Hotel Reservation DBMS

Billy Kim, Catalina Lamboglia, Christina Ng, Mai Vu

Team Meteor ☆彡

Introduction

In this project, we create a database management system for managing bookings at a hotel. According to our design, guests can book available rooms, with each room providing a certain service to guests. To do this, we design eight schemas: Guest, Room, Service, Booked, Available, Employee, Transaction, and ArchivedGuest. Guest contains relevant information relating to each guest within the system, such as their ID, name, age, whether or not they currently have a reservation, and the date their information was last updated. Room provides information about each room at the hotel, including the room ID, type of room, and price of booking. Service indicates what services each room provides. Booked and Available are used to organize data regarding which rooms are booked and which are available. Employee is similar to Guest in that it contains information regarding employees of the hotel. Transaction records the payments of guests at the hotel who have booked a room. ArchivedGuest is a schema that contains archived guest information that have not been updated since a certain date. Through the creation of these schemas, we effectively simulate how a hotel would manage its bookings.

Database schema

```
/* DELETE THE DATABASE IF IT ALREADY EXISTS, THEN CREATE AND USE DATABASE */
drop database if exists hotel_reservation;
create database hotel_reservation;
use hotel_reservation;

/* DELETE THE TABLES IF THEY ALREADY EXIST */
drop table if exists Guest;
drop table if exists Room;
drop table if exists Service;
drop table if exists Booked;
drop table if exists Available;
drop table if exists Employee;
drop table if exists Transaction;
drop table if exists ArchivedGuest;

/* CREATE THE SCHEMA FOR OUR TABLES */
create table Guest(cID int primary key, name text, age int, reservation int,
updatedAt date default '2020-01-01');

create table Room(rID int primary key, type text, price int);

create table Service(rID int, serviceCode int, service text, primary key(rID),
foreign key(rID) references Room(rID) on update cascade on delete cascade);

create table Booked(rID int, cID int, PRIMARY KEY (rID), date_booked date,
foreign key (rID) references Room (rID) on update cascade,
foreign key (cID) references Guest (cID) on update cascade on delete set null);

create table Available(rID int, PRIMARY KEY (rID), date_available date,
foreign key (rID) references Room (rID) on update cascade on delete cascade);

create table Employee(employee_ID int primary key, employee_name text, position text,
phone_number text);

create table Transaction(transaction_id int primary key, cID int, rID int, days int,
amount_due int, time_of_purchase datetime, checkIn date, checkOut date,
```

```
 foreign key (cID) references Guest (cID) on update cascade on delete set null,
 foreign key (rID) references Room (rID) on update cascade on delete set null);

create table ArchivedGuest(cID int primary key, name text, age int, reservation int,
updatedAt date);
```

Data Set

We did not use a public data set, but instead populated our schema independently.

```
/* POPULATE THE TABLES WITH OUR DATA */
insert into Guest values
( 1001, 'Austin Lee', 35, 0, '2021-10-01' ),
( 1002, 'Ryan Baker', 52, 1, '2021-05-01' ),
( 1003, 'Julia Flores', 18, 0, '2021-10-09' ),
( 1004, 'Katherine Adams', 42, 0, '2021-09-01' ),
( 1005, 'Patrick Mitchell', 35, 1, '2021-09-01' ),
( 1006, 'Susan Clark', 25, 1, '2021-10-01' ),
( 1007, 'Daniel Reed', 21, 1, '2021-05-01' ),
( 1008, 'Angela Garcia', 37, 1, '2021-10-11' ),
( 1009, 'Thomas Price', 74, 0, '2021-10-01' ),
( 1010, 'Lois Brown', 68, 1, '2021-10-11' ),
( 1011, 'Denise Smith', 35, 1, '2021-09-01' ),
( 1012, 'Jesse James', 42, 1, '2021-10-01' );

insert into Room values
( 101, 'single', 80 ),
( 102, 'single', 85 ),
( 103, 'double', 100 ),
( 104, 'double', 105 ),
( 105, 'triple', 120 ),
( 201, 'single', 85 ),
( 202, 'single', 85 ),
( 203, 'double', 95 ),
( 204, 'double', 115 ),
( 205, 'triple', 130 ),
( 301, 'single', 80 ),
( 302, 'single', 80 ),
( 303, 'double', 105 ),
( 304, 'double', 120 ),
( 305, 'triple', 125 );

insert into Service values
(101, 3001, 'minibar'),
(102, 3002, 'spa'),
(103, 3003, 'special meal'),
(104, 3004, 'spa'),
(105, 3005, 'minibar'),
(201, 3006, 'spa'),
(202, 3007, 'spa'),
(203, 3008, 'special meal'),
(204, 3009, 'special meal'),
(205, 3010, 'minibar'),
(301, 3011, 'special meal'),
(302, 3012, 'special meal'),
```

```
(303, 3013, 'minibar'),
(304, 3014, 'minibar'),
(305, 3015, 'special meal');


insert into Booked values
(101, 1001, '2021-10-01'),
(102, 1002, '2021-10-01'),
(103, 1003, '2021-10-01'),
(201, 1004, '2021-10-01'),
(203, 1005, '2021-10-01'),
(301, 1006, '2021-10-01'),
(305, 1010, '2021-10-01');

insert into Available values
(104, '2021-10-01'),
(105, '2021-10-01'),
(202, '2021-10-01'),
(204, '2021-10-01'),
(205, '2021-10-01'),
(302, '2021-10-01'),
(303, '2021-10-01'),
(304, '2021-10-01');

insert into Employee values
( 0, 'Arnoldo Demetrio', 'Concierge', '555-1234' ),
( 1, 'Lila Prudenzio', 'Manager', '555-1111' ),
( 2, 'Stig Tahmina', 'Front Desk', '555-1112' ),
( 3, 'Erhan Jozafat', 'Valet', '555-1121' ),
( 4, 'Eini Finn', 'Janitor', '555-1211' ),
( 5, 'Melanija Javiera', 'Sales', '555-1311' ),
( 6, 'Suman Gundisalvus', 'Housekeeper', '555-1131' ),
( 7, 'Solly Ivanka', 'Security', '555-1114' );

insert into Transaction values
( 0, 1001, 101, 3, 235, '2021-10-11 10:34:09', '2021-10-11', '2021-10-14' ),
( 1, 1002, 102, 1, 75, '2021-11-11 22:34:09', '2021-11-11', '2021-11-12' ),
( 2, 1003, 103, 2, 155, '2021-10-11 5:22:09', '2021-10-12', '2021-10-14' ),
( 3, 1004, 104, 5, 455, '2021-05-11 10:34:09', '2021-05-11', '2021-05-16' ),
( 4, 1005, 105, 2, 125, '2021-10-11 10:34:09', '2021-10-11', '2021-10-13' ),
( 5, 1006, 201, 3, 345, '2021-10-11 10:34:09', '2021-10-11', '2021-10-14' ),
( 6, 1007, 202, 4, 535, '2021-10-11 10:34:09', '2021-10-11', '2021-10-15' ),
( 7, 1008, 203, 5, 735, '2021-10-11 10:34:09', '2021-10-11', '2021-10-16' );
```

Relation Schemas are in BCNF or in 3NF

Relation Guest is in BCNF

- The relation is in 3NF because there is no transitive dependency
- The only key is {cID}
- cID -> name age reservation, updateAt holds in the relation (cID is a super key)

Relation Room is in BCNF

- The relation is in 3NF because there is no transitive dependency. For example: price does not depend on type
- The only key is {rID}
- rID -> type price holds in the relation (rID is a super key)

Relation Service is in BCNF

- The relation is in 3NF because there is no transitive dependency.
- The only key is {rID}
- rID -> serviceCode service holds in the relation (rID is a super key)

Relation Booked is in BCNF

- The relation is in 3NF because there is no transitive dependency. For example: date_booked does not depend on cID
- The only key is {rID}
- rID -> cID date_booked holds in the relation (rID is a super key)

Relation Available is in BCNF

- The relation is in 3NF because there is no transitive dependency.
- The only key is {rID}
- rID -> date_available holds in the relation (rID is a super key)

Relation Employee is in BCNF

- The relation is in 3NF because there is no transitive dependency. For example: phone_number does not depend on position.
- The only key is {employee_ID}
- employee_ID -> employee_name position phone_number holds in the relation (employee_ID is a super key)

Relation Transaction is in BCNF

- The relation is in 3NF because there is no transitive dependency.
- The only key is {transaction_id}
- transaction_id -> cID rID days amount_due time_of_purchase checkIn checkOut holds in the relation (transaction_id is a super key)

Relations After Populated With Data

Guest

cID	name	age	reservation	updatedAt
1001	Austin Lee	35	0	2021-10-01
1002	Ryan Baker	52	1	2021-05-01
1003	Julia Flores	18	0	2021-10-09
1004	Katherine Adams	42	0	2021-09-01

1005	Patrick Mitchell	35	1	2021-09-01
1006	Susan Clark	25	1	2021-10-01
1007	Daniel Reed	21	1	2021-05-01
1008	Angela Garcia	37	1	2021-10-11
1009	Thomas Price	74	0	2021-10-01
1010	Lois Brown	68	1	2021-10-11
1011	Denise Smith	35	1	2021-09-01
1012	Jesse James	42	1	2021-10-01

Room

rID	type	price
101	single	80
102	single	85
103	double	100
104	double	105
105	triple	120
201	single	85
202	single	85
203	double	95
204	double	115
205	triple	130
301	single	80
302	single	80
303	double	105
304	double	120
305	triple	125

Service

rID	serviceCode	service
101	3001	minibar
102	3002	spa
103	3003	special meal
104	3004	spa
105	3005	minibar
201	3006	spa
202	3007	spa
203	3008	special meal
204	3009	special meal
205	3010	minibar
301	3011	special meal
302	3012	special meal
303	3013	minibar
304	3014	minibar
305	3015	special meal

Booked

rID	cID	date_booked

101	1001	2021-10-01
102	1002	2021-10-01
103	1003	2021-10-01
201	1004	2021-10-01
203	1005	2021-10-01
301	1006	2021-10-01
305	1010	2021-10-01

Available

rID	date_available
104	2021-10-01
105	2021-10-01
202	2021-10-01
204	2021-10-01
205	2021-10-01
302	2021-10-01
303	2021-10-01
304	2021-10-01

Employee

employee_ID	employee_name	position	phone_number
0	Arnoldo Demetrio	Concierge	555-1234
1	Lila Prudenzio	Manager	555-1111
2	Stig Tahmina	Front Desk	555-1112
3	Erhan Jozafat	Valet	555-1121
4	Eini Finn	Janitor	555-1211
5	Melanija Javiera	Sales	555-1311
6	Suman Gundisalvus	Housekeeper	555-1131
7	Solly Ivanka	Security	555-1114

Transaction

transaction_id	cID	rID	days	amount_due	time_of_purchase	checkIn	checkOut
0	1001	101	3	235	2021-10-11 10:34:09	2021-10-11	2021-10-14
1	1002	102	1	75	2021-11-11 22:34:09	2021-11-11	2021-11-12
2	1003	103	2	155	2021-10-11 05:22:09	2021-10-12	2021-10-14
3	1004	104	5	455	2021-05-11 10:34:09	2021-05-11	2021-05-16
4	1005	105	2	125	2021-10-11 10:34:09	2021-10-11	2021-10-13
5	1006	201	3	345	2021-10-11 10:34:09	2021-10-11	2021-10-14
6	1007	202	4	535	2021-10-11 10:34:09	2021-10-11	2021-10-15
7	1008	203	5	735	2021-10-11 10:34:09	2021-10-11	2021-10-16

Functional Requirements

Requests Available To The User

Use Case	SQL Statement
1. Get list of guests in alphabetical order	`select cID, name, age from Guest order by name;`
2. Find the cheapest double room	**`select * from Room where type='double' and price <= all (select price from Room R2 where type='double');`**
3. Sort the rooms under $100 from cheapest to most expensive	`select * from Room where price < 100 order by price, rID;`
4. Find types of available rooms where the maximum price is 100	**`select type, max(price) as max_price from Available join Room where Available.rID = Room.rID group by type having max(price) <= 100;`**
5. Need to display all employees for managers or HR, etc	`select * from Employee;`
6. Select transactions where the check in date is in the future	`select * from Transaction where checkIn > curdate();`
7. Someone needs to request room cleaning	`select * from Employee where position like 'housekeeper';`
8. Check customer transaction history	`select * from Transaction where cID in ( select cID from Guest where name like 'customer name here');`
9. Find all available single rooms	`select * from Available where rID in (select rID from Room where type="single");`
10. Find all available rooms with more than one bed	`select * from Available where rID in (select rID from Room where type <> "single");`
11. Find all available rooms that come with a spa	`select * from Available where rID in (select Room.rID from Room, Service where Room.rID=Service.rID and service="spa");`
12. Find all booked rooms with guests over 50	`select * from Booked where cID in (select cID from Guest where age > 50);`
13. Find all available rooms that request minibar for an extra service:	`select * from Room where rID in ( select Service.rID from Service, Available where Service.rID=Available.rID and service = 'minibar');`
14. Find all available rooms that request spa or special meal for an extra service	`select * from Room where rID in ( select Service.rID from Service, Available where`

	Service.rID=Available.rID and (service = 'spa' or service = 'special meal'));
15. Find all single rooms that request spa for an extra service	select * from Room where rID in (select rID from Service where Room.rID = Service.rID and Room.type = 'single' and Service.service = 'spa');
16. Find all available service with triple room	select distinct service from Service where rID in (select rID from Room where Room.rID = Service.rID and Room.type = 'triple');
17. Produce report that shows all rooms and their services	**select * from Room left outer join Service on Room.rID = Service.rID;**

Screenshots

1. All guests in alphabetical order

```
private static void guestsInAlphaOrder(Statement statement) throws SQLException {
    ResultSet rs = statement.executeQuery( sql: "select * from Guest order by name");
    System.out.println("\nGetting a lit of all guests in alphabetical order:");
    //printResultSetfromGuest(rs);
    while (rs.next()) {
        int cID = rs.getInt( columnLabel: "cID");
        String name = rs.getString( columnLabel: "name");
        int age = rs.getInt( columnLabel: "age");
        System.out.println("Guest ID: " + cID + " Name: " + name + " Age: " + age);
    }
}
```

```
Getting a lit of all guests in alphabetical order:
Guest ID: 1008 Name: Angela Garcia Age: 37
Guest ID: 1001 Name: Austin Lee Age: 35
Guest ID: 1007 Name: Daniel Reed Age: 21
Guest ID: 1011 Name: Denise Smith Age: 35
Guest ID: 1012 Name: Jesse James Age: 42
Guest ID: 1003 Name: Julia Flores Age: 18
Guest ID: 1004 Name: Katherine Adams Age: 42
Guest ID: 1010 Name: Lois Brown Age: 68
Guest ID: 1005 Name: Patrick Mitchell Age: 35
Guest ID: 1002 Name: Ryan Baker Age: 52
Guest ID: 1006 Name: Susan Clark Age: 25
Guest ID: 1009 Name: Thomas Price Age: 74
```

2. Finding the cheapest double room

```
private static void findCheapestDouble(Statement statement) throws SQLException {
    ResultSet rs = statement.executeQuery(
            sql: "select * from Room where type='double' " +
                    "and price <= all (select price from Room R2 where type='double')");
    System.out.println("\nFinding the cheapest double room:");
    printResultSetfromRoom(rs);
}
```

```
Finding the cheapest double room:
Room ID: 203 Type: double Price: 95
```

3. Sorting the rooms under $100 from cheapest to most expensive

```
private static void sortUnder100(Statement statement) throws SQLException {
    ResultSet rs = statement.executeQuery
            ( sql: "select * from Room where price < 100 order by price, rID");
    System.out.println("\nSorting the rooms under $100 from cheapest to most expensive:");
    printResultSetfromRoom(rs);
}
```

```
Sorting the rooms under $100 from cheapest to most expensive:
Room ID: 101 Type: single Price: 80
Room ID: 301 Type: single Price: 80
Room ID: 302 Type: single Price: 80
Room ID: 102 Type: single Price: 85
Room ID: 201 Type: single Price: 85
Room ID: 202 Type: single Price: 85
Room ID: 203 Type: double Price: 95
```

4. Finding types of available rooms where the maximum price is $100

```
private static void typesMax100(Statement statement) throws SQLException {
    ResultSet rs = statement.executeQuery
            ( sql: "select type, max(price) as max_price from Available join Room " +
                    "where Available.rID = Room.rID group by type having max(price) <= 100");
    System.out.println("\nFinding the types of available rooms where the max price is $100:");
    while (rs.next()) {
        String type = rs.getString( columnLabel: "type");
        int maxPrice = rs.getInt( columnLabel: "max_price");
        System.out.println("Type: " + type + " Max_Price: " + maxPrice);
    }
}
```

```
Finding the types of available rooms where the max price is $100:
Type: single Max_Price: 85
```

5. Displaying all employees for the manager or HR

```java
private static void allEmployees(Statement statement) throws SQLException {
    ResultSet rs = statement.executeQuery( sql: "select * from Employee");
    System.out.println("\nFinding all employees:");
    printResultSetfromEmployee(rs);
}
```

```
Finding all employees:
Employee ID: 0 Name: Arnoldo Demetrio Position: Concierge Number: 555-1234
Employee ID: 1 Name: Lila Prudenzio Position: Manager Number: 555-1111
Employee ID: 2 Name: Stig Tahmina Position: Front Desk Number: 555-1112
Employee ID: 3 Name: Erhan Jozafat Position: Valet Number: 555-1121
Employee ID: 4 Name: Eini Finn Position: Janitor Number: 555-1211
Employee ID: 5 Name: Melanija Javiera Position: Sales Number: 555-1311
Employee ID: 6 Name: Suman Gundisalvus Position: Housekeeper Number: 555-1131
Employee ID: 7 Name: Solly Ivanka Position: Security Number: 555-1114
```

6. Finding transactions where the check-in date is in the future

```java
private static void futureTransactions(Statement statement) throws SQLException
{
    ResultSet rs = statement.executeQuery( sql: "select * from Transaction where checkIn > \"2021-10-11\"");
    System.out.println("\nFinding all transactions where the check in date is after '2021-10-11':");
    printResultSetfromTransaction(rs);
}
```

```
Finding all transactions where the check in date is after '2021-10-11':
Transaction ID: 1 cID: 1002 rID: 102 Days: 1 Amount Due: $75 Time of Purchase:
  2021-11-11 22:34:09 Check In: 2021-11-11 Check Out: 2021-11-12
Transaction ID: 2 cID: 1003 rID: 103 Days: 2 Amount Due: $155 Time of Purchase:
  2021-10-11 05:22:09 Check In: 2021-10-12 Check Out: 2021-10-14
```

7. Finding someone that is able to clean a room

```java
private static void roomCleaning(Statement statement) throws SQLException
{
    ResultSet rs = statement.executeQuery( sql: "select * from Employee where position like 'housekeeper'");
    System.out.println("\nFinding people for room cleaning:");
    printResultSetfromEmployee(rs);
}
```

```
Finding people for room cleaning:
Employee ID: 6 Name: Suman Gundisalvus Position: Housekeeper Number: 555-1131
```

8. Checking customer transaction history

```java
private static void checkTransactionHistory(Statement statement) throws SQLException
{
    ResultSet rs = statement.executeQuery( sql: "select * from Transaction where cID in " +
                                    "( select cID from Guest where name like 'Austin Lee');");
    System.out.println("\nChecking customer (Austin Lee)'s transaction history:");
    printResultSetfromTransaction(rs);
}
```

```
Checking customer (Austin Lee)'s transaction history:
Transaction ID: 0 cID: 1001 rID: 101 Days: 3 Amount Due: $235 Time of Purchase: 2021-10-11
  10:34:09 Check In: 2021-10-11 Check Out: 2021-10-14
```

9. Finding all available single rooms

```java
private static void findSingleRooms(Statement statement) throws SQLException {
    System.out.println("\nFinding all available single rooms:");
    ResultSet rs = statement.executeQuery( sql: "select * from Available where rID in " +
                                        "(select rID from Room where type = \"single\")");
    printResultSetfromAvailable(rs);
}
```

```
Finding all available single rooms:
Room ID: 202 Date Available: 2021-10-01
Room ID: 302 Date Available: 2021-10-01
```

10. Finding all available rooms with more than one bed

```java
private static void findNonSingleRooms(Statement statement) throws SQLException {
    System.out.println("Finding all available rooms with more than one bed:");
    ResultSet rs = statement.executeQuery( sql: "select * from Available where rID in " +
                                        "(select rID from Room where type <> \"single\")");
    printResultSetfromAvailable(rs);
}
```

```
Finding all available rooms with more than one bed:
Room ID: 104 Date Available: 2021-10-01
Room ID: 105 Date Available: 2021-10-01
Room ID: 204 Date Available: 2021-10-01
Room ID: 205 Date Available: 2021-10-01
Room ID: 303 Date Available: 2021-10-01
Room ID: 304 Date Available: 2021-10-01
```

11. Finding all available rooms that come with a spa

```
private static void findSpaRooms(Statement statement) throws SQLException {
    System.out.println("Finding all available rooms that come with a spa:");
    ResultSet rs = statement.executeQuery( sql: "select * from Available where rID in " +
            "(select Room.rID from Room, Service where Room.rID=Service.rID and service=\"spa\")");
    printResultSetfromAvailable(rs);
}
```

```
Finding all available rooms that come with a spa:
Room ID: 104 Date Available: 2021-10-01
Room ID: 202 Date Available: 2021-10-01
```

12. Finding all booked rooms with guests over 50

```
private static void findOlderBooked(Statement statement) throws SQLException {
    System.out.println("Finding all booked rooms with guests over 50:");
    ResultSet rs = statement.executeQuery( sql: "select * from Booked where cID in " +
                                               "(select cID from Guest where age > 50)");
    printResultSetfromBooked(rs);
}
```

```
Finding all booked rooms with guests over 50:
Room ID: 102 cID: 1002 Date Booked: 2021-10-01
Room ID: 305 cID: 1010 Date Booked: 2021-10-01
```

13. Finding all available rooms that request a minimbar for an extra service

```
private static void findMinibarRooms(Statement statement) throws SQLException {
    System.out.println("Finding all available rooms with a minibar:");
    ResultSet rs = statement.executeQuery( sql: "select * from Room where rID in " +
                    "( select Service.rID from Service, Available " +
                    "where Service.rID=Available.rID and service = 'minibar')");
    printResultSetfromRoom(rs);
}
```

```
Finding all available rooms with a minibar:
Room ID: 105 Type: triple Price: 120
Room ID: 205 Type: triple Price: 130
Room ID: 303 Type: double Price: 105
Room ID: 304 Type: double Price: 120
```

14. Finding all available rooms that request a spa or a special meal for an extra service

```java
private static void findSpaOrMealRooms(Statement statement) throws SQLException {
    System.out.println("Finding all available rooms with a spa or special meal:");
    ResultSet rs = statement.executeQuery( sql: "select * from Room where rID in " +
            "( select Service.rID from Service, Available " +
            "where Service.rID=Available.rID and (service = 'spa' or service = 'special meal'))");
    printResultSetfromRoom(rs);
}
```

```
Finding all available rooms with a spa or special meal:
Room ID: 104 Type: double Price: 105
Room ID: 202 Type: single Price: 85
Room ID: 204 Type: double Price: 115
Room ID: 302 Type: single Price: 80
```

15. Finding all single rooms that request spa for an extra service

```java
private static void findSingleSpaRooms(Statement statement) throws SQLException {
    System.out.println("Finding all single rooms that request spa for an extra service:");
    ResultSet rs = statement.executeQuery( sql: "select * from Room where rID in " +
            "(select rID from Service where Room.rID = Service.rID and Room.type = 'single' " +
            "and Service.service = 'spa')");
    printResultSetfromRoom(rs);
}
```

```
Finding all single rooms that request spa for an extra service:
Room ID: 102 Type: single Price: 85
Room ID: 201 Type: single Price: 85
Room ID: 202 Type: single Price: 85
```

16. Finding all available services with triple rooms

```java
private static void findTripleRoomServices(Statement statement) throws SQLException {
    System.out.println("\nFinding all available services with triple rooms:");
    ResultSet rs = statement.executeQuery( sql: "select distinct service from Service where rID in " +
            "(select rID from Room where Room.rID = Service.rID and Room.type = 'triple')");

    while(rs.next())
    {
        String service = rs.getString( columnLabel: "service");
        System.out.println("Service: " + service);
    }
}
```

```
Finding all available services with triple rooms:
Service: minibar
Service: special meal
```

Primary key constraint violation

```
private static void primaryKeyViolation(Statement statement) throws SQLException {
    ResultSet rs = statement.executeQuery( sql: "SELECT * FROM Room WHERE rID=101");
    System.out.println("Find if room 101 exists:");
    printResultSetfromRoom(rs);

    System.out.println("Attempt to execute 'INSERT INTO Room VALUES (101, 'single', 100)'");
    statement.executeUpdate( sql: "INSERT INTO Room VALUES (101, 'single', 100)");
}
```

```
====== Primary Key Constraint Violation ======
Find if room 101 exists:
Room ID: 101 Type: single Price: 80
Attempt to execute 'INSERT INTO Room VALUES (101, 'single', 100)'
java.sql.SQLIntegrityConstraintViolationException Create breakpoint : Duplicate entry '101' for key 'room.PRIMARY'
    at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:117)
    at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:122)
    at com.mysql.cj.jdbc.StatementImpl.executeUpdateInternal(StatementImpl.java:1340)
    at com.mysql.cj.jdbc.StatementImpl.executeLargeUpdate(StatementImpl.java:2089)
    at com.mysql.cj.jdbc.StatementImpl.executeUpdate(StatementImpl.java:1251)
    at Hotel.primaryKeyViolation(Hotel.java:317)
    at Hotel.main(Hotel.java:68)
```

Foreign key constraint violation

```
private static void foreignKeyViolation(Statement statement) throws SQLException {
    System.out.println("Find if room 101 is booked");
    ResultSet rs = statement.executeQuery( sql: "SELECT * FROM Booked WHERE rID=101");
    printResultSetfromBooked(rs);

    System.out.println("Attempt to execute 'DELETE FROM Room WHERE rID=101'");
    statement.executeUpdate( sql: "DELETE FROM Room WHERE rID=101");
}
```

```
====== Foreign Key Constraint Violation ======
Find if room 101 is booked
Room ID: 101 cID: 1001 Date Booked: 2021-10-01
Attempt to execute 'DELETE FROM Room WHERE rID=101'
java.sql.SQLIntegrityConstraintViolationException Create breakpoint : Cannot delete or update a parent row: a
  foreign key constraint fails (`hotel_reservation`.`booked`, CONSTRAINT `booked_ibfk_1` FOREIGN KEY (`rID`)
  REFERENCES `room` (`rID`) ON UPDATE CASCADE)
    at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:117)
    at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:122)
    at com.mysql.cj.jdbc.StatementImpl.executeUpdateInternal(StatementImpl.java:1340)
    at com.mysql.cj.jdbc.StatementImpl.executeLargeUpdate(StatementImpl.java:2089)
    at com.mysql.cj.jdbc.StatementImpl.executeUpdate(StatementImpl.java:1251)
    at Hotel.foreignKeyViolation(Hotel.java:337)
    at Hotel.main(Hotel.java:78)
```

Archived Function

```java
private static void callArchiveProcedure() throws SQLException
{
    System.out.println("\nCalling the procedure Archive with cutoff date: '2021-09-10'");
    String sql = "{CALL Archive(?)}";
    CallableStatement cs = conn.prepareCall(sql);
    cs.setString( parameterIndex: 1, x: "2021-09-10");
    ResultSet rs = cs.executeQuery();
}
```

```
Calling the procedure Archive with cutoff date: '2021-09-10'

Printing Archived Guests...
Guest ID: 1002 Name: Ryan Baker Age: 52 Reservation: 1 Updated At: 2021-05-01
Guest ID: 1004 Name: Katherine Adams Age: 42 Reservation: 0 Updated At: 2021-09-01
Guest ID: 1005 Name: Patrick Mitchell Age: 35 Reservation: 1 Updated At: 2021-09-01
Guest ID: 1007 Name: Daniel Reed Age: 21 Reservation: 1 Updated At: 2021-05-01
Guest ID: 1011 Name: Denise Smith Age: 35 Reservation: 1 Updated At: 2021-09-01

Printing Guests...
Guest ID: 1001 Name: Austin Lee Age: 35 Reservation: 0 Updated At: 2021-10-01
Guest ID: 1003 Name: Julia Flores Age: 18 Reservation: 0 Updated At: 2021-10-09
Guest ID: 1006 Name: Susan Clark Age: 25 Reservation: 1 Updated At: 2021-10-01
Guest ID: 1008 Name: Angela Garcia Age: 37 Reservation: 1 Updated At: 2021-10-11
Guest ID: 1009 Name: Thomas Price Age: 74 Reservation: 0 Updated At: 2021-10-01
Guest ID: 1010 Name: Lois Brown Age: 68 Reservation: 1 Updated At: 2021-10-11
Guest ID: 1012 Name: Jesse James Age: 42 Reservation: 1 Updated At: 2021-10-01
```

SQL Statements, Triggers, and Stored Procedures

Five significantly different types of SQL (one correlated subquery, group by and having, aggregation, outer join, and mathematical set operation) are highlighted in bold.

SQL Select Statements List

`select cID, name, age from Guest order by name;`
`select * from Room where type='double' and price <= all (select price from Room R2 where type='double');`
`select * from Room where price < 100 order by price, rID;`
`// Has group by, having, and an aggregation` **`select type, max(price) as max_price from Available join Room where Available.rID = Room.rID group by type having max(price) <= 100;`**
`select * from Employee;`
`select * from Transaction where checkIn > curdate();`

```
select * from Employee where position like 'housekeeper';
```

```
select * from Transaction where cID in ( select cID from Guest where name like
'customer name here');
```

```
select * from Available where rID in (select rID from Room where type="single");
```

```
select * from Available where rID in (select rID from Room where type <> "single");
```

```
select * from Available where rID in (select Room.rID from Room, Service where
Room.rID=Service.rID and service="spa");
```

```
select * from Booked where cID in (select cID from Guest where age > 50);
```

```
select * from Room where rID in ( select Service.rID from Service, Available where
Service.rID=Available.rID and service = 'minibar');
```

```
select * from Room where rID in ( select Service.rID from Service, Available where
Service.rID=Available.rID and (service = 'spa' or service = 'special meal'));
```

```
select * from Room where rID in (select rID from Service where Room.rID =
Service.rID and Room.type = 'single' and Service.service = 'spa');
```

```
select distinct service from Service where rID in (select rID from Room where
Room.rID = Service.rID and Room.type = 'triple');
```

```
select * from Room left outer join Service on Room.rID = Service.rID;
```

SQL Update Statements List

SQL Delete Statements List

```
drop trigger if exists ServiceD;
delimiter //
create trigger ServiceD
after delete on Room
for each row
begin
       delete from Service
       where rID = old.rID;
end;//
```

```
DROP TRIGGER IF EXISTS BookedTrigger;
delimiter //
CREATE TRIGGER BookedTrigger
AFTER INSERT ON Booked
FOR EACH ROW
BEGIN
    DELETE FROM Available
    WHERE rID = new.rID;
```

```
END;//
```

```
DROP TRIGGER IF EXISTS AvailableTrigger;
delimiter //
CREATE TRIGGER AvailableTrigger
AFTER INSERT ON Available
FOR EACH ROW
BEGIN
    DELETE FROM Booked
    WHERE rID = new.rID;
END;
//
```

```
DROP TRIGGER IF EXISTS DelFromBooked//
delimiter //
CREATE TRIGGER DelFromBooked
AFTER DELETE ON Booked
FOR EACH ROW
BEGIN
    INSERT INTO Available values (OLD.rID, CURRENT_DATE());
END;//
```

SQL Insert Statements List

```
insert into Guest values
( 1001, 'Austin Lee', 35, 0, '2021-10-01' ),
( 1002, 'Ryan Baker', 52, 1, '2021-05-01' ),
( 1003, 'Julia Flores', 18, 0, '2021-10-09' ),
( 1004, 'Katherine Adams', 42, 0, '2021-09-01' ),
( 1005, 'Patrick Mitchell', 35, 1, '2021-09-01' ),
( 1006, 'Susan Clark', 25, 1, '2021-10-01' ),
( 1007, 'Daniel Reed', 21, 1, '2021-05-01' ),
( 1008, 'Angela Garcia', 37, 1, '2021-10-11' ),
( 1009, 'Thomas Price', 74, 0, '2021-10-01' ),
( 1010, 'Lois Brown', 68, 1, '2021-10-11' ),
( 1011, 'Denise Smith', 35, 1, '2021-09-01' ),
( 1012, 'Jesse James', 42, 1, '2021-10-01' );
```

```
insert into Room values
( 101, 'single', 80 ),
( 102, 'single', 85 ),
( 103, 'double', 100 ),
( 104, 'double', 105 ),
( 105, 'triple', 120 ),
( 201, 'single', 85 ),
( 202, 'single', 85 ),
( 203, 'double', 95 ),
( 204, 'double', 115 ),
( 205, 'triple', 130 ),
( 301, 'single', 80 ),
( 302, 'single', 80 ),
( 303, 'double', 105 ),
```

```
( 304, 'double', 120 ),
( 305, 'triple', 125 );
```

```
/* 001: special meal, 002: minibar, 004: spa */
insert into Service values
(101, 3001, 'minibar'),
(102, 3002, 'spa'),
(103, 3003, 'special meal'),
(104, 3004, 'spa'),
(105, 3005, 'minibar'),
(201, 3006, 'spa'),
(202, 3007, 'spa'),
(203, 3008, 'special meal'),
(204, 3009, 'special meal'),
(205, 3010, 'minibar'),
(301, 3011, 'special meal'),
(302, 3012, 'special meal'),
(303, 3013, 'minibar'),
(304, 3014, 'minibar'),
(305, 3015, 'special meal');
```

```
insert into Booked values
(101, 1001, '2021-10-01'),
(102, 1002, '2021-10-01'),
(103, 1003, '2021-10-01'),
(201, 1004, '2021-10-01'),
(203, 1005, '2021-10-01'),
(301, 1006, '2021-10-01'),
(305, 1010, '2021-10-01');
```

```
insert into Available values
(104, '2021-10-01'),
(105, '2021-10-01'),
(202, '2021-10-01'),
(204, '2021-10-01'),
(205, '2021-10-01'),
(302, '2021-10-01'),
(303, '2021-10-01'),
(304, '2021-10-01');
```

```
insert into Employee values
( 0, 'Arnoldo Demetrio', 'Concierge', '555-1234' ),
( 1, 'Lila Prudenzio', 'Manager', '555-1111' ),
( 2, 'Stig Tahmina', 'Front Desk', '555-1112' ),
( 3, 'Erhan Jozafat', 'Valet', '555-1121' ),
( 4, 'Eini Finn', 'Janitor', '555-1211' ),
( 5, 'Melanija Javiera', 'Sales', '555-1311' ),
( 6, 'Suman Gundisalvus', 'Housekeeper', '555-1131' ),
( 7, 'Solly Ivanka', 'Security', '555-1114' );
```

```
insert into Transaction values
( 0, 1001, 101, 3, 235, '2021-10-11 10:34:09', '2021-10-11', '2021-10-14' ),
( 1, 1002, 102, 1, 75, '2021-11-11 22:34:09', '2021-11-11', '2021-11-12' ),
( 2, 1003, 103, 2, 155, '2021-10-11 5:22:09', '2021-10-12', '2021-10-14' ),
( 3, 1004, 104, 5, 455, '2021-05-11 10:34:09', '2021-05-11', '2021-05-16' ),
```

```
( 4, 1005, 105, 2, 125, '2021-10-11 10:34:09', '2021-10-11', '2021-10-13' ),
( 5, 1006, 201, 3, 345, '2021-10-11 10:34:09', '2021-10-11', '2021-10-14' ),
( 6, 1007, 202, 4, 535, '2021-10-11 10:34:09', '2021-10-11', '2021-10-15' ),
( 7, 1008, 203, 5, 735, '2021-10-11 10:34:09', '2021-10-11', '2021-10-16' );
```

```
DROP TRIGGER IF EXISTS DelFromBooked//
delimiter //
CREATE TRIGGER DelFromBooked
AFTER DELETE ON Booked
FOR EACH ROW
BEGIN
    INSERT INTO Available values (OLD.rID, CURRENT_DATE());
END;//
```

```
DROP PROCEDURE IF EXISTS Archive;
delimiter //
CREATE PROCEDURE Archive(IN cutoff date)
BEGIN
       insert into ArchivedGuest (select * from Guest where updatedAt < cutoff);
       delete from Guest where updatedAt < cutoff;
END;//
```

SQL Triggers List

```
drop trigger if exists ServiceD;
delimiter //
create trigger ServiceD
after delete on Room
for each row
begin
       delete from Service
       where rID = old.rID;
end;//
```

```
DROP TRIGGER IF EXISTS BookedTrigger;
delimiter //
CREATE TRIGGER BookedTrigger
AFTER INSERT ON Booked
FOR EACH ROW
BEGIN
    DELETE FROM Available
    WHERE rID = new.rID;
END;//
```

```
DROP TRIGGER IF EXISTS AvailableTrigger;
delimiter //
CREATE TRIGGER AvailableTrigger
AFTER INSERT ON Available
FOR EACH ROW
BEGIN
    DELETE FROM Booked
    WHERE rID = new.rID;
```

```
END;
//
```

```
DROP TRIGGER IF EXISTS DelFromBooked//
delimiter //
CREATE TRIGGER DelFromBooked
AFTER DELETE ON Booked
FOR EACH ROW
BEGIN
    INSERT INTO Available values (OLD.rID, CURRENT_DATE());
END;//
```

SQL Stored Procedures List

```
DROP PROCEDURE IF EXISTS Archive;
delimiter //
CREATE PROCEDURE Archive(IN cutoff date)
BEGIN
       insert into ArchivedGuest (select * from Guest where updatedAt < cutoff);
       delete from Guest where updatedAt < cutoff;
END;//
```

* Last modification date: 12/4/21