Chapter 3

How to retrieve data from a single table

Objectives

Applied

1. Code SELECT statements that require any of the language elements presented in this chapter.

Knowledge

1. Distinguish between the base table values and the calculated values in SELECT statements.
2. Describe the use of a column alias.
3. Describe the order of precedence and the use of parentheses for arithmetic expressions.
4. Describe the use of the CONCAT function in string expressions.
5. Describe the use of functions with strings, dates, and numbers.
6. Describe the use of the DISTINCT keyword.

Objectives (continued)

7. Describe the use of comparison operators, logical operators, and parentheses in WHERE clauses.
8. Describe the use of the IN, BETWEEN, and LIKE operators in WHERE clauses.
9. Describe the use of IS NULL in a WHERE clause.
10. Describe the use of column names, column aliases, calculated values, and column numbers in ORDER BY clauses.

The basic syntax of the SELECT statement

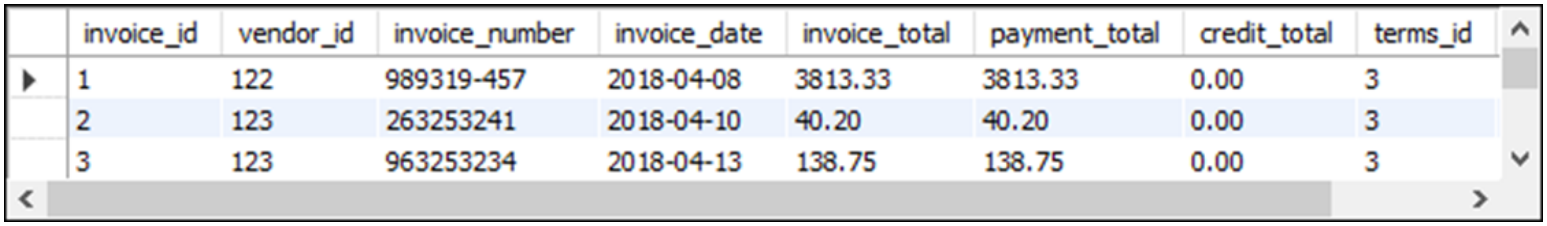
```
SELECT select_list
[FROM table_source]
[WHERE search_condition]
[ORDER BY order_by_list]
[LIMIT row_limit]
```

The five clauses of the SELECT statement

- SELECT
- FROM
- WHERE
- ORDER BY
- LIMIT

A simple SELECT statement

```
SELECT * FROM invoices
```



invoice_id	vendor_id	invoice_number	invoice_date	invoice_total	payment_total	credit_total	terms_id
1	122	989319-457	2018-04-08	3813.33	3813.33	0.00	3
2	123	263253241	2018-04-10	40.20	40.20	0.00	3
3	123	963253234	2018-04-13	138.75	138.75	0.00	3

```
(114 rows)
```

A SELECT statement that retrieves and sorts rows

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
ORDER BY invoice_total DESC
```

invoice_number	invoice_date	invoice_total
0-2058	2018-05-28	37966.19
P-0259	2018-07-19	26881.40
0-2060	2018-07-24	23517.58

```
(114 rows)
```

A SELECT statement that retrieves a calculated value

```
SELECT invoice_id, invoice_total,
       credit_total + payment_total AS total_credits
FROM invoices
WHERE invoice_id = 17
```

	invoice_id	invoice_total	total_credits
▶	17	10.00	10.00

A SELECT statement that retrieves all invoices between given dates

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
WHERE invoice_date BETWEEN '2018-06-01' AND '2018-06-30'
ORDER BY invoice_date
```

invoice_number	invoice_date	invoice_total
989319-437	2018-06-01	2765.36
111-92R-10094	2018-06-01	19.67
40318	2018-06-01	21842.00

```
(37 rows)
```

A SELECT statement that returns an empty result set

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
WHERE invoice_total > 50000
```

	invoice_number	invoice_date	invoice_total

The expanded syntax of the SELECT clause

```
SELECT [ALL|DISTINCT]
       column_specification [[AS] result_column]
    [, column_specification [[AS] result_column]] ...
```

Four ways to code column specifications

- All columns in a base table
- Column name in a base table
- Calculation
- Function

Column specifications that use base table values

The * is used to retrieve all columns

```
SELECT *
```

Column names are used to retrieve specific columns

```
SELECT vendor_name, vendor_city, vendor_state
```

Column specifications that use calculated values

An arithmetic expression that calculates the balance due

```
SELECT invoice_total - payment_total - credit_total
    AS balance_due
```

A function that returns the full name

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
```

A SELECT statement that renames the columns in the result set

```
SELECT invoice_number AS "Invoice Number",
    invoice_date AS Date, invoice_total AS Total
FROM invoices
```

	Invoice Number	Date	Total
▶	989319-457	2018-04-08	3813.33
	263253241	2018-04-10	40.20
	963253234	2018-04-13	138.75
	2-000-2993	2018-04-16	144.70
	963253251	2018-04-16	15.50
	963253261	2018-04-16	42.75

```
(114 rows)
```

A SELECT statement that doesn't name a calculated column

```
SELECT invoice_number, invoice_date, invoice_total,
       invoice_total - payment_total - credit_total
FROM invoices
```

invoice_number	invoice_date	invoice_total	invoice_total - payment_total - credit_total
989319-457	2018-04-08	3813.33	0.00
263253241	2018-04-10	40.20	0.00
963253234	2018-04-13	138.75	0.00
2-000-2993	2018-04-16	144.70	0.00
963253251	2018-04-16	15.50	0.00
963253261	2018-04-16	42.75	0.00

```
(114 rows)
```

The arithmetic operators in order of precedence

Operator	Name	Order of precedence
*	Multiplication	1
/	Division	1
DIV	Integer division	1
% (MOD)	Modulo (remainder)	1
+	Addition	2
-	Subtraction	2

A SELECT statement that calculates the balance due

```
SELECT invoice_total, payment_total, credit_total,
       invoice_total - payment_total - credit_total
       AS balance_due
FROM invoices
```

	invoice_total	payment_total	credit_total	balance_due
▶	3813.33	3813.33	0.00	0.00
	40.20	40.20	0.00	0.00
	138.75	138.75	0.00	0.00

Use parentheses to control the sequence of operations

```
SELECT invoice_id,
       invoice_id + 7 * 3 AS multiply_first,
       (invoice_id + 7) * 3 AS add_first
FROM invoices
ORDER BY invoice_id
```

invoice_id	multiply_first	add_first
1	22	24
2	23	27
3	24	30

Use the DIV and modulo operators

```
SELECT invoice_id,
       invoice_id / 3 AS decimal_quotient,
       invoice_id DIV 3 AS integer_quotient,
       invoice_id % 3 AS remainder
FROM invoices
ORDER BY invoice_id
```

invoice_id	decimal_quotient	integer_quotient	remainder
1	0.3333	0	1
2	0.6667	0	2
3	1.0000	1	0

What determines the sequence of operations

- Order of precedence
- Parentheses

The syntax of the CONCAT function

```
CONCAT(string1[, string2]...)
```

How to concatenate string data

```
SELECT vendor_city, vendor_state,
    CONCAT(vendor_city, vendor_state)
FROM vendors
```

vendor_city	vendor_state	CONCAT(vendor_city, vendor_state)
Madison	WI	MadisonWI
Washington	DC	WashingtonDC

```
(122 rows)
```

How to format string data using literal values

```
SELECT vendor_name,
       CONCAT(vendor_city, ', ', vendor_state, ' ',
              vendor_zip_code) AS address
FROM vendors
```

vendor_name	address
US Postal Service	Madison, WI 53707
National Information Data Ctr	Washington, DC 20120

```
(122 rows)
```

How to include apostrophes in literal values

```
SELECT CONCAT(vendor_name, '''s Address: ') AS Vendor,
       CONCAT(vendor_city, ', ', vendor_state, ' ',
              vendor_zip_code) AS Address
FROM vendors
```

Vendor	Address
US Postal Service's Address:	Madison, WI 53707
National Information Data Ctr's Address:	Washington, DC 20120

```
(122 rows)
```

Terms to know

- Function
- Parameter
- Argument
- Concatenate

The syntax of the LEFT function

```
LEFT(string, number_of_characters)
```

A SELECT statement that uses the LEFT function

```
SELECT vendor_contact_first_name, vendor_contact_last_name,
       CONCAT(LEFT(vendor_contact_first_name, 1),
              LEFT(vendor_contact_last_name, 1)) AS initials
FROM vendors
```

vendor_contact_first_name	vendor_contact_last_name	initials
Francesco	Alberto	FA
Ania	Irvin	AI
Lukas	Liana	LL

```
(122 rows)
```

The syntax of the DATE_FORMAT function

```
DATE_FORMAT(date, format_string)
```

A SELECT statement that uses the DATE_FORMAT function

```
SELECT invoice_date,
  DATE_FORMAT(invoice_date, '%m/%d/%y') AS 'MM/DD/YY',
  DATE_FORMAT(invoice_date, '%e-%b-%Y') AS 'DD-Mon-YYYY'
FROM invoices
ORDER BY invoice_date
```

invoice_date	MM/DD/YY	DD-Mon-YYYY
2018-04-08	04/08/18	8-Apr-2018
2018-04-10	04/10/18	10-Apr-2018
2018-04-13	04/13/18	13-Apr-2018

```
(114 rows)
```

Note

- To specify the format of a date, you use the percent sign (%) to identify a format code.

The syntax of the ROUND function

```
ROUND(number[, number_of_decimal_places])
```

A SELECT statement that uses the ROUND function

```
SELECT invoice_date, invoice_total,
       ROUND(invoice_total) AS nearest_dollar,
       ROUND(invoice_total, 1) AS nearest_dime
FROM invoices
ORDER BY invoice_date
```

invoice_date	invoice_total	nearest_dollar	nearest_dime
2018-04-08	3813.33	3813	3813.3
2018-04-10	40.20	40	40.2
2018-04-13	138.75	139	138.8

```
(114 rows)
```

A SELECT statement that tests a calculation

```
SELECT 1000 * (1 + .1) AS "10% More Than 1000"
```

	10% More Than 1000
▶	1100.0

A SELECT statement that tests the CONCAT function

```
SELECT "Ed" AS first_name, "Williams" AS last_name,
    CONCAT(LEFT("Ed", 1), LEFT("Williams", 1)) AS
initials
```

	first_name	last_name	initials
▶	Ed	Williams	EW

A SELECT statement that tests the DATE_FORMAT function

```
SELECT CURRENT_DATE,
    DATE_FORMAT(CURRENT_DATE, '%m/%d/%y') AS 'MM/DD/YY',
    DATE_FORMAT(CURRENT_DATE, '%e-%b-%Y') AS 'DD-Mon-YYYY'
```

	CURRENT_DATE	MM/DD/YY	DD-Mon-YYYY
▶	2018-11-06	11/06/18	6-Nov-2018

A SELECT statement that tests the ROUND function

```
SELECT CURRENT_DATE,
    DATE_FORMAT(CURRENT_DATE, '%m/%d/%y') AS 'MM/DD/YY',
    DATE_FORMAT(CURRENT_DATE, '%e-%b-%Y') AS 'DD-Mon-YYYY'
```

	CURRENT_DATE	MM/DD/YY	DD-Mon-YYYY
▶	2018-11-06	11/06/18	6-Nov-2018

A SELECT statement that returns all rows

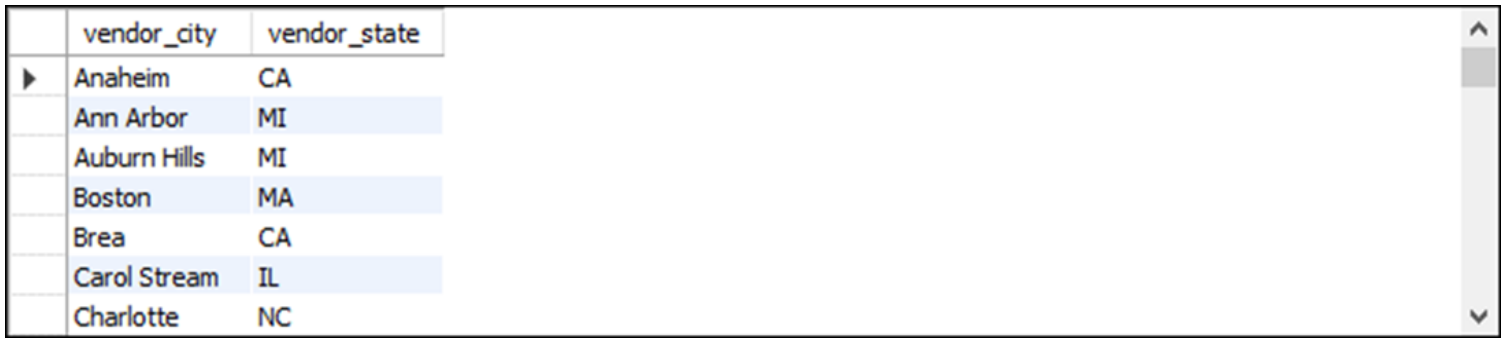
```
SELECT vendor_city, vendor_state
FROM vendors
ORDER BY vendor_city
```

vendor_city	vendor_state
Anaheim	CA
Anaheim	CA
Ann Arbor	MI
Auburn Hills	MI
Boston	MA
Boston	MA
Boston	MA

```
(122 rows)
```

A SELECT statement that eliminates duplicate rows

```
SELECT DISTINCT vendor_city, vendor_state
FROM vendors
ORDER BY vendor_city
```



vendor_city	vendor_state
Anaheim	CA
Ann Arbor	MI
Auburn Hills	MI
Boston	MA
Brea	CA
Carol Stream	IL
Charlotte	NC

```
(53 rows)
```

The syntax of the WHERE clause with comparison operators

```
WHERE expression_1 operator expression_2
```

The comparison operators

```
=
<
>
<=
>=
<>
!=
```

Examples of WHERE clauses that retrieve…

Vendors located in Iowa

```
WHERE vendor_state = 'IA'
```

Invoices with a balance due (two variations)

```
WHERE invoice_total - payment_total - credit_total > 0
WHERE invoice_total > payment_total + credit_total
```

Vendors with names from A to L

```
WHERE vendor_name < 'M'
```

Invoices on or before a specified date

```
WHERE invoice_date <= '2018-07-31'
```

Invoices on or after a specified date

```
WHERE invoice_date >= '2018-07-01'
```

Invoices with credits that don't equal zero (two variations)

```
WHERE credit_total <> 0
WHERE credit_total != 0
```

The syntax of the WHERE clause with logical operators

```
WHERE [NOT] search_condition_1 {AND|OR}
      [NOT] search_condition_2 ...
```

Examples of WHERE clauses that use logical operators

The AND operator

```
WHERE vendor_state = 'NJ' AND vendor_city = 'Springfield'
```

The OR operator

```
WHERE vendor_state = 'NJ' OR vendor_city = 'Pittsburg'
```

The NOT operator

```
WHERE NOT vendor_state = 'CA'
```

Examples of WHERE clauses that use logical operators (continued)

The NOT operator in a complex search condition

```
WHERE NOT (invoice_total >= 5000
   OR NOT invoice_date <= '2018-08-01')
```

The same condition rephrased to eliminate the NOT operator

```
WHERE invoice_total < 5000
  AND invoice_date <= '2018-08-01'
```

A compound condition without parentheses

```
WHERE invoice_date > '2018-07-03' OR invoice_total > 500
  AND invoice_total - payment_total - credit_total > 0
```

invoice_number	invoice_date	invoice_total	balance_due
203339-13	2018-07-05	17.50	0.00
111-92R-10093	2018-07-06	39.77	0.00
963253258	2018-07-06	111.00	0.00

```
(33 rows)
```

The order of precedence for compound conditions

- NOT
- AND
- OR

The same compound condition with parentheses

```
WHERE (invoice_date > '2018-07-03' OR invoice_total > 500)
  AND invoice_total - payment_total - credit_total > 0
```

invoice_number	invoice_date	invoice_total	balance_due
39104	2018-07-10	85.31	85.31
963253264	2018-07-18	52.25	52.25
31361833	2018-07-21	579.42	579.42

```
(11 rows)
```

The syntax of the WHERE clause with an IN phrase

```
WHERE test_expression [NOT] IN
      ({subquery|expression_1 [, expression_2]...})
```

Examples of the IN phrase

An IN phrase with a list of numeric literals

```
WHERE terms_id IN (1, 3, 4)
```

An IN phrase preceded by NOT

```
WHERE vendor_state NOT IN ('CA', 'NV', 'OR')
```

An IN phrase with a subquery

```
WHERE vendor_id IN
    (SELECT vendor_id
     FROM invoices
     WHERE invoice_date = '2018-07-18')
```

The syntax of the WHERE clause with a BETWEEN phrase

```
WHERE test_expression [NOT] BETWEEN
      begin_expression AND end_expression
```

Examples of the BETWEEN phrase

A BETWEEN phrase with literal values

```
WHERE invoice_date BETWEEN '2018-06-01' AND '2018-06-30'
```

A BETWEEN phrase preceded by NOT

```
WHERE vendor_zip_code NOT BETWEEN 93600 AND 93799
```

A BETWEEN phrase with a test expression coded as a calculated value

```
WHERE invoice_total - payment_total - credit_total
    BETWEEN 200 AND 500
```

A BETWEEN phrase with upper and lower limits

```
WHERE payment_total
    BETWEEN credit_total AND credit_total + 500
```

The syntax of the WHERE clause with a LIKE phrase

```
WHERE match_expression [NOT] LIKE pattern
```

Wildcard symbols

%

_

WHERE clauses that use the LIKE operator

Example 1

```
WHERE vendor_city LIKE 'SAN%'
```

Cities that will be retrieved

"San Diego", "Santa Ana"

Example 2

```
WHERE vendor_name LIKE 'COMPU_ER%'
```

Vendors that will be retrieved

"Compuserve", "Computerworld"

The syntax of the WHERE clause with a REGEXP phrase

```
WHERE match_expression [NOT] REGEXP pattern
```

REGEXP special characters and constructs

```
^
$
.
[charlist]
[char1-char2]
|
```

WHERE clauses that use REGEXP (part 1)

Example 1

```
WHERE vendor_city REGEXP 'SA'
```

Cities that will be retrieved

"Pasadena", "Santa Ana"

Example 2

```
WHERE vendor_city REGEXP '^SA'
```

Cities that will be retrieved

"Santa Ana", "Sacramento"

Example 3

```
WHERE vendor_city REGEXP 'NA$'
```

Cities that will be retrieved

"Gardena", "Pasadena", "Santa Ana"

WHERE clauses that use REGEXP (part 2)

Example 4

```
WHERE vendor_city REGEXP 'RS|SN'
```

Cities that will be retrieved

"Traverse City", "Fresno"

Example 5

```
WHERE vendor_state REGEXP 'N[CV]'
```

States that will be retrieved

"NC" and "NV" but not "NJ" or "NY"

Example 6

```
WHERE vendor_state REGEXP 'N[A-J]'
```

States that will be retrieved

"NC" and "NJ" but not "NV" or "NY"

WHERE clauses that use REGEXP (part 3)

Example 7

```
WHERE vendor_contact_last_name REGEXP 'DAMI[EO]N'
```

Last names that will be retrieved

"Damien" and "Damion"

Example 8

```
WHERE vendor_city REGEXP '[A-Z][AEIOU]N$'
```

Cities that will be retrieved

"Boston", "Mclean", "Oberlin"

The syntax of the WHERE clause with the IS NULL clause

```
WHERE expression IS [NOT] NULL
```

The contents of the Null_Sample table

```
SELECT * FROM null_sample
```

invoice_id	invoice_total
1	125.00
2	0.00
3	NULL
4	2199.99
5	0.00

A SELECT statement that retrieves rows with zero values

```
SELECT * FROM null_sample
WHERE invoice_total = 0
```

invoice_id	invoice_total
2	0.00
5	0.00

A SELECT statement that retrieves rows with non-zero values

```
SELECT * FROM null_sample
WHERE invoice_total <> 0
```

invoice_id	invoice_total
1	125.00
4	2199.99

A SELECT statement that retrieves rows with null values

```
SELECT *
FROM null_sample
WHERE invoice_total IS NULL
```

	invoice_id	invoice_total
▶	3	NULL

A SELECT statement that retrieves rows without null values

```
SELECT *
FROM null_sample
WHERE invoice_total IS NOT NULL
```

	invoice_id	invoice_total
▶	1	125.00
	2	0.00
	4	2199.99
	5	0.00

The expanded syntax of the ORDER BY clause

```
ORDER BY expression [ASC|DESC][, expression [ASC|DESC]]
...
```

An ORDER BY clause that sorts by one column

```
SELECT vendor_name,
    CONCAT(vendor_city, ', ', vendor_state, ' ',
    vendor_zip_code) AS address
FROM vendors
ORDER BY vendor_name
```

vendor_name	address
Abbey Office Furnishings	Fresno, CA 93722
American Booksellers Assoc	Tarrytown, NY 10591
American Express	Los Angeles, CA 90096
ASC Signs	Fresno, CA 93703

The default sequence for an ascending sort

- Null values
- Special characters
- Numbers
- Letters

Note

- Null values appear first in the sort sequence, even if you're using DESC.

An ORDER BY clause that sorts by one column in descending sequence

```
SELECT vendor_name,
    CONCAT(vendor_city, ', ', vendor_state, ' ',
    vendor_zip_code) AS address
FROM vendors
ORDER BY vendor_name DESC
```

vendor_name	address
Zylka Design	Fresno, CA 93711
Zip Print & Copy Center	Fresno, CA 93777
Zee Medical Service Co	Washington, IA 52353
Yesmed, Inc	Fresno, CA 93718

An ORDER BY clause that sorts by three columns

```
SELECT vendor_name,
    CONCAT(vendor_city, ', ', vendor_state, ' ',
    vendor_zip_code) AS address
FROM vendors
ORDER BY vendor_state, vendor_city, vendor_name
```

vendor_name	address
AT&T	Phoenix, AZ 85062
Computer Library	Phoenix, AZ 85023
Wells Fargo Bank	Phoenix, AZ 85038
Aztek Label	Anaheim, CA 92807

An ORDER BY clause that uses an alias

```
SELECT vendor_name,
    CONCAT(vendor_city, ', ', vendor_state, ' ',
    vendor_zip_code) AS address
FROM vendors
ORDER BY address, vendor_name
```

vendor_name	address
Aztek Label	Anaheim, CA 92807
Blue Shield of California	Anaheim, CA 92850
Malloy Lithographing Inc	Ann Arbor, MI 48106
Data Reproductions Corp	Auburn Hills, MI 48326

An ORDER BY clause that uses an expression

```
SELECT vendor_name,
    CONCAT(vendor_city, ', ', vendor_state, ' ',
    vendor_zip_code) AS address
FROM vendors
ORDER BY CONCAT(vendor_contact_last_name,
                vendor_contact_first_name)
```

vendor_name	address
Dristas Groom & McCormick	Fresno, CA 93720
Internal Revenue Service	Fresno, CA 93888
US Postal Service	Madison, WI 53707
Yale Industrial Trucks-Fresno	Fresno, CA 93706

An ORDER BY clause that uses column positions

```
SELECT vendor_name,
    CONCAT(vendor_city, ', ', vendor_state, ' ',
    vendor_zip_code) AS address
FROM vendors
ORDER BY 2, 1
```

vendor_name	address
Aztek Label	Anaheim, CA 92807
Blue Shield of California	Anaheim, CA 92850
Malloy Lithographing Inc	Ann Arbor, MI 48106
Data Reproductions Corp	Auburn Hills, MI 48326

The expanded syntax of the LIMIT clause

```
LIMIT [offset,] row_count
```

A SELECT statement with a LIMIT clause that starts with the first row

```
SELECT vendor_id, invoice_total
FROM invoices
ORDER BY invoice_total DESC
LIMIT 5
```

vendor_id	invoice_total
110	37966.19
110	26881.40
110	23517.58
72	21842.00
110	20551.18

A SELECT statement with a LIMIT clause that starts with the third row

```
SELECT invoice_id, vendor_id, invoice_total
FROM invoices
ORDER BY invoice_id
LIMIT 2, 3
```

invoice_id	vendor_id	invoice_total
3	123	138.75
4	123	144.70
5	123	15.50

A SELECT statement with a LIMIT clause that starts with the 101st row

```
SELECT invoice_id, vendor_id, invoice_total
FROM invoices
ORDER BY invoice_id
LIMIT 100, 1000
```

invoice_id	vendor_id	invoice_total
101	123	30.75
102	110	20551.18
103	122	2051.59
104	123	44.44

```
(14 rows)
```