San José State University Department of Applied Data Science

DATA 225

Database Systems for Analytics

Instructor: Simon Shim

LAB GROUP PROJECT REPORT 1

SJSU RENTAL DATABASE SYSTEM

Group 5

Group Members:

Koushik Modekurti

Vrushali Pravin Menthe

Raaj Kiran Reddy Anumula

Swathi Kasarabada

Lokesh Eravelli

TABLE OF CONTENT

PROBLEM STATEMENT:

SJSU RENTAL DATABASE is a system where Customers will have a fantastic, simple, and an ideal location to stay when visiting San Jose. It will also provide many people the chance to earn additional money by lending their properties on lease for a particular period of time. Not everyone can afford to buy a home and living. Some individuals might favor renting the home. These people can find such houses, select amenities according to their needs and book their stay. A database is required to keep a track of every single detail and transaction of customer, hosting, amenities, listing, booking, payments, reviews and so on.

The cost of a listing is influenced by a variety of factors. Our database will provide the details about the listings and amenities associated with the particular listings. If one customer wants any particular amenities, he has the option to select those amenities out of all the listing and amenities available. All the the listings will have different kinds of amenities. Therefore, Detailed description of the houses with available listing and amenities will be offered by the system. The customer only has to select options according to their likings and budget.

In the Database and the system there is an section of reviews available through which any customer can see and come to know more about the house, location, and amenities. They can make their choice after going through the reviews of previous customers. These reviews can be a good way to learn about the listing before making any bookings. However, in reality, sometimes the reviews can be manipulated, thus the system provides a feature to determine the most popular property listing based on bookings of the customers.

SOLUTION REQUIREMENTS:

This is a one-stop application which people can use to rent a particular listing for a period of time. From the perspective of the customer, this actor must sign up for the system and go through the mobile number and email verification processes. Customer then lands into a zone where different properties listed by the host are displayed. Host is the actor who owns the listing. Host undergoes the same process which Customer passes through. In addition, the host supplies the system with information on the listings. Customers can view these listings and book a property of their choice. Moreover, user has the privilege to apply customizations to retrieve the desired listing results. Once user-desired listings are displayed, customer views a particular listing and if the listing matches his expectations, he performs booking for the property. Host enforces certain minimum and maximum threshold for number of days a listing can be booked for. Considering these constraints a customer books a listing by making a payment. In accordance to the payment transaction status, booking for a particular listing succeeds or fails. In this way, customer and host leverage the system to enjoy the unique experience of San Jose Rental Database.

LIMITATIONS:

Limitations prevail for most of the systems, for instance, in this system, customers do not get an option to experience virtual tour for the listing before booking. Listing Description available in the system such as the amenities available, the condition of the houses, the prices and so on facilitate customer’s choice to make a booking. Another such limitation is factors such as the location safety, the neighbourhood, etc. are unknown to the customer for performing a booking. Similarly, the host is unknown about the customer. Apart from the customer name, none of the details are revealed on the system. The reviews which are available on the system can not be trusted completely as they can be biased at times. The positive and negative reviews depends on each customer. Each customer will give a review according to their likings. Customer preferences differ, hence reviews might affect the listing popularity.

CONCEPTUAL DATABASE DESIGN:

To implement the SJSU rental system, the system will require the below entities to store the data about the customers and to search the available listings.

SJSU rental system has a total of 12 entities and the details are as follows:

1. Bookings	7. Amentities
2. State	8.Host
3.Country	9.Listings
4.Verification	10.Amenities_Provided
5.Customer	11.Reviews
6.Payment	12.Complaints

1.Bookings : This entity stores the details about the bookings made by the customer. The attributes in this entity are : Booking_ID,Customer_ID,Customer_Name,Listing_ID ,Unit_Alloted ,Price_Incurred ,Booking_From ,Booking_To ,Booking_Category ,Payment_ID,

Primary_Key : Booking_ID

Foreign_Keys : Customer_ID,Listing_ID from the tables Customer and Listings respectively

2.State : This entity stores the details about the state. Attributes details for State entity:
State_ID,Name

Primary_Key : State_ID

3.Country : This entity stores the details about country Attributes details for Country entity:
Country_ID,Name

Primary_Key : Country_ID

4.Verification : This entity stores the details about Verification information whether the customer is successfully verified with the email or phone_number. Attributes details for Verification entity:
Verification_ID ,Category

Primary_Key : Verification_ID

5.Customer : This entity stores the details about the customer information like Name, age email_ID etc., Attributes details for Customer entity: Customer_ID, First_Name ,Last_Name, Gender , Email_ID ,Primary_Phone, Secondary_Phone , Age , Gov_ID, Apt_no , Street_Address, City , State_ID , Zip_Code, Country_ID, Verification_status

Primary_Key : Customer_ID

Foreign_Keys:State_ID,Country_ID,Verification_status from the tables State,Country and Verification respectively

6.Payment: This entity stores the details payment information whether the payment is successfully completed or not and the status of the payment. Attributes details for Payment entity :
Payment_ID,Customer_ID,Booking_ID,Payment_Mode,Comments.

Primary_Key : Payment_ID

Foreign_Keys:Customer_ID,Booking_ID from the tables customer and bookings respectively. We are creating an composite primary key with this attributes : (`Payment_ID`,`Customer_ID`,`Booking_ID`)

7.Amenities : This entity stores the details about amenities provided by the particular listing. Attributes details for Amenities entity: Amenities_ID ,Name

Primary_Key : Amenities_ID

8.Host : This entity stores the details about the Host and the details about the particular listings. Attributes details for Host entity : Host_ID, Name,Age, Gender, Unit_no, Street_Address, City, Zip_code, State_ID, Country_ID, About , Host_since, Response_Time, Acceptance_Rate, Listings_count, License_Number, Verification_status

Primary_Key : Host_ID

Foreign_Keys:State_ID,Country_ID,Verification _status from the tables State,Country and Verification respectively

9.Listings : This entity stores the details about Listings with the images and also to which host it is related.Attributes details for Listings entity: Listings_ID, Name, Host_ID, Neighbourhood, Neighbourhood_Group,City,State_ID, Country_ID,Property_Type,Allowed_Occupants ,Min_Booking_Days, Max_Booking, 1B1B_Units, 1B1B_Price , 2B1B_Units, 2B1B_Price, `2B1.5B_Units` , `2B1.5B_Price, 2B2B_Units, 2B2B_Price , 3B2B_Units, 3B2B_Price, `3B2.5B_Units` , `3B2.5B_Price, 3B3B_Units, 3B3B_Price , 4B3B_Units, 4B3B_Price,`4B3.5B_Units, `4B3.5B_Price, 4B4B_Units, 4B4B_Price, Construction_Year, Images_URL, Custom_Unit_Desc, Custom_Unit_Available, Custom_unit_price.

Primary_Key : Listings_ID

Foreign_Keys: Host_ID, State_ID, Country_ID from the tables Host,State and Country respectively

10.Amenities_Provided : This entity stores the details about Amenities provided by the particular listings.Attributes details for Amenities_Provided entity: Listing_ID,Amenities_ID,

Composite_Primary_Key: Amenities_ID, Listing_ID

Foreign_Keys:Listing_ID,Amenities_ID from the tables Listings and Amenities respectively

11.Reviews : This entity stores the details about reviews given by the customers based on the experience during their stay in the particular listing. Attributes details for Reviews entity : ID ,Reviewer_ID ,Reviewer_Name ,Listing_ID ,Rating ,Comments ,Review_Date

Primary_Key : ID

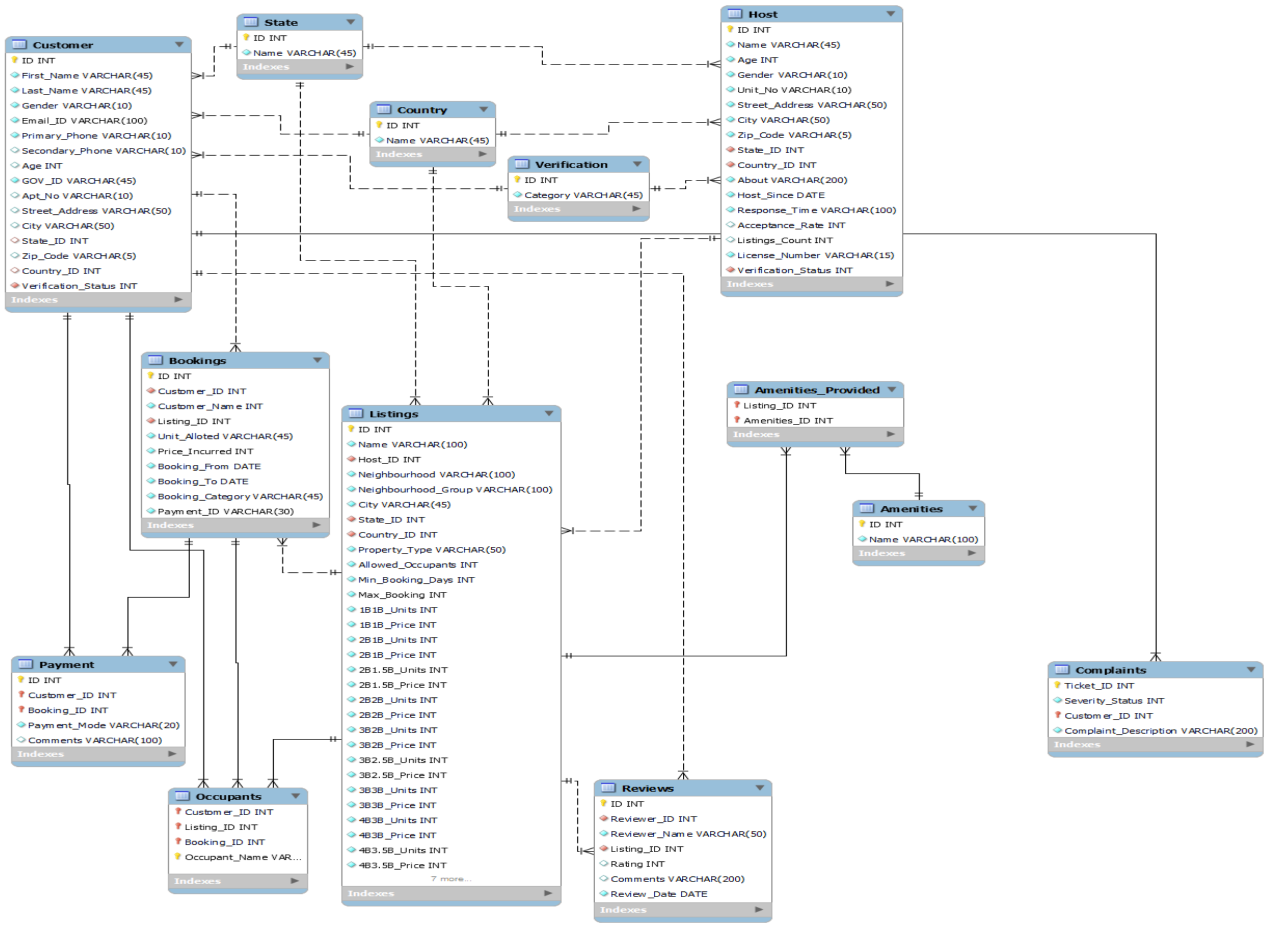
Foreign_Keys:Customer_ID,Listing_ID from the tables customer and Listings respectively.

12.Complaints: This entity stores the details about complaints given by the customers based on the experience during their stay in the particular listing.Attributes details for Complaints entity : Ticket ID, Severity Status, Customer ID, Complaint Description.

Primary_Key : Ticket ID

Foreign_Keys:Customer_ID from the tables customer respectively.

ENTITY RELATIONSHIP DIAGRAM:



FUNCTIONAL ANALYSIS:

We have entities like: Customers, Host, Listings, Bookings, amenities, verification, payment, country, state, reviews. Eachentity has its own importances. When a customer or host views the browser they register prior filling out some basic information like their Name, Phone_Number, Gender, Email etc. which are an essential part of the booking process.

The details of the host or customer will be stored in the their respective relations. The user after their successful signupor signin, will be directed to their corresponding dashboards.

We have some attributes like Customer_ID,First_Name,Last_name,Gender etc and in host we have Host_id,Name,Gender,Unit no etc. After Customer's & Host's Profile Information Entry, the system verifies the identity of the customer or host leveraging the modes of verification in verification table i.e. verification ID and category - Mobile Verification, Email Verification & Both.

After completing the verification process, the customer is directed to the listings page where he can choose the booking of his choice like 1B1B Unit, 2B2B Unit and also the property type which includes Villa, Apartment or a Group House. Some additional factors like price, the number of guests will be provided as filters with a particular limit and length of the booking. The listing entity has attributes like Host id, state ID, and Max Booking. The user will be presented with amenities that the host provides like Wifi, Jacuzzi, Barbeque etc in the form of checkboxes. The user can select his requirements and based on his/her input, system displays the recommendations. The checkboxes will be taken from a stored procedure/entity called Amenities_Provided which has a linear connection with the listings table.

When a customer selects the space of his choice, the booking table appears with attributes like the price and length of stay as well as units allotted and price that the user can recheck. However, before the customer can receive his booking id, he must make a payment, at which point the payment table.After a successful payment, booking id and payment id will be generated in respective columns of our functional tables which are Bookings and Payments.

SQL CODE SNIPPETS/QUERIES:

[1] Get relation between Amenities_Provided and Listings

SELECT L.Listings_ID, L.Name, L.Neighbourhood, L.Neighbourhood_Group, L.City, L.Property_Type, L.Allowed_Occupants FROM amenities_provided AP INNER JOIN listings L ON AP.Listing_ID = L.Listings_ID GROUP BY L.Listings_ID , L.Name , L.Neighbourhood , L.Neighbourhood_Group , L.City , L.Property_Type , L.Allowed_Occupants;

[2]Get Relation between Host and Listings

SELECT H.Host_ID, H.Age, H.Gender, H.Unit_no, H.Street_Address, H.City, H.Zip_code, H.About, H.Host_Since FROM host H INNER JOIN listings L ON L.Host_ID = H.Host_ID GROUP BY H.Host_Id , H.Age , H.Gender , H.Unit_no , H.Street_Address , H.City , H.Zip_code , H.About , H.Host_Since , H.Response_Time;

[3]Get relation between Payment and Bookings

SELECT B.Booking_ID, B.Customer_Id, B.Listing_Id, B.Payment_Id, B.Unit_Alloted, B.Price_Incurred, B.Booking_From, B.Booking_To, P.Payment_mode FROM bookings B INNER JOIN payment P ON B.Booking_ID = P.Booking_ID GROUP BY B.Booking_ID , B.Customer_Id , B.Listing_Id , B.Payment_Id , B.Unit_Alloted , B.Price_Incurred , B.Booking_From , B.Booking_To , P.Payment_mode;

[4]Get the relation between the host(Describes the hosts intital responses and the frequency that the host responds after recieving a booking request) and verification.

SELECT Host_ID, Response_Time, Acceptance_Rate, License_Number, Verification_Status, Category FROM Host INNER JOIN verification ON Verification_Status = Verification_Id ORDER BY Host_Id;

[5]Gets the relation between Customer and the verification process.

SELECT C.Customer_ID, C.First_Name, C.Last_Name, C.Gender,C.Email_ID, V.Category FROM customer C INNER JOIN verification V ON C.Verification_status = V.Verification_ID ORDER BY Customer_Id

[6]List out grouped listings and amenities for property type apartments.

SELECT AP.Amenities_ID, AP.Listing_ID, A.Name, L.Name, L.City, L.Allowed_Occupants, L.Property_Type FROM listings L INNER JOIN amenities_provided AP ON AP.Listing_ID = L.Listings_ID INNER JOIN amenities A ON A.Amenities_ID = AP.Amenities_ID INNER JOIN host H ON L.Host_ID = H.Host_ID WHERE L.Property_Type = 'Apartment' GROUP BY AP.Amenities_ID , AP.Listing_ID , A.Name , L.City , L.Allowed_Occupants , L.Property_Type;

[7]Returns host with the biggest number of listings.

SELECT H.Host_ID, H.Name, L.Neighbourhood, L.Neighbourhood_Group, H.City, L.Property_Type FROM listings L INNER JOIN host H ON H.Host_ID = L.Host_ID GROUP BY H.Host_ID, H.Name, L.Neighbourhood, L.Neighbourhood_Group, H.City, L.Property_Type HAVING MAX(H.Listings_Count);

[8]Get the average of prices per neighbourhood_group and room types.

SELECT L.Neighbourhood_Group, L.Property_Type, ROUND(AVG(1B1B_Price), 1) 1Bed, ROUND(AVG(2B1B_Price), 1) 2and1Bed, ROUND(AVG(2B2B_Price), 1) 2and2Bed, ROUND(AVG(3B2B_Price), 1) 3and2Bed, ROUND(AVG(3B3B_Price), 1) 3and3Bed, ROUND(AVG(4B3B_Price), 1) 4and3Bed, ROUND(AVG(4B4B_Price), 1) 4and4Bed FROM listings L GROUP BY L.Neighbourhood_Group , L.Property_Type , 1B1B_Price , 2B1B_Price , 2B2B_Price , 3B2B_Price , 3B3B_Price , 4B3B_Price , 4B4B_Price ORDER BY L.Neighbourhood_Group;

[9]Get the relationships between prices and minimum_nights

SELECT L.Min_Booking_Days, L.Property_Type, ROUND(AVG(L.1B1B_Price), 1) 1Bed, ROUND(AVG(L.2B1B_Price), 1) 2and1Bed, ROUND(AVG(L.2B2B_Price), 1) 2and2Bed, ROUND(AVG(L.3B2B_Price), 1) 3and2Bed, ROUND(AVG(L.3B3B_Price), 1) 3and3Bed, ROUND(AVG(L.4B3B_Price), 1) 4and3Bed, ROUND(AVG(L.4B4B_Price), 1) 4and4Bed FROM listings L INNER JOIN reviews R ON L.Listings_ID = R.Listing_ID GROUP BY L.Min_Booking_Days , L.Property_Type , L.1B1B_Price , L.2B1B_Price , L.2B2B_Price , L.3B2B_Price , L.3B3B_Price , L.4B3B_Price , L.4B4B_Price;

[10]Get the relationships between prices and number_of_reviews

SELECT L.Name, R.Reviewer_Name, R.Rating, R.Review_Date, ROUND(AVG(L. 1B1B_Price), 1) 1Bed, ROUND(AVG(L. 2B1B_Price), 1) 2and1Bed, ROUND(AVG(L. 2B2B_Price), 1) 2and2Bed, ROUND(AVG(L. 3B2B_Price), 1) 3and2Bed, ROUND(AVG(L. 3B3B_Price), 1) 3and3Bed, ROUND(AVG(L. 4B3B_Price), 1) 4and3Bed, ROUND(AVG(L. 4B4B_Price), 1) 4and4Bed, L.Images_Url FROM listings L INNER JOIN reviews R ON L.Listings_ID = R.Listing_ID GROUP BY L.Name , L. 1B1B_Price , L. 2B1B_Price , L. 2B2B_Price , L. 3B2B_Price , L. 3B3B_Price , L. 4B3B_Price , L. 4B4B_Price , L.Images_Url , R.Reviewer_Name , R.Rating , R.Review_Date;

STORED PROCEDURES:

[1]Average Rating of Listing based on Reviews

```
DELIMITER $$
CREATE PROCEDURE average_of_reviews()
BEGIN
SELECT Listing_ID, ROUND(AVG(Rating),1)
FROM reviews GROUP BY Listing_ID; END
$$
CALL average_of_reviews();
```

[2]Extract Customer Details and their given Rating

```
DELIMITER //
CREATE PROCEDURE Customer_Rating(IN
Customer_ID INT)
BEGIN
SELECT C.Customer_ID, C.First_Name,
C.Last_Name, C.Gender, C.Email_ID, C.Age,
C.City, R.Rating FROM
customer C
JOIN
reviews R ON C.Customer_ID =
R.Reviewer_ID
WHERE C.Customer_ID = Customer_ID;
END //
DELIMITER
CALL Customer_Rating(3);
```

TRIGGERS:

[1] Primary Phone updation trigger for customers

```
DELIMITER $$
CREATE TRIGGER
CUSTOMER_PRIMARY_PHONE_UPDATE
AFTER UPDATE ON
sjsu_rental_database.customer
FOR EACH ROW
BEGIN
INSERT INTO Trigger_Phone_Update
VALUES(
OLD.Primary_Phone,
NEW.Primary_Phone);
END$$
```

[2]Displays Payment Details Message

```
use sjsu_rental_database;
drop trigger if exists after_insert_payment;
delimiter //
create trigger after_insert_payment
after insert on payment for each row
begin
declare message varchar(100);
set message = 'The customer' + new.customer_id
+ 'did the payment via' + new.Payment_Mode;
end //
delimiter ;
```

[3]Captures Review Details with Rating < 2

```
use sjsu_rental_database;
drop trigger if exists after_insert_reviews;
delimiter //
create trigger after_insert_reviews
after insert on reviews for each row
begin
        declare message varchar(100);
  set message = Reviewer_Name+'has given the'
+ Rating + 'for the listing_ID' + Listing_ID;
   if new.Rating < 2 then
   signal sqlstate  '45000'
   set message_text = message;
   end if;
end //
delimiter ;
```

ACCESS PRIVILEGES:

USER CREATIONS FOR ACCESS PRIVILEGES

```
SELECT Host,User FROM mysql.user;
CREATE USER 'kruthi@DBADMIN'
IDENTIFIED BY 'kruthi123';
```

```
CREATE USER 'latha@DBADMIN' IDENTIFIED BY 'latha123';
GRANT ALL PRIVILEGES ON sjsu_rental_database.* TO 'kruthi@DBADMIN'@'localhost' WITH GRANT OPTION;
```

LOGGING OF DATABASE:

This System's Database (AWS RDS) is backed with the potential of AWS Cloudwatch for logging General Logs, Slow Query Logs and Error Logs. These logs can be obtained from the AWS Cloudwatch anytime. It is achieved by configuring Cloudwatch for the RDS Instance while creating a RDS Instance (Logs can also be configured later at any point of time). It is in the following format:

```
22-10-17T17:41:23.909468Z         109
arning]    [MY-010055]    [Server]    IP
dress  '205.210.31.136'  could  not  be
solved: Name or service not known
```

```
22-10-17T17:54:14.774959Z         115
arning]    [MY-010055]    [Server]    IP
dress   '98.51.4.41'   could   not   be
solved: Name or service not known
```

```
22-10-17T18:16:36.468768Z         123
arning]    [MY-010055]    [Server]    IP
dress  '152.32.142.133'  could  not  be
solved: Name or service not known
```

```
22-10-17T19:39:57.913806Z         142
arning]    [MY-010057]    [Server]    IP
dress   '174.249.129.228'   has   been
solved     to     the     host     name
28.sub-174-249-129.myvzw.com',    which
sembles IPv4-address itself.
```

```
22-10-17T19:39:58.864427Z         143
arning]    [MY-010057]    [Server]    IP
dress   '174.249.129.228'   has   been
solved     to     the     host     name
28.sub-174-249-129.myvzw.com',    which
sembles IPv4-address itself.
```

```
22-10-17T21:39:58.007205Z         167
arning]    [MY-010055]    [Server]    IP
dress  '130.65.254.10'  could  not  be
solved: Name or service not known
```

AWS CONNECTIVITY WITH PYTHON

These days, every business enterprise are developing their applications backed by a database sitting in the cloud, it is because Cloud aids the firms with provisions of scalability, networking, maintenance. Therefore, this system's database is also migrated from MySQL Workbench's local instance to an instance created in AWS RDS - a service by AWS which provides its customers with the features of configuring the capacity, parameter groups, security groups, database engine, RAM Capacity, no. of virtual CPUs to be deployed for any application.

Firstly, for moving data from a local MySQL Workbench Instance to AWS RDS Instance, a connection needs to be established to the AWS RDS Instance with the username, password and RDS Instance's endpoint. After establishing a successful connection, data migration can be performed by exporting the database using Data Export Option in the Administration section of MySQL Workbench and dumping it in the self-contained file at a custom file location. This file can be later used to import database into AWS RDS Instance's Connection.

DB Connection can also be checked using Python with the help of testing it in Python's Embedded SQL. Following is the demonstration for loading the Listing Name which is the oldest (CONSTRUCTION YEAR) from the database srd-aws (database instance created in AWS RDS):

```
import mysql.connector
from mysql.connector import errorcode
try:
   cnx =
mysql.connector.connect(user='koushik',
password='koushik123',

host='srd-aws.cnbysbbelnea.us-west-2.rds.amaz
onaws.com',

database='sjsu_rental_database')
   cursor = cnx.cursor()
   query = ("SELECT Name, Host_ID, City,
MIN(Construction_Year) AS 'Oldest Listing'
FROM listings")
   cursor.execute(query)
   for (Name, Host_ID, City, Construction_Year)
in cursor:
      print("{} is the oldest listing constructed in
{} in {} and is listed by the Host ID
{}".format(Name, Construction_Year, City,
Host_ID))
   cursor.close()
   cnx.close()
except mysql.connector.Error as err:
  if err.errno ==
errorcode.ER_ACCESS_DENIED_ERROR:
   print("Something is wrong with your user
name or password")
  elif err.errno ==
errorcode.ER_BAD_DB_ERROR:
   print("Database does not exist")
  else:
   print(err)
else:
  cnx.close()
```

PERFORMANCE MEASUREMENT:

MySQL Workbench has the potential to generate a variety of reports with substantial analysis on Memory Usage, Hot Spots for I/O, High Cost SQL Statements, Database Schema statistics, Wait Event Times, InnoDB Statistics, User Resource Use. This project’s scope includes analysis of High Cost SQL Statements. From the study and reports generated by using Performance Report Option in Server Menu, it can be observed that DDL(Data Definition Language) commands fall under the category of Top 5% High Cost SQL Statements. Complex Join Queries, Inner queries, DML(Data Manipulation) Commands have a relatively lower execution time and thus are faster than DDL.