Task 2: Complex Query Writing and Task 3: Query Optimisation

Part 1: Queries on the Fact and Dimension Tables (Star Schema)

The following queries are based on the star schema design with **fact** and **dimension** tables, representing a traditional data warehouse design.

1. High-Risk Patients:

Retrieve a list of patients with high-priority appointments who have a probability of DNA (did not attend) greater than 0.9.

```
SELECT
    p.first_name,
    p.last_name,
    a.priority,
    a.specialty,
    a.hospital,
    d.fulldate AS scheduled_date,
    pr.probability_dna
FROM
    FACT_APPOINTMENT fa
JOIN
    DIM_PATIENT p ON fa.patient_id = p.patient_id
JOIN
    DIM_APPOINTMENT a ON fa.appointment_id = a.appointment_id
JOIN
    FACT_PREDICTION fp ON fa.factap_pk = fp.factap_pk
JOIN
    DIM_PREDICTION pr ON fp.prediction_id = pr.prediction_id
JOIN
    DIM_DATE d ON fa.scheduled_date_id = d.date_id
WHERE
    a.priority = 'High'
    AND pr.probability_dna > 0.9;
```

2. Clinic Performance:

Calculate the average probability of DNA across all appointments for each clinic. Return the top 3 clinics with the highest averages.

```sql
SELECT
        c.clinic_name,
        round(AVG(pr.probability_dna),2) AS avg_probability_dna
    FROM
        FACT_APPOINTMENT fa
    JOIN
        DIM_CLINIC c ON fa.clinic_id = c.clinic_id
    JOIN
        FACT_PREDICTION fp ON fa.factap_pk = fp.factap_pk
    JOIN
        DIM_PREDICTION pr ON fp.prediction_id = pr.prediction_id
    GROUP BY
        c.clinic_name
    ORDER BY
        avg_probability_dna DESC
    LIMIT 3;
```

3. Patient Engagement Rate:

Calculate the engagement rate for each patient, defined as the percentage of appointments where the probability of DNA exceeds 0.8.

```sql
SELECT
    p.first_name,
    p.last_name,
    (COUNT(CASE WHEN pr.probability_dna > 0.8 THEN 1 END)
     * 100.0 / COUNT(*)) AS engagement_rate
FROM
    FACT_APPOINTMENT fa
JOIN
    DIM_PATIENT p ON fa.patient_id = p.patient_id
JOIN
    FACT_PREDICTION fp ON fa.factap_pk = fp.factap_pk
JOIN
    DIM_PREDICTION pr ON fp.prediction_id = pr.prediction_id
GROUP BY
    p.first_name, p.last_name
ORDER BY
    engagement_rate DESC;
```

4. Missed Appointments:

Identify the patients who have missed appointments (probability of DNA > 0.85) more than three times.

```sql
SELECT
    p.first_name,
    p.last_name,
    COUNT(*) AS missed_appointments
FROM
    FACT_APPOINTMENT fa
JOIN
    DIM_PATIENT p ON fa.patient_id = p.patient_id
JOIN
    FACT_PREDICTION fp ON fa.factap_pk = fp.factap_pk
JOIN
    DIM_PREDICTION pr ON fp.prediction_id = pr.prediction_id
WHERE
    pr.probability_dna > 0.85
GROUP BY
    p.first_name, p.last_name
HAVING
    COUNT(*) > 3;
```

Part 2: Queries on the One Big Table (OBT)

The following queries are written based on the **one big table** structure, where data is denormalized and consolidated into a single table. This reduces the need for joins, making queries more straightforward and performant for analytics.

1. High-Risk Patients:

Retrieve a list of patients with high-priority appointments who have a probability of DNA greater than 0.9.

```sql
SELECT
    first_name,
    last_name,
    priority,
    specialty,
    hospital,
    scheduled_fulldate AS scheduled_date,
    probability_dna
FROM
    Unified_Predictions
WHERE
    priority = 'High'
    AND probability_dna > 0.9;
```

2. Clinic Performance:

Calculate the average probability of DNA across all appointments for each clinic. Return the top 3 clinics with the highest averages.

```
    SELECT
        clinic_name,
        round(AVG(probability_dna),2) AS average_probability_dna
    FROM
        Unified_Predictions
    GROUP BY
        clinic_name
    ORDER BY
        average_probability_dna DESC
    LIMIT 3;
```

3. Patient Engagement Rate:

Calculate the engagement rate for each patient, defined as the percentage of appointments where the probability of DNA exceeds 0.8.

```
    SELECT
        patient_id,
        first_name,
        last_name,
        ROUND((SUM(CASE WHEN probability_dna > 0.8 THEN 1 ELSE 0 END)::decimal / COUNT(*))
* 100, 2) AS engagement_rate_percentage
    FROM
        Unified_Predictions
    GROUP BY
        patient_id, first_name, last_name
    ORDER BY
        engagement_rate_percentage DESC;
```

4. Missed Appointments:

Identify the patients who have missed appointments (probability of DNA > 0.85) more than three times.

```
    SELECT
        patient_id,
        first_name,
        last_name,
        COUNT(*) AS missed_appointments_count
    FROM
```

```
    Unified_Predictions
WHERE
    probability_dna > 0.85
GROUP BY
    patient_id, first_name, last_name
HAVING
    COUNT(*) > 3
ORDER BY
    missed_appointments_count DESC;
```

Part 3: Queries on the Original Structure

These queries are based on the original schema provided for the system.

1. High-Risk Patients:

Retrieve a list of patients with high-priority appointments who have a probability of DNA greater than 0.9.

```
SELECT
    p.patient_id,
    p.first_name,
    p.last_name,
    a.scheduled_date,
    a.specialty,
    a.hospital
FROM
    Patient p
JOIN
    Appointment a ON p.patient_id = a.patient_id
JOIN
    Prediction pred ON a.appointment_id = pred.appointment_id
WHERE
    a.priority = 'high' AND
    pred.probability_dna > 0.9;
```

2. Clinic Performance:

Calculate the average probability of DNA across all appointments for each clinic. Return the top 3 clinics with the highest averages.

```
    SELECT
        a.clinic_id,
        AVG(pred.probability_dna) AS avg_probability
    FROM
        Appointment a
    JOIN
        Prediction pred ON a.appointment_id = pred.appointment_id
    GROUP BY
        a.clinic_id
    ORDER BY
        avg_probability DESC
    LIMIT 3;
```

3. Patient Engagement Rate:

Calculate the engagement rate for each patient, defined as the percentage of appointments where the probability of DNA exceeds 0.8.

```
    SELECT
        p.patient_id,
        p.first_name,
        p.last_name,
        COUNT(CASE WHEN pred.probability_dna > 0.8 THEN 1 END) * 100.0 / COUNT(*) AS
engagement_rate
    FROM
        Patient p
    JOIN
        Appointment a ON p.patient_id = a.patient_id
    JOIN
        Prediction pred ON a.appointment_id = pred.appointment_id
    GROUP BY
        p.patient_id
    ORDER BY
        engagement_rate DESC;
```

4. Missed Appointments:

Identify the patients who have missed appointments (probability of DNA > 0.85) more than three times.

```
SELECT
    p.patient_id,
    p.first_name,
    p.last_name,
    COUNT(*) AS missed_appointments
FROM
    Patient p
JOIN
    Appointment a ON p.patient_id = a.patient_id
JOIN
    Prediction pred ON a.appointment_id = pred.appointment_id
WHERE
    pred.probability_dna > 0.85
GROUP BY
    p.patient_id
HAVING
    COUNT(*) > 3;
```

Optimisation Strategies

1. Move to Columnar Format (Redshift Migration)

The tables are currently stored in **Amazon RDS**, a row-oriented database optimized for transactional workloads (OLTP), which can be suboptimal for analytical queries involving large data scans. **Migrate to Amazon Redshift**, a **columnar database** that is optimized for analytical queries and data warehousing (OLAP). Redshift stores data in a columnar format, which significantly improves performance for queries that scan a large number of rows.

1. **Improved Query Performance**: Since analytical queries typically involve aggregations, filtering, and calculations on a subset of columns, columnar storage reduces the amount of data that needs to be read from disk.
2. **Efficient Compression**: Redshift's columnar format allows for higher compression rates and provides lower storage costs and better I/O performance.
3. **Scalability**: Redshift can handle large-scale data sets and scales easily that why it is ideal for growing datasets like those in the UNIFIED_PREDICTIONS and UNIFIED_APPOINTMENT tables.

2. Query specific optimisation

I have tried to incorporate several query optimisation strategies such as indexing, materialised views and window functions

High-Risk Patients: Create an index on the priority and probability_dna columns to speed up the filtering process. This will allow the database to quickly locate rows where priority = 'High' and probability_dna > 0.9.

```
CREATE INDEX idx_unified_priority_probability ON Unified_Predictions(priority,
probability_dna);
```

Clinic Performance: I have implemented **materialized view** here for aggregations that are frequently run. since calculating the average `probability_dna` by clinic is something that is commonly performed operation, I have store precomputed results in a materialized view which reduces the need for repeated calculations.

```
    CREATE MATERIALIZED VIEW clinic_performance_avg AS
    SELECT
        clinic_name,
        AVG(probability_dna) AS average_probability_dna
    FROM
        Unified_Predictions
    GROUP BY
        clinic_name;

    -- Then use this view for querying the top clinics
    SELECT * FROM clinic_performance_avg
    ORDER BY average_probability_dna DESC
    LIMIT 3;
```

Patient Engagement Rate: Use **window functions** instead of GROUP BY for calculating percentages, which can be more efficient in retrieving records and provide more flexibility. This eliminates the need for a full table scan and reduces the number of rows involved in the calculations.

```
    SELECT
        patient_id,
        first_name,
        last_name,
        ROUND((SUM(CASE WHEN probability_dna > 0.8 THEN 1 ELSE 0 END) OVER (PARTITION BY
patient_id)::decimal / COUNT(*) OVER (PARTITION BY patient_id)) * 100, 2) AS
engagement_rate_percentage
    FROM
        Unified_Predictions
    ORDER BY
        engagement_rate_percentage DESC;
```

Missed Appointments: Add an index on `patient_id` and `probability_dna`, as this will optimize both the filtering on `probability_dna > 0.85` and the `GROUP BY` operation.

```
CREATE INDEX idx_patient_id_probability_dna_missed ON Unified_Predictions (patient_id,
probability_dna)
```