

# Hotel Database Management System

Toronto Metropolitan University

Yijuan Huang - 501160292

Nigel Siddeley - 501186392

CPS510: Database Systems I

Professor Abhari Abdolreza

Section 5

November 30, 2024

# Table Of Contents

Application Description	2
ER Diagram	4
Schema Design And Database Construction	5
Design Views And Simple Queries	8
Advanced Queries	12
Unix Shell	14
Functional Dependencies	15
Normalization Into 3NF	16
Normalized Using Bcnf/Bernstein Algorithm	18
Java UI	20
Relational Algebra For Queries	23

## Description

Hotels have always kept ledgers for bookkeeping purposes, as it is self-evident that a hotel should be able to keep track of guests and current room reservations to provide service to incipient and current guests. The near universal proliferation of computer technology over the past few decades has seen most hotels transition from cumbersome physical ledgers for storing information to computerized databases. While a simple excel sheet may suffice for a smaller motel, larger hotels will have to rely on more comprehensive data models to keep track of rooms, availability, employee status, guests and numerous other factors. This complexity is especially pronounced in chain hotels, which could have hundreds of different locations spread over

numerous nations and with employee counts in the thousands, all of which will require detailed payroll information. This requires an accessible and logical database to keep note of all a hotel chain's operations and employees. This database should be able to output both individualized items or groupings based on queries given by the user and should be available for use by any high enough manager in the hotel firm to manage a hotel.

By necessity, any hotel management system will have to keep track of a hotel's various rooms, maintaining detailed information about a room's number, hotel location, suite, occupancy status, price, and other details such as the number of beds and requested client accommodations. All this information should be displayed in a concise and logical manner for the benefit of the user. The primary key to a room is its number, which should be unique for a given hotel. There should also be the ability to view attached notes for a given room to see what services and cleaning must be done, such as restocking a minibar or repainting of the walls. Rooms rendered unavailable for reasons other than a guest should also have a system to explain why the room cannot be booked.

Guests who will be staying at a hotel will need to also be accounted for, with each being given a

unique customer ID as a primary attribute with other accompanying information such as name, age, room occupancy, length of stay and any special accommodations needed. A manager should

also be able to blacklist troublesome guests to prevent them from utilizing the services of a hotel.

Employees will also be assigned a unique ID for identification purposes, and among other things their info will require their hiring date, name, age, position and salary.

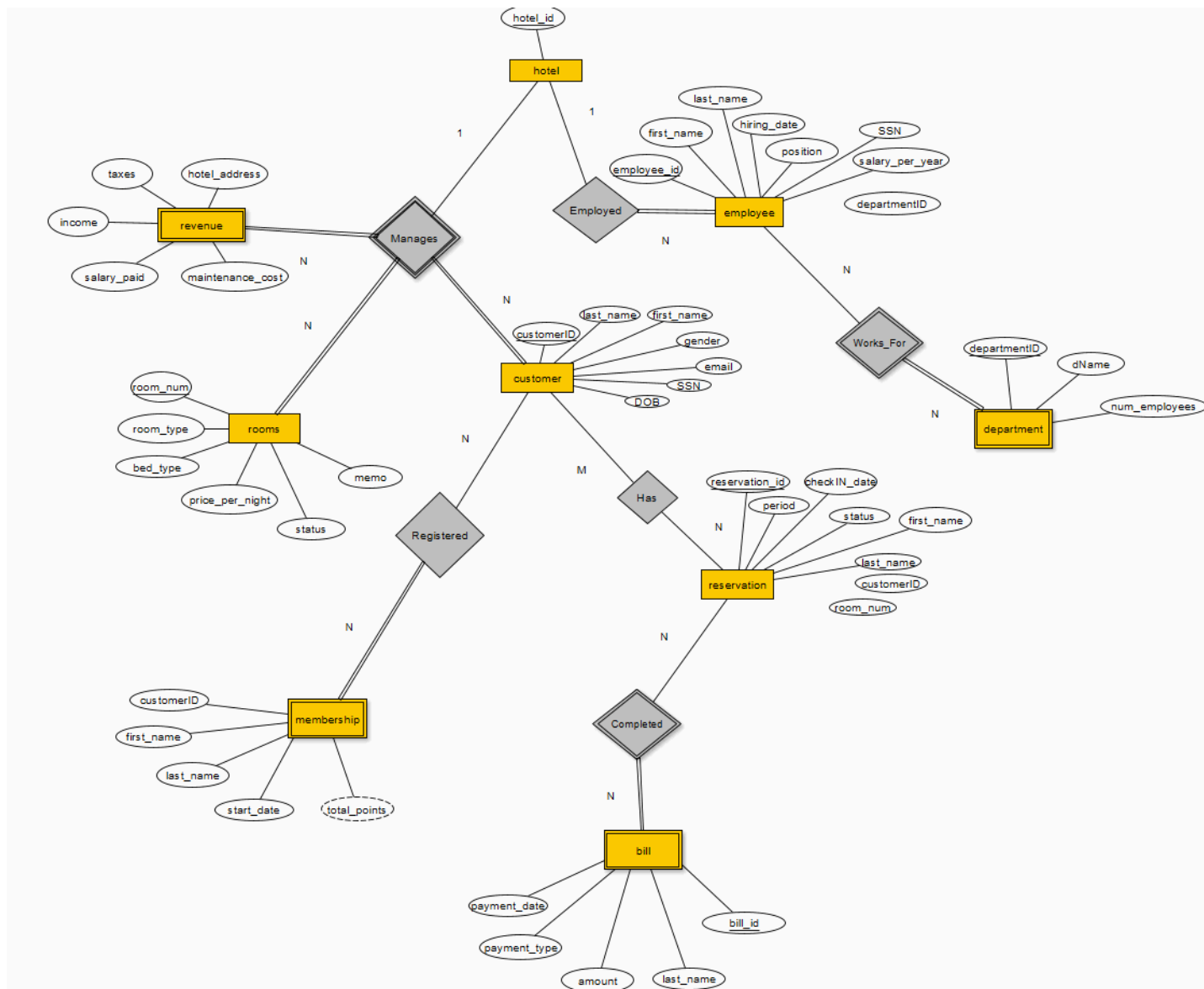
A manager should also be allowed to take a glance at a hotel's revenue and expenses, with relations indicating things such as income, salaries, taxes, operational overhead and maintenance

costs, and other unexpected expenses such as lawsuits and repairs. This information should be displayed in a segregated view for individual revenues and expenses but also sum to an overall cash flow figure to see the financial health of the hotel.

Incoming reservations should also be managed with a unique identifying number as a primary key. A reservation should be displayed along with its length, price, starting date and ending date, as well as the guest(s) who called for the reservation.

With this framework in mind, we hope to model an effective hotel DBMS that allows a manager to get a complete picture of their hotel's current operations so that they may undertake business decisions in a more sophisticated manner armed with accurate information.

# ER Diagram



# Schema Design And Database Construction

## Create tables

```

hotel_tables.sql
SQL Worksheet History
Worksheet Query Builder

create table Hotel (
    hotel_id number(9) PRIMARY KEY
);
--drop table Revenue;

create table Revenue(
    income number(11,2) DEFAULT 0,
    taxes number(9,2) DEFAULT 0,
    salary_paid number(9,2) DEFAULT 0,
    maintenance_cost number(8,2) DEFAULT 0,
    hotel_address varchar2(100) -- Composite attribute
);

--drop table Department;
create table Department(
    department_id number(2) PRIMARY KEY,
    department_name varchar2(30),
    num_employees number(3)
);

--drop table Employee;
create table Employee (
    employee_id number(5) PRIMARY KEY,
    first_name varchar2(10),
    last_name varchar2(10),
    hiring_date DATE,
    position_ varchar2(30),
    SSN number(9),
    salaryPer_year NUMBER(10,2),
    department_id number(2),
    FOREIGN KEY (department_id)
        REFERENCES Department(department_id)
);

create table Customer(
    customer_id varchar2(20) PRIMARY KEY,
    first_name varchar2(10),
    last_name varchar2(10),
    gender varchar2(10),
    email varchar(40),
    SSN number(9),
    DOB DATE NOT NULL
);

create table Reservation(
    reservation_id varchar(30) PRIMARY KEY,
    customer_id varchar2(20),
    room_num number(6),
    stay_period INT,
    checkIN_date DATE,
    status char(20) CHECK (status IN ('confirmed','checked-in','checked-out')),
    first_name varchar2(10),
    last_name varchar2(10)
);

```

```

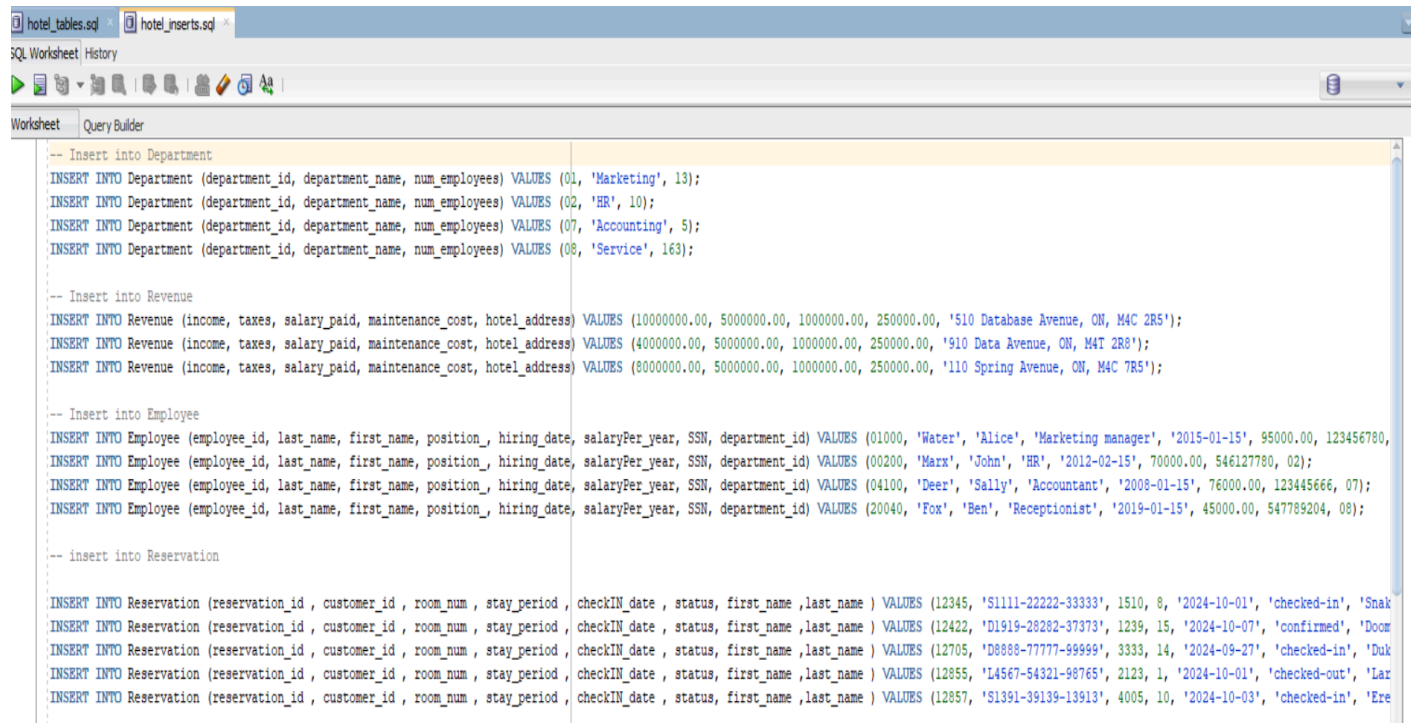
create table Bill(
    bill_id number(9) PRIMARY KEY,
    last_name varchar2(10),
    amount number(10,2),
    payment_type char(10),
    payment_date DATE
);

create table Room(
    room_num number(6) PRIMARY KEY,
    bed_type char(30),
    room_type char(30),
    price_per_night INT,
    status varchar2(20) CHECK (status IN ('available','occupied','under maintenance')), --This constraint ensur
    memo varchar2(100)
);

create table Membership(
    customer_id varchar2(20),
    first_name varchar2(10),
    last_name varchar2(10),
    start_date DATE,
    total_points number(7),
    FOREIGN KEY (customer_id)
        REFERENCES Customer(customer_id)
);

```

## Populate tables with data



```

-- Insert into Department
INSERT INTO Department (department_id, department_name, num_employees) VALUES (01, 'Marketing', 13);
INSERT INTO Department (department_id, department_name, num_employees) VALUES (02, 'HR', 10);
INSERT INTO Department (department_id, department_name, num_employees) VALUES (07, 'Accounting', 5);
INSERT INTO Department (department_id, department_name, num_employees) VALUES (08, 'Service', 163);

-- Insert into Revenue
INSERT INTO Revenue (income, taxes, salary_paid, maintenance_cost, hotel_address) VALUES (10000000.00, 5000000.00, 1000000.00, 250000.00, '510 Database Avenue, ON, M4C 2R5');
INSERT INTO Revenue (income, taxes, salary_paid, maintenance_cost, hotel_address) VALUES (4000000.00, 5000000.00, 1000000.00, 250000.00, '910 Data Avenue, ON, M4T 2R8');
INSERT INTO Revenue (income, taxes, salary_paid, maintenance_cost, hotel_address) VALUES (8000000.00, 5000000.00, 1000000.00, 250000.00, '110 Spring Avenue, ON, M4C 7R5');

-- Insert into Employee
INSERT INTO Employee (employee_id, last_name, first_name, position, hiring_date, salaryPer_year, SSN, department_id) VALUES (01000, 'Water', 'Alice', 'Marketing manager', '2015-01-15', 95000.00, 123456780, 01);
INSERT INTO Employee (employee_id, last_name, first_name, position, hiring_date, salaryPer_year, SSN, department_id) VALUES (00200, 'Marx', 'John', 'HR', '2012-02-15', 70000.00, 546127780, 02);
INSERT INTO Employee (employee_id, last_name, first_name, position, hiring_date, salaryPer_year, SSN, department_id) VALUES (04100, 'Deer', 'Sally', 'Accountant', '2008-01-15', 76000.00, 12345666, 07);
INSERT INTO Employee (employee_id, last_name, first_name, position, hiring_date, salaryPer_year, SSN, department_id) VALUES (20040, 'Fox', 'Ben', 'Receptionist', '2019-01-15', 45000.00, 547789204, 08);

-- insert into Reservation
INSERT INTO Reservation (reservation_id, customer_id, room_num, stay_period, checkIN_date, status, first_name, last_name) VALUES (12345, 'S1111-22222-33333', 1510, 8, '2024-10-01', 'checked-in', 'Snak', 'Doom');
INSERT INTO Reservation (reservation_id, customer_id, room_num, stay_period, checkIN_date, status, first_name, last_name) VALUES (12422, 'D1919-28282-37373', 1239, 15, '2024-10-07', 'confirmed', 'Doom', 'Doom');
INSERT INTO Reservation (reservation_id, customer_id, room_num, stay_period, checkIN_date, status, first_name, last_name) VALUES (12705, 'D8888-77777-99999', 3333, 14, '2024-09-27', 'checked-in', 'Duk', 'Duk');
INSERT INTO Reservation (reservation_id, customer_id, room_num, stay_period, checkIN_date, status, first_name, last_name) VALUES (12855, 'L4567-54321-98765', 2123, 1, '2024-10-01', 'checked-out', 'Lar', 'Lar');
INSERT INTO Reservation (reservation_id, customer_id, room_num, stay_period, checkIN_date, status, first_name, last_name) VALUES (12857, 'S1391-39139-13913', 4005, 10, '2024-10-03', 'checked-in', 'Ere', 'Ere');

```

```

-- insert into Rooms
INSERT INTO Room (room_num, bed_type, room_type, price_per_night, status) VALUES (1510, 'Double', 'Single Room', 60, 'occupied');
INSERT INTO Room (room_num, bed_type, room_type, price_per_night, status) VALUES (1239, 'Double', 'Double Room', 100, 'occupied');
INSERT INTO Room (room_num, bed_type, room_type, price_per_night, status) VALUES (3333, 'King', 'Deluxe Room', 300, 'occupied');
INSERT INTO Room (room_num, bed_type, room_type, price_per_night, status) VALUES (2123, 'Queen', 'Queen Room', 150, 'available');
INSERT INTO Room (room_num, bed_type, room_type, price_per_night, status) VALUES (4005, 'King', 'Presidential Suite', 2000, 'occupied');
INSERT INTO Room (room_num, bed_type, room_type, price_per_night, status) VALUES (1414, 'Double', 'Single Room', 60, 'under maintenance');
INSERT INTO Room (room_num, bed_type, room_type, price_per_night, status) VALUES (2250, 'King', 'King Room', 200, 'available');
INSERT INTO Room (room_num, bed_type, room_type, price_per_night, status) VALUES (4001, 'Queen', 'Suite', 1000, 'occupied');
INSERT INTO Room (room_num, bed_type, room_type, price_per_night, status) VALUES (3346, 'Double', 'Studio', 400, 'under maintenance');

-- insert into Bill
INSERT INTO Bill (bill_id, last_name, amount, payment_type, payment_date) VALUES (000000011, 'White', 22568.52, 'Cash', '2004-02-15');
INSERT INTO Bill (bill_id, last_name, amount, payment_type, payment_date) VALUES (001234007, 'Bond', 1507.29, 'Credit', '1998-06-01');
INSERT INTO Bill (bill_id, last_name, amount, payment_type, payment_date) VALUES (000013139, 'Jaeger', 663.13, 'Debit', '2013-08-12');
INSERT INTO Bill (bill_id, last_name, amount, payment_type, payment_date) VALUES (001234567, 'Smith', 213.57, 'Credit', '2024-09-29');
INSERT INTO Bill (bill_id, last_name, amount, payment_type, payment_date) VALUES (000007824, 'Andrews', 10000.01, 'Credit', '2023-07-15');

-- insert into Customer
INSERT INTO Customer (customer_id, first_name, last_name, gender, email, ssn, dob) VALUES ('S1111-22222-33333', 'Snake', 'Flisken', 'Male', 'metalgearman@gmail.com', 132435467, '1990-05-05');
INSERT INTO Customer (customer_id, first_name, last_name, gender, email, ssn, dob) VALUES ('D8888-77777-99999', 'Duke', 'Mukem', 'Male', 'thenukeduke@gmail.com', 555555555, '1989-07-29');
INSERT INTO Customer (customer_id, first_name, last_name, gender, email, ssn, dob) VALUES ('S4125-59190-59326', 'John', 'Smith', 'Male', 'thejohnsmith12@gmail.com', 332130382, '1955-02-25');
INSERT INTO Customer (customer_id, first_name, last_name, gender, email, ssn, dob) VALUES ('S1391-39139-13913', 'Eren', 'Jaeger', 'Male', 'titanman@paradis.com', 132000139, '1990-03-30');
INSERT INTO Customer (customer_id, first_name, last_name, gender, email, ssn, dob) VALUES ('K1234-12345-67890', 'Sally', 'Wrinkle', 'Female', 'imold123@fossil.com', 000000001, '1812-01-01');
INSERT INTO Customer (customer_id, first_name, last_name, gender, email, ssn, dob) VALUES ('I1111-11111-11111', 'Nigel', 'Siddeley', 'Male', 'nsiddeley@gmail.com', 123123123, '2004-02-15');

-- insert into Membership
INSERT INTO Membership (customer_id, first_name, last_name, start_date, total_points) VALUES ('S1111-22222-33333', 'Snake', 'Flisken', '2022-01-25', 5055);
INSERT INTO Membership (customer_id, first_name, last_name, start_date, total_points) VALUES ('D8888-77777-99999', 'Duke', 'Mukem', '2023-11-11', 8888);
INSERT INTO Membership (customer_id, first_name, last_name, start_date, total_points) VALUES ('S4125-59190-59326', 'John', 'Smith', '2000-05-05', 12345);
INSERT INTO Membership (customer_id, first_name, last_name, start_date, total_points) VALUES ('S1391-39139-13913', 'Eren', 'Jaeger', '2017-07-10', 1391);
INSERT INTO Membership (customer_id, first_name, last_name, start_date, total_points) VALUES ('K1234-12345-67890', 'Sally', 'Wrinkle', '1900-02-02', 999999);

```



# Views And Simple Queries

## Creating views

```
create view vwToDoSalesPromotion(first_name, last_name, email, total_points)
as
    select  Customer.first_name,
            Customer.last_name,
            Customer.email,
            Membership.total_points
    from Customer
    inner join Membership on Customer.customer_id=Membership.customer_id
    where Membership.total_points > 10000;

create view vwRoomsAvailability(current_status, room_num, price_per_night, room_type)
as
    (select status,
            room_num,
            price_per_night,
            room_type
    from Room
    where status='available');

create view vwEmployedLength(employee_id, first_name, position_, hiring_date)
as
    (select employee_id,
            first_name,
            position_,
            hiring_date
    from Employee
    where hiring_date < '2015-01-01');

select * from vwToDoSalesPromotion;
select * from vwRoomsAvailability;
select * from vwEmployedLength;
```

Script Output x

Task completed in 0.067 seconds

View VWTODOSALASPROMOTION created.

View VWROOMSAVAILABILITY created.

View VWEMPLOYEDLENGTH created.

## Outputs of the views

Script Output x Query Result x Query Result 1 x Query Result 2 x

SQL | All Rows Fetched: 2 in 0.07 seconds

	FIRST_NAME	LAST_NAME	EMAIL	TOTAL_POINTS
1	John	Smith	thejohnsmith12@gmail.com	12345
2	Sally	Wrinkle	imold123@fossil.com	999999

Script Output x Query Result x Query Result 1 x Query Result 2 x

SQL | All Rows Fetched: 3 in 0.015 seconds

	CURRENT_STATUS	PRICE_PER_NIGHT	ROOM_TYPE
1	available	60	Single Room
2	available	150	Queen Room
3	available	200	King Room

Script Output x Query Result x Query Result 1 x Query Result 2 x

SQL | All Rows Fetched: 2 in 0.024 seconds

	EMPLOYEE_ID	FIRST_NAME	POSITION_	HIRING_DATE
1	200	John	HR	15-FEB-13
2	4100	Sally	Accountant	15-JAN-08

## Simple Queries

```
-- Query that uses ORDER BY to get employees'salary in descending order.
select
    employee_id,
    last_name,
    position_,
    salaryPer_year as salary
from Employee
order by salary DESC;
```

Script Output x Query Result x

SQL | All Rows Fetched: 4 in 0.018 seconds

	EMPLOYEE_ID	LAST_NAME	POSITION_	SALARY
1	1000	Water	Marketing manager	95000
2	4100	Deer	Accountant	76000
3	200	Marx	HR	70000
4	20040	Fox	Receptionist	45000

```
-- Query to find hotels' income that are greater than a specific number which is 5,000,000
select
    income,
    hotel_address
from Revenue
where income >= 5000000;
```

Script Output x Query Result x

SQL | All Rows Fetched: 2 in 0.014 seconds

	INCOME	HOTEL_ADDRESS
1	10000000	510 Database Avenue, ON, M4C 2R5
2	8000000	110 Spring Avenue, ON, M4C 7R5

```
-- Query to find bills that exceed $10,000 in descending order
```

```
select
    bill_id,
    last_name,
    payment_date,
    amount
from Bill
where amount >= 10000 order by amount DESC;
```

Query Result x

SQL | All Rows Fetched: 2 in 0.016 seconds

	BILL_ID	LAST_NAME	PAYMENT_DATE	AMOUNT
1	11	White	15-FEB-04	22568.52
2	7824	Andrews	15-JUL-23	10000.01

```
-- Query to find departments with 100 or more employees
```

```
select
    department_name,
    num_employees
from Department
where num_employees >= 100;
```

Query Result x

SQL | All Rows Fetched: 1 in 0.016 seconds

	DEPARTMENT_NAME	NUM_EMPLOYEES
1	Service	163

```
-- Query to find available rooms
```

```
select
    room_num,
    bed_type,
    room_type,
    price_per_night
from Room
where status = 'available';
```

Query Result x

SQL | All Rows Fetched: 2 in 0.028 seconds

	ROOM_NUM	BED_TYPE	ROOM_TYPE	PRICE_PER_NIGHT
1	2123	Queen	Queen Room	150
2	2250	King	King Room	200

```
-- Query to find available rooms
```

```
select
    room_num,
    bed_type,
    room_type,
    price_per_night
from Room
where status = 'available';
```

Query Result x

SQL | All Rows Fetched: 2 in 0.028 seconds

	ROOM_NUM	BED_TYPE	ROOM_TYPE	PRICE_PER_NIGHT
1	2123	Queen	Queen Room	150
2	2250	King	King Room	200

## Advanced queries

```
-- =====
-- ADVANCED QUERIES BELOW
-- =====

-- showing customers who have checked-in in the month of October 2024
select
  customer_id,
  first_name,
  last_name
from Customer
where customer_id
  in (select customer_id
      from Reservation
      where checkIN_date >= TO_DATE('2024-10-01','YYYY-MM-DD') and checkIN_date <= TO_DATE('2024-10-31','YYYY-MM-DD'));
```

Query Result x

SQL | All Rows Fetched: 2 in 0.014 seconds

	CUSTOMER_ID	FIRST_NAME	LAST_NAME
1	S1111-22222-33333	Snake	Plisken
2	S1391-39139-13913	Eren	Jaeger

```
-- lists total revenue sorted by payment type
SELECT payment_type, SUM(amount) AS total_revenue
FROM Bill
GROUP BY payment_type
HAVING SUM(amount) > 0;
```

Query Result x

SQL | All Rows Fetched: 3 in 0.022 seconds

	PAYMENT_TYPE	TOTAL_REVENUE
1	Debit	663.13
2	Cash	22568.52
3	Credit	11720.87

```
-- lists rooms that are available or occupied
SELECT room_num, bed_type, room_type, status
FROM Room
WHERE status = 'available'
UNION
SELECT room_num, bed_type, room_type, status
FROM Room
WHERE status = 'occupied';
```

Query Result x

SQL | All Rows Fetched: 7 in 0.017 seconds

	ROOM_NUM	BED_TYPE	ROOM_TYPE	STATUS
1	1239	Double	Double Room	occupied
2	1510	Double	Single Room	occupied
3	2123	Queen	Queen Room	available
4	2250	King	King Room	available
5	3333	King	Deluxe Room	occupied
6	4001	Queen	Suite	occupied
7	4005	King	Presidential Suite	occupied

```
-- lists average price of each room type
SELECT room_type, AVG(price_per_night) AS avg_price
FROM Room
GROUP BY room_type
HAVING AVG(price_per_night) > 150;
```

Query Result x

SQL | All Rows Fetched: 5 in 0.017 seconds

	ROOM_TYPE	AVG_PRICE
1	King Room	200
2	Presidential Suite	2000
3	Suite	1000
4	Studio	400
5	Deluxe Room	300

```
-- lists departments that have employees hired before 2014
SELECT department_id, department_name
FROM Department d
WHERE EXISTS (
  SELECT 1
  FROM Employee e
  WHERE e.department_id = d.department_id
  AND e.hiring_date < TO_DATE('2014-01-01', 'YYYY-MM-DD')
);
```

Query Result x

SQL | All Rows Fetched: 2 in 0.023 seconds

	DEPARTMENT_ID	DEPARTMENT_NAME
1	2	HR
2	7	Accounting



# Functional Dependency

FD for strong entities:

## Department

{department\_id} → department\_name, num\_employees

## Employee

{employee\_id} → last\_name, first\_name, position, hiring\_date, salaryPer\_year, SSN,  
department\_id

## Customer

{customer\_id} → first\_name, last\_name, gender, email, ssn, dob

## Room

{room\_num} → bed\_type, room\_type, price\_per\_night, status, memo

## Reservation

{reservation\_id} → customer\_id, room\_num, stay\_period, checkIN\_date, status,  
first\_name, last\_name

FD for weak entities:

## Bill

{reservation\_id, bill\_id} → last\_name, amount, payment\_type, payment\_date

## Membership

{customer\_id} → first\_name, last\_name, start\_date, total\_points



## 3NF

### Ex. 2: Table with a transitive FD to 3NF

Employee(emp\_id, emp\_name, dept\_id, dept\_name, location)

The above table schema has the following FDs:

$\text{emp\_id} \rightarrow \text{emp\_name}, \text{dept\_id}$

$\text{dept\_id} \rightarrow \text{dept\_name}, \text{location}$

The FD in red is a **transitive dependency**, as dept\_name and location are determined by dept\_id, which is determined by emp\_id, so dept\_name and location are indirectly dependent on emp\_id meaning this table violates 3NF. In order to solve this we must decompose the table into two separate tables: Employee and Department.

Employee(emp\_id, emp\_name, dept\_id)

$\{\text{emp\_id}\} \rightarrow \{\text{emp\_name}, \text{dept\_id}\}$

Department(dept\_id, dept\_name, location)

$\{\text{dept\_id}\} \rightarrow \{\text{dept\_name}, \text{location}\}$

Now both tables are in 2NF, and have no transitive dependencies, making them 3NF.

### Hotel: Not In 3NF

Hotel(hotel\_id, hotel\_address)

The hotel table has a composite attribute hotel\_address (violates 1NF) so it must be broken down into its components

Hotel(hotel\_id, province, city, street\_name, street\_number, postal\_code)

$\{\text{hotel\_id}\} \rightarrow \{\text{province}, \text{city}, \text{street\_name}, \text{street\_number}, \text{postal\_code}\}$

The schema now satisfies 1NF, and has no partial or transitive dependencies meaning it is **in 3NF**

### Revenue: Not In 3NF

Revenue(hotel\_id, fiscal\_year, fiscal\_quarter, income, taxes, salary\_paid, maintenance\_cost, hotel\_address)

The revenue table has a partial dependency  $\{\text{hotel\_id}\} \rightarrow \{\text{hotel\_address}\}$  (violates 2NF) so in order to fix it we must remove hotel\_address from the Revenue table:

Revenue(hotel\_id, fiscal\_year, fiscal\_quarter, income, taxes, salary\_paid, maintenance\_cost)

$\{\text{hotel\_id}, \text{fiscal\_year}, \text{fiscal\_quarter}\} \rightarrow \{\text{income}, \text{taxes}, \text{salary\_paid}, \text{maintenance\_cost}\}$

This schema now satisfies 2NF and has no transitive dependencies meaning it is **in 3NF**

**Reservation: Not In 3NF**

Reservation(res\_id, customer\_id, room\_num, stay\_period, check\_in\_date, status, f\_name, l\_name)

The reservation table has a partial dependency {customer\_id} -> {f\_name, l\_name} (violates 2NF) so in order to fix it we must remove f\_name and l\_name from the table:

Reservation(res\_id, customer\_id, room\_num, stay\_period, check\_in\_date, status)  
 {res\_id} → {customer\_id, room\_num, stay\_period, check\_in\_date, status}

This schema now satisfies 2NF and has no transitive dependencies meaning it is **in 3NF**

**Bill: Not In 3NF**

Bill(bill\_id, l\_name, amount, payment\_type, payment\_date)

To make into a 3NF, remove last\_name, and replace by "customer\_id", so that  
 The last\_name can be accessed by "customer\_id"

Bill(bill\_id, customer\_id, amount, payment\_type, payment\_date)  
 {bill\_id} → {customer\_id, amount, payment\_type, payment\_date}.

**Room: In 3NF**

Room(room\_num, bed\_type, room\_type, price\_per\_night, status, memo)  
 {room\_num} → {bed\_type, room\_type, price\_per\_night, status, memo}

**Membership: Not In 3NF**

Membership(customer\_id, f\_name, l\_name, start\_date, total\_points)

The membership table has redundant attributes f\_name and l\_name (connected by customer\_id) so we must remove them

Membership(customer\_id, start\_date, total\_points)  
 {customer\_id} → {start\_date, total\_points}

This schema now satisfies 2NF and has no transitive dependencies meaning it is **in 3NF**

## BCNF And Bernstein Algorithm

Find one table that has partial dependency and transitive dependency, and then use both BCNF and Bernstein algorithm to decompose the table.

Revenue(hotel\_id, fiscal\_year, fiscal\_quarter, income, taxes, salary\_paid, maintenance\_cost, hotel\_address, postal\_code)

Transitive dependency:

Hotel\_address -> postal\_code

Partial dependency:

{hotel\_id} -> hotel\_address

{hotel\_id} -> postal\_code

Map:

hotel\_id = A, fiscal\_year = B, fiscal\_quarter = C

income = D, taxes = E, salary\_paid = F, maintenance\_cost = G,

hotel\_address = H, postal\_code = I

Using Bernstein algorithm for Revenue table

1. FD={ABC -> DEFG,

A -> HI,

H -> I}

2. FD={ABC -> D,

ABC -> E,

ABC -> F,

ABC -> G,

A -> H,

A -> I,

H -> I}

ABC -> D: {ABC}<sup>+</sup>={ABCEFGHI}

no D, not redundant

ABC -> E: {ABC}<sup>+</sup>={ABCEFGHI}

no E, not redundant

ABC -> F: {ABC}<sup>+</sup>={ABCEFGHI}

no F, not redundant

ABC -> G: {ABC}<sup>+</sup>={ABCEFGHI}

no G, not redundant

A -> H: {A}<sup>+</sup>={AI}

no H, not redundant

A -> I: {A}<sup>+</sup>={AHI}

yes I, redundant

H -> I: {H}<sup>+</sup>={H}

no I, not redundant

After removing redundancies: FD={ABC -> D, ABC -> E, ABC -> F, ABC -> G, A -> H, H -> I,}

Minimizing left side by removing partial dependencies:

ABC -> D    {A}<sup>+</sup>={AHI}    {B}<sup>+</sup>={B}    {C}<sup>+</sup>={C}    no D, FD is fully dependent

ABC -> E    {A}<sup>+</sup>={AHI}    {B}<sup>+</sup>={B}    {C}<sup>+</sup>={C}    no E, FD is fully dependent

ABC -> F    {A}<sup>+</sup>={AHI}    {B}<sup>+</sup>={B}    {C}<sup>+</sup>={C}    no F, FD is fully dependent

$ABC \rightarrow G$      $\{A\}^+ = \{AHI\}$      $\{B\}^+ = \{B\}$      $\{C\}^+ = \{C\}$     no G, FD is fully dependent

3.  $\{DEFG\}$  only appear on RHS so they are not part of keys

$\{ABC\}$  only appear on LHS so they are part of keys

Thus,  $\{ABC\}^+ = \{ABCDEFGHI\}$

$\{ABCH\}^+ = \{ABCDEFGHI\}$

$\{ABCI\}^+ = \{ABCDEFGHI\}$

all are keys

4.  $FD = \{ABC \rightarrow D, ABC \rightarrow E, ABC \rightarrow F, ABC \rightarrow G, A \rightarrow H, H \rightarrow I, \}$

$FD = \{ABC \rightarrow DEFG, A \rightarrow H, H \rightarrow I, \}$

New tables

$R1 = (ABCDEFG)$      $R2 = (AH)$      $R3 = (HI)$

Using BCNF for Revenue table

$R = (ABCDEFGHI)$

$FD = \{ABC \rightarrow DEFG,$

$A \rightarrow HI,$

$H \rightarrow I\}$

$\{ABC\}^+ = \{ABCDEFGHI\}$

$\{A\}^+ = \{AHI\}$

$\{H\}^+ = \{HI\}$

Since R is not bcnf with respect to  $A \rightarrow HI$  and  $H \rightarrow I$

Decompose into R1 and R2, where

$R1 = (ABCDEFG)$  is bcnf

$R2 = (AHI)$  is bcnf

Thus, bcnf schema of R is

$R1 = (ABCDEFG)$

$R2 = (AHI)$

# JAVA UI

```

9 public class Jdbc {
10
11     private static final String DB_URL = ;
12
13     public static void main(String[] args) {
14         String response = "a";
15         Scanner in = new Scanner(System.in);
16
17         while (response != null) {
18             System.out.println("=====");
19             System.out.println("Welcome to the database manager");
20             System.out.println("");
21             System.out.println("( 1 ) > Drop Tables");
22             System.out.println("( 2 ) > Create Tables");
23             System.out.println("( 3 ) > Populate Tables");
24             System.out.println("( 4 ) > Query Tables");
25             System.out.println("( 5 ) > Update Tables");
26             System.out.println("( 6 ) > Alter Tables");
27             System.out.println("");
28             System.out.println("( E ) > Exit");
29             System.out.println("");
30             System.out.println("=====");
31             System.out.println("");
32             System.out.println("Enter option > ");
33
34             response = in.nextLine();
35
36             if(response.equals("1")) {
37                 dropTables();
38             } else if(response.equals("2")) {
39                 createTables();
40             } else if(response.equals("3")) {
41                 populateTables();
42             } else if(response.equals("4")) {
43                 customQuery();
44             } else if(response.equals("5")) {
45                 customUpdate();
46             } else if(response.equals("6")) {
47                 customAlter();
48             } else if(response.toUpperCase().equals("E")) {
49                 in.close();
50                 System.exit(0);
51             } else {
52                 System.out.println("Please enter a valid option");
53             }
54         }
55     }
56 }

```

```

57     private static void dropTables(String dbURL) {
58
59         String[] statements = {
60             "drop table room cascade constraints",
61             "drop table revenue cascade constraints",
62             "drop table reservation cascade constraints",
63             "drop table membership cascade constraints",
64             "drop table hotel cascade constraints",
65             "drop table employee cascade constraints",
66             "drop table department cascade constraints",
67             "drop table customer cascade constraints",
68             "drop table bill cascade constraints"
69         };
70
71         for(String drop : statements) {
72
73             try (Connection conn = DriverManager.getConnection(dbURL);
74                 Statement stmt = conn.createStatement();)
75             {
76
77                 stmt.executeUpdate(drop);
78                 System.out.println("Table dropped.");
79                 conn.close();
80
81             } catch (SQLException e) {
82                 System.out.println("error: ");
83                 e.printStackTrace();
84             }
85         }
86     }
87
88 }

```



```

263 private static void customQuery(String dbURL) {
264
265     Scanner scan = new Scanner(System.in);
266     System.out.println("Enter a query with proper SQL syntax > ");
267     String query = scan.nextLine();
268
269     try {
270         Connection conn = DriverManager.getConnection(dbURL);
271         Statement stmt = conn.createStatement();
272
273         ResultSet rs = stmt.executeQuery(query);
274         ResultSetMetaData rsmd = rs.getMetaData();
275
276         int numCols = rsmd.getColumnCount();
277
278         for(int i=1; i<=numCols; i++) {
279             String colName = rsmd.getColumnName(i);
280             System.out.printf("%-25s", colName);
281         }
282         System.out.print("\n");
283
284         while(rs.next()) {
285             for(int i=1; i<=numCols; i++) {
286                 System.out.printf("%-25s", rs.getObject(i).toString());
287             }
288             System.out.print("\n");
289         }
290
291         conn.close();
292     } catch (SQLException e) {
293         System.out.println("error: ");
294         e.printStackTrace();
295     }
296 }
297
298

```

```

300 private static void customUpdate(String dbURL) {
301
302     Scanner scan = new Scanner(System.in);
303     System.out.println("Enter one of the following statements (INSERT, UPDATE, DELETE) with proper SQL syntax > ");
304     String update = scan.nextLine();
305
306     try {
307         Connection conn = DriverManager.getConnection(dbURL);
308         Statement stmt = conn.createStatement();
309
310         int rowsAff = stmt.executeUpdate(update);
311         System.out.println(String.format("%d rows affected", rowsAff));
312
313         conn.close();
314     } catch (SQLException e) {
315         System.out.println("error: ");
316         e.printStackTrace();
317     }
318 }
319
320

```

```

321 private static void customAlter(String dbURL) {
322
323     Scanner scan = new Scanner(System.in);
324     System.out.println("Enter an ALTER statement with proper SQL syntax > ");
325     String update = scan.nextLine();
326
327     try {
328         Connection conn = DriverManager.getConnection(dbURL);
329         Statement stmt = conn.createStatement();
330
331         int rowsAff = stmt.executeUpdate(update);
332         System.out.println("Column altered");
333         System.out.println(String.format("%d rows affected", rowsAff));
334
335         conn.close();
336     } catch (SQLException e) {
337         System.out.println("error: ");
338         e.printStackTrace();
339     }
340 }
341
342

```

## Example output:

```

=====
Welcome to the database manager

( 1 ) > Drop Tables
( 2 ) > Create Tables
( 3 ) > Populate Tables
( 4 ) > Query Tables
( 5 ) > Update Tables
( 6 ) > Alter Tables

( E ) > Exit

=====

Enter option >
4
Enter a query with proper SQL syntax >
select * from employee

```

EMPLOYEE_ID	LNAM	FNAME	HIRE_DATE	POSITION_	SSN	SALARY	DEPARTMENT_ID
1000	Water	Alice	2015-01-15 00:00:00.0	Marketing manager	123456780	95000	1
200	Marx	John	2012-02-15 00:00:00.0	HR	546127780	70000	2
4100	Deer	Sally	2008-01-15 00:00:00.0	Accountant	123445666	76000	7
20040	Fox	Ben	2019-01-15 00:00:00.0	Receptionist	547789204	45000	8
20041	Bobson	Bob	2019-01-15 00:00:00.0	Janitor	547789205	35000	8

## Relational Algebra For Queries

<pre>select   department_name,   num_employees from Department where num_employees &gt;= 100;</pre>	$\sigma_{\text{num\_employees} \geq 100}(\text{Department})$
<pre>select   first_name,   last_name,   stay_period,   status from Reservation where status = 'checked-in';</pre>	$\sigma_{\text{status} = \text{'checked-in'}}(\text{Reservation})$
<pre>SELECT room_num, bed_type, room_type, status FROM Room WHERE status = 'available' UNION SELECT room_num, bed_type, room_type, status FROM Room WHERE status = 'occupied';</pre>	$\pi_{\text{room\_num}, \text{bed\_type}, \text{room\_type}, \text{status}}(\sigma_{\text{status} = \text{'available'}}(\text{Room}))$ $\cup$ $\pi_{\text{room\_num}, \text{bed\_type}, \text{room\_type}, \text{status}}(\sigma_{\text{status} = \text{'occupied'}}(\text{Room}))$



```
SELECT customer_id, first_name, last_name
FROM Customer
MINUS
SELECT customer_id, first_name, last_name
FROM Membership;
```

$$\pi_{\text{customer\_id, first\_name, last\_name}}(\text{Customer})$$

$$-$$

$$\pi_{\text{customer\_id, first\_name, last\_name}}(\text{Membership})$$

```
select
  first_name,
  last_name,
  total_points
from Membership
where total_points >= 10000 order by total_points DESC;
```

$$\tau_{\text{total points}} \text{ } \text{DESC } \pi_{\text{first name, last name, total points}}(\sigma_{\text{total points} \geq 10000}(\text{Membership}))$$