select \*from `examples.diabetic `

---Retrieve the Patient\_id and ages of all patients.

select patient\_id, age from `examples.diabetic `

--Select all female patients who are older than 40.

select EmployeeName, age from `examples.diabetic ` where gender='Female' and age>40

---Calculate the average BMI of patients.

select AVG(bmi) as avg\_bmi from `examples.diabetic `

---List patients in descending order of blood glucose levels.

select EmployeeName,blood\_glucose\_level from `examples.diabetic `

ORDER BY blood\_glucose\_level desc;

---Find patients who have hypertension and diabetes.

select EmployeeName from `examples.diabetic `

WHERE diabetes = 1;

---Determine the number of patients with heart disease.

select EmployeeName from `examples.diabetic ` WHERE heart\_disease =1

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

SELECT smoking\_history, COUNT(\*) as count

FROM `examples.diabetic `

GROUP BY smoking\_history;

----Retrieve the Patient\_ids of patients who have a BMI greater than the average BMI.

select Patient\_id from `examples.diabetic ` WHERE bmi > (select avg(bmi) from `examples.diabetic `)

---Find the patient with the highest HbA1c level and the patient with the lowest HbA1clevel.

select EmployeeName  from `examples.diabetic `

WHERE HbA1c\_level = (SELECT MAX(HbA1c\_level) FROM `examples.diabetic `)

   OR HbA1c\_level = (SELECT MIN(HbA1c\_level) FROM `examples.diabetic `);

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

SELECT Patient\_id, EmployeeName, age, EXTRACT(YEAR FROM CURRENT\_DATE) - Age AS Estimated\_Birth\_Year

from `examples.diabetic `;

--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\_Level\_Rank

from `examples.diabetic `;

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

SELECT EmployeeName, age

from `examples.diabetic `

WHERE smoking\_history = 'not current' AND age > 50;

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

insert into `examples.diabetic `(Patient\_id, EmployeeName,gender, age,  hypertension,  heart\_disease  ,smoking\_history, bmi,  HbA1c\_level ,blood\_glucose\_level,   diabetes) values('PT100101', 'john',55,'Male',58,0,'current',25.8,5.5,100,1)

INSERT INTO `examples.diabetic ` (Patient\_id, EmployeeName, age, gender, hypertension, heart\_disease, smoking\_history, bmi, HbA1c\_level, blood\_glucose\_level, diabetes)

VALUES ('PT100101', 'john', 55, 'Male', 58, 0, 'current', 25.8, 5.5, 100, 1);

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

DELETE from `examples.diabetic ` WHERE heart\_disease =1

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

SELECT Patient\_id, EmployeeName

FROM `examples.diabetic ` d1

WHERE hypertension = 1

AND NOT EXISTS (

    SELECT 1

    FROM `examples.diabetic ` d2

    WHERE d1.Patient\_id = d2.Patient\_id

    AND diabetes = 1

);

--- Define a unique constraint on the "patient\_id" column to ensure its values are unique.

ALTER TABLE `examples.diabetic `

ADD CONSTRAINT UC\_Patient\_id UNIQUE (Patient\_id);

CREATE VIEW `examples.diabetic\_patient\_info` AS

SELECT Patient\_id, age, BMI

FROM `examples.diabetic `;

SELECT \* FROM `examples.diabetic\_patient\_info`;

---add DOB

ALTER TABLE `examples.diabetic `

ADD COLUMN DATE\_OF\_BIRTH DATE;

UPDATE `examples.diabetic `

SET DATE\_OF\_BIRTH = DATE\_ADD(CURRENT\_DATE, INTERVAL -CAST(age AS INT64) YEAR) WHERE TRUE

select \*from `examples.diabetic `

**Normalization:**

Break down tables into smaller, related tables to eliminate redundant data.

Normalize the schema to at least the third normal form to minimize data duplication.

**Primary and Foreign Keys:**

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

Use foreign keys to establish relationships between tables, maintaining referential integrity.

**Use Enums or Lookup Tables:**

Replace repetitive text columns with enums or lookup tables to reduce redundancy.

For example, create a table for smoking history with smoking statuses.

**Consistent Naming Conventions:**

Adopt a consistent naming convention for tables and columns.

Use meaningful names to enhance readability and maintainability.

**Avoid Storing Derived Data:**

Avoid storing data that can be derived from other columns, as it may lead to inconsistencies.

Calculate derived values dynamically in queries or views.

**Handle NULL Values Appropriately:**

Use NULL values for optional or unknown information.

Define columns as NOT NULL where appropriate.

**Document the Schema:**

Document the relationships, constraints, and purpose of each table and column.

Maintain clear documentation to aid future developers and administrators.

**Indexes:**

Add indexes to columns frequently used in search conditions to speed up query performance.

Balance the number and types of indexes to avoid unnecessary overhead.

**Regular Maintenance:**

Regularly review and update the schema based on evolving requirements.

Consider performance tuning and optimization as the data grows.

**Unique Constraints:**

Use unique constraints to enforce uniqueness where necessary.

Ensure that columns with unique constraints are indexed.

**Data Types:**

Use appropriate data types for each column to optimize storage and retrieval.

Avoid using generic data types for specific purposes.

**Partitioning and Clustering:**

Consider partitioning large tables based on a relevant column.

Use clustering to organize data within partitions.

**Audit Trails:**

Implement audit trails to track changes to critical data for accountability.

Include timestamp and user information in audit tables.

**Materialized Views:**

Consider using materialized views for frequently queried aggregations.

Update materialized views periodically to reflect changes in the underlying data.

**Referential Integrity Checks:**

Enforce referential integrity through foreign key constraints.

Use cascading actions judiciously to maintain consistency.