## RELATIONAL DESIGN [figure 1]

## TABLES

**FARM**

This acts as the central entity to the data base; most outer tables refer to this table. This table contains information directly relevant to each instance of a farm. I kept *water_source* in the farm table, I decided creating a separate table and a join is over complex for the use case. Upon expanding however, it may be easier to create a separate table to prevent multiple instances of the same water source across rows.

**CROP_TYPE, CROP_APPLICATION**

I decided to break down the crop data into two tables (CROP_TYPE, CROP_APPLICATION). CROP_APPLICATION contains information for one instance of a crop application. This keeps all information for one instance of a crop in a single table, allowing for the same crop to be applied in multiple instances. CROP_TYPE is a lookup table storing the information for a crop type.

**RESOURCE_APPLICATION, RESOURCE_TYPE**

Similarly, I decided to split the resource data into two tables (RESOURCE_TYPE and RESOURCE_APPLICATION). Each time a resource is used, it is given a *resource_application_id*. This allows us to follow every application of a particular *resource_type*. Where before, the *resource_id* was explicitly representative of only the *resource_type* making *resource_id* redundant information. Previously to understand where each resource was applied, one would have to consider the *resource_type* and the associated *crop_application_id*.

**SOIL**

The soil data is stored as a separate entity, keeping the farm entity less cluttered. Furthermore, it allows for easier modification of the soil data. The primary key for this data type is *farm_id* (foreign key refers to farm(farm_id)). This enforces uniqueness and removes the redundant *soil_id*.

**SUSTAINABILITY_INITIATIVE_APPLICATION, SUSTAINABILITY_INITIATIVE**

Sustainability initiative is broken down into an initiatives sub table and initiative applications. This allows for multiple farms to implement the same initiative if desired and helps remove redundant information from the database.

## RELATIONSHIPS/CARDINALITIES

**SOIL → FARM (one to one)**

I constructed this cardinality based purely on the given data, whereby each farm has exactly one soil profile and each soil profile belongs to exactly one farm. We assume that soil across the farm remains consistent. In this instance, the foreign key could theoretically be placed in either the FARM entity or the SOIL entity. I decided to place *farm_id* as a foreign key in the SOIL entity because the existence of the soil sample depends on the existence of a farm. *soil_id* becomes redundant and was therefore removed.

**FARM → CROP_APPLICATION (one to many**)

Exactly one farm can plant zero, one, or many crops. In the case of the data, each farm plants two crops. The foreign key (*farm_id*) is placed on the many side of the relationship.

**CROP_TYPE → CROP_APPLICATION (one to many)**

Each application must be of exactly one crop type, but one crop type can be planted in many applications. In the data provided, a single crop type is never planted more than once; however, I wanted to build based on the potential for multiple applications of the same crop in the future. The foreign key (*crop_type_id*) is placed on the many side of the relationship.

**SUSTAINABILITY_INITIATIVE_APPLICATION → FARM (many to one)**

A farm may have many applications of sustainability initiatives. In the provided data, this is not the case. I wanted, however, to prepare for future expansion whereby a farm may employ multiple initiatives simultaneously. The foreign key (*farm_id*) is placed on the many side of the relationship.

**SUSTAINABILITY_INITIATIVE → SUSTAINABILITY_INITIATIVE_APPLICATION (one to many)**

I wanted to build the database so that a single initiative may be applied at multiple farms. The foreign key (*initiative_id*) is placed on the many side of the relationship.

**RESOURCE_APPLICATION → CROP_APPLICATION (many to one)**

A crop application may receive multiple resources. The foreign key (*crop_application_id*) is placed on the many side of the relationship.

**RESOURCE_TYPE → RESOURCE_APPLICATION (one to many)**

A type of resource can be applied in multiple scenarios; for example, water is used across multiple crop applications. The foreign key (*resource_type_id*) is placed on the many side of the relationship.

## NORMALISATION – Achieving Third Normal Form

**First Normal Form (1NF)**

The database is constructed so that all tables contain atomic values. An example of where this is achieved: *farm_location* was broken down into *farm_name* and *farm_location*. All tables have a defined primary key and there are no repeating groups.

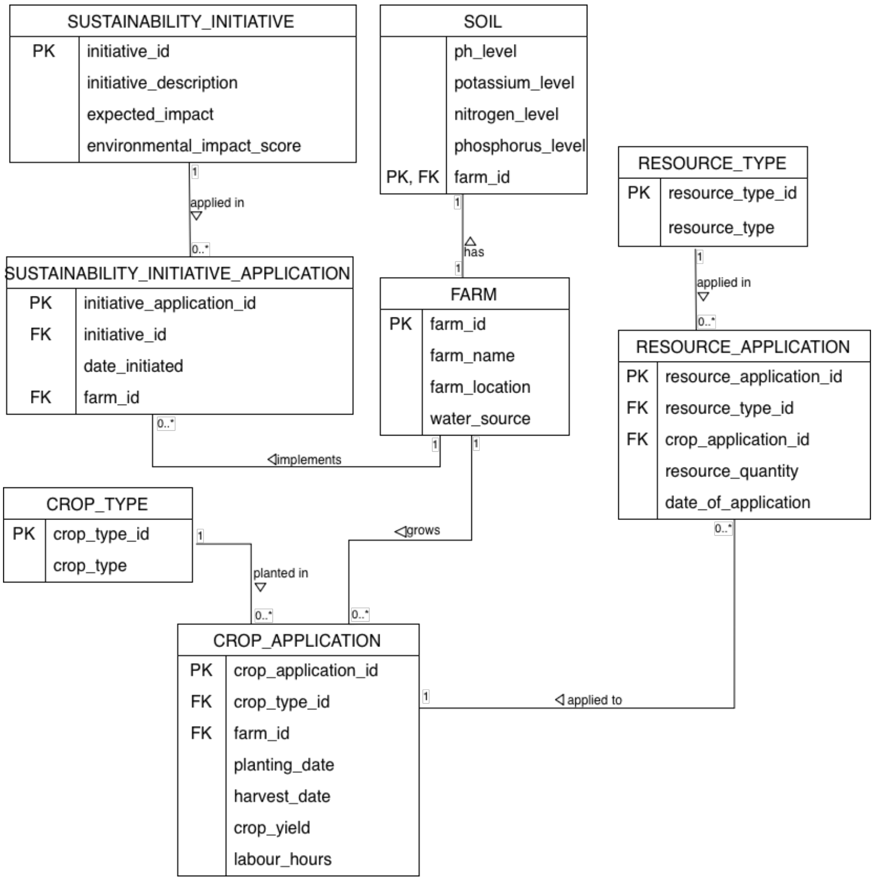
**Second Normal Form (2NF)**

To achieve, second normal form, some partial dependencies were removed. This was mostly achieved through the application/lookup table structure. Take the resources for example. Initially, the attribute resource quantity relied on a composite key consisting of the *resource_id* and the *crop_id.* Moving all attributes to separate lookup tables and giving each individual instance of the resource usage a *resource_application_id* mitigates this issue, with full dependency now on the new primary keys.

**Third Normal Form (3NF)**

Similarly, the lookup/application table structure mitigates the issue of transitive dependencies by storing all descriptive attributes in dedicated lookup tables. For example, if I were to store *initiative_description* and environmental *environmental_impact_score* in the SUSTAINABILITY_INITIATIVE_APPLICATION table, the *environmental_impact_score* could be inferred through the *initiative_description* rather than the primary key.

[Figure 1]



## RESTFUL API DESIGN

The design aims to provide the user with maximum capability whilst also maintaining a level of order. The endpoints allow for retrieval/modification of a collection of resources or specific instances whilst supplying an id. Additional queries may be used for viewing methods such as filtering by an attribute.

## A Note on Verification

When designing the API, the user should also consider access restraints on certain resources. For example, a farmer should not be able to delete information for a farm that he did not create. To solve this, verification such as authentication tokens should be used to make sure users can only perform destructive/editorial operations on resources they own. Authentication tokens are an example of a verification method that could be used whereby the requesting user's identity is checked against the owner of the resource before operations are allowed.

## Error Codes

As well as the response, the user will also receive one of the following error codes that can be used to determine the outcome of the request.

**200 -** Okay
**201** – New resource successfully created
**400** – Client-side error (bad request)
**401 –** Unauthorised (authentication required)
**403 –** Forbidden access (access denied)
**404** – Not Found
**500** – Internal server error

*Below I have listed all the API methods, routes and endpoints:*
*NOTE: PUT methods update all attributes of the resource, PATCH updates only the attribute specified in the body*

### FARM

**GET /farms** – retrieve all farms and their attributes | path parameters: none | query parameters: location (optional, filter by location), water_source (optional, filter by water source) | response: farm_id, farm_name, farm_location, water_source
**GET /farms/{farm_id}** – retrieve specific farm and attributes | path parameters: farm_id (required) | response: farm_id, farm_name, farm_location, water_source
**POST /farms** – create a new farm | body parameters: farm_name (required), farm_location (required), water_source (required) | response: farm_id, farm_name, farm_location, water_source
**PUT/PATCH /farms/{farm_id}** – update a farm | path parameters: farm_id (required) | body parameters: farm_name (optional), farm_location (optional), water_source (optional) | response: farm_id, farm_name, farm_location, water_source
**DELETE /farms/{farm_id}** – delete a farm and associated data | path parameters: farm_id (required) response: none

### SOIL

**GET /farms/{farm_id}/soil** – retrieve soil data for a farm | path parameters: farm_id (required) | response: farm_id, ph_level, nitrogen_level, phosphorus_level, potassium_level
**POST /farms/{farm_id}/soil** – create soil data for a farm | path parameters: farm_id (required) | body parameters: ph_level (optional), potassium_level (optional), nitrogen_level (optional), phosphorus_level (optional) | response: farm_id, ph_level, nitrogen_level, phosphorus_level, potassium_level

**PUT/PATCH /farms/{farm_id}/soil** – update soil data for a farm | path parameters: farm_id (required) | body parameters: ph_level (optional), potassium_level (optional), nitrogen_level (optional), phosphorus_level (optional) | response: farm_id, ph_level, nitrogen_level, phosphorus_level, potassium_level

**DELETE /farms/{farm_id}/soil** – delete soil data for a farm | path parameters: farm_id (required) | response: none

## SUSTAINABILITY_INITIATIVE

**GET /initiatives** – retrieve all sustainability initiatives | path parameters: none | query parameters: environmental_impact_score (optional, filter by score) | response: initiative_id, initiative_description, expected_impact, environmental_impact_score

**GET /initiatives/{initiative_id}** – retrieve specified initiative | path parameters: initiative_id (required) | response: initiative_id, initiative_description, expected_impact, environmental_impact_score

**POST /initiatives** – create a new initiative | body parameters: initiative_description (required), expected_impact (optional) | environmental_impact_score (optional) | response: initiative_id, initiative_description, expected_impact, environmental_impact_score

**PUT/PATCH /initiatives/{initiative_id}** – update specified initiative | path parameters: initiative_id (required) | body parameters: initiative_description (optional), expected_impact (optional), environmental_impact_score (optional) | response: initiative_id, initiative_description, expected_impact, environmental_impact_score

**DELETE /initiatives/{initiative_id}** – delete specified initiative | path parameters: initiative_id (required) | response: none

## SUSTAINABILITY_INITIATIVE_APPLICATION

**GET /initiative_applications** – retrieve all initiative applications | path parameters: none | query parameters: initiative_id (optional, filter by initiative), farm_id (optional, filter by farm) | response: initiative_application_id, initiative_id, farm_id, date_initiated

**GET /initiative_applications/{initiative_application_id}** – retrieve specified application | path parameters: initiative_application_id (required) | response: initiative_application_id, initiative_id, farm_id, date_initiated

**GET /farms/{farm_id}/initiative_applications** – retrieve all initiative applications for a farm | path parameters: farm_id (required) | response: initiative_application_id, initiative_id, farm_id, date_initiated

**GET /initiatives/{initiative_id}/initiative_applications** – retrieve all applications of a specified initiative | path parameters: initiative_id (required) | response: initiative_application_id, initiative_id, farm_id, date_initiated

**POST /farms/{farm_id}/initiative_applications** – create a new initiative application for a farm | path parameters: farm_id (required) | body parameters: initiative_id (required), date_initiated (optional) | response: initiative_application_id, initiative_id, farm_id, date_initiated

**PUT/PATCH /initiative_applications/{initiative_application_id}** – update an initiative application | path parameters: initiative_application_id (required) | body parameters: initiative_id (optional), date_initiated (optional) | response: initiative_application_id, initiative_id, farm_id, date_initiated

**DELETE /initiative_applications/{initiative_application_id}** – delete an initiative application | path parameters: initiative_application_id (required) response: none

## CROP_TYPES

**GET /crop_types** – retrieve all crop types and their ids | path parameters: none | response: crop_type_id, crop_type

**POST /crop_types** – create a new crop type | body parameters: crop_type (required) | response: crop_type_id, crop_type

**DELETE /crop_types/{crop_type_id}** – delete specified crop type | path parameters: crop_type_id (required) | response: none

## CROP_APPLICATION

**GET /crop_applications** – retrieve all crop applications and their attributes | path parameters: none | query parameters: sort_by (optional: planting_date, harvest_date, crop_yield) | response: crop_application_id, crop_type_id, farm_id, planting_date, harvest_date, crop_yield, labour_hours

**GET /crop_applications/{crop_application_id}** – retrieve data for specified crop application | path parameters: crop_application_id (required) | response: crop_application_id, crop_type_id, farm_id, planting_date, harvest_date, crop_yield, labour_hours

**GET /farms/{farm_id}/crop_applications** – retrieve all crop applications for *specified farm* | path parameters: farm_id (required) | query parameters: sort_by (optional: planting_date, harvest_date, crop_yield) | response: crop_application_id, crop_type_id, farm_id, planting_date, harvest_date, crop_yield, labour_hours

**GET /crop_types/{crop_type_id}/crop_applications** – retrieve all applications of a specified crop type | path parameters: crop_type_id (required)| response: crop_application_id, crop_type_id, farm_id, planting_date, harvest_date, crop_yield, labour_hours

**POST /farms/{farm_id}/crop_applications** – create a new crop application | path parameters: farm_id (required) | body parameters: crop_type_id (required), planting_date (required), harvest_date (required), crop_yield (required), labour_hours (required) | response: crop_application_id, crop_type_id, farm_id, planting_date, harvest_date, crop_yield, labour_hours

**PUT/PATCH /crop_applications/{crop_application_id}** – update a crop application | path parameters: crop_application_id (required) | body parameters: crop_type_id (optional), planting_date (optional), harvest_date (optional), crop_yield (optional), labour_hours (optional) | response: crop_application_id, crop_type_id, farm_id, planting_date, harvest_date, crop_yield, labour_hours

**DELETE /crop_applications/{crop_application_id}** – delete a crop application | path parameters: crop_application_id (required) | response: none

## RESOURCE_TYPES

**GET /resource_types** – retrieve all resource types and their id | path parameters: none | response: resource_type_id, resource_type

**POST /resource_types** – create a new resource type | body parameters: resource_type (required) | response: resource_type_id, resource_type

**DELETE /resource_types/{resource_type_id}** – delete specified resource type | path parameters: resource_type_id (required) | response: none

## RESOURCE_APPLICATION

**GET /resource_applications** – retrieve all resource applications and their attributes | path parameters: none | query parameters: sort_by (optional: date_of_application, resource_quantity) | response: resource_application_id, resource_type_id, crop_application_id, resource_quantity, date_of_application

**GET /resource_applications/{resource_application_id}** – retrieve data for specified resource application | path parameters: resource_application_id (required) | response: resource_application_id, resource_type_id, crop_application_id, resource_quantity, date_of_application

**GET /farms/{farm_id}/resource_applications** – retrieve all resource applications for specified farm | path parameters: farm_id (required) | response: resource_application_id, resource_type_id, crop_application_id, resource_quantity, date_of_application

**GET /crop_applications/{crop_application_id}/resource_**applications – retrieve all resource applications for specified crop application | path parameters: crop_application_id (required) | response: resource_application_id, resource_type_id, crop_application_id, resource_quantity, date_of_application

**GET /resource_types/{resource_type_id}/resource_applications** – retrieve all applications of a specified resource type | path parameters: resource_type_id (required) | response: resource_application_id, resource_type_id, crop_application_id, resource_quantity, date_of_application

**POST /crop_applications/{crop_application_id}/resource_applications** – create a new resource application | path parameters: crop_application_id (required) | body parameters: resource_type_id (required), resource_quantity (required), date_of_application (required) | response: resource_application_id, resource_type_id, crop_application_id, resource_quantity, date_of_application

**PUT/PATCH /resource_applications/{resource_application_id}** – update a resource application | path parameters: resource_application_id (required) | body parameters: resource_type_id (optional), resource_quantity (optional), date_of_application (optional) | response: resource_application_id, resource_type_id, crop_application_id, resource_quantity, date_of_application

**DELETE /resource_applications/{resource_application_id}** – delete a resource application | path parameters: resource_application_id (required) | response: none

**DOCUMENT-BASED DESIGN** [Figure 2]

For an alternative "document-based" design, I have chosen to employ MongoDB and its non-relational database structure. The database is to be structured as followed. One MAIN ***farms*** COLLECTION. Each farm is to be stored as a single document. NESTED within farm: ***soil*** – each farm has exactly one soil profile; ***sustainability_initiative_applications*** – an array containing the initiatives the farm has implemented; ***crop_applications*** – an array containing all the crop applications. Nested within *crop_applications* – An array containing all ***resource_applications***. This maintains the hierarchical structure and read pattern of the data.

There will be three *referenced* COLLECTIONs – these store the lookup information that any farm can access. ***crop_types*** – list of all crop types, ***resource_types*** – list of all resource types, ***sustainability_initiatives*** – list of all sustainability initiatives.

In this design, data that is specific to one farm is embedded within a single farm document which is stored in the farm collection, whilst shared resources are stored in referenceable collections.

**SCALABILTY**

SQL traditionally scales vertically. This is achieved through adding more power (CPU, RAM, storage) to a single server. This of course has physical limits and can become expensive. While some relational data bases support horizontal scaling, MongoDB is *designed* to scale horizontally by distributing the data across multiple servers.

**PERFORMANCE**

Retrieving related data in SQL requires multiple joins, each at a computational expense. In the context of our DB, a complete farm profile can require up to 8 joins. MongoDB on the other hand retrieves the core farm data in one query. In our case referential lookups are still required but the overall performance gains are substantial in the document-based design for single farm queries. (more on this later). Writing data on the other hand can be more expensive in MongoDB, where duplicated data must be individually updated across each document as opposed to the relational structure of SQL whereby only one update is usually required.

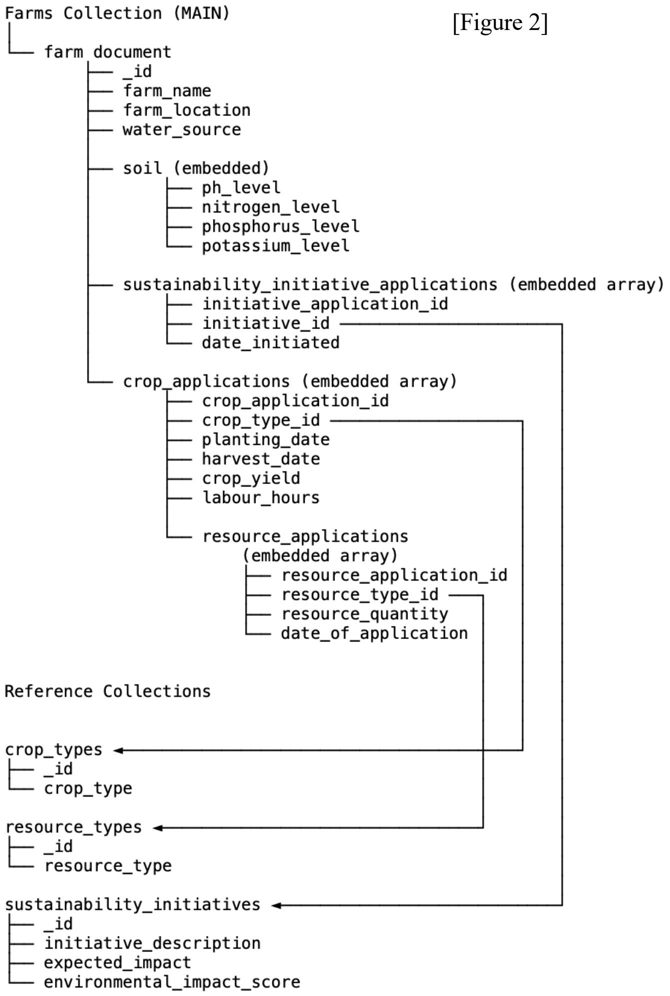
**DATA INTEGRITY**

SQL enforces referential integrity through native foreign keys. This prevents invalid references at the database level by default. For example you cannot delete a crop_type that is still referenced by existing crop_application. MongoDB on the other hand does not enforce this. The application using the DB is responsible for validating references.

**USE CASES**

In the case of the farm data base, MongoDB would be preferred if modifying and viewing individual farm profiles is the main use case. E.g. the main user just wants to view and edit their own farm. The document-based structure allows for easy access to full farm profiles at a low computational cost with minimal joins. Furthermore, MongoDB suits a use case where the number of farm profiles is likely to grow due to its inherent ability to scale horizontally. Adding additional server storage is relatively easy and data can be easily stored across servers.

SQL would be preferred if the application requires complex cross referencing and joins such as statistical analysis. SQLs built in join functions are far superior in this case, handling analytical queries more efficiently and allowing more flexibility for types of joins. (It is far easier to group-by or filter-by) The foreign key constraints also prevent non-existent references ensuring data accuracy is maintained in these complex operations.



[Figure 2]

```
Farms Collection (MAIN)
│
└── farm document
        ├── _id
        ├── farm_name
        ├── farm_location
        ├── water_source
        │
        ├── soil (embedded)
        │       ├── ph_level
        │       ├── nitrogen_level
        │       ├── phosphorus_level
        │       └── potassium_level
        │
        ├── sustainability_initiative_applications (embedded array)
        │       ├── initiative_application_id
        │       ├── initiative_id
        │       └── date_initiated
        │
        └── crop_applications (embedded array)
                ├── crop_application_id
                ├── crop_type_id
                ├── planting_date
                ├── harvest_date
                ├── crop_yield
                ├── labour_hours
                │
                └── resource_applications
                        (embedded array)
                        ├── resource_application_id
                        ├── resource_type_id
                        ├── resource_quantity
                        └── date_of_application

Reference Collections


crop_types
├── _id
└── crop_type

resource_types
├── _id
└── resource_type

sustainability_initiatives
├── _id
├── initiative_description
├── expected_impact
└── environmental_impact_score
```

[Example farm document]

```
{
 "_id": 1,
 "farm_name": "South Farm",
 "farm_location": "Kent",
 "water_source": "River",

 "soil": {
  "ph_level": 6.5,
  "nitrogen_level": 50,
  "phosphorus_level": 20,
  "potassium_level": 180
 },

 "sustainability_initiative_applications": [
  {
   "initiative_application_id": 1,
   "initiative_id": 1,
   "date_initiated": "2023-01-01"
  }
 ],

 "crop_applications": [
  {
   "crop_application_id": 101,
   "crop_type_id": 1,
   "planting_date": "2023-03-15",
   "harvest_date": "2023-08-15",
   "crop_yield": 3000,
   "labour_hours": 150,
   "resource_applications": [
    {
     "resource_application_id": 1,
     "resource_type_id": 1,
     "resource_quantity": 1000.0,
     "date_of_application": "2023-03-10"
    },
    {
     "resource_application_id": 2,
     "resource_type_id": 3,
     "resource_quantity": 360000.0,
     "date_of_application": "2023-05-30"
    }
   ]
  }
 ]
}
}
```