DATA ANALYST INTERNSHIP

Diabetes Prediction Assessment:

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

```
select Patient_id, age from diabetes;
```

Patient_id	age
PT101	80
PT102	54
PT103	28
PT104	36
PT105	76
PT106	20
PT107	44
PT108	79
PT109	42

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

```
select * from diabetes where gender = 'Female' and age > 40;
```

EmployeeName	Patient_id	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
NATHANIEL FORD	PT101	Female	80	0	1	never	25.19	6.6	140	0
GARY JIMENEZ	PT102	Female	54	0	0	No Info	27.32	6.6	80	0
ALSON LEE	PT107	Female	44	0	0	never	19.31	6.5	200	1
DAVID KUSHNER	PT108	Female	79	0	0	No Info	23.86	5.7	85	0
ARTHUR KENNEY	PT111	Female	53	0	0	never	27.32	6.1	85	0
PATRICIA JACKSON	PT112	Female	54	0	0	former	54.7	6	100	0
EDWARD HARRINGTON	PT113	Female	78	0	0	former	36.05	5	130	0
JOHN MARTIN	PT114	Female	67	0	0	never	25.69	5.8	200	0
DAVID FRANKLIN	PT115	Female	76	0	0	No Info	27.32	5	160	0

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

```
select avg(bmi) from diabetes;
```

Avg_bmi
27.32

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

```
select Patient_id, blood_glucose_level from diabetes order by blood_glucose_level desc;
```

Patient_is	blood_glucose_level
PT48347	300
PT32466	300
PT20366	300
PT22786	300
PT25412	300
PT39930	300
PT3341	300
PT15448	300

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

```
select Patient_id, hypertension, diabetes from diabetes
where hypertension = 1 and diabetes = 1;
```

Patient_is	hypertension	diabetes
PT10007	1	1
PT10083	1	1
PT10159	1	1
PT10311	1	1
PT10315	1	1
PT10318	1	1
PT10476	1	1
PT10498	1	1
PT10537	1	1

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

```
Select count(heart_disease) from diabetes where heart_disease = 1;
```

count(heart_disease)
3942

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

```
select smoking_history,count(Patient_id) as smokers
from diabetes group by smoking_history;
```

smoking_history	smokers
current	6442
ever	2716
No Info	24718
never	24221
former	6494
not current	4450

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

```
Select Patient_id, bmi from diabetes where bmi > (select avg(bmi) from diabetes);
```

Patient_id	bmi
PT109	33.64
PT112	54.7
PT113	36.05
PT117	30.36
PT121	36.38
PT124	27.94
PT126	33.76

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

```
Select Patient_id, HbA1c_level from diabetes order by HbA1c_level desc limit 1;
```

Patient_id	HbA1c_level
PT141	9

```
Select Patient_id, HbA1c_level from diabetes order by HbA1c_level asc limit 1;
```

Patient_id	HbA1c_level
PT120	3.5

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

```
select Patient_id, age from diabetes;
```

Patient_is	age
PT1000	34
PT10000	46
PT10001	27
PT10002	48
PT10003	60
PT10004	56
PT10005	53
PT10006	68
PT10007	67
PT10008	26

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

```
Select Patient_id, blood_glucose_level, gender, rank ()
over (partition by gender order by blood_glucose_level desc) from diabetes;
```

Patient_id	blood_glucose_level	gender	rank() over (partition by gender order by blood_glucose_level desc)
PT97622	300	Female	1
PT96814	300	Female	1
PT96815	300	Female	1
PT97708	300	Female	1
PT96902	300	Female	1
PT97955	300	Female	1
PT97141	300	Female	1

12. Insert a new patient into the database with sample data.

```
Update diabetes set smoking_history = "Ex-Smoker"
where age > 50 and Patient_id is not null;
```

PT10002	48	Ex-smoker
PT10003	60	Ex-smoker
PT10004	56	Ex-smoker
PT10005	53	Ex-smoker
PT10006	68	Ex-smoker
PT10007	67	Ex-smoker
PT10008	26	Ex-smoker
PT10009	33	Ex-smoker

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

```
Insert into diabetes values('CRYSTAL CHANG','PT100101',
                            'Female',37,0,0,'former',25.96,5.8,126,0);
```

VIVIAN CHU	PT100099	Female	24	0	0	never	35.42	4	100	0
SAVITREE SATRAM	PT100100	Female	57	0	0	current	22.43	6.6	90	0
CRYSTAL CHANG	PT100101	Female	37	0	0	former	25.96	5.8	126	0

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

```
delete from diabetes where heart_disease=1;
```

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

```
Select * from diabetes where hypertension = 1 and
Patient_id not in (Select 1 from diabetes where heart_disease =1);
```

EmployeeName	Patient_id	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
PATRICK GARDNER	PT105	Male	76	1	1	current	20.14	4.8	155	0
DENISE SCHMITT	PT129	Male	45	1	0	never	26.47	4	158	0
JONES WONG	PT139	Male	50	1	0	current	27.32	5.7	260	1
THOMAS SIRAGUSA	PT143	Female	77	1	1	never	32.02	5	159	0
RAY CRAWFORD	PT155	Female	45	1	0	never	23.05	4.8	130	0
KENNETH SMITH	PT161	Male	44	1	0	current	27.86	6.6	145	0
PATRIC STEELE	PT205	Female	80	1	0	never	27.32	6.8	280	1
CHARLES SCOTT	PT215	Female	55	1	0	never	34.2	5.7	140	0

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

```
Alter table diabetes add constraint patient_id unique (Patient_id);
```

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

```
Create View Patient as Select Patient_id, age, bmi from diabetes;
Select * from patient;
```

EmployeeName	Patient_id	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
NATHANIEL FORD	PT101	Female	80	0	1	never	25.19	6.6	140	0
GARY JIMENEZ	PT102	Female	54	0	0	No Info	27.32	6.6	80	0
ALBERT PARDINI	PT103	Male	28	0	0	never	27.32	5.7	158	0
CHRISTOPHER CHONG	PT104	Female	36	0	0	current	23.45	5	155	0
PATRICK GARDNER	PT105	Male	76	1	1	current	20.14	4.8	155	0
DAVID SULLIVAN	PT106	Female	20	0	0	never	27.32	6.6	85	0
ALSON LEE	PT107	Female	44	0	0	never	19.31	6.5	200	1

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

Optimizing your database schema design is crucial for enhancing the efficiency and functionality of your application. To achieve this, consider applying the following general principles tailored to your specific needs and objectives:

Naming Conventions: Employ appropriate naming conventions for tables, columns, indexes, and other elements. This practice enhances the readability and maintainability of your schema.

Normalization: Organize your data through normalization to eliminate redundancy and ensure consistency. Normalize your data into logically structured tables with minimal duplication and maximum integrity.

Indexes: Improve data retrieval speed by strategically using indexes. These data structures store a subset of your table's data for faster access. Create indexes on frequently queried columns like primary keys, foreign keys, or those with high cardinality.

Partitioning: Manage large tables more efficiently by employing partitioning. This technique involves dividing your table's data into smaller, more manageable units based on criteria such as date, range, or hash. This enhances query performance and simplifies maintenance and backup processes.

Denormalization: Boost the performance of complex queries involving multiple tables through denormalization. This process entails introducing redundant data to minimize the need for joins and aggregations. Use techniques like materialized views, summary tables, or pre-computed columns, but exercise caution to maintain data integrity.

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

Numerous factors influence the performance of SQL queries, ranging from dataset size and structure to query complexity and server resources. To optimize SQL queries,

Query Analysis with EXPLAIN: Utilize EXPLAIN or EXPLAIN ANALYZE commands to scrutinize query execution plans. This reveals potential bottlenecks, inefficiencies, and provides insights into the database engine's processing, including used indexes, algorithms, time, memory consumption, and row scans or returns.

Indexing Strategies: Accelerate data retrieval by judiciously implementing indexes. These structures store a subset of table data for quicker access. Create indexes on frequently queried columns like primary keys, foreign keys, or high cardinality columns. Exercise caution to avoid overusing indexes, considering their impact on storage space, maintenance, and potential slowdown of insertion and update operations.

Effective Joins: Leverage joins for combining data from multiple tables based on common criteria. While joins are powerful, they can be resource-intensive. Choose the appropriate join type (inner, left, right, or full) based on table relationships and cardinality. Specify the join condition using the ON clause, and refrain from using the WHERE clause for filtering joined data to prevent unnecessary scans and computations.

Subquery Optimization: Break down complex queries into manageable components using subqueries. These nested queries perform calculations, aggregations, or comparisons on subsets of data. Differentiate between correlated and uncorrelated subqueries, favoring uncorrelated ones for efficiency and ease of optimization.

Optimizing Views: Employ views to store and reuse frequently used queries. Views, virtual tables defined by queries, simplify and organize queries while enhancing data security and consistency. Be mindful of performance implications and consider using materialized views for complex or resource-intensive queries, storing results in physical tables.