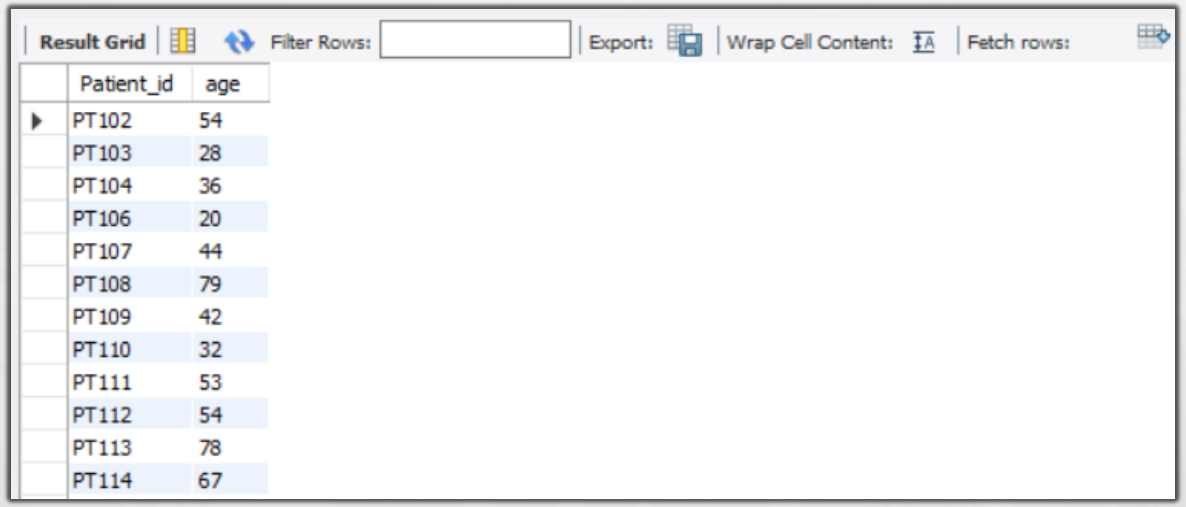

# PSYLIQ – DATA ANALYST INTERNSHIP

## DIABETES PREDICTION ASSESSMENT

Author : Pranav Dharpure

1. **Retrieve the Patient_id and ages of all patients**.

```
SELECT Patient_id, age FROM patient;
```



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

| Patient_id | age |
|---|---|
| PT102 | 54 |
| PT103 | 28 |
| PT104 | 36 |
| PT106 | 20 |
| PT107 | 44 |
| PT108 | 79 |
| PT109 | 42 |
| PT110 | 32 |
| PT111 | 53 |
| PT112 | 54 |
| PT113 | 78 |
| PT114 | 67 |

## 2. Select all female patients who are older than 40.

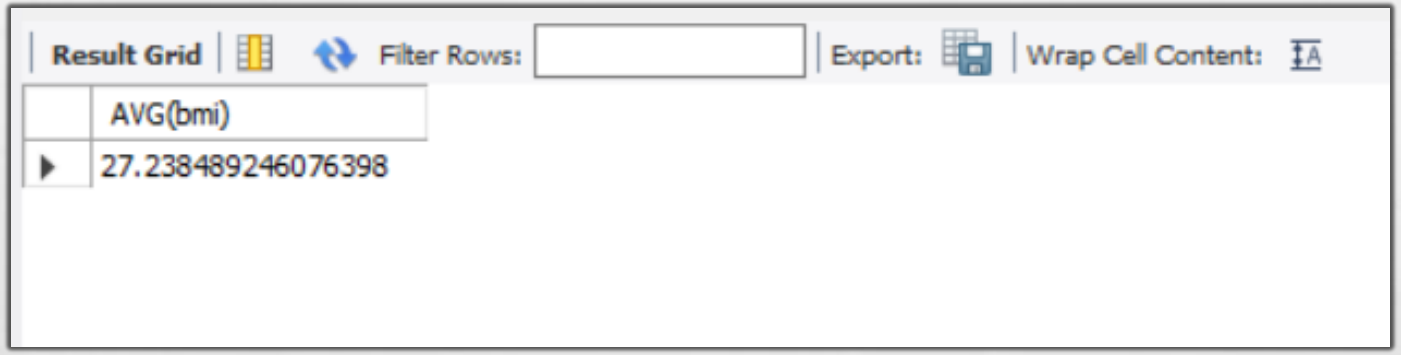
```
SELECT EmployeeName, Patient_id, gender, age
FROM patient
WHERE gender = 'Female' AND age > 40;
```

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

| EmployeeName | Patient_id | gender | age |
|---|---|---|---|
| GARY JIMENEZ | PT102 | Female | 54 |
| ALSON LEE | PT107 | Female | 44 |
| DAVID KUSHNER | PT108 | Female | 79 |
| ARTHUR KENNEY | PT111 | Female | 53 |
| PATRICIA JACKSON | PT112 | Female | 54 |
| EDWARD HARRINGTON | PT113 | Female | 78 |
| JOHN MARTIN | PT114 | Female | 67 |
| DAVID FRANKLIN | PT115 | Female | 76 |
| SEBASTIAN WONG | PT118 | Female | 42 |
| MARTY ROSS | PT119 | Female | 42 |
| GEORGE GARCIA | PT123 | Female | 69 |
| HARLAN KELLY-JR | PT131 | Female | 53 |

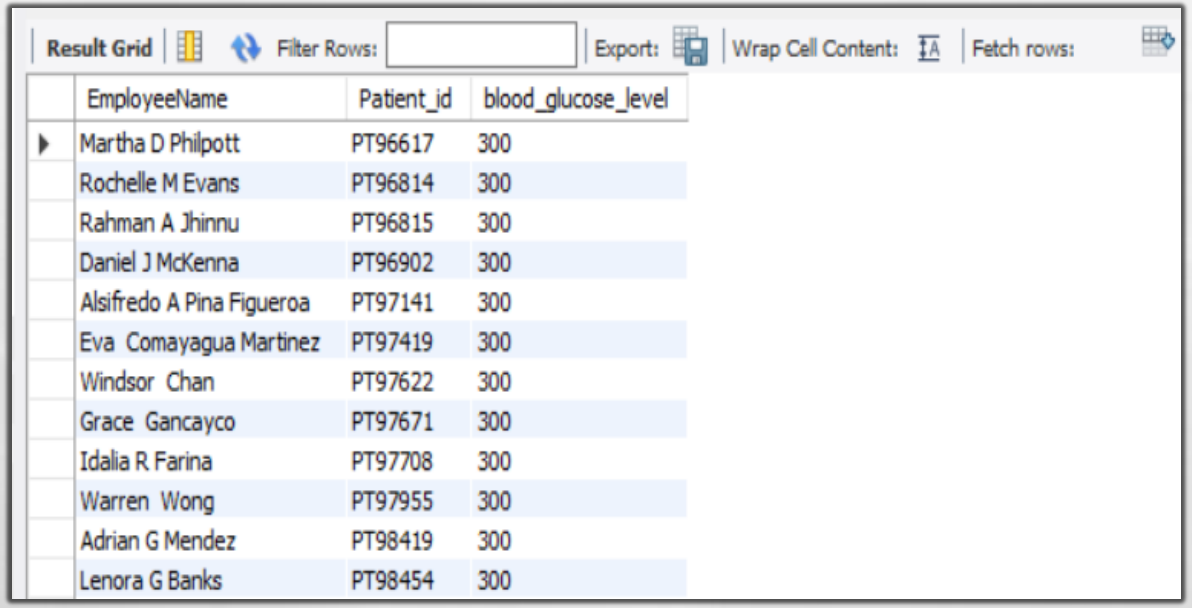
**3. Calculate the average BMI of patients.**

```
SELECT AVG(bmi) FROM patient;
```



## 4. List patients in descending order of blood glucose levels.

```
SELECT EmployeeName, Patient_id, blood_glucose_level
FROM patient
ORDER BY  blood_glucose_level DESC;
```



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

| EmployeeName | Patient_id | blood_glucose_level |
|---|---|---|
| Martha D Philpott | PT96617 | 300 |
| Rochelle M Evans | PT96814 | 300 |
| Rahman A Jhinnu | PT96815 | 300 |
| Daniel J McKenna | PT96902 | 300 |
| Alsifredo A Pina Figueroa | PT97141 | 300 |
| Eva  Comayagua Martinez | PT97419 | 300 |
| Windsor  Chan | PT97622 | 300 |
| Grace  Gancayco | PT97671 | 300 |
| Idalia R Farina | PT97708 | 300 |
| Warren  Wong | PT97955 | 300 |
| Adrian G Mendez | PT98419 | 300 |
| Lenora G Banks | PT98454 | 300 |

## 5. Find patients who have hypertension and diabetes.

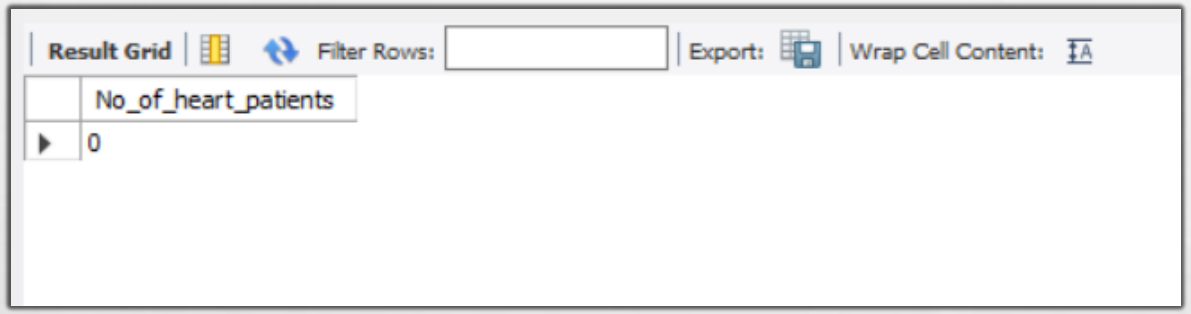
```
SELECT EmployeeName, Patient_id, hypertension, diabetes
FROM patient
WHERE hypertension = 1 AND diabetes = 1;
```

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

| EmployeeName | Patient_id | hypertension | diabetes |
|---|---|---|---|
| JONES WONG | PT139 | 1 | 1 |
| PATRIC STEELE | PT205 | 1 | 1 |
| CHAD LAW | PT355 | 1 | 1 |
| CATHERINE JAMES | PT451 | 1 | 1 |
| JOHN HART | PT565 | 1 | 1 |
| JOHN BARKER | PT567 | 1 | 1 |
| ROBERT BONNET | PT632 | 1 | 1 |
| VITANI BENJAMIN | PT727 | 1 | 1 |
| LANNIE ADELMAN | PT828 | 1 | 1 |
| JOEL DELIZONNA | PT852 | 1 | 1 |
| KAREN KUBICK | PT861 | 1 | 1 |
| ANA GONZALEZ | PT983 | 1 | 1 |

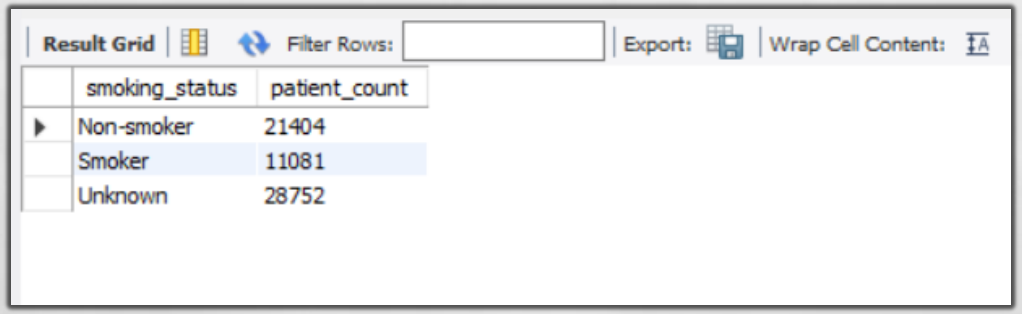
**6. Determine the number of patients with heart disease.**

```
SELECT COUNT(Patient_id) AS No_of_heart_patients
FROM patient
WHERE heart_disease = 1;
```



**7. Group patients by smoking history and count how many smokers and nonsmokers there are.**

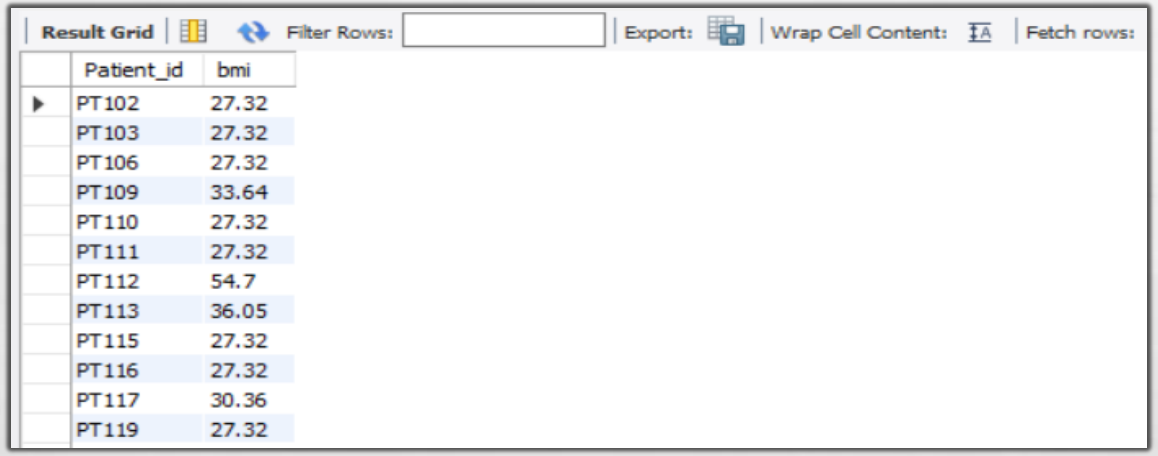
```
SELECT
CASE
WHEN smoking_history IN ('current', 'former', 'ever') THEN 'Smoker'
WHEN smoking_history = 'never' THEN 'Non-smoker'
ELSE 'Unknown'
END AS smoking_status,  COUNT(*) AS patient_count
FROM patient
WHERE smoking_history IN ('never', 'No Info', 'current', 'former', 'ever', 'not current')
GROUP BY smoking_status;
```




| smoking_status | patient_count |
|---|---|
| Non-smoker | 21404 |
| Smoker | 11081 |
| Unknown | 28752 |

**8. Retrieve the Patient_ids of patients who have a BMI greater than the average BMI.**

```
SELECT Patient_id, bmi
FROM patient
WHERE bmi > (SELECT AVG(bmi) FROM patient);
```

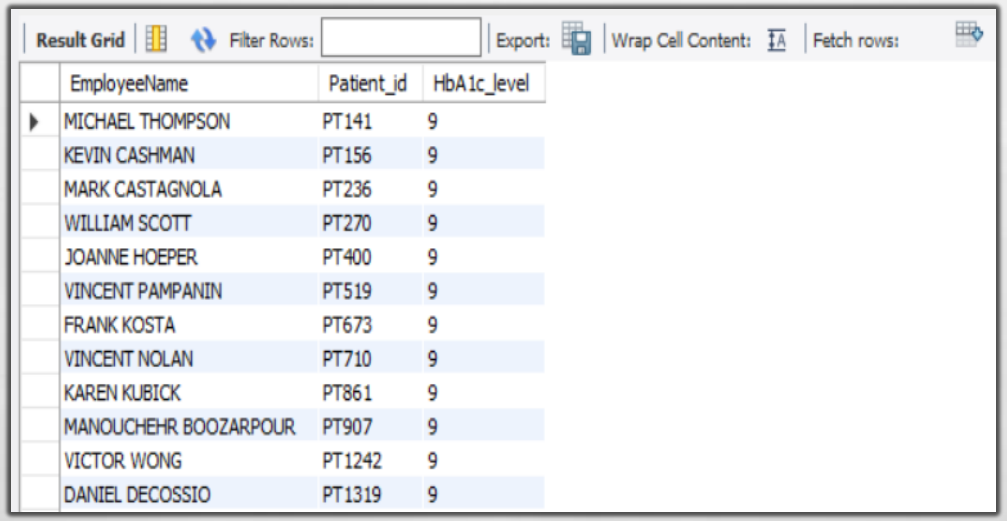


Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| Patient_id | bmi |
|---|---|
| PT102 | 27.32 |
| PT103 | 27.32 |
| PT106 | 27.32 |
| PT109 | 33.64 |
| PT110 | 27.32 |
| PT111 | 27.32 |
| PT112 | 54.7 |
| PT113 | 36.05 |
| PT115 | 27.32 |
| PT116 | 27.32 |
| PT117 | 30.36 |
| PT119 | 27.32 |

**9. Find the patient with the highest HbA1c level and the patient with the lowest HbA1clevel.**

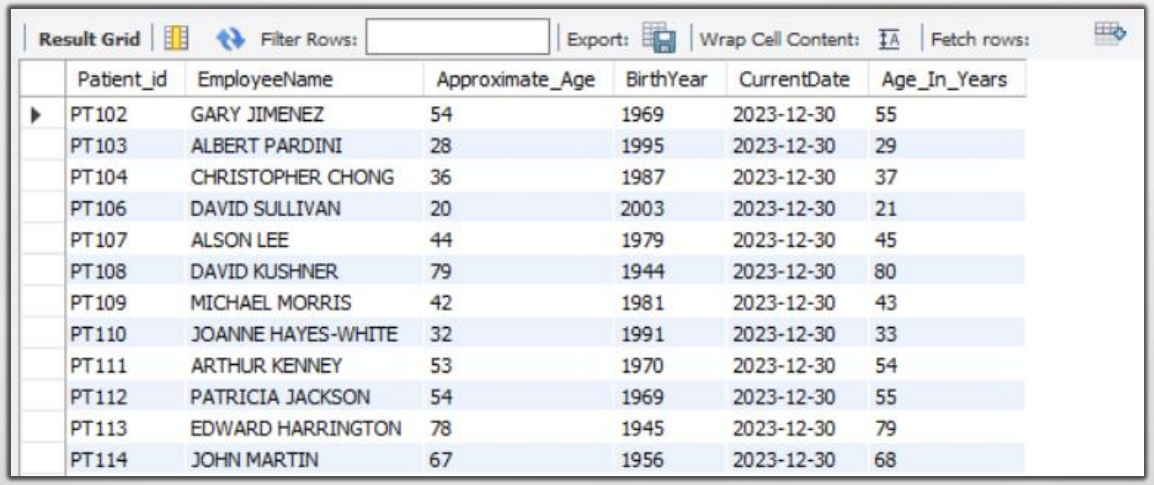
```
SELECT EmployeeName, Patient_id, HbA1c_level
FROM patient
WHERE HbA1c_level = (SELECT MAX(HbA1c_level) FROM patient)
UNION ALL
SELECT EmployeeName, Patient_id, HbA1c_level
FROM patient
WHERE HbA1c_level = (SELECT MIN(HbA1c_level) FROM patient);
```




| EmployeeName | Patient_id | HbA1c_level |
|---|---|---|
| MICHAEL THOMPSON | PT141 | 9 |
| KEVIN CASHMAN | PT156 | 9 |
| MARK CASTAGNOLA | PT236 | 9 |
| WILLIAM SCOTT | PT270 | 9 |
| JOANNE HOEPER | PT400 | 9 |
| VINCENT PAMPANIN | PT519 | 9 |
| FRANK KOSTA | PT673 | 9 |
| VINCENT NOLAN | PT710 | 9 |
| KAREN KUBICK | PT861 | 9 |
| MANOUCHEHR BOOZARPOUR | PT907 | 9 |
| VICTOR WONG | PT1242 | 9 |
| DANIEL DECOSSIO | PT1319 | 9 |

**10. Calculate the age of patients in years (assuming the current date as of now).**

```
SELECT  Patient_id,  EmployeeName,  Age AS Approximate_Age,
FLOOR(YEAR(CURDATE()) - Age) AS BirthYear,
CURDATE() AS CurrentDate,
FLOOR(DATEDIFF(CURDATE(),
STR_TO_DATE(CONCAT(YEAR(CURDATE()) - Age, '-01-01'), '%Y-%m-%d')) / 365) AS Age_In_Years
FROM patient;
```



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

| Patient_id | EmployeeName | Approximate_Age | BirthYear | CurrentDate | Age_In_Years |
|---|---|---|---|---|---|
| PT102 | GARY JIMENEZ | 54 | 1969 | 2023-12-30 | 55 |
| PT103 | ALBERT PARDINI | 28 | 1995 | 2023-12-30 | 29 |
| PT104 | CHRISTOPHER CHONG | 36 | 1987 | 2023-12-30 | 37 |
| PT106 | DAVID SULLIVAN | 20 | 2003 | 2023-12-30 | 21 |
| PT107 | ALSON LEE | 44 | 1979 | 2023-12-30 | 45 |
| PT108 | DAVID KUSHNER | 79 | 1944 | 2023-12-30 | 80 |
| PT109 | MICHAEL MORRIS | 42 | 1981 | 2023-12-30 | 43 |
| PT110 | JOANNE HAYES-WHITE | 32 | 1991 | 2023-12-30 | 33 |
| PT111 | ARTHUR KENNEY | 53 | 1970 | 2023-12-30 | 54 |
| PT112 | PATRICIA JACKSON | 54 | 1969 | 2023-12-30 | 55 |
| PT113 | EDWARD HARRINGTON | 78 | 1945 | 2023-12-30 | 79 |
| PT114 | JOHN MARTIN | 67 | 1956 | 2023-12-30 | 68 |

**11. Rank patients by blood glucose level within each gender group**.

```
SELECT  Patient_id,  EmployeeName,  gender,  blood_glucose_level,
RANK() OVER (PARTITION BY Gender ORDER BY blood_glucose_level) AS
Glucose_Rank_Within_Gender
FROM patient;
```

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

| Patient_id | EmployeeName | gender | blood_glucose_level | Glucose_Rank_Within_Gender |
|---|---|---|---|---|
| PT98546 | Adan J Subia | Female | 80 | 1 |
| PT99450 | Simon K Ng | Female | 80 | 1 |
| PT99113 | David J Thompson | Female | 80 | 1 |
| PT99454 | Thomas M Gragasin Jr | Female | 80 | 1 |
| PT98564 | Yue L Yu | Female | 80 | 1 |
| PT99464 | Jeanpierre V Concepcion | Female | 80 | 1 |
| PT99702 | Trina R Pellette | Female | 80 | 1 |
| PT98754 | Gerome C Segarra | Female | 80 | 1 |
| PT99757 | Vandora  Sione | Female | 80 | 1 |
| PT99682 | Gerardina D Cruz | Female | 80 | 1 |
| PT99894 | Jacqueline C Roberts | Female | 80 | 1 |
| PT100015 | Joshua R Mcdonald | Female | 80 | 1 |

**12. Update the smoking history of patients who are older than 50 to "Ex-smoker.“**

```
UPDATE patient
SET smoking_history = 'Ex-smoker’
WHERE age > 50;
```

**13. Insert a new patient into the database with sample data.**

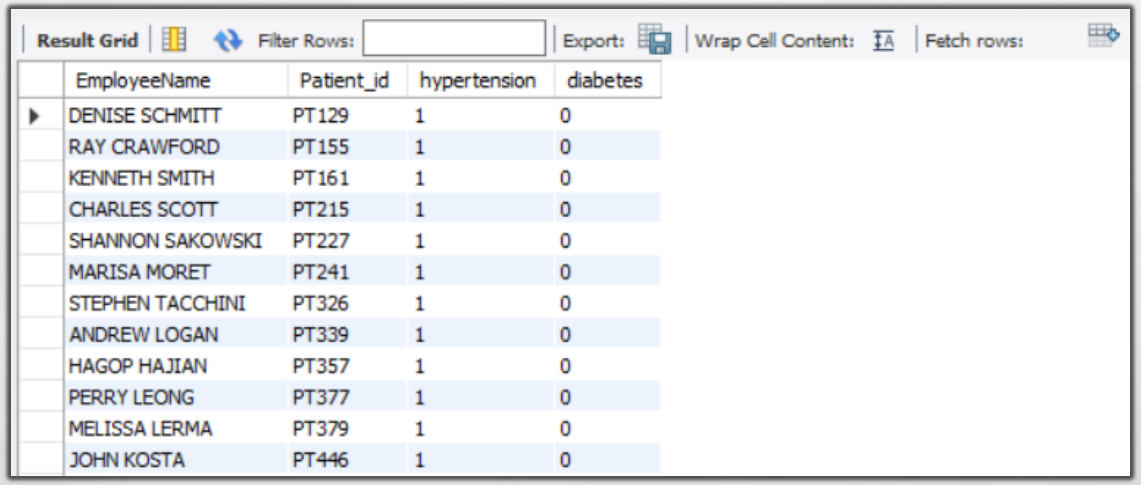
```
 INSERT INTO patient
(Patient_id, EmployeeName, gender, age, smoking_history, blood_glucose_level)
VALUES  (1, 'John Doe', 'Male', 35, 'Non-smoker', 120),
(2, 'Jane Smith', 'Female', 45, 'Current smoker', 140),
-- Add more sample data as needed
(3, 'Sam Johnson', 'Male', 55, 'Former smoker', 130);
```

**14. Delete all patients with heart disease from the database**.

```
DELETE FROM patientWHERE heart_disease = 1;
```

**15. Find patients who have hypertension but not diabetes using the EXCEPT operator.**

```
SELECT EmployeeName, Patient_id, hypertension, diabetes
FROM patient
WHERE Hypertension = 1
AND NOT EXISTS (
SELECT 1  FROM patient AS p2  WHERE p2.Patient_id = patient.Patient_id AND p2.diabetes = 1);
```



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

| EmployeeName | Patient_id | hypertension | diabetes |
|---|---|---|---|
| DENISE SCHMITT | PT129 | 1 | 0 |
| RAY CRAWFORD | PT155 | 1 | 0 |
| KENNETH SMITH | PT161 | 1 | 0 |
| CHARLES SCOTT | PT215 | 1 | 0 |
| SHANNON SAKOWSKI | PT227 | 1 | 0 |
| MARISA MORET | PT241 | 1 | 0 |
| STEPHEN TACCHINI | PT326 | 1 | 0 |
| ANDREW LOGAN | PT339 | 1 | 0 |
| HAGOP HAJIAN | PT357 | 1 | 0 |
| PERRY LEONG | PT377 | 1 | 0 |
| MELISSA LERMA | PT379 | 1 | 0 |
| JOHN KOSTA | PT446 | 1 | 0 |

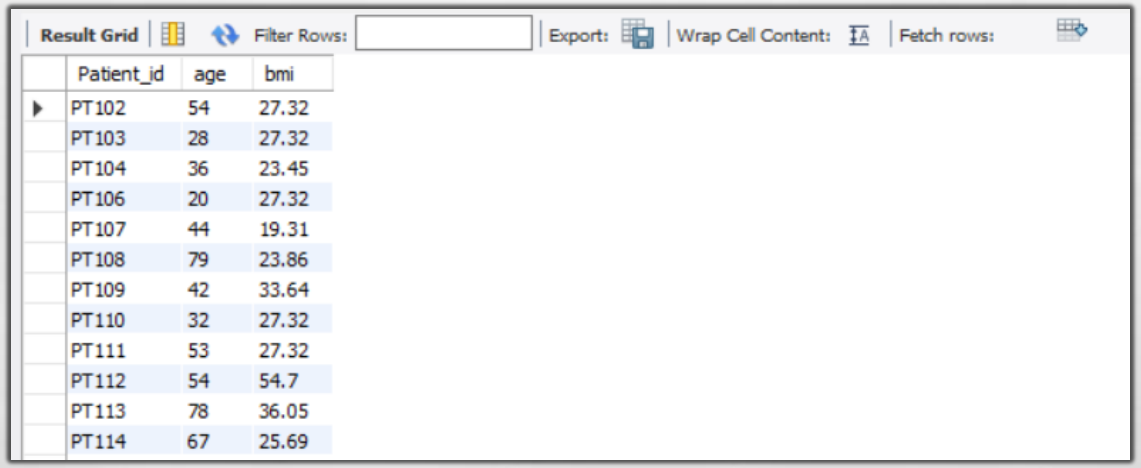
**16. Define a unique constraint on the "patient_id" column to ensure its values are unique.**

```
ALTER TABLE patient
ADD CONSTRAINT unique_Patient_id UNIQUE (Patient_id);
```

**17. Create a view that displays the Patient_ids, ages, and BMI of patients.**

```
CREATE VIEW patient_info_view AS
SELECT  Patient_id,  age,  bmi
FROM  patient;

SELECT * FROM patient_info_view;
```




| Patient_id | age | bmi |
|---|---|---|
| PT102 | 54 | 27.32 |
| PT103 | 28 | 27.32 |
| PT104 | 36 | 23.45 |
| PT106 | 20 | 27.32 |
| PT107 | 44 | 19.31 |
| PT108 | 79 | 23.86 |
| PT109 | 42 | 33.64 |
| PT110 | 32 | 27.32 |
| PT111 | 53 | 27.32 |
| PT112 | 54 | 54.7 |
| PT113 | 78 | 36.05 |
| PT114 | 67 | 25.69 |

**18. Suggest improvements in the database schema to reduce data redundancy and improve data integrity.**

To reduce data redundancy and improve data integrity in a database schema, you can consider the following improvements:

**1. Normalization:** Apply normalization techniques to eliminate redundancy and dependency issues. Break down large tables into smaller, related tables to store data more efficiently.

**2. Use Primary and Foreign Keys:** Ensure that each table has a primary key to uniquely identify records. Use foreign keys to establish relationships between tables, enforcing referential integrity.

**3. Avoid Storing Calculated Data:** Avoid storing derived or calculated values in the database. Instead, calculate them on-the-fly when needed. This helps prevent data inconsistency and redundancy.

**4. Use Enumerations or Lookup Tables:** Replace repeating values with enumeration types or lookup tables. For example, create a table for gender, and reference it in the main table using foreign keys, instead of storing gender strings in each record.

**5. Default Values and Constraints:** Define default values for columns where appropriate. Use constraints such as NOT NULL, UNIQUE, and CHECK constraints to enforce data integrity rules.

**8. Avoid Redundant Columns:** Avoid having redundant columns that store the same information. If information can be derived from other columns, consider calculating it dynamically or creating a separate table.

**7. Review Data Types:** Choose appropriate data types for columns to minimize storage space. Use integer types for numerical values, and choose string types based on the expected length of the data.

**8. Document and Enforce Data Standards:** Establish and document data standards for naming conventions, capitalization, and formatting. Enforce these standards consistently across the database schema.

**9. Audit Trails:** Implement audit trails or change tracking to keep a record of modifications to the data. This enhances accountability and makes it easier to trace changes over time.

**10. Indexing:** Use indexes wisely to improve query performance. Index columns used frequently in search conditions, but avoid excessive indexing, as it can impact write performance.

**11. Partitioning:** Consider partitioning large tables to improve manageability and query performance. Partitioning can be based on a range of values, such as dates or numeric ranges.

**12. Regular Maintenance:** Perform regular maintenance tasks, such as database backups, integrity checks, and optimization to ensure the ongoing health and performance of the database.

## 19. Explain how you can optimize the performance of SQL queries on this dataset.

Optimizing the performance of SQL queries on a dataset involves various strategies to enhance efficiency and reduce execution times. Here are some general tips to optimize SQL queries:

**1. Use Indexing:** Create indexes on columns frequently used in WHERE clauses or JOIN conditions. Indexing speeds up data retrieval but be mindful of the trade-off with write operations.

```
CREATE INDEX idx_column_name ON table_name (column_name);
```

**2. Optimize JOIN Operations:** Use INNER JOINs instead of OUTER JOINs when possible. Ensure that columns involved in JOIN conditions are indexed.

```
SELECT * FROM table1 INNER JOIN table2 ON table1.id = table2.table1_id;
```

***3. Avoid SELECT *:*** Instead of selecting all columns using **SELECT ***, explicitly specify only the columns needed. This reduces the amount of data transferred and can improve query performance.

```
SELECT column1, column2 FROM table_name WHERE condition;
```

**4. Use WHERE Clause Efficiently:** Filter rows as early as possible using the WHERE clause. This reduces the number of rows processed and speeds up query execution.

```
SELECT * FROM table_name WHERE column_name = 'value’;
```

**5. Optimize Subqueries:** Where possible, replace subqueries with JOINs. Subqueries can be less efficient, especially if they are executed for each row.

-- Subquery SELECT * FROM table1 WHERE column_name IN (SELECT column_name FROM table2 WHERE condition); -- JOIN SELECT table1.* FROM table1 JOIN table2 ON table1.column_name = table2.column_name AND table2.condition;

**6. Limit and Offset Results:** Use **LIMIT** and **OFFSET** to limit the number of returned rows. This is particularly important when dealing with large datasets.

SELECT * FROM table_name LIMIT 10 OFFSET 20;

**7. Avoid Using DISTINCT Unnecessarily:** Use **DISTINCT** only when necessary. It can be resource-intensive, especially on large datasets.

SELECT DISTINCT column_name FROM table_name;

**8. Consider Denormalization:** In some cases, denormalizing tables by duplicating certain data can improve query performance. This is a trade-off and depends on specific use cases.

**9. Monitor and Analyze Execution Plans:** Use tools to analyze query execution plans to identify bottlenecks. Adjust queries and indexes based on the analysis.

SELECT column1, column2 FROM table_name WHERE condition;

**10. Regular Database Maintenance:** Schedule regular maintenance tasks, such as index rebuilds and statistics updates, to keep the database in optimal condition.

-- Rebuild indexes ALTER INDEX ALL ON table_name REBUILD;