# SQL Server Functions

# SQL Server Functions

- **Aggregate Functions**
- Date Functions
- String Functions
- System Functions
- Window Functions

Σ

Aggregate functions are built in SQL Server functions and applied to sets of records rather than to a single record

<null>

Except for COUNT(*), aggregate functions ignore null values.

Aggregate functions are often used with the GROUP BY clause of the SELECT statement.

# SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

| Aggregate function | Description |
|---|---|
| AVG | Calculates the average of non-NULL values in a set. |
| COUNT | Returns the number of rows in a group, including rows with NULL values. |
| MAX | Returns the highest value (maximum) in a set of non-NULL values |
| MIN | Returns the lowest value (minimum) in a set of non-NULL values. |
| SUM | Returns the summation of all non-NULL values a set. |

## SQL Server Functions

- **Aggregate Functions**
- Date Functions
- String Functions
- System Functions
- Window Functions

| Aggregate function | Description |
|---|---|
| CHECKSUM_AGG | Calculates a checksum value based on a group of rows. |
| COUNT_BIG | Returns the number of rows (with BIGINT data type) in a group, including rows with NULL values |
| STDEV | Returns the statistical standard deviation of all values provided in the expression based on a sample of the data population. |
| STDEVP | Returns the standard deviation for all values in the provided expression, but does so based on the entire data population. |
| VAR | Returns the summation of all non-NULL values a set. |
| VARP | Returns the statistical variance of values in an expression but does so based on the entire data population. |

# SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

**Constructing date and time from their parts**

| Function | Description |
|---|---|
| DATEFROMPARTS | Return a DATE value from the year, month, and day |
| DATETIME2FROMPARTS | Returns a DATETIME2 value from the date and time arguments |
| DATETIMEOFFSETFROMPARTS | Returns a DATETIMEOFFSET value from the date and time arguments |
| TIMEFROMPARTS | Returns a TIME value from the time parts with the precisions |

# SQL Server Functions

- Aggregate Functions
- Date Functions
- **String Functions**
- System Functions
- Window Functions

| | |
|---|---|
| LTRIM() | Removes blanks on the left side of the character expression |
| LOWER() | Converts all characters to lower case letters |
| UPPER() | Converts all characters to upper case letters |
| REVERSE() | Reverses all the characters in the string |
| SUBSTRING() | Gives a substring from the original string |

# SQL Server Functions

- Aggregate Functions
- Date Functions
- String Functions
- **System Functions**
- Window Functions

| Function | Description |
|---|---|
| CAST | Cast a value of one type to another |
| CONVERT | Convert a value of one type to another |
| CHOOSE | Return one of the two values based on the result of the first argument |
| ISNULL | Replace NULL with a specified value |
| ISNUMERIC | Check if an expression is a valid numeric type |
| IIF | Add if-else logic to a query |

# SQL Server Functions

- Aggregate Functions
- Date Functions
- String Functions
- **System Functions**
- Window Functions

| Function | Description |
|---|---|
| TRY_CAST | Cast a value of one type to another and return NULL if the cast fails |
| TRY_CONVERT | Convert a value of one type to another and return the value to be translated into the specified type. It returns NULL if the cast fails |
| TRY_PARSE | Convert a string to a date/time or a number and return NULL if the conversion fails |
| Convert datetime to string | Show you how to convert a datetime value to a string in a specified format |
| Convert string to datetime | Describe how to convert a string to a datetime value |
| Convert datetime to date | Convert a datetime to a date |

# SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

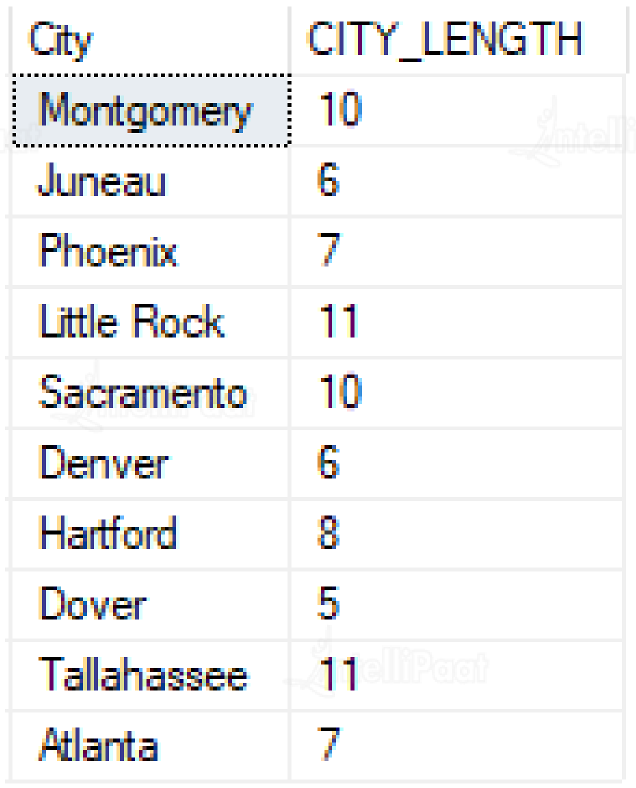
| Function | Description |
|---|---|
| CUME_DIST | Calculate the cumulative distribution of a value in a set of values |
| DENSE_RANK | Assign a rank value to each row within a partition of a result, with no gaps in rank values |
| FIRST_VALUE | Get the value of the first row in an ordered partition of a result set |
| LAG | Provide access to a row at a given physical offset that comes before the current row |
| LAST_VALUE | Get the value of the last row in an ordered partition of a result set |

# STRING FUNCTIONS

Write a query to find the length of the string in the City column

**LEN():** This function is used to find the length of a string.
**Syntax:** *LEN(string);*

```
SELECT City, LEN( City ) AS
CITY_LENGTH FROM Store_Details
```



| City | CITY_LENGTH |
|---|---|
| Montgomery | 10 |
| Juneau | 6 |
| Phoenix | 7 |
| Little Rock | 11 |
| Sacramento | 10 |
| Denver | 6 |
| Hartford | 8 |
| Dover | 5 |
| Tallahassee | 11 |
| Atlanta | 7 |

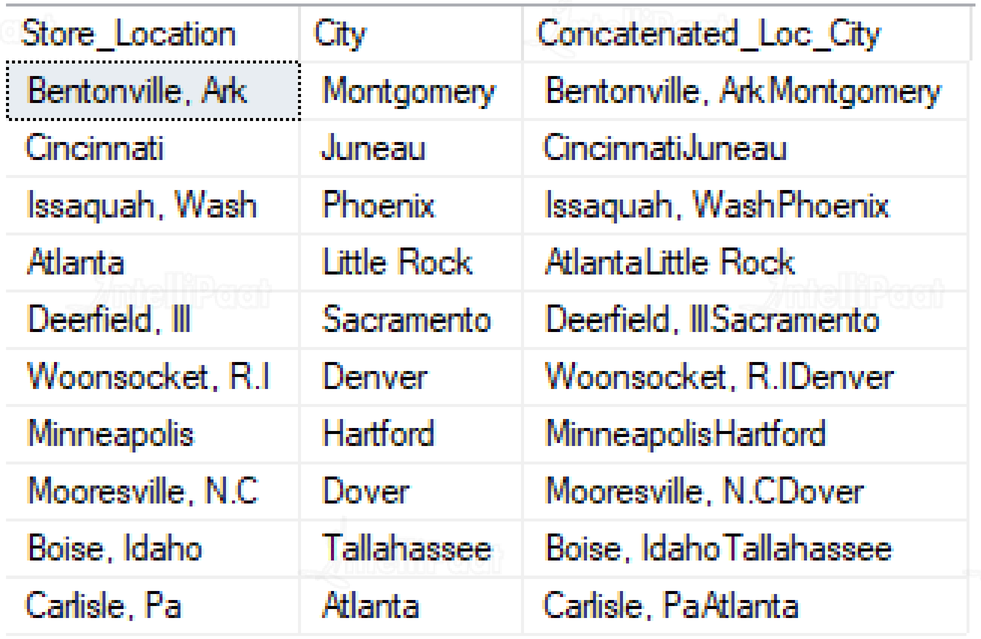
## STRING FUNCTIONS

Write a query to concatenate Store_Location column with City column and display the outputs

**CONCAT():** This function returns a string resulting from the concatenation, or joining, of two or more string values in an end-to-end manner.

**Syntax:** *CONCAT ( string1, string2,…, string_n)*

```
Select Store_Location,City,
CONCAT( Store_Location, City) as
Concatenated_Loc_City from Store_Details
```



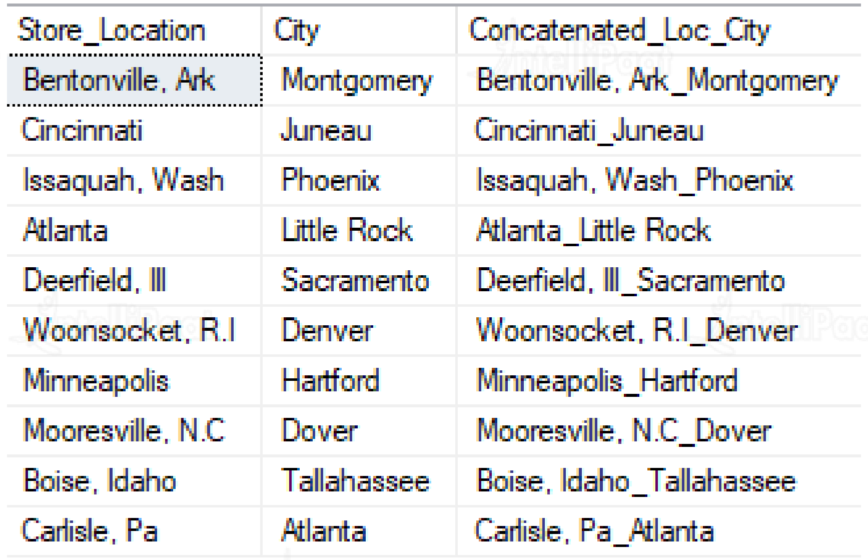
| Store_Location | City | Concatenated_Loc_City |
|---|---|---|
| Bentonville, Ark | Montgomery | Bentonville, ArkMontgomery |
| Cincinnati | Juneau | CincinnatiJuneau |
| Issaquah, Wash | Phoenix | Issaquah, WashPhoenix |
| Atlanta | Little Rock | AtlantaLittle Rock |
| Deerfield, Ill | Sacramento | Deerfield, IllSacramento |
| Woonsocket, R.I | Denver | Woonsocket, R.IDenver |
| Minneapolis | Hartford | MinneapolisHartford |
| Mooresville, N.C | Dover | Mooresville, N.CDover |
| Boise, Idaho | Tallahassee | Boise, IdahoTallahassee |
| Carlisle, Pa | Atlanta | Carlisle, PaAtlanta |

# STRING FUNCTIONS

Write a query to concatenate Store_Location column with City column and display the outputs

**CONCAT_WS():** This function is used to add two or more strings together with a separator.
**Syntax :**
*CONCAT_WS(separator,string1, string2, ...., string_n)*

```
Select Store_Location,City,
CONCAT( Store_Location, City) as
Concatenated_Loc_City from Store_Details
```



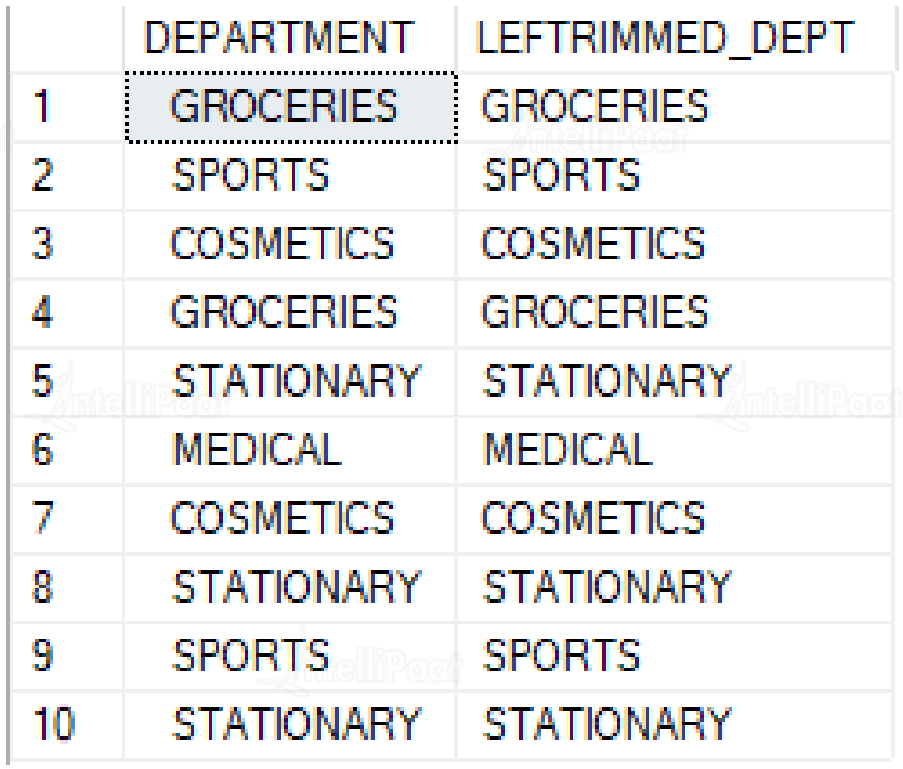
| Store_Location | City | Concatenated_Loc_City |
|---|---|---|
| Bentonville, Ark | Montgomery | Bentonville, Ark_Montgomery |
| Cincinnati | Juneau | Cincinnati_Juneau |
| Issaquah, Wash | Phoenix | Issaquah, Wash_Phoenix |
| Atlanta | Little Rock | Atlanta_Little Rock |
| Deerfield, Ill | Sacramento | Deerfield, Ill_Sacramento |
| Woonsocket, R.I | Denver | Woonsocket, R.I_Denver |
| Minneapolis | Hartford | Minneapolis_Hartford |
| Mooresville, N.C | Dover | Mooresville, N.C_Dover |
| Boise, Idaho | Tallahassee | Boise, Idaho_Tallahassee |
| Carlisle, Pa | Atlanta | Carlisle, Pa_Atlanta |

# STRING FUNCTIONS

Write a query to trim the spaces at the beginning of the Department column in the Store_Details table

**LTRIM():** This function removes leading spaces from a string
**Syntax:** *LTRIM (string)*

```
SELECT DEPARTMENT,LTRIM(DEPARTMENT)
AS LEFTRIMMED_DEPT from Store_Details;
```



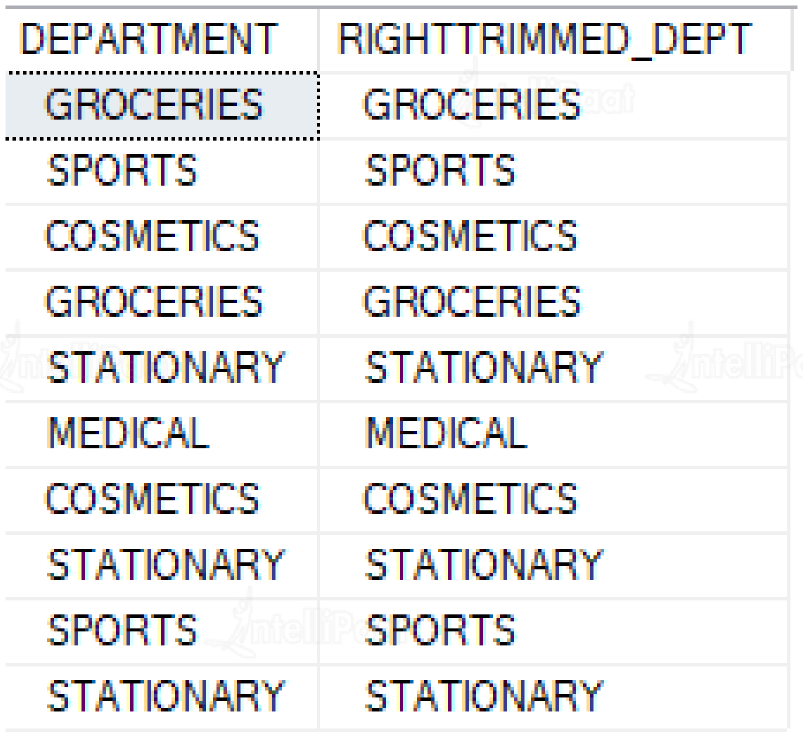
| | DEPARTMENT | LEFTRIMMED_DEPT |
|---|---|---|
| 1 | GROCERIES | GROCERIES |
| 2 | SPORTS | SPORTS |
| 3 | COSMETICS | COSMETICS |
| 4 | GROCERIES | GROCERIES |
| 5 | STATIONARY | STATIONARY |
| 6 | MEDICAL | MEDICAL |
| 7 | COSMETICS | COSMETICS |
| 8 | STATIONARY | STATIONARY |
| 9 | SPORTS | SPORTS |
| 10 | STATIONARY | STATIONARY |

# STRING FUNCTIONS

Write a query to trim the spaces at the beginning of the Department column in the Store_Details table

**RTRIM():** This function removes space character char(32) or other specified characters from the end of a string.
**Syntax:** *RTRIM (string)*

```
SELECT  DEPARTMENT,RTRIM(DEPARTMENT) AS
RIGHTTRIMMED_DEPT from Store_Details;
```



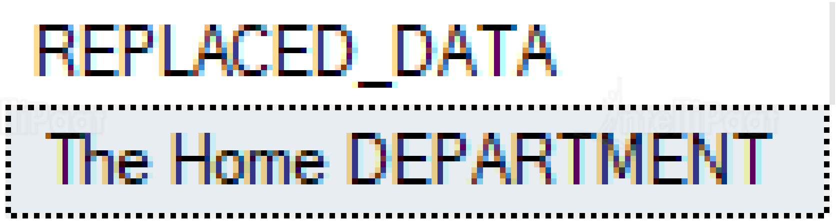
| DEPARTMENT | RIGHTTRIMMED_DEPT |
|---|---|
| GROCERIES | GROCERIES |
| SPORTS | SPORTS |
| COSMETICS | COSMETICS |
| GROCERIES | GROCERIES |
| STATIONARY | STATIONARY |
| MEDICAL | MEDICAL |
| COSMETICS | COSMETICS |
| STATIONARY | STATIONARY |
| SPORTS | SPORTS |
| STATIONARY | STATIONARY |

## STRING FUNCTIONS

Write a query to replace the substring Depot with Department in The Home Depot from the Department column in the Store_Details table.

**REPLACE():** Replaces all occurrences of a specified string value with another string value.
**Syntax:** *REPLACE ( string_expression , string_pattern , string_replacement )*

```
SELECT REPLACE('TheHomeDepot','Depot','DEPARTMENT')
AS REPLACED_DATA;
```



```
REPLACED_DATA
The Home DEPARTMENT
```

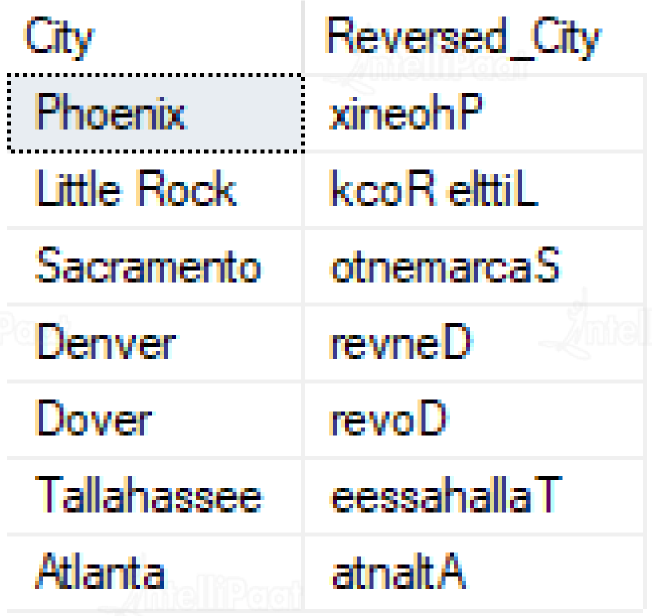
## STRING FUNCTIONS

Write a query to display the reversed values of City column in Store_details table

**REVERSE():** Returns the reverse order of a string value.

**Syntax:** *REVERSE ( string_expression )*

```
SELECT REPLACE('The Home Depot','Depot','DEPARTMENT')
AS REPLACED_DATA;
```



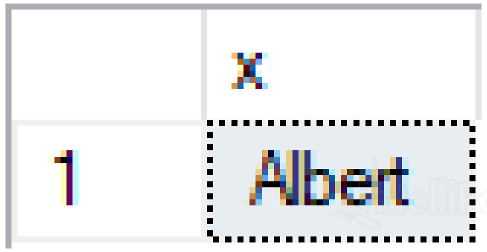
| City | Reversed_City |
|---|---|
| Phoenix | xineohP |
| Little Rock | kcoR elttiL |
| Sacramento | otnemarcaS |
| Denver | revneD |
| Dover | revoD |
| Tallahassee | eessahallaT |
| Atlanta | atnaltA |

## STRING FUNCTIONS

Write a query to return the substring 'ALBERT' from 'Albertsons Companies' value from the Store_name column of the Store_details table.

**SUBSTRING():** Returns part of a character, binary, text, or image expression in SQL Server.
**Syntax:** ***SUBSTRING ( expression ,start , length )***

```
SELECT x = SUBSTRING('Albertsons Companies', 1, 6);
```



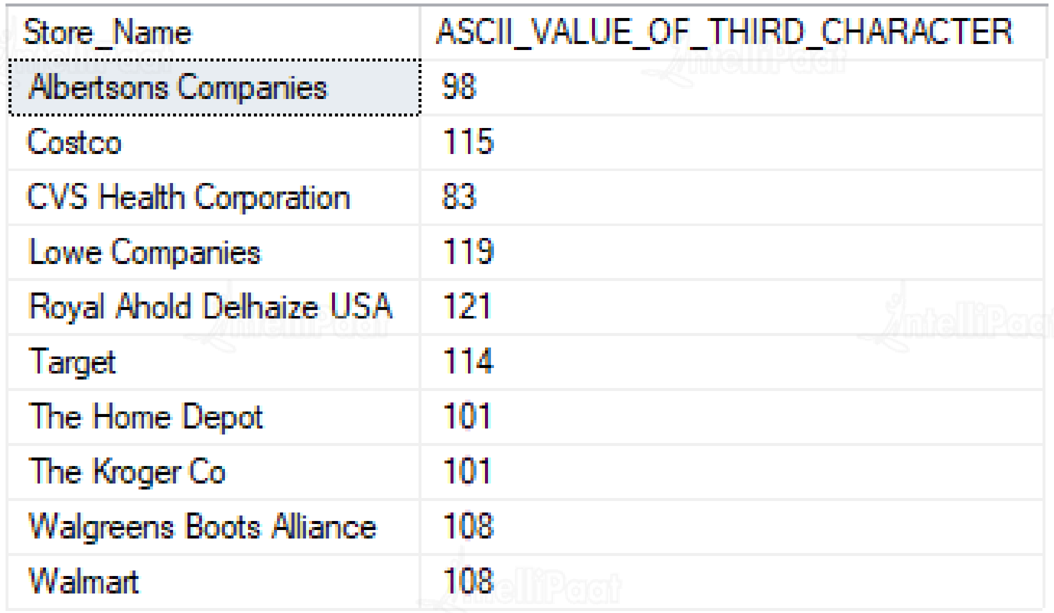
|   | x |
|---|---|
| 1 | Albert |

# STRING FUNCTIONS

Display the ASCII value of the third character from the column Store_Name

**ASCII():** This function is used to find the ASCII value of a character.
**SYNTAX:** *SELECT ASCII(character);*

```
SELECT Store_Name,
(ASCII(SUBSTRING(Store_Name,3,1)))
AS
ASCII_VALUE_OF_THIRD_CHARACTER
FROM Store_Details
```



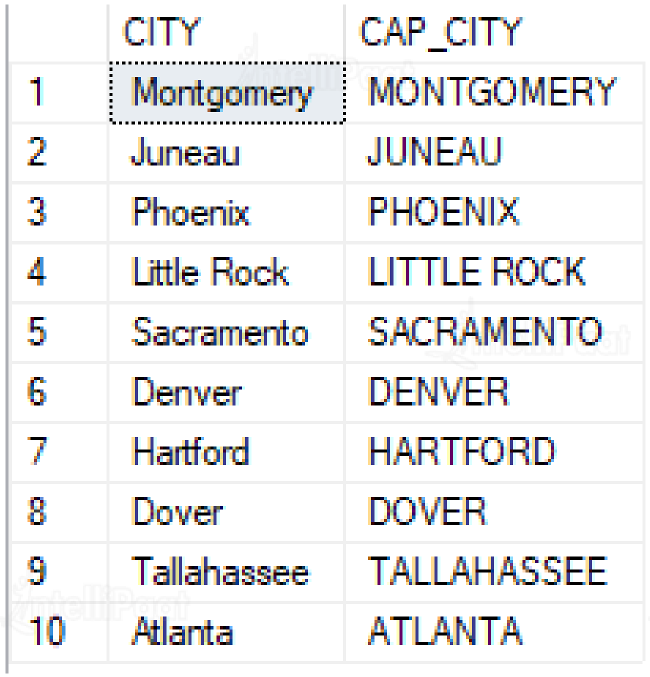
| Store_Name | ASCII_VALUE_OF_THIRD_CHARACTER |
|---|---|
| Albertsons Companies | 98 |
| Costco | 115 |
| CVS Health Corporation | 83 |
| Lowe Companies | 119 |
| Royal Ahold Delhaize USA | 121 |
| Target | 114 |
| The Home Depot | 101 |
| The Kroger Co | 101 |
| Walgreens Boots Alliance | 108 |
| Walmart | 108 |

## STRING FUNCTIONS

Write a query to convert the values in the City column from the Store_details table into uppercase.

**UPPER():** Returns a character expression with lowercase character data converted to uppercase.
**Syntax:** ***UPPER ( character_expression )***

```
SELECT CITY, UPPER(CITY) AS CAP_CITY  FROM Store_Details
```



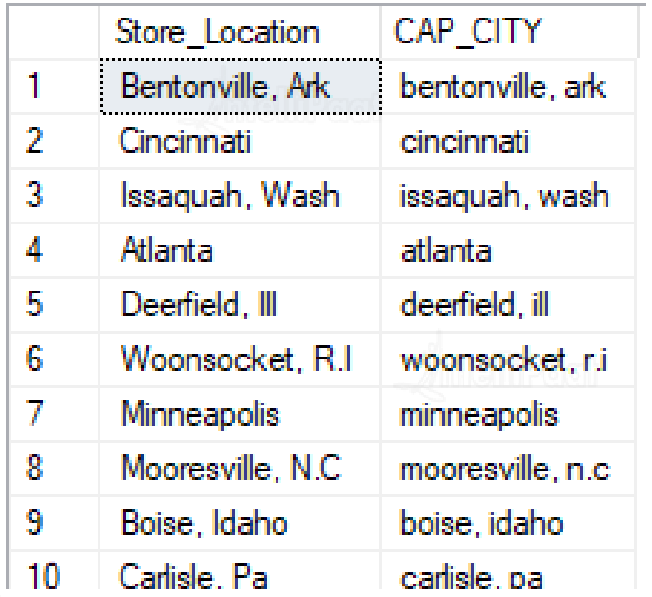
| | CITY | CAP_CITY |
|---|---|---|
| 1 | Montgomery | MONTGOMERY |
| 2 | Juneau | JUNEAU |
| 3 | Phoenix | PHOENIX |
| 4 | Little Rock | LITTLE ROCK |
| 5 | Sacramento | SACRAMENTO |
| 6 | Denver | DENVER |
| 7 | Hartford | HARTFORD |
| 8 | Dover | DOVER |
| 9 | Tallahassee | TALLAHASSEE |
| 10 | Atlanta | ATLANTA |

# STRING FUNCTIONS

Write a query to convert the values in the Store_location column from the Store_details table into lowercase.

**LOWER():** Returns a character expression after converting uppercase character data to lowercase.
**Syntax:** *LOWER ( character_expression )*

```
SELECT Store_Location, LOWER(Store_Location) AS CAP_CITY
FROM Store_Details
```

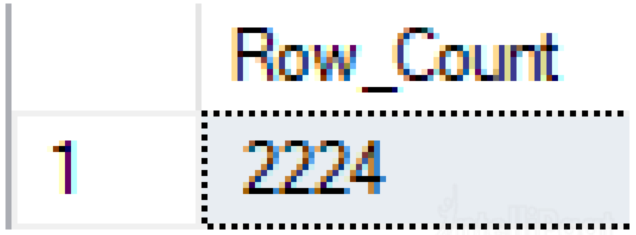


| | Store_Location | CAP_CITY |
|---|---|---|
| 1 | Bentonville, Ark | bentonville, ark |
| 2 | Cincinnati | cincinnati |
| 3 | Issaquah, Wash | issaquah, wash |
| 4 | Atlanta | atlanta |
| 5 | Deerfield, Ill | deerfield, ill |
| 6 | Woonsocket, R.I | woonsocket, r.i |
| 7 | Minneapolis | minneapolis |
| 8 | Mooresville, N.C | mooresville, n.c |
| 9 | Boise, Idaho | boise, idaho |
| 10 | Carlisle, Pa | carlisle, pa |

## MATHEMATICAL FUNCTIONS

Count the number of rows where the temperature is greater than 50 and unemployment is more than 8.

**COUNT():** This function returns the number of items found in a group. COUNT always returns an int data type value.
**Syntax:** ***SELECT COUNT(column_name) FROM table_name WHERE condition;***



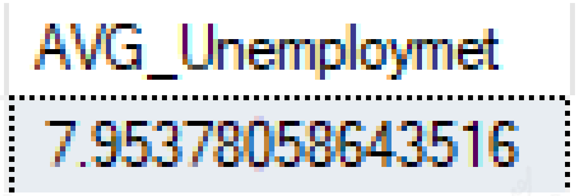
|   | Row_Count |
|---|---|
| 1 | 2224 |

```
Select Count(*) as Row_Count from Features where
Temperature>50.00  and Unemployment>8.00
```

# MATHEMATICAL FUNCTIONS

Calculate the average of Unemployment where temperature is more than 60 and fuel price is more than 3.

**AVG()** - This function returns the average value of a numeric column.
**Syntax** - ***SELECT AVG(column_name) FROM table_name WHERE condition;***



```
AVG_Unemploymet
7.95378058643516
```

```
Select AVG(Unemployment) as AVG_Unemploymet from
Features where Temperature>60 and Fuel_Price>3
```

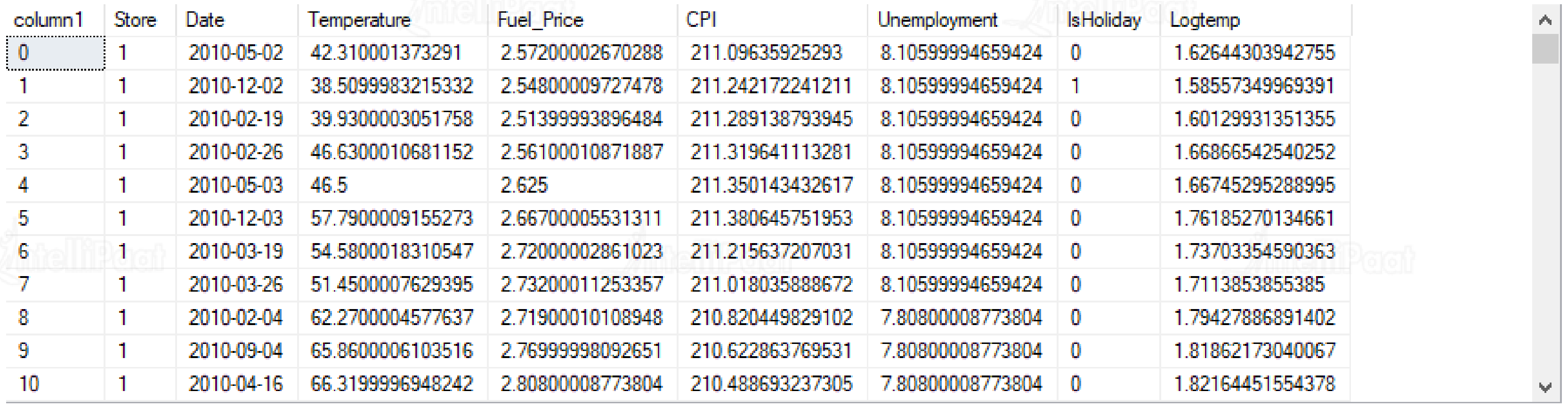
# MATHEMATICAL FUNCTIONS

Create a new column with the name "LogTemp" which will be holding the log10 value of the Temperature Column.

**LOG10():** Returns the base-10 logarithm of the specified expression

**Syntax:** *LOG10 ( expression )*

Alter table Features Add Logtemp as log10(Temperature)



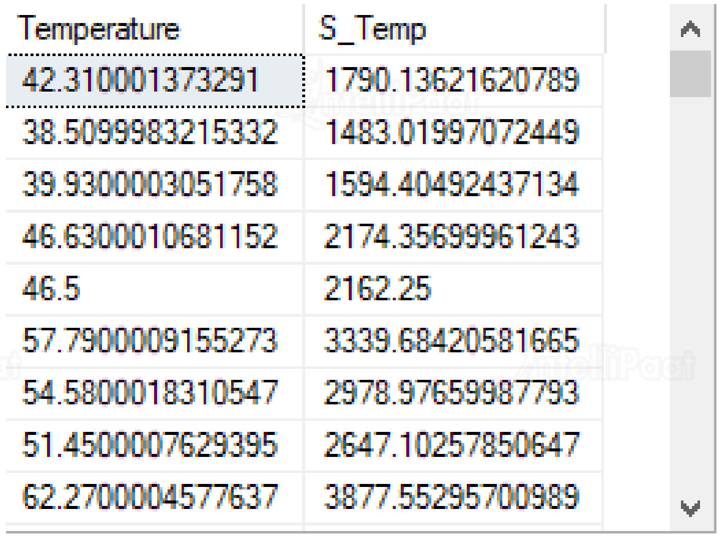
| column1 | Store | Date | Temperature | Fuel_Price | CPI | Unemployment | IsHoliday | Logtemp |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2010-05-02 | 42.310001373291 | 2.57200002670288 | 211.09635925293 | 8.10599994659424 | 0 | 1.62644303942755 |
| 1 | 1 | 2010-12-02 | 38.5099983215332 | 2.54800009727478 | 211.242172241211 | 8.10599994659424 | 1 | 1.58557349969391 |
| 2 | 1 | 2010-02-19 | 39.9300003051758 | 2.51399993896484 | 211.289138793945 | 8.10599994659424 | 0 | 1.60129931351355 |
| 3 | 1 | 2010-02-26 | 46.6300010681152 | 2.56100010871887 | 211.319641113281 | 8.10599994659424 | 0 | 1.66866542540252 |
| 4 | 1 | 2010-05-03 | 46.5 | 2.625 | 211.350143432617 | 8.10599994659424 | 0 | 1.66745295288995 |
| 5 | 1 | 2010-12-03 | 57.7900009155273 | 2.66700005531311 | 211.380645751953 | 8.10599994659424 | 0 | 1.76185270134661 |
| 6 | 1 | 2010-03-19 | 54.5800018310547 | 2.72000002861023 | 211.215637207031 | 8.10599994659424 | 0 | 1.73703354590363 |
| 7 | 1 | 2010-03-26 | 51.4500007629395 | 2.73200011253357 | 211.018035888672 | 8.10599994659424 | 0 | 1.7113853855385 |
| 8 | 1 | 2010-02-04 | 62.2700004577637 | 2.71900010108948 | 210.820449829102 | 7.80800008773804 | 0 | 1.79427886891402 |
| 9 | 1 | 2010-09-04 | 65.8600006103516 | 2.76999998092651 | 210.622863769531 | 7.80800008773804 | 0 | 1.81862173040067 |
| 10 | 1 | 2010-04-16 | 66.3199996948242 | 2.80800008773804 | 210.488693237305 | 7.80800008773804 | 0 | 1.82164451554378 |

# MATHEMATICAL FUNCTIONS

Create a query to fetch the squared values of the Temperature Column.

**SQUARE():** Returns the square of a value.
**Syntax:** ***SQUARE (number)***

```
Select Square(Temperature) as S_Temp from
Features
```



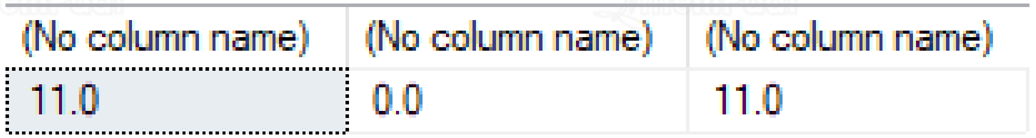
| Temperature | S_Temp |
|---|---|
| 42.310001373291 | 1790.13621620789 |
| 38.5099983215332 | 1483.01997072449 |
| 39.9300003051758 | 1594.40492437134 |
| 46.6300010681152 | 2174.35699961243 |
| 46.5 | 2162.25 |
| 57.7900009155273 | 3339.68420581665 |
| 54.5800018310547 | 2978.97659987793 |
| 51.4500007629395 | 2647.10257850647 |
| 62.2700004577637 | 3877.55295700989 |

# MATHEMATICAL FUNCTIONS

Write a query to fetch the absolute value of a given numbers: –11.0, 0.0, 11.0

**ABS():** A mathematical function that returns the absolute (positive) value of the specified numeric expression.
**Syntax:** ***ABS ( numeric_expression )***

```
SELECT ABS(-11.0), ABS(0.0), ABS(11.0)
```



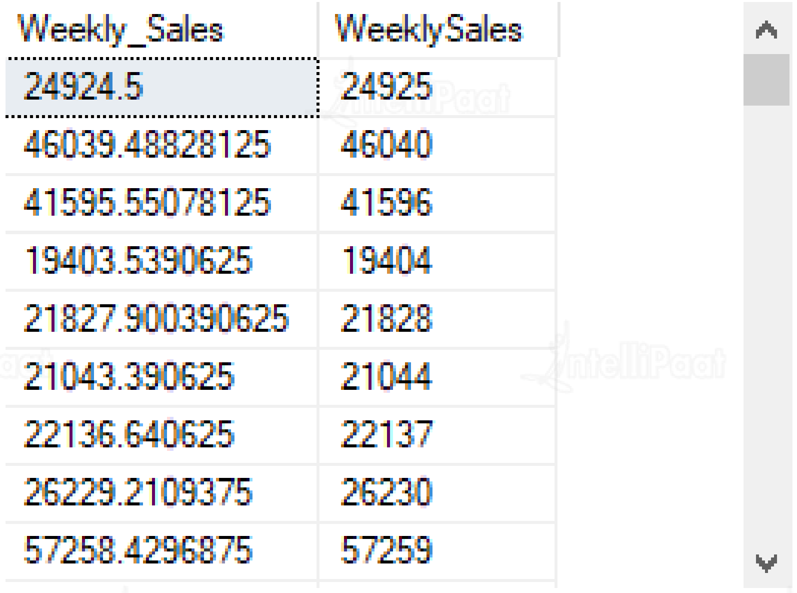
| (No column name) | (No column name) | (No column name) |
|---|---|---|
| 11.0 | 0.0 | 11.0 |

## MATHEMATICAL FUNCTIONS

Write a query to check the application of the ceiling function on the Weekly_sales column from the Sales table.

**CEILING():** This function returns the smallest integer greater than, or equal to, the specified numeric expression.
**Syntax:** *CEILING ( numeric_expression )*

```
SELECT Weekly_Sales,CEILING(Weekly_Sales)as WeeklySales
from sales
```



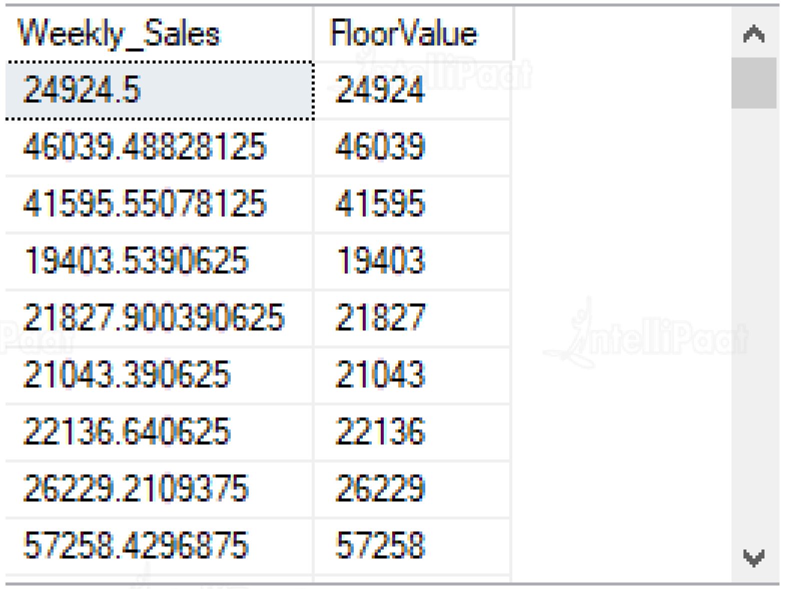
| Weekly_Sales | WeeklySales |
|---|---|
| 24924.5 | 24925 |
| 46039.48828125 | 46040 |
| 41595.55078125 | 41596 |
| 19403.5390625 | 19404 |
| 21827.900390625 | 21828 |
| 21043.390625 | 21044 |
| 22136.640625 | 22137 |
| 26229.2109375 | 26230 |
| 57258.4296875 | 57259 |

# MATHEMATICAL FUNCTIONS

Write a query to check the application of the ceiling function on the Weekly_sales column from the Sales table.

**FLOOR():** Returns the largest integer less than or equal to the specified numeric expression.
**Syntax:** *FLOOR ( numeric_expression )*

```
SELECT Weekly_Sales,FLOOR(Weekly_Sales) AS FloorValue
from sales
```



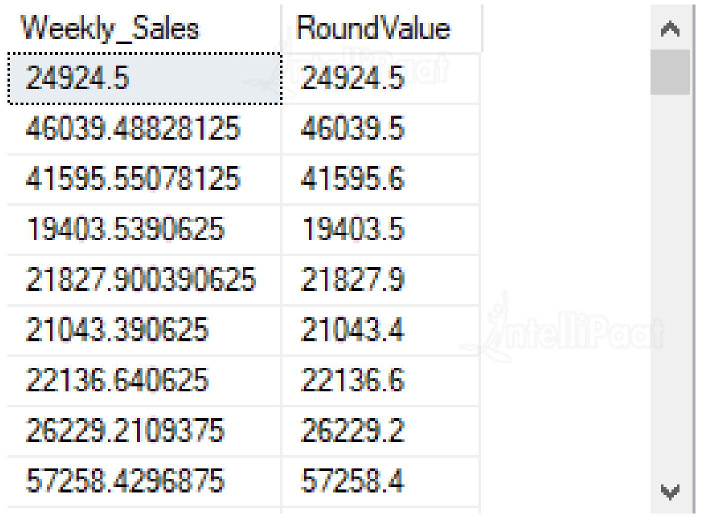
| Weekly_Sales | FloorValue |
|---|---|
| 24924.5 | 24924 |
| 46039.48828125 | 46039 |
| 41595.55078125 | 41595 |
| 19403.5390625 | 19403 |
| 21827.900390625 | 21827 |
| 21043.390625 | 21043 |
| 22136.640625 | 22136 |
| 26229.2109375 | 26229 |
| 57258.4296875 | 57258 |

# MATHEMATICAL FUNCTIONS

Write a query to check the application of the round function on the Weekly_sales column from the Sales table.

**ROUND():** Returns a numeric value, rounded to the specified length or precision.
**Syntax:** *ROUND (number,decimal,operation )*

```
SELECT Weekly_Sales, Round(Weekly_Sales,1) AS
RoundValue from sales
```



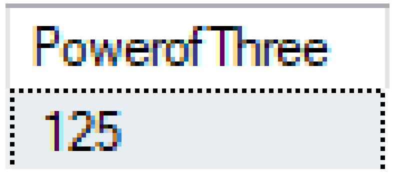
| Weekly_Sales | RoundValue |
|---|---|
| 24924.5 | 24924.5 |
| 46039.48828125 | 46039.5 |
| 41595.55078125 | 41595.6 |
| 19403.5390625 | 19403.5 |
| 21827.900390625 | 21827.9 |
| 21043.390625 | 21043.4 |
| 22136.640625 | 22136.6 |
| 26229.2109375 | 26229.2 |
| 57258.4296875 | 57258.4 |

# MATHEMATICAL FUNCTIONS

Write a query to check the application of the power function using any random value.

**POWER():** Returns the value of the specified expression to the specified power.
**Syntax:** *POWER ( float_expression , y )*

```
SELECT POWER(5, 3)as PowerofThree
```



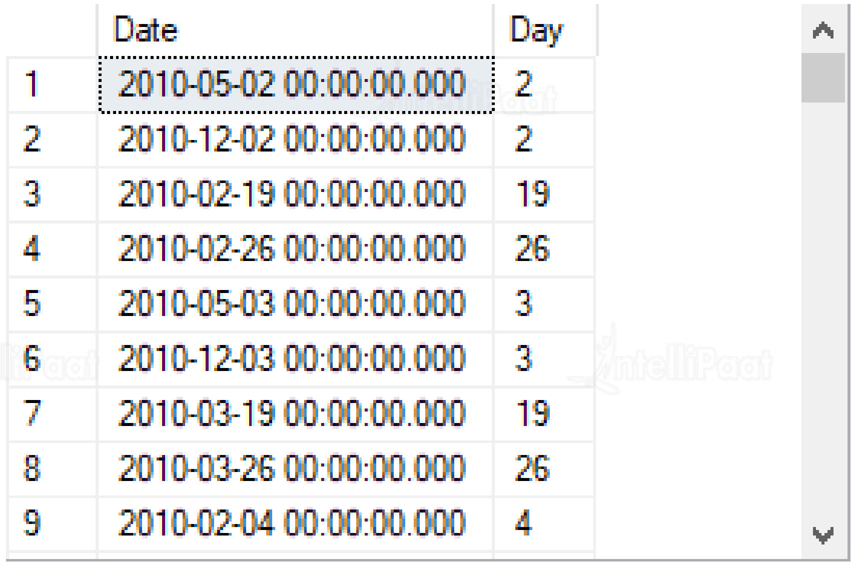
```
PowerofThree
125
```

# DATE TIME FUNCTIONS

Write a query to fetch only the day from the date column

**Day():** Returns the day of a specified date as an integer
**Syntax:** ***DAY ( date)***

```
Select Date, Day([Date]) as Day from features
```



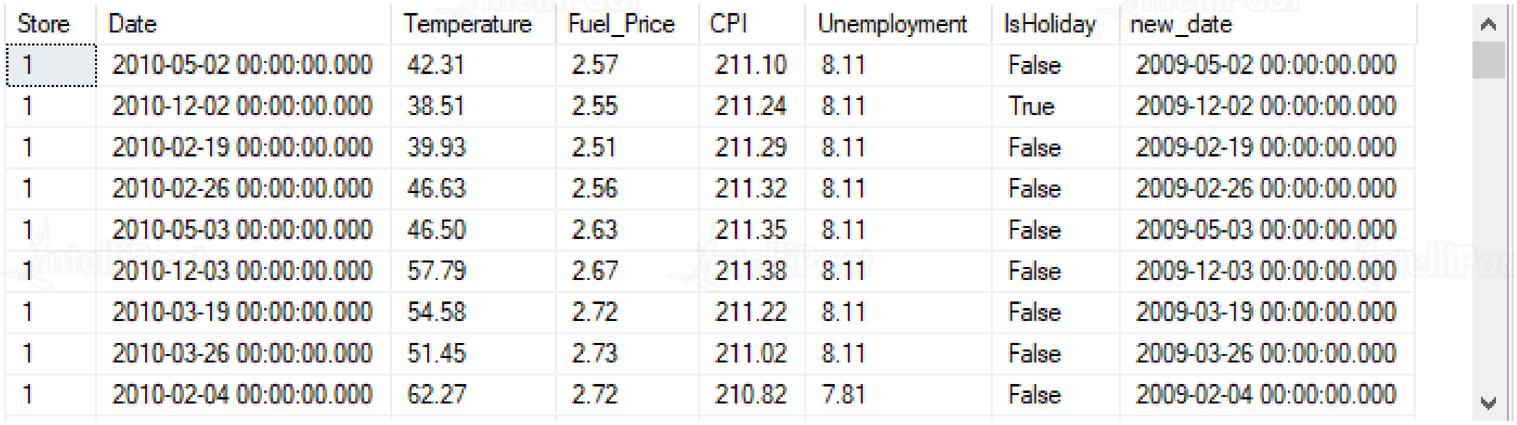
| | Date | Day |
|---|---|---|
| 1 | 2010-05-02 00:00:00.000 | 2 |
| 2 | 2010-12-02 00:00:00.000 | 2 |
| 3 | 2010-02-19 00:00:00.000 | 19 |
| 4 | 2010-02-26 00:00:00.000 | 26 |
| 5 | 2010-05-03 00:00:00.000 | 3 |
| 6 | 2010-12-03 00:00:00.000 | 3 |
| 7 | 2010-03-19 00:00:00.000 | 19 |
| 8 | 2010-03-26 00:00:00.000 | 26 |
| 9 | 2010-02-04 00:00:00.000 | 4 |

# DATE TIME FUNCTIONS

Write a query that subtracts a 1 year time interval from the date column

**Dateadd():** Adds a value to a date part of a date and returns the new date value
**Syntax:** *DATEADD (datepart , number , date )*

```
SELECT *, DATEADD(Year,-1,[Date])as new_date from features
```



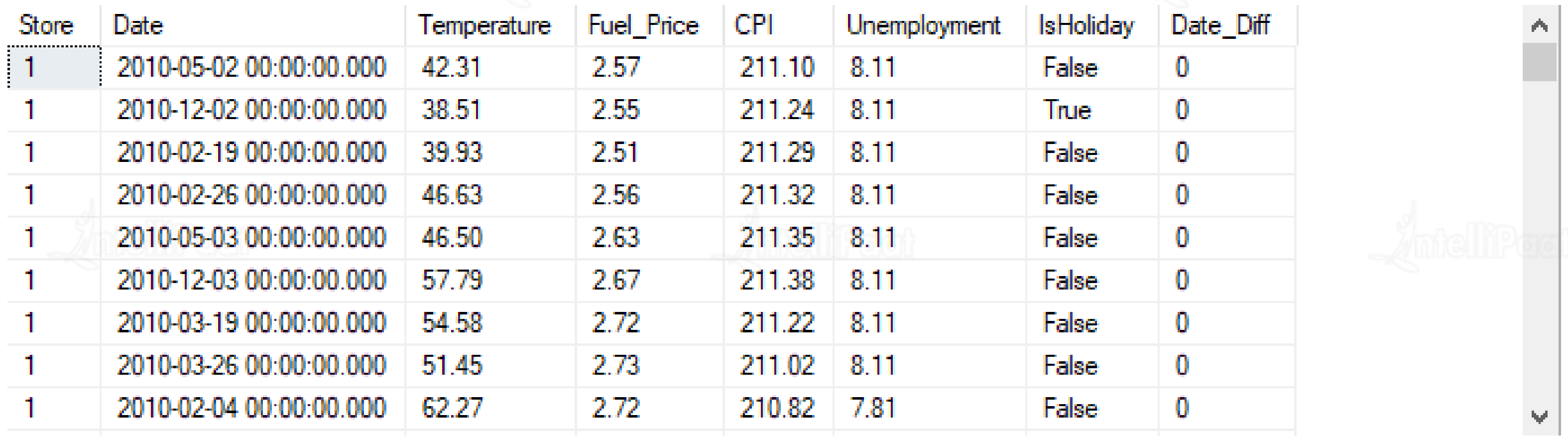
| Store | Date | Temperature | Fuel_Price | CPI | Unemployment | IsHoliday | new_date |
|---|---|---|---|---|---|---|---|
| 1 | 2010-05-02 00:00:00.000 | 42.31 | 2.57 | 211.10 | 8.11 | False | 2009-05-02 00:00:00.000 |
| 1 | 2010-12-02 00:00:00.000 | 38.51 | 2.55 | 211.24 | 8.11 | True | 2009-12-02 00:00:00.000 |
| 1 | 2010-02-19 00:00:00.000 | 39.93 | 2.51 | 211.29 | 8.11 | False | 2009-02-19 00:00:00.000 |
| 1 | 2010-02-26 00:00:00.000 | 46.63 | 2.56 | 211.32 | 8.11 | False | 2009-02-26 00:00:00.000 |
| 1 | 2010-05-03 00:00:00.000 | 46.50 | 2.63 | 211.35 | 8.11 | False | 2009-05-03 00:00:00.000 |
| 1 | 2010-12-03 00:00:00.000 | 57.79 | 2.67 | 211.38 | 8.11 | False | 2009-12-03 00:00:00.000 |
| 1 | 2010-03-19 00:00:00.000 | 54.58 | 2.72 | 211.22 | 8.11 | False | 2009-03-19 00:00:00.000 |
| 1 | 2010-03-26 00:00:00.000 | 51.45 | 2.73 | 211.02 | 8.11 | False | 2009-03-26 00:00:00.000 |
| 1 | 2010-02-04 00:00:00.000 | 62.27 | 2.72 | 210.82 | 7.81 | False | 2009-02-04 00:00:00.000 |

# DATE TIME FUNCTIONS

Create a new column “date_diff" that will contain a difference of dates column with the given date "11/04/2010"

**Datediff():** Returns a difference in a date part between two dates
**Syntax:** DATEDIFF ( datepart , startdate , enddate )

```
Alter table Features  Add Date_Diff as DATEDIFF(yy,'11/04/2010',[Date])
```



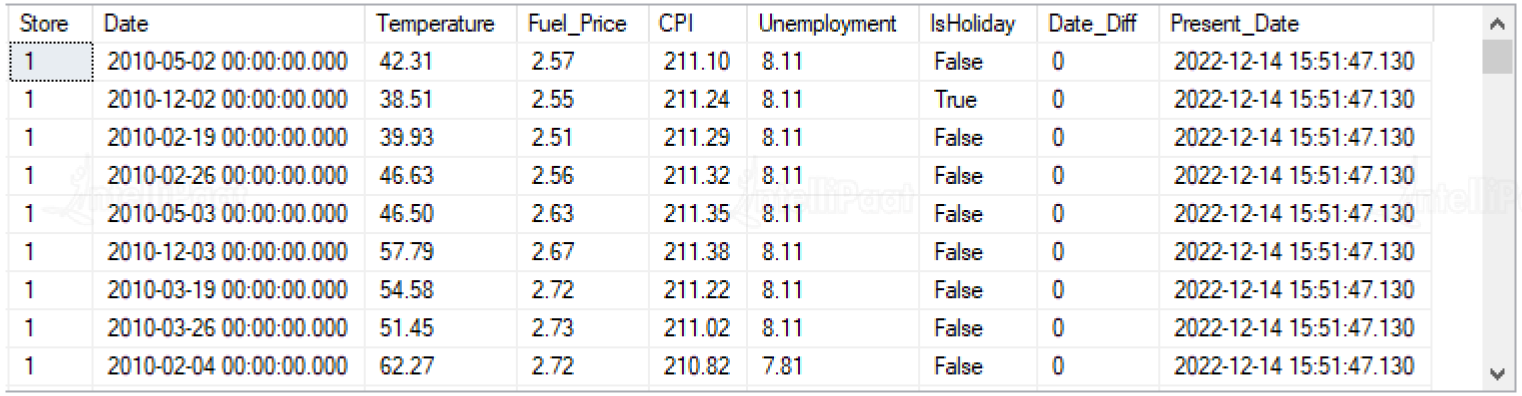
| Store | Date | Temperature | Fuel_Price | CPI | Unemployment | IsHoliday | Date_Diff |
|---|---|---|---|---|---|---|---|
| 1 | 2010-05-02 00:00:00.000 | 42.31 | 2.57 | 211.10 | 8.11 | False | 0 |
| 1 | 2010-12-02 00:00:00.000 | 38.51 | 2.55 | 211.24 | 8.11 | True | 0 |
| 1 | 2010-02-19 00:00:00.000 | 39.93 | 2.51 | 211.29 | 8.11 | False | 0 |
| 1 | 2010-02-26 00:00:00.000 | 46.63 | 2.56 | 211.32 | 8.11 | False | 0 |
| 1 | 2010-05-03 00:00:00.000 | 46.50 | 2.63 | 211.35 | 8.11 | False | 0 |
| 1 | 2010-12-03 00:00:00.000 | 57.79 | 2.67 | 211.38 | 8.11 | False | 0 |
| 1 | 2010-03-19 00:00:00.000 | 54.58 | 2.72 | 211.22 | 8.11 | False | 0 |
| 1 | 2010-03-26 00:00:00.000 | 51.45 | 2.73 | 211.02 | 8.11 | False | 0 |
| 1 | 2010-02-04 00:00:00.000 | 62.27 | 2.72 | 210.82 | 7.81 | False | 0 |

# DATE TIME FUNCTIONS

Create a new column that is containing the current date and time of the system.

**GETDATE():** Returns the current database system timestamp as a datetime value without the database time zone offset.
**Syntax:** *GETDATE()*

```
Alter Table Features Add Present_Date as getdate();
```



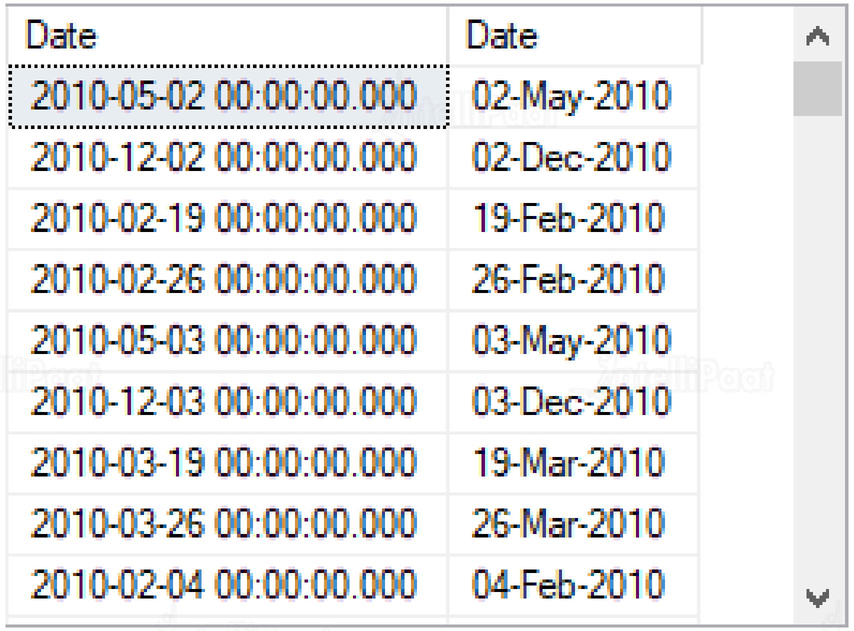
| Store | Date | Temperature | Fuel_Price | CPI | Unemployment | IsHoliday | Date_Diff | Present_Date |
|---|---|---|---|---|---|---|---|---|
| 1 | 2010-05-02 00:00:00.000 | 42.31 | 2.57 | 211.10 | 8.11 | False | 0 | 2022-12-14 15:51:47.130 |
| 1 | 2010-12-02 00:00:00.000 | 38.51 | 2.55 | 211.24 | 8.11 | True | 0 | 2022-12-14 15:51:47.130 |
| 1 | 2010-02-19 00:00:00.000 | 39.93 | 2.51 | 211.29 | 8.11 | False | 0 | 2022-12-14 15:51:47.130 |
| 1 | 2010-02-26 00:00:00.000 | 46.63 | 2.56 | 211.32 | 8.11 | False | 0 | 2022-12-14 15:51:47.130 |
| 1 | 2010-05-03 00:00:00.000 | 46.50 | 2.63 | 211.35 | 8.11 | False | 0 | 2022-12-14 15:51:47.130 |
| 1 | 2010-12-03 00:00:00.000 | 57.79 | 2.67 | 211.38 | 8.11 | False | 0 | 2022-12-14 15:51:47.130 |
| 1 | 2010-03-19 00:00:00.000 | 54.58 | 2.72 | 211.22 | 8.11 | False | 0 | 2022-12-14 15:51:47.130 |
| 1 | 2010-03-26 00:00:00.000 | 51.45 | 2.73 | 211.02 | 8.11 | False | 0 | 2022-12-14 15:51:47.130 |
| 1 | 2010-02-04 00:00:00.000 | 62.27 | 2.72 | 210.82 | 7.81 | False | 0 | 2022-12-14 15:51:47.130 |

## DATE TIME FUNCTIONS

Create a new column that is containing the current date and time of the system.

**Format():** Returns a value formatted with the specified format and optional culture. Use the FORMAT function for locale-aware formatting of date/time and number values as strings.
**Syntax:** ***FORMAT( value, format , culture )***

Select Date, Format([Date],'dd-MMM-yyyy') as Date from Features



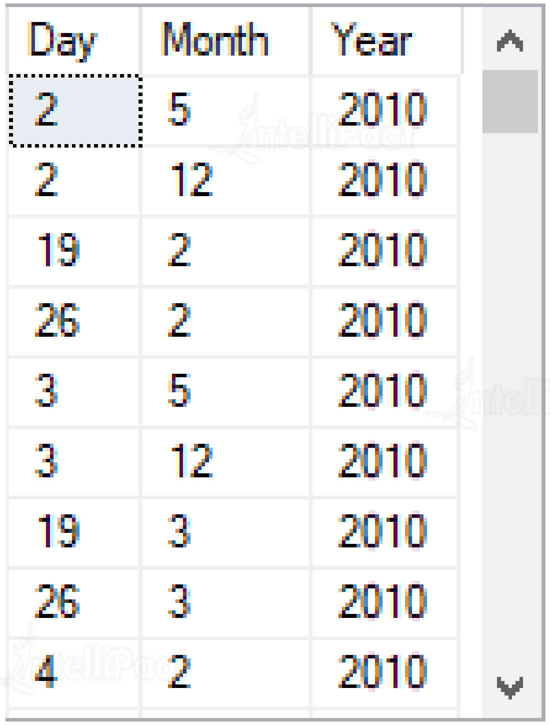
| Date | Date |
|---|---|
| 2010-05-02 00:00:00.000 | 02-May-2010 |
| 2010-12-02 00:00:00.000 | 02-Dec-2010 |
| 2010-02-19 00:00:00.000 | 19-Feb-2010 |
| 2010-02-26 00:00:00.000 | 26-Feb-2010 |
| 2010-05-03 00:00:00.000 | 03-May-2010 |
| 2010-12-03 00:00:00.000 | 03-Dec-2010 |
| 2010-03-19 00:00:00.000 | 19-Mar-2010 |
| 2010-03-26 00:00:00.000 | 26-Mar-2010 |
| 2010-02-04 00:00:00.000 | 04-Feb-2010 |

# DATE TIME FUNCTIONS

Display Days, Months, and Years from the Date column in the features table

**Day():** Returns the day of a specified date as an integer
**Syntax:** *DAY (date)*
**Month():** Returns the month of a specified date as an integer
**Syntax:** *MONTH (date)*
**Year():** Returns the year of a specified date as an integer
**Syntax:** *YEAR (date)*

```
SELECT Day([date])as Day, MONTH(date)as Month,
YEAR(date)as Year from features
```



| Day | Month | Year |
|---|---|---|
| 2 | 5 | 2010 |
| 2 | 12 | 2010 |
| 19 | 2 | 2010 |
| 26 | 2 | 2010 |
| 3 | 5 | 2010 |
| 3 | 12 | 2010 |
| 19 | 3 | 2010 |
| 26 | 3 | 2010 |
| 4 | 2 | 2010 |

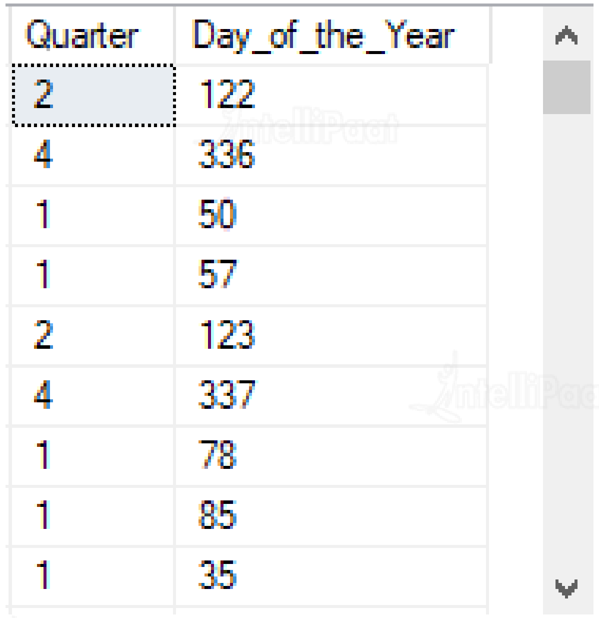
# DATE TIME FUNCTIONS

Display Quarter, Day of the year from the Date column in the features table using datepart function

**Datepart():** Returns the date part of a specified date as an integer number

**Syntax:** *DATEPART ( datepart , date )*

```
SELECT DATEPART(QQ,[date])as Quarter,
DATEPART(DY,[date])as Day_of_the_Year from features
```



| Quarter | Day_of_the_Year |
|---|---|
| 2 | 122 |
| 4 | 336 |
| 1 | 50 |
| 1 | 57 |
| 2 | 123 |
| 4 | 337 |
| 1 | 78 |
| 1 | 85 |
| 1 | 35 |

## AGGREGATE FUNCTIONS

An aggregate function performs a calculation on a set of values, and returns a single value

Different types of aggregate functions -

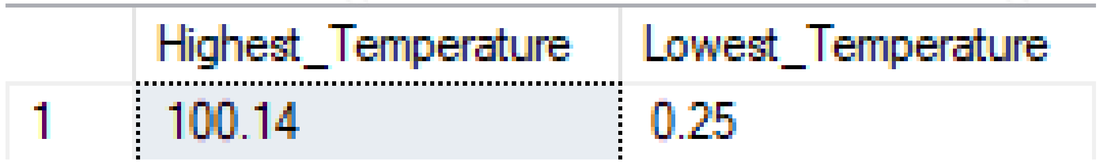
**MIN()** - Gives the minimum value from the record.
**MAX()** - Gives the maximum value from the record.
**SUM()** - Adds and gives the sum of the entire record.
**AVG()** - Gives the average value of an entire record.
**COUNT()** - Gives the count of the no of  rows.

# AGGREGATE FUNCTIONS

Check the highest and lowest temperature from the Featured table.

```
Select MAX(Temperature) as Highest_Temperature,
MIN(Temperature) as Lowest_Temperature from Features
```

OUTPUT



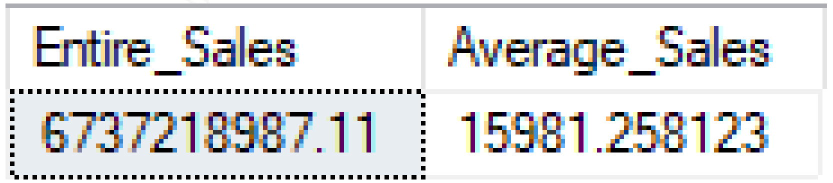
| | Highest_Temperature | Lowest_Temperature |
|---|---|---|
| 1 | 100.14 | 0.25 |

## AGGREGATE FUNCTIONS

Display the entire Weekly_Sales and the average Weekly_Sales.

```
Select SUM(Weekly_Sales) as Entire_Sales,
AVG(Weekly_Sales)as Average_Sales from Sales
```

OUTPUT



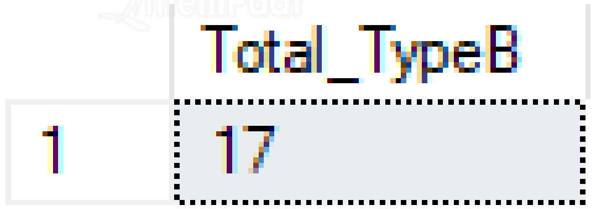
| Entire_Sales | Average_Sales |
|---|---|
| 6737218987.11 | 15981.258123 |

# AGGREGATE FUNCTIONS

How many types of 'B' records are present in the Stores table

```
Select COUNT(Type) as Total_TypeB from Stores Where Type ='B'
```

OUTPUT



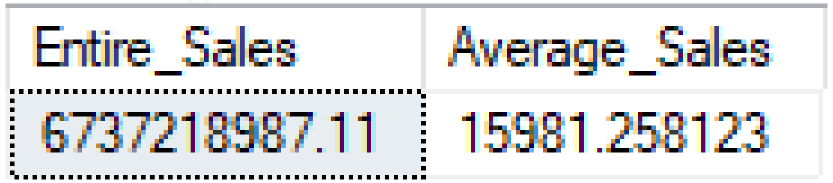
```
   Total_TypeB
1  17
```

# AGGREGATE FUNCTIONS

Display the entire Weekly_Sales and the average Weekly_Sales.

```
Select SUM(Weekly_Sales) as Entire_Sales,
AVG(Weekly_Sales)as Average_Sales from Sales
```

OUTPUT



| Entire_Sales | Average_Sales |
|---|---|
| 6737218987.11 | 15981.258123 |

## RANK FUNCTIONS

Ranking functions return a ranking value for each row in a partition. Depending on the function that is used, some rows might receive the same value as other rows.

**Ranking functions are nondeterministic.**

- **RANK**
- **DENSE RANK**
- **NTILE**
- **ROW NUMBER**

# RANK( Transact-SQL)

If two or more rows tie for a rank, each tied rows receives the same rank.

For example:

If the two top salespeople have the same SalesYTD value,
they are both ranked one.

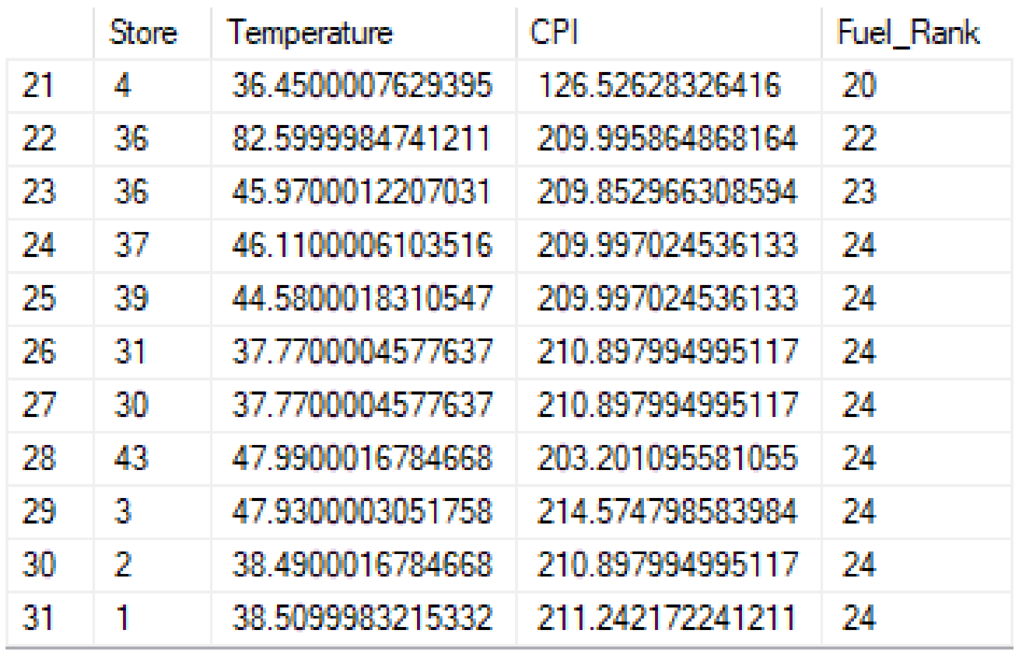
The salesperson with the next highest SalesYTD is ranked
number three, because there are two rows that are ranked higher.
Therefore, the RANK function does not always return consecutive integers.

# RANK( Transact-SQL)

Find out the rank of fuel price from the Features table and display it along with Store, Temperature and CPI.

**RANK()**
The rank function adds repeated rank to the repeated rows.
**Syntax:**
***RANK() OVER ( [PARTITION BY expression, ] ORDER BY expression (ASC | DESC) );***

```
 SELECT Store, Temperature,
CPI,  Rank() OVER (
ORDER by Fuel_Price)as Fuel_Rank
 from Features
```



| | Store | Temperature | CPI | Fuel_Rank |
|---|---|---|---|---|
| 21 | 4 | 36.4500007629395 | 126.52628326416 | 20 |
| 22 | 36 | 82.5999984741211 | 209.995864868164 | 22 |
| 23 | 36 | 45.9700012207031 | 209.852966308594 | 23 |
| 24 | 37 | 46.1100006103516 | 209.997024536133 | 24 |
| 25 | 39 | 44.5800018310547 | 209.997024536133 | 24 |
| 26 | 31 | 37.7700004577637 | 210.897994995117 | 24 |
| 27 | 30 | 37.7700004577637 | 210.897994995117 | 24 |
| 28 | 43 | 47.9900016784668 | 203.201095581055 | 24 |
| 29 | 3 | 47.9300003051758 | 214.574798583984 | 24 |
| 30 | 2 | 38.4900016784668 | 210.897994995117 | 24 |
| 31 | 1 | 38.5099983215332 | 211.242172241211 | 24 |

## ROW NUMBER( )

Returns the serial number of the row order by specified column.

Example for ROW NUMBER()

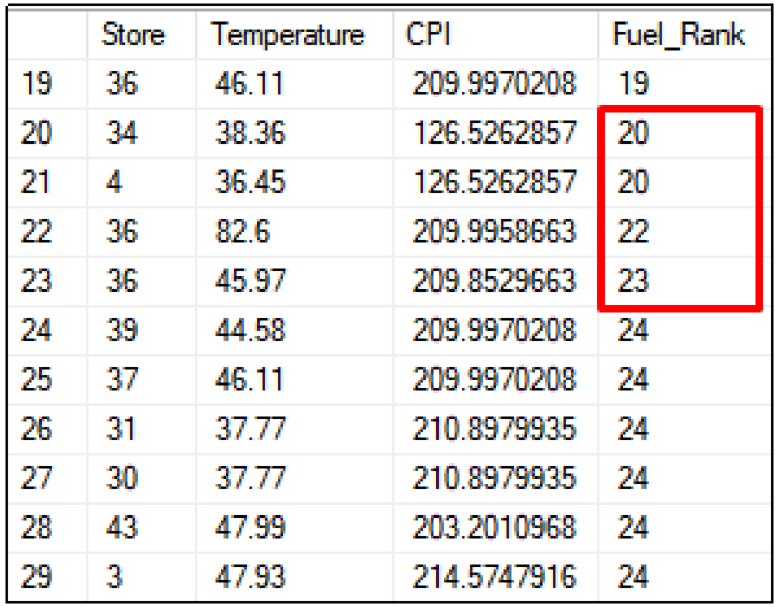
```
select Name,Subject,Marks,
ROW_NUMBER
over(order by Name)
RowNumber From
ExamResult
order by name,subject
```

# ROW_NUMBER()

Create a separate row number for the Sales table

**ROW_NUMBER()**
It is a basic rank function. It gives a result in sequential order.
**Syntax :**
***ROW_NUMBER ( ) OVER ( [ PARTITION BY value_expression , ... [ n ] ] order_by_clause )***

```
SELECT *, Row_Number() OVER (
ORDER by Dept)as RowNumber
 from Sales
```



|  | Store | Temperature | CPI | Fuel_Rank |
|---|---|---|---|---|
| 19 | 36 | 46.11 | 209.9970208 | 19 |
| 20 | 34 | 38.36 | 126.5262857 | 20 |
| 21 | 4 | 36.45 | 126.5262857 | 20 |
| 22 | 36 | 82.6 | 209.9958663 | 22 |
| 23 | 36 | 45.97 | 209.8529663 | 23 |
| 24 | 39 | 44.58 | 209.9970208 | 24 |
| 25 | 37 | 46.11 | 209.9970208 | 24 |
| 26 | 31 | 37.77 | 210.8979935 | 24 |
| 27 | 30 | 37.77 | 210.8979935 | 24 |
| 28 | 43 | 47.99 | 203.2010968 | 24 |
| 29 | 3 | 47.93 | 214.5747916 | 24 |

## DENSE Rank

Returns the rank of rows within the partition of a result set, without any gaps in the ranking.

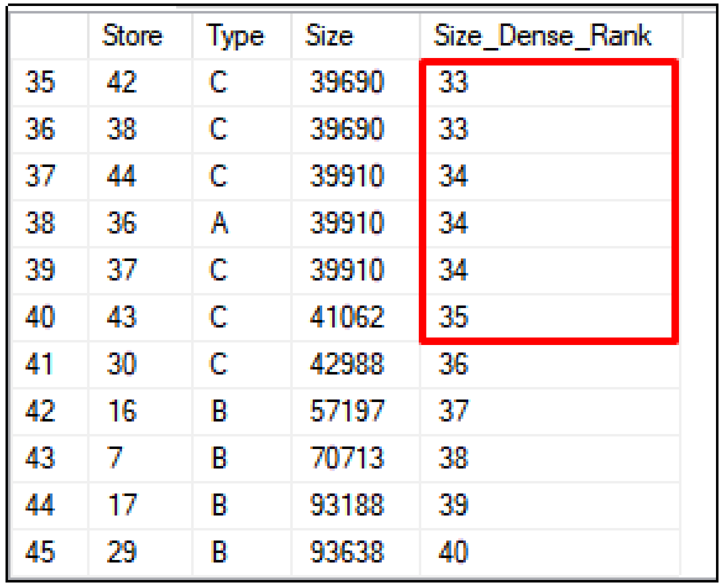
The rank of a row is one plus the number of distinct ranks that come before the row in question.

# DENSE_RANK()

Check the dense rank based on size from the Stores table.

**DENSE_RANK()**
The dense rank will never give any gaps.
**Syntax :**
*DENSE_RANK ( ) OVER ( [ <partition_by_clause> ] < order_by_clause > )*

```
SELECT *, Dense_Rank() OVER (
ORDER by Size)as
Size_Dense_Rank from Stores
```



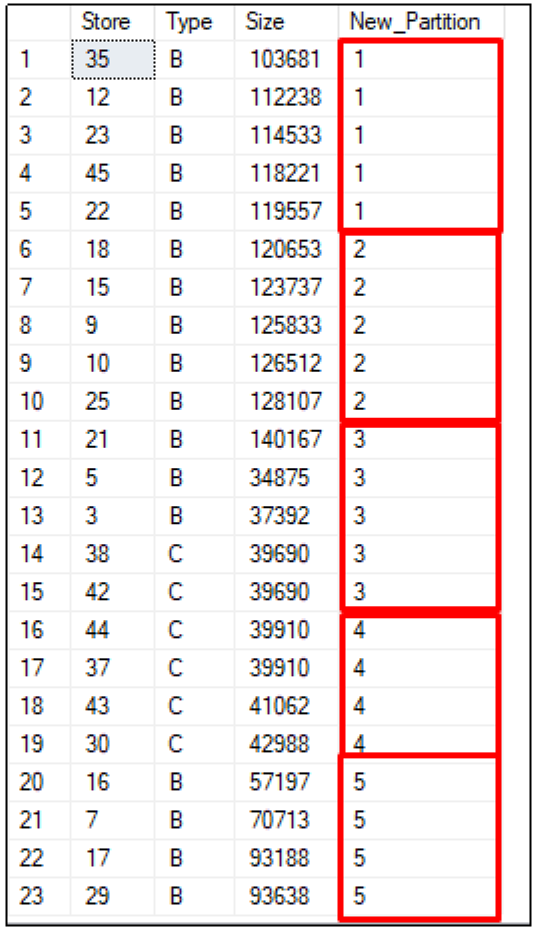
| | Store | Type | Size | Size_Dense_Rank |
|---|---|---|---|---|
| 35 | 42 | C | 39690 | 33 |
| 36 | 38 | C | 39690 | 33 |
| 37 | 44 | C | 39910 | 34 |
| 38 | 36 | A | 39910 | 34 |
| 39 | 37 | C | 39910 | 34 |
| 40 | 43 | C | 41062 | 35 |
| 41 | 30 | C | 42988 | 36 |
| 42 | 16 | B | 57197 | 37 |
| 43 | 7 | B | 70713 | 38 |
| 44 | 17 | B | 93188 | 39 |
| 45 | 29 | B | 93638 | 40 |

# NTILE()

Show the partition of 5 in your Stores table.

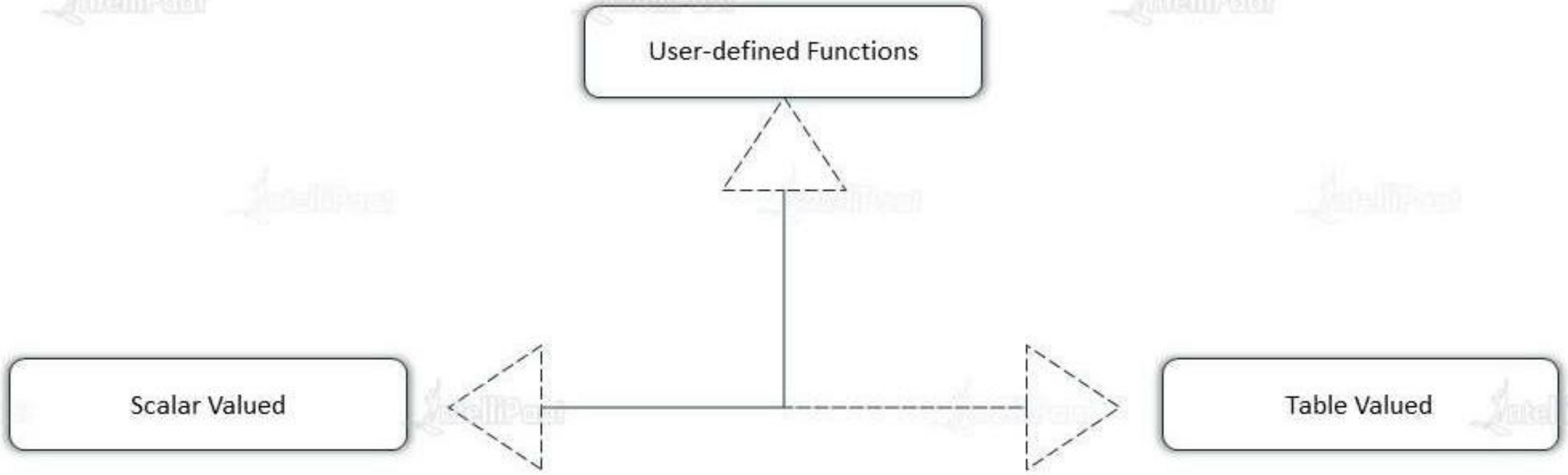
**NTILE()**
The NTILE is used to divide or try to partition the rows equally.
**Syntax :**
*NTILE (integer_expression) OVER ( [ <partition_by_clause> ] < order_by_clause > )*

```
SELECT *, NTILE(5) OVER (
ORDER by Size)as New_Partition
 from Stores
```



| | Store | Type | Size | New_Partition |
|---|---|---|---|---|
| 1 | 35 | B | 103681 | 1 |
| 2 | 12 | B | 112238 | 1 |
| 3 | 23 | B | 114533 | 1 |
| 4 | 45 | B | 118221 | 1 |
| 5 | 22 | B | 119557 | 1 |
| 6 | 18 | B | 120653 | 2 |
| 7 | 15 | B | 123737 | 2 |
| 8 | 9 | B | 125833 | 2 |
| 9 | 10 | B | 126512 | 2 |
| 10 | 25 | B | 128107 | 2 |
| 11 | 21 | B | 140167 | 3 |
| 12 | 5 | B | 34875 | 3 |
| 13 | 3 | B | 37392 | 3 |
| 14 | 38 | C | 39690 | 3 |
| 15 | 42 | C | 39690 | 3 |
| 16 | 44 | C | 39910 | 4 |
| 17 | 37 | C | 39910 | 4 |
| 18 | 43 | C | 41062 | 4 |
| 19 | 30 | C | 42988 | 4 |
| 20 | 16 | B | 57197 | 5 |
| 21 | 7 | B | 70713 | 5 |
| 22 | 17 | B | 93188 | 5 |
| 23 | 29 | B | 93638 | 5 |

# User-defined functions



## Scaler Valued Functions

A Scalar valued function will take one or many inputs and returns a single value whenever the function is invoked.

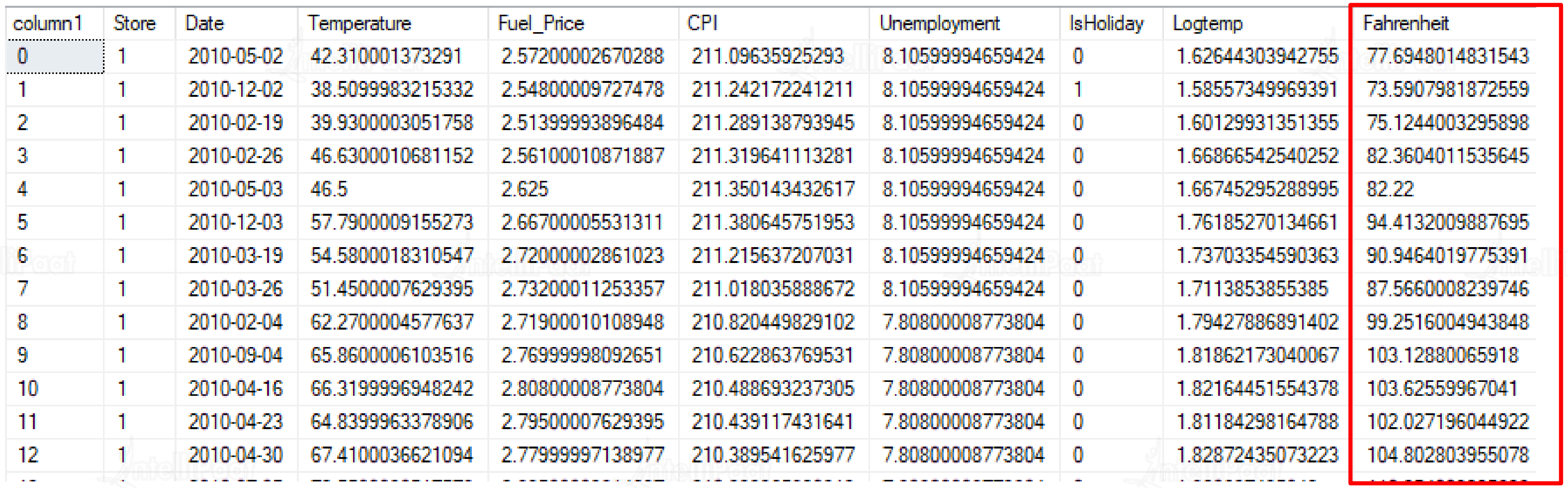
Create a user-defined function to convert the temperature to Fahrenheit.

```
//Scalar Valued Function Creation
 create function fahrenheit (@val float)
 returns float as begin return (@val * 1.08)+32 end

//To call the function
 select *, dbo.fahrenheit(Temperature) as Fahrenheit from Features
```

## SCALER VALUED FUNCTIONS

OUTPUT



| column1 | Store | Date | Temperature | Fuel_Price | CPI | Unemployment | IsHoliday | Logtemp | Fahrenheit |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2010-05-02 | 42.310001373291 | 2.57200002670288 | 211.09635925293 | 8.10599994659424 | 0 | 1.62644303942755 | 77.6948014831543 |
| 1 | 1 | 2010-12-02 | 38.5099983215332 | 2.54800009727478 | 211.242172241211 | 8.10599994659424 | 1 | 1.58557349969391 | 73.5907981872559 |
| 2 | 1 | 2010-02-19 | 39.9300003051758 | 2.51399993896484 | 211.289138793945 | 8.10599994659424 | 0 | 1.60129931351355 | 75.1244003295898 |
| 3 | 1 | 2010-02-26 | 46.6300010681152 | 2.56100010871887 | 211.319641113281 | 8.10599994659424 | 0 | 1.66866542540252 | 82.3604011535645 |
| 4 | 1 | 2010-05-03 | 46.5 | 2.625 | 211.350143432617 | 8.10599994659424 | 0 | 1.66745295288995 | 82.22 |
| 5 | 1 | 2010-12-03 | 57.7900009155273 | 2.66700005531311 | 211.380645751953 | 8.10599994659424 | 0 | 1.76185270134661 | 94.4132009887695 |
| 6 | 1 | 2010-03-19 | 54.5800018310547 | 2.72000002861023 | 211.215637207031 | 8.10599994659424 | 0 | 1.73703354590363 | 90.9464019775391 |
| 7 | 1 | 2010-03-26 | 51.4500007629395 | 2.73200011253357 | 211.018035888672 | 8.10599994659424 | 0 | 1.7113853855385 | 87.5660008239746 |
| 8 | 1 | 2010-02-04 | 62.2700004577637 | 2.71900010108948 | 210.820449829102 | 7.80800008773804 | 0 | 1.79427886891402 | 99.2516004943848 |
| 9 | 1 | 2010-09-04 | 65.8600006103516 | 2.76999998092651 | 210.622863769531 | 7.80800008773804 | 0 | 1.81862173040067 | 103.12880065918 |
| 10 | 1 | 2010-04-16 | 66.3199996948242 | 2.80800008773804 | 210.488693237305 | 7.80800008773804 | 0 | 1.82164451554378 | 103.62559967041 |
| 11 | 1 | 2010-04-23 | 64.8399963378906 | 2.79500007629395 | 210.439117431641 | 7.80800008773804 | 0 | 1.81184298164788 | 102.027196044922 |
| 12 | 1 | 2010-04-30 | 67.4100036621094 | 2.77999997138977 | 210.389541625977 | 7.80800008773804 | 0 | 1.82872435073223 | 104.802803955078 |

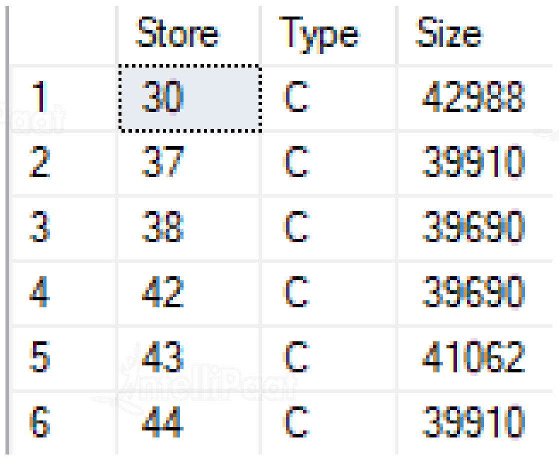
# TABLE VALUED FUNCTIONS

A Table valued function can be created with or without parameters and returns a table data whenever the function is invoked.

Create a user-defined function for the Stores table to fetch a particular store type based upon the user’s preference.

```
//Table Valued Function Creation with Parameter
 Create function Type_A (@type varchar(5))
  returns table as return
  select * from Stores where Type=@type

//To call the function
  select * from Type_A('C')
```



|  | Store | Type | Size |
|---|---|---|---|
| 1 | 30 | C | 42988 |
| 2 | 37 | C | 39910 |
| 3 | 38 | C | 39690 |
| 4 | 42 | C | 39690 |
| 5 | 43 | C | 41062 |
| 6 | 44 | C | 39910 |