Updated & Last\_Time\_Updated fields in any table because we are taking a snapshot of the source data once and after that are treating the data as final for our project.

***5.4.1 Creating Permit table:***

The scheme of Permit table is:

Permit ( Permit\_License\_Number , Active, Permit\_License\_Number , Permit\_Owner\_Name , Expiration\_Date , *Vehicle\_VIN\_Number* , Suspension\_Date , Reason)

The create table as select query for Permit table is listed below.

CREATE TABLE PERMIT\_ AS

(SELECT Distinct Active, Permit\_License\_Number, Permit\_Owner\_Name, Expiration\_Date,*Vehicle\_VIN\_Number*,Reason,Suspension\_Date FROM shl\_orig);

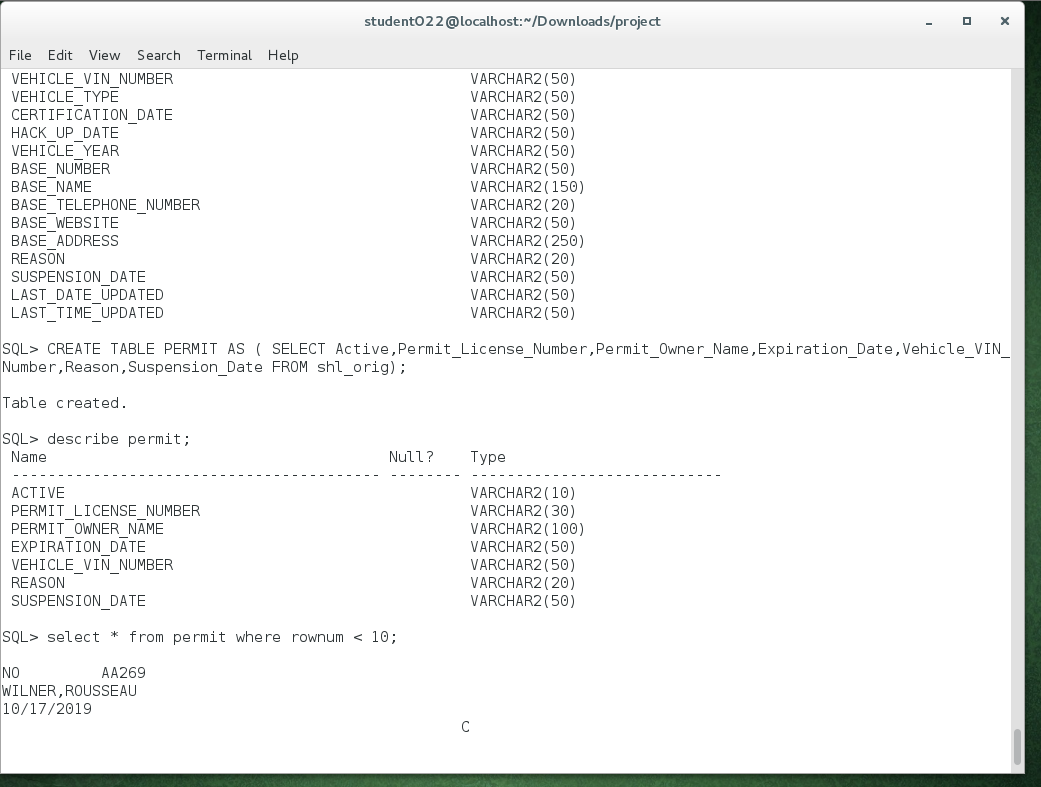


Figure (CREATE TABLE PERMIT)

***5.4.2 Creating Vehicles table:***

The scheme of *Vehicles* table is:

Vehicles (Vehicle\_VIN\_Number , DMV\_License\_Plate\_Number, Vehicle\_Type, Certification\_Date, Hack\_Up\_Date, Vehicle\_Year , *Base\_Number* )

The create table as select query for Permit table is listed below.

CREATE TABLE VEHICLES AS

(SELECT Distinct Vehicle\_VIN\_Number, Vehicle\_License\_Number, Certification\_Date,Hack\_Up\_Date,Vehicle\_Year,*Base\_Number* FROM shl\_orig);

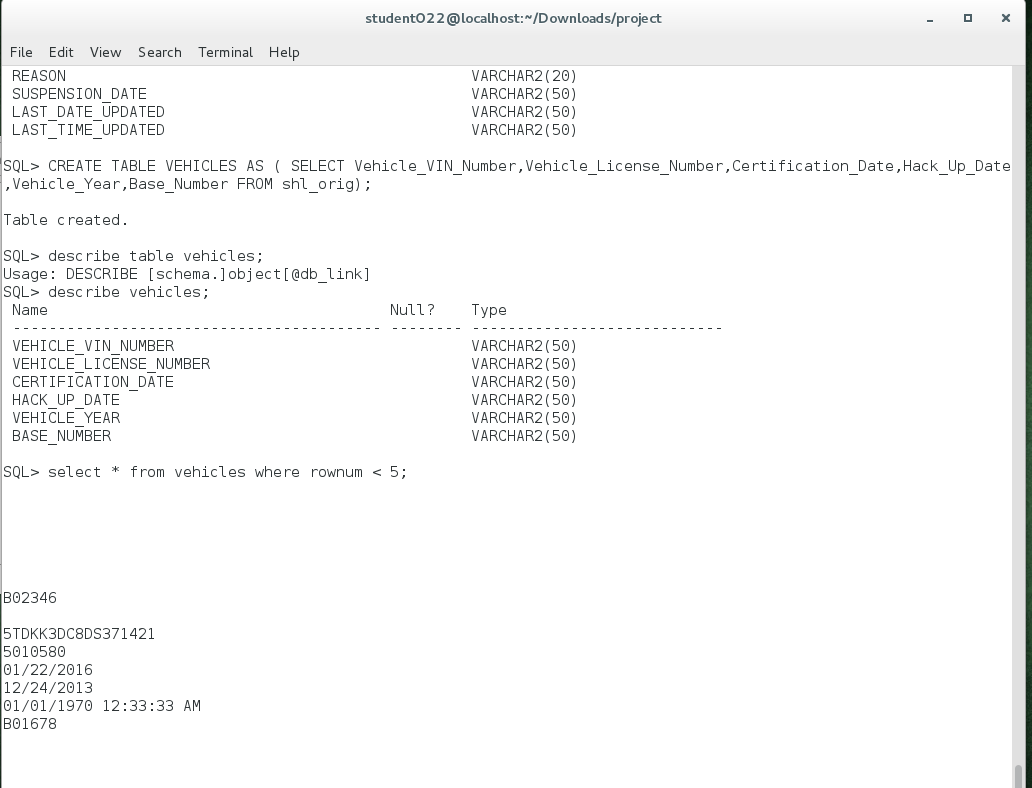


Figure (CREATE TABLE VEHICLES)

***5.4.3 Creating Bases table:***

The scheme of *Vehicles* table is:

Bases (Base\_Number -> Base\_Name , Base\_Telephone\_Number , Base\_Website , Base\_Address)

The create table as select query for Bases table is listed below.

CREATE TABLE BASES AS

(Select Distinct Base\_Number, Base\_Name,Base\_Telephone\_Number,Base\_Website,Base\_Address from shl\_orig);

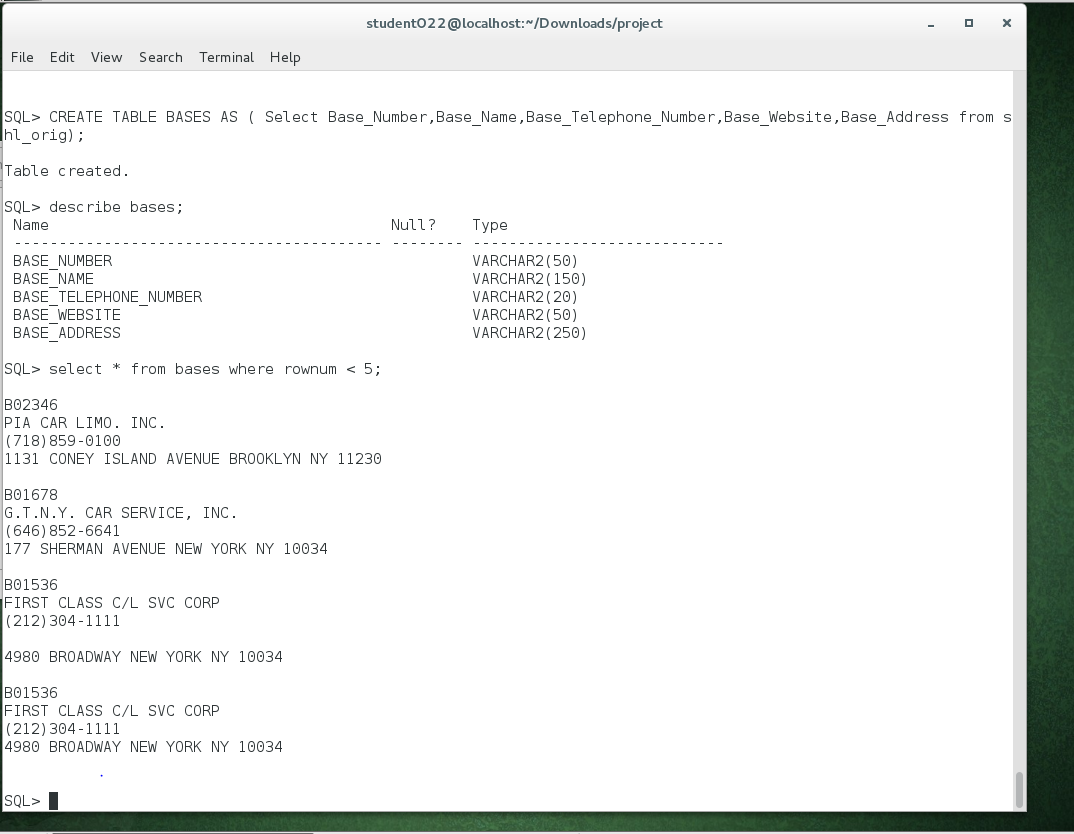


Figure (CREATE TABLE BASES)

# 6. Constraining the Normalized Tables

**6.1** **Reviewing the implied constraints**

In section 5.3 (Normalized Tables), we established the primary and foreign keys based on functional dependencies identified in section 5.2. We identified 3 primary keys for each of permit, vehicle and base table. We also identified two foreign keys, one of them needed to be created in permit table and linking a permit to a licensed vehicle registered with the permit which exists in vehicle table. The other foreign key existed in Vehicle table for Base\_Number which linked a vehicle to a base in Bases table (The vehicle is linked to the base station it is affiliated with).

While creating the constraints for the 3 tables, we will be referencing each table’s structure and keys setup listed in section 5.3.

Note: We are constraining the tables in opposite order as we created them in. This would be easy to understand in next step but the reason is the dependency of tables. Bases table is the most independent table and should be the easiest to constraint since it has no foreign key constraint. Then we will constraint vehicle table, which has a foreign key constraint linked to bases table which would be already constrained in that step. At the end, we will constraint permit table.

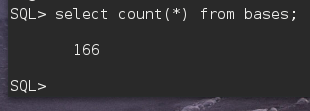
**6.2 Constraining Bases table**

In section 5.4.3 *(Creating Bases table)* , we created the bases table from shl\_orig raw data table with a select as query and using distinct to restrict duplication of data. The structure of created bases table is shown below.

1. **Base ( Base\_Number** -> Base\_Name , Base\_Telephone\_Number , Base\_Website , Base\_Address **)**

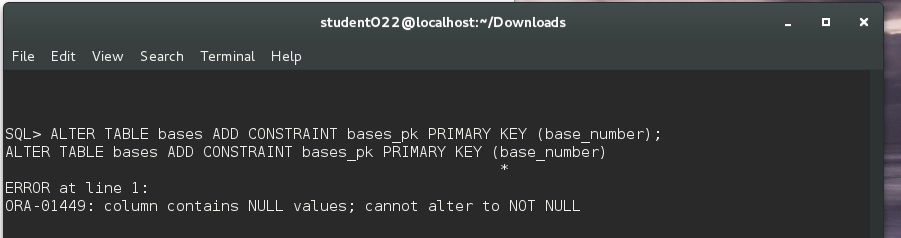
**Primary Key: Base\_Number**

The number of rows in bases table is 166 which can be found by following query.

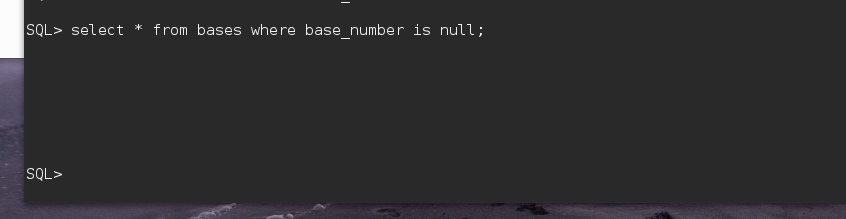


We will be using alter table statement to add a primary key to bases table. The SQL query is listed below

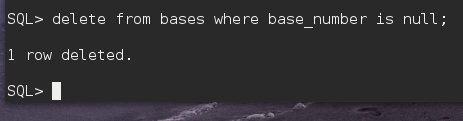
ALTER TABLE bases ADD CONSTRAINT bases\_pk PRIMARY KEY (base\_number);

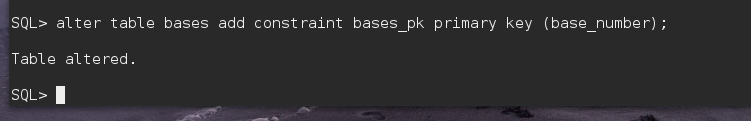


As we see the error above, it’s clear we have some rows in bases table where base\_number field is null. To further examine the issue, we run the following command to find the rows where this condition is true.



The result above shows a row with all null values which is preventing us from setting the primary key. We can drop this row by using Delete from table where base\_number is null;

  
  
Now, we will try to run the alter table query again to set the base\_number field as the primary key of the bases table.



The alter table query runs successfully and we now have successfully set the primary key of table bases. Running a describe on bases will now show that the base\_number field is set to NOT NULL which is a pre-requisite of primary key.



**6.3 Constraining Vehicles table**

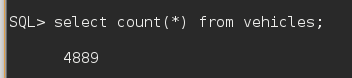
In section 5.4.2 *(Creating Vehicles table)* , we created the Vehicles table from shl\_orig raw data table with a select as query and using distinct to restrict duplication of data. The structure of created Vehicles table is shown below.

1. **Vehicle** ( **Vehicle\_VIN\_Number** , DMV\_License\_Plate\_Number, Vehicle\_Type, Certification\_Date, Hack\_Up\_Date, Vehicle\_Year , Base\_Number)

Primary Key: **Vehicle\_VIN\_Number**

Foreign Key: Base\_Number – References Base (Base\_Number)

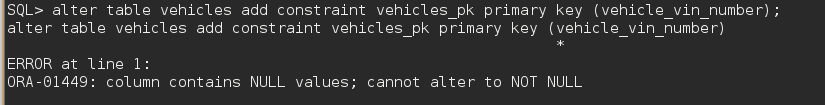
The number of rows in permit table is 4889 which can be found by following query.



We will be using alter table statement to add a primary key to Permits table. The SQL query is listed below

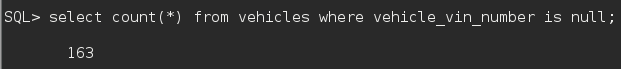
alter table vehicles add constraint vehicles\_pk primary key (vehicle\_vin\_number);

Running the above query gives us following error. To solve table’s constraint error, we will be looking at null values and duplicates if any.



We run the following query to find out number of rows with null vin number.

SQL> select count(\*) from vehicles where vehicle\_vin\_number is null;

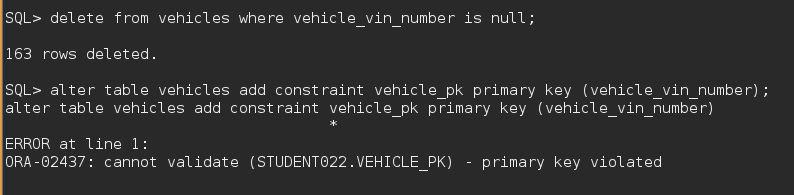


This gives us 163 rows with a null vin number. We do know that the raw data table had the data with inactive permits and it makes sense to see rows with null vin numbers since the inactive permits will have no vehicles licensed to the permit and hence a null vin number. These rows will also have other vehicle related fields as null since there is no vehicle in the permit row. Therefore, we will drop these rows simply because they are of no use to us.

We delete these null rows from vehicle table using following query.

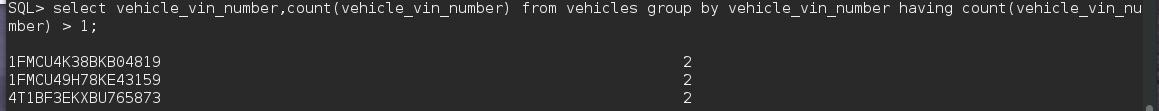
SQL> delete from vehicles where vehicle\_vin\_number is null;

After running this delete query, we run the alter table query again to add a primary key and it fails again. The sqlplus output is shown below.

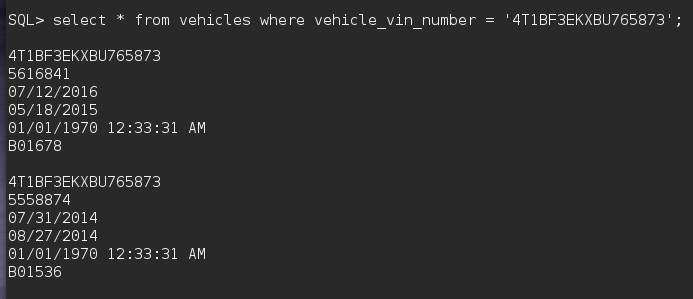


We now can make a fair guess that the error must be coming from duplicated vin numbers. We will check that by following query.

SQL> select vehicle\_vin\_number,count(vehicle\_vin\_number) from vehicles group by vehicle\_vin\_number having count(vehicle\_vin\_number) > 1;



We find three rows of vehicles with duplicated vehicle\_vin\_numbers. On further investigation, we find out that these vehicles are duplicated with an older record with a base and a newer record with a different base. Therefore, these rows are example of a vehicle with a base transfer and the raw data having an older record of the vehicle. We do not have any use of the older record of vehicles previous base, so we will be deleting these rows manually since there are only 3 duplicated rows.



We can use the base number as a condition along with vin number to find the older records. In above example, we aim to delete the second row with base B01536 and vehicle license number 555874. We can use any of the base or vehicle license condition along with vin number to find the exact record and delete it.

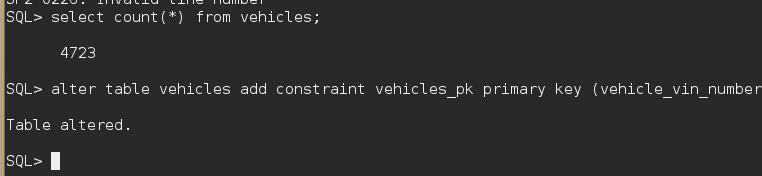
Below are the queries we use with one using base\_number and another using vehicle\_license\_number. Both achieve the same result.

SQL> delete from vehicles where vehicle\_vin\_number = '1FMCU4K38BKB04819' AND vehicle\_license\_number = ‘5696952’;

SQL> delete from vehicles where vehicle\_vin\_number = '4T1BF3EKXBU765873' AND base\_number = 'B01536';



We now run the alter table again which runs successfully and sets the primary key. We also run a count to get the total remaining records in vehicles table;

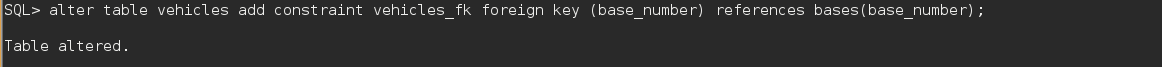


**Foreign Key**

We will now setup the foregin key in this table. We previously explained that the vehicle table is linked to bases with base\_number as a foreign key residing in vehicle table. This relationship is based on the fact that each vehicle has to be affiliated with one and only one base.

The alter table query to run is listed below.

SQL> alter table vehicles add constraint vehicles\_fk foreign key (base\_number) references bases(base\_number);



The query runs successfully and sets the required foreign key.

**6.4 Constraining Permits table**

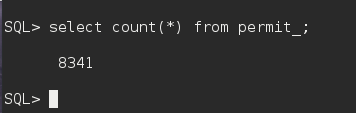
In section 5.4.1 *(Creating Permit table)* , we created the permit table from shl\_orig raw data table with a select as query and using distinct to restrict duplication of data. The structure of created permit table is shown below.

**Permit** ( **Permit\_License\_Number** , Active, Permit\_License\_Number , Permit\_Owner\_Name , Expiration\_Date , Vehicle\_License\_Number , Suspension\_Date , Reason , *Vehicle\_VIN\_Number*)

**Primary Key**: Permit\_License\_Number

**Foreign Key**: Vehicle\_VIN\_Number - References **Vehicle (Vehicle\_VIN\_Number)**

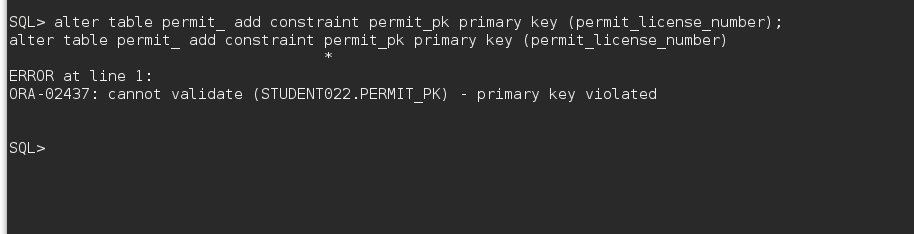
The number of rows in permit table is 8341 which can be found by following query.



We will be using alter table statement to add a primary key to Permits table. The SQL query is listed below.

alter table permit\_ add constraint permit\_pk primary key (permit\_license\_number);

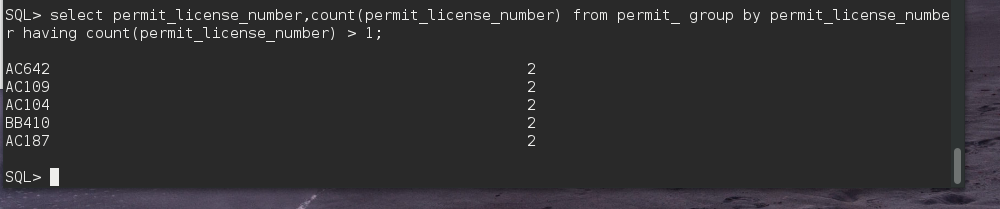
The query ends up giving an error confirming a violation of primary key. Therefore, there must be a null or duplicate permit\_license\_number.



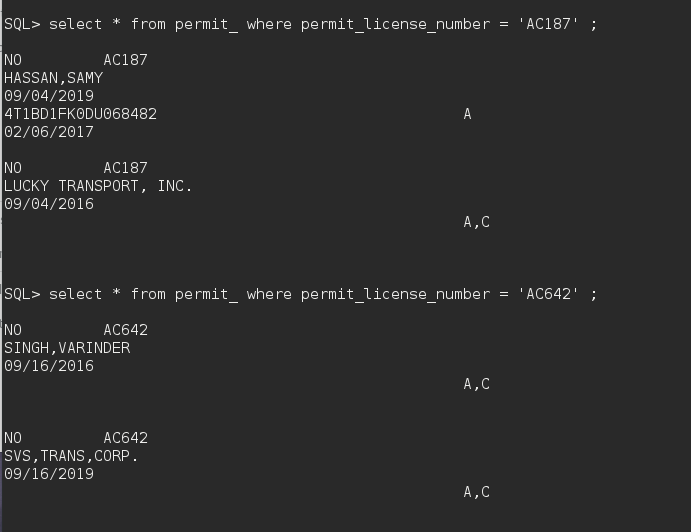
We can find out the permit\_license\_numbers who have a count of more than 1. We run the following query

select permit\_license\_number,count(permit\_license\_number) from permit\_ group by permit\_license\_number having count(permit\_license\_number) > 1;

And see the following result. There are 5 rows with duplicate permit\_license\_number;



We can do further investigation by looking at these records since there are only 10 tuples with issues. While doing the investigation, it was found that this situation could possibly happen as a result of one permit owner selling permit to another owner. But they key takeaway is that we only need to care about a permit which is active since that permit will have a linked vehicle. We found that most of these permits were inactive, except two permits being active and having licensed vehicle. Below is one of record.

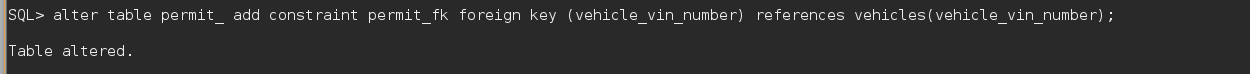


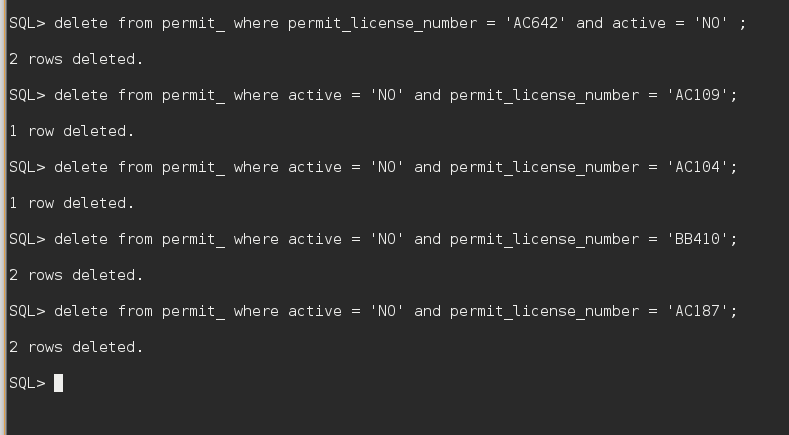
Now, since there cannot be more than one active permit with same permit license number, we can simply delete all inactive permits found in previous query or delete one tuple from each duplicated pair. We need to preserve the two active permits.

Here’s how we deleted the duplicated pair of inactive permits with permit license number ‘AC642’. Similarly, we followed this process for other four rows.

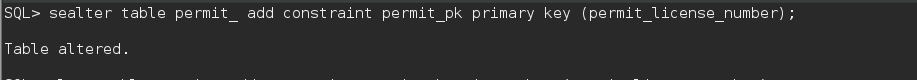
A sample query is shown below. The terminal output is shown next.

delete from permit\_ where active = 'NO' and permit\_license\_number = 'AC109';





Now, we can run the alter table query again to set the primary key and it runs successfully as shown below.

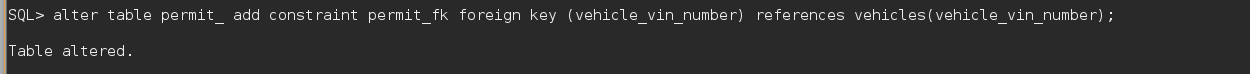


**Foreign Key**

We will now setup the foregin key in this table. We previously explained that the permit table is linked to vehicles with vehicle\_vin\_number as a foreign key residing in permit table. This relationship is based on the fact that each vehicle is licensed under a permit and only one permit.

SQL> alter table permit\_ add constraint permit\_fk foreign key (vehicle\_vin\_number) references vehicles(vehicle\_vin\_number);

The alter table query to run is listed below.



***Therefore, this section completes setting up primary key and foreign key constraints for the three tables.***