Group 7 Database Final Project

Views

Caballero-Employees

As you might have guessed, this view returns every employee. The usefulness of this view can be applied to a number of different situations. For instance, if someone in a managerial or administrative situation needed to find a specific employee, then they would be able to by using this view. Another reason would be to verify that a new employee has been added to the list or to verify that the update of an existing employee's information was successful. In addition, if an employee needed to look up their own or another employee's ID number or any other information within the scope of the table, then they could do that.

```
create view vw_employee as
SELECT
        employee.EmployeeID as "Employee ID",
        employee.FName as "First Name",
        employee.LName as "Last Name",
        employee.JoinDate as "Join Date",
        manager.FName as "Manager First Name",
        manager.LName as "Manager Last Name",
        job.JobName as "Job Name",
        department.DepartmentName as "Department Name"
FROM (employee, manager, job, department)
join manager as Manager_ID on employee.ManagerID=manager.ManagerID
join job as Job_ID on employee.JobID=job.JobID
join department as Department_ID on employee.DepartmentID=department.DepartmentID
GROUP BY employee.EmployeeID
ORDER BY employee.LName;
```

```
select*from vw_employee;
```

Employee ID	First Name	Last Name	Join Date	Manager ID	Job ID	Department ID
75718	Wake	Belliss	2016-12-21	550024	60624	910002
75556	Friederike	Borgesio	2018-04-11	550013	60618	910005
75946	Dacia	Brailsford	2015-09-05	550022	60632	910006
75960	Rosamund	Caruth	2012-03-05	550015	60625	910002
75959	Gunter	Corish	2012-02-05	550020	60631	910006
75460	Joelly	Dyas	2014-10-08	550024	60626	910002
75709	Thurston	Fincham	2013-03-18	550017	60619	910005
75841	Mose	Geffcock	2011-03-25	550014	60623	910001
75801	Lowe	Geggie	2011-05-05	550021	60629	910004
75174	Igor	Gillott	2019-04-26	550018	60627	910003
75813	Rafaello	Hemstead	2010-05-13	550017	60617	910005
75920	Leola	Hincks	2013-07-11	550013	60618	910005
75744	Joshuah	Huband	2015-04-26	550014	60620	910001
75757	Fannie	Hullbrook	2015-10-10	550015	60624	910002
75393	Giffer	Inmett	2013-03-03	550023	60627	910003
75287	Martainn	Ipsgrave	2014-12-11	550021	60628	910004
75115	Kaylyn	Jickells	2013-02-14	550022	60631	910006
75996	Myles	Kenewell	2011-11-15	550014	60621	910001
75061	Maribel	Klauber	2020-11-12	550016	60622	910001
75201	Brucie	Laraway	2010-02-03	550016	60622	910001
75032	Mayer	McLucas	2014-11-27	550019	60630	910004
75445	Georgy	Pechet	2015-05-13	550016	60620	910001
75905	Royal	Pymm	2013-04-20	550014	60623	910001
75935	Demott	Rosenfarb	2011-11-20	550013	60618	910005
75796	Mead	Seadon	2015-01-25	550019	60629	910004
75837	Anette	Tidcombe	2016-04-12	550014	60621	910001

w_employee 1

Caballero-Active Reservations

This view returns any reservations that are currently active. Active in this case means that the end date isn't past the current date. The current date being used in this particular sample run is 12/08/2020. Another thing to note is that aside form the reservation ID, a few of the entries present in the sample run are the same. This is from testing the Add Room Reservation stored procedure with the same data.

As for why this is important for the database, there are a few scenarios that could play out where this view would be needed. For instance, when a guest is checking into their room, this view could be used to verify that the guest has a reservation. This view can also be used to make sure that reservations don't conflict with each other. This can also be used at checkout when the guest needs to pay since the amount due is a part of this view.

```
create view vw_active_reservations as
SELECT
    reservations.ReservationID as "Reservation ID",
    reservations.Service as "Service",
    employee.LName as "Employee Last Name",
    guests.Lname as "Guest Last Name",
    reservations.StartDate as "Start Date",
    reservations.StartTime as "Start Time",
    reservations.EndDate as "End Date",
    reservations.EndTime as "End Time",
    payment.Amount as "Amount Due",
    AmenitiesListID as "Amenities List ID"
FROM
    (reservations, payment, guests, employee)
join payment as payment_ID on reservations.PaymentID=payment.PaymentID
join guests as guest_ID on reservations.GuestID=guests.GuestID
join employee as employee_ID on reservations.EmployeeID=employee.EmployeeID
WHERE
    ReservationActive=1
GROUP BY reservations.ReservationID;

select*from vw_active_reservations;
```

```
42 •  select*from vw_active_reservations;
43
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Reservation ID	Service	Employee Last Name	Guest Last Name	Start Date	Start Time	End Date	End Time	Amount Due	Amenities List ID
2136814	Room	Hullbrook	Miller	2020-12-08	15:30:00	2020-12-11	10:30:00	958.31	57743165
2314083	Room	Hullbrook	Miller	2020-12-08	15:30:00	2020-12-11	10:30:00	958.31	57743165
2433773	Room	Hullbrook	Davis	2020-12-04	11:38:00	2020-12-05	09:18:00	2524.62	729300
2496136	Yoga	Brailsford	Jones	2020-12-11	11:00:00	2020-12-11	12:00:00	4144.56	85730823
2591058	Room	Hullbrook	Jackson	2020-12-08	15:30:00	2020-12-11	10:30:00	958.31	57743165
2644580	Room	Hullbrook	Jackson	2020-12-08	15:30:00	2020-12-11	10:30:00	958.31	57743165
2955411	Room	Hullbrook	Jackson	2020-12-08	15:30:00	2020-12-11	10:30:00	958.31	57743165
2979320	Dinner	Fincham	Davis	2020-12-04	18:00:00	2020-12-04	20:00:00	3890.99	85730823

Caballero-Greater than $1000

This view returns any reservations that have more than $1000 as well as how much is due. The usefulness of this view resides more in the accounting side of things. Someone working in accounting could use this information to establish a priority hierarchy for charging guests. This can also be used for identifying guests with upstanding charges.

```
create view vw_greater_than_one_thousand as
SELECT
    reservations.ReservationID as "Reservation ID",
    guests.Lname as "Guest Last Name",
    payment.Amount as "Amount Due"
FROM
    (reservations, guests, payment)
join reservations as `payment_ID` on reservations.PaymentID=payment.PaymentID
join reservations as `guest_ID` on reservations.GuestID=guests.GuestID
WHERE
    payment.Amount > 1000
GROUP BY
    reservations.ReservationID
ORDER BY
    payment.Amount;

select*from vw_greater_than_one_thousand;
```

Reservation ID	Guest Last Name	Amount Due
2925621	Miller	1062.93
2559348	Jones	1416.66
2587819	Taylor	1416.66
2403776	Smith	1940.09
2455541	Harris	2319.34
2624169	Johnson	2319.34
2433773	Davis	2524.62
2671413	Lee	3645.44
2979320	Davis	3890.99
2496136	Jones	4144.56
2225410	Smith	4399.01

Caballero-Room and Room Type

This view displays all the views available as well as each of their room types. This could prove useful for either an employee or a guest. For an employee, this could be used to help a guest find the type of room they are looking for. Using this information will help make the process of creating new room reservations a bit easier. For a guest, if this was displayed on the hotel's website, then a guest could go into making a reservation knowing exactly what they want, which will also make the process of making a room reservation go much faster.

```sql
create view roomAndRoomType as
SELECT
    room.RoomNumber as "Room Number",
    roomtype.RoomTypeName as "Room Type",
    roomtype.MaxGuestQuantity as "Max Guest Quantity",
    roomtype.CostPerNight as "Cost Per Night"
FROM
    (room, roomtype)
join roomtype as roomtype_ID on room.RoomTypeID=roomtype.RoomTypeID
GROUP BY room.RoomNumber;

select*from roomAndRoomType;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Room Number	Room Type	Max Guest Quantity	Cost Per Night
101	Studio	2	80.00
102	Studio	2	80.00
103	Queen	3	100.00
104	Studio	2	80.00
105	King	5	150.00
106	King	5	150.00
107	Studio	2	80.00
108	Studio	2	80.00
109	Queen	3	100.00
110	King	5	150.00
201	King	5	150.00
202	Queen	3	100.00
203	Queen	3	100.00
204	Studio	2	80.00
205	Studio	2	80.00
206	Studio	2	80.00
207	King	5	150.00
208	King	5	150.00
209	Studio	2	80.00
210	Studio	2	80.00

Delgado - Guest View

This view shows the information for the Guest table. All guest information will be shown and presented to the user with column names that properly represent the data within the column. Columns such as “FName” and “LName” will be shown as “First Name” and “Last

Name” respectively. This view is specifically useful to see if multiple rooms are booked under the same name, if a guest is blacklisted from staying at the hotel and to show general information regarding the guests at the hotel.

```
SELECT GuestID AS "Guest ID", FName AS "First Name", LName AS "Last Name",
Email, RoomNumber AS "Room Number", Blacklisted FROM `GuestsView`
```

```
SELECT GuestID AS "Guest ID", FName AS "First Name", LName AS "Last Name", Email, RoomNumber AS "Room Number", Blacklisted FROM
`GuestsView`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

Guest ID	First Name	Last Name	Email	Room Number	Blacklisted
2347	Mary	Jones	maryjane@yahoo.com	104	1
2356	Jennifer	Miller	jennifill123@gmail.com	107	0
2357	Anthony	Young	younganth321@yahoo.com	210	0
2361	Joseph	Thomas	jthoms420@gmail.com	202	0
2635	Richard	Anderson	randerson@yahoo.com	201	1
3443	Michael	Davis	mikedave@hotmail.com	106	0
3454	Jessica	Martin	jessmua69@yahoo.com	206	0
4523	James	Smith	james345@gmail.com	101	0
5225	Patricia	Brown	Pattybrown00@gmail.com	105	0
5437	Daniel	Garcia	danisaG@hotmail.com	208	1
5622	Linda	Wilson	WilsonL8@yahoo.com	108	0
6832	Barbara	Jackson	jackbar360@gmail.com	203	0
6853	William	Moore	morewill10@yahoo.com	109	0
7523	Robert	Williams	robwill@yahoo.com	103	1
7633	Thomas	White	thomsthewhite54@gmail.com	204	0
7653	Karen	Thompson	thompskar2020@gmail.com	207	0
8247	John	Johnson	johnsonJJ@hotmail.com	102	1
8454	Susan	Harris	harriss20@hotmail.com	205	0
9236	Lisa	Lee	leelee@hotmail.com	209	0
9345	David	Taylor	daytay11@hotmail.com	110	0

Delgado - Sponsors View

This view is useful for seeing the Vendors or Sponsors who supply the hotel with items for the guests rooms. The data that is presented in the query is related to the Sponsor table where the Sponsor ID is directly related to the name of the sponsor that is supplying the items to the hotel. The reason that this view is so succinct in presented data is because a user who runs this query can simply see that if they needed to order a certain supply, you can quickly look to the Sponsor ID that provides that supply, then go to the Sponsor table for further information

regarding that Sponsor. This view makes it easy to look up the product supplied by the sponsor for ordering purposes.

```
1 SELECT SponsorID AS "Sponsor ID", SupplyName AS "Supply Name", SuppliesID AS
"Supplies ID" FROM `SponsorView`
```

Showing rows 0 - 9 (10 total, Query took 0.0019 seconds.)

```
SELECT SponsorID AS "Sponsor ID", SupplyName AS "Supply Name", SuppliesID AS "Supplies ID" FROM `SponsorView`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

Sponsor ID	Supply Name	Supplies ID
6649155	Coffee	5
8746489	Soap	1
15622664	Sheets	4
22377034	Toothpaste	6
35063219	TV	7
37351448	Towels	3
62316949	Glassware	8
73152702	Pillows	9
73218454	Toilet Paper	10
96861695	Shampoo	2

Stored Procedures

Delgado and Perez - Select Guest

This stored procedure is used to allow a user to look up the information for a user stored in the Guest table simply by querying with the guests first name. This information would be useful for contacting a hotel guest with the proper information regarding their reservation. This stored procedure allows the user to see the information presented to them with proper column names, rather than the column names that were given in SQL code.

```
DELIMITER $$
CREATE PROCEDURE SelectGuest(IN Firstname varchar(16))
BEGIN
    SELECT GuestID AS "Guest ID", Fname AS "First Name",
    Lname AS "Last Name", Email, RoomNumber AS "Room Number",
    Blacklisted FROM Guests WHERE Fname = Firstname;
END $$

DELIMITER ;
```

```
CALL SelectGuest('Michael')
```

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

Guest ID	First Name	Last Name	Email	Room Number	Blacklisted
3443	Michael	Davis	mikedave@hotmail.com	106	0

Delgado - Employee Promotion

This stored procedure is to allow a user to add an employee's promoted status and update which department they're working in and their manager ID. The stored procedure was created successfully but we were unsuccessful in querying data with it.

```
CREATE PROCEDURE EmployeePromotion(IN Employee_ID int(8), IN FirstName varchar(16), IN LastName varchar(16), IN NewManagerID
int(6)) BEGIN DECLARE FirstName varchar(16); DECLARE LastName varchar(16); DECLARE NewManagerID int(6); SELECT FName INTO
FirstName FROM Employee WHERE EmployeeID=EmployeeID; SELECT LName INTO LastName FROM Employee WHERE EmployeeID=Employee_ID;
SELECT FLOOR(RAND()*(910010-910000+1))+910000 INTO NewManagerID FROM Employee Where EmployeeID=Employee_ID; INSERT INTO Manager
(FName, LName, ManagerID) VALUES (FirstName, LastName, NewManagerID); END
```

Delgado - Payment Type Checker

This stored procedure allows a user to run a query that will show payments of a certain type such as checks, VISA, MasterCard and more. This information could be extremely useful for "vault" workers who are responsible for keeping track of the money that has come in and out of the hotel. In the screenshots, the example I've provided is checking for payments made with checks. Checks are important to keep track of and usually need to be held for a few days time, if not longer, for processing and making sure that the check doesn't bounce. The stored procedure shows the amount due, the payment ID and the payment type so it is easy to find the guest associated with this payment in case any issues were to arise with their payment.

```
DELIMITER $$

CREATE PROCEDURE PaymentTypeLookup(IN PaymentCheck varchar(16))
BEGIN
    SELECT PaymentID AS "Payment ID", Amount AS "Amount", PaymentType AS "Payment Type" FROM Payment
    WHERE PaymentType = PaymentCheck
END $$

DELIMITER ;
```

```
CALL PaymentTypeLookup('Check')
```

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

Payment ID	Amount	Payment Type
54963966	941.36	Check
57806020	2957.65	Check
68055266	2524.62	Check
76995303	958.31	Check
81591894	4399.01	Check

Delgado - Add New Employee

This stored procedure allows a user to add a new employee to the Employee table. The user inputs data for each column within the table and the data for the DepartmentID, JobID and ManagerID must exist in the tables that those entities are primary keys in. All columns must have data entered in when the user calls this stored procedure. Columns cannot be left null upon calling this stored procedure or the call will result in an error. This stored procedure is useful for adding a user or multiple users to the Employee table, for example, after having just hired an employee or group of employees for the hotel.

```
DELIMITER $$
CREATE PROCEDURE AddNewEmployee(IN DepartmentID int(8), IN EmployeeID int(8), IN FName varchar(16), IN JobID int(8), IN JoinDate datetime, IN LName varchar(16), IN ManagerID varchar(16))
BEGIN
    INSERT INTO Employee (DepartmentID, EmployeeID, FName, JobID, JoinDate, LName, ManagerID)
    VALUES (DepartmentID, EmployeeID, FName, JobID, JoinDate, LName, ManagerID);
END $$
DELIMITER ;
```

```
CALL AddNewEmployee('910004','99999','Justin','60630','2020-12-09','Delgado','550019')
```

	EmployeeID	FName	LName	JoinDate	ManagerID	JobID	DepartmentID
Edit Copy Delete	75935	Demott	Rosenfarb	2011-11-20	550013	60618	910005
Edit Copy Delete	75946	Dacia	Brailsford	2015-09-05	550022	60632	910006
Edit Copy Delete	75959	Gunter	Corish	2012-02-05	550020	60631	910006
Edit Copy Delete	75960	Rosamund	Caruth	2012-03-05	550015	60625	910002
Edit Copy Delete	75996	Myles	Kenewell	2011-11-15	550014	60621	910001
Edit Copy Delete	99999	Justin	Delgado	2020-12-09	550019	60630	910004

Delgado - Employee Department Swap

This stored procedure allows a user to switch the department of an employee by specifying an EmployeeID and then updating DepartmentID in the Employee table. This is an important operation for a user to have access to because employees will often be trained in more than one position, in more than one department. By making this stored procedure, a user can easily update an employee's department which can then assist someone like a scheduling manager who can then sort the employees by DepartmentID to see who all is working in a department. (When creating this stored procedure, I wasn't able to run it using PHPMyAdmin but sent it to Nick and it worked first try in MySQLWorkbench.)

```
DELIMITER //
create procedure Employee_Dept_Swap(IN employee_ID int, IN department_ID int)
BEGIN
    UPDATE employee SET DepartmentID=department_ID WHERE EmployeeID=employee_ID;
END//
DELIMITER ;

call Employee_Dept_Swap(75032,910004);
```

EmployeeID	FName	LName	JoinDate	ManagerID	JobID	DepartmentID
75032	Mayer	McLucas	2014-11-27	550019	60630	910004
75061	Maribel	Klauber	2020-11-12	550016	60622	910001
75115	Kaylyn	Jickells	2013-02-14	550022	60631	910006
75174	Igor	Gillott	2019-04-26	550018	60627	910003
75287	Martainn	Ipsgrave	2014-12-11	550021	60628	910004
75321	Erin	Umpleby	2014-06-01	550013	60617	910005
75335	Arvie	Watmough	2010-05-12	550019	60628	910004
75393	Giffer	Inmett	2013-03-03	550023	60627	910003
75445	Georgy	Pechet	2015-05-13	550016	60620	910001
75460	Joelly	Dyas	2014-10-08	550024	60626	910002

EmployeeID	FName	LName	JoinDate	ManagerID	JobID	DepartmentID
75032	Mayer	McLucas	2014-11-27	550019	60630	910003
75061	Maribel	Klauber	2020-11-12	550016	60622	910001
75115	Kaylyn	Jickells	2013-02-14	550022	60631	910006
75174	Igor	Gillott	2019-04-26	550018	60627	910003
75287	Martainn	Ipsgrave	2014-12-11	550021	60628	910004
75321	Erin	Umpleby	2014-06-01	550013	60617	910005
75335	Arvie	Watmough	2010-05-12	550019	60628	910004
75393	Giffer	Inmett	2013-03-03	550023	60627	910003
75445	Georgy	Pechet	2015-05-13	550016	60620	910001
75460	Joelly	Dyas	2014-10-08	550024	60626	910002

employee 5

Caballero-Department Staff

This stored procedure takes in a department name and returns every employee within a department. This could be used to do a few things. For instance, if an employee from one department needed help from an employee in a different department, then they could use this stored procedure to find a list of people to help them out. In addition, if this were integrated into a piece of software, it could be used as a filter option for searching for employees. Also, a

department manager could also use this to search for a specific employee under their management. They might want to do this if they need to find a specific employee to do a job for them.

```
DELIMITER //
create procedure departmentStaff(IN department_Name varchar(15))
BEGIN
    SELECT
        employee.FName as "First Name",
        employee.LName as "Last Name",
        department.DepartmentName as "Department Name"
    FROM (employee, department)
    join employee as employeeTable on department.DepartmentID=employee.DepartmentID
    WHERE
        department.DepartmentName=department_Name AND department.DepartmentID=employee.DepartmentID
    GROUP BY employee.FName, employee.LName;
END//
DELIMITER ;

CALL departmentStaff("Human Resources");
```

First Name	Last Name	Department Name
Mayer	McLucas	Human Resources
Martainn	Ipsgrave	Human Resources
Arvie	Watmough	Human Resources
Mead	Seadon	Human Resources
Lowe	Geggie	Human Resources

Caballero- Delete Employee

This stored procedure removes an employee from the employee list. Despite its simplicity, this can be used in a few different ways. The most obvious way would be to remove the employee form the list when they are not employed with the hotel for whatever reason. This could also be used in conjunction with a piece of software. In this instance, this could be used as part of a delete functionality to create a CRUD system.

```
DELIMITER //
create procedure deleteEmployee(IN employeeID int)
BEGIN
    DELETE FROM employee WHERE employee.EmployeeID=employeeID;
END //
DELIMITER ;

CALL deleteEmployee(75201);
```

```
82 •  select*from employee;
83
```

Result Grid | Filter Rows: | Edit: | Export/Import:

EmployeeID	FName	LName	JoinDate	ManagerID	JobID	DepartmentID
75032	Mayer	McLucas	2014-11-27	550019	60630	910004
75061	Maribel	Klauber	2020-11-12	550016	60622	910001
75115	Kaylyn	Jickells	2013-02-14	550022	60631	910006
75174	Igor	Gillott	2019-04-26	550018	60627	910003
75201	Brucie	Laraway	2010-02-03	550016	60622	910001
75287	Martainn	Ipsgrave	2014-12-11	550021	60628	910004
75321	Erin	Umpleby	2014-06-01	550013	60617	910005
75335	Arvie	Watmough	2010-05-12	550019	60628	910004
75393	Giffer	Inmett	2013-03-03	550023	60627	910003
75445	Georgy	Pechet	2015-05-13	550016	60620	910001

```
CALL deleteEmployee(75201);

select*from employee;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

EmployeeID	FName	LName	JoinDate	ManagerID	JobID	DepartmentID
75032	Mayer	McLucas	2014-11-27	550019	60630	910004
75061	Maribel	Klauber	2020-11-12	550016	60622	910001
75115	Kaylyn	Jickells	2013-02-14	550022	60631	910006
75174	Igor	Gillott	2019-04-26	550018	60627	910003
75287	Martainn	Ipsgrave	2014-12-11	550021	60628	910004
75321	Erin	Umpleby	2014-06-01	550013	60617	910005
75335	Arvie	Watmough	2010-05-12	550019	60628	910004
75393	Giffer	Inmett	2013-03-03	550023	60627	910003
75445	Georgy	Pechet	2015-05-13	550016	60620	910001
75460	Joelly	Dyas	2014-10-08	550024	60626	910002
75556	Friederike	Borgesio	2018-04-11	550013	60618	910005
75594	Linette	Whitchurch	2019-02-14	550017	60619	910005
75698	Stoddard	Waterhouse	2011-01-07	550017	60617	910005
75709	Thurston	Fincham	2013-03-18	550017	60619	910005
75718	Wake	Belliss	2016-12-21	550024	60624	910002
75744	Joshuah	Huband	2015-04-26	550014	60620	910001
75757	Fannie	Hullbrook	2015-10-10	550015	60624	910002
75796	Mead	Seadon	2015-01-25	550019	60629	910004
75801	Lowe	Geggie	2011-05-05	550021	60629	910004
75813	Rafaello	Hemstead	2010-05-13	550017	60617	910005
75837	Anette	Tidcombe	2016-04-12	550014	60621	910001
75841	Mose	Geffcock	2011-03-25	550014	60623	910001
75905	Royal	Pymm	2013-04-20	550014	60623	910001
75920	Leola	Hincks	2013-07-11	550013	60618	910005
75935	Demott	Rosenfarb	2011-11-20	550013	60618	910005
75946	Dacia	Brailsford	2015-09-05	550022	60632	910006
75959	Gunter	Corish	2012-02-05	550020	60631	910006
75960	Rosamund	Caruth	2012-03-05	550015	60625	910002
75996	Myles	Kenewell	2011-11-15	550014	60621	910001
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Caballero-Add Room Reservation

This stored procedure adds a room reservation to the reservation list. It’s purpose may be very niche, but it makes up for it with usefulness. Without a way to quickly make room reservations, then the hotel wouldn’t be able to properly function, or at the very least it wouldn’t be able to

function efficiently. By doing this, everything on the guest side of the hotel's operations can flow much quicker and without too many issues.

```
DELIMITER //
create procedure addRoomReservation(IN employee_ID int, IN guest_ID int, IN start_Date date, IN start_Time time,
IN end_Date date, IN end_Time time, IN payment_ID int, IN amenities_ID int, IN room_ID int, IN roomkey_ID int)
BEGIN
    IF NOT EXISTS(select*from roomkey, room where room.RoomNumber=room_ID
    AND roomkey.RoomKeyID=roomkey_ID AND roomkey.Active=1) THEN
        insert into reservations (ReservationID, Service, EmployeeID, GuestID, StartDate, StartTime,
        EndDate, EndTime, PaymentID, ReservationActive, AmenitiesListID)
        values ((FLOOR(RAND()*(2999999-2000000+1))+2000000), 'Room', employee_ID, guest_ID,
        start_Date, start_Time, end_Date, end_Time, payment_ID, 1, amenities_ID);
        UPDATE guests SET RoomNumber=room_ID WHERE GuestID=guest_ID;
        select*from guests where GuestID=guest_ID;
        UPDATE room SET RoomKeyID=roomkey_ID WHERE RoomNumber=room_ID;
        select*from room where RoomNumber=room_ID;
        UPDATE roomkey SET roomkey.Active=1 WHERE RoomKeyID=roomkey_ID;
        select*from roomkey where RoomKeyID=roomkey_ID;
    END IF;
END //
DELIMITER ;
```

```
104 •   CALL addRoomReservation(75757, 2356, '2020-12-08', '15:30:00', '2020-12-11', '10:30:00',76995303, 57743165, 205, 14);
105
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

GuestID	Fname	Lname	Email	RoomNumber	Blacklisted
2356	Jennifer	Miller	jennifill123@gmail.com	205	0

```
104 •   CALL addRoomReservation(75757, 2356, '2020-12-08', '15:30:00', '2020-12-11', '10:30:00',76995303, 57743165, 205, 14);
105
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

RoomNumber	RoomTypeID	FloorNumber	RoomKeyID	CostOfRoomPerNight
205	205	2	14	NULL

```
104 •   CALL addRoomReservation(75757, 2356, '2020-12-08', '15:30:00', '2020-12-11', '10:30:00',76995303, 57743165, 205, 14);
105
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

RoomKeyID	Barcode	Active	IssueDate
14	548102596	1	2020-12-15

Caballero-Checkout

This stored procedure is used to checkout a guest. It also marks that the guest's reservation is inactive and that the room key associated with the room the guest was staying is also marked inactive. This stored procedure is also a bit niche, but because of it the hotel can operate at maximum efficiency. With the help of this stored procedure, guests can be checked out relatively easily and without a lot of time. On the hotel's side, the staff don't have to go looking for a reservation or a room key to mark it as inactive. This saves a lot of time that can be used for adding to the productivity of the hotel.

```
DELIMITER //
CREATE PROCEDURE checkout(IN reservation_ID int,IN guest_ID int)
BEGIN
    DECLARE currentGuestRoomNumber INT;
    DECLARE currentRoomKeyID INT;
    SELECT RoomNumber INTO currentGuestRoomNumber FROM guests WHERE GuestID=guest_ID;
    SELECT RoomKeyID INTO currentRoomKeyID FROM room WHERE RoomNumber=currentGuestRoomNumber;
    UPDATE roomkey SET roomkey.Active=0 WHERE RoomKeyID=currentRoomKeyID;
    UPDATE reservations SET ReservationActive=0 WHERE ReservationID=reservation_ID;
END//
DELIMITER ;
```

ReservationID	Service	EmployeeID	GuestID	StartDate	StartTime	EndDate	EndTime	PaymentID	ReservationActive	AmenitiesListID
2025817	Room	75757	2356	2020-12-08	15:30:00	2020-12-11	10:30:00	76995303	1	57743165
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

RoomKeyID	Barcode	Active	IssueDate
14	548102596	1	2020-12-15
NULL	NULL	NULL	NULL

ReservationID	Service	EmployeeID	GuestID	StartDate	StartTime	EndDate	EndTime	PaymentID	ReservationActive	AmenitiesListID
2025817	Room	75757	2356	2020-12-08	15:30:00	2020-12-11	10:30:00	76995303	0	57743165
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

RoomKeyID	Barcode	Active	IssueDate
14	548102596	0	2020-12-15
NULL	NULL	NULL	NULL

Caballero-Amenities Packages

This stored procedure returns all the amenities package IDs of any packages with a specific feature. This was made with the intention of making it easier for a guest to decide on what package they would like. For instance, if a future guest is looking for a package with Wifi access, then this procedure would help with showing which packages have that feature.

```
DELIMITER //
create procedure amenitiesPackages(IN packageFeature varchar(30))
BEGIN
SELECT lower(packageFeature) into packageFeature;
    CASE
        WHEN packageFeature="free continental breakfast" THEN select AmenitiesListID from amenities_list WHERE FreeContinentalBreakfast=1;
        WHEN packageFeature="gym access" THEN select AmenitiesListID from amenities_list WHERE GymAcces=1;
        WHEN packageFeature="pool access" THEN select AmenitiesListID from amenities_list WHERE PoolAcces=1;
        WHEN packageFeature="parking" THEN select AmenitiesListID from amenities_list WHERE Parking=1;
        WHEN packageFeature="wifi access" THEN select AmenitiesListID from amenities_list WHERE WiFiAccess=1;
        ELSE SELECT 'Invalid input. Please make sure that this is a valid package name';
    END CASE;
END//
DELIMITER ;

call amenitiesPackages("parking");
```

AmenitiesListID
15742182
19149671
27950962
29352136
34480132
41543897
57743165
59325358
62745744
65275643

Caballero-Blacklist

This stored procedure will be used to blacklist any guests who are no longer welcome at the hotel. This stored procedure is simple, but is a powerful tool for ensuring that the environment of the hotel is pleasant for all guests. With this, it will be a lot easier for the hotel to keep those who aren’t welcome away from the hotel.

GuestID	Fname	Lname	Email	RoomNumber	Blacklisted
2347	Mary	Jones	maryjane@yahoo.com	206	1
2356	Jennifer	Miller	jennifill123@gmail.com	205	0
2357	Anthony	Young	younganth321@yahoo.com	206	0
2361	Joseph	Thomas	jthoms420@gmail.com	206	0
2635	Richard	Anderson	randerson@yahoo.com	206	1
3443	Michael	Davis	mikedave@hotmail.com	206	0
3454	Jessica	Martin	jessmua69@yahoo.com	206	0
4523	James	Smith	james345@gmail.com	206	0
5225	Patricia	Brown	Pattybrown00@gmail.com	206	0
5437	Daniel	Garcia	danisaG@hotmail.com	206	1
5622	Linda	Wilson	WilsonL8@yahoo.com	206	0
6832	Barbara	Jackson	jackbar360@gmail.com	206	0
6853	William	Moore	morewill10@yahoo.com	206	0
7523	Robert	Williams	robwill@yahoo.com	206	1

```
DELIMITER //
CREATE PROCEDURE blacklistGuest(IN Guest_ID int)
BEGIN
    UPDATE guests SET blacklisted=1 WHERE GuestID=Guest_ID;
END //
DELIMITER ;

select*from guests;

call blacklistGuest(2356);

select*from guests;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

GuestID	Fname	Lname	Email	RoomNumber	Blacklisted
2347	Mary	Jones	maryjane@yahoo.com	206	1
2356	Jennifer	Miller	jennifill123@gmail.com	205	1
2357	Anthony	Young	younganth321@yahoo.com	206	0
2361	Joseph	Thomas	jthoms420@gmail.com	206	0
2635	Richard	Anderson	randerson@yahoo.com	206	1
3443	Michael	Davis	mikedave@hotmail.com	206	0
3454	Jessica	Martin	jessmua69@yahoo.com	206	0
4523	James	Smith	james345@gmail.com	206	0
5225	Patricia	Brown	Pattybrown00@gmail.com	206	0
5437	Daniel	Garcia	danisaG@hotmail.com	206	1
5622	Linda	Wilson	WilsonL8@yahoo.com	206	0
6832	Barbara	Jackson	jackbar360@gmail.com	206	0
6853	William	Moore	morewill10@yahoo.com	206	0
7523	Robert	Williams	robwill@yahoo.com	206	1

Perez - Guest Amenities

This stored procedure is used to show the amenities that a hotel guest has access to for their reservation. This is useful to see if hotel guests are able to participate in certain amenities that are on the hotel premises. The query needs a first name to perform the query and the name of the guest/guests matching that name will show with their AmenitiesID and the range of amenities, along with the privileges they may or may not have in boolean form.

```
CREATE PROCEDURE GuestAmenities(IN FirstName varchar(16)) BEGIN Select Guests.FName as "First Name", Guests.LName as "Last
Name", Amenities_List.AmenitiesListID as "Amenities ID", Amenities_List.FreeContinentalBreakfast as "Free Breakfast",
Amenities_List.GymAccess as "Gym Access", Amenities_List.PoolAccess as "Pool Access", Amenities_List.Parking as "Parking",
Amenities_List.WiFiAccess as "Wifi Access" From (Guests, Reservations, Amenities_List) join Guests as guestsTable on
Reservations.GuestID=Guests.GuestID join Amenities_List as Amenities_ListTable on
Reservations.AmenitiesListID=Amenities_List.AmenitiesListID WHERE Guests.FName = FirstName GROUP BY Guests.Fname, Guests.Lname;
END
```

```
Call GuestAmenities('Anthony')
```

[Edit inline] [Edit] [

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

First Name	Last Name	Amenities ID	Free Breakfast	Gym Access	Pool Access	Parking	Wifi Access
Anthony	Young	729300	0	0	0	0	1

Perez - ActiveCheck

This stored procedure makes it really easy for employees to look up and see if a certain room they are looking for is either occupied or not. All the procedure needs to be inputted is a 1, which in our case, means that it is active and 0 that means that the room is not active. Along with this, the employee will see the room number, floor number and that same key for the room.

```
DELIMITER //
CREATE PROCEDURE ActiveCheck(IN Active_Check boolean)
BEGIN
    SELECT DISTINCT
        Room.RoomNumber as "Room",
        Room.FloorNumber as "Floor",
        RoomType.RoomTypeName as "Type",
        RoomKey.`Active` as "Active"
    From (Room, RoomType, RoomKey)
    join RoomType as RoomTypeTable on Room.RoomTypeID=RoomType.RoomTypeID
    join RoomKey as RoomKeyTable on Room.RoomKeyID=RoomKey.RoomKeyID
    WHERE
        RoomKey.`Active`=Active_Check;
END//
DELIMITER ;
```

```
CALL ActiveCheck('1')
```

Show all | Number of rows:

+ Options

Room	Floor	Type	Active
102	1	Studio	1
103	1	Queen	1
107	1	Studio	1
108	1	Studio	1
110	1	King	1
202	2	Queen	1
203	2	Queen	1
205	2	Studio	1
209	2	Studio	1
210	2	Studio	1

Perez - RoomFloors

This was a very simple procedure to show exactly how many rooms there are on each floor. This can be a great tool for new hires so that they can be familiar with the hotel and not get lost the first couple of days.

```mysql
DELIMITER //
CREATE PROCEDURE RoomFloors(IN Floor_Number INT(8))
BEGIN
    SELECT DISTINCT
        Room.FloorNumber as "Floor",
        Room.RoomNumber as "Room",
        RoomType.RoomTypeName as "Type"
    From (Room, RoomType)
    join RoomType as RoomTypeTable on Room.RoomTypeID=RoomType.RoomTypeID
    WHERE
        Room.FloorNumber=Floor_Number;
END//
DELIMITER ;
```

```mysql
CALL RoomFloors("1");
```

Result Grid | Filter Rows:

Floor	Room	Type
1	101	Studio
1	102	Studio
1	103	Queen
1	104	Studio
1	105	King
1	106	King
1	107	Studio
1	108	Studio
1	109	Queen
1	110	King