Database Project

Yves Semana Gisubizo

Domain Description

Title: Hotel Database

The Hotel Database is designed to store and manage information related to a typical hotel. This database will include information about the hotel's employees, transactions, customers, and other relevant operations.

This database will help in authenticating, storing, and managing a specified hotel business and

Entities and Relationships

The Hotel Database will include several key entities, each of which will be related to one another in various ways. These entities include:

Employees: This entity will hold all the information regarding the hotel's employees such as their unique id, name, phone, address, gender, contract start and end date, and more. The employee id and username should be unique for the identification and authentication purposes of each individual employee.

Orders: This entity will contain all the information regarding the orders that take place in the hotel such as the order id, the product being ordered, amount, order duration, the ordering customer info, the employee in charge, and more. It will store all the orders in chronological order with the customer and payments included.

Inventory: This entity will hold information about the hotel's inventory, such as the availability and stock levels of food and beverage items, toiletries, linens, and other amenities.

Suppliers: This entity will contain information about the hotel's suppliers for the different products or services such as the supplier id, name, phone, address, product and more.

Rooms: This entity will contain information about the room services of the hotel, including the room number, price, location, number of people, amenities, availability status, and more. Each room will have a unique room number to ensure no double-booking issues of the rooms.

Reservations: This entity will act as the orders entity but for the rooms. It will contain information about the bookings of the rooms such as reservation id, room information, customer information, sign-in and sign-out dates, and more. Every reservation will be assigned a unique reservation id. The room can only be reserved if it's availability condition is true.

Payments: This entity will hold details about the customer's payment method, amount, payment history, whether paid or debt and outstanding balances. This entity will be referenced with the unique order numbers to make sure we can identify where every amount has originated.

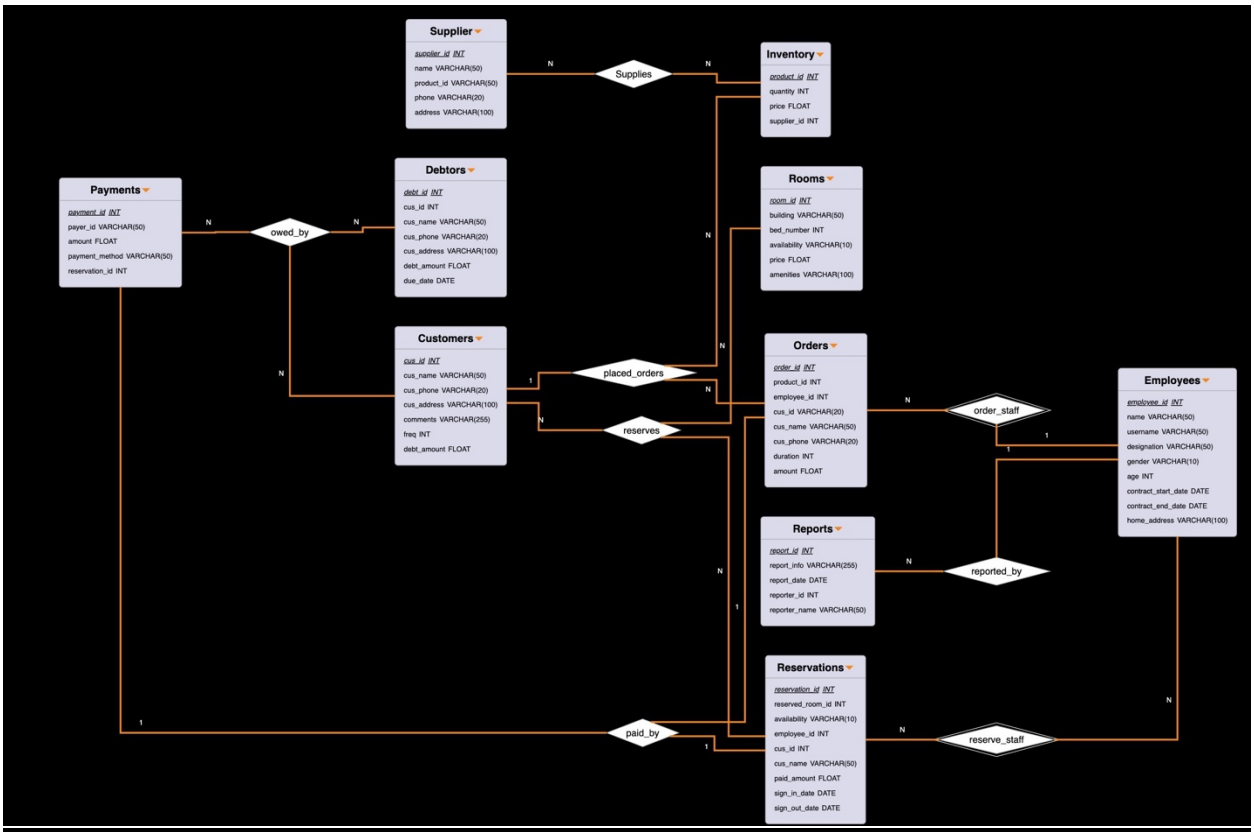
Debtors: This entity will hold the information regarding customers who have acquired hotel services as a debt such as the debt amount, due date, customer information and more. Every debt will be assigned a unique debt id for identifying the debts easily.

Customers: This entity will contain information about the customers for this hotel such as the customer names, phone numbers, addresses, money owed to the hotel (if any) and more. This relationship will allow the database to store and track who are the hotel's customers and link them to their order information. This data will also be used to analyze what are the hotel's most frequent customers, orders, etc.

Reports: This entity will contain information about different reports generated automatically or manually for different causes, including report id, report information, and reporting employee's information such as the employee id and name. It will use a unique report id to be able to identify every report individually.

Entity-Relationship Diagram

TRINITY COLLEGE · DEPARTMENT OF COMPUTER SCIENCE
CPSC 372 Database Fundamentals, Spring 2023
Database Project, Yves Semana Gisubizo



Relational Schema

Employees (<u>employee_id</u>, name, username, designation, gender, age, contract_start_date, contract_end_date,home_address)
Orders (<u>order_id</u>, product_id, employee_id, cus_id, cus_name, cus_phone, duration, amount)
Inventory (<u>product_id</u>, quantity, price, supplier_id)
Supplier (<u>supplier_id</u>, name, product_id, phone, address)
Rooms (<u>room_id</u>, building, bed_number, availability, price, amenities)
Reservations (<u>reservation_id</u>, reserved_room_id, availability, cus_id, cus_name, paid_amount, sign_in_date, sign_out_date)
Payments (<u>payment_id</u>, payer, amount, payment_method)
Debtors (<u>debt_id</u>, cus_id, cus_name, cus_phone, cus_address, debt_amount, due_date)
Customers (<u>cus_id</u>, cus_name, cus_phone, cus_address, comments, freq, debt_amount)
Reports (<u>report_id</u>, report_info, report_date, reporter_id, reporter_name)

Boyce–Codd Normal Form Decomposition

Assuming there are no transitive dependencies, here are the functional dependencies from my database with their candidate keys:

1. **Employees** (employee_id, name, username, gender, age, contract_start_date, contract_end_date, home_address)

   Non-trivial functional dependencies:
   employee_id -> name, username, gender, age, contract_start_date, contract_end_date, home_address
   username -> employee_id, name, gender, age, contract_start_date, contract_end_date, home_address

   Result: Yes, it is in BCNF. All the attributes on the right side of the dependency are fully dependent on the primary key (employee_id), and username is also a superkey (employees cannot share a username)

2. **Orders** (order_id, product_id, employee_id, cus_id, cus_name, cus_phone, duration, amount)

   Non-trivial functional dependencies:
   order_id -> product_id, employee_id, cus_id, cus_name, cus_phone, duration, amount

   Result: Yes, it is in BCNF because order_id is a super key.

3. **Inventory** (product_id, quantity, price, supplier_id)

   Non-trivial functional dependencies:
   product_id -> quantity, price, supplier_id

   Result: Yes, it is in BCNF because product_id is a super key.

4. **Supplier** (supplier_id, supplier_name, supplier_phone, supplier_address)

   Non-trivial functional dependencies:
   supplier_id -> supplier_name, supplier_phone, supplier_address

   Result: Yes, it is in BCNF because supplier_id is a super key.

5. **Rooms** (room_id, building, bed_number, availability, price, amenities)

   Non-trivial functional dependencies:
   room_id -> building, bed_number, availability, price, amenities

   Result: Yes, it is in BCNF because room_id is a super key.

6. **Reservations** (reservation_id, reserved_room_id, availability, cus_id, cus_name, paid_amount, sign_in_date, sign_out_date)

   Non-trivial functional dependencies:
   reservation_id -> reserved_room_id, availability, cus_id, cus_name, paid_amount, sign_in_date, sign_out_date

   Result: Yes, it is in BCNF because reservation_id is a super key.

7. **Payments** (payment_id, payer, amount, payment_method)

   Non-trivial functional dependencies:
   payment_id -> payer, amount, payment_method

   Result: Yes, it is in BCNF because payment_id is a super key.

8. **Debtors** (debt_id, cus_id, cus_name, cus_phone, cus_address, debt_amount, due_date)

   Non-trivial functional dependencies:
   debt_id -> cus_id, cus_name, cus_phone, cus_address, debt_amount, due_date
   cus_id -> cus_name, cus_phone, cus_address, debt_amount, due_date

   Result: Yes, it is BCNF. All the attributes on the right side of the dependencies are fully dependent on the primary keys (debt_id and cus_id)

9. **Customers** (cus_id, cus_name, cus_phone, cus_address, comments, freq, debt_amount)

   Non-trivial functional dependencies:
   cus_id -> cus_name, cus_phone, cus_address, comments, freq, debt_amount

   Result: Yes, it is in BCNF because cus_id is a super key.

10. **Reports** (report_id, report_info, report_date, reporter_id, reporter_name)

    Non-trivial functional dependencies:
    report_id -> report_info, report_date, reporter_id, reporter_name

    Result: Yes, it is in BCNF because report_id is a super key.

In all the tables, the non-trivial functional dependencies hold because the tables are designed in such a way that each attribute is fully dependent on the primary key of the table, and there are no non-trivial dependencies between non-key attributes.

TRINITY COLLEGE · DEPARTMENT OF COMPUTER SCIENCE
CPSC 372 Database Fundamentals, Spring 2023
Database Project, Yves Semana Gisubizo

Transaction and Query Executions

(All the queries below can also be found in a single SQL file named *sample-quieries.sql*)

//At least two queries should involve four or more relations

QUERY 1

Query description:
Retrieve the details of all reservations along with the corresponding customer information, reserved room details, and payment information.

Query:

```
SELECT *
  FROM Reservations r
       JOIN
       Customers c ON r.cus_id = c.cus_id
       JOIN
       Rooms rm ON r.reserved_room_id = rm.room_id
       JOIN
       Payments p ON r.reservation_id = p.reservation_id;
```

Intended result set:
The intended result is a list of all reservations along with the corresponding customer name, phone number, address, reserved room building, bed number, price, and amenities, and payment details such as the payer, amount paid, and payment method.

	reservati	reserved	availabil	employe	cus_id	cus_name	paid_am	sign_in_date	sign_out_dat	cus_id:1	cus_name:1	cus_phone	cus_address	comments	freq	debt_am	room_id	building	bed_nun	availabil	price	amenities	payment	payer_ic	amount	payment_m	reservati
1	1	101	TRUE	2	1	Charlie Gray	500	2023-05-01	2023-05-03	1	Charlie Gray	345-678-9012	56 Mbaga St,Nairobi,Kenya	VIP	3	500	101	A	1	TRUE	50	TV,Wifi	1	3	100	Debt Card	1
2	2	415	TRUE	1	3	Frank Chen	750	2023-05-02	2023-05-05	3	Frank Chen	333-444-5555	64 Manick St,Texas,USA	NULL	1	1000	415	B	2	TRUE	75	TV,Wifi,Mini fridge	2	2	50	Cash	2
3	3	210	TRUE	2	2	Ellen Kim	1000	2023-05-04	2023-05-08	2	Ellen Kim	222-333-4444	453 Kimina St,Kigali,Rwanda	NULL	2	0	210	C	2	FALSE	100	TV,Wifi,Mini fridge,Microwave	3	1	25	Debt	3
4	4	104	TRUE	3	7	Enock Niyonkuru	1250	2023-05-07	2023-05-10	7	Enock Niyonkuru	222-987-0001	33 Mana St,Kigali,Rwanda	COMFY BEDS	1	0	104	B	2	TRUE	75	TV,Wifi,Mini fridge	4	4	75	PayPal	4
5	5	101	TRUE	3	6	Semg Dile	1500	2023-05-09	2023-05-12	6	Semg Dile	234-123-4567	13 Cheka St,Dodoma,Tanzania	NULL	1	0	101	A	1	TRUE	50	TV,Wifi	5	5	200	Credit Card	5

QUERY 2

Query description:
Retrieve the list of all orders along with the product details, supplier information, and employee details involved in each order.

Query:

```
SELECT o.order_id,
       o.product_id,
       i.quantity,
       i.price,
       s.supplier_id,
       s.name AS supplier_name,
       s.product_id AS supplier_product,
       e.employee_id,
       e.name AS employee_name,
       e.designation,
       e.gender,
       e.age
  FROM orders o
       JOIN
       Inventory i ON o.product_id = i.product_id
       JOIN
       Supplier s ON i.supplier_id = s.supplier_id
       JOIN
       Employees e ON o.employee_id = e.employee_id;
```

Intended result set:
The intended result is a list of all orders along with the corresponding product quantity and price, supplier ID, name, and product, and employee ID, name, designation, gender, and age.

	order_id	product_	quantity	price	supplier_	supplier_name	supplier_product	employe	employee_r	designation	gender	age
1	1	1	400	20	1	Inkoko Inc.	1	1	John Doe	Manager	Male	40
2	2	2	55	10	2	Tura Tugabane Corp.	2	5	David Kim	Stock Manager	Male	29
3	3	5	9	15	4	Gisubizo Industries	5	2	Jane Smith	Assistant Manager	Female	35

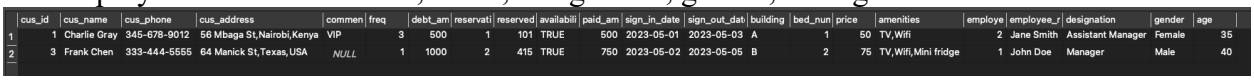
QUERY 3

Query description:
Retrieve the details of all customers who have a debt amount greater than or equal to $100, along with their corresponding reservations, reserved room details, and employee details.

Query:

```
SELECT c.cus_id,
       c.cus_name,
       c.cus_phone,
       c.cus_address,
       c.comments,
       c.freq,
       c.debt_amount,
       r.reservation_id,
       r.reserved_room_id,
       r.availability,
       r.paid_amount,
       r.sign_in_date,
       r.sign_out_date,
       rm.building,
       rm.bed_number,
       rm.price,
       rm.amenities,
       e.employee_id,
       e.name AS employee_name,
       e.designation,
       e.gender,
       e.age
  FROM Customers c
       JOIN
       Reservations r ON c.cus_id = r.cus_id
       JOIN
       Rooms rm ON r.reserved_room_id = rm.room_id
       JOIN
       Employees e ON r.employee_id = e.employee_id
 WHERE c.debt_amount >= 100;
```

Intended result set:
The intended result is a list of all such customers with a debt amount greater than 100 along with their corresponding reservation ID, reserved room building, bed number, price, and amenities, and employee details such as ID, name, designation, gender, and age.



	cus_id	cus_name	cus_phone	cus_address	commen	freq	debt_am	reservati	reserved	availabili	paid_am	sign_in_date	sign_out_dat	building	bed_nun	price	amenities	employe	employee_r	designation	gender	age
1	1	Charlie Gray	345-678-9012	56 Mbaga St,Nairobi,Kenya	VIP	3	500	1	101	TRUE	500	2023-05-01	2023-05-03	A	1	50	TV,Wifi	2	Jane Smith	Assistant Manager	Female	35
2	3	Frank Chen	333-444-5555	64 Manick St,Texas,USA	NULL	1	1000	2	415	TRUE	750	2023-05-02	2023-05-05	B	2	75	TV,Wifi,Mini fridge	1	John Doe	Manager	Male	40

//At least one query should involve outer joins

QUERY 4

Query description:
Retrieve a list of all customers and their associated orders, including customers who have not yet placed any orders.

Query:

```
SELECT Customers.cus_name,
       Orders.order_id,
       Orders.product_id,
       Orders.amount
  FROM Customers
       LEFT JOIN
       Orders ON Customers.cus_id = Orders.cus_id;
```

Intended result set:
This query uses a LEFT OUTER JOIN to join the Customers and Orders tables. The result will include all customers, even those who haven't placed any orders yet. If a customer doesn't have any orders, the fields from the Orders table will be NULL.

	cus_name	order_id	product_id	amount
1	Charlie Gray	3	5	25
2	Ellen Kim	2	2	50
3	Frank Chen	1	1	100
4	Alice Brown	NULL	NULL	NULL
5	Bob Green	NULL	NULL	NULL
6	Semg Dile	NULL	NULL	NULL
7	Enock Niyonkuru	NULL	NULL	NULL
8	Lincoln Chapata	NULL	NULL	NULL
9	Benny Simoes	NULL	NULL	NULL

QUERY 5

Query description:
Retrieve a list of products that have not yet been ordered by any customers, their current inventory levels, and their suppliers.

Query:
```
SELECT Inventory.product_id,
       Inventory.quantity,
       Supplier.name AS supplier_name
  FROM Inventory
       LEFT JOIN
       Supplier ON Inventory.supplier_id = Supplier.supplier_id
       LEFT JOIN
       Orders ON Inventory.product_id = Orders.product_id
 WHERE Orders.product_id IS NULL;
```

Intended result set:
This query uses a LEFT OUTER JOIN to join the Inventory and Supplier tables, and another LEFT OUTER JOIN to join the Orders table. The WHERE clause filters out products that have been ordered by customers, so the result will include all products that haven't been ordered yet. For those products, the fields from the Orders table will be NULL.

	product_id	quantity	supplier_name
1	3	320	Paradise Medal LLC
2	4	100	Tura Tugabane Corp.
3	6	700	Manaturikumwe Shop

//At least one query should use an aggregate function

QUERY 6

Query description:
Retrieve the total amount of all orders made by each customer.

Query:
```
SELECT Customers.cus_name,
       SUM(Orders.amount) AS total_orders
  FROM Customers
       INNER JOIN
       Orders ON Customers.cus_id = Orders.cus_id
 GROUP BY Customers.cus_name;
```

Intended result set:
This query uses the SUM() aggregate function to calculate the total amount of all orders made by each customer. It then groups the results by customer name, so the output will include one row for each customer, with their name and the total amount of their orders.

	cus_name	total_orders
1	Charlie Gray	25
2	Ellen Kim	50
3	Frank Chen	100

QUERY 7

Query description:
Retrieve the average price and total quantity of all products supplied by each supplier.

Query:
```
SELECT Supplier.name,
       AVG(Inventory.price) AS avg_price,
       SUM(Inventory.quantity) AS total_quantity
  FROM Supplier
       INNER JOIN
       Inventory ON Supplier.supplier_id = Inventory.supplier_id
 GROUP BY Supplier.name;
```

Intended result set:

This query uses the AVG() and SUM() aggregate functions to calculate the average price and total quantity of all products supplied by each supplier. It then groups the results by supplier name, so the output will include one row for each supplier, with their name, the average price of their products, and the total quantity of their products.

	name	avg_price	total_quantity
1	Gisubizo Industries	15	9
2	Inkoko Inc.	20	400
3	Manaturikumwe Shop	50	700
4	Paradise Medal LLC	5	320
5	Tura Tugabane Corp.	15	155

//At least three queries should use subqueries in a non-trivial way
- //One of those should use a set comparison (e.g. > ***some***)

QUERY 8

Query description:
Retrieve the names of all customers who have made at least one order or have booked a room at least once.

Query:
```
SELECT cus_name
  FROM Customers
 WHERE cus_id IN (
           SELECT DISTINCT cus_id
             FROM Orders
           UNION
           SELECT DISTINCT cus_id
             FROM Reservations
       );
```

Intended result set:
This query uses a subquery to find all distinct customer IDs that appear in the Orders table or the Reservations table. It then selects the names of all customers whose IDs are in that list. The intended result is a list of customer names who have made at least one order or booked one room.

	cus_name
1	Charlie Gray
2	Ellen Kim
3	Frank Chen
4	Semg Dile
5	Enock Niyonkuru

QUERY 9

Query description:

Retrieve the product IDs and their suppliers' IDs of all products that have a lower price than the average price of all products.

Query:

```
SELECT product_id,
       supplier_id
  FROM Inventory
 WHERE price < (
                   SELECT AVG(price)
                     FROM Inventory
               );
```

Intended result set:
This query uses a subquery to calculate the average price of all products in the Inventory table. It then selects the product IDs and their suppliers' IDs of all products whose price is lower than that average. The intended result is a list of products with below-average prices.

	product_id	supplier_id
1	2	2
2	3	3
3	5	4

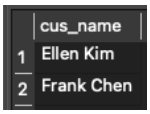
QUERY 10

Query description:
Retrieve the names of all customers who have made at least one order of a product with a quantity greater than 10.

Query:

```
SELECT cus_name
  FROM Customers
 WHERE cus_id IN (
           SELECT DISTINCT cus_id
             FROM Orders
            WHERE product_id IN (
                      SELECT product_id
                        FROM Inventory
                       WHERE quantity > 10
                  )
       );
```

Intended result set:
This query uses two subqueries to find all distinct customer IDs that have made an order for a product with a quantity greater than 10. It then selects the names of all customers whose IDs are in that list. The intended result is a list of customer names who have made at least one order of a high-quantity product.



	cus_name
1	Ellen Kim
2	Frank Chen

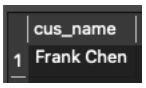
QUERY 11

Query description:
Retrieve the names of all customers who have made orders for products supplied by a supplier with the name ‘Inkoko Inc.’.

Query:

```
SELECT cus_name
  FROM Customers
 WHERE cus_id IN (
           SELECT DISTINCT cus_id
             FROM Orders
            WHERE product_id IN (
                      SELECT product_id
                        FROM Inventory
                       WHERE supplier_id = (
                                               SELECT supplier_id
                                                 FROM Supplier
                                                WHERE name = 'Inkoko Inc.'
                                           )
                  )
       );
```

Intended result set:
This query uses three subqueries to find all distinct customer IDs that have made an order for a product supplied by a supplier with the name 'Inkoko Inc.'. It then selects the names of all customers whose IDs are in that list. The intended result is a list of customer names who have made orders for products supplied by that particular supplier.



	cus_name
1	Frank Chen

QUERY 12

Query description:
Retrieve the names of all customers who have a debt amount greater than the maximum debt amount of any other customer using a self-join.

Query:

```
SELECT DISTINCT c1.cus_name
  FROM Customers c1
       JOIN
       Customers c2 ON c1.cus_id <> c2.cus_id AND
                       c1.debt_amount > c2.debt_amount;
```

Intended result set:
This query uses a self-join to compare each customer's debt amount with the debt amount of all other customers. We join the Customers table with itself on the condition that the cus_id of one row is not equal to the cus_id of the other row, and the debt_amount of the first row is greater than the debt_amount of the second row. We then select only the distinct cus_name values from the first row. The intended result is a list of customer names with the highest debt amount in the table.

	cus_name
1	Charlie Gray
2	Frank Chen
3	Alice Brown

//At least two queries should use grouping
- //At least one of those should use ***having***

QUERY 13

Query description:
List the total number of reservations made by each customer who has made at least 3 reservations.

Query:
```
SELECT cus_id,
       cus_name,
       COUNT( * ) AS num_reservations
  FROM Reservations
 GROUP BY cus_id
HAVING COUNT( * ) >= 3;
```

Intended result set:
This query first groups the reservations table by the customer ID, name, and counts the number of reservations for each customer. The HAVING clause filters the result to only include customers with at least 3 reservations. The output shows the customer ID and the total number of reservations for each qualifying customer.

	cus_id	cus_name	num_reservations
1	7	Enock Niyonkuru	3

QUERY 14

Query description:
Find the average price of products for each supplier whose total quantity of products is greater than 500.

Query:
```
SELECT Supplier.name,
       AVG(Inventory.price) AS avg_price
  FROM Inventory
       JOIN
       Supplier ON Inventory.supplier_id = Supplier.supplier_id
 GROUP BY Inventory.supplier_id
HAVING SUM(Inventory.quantity) > 500;
```

Intended result set:
This query joins the Inventory and Supplier tables on the supplier ID and groups the result by supplier ID. The HAVING clause filters the result to only include suppliers with a total quantity of products greater than 500. The output shows the supplier name and the average price of products for each qualifying supplier.

	name	avg_price
1	Manaturikumwe Shop	50

QUERY 15

Query description:
List the total amount paid by each customer who has at least one payment greater than $95.

Query:
```
SELECT payer_id, SUM(amount) AS total_amount_paid
FROM Payments
GROUP BY payer_id
HAVING MAX(amount) > 95;
```

Intended result set:
This query groups the Payments table by customer ID and sums up the amount paid by each customer. The HAVING clause filters the result to only include customers who have made at least one payment greater than $95. The output shows the customer ID and the total amount paid by each qualifying customer.

	payer_id	total_amount_paid
1	3	100
2	5	200

//At least one query should use set operations

QUERY 16

Query description:
Retrieves the employee ID, name and table name that have the same number and order of columns.

Query:
```
SELECT employee_id,
       name,
       'employee' AS table_name
  FROM Employees
UNION ALL
SELECT cus_id,
       cus_name,
       'customer' AS table_name
  FROM Customers;
```

Intended result set:
This query uses the UNION ALL set operation to combine the results of two SELECT statements that have the same number and order of columns. The result set of this query is a combination of all rows from the two tables, including duplicates if any.

	employee_id	name	table_name
1	1	John Doe	employee
2	2	Jane Smith	employee
3	3	Bob Johnson	employee
4	4	Alice Lee	employee
5	5	David Kim	employee
6	1	Charlie Gray	customer
7	2	Ellen Kim	customer
8	3	Frank Chen	customer
9	4	Alice Brown	customer
10	5	Bob Green	customer
11	6	Semg Dile	customer
12	7	Enock Niyonkuru	customer
13	8	Lincoln Chapata	customer
14	9	Benny Simoes	customer

QUERY 17

Query description:
Finds the product ID values that are both in the Inventory table with a quantity greater than 0 and in the Orders table using INTERSECT set operation.

Query:

```
SELECT product_id
  FROM Inventory
 WHERE quantity > 0
INTERSECT
SELECT product_id
  FROM Orders;
```

Intended result set:
The result set of this query is a list of product ID values that are in both tables.

	product_id
1	1
2	2
3	5

[Closing note: Remove all grey instructions like this one from your final report. Use this template to organize your final report. Submit your report as a PDF per the project instructions. Be sure everything is neat, organized, and readable. Feel free to add other sections or relevant information that may not have been included in this template. All your SQL must be executable in SQLite.]

CPSC 372 – Project

Domain: Hotel Database
Entities:

References:

TRINITY COLLEGE · DEPARTMENT OF COMPUTER SCIENCE
CPSC 372 Database Fundamentals, Spring 2023
Database Project, Yves Semana Gisubizo

https://www.freeprojectz.com/entity-relationship/hotel-management-system-er-diagram
https://www.edrawmax.com/templates/1000527/