ISM 6208: Data Warehousing

Assignment 2: Analytic SQL

Group 8

ANKUR SRIVASTAVA

GOUTHAM BEERAM

PRASAD ACHARYA

RISHABH MITTAL

RITIK GUPTA

PART I: Query writing tasks

Query 1: Aggregations with CUBE and ROLLUP

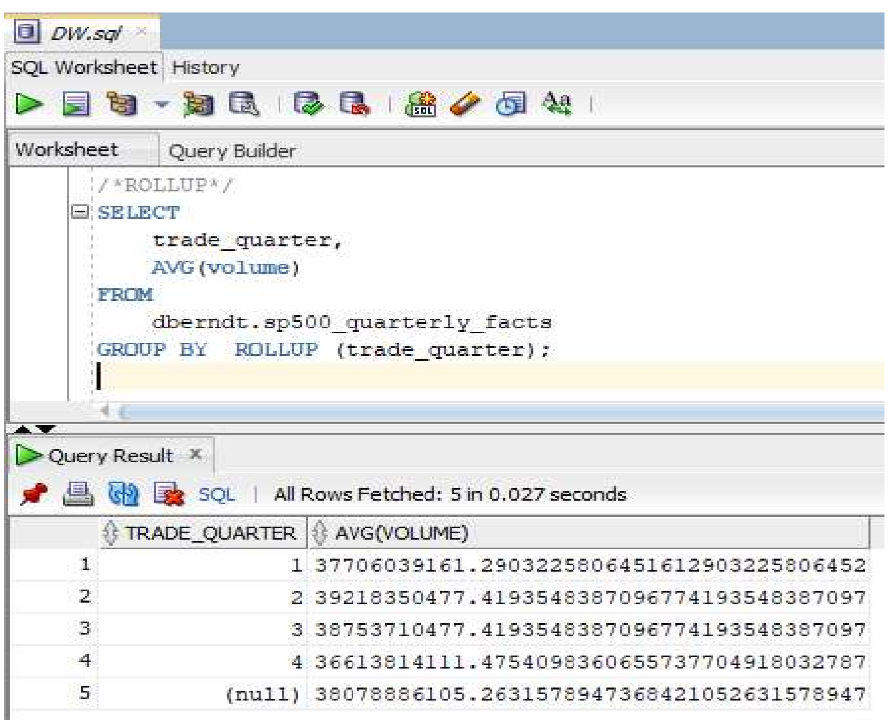
ROLLUP and CUBE based on single column -

This Simple query aggregated the data for each quarter for all the years in quarterly facts table. As we know, NULL in ROLLUP represents the aggregate of all the rows provided in the group by statement. So, we can see null in this query represents the average of all the QUARTERS for all the years in the dataset. Since grouping is based on single column there is no difference in ROLLUP and CUBE output as shown below.

<u>SQL Query:</u>

```
SELECT
   trade_quarter,
   AVG(volume)
FROM
   dberndt.sp500_quarterly_facts
GROUP BY  ROLLUP (trade_quarter);
```

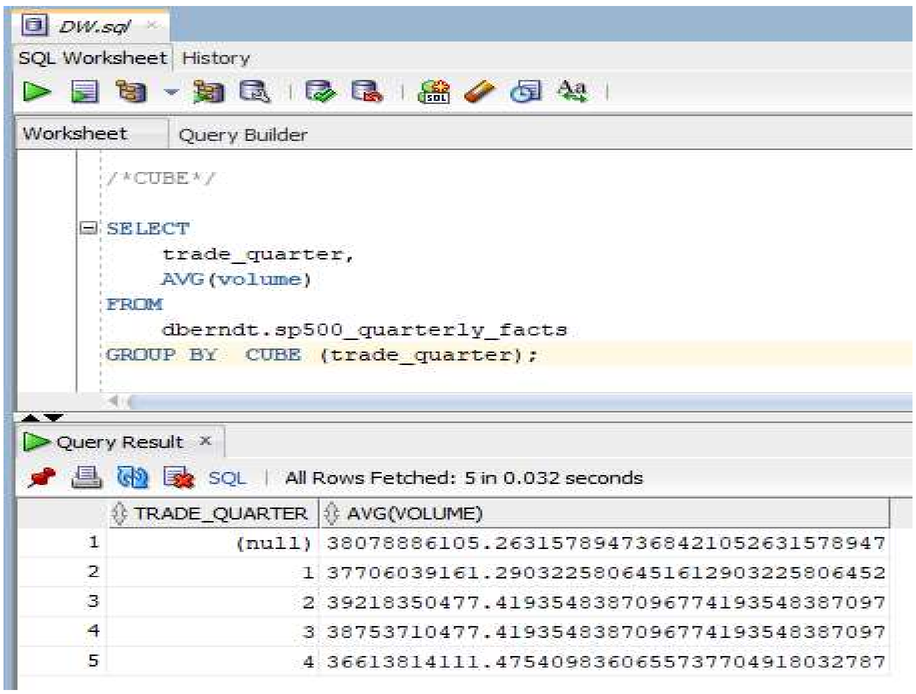
<u>Output screenshot:</u>



SQL Query:

```
SELECT
   trade_quarter,
   AVG(volume)
FROM
   dberndt.sp500_quarterly_facts
GROUP BY  CUBE (trade_quarter);
```

Output screenshot:



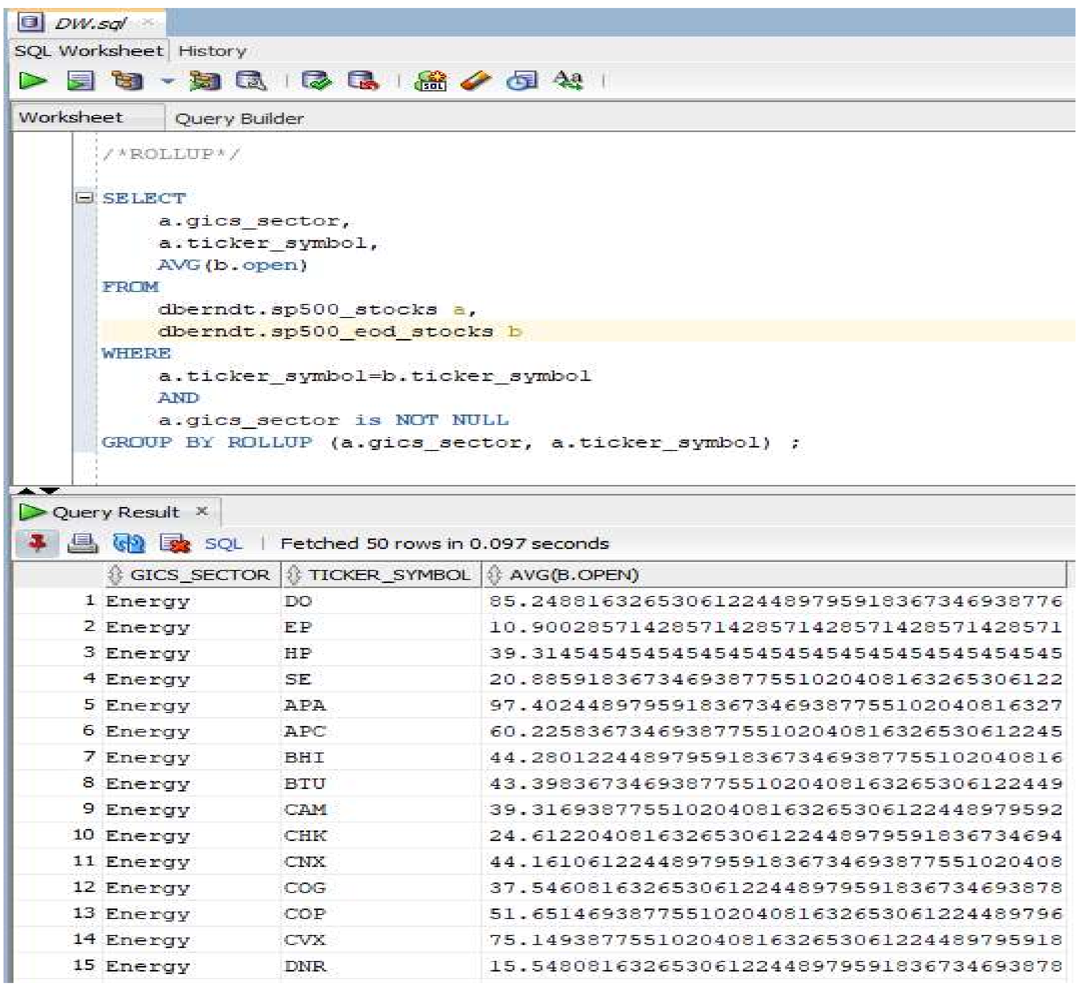
ROLLUP and CUBE based on two columns -

If we want to get the sector ticker symbol and the average of the opening stock value using ROLLUP we must join two tables - STOCKS and EOD STOCKS. Below is the query in which we have considered these 2 tables and 2 columns - gics_sector and ticker_symbol. ROLLUP query returned 494 rows. ROLLUP result set shows aggregates for a hierarchy of values in the selected columns.

SQL Query:

SELECT
   a.gics_sector,
   a.ticker_symbol,
   AVG(b.open)
FROM
   dberndt.sp500_stocks a,
   dberndt.sp500_eod_stocks b
WHERE
   a.ticker_symbol=b.ticker_symbol
   AND
   a.gics_sector is NOT NULL
GROUP BY ROLLUP (a.gics_sector, a.ticker_symbol) ;

Output screenshot:



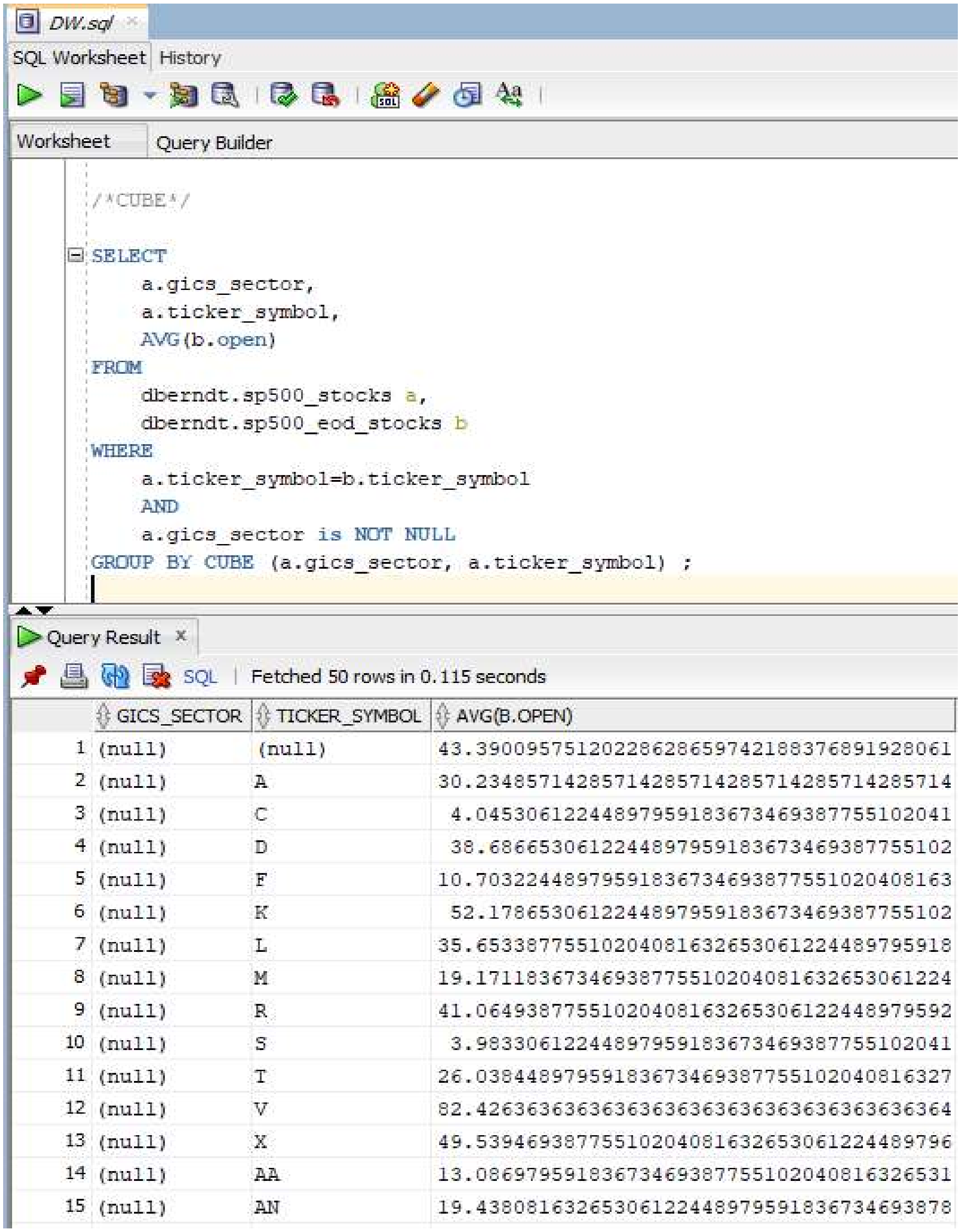
	GICS_SECTOR	TICKER_SYMBOL	AVG(B.OPEN)
1	Energy	DO	85.2488163265306122448979591836734693877 6
2	Energy	EP	10.9002857142857142857142857142857142857 1
3	Energy	HP	39.3145454545454545454545454545454545454 5
4	Energy	SE	20.8859183673469387755102040816326530612 2
5	Energy	APA	97.4024489795918367346938775510204081632 7
6	Energy	APC	60.2258367346938775510204081632653061224 5
7	Energy	BHI	44.2801224489795918367346938775510204081 6
8	Energy	BTU	43.3983673469387755102040816326530612244 9
9	Energy	CAM	39.3169387755102040816326530612244897959 2
10	Energy	CHK	24.6122040816326530612244897959183673469 4
11	Energy	CNX	44.1610612244897959183673469387755102040 8
12	Energy	COG	37.5460816326530612244897959183673469387 8
13	Energy	COP	51.6514693877551020408163265306122448979 6
14	Energy	CVX	75.1493877551020408163265306122448979591 8
15	Energy	DNR	15.5480816326530612244897959183673469387 8

When we run same query by using CUBE function, CUBE returned 977 rows. We found that CUBE aggregates at every level unlike ROLLUP and generated grouping sets for all possible combinations of dimensions(columns). CUBE result set shows aggregates for all combinations of values in the selected columns.

SQL Query:

```
SELECT
   a.gics_sector,
   a.ticker_symbol,
   AVG(b.open)
FROM
   dberndt.sp500_stocks a,
   dberndt.sp500_eod_stocks b
WHERE
   a.ticker_symbol=b.ticker_symbol
   AND
   a.gics_sector is NOT NULL
GROUP BY CUBE(a.gics_sector, a.ticker_symbol) ;
```

Output screenshot:



	GICS_SECTOR	TICKER_SYMBOL	AVG(B.OPEN)
1	(null)	(null)	43.39009575120228628659742188376891928061
2	(null)	A	30.23485714285714285714285714285714285714
3	(null)	C	4.045306122448979591836734693877551020 41
4	(null)	D	38.686653061224489795918367346938775510 2
5	(null)	F	10.70322448979591836734693877551020408163
6	(null)	K	52.178653061224489795918367346938775510 2
7	(null)	L	35.65338775510204081632653061224489795918
8	(null)	M	19.17118367346938775510204081632653061224
9	(null)	R	41.06493877551020408163265306122448979592
10	(null)	S	3.983306122448979591836734693877551020 41
11	(null)	T	26.03844897959183673469387755102040816327
12	(null)	V	82.42636363636363636363636363636363636364
13	(null)	X	49.53946938775510204081632653061224489796
14	(null)	AA	13.08697959183673469387755102040816326531
15	(null)	AN	19.43808163265306122448979591836734693878

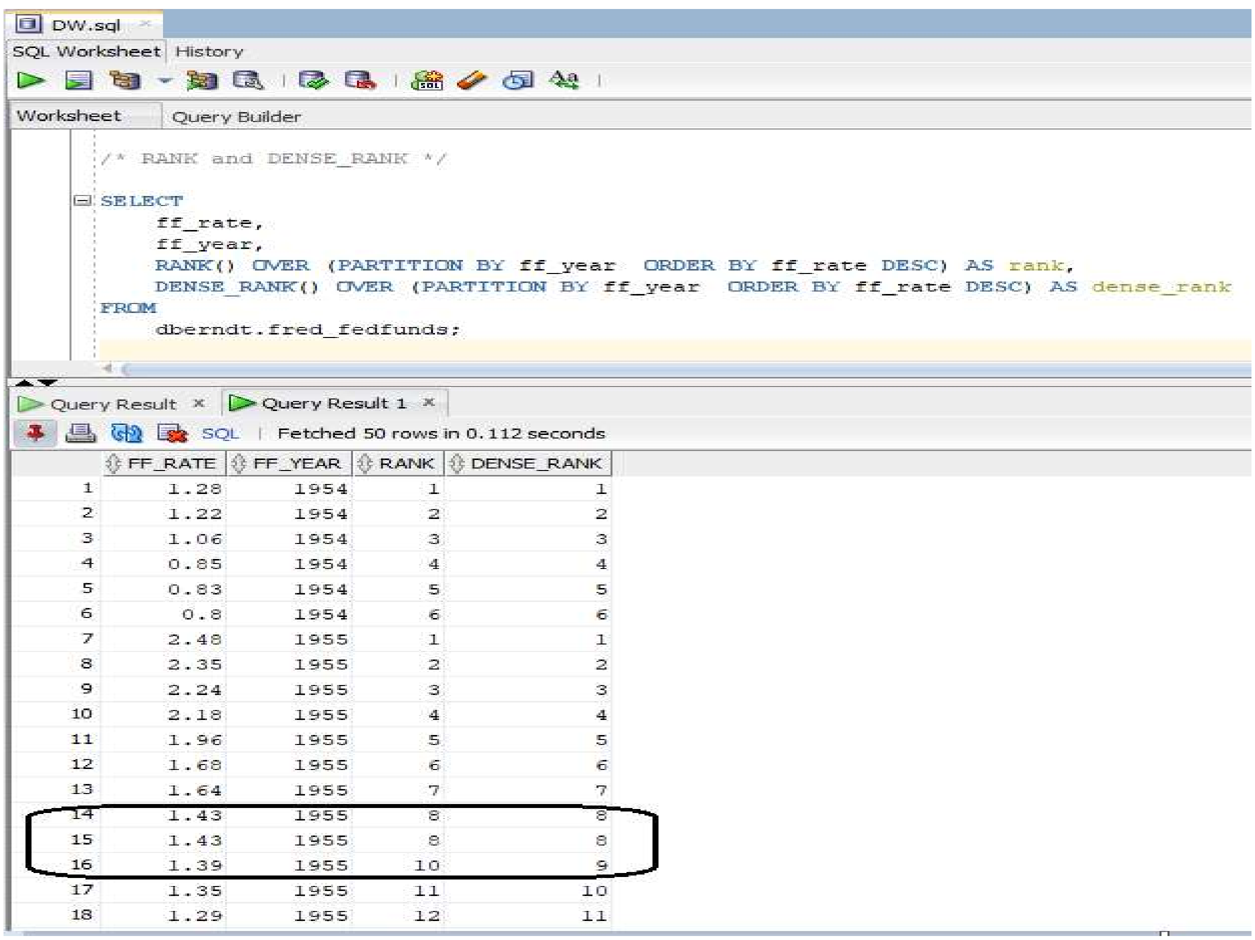
Query 2: Computing RANKs

This query computes a RANK and DENSE_RANK. We have Assigned a RANK and DENSE_RANK to each year based on the federal funds rate (using the FRED_FEDFUNDS data). We observed that RANK gives the value 10 after two consecutive 8's and DENSE_RANK gives value 9 after two 8's.

SQL Query:

```
SELECT
   ff_rate,
   ff_year,
   RANK() OVER (PARTITION BY ff_year  ORDER BY ff_rate DESC) AS rank,
   DENSE_RANK() OVER (PARTITION BY ff_year  ORDER BY ff_rate DESC) AS dense_rank
FROM
   dberndt.fred_fedfunds;
```

Output screenshot:




```
/* RANK and DENSE_RANK */

SELECT
     ff_rate,
     ff_year,
     RANK() OVER (PARTITION BY ff_year  ORDER BY ff_rate DESC) AS rank,
     DENSE_RANK() OVER (PARTITION BY ff_year  ORDER BY ff_rate DESC) AS dense_rank
FROM
     dberndt.fred_fedfunds;
```

	FF_RATE	FF_YEAR	RANK	DENSE_RANK
1	1.28	1954	1	1
2	1.22	1954	2	2
3	1.06	1954	3	3
4	0.85	1954	4	4
5	0.83	1954	5	5
6	0.8	1954	6	6
7	2.48	1955	1	1
8	2.35	1955	2	2
9	2.24	1955	3	3
10	2.18	1955	4	4
11	1.96	1955	5	5
12	1.68	1955	6	6
13	1.64	1955	7	7
14	1.43	1955	8	8
15	1.43	1955	8	8
16	1.39	1955	10	9
17	1.35	1955	11	10
18	1.29	1955	12	11

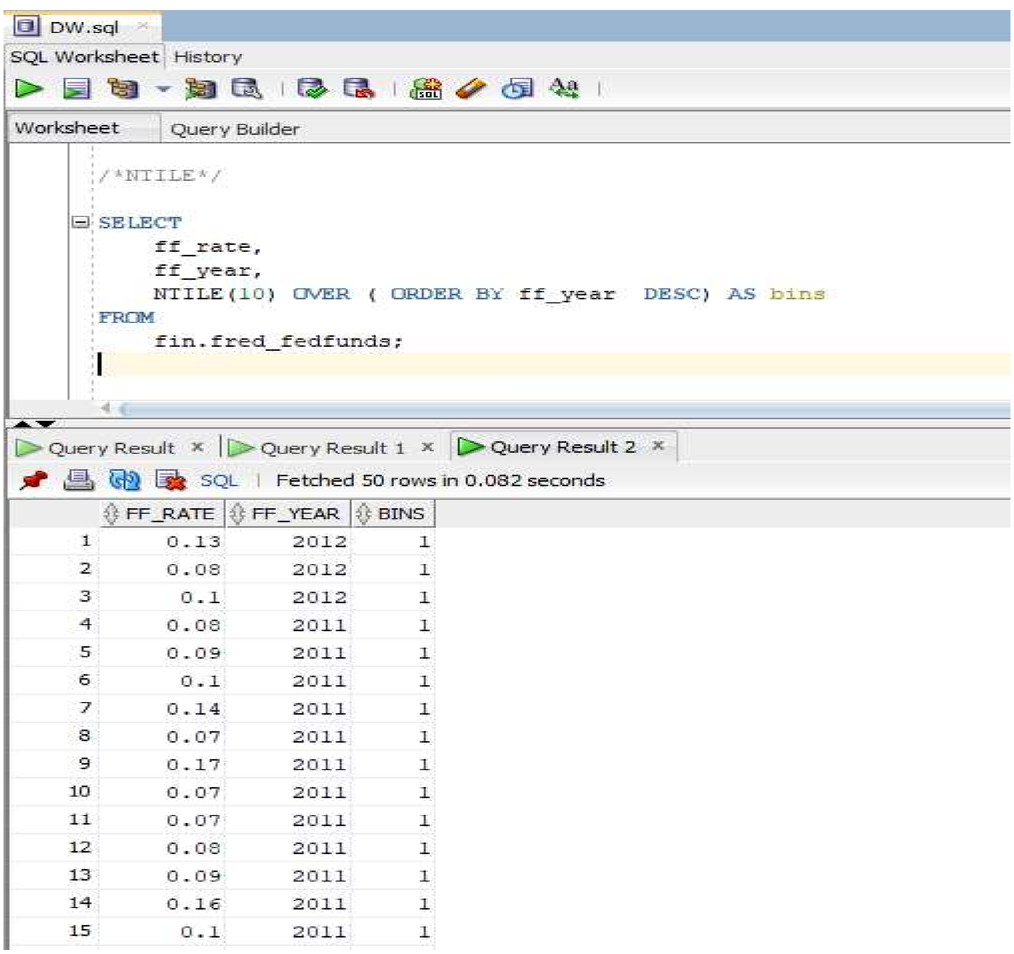
Query 3: Creating Bins with NTILE

This query uses NTILE to create deciles that bin the different years. The following example divides the values in the year column of the fin.fred_fedfunds table into 10 buckets. The year column has total 693 values, so the three extra values (the remainder of 693 / 10) are allocated to buckets 1, 2 and 3, which therefore have one more value than buckets 4-10.

SQL Query:

```
SELECT
   ff_rate,
   ff_year,
   NTILE(10) OVER ( ORDER BY ff_year  DESC) AS bins
FROM
   fin.fred_fedfunds;
```

Output screenshot:

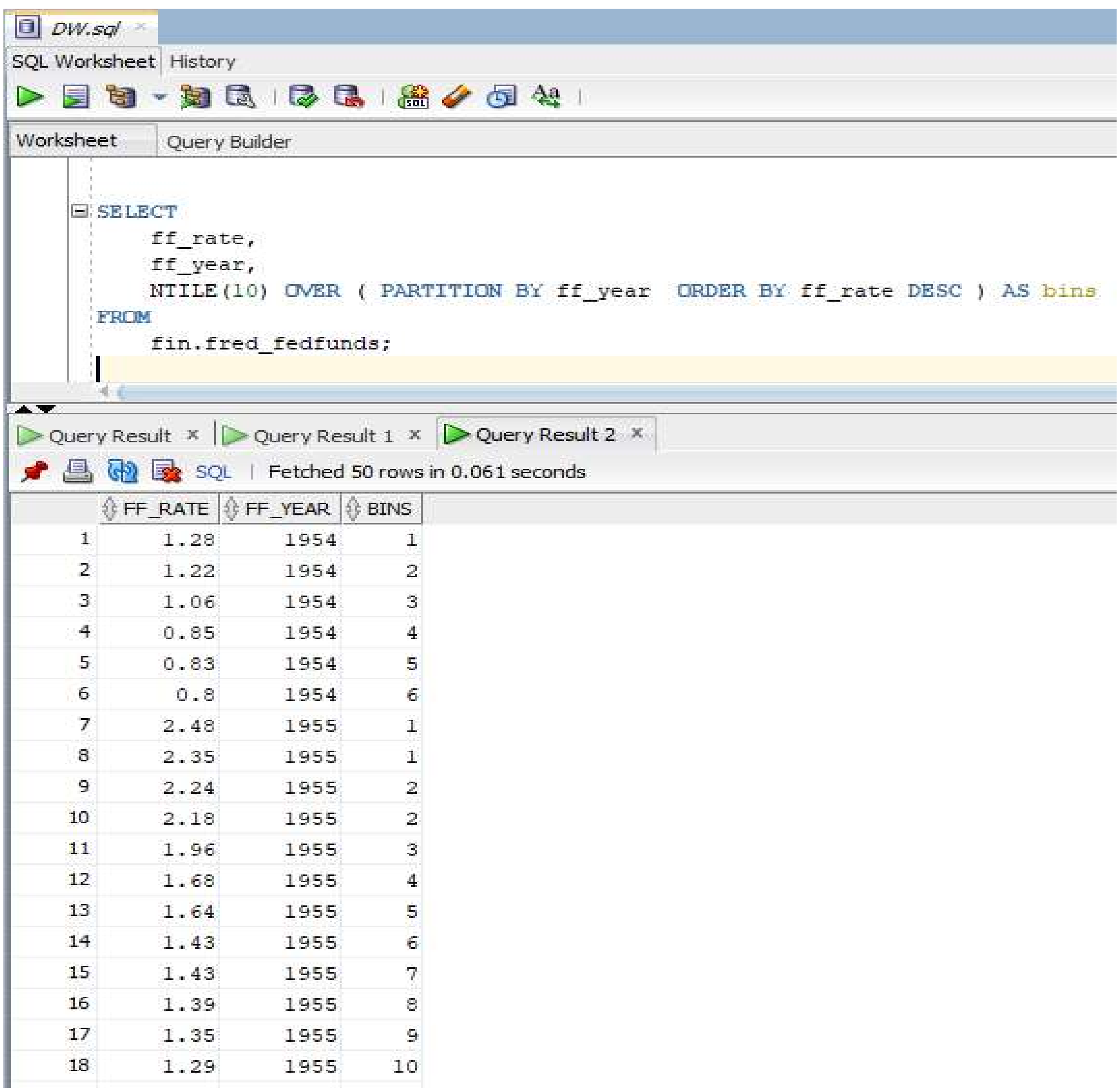


In this second query we grouped the data by year and used NTILE within the partitions to assign values.

SQL Query:

```
SELECT
   ff_rate,
   ff_year,
   NTILE(10) OVER ( PARTITION BY ff_year  ORDER BY ff_rate DESC ) AS bins
FROM
   fin.fred_fedfunds;
```

Output screenshot:



	FF_RATE	FF_YEAR	BINS
1	1.28	1954	1
2	1.22	1954	2
3	1.06	1954	3
4	0.85	1954	4
5	0.83	1954	5
6	0.8	1954	6
7	2.48	1955	1
8	2.35	1955	1
9	2.24	1955	2
10	2.18	1955	2
11	1.96	1955	3
12	1.68	1955	4
13	1.64	1955	5
14	1.43	1955	6
15	1.43	1955	7
16	1.39	1955	8
17	1.35	1955	9
18	1.29	1955	10

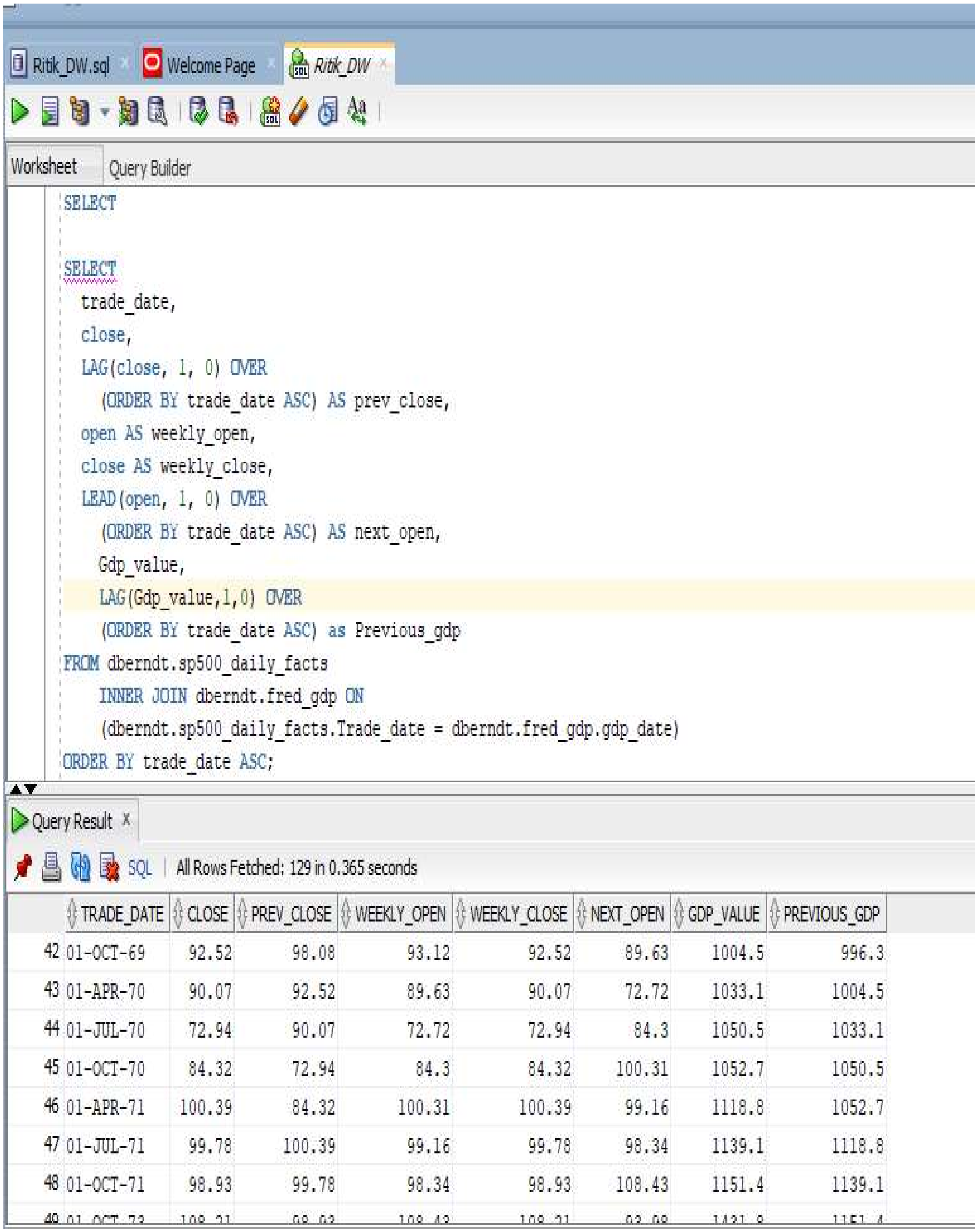
Query 5: Leading and Lagging Indicators

This Query to select stock open, stock close, previous close(lag), Nextopen(lead) from sp500 data along with its impact on gdp, with previous gdp into account.

SQL Query:

```
SELECT
    trade_date,
    close,
    LAG(close, 1, 0) OVER (ORDER BY trade_date ASC) AS prev_close,
    open AS weekly_open,
    close AS weekly_close,
    LEAD(open, 1, 0) OVER (ORDER BY trade_date ASC) AS next_open,
    Gdp_value,
    LAG(Gdp_value,1,0) OVER (ORDER BY trade_date ASC) AS Previous_gdp
FROM
    dberndt.sp500_daily_facts
    INNER JOIN dberndt.fred_gdp ON
    (dberndt.sp500_daily_facts.Trade_date = dberndt.fred_gdp.gdp_date)
ORDER BY
    trade_date ASC;
```

Output screenshot:




```sql
SELECT

SELECT
  trade_date,
  close,
  LAG(close, 1, 0) OVER
    (ORDER BY trade_date ASC) AS prev_close,
  open AS weekly_open,
  close AS weekly_close,
  LEAD(open, 1, 0) OVER
    (ORDER BY trade_date ASC) AS next_open,
    Gdp_value,
    LAG(Gdp_value,1,0) OVER
    (ORDER BY trade_date ASC) as Previous_gdp
FROM dberndt.sp500_daily_facts
    INNER JOIN dberndt.fred_gdp ON
    (dberndt.sp500_daily_facts.Trade_date = dberndt.fred_gdp.gdp_date)
ORDER BY trade_date ASC;
```

Query Result

SQL | All Rows Fetched: 129 in 0.365 seconds

	TRADE_DATE	CLOSE	PREV_CLOSE	WEEKLY_OPEN	WEEKLY_CLOSE	NEXT_OPEN	GDP_VALUE	PREVIOUS_GDP
42	01-OCT-69	92.52	98.08	93.12	92.52	89.63	1004.5	996.3
43	01-APR-70	90.07	92.52	89.63	90.07	72.72	1033.1	1004.5
44	01-JUL-70	72.94	90.07	72.72	72.94	84.3	1050.5	1033.1
45	01-OCT-70	84.32	72.94	84.3	84.32	100.31	1052.7	1050.5
46	01-APR-71	100.39	84.32	100.31	100.39	99.16	1118.8	1052.7
47	01-JUL-71	99.78	100.39	99.16	99.78	98.34	1139.1	1118.8
48	01-OCT-71	98.93	99.78	98.34	98.93	108.43	1151.4	1139.1

Query 8: Superstore Product Correlations

This query will find correlation between sales amount and orders sold for all years.

SQL Query:

```
SELECT
        d.month_num,
        CORR (SUM(s.sales),
        SUM(s.order_quantity)) OVER (ORDER BY d.month_num) as CUM_CORR
 FROM
         superstore.sales_fact s,
         superstore.date_dim d
 WHERE
        s.order_date_key = d.date_key
 GROUP BY
        d.month_num
 ORDER BY
        d.month_num;
```

Output screenshot:

```
SELECT d.month_num,CORR (SUM(s.sales), SUM(s.order_quantity)) OVER (ORDER BY d.month_num) as CUM_CORR
   FROM superstore.sales_fact s, superstore.date_dim d
      WHERE s.order_date_key = d.date_key --AND d.year_num = 2007
      GROUP BY d.month_num
      ORDER BY d.month_num;
```

Script Output × | Query Result × | Query Result 1 × | Query Result 2 ×

SQL | All Rows Fetched: 12 in 0.069 seconds

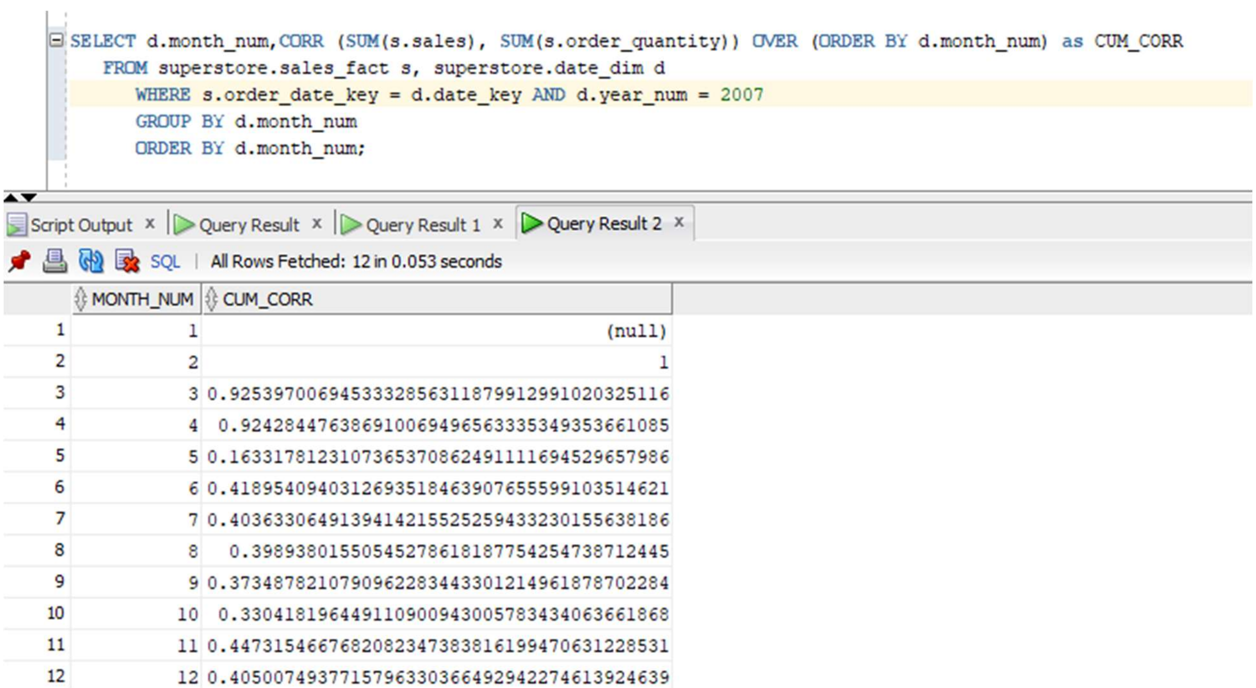
	MONTH_NUM	CUM_CORR
1	1	(null)
2	2	1
3	3	0.9287299039196861886828779175969933642093
4	4	0.8521906961723612142501811971593111354775
5	5	-0.5061617311250947501192764643823162473857
6	6	-0.0894867116172802660399600280156769461888
7	7	-0.0937176372266114689743920121466376875335
8	8	-0.1046717333494684630685396889983316325625
9	9	-0.0228832523831835461443131816159078201371
10	10	0.0532582933419837754344589876731333787644
11	11	0.1456586607036591729252829426278411103299
12	12	0.1962357877093206706012963733968439947368

Now, in this second query we have added year condition limiting the results to one calendar year (say 2007).

SQL Query:

```
SELECT
        d.month_num,
        CORR (SUM(s.sales),
        SUM(s.order_quantity)) OVER (ORDER BY d.month_num) as CUM_CORR
FROM
        superstore.sales_fact s,
        superstore.date_dim d
WHERE
        s.order_date_key = d.date_key AND d.year_num = 2007
GROUP BY
        d.month_num
ORDER BY
        d.month_num;
```

Output screenshot:



```
SELECT d.month_num,CORR (SUM(s.sales), SUM(s.order_quantity)) OVER (ORDER BY d.month_num) as CUM_CORR
  FROM superstore.sales_fact s, superstore.date_dim d
    WHERE s.order_date_key = d.date_key AND d.year_num = 2007
    GROUP BY d.month_num
    ORDER BY d.month_num;
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 12 in 0.053 seconds

	MONTH_NUM	CUM_CORR
1	1	(null)
2	2	1
3	3	0.925397006945333285631187991299102032511 6
4	4	0.924284476386910069496563335349353661085
5	5	0.16331781231073653708624911116945296579 86
6	6	0.41895409403126935184639076555991035146 21
7	7	0.40363306491394142155252594332301556381 86
8	8	0.398938015505452786181877542547387124 45
9	9	0.37348782107909622834433012149618787022 84
10	10	0.330418196449110900943005783434063661868
11	11	0.44731546676820823473838161994706312285 31
12	12	0.40500749377157963303664929422746139246 39

PART II: Interesting Queries

Query 1:

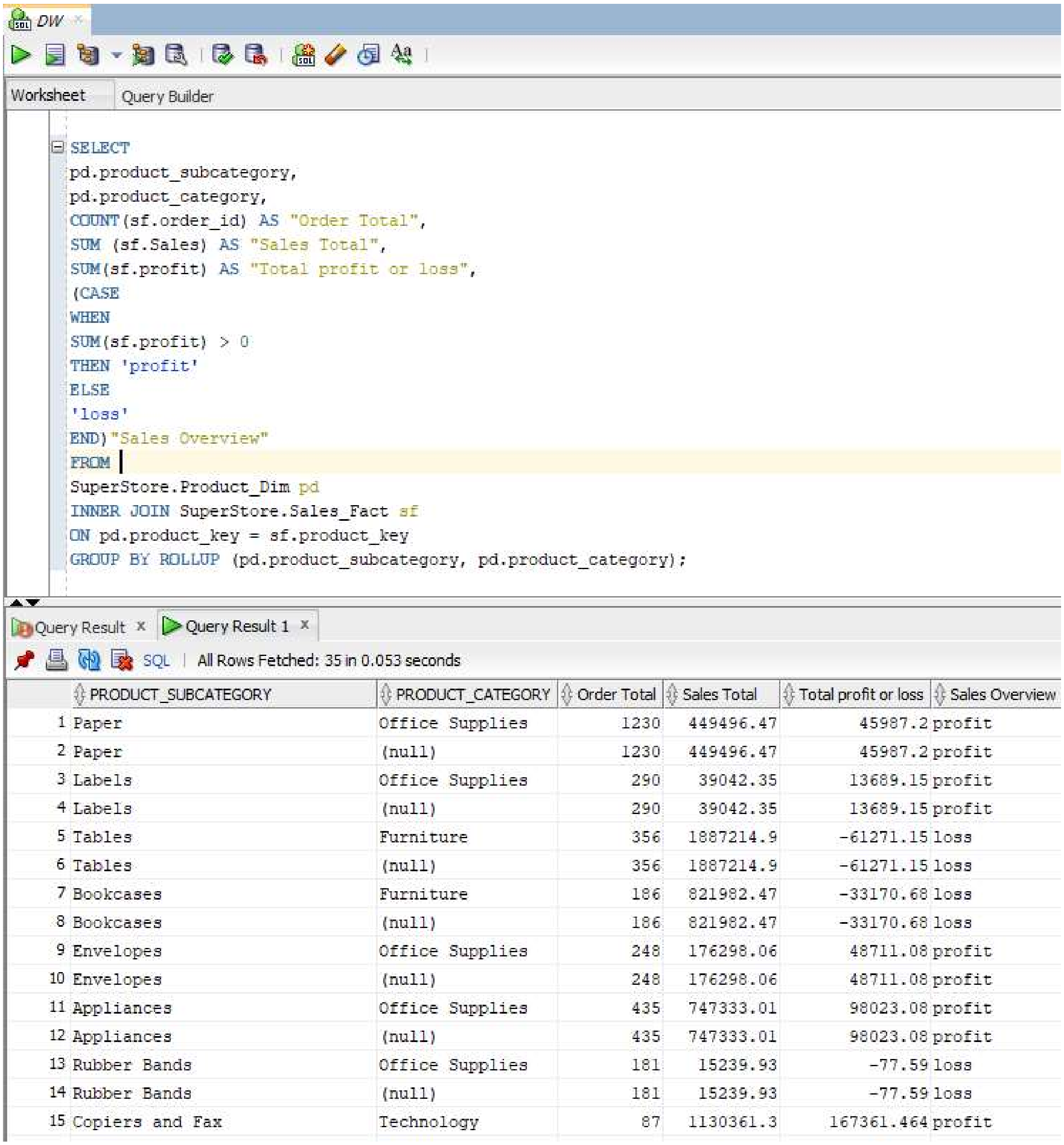
This query is giving us insight about the revenue generated from different type of product category and their sub categories.

Here the column "Sales Overview" tells us whether there has been a profit or loss in the store.

SQL Query:

```
SELECT
    pd.product_subcategory,
    pd.product_category,
    COUNT(sf.order_id) AS "Order Total",
    SUM (sf.Sales) AS "Sales Total",
    SUM(sf.profit) AS "Total profit or loss",
    (CASE
        WHEN
            SUM(sf.profit) > 0
        THEN 'profit'
        ELSE 'loss'
    END)"Sales Overview"
FROM
    SuperStore.Product_Dim pd
    INNER JOIN SuperStore.Sales_Fact sf
    ON pd.product_key = sf.product_key
GROUP BY ROLLUP ((pd.product_subcategory, pd.product_category);
```

Output screenshot:



DW

Worksheet | Query Builder

```sql
SELECT
pd.product_subcategory,
pd.product_category,
COUNT(sf.order_id) AS "Order Total",
SUM (sf.Sales) AS "Sales Total",
SUM(sf.profit) AS "Total profit or loss",
(CASE
WHEN
SUM(sf.profit) > 0
THEN 'profit'
ELSE
'loss'
END)"Sales Overview"
FROM
SuperStore.Product_Dim pd
INNER JOIN SuperStore.Sales_Fact sf
ON pd.product_key = sf.product_key
GROUP BY ROLLUP (pd.product_subcategory, pd.product_category);
```

Query Result × | Query Result 1 ×

SQL | All Rows Fetched: 35 in 0.053 seconds

	PRODUCT_SUBCATEGORY	PRODUCT_CATEGORY	Order Total	Sales Total	Total profit or loss	Sales Overview
1	Paper	Office Supplies	1230	449496.47	45987.2	profit
2	Paper	(null)	1230	449496.47	45987.2	profit
3	Labels	Office Supplies	290	39042.35	13689.15	profit
4	Labels	(null)	290	39042.35	13689.15	profit
5	Tables	Furniture	356	1887214.9	-61271.15	loss
6	Tables	(null)	356	1887214.9	-61271.15	loss
7	Bookcases	Furniture	186	821982.47	-33170.68	loss
8	Bookcases	(null)	186	821982.47	-33170.68	loss
9	Envelopes	Office Supplies	248	176298.06	48711.08	profit
10	Envelopes	(null)	248	176298.06	48711.08	profit
11	Appliances	Office Supplies	435	747333.01	98023.08	profit
12	Appliances	(null)	435	747333.01	98023.08	profit
13	Rubber Bands	Office Supplies	181	15239.93	-77.59	loss
14	Rubber Bands	(null)	181	15239.93	-77.59	loss
15	Copiers and Fax	Technology	87	1130361.3	167361.464	profit

Query 2:

BI Scenario: To find out what factors are driving profits in some states and how these factors can be incorporated into other states.

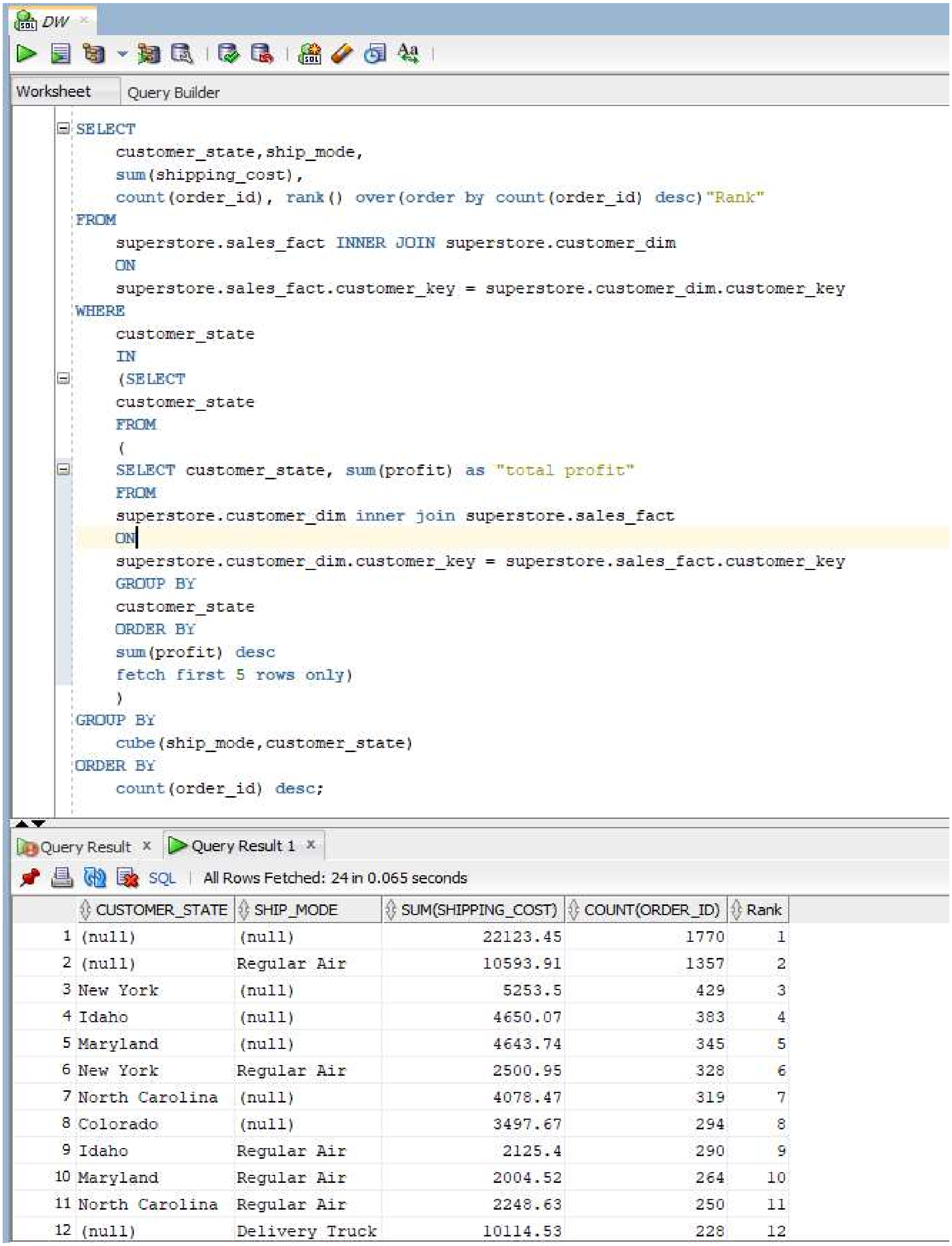
Is shipping cost affecting profits: Below is the analytical query for that.

In order to do so, In the query first 5 states are found out with high profits which is then cubed with shipping details

SQL Query:

```
SELECT
        customer_state,
        ship_mode,
        SUM(shipping_cost),
        COUNT(order_id),
        RANK() over(order by count(order_id) desc)"Rank"
FROM
        superstore.sales_fact INNER JOIN superstore.customer_dim
        ON
        superstore.sales_fact.customer_key = superstore.customer_dim.customer_key
WHERE
        customer_state
        IN
        (SELECT
        customer_state
        FROM
         (
                SELECT customer_state, sum(profit) as "total profit"
                FROM
                superstore.customer_dim inner join superstore.sales_fact
                ON
                superstore.customer_dim.customer_key = superstore.sales_fact.customer_key
                GROUP BY
                customer_state
                ORDER BY
                SUM(profit) DESC
                fetch first 5 rows only)
         )
GROUP BY
         CUBE(ship_mode,customer_state)
ORDER BY
        COUNT(order_id) DESC;
```

Output screenshot:



DW

Worksheet Query Builder

```sql
SELECT
    customer_state,ship_mode,
    sum(shipping_cost),
    count(order_id), rank() over(order by count(order_id) desc)"Rank"
FROM
    superstore.sales_fact INNER JOIN superstore.customer_dim
    ON
    superstore.sales_fact.customer_key = superstore.customer_dim.customer_key
WHERE
    customer_state
    IN
    (SELECT
    customer_state
    FROM
    (
    SELECT customer_state, sum(profit) as "total profit"
    FROM
    superstore.customer_dim inner join superstore.sales_fact
    ON
    superstore.customer_dim.customer_key = superstore.sales_fact.customer_key
    GROUP BY
    customer_state
    ORDER BY
    sum(profit) desc
    fetch first 5 rows only)
    )
GROUP BY
    cube(ship_mode,customer_state)
ORDER BY
    count(order_id) desc;
```

Query Result × Query Result 1 ×

SQL | All Rows Fetched: 24 in 0.065 seconds

	CUSTOMER_STATE	SHIP_MODE	SUM(SHIPPING_COST)	COUNT(ORDER_ID)	Rank
1	(null)	(null)	22123.45	1770	1
2	(null)	Regular Air	10593.91	1357	2
3	New York	(null)	5253.5	429	3
4	Idaho	(null)	4650.07	383	4
5	Maryland	(null)	4643.74	345	5
6	New York	Regular Air	2500.95	328	6
7	North Carolina	(null)	4078.47	319	7
8	Colorado	(null)	3497.67	294	8
9	Idaho	Regular Air	2125.4	290	9
10	Maryland	Regular Air	2004.52	264	10
11	North Carolina	Regular Air	2248.63	250	11
12	(null)	Delivery Truck	10114.53	228	12

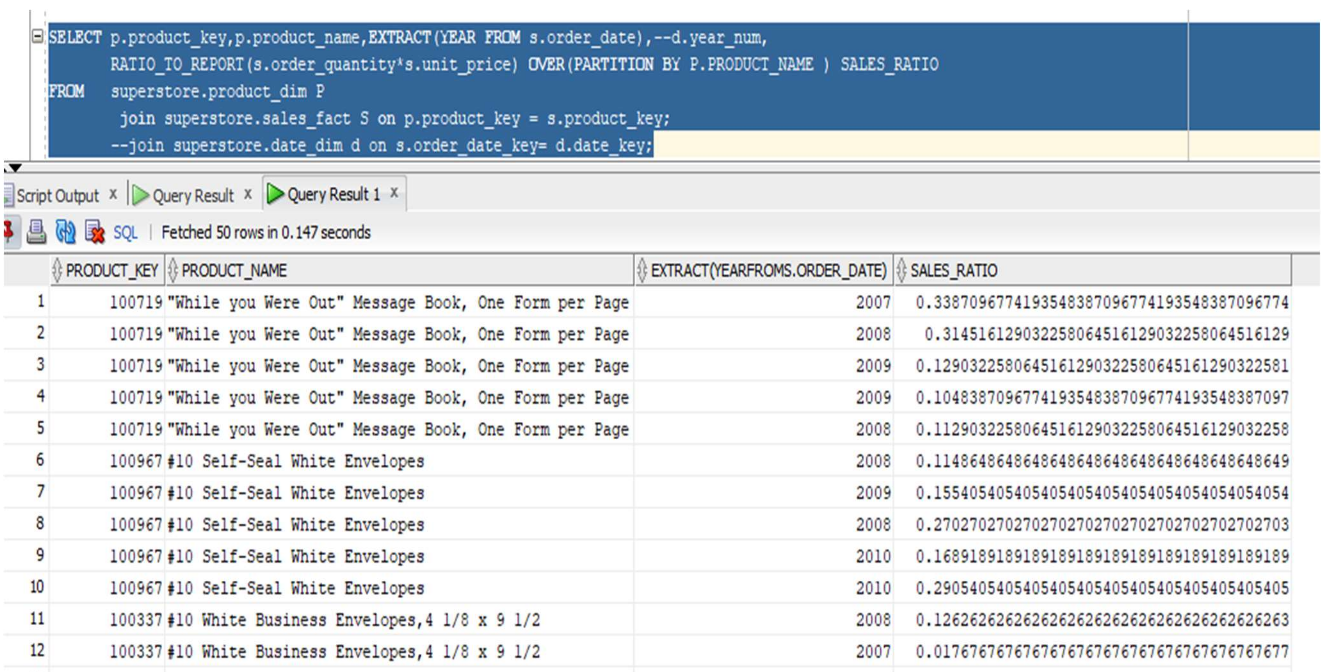
Query 3:

This query will find the correlations between products and categories within sales.

SQL Query:

SELECT
    p.product_key,
    p.product_name,
    EXTRACT(YEAR FROM s.order_date),
    RATIO_TO_REPORT(s.order_quantity*s.unit_price)
    OVER(PARTITION BY p.product_name) SALES_RATIO
FROM
    superstore.product_dim p
    JOIN superstore.sales_fact s
      ON p.product_key = s.product_key;

Output screenshot:



```
SELECT p.product_key,p.product_name,EXTRACT(YEAR FROM s.order_date),--d.year_num,
       RATIO_TO_REPORT(s.order_quantity*s.unit_price) OVER(PARTITION BY P.PRODUCT_NAME ) SALES_RATIO
FROM   superstore.product_dim P
        join superstore.sales_fact S on p.product_key = s.product_key;
       --join superstore.date_dim d on s.order_date_key= d.date_key;
```

Script Output | Query Result | Query Result 1

SQL | Fetched 50 rows in 0.147 seconds

	PRODUCT_KEY	PRODUCT_NAME	EXTRACT(YEARFROMS.ORDER_DATE)	SALES_RATIO
1	100719	"While you Were Out" Message Book, One Form per Page	2007	0.3387096774193548387096774193548387096774
2	100719	"While you Were Out" Message Book, One Form per Page	2008	0.314516129032258064516129032258064516129
3	100719	"While you Were Out" Message Book, One Form per Page	2009	0.1290322580645161290322580645161290322581
4	100719	"While you Were Out" Message Book, One Form per Page	2009	0.1048387096774193548387096774193548387097
5	100719	"While you Were Out" Message Book, One Form per Page	2008	0.1129032258064516129032258064516129032258
6	100967	#10 Self-Seal White Envelopes	2008	0.1148648648648648648648648648648648648649
7	100967	#10 Self-Seal White Envelopes	2009	0.1554054054054054054054054054054054054054
8	100967	#10 Self-Seal White Envelopes	2008	0.2702702702702702702702702702702702702703
9	100967	#10 Self-Seal White Envelopes	2010	0.1689189189189189189189189189189189189189
10	100967	#10 Self-Seal White Envelopes	2010	0.2905405405405405405405405405405405405405
11	100337	#10 White Business Envelopes,4 1/8 x 9 1/2	2008	0.1262626262626262626262626262626262626263
12	100337	#10 White Business Envelopes,4 1/8 x 9 1/2	2007	0.0176767676767676767676767676767676767677
13	100337	#10 White Business Envelopes,4 1/8 x 9 1/2	2009	0.0580808080808080808080808080808080808081

Query 4:

This query will display the number of sales yearly that are more than average of sales for all years.

SQL Query:

```
SELECT
        p.product_name,
        EXTRACT(YEAR FROM s.order_date),
        s.order_quantity
FROM
        superstore.product_dim p
         JOIN superstore.sales_fact s
           ON p.product_key = s.product_key
WHERE
        s.order_quantity >  (SELECT avg(order_quantity)
                              FROM superstore.sales_fact Sub
                              WHERE sub.product_key = s.product_key);
```

Output screenshot:

```
SELECT p.product_name, EXTRACT(YEAR FROM s.order_date),s.order_quantity
FROM   superstore.product_dim p
       join superstore.sales_fact S on p.product_key = s.product_key
WHERE  s.order_quantity >  (select avg(order_quantity) from superstore.sales_fact Sub
                            where sub.product_key = s.product_key);
```

Script Output × | Query Result × | Query Result 1 × | Query Result 2 ×

SQL | Fetched 50 rows in 0.131 seconds

	PRODUCT_NAME	EXTRACT(YEARFROMS.ORDER_DATE)	ORDER_QUANTITY
4	Deflect-o EconoMat Studded, No Bevel Mat for Low Pile Carpeting	2008	35
5	Holmes Replacement Filter for HEPA Air Cleaner, Very Large Room, HEPA Filter	2010	29
6	Holmes Replacement Filter for HEPA Air Cleaner, Very Large Room, HEPA Filter	2008	27
7	Sharp EL500L Fraction Calculator	2009	42
8	Hon Rectangular Conference Tables	2007	45
9	OIC Colored Binder Clips, Assorted Sizes	2010	32
10	OIC Colored Binder Clips, Assorted Sizes	2007	30
11	Fellowes Stor/Drawer® Steel Plus™ Storage Drawers	2008	48
12	Memorex 4.7GB DVD+RW, 3/Pack	2010	43
13	Memorex 4.7GB DVD+RW, 3/Pack	2008	42
14	Memorex 4.7GB DVD+RW, 3/Pack	2010	50
15	Imation 3.5", RTS 247544 3M 3.5 DSDD, 10/Pack	2008	37
16	Wilson Jones 14 Line Acrylic Coated Pressboard Data Binders	2007	38