Query Proposals

This section details the key information requirements for the system, along with SQL queries for data retrieval and their business context, all this based on the ER diagrama presented on the initial database architecture:

User Information and Role Management

- **Description:** The system must store and manage detailed information about each user, including personal identification data (name, email, password hash), role (e.g., farmer, analyst, administrator), associated agricultural fields, and user preferences (such as alert preferences or notification channels). The system must also associate users with system access controls to enforce role-based permissions.
- **SQL Query (PostgreSQL):**

```sql
SELECT user_id, user_name, role, email
FROM User
WHERE role = Role;
```

- **Purpose:** Retrieves a list of all user with that role.
- **Insight:** Supports role-based access, user segmentation, and administration.
- **Use Case:** Analytics (for admin dashboards or user management tools).

Climate Risk Prediction Data

- **Description:** The system must store and retrieve data related to climate risk predictions, including event type (e.g., flood, drought, frost), location, probability, associated severity level, and prediction timestamp. Each prediction must be traceable to the machine learning model used for generation.
- **SQL Query (PostgreSQL):**

```sql
SELECT p.prediction_id, e.event_name, l.name AS location, p.probability, s.severity_type, p.timestamp
FROM Prediction p
JOIN Event_type e ON p.event_type_id = e.event_type_id
JOIN Severity_level s ON p.severity_level_type_id = s.severity_level_id
JOIN Location l ON p.location_id = l.location_id
WHERE p.timestamp >= CURRENT_DATE - INTERVAL '7 days';
```

- **Purpose:** Retrieves recent predictions by event type and location.
- **Insight:** Helps users understand imminent or frequent risks.
- **Use Case:** Analytics (historical or weekly reports for managers and dashboards).

Personalized Agricultural Recommendations

- **Description:** The system must store actionable recommendations linked to specific climate predictions and tailored to the crop type, field location, and user context. Each recommendation includes a textual description and reference to its originating prediction.
- **SQL Query (PostgreSQL):**

```sql
SELECT u.user_name, c.crop_name, af.field_id, r.recommen_descrip, p.timestamp
FROM Recommendation r
JOIN Prediction p ON r.prediction_id = p.prediction_id
JOIN Agriculture_Field af ON r.field_id = af.field_id
JOIN Crop c ON af.crop_id = c.crop_id
```

```
JOIN User u ON af.user_id = u.user_id
WHERE u.user_id = 1
ORDER BY p.timestamp DESC;
```

- **Purpose:** Retrieves the recommendations for a given user.
- **Insight:** Shows which actions have been recommended based on the current crop and field context.
- **Use Case:** Real-time access for end-users.

Real-Time and Historical Weather Data

- **Description:** The system must collect and manage granular weather data (temperature, humidity, wind speed, timestamp) for each weather station and location. This data should be both real-time and historically queryable.
- **SQL Query (PostgreSQL):**

```
SELECT wd.timestamp, wd.temperature, wd.humidity, ws.station_name, l.name AS location
FROM Weather_Data wd
JOIN Weather_Station ws ON wd.station_id = ws.station_id
JOIN Location l ON ws.location_id = l.location_id
WHERE wd.timestamp BETWEEN NOW() - INTERVAL '1 DAY' AND NOW();
```

- **Purpose:** Retrieves real-time weather data for the last 24 hours.
- **Insight:** Enables immediate decision-making for field operations like irrigation.
- **Use Case:** Real-time access for farmers and analytics for analysts.

Agricultural Field and Crop Information

- **Description:** The system must maintain detailed records of users' agricultural fields, including crop type, field ID, and location coordinates. Each user can be associated with one or more fields.
- **SQL Query (PostgreSQL):**

```
SELECT af.field_id, af.field_size, l.name AS location, c.crop_name, u.user_name
FROM Agriculture_Field af
JOIN Crop c ON af.crop_id = c.crop_id
JOIN Location l ON af.location_id = l.location_id
JOIN User u ON af.user_id = u.user_id
WHERE u.user_id = 1;
```

- **Purpose:** Retrieves crop and field information for a specific user.
- **Insight:** Used to personalize recommendations and predictions.
- **Use Case:** Analytics for reports and real-time access to visualize fields on the app.

Alert and Notification Records

- **Description:** The system must store alert records that include which user received what alert, for which prediction, when it was sent, and the current status (e.g., delivered, read, acknowledged).
- **SQL Query (PostgreSQL):**

```sql
SELECT u.user_name, e.event_name, a.status, a.sent_at
FROM Alert a
JOIN User u ON a.user_id = u.user_id
JOIN Prediction p ON a.prediction_id = p.prediction_id
JOIN Event_type e ON p.event_type_id = e.event_type_id
WHERE u.user_id = 1
ORDER BY a.sent_at DESC;
```

- **Purpose:** Displays all alerts received by a user, including status and associated risk.
- **Insight:** Helps users and admins track how alerts are being delivered and acknowledged.
- **Use Case:** Real-time access for farmers, analytics for admins.

User Interaction Logs and Feedback

- **Description:** The system must store user interaction data and optional feedback related to recommendations, alerts, and platform usage. These logs include timestamps, device type, interaction type (e.g., viewed recommendation, dismissed alert, submitted feedback), and additional metadata like geolocation, response time, or comments. This information can vary in structure depending on the event or action.
- **NoSQL Query (MongoDB):**

```javascript
db.interaction_logs.find({
  interaction_type: "recommendation_viewed",
  user_id: "u001"
}).sort({ timestamp: -1 }).limit(5);
```

- **Purpose:** Retrieves the 5 most recent interactions where the user with `ID u001` viewed a recommendation.
- **Insight:** Helps monitor user engagement with the recommendation system, understand behavior patterns, and identify which recommendations are being read or ignored.
- **Use Case:** Real-time access for activity monitoring in the user dashboard. Analytics for `UX` evaluation and system improvement.

Raw API Response Storage

- **Description:** The system must optionally store raw `JSON` payloads from external weather `APIs` (such as `OpenWeather`) for auditing, reprocessing, and debugging purposes. These payloads may vary in structure depending on the source, `API` version, or requested endpoint.
- **NoSQL Query (MongoDB):**

```javascript
db.api_responses.find({
  source: "OpenWeather",
  "location.name": "Cali",
  timestamp: { $gte: ISODate("2025-05-27T00:00:00Z") }
}).sort({ timestamp: -1 }).limit(10);
```

- **Purpose:** Retrieves the last 10 raw weather `API` responses from the `OpenWeather` source for the city of Cali.
- **Insight:** Supports auditability of predictions, debugging of data pipelines, and retraining of climate models using original `API` inputs.
- **Use Case:** Analytics for historical comparison and model validation. Data engineering use for reprocessing or feature extraction.