澳門大學

UNIVERSIDADE DE MACAU

UNIVERSITY OF MACAU

CISC3000 Course Project

Name/Student Number

Content

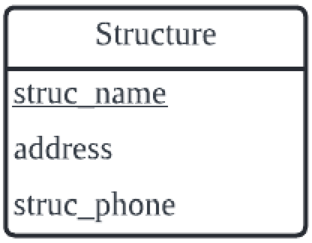
A. Application Domain

Databases are crucial in hotel management systems, storing and managing vital information such as room details, customer data, employee records, financial transactions, and facility services. By efficiently integrating and processing this data, hotels can not only efficiently manage their internal information and business within the structual organizaions, but also offer personalized customer services, optimize room utilization, simplify the booking process, and enhance overall management capabilities and customer satisfaction. Additionally, databases ensure the security and privacy of information, making them an indispensable technological support for modern hotel management.

B. Database Requirements

B.1 Entities

1. **Structure**
   Structure is the organizations or departments of the hotel with different functions.



Structure

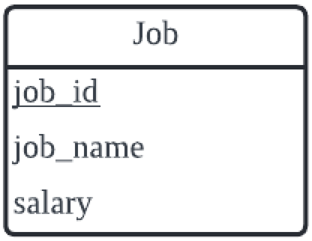
struc_name

address

struc_phone

- **struc_name**: (primary key) The name of structures in the hotel. It uniquely identifies different structures in the hotel.
- **address**: The address or position of specific structure.
- **struc_phone**: The phone number of specific structure.

2. **Job**
   Job is the job roles or positions within the hotel.



Job

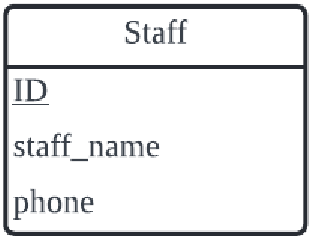
job_id

job_name

salary

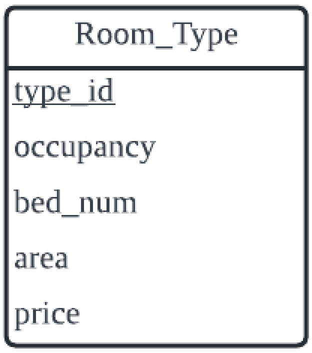
- **job_id**: (primary key) The ID of jobs in the hotel. It uniquely identifies different jobs in the hotel.
- **job_name**: The names of the job. The job name may be duplicated with others.
- **salary**: The salary for the job.

3. **Staff**
   Staff is all members working in the hotel.



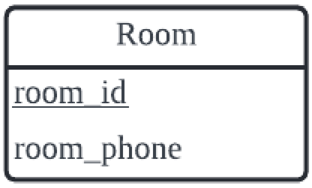
- **ID:** (primary key) The ID of staffs in the hotel. It uniquely identifies different staffs in the hotel.
- **staff_name**: The name of the staff. The name do not contain numbers or special characters. Each staff's name may be duplicated with others.
- **phone**: The phone number of the staff.

4. **Room_Type**
   Room_Type is the room types in the hotel.



- **type_id**: (primary key) The ID of room types. It uniquely identifies different room types in the hotel.
- **occupancy**: The room capacity or maximum occupancy of the room. It should be greater than 0.
- **bed_num**: The number of beds in the room. It should be greater than 0.
- **area**: The area (m^2) of the room. It should be greater than 0.
- **price**: The price of specific room type. It should be greater than 0.

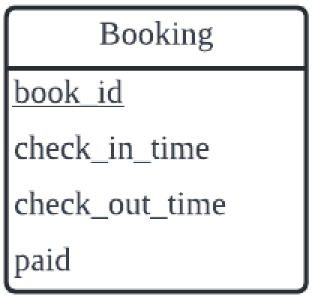
5. **Room**
   Room is the rooms in the hotel.



- **room_id**: (primary key) The ID of different rooms in the hotel. It uniquely identifies different rooms in the hotel.
- **room_phone**: The phone of the room.

6. Booking

Booking is the customer's hotel room booking or reservation orders.



Booking

book_id

check_in_time

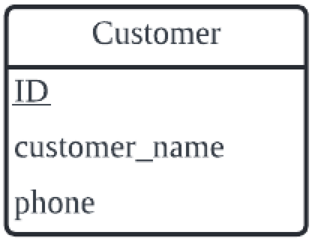
check_out_time

paid

- **book_id**: (primary key) The ID of room booking orders of the hotel. It uniquely identifies different booking rooms orders.
- **check_in_time**: Customer’s check-in time for the reservation. It is stored in the form of "YYYY-MM-DD hh:MM:ss". What’s more, it is not allowed to have overlapping Booking times in the same room.
- **check_out_time**: Customer’s check-in time for the reservation. It is stored in the form of "YYYY-MM-DD hh:MM:ss". What’s more, it is not allowed to have overlapping Booking times in the same room.
- **paid**: Check whether this order has been paid. The values of paid is either 1 or 0, indicating paid and unpaid respectively.

7. Customer

Customer is the customers or guests who booked a room or already stayed at the hotel.



Customer

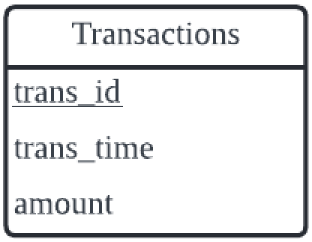
ID

customer_name

phone

- **ID**: (primary key) The ID of the customers. It uniquely identifies different customers.
- **customer_name**: The name of the customer. The name do not contain numbers or special characters. Each customer’s name may be duplicated with others.
- **phone**: The phone of the customer.

8. **Transactions**
   Transactions is the financial transaction records between the persons who made room bookings, and the hotel.



- **trans_id**: (primary key) The ID of the transaction records. It uniquely identifies different transaction records.
- **trans_time**: The transaction time. It is stored in the form of "YYYY-MM-DD hh:MM:ss".
- **amount**: The amount of the transaction. If the transaction is for a room booking order, the corresponding booking orders should display as paid.

B.2 Relationships

1. **Job_Struc**
   Job_Struc represents that Organizational structures manage job roles.
   Job_Struc is a Many-to-One relationship between Job and Structure. A job can only belong to one structure, while a structure could have multiple jobs.
   - **job_id**: (primary key) The foreign key referencing Job.
   - **struc_name**: the foreign key referencing Structure.

2. **Superv**
   Superv represents a hierarchical (supervisory) relationship among job roles.
   Superv is a Many-to-One relationship between Job and it's supervisory Job. A job can only have one supervisory job, while one supervisory job can supervise several jobs.
   - **job_id**: (primary key) The foreign key referencing Job.
   - **supervisory_job_id**: The foreign key referencing Job.

3. **Staff_Job**
   Staff_Job represents that the staff members have specific job roles.
   Staff_Job is a Many-to-One relationship between Staff and Job. A staff is associated with one job, while a job can associated with many (including 0) staffs.
   - **ID**: (primary key) The foreign key referencing Staff.
   - **job_id**: The foreign key referencing Job.

4. **Room_Typ**
   Room_Typ represents that rooms are classified by room type.
   Room_Typ is a Many-to-One relationship between Room and Room_Type. A room only belongs to one room type, while a room type can associated with many rooms.
   - **room_id**: (primary key) The foreign key referencing Room.
   - **type_id**: The foreign key referencing Room_Type.

5. **Book_Room**
   Book_Room represents that bookings are made for specific rooms.
   Book_Room is a Many-to-One relationship between Booking and Room. A booking order is only associated with one room, while a room can associated with many (including 0) booking orders.
   - **book_id**: (primary key) The foreign key referencing Booking.
   - **room_id**: The foreign key referencing Room.

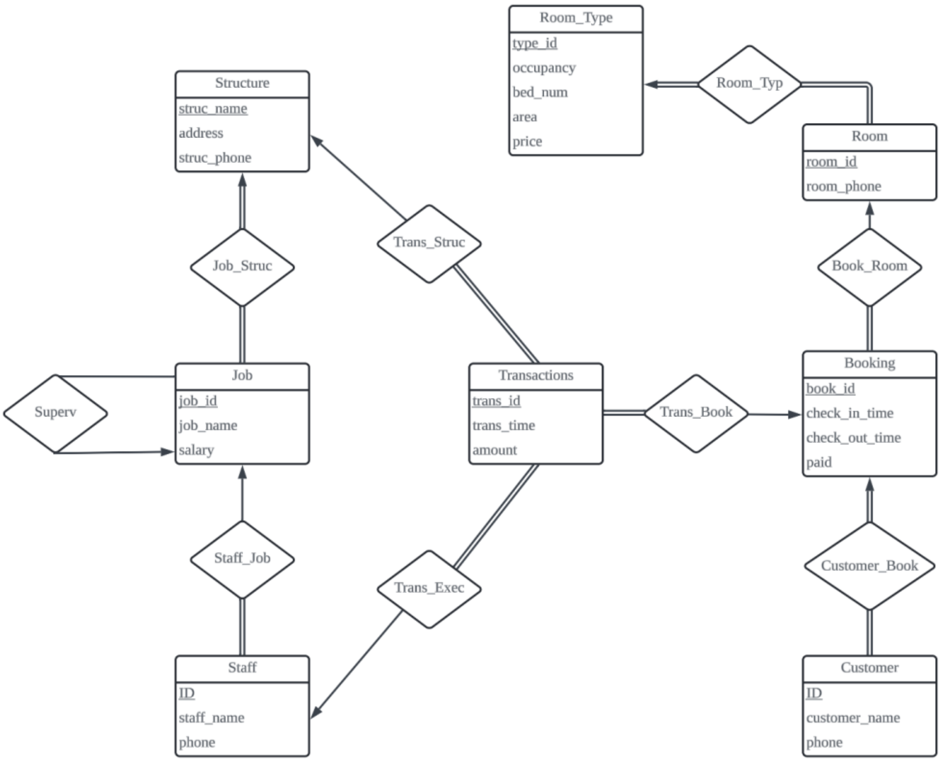
6. **Customer_Book**
   Customer_Book represents that bookings are made by customers.
   Customer_Book is a Many-to-One relationship between Customer and Booking. A customer can only have one booking order, while a booking order can be associated with many customers.
   - **ID**: (primary key) The foreign key referencing Customer.
   - **book_id**: The foreign key referencing Booking.

7. **Trans_Book**
   Trans_Book represents that transactions are financial entries for the bookings.
   Trans_Book is a Many-to-One relationship between Transactions and Booking. A transaction can be conducted by the people of one booking order, while the people of a booking order can conduct many (possibly 0) transactions.
   - **trans_id**: (primary key) The foreign key referencing Transactions.
   - **book_id**: The foreign key referencing Booking.

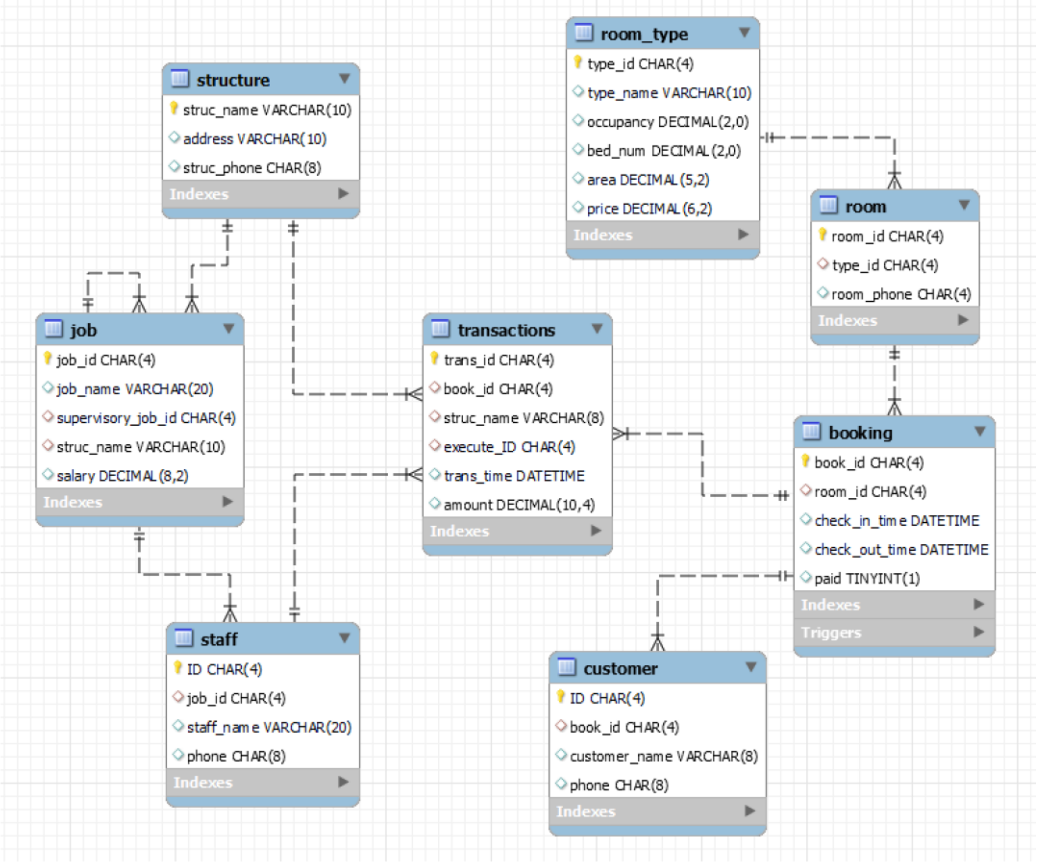
8. **Trans_Struc**
   Trans_Struc represents that transaction are recorded under specific hotel structures.
   Trans_Struc is a Many-to-One relationship between Transactions and Struc. A transaction record is for transacting with one structure, while a structure can be involved in many (possibly 0) transactions.
   - **trans_id**: (primary key) The foreign key referencing Transactions.
   - **struc_name**: the foreign key referencing Structure.

9. **Trans_Exec**
   Trans_Exec represents that transactions are executed by staff members.
   Trans_Exec is a Many-to-One relationship between Transactions and Staff. A transactions is executed by one staff, while a staff can execute many (possibly 0) transactions.
   - **trans_id**: (primary key) The foreign key referencing Transactions.
   - **execute_ID**: The foreign key referencing Staff.

C. ER Diagram



D. EER Diagram



E. DDL

E.1. Tables

When creating tables, we first create the tables with no foreigner keys, and then create the tables containing the foreigner key referencing previous tables.

```
-- 1.Structure
create table Structure (
struc_name varchar(10),
address varchar(10),
struc_phone char(8),
primary key (struc_name)
);
```

```
-- 2.Job
create table Job (
job_id char(4),
job_name varchar(20),
supervisory_job_id char(4),  #
struc_name varchar(10),  #
salary numeric(8,2) check (salary > 0),
primary key (job_id),
foreign key (supervisory_job_id) references Job(job_id),
foreign key (struc_name) references Structure (struc_name)
);
```

```
-- 3.Staff
create table Staff (
ID char(4),
job_id char(4),  #
staff_name varchar(20) check (staff_name regexp ’^[A-Za-z ]+$’),
phone char(8),
primary key (ID),
foreign key (job_id) references Job (job_id)
);
```

```
-- 4.Room_Type
create table Room_Type (
type_id char(4),
type_name varchar(10),
occupancy numeric (2, 0) check (occupancy > 0),
bed_num numeric (2, 0) check (bed_num > 0),
area numeric (5, 2) check (area > 0),
price numeric (6, 2) check (price > 0),
primary key (type_id)
);
```

```sql
-- 5.Room
create table Room (
room_id char(4),
type_id char(4),  #
room_phone char(4),
primary key (room_id),
foreign key (type_id) references Room_type (type_id)
);
```

```sql
-- 6.Booking
create table Booking (
book_id char(4),
room_id char(4),  #
check_in_time datetime,
check_out_time datetime,
paid bool,
primary key (book_id),
foreign key (room_id) references Room (room_id)
);
```

```sql
-- 7.Customer
create table Customer (
ID char(4),
book_id char(4),  #
customer_name varchar(8) check (customer_name regexp ’^[A-Za-z ]+$’),
phone char(8),
primary key (ID),
foreign key (book_id) references Booking (book_id)
);
```

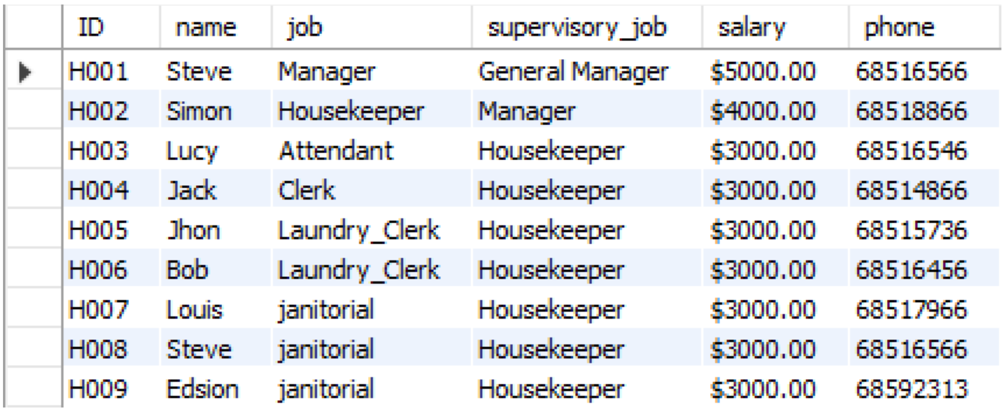
```sql
-- 8.Transactions
create table Transactions (
trans_id char(4),
book_id char(4),  #
struc_name  varchar(8),  #
execute_ID char(4),  #
trans_time datetime,
amount numeric(10,4),
primary key (trans_id),
foreign key (book_id) references Booking (book_id),
foreign key (struc_name) references Structure (struc_name),
foreign key (execute_ID) references Staff (ID)
);
```

E.2. Views

1. To view the detailed information of the staff in Rooms structure, such as ID, name, job, supervisory job, salary and phone number.

```
create view Staff_Infor (ID, name, job, supervisory_job, salary, phone) as
    select ID, staff_name, J.job_name, S.job_name, concat(’$’, J.salary), phone
    from Staff join (
    Job J left outer join Job S on (J.supervisory_job_id = S.job_id)
    ) on (Staff.job_id = J.job_id)
    where J.struc_name = ’Rooms’;
```

Results:

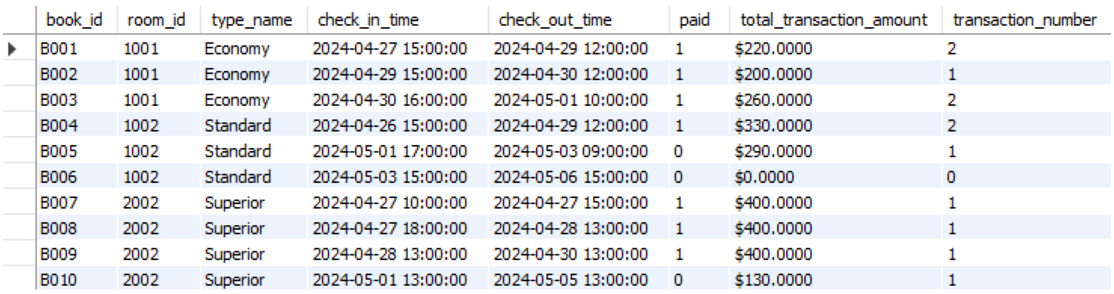


	ID	name	job	supervisory_job	salary	phone
▶	H001	Steve	Manager	General Manager	$5000.00	68516566
	H002	Simon	Housekeeper	Manager	$4000.00	68518866
	H003	Lucy	Attendant	Housekeeper	$3000.00	68516546
	H004	Jack	Clerk	Housekeeper	$3000.00	68514866
	H005	Jhon	Laundry_Clerk	Housekeeper	$3000.00	68515736
	H006	Bob	Laundry_Clerk	Housekeeper	$3000.00	68516456
	H007	Louis	janitorial	Housekeeper	$3000.00	68517966
	H008	Steve	janitorial	Housekeeper	$3000.00	68516566
	H009	Edsion	janitorial	Housekeeper	$3000.00	68592313

2. To view the room booking information and transactions summaries.

```
create view Book_Trans_Infor (book_id, room_id, type_name,
                              check_in_time, check_out_time, paid,
                              total_transaction_amount,
transaction_number) as
    select book_id, room_id,
        (select Typ.type_name
        from Room R join Room_Type Typ using (type_id)
        where R.room_id = B.room_id),
        check_in_time, check_out_time, paid,
        (select concat(’$’, ifnull(sum(amount), 0))
        from Transactions T1
        where T1.book_id = B.book_id),
        (select count(*)
        from Transactions T2
        where T2.book_id = B.book_id)
    from Booking B;
```

Results:



book_id	room_id	type_name	check_in_time	check_out_time	paid	total_transaction_amount	transaction_number
B001	1001	Economy	2024-04-27 15:00:00	2024-04-29 12:00:00	1	$220.0000	2
B002	1001	Economy	2024-04-29 15:00:00	2024-04-30 12:00:00	1	$200.0000	1
B003	1001	Economy	2024-04-30 16:00:00	2024-05-01 10:00:00	1	$260.0000	2
B004	1002	Standard	2024-04-26 15:00:00	2024-04-29 12:00:00	1	$330.0000	2
B005	1002	Standard	2024-05-01 17:00:00	2024-05-03 09:00:00	0	$290.0000	1
B006	1002	Standard	2024-05-03 15:00:00	2024-05-06 15:00:00	0	$0.0000	0
B007	2002	Superior	2024-04-27 10:00:00	2024-04-27 15:00:00	1	$400.0000	1
B008	2002	Superior	2024-04-27 18:00:00	2024-04-28 13:00:00	1	$400.0000	1
B009	2002	Superior	2024-04-28 13:00:00	2024-04-30 13:00:00	1	$400.0000	1
B010	2002	Superior	2024-05-01 13:00:00	2024-05-05 13:00:00	0	$130.0000	1

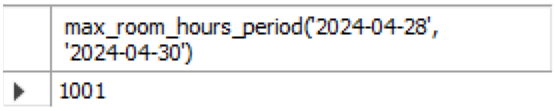
E.3. Functions

1. Get the room with the longest total booking time during the given date period.

```
delimiter $$
create function max_room_hours_period (input_start_date char(10),
                                       input_end_date char(10))
returns char(4)
reads sql data
begin
    declare max_hours_id char(4);
    declare input_start_time datetime;
    declare input_end_time datetime;
    select concat(str_to_date(input_start_date, ’%Y-%m-%d’), ’ 00:00:00’)  into input_start_time;
    select concat(str_to_date(input_end_date, ’%Y-%m-%d’), ’ 23:59:59’) into input_end_time;

    with union_period_books as (
        (select room_id, book_id, check_in_time as s_time, check_out_time as e_time
        from Booking
        where input_start_time <= check_in_time
        and check_out_time <= input_end_time)
        union
        (select room_id, book_id, input_start_time as s_time, check_out_time as e_time
        from Booking
        where check_in_time < input_start_time
        and input_start_time < check_out_time
        and check_out_time <= input_end_time)
        union
        (select room_id, book_id, check_in_time as s_time, input_end_time as e_time
        from Booking
        where input_start_time <= check_in_time
        and check_in_time < input_end_time
        and  input_end_time < check_out_time)
        union
        (select room_id, book_id, input_start_time as s_time, input_end_time as e_time
        from Booking
        where check_in_time < input_start_time
        and  input_end_time < check_out_time)
    ),
    sum_group_hours as (
        select room_id, sum(timestampdiff(hour, s_time, e_time)) as sum_hours
        from union_period_books
        group by room_id
    )
    select room_id into max_hours_id
    from sum_group_hours
    where sum_hours = (select max(sum_hours) from sum_group_hours);
```

```
        return max_hours_id;
end$$
delimiter ;

-- call. get the room with the longest total booking time between 2024-04-28 and 2024-04-30
select max_room_hours_period(’2024-04-28’, ’2024-04-30’);
```

Results:

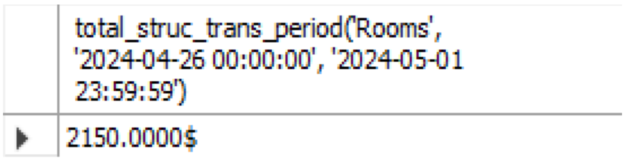


	max_room_hours_period('2024-04-28', '2024-04-30')
▶	1001

2. Get the total transaction amount of a specific structure during a certain period of time.

```
delimiter $$
create function total_struc_trans_period (input_struc_name char(20),
                                          input_start_time char(19),
                                          input_end_time char(19))
returns varchar(20)
reads sql data
begin
    declare total_amount varchar(20);
    declare input_start_time_ datetime;
    declare input_end_time_ datetime;

    set input_start_time_ = str_to_date(input_start_time, ’%Y-%m-%d %H:%i:%s’);

    if input_end_time is NULL or input_end_time = ’’ then
        set input_end_time_ = date_format(now(), ’%Y-%m-%d %H:%i:%s’);
    else
        set input_end_time_ = str_to_date(input_end_time, ’%Y-%m-%d %H:%i:%s’);
    end if;

    select concat(sum(amount), ’$’) into total_amount
    from Transactions
    where struc_name = input_struc_name
    and trans_time >= input_start_time_ and trans_time <= input_end_time_;

    return total_amount;
end$$
delimiter ;

-- call. get the total transaction amount of Rooms structure
--       during the time period from 2024-04-26 00:00:00 to 2024-05-01 23:59:59
select total_struc_trans_period(’Rooms’, ’2024-04-26 00:00:00’, ’2024-05-01 23:59:59’);
```

Results:



	total_struc_trans_period('Rooms', '2024-04-26 00:00:00', '2024-05-01 23:59:59')
▶	2150.0000$

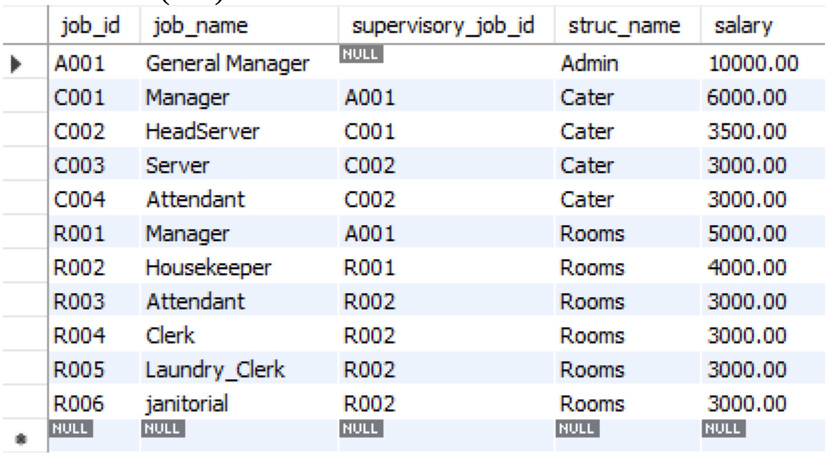
E.4. Procedures

1. Increase the salaries of staffs in Rooms structure at a certain rate.

```
delimiter $$
create procedure raise_Rooms_salary (in rate numeric(2,2))
begin
    if rate > 0 then
        update Job
        set salary = salary * (1 + rate)
        where struc_name = ’Rooms’;
    else
        signal sqlstate ’45000’ set message_text = ’raise: rate should be > 0’;
    end if;
end$$
delimiter ;

-- call. increase the salary at the rate of 0.2
START TRANSACTION;
call raise_Rooms_salary (0.2);
ROLLBACK;
```

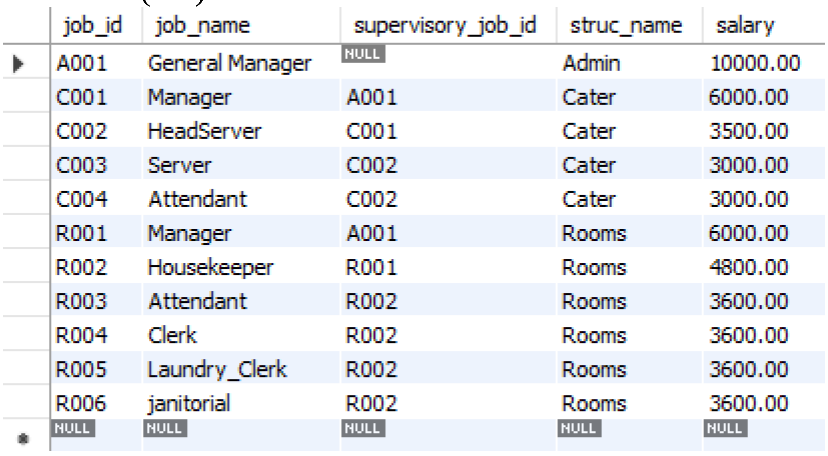
Results:

Before Call (Job)



job_id	job_name	supervisory_job_id	struc_name	salary
A001	General Manager	NULL	Admin	10000.00
C001	Manager	A001	Cater	6000.00
C002	HeadServer	C001	Cater	3500.00
C003	Server	C002	Cater	3000.00
C004	Attendant	C002	Cater	3000.00
R001	Manager	A001	Rooms	5000.00
R002	Housekeeper	R001	Rooms	4000.00
R003	Attendant	R002	Rooms	3000.00
R004	Clerk	R002	Rooms	3000.00
R005	Laundry_Clerk	R002	Rooms	3000.00
R006	janitorial	R002	Rooms	3000.00
NULL	NULL	NULL	NULL	NULL

After Call (Job)



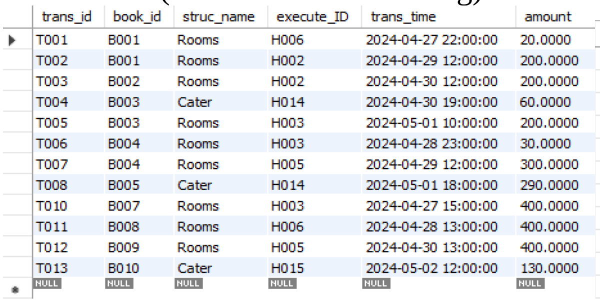
job_id	job_name	supervisory_job_id	struc_name	salary
A001	General Manager	NULL	Admin	10000.00
C001	Manager	A001	Cater	6000.00
C002	HeadServer	C001	Cater	3500.00
C003	Server	C002	Cater	3000.00
C004	Attendant	C002	Cater	3000.00
R001	Manager	A001	Rooms	6000.00
R002	Housekeeper	R001	Rooms	4800.00
R003	Attendant	R002	Rooms	3600.00
R004	Clerk	R002	Rooms	3600.00
R005	Laundry_Clerk	R002	Rooms	3600.00
R006	janitorial	R002	Rooms	3600.00
NULL	NULL	NULL	NULL	NULL

2. The procedure is for customers to pay for their rooms. When a customer pays the room fee, the transaction amount should be recorded as the price of the room booked by the customer, and the Rooms structure is considered to have made the transactions with it. What’s more, the room booking order should be modified to paid.

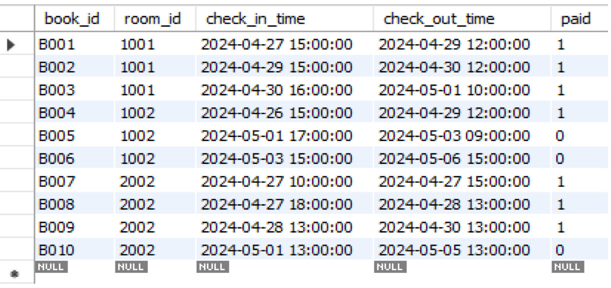
```
delimiter $$
create procedure pay_for_room (in new_trans_id char(4),
                               in new_book_id char(4),
                               in new_execute_id char(4),
                               in new_trans_time datetime)
begin
    if exists (select * from booking
               where book_id = new_book_id and paid = 0) then
        # Transaction add a new row
        insert into Transactions (trans_id, book_id, struc_name, execute_ID,  trans_time, amount)
        select new_trans_id, new_book_id, ’Rooms’, new_execute_ID, new_trans_time, price
        from Room_Type join Room using (type_id) join Booking using (room_id)
        where book_id = new_book_id;

        # set Booking paid = 1
        update Booking
        set paid = 1
        where book_id = new_book_id;
    else
        signal sqlstate ’45000’ set message_text = ’has already been paid’;
    end if;
end$$
delimiter ;

-- call. Add a new transaction record with ID T014,
--       which is for room payment for booking order B010.
--       The transaction is the executed by staff with ID H005.
--       The transaction time is 2024-05-03 12:00:00.
START TRANSACTION;
call pay_for_room (’T014’, ’B010’, ’H005’, ’2024-05-03 12:00:00’);
select * from Transactions;
select * from Booking;
ROLLBACK;
```

Results:

Before Call (Transaction & Booking)

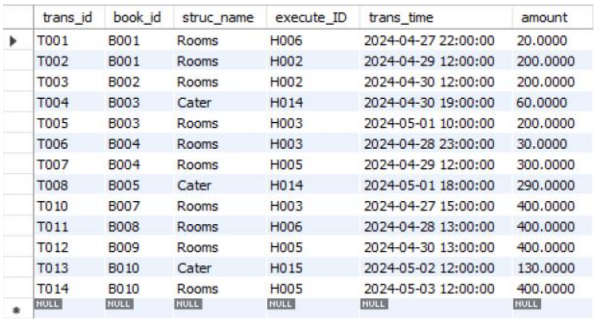


trans_id	book_id	struc_name	execute_ID	trans_time	amount
T001	B001	Rooms	H006	2024-04-27 22:00:00	20.0000
T002	B001	Rooms	H002	2024-04-29 12:00:00	200.0000
T003	B002	Rooms	H002	2024-04-30 12:00:00	200.0000
T004	B003	Cater	H014	2024-04-30 19:00:00	60.0000
T005	B003	Rooms	H003	2024-05-01 10:00:00	200.0000
T006	B004	Rooms	H003	2024-04-28 23:00:00	30.0000
T007	B004	Rooms	H005	2024-04-29 12:00:00	300.0000
T008	B005	Cater	H014	2024-05-01 18:00:00	290.0000
T010	B007	Rooms	H003	2024-04-27 15:00:00	400.0000
T011	B008	Rooms	H006	2024-04-28 13:00:00	400.0000
T012	B009	Rooms	H005	2024-04-30 13:00:00	400.0000
T013	B010	Cater	H015	2024-05-02 12:00:00	130.0000
NULL	NULL	NULL	NULL	NULL	NULL

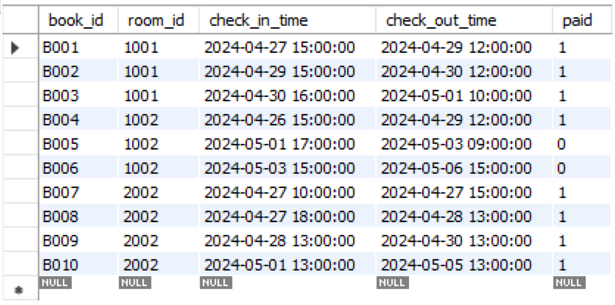


book_id	room_id	check_in_time	check_out_time	paid
B001	1001	2024-04-27 15:00:00	2024-04-29 12:00:00	1
B002	1001	2024-04-29 15:00:00	2024-04-30 12:00:00	1
B003	1001	2024-04-30 16:00:00	2024-05-01 10:00:00	1
B004	1002	2024-04-26 15:00:00	2024-04-29 12:00:00	1
B005	1002	2024-05-01 17:00:00	2024-05-03 09:00:00	0
B006	1002	2024-05-03 15:00:00	2024-05-06 15:00:00	0
B007	2002	2024-04-27 10:00:00	2024-04-27 15:00:00	1
B008	2002	2024-04-27 18:00:00	2024-04-28 13:00:00	1
B009	2002	2024-04-28 13:00:00	2024-04-30 13:00:00	1
B010	2002	2024-05-01 13:00:00	2024-05-05 13:00:00	0
NULL	NULL	NULL	NULL	NULL

After Call (Transaction & Booking)



trans_id	book_id	struc_name	execute_ID	trans_time	amount
T001	B001	Rooms	H006	2024-04-27 22:00:00	20.0000
T002	B001	Rooms	H002	2024-04-29 12:00:00	200.0000
T003	B002	Rooms	H002	2024-04-30 12:00:00	200.0000
T004	B003	Cater	H014	2024-04-30 19:00:00	60.0000
T005	B003	Rooms	H003	2024-05-01 10:00:00	200.0000
T006	B004	Rooms	H003	2024-04-28 23:00:00	30.0000
T007	B004	Rooms	H005	2024-04-29 12:00:00	300.0000
T008	B005	Cater	H014	2024-05-01 18:00:00	290.0000
T010	B007	Rooms	H003	2024-04-27 15:00:00	400.0000
T011	B008	Rooms	H006	2024-04-28 13:00:00	400.0000
T012	B009	Rooms	H005	2024-04-30 13:00:00	400.0000
T013	B010	Cater	H015	2024-05-02 12:00:00	130.0000
T014	B010	Rooms	H005	2024-05-03 12:00:00	400.0000
NULL	NULL	NULL	NULL	NULL	NULL



book_id	room_id	check_in_time	check_out_time	paid
B001	1001	2024-04-27 15:00:00	2024-04-29 12:00:00	1
B002	1001	2024-04-29 15:00:00	2024-04-30 12:00:00	1
B003	1001	2024-04-30 16:00:00	2024-05-01 10:00:00	1
B004	1002	2024-04-26 15:00:00	2024-04-29 12:00:00	1
B005	1002	2024-05-01 17:00:00	2024-05-03 09:00:00	0
B006	1002	2024-05-03 15:00:00	2024-05-06 15:00:00	0
B007	2002	2024-04-27 10:00:00	2024-04-27 15:00:00	1
B008	2002	2024-04-27 18:00:00	2024-04-28 13:00:00	1
B009	2002	2024-04-28 13:00:00	2024-04-30 13:00:00	1
B010	2002	2024-05-01 13:00:00	2024-05-05 13:00:00	1
NULL	NULL	NULL	NULL	NULL

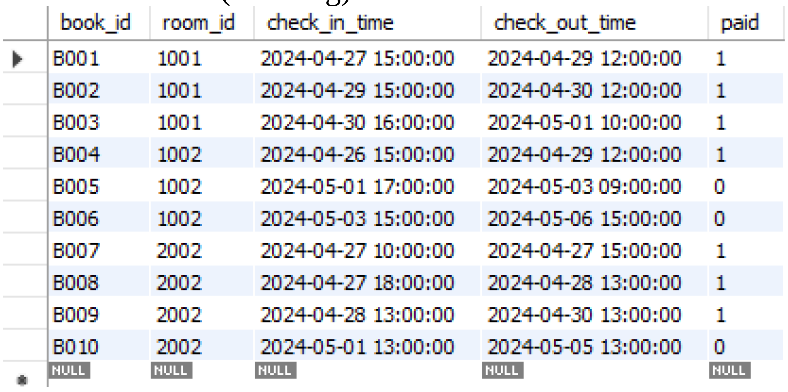
E.5. Triggers

1. To avoid people booking a time slot that conflicts or overlaps with those already booked by other people.

```
delimiter $$
create trigger New_Booking
before insert on booking
for each row
begin
if exists(
select *
from booking
where new.room_id = room_id
and not(
new.check_in_time > check_out_time
or new.check_out_time < check_in_time)
)
then signal sqlstate ’45000’ set message_text = ’Time Conflict’;
end if;
end$$
delimiter ;

-- Insert a record with a time slot that conflicts with others.
START TRANSACTION;
insert into Booking (book_id, room_id, check_in_time, check_out_time,
paid)
values (’B011’, ’1001’, ’2024-04-30 15:00:00’, ’2024-04-30 17:00:00’, 0);
ROLLBACK;
```

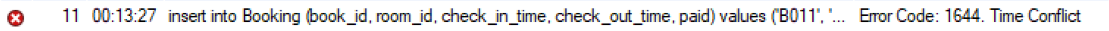
Result:

Before Insertion (Booking)

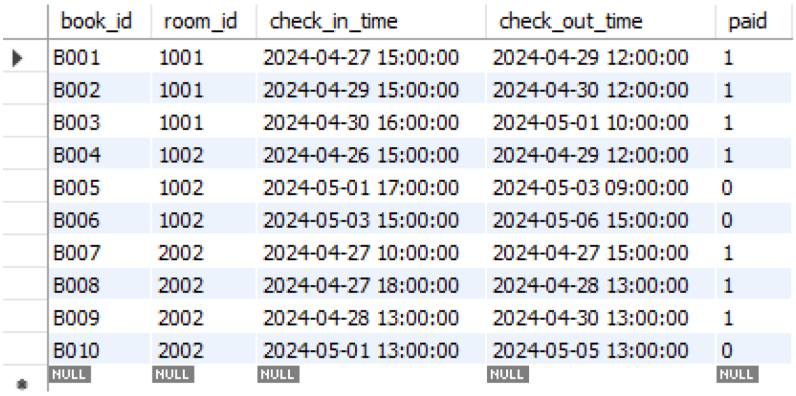


book_id	room_id	check_in_time	check_out_time	paid
B001	1001	2024-04-27 15:00:00	2024-04-29 12:00:00	1
B002	1001	2024-04-29 15:00:00	2024-04-30 12:00:00	1
B003	1001	2024-04-30 16:00:00	2024-05-01 10:00:00	1
B004	1002	2024-04-26 15:00:00	2024-04-29 12:00:00	1
B005	1002	2024-05-01 17:00:00	2024-05-03 09:00:00	0
B006	1002	2024-05-03 15:00:00	2024-05-06 15:00:00	0
B007	2002	2024-04-27 10:00:00	2024-04-27 15:00:00	1
B008	2002	2024-04-27 18:00:00	2024-04-28 13:00:00	1
B009	2002	2024-04-28 13:00:00	2024-04-30 13:00:00	1
B010	2002	2024-05-01 13:00:00	2024-05-05 13:00:00	0
NULL	NULL	NULL	NULL	NULL

After Insertion (Booking)



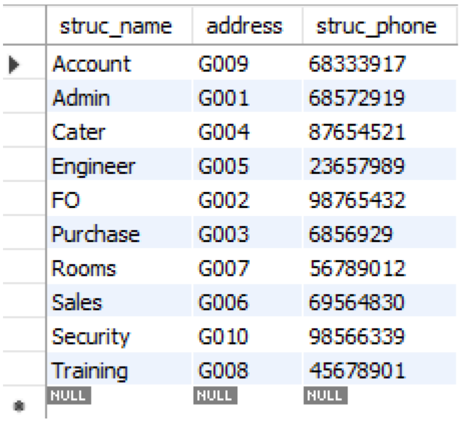
```
11  00:13:27  insert into Booking (book_id, room_id, check_in_time, check_out_time, paid) values ('B011', '...   Error Code: 1644. Time Conflict
```



book_id	room_id	check_in_time	check_out_time	paid
B001	1001	2024-04-27 15:00:00	2024-04-29 12:00:00	1
B002	1001	2024-04-29 15:00:00	2024-04-30 12:00:00	1
B003	1001	2024-04-30 16:00:00	2024-05-01 10:00:00	1
B004	1002	2024-04-26 15:00:00	2024-04-29 12:00:00	1
B005	1002	2024-05-01 17:00:00	2024-05-03 09:00:00	0
B006	1002	2024-05-03 15:00:00	2024-05-06 15:00:00	0
B007	2002	2024-04-27 10:00:00	2024-04-27 15:00:00	1
B008	2002	2024-04-27 18:00:00	2024-04-28 13:00:00	1
B009	2002	2024-04-28 13:00:00	2024-04-30 13:00:00	1
B010	2002	2024-05-01 13:00:00	2024-05-05 13:00:00	0
NULL	NULL	NULL	NULL	NULL

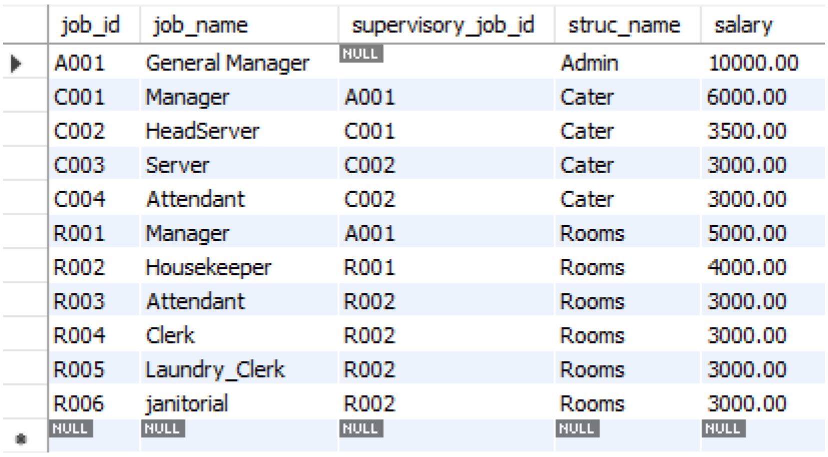
F. Insert Records

1. Structure



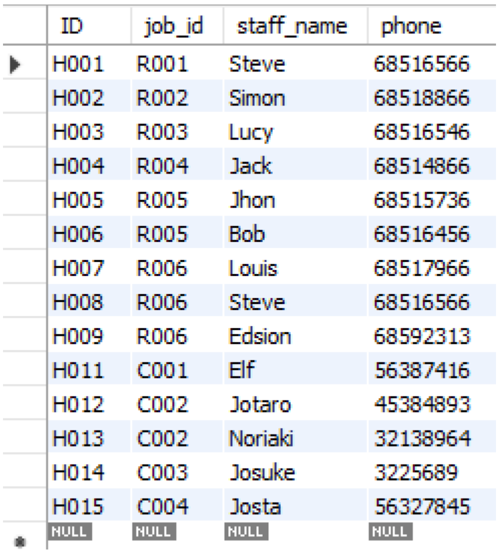
struc_name	address	struc_phone
Account	G009	68333917
Admin	G001	68572919
Cater	G004	87654521
Engineer	G005	23657989
FO	G002	98765432
Purchase	G003	6856929
Rooms	G007	56789012
Sales	G006	69564830
Security	G010	98566339
Training	G008	45678901
NULL	NULL	NULL

2. Job



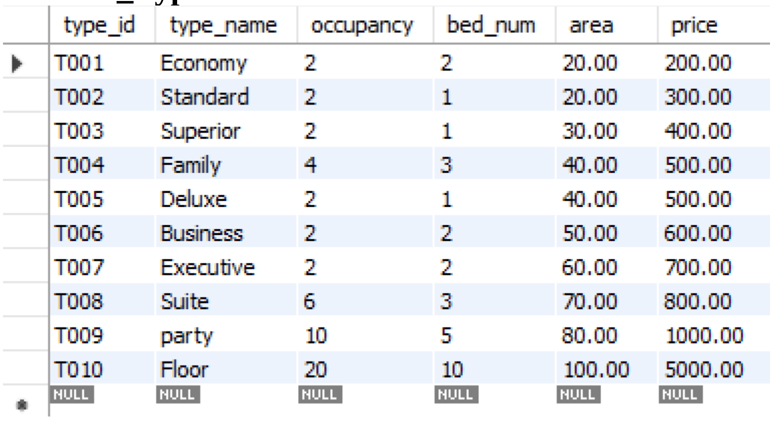
job_id	job_name	supervisory_job_id	struc_name	salary
A001	General Manager	NULL	Admin	10000.00
C001	Manager	A001	Cater	6000.00
C002	HeadServer	C001	Cater	3500.00
C003	Server	C002	Cater	3000.00
C004	Attendant	C002	Cater	3000.00
R001	Manager	A001	Rooms	5000.00
R002	Housekeeper	R001	Rooms	4000.00
R003	Attendant	R002	Rooms	3000.00
R004	Clerk	R002	Rooms	3000.00
R005	Laundry_Clerk	R002	Rooms	3000.00
R006	janitorial	R002	Rooms	3000.00
NULL	NULL	NULL	NULL	NULL

3. Staff



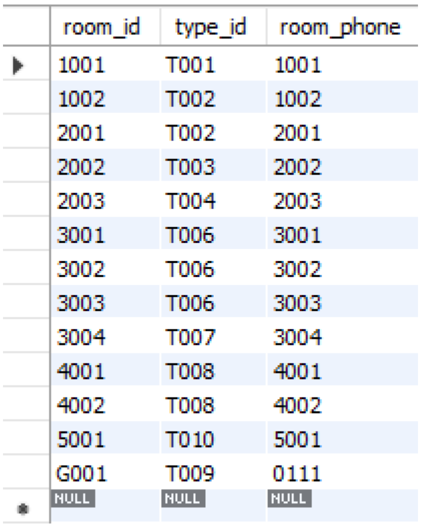
ID	job_id	staff_name	phone
H001	R001	Steve	68516566
H002	R002	Simon	68518866
H003	R003	Lucy	68516546
H004	R004	Jack	68514866
H005	R005	Jhon	68515736
H006	R005	Bob	68516456
H007	R006	Louis	68517966
H008	R006	Steve	68516566
H009	R006	Edsion	68592313
H011	C001	Elf	56387416
H012	C002	Jotaro	45384893
H013	C002	Noriaki	32138964
H014	C003	Josuke	3225689
H015	C004	Josta	56327845
NULL	NULL	NULL	NULL

4. Room_Type



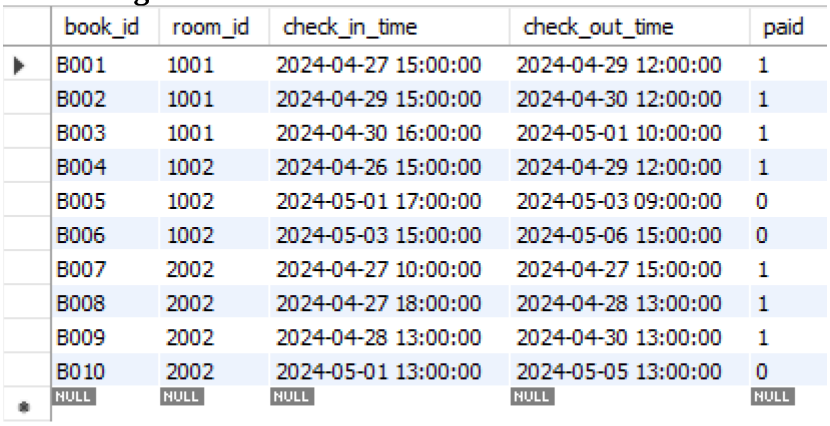
type_id	type_name	occupancy	bed_num	area	price
T001	Economy	2	2	20.00	200.00
T002	Standard	2	1	20.00	300.00
T003	Superior	2	1	30.00	400.00
T004	Family	4	3	40.00	500.00
T005	Deluxe	2	1	40.00	500.00
T006	Business	2	2	50.00	600.00
T007	Executive	2	2	60.00	700.00
T008	Suite	6	3	70.00	800.00
T009	party	10	5	80.00	1000.00
T010	Floor	20	10	100.00	5000.00
NULL	NULL	NULL	NULL	NULL	NULL

5. Room



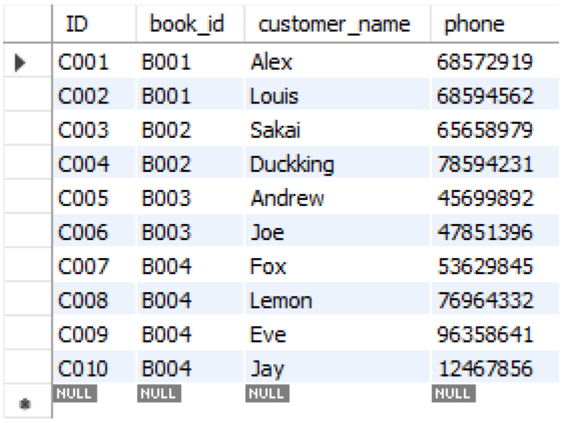
room_id	type_id	room_phone
1001	T001	1001
1002	T002	1002
2001	T002	2001
2002	T003	2002
2003	T004	2003
3001	T006	3001
3002	T006	3002
3003	T006	3003
3004	T007	3004
4001	T008	4001
4002	T008	4002
5001	T010	5001
G001	T009	0111
NULL	NULL	NULL

6. Booking



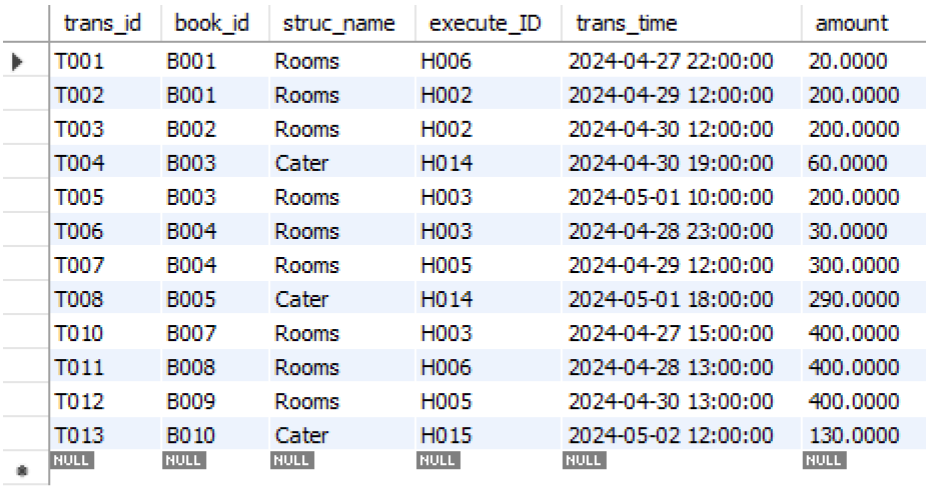
book_id	room_id	check_in_time	check_out_time	paid
B001	1001	2024-04-27 15:00:00	2024-04-29 12:00:00	1
B002	1001	2024-04-29 15:00:00	2024-04-30 12:00:00	1
B003	1001	2024-04-30 16:00:00	2024-05-01 10:00:00	1
B004	1002	2024-04-26 15:00:00	2024-04-29 12:00:00	1
B005	1002	2024-05-01 17:00:00	2024-05-03 09:00:00	0
B006	1002	2024-05-03 15:00:00	2024-05-06 15:00:00	0
B007	2002	2024-04-27 10:00:00	2024-04-27 15:00:00	1
B008	2002	2024-04-27 18:00:00	2024-04-28 13:00:00	1
B009	2002	2024-04-28 13:00:00	2024-04-30 13:00:00	1
B010	2002	2024-05-01 13:00:00	2024-05-05 13:00:00	0
NULL	NULL	NULL	NULL	NULL

7. Customer



ID	book_id	customer_name	phone
C001	B001	Alex	68572919
C002	B001	Louis	68594562
C003	B002	Sakai	65658979
C004	B002	Duckking	78594231
C005	B003	Andrew	45699892
C006	B003	Joe	47851396
C007	B004	Fox	53629845
C008	B004	Lemon	76964332
C009	B004	Eve	96358641
C010	B004	Jay	12467856
NULL	NULL	NULL	NULL

8. Transactions



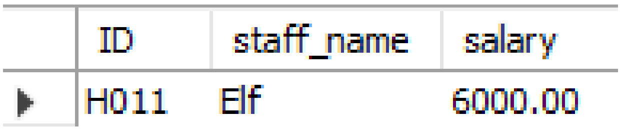
	trans_id	book_id	struc_name	execute_ID	trans_time	amount
▶	T001	B001	Rooms	H006	2024-04-27 22:00:00	20.0000
	T002	B001	Rooms	H002	2024-04-29 12:00:00	200.0000
	T003	B002	Rooms	H002	2024-04-30 12:00:00	200.0000
	T004	B003	Cater	H014	2024-04-30 19:00:00	60.0000
	T005	B003	Rooms	H003	2024-05-01 10:00:00	200.0000
	T006	B004	Rooms	H003	2024-04-28 23:00:00	30.0000
	T007	B004	Rooms	H005	2024-04-29 12:00:00	300.0000
	T008	B005	Cater	H014	2024-05-01 18:00:00	290.0000
	T010	B007	Rooms	H003	2024-04-27 15:00:00	400.0000
	T011	B008	Rooms	H006	2024-04-28 13:00:00	400.0000
	T012	B009	Rooms	H005	2024-04-30 13:00:00	400.0000
	T013	B010	Cater	H015	2024-05-02 12:00:00	130.0000
*	NULL	NULL	NULL	NULL	NULL	NULL

G. Reasonable Queries

1. (join, max)
Find the highest paid staff.

```
with Staff_Job as(
    select *
    from Staff join Job using (job_id))
select ID, staff_name
from Staff_job
where salary = (select MAX(salary) from Staff_Job);
```

Result:

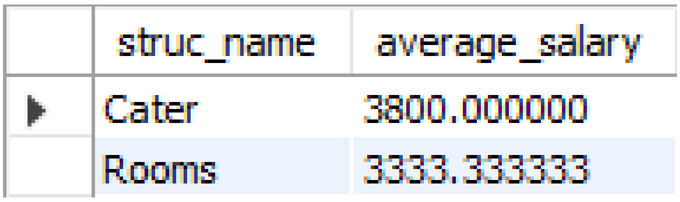


	ID	staff_name	salary
▶	H011	Elf	6000.00

2. (join, avg, group by)
Calculate the average salary for each structure

```
with Staff_Job as(
    select *
    from Staff join Job using (job_id))
select struc_name, avg(salary) as average_salary
from Staff_Job
group by struc_name;
```

Result:



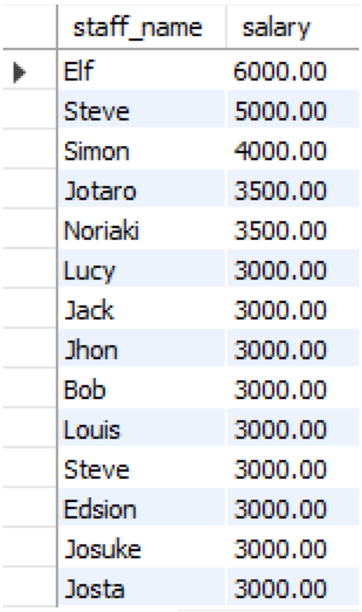
struc_name	average_salary
Cater	3800.000000
Rooms	3333.333333

3. (join, order by)

Display the salaries of all staffs and sort them in descending order.

```
select s.staff_name, j.salary
from Staff s
join Job j on s.job_id = j.job_id
order by j.salary desc;
```

Result:



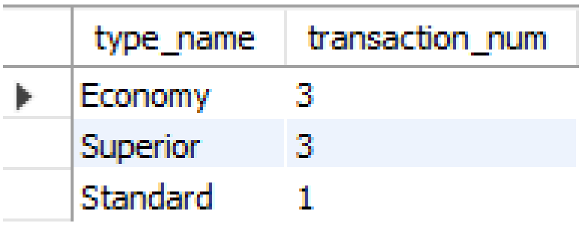
staff_name	salary
Elf	6000.00
Steve	5000.00
Simon	4000.00
Jotaro	3500.00
Noriaki	3500.00
Lucy	3000.00
Jack	3000.00
Jhon	3000.00
Bob	3000.00
Louis	3000.00
Steve	3000.00
Edsion	3000.00
Josuke	3000.00
Josta	3000.00

4. (count, join, group by, order by)

Find the rooms with the top 3 highest transaction count in the paid room booking record.

```
select t.type_name, count(*) as transaction_num
from Room_Type t
join Room r on r.type_id = t.type_id
join Booking b on b.room_id = r.room_id
where b.paid = 1
group by t.type_name
order by transaction_num desc
limit 3;
```

Result:



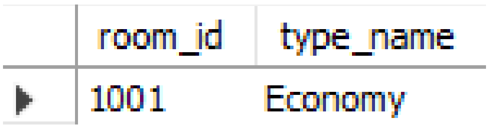
type_name	transaction_num
Economy	3
Superior	3
Standard	1

5. (join)

Query the room IDs of ‘Economy’ type.

```
select r.room_id, t.type_name
from Room r
join Room_Type t on t.type_id = r.type_id
where t.type_name = ’Economy’;
```

Result:



room_id	type_name
1001	Economy

6. (ifnull, sum, left outer join, group by, correlated subquery)

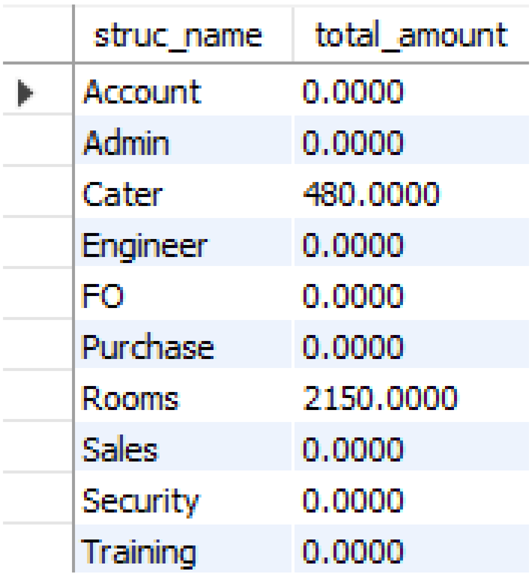
Query the total transaction amount of different structures.

```
select struc_name, ifnull(sum(amount), 0) as total_amount
from Structure left outer join Transactions using (struc_name)
group by struc_name;
```

or

```
select struc_name, (select ifnull(sum(amount), 0)
                     from Transactions T
                     where T.struc_name = S.struc_name) as total_amount
from Structure S;
```

Result:

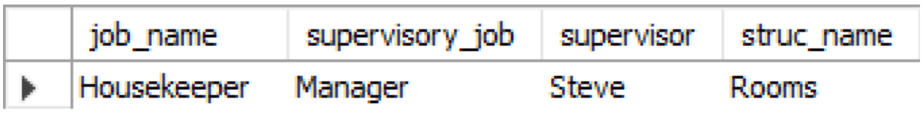


	struc_name	total_amount
▶	Account	0.0000
	Admin	0.0000
	Cater	480.0000
	Engineer	0.0000
	FO	0.0000
	Purchase	0.0000
	Rooms	2150.0000
	Sales	0.0000
	Security	0.0000
	Training	0.0000

7. (left outer join, join)
Find staff Simon’s job and his/her supervisor and supervisor’s job.

```
select J.job_name, S.job_name as supervisory_job, stf_s.staff_name as
supervisor, J.struc_name
from (Job J left outer join Job S on J.supervisory_job_id = S.job_id)
join Staff stf_j on stf_j.job_id = J.job_id
join Staff stf_s on stf_s.job_id = S.job_id
where stf_j.staff_name = ’Simon’;
```

Result:

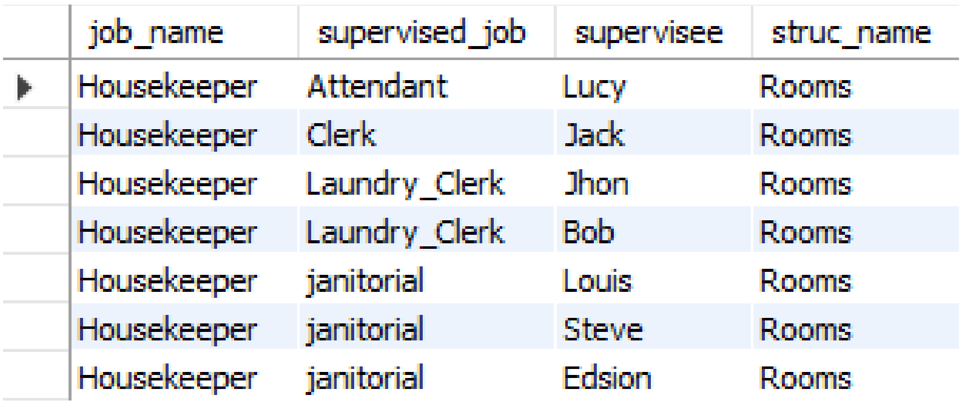


	job_name	supervisory_job	supervisor	struc_name
▶	Housekeeper	Manager	Steve	Rooms

8. (left outer join, join)
Find staff Simon’s job and his/her supervisees and the jobs of supervisees.

```
select J.job_name, Sed.job_name as supervised_job, stf_sed.staff_name as
supervisee, J.struc_name
from (Job J left outer join Job Sed on J.job_id = Sed.supervisory_job_id)
join Staff stf_j on stf_j.job_id = J.job_id
join Staff stf_sed on stf_sed.job_id = Sed.job_id
where stf_j.staff_name = ’Simon’;
```

Result:



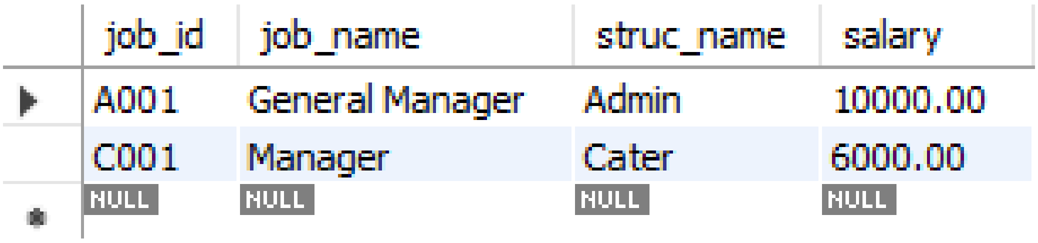
job_name	supervised_job	supervisee	struc_name
Housekeeper	Attendant	Lucy	Rooms
Housekeeper	Clerk	Jack	Rooms
Housekeeper	Laundry_Clerk	Jhon	Rooms
Housekeeper	Laundry_Clerk	Bob	Rooms
Housekeeper	janitorial	Louis	Rooms
Housekeeper	janitorial	Steve	Rooms
Housekeeper	janitorial	Edsion	Rooms

9. (all)

Find the job ID, job name, structure and salary, where the salary for this job is greater then the salary for all jobs in Rooms structure.

```
select job_id, job_name, struc_name, salary
from Job
where salary > all (select salary
                    from Job
                    where struc_name = ’Rooms’);
```

Result:



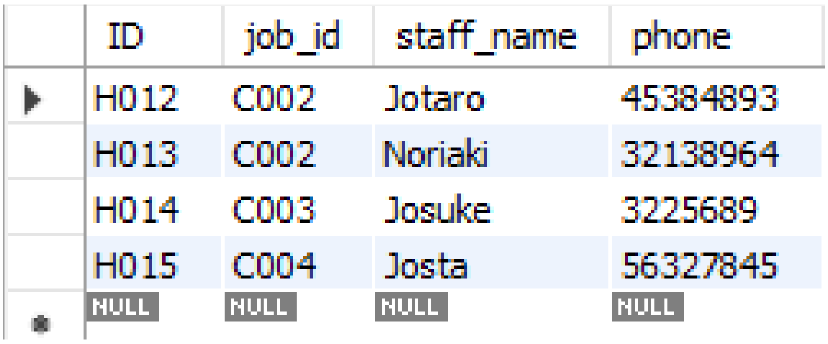
job_id	job_name	struc_name	salary
A001	General Manager	Admin	10000.00
C001	Manager	Cater	6000.00
NULL	NULL	NULL	NULL

10. (like)

Find the staffs whose names start with ‘Jo’, or have a length of 7 and end with ‘ki’.

```
select *
from Staff
where staff_name like ’Jo%’
or staff_name like ’_____ki’;
```

Result:

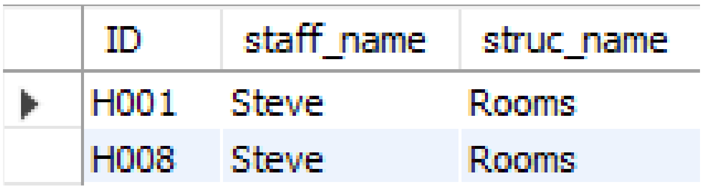


ID	job_id	staff_name	phone
H012	C002	Jotaro	45384893
H013	C002	Noriaki	32138964
H014	C003	Josuke	3225689
H015	C004	Josta	56327845
NULL	NULL	NULL	NULL

11. (join, exists, correlated subquery)
Find the staffs with same name in the same structure.

```
select ID, staff_name, struc_name
from Staff S1 join Job J1 using (job_id)
where exists (select *
              from Staff S2 join Job J2 using (job_id)
              where J2.struc_name = J1.struc_name
              and S2.staff_name = S1.staff_name
              and S2.ID != S1.ID);
```

Result:

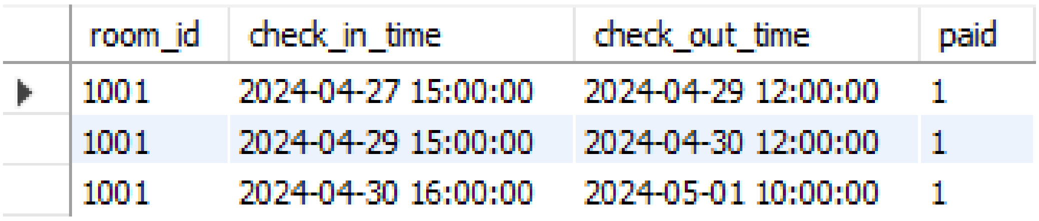


ID	staff_name	struc_name
H001	Steve	Rooms
H008	Steve	Rooms

12. (in, join, order by)
Find all the time period that have been booked for the rooms satisfying the certain conditions (occupancy and bed number) within a specific date range.

```
set @new_occupancy = 2;
set @new_bed_num = 2;
set @new_check_in_date = ’2024-04-29’;
set @new_check_out_date = ’2024-05-01’;

select room_id, check_in_time, check_out_time, paid
from Booking
where room_id in (
    select room_id
    from Room join Room_Type using (type_id)
    where occupancy = @new_occupancy and bed_num = @new_bed_num
)
and not(
    date(check_out_time) < @new_check_in_date
    or
    date(check_in_time) > @new_check_out_date
)
order by room_id, check_in_time;
```

Result:

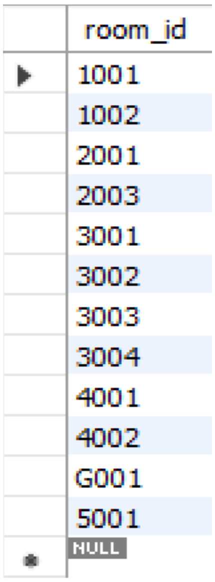


room_id	check_in_time	check_out_time	paid
1001	2024-04-27 15:00:00	2024-04-29 12:00:00	1
1001	2024-04-29 15:00:00	2024-04-30 12:00:00	1
1001	2024-04-30 16:00:00	2024-05-01 10:00:00	1

13. (not exists)
Find all the rooms that have not been booked within a given period of time.

```
set @new_check_in_time = ’2024-04-29 13:00:00’;
set @new_check_out_time = ’2024-04-29 14:00:00’;

select room_id
from Room R
where not exists(
select *
from Booking
where Booking.room_id = R.room_id
and not(
@new_check_in_time > check_out_time
or @new_check_out_time < check_in_time));
```

Result:



room_id
1001
1002
2001
2003
3001
3002
3003
3004
4001
4002
G001
5001
NULL