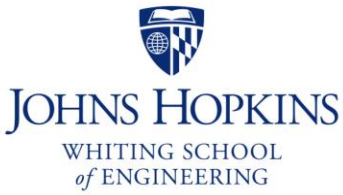


Property Rentals Platform

Course Section: CS605.641.81
Summer, 2020

Prepared by

Anh Ta
08/05/2020

Table of Contents

1. Introduction

At the time of this project implementation, COVID-19 has ravaged the travel and leisure industry. Working as an investment analyst, I have seen many hospitality investments gone sour due to this pandemic. Nevertheless, the hospitality industry remained an interesting industry to me. Therefore, this project is geared to model a property rentals application, which serves as a platform connecting hosts and guests. Hosts can generate income by having their properties rented out while guests can spend their time staying at one of their desired properties owned by hosts. The business model, which connects buyers/owners and sellers/renters, have disrupted many long-standing industries (i.e. the Uber and Lyft model disrupting the taxi business). Hence, I thought this would be a great project to implement my database knowledge by building a 'Property Rentals' database.

1.1. Scope and Purpose of Document

The document discusses the i) project objective, ii) requirements, iii) conceptual and logical design, and iv) database implementation. The project objective section covers the objectives of the 'Property Rentals' application. The requirements section covers hardware, software, functional, and database requirements. The conceptual and logical design covers database design rationale, E/R model, relational model, security, and database backup, and case tool. The database implementation covers data dictionary, advanced features, and queries.

1.2. Project Objective

The project's objective is to design a 'Property Rentals' database using a relational Database Management System. The database application 'Property Rentals' is a one-stop shop for connecting i) Hosts who wish to list their properties for rent and ii) Guests who wish to stay at the leased properties either for vacations, mid to long-term stays, or other purposes.

The database project creates a platform allowing:

1) Hosts to list their Properties for rent, which is browsable by Guests.
2) Guests to book Reservations with Hosts.
3) Guests to conduct their Stays at their chosen Properties after their Reservations are accepted by the Hosts.
4) Guests and Hosts to Transact securely on the platform.
5) Guest and Host can write Reviews on each other; Guest can write Reviews on Properties, which can then be browsed by other Guests.
6) Guests and Hosts can look up Reviews before making rental decisions (i.e.Yelp concepts).
7) 'Property Rentals' application will earn a fee for each of the Payment Transaction conducted by Hosts and Guests (the app main source of app revenue).

2. System Requirements

My choice of software is MySQL. Hence the requirements stated below (both minimum and preferred requirements) are for supporting MySQL on a local machine, where the database project is going to be launched. In addition, if developers also want to implement web-based application in conjunction with the database, the system requirements should also support those other tools.

2.1 Hardware Requirements

As mentioned in the "MySQL Hardware Requirements" on the MySQL documentation website in the Reference section, MySQL's minimum hardware requirements include - i) 2 CPU Cores, ii) 2GB RAM, and iii) Disk I/O subsystem applicable to a write-intensive database ("MySQL Enterprise Monitor 4.0 Manual: 3.2.1 System Requirements").

The required disk space to run MySQL locally is 800MB for a Service Manager Minimum Disk Space on a Windows x86 64-bit machine.

However, for the best performance, the following configuration, which is needed to run the database on a local machine, includes – i) 4 CPU Cores or more, ii) 8 GB RAM of more, and iii) RAID10 or RAID 0+1 disk set up.

2.2 Software Requirements

The DBMS software for this project is MySQL, which is a robust RDBMS software and supports fully every feature of this project. MySQL can be deployed locally on Linux (both 32-bit and 64-bit versions), Solaris (both x-86 and Sparc versions), Mac OS X, and Windows (both 32-bit and 64-bit versions) ("Chapter 3. Installing and Launching MySQL Workbench").

2.3 Functional Requirements

The database application should support:

1) Entity Insertion, Deletion, and Modification:

- New users (i.e. Guests and Hosts) can be created on the platform via Insertion command. Users on the platform can also request their information to be deactivated (however, not deleted from the database). The Host's deactivation will render the associated Properties' verification to be False. In addition, users can also update their information on the application, which is going to be reflected accordingly in the database.
- These actions above also apply to non-user entities on the platform, which include Reservation, Stay, Transaction Payment, Reviews, and Verifications:
  i) Guests and Hosts are required to create a Reservation, which can only be created if the requested Property is verified.
  ii) A Reservation can be converted into a Stay; after a Stay is finished, a Payment associated with that Stay will occur. The Stay information can be deleted if an error was made during data entry.

iii) Guests can write reviews on Properties; Guests and Hosts can write review on one another. Reviews can be deleted if they are found to be inappropriate by the platform management team.

iv) Property verifications can be updated to check the latest verification status of a Property.

2) Querying for information:
   i) The database supports querying user entities such as Guests and Hosts
   ii) The database also supports querying information on non-user entities such as Reservation, Stay, Transaction Payment, Reviews, and Verifications.

3) Supporting performance:
   i) The database will avoid creating weak entities and generating composite keys in the process as much as possible to improve database performance. In lieu of creating Composite Keys, the database will rely on creating Foreign Keys with Non-Null constraints.

2.4 Database Requirements

Product	Vendor	Version	Comments
MySQL	Oracle	8.0.21	Relational Database Management System

3. Database Design Description

The database is geared to support the "Property Rentals" application. Hence, the database will contain i) user entities such as GUEST and HOST and ii) non-user entities such as RESERVATION, STAY, TRANSACTION_PAYMENT, PAYMENT_INFO, PROPERTY, ZIP_CODE, ROOM, PHOTO, PROPERTY_VERIFICATION, GUEST_REVIEW, PROPERTY_REVIEW, and HOST_REVIEW. In total there are fourteen entities within the "Property Rentals" database schema. There is also a back-up table called archiveTransactionPayments, which archive deleted transactions. The 'Property Rentals' database complies with the Third Normal Form, which is ideal for maintaining database integrity while preserving performance speed.

3.1 Design Rationale

The 'Property Rentals' database is designed to achieve balance of i) preserving sound relationship among entities and ii) maintaining good performance. Initially, I designed the database with almost all weak entities which generated a lot of composite keys and slowed down performance considerably. In addition, a lot of the composite keys' implementation violate the second normal form as attributes are dependent only on one of the composite keys but not on all of them.

Please see below for an example. I have a Host-Guest Review-Guest relationship. Hosts can write many reviews on Guests and vice versa. I can certainly design a many-to-many weak entity construct with composite keys of Guest_id and Host_id (Figure 1). However,

this design will be a drag on performance because the ordering of the composite keys is important in determining the database performance. Instead, I create the GUEST_REVIEW relation with Guest_id and Host_id as Foreign Key setting to NON-NULL (Figure 2). In this way, I can enhance database performance, while maintaining that the attribute for Guest_id and Host_id to be supplied by the GUEST and HOST relation respectively before the creation of a GUEST_REVIEW tuple.

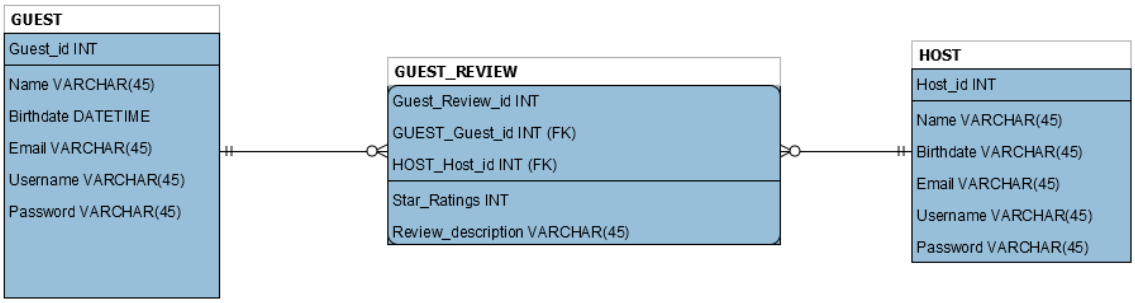


Figure 1: Weak entity approach with composite keys creation dragging performance

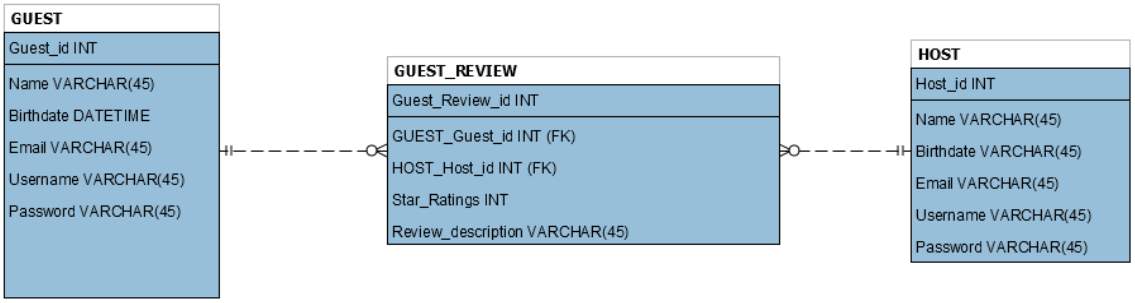


Figure 2: Strong entity approach with single primary key and two NON-NULL foreign keys (Guest_id and Host_id), enhancing performance

In addition, in my database, there is no one to one relationship as this can cause a loop constraint. I opted instead for "zero to many" relationship to make the implementation of the database system feasible.

3.2 E/R Model

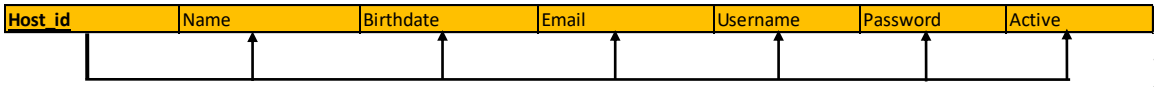
There are three groups of entities which include i) user-related entities including GUEST and HOST relations, ii) property-related entities including PROPERTY, PROPERTY_VERIFICATION, ZIP_CODE, TRANSACTION_PAYMENT, PAYMENT_INFO, RESERVATION, STAY, ROOM, and PHOTO, and iii) reviews-related entities including HOST_REVIEW, PROPERTY_REVIEW, and GUEST_REVIEW.

3.2.1 Entities

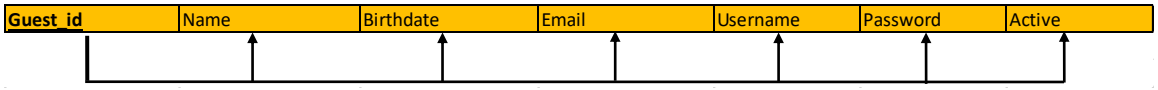
Entities will roll up into user-related entities and non-user related entities (property-related and reviews-related entities).

User-related entities:

HOST entity consists of i) Host_id as Primary Key and ii) Name, Birthdate, Email, Username, Password, and Active.

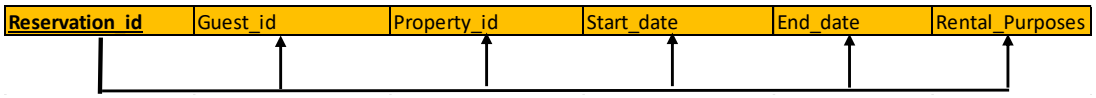


GUEST entity consists of i) Guest_id as Primary Key and ii) Name, Birthdate, Email, Username, Password, and Active.

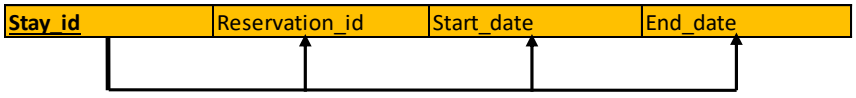


Non user-related entities:

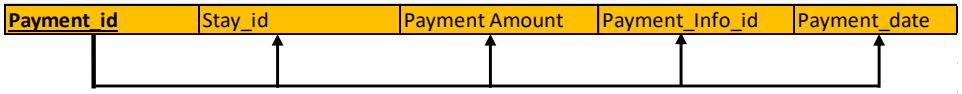
RESERVATION entity consists of i) Reservation_id as Primary Key, ii) Guest_id and Property_id as NON-NULL Foreign Keys, and iii) Start_date, End_date, and Rental_Purposes.



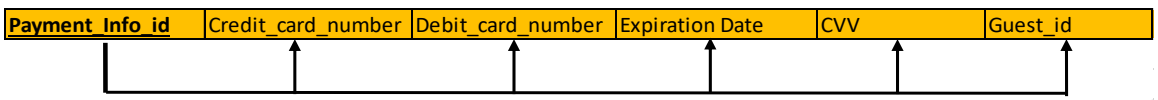
STAY entity consists of i) Stay_id as Primary Key, ii) Reservation_id as NON-NULL Foreign Key, and iii) Start_date and End_date.



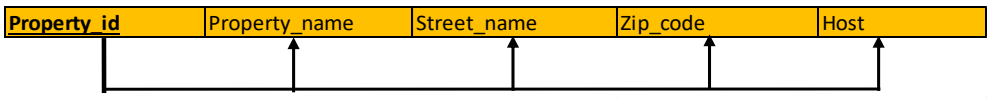
TRANSACTION_PAYMENT consists of i) Payment_id as Primary Key, ii) Payment Amount, and iii) Stay_id and Payment_Info_id as NON-NULL Foreign Keys.



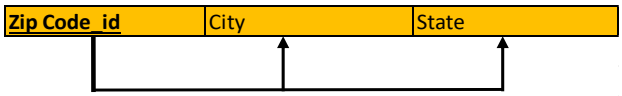
PAYMENT_INFO entity consists of i) Payment_Info_id as Primary Key, ii) Credit_Card_number, and Debt_card_number, Expiration_date, and CVV and iii) Guest_id as NON-NULL Foreign Key.



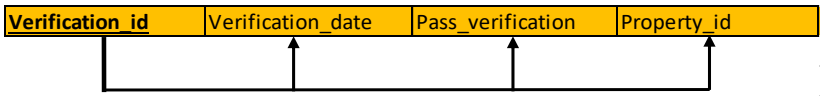
PROPERTY entity consists of i) Property_id as Primary Key, ii) Host as NON-NULL Foreign Key, and iii) Property_name, Street_name, and Zip_code



ZIP_CODE entity consists of i) Zip_Code_id as Primary Key and ii) City and State.



PROPERTY_VERIFICATION entity consists of i) Verification_id as Primary Key, ii) Verification_date, Pass_verification, and iii) Property_id as NON-NULL Foreign Key.



ROOM entity consists of i) Room_id and Property_id as Composite Key and ii) Room_type and Smoking.

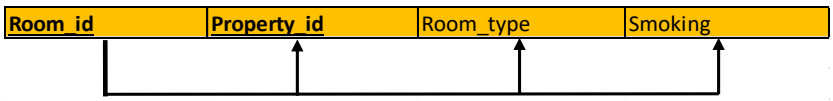
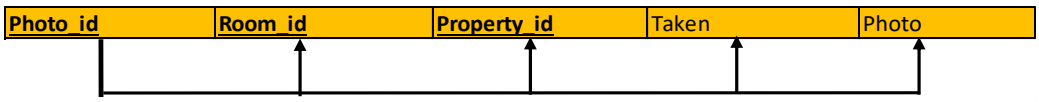
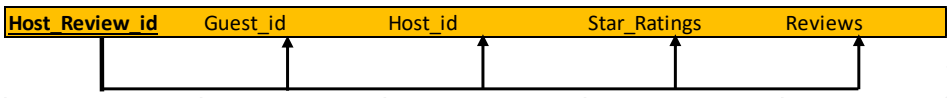


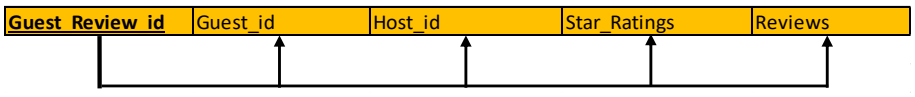
PHOTO entity consists of i) Photo_id, Room_id, and Property_id as Composite Key and ii) Taken and Photo.



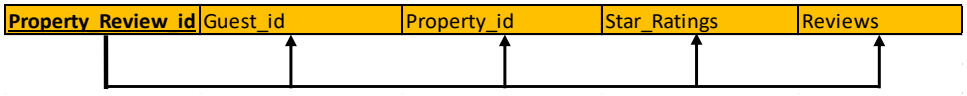
HOST_REVIEW entity consists of i) Host_Review_id as Primary Key, ii) Guest_id and Host_id as NON-NULL Foreign Keys, and iii) Star_Ratings and Reviews.



GUEST_REVIEW entity consists of i) Guest_Review_id as Primary Key, ii) Guest_id and Host_id as NON-NULL Foreign Keys, and iii) Star_Ratings and Reviews.

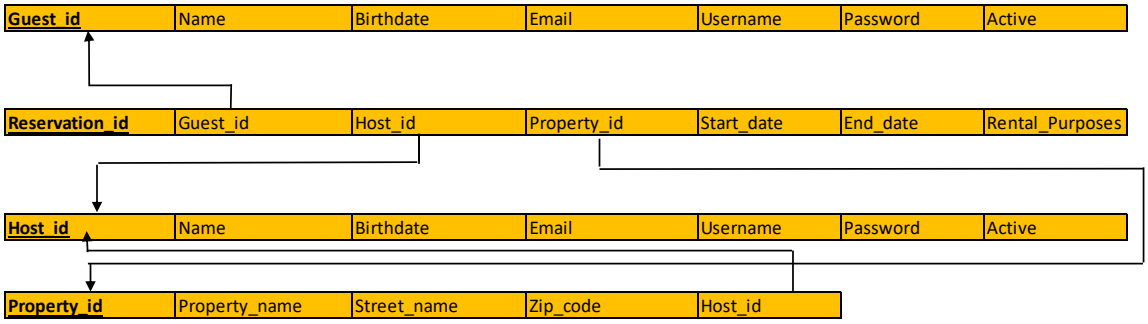


PROPERTY_REVIEW entity consists of i) Property_Review_id as Primary Key, ii) Guest_id and Property_id as NON-NULL Foreign Keys, and iii) Star_Ratings and Reviews.

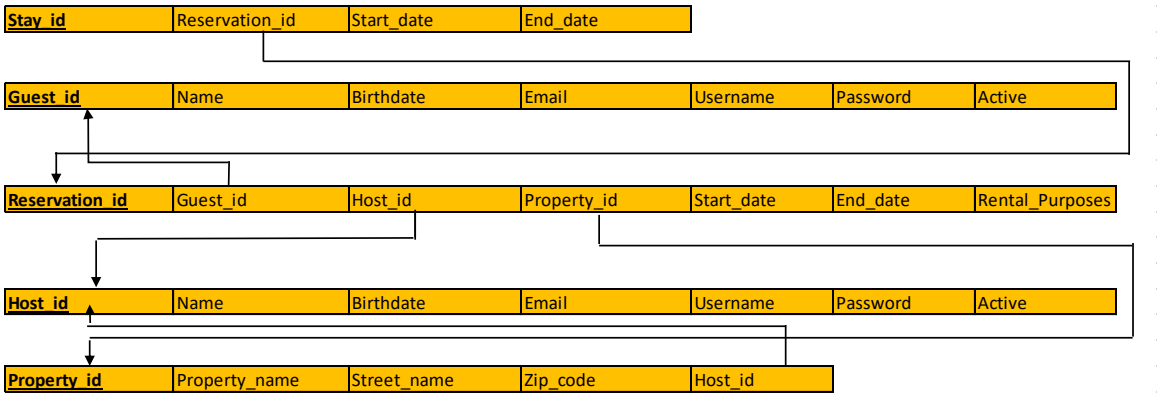


3.2.2 Relationships

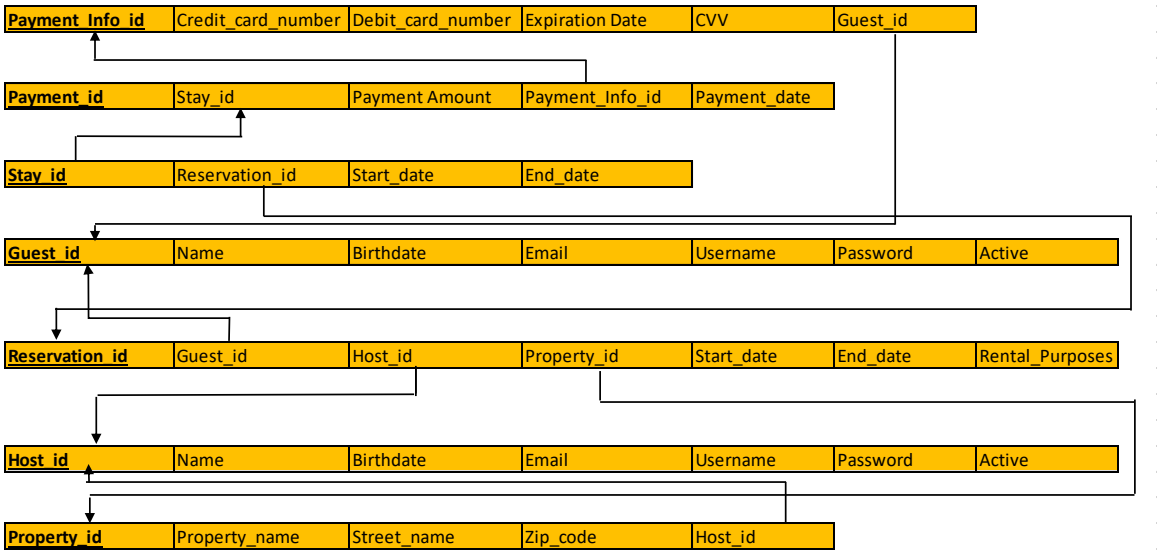
GUESTS can set up multiple RESERVATIONS.
A PROPERTY can also have multiple RESERVATIONS from GUESTS.
A HOST can own multiple PROPERTIES.



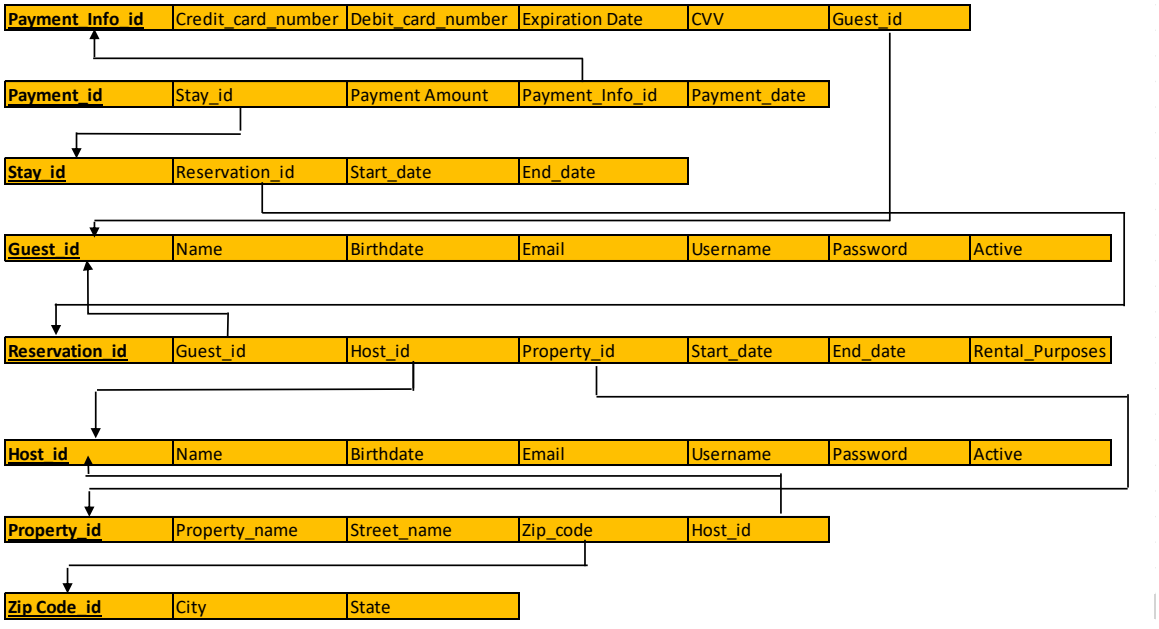
Once GUEST commit to a STAY, a STAY Entity will be created.
A RESERVATION can have multiple STAYS.



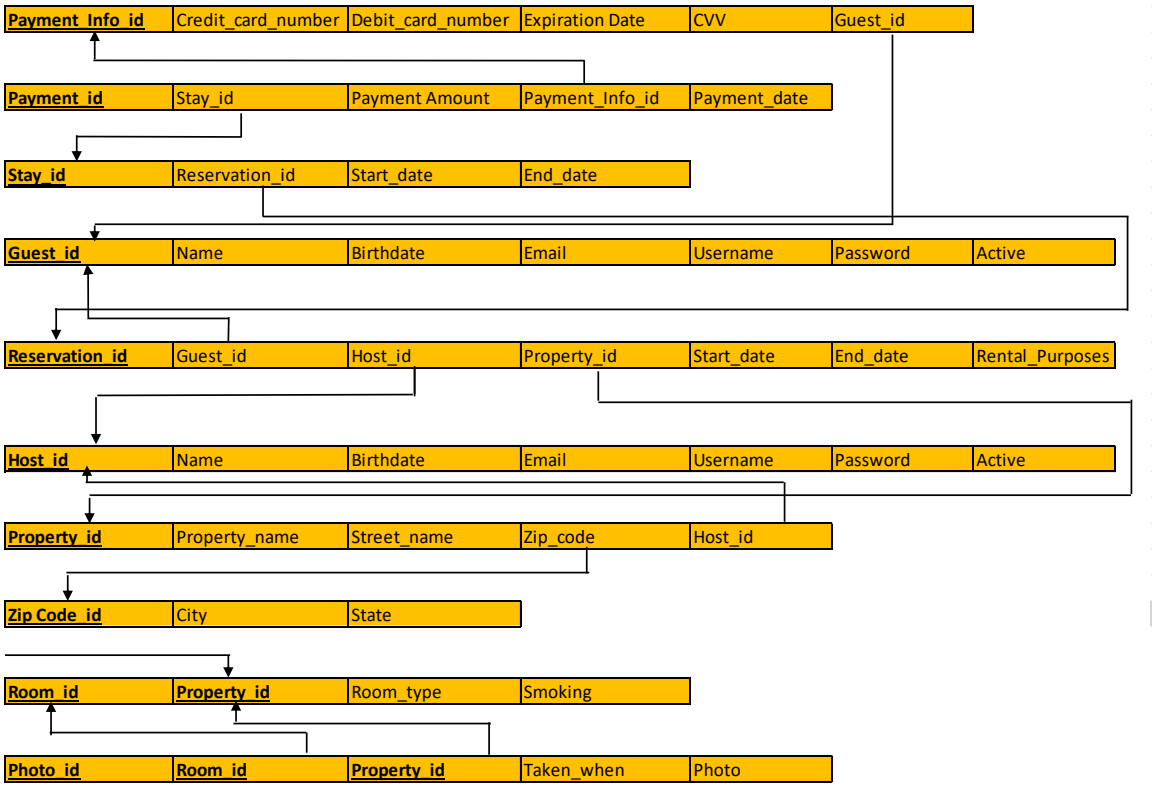
Once GUESTS finish the STAY, they will be paying for the STAY in PAYMENT_TRANSACTIONS.
A STAY can have multiple TRANSACTION_PAYMENTS.
A GUEST can have multiple PAYMENT_INFOS.



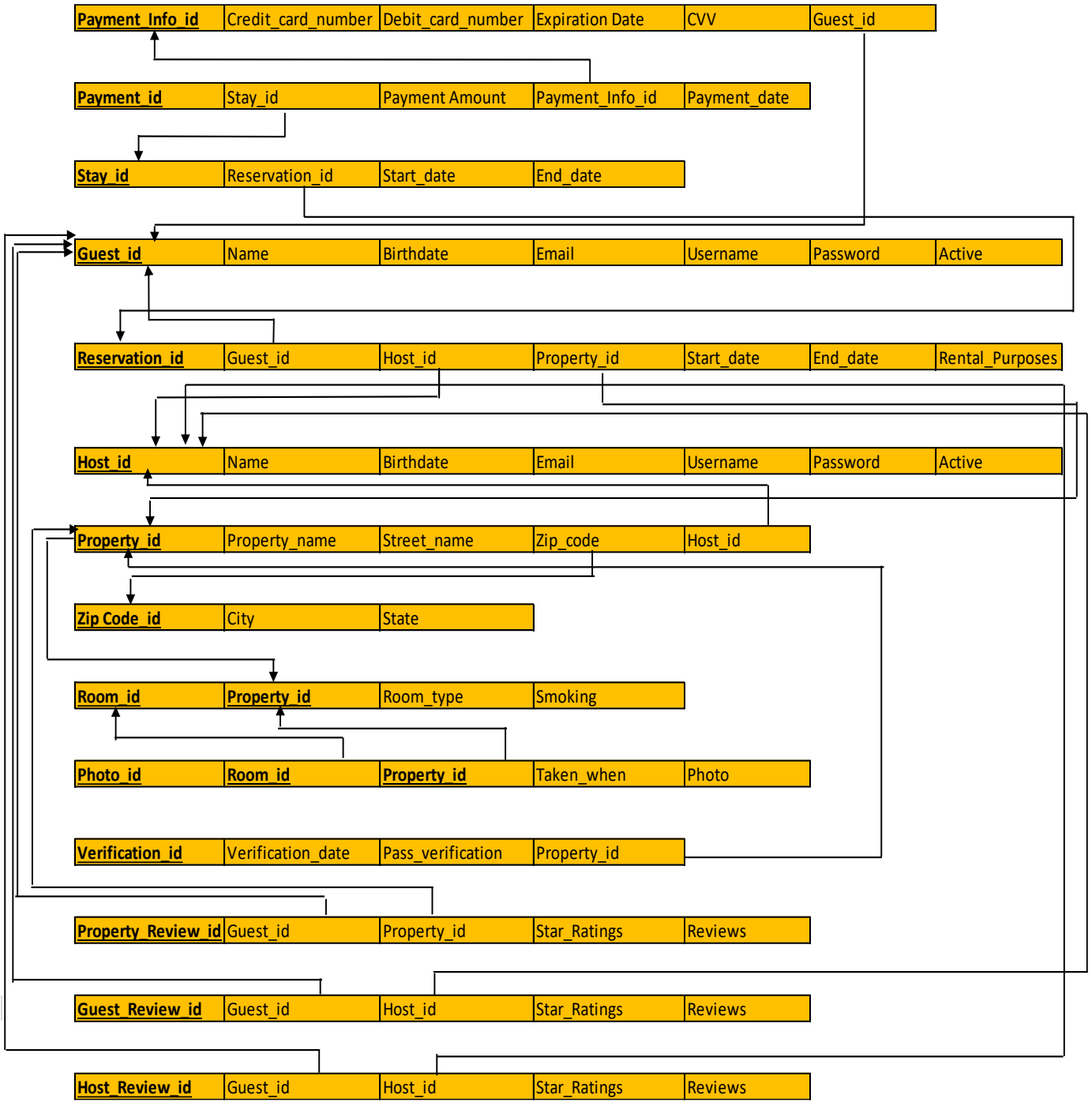
A ZIP_CODE can have many PROPERTIES.



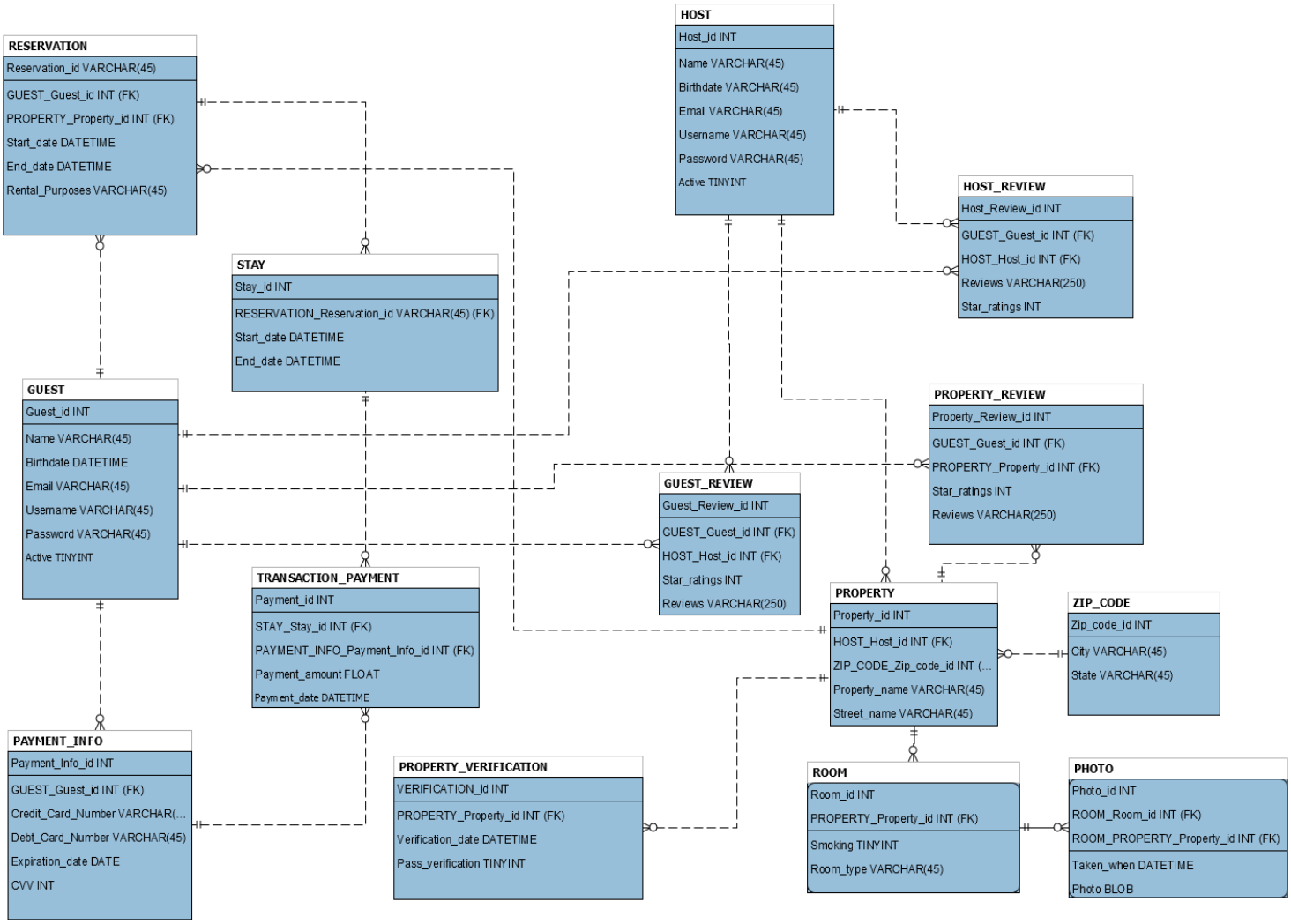
A PROPERTY can have many ROOMS.
A ROOM can have many PHOTOS attached.



A PROPERTY can have multiple VERIFICATIONS.
GUESTS and HOSTS can have multiple GUEST_REVIEWS.
GUESTS and HOSTS can also have multiple HOST_REVIEWS.
GUESTS and PROPERTIES can have multiple PROPERTY_REVIEWS.



3.2.3 E/R Diagram



Please also see an enlarged version at the back of the document.

3.3 Relational Model

3.3.1 Data Dictionary

User-Related Data Dictionary

HOST

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Host_id	Index of Guest	INT	4 bytes	Primary Key	Y	Greater than zero
Name	Name of the host	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Birthdate	Birthdate of the host	DATETIME	8 bytes	None	N	After 1/1/1900
Email	Email of the host	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Username	Username of the host	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Password	Password of the host	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Active	Whether the host is active?	TINYINT	1 byte	None	N	True or False

GUEST

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Guest_id	Index of Guest	INT	4 bytes	Primary Key	Y	Greater than zero
Name	Name of the guest	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Birthdate	Birthdate of the guest	DATETIME	8 bytes	None	N	After 1/1/1900
Email	Email of the guest	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Username	Username of the guest	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Password	Password of the guest	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Active	Whether the guest is active?	TINYINT	1 byte	None	N	True or False

Non-User-Related Data Dictionary

RESERVATION

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Reservation_id	Index	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key referencing Guest	Y	Greater than zero
Property_id	Property Identification	INT	4 bytes	Foreign Key referencing Property	Y	Greater than zero
Start_date	Reservation Start Date	DATETIME	8 bytes	None	Y	After 1/1/1900
End_date	Reservation End Date	DATETIME	8 bytes	None	Y	End_date > Start_date

STAY

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Stay_id	Index	INT	4 bytes	Primary Key	Y	Greater than zero
Reservation_id	Reservation Identification	INT	4 bytes	Foreign Key Referencing Reservation	Y	Greater than zero
Start_date	Stay Start Date	DATETIME	8 bytes	None	Y	After 1/1/1900
End_date	Stay End Date	DATETIME	8 bytes	None	Y	End_date > Start_date

TRANSACTION_PAYMENT

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Payment_id	Index	INT	4 bytes	Primary Key	Y	Greater than zero
Stay_id	Stay Identification	INT	4 bytes	Foreign Key Referencing Stay	Y	Greater than zero
Payment_Info	Payment_Info Identification	INT	4 bytes	Foreign Key Referencing Payment_Info	Y	Greater than zero
Payment_amount	Transacted payment amount	FLOAT	4 bytes	None	Y	Greater than zero

PAYMENT_INFO

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Payment_Info_id	Index	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key Refering Guest	Y	Greater than zero
Credit_card_number	Credit Card Number	VARCHAR	45 bytes	None	N	Maximum 45 characters
Debit_card_number	Debit Card Number	VARCHAR	45 bytes	None	N	Maximum 45 characters
Expiration Date	Debit/Credit Card Expiration Date	DATETIME	8 bytes	None	N	After 1/1/1900
CVV	Numbers at the back of card	INT	4 bytes	None	N	Greater than zero

HOST_REVIEW

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Host_review_id	Index of Property Review	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key referencing Guest	Y	Greater than zero
Host_id	Host Identification	INT	4 bytes	Foreign Key referencing Host	Y	Greater than zero
Star Ratings	Rating of Host	INT	4 bytes	None	Y	Between 1 to 10
Reviews	Review of the Host	VARCHAR	250 bytes	None	Y	Maximum 250 characters

PROPERTY REVIEW

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Property_Review_id	Index of Property Review	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key referencing Guest	Y	Greater than zero
Property_id	Index of Property	INT	4 bytes	Foreign Key referencing Property	Y	Greater than zero
Star Ratings	Rating of Property	INT	4 bytes	None	Y	Between 1 to 10
Reviews	Review of the Property	VARCHAR	250 bytes	None	Y	Maximum 250 characters

GUEST_REVIEW

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Guest_review_id	Index of Property Review	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key referencing Guest	Y	Greater than zero
Host_id	Host Identification	INT	4 bytes	Foreign Key referencing Host	Y	Greater than zero
Star Ratings	Rating of Host	INT	4 bytes	None	Y	Between 1 to 10
Reviews	Review of the Host	VARCHAR	250 bytes	None	Y	Maximum 250 characters

PROPERTY

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Property_id	Index of Property	INT	4 bytes	Primary Key	Y	Greater than zero
Zip_code_id	Zip Code Identification	INT	5 bytes	Foreign Key referencing Zip_Code	Y	Maximum 5 digits
Host_id	Host Identification	INT	4 bytes	Foreign Key referencing Host	Y	Greater than zero
Property_Name	Name of the Property	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Street_Name	Name of the Street	VARCHAR	45 bytes	None	Y	Maximum 45 characters

PROPERTY_VERIFICATION

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Property_Verification_id	Index of Property Verification	INT	4 bytes	Primary Key	Y	Greater than zero
Property_id	Index of Property	INT	4 bytes	Foreign Key referencing Property	Y	Greater than zero
Verification_Date	Verification Date of the Property	DATETIME	8 bytes	None	Y	After 1/1/1900
Pass_Verification	Does Property pass the test?	TINYINY	1 byte	None	Y	True or False

ZIP_CODE

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Zip_code_id	Index of Zip Code	INT	4 bytes	Primary Key	Y	Maximum 5 digits
City	City referenced by the zip code	VARCHAR	45 bytes	None	Y	Maximum 45 characters
State	State referenced by the zip code	VARCHAR	45 bytes	None	Y	Maximum 45 characters

ROOM

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Room_id	Index of Room	INT	4 bytes	Composite Key	Y	Greater than zero
Property_id	Property Identification	INT	4 bytes	Composite Key/Foreign Key referencing Property	Y	Greater than zero
Room_type	Type of the room	INT	4 bytes	None	N	Greater than zero
Smoking	Does Room allow smoking?	TINYINT	1 byte	None	N	True or False

PHOTO

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Photo_id	Index of Photo	INT	4 bytes	Composite Key	Y	Greater than zero
Room_id	Room Identification	INT	4 bytes	Composite Key/Foreign Key referencing Room	Y	Greater than zero
Property_id	Property Identification	INT	4 bytes	Composite Key/Foreign Key referencing Room	Y	Greater than zero
Taken_when	Date of the photo taken	DATETIME	8 bytes	None	N	After 1/1/1900
Photo	The actual photo	BLOB	65,535 bytes	None	Y	Image Type

3.3.2 Integrity Rules

I handled mandatory fields by setting attributes as NON-NULL. String inputs are set as VARCHAR; integer inputs are set as INT; payment inputs are set as FLOAT to handle decimals. I set positive numbers as unsigned to make sure that only positive numbers are accepted.

Below are the references in my database:

- RESERVATION table references Guest_id in GUEST and Host_id in HOST
- STAY table references the Reservation_id in the RESERVATION table.
- TRANSACTION_PAYMENT table references Stay_id in the STAY table and Payment_Info_id in the PAYMENT_INFO table.
- PAYMENT_INFO table references Guest_id in the GUEST table.
- HOST_REVIEW table references Host_id in the HOST table and Guest_id in the GUEST table.
- GUEST_REVIEW table references Guest_id in the GUEST table and Host_id in the HOST table.
- PROPERTY_REVIEW table references Guest_id in the GUEST table and Property_id in the PROPERTY table.
- PROPERTY table references Zip_code_id in the ZIP_CODE table and Host_id in the HOST table.
- ROOM table references Property_id in the PROPERTY table.

- PHOTO table references Room_id in the ROOM table and Property_id in PROPERTY table.

3.3.3 Operational Rules

There is no deletion of a HOST or a GUEST; we can only set the status of a HOST and a GUEST to be inactive. Once a HOST is set to be inactive, the PROPERTIES, which are owned by that HOST, will have their verification status set to False. We can then look at that flag and elect to display the PROPERTIES to a client on the website or include those PROEPRTIES in certain calculations.

We cannot delete a GUEST, HOST, PROPERTY, or RESERVATION from our database; we can only make GUEST and HOST to be inactive to i) maintain the information asset of our platform and ii) make sure that our children don't get deleted with the parents entity getting deleted (i.e. children entities of HOST include PROPERTY).

We could delete GUEST_REVIEW, HOST_REVIEW, and PROPERTY_REVIEW (children records). We can choose to delete a record in the STAY table if they are caused by incorrect data entry (but we need to delete the associated payments/children records first).

3.3.4 Operations

There are multiple operations in the database, including inserting/updating/retrieving GUETS, HOSTS, PROPERTIES, ZIP_CODE, RESERVATIONS, PAYMENT_INFO. We also support deletion of ROOM, PHOTO, TRANSACTION_PAYMENTS, and REVIEWS.

Booking a RESERVATION would involve retrieving the ids of a GUEST and a HOST (if already exist) and inserting the estimated starting and ending dates. Making a Payment requires retrieving a Stay id and the associated payment amount. Writing a Review requires i) retrieving the respective parties who receive the review and ii) retrieving the respective parties who write the review. Lastly, verifying a PROPERTY requires retrieving Property's id.

3.4 Security

GUESTS' financial information is sensitive information which is stored in a separate PAYMENT_INFO table without the GUESTS' other information such as Name, Username, etc. An attacker will need access to two tables to piece out the client's information and full identity.

There are also several risks related to SQL injections, which include SQL injection such as bypassing authentication, performing privilege escalation, denial of service, etc. The 'Property Rentals' application will also implement techniques to prevent SQL injection techniques such as input validation (to be implemented on the client side).

I also create a trigger (please see the Advanced Features section) which creates an *audit trail* whenever a payment transaction is deleted for regulatory and compliance reason. This enhances the security of the database system.

3.5 Database Backup and Recovery

According to the MySQL documentation, MySQL offers several types of database backup, which includes logical versus physical, full versus incremental, etc. Physical backup includes raw copies of the directories and files that store database contents; Logical backups save information represented as logical database structure; Online backups stores database information in a server while MySQL server is running; Offline backups occur when the server is stopped ("Chapter 7 Backup and Recovery").

MySQL also supports two types of recovery – i) full recovery and ii) incremental recovery. Full recovery restores all data from a full backup. Incremental recovery only recovers changes made during a specific period.

3.6 Using Database Design or CASE Tool

MySQL Workbench supports CASE tool in drawing out the different tables and their associated relationships. In addition, MySQL Workbench also supports "Forward Engineering", which generates SQL code in constructing tables and the various relationships without having to write complete SQL code. I find these tools to be especially useful in supporting fast database development.

3.7 Other Possible E/R Relationships

I have considered a one-to-many relationship between i) TRANSACTION_PAYMENT (weak cross section entity) and GUEST and HOST. However, since a TRANSACTION_PAYMENT is always linked with a STAY, I can always retrieve the identity of the respective GUESTS and HOSTS via joining with STAY and RESERVATION.

4. Implementation Description

I used MySQL Workbench to create an ERD diagram There are fourteen tables in total, which is the same as in the modeling section above.

4.1 Data Dictionary

The below command is used to run to describe the implemented MySQL Dictionary

```
1  USE PROPERTY_RENTALS;
2
3  DESCRIBE GUEST;
4  DESCRIBE HOST;
5  DESCRIBE PROPERTY;
6  DESCRIBE PROPERTY_VERIFICATION;
7  DESCRIBE ROOM;
8  DESCRIBE PHOTO;
9  DESCRIBE ZIP_CODE;
10 DESCRIBE RESERVATION;
11 DESCRIBE STAY;
12 DESCRIBE TRANSACTION_PAYMENT;
13 DESCRIBE PAYMENT_INFO;
14 DESCRIBE GUEST_REVIEW;
15 DESCRIBE HOST_REVIEW;
16 DESCRIBE PROPERTY_REVIEW;
```

DESCRIBE GUEST;

Field	Type	Null	Key	Default	Extra
Guest_id	int unsigned	NO	PRI	NULL	
Name	varchar(45)	NO		NULL	
Birthdate	datetime	YES		NULL	
Email	varchar(45)	NO		NULL	
Username	varchar(45)	NO		NULL	
Password	varchar(45)	NO		NULL	
Active	tinyint	YES		1	

DESCRIBE HOST;

Field	Type	Null	Key	Default	Extra
Host_id	int unsigned	NO	PRI	NULL	
Name	varchar(45)	NO		NULL	
Birthdate	varchar(45)	YES		NULL	
Email	varchar(45)	NO		NULL	
Username	varchar(45)	NO		NULL	
Password	varchar(45)	NO		NULL	
Active	tinyint	YES		1	

DESCRIBE PROPERTY;

Field	Type	Null	Key	Default	Extra
Property_id	int unsigned	NO	PRI	NULL	
HOST_Host_id	int unsigned	NO	MUL	NULL	
ZIP_CODE_Zip_code_id	int unsigned	NO	MUL	NULL	
Property_name	varchar(45)	NO		NULL	
Street_name	varchar(45)	NO		NULL	

DESCRIBE PROPERTY_VERIFICATION;

Field	Type	Null	Key	Default	Extra
VERIFICATION_id	int unsigned	NO	PRI	NULL	
PROPERTY_Property_id	int unsigned	NO	MUL	NULL	
Verification_date	datetime	NO		NULL	
Pass_verification	tinyint	NO		NULL	

DESCRIBE ROOM;

Field	Type	Null	Key	Default	Extra
Room_id	int unsigned	NO	PRI	NULL	
PROPERTY_Property_id	int unsigned	NO	PRI	NULL	
Smoking	tinyint	YES		1	
Room_type	varchar(45)	YES		General Purposes	

DESCRIBE PHOTO;

Field	Type	Null	Key	Default	Extra
Photo_id	int unsigned	NO	PRI	NULL	
ROOM_Room_id	int unsigned	NO	PRI	NULL	
ROOM_PROPERTY_Property_id	int unsigned	NO	PRI	NULL	
Taken_when	datetime	YES		NULL	
Photo	blob	NO		NULL	

DESCRIBE ZIP_CODE;

Field	Type	Null	Key	Default	Extra
Zip_code_id	int unsigned	NO	PRI	NULL	
City	varchar(45)	NO		NULL	
State	varchar(45)	NO		NULL	

DESCRIBE RESERVATION;

Field	Type	Null	Key	Default	Extra
Reservation_id	varchar(45)	NO	PRI	NULL	
GUEST_Guest_id	int unsigned	NO	MUL	NULL	
PROPERTY_Property_id	int unsigned	NO	MUL	NULL	
Start_date	datetime	NO		NULL	
End_date	datetime	NO		NULL	
Rental_Purposes	varchar(45)	YES		Others	

DESCRIBE STAY;

Field	Type	Null	Key	Default	Extra
Stay_id	int unsigned	NO	PRI	NULL	
RESERVATION_Reservation_id	varchar(45)	NO	MUL	NULL	
Start_date	datetime	NO		NULL	
End_date	datetime	NO		NULL	

DESCRIBE TRANSACTION_PAYMENT;

Field	Type	Null	Key	Default	Extra
Payment_id	int unsigned	NO	PRI	NULL	
STAY_Stay_id	int unsigned	NO	MUL	NULL	
PAYMENT_INFO_Payment_Info_id	int unsigned	NO	MUL	NULL	
Payment_amount	float unsigned	YES		0	
Payment_date	datetime	YES		NULL	

DESCRIBE PAYMENT_INFO;

Field	Type	Null	Key	Default	Extra
Payment_Info_id	int unsigned	NO	PRI	NULL	
GUEST_Guest_id	int unsigned	NO	MUL	NULL	
Credit_Card_Number	varchar(45)	YES		NULL	
Debt_Card_Number	varchar(45)	YES		NULL	
Expiration_date	date	YES		NULL	
CVV	int	YES		NULL	

DESCRIBE GUEST_REVIEW;

Field	Type	Null	Key	Default	Extra
Guest_Review_id	int unsigned	NO	PRI	NULL	
GUEST_Guest_id	int unsigned	NO	MUL	NULL	
HOST_Host_id	int unsigned	NO	MUL	NULL	
Star_ratings	int	NO		NULL	
Reviews	varchar(250)	YES		Default - No Comment	

DESCRIBE HOST_REVIEW;

Field	Type	Null	Key	Default	Extra
Host_Review_id	int unsigned	NO	PRI	NULL	
GUEST_Guest_id	int unsigned	NO	MUL	NULL	
HOST_Host_id	int unsigned	NO	MUL	NULL	
Reviews	varchar(250)	YES		Default - No Comment	
Star_ratings	int	NO		NULL	

DESCRIBE PROPERTY_REVIEW;

Field	Type	Null	Key	Default	Extra
Property_Review_id	int unsigned	NO	PRI	NULL	
GUEST_Guest_id	int unsigned	NO	MUL	NULL	
PROPERTY_Property_id	int unsigned	NO	MUL	NULL	
Star_ratings	int	NO		NULL	
Reviews	varchar(250)	YES		Default - No Comment	

4.2 Advanced Features

Stored Procedure Implementation

One of the most frequent used functions of any business is to calculate the revenue generated. I will implement a *calculatePlatformRevenueByState* stored procedure for the 'Property Rentals' application. The stored procedure will take in two variables Start_date and End_date and calculates how much transaction/revenue (in dollar term) occurring on the platform in the period between those two dates by state. This function will be useful to the management team to draw out analytics on performing and non-performing states.

```
DELIMITER $$
CREATE PROCEDURE calculatePlatformRevenueByState(IN Start_date DATE, IN
End_date DATE)
BEGIN
SELECT Z.State, SUM(TP.Payment_amount) AS REVENUE_BY_STATE
FROM TRANSACTION_PAYMENT AS TP
LEFT JOIN STAY as S
ON TP.STAY_Stay_id = S.Stay_id
LEFT JOIN RESERVATION AS R
ON S.RESERVATION_Reservation_id = R.Reservation_id
LEFT JOIN PROPERTY AS P
ON R.PROPERTY_Property_id = P.Property_id
LEFT JOIN ZIP_CODE AS Z
ON P.ZIP_CODE_Zip_code_id = Z.Zip_code_id
WHERE TP.Payment_date BETWEEN Start_date AND End_date
GROUP BY Z.State;
END$$
DELIMITER ;

CALL calculatePlatformRevenueByState('2020-01-01', '2020-03-31');
```

Result (can also apply a multiplier (5 cents/dollar) to calculate commission fees):

	State	REVENUE_BY_STATE
▶	Maryland	1000
	California	1250
	New York	2250
	Georgia	3500

Triggers Implementation

a. Update Property Verification status to be false when a host deactivates their profile

If a HOST, through a web client, wants to delete his or her profile. We will update their properties' verification to be False so that their PROPERTIES will not be displayed on the web client. I will implement a TRIGGER after an update on the HOST; that TRIGGER will then assign Pass_verification in the PROPERTY_VERIFICATION table to be False if there is a HOST with a new deactivated status.

```
DROP TRIGGER IF EXISTS updatePropertyVerification ;
DELIMITER $$
CREATE TRIGGER updatePropertyVerification AFTER UPDATE ON HOST
        FOR EACH ROW
  BEGIN
                UPDATE PROPERTY_VERIFICATION PV
                INNER JOIN PROPERTY AS P
                ON PV.PROPERTY_Property_id = P.Property_id
                INNER JOIN HOST AS H
```

```
            ON H.Host_id = P.HOST_Host_id
            SET PV.Pass_verification = False
            WHERE H.Active = False;
   END;
$$
DELIMITER ;

-- Show the implemented Triggers
SHOW TRIGGERS;

-- Deactive the first Host
UPDATE HOST AS H
SET H.Active = False
WHERE H.Host_id = 1;

SELECT * FROM HOST;
SELECT * FROM PROPERTY_VERIFICATION;
```

Host (id = 1) becomes deactivated (Active = 0).

Host_id	Name	Birthdate	Email	Username	Password	Active
1	James Keller	1990-11-10	jameskeller@gmail.com	jameskeller	IamJamesKeller	0
2	Michael Jones	1980-10-10	michaeljones@gmail.com	michaeljones	IamMichaelJones	1
3	Allie Johnson	1985-09-02	alliejohnson@gmail.com	alliejohnson	IamAllieJohnson	1
4	Jackson Williams	1987-01-01	jacksonwilliams@gmail.com	jacksonwilliams	IamJacksonWilliams	1
5	Joshua Stach	1987-03-22	joshuastach@gmail.com	joshuastach	IamJoshuaStach	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Their Properties' Pass_verification status will be set to 0 (highlighted in blue).

VERIFICATION_id	PROPERTY_Property_id	Verification_date	Pass_verification
1	1	2020-01-01 00:00:00	0
2	2	2020-02-01 00:00:00	1
3	3	2020-03-01 00:00:00	0
4	4	2020-02-01 00:00:00	1
5	5	2020-04-01 00:00:00	0
6	6	2020-04-01 00:00:00	1
NULL	NULL	NULL	NULL

b. Archive a Transaction Payment into a separate table if we delete one of the transactions

We would want to delete a transaction payment if that payment was made by mistake. However, because payment is sensitive information, we might want to archive those transactions (at least for 6 months) to comply with laws and regulations. I would create a backup table called *archiveTransactionPayments* which contain the deleted transaction payments. I create a trigger called *archiveTransactionPayments* to record the deleted payments into that table upon a deletion occurring.

```
CREATE TABLE ARCHIVE_TRANSACTION_PAYMENT
(
        Delete_id INT NOT NULL AUTO_INCREMENT,
   Payment_id INT NOT NULL,
        STAY_Stay_id INT,
   PAYMENT_INFO_Payment_Info_id INT,
   Payment_amount FLOAT,
   Payment_date DATE,
   Delete_date DATE,
   PRIMARY KEY(Delete_id)
);

DROP TRIGGER  IF EXISTS archiveTransactionPayments;
DELIMITER $$
CREATE TRIGGER archiveTransactionPayments
BEFORE DELETE ON TRANSACTION_PAYMENT FOR EACH ROW
BEGIN
  INSERT INTO ARCHIVE_TRANSACTION_PAYMENT (Payment_id, STAY_Stay_id,
PAYMENT_INFO_Payment_Info_id, Payment_amount, Payment_date, Delete_date)
VALUES
  (OLD.Payment_id, OLD.STAY_Stay_id, OLD.PAYMENT_INFO_Payment_Info_id,
OLD.PAYMENT_amount, OLD.Payment_date, NOW());
END$$
DELIMITER ;
```

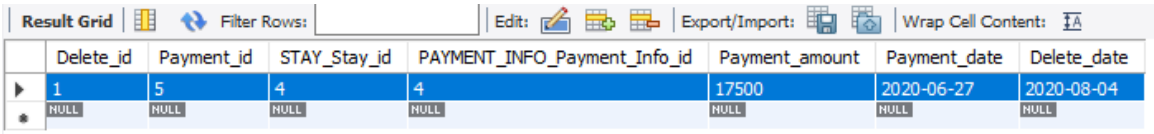
There was another payment added to the table with Payment_id = 5 (highlighted in blue).

```
INSERT INTO TRANSACTION_PAYMENT(Payment_id, STAY_Stay_id,
PAYMENT_INFO_Payment_Info_id, Payment_amount, Payment_date) VALUES
(5,4,4, 17500, '2020-06-27');
SELECT * FROM TRANSACTION_PAYMENT;
```

Payment_id	STAY_Stay_id	PAYMENT_INFO_Payment_Info_id	Payment_amount	Payment_date
1	1	1	1000	2020-02-09 00:00:00
2	2	2	1250	2020-03-12 00:00:00
3	3	3	2250	2020-01-12 00:00:00
4	4	4	3500	2020-01-27 00:00:00
5	4	4	17500	2020-06-27 00:00:00
NULL	NULL	NULL	NULL	NULL

Delete that Payment_id = 5 and the trigger will be called. The table will store the Payment_id = 5 in the backup table Archive_Transaction_Payment (highlighted in blue).

```
DELETE FROM TRANSACTION_PAYMENT
WHERE Payment_id = 5;
SELECT * FROM TRANSACTION_PAYMENT;
SELECT * FROM ARCHIVE_TRANSACTION_PAYMENT;
```



Delete_id	Payment_id	STAY_Stay_id	PAYMENT_INFO_Payment_Info_id	Payment_amount	Payment_date	Delete_date
1	5	4	4	17500	2020-06-27	2020-08-04
NULL	NULL	NULL	NULL	NULL	NULL	NULL

4.3 Queries

The main purpose of the query below is to demonstrate the database capabilities as well as valuable business logics for data analytic purposes.

4.3.1 List of Properties Within a Certain Zip Code

The front-end client can show the users what are the available properties within a certain ZIP_CODE so that the clients can choose from. In the example below, the client wants to find all the PROPERTIES within the zip code of 30067.

```
SELECT PROPERTY.Property_Name, PROPERTY.Street_name
FROM PROPERTY
INNER JOIN ZIP_CODE
ON
PROPERTY.ZIP_CODE_Zip_code_id = ZIP_CODE.Zip_code_id
WHERE PROPERTY.ZIP_CODE_Zip_code_id  = 30067
```

4.3.2 Retrieve the Top Three Host Rating and Their Associated Properties and Information

The guests might be interested in knowing the top three highest star rating HOST so that they can choose PROPERTIES of HOSTS with the best star rating. The query also returns the associated name and email so that the interested users can reach out.

```
SELECT PROPERTY.Property_Name, PROPERTY.Street_name,
PROPERTY.ZIP_CODE_Zip_Code_id, H.Name, H.email, TOP_HOST.SCORE
FROM PROPERTY
INNER JOIN
(SELECT HOST_REVIEW.HOST_Host_id, AVG(HOST_REVIEW.Star_ratings) as
SCORE
FROM HOST_REVIEW
GROUP BY HOST_REVIEW.HOST_Host_id
ORDER BY SCORE DESC LIMIT 3) AS TOP_HOST
ON PROPERTY.HOST_host_id = TOP_HOST.HOST_host_id
INNER JOIN  HOST as H
ON TOP_HOST.HOST_host_id = H.Host_id;
```

4.3.3 Retrieve a List of Unverified Properties and Their Associated Host

PROPERTIES need to be verified. The ‘Property Rentals’ management is interested to know which PROPERTIES are not currently verified and their associated HOSTS. Management can then work with the HOSTS to get the PROPERTIES verified.

```
SELECT PROPERTY.Property_Name, PROPERTY.Street_name, HOST.Name,
HOST.Email
FROM PROPERTY
INNER JOIN PROPERTY_VERIFICATION
ON PROPERTY.Property_id = PROPERTY_VERIFICATION.PROPERTY_Property_id
INNER JOIN HOST
ON PROPERTY.HOST_host_id = HOST.Host_id
WHERE PROPERTY_VERIFICATION.Pass_verification = FALSE;
```

4.3.4 Retrieve All Pending Reservations (Not Converted to a STAY Yet) and the Associated Guests and Properties

It is important for the platform to know which RESERVATIONS have not been converted to a STAY yet so the agents can reach out to the clients to see whether if there is any change in a plan to inform the HOSTS of appropriate outcomes.

```
SELECT R.Reservation_id, G.Name, P.Property_name, P.Street_name, Z.City, Z.State,
Z.Zip_Code_id
FROM RESERVATION AS R
INNER JOIN GUEST AS G
ON R.GUEST_Guest_id = G.Guest_id
INNER JOIN PROPERTY AS P
ON R.PROPERTY_Property_id= P.Property_id
INNER JOIN ZIP_CODE AS Z
ON P.ZIP_CODE_Zip_code_id = Z.Zip_code_id
WHERE Reservation_id NOT IN
(SELECT RESERVATION_Reservation_id
FROM STAY);
```

4.3.5 Retrieve a List of Properties Not Been Booked for a Reservation Yet

The platform can use the above queries to retrieve those PROPERTIES, inspect the reasons for the inactivity, and implement promotions on these PROPERTIES.

```
SELECT P.Property_id, P.Property_name, P.Street_name, Z.City, Z.State,
Z.Zip_Code_id
FROM PROPERTY AS P
LEFT JOIN PROPERTY_VERIFICATION AS PV
ON P.Property_id = PV.PROPERTY_Property_id
LEFT JOIN ZIP_CODE AS Z
ON P.ZIP_CODE_Zip_code_id = Z.Zip_code_id
WHERE P.Property_id
NOT IN
(
SELECT R.PROPERTY_Property_id
FROM RESERVATION AS R
)
AND
PV.Pass_Verification = True;
```

4.3.6 Retrieve Top Three Generating Revenue Properties and Its Associated Hosts

Management team can know which are the highest grossing PROPERTIES and study why they are so successful in attracting sales.

```
SELECT P.Property_id, P.Property_name, P.Street_name, Z.City, Z.State,
Z.Zip_Code_id, SUM(TP.Payment_amount)
FROM TRANSACTION_PAYMENT AS TP
LEFT JOIN STAY AS S
ON TP.STAY_Stay_id = S.Stay_id
LEFT JOIN RESERVATION AS R
ON S.RESERVATION_Reservation_id = R.Reservation_id
LEFT JOIN PROPERTY AS P
ON R.PROPERTY_Property_id = P.Property_id
LEFT JOIN ZIP_CODE AS Z
ON P.ZIP_CODE_Zip_code_id = Z.Zip_code_id
GROUP BY P.Property_id
ORDER BY SUM(TP.Payment_amount) DESC LIMIT 3;
```

4.3.7 Retrieve Customers Name Who Completed a Stay But Has Not Paid Yet Together with the Associated Stay ID, Property Name, and Trip Start and End Date

This query is important for revenue management because we can track down who has not paid for the rental yet despite spending their time on the PROPERTIES.

```
SELECT S.Stay_id, G.Name, P.Property_Name, P.Street_name, Z.City, Z.State,
Z.Zip_Code_id
FROM STAY AS S
LEFT JOIN RESERVATION AS R
ON S.RESERVATION_Reservation_id = R.Reservation_id
LEFT JOIN PROPERTY AS P
ON R.PROPERTY_Property_id = P.Property_id
LEFT JOIN ZIP_CODE AS Z
ON P.ZIP_CODE_Zip_code_id = Z.Zip_code_id
LEFT JOIN GUEST AS G
ON R.GUEST_Guest_id = G.Guest_id
WHERE S.STAY_id
NOT IN
(SELECT TP.STAY_Stay_id
FROM TRANSACTION_PAYMENT AS TP)
AND S.End_date < CURDATE();
```

4.3.8 Retrieve Hosts with the Highest Number of Verified Properties on the Platform

It would be great to know to know which HOSTS have the highest number of verified PROPERTIES on the platform so we can offer special rates in the future. We also want to make those HOSTS happy to continue listing their PROPERTIES on our platform.

```
SELECT P.HOST_Host_id, COUNT(P.PROPERTY_id)
FROM PROPERTY AS P
LEFT JOIN HOST as H
ON P.HOST_Host_id = H.Host_id
LEFT JOIN PROPERTY_VERIFICATION as PV
ON P.Property_id = PV.PROPERTY_Property_id
WHERE PV.Pass_verification = True
GROUP BY P.HOST_Host_id
ORDER BY COUNT(P.PROPERTY_id) DESC LIMIT 5;
```

5. CRUD Matrix

The CRUD Matrix stands for CREATE, UPDATE, and DELETE, which examines the interaction of data and processes.

5.1 List of Entity Types

E1: GUEST
E2: HOST
E3: PROPERTY
E4: ROOM
E5: PHOTO
E6: ZIP_CODE
E7: PROPERTY_VERIFICATION
E8: RESERVATION
E9: STAY
E10: TRANSACTION_PAYMENT
E11: PAYMENT_INFO
E12: GUEST_REVIEW
E13: HOST_REVIEW
E14: PROPERTY_REVIEW
E15: archivedTransactionsPayment

5.2 List of Functions

F1: Insert/update/retrieve a GUEST
F2: Insert/update/retrieve a HOST
F3: Insert/update/delete/retrieve a PROPERTY
F4: Insert/update/delete/retrieve a ROOM
F5: Insert/update/delete/retrieve a PHOTO
F6: Insert/update /retrieve a ZIP_CODE
F7: Insert/update/delete/retrieve a PROPERTY_VERIFICATION
F8: Insert/update retrieve a RESERVATION

F9: Insert/update/delete/retrieve a STAY
F10: Insert/update/delete/retrieve a TRANSACTION_PAYMENT
F11: Insert/update/retrieve a PAYMENT_INFO
F12: Insert/update/delete/retrieve a GUEST_REVIEW
F13: Insert/update/delete/retrieve a HOST_REVIEW
F14: Insert/update/delete/retrieve a PROPERTY_REVIEW

Function/Entity Interaction	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15
F1	CRU														
F2		CRU													
F3			CRU												
F4				CRUD	D										
F5					CRUD										
F6						CRU									
F7							CRUD								
F8								CRU							
F9									CRUD	D					
F10										CRUD					U
F11											CRU				
F12												CRUD			
F13													CRUD		
F14														CRUD	

For the CRUD Matrix above, we cannot delete a GUEST, HOST, PROPERTY, or RESERVATION from our database; we can only make GUEST and HOST to be inactive to i) maintain the information asset of our platform and ii) make sure that our children don't get deleted with the parents entity getting deleted (i.e. children entities of Host include PROPERTY).

If the users ask for their information to be deleted, we can set their status to be inactive. In addition, if the PROPERTY should not be listed on the site, we can set the property verification status to be false so that it will not be shown on the site.

6. Concluding Remark

The 'Property Rentals' database is a well-structured and comprehensive database. The constructed database, for the most part, accurately depicts the more common entities and database operations which a property rental platform is going to perform. The database prefers the implementation of non-null foreign keys instead of composite keys, which increase the database performance. The database adheres to third normal form and there should be no duplicate data among relations. One of the database weakness is the lack of lookup tables to instantly fetch certain calculations (i.e. top 3 power users). However, this might include duplicated data. If I have had more time, I would implement a web-based client to query the database (i.e. query for list of properties) and demonstrate the database capabilities to be used in a client-server setting.

Lesson learned is that the planning and designing phase is critical to the sucess of a database project. Without careful planning and designing phase, we might miss out on entities & relationships and will take a lot of times for database revisions in the middle of a project. In addition, there are also a variety of equivalent ways to implement a database

(i.e. usage of composite keys and non-null foreign keys). Furthermore, I also learn that we do not normally delete a user in a database given that users' information is Company's asset; we just render that user to be "Inactive" so that we can filter out deactivated users from our database for certain calculations and displays. Lastly, I also learn how to create an audit trail to archive deleted data for regulatory and compliance reasons.

Appendix A - DDL, INSERT, SELECT Statements

CREATE statements for creating database objects; INSERT statements to populate test data into the database; SELECT statements to display the test data

Appendix B - Data Dictionary Index

Index to the data dictionary (e.g., column_name with primary keys arranged first and table_names)

Appendix C – Enlarged Entities Relationship Diagram

References

"MySQL Enterprise Monitor 4.0 Manual: 3.2.1 System Requirements." *MySQL*, dev.mysql.com/doc/mysql-monitor/4.0/en/system-prereqs-reference.html.

"Chapter 7 Backup and Recovery." *MySQL*, dev.mysql.com/doc/refman/8.0/en/backup-and-recovery.html.

"Chapter 3. Installing and Launching MySQL Workbench", docs.oracle.com/cd/E19078-01/mysql/mysql-workbench/wb-installing.html.

Appendix A - DDL, INSERT, SELECT Statements

```
-- MySQL Workbench Forward Engineering
-- CREATE TABLE

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN
_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUB
STITUTION';

-- -----------------------------------------------------
-- Schema PROPERTY_RENTALS
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema PROPERTY_RENTALS
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `PROPERTY_RENTALS` DEFAULT
CHARACTER SET utf8 ;
USE `PROPERTY_RENTALS` ;

-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`GUEST`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`GUEST` (
  `Guest_id` INT UNSIGNED NOT NULL,
  `Name` VARCHAR(45) NOT NULL,
  `Birthdate` DATETIME NULL,
  `Email` VARCHAR(45) NOT NULL,
  `Username` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `Active` TINYINT NULL DEFAULT 1,
  PRIMARY KEY (`Guest_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`HOST`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`HOST` (
  `Host_id` INT UNSIGNED NOT NULL,
  `Name` VARCHAR(45) NOT NULL,
  `Birthdate` VARCHAR(45) NULL,
```

```
  `Email` VARCHAR(45) NOT NULL,
  `Username` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `Active` TINYINT NULL DEFAULT 1,
  PRIMARY KEY (`Host_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`ZIP_CODE`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`ZIP_CODE` (
  `Zip_code_id` INT UNSIGNED NOT NULL,
  `City` VARCHAR(45) NOT NULL,
  `State` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Zip_code_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`PROPERTY`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`PROPERTY` (
  `Property_id` INT UNSIGNED NOT NULL,
  `HOST_Host_id` INT UNSIGNED NOT NULL,
  `ZIP_CODE_Zip_code_id` INT UNSIGNED NOT NULL,
  `Property_name` VARCHAR(45) NOT NULL,
  `Street_name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Property_id`),
  INDEX `fk_PROPERTY_HOST1_idx` (`HOST_Host_id` ASC) VISIBLE,
  INDEX `fk_PROPERTY_ZIP_CODE1_idx` (`ZIP_CODE_Zip_code_id` ASC)
VISIBLE,
  CONSTRAINT `fk_PROPERTY_HOST1`
    FOREIGN KEY (`HOST_Host_id`)
    REFERENCES `PROPERTY_RENTALS`.`HOST` (`Host_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_PROPERTY_ZIP_CODE1`
    FOREIGN KEY (`ZIP_CODE_Zip_code_id`)
    REFERENCES `PROPERTY_RENTALS`.`ZIP_CODE` (`Zip_code_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
```

```
-- Table `PROPERTY_RENTALS`.`RESERVATION`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`RESERVATION` (
  `Reservation_id` VARCHAR(45) NOT NULL,
  `GUEST_Guest_id` INT UNSIGNED NOT NULL,
  `PROPERTY_Property_id` INT UNSIGNED NOT NULL,
  `Start_date` DATETIME NOT NULL,
  `End_date` DATETIME NOT NULL,
  `Rental_Purposes` VARCHAR(45) NULL DEFAULT 'Others',
  PRIMARY KEY (`Reservation_id`),
  INDEX `fk_RESERVATION_GUEST_idx` (`GUEST_Guest_id` ASC) VISIBLE,
  INDEX `fk_RESERVATION_PROPERTY1_idx` (`PROPERTY_Property_id` ASC)
VISIBLE,
  UNIQUE INDEX `Reservation_id_UNIQUE` (`Reservation_id` ASC) VISIBLE,
  CONSTRAINT `fk_RESERVATION_GUEST`
    FOREIGN KEY (`GUEST_Guest_id`)
    REFERENCES `PROPERTY_RENTALS`.`GUEST` (`Guest_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_RESERVATION_PROPERTY1`
    FOREIGN KEY (`PROPERTY_Property_id`)
    REFERENCES `PROPERTY_RENTALS`.`PROPERTY` (`Property_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`STAY`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`STAY` (
  `Stay_id` INT UNSIGNED NOT NULL,
  `RESERVATION_Reservation_id` VARCHAR(45) NOT NULL,
  `Start_date` DATETIME NOT NULL,
  `End_date` DATETIME NOT NULL,
  PRIMARY KEY (`Stay_id`),
  INDEX `fk_STAY_RESERVATION1_idx` (`RESERVATION_Reservation_id` ASC)
VISIBLE,
  CONSTRAINT `fk_STAY_RESERVATION1`
    FOREIGN KEY (`RESERVATION_Reservation_id`)
    REFERENCES `PROPERTY_RENTALS`.`RESERVATION` (`Reservation_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`PAYMENT_INFO`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`PAYMENT_INFO` (
  `Payment_Info_id` INT UNSIGNED NOT NULL,
  `GUEST_Guest_id` INT UNSIGNED NOT NULL,
  `Credit_Card_Number` VARCHAR(45) NULL,
  `Debt_Card_Number` VARCHAR(45) NULL,
  `Expiration_date` DATE NULL,
  `CVV` INT NULL,
  PRIMARY KEY (`Payment_Info_id`),
  INDEX `fk_PAYMENT_INFO_GUEST1_idx` (`GUEST_Guest_id` ASC) VISIBLE,
  CONSTRAINT `fk_PAYMENT_INFO_GUEST1`
    FOREIGN KEY (`GUEST_Guest_id`)
    REFERENCES `PROPERTY_RENTALS`.`GUEST` (`Guest_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`TRANSACTION_PAYMENT`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS
`PROPERTY_RENTALS`.`TRANSACTION_PAYMENT` (
  `Payment_id` INT UNSIGNED NOT NULL,
  `STAY_Stay_id` INT UNSIGNED NOT NULL,
  `PAYMENT_INFO_Payment_Info_id` INT UNSIGNED NOT NULL,
  `Payment_amount` FLOAT UNSIGNED NULL DEFAULT 0,
  `Payment_date` DATETIME NULL,
  PRIMARY KEY (`Payment_id`),
  INDEX `fk_TRANSACTION_PAYMENT_STAY1_idx` (`STAY_Stay_id` ASC)
VISIBLE,
  INDEX `fk_TRANSACTION_PAYMENT_PAYMENT_INFO1_idx`
(`PAYMENT_INFO_Payment_Info_id` ASC) VISIBLE,
  CONSTRAINT `fk_TRANSACTION_PAYMENT_STAY1`
    FOREIGN KEY (`STAY_Stay_id`)
    REFERENCES `PROPERTY_RENTALS`.`STAY` (`Stay_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_TRANSACTION_PAYMENT_PAYMENT_INFO1`
    FOREIGN KEY (`PAYMENT_INFO_Payment_Info_id`)
    REFERENCES `PROPERTY_RENTALS`.`PAYMENT_INFO` (`Payment_Info_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`HOST_REVIEW`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`HOST_REVIEW` (
  `Host_Review_id` INT UNSIGNED NOT NULL,
  `GUEST_Guest_id` INT UNSIGNED NOT NULL,
  `HOST_Host_id` INT UNSIGNED NOT NULL,
  `Reviews` VARCHAR(250) NULL DEFAULT 'Default - No Comment',
  `Star_ratings` INT NOT NULL,
  INDEX `fk_HOST_REVIEW_HOST1_idx` (`HOST_Host_id` ASC) VISIBLE,
  PRIMARY KEY (`Host_Review_id`),
  CONSTRAINT `fk_HOST_REVIEW_GUEST1`
    FOREIGN KEY (`GUEST_Guest_id`)
    REFERENCES `PROPERTY_RENTALS`.`GUEST` (`Guest_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_HOST_REVIEW_HOST1`
    FOREIGN KEY (`HOST_Host_id`)
    REFERENCES `PROPERTY_RENTALS`.`HOST` (`Host_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`ROOM`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`ROOM` (
  `Room_id` INT UNSIGNED NOT NULL,
  `PROPERTY_Property_id` INT UNSIGNED NOT NULL,
  `Smoking` TINYINT NULL DEFAULT 1,
  `Room_type` VARCHAR(45) NULL DEFAULT 'General Purposes',
  PRIMARY KEY (`Room_id`, `PROPERTY_Property_id`),
  INDEX `fk_ROOM_PROPERTY1_idx` (`PROPERTY_Property_id` ASC) VISIBLE,
  CONSTRAINT `fk_ROOM_PROPERTY1`
    FOREIGN KEY (`PROPERTY_Property_id`)
    REFERENCES `PROPERTY_RENTALS`.`PROPERTY` (`Property_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`PHOTO`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`PHOTO` (
```

```
  `Photo_id` INT UNSIGNED NOT NULL,
  `ROOM_Room_id` INT UNSIGNED NOT NULL,
  `ROOM_PROPERTY_Property_id` INT UNSIGNED NOT NULL,
  `Taken_when` DATETIME NULL,
  `Photo` BLOB NOT NULL,
  PRIMARY KEY (`Photo_id`, `ROOM_Room_id`,
`ROOM_PROPERTY_Property_id`),
  INDEX `fk_PHOTO_ROOM1_idx` (`ROOM_Room_id` ASC,
`ROOM_PROPERTY_Property_id` ASC) VISIBLE,
  CONSTRAINT `fk_PHOTO_ROOM1`
    FOREIGN KEY (`ROOM_Room_id` , `ROOM_PROPERTY_Property_id`)
    REFERENCES `PROPERTY_RENTALS`.`ROOM` (`Room_id` ,
`PROPERTY_Property_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`PROPERTY_REVIEW`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`PROPERTY_REVIEW`
(
  `Property_Review_id` INT UNSIGNED NOT NULL,
  `GUEST_Guest_id` INT UNSIGNED NOT NULL,
  `PROPERTY_Property_id` INT UNSIGNED NOT NULL,
  `Star_ratings` INT NOT NULL,
  `Reviews` VARCHAR(250) NULL DEFAULT 'Default - No Comment',
  INDEX `fk_PROPERTY_REVIEW_PROPERTY1_idx` (`PROPERTY_Property_id`
ASC) VISIBLE,
  PRIMARY KEY (`Property_Review_id`),
  CONSTRAINT `fk_PROPERTY_REVIEW_GUEST1`
    FOREIGN KEY (`GUEST_Guest_id`)
    REFERENCES `PROPERTY_RENTALS`.`GUEST` (`Guest_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_PROPERTY_REVIEW_PROPERTY1`
    FOREIGN KEY (`PROPERTY_Property_id`)
    REFERENCES `PROPERTY_RENTALS`.`PROPERTY` (`Property_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`GUEST_REVIEW`
```

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `PROPERTY_RENTALS`.`GUEST_REVIEW` (
  `Guest_Review_id` INT UNSIGNED NOT NULL,
  `GUEST_Guest_id` INT UNSIGNED NOT NULL,
  `HOST_Host_id` INT UNSIGNED NOT NULL,
  `Star_ratings` INT NOT NULL,
  `Reviews` VARCHAR(250) NULL DEFAULT 'Default - No Comment',
  PRIMARY KEY (`Guest_Review_id`),
  INDEX `fk_GUEST_REVIEW_GUEST1_idx` (`GUEST_Guest_id` ASC) VISIBLE,
  INDEX `fk_GUEST_REVIEW_HOST1_idx` (`HOST_Host_id` ASC) VISIBLE,
  CONSTRAINT `fk_GUEST_REVIEW_GUEST1`
    FOREIGN KEY (`GUEST_Guest_id`)
    REFERENCES `PROPERTY_RENTALS`.`GUEST` (`Guest_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_GUEST_REVIEW_HOST1`
    FOREIGN KEY (`HOST_Host_id`)
    REFERENCES `PROPERTY_RENTALS`.`HOST` (`Host_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `PROPERTY_RENTALS`.`PROPERTY_VERIFICATION`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS
`PROPERTY_RENTALS`.`PROPERTY_VERIFICATION` (
  `VERIFICATION_id` INT UNSIGNED NOT NULL,
  `PROPERTY_Property_id` INT UNSIGNED NOT NULL,
  `Verification_date` DATETIME NOT NULL,
  `Pass_verification` TINYINT NOT NULL,
  PRIMARY KEY (`VERIFICATION_id`),
  INDEX `fk_PROPERTY_VERIFICATION_PROPERTY1_idx`
(`PROPERTY_Property_id` ASC) VISIBLE,
  CONSTRAINT `fk_PROPERTY_VERIFICATION_PROPERTY1`
    FOREIGN KEY (`PROPERTY_Property_id`)
    REFERENCES `PROPERTY_RENTALS`.`PROPERTY` (`Property_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

```
-- INSERT STATEMENT

USE PROPERTY_RENTALS;

-- Insert new Guests
INSERT INTO GUEST(Guest_id, Name, BirthDate, Email, Username, Password,
Active) VALUES
(1,'Michael Winchester','1975-07-10','michaelwinchester@gmail.com',
'mihchaelwinchester','IamMichaelWinchester', True);
INSERT INTO GUEST(Guest_id, Name, BirthDate, Email, Username, Password,
Active) VALUES
(2,'Jordan Olla','1991-06-02','jordanolla@gmail.com', 'jordanolla','IamJordanOlla', True);
INSERT INTO GUEST(Guest_id, Name, BirthDate, Email, Username, Password,
Active) VALUES
(3,'Alex Chen','1986-06-12','alexchen@gmail.com', 'alexchen','IamAlexChen', True);
INSERT INTO GUEST(Guest_id, Name, BirthDate, Email, Username, Password,
Active) VALUES
(4,'Dominic Cain','1982-02-12','dominiccain@gmail.com',
'dominiccain','IamDominicCain', True);
INSERT INTO GUEST(Guest_id, Name, BirthDate, Email, Username, Password,
Active) VALUES
(5,'Virginia Lyons','1993-03-18','virginialyons@gmail.com',
'virginialyons','IamVirginiaLyons', True);

-- Insert new Guest Payment Information
INSERT INTO PAYMENT_INFO(Payment_Info_id, GUEST_Guest_id,
Credit_card_number, Expiration_date, CVV) VALUES
(1,1,'1000-1000-1000-1000','2025-01-01', 655);
INSERT INTO PAYMENT_INFO(Payment_Info_id, GUEST_Guest_id,
Credit_card_number, Expiration_date, CVV) VALUES
(2,2,'3001-1000-3001-3001','2025-11-12', 300);
INSERT INTO PAYMENT_INFO(Payment_Info_id, GUEST_Guest_id,
Credit_card_number, Expiration_date, CVV) VALUES
(3,3,'4550-5550-4550-4550','2021-10-15', 444);
INSERT INTO PAYMENT_INFO(Payment_Info_id, GUEST_Guest_id,
Credit_card_number, Expiration_date, CVV) VALUES
(4,4,'5550-5550-5550-5550','2023-10-15', 678);
INSERT INTO PAYMENT_INFO(Payment_Info_id, GUEST_Guest_id,
Credit_card_number, Expiration_date, CVV) VALUES
(5,5,'6780-5550-5550-6780','2022-04-06', 745);

-- Insert new Host
INSERT INTO HOST(Host_id, Name, BirthDate, Email, Username, Password, Active)
VALUES
```

```
(1,'James Keller','1990-11-10','jameskeller@gmail.com', 'jameskeller','IamJamesKeller',
True);
INSERT INTO HOST(Host_id, Name, BirthDate, Email, Username, Password, Active)
VALUES
(2,'Michael Jones','1980-10-10','michaeljones@gmail.com',
'michaeljones','IamMichaelJones', True);
INSERT INTO HOST(Host_id, Name, BirthDate, Email, Username, Password, Active)
VALUES
(3,'Allie Johnson','1985-09-02','alliejohnson@gmail.com',
'alliejohnson','IamAllieJohnson', True);
INSERT INTO HOST(Host_id, Name, BirthDate, Email, Username, Password, Active)
VALUES
(4,'Jackson Williams','1987-01-
01','jacksonwilliams@gmail.com','jacksonwilliams','IamJacksonWilliams', True);
INSERT INTO HOST(Host_id, Name, BirthDate, Email, Username, Password, Active)
VALUES
(5,'Joshua Stach','1987-03-22','joshuastach@gmail.com','joshuastach','IamJoshuaStach',
True);

-- Insert new Zip Code, will need to have bigger list of zip codes; this is just for demo
INSERT INTO ZIP_CODE(Zip_code_id, City, State) VALUES
(30067,'Marietta','Georgia');
INSERT INTO ZIP_CODE(Zip_code_id, City, State) VALUES
(10010,'Mahattan','New York');
INSERT INTO ZIP_CODE(Zip_code_id, City, State) VALUES
(90009,'Los Angeles','California');
INSERT INTO ZIP_CODE(Zip_code_id, City, State) VALUES
(21201,'Baltimore','Maryland');
INSERT INTO ZIP_CODE(Zip_code_id, City, State) VALUES
(33101,'Miami','Florida');

-- Insert new Properties owned by Hosts
INSERT INTO PROPERTY(Property_id, HOST_Host_id, ZIP_CODE_Zip_code_id,
Property_name, Street_name) VALUES
(1,1,30067,'House 1', '200 Madison');
INSERT INTO PROPERTY(Property_id, HOST_Host_id, ZIP_CODE_Zip_code_id,
Property_name, Street_name) VALUES
(2,2,10010,'House 2', '300 Vanderbilt');
INSERT INTO PROPERTY(Property_id, HOST_Host_id, ZIP_CODE_Zip_code_id,
Property_name, Street_name) VALUES
(3,3,90009,'House 3', '400 Powers Ferry');
INSERT INTO PROPERTY(Property_id, HOST_Host_id, ZIP_CODE_Zip_code_id,
Property_name, Street_name) VALUES
(4,4,21201,'House 4', '500 South Drive');
INSERT INTO PROPERTY(Property_id, HOST_Host_id, ZIP_CODE_Zip_code_id,
Property_name, Street_name) VALUES
```

```
(5,5,33101,'House 5', '600 North Drive');
INSERT INTO PROPERTY(Property_id, HOST_Host_id, ZIP_CODE_Zip_code_id,
Property_name, Street_name) VALUES
(6,5,33101,'House 6', '800 West Drive');


-- Insert new Rooms within each Property
INSERT INTO ROOM(Room_id, PROPERTY_Property_id, Smoking, Room_type)
VALUES
(1, 2, True, 'Bedroom');
INSERT INTO ROOM(Room_id, PROPERTY_Property_id, Smoking, Room_type)
VALUES
(2, 2, True, 'Library');
INSERT INTO ROOM(Room_id, PROPERTY_Property_id, Smoking, Room_type)
VALUES
(3, 3, False, 'Kitchen');
INSERT INTO ROOM(Room_id, PROPERTY_Property_id, Smoking, Room_type)
VALUES
(4, 4, False, 'Lounge');
INSERT INTO ROOM(Room_id, PROPERTY_Property_id, Smoking, Room_type)
VALUES
(5, 4, False, 'Bedroom');

-- Insert new Property Verification, need to check if there is multiple verification date
INSERT INTO PROPERTY_VERIFICATION(Verification_id,
PROPERTY_Property_id, Verification_Date, Pass_Verification) VALUES
(1, 1, '2020-01-01', True);
INSERT INTO PROPERTY_VERIFICATION(Verification_id,
PROPERTY_Property_id, Verification_Date, Pass_Verification) VALUES
(2, 2, '2020-02-01', True);
INSERT INTO PROPERTY_VERIFICATION(Verification_id,
PROPERTY_Property_id, Verification_Date, Pass_Verification) VALUES
(3, 3, '2020-03-01', False);
INSERT INTO PROPERTY_VERIFICATION(Verification_id,
PROPERTY_Property_id, Verification_Date, Pass_Verification) VALUES
(4, 4, '2020-02-01', True);
INSERT INTO PROPERTY_VERIFICATION(Verification_id,
PROPERTY_Property_id, Verification_Date, Pass_Verification) VALUES
(5, 5, '2020-04-01', False);
INSERT INTO PROPERTY_VERIFICATION(Verification_id,
PROPERTY_Property_id, Verification_Date, Pass_Verification) VALUES
(6, 6, '2020-04-01', True);


-- Insert new Reservation
```

```
INSERT INTO RESERVATION(Reservation_id, GUEST_Guest_id,
PROPERTY_Property_id, Start_date, End_date, Rental_Purposes) VALUES
(1,1,4, '2020-02-01', '2020-02-07','Vacationing');
INSERT INTO RESERVATION(Reservation_id, GUEST_Guest_id,
PROPERTY_Property_id, Start_date, End_date, Rental_Purposes) VALUES
(2,2,3, '2020-03-03', '2020-03-10','Vacationing');
INSERT INTO RESERVATION(Reservation_id, GUEST_Guest_id,
PROPERTY_Property_id, Start_date, End_date, Rental_Purposes) VALUES
(3,3,2, '2020-01-02', '2020-01-10','Working');
INSERT INTO RESERVATION(Reservation_id, GUEST_Guest_id,
PROPERTY_Property_id, Start_date, End_date, Rental_Purposes) VALUES
(4,4,1, '2020-01-02', '2020-01-10','Working');
INSERT INTO RESERVATION(Reservation_id, GUEST_Guest_id,
PROPERTY_Property_id, Start_date, End_date, Rental_Purposes) VALUES
(5,5,5, '2020-01-22', '2020-01-29','Others');
INSERT INTO RESERVATION(Reservation_id, GUEST_Guest_id,
PROPERTY_Property_id, Start_date, End_date, Rental_Purposes) VALUES
(6,1,2, '2020-03-05', '2020-03-08','Vacationing');
INSERT INTO RESERVATION(Reservation_id, GUEST_Guest_id,
PROPERTY_Property_id, Start_date, End_date, Rental_Purposes) VALUES
(7,1,3, '2020-06-05', '2020-06-09','Others');

-- Insert new Stay as a Reservation is converted into a STAY; Stay start date might be
different from End_date
INSERT INTO STAY(Stay_id, RESERVATION_Reservation_id, Start_date, End_date)
VALUES
(1,1,'2020-02-02', '2020-02-08');
INSERT INTO STAY(Stay_id, RESERVATION_Reservation_id, Start_date, End_date)
VALUES
(2,2,'2020-03-04', '2020-03-11');
INSERT INTO STAY(Stay_id, RESERVATION_Reservation_id, Start_date, End_date)
VALUES
(3,3,'2020-01-02', '2020-01-10');
INSERT INTO STAY(Stay_id, RESERVATION_Reservation_id, Start_date, End_date)
VALUES
(4,4,'2020-01-20', '2020-01-25');
INSERT INTO STAY(Stay_id, RESERVATION_Reservation_id, Start_date, End_date)
VALUES
(5,6,'2020-03-05', '2020-03-08');

-- Insert new Transaction Payment
INSERT INTO TRANSACTION_PAYMENT(Payment_id, STAY_Stay_id,
PAYMENT_INFO_Payment_Info_id, Payment_amount, Payment_date) VALUES
(1,1,1, 1000,'2020-02-09');
INSERT INTO TRANSACTION_PAYMENT(Payment_id, STAY_Stay_id,
PAYMENT_INFO_Payment_Info_id, Payment_amount, Payment_date) VALUES
```

```
(2,2,2, 1250,'2020-03-12');
INSERT INTO TRANSACTION_PAYMENT(Payment_id, STAY_Stay_id,
PAYMENT_INFO_Payment_Info_id, Payment_amount, Payment_date) VALUES
(3,3,3, 2250, '2020-01-12');
INSERT INTO TRANSACTION_PAYMENT(Payment_id, STAY_Stay_id,
PAYMENT_INFO_Payment_Info_id, Payment_amount, Payment_date) VALUES
(4,4,4, 3500, '2020-01-27');


-- Insert new Host Review
INSERT INTO HOST_REVIEW(Host_Review_id, GUEST_Guest_id, HOST_Host_id,
Reviews, Star_Ratings) VALUES
(1,1,4,"He is such a fantastic host. It was a pleasure residing at his property", 10);
INSERT INTO HOST_REVIEW(Host_Review_id, GUEST_Guest_id, HOST_Host_id,
Reviews, Star_Ratings) VALUES
(2,2,3,"He could keep his property tidier. The condition of the room is not very good.",
6);
INSERT INTO HOST_REVIEW(Host_Review_id, GUEST_Guest_id, HOST_Host_id,
Reviews, Star_Ratings) VALUES
(3,3,2,"He is pleasant to work with. He is on time but was for me late to return the keys at
the end.", 8);
INSERT INTO HOST_REVIEW(Host_Review_id, GUEST_Guest_id, HOST_Host_id,
Reviews, Star_Ratings) VALUES
(4,4,1,"It was a surprisingly good experience. I would recommend him again for future
stay.", 9);

-- Insert new Guest Review
INSERT INTO GUEST_REVIEW(Guest_Review_id, GUEST_Guest_id,
HOST_Host_id, Reviews, Star_Ratings) VALUES
(1,1,4,"It was a pleasure to host Michael. He kept the place very clean.", 9);
INSERT INTO GUEST_REVIEW(Guest_Review_id, GUEST_Guest_id,
HOST_Host_id, Reviews, Star_Ratings) VALUES
(2,2,3,"It was a good experience hosting Jordan. He is a bit late to our first meeting but it
works out in the end. ", 8);

-- Insert new Property Review
INSERT INTO PROPERTY_REVIEW(Property_Review_id, GUEST_Guest_id,
PROPERTY_Property_id, Reviews, Star_Ratings) VALUES
(1,1,4,"It was a very nice and clean property with excellent location to nearby
amenities.", 9);
INSERT INTO PROPERTY_REVIEW(Property_Review_id, GUEST_Guest_id,
PROPERTY_Property_id, Reviews, Star_Ratings) VALUES
(2,2,3,"It was an ok property to stay in. I have no negative opinions of the property.", 7);

-- SELECT STATEMENT
```

```
-- List of Properties Within a Certain Zip Code
SELECT PROPERTY.Property_Name, PROPERTY.Street_name
FROM PROPERTY
INNER JOIN ZIP_CODE
ON
PROPERTY.ZIP_CODE_Zip_code_id = ZIP_CODE.Zip_code_id
WHERE PROPERTY.ZIP_CODE_Zip_code_id  = 30067;

-- Retrieve the Top 3 Host Rating and Their Associated Properties and Information
SELECT PROPERTY.Property_Name, PROPERTY.Street_name,
PROPERTY.ZIP_CODE_Zip_Code_id, H.Name, H.email, TOP_HOST.SCORE
FROM PROPERTY
INNER JOIN
(SELECT HOST_REVIEW.HOST_Host_id, AVG(HOST_REVIEW.Star_ratings) as
SCORE
FROM HOST_REVIEW
GROUP BY HOST_REVIEW.HOST_Host_id
ORDER BY SCORE DESC LIMIT 3) AS TOP_HOST
ON PROPERTY.HOST_host_id = TOP_HOST.HOST_host_id
INNER JOIN  HOST as H
ON TOP_HOST.HOST_host_id = H.Host_id;

-- Retrieve a List of Unverified Properties and Their Associated Host
SELECT PROPERTY.Property_Name, PROPERTY.Street_name, HOST.Name,
HOST.Email
FROM PROPERTY
INNER JOIN PROPERTY_VERIFICATION
ON PROPERTY.Property_id =
PROPERTY_VERIFICATION.PROPERTY_Property_id
INNER JOIN HOST
ON PROPERTY.HOST_host_id = HOST.Host_id
WHERE PROPERTY_VERIFICATION.Pass_verification = FALSE;

-- Retrieve All Pending Reservations (Not Converted to a STAY Yet) and the Associated
Guests and Properties
SELECT R.Reservation_id, G.Name, P.Property_name, P.Street_name, Z.City, Z.State,
Z.Zip_Code_id
FROM RESERVATION AS R
INNER JOIN GUEST AS G
ON R.GUEST_Guest_id = G.Guest_id
INNER JOIN PROPERTY AS P
ON R.PROPERTY_Property_id= P.Property_id
INNER JOIN ZIP_CODE AS Z
ON P.ZIP_CODE_Zip_code_id = Z.Zip_code_id
WHERE Reservation_id NOT IN
(SELECT RESERVATION_Reservation_id
```

```
FROM STAY);

-- Retrieve a List of Properties Not Been Booked for a Reservation Yet
SELECT P.Property_id, P.Property_name, P.Street_name, Z.City, Z.State,
Z.Zip_Code_id
FROM PROPERTY AS P
LEFT JOIN PROPERTY_VERIFICATION AS PV
ON P.Property_id = PV.PROPERTY_Property_id
LEFT JOIN ZIP_CODE AS Z
ON P.ZIP_CODE_Zip_code_id = Z.Zip_code_id
WHERE P.Property_id
NOT IN
(
SELECT R.PROPERTY_Property_id
FROM RESERVATION AS R
)
AND
PV.Pass_Verification = True;

-- Retrieve Top Three Generating Revenue Properties and Its Associated Hosts
SELECT P.Property_id, P.Property_name, P.Street_name, Z.City, Z.State,
Z.Zip_Code_id, SUM(TP.Payment_amount)
FROM TRANSACTION_PAYMENT AS TP
LEFT JOIN STAY AS S
ON TP.STAY_Stay_id = S.Stay_id
LEFT JOIN RESERVATION AS R
ON S.RESERVATION_Reservation_id = R.Reservation_id
LEFT JOIN PROPERTY AS P
ON R.PROPERTY_Property_id = P.Property_id
LEFT JOIN ZIP_CODE AS Z
ON P.ZIP_CODE_Zip_code_id = Z.Zip_code_id
GROUP BY P.Property_id
ORDER BY SUM(TP.Payment_amount) DESC LIMIT 3;

-- Retrieve Customers Name Who Completed a Stay But Has Not Paid Yet Together with
the Associated Stay ID, Property Name, and Trip Start and End Date
SELECT S.Stay_id, G.Name, P.Property_Name, P.Street_name, Z.City, Z.State,
Z.Zip_Code_id
FROM STAY AS S
LEFT JOIN RESERVATION AS R
ON S.RESERVATION_Reservation_id = R.Reservation_id
LEFT JOIN PROPERTY AS P
ON R.PROPERTY_Property_id = P.Property_id
LEFT JOIN ZIP_CODE AS Z
ON P.ZIP_CODE_Zip_code_id = Z.Zip_code_id
LEFT JOIN GUEST AS G
```

```
ON R.GUEST_Guest_id = G.Guest_id
WHERE S.STAY_id
NOT IN
(SELECT TP.STAY_Stay_id
FROM TRANSACTION_PAYMENT AS TP)
AND S.End_date < CURDATE();

-- Retrieve Hosts with the Highest Number of Verified Properties on the Platform
SELECT P.HOST_Host_id, COUNT(P.PROPERTY_id)
FROM PROPERTY AS P
LEFT JOIN HOST as H
ON P.HOST_Host_id = H.Host_id
LEFT JOIN PROPERTY_VERIFICATION as PV
ON P.Property_id = PV.PROPERTY_Property_id
WHERE PV.Pass_verification = True
GROUP BY P.HOST_Host_id
ORDER BY COUNT(P.PROPERTY_id) DESC LIMIT 5;
```

Appendix B - Data Dictionary Index

GUEST

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Guest_id	Index of Guest	INT	4 bytes	Primary Key	Y	Greater than zero
Name	Name of the guest	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Birthdate	Birthdate of the guest	DATETIME	8 bytes	None	N	After 1/1/1900
Email	Email of the guest	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Username	Username of the guest	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Password	Password of the guest	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Active	Whether the guest is active?	TINYINT	1 byte	None	N	True or False

HOST

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Host_id	Index of Guest	INT	4 bytes	Primary Key	Y	Greater than zero
Name	Name of the host	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Birthdate	Birthdate of the host	DATETIME	8 bytes	None	N	After 1/1/1900
Email	Email of the host	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Username	Username of the host	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Password	Password of the host	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Active	Whether the host is active?	TINYINT	1 byte	None	N	True or False

RESERVATION

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Reservation_id	Index	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key referencing Guest	Y	Greater than zero
Property_id	Property Identification	INT	4 bytes	Foreign Key referencing Property	Y	Greater than zero
Start_date	Reservation Start Date	DATETIME	8 bytes	None	Y	After 1/1/1900
End_date	Reservation End Date	DATETIME	8 bytes	None	Y	End_date > Start_date

STAY

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Stay_id	Index	INT	4 bytes	Primary Key	Y	Greater than zero
Reservation_id	Reservation Identification	INT	4 bytes	Foreign Key Referencing Reservation	Y	Greater than zero
Start_date	Stay Start Date	DATETIME	8 bytes	None	Y	After 1/1/1900
End_date	Stay End Date	DATETIME	8 bytes	None	Y	End_date > Start_date

TRANSACTION_PAYMENT

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Payment_id	Index	INT	4 bytes	Primary Key	Y	Greater than zero
Stay_id	Stay Identification	INT	4 bytes	Foreign Key Referencing Stay	Y	Greater than zero
Payment_Info	Payment_Info Identification	INT	4 bytes	Foreign Key Referencing Payment_Info	Y	Greater than zero
Payment_amount	Transacted payment amount	FLOAT	4 bytes	None	N	Greater than zero
Payment_Date	Date of Payment	DATETIME	8 bytes	None	N	After 1/1/1900

PAYMENT_INFO

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Payment_Info_id	Index	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key Refering Guest	Y	Greater than zero
Credit_card_number	Credit Card Number	VARCHAR	45 bytes	None	N	Maximum 45 characters
Debit_card_number	Debit Card Number	VARCHAR	45 bytes	None	N	Maximum 45 characters
Expiration Date	Debit/Credit Card Expiration Date	DATETIME	8 bytes	None	N	After 1/1/1900
CVV	Numbers at the back of card	INT	4 bytes	None	N	Greater than zero

HOST_REVIEW

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Host_review_id	Index of Property Review	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key referencing Guest	Y	Greater than zero
Host_id	Host Identification	INT	4 bytes	Foreign Key referencing Host	Y	Greater than zero
Star Ratings	Rating of Host	INT	4 bytes	None	Y	Between 1 to 10
Reviews	Review of the Host	VARCHAR	250 bytes	None	Y	Maximum 250 characters

PROPERTY REVIEW

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Property_Review_id	Index of Property Review	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key referencing Guest	Y	Greater than zero
Property_id	Index of Property	INT	4 bytes	Foreign Key referencing Property	Y	Greater than zero
Star Ratings	Rating of Property	INT	4 bytes	None	Y	Between 1 to 10
Reviews	Review of the Property	VARCHAR	250 bytes	None	Y	Maximum 250 characters

GUEST_REVIEW

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Guest_review_id	Index of Property Review	INT	4 bytes	Primary Key	Y	Greater than zero
Guest_id	Guest Identification	INT	4 bytes	Foreign Key referencing Guest	Y	Greater than zero
Host_id	Host Identification	INT	4 bytes	Foreign Key referencing Host	Y	Greater than zero
Star Ratings	Rating of Host	INT	4 bytes	None	Y	Between 1 to 10
Reviews	Review of the Host	VARCHAR	250 bytes	None	Y	Maximum 250 characters

PROPERTY

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Property_id	Index of Property	INT	4 bytes	Primary Key	Y	Greater than zero
Zip_code_id	Zip Code Identification	INT	5 bytes	Foreign Key referencing Zip_Code	Y	Maximum 5 digits
Host_id	Host Identification	INT	4 bytes	Foreign Key referencing Host	Y	Greater than zero
Property_Name	Name of the Property	VARCHAR	45 bytes	None	Y	Maximum 45 characters
Street_Name	Name of the Street	VARCHAR	45 bytes	None	Y	Maximum 45 characters

PROPERTY_VERIFICATION

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Property_Verification_id	Index of Property Verification	INT	4 bytes	Primary Key	Y	Greater than zero
Property_id	Index of Property	INT	4 bytes	Foreign Key referencing Property	Y	Greater than zero
Verification_Date	Verification Date of the Property	DATETIME	8 bytes	None	Y	After 1/1/1900
Pass_Verification	Does Property pass the test?	TINYINY	1 byte	None	Y	True or False

ZIP_CODE

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Zip_code_id	Index of Zip Code	INT	4 bytes	Primary Key	Y	Maximum 5 digits
City	City referenced by the zip code	VARCHAR	45 bytes	None	Y	Maximum 45 characters
State	State referenced by the zip code	VARCHAR	45 bytes	None	Y	Maximum 45 characters

ROOM

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Room_id	Index of Room	INT	4 bytes	Composite Key	Y	Greater than zero
Property_id	Property Identification	INT	4 bytes	Composite Key/Foreign Key referencing Property	Y	Greater than zero
Room_type	Type of the room	INT	4 bytes	None	N	Greater than zero
Smoking	Does Room allow smoking?	TINYINT	1 byte	None	N	True or False

PHOTO

Column name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Photo_id	Index of Photo	INT	4 bytes	Composite Key	Y	Greater than zero
Room_id	Room Identification	INT	4 bytes	Composite Key/Foreign Key referencing Room	Y	Greater than zero
Property_id	Property Identification	INT	4 bytes	Composite Key/Foreign Key referencing Room	Y	Greater than zero
Taken_when	Date of the photo taken	DATETIME	8 bytes	None	N	After 1/1/1900
Photo	The actual photo	BLOB	65,535 bytes	None	N	Image Type

Appendix C – Enlarged ERD Version