

Internship Program: Soulvibe.Tech

“Farmer & Market Insights Using SQL”

Batch Name: SVT/DAINT/2025/07-B11



Introduction



Overview of the main objectives

In this task, I was assigned to analyze agricultural data using SQL with a focus on two key tables: FarmerAdvisor and MarketResearcher. The objective was to extract meaningful insights related to crop performance, farmer behavior, and market dynamics. Using SQL, I performed various operations such as filtering, grouping, joining tables, and applying window functions to answer specific business questions. These included identifying high-growth crops, ranking profitability, tracking market price changes, and understanding patterns in farmer-crop relationships. The goal was to simulate real-world data analysis scenarios to support better decision-making in agriculture and advisory services by leveraging structured query logic and analytical thinking.

Top 5 locations where the maximum number of farmers with advisors.

The screenshot shows the SQL Developer interface. The top toolbar includes icons for running queries, saving, and other database functions. The main window is divided into two panes: 'Worksheet' and 'Query Builder'. The 'Worksheet' pane contains the following SQL query:

```
SELECT * FROM (
  SELECT Location, COUNT(*) AS FarmerCount
  FROM FarmerAdvisor_Mock
  GROUP BY Location
  ORDER BY FarmerCount DESC
) WHERE ROWNUM <= 5;
```

The 'Query Result' pane displays the results of the query in a table format:

	LOCATION	FARMERCOUNT
1	Guntur	3334
2	Vijayawada	3333
3	Hyderabad	3333

The status bar at the bottom indicates 'All Rows Fetched: 3 in 0.299 seconds'.

2

The average market price and sort in descending order.

The screenshot displays the SQL Developer interface. The top toolbar includes icons for running queries, saving, and other standard database operations. The 'Query Builder' tab is active, showing a SQL query that selects the product name and the average market price per ton, grouped by product and sorted in descending order of the average price. The 'Query Result' tab at the bottom shows the execution results, indicating that 4 rows were fetched in 0.108 seconds. The results are presented in a table with two columns: 'PRODUCT' and 'AVG_MARKET_PRICE'.

```
SELECT
    Product,
    AVG(Market_Price_per_ton) AS Avg_Market_Price
FROM
    MarketResearcher
GROUP BY
    Product
ORDER BY
    Avg_Market_Price DESC;
```

	PRODUCT	AVG_MARKET_PRICE
1	Rice	300.840172927319948563061304178055447091
2	Corn	300.485150617787159612244897959183673469
3	Soybean	299.146857713163046420047732696897374702
4	Wheat	298.08435875293767565252525252525253

3

Count how many unique crops each farmer is associated with.

The screenshot displays a SQL query builder window with a 'Query Builder' tab. The query is as follows:

```
SELECT
    Farm_ID,
    COUNT(DISTINCT Crop_Type) AS Unique_Crop_Count
FROM
    FarmerAdvisor
GROUP BY
    Farm_ID;
```

Below the query editor, the 'Query Result' tab shows the execution results. It indicates that 50 rows were fetched in 0.375 seconds. The results are presented in a table with two columns: FARM_ID and UNIQUE_CROP_COUNT.

	FARM_ID	UNIQUE_CROP_COUNT
1	151	1
2	152	1
3	247	1
4	403	1
5	472	1
6	477	1
7	544	1
8	647	1
9	677	1
10	691	1
11	692	1
12	701	1
13	756	1

4 Farmers who are growing more than 3 different types of crops.

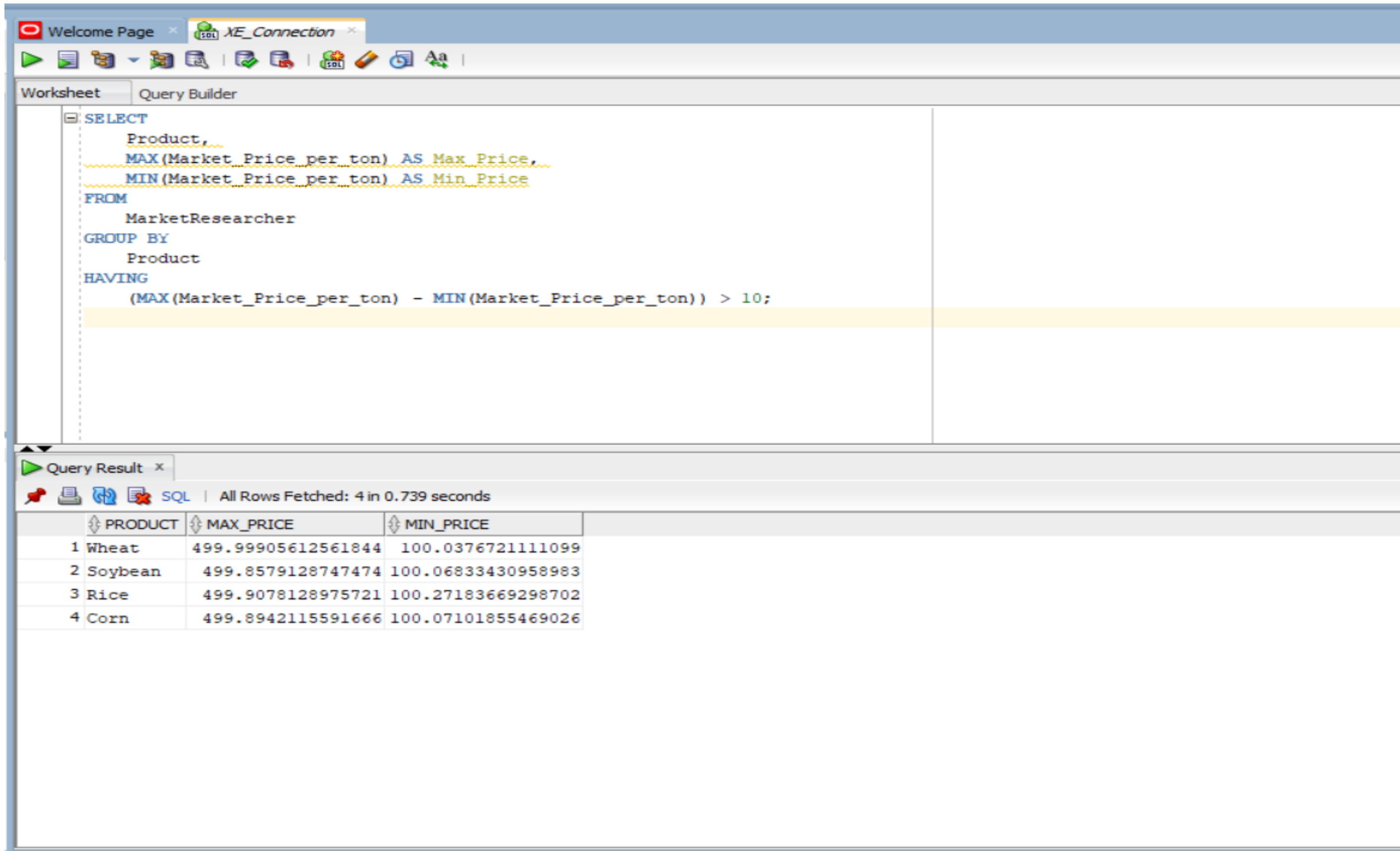
The screenshot displays a SQL query builder window with the following components:

- Worksheet Tab:** Contains the SQL query:

```
SELECT
  Farm_ID
FROM
  FarmerAdvisor
GROUP BY
  Farm_ID
HAVING
  COUNT(DISTINCT Crop_Type) > 3;
```
- Query Result Tab:** Shows the execution status: "All Rows Fetched: 0 in 0.179 seconds".
- Field List:** A list of fields is visible, with "FARM_ID" selected.

5

Crops where the max and min market prices differ by more than ₹10 per kg.



The screenshot shows a SQL query builder interface with a query window and a results window. The query is as follows:

```
SELECT
    Product,
    MAX(Market_Price_per_ton) AS Max_Price,
    MIN(Market_Price_per_ton) AS Min_Price
FROM
    MarketResearcher
GROUP BY
    Product
HAVING
    (MAX(Market_Price_per_ton) - MIN(Market_Price_per_ton)) > 10;
```

The results window shows the following data:

	PRODUCT	MAX_PRICE	MIN_PRICE
1	Wheat	499.99905612561844	100.0376721111099
2	Soybean	499.8579128747474	100.06833430958983
3	Rice	499.9078128975721	100.27183669298702
4	Corn	499.8942115591666	100.07101855469026

All advisors who guide farmers growing the same crop in different districts.

Welcome Page | XE_Connection

Worksheet | Query Builder

SELECT *

FROM FarmerAdviso_Mock

WHERE (AdvisorID, Crop_Type) IN (

SELECT AdvisorID, Crop_Type

FROM FarmerAdvisor_Mock

GROUP BY AdvisorID, Crop_Type

HAVING COUNT(DISTINCT Location) > 1

);

Script Output | Query Result

SQL | Fetched 50 rows in 0.496 seconds

	FARM_ID	CROP_TYPE	ADVISORID	LOCATION
1	1	Wheat	2	Guntur
2	2	Soybean	3	Vijayawada
3	3	Corn	4	Hyderabad
4	4	Wheat	5	Guntur
5	5	Corn	1	Vijayawada
6	6	Rice	2	Hyderabad
7	7	Soybean	3	Guntur
8	8	Soybean	4	Vijayawada
9	9	Wheat	5	Hyderabad
10	10	Soybean	1	Guntur
11	11	Corn	2	Vijayawada
12	12	Corn	3	Hyderabad
13	13	Rice	4	Guntur

Rank crops by profit per unit using RANK().

The screenshot shows a SQL query editor with a query that ranks crops by profit per unit. The query is as follows:

```
SELECT
  MR.PRODUCT AS CROP_NAME,
  ROUND(MR.MARKET_PRICE_PER_TON - FA.AVG_COST, 2) AS PROFIT_PER_UNIT,
  RANK() OVER (ORDER BY (MR.MARKET_PRICE_PER_TON - FA.AVG_COST) DESC) AS PROFIT_RANK
FROM
  MARKETRESEARCHER MR
JOIN (
  SELECT
    CROP_TYPE,
    ROUND(AVG(NVL(FERTILIZER_USAGE_KG, 0) + NVL(PESTICIDE_USAGE_KG, 0)), 2) AS AVG_COST
  FROM
    FARMERADVISOR
  GROUP BY
    CROP_TYPE
) FA
ON
  MR.PRODUCT = FA.CROP_TYPE;
```

The query result is displayed below the editor, showing 13 rows of data. The columns are CROP_NAME, PROFIT_PER_UNIT, and PROFIT_RANK. The data is as follows:

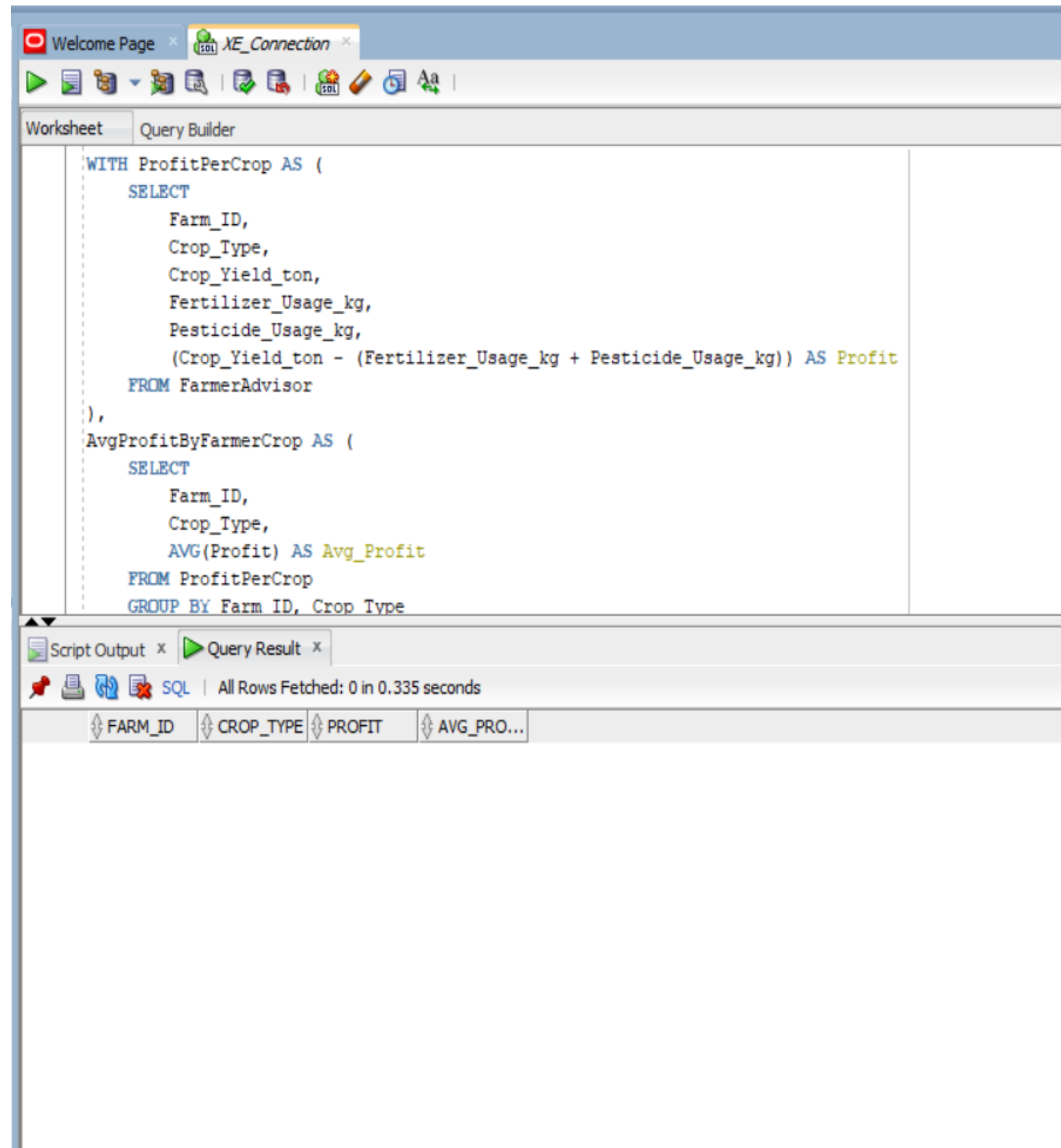
	CROP_NAME	PROFIT_PER_UNIT	PROFIT_RANK
1	Soybean	365.11	1
2	Soybean	364.98	2
3	Soybean	364.79	3
4	Soybean	364.51	4
5	Soybean	364.44	5
6	Soybean	364.39	6
7	Soybean	364.38	7
8	Soybean	364.32	8
9	Soybean	364.17	9
10	Rice	364.11	10
11	Rice	363.99	11
12	Soybean	363.99	12
13	Wheat	363.87	13

8 Locations where the current market price of a crop is more than 20% above the average price of that crop across all locations.

WITH AvgPrice AS (
SELECT Product, AVG(Market_Price_per_ton) AS Avg_Price
FROM MarketResearcher
GROUP BY Product
)
SELECT m.*
FROM MarketResearcher m
JOIN AvgPrice a ON m.Product = a.Product
WHERE m.Market_Price_per_ton > a.Avg_Price * 1.2;

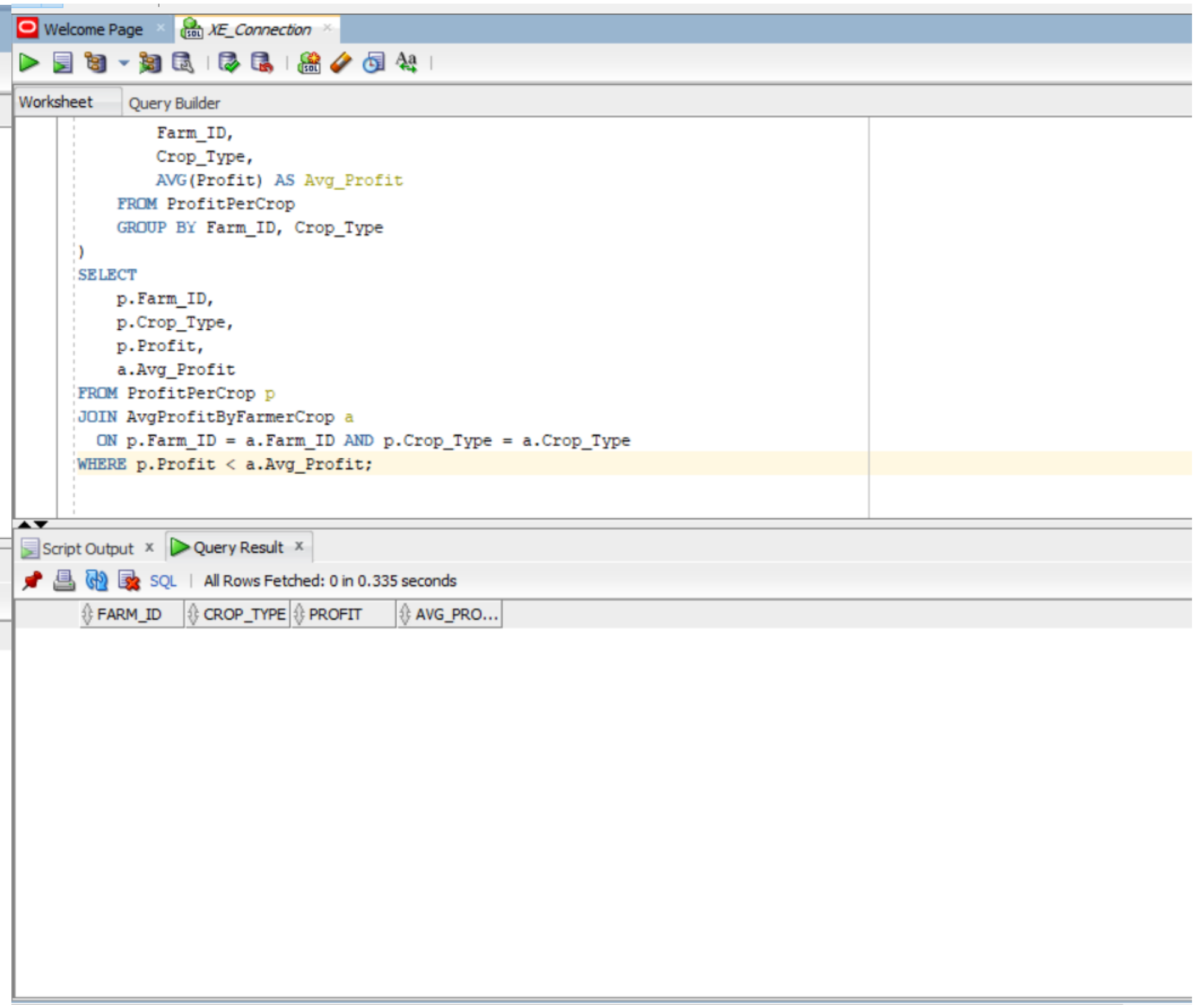
	MARKET_ID	PRODUCT	MARKET_PRICE_PER_TON	DEMAND_INDEX	SUPPLY_INDEX	COMPETITOR_PRICE_PER_TON	ECONOMIC_INDICATOR	WEATHER_IMPACT_SCORE	SEASONAL
1	2	Rice	420.52796955955307	188.45240008580845	150.789482977595	492.097797644394	0.5263068815761089	70.97806264396277	High
2	3	Wheat	457.2603975211701	171.17938395811063	78.98932630036724	323.00334230465387	1.29239334879865	80.8535915904238	Low
3	6	Corn	389.6482051471298	123.08872887995155	190.90647508558038	181.9354392180609	0.8761502603945046	81.36985531316705	Low
4	9	Soybean	447.4117730632625	188.31754490852956	171.82949278671953	210.50295551774195	0.7078028072550187	29.150676979735856	Low
5	14	Soybean	481.94841517479534	164.5570086363353	117.89743276774203	315.894413578798	0.996883786192946	81.29448800288094	Medium
6	16	Corn	400.6447487209915	192.60030048359053	68.85381701077962	268.40720091667856	0.8406059954568866	98.0521766129821	Medium
7	20	Soybean	468.08534637097773	64.18598493416194	108.30769511092905	375.33084499206666	0.9957607808823131	67.39407678002567	Low
8	21	Soybean	404.3796066798186	170.12732957924175	198.9322484582829	367.5274051914306	1.195187014934815	72.08959196466841	High
9	22	Wheat	444.2102233468652	132.45943830884892	85.20097918199315	355.1809845681516	1.2960913631131243	8.891188062752187	Low
10	24	Wheat	431.8034792818116	101.57379387931167	62.578446491784575	393.4910002637361	0.8898999419714463	5.924634912520432	High
11	27	Soybean	443.2644718458053	176.67465539540592	163.34341019096973	489.31539884979804	0.6639799275587567	23.220849964990066	High
12	33	Rice	426.99328668626333	66.85687441609332	73.33422299655902	166.84376039274733	1.4116215192869785	36.30247044148278	High
13	35	Corn	419.87995300664267	187.06816235053117	164.57811954468576	382.46982659607687	1.4104332604987757	75.4632025945921	High

10 . CTE showing average profit per crop type by farmer and use it to list farmers making below-average profits on any crop.



The screenshot shows the SQL Server Enterprise Manager interface. The 'Query Builder' tab is active, displaying a CTE query. The query defines two CTEs: 'ProfitPerCrop' and 'AvgProfitByFarmerCrop'. The 'ProfitPerCrop' CTE selects columns from 'FarmerAdvisor' and calculates profit. The 'AvgProfitByFarmerCrop' CTE calculates the average profit for each farmer and crop type. The main query selects from 'ProfitPerCrop' and joins it with 'AvgProfitByFarmerCrop' to find farmers whose profit is below the average for their crop type.

```
WITH ProfitPerCrop AS (  
    SELECT  
        Farm_ID,  
        Crop_Type,  
        Crop_Yield_ton,  
        Fertilizer_Usage_kg,  
        Pesticide_Usage_kg,  
        (Crop_Yield_ton - (Fertilizer_Usage_kg + Pesticide_Usage_kg)) AS Profit  
    FROM FarmerAdvisor  
),  
AvgProfitByFarmerCrop AS (  
    SELECT  
        Farm_ID,  
        Crop_Type,  
        AVG(Profit) AS Avg_Profit  
    FROM ProfitPerCrop  
    GROUP BY Farm ID, Crop Type  
)  
SELECT  
    p.Farm_ID,  
    p.Crop_Type,  
    p.Profit,  
    a.Avg_Profit  
FROM ProfitPerCrop p  
JOIN AvgProfitByFarmerCrop a  
    ON p.Farm_ID = a.Farm_ID AND p.Crop_Type = a.Crop_Type  
WHERE p.Profit < a.Avg_Profit;
```



The screenshot shows the same SQL Server Enterprise Manager interface, but with the 'Query Result' tab active. The query is the same as in the previous screenshot. The 'Query Result' tab shows the results of the query, with columns 'FARM_ID', 'CROP_TYPE', 'PROFIT', and 'AVG_PRO...'. The results are displayed in a table format.

```
Farm_ID,  
Crop_Type,  
AVG(Profit) AS Avg_Profit  
FROM ProfitPerCrop  
GROUP BY Farm_ID, Crop_Type  
)  
SELECT  
    p.Farm_ID,  
    p.Crop_Type,  
    p.Profit,  
    a.Avg_Profit  
FROM ProfitPerCrop p  
JOIN AvgProfitByFarmerCrop a  
    ON p.Farm_ID = a.Farm_ID AND p.Crop_Type = a.Crop_Type  
WHERE p.Profit < a.Avg_Profit;
```


11

Use a window function to find the price change of each crop over the last 3 entries

Worksheet | Query Builder

```

SELECT
  curr.Crop_Type,
  curr.PriceDate,
  curr.Market_Price_per_ton AS Current_Price,
  NVL(curr.Market_Price_per_ton - prev1.Market_Price_per_ton, 0) AS Change_1,
  NVL(curr.Market_Price_per_ton - prev2.Market_Price_per_ton, 0) AS Change_2,
  NVL(curr.Market_Price_per_ton - prev3.Market_Price_per_ton, 0) AS Change_3
FROM
  MarketResearcher_Mock curr
LEFT JOIN
  MarketResearcher_Mock prev1
  ON curr.Crop_Type = prev1.Crop_Type AND prev1.PriceDate = curr.PriceDate - 1
LEFT JOIN
  MarketResearcher_Mock prev2
  ON curr.Crop_Type = prev2.Crop_Type AND prev2.PriceDate = curr.PriceDate - 2
LEFT JOIN
  MarketResearcher_Mock prev3
  ON curr.Crop_Type = prev3.Crop_Type AND prev3.PriceDate = curr.PriceDate - 3
ORDER BY
  curr.Crop_Type, curr.PriceDate;

```

Script Output x | Query Result x

SQL | Fetched 100 rows in 0.846 seconds

	CROP_TYPE	PRICEDATE	CURRENT_PRICE	CHANGE_1	CHANGE_2	CHANGE_3
1	Maize	03-07-25	420.52796955955307	0	0	0
2	Maize	06-07-25	324.03292488473636	0	0	-96.49504467481671
3	Maize	09-07-25	321.72672767442924	0	0	-2.30619721030712
4	Maize	12-07-25	101.52623915610275	0	0	-220.20048851832649
5	Maize	15-07-25	481.94841517479534	0	0	380.42217601869259
6	Maize	18-07-25	173.45643527061497	0	0	-308.49197990418037
7	Maize	21-07-25	468.08534637097773	0	0	294.62891110036276
8	Maize	24-07-25	172.04503923748314	0	0	-296.04030713349459
9	Maize	27-07-25	111.17576201444686	0	0	-60.86927722303628
10	Maize	30-07-25	178.39574761159275	0	0	67.21998559714589
11	Maize	02-08-25	284.2910531916458	0	0	105.89530558005305
12	Maize	05-08-25	419.87995300664267	0	0	135.58889981499687
13	Maize	08-08-25	251.5614862336316	0	0	-168.31846677301107

Worksheet | Query Builder

```

SELECT
  NVL(curr.Market_Price_per_ton - prev3.Market_Price_per_ton, 0) AS Change_3
FROM
  MarketResearcher_Mock curr
LEFT JOIN
  MarketResearcher_Mock prev1
  ON curr.Crop_Type = prev1.Crop_Type AND prev1.PriceDate = curr.PriceDate - 1
LEFT JOIN
  MarketResearcher_Mock prev2
  ON curr.Crop_Type = prev2.Crop_Type AND prev2.PriceDate = curr.PriceDate - 2
LEFT JOIN
  MarketResearcher_Mock prev3
  ON curr.Crop_Type = prev3.Crop_Type AND prev3.PriceDate = curr.PriceDate - 3
ORDER BY
  curr.Crop_Type, curr.PriceDate;

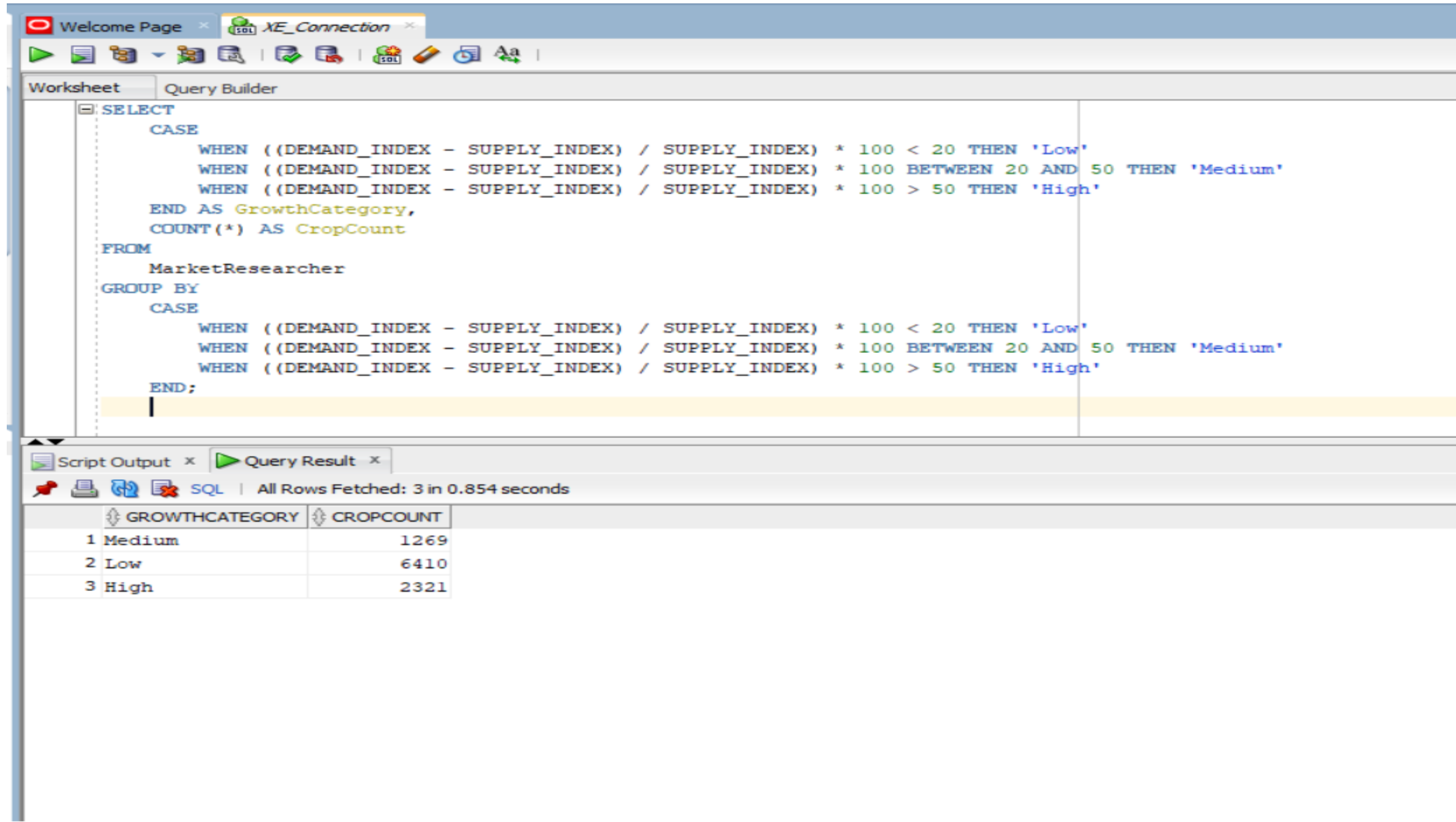
```

Script Output x | Query Result x

SQL | Fetched 100 rows in 0.846 seconds

	CROP_TYPE	PRICEDATE	CURRENT_PRICE	CHANGE_1	CHANGE_2	CHANGE_3
1	Maize	03-07-25	420.52796955955307	0	0	0
2	Maize	06-07-25	324.03292488473636	0	0	-96.49504467481671
3	Maize	09-07-25	321.72672767442924	0	0	-2.30619721030712
4	Maize	12-07-25	101.52623915610275	0	0	-220.20048851832649
5	Maize	15-07-25	481.94841517479534	0	0	380.42217601869259
6	Maize	18-07-25	173.45643527061497	0	0	-308.49197990418037
7	Maize	21-07-25	468.08534637097773	0	0	294.62891110036276
8	Maize	24-07-25	172.04503923748314	0	0	-296.04030713349459
9	Maize	27-07-25	111.17576201444686	0	0	-60.86927722303628
10	Maize	30-07-25	178.39574761159275	0	0	67.21998559714589
11	Maize	02-08-25	284.2910531916458	0	0	105.89530558005305
12	Maize	05-08-25	419.87995300664267	0	0	135.58889981499687
13	Maize	08-08-25	251.5614862336316	0	0	-168.31846677301107

12 Create a new column that classifies crop growth Count the number of crops in each category.



The screenshot displays the SQL Developer interface. The top pane shows a SQL query in the Query Builder tab. The query uses a CASE statement to classify crop growth into 'Low', 'Medium', or 'High' based on the ratio of Demand Index to Supply Index, and counts the number of crops in each category using COUNT(*).

```
SELECT
  CASE
    WHEN ((DEMAND_INDEX - SUPPLY_INDEX) / SUPPLY_INDEX) * 100 < 20 THEN 'Low'
    WHEN ((DEMAND_INDEX - SUPPLY_INDEX) / SUPPLY_INDEX) * 100 BETWEEN 20 AND 50 THEN 'Medium'
    WHEN ((DEMAND_INDEX - SUPPLY_INDEX) / SUPPLY_INDEX) * 100 > 50 THEN 'High'
  END AS GrowthCategory,
  COUNT(*) AS CropCount
FROM
  MarketResearcher
GROUP BY
  CASE
    WHEN ((DEMAND_INDEX - SUPPLY_INDEX) / SUPPLY_INDEX) * 100 < 20 THEN 'Low'
    WHEN ((DEMAND_INDEX - SUPPLY_INDEX) / SUPPLY_INDEX) * 100 BETWEEN 20 AND 50 THEN 'Medium'
    WHEN ((DEMAND_INDEX - SUPPLY_INDEX) / SUPPLY_INDEX) * 100 > 50 THEN 'High'
  END;
```

The bottom pane shows the Query Result tab with the following data:

	GROWTHCATEGORY	CROPCOUNT
1	Medium	1269
2	Low	6410
3	High	2321

13 Join the tables and display all farmers whose crop is sold in the same district at the highest price.

Worksheet Query Builder

```

WITH MaxPricePerCrop AS (
    SELECT
        PRODUCT,
        MAX(MARKET_PRICE_PER_TON) AS MaxPrice
    FROM
        MarketResearcher
    GROUP BY
        PRODUCT
)

SELECT
    f.FARM_ID,
    f.CROP_TYPE,
    f.CROP_YIELD_TON,
    m.MARKET_PRICE_PER_TON
FROM
    FarmerAdvisor f

```

Script Output x Query Result x

SQL | Fetched 50 rows in 0.239 seconds

	FARM_ID	CROP_TYPE	CROP_YIELD_TON	MARKET_PRICE_PER_TON
1	1	Wheat	1.5769199239944838	499.99905612561844
2	2	Soybean	3.824685657452959	499.8579128747474
3	3	Corn	1.1331976262416603	499.8942115591666
4	4	Wheat	8.870540370316064	499.99905612561844
5	5	Corn	8.779317423442908	499.8942115591666
6	6	Rice	2.0437081720414625	499.9078128975721
7	7	Soybean	6.177211062248057	499.8579128747474
8	8	Soybean	5.218131637064065	499.8579128747474
9	9	Wheat	7.801282070734073	499.99905612561844
10	10	Soybean	6.221388077188259	499.8579128747474
11	11	Corn	2.27026393857822	499.8942115591666
12	12	Corn	2.847876691130483	499.8942115591666
13	13	Rice	9.944867740520532	499.9078128975721

Worksheet Query Builder

```

f.CROP_TYPE,
f.CROP_YIELD_TON,
m.MARKET_PRICE_PER_TON
FROM
    FarmerAdvisor f
JOIN
    MarketResearcher m
ON f.CROP_TYPE = m.PRODUCT
JOIN
    MaxPricePerCrop mp
ON mp.PRODUCT = m.PRODUCT
AND mp.MaxPrice = m.MARKET_PRICE_PER_TON;

```

Script Output x Query Result x

SQL | Fetched 50 rows in 0.239 seconds

	FARM_ID	CROP_TYPE	CROP_YIELD_TON	MARKET_PRICE_PER_TON
1	1	Wheat	1.5769199239944838	499.99905612561844
2	2	Soybean	3.824685657452959	499.8579128747474
3	3	Corn	1.1331976262416603	499.8942115591666
4	4	Wheat	8.870540370316064	499.99905612561844
5	5	Corn	8.779317423442908	499.8942115591666
6	6	Rice	2.0437081720414625	499.9078128975721
7	7	Soybean	6.177211062248057	499.8579128747474
8	8	Soybean	5.218131637064065	499.8579128747474
9	9	Wheat	7.801282070734073	499.99905612561844
10	10	Soybean	6.221388077188259	499.8579128747474
11	11	Corn	2.27026393857822	499.8942115591666
12	12	Corn	2.847876691130483	499.8942115591666
13	13	Rice	9.944867740520532	499.9078128975721

14

Count how many of their farmers grow crops that fall under the “High” growth classification

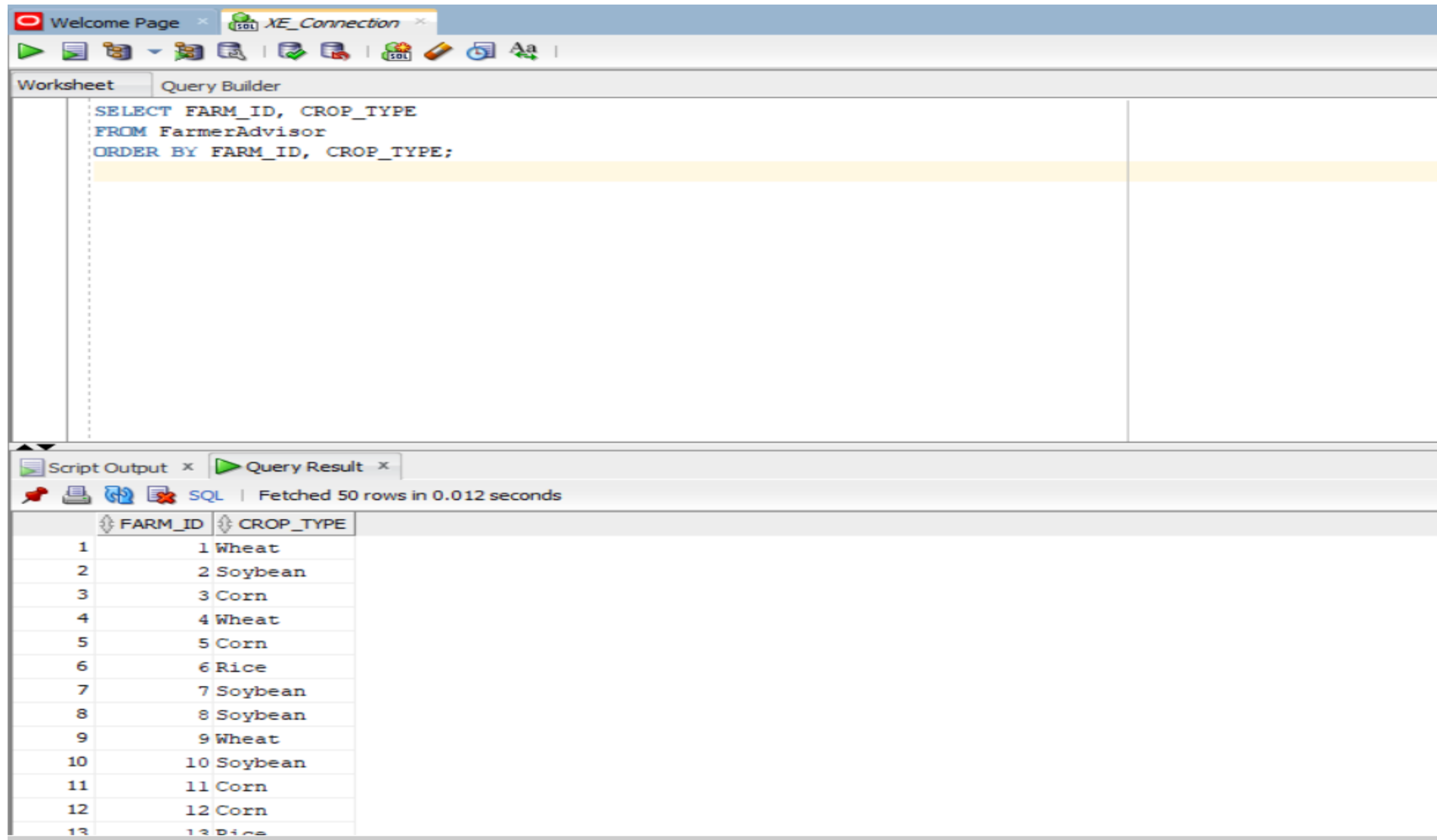
The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows a query in the Query Builder tab. The query is as follows:

```
SELECT
    COUNT(DISTINCT Farm_ID) AS HighGrowthFarmerCount
FROM
    FarmerAdvisor
WHERE
    Crop_Type IN (
        SELECT Crop_Type
        FROM MarketResearcher
        GROUP BY Crop_Type
        HAVING MAX(Market_Price_per_ton) - MIN(Market_Price_per_ton) > 15
    );
```

The bottom pane shows the Query Result tab, which displays the results of the query. The results are as follows:

	HIGHGROWTHFARMERCOUNT
1	10000

15 Identify if any farmer has duplicate crop entries.



The screenshot shows a SQL query tool interface. The top pane, labeled 'Query Builder', contains the following SQL query:

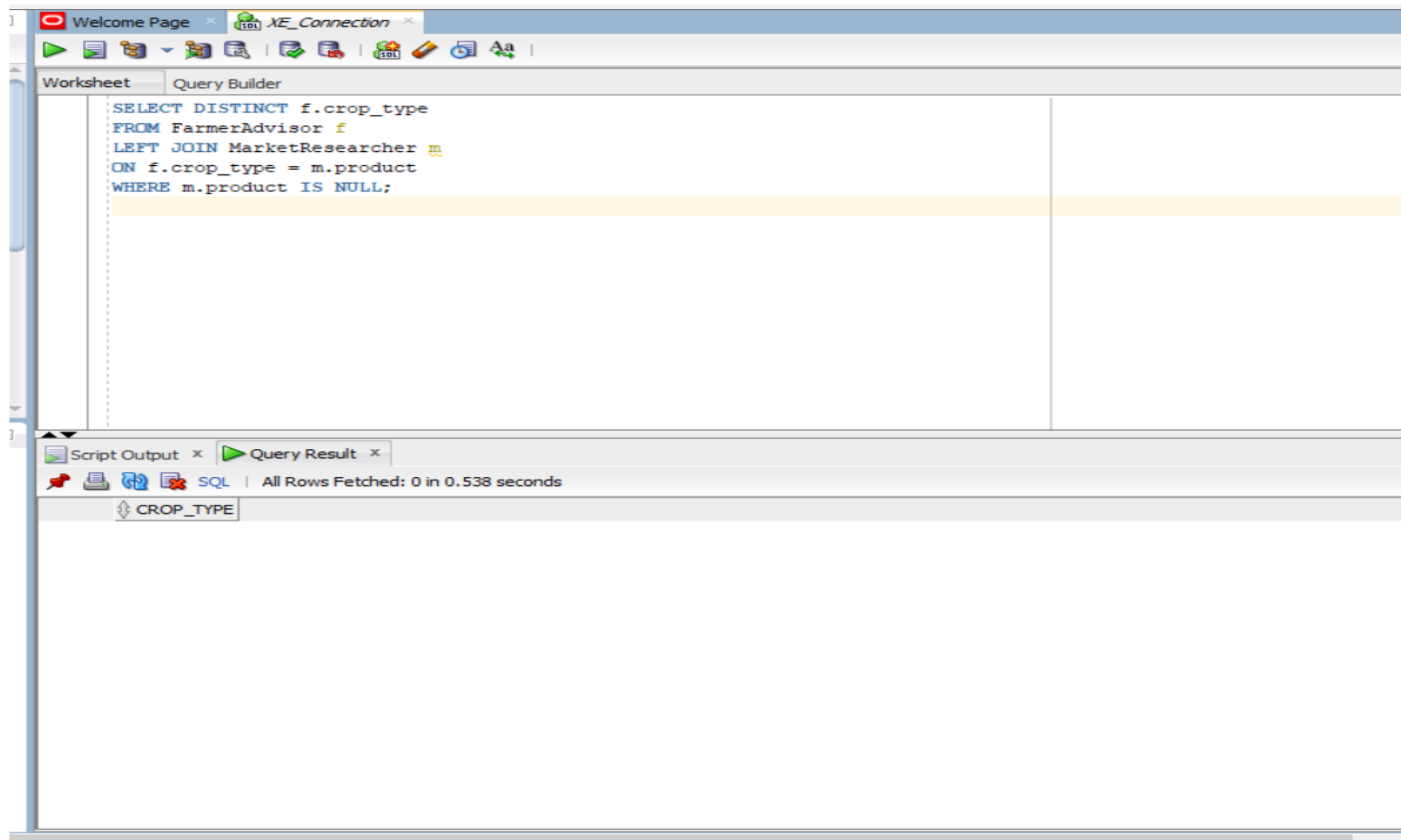
```
SELECT FARM_ID, CROP_TYPE
FROM FarmerAdvisor
ORDER BY FARM_ID, CROP_TYPE;
```

The bottom pane, labeled 'Query Result', shows the results of the query. It indicates that 50 rows were fetched in 0.012 seconds. The results are displayed in a table with two columns: FARM_ID and CROP_TYPE.

	FARM_ID	CROP_TYPE
1	1	Wheat
2	2	Soybean
3	3	Corn
4	4	Wheat
5	5	Corn
6	6	Rice
7	7	Soybean
8	8	Soybean
9	9	Wheat
10	10	Soybean
11	11	Corn
12	12	Corn
13	13	Rice











16

All crops grown by farmers that are not listed in the MarketResearcher table.



17 Determine which location has the highest average profit margin.







Welcome Page xXE_Connection x



WorksheetQuery Builder

```
SELECT
    Product,
    AVG (Market_Price_per_ton - Competitor_Price_per_ton) AS Avg_Profit_Margin
FROM
    MarketResearcher
GROUP BY
    Product;
```

Script Output xQuery... x



SQL | All Rows Fetched: 4 in 0.514 seconds

	PRODUCT	AVG_PROFIT_MARGIN
1	Wheat	-2.60714112900203460202020202020202
2	Soybean	-2.46146452034653447494033412887828162291
3	Rice	-0.9352008352696401952362358453729012104647
4	Corn	1.37964619527439205306122448979591836735

18 Use window functions to find the second-highest market price crop per district.

The screenshot shows a SQL query editor window with the following SQL code:

```
SELECT Market_ID, Product, Market_Price_per_ton
FROM (
    SELECT
        Market_ID,
        Product,
        Market_Price_per_ton,
        DENSE_RANK() OVER (PARTITION BY Market_ID ORDER BY Market_Price_per_ton DESC) AS price_rank
    FROM MarketResearcher
) ranked_prices
WHERE price_rank = 2;
```

The query is executed, and the results are displayed in the Query Result pane. The results show the second-highest market price crop per district.

MARKET_ID	PRODUCT	MARKET_PRICE_PER_TON
1	Wheat	1200
1	Rice	1100
2	Wheat	1100
2	Rice	1000
3	Wheat	1000
3	Rice	900

19

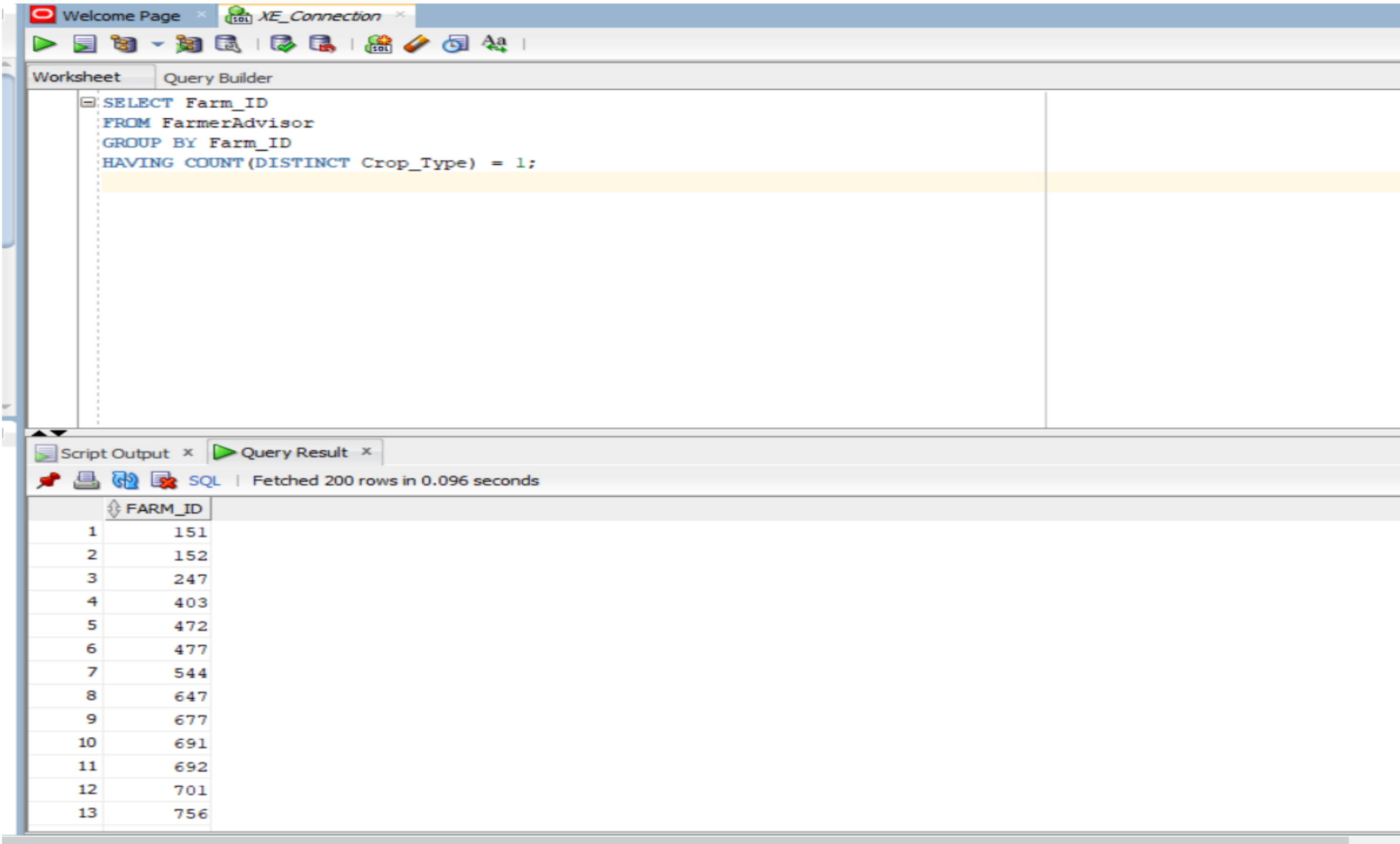
All advisors associated with more than 5 distinct crop types.

The screenshot shows a SQL query editor window with a toolbar at the top containing icons for execution, saving, and other functions. The main area is divided into a 'Worksheet' tab and a 'Query Builder' tab. The 'Query Builder' tab is active, displaying the following SQL query:

```
SELECT Farm_ID, COUNT(DISTINCT Crop_Type) AS Crop_Count  
FROM FarmerAdvisor  
GROUP BY Farm_ID  
HAVING COUNT(DISTINCT Crop_Type) > 5;
```

Below the query editor, there is a 'Script Output' tab and a 'Query Result' tab. The 'Query Result' tab is active, showing the status 'All Rows Fetched: 0 in 0.326 seconds'. Below this, there are two columns labeled 'FARM_ID' and 'CROP_CO...'.

20 Farmers who consistently grow the same crop type for all seasons



The screenshot shows a SQL query editor with a query that filters for farmers who grow only one crop type. The query is as follows:

```
SELECT Farm_ID
FROM FarmerAdvisor
GROUP BY Farm_ID
HAVING COUNT(DISTINCT Crop_Type) = 1;
```

The query results are displayed in a table with 13 rows, showing the Farm_ID for each farmer who consistently grows the same crop type.

	FARM_ID
1	151
2	152
3	247
4	403
5	472
6	477
7	544
8	647
9	677
10	691
11	692
12	701
13	756

Conclusion

Through this SQL-based exploration of the agricultural dataset involving the *FarmerAdvisor* and *MarketResearcher* tables, I gained practical experience in writing queries to extract, join, filter, group, and summarize data efficiently. By solving multiple scenario-based questions, I was able to:

- Identify crop growth and profitability patterns
- Analyze farmer-advisor relationships
- Track market price trends across crops
- Derive insights from regional and crop-specific data

Key Takeaways:

- SQL enables powerful data analysis and insight generation directly from relational databases.
- Writing diverse queries improved my understanding of database structure, keys, and data relationships.
- The insights obtained can support better decisions in agriculture, such as advisor assignments, market pricing strategy, and crop planning.
- This exercise also forms a strong base for integrating SQL findings with data visualization tools like Power BI or Tableau for more intuitive presentations.



Thank You

