CPSC 304 MileStone4

Project Name: Seed Gem - A gardening management tool

Milestone #: __ 4_____

Date: ___29th_Nov 2024_________

Group Number: ____14_____

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Shu (Charlie) Chen	61628137	i8r6v	schen622@student.ubc.ca
David Tianyi Yin	49385537	k1l2s	tianyiy.ubc@gmail.com
Lewis Li	39058169	b9d8f	weitianl@student.ubc.ca

Link to Repo:

https://github.students.cs.ubc.ca/CPSC304-2024W-T1/project_b9d8f_i8r6v_k1l2s.git

Project description

Seed Germ is a comprehensive system for managing and tracking plants within a garden, focusing on plant growth data analytics. The system accommodates various plant types and stages, tracks scheduled and ad-hoc events, and provides traceability from seed purchase through to harvest and distribution. Its functionality includes maintaining records of plant events such as watering, weeding, etc., observations such as bud breaking, fruiting, etc., and generating timelines and analytics for efficient garden management.

Changes of schema from the last submission

- **Cultivar Representation**:
    - **Old Schema**: Cultivar was represented as an ISA relationship under Plant.
    - **Current Schema**: Cultivar has been simplified to focus directly on plants with attributes like `expected_yield_weight` and `tags`.
    - **Reason**: Cultivar shares many attributes with the plant, by simplifying the data stored in cultivar, we can reduce redundancies.
- **Plant Stages**:
    - **Old Schema**: The `Plant goes_through Stage` relationship tracked plant stages.
    - **Current Schema**: The `Stage` is a weak entity of `Plant` and is in a many-to-many relationship with `Batch`.
    - **Reason**: As `Batch` is used to track the changes of plants, we want to use this relationship to facilitate the recording of the plant stage.
- **Order Structure**:
    - **Old Schema**: Orders included total participation constraints for plants and suppliers.
    - **Current Schema**: Order now has a weak entity Order_item, which has unique suppliers and plants.
    - Reason: Now order items can uniquely determine Plants and make tracking possible.
- **Soil Condition**:
    - **Old Schema**: The relationship `Location distinguished_by Soil_condition` existed.

- **Current Schema**: Soil condition is now in a many-to-many relationship with Location and has new attributes, such as pH and organic matter concentration.
- Reason: Now a location can have different soil types, allowing the field to have pots containing different soils in the field.

- **Batch Tracking**:

  - **Old Schema**: Batch relationships (e.g., Batch is_at Stage, Batch goes_to Location) were defined with attributes like `current_stage` and `care_notes`.
  - **Current Schema**: The Stage is listed as a separate entity, with additional attributes like yield weight to collect more data.
  - Reason: Enhanced focus on batch tracking, with more granularity on plant events and stages.

Dynamically Coded Queries

A list of all SQL queries used to satisfy the rubric items and where each query can

be found in the code (file name and line number(s)).

2.1.1 Insert

On all pages, the user can insert tuples. As an example, At:

project_b9d8f_i8r6v_k1l2s/appServices/CultivarsAppService.js, Line 33:

```
INSERT INTO CULTIVAR (plant_ID, cultivar_name)
       VALUES (:plant_ID, :cultivar_name);
```

2.1.2 Update

On all pages, the user can update tuples. As an example, At:

project_b9d8f_i8r6v_k1l2s/appServices/PlantsAppServices.js, Line 96

```
UPDATE PLANT SET
                  plant_ID = :plant_id,
                  yield_type = :yield_type,
                  common_name = :common_name,
                  scientific_name = :scientific_name,
                  overview_notes = :overview_notes
      WHERE plant_ID=:plant_id
```

2.1.3 Delete

On all pages, the user can delete tuples. As an example, At:

project_b9d8f_i8r6v_k1l2s/appServices/SoilConditionsAppService.js,Line 80:

```
`DELETE FROM SOIL_CONDITION
```

```
                WHERE soil_type = :soil_type`
```

Note: on delete cascade implemented in “distinguished_by”, found in: project_b9d8f_i8r6v_k1l2s/seed_gem.sql, line 99.

2.1.4 Selection

On the Soil Condition page, users can filter data based on specific conditions and selected attributes, combining these criteria using AND/OR logic.

At:

project_b9d8f_i8r6v_k1l2s/appServices/SoilConditionsAppService.js, Line 105:

```
SELECT soil_type, pH, organic_matter_concentration
                   FROM Soil_condition
                   WHERE ${conditions}
```

2.1.5 Projection

On the All Orders page, the user can choose to only display the selected columns.

At:

project_b9d8f_i8r6v_k1l2s/appServices/OrdersAppServices.js, Line 108:

```
const query = `SELECT ${selection} FROM Orders`;
```

2.1.6 Join

In the Condition of the soil in each area page, users can decide the information of which field they want to view with the corresponding soil_condition.

At project_b9d8f_i8r6v_k1l2s/appServices/ DistinguishedByAppService.js, line 202

```
SELECT
        Location.field_name,
        Location.zone_id,
        Location.is_outdoor,
        Soil_condition.soil_type,
        Soil_condition.pH,
        Soil_condition.organic_matter_concentration
      FROM
        distinguished_by
      JOIN
        Location
      ON
        distinguished_by.field_name = Location.field_name
        AND distinguished_by.zone_id = Location.zone_id
      JOIN
        Soil_condition
      ON
        distinguished_by.soil_type = Soil_condition.soil_type
      WHERE
        Location.field_name = :field_name
```

Hard Coded Queries:

2.1.7 Aggregation with Group By

Count Plants by Yield Type.

At:

project_b9d8f_i8r6v_k1l2s/appServices/PlantsAppServices.js, Line 152:

```
SELECT yield_type, COUNT(*) AS count
      FROM Plant
      GROUP BY yield_type
```

2.1.8 Aggregation with Having

Highlights yield_types with more diversity (i.e., containing >=2 different plants)

At:

project_b9d8f_i8r6v_k1l2s/appServices/PlantsAppServices.js, Line 172:

```
SELECT yield_type, COUNT(*) AS count
      FROM Plant
      GROUP BY yield_type
      HAVING COUNT(*) >= 2
```

Count plants that yield fruits:

At:

project_b9d8f_i8r6v_k1l2s/appServices/PlantsAppServices.js, Line 137:

```
SELECT COUNT(*) AS fruit_count
       FROM Plant
       GROUP BY yield_type
       HAVING LOWER(yield_type) = 'fruit
```

2.1.9 Nested Aggregation with Group By

Show Good Locations:

A "Good Location" is defined as a location where the organic matter concentration is greater than or equal to the averages of concentration of all different fields.

At:

project_b9d8f_i8r6v_k1l2s/appServices/DistinguishedByAppService.js, Line 114:

```
SELECT
    Location.field_name,
    location.zone_ID,
    location.is_outdoor,
    distinguished_by.soil_type,
    Soil_condition.pH,
    Soil_condition.organic_matter_concentration
FROM
    distinguished_by, Location, Soil_condition
WHERE
    distinguished_by.field_name = Location.field_name
    AND
    distinguished_by.zone_id = Location.zone_id
    AND
    distinguished_by.soil_type = Soil_condition.soil_type
    AND
    Soil_condition.organic_matter_concentration >= ALL (
        SELECT
            AVG(Soil_condition.organic_matter_concentration)
        FROM
            distinguished_by
        JOIN
            Location
        ON
            distinguished_by.field_name = Location.field_name
            AND distinguished_by.zone_id = Location.zone_id
        JOIN
            Soil_condition
        ON
            distinguished_by.soil_type = Soil_condition.soil_type
        GROUP BY
            Location.field_name
    );
```

2.1.10 Division

Find Super Fields (Division):

A "Super Field" is defined as a field that is associated with all types of soil conditions in the database.

At:

project_b9d8f_i8r6v_k1l2s/appServices/DistinguishedByAppService.js, Line 158:

```
SELECT DISTINCT d.field_name, d.zone_id
FROM distinguished_by d
WHERE NOT EXISTS (
    SELECT s.soil_type
    FROM Soil_condition s
    WHERE NOT EXISTS (
        SELECT 1
        FROM distinguished_by d2
        WHERE d2.field_name = d.field_name
          AND d2.soil_type = s.soil_type
```

Bonus: Enforce many-to-many total participation with triggers

We added a few triggers to enforce the total participation between the location and soil condition.

First, we have a trigger to reject a deletion if the tuple deleted is the last one that contains a certain location. If the tuple is removed, this location will be missing a soil condition, violating the total participation constraint.

At:

project_b9d8f_i8r6v_k1l2s/seed_gem.sql, Line: 450:

CREATE OR REPLACE TRIGGER prevent_orphaned_location

```
BEFORE DELETE ON distinguished_by
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    -- Check if the deletion will leave the Location without any
distinguished_by entries
    SELECT COUNT(*) INTO v_count
    FROM distinguished_by
    WHERE field_name = :OLD.field_name AND zone_ID = :OLD.zone_ID;

    -- If this is the last entry, raise an error
    IF v_count = 1 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Cannot delete the last distinguished_by
entry for a Location.');
    END IF;
END;
/
```

We also have a trigger that ensures each location tuple has an associated soil condition in the table distinguish_by.

project_b9d8f_i8r6v_k1l2s/seed_gem.sql, Line: 476:

```
CREATE OR REPLACE TRIGGER prevent_orphaned_location
BEFORE DELETE ON distinguished_by
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    -- Check if the deletion will leave the Location without any
distinguished_by entries
    SELECT COUNT(*) INTO v_count
    FROM distinguished_by
    WHERE field_name = :OLD.field_name AND zone_ID = :OLD.zone_ID;

    -- If this is the last entry, raise an error
    IF v_count = 1 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Cannot delete the last distinguished_by
entry for a Location.');
    END IF;
END;
/
```