DIABETES PREDICTION -SQL

HIRAN JOSEPH

PSYLIQ

DIABETES PREDICTION OVERVIEW

Number of Records:

The dataset consists of 100,001 records, providing a substantial volume of data for analysis.

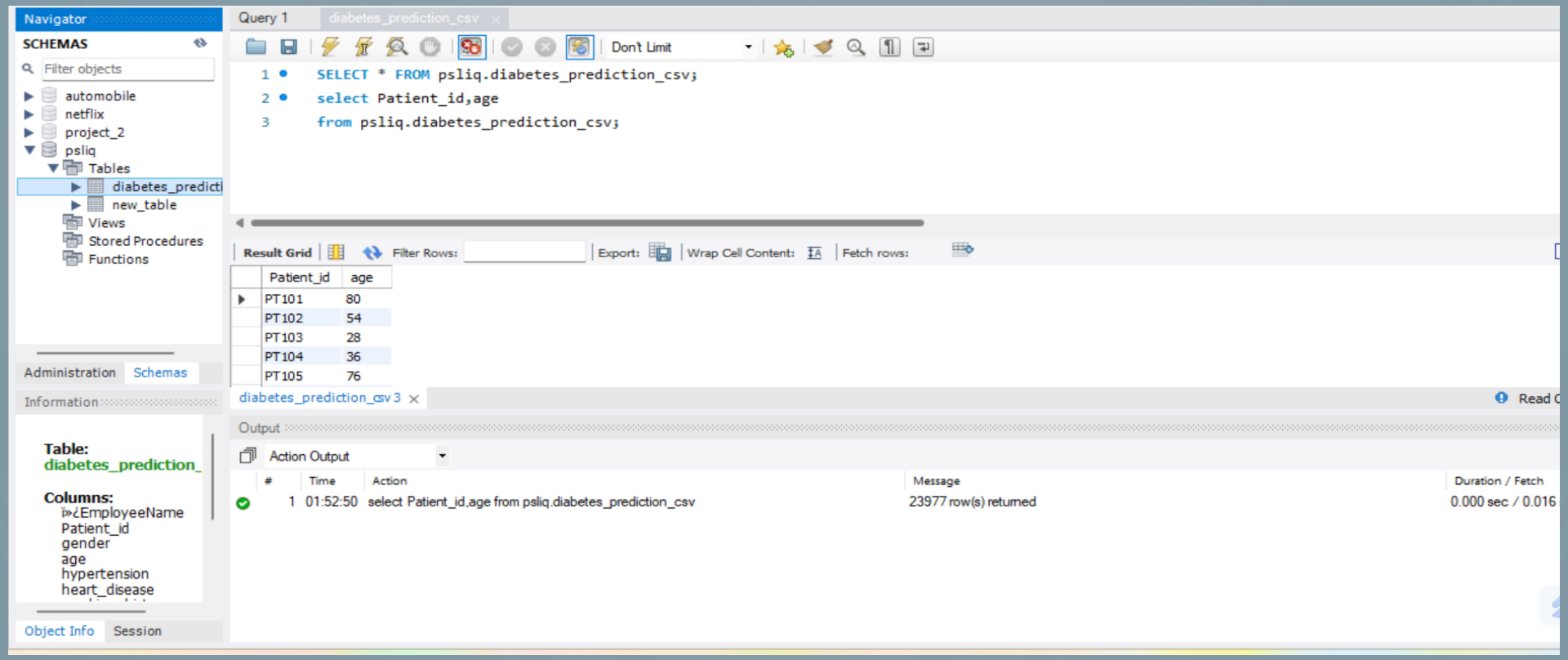
Main Attributes:

The primary attributes include EmployeeName, Patient_id, gender, age, hypertension, heart_disease, smoking_history, bmi, HbA1c_level, blood_glucose_level, and diabetes..

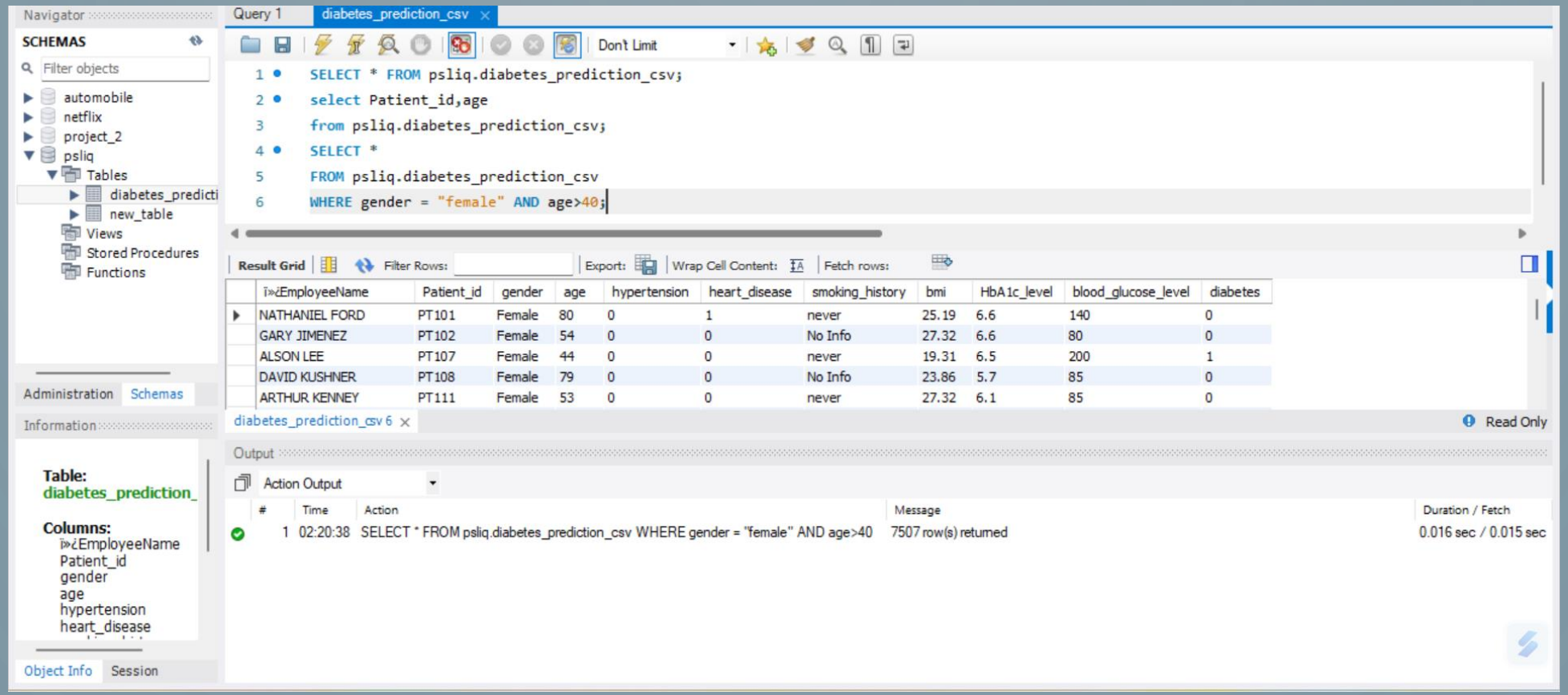
Significance of Attributes:

Each attribute plays a crucial role in understanding diabetes-related factors, offering insights into patient demographics, health conditions, and lifestyle choices.

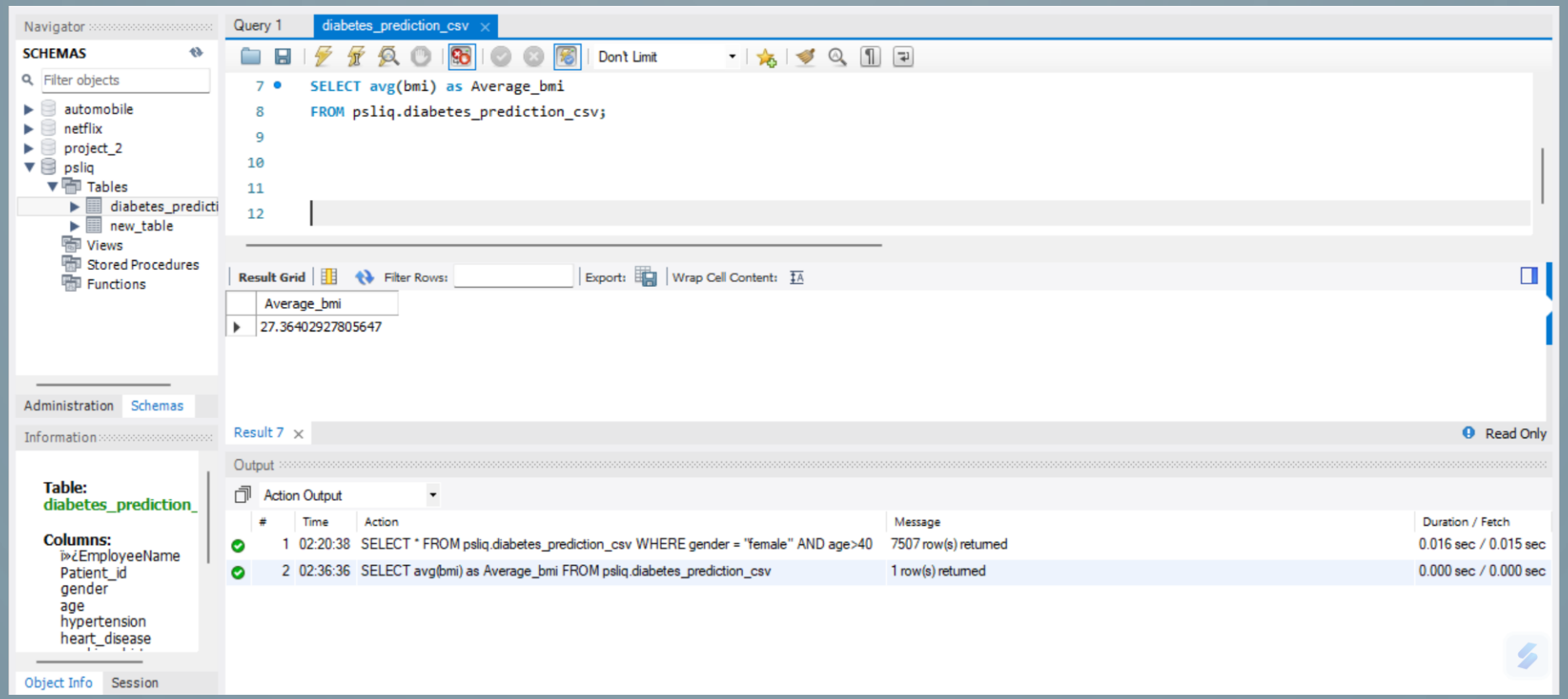
1. RETRIEVE THE PATIENT_ID AND AGES OF ALL PATIENTS.



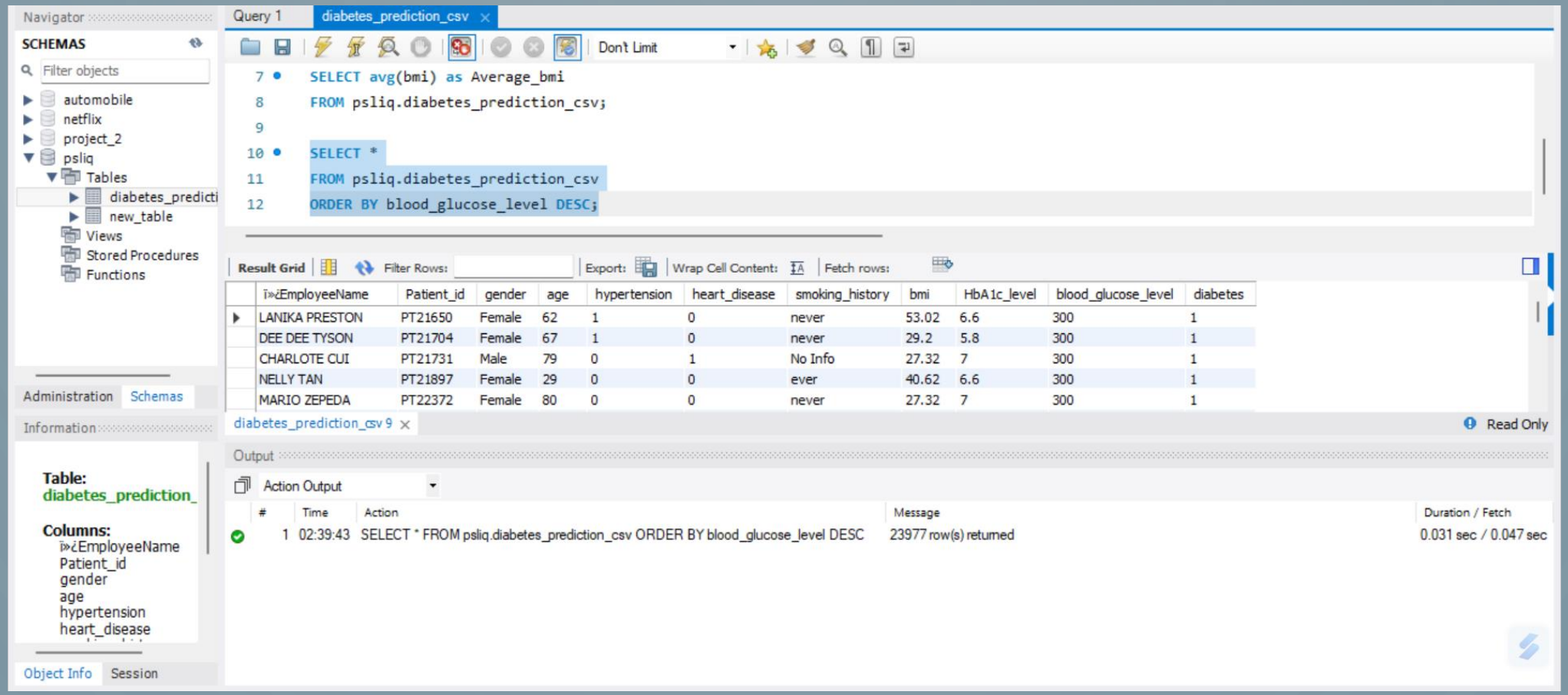
2. SELECT ALL FEMALE PATIENTS WHO ARE OLDER THAN 40.



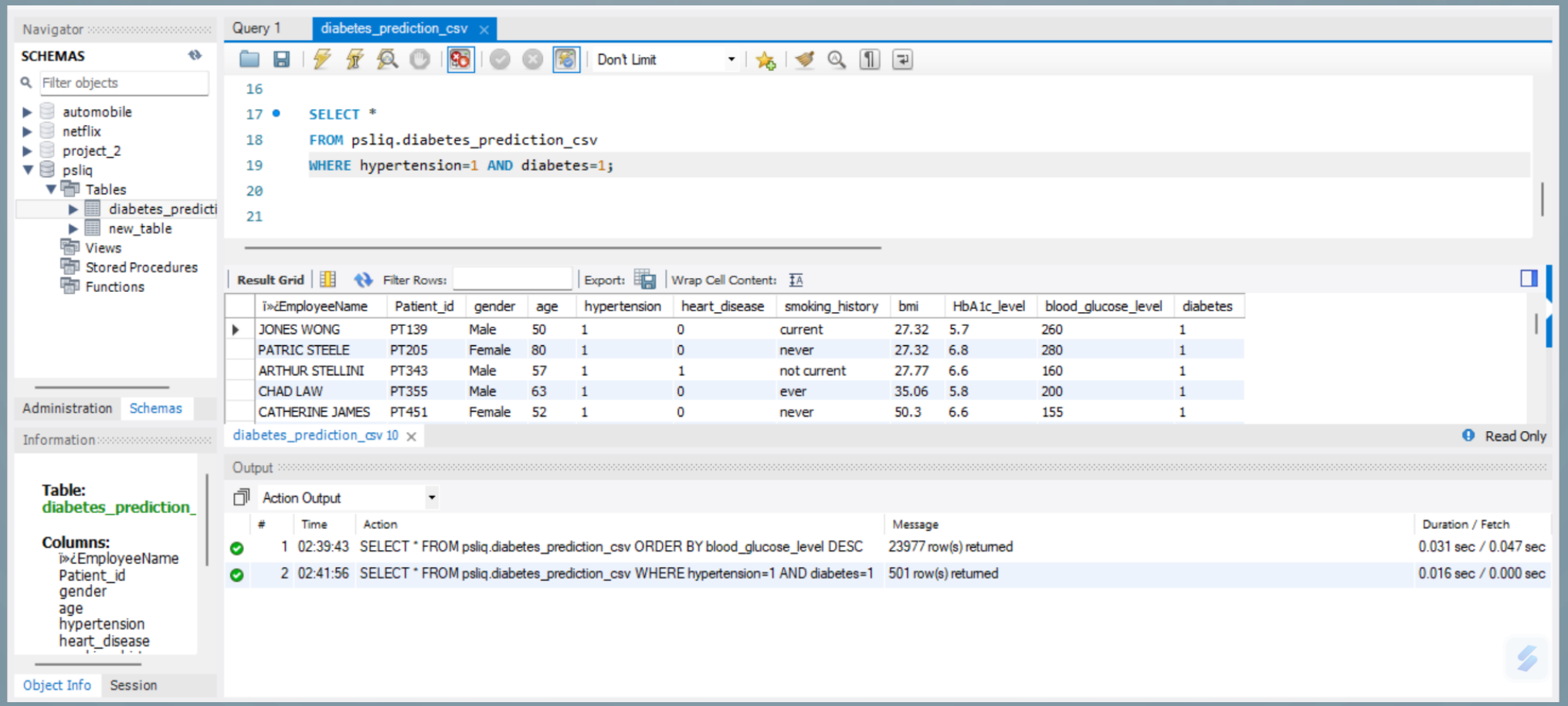
3. CALCULATE THE AVERAGE BMI OF PATIENTS.



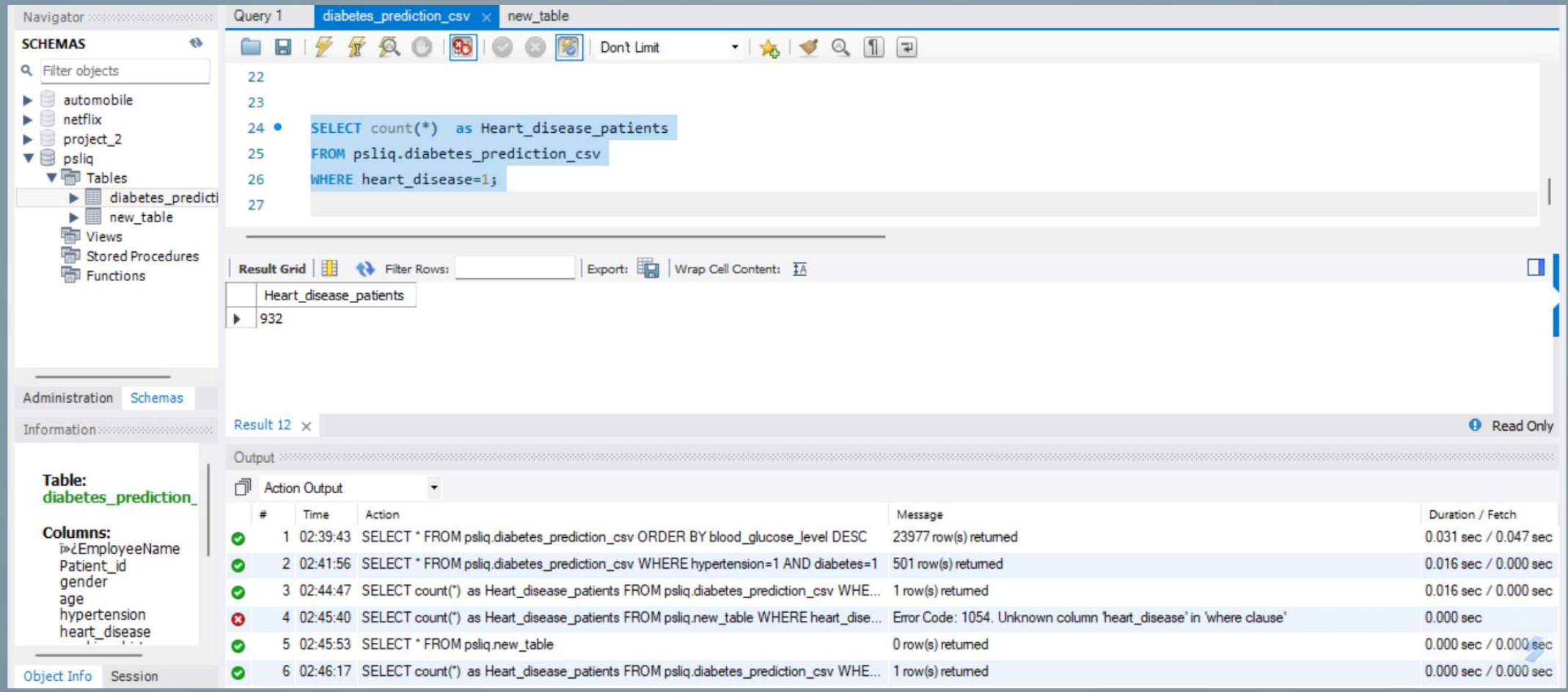
4. LIST PATIENTS IN DESCENDING ORDER OF BLOOD GLUCOSE LEVELS.



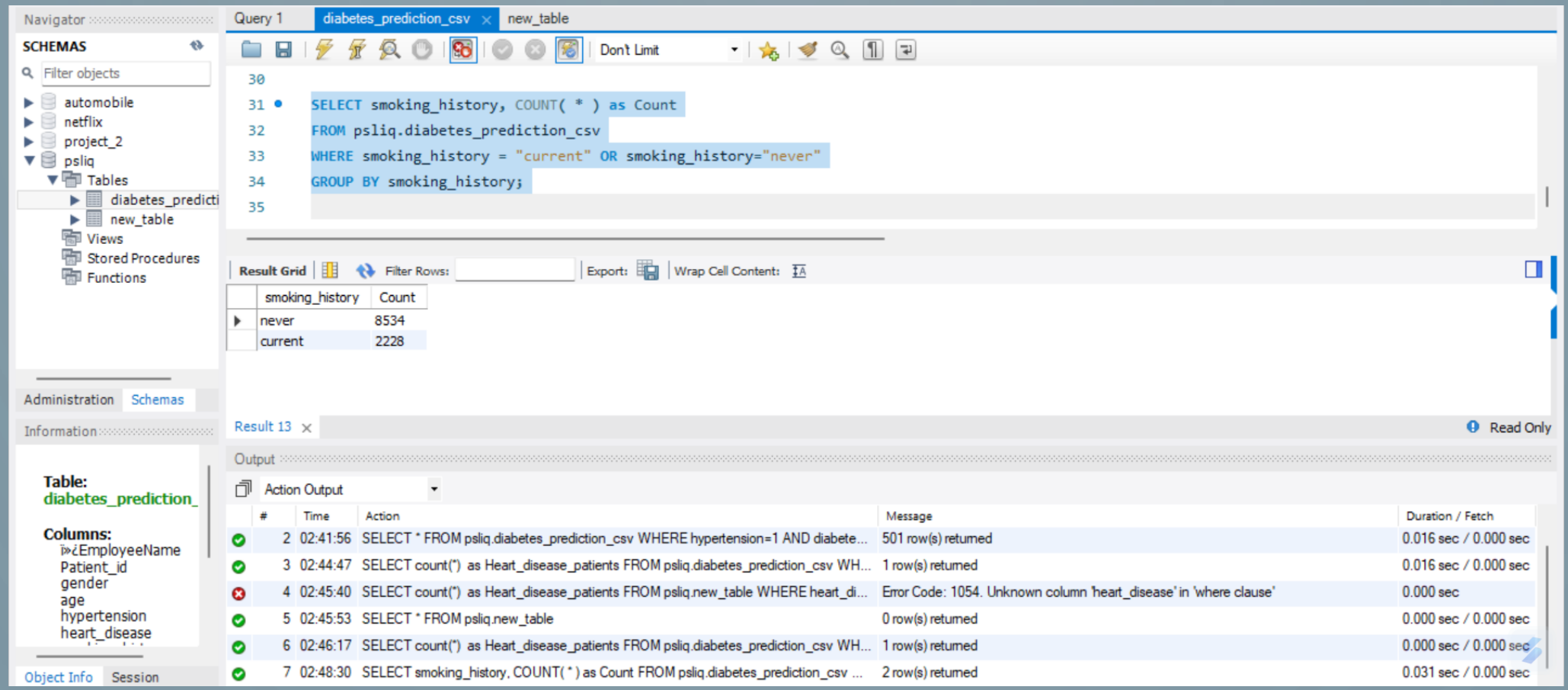
5. FIND PATIENTS WHO HAVE HYPERTENSION AND DIABETES.



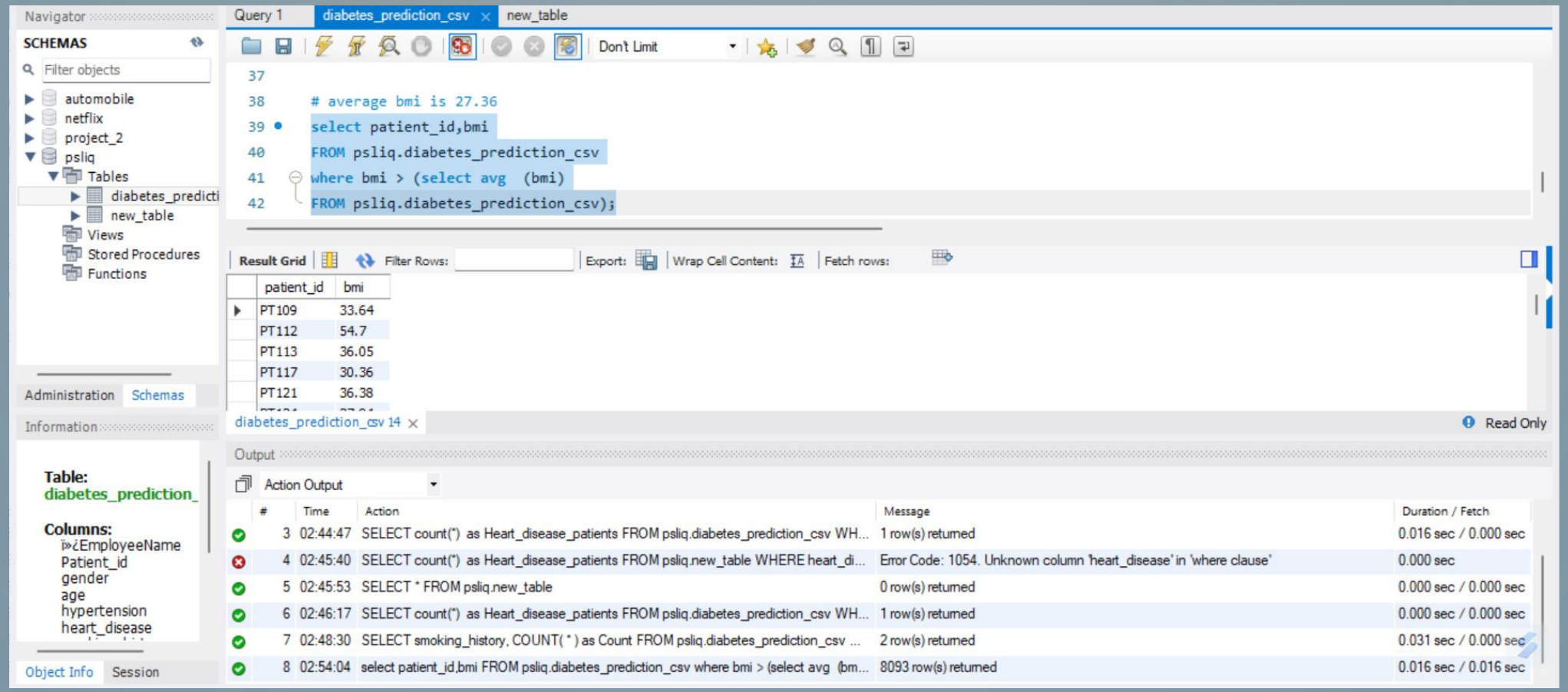
6. DETERMINE THE NUMBER OF PATIENTS WITH HEART DISEASE.



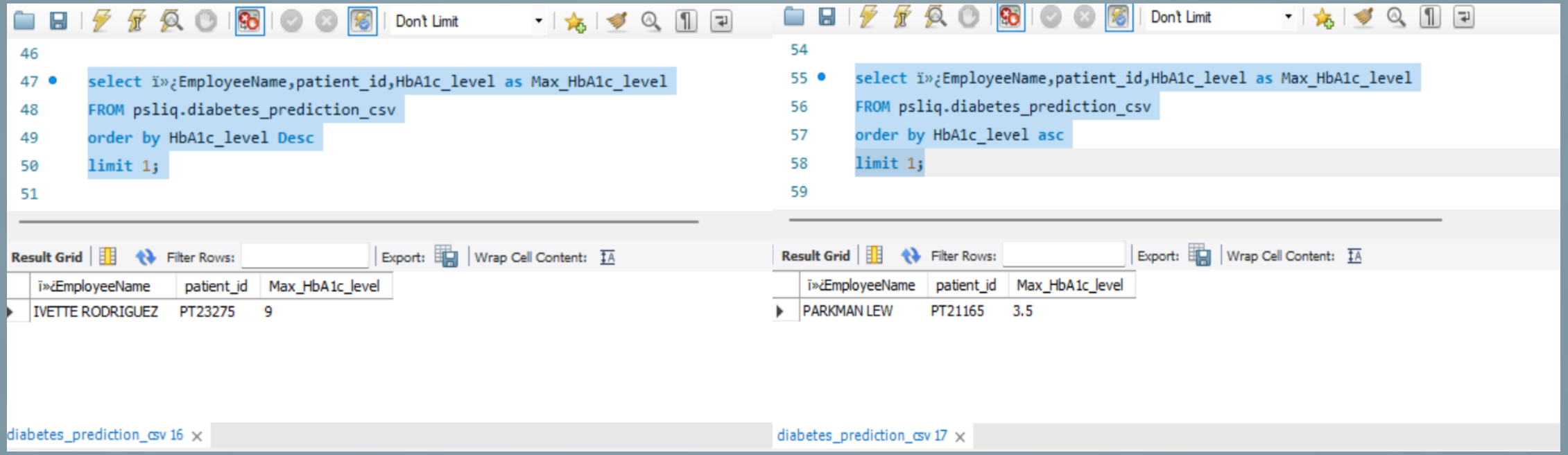
7. GROUP PATIENTS BY SMOKING HISTORY AND COUNT HOW MANY SMOKERS AND NON SMOKERS THERE ARE.



8. RETRIEVE THE PATIENT_IDS OF PATIENTS WHO HAVE A BMI GREATER THAN THE AVERAGE BMI.



9. FIND THE PATIENT WITH THE HIGHEST HBA1C LEVEL AND THE PATIENT WITH THE LOWEST HBA1CLEVEL.



```
select ï»¿EmployeeName,patient_id,HbA1c_level as Max_HbA1c_level
FROM psliq.diabetes_prediction_csv
order by HbA1c_level Desc
limit 1;
```

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

ï»¿EmployeeName	patient_id	Max_HbA1c_level
IVETTE RODRIGUEZ	PT23275	9

diabetes_prediction_csv 16

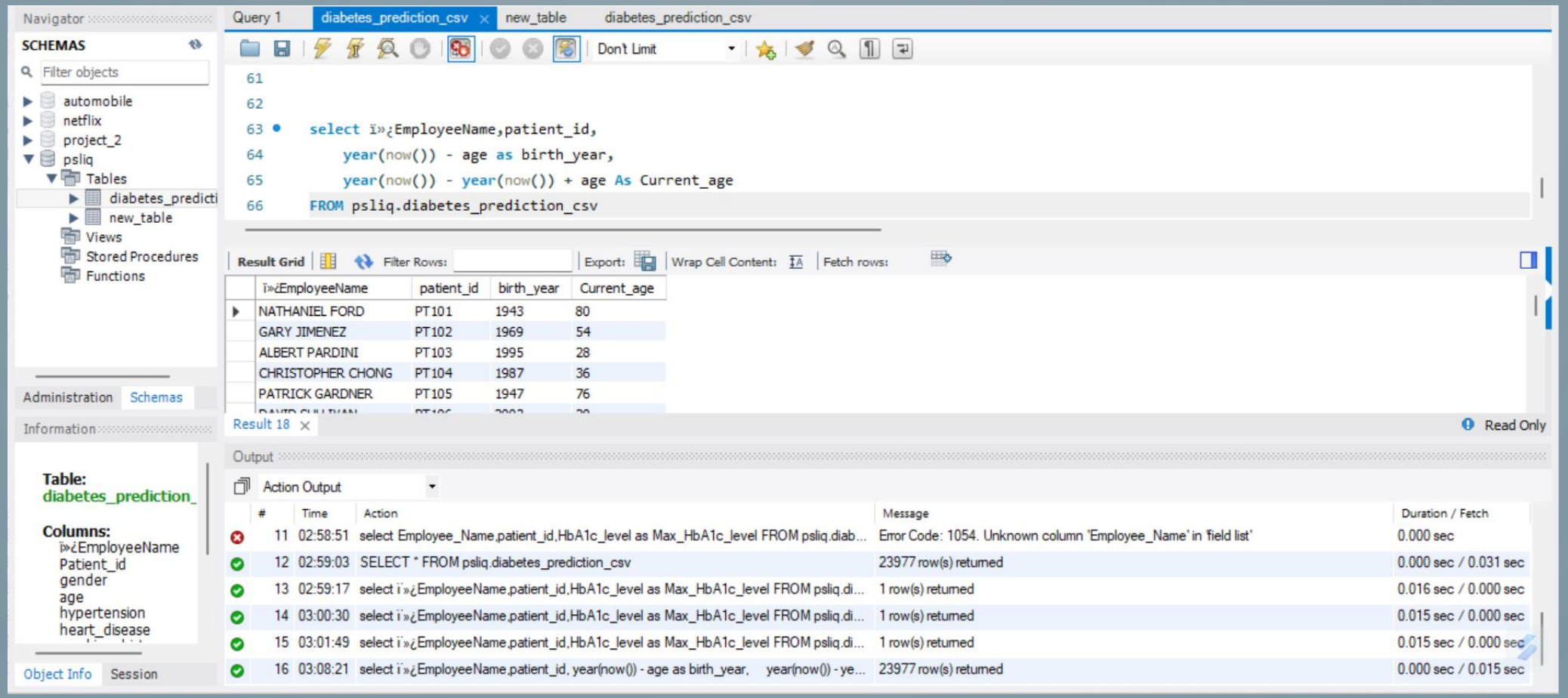
```
select ï»¿EmployeeName,patient_id,HbA1c_level as Max_HbA1c_level
FROM psliq.diabetes_prediction_csv
order by HbA1c_level asc
limit 1;
```

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

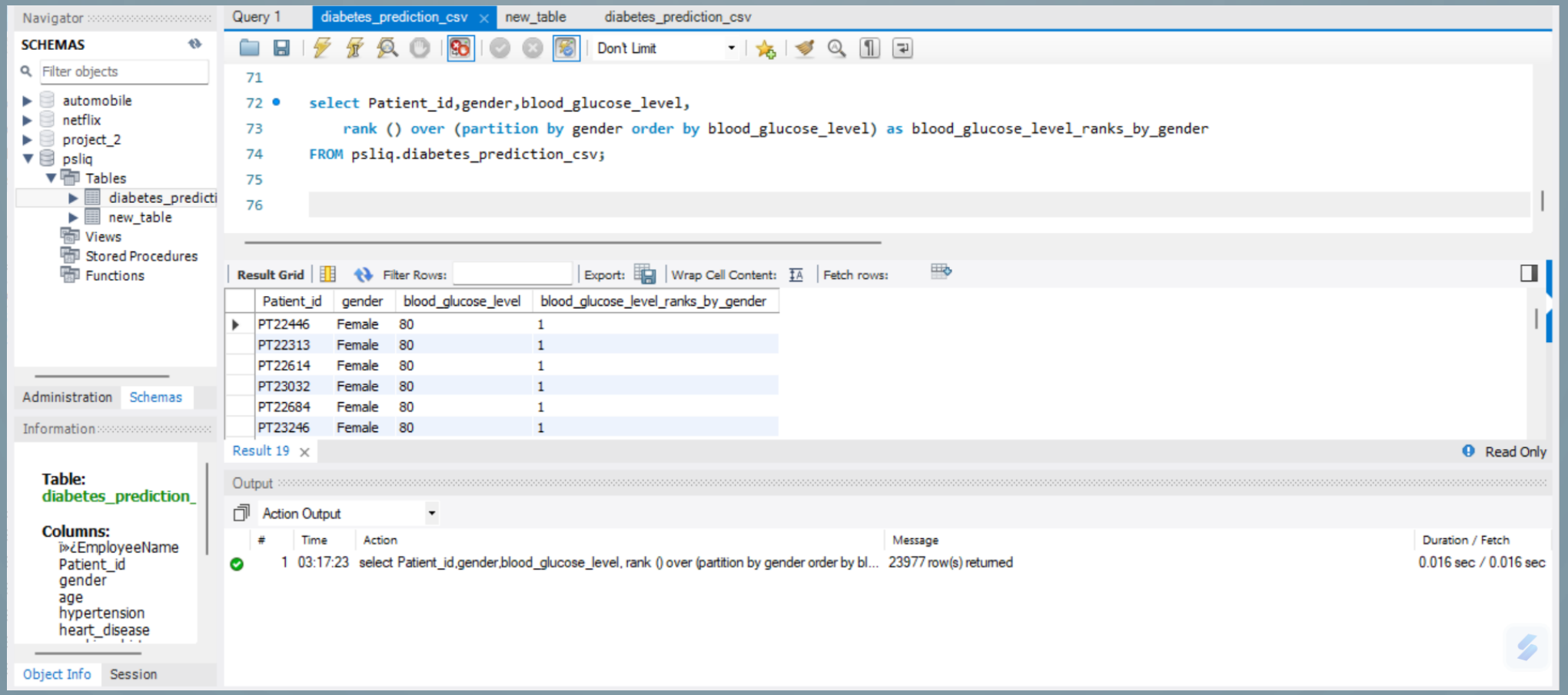
ï»¿EmployeeName	patient_id	Max_HbA1c_level
PARKMAN LEW	PT21165	3.5

diabetes_prediction_csv 17

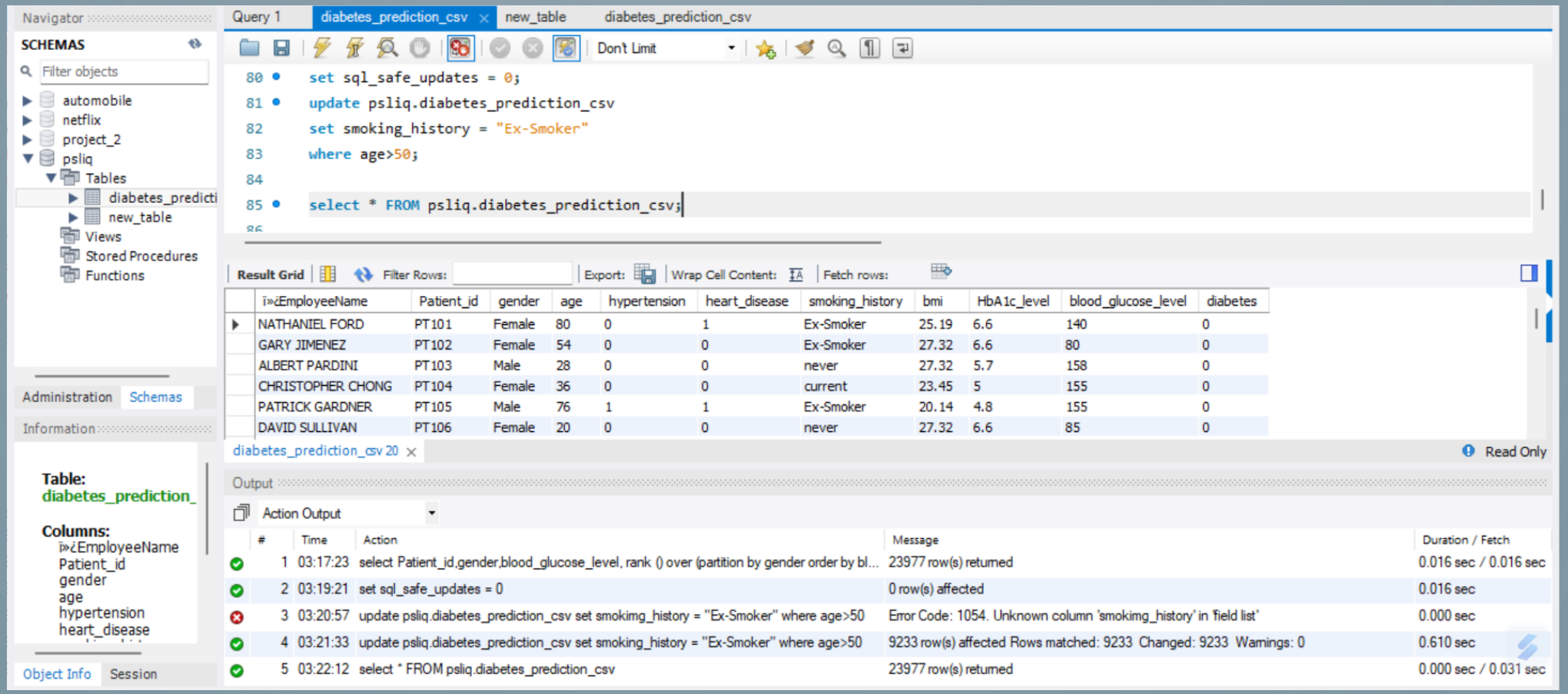
10. CALCULATE THE AGE OF PATIENTS IN YEARS (ASSUMING THE CURRENT DATE AS OF NOW).



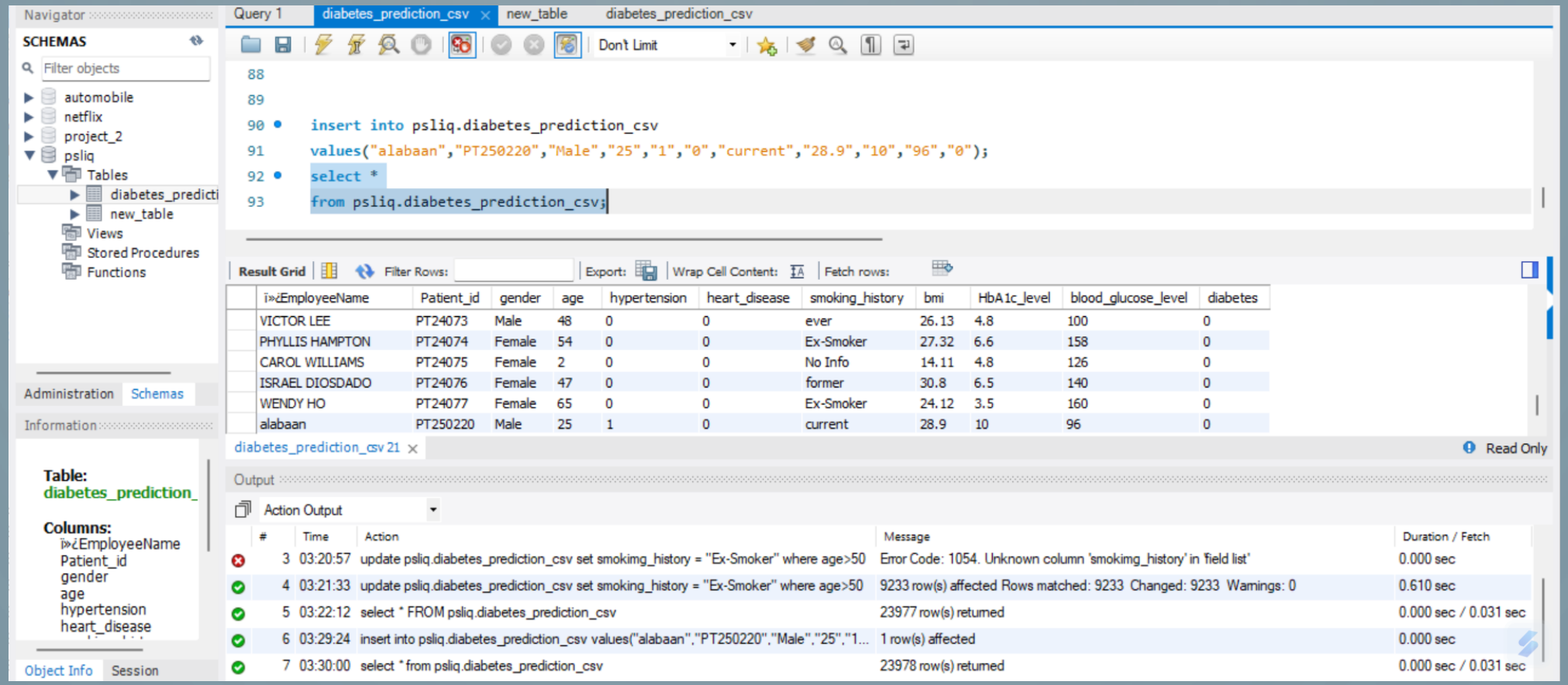
11. RANK PATIENTS BY BLOOD GLUCOSE LEVEL WITHIN EACH GENDER GROUP.



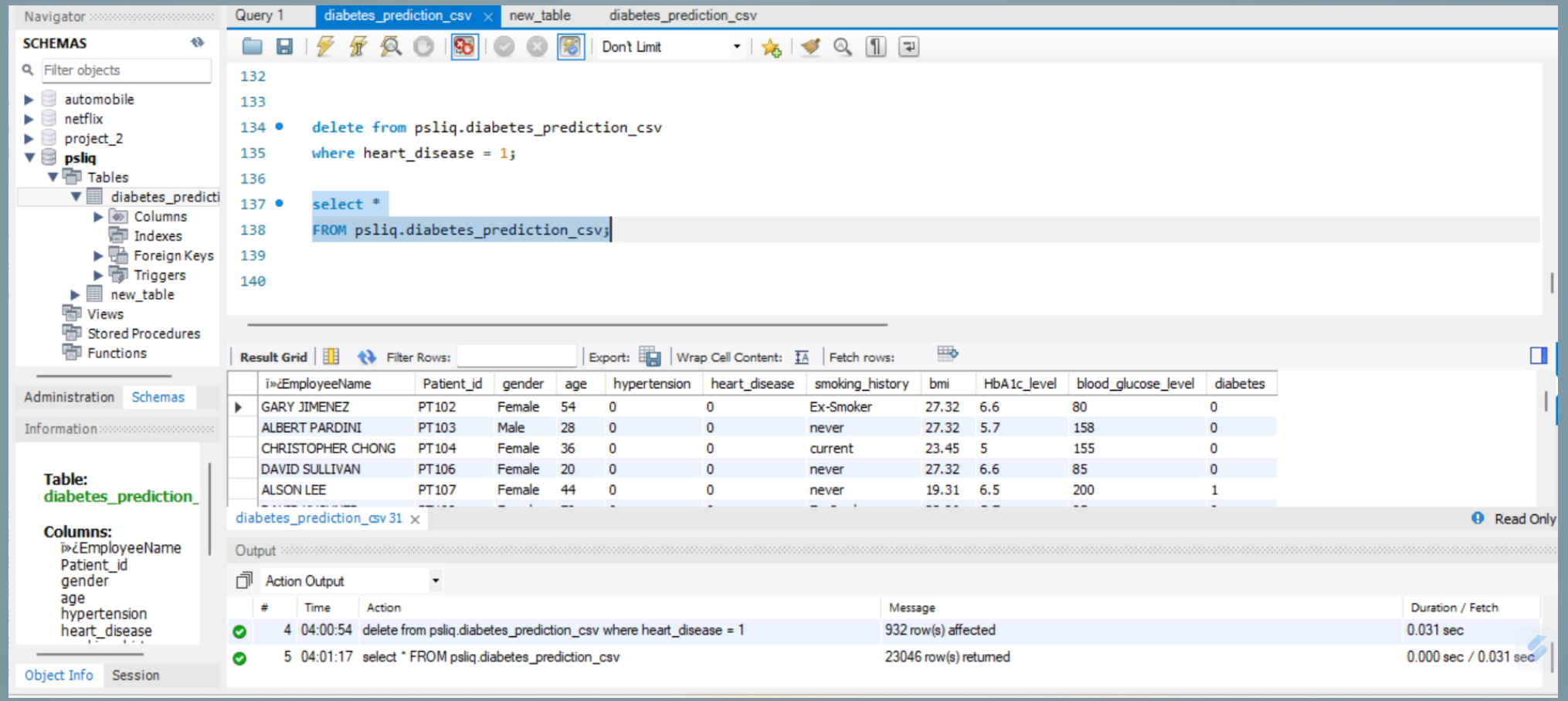
12. UPDATE THE SMOKING HISTORY OF PATIENTS WHO ARE OLDER THAN 50 TO "EX-SMOKER."



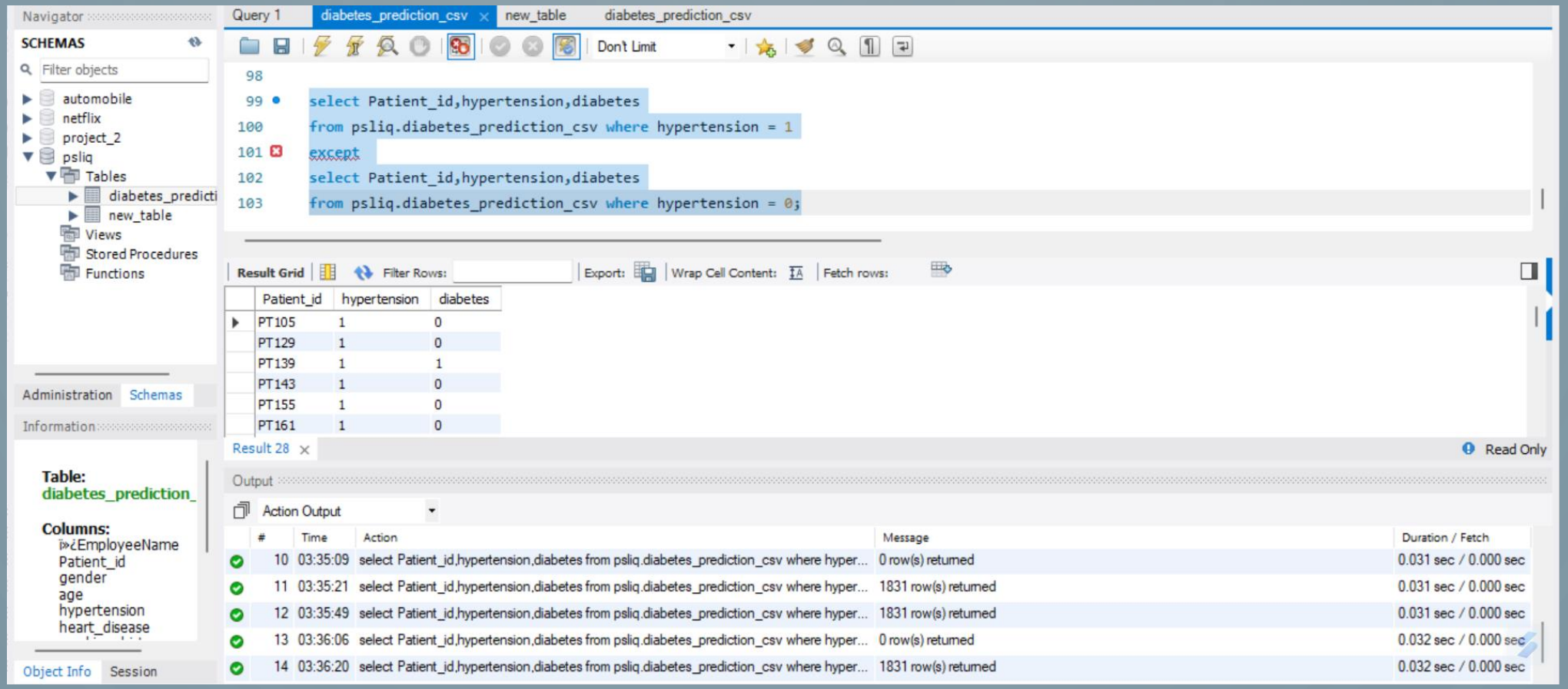
13. INSERT A NEW PATIENT INTO THE DATABASE WITH SAMPLE DATA.



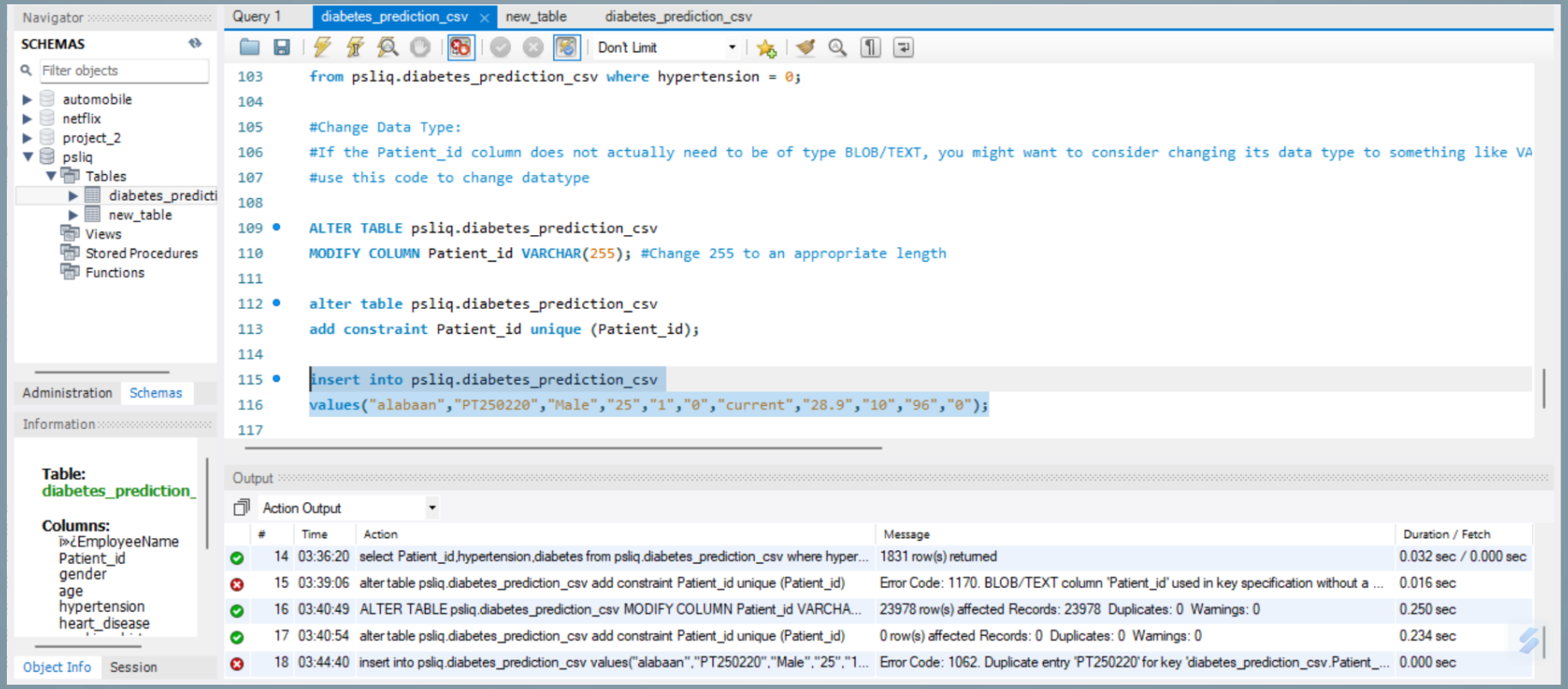
14. DELETE ALL PATIENTS WITH HEART DISEASE FROM THE DATABASE.



15. FIND PATIENTS WHO HAVE HYPERTENSION BUT NOT DIABETES USING THE EXCEPT OPERATOR.

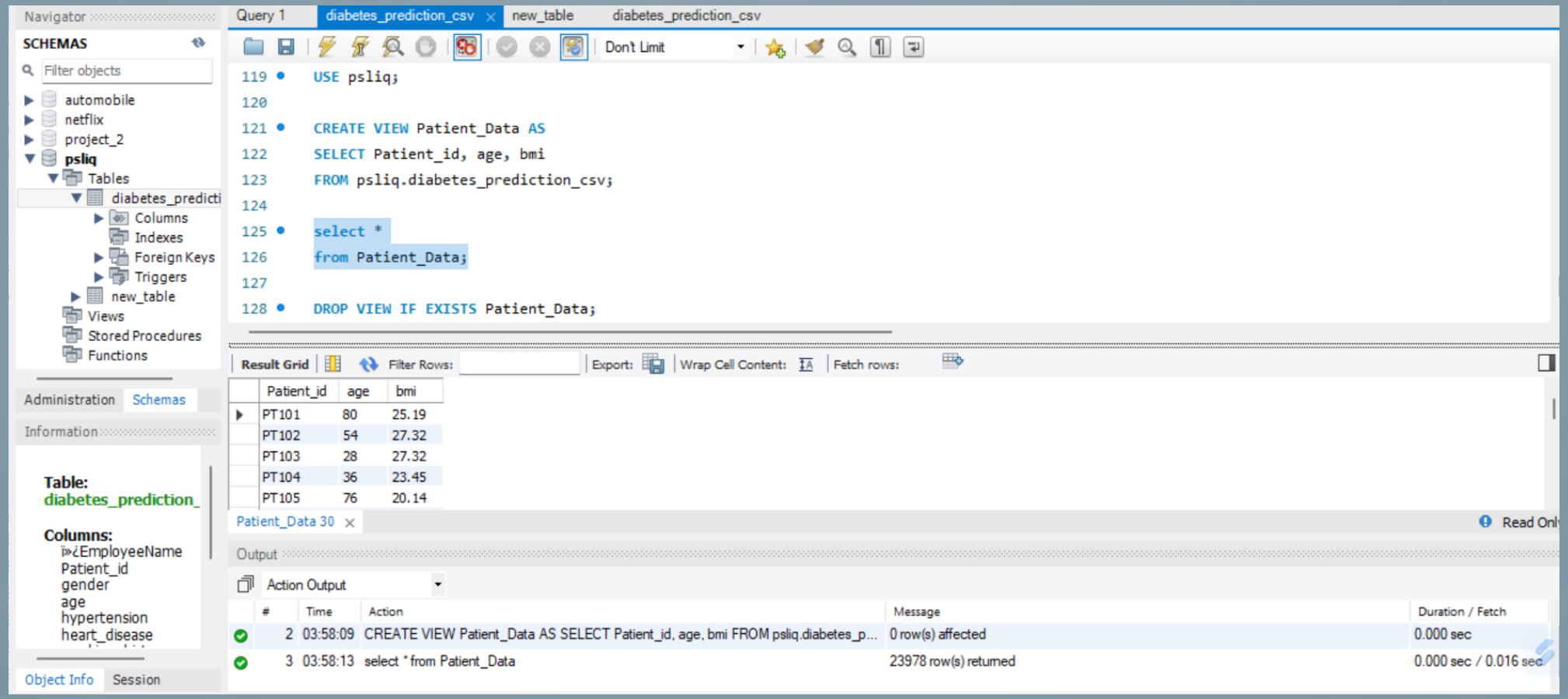


16. DEFINE A UNIQUE CONSTRAINT ON THE “PATIENT_ID" COLUMN TO ENSURE ITS VALUES ARE UNIQUE.



Note : Since we added a UNIQUE constraint to 'Patient_id' , when we insert a duplicate 'Patient_id' it should throw an error

17. CREATE A VIEW THAT DISPLAYS THE PATIENT_IDS, AGES, AND BMI OF PATIENTS.



18. SUGGEST IMPROVEMENTS IN THE DATABASE SCHEMA TO REDUCE DATA REDUNDANCY AND IMPROVE DATA INTEGRITY.

- **Foreign Key Relationships:**

•Establish and enforce foreign key relationships between tables to maintain referential integrity.

•This helps prevent orphaned records and ensures consistency in the data.

- **Normalization:**

•Break down large tables into smaller, related tables through normalization to reduce redundancy.

•Ensure that each table has a primary key to uniquely identify records.

- **Use of Unique Constraints:**

•Implement unique constraints on columns where applicable to prevent duplicate values.

•Utilize unique indexes to enforce uniqueness efficiently.

- **Data Types and Sizes Optimization:**

•Optimize data types and sizes for columns to reduce storage requirements.

•Avoid using excessively large data types if smaller ones are sufficient.

- **Denormalization (where appropriate):**

•Evaluate the need for denormalization for performance gains, considering read vs. write operations.

•Use denormalization selectively, and document reasons for doing so

19. EXPLAIN HOW YOU CAN OPTIMIZE THE PERFORMANCE OF SQL QUERIES ON THIS DATASET.

- **Use Indexing:**

•Identify and create indexes on columns frequently used in WHERE clauses, JOIN conditions, and ORDER BY clauses.

•However, be cautious not to over-index, as it can lead to overhead during data modification operations.

- **Optimize SELECT Queries:**

•Retrieve only the necessary columns rather than using SELECT *.

•Avoid using SELECT DISTINCT unless absolutely necessary, as it can be resource-intensive.

- **JOIN Optimization:**

•Use INNER JOINs instead of OUTER JOINs whenever possible, as INNER JOINs are generally more efficient.

•Consider using appropriate join algorithms (e.g., HASH JOIN, MERGE JOIN) based on the characteristics of your data.

- **Subquery Optimization:**

•Refactor subqueries to JOINs where feasible, as JOINs are often more performant than subqueries.

•Use EXISTS or IN clauses judiciously, as they can impact performance.

- **Avoid SELECT * in Production:**

•Explicitly list the columns you need in the SELECT statement instead of using SELECT *.
•This reduces the amount of data transferred and improves query performance.

- **Partitioning Tables:**

•Consider partitioning large tables based on a column that is frequently used in queries.
•This can significantly enhance query performance, especially in scenarios where only a subset of data is needed.

- **Optimize WHERE Clauses:**

•Be mindful of the conditions in the WHERE clause and ensure they can leverage existing indexes.
•Use appropriate comparison operators and avoid unnecessary functions in the WHERE clause.

- **Database Configuration:**

•Configure database server settings such as memory allocation, buffer pool size, and parallelism to align with the specific requirements of your workload.

- **Query Plan Analysis:**

•Analyze query execution plans using tools like EXPLAIN (in MySQL) to understand how the database engine is processing the query.
•Optimize queries based on the information provided by the query plan.

THANK YOU