An Oracle SQL Database-Based Project,

AgroNex

Empowering Farmers With Data

Name: Choyan Mitra Barua Bijoy
Roll : 2007101
Year: 3rd
Term: 1st
Department: Computer Science And Engineering

PROJECT INTRODUCTION

✪Short Description:

"AgroNex" is like a digital assistant for farmers. It helps them keep track of everything on their farm – from what crops they're growing to when they need to plant or harvest. With "AgroNex", farmers can also manage their equipment and keep an eye on sales. It's all in one easy-to-use app, so farmers can spend less time on paperwork and more time doing what they love – farming!

✪Scope And Importance Of AgroNex:

☞Scope:

AgroNex helps farmers manage their farms better. It keeps track of things like what crops to plant, how to take care of fields, what equipment to use, and how much they're selling. It works for all kinds of farmers, whether they have big farms or small ones.

☞Importance:

AgroNex is really important because it gives farmers useful information to make good decisions. It helps them grow more crops, use fewer resources, and work together with others in farming. This means better food for everyone and helps farmers make more money too. Also It helps farmers store data and retrieve it easily where the collection of data is well organized so that it is easily available for use.

PROJECT OBJECTIVES

1. Optimized Relational Database Design:

Design a relational database schema that efficiently organizes farm and farmer-related data such that field-data, harvest-data etc, ensuring normalized tables, appropriate indexing, and optimal relationships between entities. This design should facilitate seamless data retrieval, manipulation, and analysis, promoting data integrity and scalability.

2. Streamlined CRUD Operations:

Implement CRUD (Create, Read, Update, Delete) functionalities within the database system to enable farmers and farm managers to easily add, retrieve, modify, and delete data related to farms, crops, equipment, and other relevant entities. These operations should be user-friendly, intuitive, and securely managed to ensure data consistency and integrity. This also supports the comparatively complex relationship between tables.

3. Performance Optimization and Management:

Employ performance optimization techniques such as query optimization, and caching to enhance the speed and efficiency of database operations. Additionally, implement robust database management practices including backup and recovery procedures, resource monitoring, and proactive maintenance to ensure optimal database performance, reliability, and availability for farm management tasks.

DATABASE DESIGN AND OVERVIEW

The “AgroNex” database schema is designed to facilitate the management of farms and farmer-related data. It consists of several interconnected tables representing various entities involved in agricultural operations. The schema is structured to ensure data integrity, scalability, and efficient data retrieval and manipulation.

1. Farmers: The central entity representing individuals engaged in farming activities. Each farmer is uniquely identified by a farmer_id and associated with farms and equipment.

2. Farms: Represents individual farms managed by farmers. Each farm has a unique farm_id and contains details such as farm name, location, and size.

3. Farmer_farm: Serves as a bridge table to establish a many-to-many relationship between farmers and farms, allowing multiple farmers to be associated with multiple farms.

4. Crops: Stores information about different types of crops that can be grown on farms. Each crop is uniquely identified by a crop_id and contains details like crop name, type, and planting season.

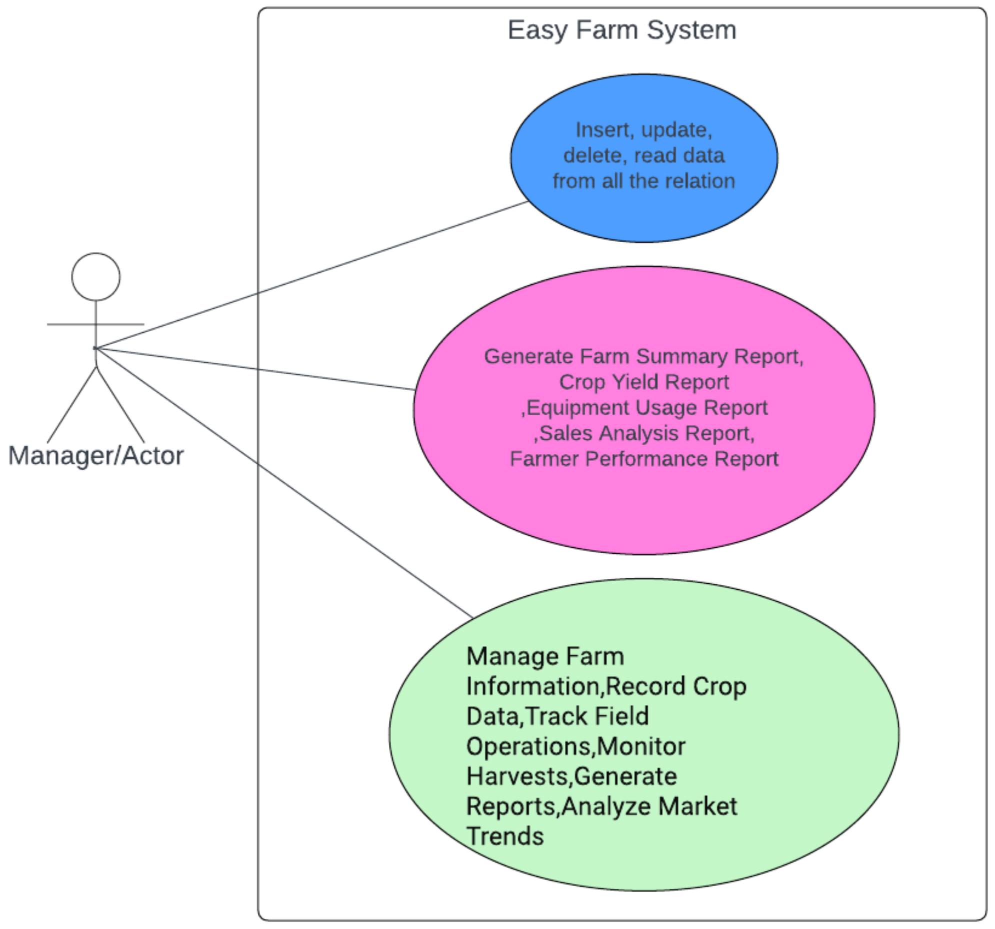
5. Fields: Represents specific fields within farms where crops are planted. Fields are associated with farms and crops and contain information such as field name, size, and planting date.

6. Equipment: Records details about equipment owned by farmers. Each equipment item is uniquely identified by an equipment_id and associated with a farmer. Details include equipment type, description, and purchase date.

7. Harvests: Tracks information about harvests from fields, including harvest date, yield, and associated crop and field.

8. Sales: Stores data about sales transactions, including sale date, amount, and associated harvest.

✪ Use Case Diagram:


Easy Farm System
Insert, update, delete, read data from all the relation
Generate Farm Summary Report, Crop Yield Report ,Equipment Usage Report ,Sales Analysis Report, Farmer Performance Report
Manage Farm Information,Record Crop Data,Track Field Operations,Monitor Harvests,Generate Reports,Analyze Market Trends
Manager/Actor

✪ Reasoning for the Design Choices:

The design for the AgroNex project prioritizes scalability, data integrity, flexibility, performance optimization, user-friendliness, and interoperability. These factors ensure the system can efficiently handle data growth, maintain accuracy, adapt to changes, optimize performance, enhance user experience, and integrate with other agricultural systems.

✪ Table Design:

```sql
-- table: farmers
-- description: stores information about farmers,
--including their name, address, phone number, and email.
create table farmers
(
  farmer_id int not null,
  name varchar(50),
  address varchar(60),
  phone_number varchar(11),
  email varchar(50),
  primary key (farmer_id)
);
-- table: farms
-- description: represents individual farms managed by farmers,
--storing details such as farm name, location, and size.
create table farms
(
  farm_id int,
  farm_name varchar(30),
  location varchar(50),
  farm_size decimal(10, 2),
  primary key (farm_id)
);
```

```sql
-- table: farmer_farm
-- description: establishes a many-to-many relationship
-- between farmers and farms, allowing multiple farmers
-- to be associated with multiple farms.


create table farmer_farm
(
  farmer_id int,
  farm_id int,
  foreign key (farm_id) references farms (farm_id),
  foreign key (farmer_id) references farmers (farmer_id)
);



-- table: crops
-- description: stores information about
-- different types of crops that can be grown on farms.
create table crops
(
  crop_id int,
  crop_name varchar(20),
  crop_type varchar(25),
  planting_season varchar(20),
  primary key (crop_id)
);
```

```
  -- table: fields
  -- description: represents specific fields within farms where crops
are planted.
  create table fields
  (
    field_id int,
    farm_id int,
    field_name varchar(30),
    field_size decimal(5, 2),
    crop_id int,
    planting_date date,
    primary key (field_id),
    foreign key (farm_id) references farms (farm_id),
    foreign key (crop_id) references crops (crop_id)
  );

  -- table: equipment
  -- description: records details about equipment owned by farmers.
  create table equipment
  (
    equipment_id int,
    farmer_id int,
    equipment_type varchar(20),
    description varchar2(250),
    purchase_date date,
    primary key (equipment_id),
    foreign key (farmer_id) references farmers (farmer_id)
  );
  -- table: harvests
  -- description: tracks information about harvests from fields.
  create table harvests
  (
    harvest_id int,
    field_id int,
    crop_id int,
    harvest_date date,
    yield number(5, 2),
    primary key (harvest_id),
    foreign key (field_id) references fields (field_id),
    foreign key (crop_id) references crops (crop_id)
  );

  -- table: sales
  -- description: stores data about sales transactions related to
harvests.
  create table sales
  (
    sale_id int,
    harvest_id int,
    sale_date date,
    sale_amount number(10),
    primary key (sale_id),
    foreign key (harvest_id) references harvests (harvest_id)
  );
```

Redundant Data and Anomalies:

Redundant data in a database refers to duplicated information stored across multiple records or tables. This redundancy can lead to inefficiency in storage and inconsistencies in data. Anomalies, such as insertion, update, and deletion anomalies, can occur due to redundant data, causing irregularities in data management operations. Normalization techniques help eliminate redundant data and minimize the risk of anomalies, ensuring data integrity and consistency.

Farmers Table: It maintains individual farmer details without redundancy or dependencies on non-key attributes, ensuring each entry contains atomic values.

Farms Table: Similarly, farm information is structured without repetition or transitive dependencies, promoting data integrity.

Farmer_farm Table: This bridge table effectively establishes many-to-many relationships between farmers and farms without introducing redundant data or transitive dependencies.

Crops, Fields, Equipment, Harvests, and Sales Tables: Each of these tables maintains atomic values and relationships without violating normalization principles.

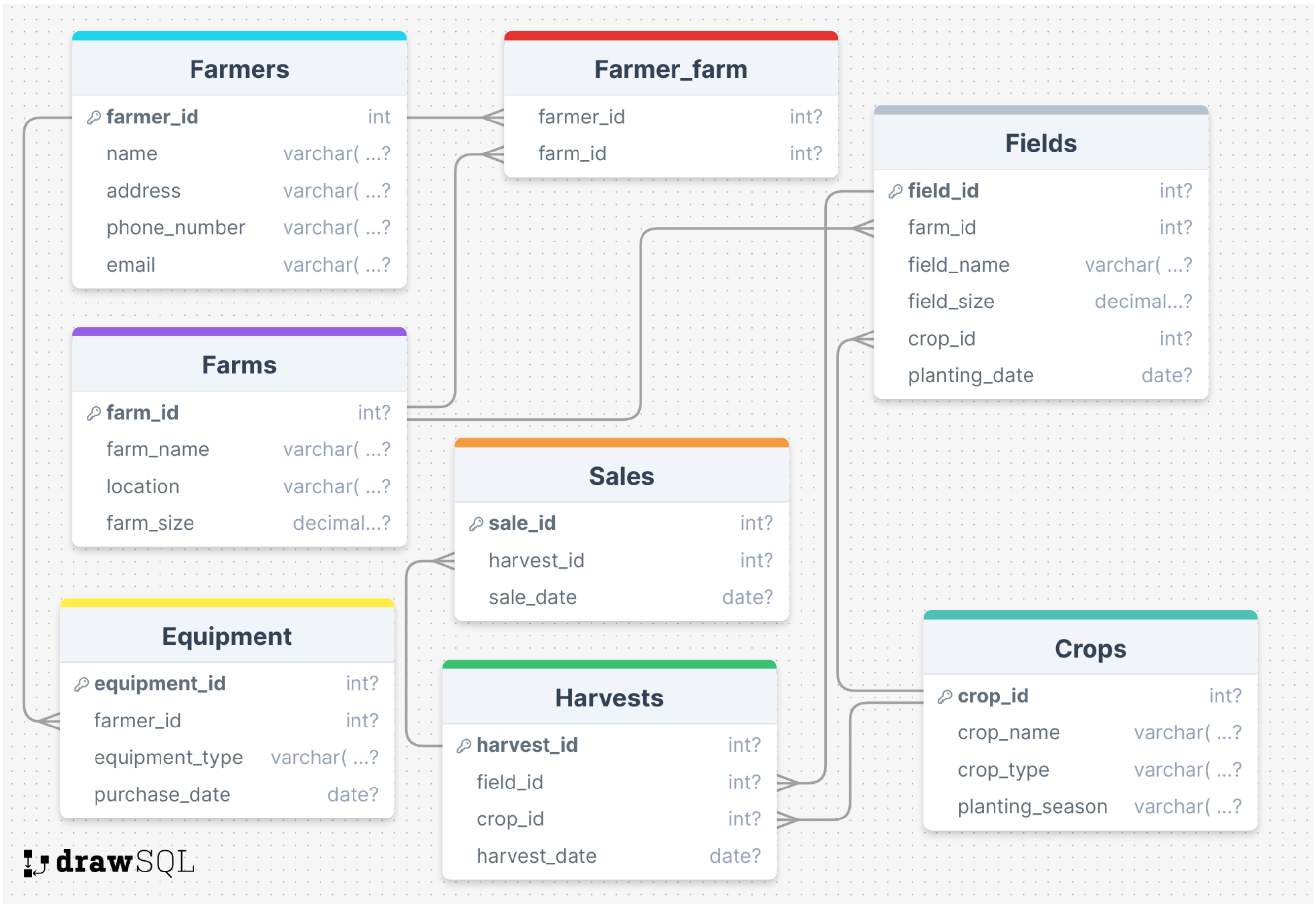
✾ *AgroNex database is organized in a manner consistent with 3NF.*

✪ Users and Data Manipulator :

- **Farmers:** Use AgroNex to manage farming activities, including inputting data about farms, crops, fields, equipment, harvests, and sales.
- **Farm Managers:** Oversee multiple farms, allocate resources efficiently, and analyze performance metrics to improve productivity.
- **Agricultural Advisors:** Provide consultancy services, offer insights based on data analysis, and make recommendations to farmers and agricultural organizations.

- **Government Agencies:** Monitor farming practices, assess compliance with regulations, and gather agricultural data for policy-making and planning purposes.
- **Research Institutions:** Analyze agricultural trends, crop yields, and farming practices based on aggregated data from AgroNex for research and development.
- **Suppliers and Distributors:** Understand market demand, track inventory levels, and plan distribution strategies for agricultural inputs and products.
- **Financial Institutions:** Assess creditworthiness, evaluate risk factors, and monitor loan performance based on farming data and financial records stored in AgroNex.

✪Entity-Relationship Diagram (E-R Diagram) :



✪SQL Queries and Functionality :

SQL queries and functionality play a crucial role in the AgroNex project, enabling users to interact with the database effectively. Here are some SQL queries and functionalities that are essential for the system

☞Data Retrieval:

✦Retrieve Farmer Information:

```
select farmer_id, name, field_id, field_size from farmers, fields where farmer_id = (
select farmer_id from farmer_farm where farmer_farm.farmer_id = farmers.farmer_id
and farmer_farm.farm_id = fields.farm_id);
```

This query retrieves information about farmers along with the fields they own. It selects the farmer's ID, name, and the ID and size of the fields they own.

✦Find The Average Sales Amount:

```
select harvest_id, avg(sale_amount) as average_sale_amount
from sales group by harvest_id;
```

This query calculates the average sales amount for each harvest. It selects the harvest ID and computes the average sale amount for each harvest.

✦Crop Information For a Specific Season:

```
select f.field_id, f.farm_id, f.field_name, f.field_size, f.planting_date, c.crop_id,
c.crop_name, c.crop_type, c.planting_season from fields f inner join crops c on
f.crop_id = c.crop_id where c.planting_season = 'Fall';
```

This query retrieves information about crops planted in a specific season (in this case, 'Fall'). It selects details about fields and associated crops where the planting season matches the specified season.

✦Fields Associated with Farm:

```
select f.farm_id, f.farm_name, count(fi.field_id) as field_count from farms f left outer
join fields fi on f.farm_id = fi.farm_id group by f.farm_id, f.farm_name;
```

This query counts the number of fields associated with each farm. It selects the farm ID, name, and the count of fields associated with each farm. It uses a left outer join to include farms even if they don't have any associated fields.

☞Searching & Updating Information:

✦Number Of Farms Associated With Farmer:

```
select count(*)  from farmer_farm where farmer_id = farmer_id_val;
```

Counts farms associated with a specific farmer.

✦Search For Field Information:

```
select f.field_id, f.field_name, h.harvest_date from fields f right outer join harvests h on
f.field_id = h.field_id where f.field_name like '%Field%';
```

Retrieves field details based on partial field name search.

✦Max Sale Amount From Harvest:

```
with max_sale_amount as ( select max(sale_amount) as max_sale_amount
from sales) select * from harvests where harvest_id in (select harvest_id from sales
where sale_amount = (select max_sale_amount from max_sale_amount));
```

Finds the maximum sale amount and retrieves associated harvest information.

☞Advanced SQL Command:

✦Harvest Records for Corresponding Crop:

```
declare
        cursor harvest_cursor is
        select h.harvest_id, h.field_id, h.harvest_date, h.yield, c.crop_id, c.crop_name,
        c.crop_type, c.planting_season from Harvests h
        join Crops c on h.crop_id = c.crop_id;

        v_harvest_id Harvests.harvest_id%type;
        v_field_id Harvests.field_id%type;
        v_harvest_date Harvests.harvest_date%type;
        v_yield Harvests.yield%type;
        v_crop_id Crops.crop_id%type;
        v_crop_name Crops.crop_name%type;
        v_crop_type Crops.crop_type%type;
        v_planting_season Crops.planting_season%type;
begin
open harvest_cursor;
```

```
        fetch harvest_cursor into v_harvest_id, v_field_id, v_harvest_date, v_yield, v_crop_id,
        v_crop_name, v_crop_type, v_planting_season;

    while harvest_cursor%FOUND loop

      DBMS_OUTPUT.PUT_LINE('Harvest ID: ' || v_harvest_id);
      DBMS_OUTPUT.PUT_LINE('Field ID: ' || v_field_id);
      DBMS_OUTPUT.PUT_LINE('Harvest Date: ' || TO_CHAR(v_harvest_date, 'YYYY-MM-DD'));
      DBMS_OUTPUT.PUT_LINE('Yield: ' || v_yield);
      DBMS_OUTPUT.PUT_LINE('Crop ID: ' || v_crop_id);
      DBMS_OUTPUT.PUT_LINE('Crop Name: ' || v_crop_name);
      DBMS_OUTPUT.PUT_LINE('Crop Type: ' || v_crop_type);
      DBMS_OUTPUT.PUT_LINE('Planting Season: ' || v_planting_season);
      DBMS_OUTPUT.PUT_LINE('-----------------------------------------');

        FETCH harvest_cursor INTO v_harvest_id, v_field_id, v_harvest_date, v_yield,
        v_crop_id,     v_crop_name, v_crop_type, v_planting_season;
        END LOOP;

    CLOSE harvest_cursor;
end;
/
```

This PL/SQL block retrieves harvest records from the AgroNex database along with corresponding crop details. It begins by declaring a cursor to select harvest information joined with crop details. The cursor is then opened, and data is fetched row by row into variables. For each fetched record, harvest and crop details are displayed using DBMS_OUTPUT.PUT_LINE. Once all records have been processed, the cursor is closed. This block enables efficient retrieval and display of harvest information, aiding in analysis and reporting within the AgroNex system.

✦Update Field Information:

```
        create or replace trigger update_harvests_field_id after update of field_id on fields
        referencing old as o new as n for each row enable begin update harvests
         set field_id = :n.field_id where field_id = :o.field_id;
        end;
        /
        update fields
        set field_id = 6
        where field_id = 1;
```

This trigger, named update_harvests_field_id, ensures that whenever a field's ID is updated in the "fields" table, corresponding records in the "harvests" table are automatically updated to reflect the change. This maintains data consistency between the two related tables in the AgroNex database.

✦Number Of Farmers And Max Sale Amount:

```
create or replace procedure farmers_and_max_sales(farmer_count out number,
max_sale_amount out number) as t_show char(30);
begin
t_show := 'from procedure: ';
select count(*) into farmer_count from farmers;
select max(sale_amount) into max_sale_amount from sales;

dbms_output.put_line(t_show);
  dbms_output.put_line('Total number of farmers: ' || farmer_count);
dbms_output.put_line('Maximum sale amount: ' || max_sale_amount);
end;
/

declare
farmer_count NUMBER;
max_sale_amount NUMBER;
begin
  farmers_and_max_sales(farmer_count, max_sale_amount);
end;
/
```

This PL/SQL procedure, named `farmers_and_max_sales`, retrieves and displays the total number of farmers and the maximum sale amount from the AgroNex database. It uses output parameters to store these values and outputs them using DBMS_OUTPUT.PUT_LINE statements. The procedure is executed within an anonymous PL/SQL block, which calls the procedure and displays the results.

✪Conclusion:

The AgroNex database project offers a robust platform for managing farming activities, analyzing agricultural data, and facilitating decision-making processes. Through its relational database schema and associated functionalities, AgroNex addresses various needs of farmers, farm managers, agricultural advisors, government agencies, suppliers, distributors, and financial institutions within the agricultural sector. However, like any project, AgroNex has its strengths and weaknesses.

✪ Pros:

- ***All Data in One Place:*** *AgroNex keeps all farm-related information organized, making it easy to find and use.*
- ***Helps Make Decisions:*** *AgroNex provides useful reports and insights that can help farmers and others make smart decisions.*
- ***Follows Rules:*** *It helps farmers follow the rules and regulations in agriculture, which is important for legal and ethical reasons.*
- ***Supports Collaboration:*** *AgroNex allows farmers and experts to work together, sharing ideas and advice.*
- ***Saves Time:*** *It automates tasks like data entry and report creation, saving farmers time and effort.*

✪ Cons:

1. ***Hard to Understand:*** *AgroNex can be complicated to use, so farmers may need training to use it properly.*
2. ***Needs Regular Maintenance:*** *It needs to be looked after regularly to keep it running smoothly and to keep data safe.*
3. ***Tough to Connect With Other Systems:*** *Connecting AgroNex with other systems or devices may be tricky.*
4. ***Worries About Security:*** *Keeping farm data safe from hackers and others is a big concern.*
5. ***Might Struggle With Growth:*** *As more farms use AgroNex, it might slow down or have other problems.*

✪ Improvement Ideas:

1. ***Use It Anywhere:*** *Create mobile apps so farmers can use AgroNex on their phones while they're out in the fields.*
2. ***Predict the Future:*** *Add features that can predict things like crop yields or market trends.*
3. ***Connect With Smart Devices:*** *Use devices like sensors to get real-time information about the farm.*

The End