Chapter 9

How to use functions

Objectives

Applied

1. Code queries that format numeric or date/time data.
2. Code queries that require any of the scalar functions presented in this chapter.
3. Code queries that require any of the ranking or analytic functions presented in this chapter.

Knowledge

1. Describe how the use of functions can solve the problems associated with (1) sorting string data that contains numeric values, and (2) doing date or time searches.
2. Describe the use of the ranking functions for ranking the rows returned by a result set.
3. Describe the use of the analytic functions for performing calculations on ordered sets of data.

Some of the string functions

```
CONCAT(str1[,str2]...)
CONCAT_WS(sep,str1[,str2]...)
LTRIM(str)
RTRIM(str)
TRIM([[BOTH|LEADING|TRAILING] [remove] FROM] str)
LENGTH(str)
LOCATE(find,search[,start])
LEFT(str,length)
RIGHT(str,length)
SUBSTRING_INDEX(str,delimiter, count)
SUBSTRING(str,start[,length])
```

Some of the string functions (continued)

```
REPLACE(search,find,replace)
INSERT(str,start,length,insert)
REVERSE(str)
LOWER(str)
UPPER(str)
LPAD(str,length,pad)
RPAD(str,length,pad)
SPACE(count)
REPEAT(str,count)
```

String function examples

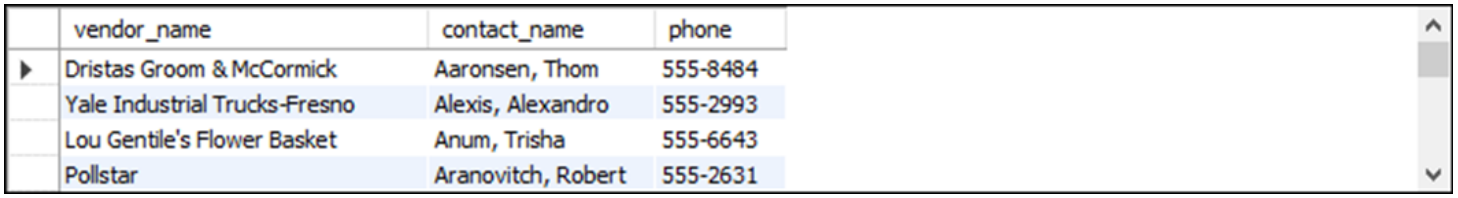
Function	Result
`CONCAT('Last', 'First')`	`'LastFirst'`
`CONCAT_WS(', ', 'Last', 'First')`	`'Last, First'`
`LTRIM('  MySQL  ')`	`'MySQL  '`
`RTRIM('  MySQL  ')`	`'  MySQL'`
`TRIM('  MySQL  ')`	`'MySQL'`
`TRIM(BOTH '*' FROM '****MySQL****')`	`'MySQL'`
`LOWER('MySQL')`	`'mysql'`
`UPPER('ca')`	`'CA'`
`LEFT('MySQL', 3)`	`'MyS'`
`RIGHT('MySQL', 3)`	`'SQL'`

String function examples (continued)

Function	Result
`SUBSTRING('(559) 555-1212', 7, 8)`	`'555-1212'`
`SUBSTRING_INDEX('http://www.murach.com', '.', -2)`	`'murach.com'`
`LENGTH('MySQL')`	`5`
`LENGTH('  MySQL  ')`	`9`
`LOCATE('SQL', '  MySQL')`	`5`
`LOCATE('-', '(559) 555-1212')`	`10`
`REPLACE(RIGHT('(559) 555-1212', 13),') ', '-')`	`'559-555-1212'`
`INSERT("MySQL", 1, 0, "Murach's ")`	`"Murach's MySQL"`
`INSERT('MySQL', 1, 0, 'Murach''s ')`	`"Murach's MySQL"`

A SELECT statement that uses three functions

```
SELECT vendor_name,
       CONCAT_WS(', ', vendor_contact_last_name,
                 vendor_contact_first_name) AS contact_name,
       RIGHT(vendor_phone, 8) AS phone
FROM vendors
WHERE LEFT(vendor_phone, 4) = '(559'
ORDER BY contact_name
```

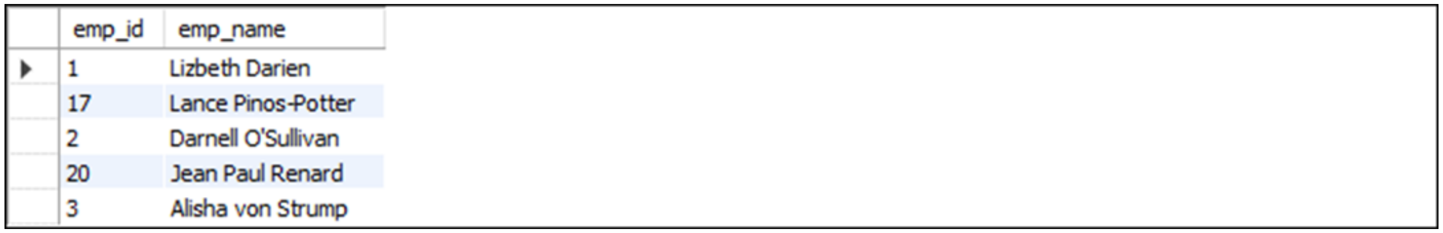


vendor_name	contact_name	phone
Dristas Groom & McCormick	Aaronsen, Thom	555-8484
Yale Industrial Trucks-Fresno	Alexis, Alexandro	555-2993
Lou Gentile's Flower Basket	Anum, Trisha	555-6643
Pollstar	Aranovitch, Robert	555-2631

How to sort by a string column that contains numbers (part 1)

Sorted by the emp_id column

```
SELECT *
FROM string_sample
ORDER BY emp_id
```

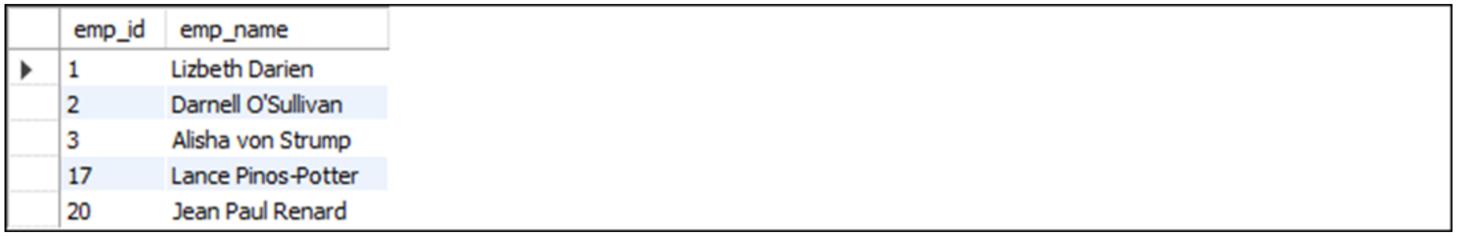


emp_id	emp_name
1	Lizbeth Darien
17	Lance Pinos-Potter
2	Darnell O'Sullivan
20	Jean Paul Renard
3	Alisha von Strump

How to sort by a string column that contains numbers (part 2)

Sorted by the emp_id column explicitly cast as an integer

```
SELECT *
FROM string_sample
ORDER BY CAST(emp_id AS SIGNED)
```

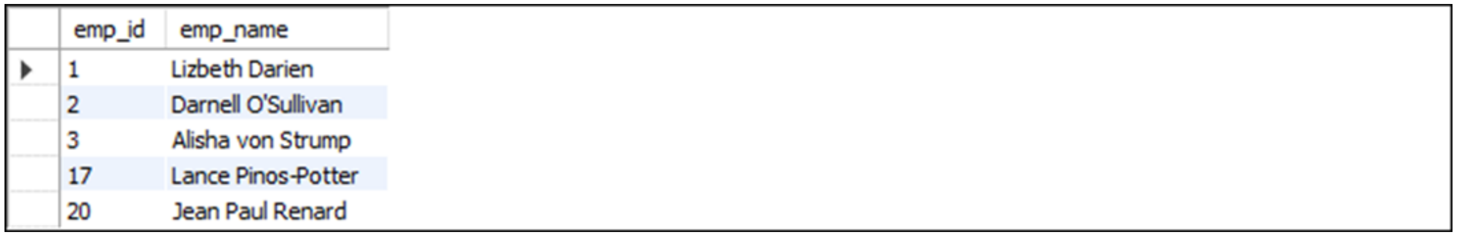


emp_id	emp_name
1	Lizbeth Darien
2	Darnell O'Sullivan
3	Alisha von Strump
17	Lance Pinos-Potter
20	Jean Paul Renard

How to sort by a string column that contains numbers (part 3)

Sorted by the emp_id column implicitly cast as an integer

```
SELECT *
FROM string_sample
ORDER BY emp_id + 0
```

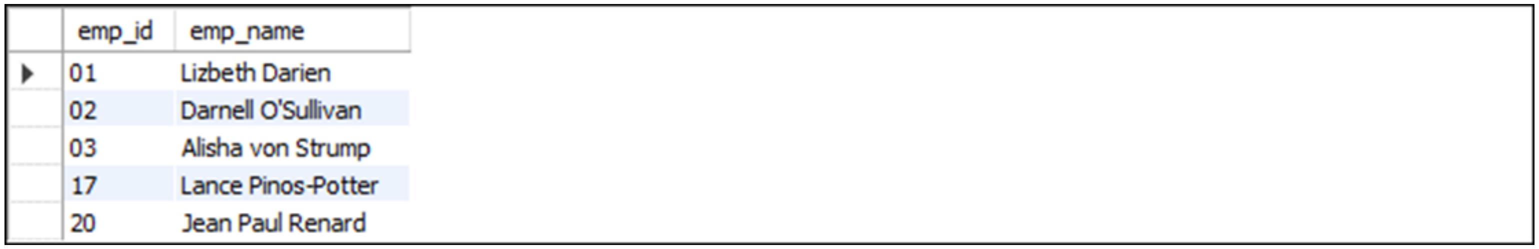


emp_id	emp_name
1	Lizbeth Darien
2	Darnell O'Sullivan
3	Alisha von Strump
17	Lance Pinos-Potter
20	Jean Paul Renard

How to sort by a string column that contains numbers (part 4)

Sorted by the emp_id column after it has been padded with leading zeros

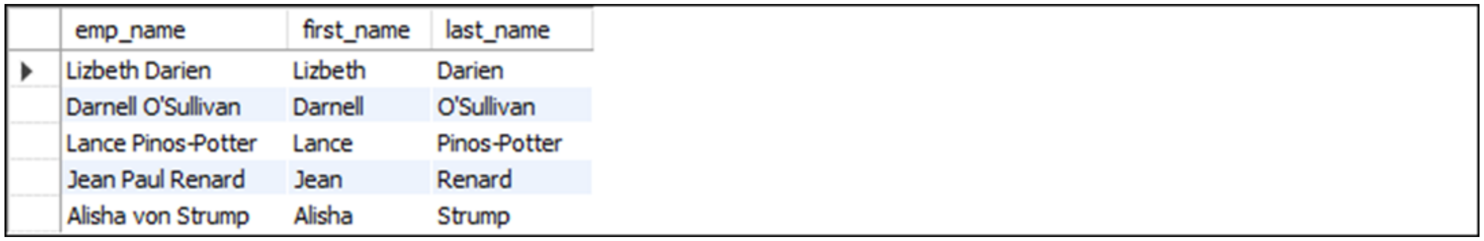
```
SELECT LPAD(emp_id, 2, '0') AS emp_id, emp_name
FROM string_sample
ORDER BY emp_id
```



emp_id	emp_name
01	Lizbeth Darien
02	Darnell O'Sullivan
03	Alisha von Strump
17	Lance Pinos-Potter
20	Jean Paul Renard

How to use the SUBSTRING_INDEX function to parse a string

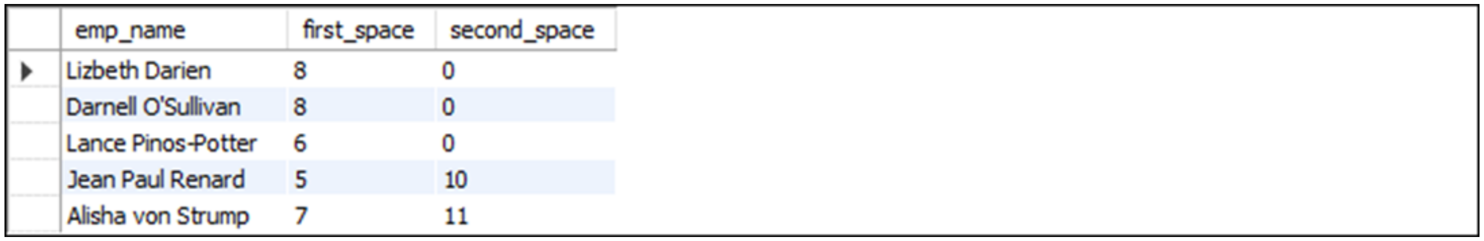
```
SELECT emp_name,
    SUBSTRING_INDEX(emp_name, ' ', 1) AS first_name,
    SUBSTRING_INDEX(emp_name, ' ', -1) AS last_name
FROM string_sample
```



emp_name	first_name	last_name
Lizbeth Darien	Lizbeth	Darien
Darnell O'Sullivan	Darnell	O'Sullivan
Lance Pinos-Potter	Lance	Pinos-Potter
Jean Paul Renard	Jean	Renard
Alisha von Strump	Alisha	Strump

How to use the LOCATE function to find a character in a string

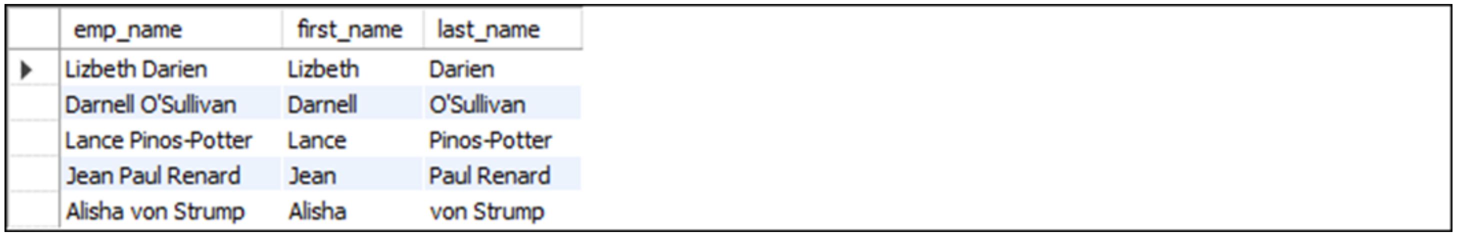
```
SELECT emp_name,
    LOCATE(' ', emp_name) AS first_space,
    LOCATE(' ', emp_name, LOCATE(' ', emp_name) + 1)
    AS second_space
FROM string_sample
```



emp_name	first_space	second_space
Lizbeth Darien	8	0
Darnell O'Sullivan	8	0
Lance Pinos-Potter	6	0
Jean Paul Renard	5	10
Alisha von Strump	7	11

How to use the SUBSTRING function to parse a string

```
SELECT emp_name,
    SUBSTRING(emp_name, 1, LOCATE(' ', emp_name) - 1)
    AS first_name,
    SUBSTRING(emp_name, LOCATE(' ', emp_name) + 1)
    AS last_name
FROM string_sample
```



emp_name	first_name	last_name
Lizbeth Darien	Lizbeth	Darien
Darnell O'Sullivan	Darnell	O'Sullivan
Lance Pinos-Potter	Lance	Pinos-Potter
Jean Paul Renard	Jean	Paul Renard
Alisha von Strump	Alisha	von Strump

Some of the numeric functions

```
ROUND(number[,length])
TRUNCATE(number,length)
CEILING(number)
FLOOR(number)
ABS(number)
SIGN(number)
SQRT(number)
POWER(number,power)
RAND([integer])
```

Examples that use the numeric functions

Function	Result
`ROUND(12.49,0)`	`12`
`ROUND(12.50,0)`	`13`
`ROUND(12.49,1)`	`12.5`
`TRUNCATE(12.51,0)`	`12`
`TRUNCATE(12.49,1)`	`12.4`

Examples that use the numeric functions (continued)

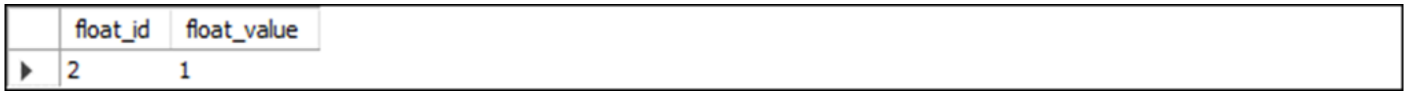
Function	Result
`CEILING(12.5)`	`13`
`CEILING(-12.5)`	`-12`
`FLOOR(-12.5)`	`-13`
`FLOOR(12.5)`	`12`
`ABS(-1.25)`	`1.25`
`ABS(1.25)`	`1.25`
`SIGN(-1.25)`	`-1`
`SIGN(1.25)`	`1`
`SQRT(125.43)`	`11.199553562530964`
`POWER(9,2)`	`81`
`RAND()`	`0.2444132019248`

The Float_Sample table

float_id	float_value
1	0.999999999999999
2	1
3	1.000000000000001
4	1234.56789012345
5	999.04440209348
6	24.04849

A search for an exact value that doesn't include two approximate values

```
SELECT *
FROM float_sample
WHERE float_value = 1
```

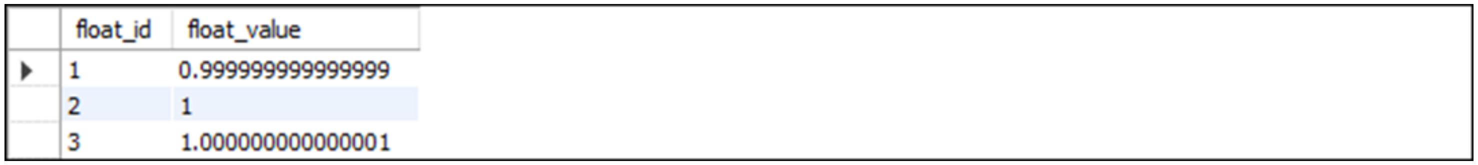


float_id	float_value
2	1

How to search for approximate values

Search for a range of values

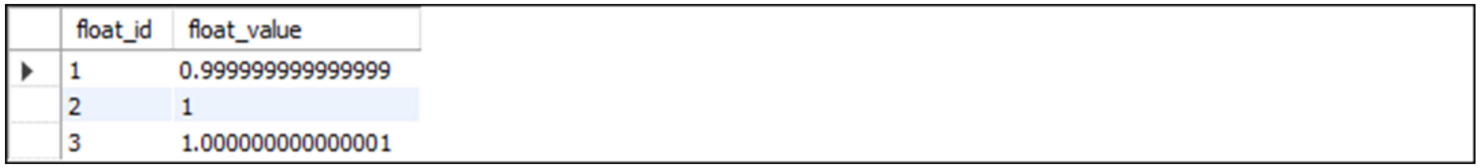
```
SELECT *
FROM float_sample
WHERE float_value BETWEEN 0.99 AND 1.01
```



float_id	float_value
1	0.999999999999999
2	1
3	1.000000000000001

Search for rounded values

```
SELECT *
FROM float_sample
WHERE ROUND(float_value, 2) = 1.00
```



float_id	float_value
1	0.999999999999999
2	1
3	1.000000000000001

Functions that get the current date and time

```
NOW()
SYSDATE()
CURRENT_TIMESTAMP()

CURDATE()
CURRENT_DATE()

CURTIME()
CURRENT_TIME()

UTC_DATE()

UTC_TIME()
```

Examples that get the current date and time

Function	Result
`NOW()`	`2018-12-06 14:12:04`
`SYSDATE()`	`2018-12-06 14:12:04`
`CURDATE()`	`2018-12-06`
`CURTIME()`	`14:12:04`
`UTC_DATE()`	`2018-12-06`
`UTC_TIME()`	`21:12:04`
`CURRENT_TIMESTAMP()`	`2018-12-06 14:12:04`
`CURRENT_DATE()`	`2018-12-06`
`CURRENT_TIME()`	`14:12:04`

Some of the date/time parsing functions

```
DAYOFMONTH(date)
MONTH(date)
YEAR(date)
HOUR(time)
MINUTE(time)
SECOND(time)
DAYOFWEEK(date)

QUARTER(date)
DAYOFYEAR(date)
WEEK(date[,first])

LAST_DAY(date)

DAYNAME(date)
MONTHNAME(date)
```

Examples that use the date/time parsing functions

Function	Result
`DAYOFMONTH('2018-12-03')`	`3`
`MONTH('2018-12-03')`	`12`
`YEAR('2018-12-03')`	`2018`
`HOUR('11:35:00')`	`11`
`MINUTE('11:35:00')`	`35`
`SECOND('11:35:00')`	`0`
`DAYOFWEEK('2018-12-03')`	`2`
`QUARTER('2018-12-03')`	`4`
`DAYOFYEAR('2018-12-03')`	`337`
`WEEK('2018-12-03')`	`48`
`LAST_DAY('2018-12-03')`	`31`
`DAYNAME('2018-12-03')`	`Monday`
`MONTHNAME('2018-12-03')`	`December`

The EXTRACT function

```
EXTRACT(unit FROM date)
```

Date/time units

Unit	Description
SECOND	Seconds
MINUTE	Minutes
HOUR	Hours
DAY	Day
MONTH	Month
YEAR	Year
MINUTE_SECOND	Minutes and seconds
HOUR_MINUTE	Hour and minutes
DAY_HOUR	Day and hours
YEAR_MONTH	Year and month
HOUR_SECOND	Hours, minutes, and seconds
DAY_MINUTE	Day, hours, and minutes
DAY_SECOND	Day, hours, minutes, and seconds

Examples that use the EXTRACT function

Function	Result
`EXTRACT(SECOND FROM '2018-12-03 11:35:00')`	`0`
`EXTRACT(MINUTE FROM '2018-12-03 11:35:00')`	`35`
`EXTRACT(HOUR FROM '2018-12-03 11:35:00')`	`11`
`EXTRACT(DAY FROM '2018-12-03 11:35:00')`	`3`
`EXTRACT(MONTH FROM '2018-12-03 11:35:00')`	`12`
`EXTRACT(YEAR FROM '2018-12-03 11:35:00')`	`2018`
`EXTRACT(MINUTE_SECOND FROM '2018-12-03 11:35:00')`	`3500`
`EXTRACT(HOUR_MINUTE FROM '2018-12-03 11:35:00')`	`1135`
`EXTRACT(DAY_HOUR FROM '2018-12-03 11:35:00')`	`311`
`EXTRACT(YEAR_MONTH FROM '2018-12-03 11:35:00')`	`201812`
`EXTRACT(HOUR_SECOND FROM '2018-12-03 11:35:00')`	`113500`
`EXTRACT(DAY_MINUTE FROM '2018-12-03 11:35:00')`	`31135`
`EXTRACT(DAY_SECOND FROM '2018-12-03 11:35:00’)`	`3113500`

Two functions for formatting dates and times

```
DATE_FORMAT(date,format)
TIME_FORMAT(time,format)
```

Common codes for date/time format strings

Code	Description
`%m`	Month, numeric (01…12)
`%c`	Month, numeric (1…12)
`%M`	Month name (January…December)
`%b`	Abbreviated month name (Jan…Dec)
`%d`	Day of the month, numeric (00…31)
`%e`	Day of the month, numeric (0…31)
`%D`	Day of the month with suffix (1st, 2nd, 3rd, etc.)
`%y`	Year, numeric, 2 digits
`%Y`	Year, numeric, 4 digits

Common codes for date/time format strings (continued)

Code	Description
`%W`	Weekday name (Sunday…Saturday)
`%a`	Abbreviated weekday name (Sun…Sat)
`%H`	Hour (00…23)
`%k`	Hour (0…23)
`%h`	Hour (01…12)
`%l`	Hour (1…12)
`%i`	Minutes (00…59)
`%r`	Time, 12-hour (hh:mm:ss AM or PM)
`%T`	Time, 24-hour (hh:mm:ss)
`%S`	Seconds (00…59)
`%p`	AM or PM

Examples that use the date/time formatting functions

Function	Result
`DATE_FORMAT('2018-12-03', '%m/%d/%y')`	`12/03/18`
`DATE_FORMAT('2018-129-03', '%W, %M %D, %Y')`	`Monday, December 3rd, 2018`
`DATE_FORMAT('2018-12-03', '%e-%b-%y')`	`3-Dec-18`
`DATE_FORMAT('2018-12-03 16:45', '%r')`	`04:45:00 PM`
`TIME_FORMAT('16:45', '%r')`	`04:45:00 PM`
`TIME_FORMAT('16:45', '%l:%i %p')`	`4:45 PM`

Some of the functions for calculating dates and times

```
DATE_ADD(date,INTERVAL expression unit)
DATE_SUB(date,INTERVAL expression unit)
DATEDIFF(date1, date2)
TO_DAYS(date)
TIME_TO_SEC(time)
```

Examples of the functions for calculating dates and times

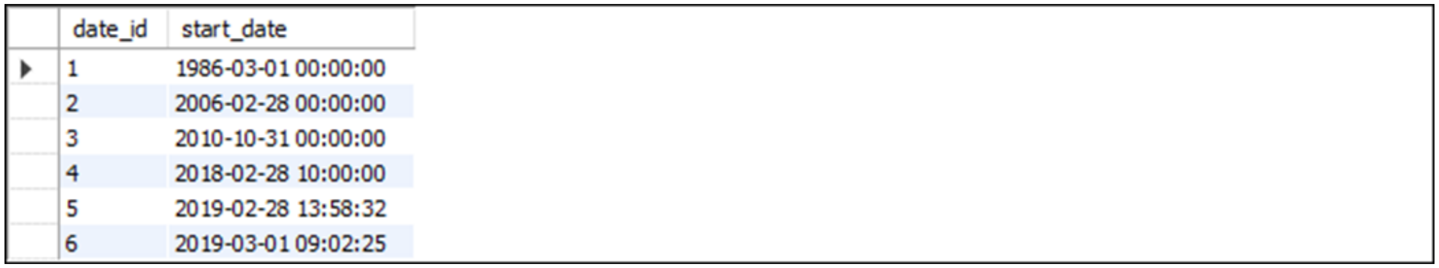
Function	Result
`DATE_ADD('2018-12-31', INTERVAL 1 DAY)`	`2019-01-01`
`DATE_ADD('2018-12-31', INTERVAL 3 MONTH)`	`2019-03-31`
`DATE_ADD('2018-12-31 23:59:59', INTERVAL 1 SECOND)`	`2019-01-01 00:00:00`
`DATE_ADD('2019-01-01', INTERVAL -1 DAY)`	`2018-12-31`
`DATE_SUB('2019-01-01', INTERVAL 1 DAY)`	`2018-12-31`
`DATE_ADD('2016-02-29', INTERVAL 1 YEAR)`	`2017-02-28`
`DATE_ADD('2018-02-29', INTERVAL 1 YEAR)`	`NULL`
`DATE_ADD('2018-12-31 12:00', INTERVAL '2 12' DAY_HOUR)`	`2019-01-03 00:00:00`

Examples of the functions for calculating dates and times (continued)

```
Function                                    Result
DATEDIFF('2018-12-30', '2018-12-03')        27
DATEDIFF('2018-12-30 23:59:59',
    '2018-12-03')                           27
DATEDIFF('2018-12-03', '2018-12-30')        -27

TO_DAYS('2018-12-30')
    - TO_DAYS('2018-12-03')                 27
TIME_TO_SEC('10:00')
    - TIME_TO_SEC('09:59')                  60
```

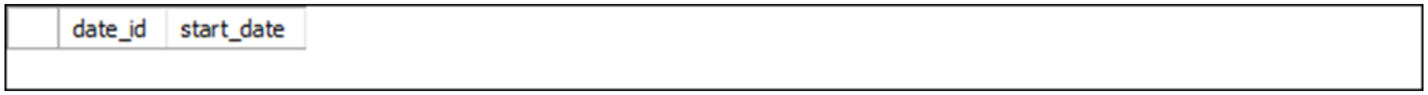
The contents of the Date_Sample table with times



date_id	start_date
1	1986-03-01 00:00:00
2	2006-02-28 00:00:00
3	2010-10-31 00:00:00
4	2018-02-28 10:00:00
5	2019-02-28 13:58:32
6	2019-03-01 09:02:25

A SELECT statement that fails to return a row

```
SELECT *
FROM date_sample
WHERE start_date = '2018-02-28'
```

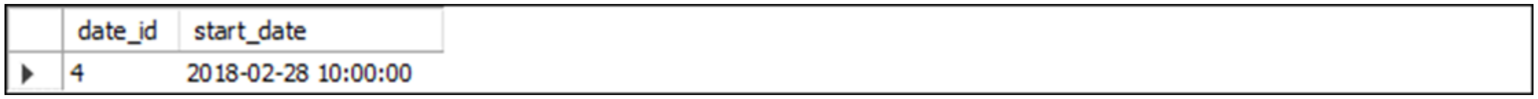


date_id	start_date

Three techniques for ignoring time values

Search for a range of dates

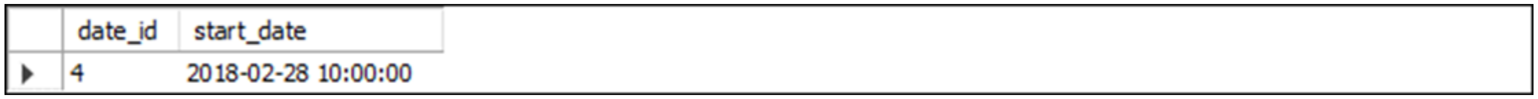
```
SELECT *
FROM date_sample
WHERE start_date >= '2018-02-28'
  AND start_date < '2018-03-01'
```



date_id	start_date
4	2018-02-28 10:00:00

Search for month, day, and year integers

```
SELECT *
FROM date_sample
WHERE MONTH(start_date) = 2 AND
      DAYOFMONTH(start_date) = 28 AND
      YEAR(start_date) = 2018
```

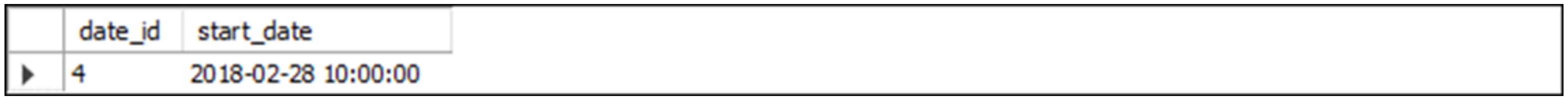


date_id	start_date
4	2018-02-28 10:00:00

Three techniques for ignoring time values (continued)

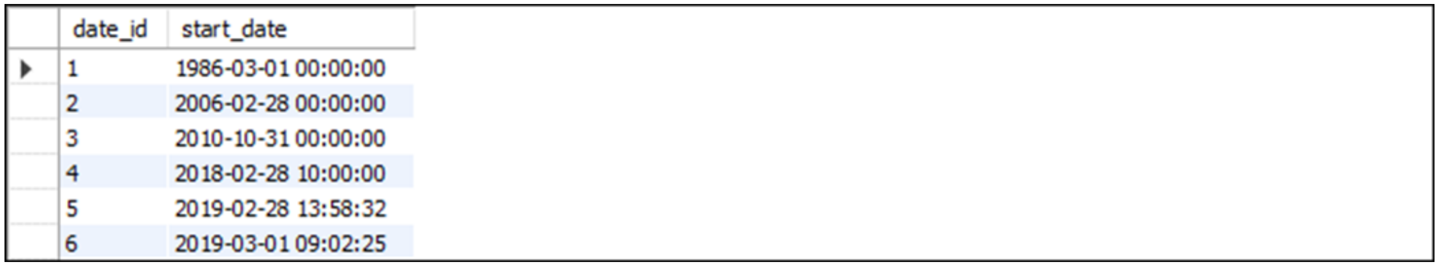
Search for a formatted date

```
SELECT *
FROM date_sample
WHERE DATE_FORMAT(start_date, '%m-%d-%Y') = '02-28-2018'
```



date_id	start_date
4	2018-02-28 10:00:00

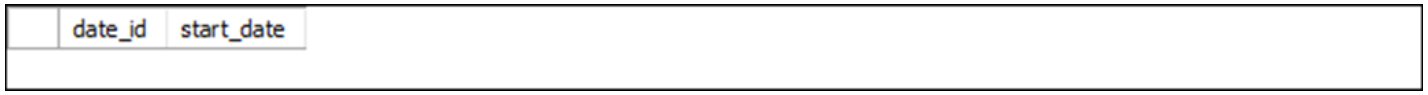
The contents of the Date_Sample table with dates



date_id	start_date
1	1986-03-01 00:00:00
2	2006-02-28 00:00:00
3	2010-10-31 00:00:00
4	2018-02-28 10:00:00
5	2019-02-28 13:58:32
6	2019-03-01 09:02:25

A SELECT statement that fails to return a row

```
SELECT * FROM date_sample
WHERE start_date = '10:00:00'
```

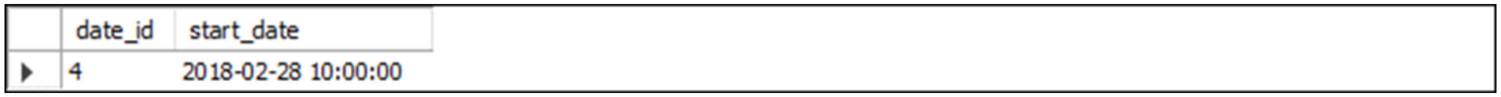


date_id	start_date

Examples that ignore date values

Search for a time that has been formatted

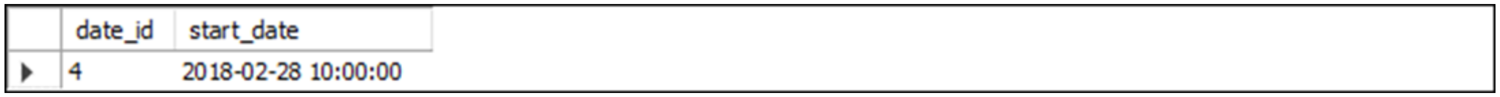
```
SELECT * FROM date_sample
WHERE DATE_FORMAT(start_date, '%T') = '10:00:00'
```



date_id	start_date
4	2018-02-28 10:00:00

Search for a time that hasn't been formatted

```
SELECT * FROM date_sample
WHERE EXTRACT(HOUR_SECOND FROM start_date) = 100000
```

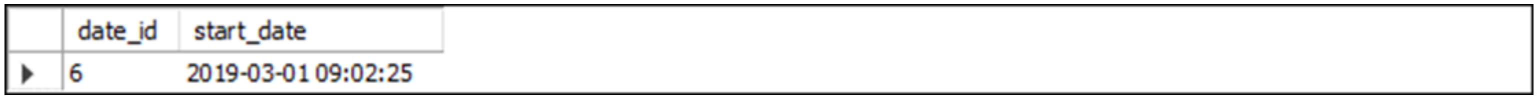


date_id	start_date
4	2018-02-28 10:00:00

Examples that ignore date values (continued)

Search for an hour of the day

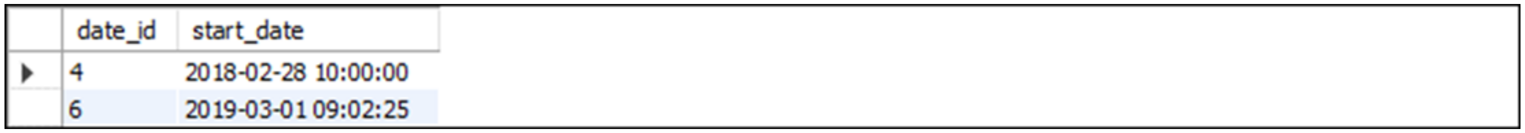
```
SELECT * FROM date_sample
WHERE HOUR(start_date) = 9
```



date_id	start_date
6	2019-03-01 09:02:25

Search for a range of times

```
SELECT * FROM date_sample
WHERE EXTRACT(HOUR_MINUTE FROM start_date)
    BETWEEN 900 AND 1200
```



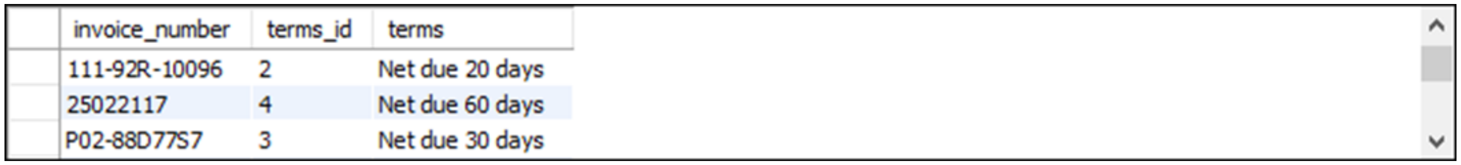
date_id	start_date
4	2018-02-28 10:00:00
6	2019-03-01 09:02:25

The syntax of the simple CASE function

```
CASE input_expression
    WHEN when_expression_1 THEN result_expression_1
   [WHEN when_expression_2 THEN result_expression_2]...
   [ELSE else_result_expression]
END
```

A statement that uses a simple CASE function

```
SELECT invoice_number, terms_id,
    CASE terms_id
        WHEN 1 THEN 'Net due 10 days'
        WHEN 2 THEN 'Net due 20 days'
        WHEN 3 THEN 'Net due 30 days'
        WHEN 4 THEN 'Net due 60 days'
        WHEN 5 THEN 'Net due 90 days'
    END AS terms
FROM invoices
```



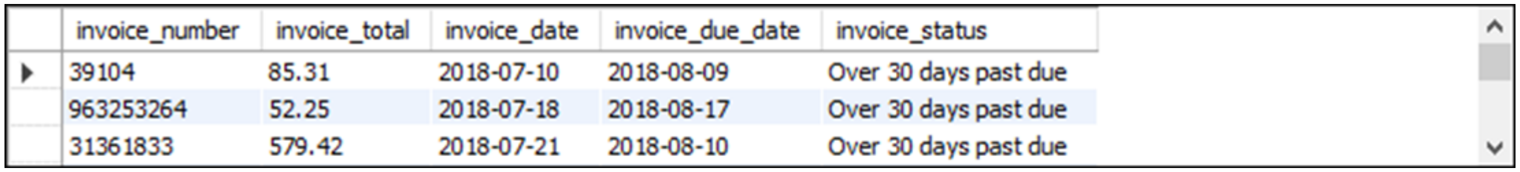
invoice_number	terms_id	terms
111-92R-10096	2	Net due 20 days
25022117	4	Net due 60 days
P02-88D77S7	3	Net due 30 days

The syntax of the searched CASE function

```
CASE
    WHEN conditional_expression_1
        THEN result_expression_1
   [WHEN conditional_expression_2
        THEN result_expression_2]...
   [ELSE else_result_expression]
END
```

A statement that uses a searched CASE function

```
SELECT invoice_number, invoice_total, invoice_date,
       invoice_due_date,
    CASE
      WHEN DATEDIFF(NOW(), invoice_due_date) > 30
        THEN 'Over 30 days past due'
      WHEN DATEDIFF(NOW(), invoice_due_date) > 0
        THEN '1 to 30 days past due'
      ELSE 'Current'
    END AS invoice_status
FROM invoices
WHERE invoice_total - payment_total - credit_total > 0
```



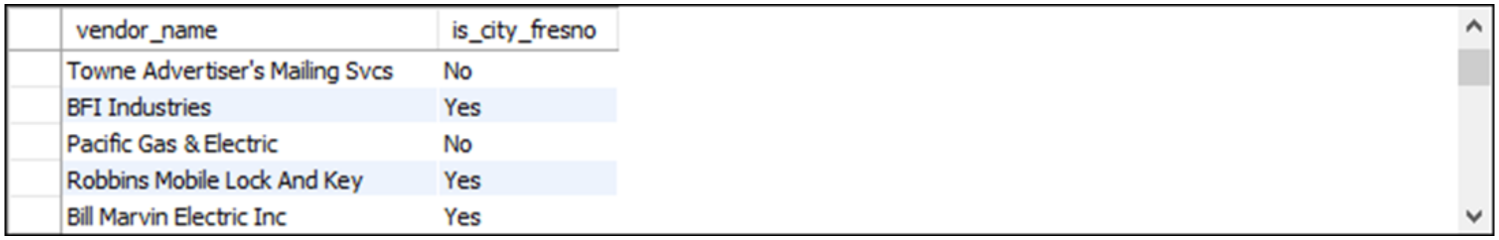
invoice_number	invoice_total	invoice_date	invoice_due_date	invoice_status
39104	85.31	2018-07-10	2018-08-09	Over 30 days past due
963253264	52.25	2018-07-18	2018-08-17	Over 30 days past due
31361833	579.42	2018-07-21	2018-08-10	Over 30 days past due

The syntax of the IF function

```
IF(test_expression, if_true_expression, else_expression)
```

A SELECT statement that uses the IF function

```
SELECT vendor_name,
    IF(vendor_city = 'Fresno', 'Yes', 'No')
    AS is_city_fresno
FROM vendors
```



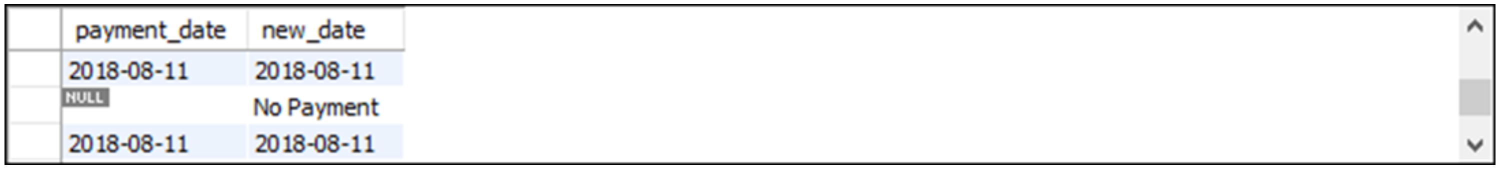
vendor_name	is_city_fresno
Towne Advertiser's Mailing Svcs	No
BFI Industries	Yes
Pacific Gas & Electric	No
Robbins Mobile Lock And Key	Yes
Bill Marvin Electric Inc	Yes

The syntax of the IFNULL function

```
IFNULL(test_expression, replacement_value)
```

A SELECT statement that uses the IFNULL function

```
SELECT payment_date,
       IFNULL(payment_date, 'No Payment') AS new_date
FROM invoices
```



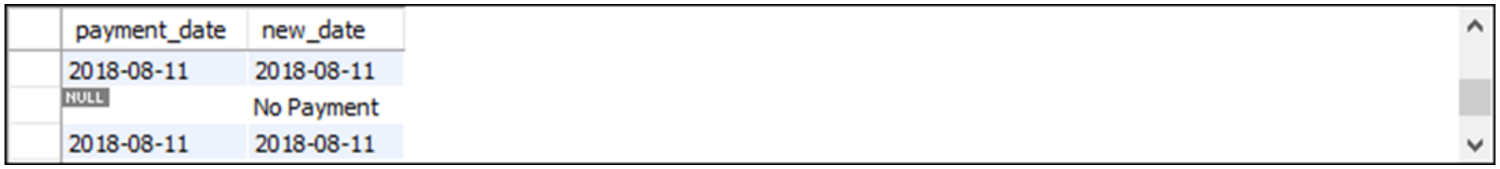
payment_date	new_date
2018-08-11	2018-08-11
NULL	No Payment
2018-08-11	2018-08-11

The syntax of the COALESCE function

```
COALESCE(expression_1[, expression_2]...)
```

A SELECT statement that uses the COALESCE function

```
SELECT payment_date,
       COALESCE(payment_date, 'No Payment') AS new_date
FROM invoices
```



payment_date	new_date
2018-08-11	2018-08-11
NULL	No Payment
2018-08-11	2018-08-11

The syntax of the regular expression functions

```
REGEXP_LIKE(expr, pattern)
REGEXP_INSTR(expr, pattern [, start])
REGEXP_SUBSTR(expr, pattern [, start])
REGEXP_REPLACE(expr, pattern, replace[, start])
```

Regular expression special characters and constructs

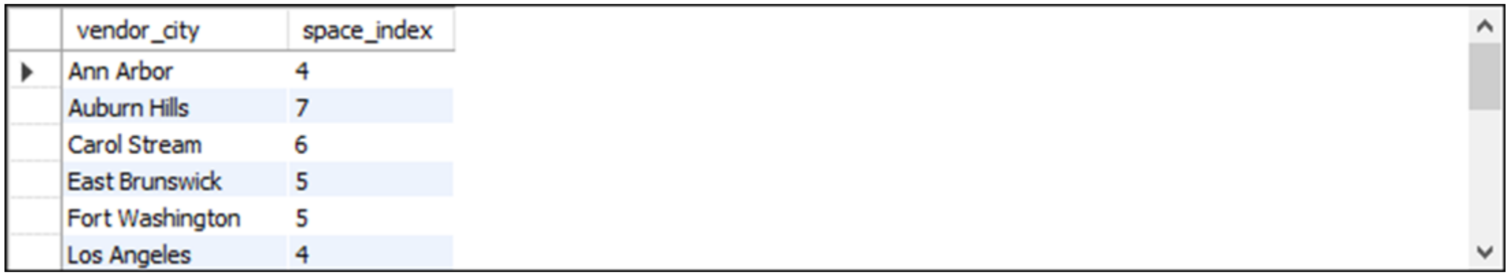
Character/ Construct	Description
`^`	Matches the pattern to the beginning of the value.
`$`	Matches the pattern to the end of the value.
`.`	Matches any single character.
`[charlist]`	Matches any single character listed within the brackets.
`[char1-char2]`	Matches any single character within the given range.
`|`	Separates two string patterns and matches either one.
`char*`	Matches zero or more occurrences of the character.
`(charlist)*`	Matches zero or more occurrences of the sequence of characters in parentheses.

Examples of the regular expression functions

Example	Result
`REGEXP_LIKE('abc123', '123')`	`1`
`REGEXP_LIKE('abc123', '^123')`	`0`
`REGEXP_INSTR('abc123', '123')`	`4`
`REGEXP_SUBSTR('abc123', '[A-Z][1-9]*$')`	`c123`
`REGEXP_REPLACE('abc123', '1|2', '3')`	`abc333`

A statement that uses the REGEXP_INSTR function

```
SELECT DISTINCT vendor_city,
    REGEXP_INSTR(vendor_city, ' ') AS space_index
FROM vendors
WHERE REGEXP_INSTR(vendor_city, ' ') > 0
ORDER BY vendor_city
```

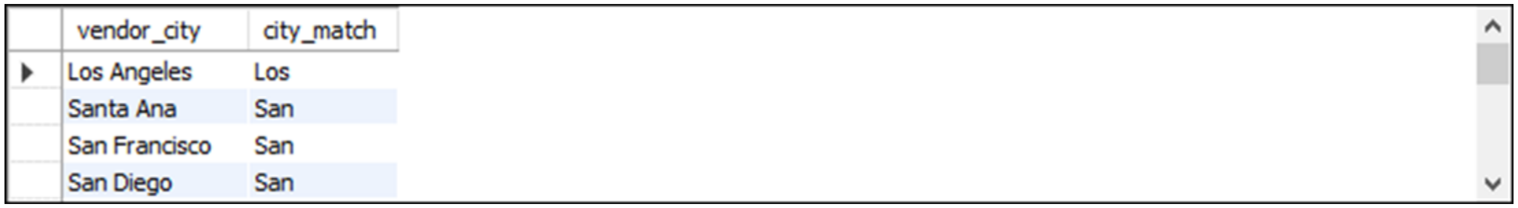


vendor_city	space_index
Ann Arbor	4
Auburn Hills	7
Carol Stream	6
East Brunswick	5
Fort Washington	5
Los Angeles	4

```
(17 rows)
```

A statement that uses the REGEXP_SUBSTR function

```
SELECT vendor_city,
    REGEXP_SUBSTR(vendor_city, '^SAN|LOS') AS city_match
FROM vendors
WHERE REGEXP_SUBSTR(vendor_city, '^SAN|LOS') IS NOT NULL
```

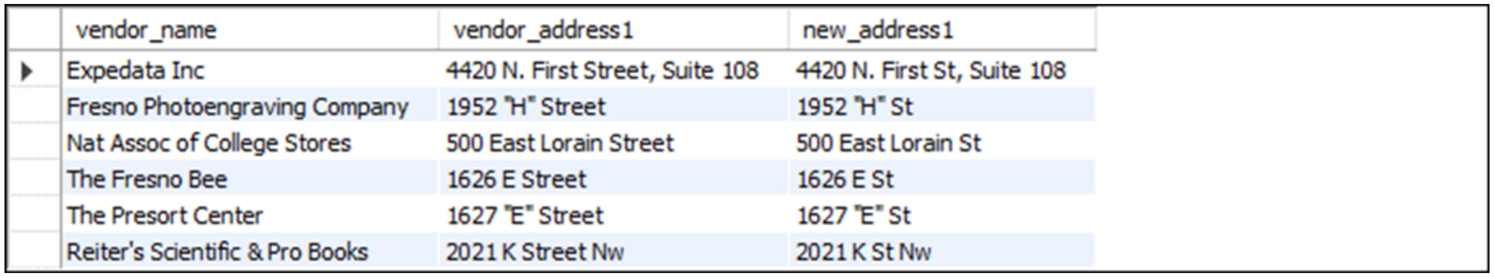


vendor_city	city_match
Los Angeles	Los
Santa Ana	San
San Francisco	San
San Diego	San

(12 rows)

A statement that uses the REGEXP_REPLACE and REGEXP_LIKE functions

```
SELECT vendor_name, vendor_address1,
       REGEXP_REPLACE(vendor_address1, 'STREET', 'St')
       AS new_address1
FROM Vendors
WHERE REGEXP_LIKE(vendor_address1, 'STREET')
```



vendor_name	vendor_address1	new_address1
Expedata Inc	4420 N. First Street, Suite 108	4420 N. First St, Suite 108
Fresno Photoengraving Company	1952 "H" Street	1952 "H" St
Nat Assoc of College Stores	500 East Lorain Street	500 East Lorain St
The Fresno Bee	1626 E Street	1626 E St
The Presort Center	1627 "E" Street	1627 "E" St
Reiter's Scientific & Pro Books	2021 K Street Nw	2021 K St Nw

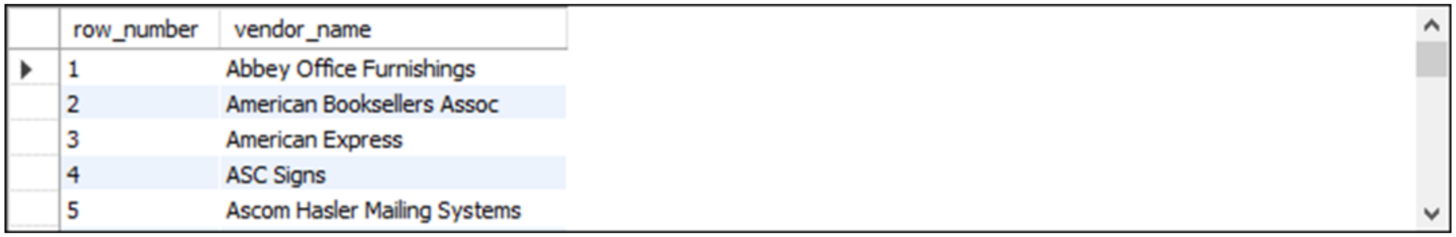
```
(4 rows)
```

The syntax of the four ranking functions

```
ROW_NUMBER()                   OVER([partition_clause] order_clause)
RANK()                         OVER([partition_clause] order_clause)
DENSE_RANK()                   OVER([partition_clause] order_clause)
NTILE(integer_expression) OVER([partition_clause] order_clause)
```

A query that uses the ROW_NUMBER function

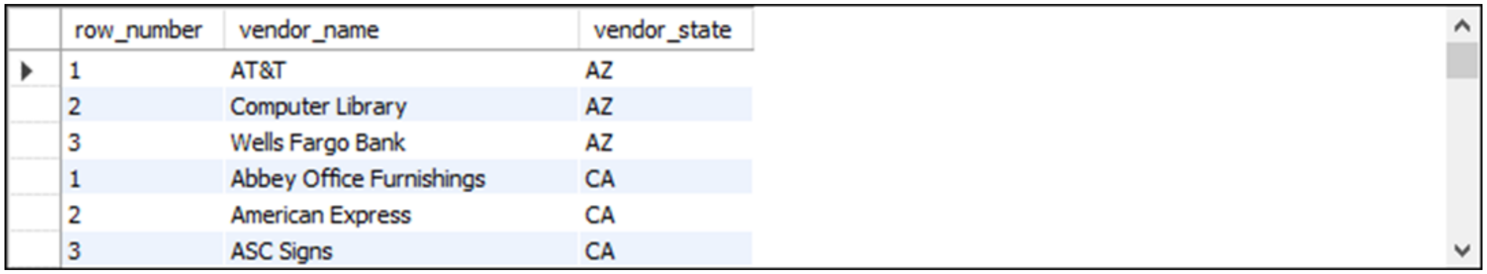
```
SELECT ROW_NUMBER() OVER(ORDER BY vendor_name)
       AS 'row_number', vendor_name
FROM vendors
```



row_number	vendor_name
1	Abbey Office Furnishings
2	American Booksellers Assoc
3	American Express
4	ASC Signs
5	Ascom Hasler Mailing Systems

A query that uses the PARTITION BY clause

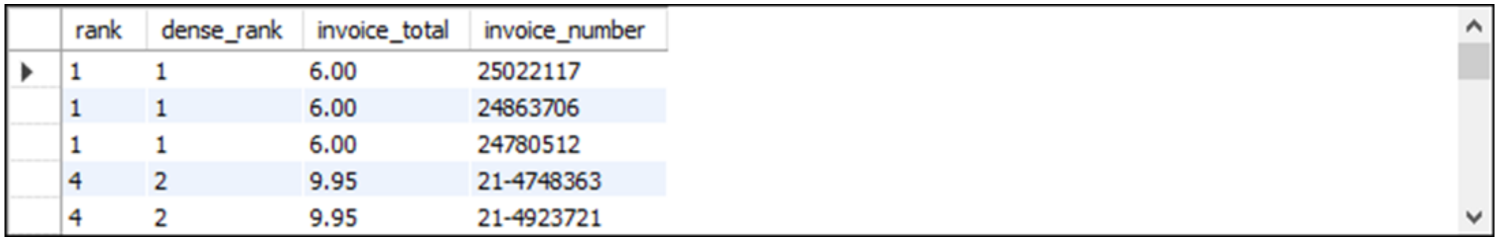
```
SELECT ROW_NUMBER() OVER(PARTITION BY vendor_state
    ORDER BY vendor_name) AS 'row_number', vendor_name,
    vendor_state
FROM vendors
```



row_number	vendor_name	vendor_state
1	AT&T	AZ
2	Computer Library	AZ
3	Wells Fargo Bank	AZ
1	Abbey Office Furnishings	CA
2	American Express	CA
3	ASC Signs	CA

A query that uses the RANK and DENSE_RANK functions

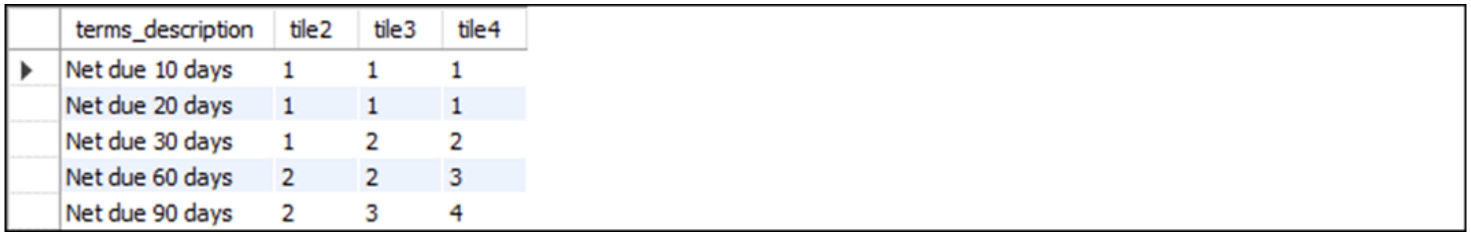
```
SELECT RANK() OVER (ORDER BY invoice_total) AS 'rank',
       DENSE_RANK() OVER (ORDER BY invoice_total)
       AS 'dense_rank', invoice_total, invoice_number
FROM invoices
```



rank	dense_rank	invoice_total	invoice_number
1	1	6.00	25022117
1	1	6.00	24863706
1	1	6.00	24780512
4	2	9.95	21-4748363
4	2	9.95	21-4923721

A query that uses the NTILE function

```
SELECT terms_description,
    NTILE(2) OVER (ORDER BY terms_id) AS tile2,
    NTILE(3) OVER (ORDER BY terms_id) AS tile3,
    NTILE(4) OVER (ORDER BY terms_id) AS tile4
FROM terms
```



terms_description	tile2	tile3	tile4
Net due 10 days	1	1	1
Net due 20 days	1	1	1
Net due 30 days	1	2	2
Net due 60 days	2	2	3
Net due 90 days	2	3	4

The syntax of the analytic functions

```
{FIRST_VALUE|LAST_VALUE|NTH_VALUE}
    (scalar_expression[, numeric_literal])
    OVER ([partition_clause] order_clause [frame_clause])
{LEAD|LAG}(scalar_expression [, offset [, default]])
    OVER ([partition_clause] order_clause)
{PERCENT_RANK()|CUME_DIST()}
    OVER ([partition_clause] order_clause)
```

The columns in the Sales_Reps table

Column name	Data type
`rep_id`	`INT`
`rep_first_name`	`VARCHAR(50)`
`rep_last_name`	`VARCHAR(50)`

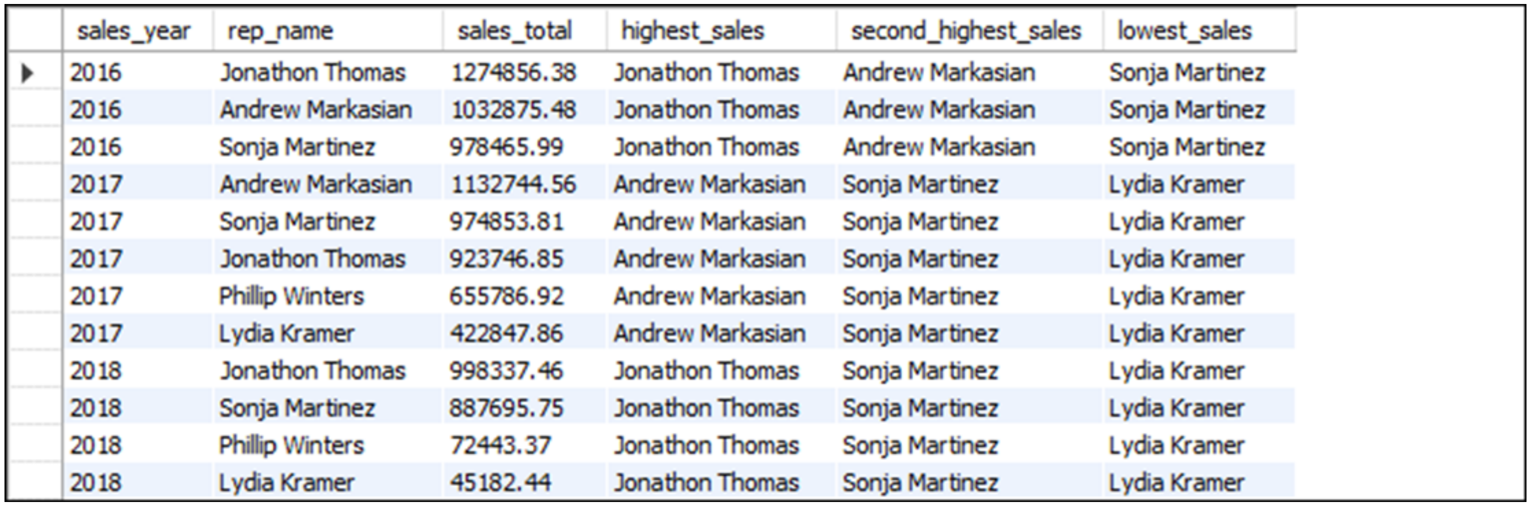
The columns in the Sales_Totals table

Column name	Data type
`rep_id`	`INT`
`sales_year`	`YEAR`
`sales_total`	`DECIMAL(9,2)`

A query that uses the FIRST_VALUE, NTH_VALUE, and LAST_VALUE functions

```
SELECT sales_year, CONCAT(rep_first_name, ' ', rep_last_name)
       AS rep_name, sales_total,
    FIRST_VALUE(CONCAT(rep_first_name, ' ', rep_last_name))
        OVER (PARTITION BY sales_year ORDER BY sales_total DESC)
        AS highest_sales,
    NTH_VALUE(CONCAT(rep_first_name, ' ', rep_last_name), 2)
        OVER (PARTITION BY sales_year ORDER BY sales_total DESC
        RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
        AS second_highest_sales,
    LAST_VALUE(CONCAT(rep_first_name, ' ', rep_last_name))
        OVER (PARTITION BY sales_year ORDER BY sales_total DESC
        RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
        AS lowest_sales
FROM sales_totals JOIN sales_reps
  ON sales_totals.rep_id = sales_reps.rep_id
```

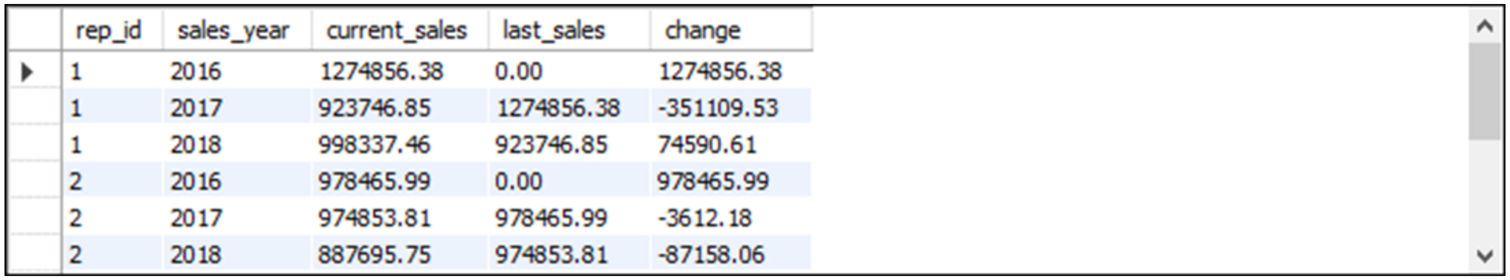
The result of the query



sales_year	rep_name	sales_total	highest_sales	second_highest_sales	lowest_sales
2016	Jonathon Thomas	1274856.38	Jonathon Thomas	Andrew Markasian	Sonja Martinez
2016	Andrew Markasian	1032875.48	Jonathon Thomas	Andrew Markasian	Sonja Martinez
2016	Sonja Martinez	978465.99	Jonathon Thomas	Andrew Markasian	Sonja Martinez
2017	Andrew Markasian	1132744.56	Andrew Markasian	Sonja Martinez	Lydia Kramer
2017	Sonja Martinez	974853.81	Andrew Markasian	Sonja Martinez	Lydia Kramer
2017	Jonathon Thomas	923746.85	Andrew Markasian	Sonja Martinez	Lydia Kramer
2017	Phillip Winters	655786.92	Andrew Markasian	Sonja Martinez	Lydia Kramer
2017	Lydia Kramer	422847.86	Andrew Markasian	Sonja Martinez	Lydia Kramer
2018	Jonathon Thomas	998337.46	Jonathon Thomas	Sonja Martinez	Lydia Kramer
2018	Sonja Martinez	887695.75	Jonathon Thomas	Sonja Martinez	Lydia Kramer
2018	Phillip Winters	72443.37	Jonathon Thomas	Sonja Martinez	Lydia Kramer
2018	Lydia Kramer	45182.44	Jonathon Thomas	Sonja Martinez	Lydia Kramer

A query that uses the LAG function

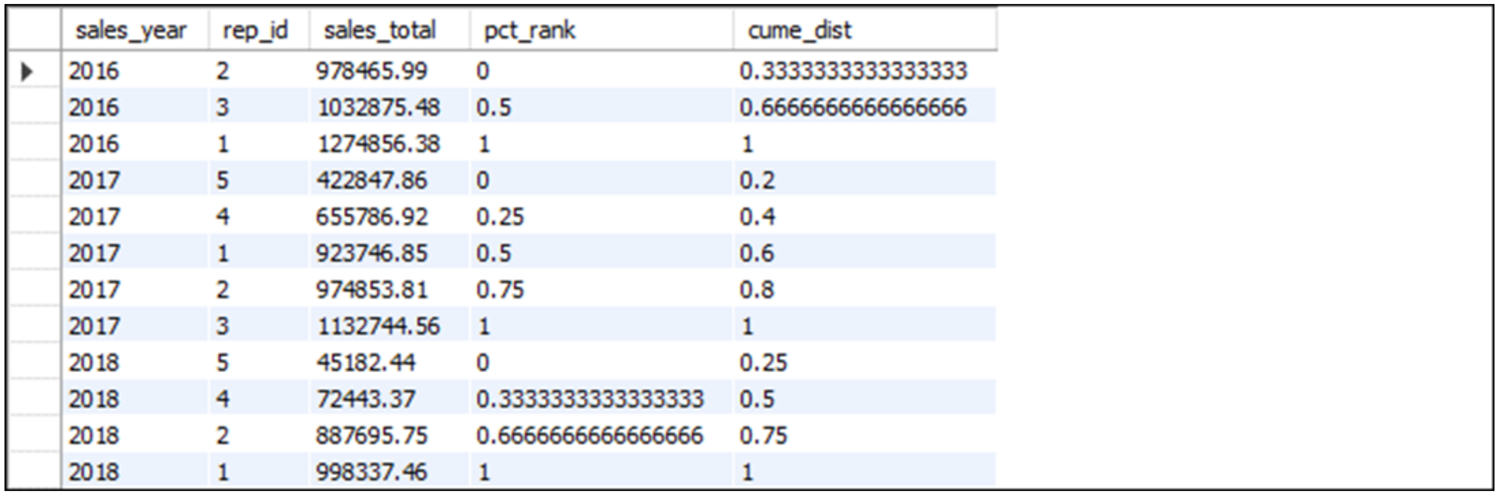
```
SELECT rep_id, sales_year, sales_total AS current_sales,
    LAG(sales_total, 1, 0)
        OVER (PARTITION BY rep_id ORDER BY sales_year)
        AS last_sales,
    Sales_total - LAG(sales_total, 1, 0)
        OVER (PARTITION BY rep_id ORDER BY sales_year)
        AS 'change'
FROM sales_totals
```



rep_id	sales_year	current_sales	last_sales	change
1	2016	1274856.38	0.00	1274856.38
1	2017	923746.85	1274856.38	-351109.53
1	2018	998337.46	923746.85	74590.61
2	2016	978465.99	0.00	978465.99
2	2017	974853.81	978465.99	-3612.18
2	2018	887695.75	974853.81	-87158.06

A query that uses the PERCENT_RANK and CUME_DIST functions

```
SELECT sales_year, rep_id, sales_total,
    PERCENT_RANK()
        OVER (PARTITION BY sales_year ORDER BY sales_total)
        AS pct_rank,
    CUME_DIST()
        OVER (PARTITION BY sales_year ORDER BY sales_total)
        AS 'cume_dist'
FROM sales_totals
```



sales_year	rep_id	sales_total	pct_rank	cume_dist
2016	2	978465.99	0	0.3333333333333333
2016	3	1032875.48	0.5	0.6666666666666666
2016	1	1274856.38	1	1
2017	5	422847.86	0	0.2
2017	4	655786.92	0.25	0.4
2017	1	923746.85	0.5	0.6
2017	2	974853.81	0.75	0.8
2017	3	1132744.56	1	1
2018	5	45182.44	0	0.25
2018	4	72443.37	0.3333333333333333	0.5
2018	2	887695.75	0.6666666666666666	0.75
2018	1	998337.46	1	1