Hotel Reservation DBMS

By Team Meteor ☆彡
Billy Kim, Catalina Lamboglia, Christina Ng, Mai Vu

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 on delete 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);

/* 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 );

/* 001: special meal, 002: minibar, 004: spa */
insert into Service values
(101, 002, 'minibar'),
(102, 004, 'spa'),
(103, 001, 'special meal'),
(104, 004, 'spa'),
(105, 002, 'minibar'),
(201, 004, 'spa'),
(202, 004, 'spa'),
(203, 001, 'special meal'),
(204, 001, 'special meal'),
(205, 002, 'minibar'),
(301, 001, 'special meal'),
(302, 001, 'special meal'),
(303, 002, 'minibar'),
(304, 002, 'minibar'),
(305, 001, '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' );
```

Triggers

If a room is deleted, its corresponding service is also deleted:

```
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;//
```

If a room becomes booked, then it is no longer available:

```
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;//
```

If a room becomes available, then it is no longer booked:

```
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;
//
```

If a room is not booked anymore, then it becomes available:

```
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;//
```

Archive Procedure

If a guest's information has not been updated since the cutoff date, then it is moved to the archived table and deleted from the guest 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;//
```

Functional Requirements

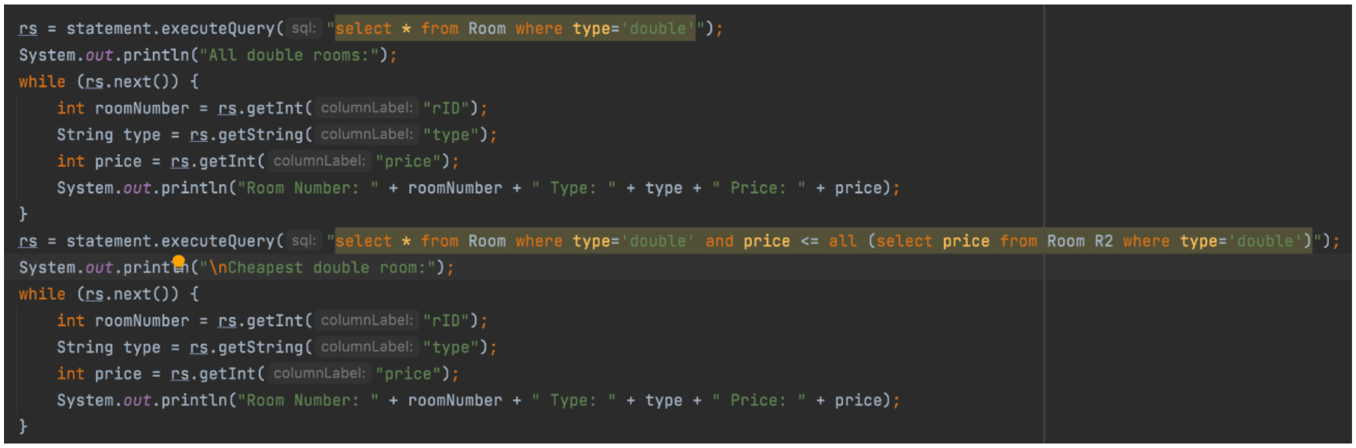
Requests Available To The User

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

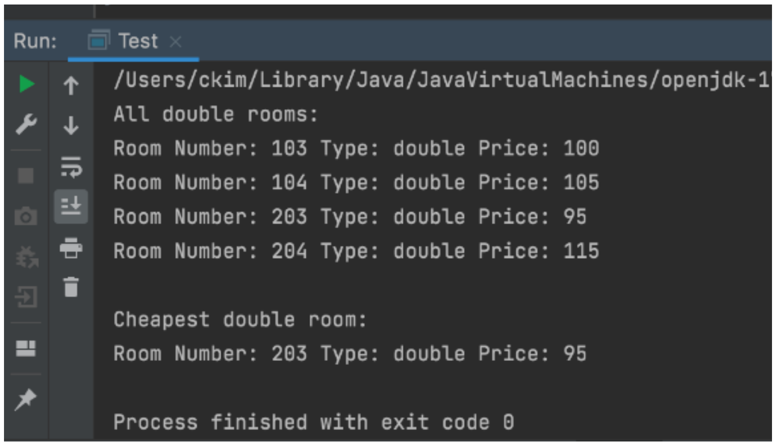
	Service.rID=Available.rID and (service = 'spa' or service = 'special meal'));
15. Find all single rooms that request spa for an extra service	select * from Room where rID in (select rID from Service where Room.rID = Service.rID and Room.type = 'single' and Service.service = 'spa');
16. Find all available 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');

Examples Of Three Different User Requests:

1. Finding the cheapest double room:

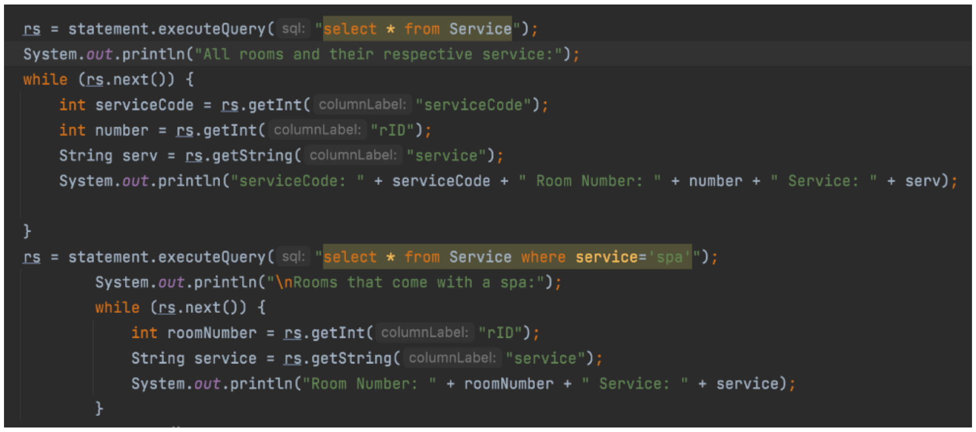


```
rs = statement.executeQuery( sql: "select * from Room where type='double'");
System.out.println("All double rooms:");
while (rs.next()) {
    int roomNumber = rs.getInt( columnLabel: "rID");
    String type = rs.getString( columnLabel: "type");
    int price = rs.getInt( columnLabel: "price");
    System.out.println("Room Number: " + roomNumber + " Type: " + type + " Price: " + price);
}
rs = statement.executeQuery( sql: "select * from Room where type='double' and price <= all (select price from Room R2 where type='double')");
System.out.println("\nCheapest double room:");
while (rs.next()) {
    int roomNumber = rs.getInt( columnLabel: "rID");
    String type = rs.getString( columnLabel: "type");
    int price = rs.getInt( columnLabel: "price");
    System.out.println("Room Number: " + roomNumber + " Type: " + type + " Price: " + price);
}
```

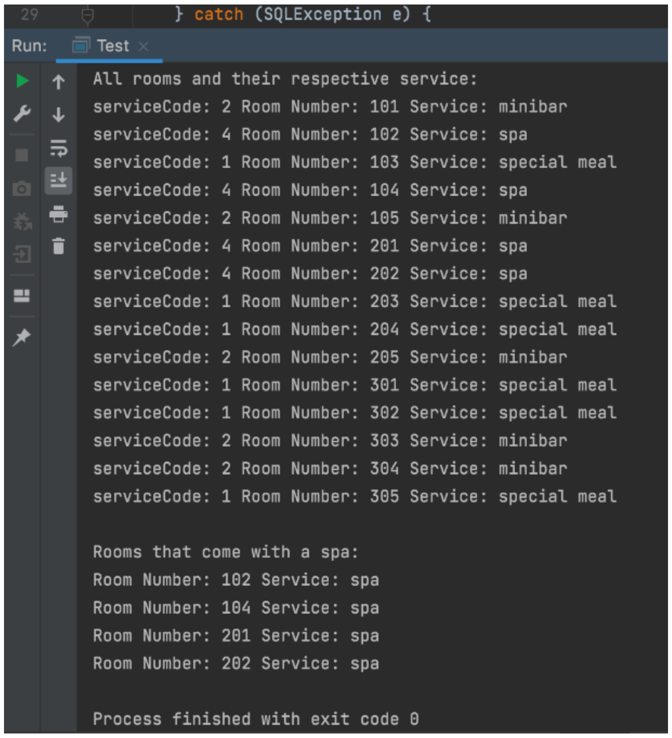


```
/Users/ckim/Library/Java/JavaVirtualMachines/openjdk-1
All double rooms:
Room Number: 103 Type: double Price: 100
Room Number: 104 Type: double Price: 105
Room Number: 203 Type: double Price: 95
Room Number: 204 Type: double Price: 115

Cheapest double room:
Room Number: 203 Type: double Price: 95

Process finished with exit code 0
```

2. Finding all rooms that come with a spa service:



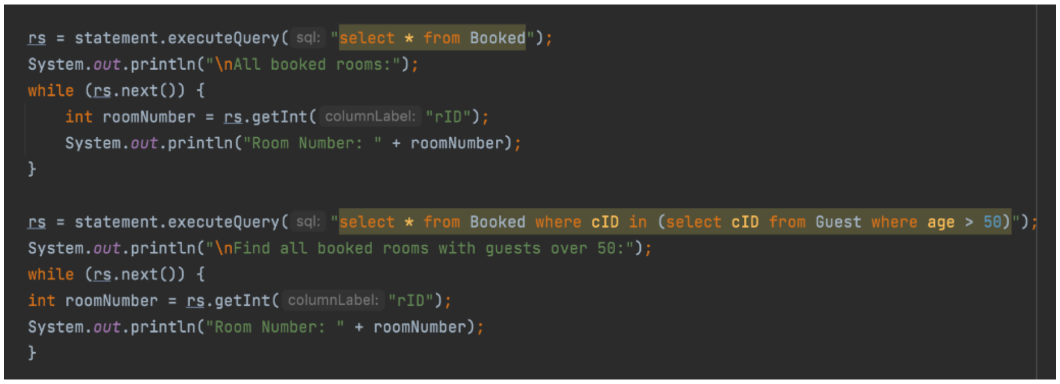
```
rs = statement.executeQuery( sql: "select * from Service");
System.out.println("All rooms and their respective service:");
while (rs.next()) {
    int serviceCode = rs.getInt( columnLabel: "serviceCode");
    int number = rs.getInt( columnLabel: "rID");
    String serv = rs.getString( columnLabel: "service");
    System.out.println("serviceCode: " + serviceCode + " Room Number: " + number + " Service: " + serv);

}
rs = statement.executeQuery( sql: "select * from Service where service='spa'");
        System.out.println("\nRooms that come with a spa:");
        while (rs.next()) {
            int roomNumber = rs.getInt( columnLabel: "rID");
            String service = rs.getString( columnLabel: "service");
            System.out.println("Room Number: " + roomNumber + " Service: " + service);
        }
```



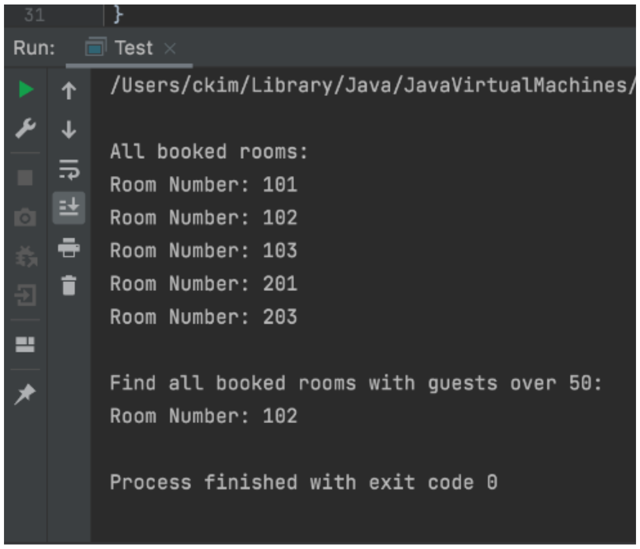
```
        } catch (SQLException e) {
```

```
All rooms and their respective service:
serviceCode: 2 Room Number: 101 Service: minibar
serviceCode: 4 Room Number: 102 Service: spa
serviceCode: 1 Room Number: 103 Service: special meal
serviceCode: 4 Room Number: 104 Service: spa
serviceCode: 2 Room Number: 105 Service: minibar
serviceCode: 4 Room Number: 201 Service: spa
serviceCode: 4 Room Number: 202 Service: spa
serviceCode: 1 Room Number: 203 Service: special meal
serviceCode: 1 Room Number: 204 Service: special meal
serviceCode: 2 Room Number: 205 Service: minibar
serviceCode: 1 Room Number: 301 Service: special meal
serviceCode: 1 Room Number: 302 Service: special meal
serviceCode: 2 Room Number: 303 Service: minibar
serviceCode: 2 Room Number: 304 Service: minibar
serviceCode: 1 Room Number: 305 Service: special meal

Rooms that come with a spa:
Room Number: 102 Service: spa
Room Number: 104 Service: spa
Room Number: 201 Service: spa
Room Number: 202 Service: spa

Process finished with exit code 0
```

3. Finding all rooms that are booked by someone over the age of 50:



```
rs = statement.executeQuery( sql: "select * from Booked");
System.out.println("\nAll booked rooms:");
while (rs.next()) {
    int roomNumber = rs.getInt( columnLabel: "rID");
    System.out.println("Room Number: " + roomNumber);
}

rs = statement.executeQuery( sql: "select * from Booked where cID in (select cID from Guest where age > 50)");
System.out.println("\nFind all booked rooms with guests over 50:");
while (rs.next()) {
int roomNumber = rs.getInt( columnLabel: "rID");
System.out.println("Room Number: " + roomNumber);
}
```



```
}
Run:   Test ×
/Users/ckim/Library/Java/JavaVirtualMachines/

All booked rooms:
Room Number: 101
Room Number: 102
Room Number: 103
Room Number: 201
Room Number: 203

Find all booked rooms with guests over 50:
Room Number: 102

Process finished with exit code 0
```

* Last modification date: 11/13/21