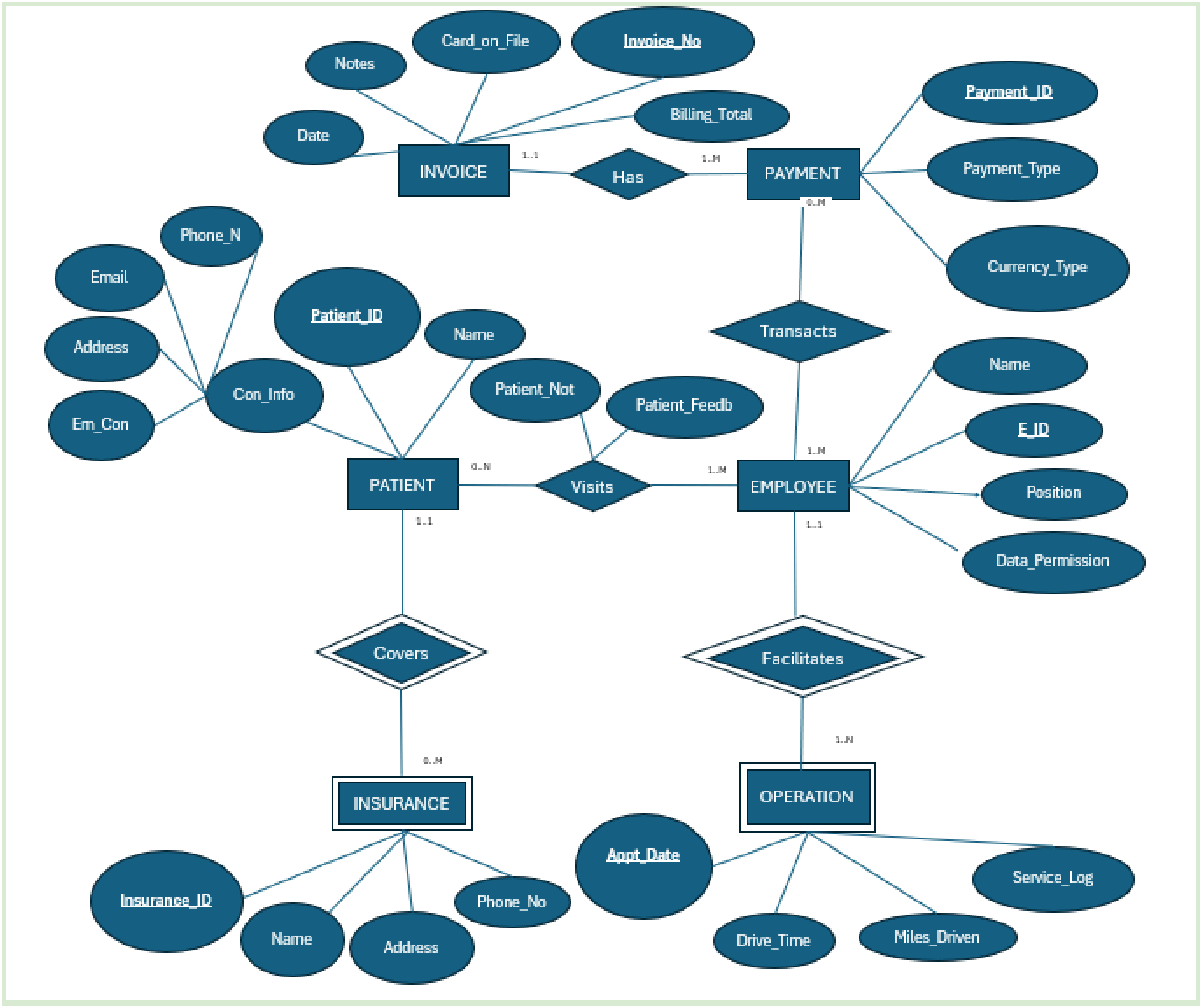
Business Problems

- All behavioral health notes submitted to their respective insurance companies are frozen from payment and sent back to (ABC) due to inefficient note-taking practices
- Their current database is outdated, while also containing repeated data and ambiguous notes
- All behavioral notes must be re-written to adhere to proper standards before receiving payment from insurance companies
- 20% of employees have resigned due to the insurance companies not paying.

Business Rules

- Centralize patient records: A complete billing record should include the following fields before submission: behavioral health notes, medical histories, billing information, driving notes, and insurance information.
- One operation can only be facilitated by one employee at a time.
- Each person receiving Rehabilitative Mental Health (RMH) services must have health insurance documented within the database.
- Patients must be given a patient ID during their first visit at ABC
- Data validation: Behavioral health notes need to include exact drive time.
    - How many minutes does it take the RMH provider to drive from one client to another?
    - The system will check to ensure all necessary information is valid and does not overlap with the provider's notes.

Conceptual design: Updated E-R Diagram



Logical Schema: Updated based on the E-R Diagram

Entities

- Insurance
  - Key Attribute: Insurance_ID
  - Attributes: Name, Address, Phone_No
- Patient
  - Key Attribute: Patient_ID
  - Attributes: Name
  - Composite Attribute - Con_Info: Phone_No, Email, Address, Emergency_Contact
- Employee
  - Key Attribute: Employee_ID
  - Attributes: Name, Position, Data_Permission
- Operation
  - Key Attribute: Appt_Date

    - Attributes: Drive_Time, Miles_Driven, Service_Log
- Payment
    - Key Attribute: Payment_ID
    - Attributes: Payment_Type, Currency Type
- Invoice
    - Key Attribute: Invoice_No
    - Attributes: Date, Notes, Card_On_File, Billing_Total

Cardinalities and relationships

1. Employee (Strong) and Operation (Weak)
    a. Because an Operation Record is dependent on an employee, if an operation record has an employee listed for it that does not exist, we will need to remove the operation
2. Patient (Strong) and Insurance (Weak)
    a. Because insurance in the database is dependent on a patient to be associated with it, insurance listed with no patient should be deleted.

Insurance covers patient - One to Many
- Patient to Insurance (0:M)
- Insurance to Patient (1:1)

Patient Visits Employee - Many to Many
- Patient to Employee (1:M)
- Employee to Patient (0:N)

Employee Facilitates Operation - One to Many
- Employee to Operation (1:N)
- Operation to Employee (1:1)

Employee Transacts Payment - Many to Many
- Employee to payment (0:M)
- Payment to employee (1:M)

Payment Has Invoice - One to Many
- Payment to Invoice (1:1)
- Invoice to payment (1:M)

Relationship Attributes

1. Patient_Visit: Patient_ID, Employee_ID, Patient_Notes, Patient_Feedback
    a. Key Attributes: Patient_ID, Employee_ID

2. Transaction: Payment_ID, Employee_ID
    a. Key Attributes: Payment_ID

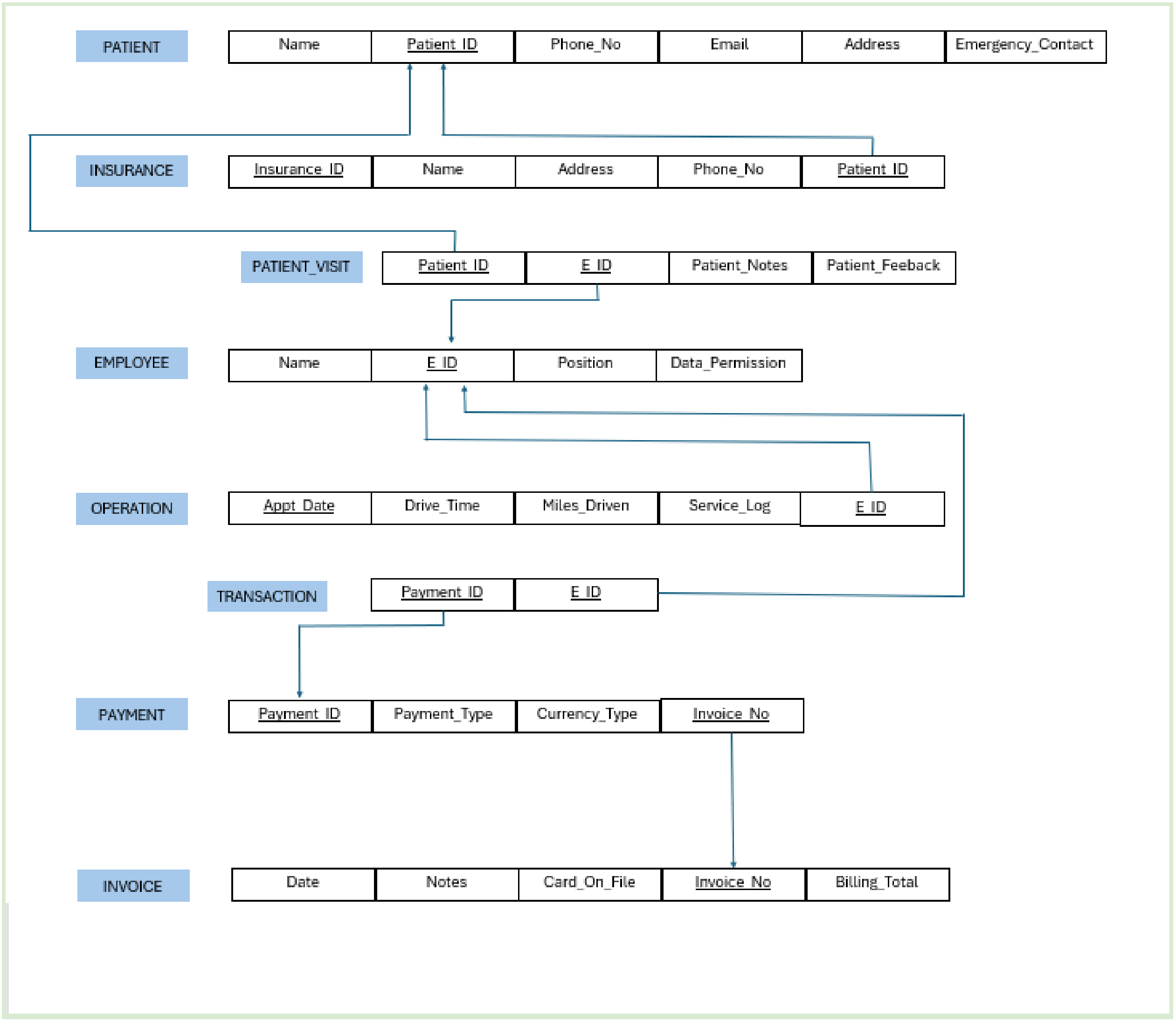


Table structure

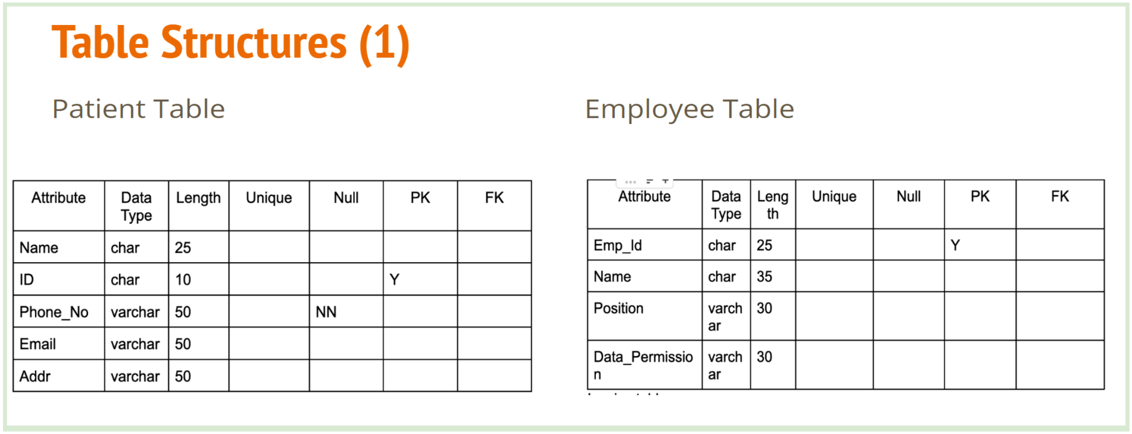


Table Structures (1)

Patient Table

Attribute	Data Type	Length	Unique	Null	PK	FK
Name	char	25				
ID	char	10			Y	
Phone_No	varchar	50		NN		
Email	varchar	50				
Addr	varchar	50				

Employee Table

Attribute	Data Type	Length	Unique	Null	PK	FK
Emp_Id	char	25			Y	
Name	char	35				
Position	varchar	30				
Data_Permission	varchar	30				

Patient Table

```
CREATE TABLE Patient(
NAME char (25),
ID Char (10) ,
Phone_No varchar(50) NOT NULL,
E_Mail varchar(50),
Addr VARCHAR (50),
Emergency_Contact char(25),
CONSTRAINT PK_Patient primary key (ID));
```

Employee Table

```
Create Table EMPLOYEE (
EMP_ID char(25) primary key,
 Name Char (35),
 Position varchar(30),
 Data_Permission varchar(30)
 );
```

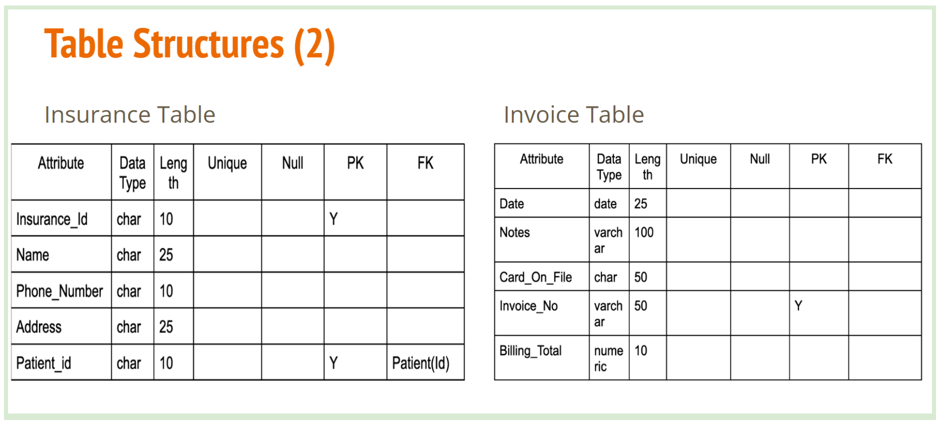


Table Structures (2)

Insurance Table

Attribute	Data Type	Length	Unique	Null	PK	FK
Insurance_Id	char	10			Y	
Name	char	25				
Phone_Number	char	10				
Address	char	25				
Patient_id	char	10			Y	Patient(Id)

Invoice Table

Attribute	Data Type	Length	Unique	Null	PK	FK
Date	date	25				
Notes	varchar	100				
Card_On_File	char	50				
Invoice_No	varchar	50			Y	
Billing_Total	numeric	10				

Insurance Table

```
Create Table INSURANCE (
 INSURANCE_ID char(10),
 Name Char (25),
 PHONE_NUMBER char(10),
 ADDRESS CHAR (25),
 PATIENT_ID char (10),
 CONSTRAINT PK_Insurance primary key (insurance_id,patient_id),
 FOREIGN KEY (patient_id) REFERENCES Patient(Id)
 ON DELETE RESTRICT
 ON UPDATE CASCADE
);
```

Invoice Table

```
CREATE TABLE INVOICE (
   Date CHAR(50),
   NOTES VARCHAR(100),
   CARD_ON_FILE CHAR(50),
   INVOICE_NO VARCHAR(50) PRIMARY KEY,
   BILLING_TOTAL NUMERIC(10)
);
```

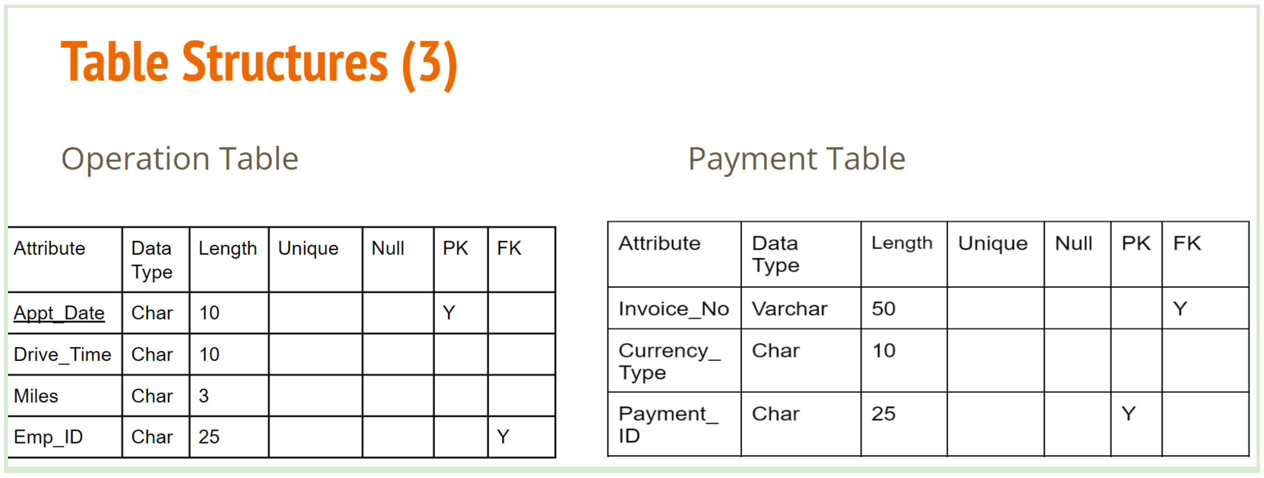


Table Structures (3)

Operation Table

Attribute	Data Type	Length	Unique	Null	PK	FK
Appt_Date	Char	10			Y	
Drive_Time	Char	10				
Miles	Char	3				
Emp_ID	Char	25				Y

Payment Table

Attribute	Data Type	Length	Unique	Null	PK	FK
Invoice_No	Varchar	50				Y
Currency_ Type	Char	10				
Payment_ ID	Char	25			Y	

Operation Table

```
Create Table OPERATION (
APPT_DATE char(10),
DRIVE_TIME char(10),
MILES CHAR (3),
EMP_ID char(25),
SERVICE_LOG CHAR (25),
CONSTRAINT PK_Operation Primary key (Emp_id, Appt_date),
CONSTRAINT FK_Operation_Emp foreign key (Emp_Id) references Employee(Emp_Id)
ON DELETE RESTRICT
ON UPDATE CASCADE
);
```

Payment Table

```
CREATE TABLE PAYMENT (
  Invoice_No varchar(50),
  Currency_Type CHAR(10),
  Payment_ID CHAR(25),
  Payment_type VARCHAR(50),
  CONSTRAINT PK_Payment Primary key (Payment_id, Invoice_No),
  FOREIGN KEY (Invoice_NO) REFERENCES INVOICE(Invoice_NO)
  ON DELETE RESTRICT
  ON UPDATE CASCADE
);
```

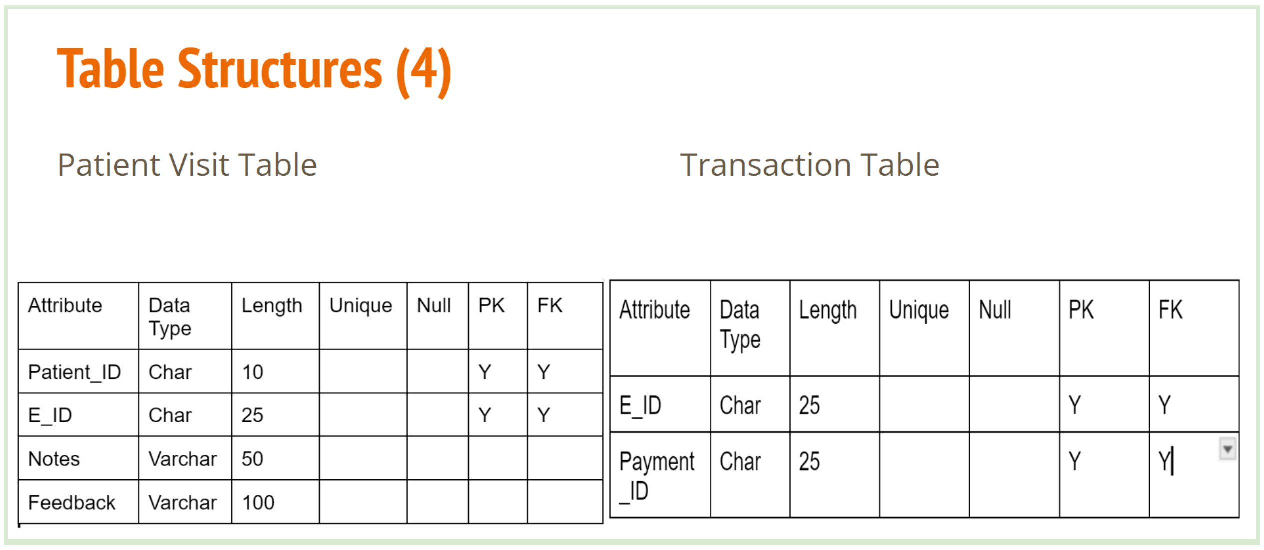


Table Structures (4)

Patient Visit Table

Attribute	Data Type	Length	Unique	Null	PK	FK
Patient_ID	Char	10			Y	Y
E_ID	Char	25			Y	Y
Notes	Varchar	50				
Feedback	Varchar	100				

Transaction Table

Attribute	Data Type	Length	Unique	Null	PK	FK
E_ID	Char	25			Y	Y
Payment _ID	Char	25			Y	Y

Patient Visit

```
CREATE TABLE Patient_Visit (
   Patient_ID char(10),
   E_ID CHAR(25),
   NOTES VARCHAR(50),
   FEEDBACK VARCHAR(100),
   CONSTRAINT PK_Patient_Visit Primary key (E_id, Patient_Id),
   FOREIGN KEY (Patient_ID) REFERENCES Patient(ID),
   FOREIGN KEY (E_ID) REFERENCES EMPLOYEE(EMP_ID)
);
```

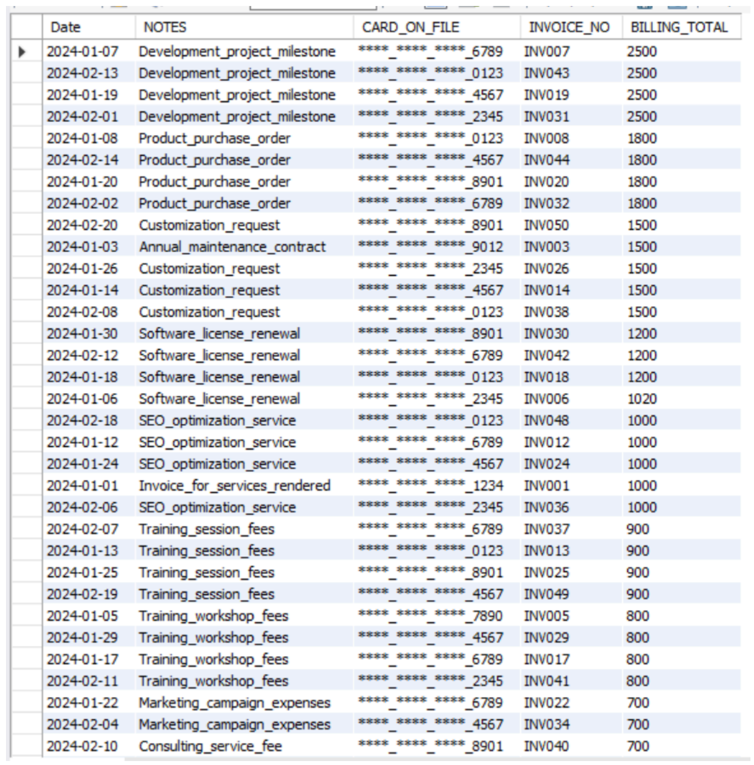
Patient Visit

```
CREATE TABLE Transaction (
   E_ID CHAR(25),
   PAYMENT_ID char(25),
   CONSTRAINT PK_Transaction Primary key (E_Id, Payment_Id),
   FOREIGN KEY (E_ID) REFERENCES Employee (EMP_ID),
         FOREIGN KEY (PAYMENT_ID) REFERENCES
PAYMENT(PAYMENT_ID)
   ON UPDATE CASCADE
   ON DELETE RESTRICT
);
```

Queries and screenshots

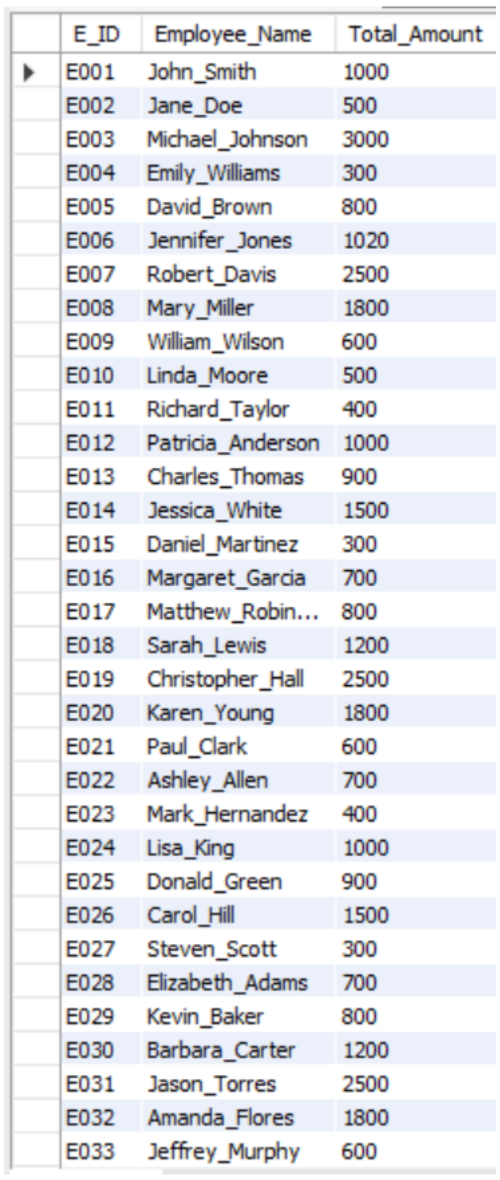
```
-- Query 1: Basic queries and operators
-- Selecting all patients who have an invoice with an amount greater than 500
-- Could be used as a way for staff to send a letter to those owing a large sum of money

SELECT *
FROM INVOICE
WHERE BILLING_TOTAL > 500
ORDER BY Billing_total DESC;
```



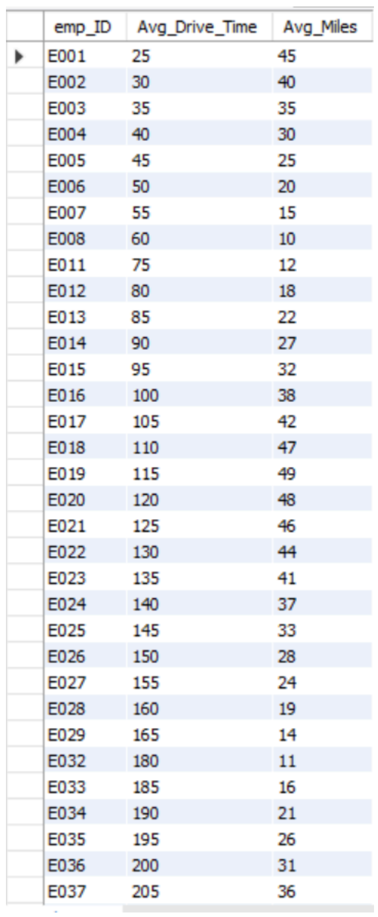
Date	NOTES	CARD_ON_FILE	INVOICE_NO	BILLING_TOTAL
2024-01-07	Development_project_milestone	****_****_****_6789	INV007	2500
2024-02-13	Development_project_milestone	****_****_****_0123	INV043	2500
2024-01-19	Development_project_milestone	****_****_****_4567	INV019	2500
2024-02-01	Development_project_milestone	****_****_****_2345	INV031	2500
2024-01-08	Product_purchase_order	****_****_****_0123	INV008	1800
2024-02-14	Product_purchase_order	****_****_****_4567	INV044	1800
2024-01-20	Product_purchase_order	****_****_****_8901	INV020	1800
2024-02-02	Product_purchase_order	****_****_****_6789	INV032	1800
2024-02-20	Customization_request	****_****_****_8901	INV050	1500
2024-01-03	Annual_maintenance_contract	****_****_****_9012	INV003	1500
2024-01-26	Customization_request	****_****_****_2345	INV026	1500
2024-01-14	Customization_request	****_****_****_4567	INV014	1500
2024-02-08	Customization_request	****_****_****_0123	INV038	1500
2024-01-30	Software_license_renewal	****_****_****_8901	INV030	1200
2024-02-12	Software_license_renewal	****_****_****_6789	INV042	1200
2024-01-18	Software_license_renewal	****_****_****_0123	INV018	1200
2024-01-06	Software_license_renewal	****_****_****_2345	INV006	1020
2024-02-18	SEO_optimization_service	****_****_****_0123	INV048	1000
2024-01-12	SEO_optimization_service	****_****_****_6789	INV012	1000
2024-01-24	SEO_optimization_service	****_****_****_4567	INV024	1000
2024-01-01	Invoice_for_services_rendered	****_****_****_1234	INV001	1000
2024-02-06	SEO_optimization_service	****_****_****_2345	INV036	1000
2024-02-07	Training_session_fees	****_****_****_6789	INV037	900
2024-01-13	Training_session_fees	****_****_****_0123	INV013	900
2024-01-25	Training_session_fees	****_****_****_8901	INV025	900
2024-02-19	Training_session_fees	****_****_****_4567	INV049	900
2024-01-05	Training_workshop_fees	****_****_****_7890	INV005	800
2024-01-29	Training_workshop_fees	****_****_****_4567	INV029	800
2024-01-17	Training_workshop_fees	****_****_****_6789	INV017	800
2024-02-11	Training_workshop_fees	****_****_****_2345	INV041	800
2024-01-22	Marketing_campaign_expenses	****_****_****_6789	INV022	700
2024-02-04	Marketing_campaign_expenses	****_****_****_4567	INV034	700
2024-02-10	Consulting_service_fee	****_****_****_8901	INV040	700

```
-- Query 2: Getting total transaction amounts by employee
-- Useful for transaction monitoring and evaluating employee performance
SELECT t.E_ID, e.Name AS Employee_Name, SUM(i.billing_Total) AS Total_Amount
FROM Transaction t INNER JOIN Payment p ON t.PAYMENT_ID = p.PAYMENT_ID
INNER JOIN Invoice i ON p.Invoice_No = i.Invoice_No
INNER JOIN Employee e ON t.E_ID = e.EMP_ID
GROUP BY t.E_ID, e.Name;
```



E_ID	Employee_Name	Total_Amount
E001	John_Smith	1000
E002	Jane_Doe	500
E003	Michael_Johnson	3000
E004	Emily_Williams	300
E005	David_Brown	800
E006	Jennifer_Jones	1020
E007	Robert_Davis	2500
E008	Mary_Miller	1800
E009	William_Wilson	600
E010	Linda_Moore	500
E011	Richard_Taylor	400
E012	Patricia_Anderson	1000
E013	Charles_Thomas	900
E014	Jessica_White	1500
E015	Daniel_Martinez	300
E016	Margaret_Garcia	700
E017	Matthew_Robin...	800
E018	Sarah_Lewis	1200
E019	Christopher_Hall	2500
E020	Karen_Young	1800
E021	Paul_Clark	600
E022	Ashley_Allen	700
E023	Mark_Hernandez	400
E024	Lisa_King	1000
E025	Donald_Green	900
E026	Carol_Hill	1500
E027	Steven_Scott	300
E028	Elizabeth_Adams	700
E029	Kevin_Baker	800
E030	Barbara_Carter	1200
E031	Jason_Torres	2500
E032	Amanda_Flores	1800
E033	Jeffrey_Murphy	600

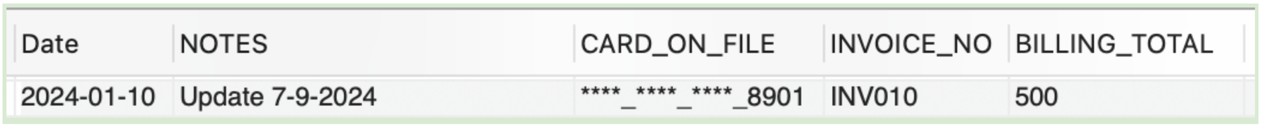
```
-- Query 3: Calculating average drive time and miles per employee
-- Provides operational insights into distances and times incurred by employees, useful for resource allocation
-- Also helps the clinic know how to spread out and allocate resources better
SELECT emp_ID, AVG(DRIVE_TIME) AS Avg_Drive_Time, AVG(MILES) AS Avg_Miles
FROM OPERATION
GROUP BY Emp_ID
ORDER BY Emp_ID;
```



emp_ID	Avg_Drive_Time	Avg_Miles
E001	25	45
E002	30	40
E003	35	35
E004	40	30
E005	45	25
E006	50	20
E007	55	15
E008	60	10
E011	75	12
E012	80	18
E013	85	22
E014	90	27
E015	95	32
E016	100	38
E017	105	42
E018	110	47
E019	115	49
E020	120	48
E021	125	46
E022	130	44
E023	135	41
E024	140	37
E025	145	33
E026	150	28
E027	155	24
E028	160	19
E029	165	14
E032	180	11
E033	185	16
E034	190	21
E035	195	26
E036	200	31
E037	205	36

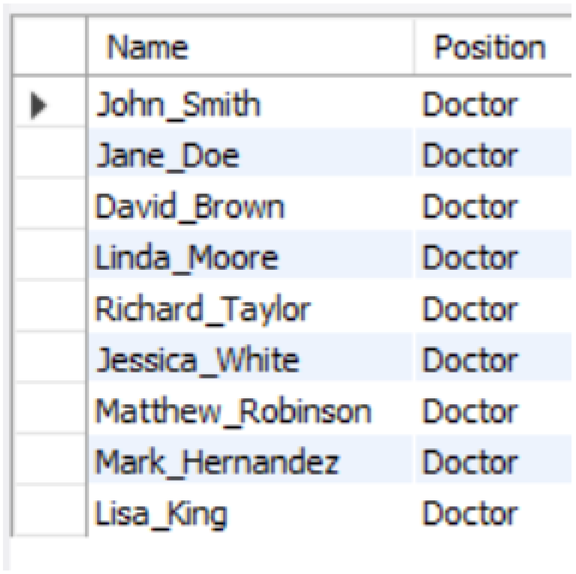
```
-- Query 4: Procedure to update invoices
-- Useful for employees when they have to update invoices
delimiter //
CREATE PROCEDURE UpdateInvoice2 (
   IN p_invoice_no VARCHAR(50),
   IN p_billing_total DECIMAL(10, 0),
   IN p_notes VARCHAR(100)
)
BEGIN
   UPDATE invoice
   SET BILLING_TOTAL = p_billing_total,
      NOTES = p_notes
   WHERE INVOICE_NO = p_invoice_no;
END //

CALL UpdateInvoice2('INV010', 500, 'Update 7-9-2024');
```



Date	NOTES	CARD_ON_FILE	INVOICE_NO	BILLING_TOTAL
2024-01-10	Update 7-9-2024	****_****_****_8901	INV010	500

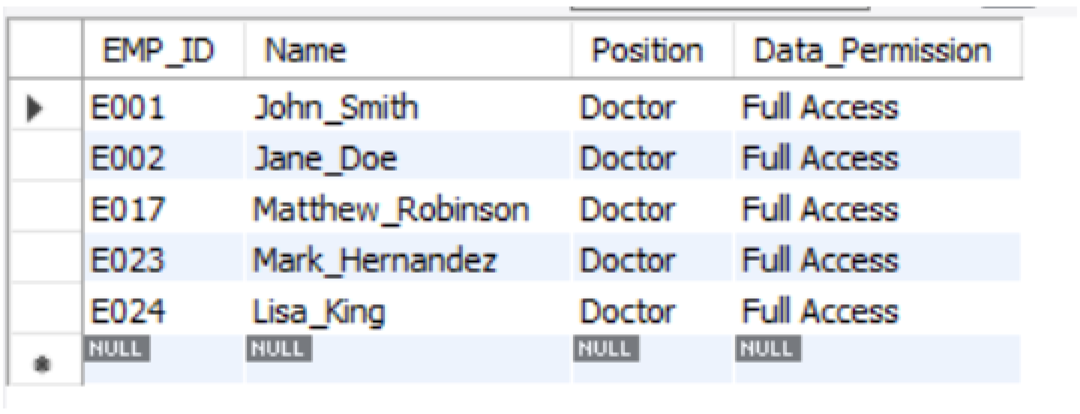
```
-- Query 5: Nested query to find all the doctors at the clinic
-- Useful if a potential patient wants to see the names of all the doctors and do their research on
them to pick one
SELECT Name, Position
FROM Employee
WHERE EMP_ID IN (
   SELECT EMP_ID
   FROM Employee
   WHERE Position = 'Doctor'
);
```



Name	Position
John_Smith	Doctor
Jane_Doe	Doctor
David_Brown	Doctor
Linda_Moore	Doctor
Richard_Taylor	Doctor
Jessica_White	Doctor
Matthew_Robinson	Doctor
Mark_Hernandez	Doctor
Lisa_King	Doctor

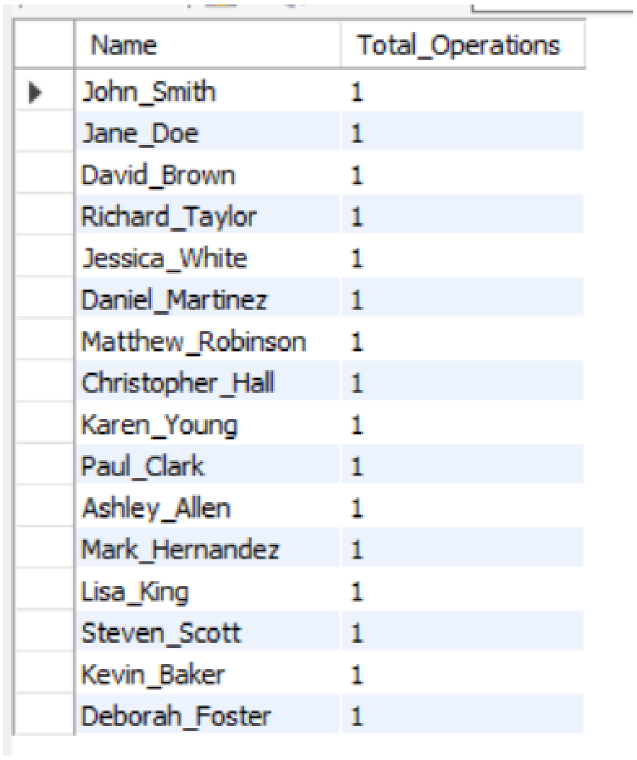
```
-- Query #6: Nested Query to retrieve all employees who are doctors and have performed
operations where the miles traveled is greater than 30:
-- This is useful to keep a record of specific employees such as 'Doctors'
-- Also useful to keep a record of specific miles driven.

SELECT *
FROM EMPLOYEE
WHERE Position = 'Doctor'
AND EMP_ID IN (
   SELECT EMP_ID
   FROM OPERATION
   WHERE CAST(MILES AS UNSIGNED) > 30
);
```



	EMP_ID	Name	Position	Data_Permission
▶	E001	John_Smith	Doctor	Full Access
	E002	Jane_Doe	Doctor	Full Access
	E017	Matthew_Robinson	Doctor	Full Access
	E023	Mark_Hernandez	Doctor	Full Access
	E024	Lisa_King	Doctor	Full Access
●	NULL	NULL	NULL	NULL

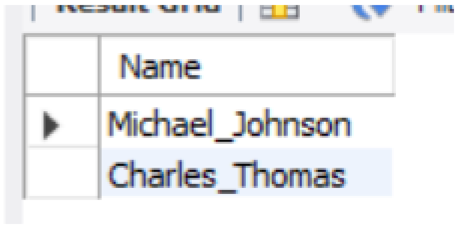
```
-- Query #7: Nested Query to Identify Doctors with full access in their data.
-- This is helpful in the case that full access to data is necessary.
SELECT Name, Total_Operations
FROM (
   SELECT e.Name, COUNT(o.APPT_DATE) AS Total_Operations
   FROM EMPLOYEE e
   JOIN OPERATION o ON e.EMP_ID = o.EMP_ID
   WHERE e.Data_Permission = 'Full Access'
   GROUP BY e.Name
) AS OperationCount
ORDER BY Total_Operations DESC;
```



	Name	Total_Operations
▶	John_Smith	1
	Jane_Doe	1
	David_Brown	1
	Richard_Taylor	1
	Jessica_White	1
	Daniel_Martinez	1
	Matthew_Robinson	1
	Christopher_Hall	1
	Karen_Young	1
	Paul_Clark	1
	Ashley_Allen	1
	Mark_Hernandez	1
	Lisa_King	1
	Steven_Scott	1
	Kevin_Baker	1
	Deborah_Foster	1

-- Query #8 Nested Query to retrieve the names of receptionists who have received patient feedback where the feedback includes the word 'courteous':
-- Helpful when evaluating provider services. One can choose a specific receptionist and request more detailed feedback due to their insight into the information given

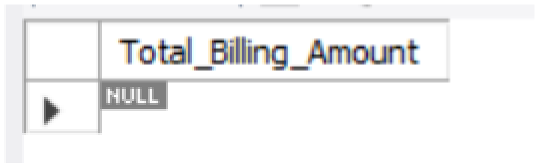
```
SELECT Name
FROM EMPLOYEE
WHERE Position = 'Receptionist'
AND EMP_ID IN (
   SELECT E_ID
   FROM Patient_Visit
   WHERE FEEDBACK LIKE '%courteous%'
);
```



```
Name
Michael_Johnson
Charles_Thomas
```

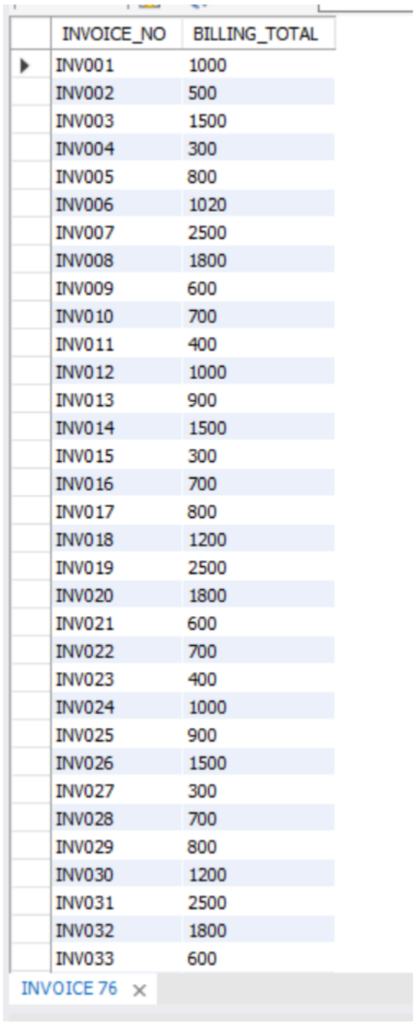
-- Query #9 Nested Query to retrieve the total billing amount for all invoices associated with transactions made by employees (E_ID) whose names start with 'E0':
-- This is helpful in the case that billing information such as amount is needed by specific employees.

```
SELECT SUM(BILLING_TOTAL) AS Total_Billing_Amount
FROM INVOICE
WHERE INVOICE_NO IN (
   SELECT PAYMENT_ID
   FROM Transaction
   WHERE E_ID IN (
      SELECT EMP_ID
      FROM EMPLOYEE
      WHERE Name LIKE '%E0%')
);
```



```
Total_Billing_Amount
NULL
```

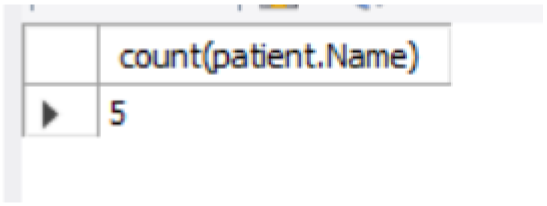
-- Query #10 Nested Query to retrieve the invoice numbers and billing totals for invoices where the payment was made using a credit card stored on file (CARD_ON_FILE):
-- Helpful when questions arise regarding billing. For example, when a client reports that he/she made a payment using a specific card, this can be located on file.
SELECT INVOICE_NO, BILLING_TOTAL
FROM INVOICE
WHERE CARD_ON_FILE IS NOT NULL;



INVOICE_NO	BILLING_TOTAL
INV001	1000
INV002	500
INV003	1500
INV004	300
INV005	800
INV006	1020
INV007	2500
INV008	1800
INV009	600
INV010	700
INV011	400
INV012	1000
INV013	900
INV014	1500
INV015	300
INV016	700
INV017	800
INV018	1200
INV019	2500
INV020	1800
INV021	600
INV022	700
INV023	400
INV024	1000
INV025	900
INV026	1500
INV027	300
INV028	700
INV029	800
INV030	1200
INV031	2500
INV032	1800
INV033	600

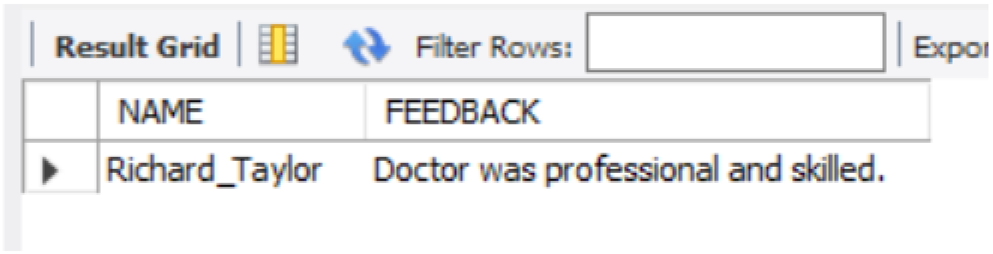
-- Query #11 Nested query to retrieve number of patients who use the insurance of Anthem
-- Useful when we compare different types of insurance.

SELECT count(patient.Name)
FROM PATIENT
INNER JOIN INSURANCE
ON PATIENT.ID = INSURANCE.PATIENT_ID
WHERE INSURANCE.NAME = 'Anthem';



	count(patient.Name)
▶	5

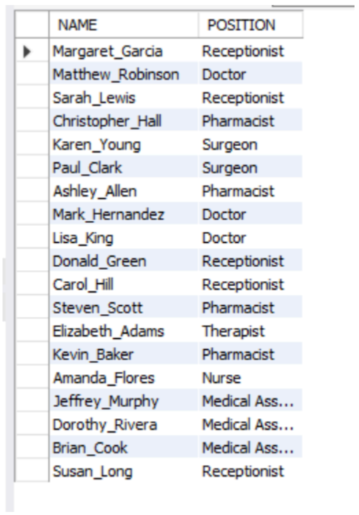
-- Query #12 Retrieve patient's feedback for Dr. Richard Taylor
-- Use for the performance of the employee
SELECT EMPLOYEE.NAME, FEEDBACK
FROM PATIENT_VISIT
INNER JOIN EMPLOYEE
ON PATIENT_VISIT.E_ID = EMPLOYEE.EMP_ID
WHERE EMPLOYEE.NAME = 'Richard_Taylor';



	NAME	FEEDBACK
▶	Richard_Taylor	Doctor was professional and skilled.

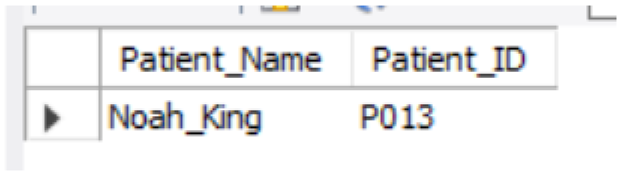
-- Query #13 List the name, position of the employee whose drive time is between 100 minutes and 200 minutes

SELECT NAME, POSITION
FROM EMPLOYEE
INNER JOIN OPERATION
ON OPERATION.EMP_ID = EMPLOYEE.EMP_ID
WHERE DRIVE_TIME BETWEEN 100 AND 200;



	NAME	POSITION
▶	Margaret_Garcia	Receptionist
	Matthew_Robinson	Doctor
	Sarah_Lewis	Receptionist
	Christopher_Hall	Pharmacist
	Karen_Young	Surgeon
	Paul_Clark	Surgeon
	Ashley_Allen	Pharmacist
	Mark_Hernandez	Doctor
	Lisa_King	Doctor
	Donald_Green	Receptionist
	Carol_Hill	Receptionist
	Steven_Scott	Pharmacist
	Elizabeth_Adams	Therapist
	Kevin_Baker	Pharmacist
	Amanda_Flores	Nurse
	Jeffrey_Murphy	Medical Ass...
	Dorothy_Rivera	Medical Ass...
	Brian_Cook	Medical Ass...
	Susan_Long	Receptionist

```
-- Query #14 . Nested query to retrieve patients who do not have insurance.
-- This is helpful when locating clients who are paying full price for services, alleviating ambiguity
in terms of how much their bill will be.
SELECT
p.NAME AS Patient_Name,
p.ID AS Patient_ID
FROM Patient p
WHERE p.ID NOT IN (SELECT PATIENT_ID FROM INSURANCE);
```



Patient_Name	Patient_ID
Noah_King	P013