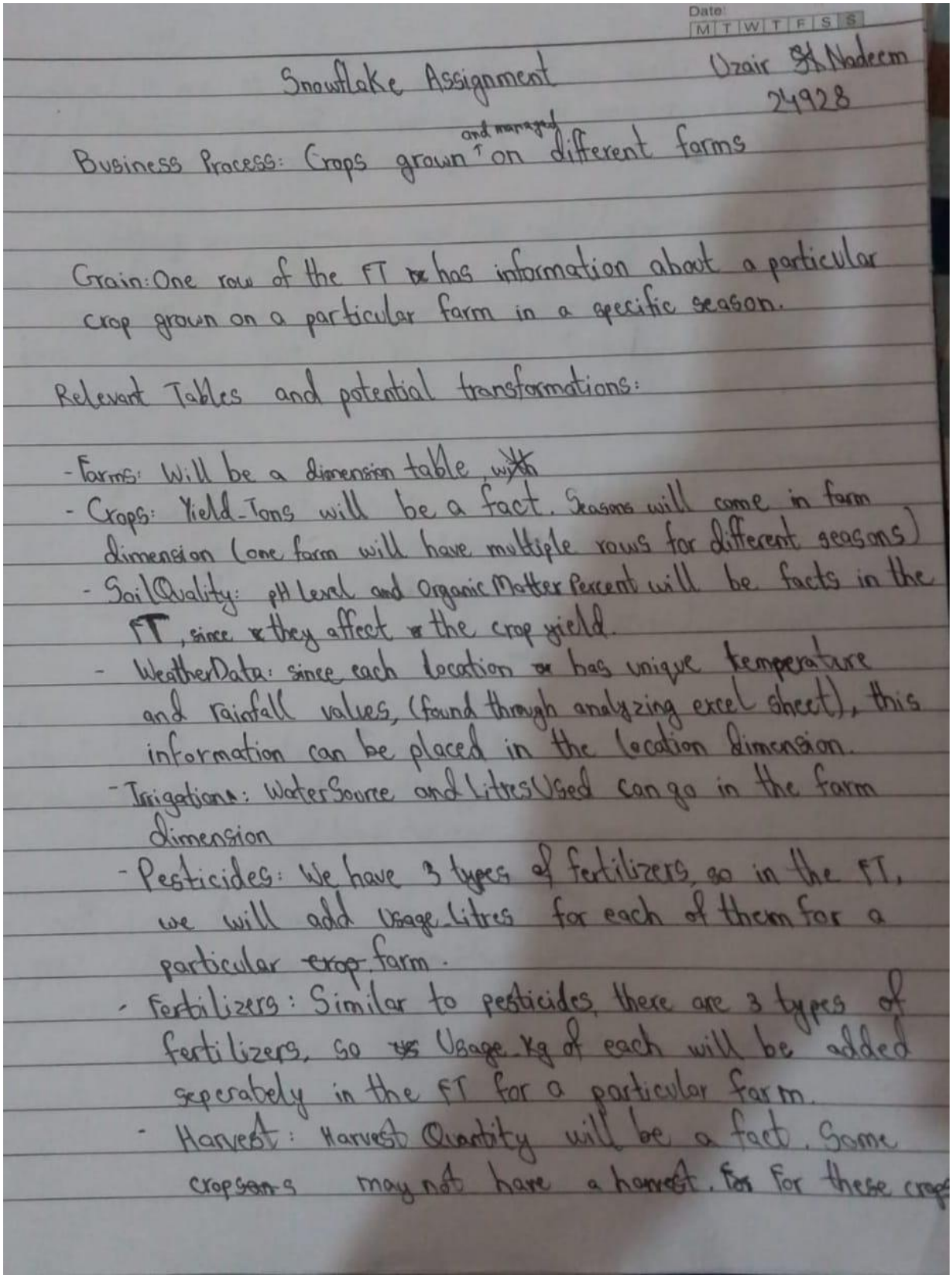
Snowflake Assignment

Uzair Nadeem 24928, Shahmeer Khan 25156

Rough Work on Paper and Star Schema:



Date: M T W T F S S

Snowflake Assignment — Uzair Nadeem 24928

Business Process: Crops grown and managed on different farms

Grain: One row of the FT has information about a particular crop grown on a particular farm in a specific season.

Relevant Tables and potential transformations:

- Farms: Will be a dimension table, with
- Crops: Yield_Tons will be a fact. Seasons will come in farm dimension (one farm will have multiple rows for different seasons)
- SoilQuality: pH Level and Organic Matter Percent will be facts in the FT, since they affect the crop yield.
- WeatherData: since each location has unique temperature and rainfall values, (found through analyzing excel sheet), this information can be placed in the location dimension.
- Irrigations: WaterSource and LitresUsed can go in the farm dimension
- Pesticides: We have 3 types of fertilizers, so in the FT, we will add Usage_Litres for each of them for a particular crop farm.
- Fertilizers: Similar to pesticides, there are 3 types of fertilizers, so Usage_Kg of each will be added seperately in the FT for a particular farm.
- Harvest: HarvestQuantity will be a fact. Some crop seasons may not have a harvest. for For these crop

Date: M T W T F S S

we can either use the yield quantity, or use 0 to indicate missing values. ~~or divide~~ The same might be the case vice-versa (crops can have harvest, but no yield).

- Crop dimension table: Will have name, average market price (will be derived from market price), and best growing season (will be derived from crop yield - the season with the highest yield for the crop will be placed here).

Dimension Tables:

- ~~FarmDim~~ DimFarms (name, size_acres, WaterSources, LitresUsed, season)
- DimCrops (name, ~~avg~~ avg_market_price, best_growing_season)
- DimLocation (name, temperature_C, Rainfall_mm, Condition)
- DimDate (dateID, Year, Month, day)
  winter

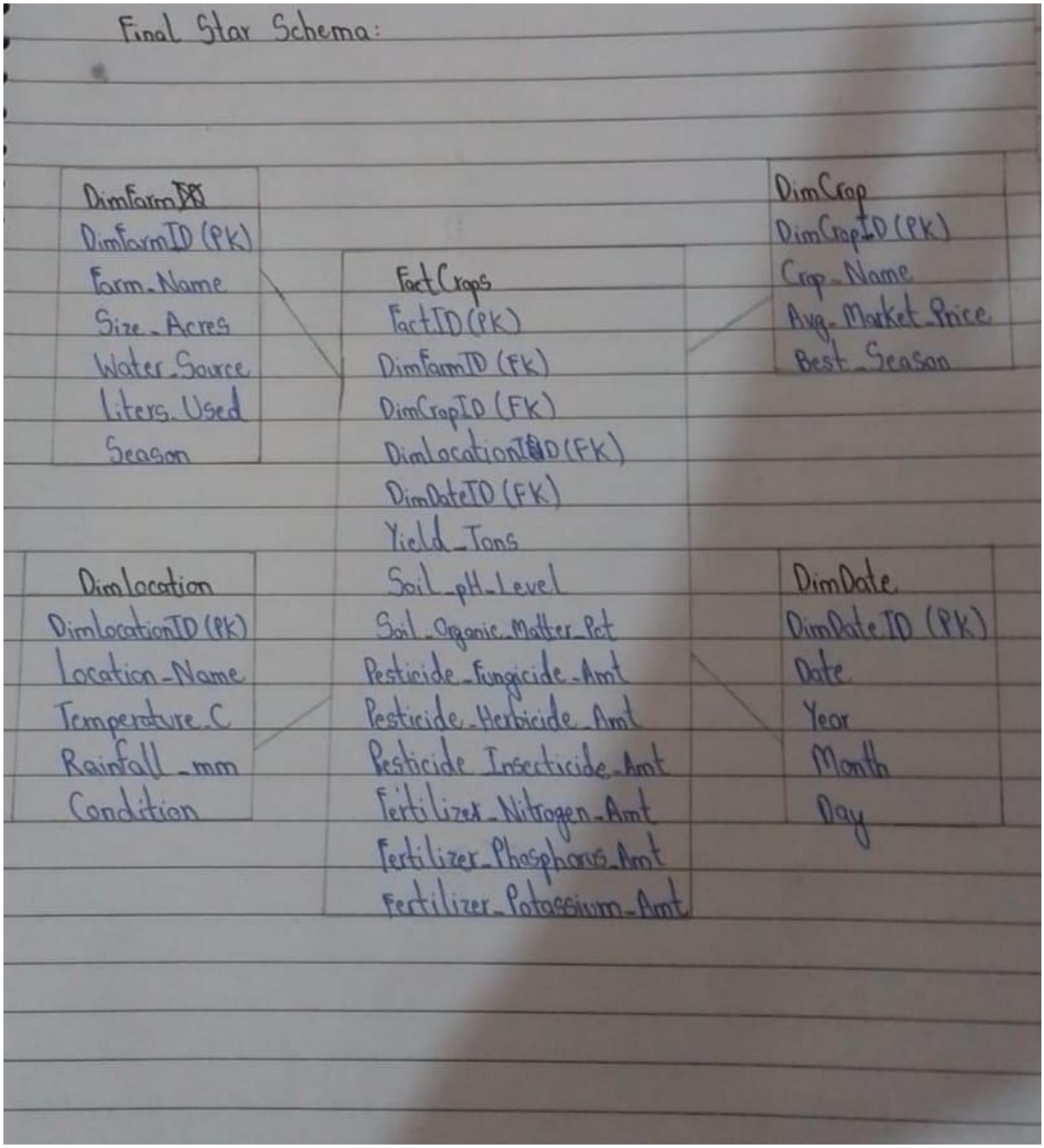
Facts:

Yield_tons, pH_Level, OrganicMatter_Percent, Fungicide_usage_litres, herbicide_usage_litres, Insecticide_usage_litres, Nitrogen_usage_kg, Potassium_usage_kg, Phosphorous_usage_kg, Harvest_Quantity_tons

Note:

- Harvest Quantity will not be considered, since the crop and farm combinations for yield in the crop table are all different from the crop and farm combinations ~~from the~~ in the harvest table.

Final Star Schema:



Major Python Functions and Scripts:

Data Ingestion:

```
import pandas as pd;
xls = pd.ExcelFile("AgricultureData.xlsx")
farm_df = xls.parse("Farms")
crop_df = xls.parse("Crops")
soilquality_df = xls.parse("SoilQuality")
irrigation_df = xls.parse("Irrigation")
pesticides_df = xls.parse("Pesticides")
harvest_df = xls.parse("Harvest")
fertilizers_df = xls.parse("Fertilizers")
marketPrice_df = xls.parse("MarketPrices")
weather_df = xls.parse("WeatherData")
```

Preparing Dimension Tables:

Farm Dimension:

```
dimFarms_df = farm_df[['FarmID', 'Name', 'Size_Acres']].copy()
dimFarms_df = dimFarms_df.merge(irrigation_df[['FarmID', 'WaterSource', 'LitersUsed']], on='FarmID', how='left')
dimFarms_df.head()
```

	FarmID	Name	Size_Acres	WaterSource	LitersUsed
0	1	Young LLC	76	River	3036.0
1	2	Owens Group	144	Well	37545.0
2	3	Davis, Taylor and Vasquez	259	NaN	NaN
3	4	Torres LLC	384	Irrigation Canal	37485.0
4	4	Torres LLC	384	River	1496.0

```
dimFarms_df.isnull().sum()
```

```
FarmID          0
Name            0
Size_Acres      0
WaterSource    37
LitersUsed     37
dtype: int64
```

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
season_df = pd.DataFrame({'Season': seasons})
dimFarms_df_expanded = dimFarms_df.merge(season_df, how='cross')
dimFarms_df_expanded.reset_index(drop=True, inplace=True)
dimFarms_df_expanded['Farm_Season_ID'] = dimFarms_df_expanded.index + 1
```

In the last cell, the farm dimension table dataframe was expanded to have four rows for each farm, so that each farm has a unique row with every season of the year (spring, summer, fall, and winter). Each row of this dimension table is considered a unique farm (The same farm name with different season labels is a unique data item/farm).

Location Dimension:

Location Dimension

```
dimLocation_df = weather_df.copy()
dimLocation_df = dimLocation_df.rename(columns={'WeatherID': 'LocationID'})
dimLocation_df = dimLocation_df.rename(columns={'Location': 'LocationName'})
dimLocation_df.head()
```

	LocationID	LocationName	Temperature_C	Rainfall_mm	Condition	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	1	Stewartland	29.5	77.8	Cloudy	Unique	NaN	NaN	NaN
1	2	North Brian	32.0	151.7	Cloudy	Unique	NaN	NaN	NaN
2	3	Stevensfort	23.6	156.1	Stormy	Unique	NaN	NaN	NaN
3	4	Michaelhaven	15.7	182.2	Cloudy	Unique	NaN	NaN	NaN
4	5	Odomshire	12.4	125.1	Sunny	Unique	NaN	NaN	NaN

```
#for dim location
dimLocation_df[['LocationName']] = farm_df[['Location']]
dimLocation_df.head()
```

	LocationID	LocationName	Temperature_C	Rainfall_mm	Condition	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	1	Lake Josephborough	29.5	77.8	Cloudy	Unique	NaN	NaN	NaN
1	2	Matthewstad	32.0	151.7	Cloudy	Unique	NaN	NaN	NaN
2	3	North Jeffshire	23.6	156.1	Stormy	Unique	NaN	NaN	NaN
3	4	New John	15.7	182.2	Cloudy	Unique	NaN	NaN	NaN
4	5	Thomasmouth	12.4	125.1	Sunny	Unique	NaN	NaN	NaN

For the location dimension, we first used the WeatherData table to get temperature, rainfall, and condition columns for each location. However, when the locations in the table were explored, it was found that there was almost no overlap between the farm locations in the farms table and the locations in the WeatherData table. An assumption was made here that the temperature and rainfall data in WeatherData was for the farm locations, and the Location column from the farms table was copied into the dimension table.

Date Dimension:

```python
import random
from datetime import datetime, timedelta

def generate_random_2025_date(season):
    """
    Generate a random date in 2025 based on the given season.
    """
    year = 2025
    season = season.lower()

    if season == 'spring':
        start = datetime(year, 3, 1)
        end = datetime(year, 5, 31)
    elif season == 'summer':
        start = datetime(year, 6, 1)
        end = datetime(year, 8, 31)
    elif season in ['autumn', 'fall']:
        start = datetime(year, 9, 1)
        end = datetime(year, 11, 30)
    elif season == 'winter':
        start = datetime(year, 12, 1)
        end = datetime(year, 12, 31)
    else:
        start = datetime(year, 1, 1)
        end = datetime(year, 12, 31)

    delta = end - start
    random_days = random.randint(0, delta.days)
    return start + timedelta(days=random_days)

# Apply to create a new 'SeasonDate' column in dimFarms_df
dimFarms_df['Date'] = dimFarms_df['Season'].apply(generate_random_2025_date)
```

```python
dimDate_df['Date'] = dimFarms_df['Date']
dimDate_df.head()
```

	DateID	Date	Year	Month	Day
0	NaN	2025-05-06	NaN	NaN	NaN
1	NaN	2025-07-24	NaN	NaN	NaN
2	NaN	2025-11-27	NaN	NaN	NaN
3	NaN	2025-12-16	NaN	NaN	NaN
4	NaN	2025-05-13	NaN	NaN	NaN

```python
dimDate_df.drop_duplicates(subset='Date', inplace=True)
```

```python
dimDate_df = dimDate_df.reset_index(drop=True)  # Ensure a clean index
dimDate_df['DateID'] = range(1, len(dimDate_df) + 1)
dimDate_df.head()
```

	DateID	Date	Year	Month	Day
0	1	2025-05-06	NaN	NaN	NaN
1	2	2025-07-24	NaN	NaN	NaN
2	3	2025-11-27	NaN	NaN	NaN
3	4	2025-12-16	NaN	NaN	NaN
4	5	2025-05-13	NaN	NaN	NaN

```python
def enrich_dim_date(dimDate_df, date_col='Date'):
    dimDate_df[date_col] = pd.to_datetime(dimDate_df[date_col])
    dimDate_df['Year'] = dimDate_df[date_col].dt.year
    dimDate_df['Month'] = dimDate_df[date_col].dt.month
    dimDate_df['Day'] = dimDate_df[date_col].dt.day

    return dimDate_df

dimDate_df = enrich_dim_date(dimDate_df)
dimDate_df
```

	DateID	Date	Year	Month	Day
0	1	2025-05-06	2025	5	6
1	2	2025-07-24	2025	7	24
2	3	2025-11-27	2025	11	27
3	4	2025-12-16	2025	12	16

Since no date data related to yield was given (harvest date was given, but we did not consider harvest date due to there being no matches between farm and crop ids in the harvest and yield tables), random dates were generated for each row in the farm dimension. These were copied into the dataframe of the date dimension, and duplicates were removed to ensure that each date in the dimension becomes unique. Each date in the table was then given a unique id.

Crop Dimension:

```python
unique_crops = crop_df[['Name']].drop_duplicates().reset_index(drop=True)
unique_crops['CropID'] = unique_crops.index + 1
dimCrop_df = unique_crops[['CropID', 'Name']]
dimCrop_df
```

	CropID	Name
0	1	Barley
1	2	Soybean
2	3	Rice
3	4	Corn
4	5	Wheat
5	6	Sugarcane

```python
avg_prices = marketPrice_df.groupby('CropName')['PricePerTon'].mean().reset_index()
avg_prices = avg_prices.rename(columns={'PricePerTon': 'Avg_Market_Price'})
dimCrop_df = dimCrop_df.merge(avg_prices, left_on='Name', right_on='CropName', how='left')
dimCrop_df
```

	CropID	Name	CropName	Avg_Market_Price
0	1	Barley	NaN	NaN
1	2	Soybean	Soybean	286.059000

```
dimCrop_df['Avg_Market_Price'] = dimCrop_df['Avg_Market_Price'].round(2)
dimCrop_df = dimCrop_df.drop(columns=['CropName'])
dimCrop_df
```

	CropID	Name	Avg_Market_Price
0	1	Barley	NaN
1	2	Soybean	286.06
2	3	Rice	275.69
3	4	Corn	268.25
4	5	Wheat	270.55
5	6	Sugarcane	NaN

```
season_yield = crop_df.groupby(['Name', 'Season'])['Yield_Tons'].sum().reset_index()
best_season = season_yield.loc[season_yield.groupby('Name')['Yield_Tons'].idxmax()]
dimCrop_df = dimCrop_df.merge(best_season[['Name', 'Season']], on='Name', how='left')
dimCrop_df.rename(columns={'Season': 'best_growing_season1'}, inplace=True)
dimCrop_df
```

	CropID	Name	Avg_Market_Price	best_growing_season1
0	1	Barley	NaN	Winter
1	2	Soybean	286.06	Summer
2	3	Rice	275.69	Spring

For the crop dimension, each unique crop was extracted from the crops table, and given a unique id in the dimension table. Then, the Market Prices table was used to calculate the average market price of each crop, which was added to the dimension table. Finally, the best season for growing the crops was found (based on the total yield for each crop in each season) and added to the dimension table dataframe.

Handling Missing Values in Dimension Tables:

After dataframes for each dimension were ready, missing values were filled using the mean for numerical columns and the mode categorical columns in each dataframe. The function for filling missing values with the mean value of the column is shown below.

```
def fill_missing_with_mean(df, column_name):
    mean_value = df[column_name].mean()
    df[column_name] = df[column_name].fillna(mean_value)
    return df

dimCrop_df = fill_missing_with_mean(dimCrop_df, 'Avg_Market_Price')
dimFarms_df_expanded = fill_missing_with_mean(dimFarms_df_expanded,'LitersUsed')
```

Creating the Fact Table:

The next step was to create a dataframe for the fact table. First, data from the farm and crop dimensions was copied in order to add the primary keys of these dimensions to the fact table based on the original keys of the crops and farm tables (So that the fact table is populated with with different combinations of crops and farms).

```
fact_df = crop_df
```

```
fact_df = fact_df.merge(
    dimFarms_df[['FarmID', 'Season', 'Farm_Season_ID']],
    on=['FarmID', 'Season'],
    how='left'
)
fact_df.head()
```

	CropID	Name	FarmID	Season	Yield_Tons	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Farm_Season_ID
0	1	Barley	67	Winter	35.08	NaN	NaN	NaN	NaN	NaN	384
1	2	Soybean	74	Fall	40.46	NaN	NaN	NaN	NaN	NaN	427
2	3	Soybean	5	Winter	29.02	NaN	NaN	NaN	NaN	NaN	24
3	4	Barley	31	Spring	3.06	NaN	NaN	NaN	NaN	NaN	189
4	5	Rice	60	Spring	29.17	NaN	NaN	NaN	NaN	NaN	337

```
fact_df = fact_df.merge(
    dimCrop_df[['Name', 'CropID']].rename(columns={'CropID': 'dimCropID'}),
    on='Name',
    how='left'
)
fact_df.head()
```

	CropID	Name	FarmID	Season	Yield_Tons	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Farm_Season_ID	dimCropID
0	1	Barley	67	Winter	35.08	NaN	NaN	NaN	NaN	NaN	384	1
1	2	Soybean	74	Fall	40.46	NaN	NaN	NaN	NaN	NaN	427	2
2	3	Soybean	5	Winter	29.02	NaN	NaN	NaN	NaN	NaN	24	2
3	4	Barley	31	Spring	3.06	NaN	NaN	NaN	NaN	NaN	189	1
4	5	Rice	60	Spring	29.17	NaN	NaN	NaN	NaN	NaN	337	3

Next, data from the Soil Quality table was brought into the fact table based on the original farm IDs (which was in the fact table, and removed after the relevant data was fetched into it).

```
# Step 1: Select and rename relevant columns from soil_quality_df
soil_subset = soilquality_df[['FarmID', 'pH_Level', 'OrganicMatter_Percent']].rename(
    columns={
        'pH_Level': 'ph_level',
        'OrganicMatter_Percent': 'organic_matter'
    }
)

# Step 2: Merge with fact_df on FarmID
fact_df = fact_df.merge(
    soil_subset,
    on='FarmID',
    how='left'
)
fact_df.head()
```

	CropID	FarmID	Yield_Tons	Farm_Season_ID	dimCropID	ph_level	organic_matter
0	1	67	35.08	384	1	4.8	3.9
1	2	74	40.46	427	2	NaN	NaN
2	3	5	29.02	24	2	NaN	NaN
3	4	31	3.06	189	1	6.5	1.2
4	4	31	3.06	189	1	5.4	1.5

After the soil quality data, pesticide and fertilizer data (amounts used) was required. Since there were 3 types of pesticides and 3 types of fertilizers, and a single farm could have multiple types of pesticides or fertilizers used on it, a function was created to make and fill columns for each different type for each different farm.

```python
def pivot_usage_table(fact_df, usage_df, value_col, category_col, farm_col='FarmID', prefix=''):
    """
    Pivot a usage table (like pesticide or fertilizer) to wide format and merge into fact_df.

    Parameters:
    - fact_df (pd.DataFrame): The fact table to merge into.
    - usage_df (pd.DataFrame): The table with usage data (e.g. pesticides, fertilizers).
    - value_col (str): Name of the column containing the usage amount (e.g. 'Usage_Liters').
    - category_col (str): Name of the column containing the type (e.g. 'Type').
    - farm_col (str): Name of the column for farm ID (default is 'FarmID').
    - prefix (str): Prefix to add to the resulting columns (e.g. 'pesticide', 'fertilizer').

    Returns:
    - pd.DataFrame: The fact_df with new columns merged in.
    """
    # Group and pivot
    usage_pivot = (
        usage_df.groupby([farm_col, category_col])[value_col]
        .sum()
        .unstack(fill_value=0)
        .reset_index()
    )

    # Optional prefixing of the new columns
    if prefix:
        usage_pivot = usage_pivot.rename(columns={
            col: f"{prefix}_{col.lower()}" for col in usage_pivot.columns if col != farm_col
        })

    # Merge into fact_df
    fact_df = fact_df.merge(usage_pivot, on=farm_col, how='left')

    return fact_df
```

This function was then applied to the Type columns of the pesticides and fertilizers tables:

```python
fact_df = pivot_usage_table(
    fact_df,
    pesticides_df,
    value_col='Usage_Liters',
    category_col='Type',
    farm_col='FarmID',
    prefix='pesticide'
)
fact_df.head()
```

```python
fact_df = pivot_usage_table(
    fact_df,
    fertilizers_df,
    value_col='Usage_Kg',
    category_col='Type',
    farm_col='FarmID',
    prefix='fertilizer'
)
fact_df.head()
```

Then, each date assigned to farms in the farm dimension table was brought into the fact table in a “Date” column, which was used to fetch a unique DateID from the date dimension table.

```python
fact_df = fact_df.merge(
    dimFarms_df[['Farm_Season_ID', 'Date']],
    on='Farm_Season_ID',
    how='left'
)
fact_df.head()
```

Python

dimCropID	ph_level	organic_matter	pesticide_fungicide	pesticide_herbicide	pesticide_insecticide	fertilizer_nitrogen	fertilizer_phosphorus	fertilizer_potassium	Date
1	4.80	3.90	0.0	0.00	0.00	49.89	0.0	0.0	2025-12-07
2	6.67	2.73	0.0	15.41	0.00	0.00	0.0	0.0	2025-09-23
2	6.67	2.73	0.0	0.00	11.63	0.00	0.0	0.0	2025-12-22

```python
fact_df = fact_df.merge(
    dimDate_df[['Date', 'DateID']],  # Select 'Date' and 'DateID' from dimDate_df
    on='Date',                       # Merge on the 'Date' column in both dataframes
    how='left'
)
fact_df.head()
```

Python

plD	ph_level	organic_matter	pesticide_fungicide	pesticide_herbicide	pesticide_insecticide	fertilizer_nitrogen	fertilizer_phosphorus	fertilizer_potassium	Date	DateID
1	4.80	3.90	0.0	0.00	0.00	49.89	0.0	0.0	2025-12-07	28
2	6.67	2.73	0.0	15.41	0.00	0.00	0.0	0.0	2025-09-23	101
2	6.67	2.73	0.0	0.00	11.63	0.00	0.0	0.0	2025-12-22	24
1	6.50	1.20	0.0	0.00	0.00	0.00	0.0	0.0	2025-05-06	1
1	5.40	1.50	0.0	0.00	0.00	0.00	0.0	0.0	2025-05-06	1

Next, the location column in the farm dimension table was brought into the fact table based on the original farm IDs. This was then used to fetch location IDs from the location dimension table.

```python
# Merge fact_df with farm_df on FarmID to bring in the 'Name' column
fact_df = fact_df.merge(farm_df[['FarmID', 'Location']], on='FarmID', how='left')
fact_df.head()
```

Python

h_level	organic_matter	pesticide_fungicide	pesticide_herbicide	pesticide_insecticide	fertilizer_nitrogen	fertilizer_phosphorus	fertilizer_potassium	DateID	Location
4.80	3.90	0.0	0.00	0.00	49.89	0.0	0.0	28	Fryehaven
6.67	2.73	0.0	15.41	0.00	0.00	0.0	0.0	101	East Amandaton
6.67	2.73	0.0	0.00	11.63	0.00	0.0	0.0	24	Thomasmouth

```python
# Merge fact_df with dimLocation based on matching 'Location' and 'LocationName'
fact_df = fact_df.merge(
    dimLocation_df[['LocationID', 'LocationName']],
    left_on='Location',
    right_on='LocationName',
    how='left'
)
fact_df.head()

# Optional: Drop the 'LocationName' column after the merge if it's no longer needed
#fact_df.drop(columns=['LocationName'], inplace=True)
```

Python

ticide_fungicide	pesticide_herbicide	pesticide_insecticide	fertilizer_nitrogen	fertilizer_phosphorus	fertilizer_potassium	DateID	Location	LocationID	LocationName
0.0	0.00	0.00	49.89	0.0	0.0	28	Fryehaven	67	Fryehaven
0.0	15.41	0.00	0.00	0.0	0.0	101	East Amandaton	74	East Amandaton
0.0	0.00	11.63	0.00	0.0	0.0	24	Thomasmouth	5	Thomasmouth
0.0	0.00	0.00	0.00	0.0	0.0	1	Tracyborough	31	Tracyborough
0.0	0.00	0.00	0.00	0.0	0.0	1	Tracyborough	31	Tracyborough

```python
fact_df.drop(columns=['LocationName'], inplace=True)
```

With this step, all the relevant foreign keys from the dimension tables and all facts that were required were now stored in the fact table.

Preparing All Dataframes for Loading Into the Warehouse:

After the fact table was complete, each table's dataframe was restructured and columns were renamed and reordered so that they could be loaded into the Snowflake cloud warehouse. This process is not shown, but can be seen the python note book named AgricultureETL.ipynb submitted along with this document.

Exporting All Dataframes to Excel Files:

Each dataframe was exported into a separate excel file which was later loaded into the Snowflake data warehouse.

```python
with pd.ExcelWriter('DimFarms.xlsx', engine='openpyxl') as writer:
    dimFarms_df.to_excel(writer, sheet_name='DimFarms', index=False)

with pd.ExcelWriter('DimCrops.xlsx', engine='openpyxl') as writer:
    dimCrop_df.to_excel(writer, sheet_name='DimCrops', index=False)

with pd.ExcelWriter('DimLocation.xlsx', engine='openpyxl') as writer:
    dimLocation_df.to_excel(writer, sheet_name='DimLocation', index=False)

with pd.ExcelWriter('DimDate.xlsx', engine='openpyxl') as writer:
    dimDate_df.to_excel(writer, sheet_name='DimDate', index=False)

with pd.ExcelWriter('Fact.xlsx', engine='openpyxl') as writer:
    fact_df.to_excel(writer, sheet_name='Fact', index=False)
```

Creating a Warehouse in Snowflake:

The first step to create a cloud data warehouse was to go to the warehouses section in snowflake and create a warehouse, then go to the databases section and create a database, and then creating a schema within the database. After the data warehouse, database, and schema were created, the next step was to use these and start creating tables for the star schema.

```
USE WAREHOUSE CROPANALYSIS_DWH;
USE DATABASE CROPANALYSISDWH;
USE SCHEMA STARSCHEMA;

CREATE OR REPLACE TABLE DimFarm (
    DimFarmID INT PRIMARY KEY,
    Farm_Name STRING,
    Size_Acres FLOAT,
    Water_Source STRING,
    Liters_Used FLOAT,
    Season STRING
);
SHOW TABLES

CREATE OR REPLACE TABLE DimCrop (
    DimCropID INT PRIMARY KEY,
    Crop_Name STRING,
    Avg_Market_Price FLOAT,
    Best_Season STRING
);

SHOW TABLES

CREATE OR REPLACE TABLE DimDate (
    DimDateID INT PRIMARY KEY,
    Date DATE,
    Year INT,
    Month INT,
    Day INT
);

CREATE OR REPLACE TABLE DimLocation (
    DimLocationID INT PRIMARY KEY,
    Location_Name STRING,
    Temperature_C FLOAT,
    Rainfall_mm FLOAT,
    Condition STRING
);

SHOW TABLES
```

```sql
CREATE OR REPLACE TABLE FactCrop (
    FactID INT PRIMARY KEY,
    DimFarmID INT,
    DimCropID INT,
    DimLocationID INT,
    DimDateID INT,
    Yield_Tons FLOAT,
    Soil_pH_Level FLOAT,
    Soil_Organic_Matter_Pct FLOAT,
    Pesticide_Fungicide_Amt FLOAT,
    Pesticide_Herbicide_Amt FLOAT,
    Pesticide_Insecticide_Amt FLOAT,
    Fertilizer_Nitrogen_Amt FLOAT,
    Fertilizer_Phosphorus_Amt FLOAT,
    Fertilizer_Potassium_Amt FLOAT,

    -- Foreign Key Constraints
    FOREIGN KEY (DimFarmID) REFERENCES DimFarm(DimFarmID),
    FOREIGN KEY (DimCropID) REFERENCES DimCrop(DimCropID),
    FOREIGN KEY (DimLocationID) REFERENCES DimLocation(DimLocationID),
    FOREIGN KEY (DimDateID) REFERENCES DimDate(DimDateID)
);
```

Loading Data into Snowflake Star Schema:

After all tables were created in Snowflake, the excel files exported earlier containing data for our dimensions and fact table were read into dataframes, a connection was made to the snowflake database, and the dataframes were loaded into their corresponding tables in the database. This was done in a separate notebook on Google Colab.

```
[ ] import pandas as pd

    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] file_crop = '/content/drive/My Drive/Snowflake/DimCrops.xlsx'
    file_farm = '/content/drive/My Drive/Snowflake/DimFarms.xlsx'
    file_date = '/content/drive/My Drive/Snowflake/DimDate.xlsx'
    file_location = '/content/drive/My Drive/Snowflake/DimLocation.xlsx'
    file_fact = '/content/drive/My Drive/Snowflake/Fact.xlsx'
```

```
[ ] dimCrop_xl = pd.ExcelFile(file_crop)
    dimFarm_xl = pd.ExcelFile(file_farm)
    dimDate_xl = pd.ExcelFile(file_date)
    dimLocation_xl = pd.ExcelFile(file_location)
    fact_xl = pd.ExcelFile(file_fact)
```

```
[ ] dimCrop_df = dimCrop_xl.parse('DimCrops')
    dimFarm_df = dimFarm_xl.parse('DimFarms')
    dimDate_df = dimDate_xl.parse('DimDate')
    dimLocation_df = dimLocation_xl.parse('DimLocation')
    fact_df = fact_xl.parse('Fact')
```

Before connecting to the snowflake database, the following command was run in the terminal to install the Snowflake connector:

```
[ ] !pip install snowflake-connector-python pandas snowflake-sqlalchemy
```

```
Collecting snowflake-connector-python
  Downloading snowflake_connector_python-3.14.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (67 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.8/67.8 kB 1.9 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Collecting snowflake-sqlalchemy
  Downloading snowflake_sqlalchemy-1.7.3-py3-none-any.whl.metadata (28 kB)
Collecting asn1crypto<2.0.0,>0.24.0 (from snowflake-connector-python)
  Downloading asn1crypto-1.5.1-py2.py3-none-any.whl.metadata (13 kB)
```

Then, a connection was made and each dataframe was loaded into the database:

```
[ ] from sqlalchemy import create_engine

    # Define the Snowflake connection string
    user = 'UZAIRNADEEM12'
    password = 'NADEEMuzair12!'
    account = 'pp60323.me-central2.gcp'
    warehouse = 'CROPANALYSIS_DWH'
    database = 'CROPANALYSISDWH'
    schema = 'STARSCHEMA'

    # Create the Snowflake connection string (SQLAlchemy format)
    connection_string = f'snowflake://{user}:{password}@{account}/{database}/{schema}?warehouse={warehouse}'

    # Create the SQLAlchemy engine
    engine = create_engine(connection_string)
```

```
dimCrop_df.to_sql('DimCrop', con=engine, if_exists='append', index=False)
```

```
6
```

```
dimFarm_df.to_sql('DimFarm', con=engine, if_exists='append', index=False)
dimDate_df.to_sql('DimDate', con=engine, if_exists='append', index=False)
dimLocation_df.to_sql('DimLocation', con=engine, if_exists='append', index=False)
```

```
fact_df.to_sql('FactCrop', con=engine, if_exists='append', index=False)
```

Query Execution in Snowflake:

After data was successfully loaded into tables in the schema, some dimensional queries were run on the warehouse.

```sql
-- 1. Which crop in South Sarah had the highest yield, and in which season was this yield produced
SELECT
    dc."Crop_Name",
    df."Season",
    fc."Yield_Tons"
FROM "FactCrop" fc
JOIN "DimCrop" dc ON fc."DimCropID" = dc."DimCropID"
JOIN "DimFarm" df ON fc."DimFarmID" = df."DimFarmID"
JOIN "DimLocation" dl ON fc."DimLocationID" = dl."DimLocationID"
WHERE dl."Location_Name" = 'South Sarah'
ORDER BY fc."Yield_Tons" DESC
LIMIT 1;
```

Results | Chart

Crop_Name	Season	Yield_Tons
Corn	Fall	25.25

```sql
-- 2. Average soil pH level for wheat in winter
SELECT AVG(fc."Soil_pH_Level") AS "Avg_pH_Level"
FROM "FactCrop" fc
JOIN "DimCrop" dc ON fc."DimCropID" = dc."DimCropID"
JOIN "DimFarm" df ON fc."DimFarmID" = df."DimFarmID"
WHERE dc."Crop_Name" = 'Wheat'
  AND df."Season" = 'Winter';
```

ults | Chart

Avg_pH_Level
6.377

```sql
-- 3. What is the best crop to grow in each month based on average yield?
SELECT
    dd."Month",
    dc."Crop_Name",
    ROUND(AVG(fc."Yield_Tons"), 2) AS Avg_Yield
FROM "FactCrop" fc
JOIN "DimCrop" dc ON fc."DimCropID" = dc."DimCropID"
JOIN "DimDate" dd ON fc."DimDateID" = dd."DimDateID"
GROUP BY dd."Month", dc."Crop_Name"
QUALIFY ROW_NUMBER() OVER (PARTITION BY dd."Month" ORDER BY AVG(fc."Yield_Tons") DESC) = 1
ORDER BY dd."Month";
```

sults | Chart

Month	Crop_Name	AVG_YIELD
3	Rice	26.28
4	Rice	39.63
5	Sugarcane	34.05

```sql
--4. What are the best farms for growing soybean based on total yield, and on average, how many litres of water do they require
--for irrigation? Also display farm size
SELECT
    f."Farm_Name",
    f."Size_Acres",
    AVG(f."Liters_Used") AS Average_Liters_Used,
    SUM(fc."Yield_Tons") AS Total_Yield
FROM "FactCrop" fc
JOIN "DimCrop" dc ON fc."DimCropID" = dc."DimCropID"
JOIN "DimFarm" f ON fc."DimFarmID" = f."DimFarmID"
WHERE dc."Crop_Name" = 'Soybean'
GROUP BY f."Farm_Name", f."Size_Acres"
ORDER BY Total_Yield DESC
LIMIT 10;
```

ults Chart

rm_Name	Size_Acres	···	AVERAGE_LITERS_USED	TOTAL_YIELD
res LLC	384		19490.5	198.96
son-Cook	107		36498	189.54

Query Details

Query duration 29ms

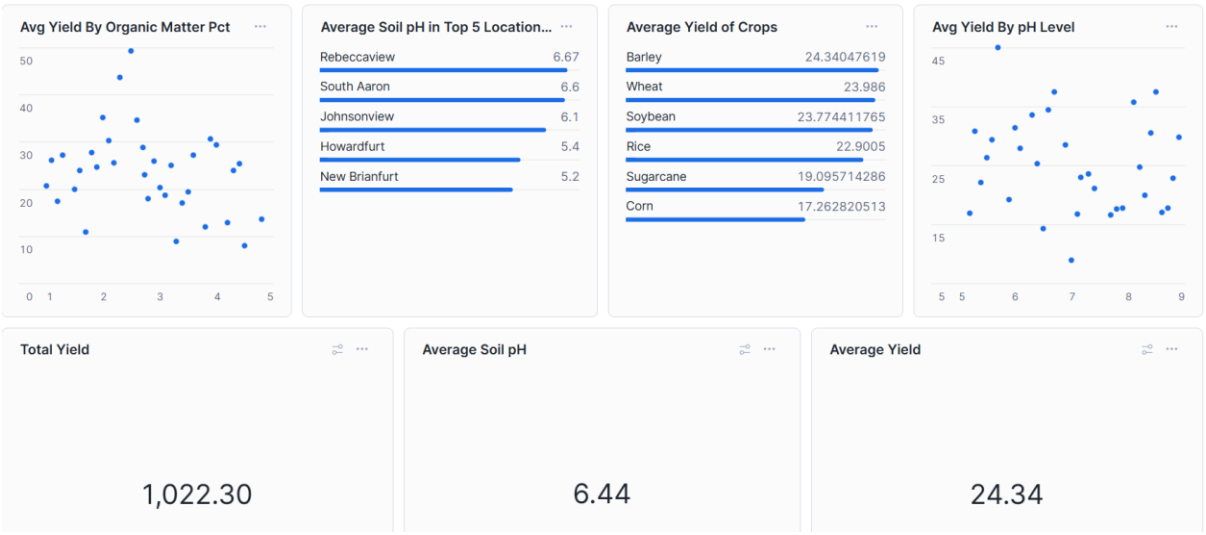
```sql
-- 5.Which location has the best weather conditions for growing corn (based on average yield)
SELECT
    dl."Location_Name",
    dl."Temperature_C",
    dl."Rainfall_mm",
    dl."Condition",
    AVG(fc."Yield_Tons") AS Avg_Yield
FROM "FactCrop" fc
JOIN "DimCrop" dc ON fc."DimCropID" = dc."DimCropID"
JOIN "DimLocation" dl ON fc."DimLocationID" = dl."DimLocationID"
WHERE dc."Crop_Name" = 'Corn'
GROUP BY
    dl."Location_Name",
    dl."Temperature_C",
    dl."Rainfall_mm",
    dl."Condition"
ORDER BY Avg_Yield DESC
LIMIT 1;
```

lts Chart

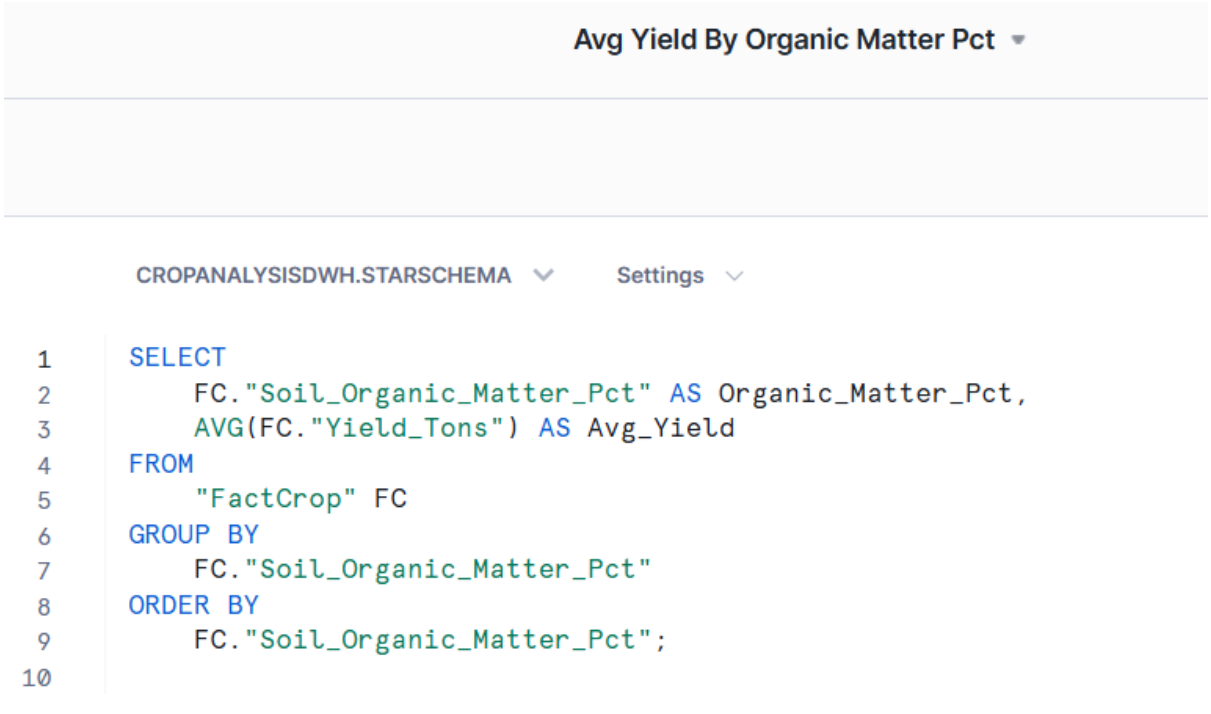
ation_Name	Temperature_C	···	Rainfall_mm	Condition	AVG_YIELD
th Chelseashire	32.9		98.7	Cloudy	45.76

Dashboarding in Snowflake:

After the execution of Dimensional Queries, a simple dashboard was created in snowflake in order to display a summarized view of the warehouse and provide brief insights about the data.



All of these visualizations were generated using results of SQL queries on the data warehouse.



Average Soil pH in Top 5 Locations by Avg Yield

CROPANALYSISDWH.STARSCHEMA Settings

```
SELECT
    DL."Location_Name" AS Location,
    AVG(FC."Soil_pH_Level") AS Avg_pH_Level
FROM
    "FactCrop" FC
JOIN
    "DimLocation" DL ON FC."DimLocationID" = DL."DimLocationID"
GROUP BY
    DL."Location_Name"
ORDER BY
    AVG(FC."Yield_Tons") DESC
LIMIT 5;
```

Average Yield of Crops

CROPANALYSISDWH.STARSCHEMA Settings

```
SELECT
    DC."Crop_Name" AS Crop,
    AVG(FC."Yield_Tons") AS Avg_Yield
FROM
    "FactCrop" FC
JOIN
    "DimCrop" DC ON FC."DimCropID" = DC."DimCropID"
GROUP BY
    DC."Crop_Name"
ORDER BY
    Avg_Yield DESC;
```

Avg Yield By pH Level

CROPANALYSISDWH.STARSCHEMA Settings

```sql
SELECT
    FC."Soil_pH_Level" AS pH_Level,
    AVG(FC."Yield_Tons") AS Avg_Yield
FROM
    "FactCrop" FC
GROUP BY
    FC."Soil_pH_Level"
ORDER BY
    FC."Soil_pH_Level";
```

— Return to DWH_Dashboard_... Total Yield

Crop Barley

Search objects

- CROPANALYSISDWH
- SNOWFLAKE
- SNOWFLAKE_LEARNING_DB
- SNOWFLAKE_SAMPLE_DA...

CROPANALYSISDWH.STARSCHEMA Settings

```sql
SELECT
    SUM(FC."Yield_Tons") AS Total_Yield
FROM
    "FactCrop" FC
JOIN
    "DimCrop" DC ON FC."DimCropID" = DC."DimCropID"
WHERE
    DC."Crop_Name" = :Crop_Name
```

Return to DWH_Dashboard_... Average Soil pH

Crop Barley

Search objects

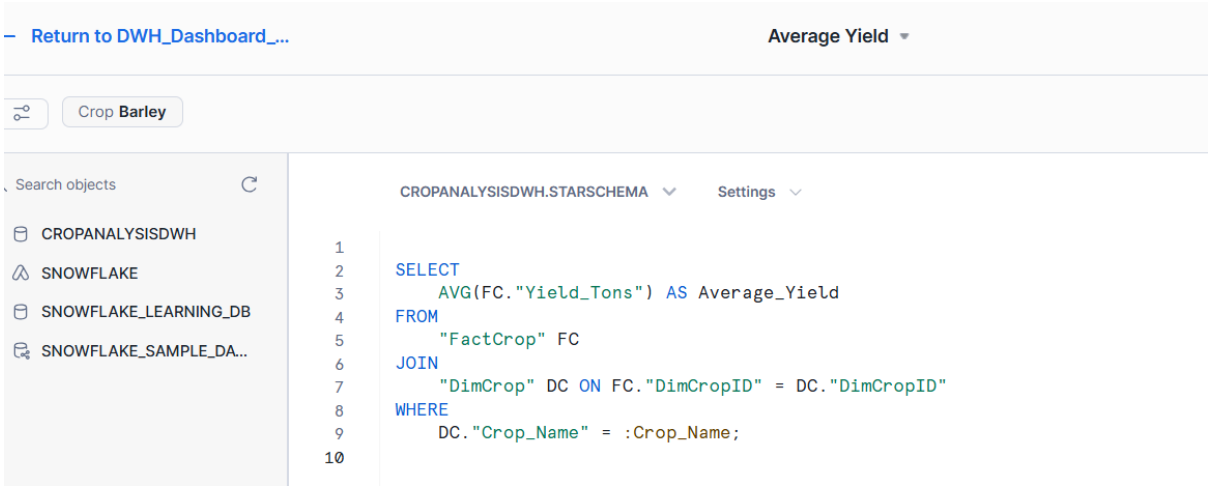
- CROPANALYSISDWH
- SNOWFLAKE
- SNOWFLAKE_LEARNING_DB
- SNOWFLAKE_SAMPLE_DA...

CROPANALYSISDWH.STARSCHEMA Settings

```sql

SELECT
    AVG(FC."Soil_pH_Level") AS Average_Soil_pH
FROM
    "FactCrop" FC
JOIN
    "DimCrop" DC ON FC."DimCropID" = DC."DimCropID"
WHERE
    DC."Crop_Name" = :Crop_Name;

```



The last 3 visualizations also use a custom filter over crop name. The scorecards only display data for the selected crop.