Tengri Documentation

Version stable, 2025-10-08

Table of Contents

Tengri **Data** is an enterprise analytics platform.

Challenges

While time-tested tools such as PostgreSQL exist for the operational activities of large companies, choosing the best solution for data analytics is still a major challenge.

Traditional database management systems can handle up to **1TB** of analysed data and support 10-20 analysts. However, further growth in analytics needs, the introduction of AI and ML, and an increase in the number of users and data domains can severely increase the timeline and reduce the reliability of results.

▼ *For more information on the **3V** big data issues — Volume, Velocity, Variety*

Traditionally, when working with big data, a list of the **3 V** problems has been identified:

1. Volume

The data to be analysed is getting bigger and bigger, we are talking about tens and hundreds of TB.

2. Velocity

There are increasing demands for speed of response, the number of analysts and models analysing the data is growing. The velocity of data growth is also increasing.

3. Variety

The number and variety of data source formats is increasing.

Later, other **V** issues were added to this list of basic problems, including:

4. **Veracity** — reliability of data
5. **Variability** — variability of data
6. **Visualisation** — visualisation of data
7. **Value** — value of data
8. **Venue** — where the data is stored and used
9. **Vocabulary** — the need for a common glossary for working with data
10. **Vagueness** — uncertainty of some data types

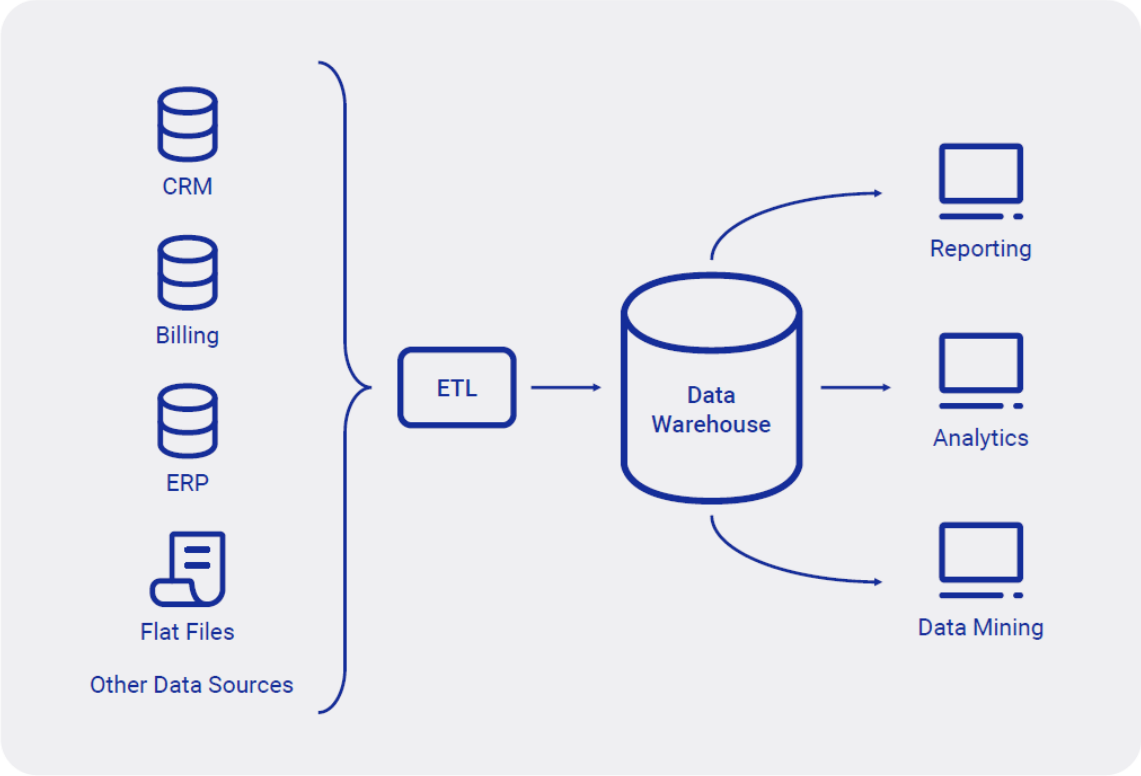
All these limitations and challenges point to the need for a fundamentally new approach when building corporate systems for analysing big data.

What Tengri offers

Separation

The separation of **Compute & Storage** principle enables data storage and compute resources to scale

flexibly and independently without multiples of infrastructure costs and adaptively fit into today's hybrid enterprise installations.



> Each request has its own resource: analytics, reporting or machine learning services do not have to compete with each other
>
> — Nikolay Golov, Tengri Data Product Director

Convenience

Using familiar language SQL and standard connections ODBC/JDBC, analysts can get answers in seconds — whether the data occupies **1 GB** or **1 PB**, and whether there are competing queries or not. And open data formats provide maximum flexibility and scalability.

> Analysts can focus on solving business problems without being distracted by the technical nuances of the platform.
>
> — Mikhail Tyurin, Tengri Data Department Director

Result

The success of this approach in the global market is confirmed by the commercial achievements of **Snowflake** and **Databricks**, whose solutions are not available to companies in Russia. Tengri provides similar and sometimes superior capabilities using an even more advanced technology stack.

> We're building a platform that opens the door to truly unlimited analytics, removing barriers to unlocking the full potential of data.

> — Nikolay Golov, Tengri Data Product Director

Architecture

This section describes the main components of the architecture Tengri.

- **Ideology of use**

  About the ideology of use Tengri

- **Compute & Storage partitioning**

  On the principle of separating compute and storage systems in the Tengri

- **Storage**

  On the storage system in Tengri

- **Calculation**

  On how computation is organised in the Tengri

- **PostgreSQL dialect and protocols**

  About dialect support SQL PostgreSQL in Tengri

- **Developed system of differentiation of rights**

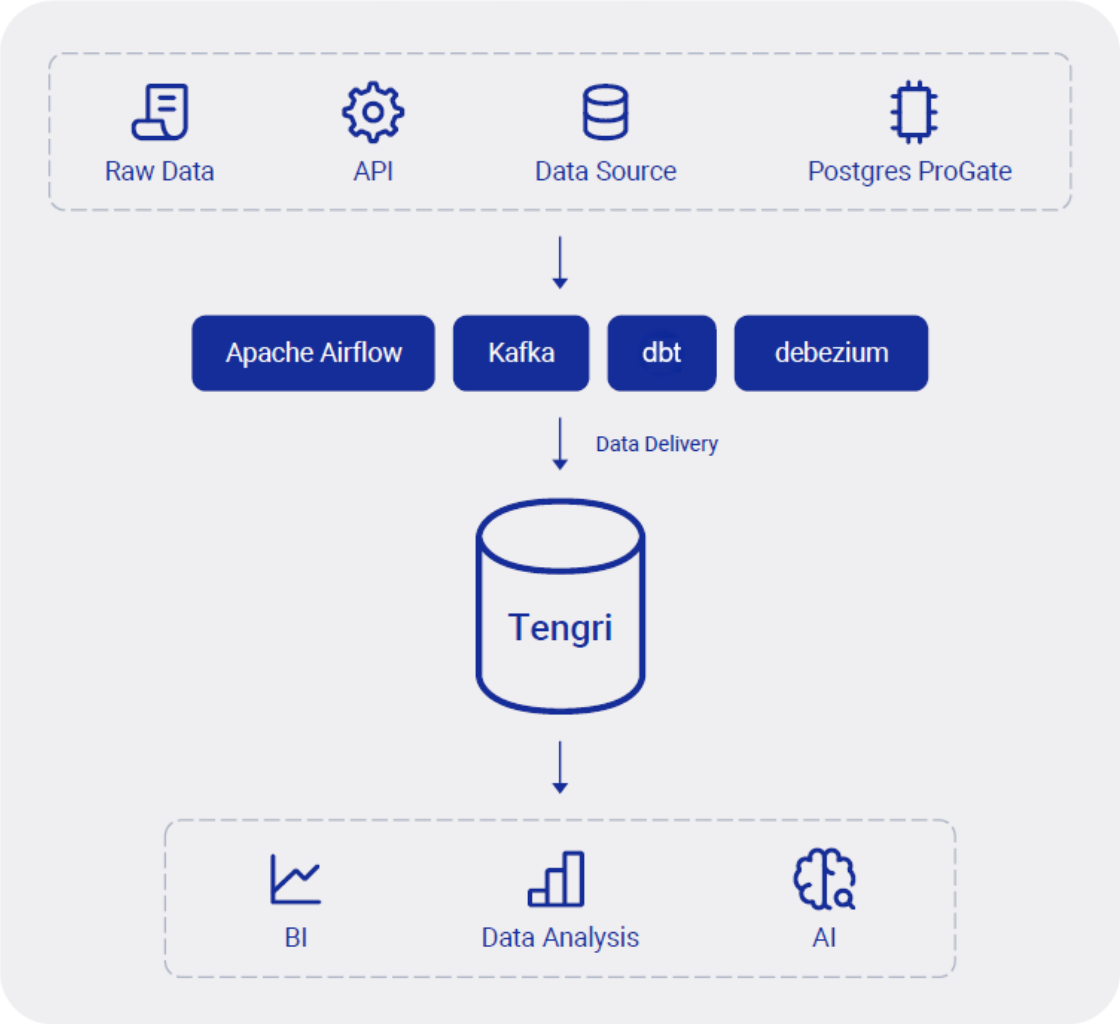
  About access rights and user roles in Tengri

Ideology of use

General usage pattern for Tengri

In Tengri data:

1. come from many different sources (API, CRM, ERP, JSON, billings, other sources);
2. are regularly uploaded into a single platform;
3. analysed and aggregated;
4. generate reports and graphs;
5. serve to train ML/AI models.



Tengri implements

- ☑ Support for OLAP SQL queries
- ☑ Easy integration with ETL/BI/ML-systems
- ☑ Unlimited scalability
- ☑ Data integrity and security guarantees
- ☑ Access rights differentiation system

Compute & Storage partitioning

Tengri — a next-generation analytics platform based on the **Compute & Storage** paradigm.

- Data is stored in a universally scalable cluster S3.
- Computing power is dynamically selected using isolated compute engines running on a containerisation platform.

This approach allows:

- Flexibly and independently scale storage and compute resources without multiples of infrastructure costs;
- Adaptively fit into modern hybrid enterprise installations.

Compute and storage systems are described at Calculation and Storage, respectively.

Storage

Apache Iceberg in S3

Table data in Tengri is stored in the **S3** cluster in the open format **Apache Iceberg**.

- **Iceberg** supports `INSERT`, `UPDATE`, and `DELETE` operations, as well as table schema changes.
- **Iceberg** supports ACID, a set of requirements for a transactional system to ensure that it operates in the most reliable and predictable manner.
- **Iceberg**-structure is formed based on **Parquet** format with column-by-column data compression.
- **Iceberg** is thousands of successful implementations by the world's largest companies to work with hundreds of **TB** and dozens of **PB** data.

Columnar storage format

By utilising the **Parquet** format, Tengri supports:

Columnar storage format

OLAP- queries run faster by reading only the required table columns.

Column-by-column data compression

OLAP- queries work faster by a factor of 5 or more by compressing table data.

Calculation

In Tengri, computation is performed in virtualised compute pools.

Compute pools

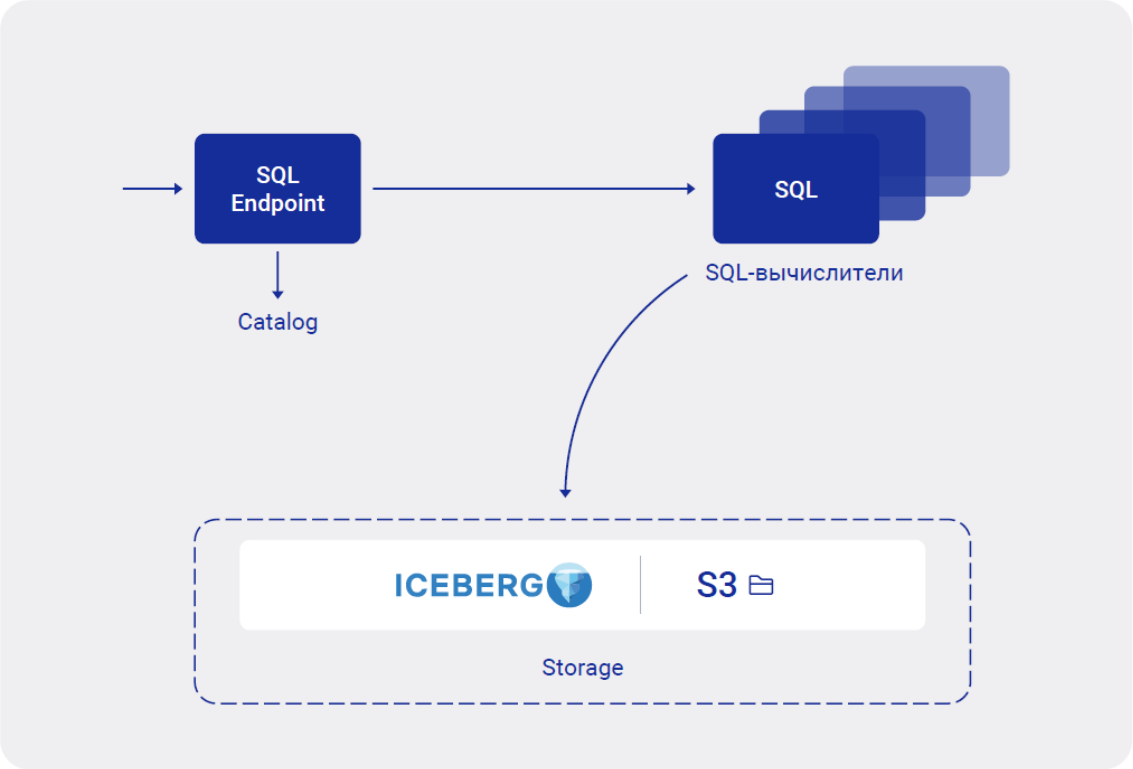
SQLA -computing pool is a lightweight container that runs on a hardware cluster in ~100 milliseconds.

SQL-computers:

- receives exclusive cores (vCPU) and RAM-budget
- loads data and executes SQL-queries
- is switched off by the system after some waiting period (if idle without load)

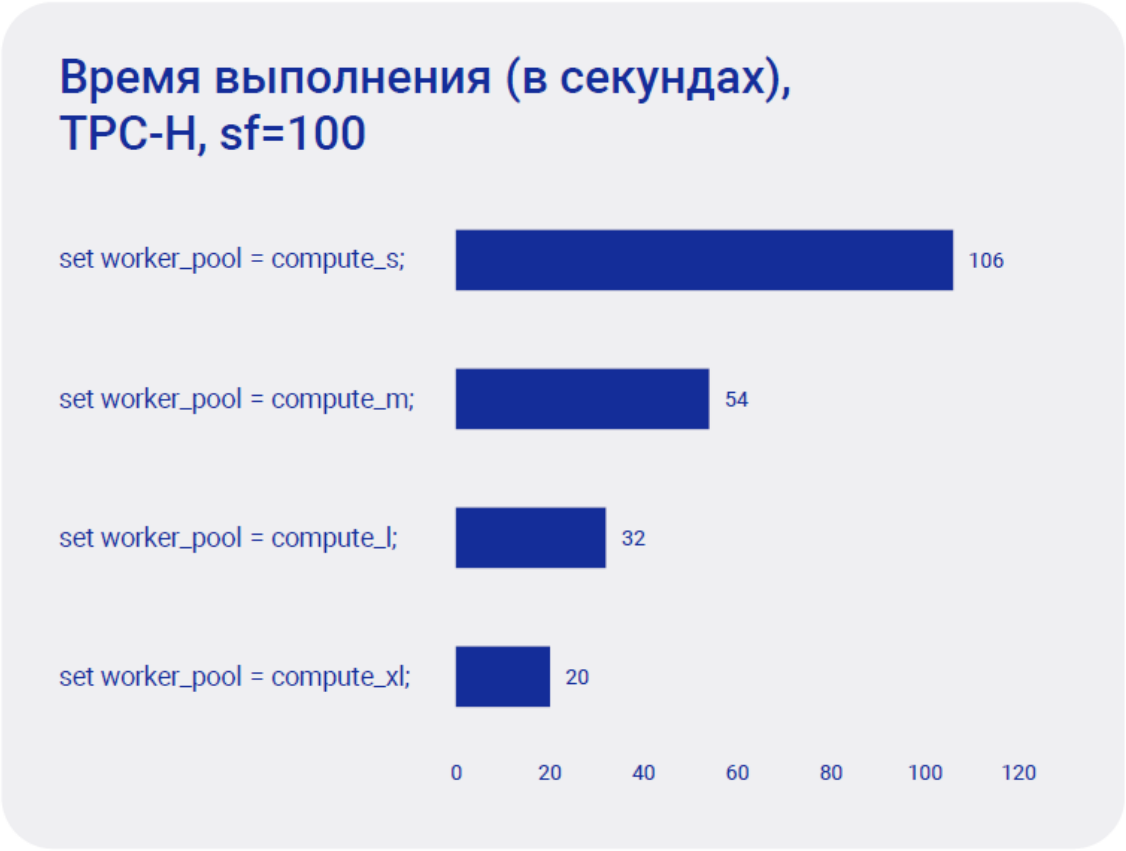
Due to this compute architecture:

- Each analyst works with its own SQL-computer.
- Analysts do not interfere with each other.



Each analyst in each session can select the size of the calculator for SQL-query:
`XS` / `S` / `M` / `L` / `XL`. Each next calculator size is twice the size of the previous one.

Execution time of **TPC-H** test on different calculator sizes:



▼ *Code used to execute this test*

```
SELECT profit.nation AS nation,
profit.o_year AS o_year,
SUM(profit.amount) AS sum_profit

FROM (SELECT s100_nation.n_name AS nation,

EXTRACT(YEAR FROM s100_orders.o_orderdate) AS o_year,

s100_lineitem.l_extendedprice * (1 - s100_lineitem.l_discount) -
s100_partsupp.ps_supplycost * s100_lineitem.l_quantity AS amount

FROM s100_part, s100_supplier, s100_lineitem,
s100_partsupp, s100_orders, s100_nation

WHERE s100_supplier.s_suppkey = s100_lineitem.l_suppkey
AND s100_partsupp.ps_suppkey = s100_lineitem.l_suppkey
AND s100_partsupp.ps_partkey = s100_lineitem.l_partkey
AND s100_part.p_partkey = s100_lineitem.l_partkey
AND s100_orders.o_orderkey = s100_lineitem.l_orderkey
AND s100_supplier.s_nationkey = s100_nation.n_nationkey
AND s100_part.p_name LIKE '%' || 'black' || '%')
AS profit

GROUP BY profit.nation, profit.o_year ORDER BY nation, o_year
DESC NULLS FIRST LIMIT 1000;
```

Computers are also supported in Tengri Python .

Accelerating calculations

To accelerate computations, Tengri supports:

Morsel-driven parallelism
: OLAP SQL-queries run faster by using **Morsel** algorithms to load all vCPU, allocated by the system.

SIMD-instructions
: OLAP SQL-queries run faster by using SIMD-instructions of modern CPU.

- Performance tests of Tengri are given in Test results.
- For more details on computing performance, see Computing pools.

PostgreSQL dialect and protocols

PostgreSQL

Support for SQL (dialect PostgreSQL) is implemented in Tengri.

In particular supported:

- Basic syntax elements of SQL dialect PostgreSQL:
  `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `JOIN`, `WITH`, `UNION`.
- Window functions
- Working with JSON via SQL
- Principles ACID
- Transactions
- Transactional data integrity
- Data isolation at the **Repeatable read** level
- JDBC-driver PostgreSQL

Data loading

The operability of data loading tools (**Airflow**, **dbt**) and analytics tools (BI, ML) is guaranteed due to the ability to connect via JDBC-driver PostgreSQL.

All BI- and ETL-tools work as if it were PostgreSQL.

> Analysts don’t need to retrain on new tools.

Developed system of differentiation of rights

An advanced system of access rights differentiation is implemented in Tengri.

Supported principles

Tengri supports the principles of DAC and RBAC.

DAC

Discretionary Access Control: each object has an owner who can grant access rights (privileges) to the object to other users.

RBAC

Role-based access control: access rights (privileges) are assigned to roles, which in turn are assigned to users.

Key concepts of a rights-based system

Protected object

An entity to which access (privileges) can be granted. If access to this protected object is not allowed, it will be denied.

Role

An entity to which access rights (privileges) can be granted. Roles can be assigned to users or to other roles. Assigning a role to another role creates a **role hierarchy**.

Privilege

A specific level of access to an object. Assigned to users or roles. Privileges assigned to roles or users allow access to objects to be protected. Can be revoked from roles or users.

Several different privileges can be used simultaneously to control the granularity of the access granted.

User

An identifier associated with a person or service. A user is an object that can be granted privileges.

For details on how to work with the rights differentiation system, refer to the sections:

- User operations

- Role operations
- Privilege operations

Test results

The following tests are used to evaluate performance:

- TPC-H (*Transaction Processing Performance Council - High Performance*) is a benchmark for evaluating the performance of decision support systems and data warehouses. It includes a set of business-oriented analytical queries and parallel data modifications that simulate real-life scenarios of using such systems.
- TPC-DS (*Transaction Processing Performance Council - Decision Support*) is a benchmark for evaluating the performance of decision support systems and analytical databases. It simulates real-world big data scenarios typical of the data centres of retail organisations and other large companies.

Results:

- **TPC-H**, Comparison with Greenplum
- **TPC-DS**

TPC-H, Comparison with Greenplum

TPC-H (*Transaction Processing Performance Council - High Performance*) is a benchmark for evaluating the performance of decision support systems and data warehouses. It includes a set of business-oriented analytical queries and parallel data modifications that simulate real-life scenarios of using such systems.

Test parameters

Benchmark	TPC-H
Scaling factor	`100`
Query	random `Q1`-`Q10`
Parameters	random
Number of threads	`1, 5, 10, 20`
Time	1 hour

Examples of test queries

▼ *See examples of test SQL-queries*

```
--Q1. Pricing Summary Report Query
--inputs = [{"depth": random.choice(range(90))}]
select
      l_returnflag,
      l_linestatus,
      sum(l_quantity) as sum_qty,
      sum(l_extendedprice) as sum_base_price,
```

```
      sum(l_extendedprice * (1-l_discount)) as sum_disc_price,
      sum(l_extendedprice * (1-l_discount) * (1+l_tax)) as sum_charge,
      avg(l_quantity) as avg_qty,
      avg(l_extendedprice) as avg_price,
      avg(l_discount) as avg_disc,
      count(*) as count_order
from
      s1_lineitem
where
      l_shipdate <= (DATE '1998-12-01') - :depth * interval '1' day
group by
      l_returnflag,
      l_linestatus
order by
      l_returnflag,
      l_linestatus;


--Q2. Minimum Cost Supplier Query
--inputs.size = random.choice(range(50))
--inputs.type = random.choice(['TIN','NICKEL', 'BRASS', 'STEEL', 'COPPER'])
--inputs.region = random.choice(['AFRICA','AMERICA','ASIA','EUROPE','MIDDLE EAST'])
select s_acctbal,s_name,n_name,p_partkey,p_mfgr,s_address,s_phone,s_comment
from s1_part, s1_supplier,s1_partsupp,s1_nation,s1_region
where p_partkey = ps_partkey
and s_suppkey = ps_suppkey
and p_size = :size
and p_type like '%' || :type
and s_nationkey = n_nationkey
and n_regionkey = r_regionkey
and r_name = :region
and ps_supplycost = (
    Select min(ps_supplycost)
    from s1_partsupp, s1_supplier, s1_nation, s1_region
    Where p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = :region
)
order by s_acctbal desc,n_name,s_name,p_partkey;
```

Systems under test

- Tengri
  - 100 vCPU
  - 200GB RAM
- **Greenplum** [1]
  - 1 coordinator + 3 nodes of 30 each vCPU, 64GB RAM

Results

Number of streams	Tengri, requests per hour	Greenplum, requests per hour
1	302	304
5	1466	1082
10	2890	1444
20	4111	1800

When the system is used by a single analyst, the performance of Tengri and **Greenplum** is comparable:

When the system is used by 10 analysts, Tengri outperforms **Greenplum** due to efficient resource utilisation without competition:

Tengri scales linearly as the number of analysts increases, efficiently utilising computing resources:

TPC-DS

TPC-DS (*Transaction Processing Performance Council - Decision Support*) is a benchmark for evaluating the performance of decision support systems and analytical databases. It simulates real-world big data scenarios typical of the data centres of retail organisations and other large companies.

Test parameters

Benchmark	TPC-DS
Test type	static test (without **ETL**)
Scaling factor	`100, 1000`
Query	random `Q1-Q100`
Parameters	random
Number of threads	`1, 10, 20`
Time	1 hour

Examples of test queries

▼ *See examples of test SQL-queries*

```
-- start Q1
WITH customer_total_return
     AS (SELECT sr_customer_sk     AS ctr_customer_sk,
                sr_store_sk        AS ctr_store_sk,
                Sum(sr_return_amt) AS ctr_total_return
         FROM   s1_store_returns,
                s1_date_dim
         WHERE  sr_returned_date_sk = d_date_sk
                AND d_year = 2001
         GROUP  BY sr_customer_sk,
                   sr_store_sk)
SELECT c_customer_id
FROM   customer_total_return ctr1,
       s1_store,
       s1_customer
WHERE  ctr1.ctr_total_return > (SELECT Avg(ctr_total_return) * 1.2
                                FROM   customer_total_return ctr2
                                WHERE  ctr1.ctr_store_sk = ctr2.ctr_store_sk)
       AND s_store_sk = ctr1.ctr_store_sk
       AND s_state = 'TN'
       AND ctr1.ctr_customer_sk = c_customer_sk
ORDER  BY c_customer_id
LIMIT 100;

-- start Q2
WITH wscs
     AS (SELECT sold_date_sk,
                sales_price
         FROM   (SELECT ws_sold_date_sk    sold_date_sk,
                        ws_ext_sales_price sales_price
                 FROM   s1_web_sales)
         UNION ALL
```

```
         (SELECT cs_sold_date_sk    sold_date_sk,
                 cs_ext_sales_price sales_price
          FROM   s1_catalog_sales)),
     wswscs
     AS (SELECT d_week_seq,
                Sum(CASE
                      WHEN ( d_day_name = 'Sunday' ) THEN sales_price
                      ELSE NULL
                    END) sun_sales,
                Sum(CASE
                      WHEN ( d_day_name = 'Monday' ) THEN sales_price
                      ELSE NULL
                    END) mon_sales,
                Sum(CASE
                      WHEN ( d_day_name = 'Tuesday' ) THEN sales_price
                      ELSE NULL
                    END) tue_sales,
                Sum(CASE
                      WHEN ( d_day_name = 'Wednesday' ) THEN sales_price
                      ELSE NULL
                    END) wed_sales,
                Sum(CASE
                      WHEN ( d_day_name = 'Thursday' ) THEN sales_price
                      ELSE NULL
                    END) thu_sales,
                Sum(CASE
                      WHEN ( d_day_name = 'Friday' ) THEN sales_price
                      ELSE NULL
                    END) fri_sales,
                Sum(CASE
                      WHEN ( d_day_name = 'Saturday' ) THEN sales_price
                      ELSE NULL
                    END) sat_sales
         FROM   wscs,
                s1_date_dim
         WHERE  d_date_sk = sold_date_sk
         GROUP  BY d_week_seq)
SELECT d_week_seq1,
       Round(sun_sales1 / sun_sales2, 2),
       Round(mon_sales1 / mon_sales2, 2),
       Round(tue_sales1 / tue_sales2, 2),
       Round(wed_sales1 / wed_sales2, 2),
       Round(thu_sales1 / thu_sales2, 2),
       Round(fri_sales1 / fri_sales2, 2),
       Round(sat_sales1 / sat_sales2, 2)
FROM   (SELECT wswscs.d_week_seq d_week_seq1,
               sun_sales         sun_sales1,
               mon_sales         mon_sales1,
               tue_sales         tue_sales1,
               wed_sales         wed_sales1,
               thu_sales         thu_sales1,
               fri_sales         fri_sales1,
               sat_sales         sat_sales1
```

```
        FROM   wswscs,
               s1_date_dim
        WHERE  s1_date_dim.d_week_seq = wswscs.d_week_seq
               AND d_year = 1998) y,
       (SELECT wswscs.d_week_seq d_week_seq2,
               sun_sales         sun_sales2,
               mon_sales         mon_sales2,
               tue_sales         tue_sales2,
               wed_sales         wed_sales2,
               thu_sales         thu_sales2,
               fri_sales         fri_sales2,
               sat_sales         sat_sales2
        FROM   wswscs,
               s1_date_dim
        WHERE  s1_date_dim.d_week_seq = wswscs.d_week_seq
               AND d_year = 1998 + 1) z
WHERE  d_week_seq1 = d_week_seq2 - 53
ORDER  BY d_week_seq1;
```

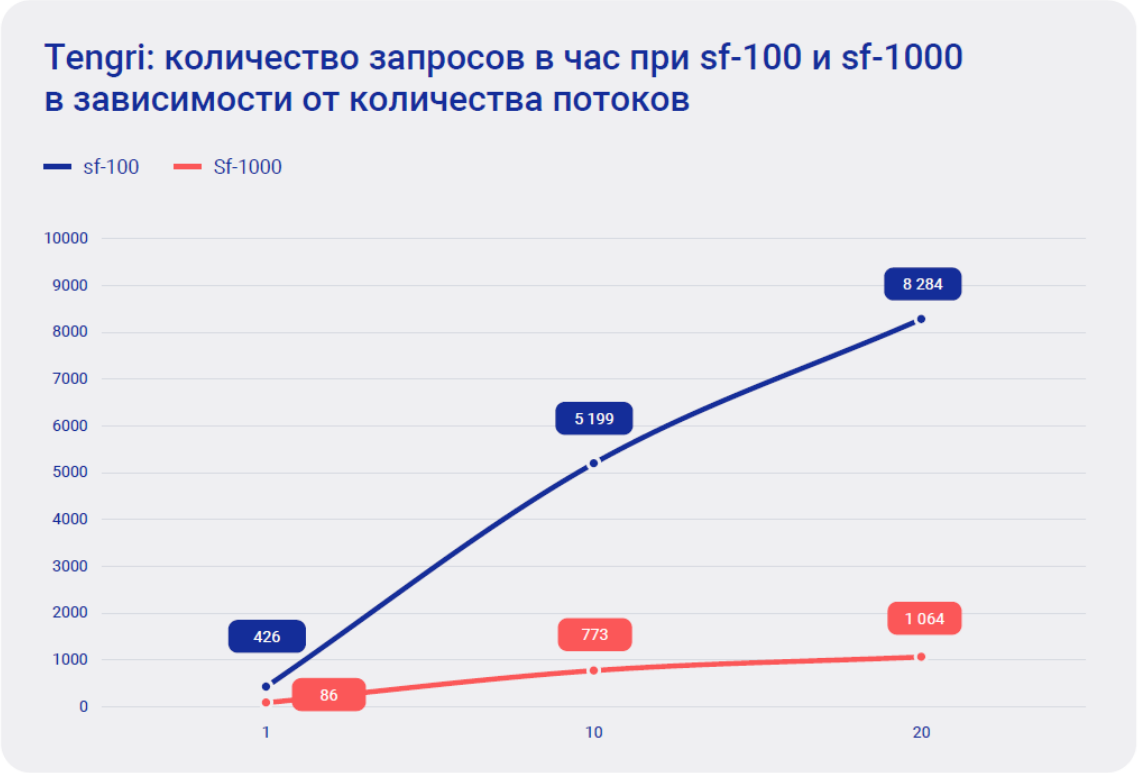
System under test

- Tengri
  - 200 vCPU
  - 400GB RAM

Results

Number of threads	`sf=100`, requests per hour	`sf=1000`, requests per hour
`1`	426	86
`10`	5199	773
`20`	8284	1064

When going from a **Scaling factor** value of `100` to a value of `1000` the operability of Tengri is preserved, the performance decreases only 5-6 times, although the data volume increases by a factor of 10.



[1] The performance data contained on this site was obtained in a strictly controlled Postgres Professional environment while simulating the intended analytical workloads and is for reference only. Results obtained in other environments may vary

Manual

- JSON type
- Types for arrays
- Binary type
- Logical type
- Type for geodata

Computing pools

In Tengri, computation is performed in virtual computing pools.

A compute pool is a lightweight container that is part of a compute pool, contains software code and has dedicated computational resources for computational tasks:

- Execution of SQL-queries
- Executing code on Python
- Running models AI

Operations with computational pools:

- `CREATE WORKER POOL` — Create a worker pool
- `ALTER WORKER POOL` — Changing the attributes of a worker pool
- `DROP WORKER POOL` — Reset a worker pool
- `ALTER WORKER POOL SET DEFAULT` — Set default attributes of the worker pool
- `SHOW WORKER POOLS` — Display a list of all worker pools
- `SHOW WORKER POOL` — Display the current worker pool
- `USE WORKER POOL` — Set the current compute pool
- Granting privileges to computing pools

Create a worker pool

```
CREATE [OR REPLACE] WORKER POOL [IF NOT EXISTS] <pool_name>
    [<worker_pool_attribute> [ ... ]];
```

Creates a new compute pool with the specified name and attributes.

If the `OR REPLACE` modifier is specified, the final action is equivalent to deleting an existing compute pool and creating a new one with the same name.

The optional `IF NOT EXISTS` modifier restricts the query to only those cases in which the specified object does not already exist.

The modifiers are mutually exclusive. Specifying them both will result in an error.

The following compute pool attributes can be specified:

- `WORKER SIZE <size>` — sets the compute size for this compute pool: `XS`, `S`, `M`, `L`, `XL`.
- `MAX WORKER COUNT <count>` — sets the maximum number of calculators.
- `WORKER IDLE TTL <seconds> SECONDS` — sets the maximum idle time of calculators.
- `SCALING STRATEGY <strategy>` — sets the scaling strategy:
  - `STD` — if there are no available calculators, immediately report their absence.
  - `ECO` — if there are no free computers, wait 10 minutes for them to appear.

▼ *See examples*

Create a computing pool with the name `my_worker_pool` and the size of calculators `S`:

```
CREATE WORKER POOL my_worker_pool
    WORKER SIZE S;
```

Create a compute pool with name `my_worker_pool`, compute size `L`, maximum number of compute `40`, maximum compute idle time `300` seconds and with scaling strategy `ECO`:

```
CREATE WORKER POOL my_worker_pool
    WORKER SIZE L
    MAX WORKER COUNT 40
    WORKER IDLE TTL 300 SECONDS
    SCALING STRATEGY ECO;
```

Changing the attributes of a worker pool

```
ALTER WORKER POOL [IF EXISTS] <pool_name> RENAME TO <new_pool_name>;

ALTER WORKER POOL [IF EXISTS] <pool_name> SET <worker_pool_attribute>;
```

Modifies the compute pool with the specified name.

It is not recommended to change the attributes of compute pools running within your workgroup, as this may cause unexpected degradation of computational performance when compute pools are used by multiple users.

Therefore, `ALTER WORKER POOL` expressions should only be used by administrators.

If you as a user need to change the size of the compute pool you are using (e.g., increase its size), it is recommended to select another larger compute pool to work in.

Supported actions:

- `RENAME TO` — renames the compute pool to the specified name `<new_pool_name>`. All attributes and configurations of the compute pool are retained.

- `SET` — updates an attribute of the compute pool. The following attributes can be changed:
  - `WORKER SIZE <size>` — specifies the calculator size (`XS`, `S`, `M`, `L`, `XL`) for this compute pool.
  - `MAX WORKER COUNT <count>` — sets the maximum number of evaluators for this compute pool.
  - `WORKER IDLE TTL <seconds> SECONDS` — sets the compute idle time for this compute pool.
  - `SCALING STRATEGY <strategy>` — sets the scaling strategy:
    - `STD` — if there are no available computors, immediately report their absence.
    - `ECO` — if there are no available computers, wait 10 minutes for them to appear.

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Reset a worker pool

```
DROP WORKER POOL [IF EXISTS] <pool_name>;
```

Deletes the compute pool with the specified name.

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Set default attributes of the worker pool

```
ALTER WORKER POOL SET DEFAULT <worker_pool_attributes>;
```

Sets the default attributes for all new compute pools on the system.

This statement allows you to set system-wide default attributes that will be applied to compute pools when they are created without explicitly specifying these attributes.

The following computing pool attributes can be specified:

- `WORKER SIZE <size>` — specifies the default calculator size (`XS`, `S`, `M`, `L`, `XL`) for this compute pool.
- `MAX WORKER COUNT <count>` — sets the maximum default number of calculators for this compute pool.
- `WORKER IDLE TTL <seconds> SECONDS` — sets the default idle time of calculators for this compute pool.
- `SCALING STRATEGY <strategy>` — sets the default scaling strategy:
  - `STD` — if there are no available computors, immediately report their absence.
  - `ECO` — if there are no free computers, wait 10 minutes for them to appear.

Display a list of all worker pools

```
SHOW WORKER POOLS;
```

Outputs a list of all compute pools in the system.

This statement displays information about all available compute pools, including their names and configurations.

The `MONITOR` privilege is required to see a compute pool in the response.

Display the current worker pool

```
SHOW WORKER POOL;
```

Outputs the compute pool that is currently being used to execute the query.

This statement displays information about the currently active compute pool, including its name and configuration.

Set the current compute pool

```
USE WORKER POOL <pool_name>;
```

Sets the specified compute pool as the current pool for query execution.

This statement changes the compute pool that will be used to execute subsequent requests. subsequent queries. The specified pool must exist, and the user must have the necessary privileges to use it.

The `USAGE` privilege is required to use this operator.

Increasing the size of the computing pool

If you as a user need to change the size of the computing pool you are using (e.g., increase it), it is recommended to select another computing pool from the list of available ones without changing the attributes of the current computing pool.

To do this, do the following.

Output the size of the used computational pool:

```
SHOW WORKER POOL;
```

```
+----------------+------+
| worker_pool    | size |
+----------------+------+
| compute_s      | S    |
+----------------+------+
```

We output the sizes of the compute pools available for use:

```
SHOW WORKER POOLS;
```

```
+----------------+-------+
| name           | value |
+----------------+-------+
| client_compute | S     |
+----------------+-------+
| compute_m      | M     |
+----------------+-------+
| compute_s      | S     |
+----------------+-------+
| compute_l      | L     |
+----------------+-------+
```

Switching to a larger compute pool:

```
USE WORKER POOL compute_m;
```

```
+--------+
| status |
+--------+
| USE    |
+--------+
```

User and rights management

A developed system of access rights differentiation is implemented in Tengri.

The principles are supported:

- DAC — *discretionary access control*
- RBAC — *role-based access control*

Key concepts of a rights differentiation system

Protected Object

An entity to which access (privileges) can be granted.

Role

An entity to which access rights (privileges) can be granted. Roles can be assigned to users or to other roles.

Privilege

A specific level of access to an entity. Assigned to users or roles.

User

An identifier associated with a person or service. A user is an object that can be granted privileges and/or roles.

Operations with objects of the rights differentiation system

User operations

- `CREATE USER` — Create a new user
- `ALTER USER` — Change user attributes
- `DROP USER` — Reset user
- `DESCRIBE USER` — Display information about the user
- `SHOW USERS` — Display a list of all users

Create a new user

```
CREATE [OR REPLACE] USER [IF NOT EXISTS] <user_name>
    [SET DEFAULT WORKER POOL <pool_id>]
    [IDENTIFIED BY (TRUST <ip_address> | PASSWORD <user_password>)];
```

Creates a new user with the specified name.

Modifier `OR REPLACE`

If the optional `OR REPLACE` modifier is specified, this action is equivalent to deleting the current user and creating a new user with the same username. the current user and creating a new user with the same username. The newly created user does not inherit any roles or attributes.

Modifier `IF NOT EXISTS`

If the optional `IF NOT EXISTS` modifier is specified, the query will only be executed if the user named `<user_name>` does not exist, otherwise nothing will happen.

The `OR REPLACE` and `IF NOT EXISTS` modifiers are mutually exclusive. Specifying them both will result in an error.

Parameter `SET DEFAULT WORKER POOL`

The optional parameter `SET DEFAULT WORKER POOL <pool_id>` sets the default compute pool `<pool_id>` for the specified user.

Parameter `IDENTIFIED BY TRUST`

The optional parameter `IDENTIFIED BY TRUST <ip_address>` sets the authentication of the specified user through the IP address `<ip_address>`.

▼ *See examples*

- ```
  CREATE USER ivanov IDENTIFIED BY TRUST '127.0.0.1';
  ```
- ```
  ALTER USER ivanov IDENTIFIED BY TRUST '127.0.0.1';
  ```

Parameter `IDENTIFIED BY PASSWORD`

The optional parameter `IDENTIFIED BY PASSWORD <user_password>` sets the authentication of the specified user through the password `<user_password>` using the `SCRAM-SHA-256` algorithm.

▼ *See examples*

- ```
  CREATE USER ivanov IDENTIFIED BY PASSWORD 'QWERTY123456';
  ```
- ```
  ALTER USER ivanov IDENTIFIED BY PASSWORD 'QWERTY123456';
  ```

Change user attributes

```
ALTER USER [IF EXISTS] <username>
    RENAME TO <new_name>;

ALTER USER [IF EXISTS] <username>
    SET DEFAULT WORKER POOL <pool_id>;

ALTER USER [IF EXISTS] <username>
    IDENTIFIED BY (TRUST <ip_address> | PASSWORD <user_password>);
```

Modifies the user's attributes.

- `RENAME TO <new_name>` is an atomic operation that renames the user to a new name. No other properties of this user (roles granted, privileges, etc.) are changed.
- `SET DEFAULT WORKER POOL <pool_id>` — see Parameter `SET DEFAULT WORKER POOL`.
- `IDENTIFIED BY TRUST <ip_address>` — see Parameter `IDENTIFIED BY TRUST`.
- `IDENTIFIED BY PASSWORD <user_password>` — see Parameter `IDENTIFIED BY PASSWORD`.

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Reset user

```
DROP USER [IF EXISTS] <username>;
```

Completely removes the user from the system.

Nothing is saved, including granted roles and access rights.

It is possible that some objects may lose access rights.

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Display information about the user

```
DESC[RIBE] USER <username>;
```

Outputs all information about the user, including their attributes and default values.

Display a list of all users

```
SHOW USERS;
```

Display a list of all registered users.

Role operations

- `CREATE ROLE` — Create a new role
- `ALTER ROLE` — Change role attributes
- `DROP ROLE` — Reset a role
- `SHOW ROLES` — Display a list of all roles
- `USE ROLE` — Set the active role
- `GRANT` — Grant attributes to roles

Create a new role

```
CREATE [OR REPLACE] ROLE [IF NOT EXISTS] <role_name>;
```

Creates a new role with the specified name.

If the `OR REPLACE` modifier is specified, the final action is equivalent to deleting the existing role and creating a new role with the same name. The newly created role does not inherit any attributes or privileges from the previous role.

The optional `IF NOT EXISTS` modifier restricts the query to only those cases in which the specified object does not already exist.

The modifiers are mutually exclusive. Specifying them both will result in an error.

Change role attributes

```
ALTER ROLE [IF EXISTS] <role_name> RENAME TO <new_role_name>;

ALTER ROLE [IF EXISTS] <role_name> WITH <attribute>;
```

Modifies an existing role.

The action can be either renaming the role or changing its attributes.

Available attributes:

- `SUPERUSER` — grants the role superuser privileges.
- `CREATEROLE` — allows the role to create new roles.

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Reset a role

```
DROP ROLE [IF EXISTS] <role_name>;
```

Removes the role from the system.

The command will fail if the role is still referenced by any objects (for example, it is granted to users or has access rights for some objects).

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Display a list of all roles

```
SHOW [TERSE] ROLES
    [LIKE '<pattern>']
    [STARTS WITH '<pattern>']
    [LIMIT <number> [FROM '<pattern>']];
```

Outputs a list of all roles in the system. The output can be filtered and restricted using various options:

- `TERSE` — displays a simplified output format.
- `LIKE '<pattern>'` — filters roles whose names match the specified pattern.
- `STARTS WITH '<pattern>'` — filters roles whose names begin with the specified pattern.
- `LIMIT <number> [FROM '<pattern>']` — limits the number of roles returned, optionally starting with the specified role name.

Templates `<pattern>` are case-sensitive string literals that may contain wildcard characters.

Set the active role

```
USE ROLE <role_name>;
```

Sets the specified role as the active role for the current session.

If a role is set as active, the user receives all privileges granted to that role for the duration of the session.

To use the specified role, the user must be granted the appropriate privileges.

Grant attributes to roles

```
GRANT <role_attributes> TO ROLE <role_name>;
```

Provides the specified role `<role_name>` with the attributes `<role_attributes>`.

This statement allows the role to have special attributes or to perform operations on other roles according to the privileges granted.

Privilege operations

- `GRANT` — Grant privileges
- `REVOKE` — Revoke privileges

Grant privileges

Scheme Privileges

```
GRANT <schema_privileges> ON SCHEMA <schema_name> TO [ROLE] <role_name>;
```

Grants the role the specified privileges to the specified schema.

`<schema_privileges>` — is a comma-separated list of privileges.

Available privileges on schemas:

- `ADMIN` — allows deletion of the schema and distribution of schema privileges.
- `MONITOR` — allows access to metainformation about the schema.
- `USAGE` — allows access to objects in the scheme.
- `MODIFY` — allows modification of schema properties.
- `CREATE TABLE` — allows you to create tables in the schema.
- `CREATE VIEW` — allows to create views in the schema.

The `ROLE` keyword before the target role name is optional.

▼ *See example*

Let's grant the `junior_admin` role the `MONITOR` and `MODIFY` privileges on the `main_schema` schema.

```
GRANT MONITOR, MODIFY
    ON SCHEMA main_schema
    TO ROLE junior_admin;
```

Table and view privileges

Direct syntax:

```
GRANT <table_privileges>
    ON (TABLE | VIEW) <table_name>
    TO [ROLE] <role_name>;
```

Temporal syntax:

```
GRANT <table_privileges>
    ON ALL [EXISTING] [[AND] FUTURE] (TABLES | VIEWS | TABLES AND VIEWS)
    IN [SCHEMA]<schema_name>
    TO [ROLE] <role_name>;
```

Grants roles the specified privileges on a table or view within the specified schema.

`<table_privileges>` — is a comma-separated list of privileges.

Available privileges on tables and views:

- `SELECT` — allows you to read data from a table.
- `INSERT` — allows you to add new rows to the table.
- `UPDATE` — allows you to modify existing rows in the table.
- `DELETE` — allows to delete rows from the table.
- `MODIFY` — allows to change the table structure.
- `OWNERSHIP` — allows to change the owner of the table.

To grant privilege to all existing (or those to be created in the future) objects within a schema, the temporal syntax is used.

The `EXISTING` modifier restricts privilege granting to existing objects only.

The `FUTURE` modifier restricts privilege granting to objects that will be created in the future.

If no modifier is specified, all objects, both existing and future, are affected.

The `ROLE` keyword before the target role name is optional.

▼ *See example*

Grant the `junior_admin` role the `SELECT` and `INSERT` privileges on all existing tables and views within the `main_schema` schema.

```
GRANT SELECT, INSERT
    ON EXISTING TABLES AND VIEWS
    IN SCHEMA main_schema
    TO ROLE junior_admin;
```

Catalogue privileges

```
GRANT <catalog_privileges> ON CATALOG TO [ROLE] <role_name>;
```

Grants the role the specified directory privileges.

`<catalog_privileges>` — is a comma-separated list of privileges.

Available privileges per catalogue:

- `ADMIN` — any action on the catalogue.
- `CREATE USER` — creating users.
- `CREATE WORKER POOL` — creating computing pools.
- `CREATE SCHEMA` — creating schemas.

The `EXISTING` modifier restricts privilege granting to existing objects only.

The `FUTURE` modifier restricts privilege granting to objects that will be created in the future.

The `ROLE` keyword before the target role name is optional.

Compute pool privileges

```
GRANT <worker_pool_privileges> ON WORKER POOL <worker_pool_name> TO [ROLE] <role_name>;
```

Grants the role the specified compute pool privileges.

`<worker_pool_privileges>` — is a comma-separated list of privileges.

Available privileges on compute pools:

- `ALL` — all possible actions on the compute pool.
- `ADMIN` — modify the compute pool (privileges to execute the command `ALTER WORKER POOL`).
- `USAGE` — using the computing pool (rights to execute the command `USE WORKER POOL`).
- `MONITOR` — monitoring of the computing pool.

▼ *See example*

Grant the `junior_analyst` role the `USAGE` and `MONITOR` privileges on the `compute_xl` compute pool.

```
GRANT USAGE, MONITOR
    ON WORKER POOL compute_xl
    TO junior_analyst;
```

Revoke privileges

```
REVOKE <schema_privileges>
    ON SCHEMA <schema_name> FROM ROLE <role_name>;

REVOKE <table_privileges>
    ON (TABLE | VIEW) <table_name> [IN SCHEMA <schema_name>] FROM ROLE <role_name>;
```

```
REVOKE <catalog_privileges>
    ON CATALOG FROM ROLE <role_name>;

REVOKE <worker_pool_privileges>
    ON WORKER POOL <worker_pool_name> FROM ROLE <role_name>;
```

Revokes the specified privileges on the specified object from the role.

`<schema_privileges>`, `<table_privileges>`, `<catalog_privileges>`, and `<worker_pool_privileges>` — these are comma-separated privilege lists. The specific privileges depend on the target object.

The `EXISTING` modifier restricts privilege revocation to existing objects only.

The `FUTURE` modifier restricts privilege revocation to objects that will be created in the future.

▼ *See example*

Let's revoke the `Junior_admin` role's `SELECT` and `INSERT` privileges on all existing tables and views within the `main_schema` schema.

```
REVOKE SELECT, INSERT
    ON EXISTING TABLES AND VIEWS
    IN SCHEMA main_schema
    FROM ROLE junior_admin;
```

Data Description (DDL)

The DDL commands are used to describe data — to create, manipulate, and delete tables, schemas, and views.

- Operations with tables
- Operations with schemes
- Operations with views

Operations with tables

- `CREATE TABLE` — Create a new table
- `INSERT` — Add data to a table
- `DROP TABLE` — Deleting a table
- `SHOW TABLES` — Display a list of all tables
- `DESCRIBE TABLE` — Display information about the table

Create a new table

```
CREATE [OR REPLACE] TABLE [IF NOT EXISTS] [<table_schema>.]<table_name>
```

```
    (<column_name> <column_type>
        [NOT NULL] [DEFAULT <default_expr>]
        [GENERATED BY DEFAULT AS IDENTITY] [GENERATED ALWAYS AS IDENTITY],
    ...
    )
    [WITH (<table_param>, ... )]
```

Creates a new table with the specified name and specified columns.

```
CREATE [OR REPLACE] TABLE [IF NOT EXISTS] [<table_schema>.]<table_name> AS
    <select_expr>
    [WITH (<table_param>, ... )]
```

Creates a new table with the specified name based on the result of a `SELECT` query.

Parameters

- `<table_name>` — name of the table to be created

- `<table_schema>` — schema of the table to be created

- `<column_name>` — column name of the table to be created

- `<column_type>` — data type of the column of the table to be created (see section Data types)

- `NOT NULL` — this column does not accept the value `NULL`

- `DEFAULT <default_expr>` — default constant or constant expression

- `GENERATED BY DEFAULT AS IDENTITY` — this column is an identity column, that is, the default values for it are automatically generated from an implicit sequence. If an `INSERT` command is executed for a table with an identity column and a value for that column is not explicitly specified, the value generated by the implicit sequence is inserted into that column.

  ▼ *See example*

  Create a table `my_table` with two columns. Set the `id` column as an identity column (autoincrement). Let's insert numbers from 5 to 10 into the other column using the function `generate_series`. As a result, the values will be automatically inserted into the identification column.

  ```
  CREATE TABLE my_table (
  ```

```
    id INT GENERATED BY DEFAULT AS IDENTITY,
    numbers INT,
    );

INSERT INTO my_table (numbers)
    SELECT unnest(generate_series(5,10));

SELECT * FROM my_table;
```

```
+----+---------+
| id | numbers |
+----+---------+
| 0  | 5       |
+----+---------+
| 1  | 6       |
+----+---------+
| 2  | 7       |
+----+---------+
| 3  | 8       |
+----+---------+
| 4  | 9       |
+----+---------+
| 5  | 10      |
+----+---------+
```

- `<select_expr>` — `SELECT` expression, the result of execution of which will be written to the table being created

---

- `<table_param>` — parameters of the table to be created

  ```
  table_param ::= [<name> = <value>]
  ```

  Possible meanings:

  - `snapshot_ttl = <duration>` — depth of snapshot (table version) storage.
    Default: 7 days, but no more than 1000 snapshots.
    For example: `'1 week'`, `'2 days'`, `'4 days 3 hours 5 minutes 30 seconds'`.
  - `order_by = <column_name>` — a sort column used to compact the table data. The data in the table is stored with arbitrary sorting, ordering is not guaranteed.

If the `OR REPLACE` modifier is specified, the final action is equivalent to deleting the existing table and creating a new one with the same name.

The optional `IF NOT EXISTS` modifier restricts the query to only those cases in which the specified object does not already exist.

The modifiers are mutually exclusive. Specifying them both will result in an error.

Add data to a table

```
INSERT INTO <target_table> [ ( <target_col_name> [ , ... ] ) ]
       {
         VALUES ( { <value> | DEFAULT | NULL } [ , ... ] ) [ , ( ... ) ]  |
         <query>
       }
```

Updates the table by inserting one or more rows into the table. The values inserted into each column of the table can be specified explicitly or obtained from a nested query.

Parameters

- `<target_table>` — the name of the target table into which the rows will be inserted

---

- `VALUES ( value | DEFAULT | NULL [ , ⋯ ] ) [ , ( ⋯ ) ]` — specifies one or more values to insert into the appropriate columns of the target table.
  - `value` — an explicitly specified value; can be a literal or an expression.
  - `DEFAULT` — the default value for the corresponding column of the target table.
  - `NULL` — an empty value.

  Values are separated by commas.

  You can insert multiple strings by specifying additional sets of values in the expression.

- `query` — a query that returns values to insert into the appropriate columns. This allows you to insert rows into the target table from one or more source tables.

Example of inserting explicitly specified values

Insert values for the `country` and `capital` columns into the `capitals` table:

```
CREATE TABLE capitals (country VARCHAR, capital VARCHAR);
INSERT INTO capitals VALUES
    ('Russia', 'Moscow'),
    ('Italy', 'Rome'),
    ('Spain', 'Madrid'),
    ('France', 'Paris');

SELECT * FROM capitals;
```

```
+---------+---------+
| country | capital |
+---------+---------+
| France  | Paris   |
+---------+---------+
| Italy   | Rome    |
+---------+---------+
| Russia  | Moscow  |
+---------+---------+
| Spain   | Madrid  |
+---------+---------+
```

Example of inserting results of a nested query

Now let's create another table `capitals_m` and insert the rows from `capitals` that will be the result of the nested `SELECT` query. Let's insert such rows from `capitals` where the value of `capital` contains `M`.

```
CREATE TABLE capitals_m (country VARCHAR, capital VARCHAR);
INSERT INTO capitals_m (country, capital)
    SELECT * FROM capitals
        WHERE capital LIKE '%M%';

SELECT * FROM capitals_m;
```

```
+---------+---------+
| country | capital |
+---------+---------+
| Russia | Moscow |
+---------+---------+
| Spain | Madrid |
+---------+---------+
```

Deleting a table

```
DROP TABLE [IF EXISTS] <table_name>;
```

Deletes the table with the specified name.

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Display a list of all tables

```
SHOW TABLES;
```

Outputs a list of all existing tables available to the user.

Display information about the table

```
DESC[RIBE] TABLE <table_name>;
```

Displays all information about the table — column names, their data types, and other properties.

Operations with schemes

- `CREATE SCHEMA` — Create a new schema
- `ALTER SCHEMA` — Change schema attributes
- `DROP SCHEMA` — Deleting a schema

Create a new schema

```
CREATE [OR REPLACE] SCHEMA [IF NOT EXISTS] <schema_name>
```

Creates a new schema with the specified name.

If the `OR REPLACE` modifier is specified, the final action is equivalent to deleting the existing schema with all objects in it and creating a new schema with the same name.

The optional `IF NOT EXISTS` modifier restricts the query to only those cases in which the specified object does not already exist.

The modifiers are mutually exclusive. Specifying them both will result in an error.

Change schema attributes

```
ALTER SCHEMA [IF EXISTS] <schema_name> RENAME TO <new_name>;
```

Modifies the pattern with the specified action.

Currently, only one action is available for changing schemes:

- The `RENAME TO` action renames the schema to the specified name `<new_name>`. All attributes and permissions are retained.

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Deleting a schema

```
DROP SCHEMA [IF EXISTS] <schema_name>;
```

Deletes the schema with the specified name.

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Operations with views

- `CREATE VIEW` — Create a new view
- `DROP VIEW` — Deleting a view

Create a new view

```
CREATE [OR REPLACE] VIEW [IF NOT EXISTS] [<view_schema>.]<view_name> AS
    <select_expr>
```

Creates a new view with the specified name based on the result of the specified `SELECT` query.

Parameters

- `<view_name>` — name of the view to be created

- `<view_schema>` — schema of the view to be created

- `<select_expr>` — `SELECT` expression, based on the result of execution of which the view will be created

If the `OR REPLACE` modifier is specified, the final action is equivalent to deleting the existing view and creating a new one with the same name.

The optional `IF NOT EXISTS` modifier restricts the query to only those cases in which the specified object does not already exist.

The modifiers are mutually exclusive. Specifying them both will result in an error.

Deleting a view

```
DROP VIEW [IF EXISTS] <view_name>;
```

Removes the view from the system.

The optional `IF EXISTS` modifier restricts the query to only those cases in which the specified object exists.

Data query syntax

SQL statements

Tengri supports data queries using the standard keywords SQL and the following basic syntax:

```
[ WITH ... ]
SELECT
   ...
[ FROM ...
   [ JOIN ... ]
[ WHERE ... ]
[ GROUP BY ...
   [ HAVING ... ] ]
[ QUALIFY ... ]
[ ORDER BY ... ]
[ LIMIT ... ]
```

Detailed descriptions for the supported keywords are provided in the sections:

- `SELECT`
- `WITH`
- `FROM`
- `JOIN`
- `UNION`
- `WHERE`
- `GROUP BY`
- `HAVING`
- `QUALIFY`
- `ORDER BY`
- `LIMIT`
- `OFFSET`
- `LIKE`
- `ILIKE`
- `SIMILAR TO`
- `AS`

Functions

- Aggregate functions
- Numerical functions
- Text functions
- Functions for regular expressions
- Functions for date and time
- Functions for JSON
- Functions for binary data
- Functions for geodata
- Window functions
- Utilities
- Python module `tngri` functions

SELECT

The `SELECT` clause specifies the list of columns to be returned by the query. Although it comes first in the query, logically the expressions in this expression are executed last. A `SELECT` clause can contain arbitrary expressions that transform output, as well as aggregate and window functions.

Syntax

Select all columns

```
[ ... ]
SELECT [ { ALL | DISTINCT } ]
       [ TOP <n> ]
       [{<object_name>|<alias>}.]*

       [ ILIKE '<pattern>' ]

       [ EXCLUDE
         {
           <col_name> | ( <col_name>, <col_name>, ... )
         }
       ]

       [ REPLACE
         {
           ( <expr> AS <col_name> [ , <expr> AS <col_name>, ... ] )
         }
       ]

       [ RENAME
```

```
        {
          <col_name> AS <col_alias>
          | ( <col_name> AS <col_alias>, <col_name> AS <col_alias>, ... )
        }
      ]
```

The following keyword combinations can be specified after `SELECT *`. The keywords must be in the order shown below:

```
SELECT * ILIKE ... REPLACE ...
```

```
SELECT * ILIKE ... RENAME ...
```

```
SELECT * ILIKE ... REPLACE ... RENAME ...
```

```
SELECT * EXCLUDE ... REPLACE ...
```

```
SELECT * EXCLUDE ... RENAME ...
```

```
SELECT * EXCLUDE ... REPLACE ... RENAME ...
```

```
SELECT * REPLACE ... RENAME ...
```

Selecting specific columns

```
[ ... ]
SELECT [ { ALL | DISTINCT } ]
       [ TOP <n> ]
       {
         [{<object_name>|<alias>}.]<col_name>
         | [{<object_name>|<alias>}.]$<col_position>
         | <expr>
       }
       [ [ AS ] <col_alias> ]
       [ , ... ]
[ ... ]
```

End comma is supported in the column list. For example, the following `SELECT` clause is supported:

```
SELECT emp_id,
```

```
    name,
    department,
FROM employees;
```

Parameters

- `ALL | DISTINCT`
  Specifies whether to perform duplicate deletion in the result set:
  - `ALL` includes all values in the result set.
  - `DISTINCT` removes duplicates from the result set.

  Default: `ALL`
  For a detailed description, see `DISTINCT` clause.

- `<object_name>` or `<alias>`
  Specifies the object identifier or object alias (as defined in the `FROM` clause).

- `*` (asterisk)
  The asterisk is an abbreviation indicating that the output must include all columns of the specified object, or all columns of all objects if `*` is not followed by an object name or alias.
  For a detailed description, see Expression with asterisk `*`.

- `<col_name>`
  Specifies the column identifier (as defined in the `FROM` clause).

- `$<col_position>`
  Indicates the position of the column (starting from 1) as defined in the `FROM` clause. If the column is referenced from the table, this number cannot exceed the maximum number of columns in the table.

- `<expr>`
  Specifies an expression (such as a mathematical expression) that evaluates to a specific value for any given row.

- `[ AS ] <col_alias>`
  Specifies the column alias assigned to the resulting expression. It is used as the display name in the top-level `SELECT` list and as the column name in the embedded view.

Do not assign such an alias to a column that will match the name of another column referenced in the query. For example, if you select columns named `prod_id` and `product_id`, do not assign the alias `product_id` to the `prod_id` column.

See also section `AS`

- `ILIKE <pattern>`
  Specifies that only columns matching the specified pattern should be included in the query results. The following expressions can be used in the pattern SQL:
  - Underscore character `_` to match any single character.
  - The percentage character `%` to match any sequence of zero or more characters.

- `EXCLUDE <col_name>` or
  `EXCLUDE (<col_name_1>, <col_name_2>, ⋯ )`
  Specifies the columns to be excluded from the query results.

- `REPLACE (<expr> AS <col_name> [ , <expr> AS <col_name>, ⋯] )`
  Replaces the value of col_name with the value of the evaluated expression expr. Replaces the value of the specified column with the result of applying the `<expr>` clause to its original value.

- `RENAME <col_name> AS <col_alias>` or
  `RENAME (<col_name_1> AS <col_alias_1>, <col_name_2> AS <col_alias_2>, ⋯)`
  Specifies the column aliases to be used in the query results.

- `TOP <n>`
  Specifies the maximum number of rows that will be the result of the query.

Examples

- Select all columns from the table named `my_table`:

```
SELECT * FROM my_table;
```

- Perform arithmetic operations on the columns of the table and specify an alias:

```
SELECT column_1 + column_2 AS sum, sqrt(column_1) AS sq_root FROM my_table;
```

- Use prefix aliases to get the same result:

```
SELECT
    sum: column_1 + column_2,
    sq_root: sqrt(column_1)
FROM my_table;
```

- Select all unique names from the `employees` table:

```
SELECT DISTINCT name FROM employees;
```

- Output the total number of rows in the `employees` table:

```
SELECT count(*) FROM employees;
```

- Select all columns except the `name` column from the `employees` table:

```
SELECT * EXCLUDE (name) FROM employees;
```

- Select all columns from the `employees` table, but replace `name` with the result of applying the `lower(name)` function:

```
SELECT * REPLACE (lower(name) AS name) FROM employees;
```

- Select all columns from the table that match the given regular expression:

```
SELECT COLUMNS('number\d+') FROM employees;
```

- Calculate the function over all given columns of the table:

```
SELECT min(COLUMNS(*)) FROM employees;
```

- Use double quotes (") to select columns with spaces or special characters:

```
SELECT "Фамилия Имя Отчество" FROM employees;
```

List of `SELECT` columns

A `SELECT` clause contains a list of expressions that define the result of the query. The `SELECT` list can refer to any columns in a `FROM` clause and combine them using expressions. Because the result of a SQL-query is a table, each expression in a `SELECT` clause also has a name. Expressions can be explicitly named using the `AS` operator (e.g., `expr AS name`). If the user does not specify a name, expressions are named automatically by the system.

Column names are case insensitive if they are specified without inverted commas.

Expression with asterisk `*`

An asterisk expression `()` is a special expression that expands to multiple expressions based on the

`contents of the FROM clause. In the simplest case,` is expanded to all expressions in the `FROM` clause.

- Select all columns from the table named `my_table`:

```
SELECT * FROM my_table;
```

`DISTINCT` clause

The `DISTINCT` clause can be used to get only unique rows in the result - so all duplicates will be filtered out.

- Select all unique names from the `employees` table:

```
SELECT DISTINCT name FROM employees;
```

Queries starting with `SELECT DISTINCT` perform deduplication, which is an expensive operation. Therefore, use `DISTINCT` only when necessary.

`DISTINCT ON` clause

The `DISTINCT ON` clause returns only one row for each unique value in the expression set, as defined in the `ON` clause. If the `ORDER BY` condition is present, the first row that occurs according to the `ORDER BY` condition is returned. If the `ORDER BY` condition is not present, the first row encountered is undefined and can be any row in the table.

Select the employee with the highest salary for each department:

```
SELECT DISTINCT ON(department) name, salary
    FROM employees
    ORDER BY salary DESC;
```

When querying large datasets, using `DISTINCT` for all columns can be a costly operation. Therefore, consider using `DISTINCT ON` for a column (or set of columns) that guarantees a sufficient degree of uniqueness in the results. For example, using `DISTINCT ON` for the key column(s) of a table guarantees complete uniqueness.

Aggregate functions

Aggregate functions are functions that combine values from multiple rows into one. When aggregate functions are present in a `SELECT` statement, the query becomes an aggregate query. In an aggregate query, **all** expressions must be either part of an aggregate function or part of a group (as specified in a `GROUP BY` clause).

- We get the total number of rows in the employee table:

```
SELECT count(*) FROM employees;
```

- Get the total number of rows in the table of employees grouped by department:

```
SELECT department, count(*)
    FROM employees
    GROUP BY department;
```

For a detailed description, see Aggregate functions.

Window functions

Window functions are functions that perform calculations for a set of strings that are related in some way to the current string. A call to a window function always contains an `OVER` clause following the window function name and arguments. The `OVER` clause specifies exactly how the query strings are to be split for processing by the window function.

- We get a `row_number` column containing incremental identifiers for each row of the salary table:

```
SELECT row_number() OVER ()
    FROM salaries;
```

- Compute the difference between the current amount and the previous amount in descending order of time:

```
SELECT amount - lag(amount) OVER (ORDER BY time)
    FROM salaries;
```

For a detailed description, see Window functions.

`WITH`

The `WITH` clause allows you to specify generic table expressions (CTE). Regular (non-recursive) common table expressions are essentially views that are limited to the scope of a particular query. CTEs can reference each other and be nested. Recursive CTEs can refer to themselves.

Syntax

Subquery

```
[ WITH
       <cte_name1> [ ( <cte_column_list> ) ] AS ( SELECT ...  )
   [ , <cte_name2> [ ( <cte_column_list> ) ] AS ( SELECT ...  ) ]
   [ , <cte_nameN> [ ( <cte_column_list> ) ] AS ( SELECT ...  ) ]
]
SELECT ...
```

Recursive CTE:

```
[ WITH [ RECURSIVE ]
    <cte_name1> ( <cte_column_list> ) AS ( anchorClause UNION ALL recursiveClause)
 [, <cte_name2> ( <cte_column_list> ) AS ( anchorClause UNION ALL recursiveClause)]
 [, <cte_nameN> ( <cte_column_list> ) AS ( anchorClause UNION ALL recursiveClause)]
]
SELECT ...
```

Where:

```
anchorClause ::=
    SELECT <anchor_column_list> FROM ...

recursiveClause ::=
    SELECT <recursive_column_list> FROM ... [ JOIN ... ]
```

Parameters

- `<cte_name1>` , `<cte_nameN>`
  CTE Name.

- `<cte_column_list>`
  Column names in the CTE (common table expression).

- `<anchor_column_list>`
  The columns used in the anchor expression for the recursive CTE. The columns in this list must match the columns defined in `<cte_column_list>`.

- `<recursive_column_list>`
  The columns used in the recursive expression for the recursive CTE. The columns in this list must match the columns defined in `<cte_column_list>`.

Examples of basic CTEs

Let's create a CTE named `cte` and use it in a basic query:

```
WITH cte AS (SELECT 33 AS amount)
    SELECT * FROM cte;
```

```
+--------+
| amount |
```

```
+--------+
| 33     |
+--------+
```

Let's create two CTEs `cte1` and `cte2`, where the second CTE refers to the first CTE:

```
WITH
    cte1 AS (SELECT 33 AS i),
    cte2 AS (SELECT i * 2 AS amount_doubled FROM cte1)
SELECT * FROM cte2;
```

```
+----------------+
| amount_doubled |
+----------------+
| 66             |
+----------------+
```

You can specify names for the CTE columns:

```
WITH my_cte(amount) AS (SELECT 33 AS i)
SELECT * FROM my_cte;
```

```
+--------+
| amount |
+--------+
| 33     |
+--------+
```

FROM

The `FROM` clause specifies the source of the data that the rest of the query should work with. Logically, the `FROM` clause is the place where the execution of the query begins.

A `FROM` clause can contain a single table, a combination of multiple tables joined using the `JOIN` clause, or another `SELECT` query in a subquery node.

Tengri also has an optional `FROM`-first syntax that allows you to execute a query without a `SELECT` statement.

Syntax

```
SELECT ...
FROM objectReference [ JOIN objectReference [ ... ] ]
[ ... ]
```

Where:

```
objectReference ::=
   {
      [<namespace>.]<object_name>
           [ AT | BEFORE ( <object_state> ) ]
           [ CHANGES ( <change_tracking_type> ) ]
           [ MATCH_RECOGNIZE ]
           [ PIVOT | UNPIVOT ]
           [ [ AS ] <alias_name> ]
     | <table_function>
           [ PIVOT | UNPIVOT ]
           [ [ AS ] <alias_name> ]
     | ( VALUES (...) )
     | ( <subquery> )
           [ [ AS ] <alias_name> ]
     | DIRECTORY( @<stage_name> )
   }
```

Parameters

- `[<namespace>.]<object_name>`
  Specifies the name of the object (table or view) being queried.

- `<table_function>`
  Specifies the system table function, UDF table function, or class method to call in the `FROM` clause.

- `VALUES`
  The `VALUES` clause may contain literal values or expressions to be used in the `FROM` clause. This expression may also contain table and column aliases (not shown in the diagram above).

- `<subquery>`
  Subquery in the `FROM` clause.

- `DIRECTORY( @stage_name )`
  Specifies the name of the stage that includes the directory table.

- `[ AS ] <alias_name>`
  Indicates the name given for the object to which it refers. Can be used with any other subqueries in a `FROM` clause. The `AS` operator may be omitted.

- `JOIN`
  Indicates the execution of a join between two (or more) tables (or views or table functions). The join can

be internal external or of another type. The join can use the `JOIN` keyword or an alternative supported join syntax.
For a detailed description, see `JOIN`.

Examples

- Select all columns from the table named `my_table`:

```
SELECT *
FROM my_table;
```

- Select all columns from the table using `FROM`-first syntax:

```
FROM my_table
SELECT *;
```

- Select all columns using `FROM`-first syntax and omitting the `SELECT` clause:

```
FROM my_table;
```

- Select all columns from a table named `my_table` through the alias `mt`:

```
SELECT mt.*
FROM my_table mt;
```

- Using the prefix alias:

```
SELECT mt.*
FROM mt: my_table;
```

- Select all columns from the `my_table` table in the `my_schema` schema:

```
SELECT *
FROM my_schema.my_table;
```

- Select column `i` from table function `range`, where the first column of the range function is renamed to `i`:

```
SELECT t.i
FROM range(1000) AS t(i);
```

- Select all columns from the subquery:

```
SELECT *
```

```
FROM (SELECT * FROM my_table);
```

- Merge the two tables:

```
SELECT *
FROM my_table
JOIN other_table
    ON my_table.key = other_table.key;
```

For a detailed description, see `JOIN`.

Syntax `FROM`-first

Tengri supports the `FROM`-first syntax, i.e., it allows you to place the `FROM` clause before the `SELECT` clause or omit the `SELECT` clause entirely. Let's demonstrate it with an example:

```
CREATE TABLE capitals (country VARCHAR, capital VARCHAR);
INSERT INTO capitals VALUES
    ('Russia', 'Moscow'),
    ('Italy', 'Rome'),
    ('Spain', 'Madrid'),
    ('France', 'Paris');
```

Syntax `FROM`-first with a `SELECT` clause

The following expression demonstrates the use of the `FROM`-first syntax:

```
FROM capitals
SELECT *;
```

```
+---------+---------+
| country | capital |
+---------+---------+
| France  | Paris   |
+---------+---------+
| Italy   | Rome    |
+---------+---------+
| Russia  | Moscow  |
+---------+---------+
| Spain   | Madrid  |
+---------+---------+
```

It's equivalent to:

```
SELECT *
```

```
FROM capitals;
```

```
+---------+---------+
| country | capital |
+---------+---------+
| France  | Paris   |
+---------+---------+
| Italy   | Rome    |
+---------+---------+
| Russia  | Moscow  |
+---------+---------+
| Spain   | Madrid  |
+---------+---------+
```

Syntax `FROM`-first without a `SELECT` clause

The following expression demonstrates the optional nature of the `SELECT` clause:

```
FROM capitals;
```

```
+---------+---------+
| country | capital |
+---------+---------+
| France  | Paris   |
+---------+---------+
| Italy   | Rome    |
+---------+---------+
| Russia  | Moscow  |
+---------+---------+
| Spain   | Madrid  |
+---------+---------+
```

This is also equivalent to a full enquiry:

```
SELECT *
FROM capitals;
```

```
+---------+---------+
| country | capital |
+---------+---------+
| France  | Paris   |
+---------+---------+
| Italy   | Rome    |
+---------+---------+
| Russia  | Moscow  |
+---------+---------+
```

```
| Spain   | Madrid  |
+---------+---------+
```

`JOIN`

Join is a fundamental relational operation used to horizontally join two tables or relations. Relationships are called the left and right sides of the join depending on how they are written in the `JOIN` clause. Each row of the result contains columns from both relations.

The join uses a rule to match pairs of rows from each relationship. Often this is a predicate, but other rules may be specified.

Syntax

The following syntax variants may be used:

```
SELECT ...
FROM <object_ref1> [
                     {
                       INNER
                       | { LEFT | RIGHT | FULL } [ OUTER ]
                     }
                   ]
                   JOIN <object_ref2>
  [ ON <condition> ]
[ ... ]
```

```
SELECT *
FROM <object_ref1> [
                     {
                       INNER
                       | { LEFT | RIGHT | FULL } [ OUTER ]
                     }
                   ]
                   JOIN <object_ref2>
  [ USING( <column_list> ) ]
[ ... ]
```

```
SELECT ...
FROM <object_ref1> [
                     {
                       | NATURAL [ { LEFT | RIGHT | FULL } [ OUTER ] ]
                       | CROSS
                     }
                   ]
                   JOIN <object_ref2>
[ ... ]
```

Parameters

- `<object_ref1>` and `<object_ref2>` +. Each `<object_ref>` represents a table or table-like data source.

- `JOIN`
  A `JOIN` keyword indicating that tables should be joined. `JOIN` is combined with other keywords (such as `INNER` or `OUTER`) to specify the type of join.

- `ON <condition>`
  A boolean expression that specifies the conditions of the connection.
  For a detailed description, see Conditional connection.

- `USING <column_list>`
  A list of columns by which the join is performed. The columns must be of the same name in the tables being joined.
  For a detailed description, see Conditional connection.

`OUTER JOIN` — external connection

Strings that have no matches can be returned if an `OUTER` join is specified. External joins can have one of the types:

- `LEFT` (All rows from the left relationship occur at least once)
- `RIGHT` (All rows from the right relationship occur at least once)
- `FULL` (All strings from both relations occur at least once)

A connection that is not outer (`OUTER`) is inner (`INNER`) — only those strings that satisfy the condition are returned.

If an unpaired row is returned, the attributes of the other table are set to `NULL`.

`CROSS JOIN` — cross-join (Cartesian product)

The simplest type of join is the `CROSS JOIN`. There are no conditions for this join type and it simply returns all possible pairs.

Let's return all pairs of strings:

```
SELECT a.*, b.*
FROM a
CROSS JOIN b;
```

This is equivalent to a query that simply omits the `JOIN` condition:

```
SELECT a.*, b.*
FROM a, b;
```

Conditional connection

Most conjunctions are specified by a predicate that connects attributes on one side to attributes on the other side. Conditions can be explicitly specified using the `ON` and `WHERE` operators.

Let's show the use of conditional joins on the example of two tables — `capitals` with countries and their capitals and `population` with countries and their population (in millions).

```
CREATE TABLE capitals (cap_country VARCHAR, capital VARCHAR);
CREATE TABLE population (pop_country VARCHAR, population_mil BIGINT);
INSERT INTO capitals VALUES
    ('Russia', 'Moscow'),
    ('Italy', 'Rome'),
    ('Spain', 'Madrid'),
    ('France', 'Paris');
INSERT INTO population VALUES
    ('Russia', 143),
    ('Spain', 48),
    ('Brazil', 211);
```

Let's output the countries with their capitals and populations using `JOIN` with the `ON` operator:

```
SELECT *
FROM capitals
JOIN population ON (cap_country = pop_country);
```

```
+-------------+---------+-------------+----------------+
| cap_country | capital | pop_country | population_mil |
+-------------+---------+-------------+----------------+
| Russia      | Moscow  | Russia      | 143            |
+-------------+---------+-------------+----------------+
| Spain       | Madrid  | Spain       | 48             |
+-------------+---------+-------------+----------------+
```

Now let's output the same data using `JOIN` with the `WHERE` operator:

```
SELECT t1.*, t2.*
FROM capitals t1, population t2
WHERE t1.cap_country = t2.pop_country
```

```
+-------------+---------+-------------+----------------+
| cap_country | capital | pop_country | population_mil |
```

```
+------------+---------+------------+----------------+
| Russia     | Moscow  | Russia     | 143            |
+------------+---------+------------+----------------+
| Spain      | Madrid  | Spain      | 48             |
+------------+---------+------------+----------------+
```

Not only equality but also other predicates can be used with the `WHERE` operator.

If the names of columns in two tables are the same and the values in them must be equal, you can use a simpler syntax `USING`.

Let's define the same tables, but with the same name columns `country`:

```
CREATE TABLE capitals (country VARCHAR, capital VARCHAR);
CREATE TABLE population (country VARCHAR, population_mil BIGINT);
INSERT INTO capitals VALUES
    ('Russia', 'Moscow'),
    ('Italy', 'Rome'),
    ('Spain', 'Madrid'),
    ('France', 'Paris');
INSERT INTO population VALUES
    ('Russia', 143),
    ('Spain', 48),
    ('Brazil', 211);
```

Now output the countries with their capital and population through a simpler query with `USING`:

```
SELECT *
FROM capitals
JOIN population USING (country);
```

```
+---------+---------+----------------+
| country | capital | population_mil |
+---------+---------+----------------+
| Russia  | Moscow  | 143            |
+---------+---------+----------------+
| Spain   | Madrid  | 48             |
+---------+---------+----------------+
```

`NATURAL JOIN` — natural connection

Natural joins join two tables based on columns with the same names.

Let's show the use of this type of join on the example of the same pair of tables with the same name columns `country`.

To join tables based on their common column, execute the command:

```
SELECT *
FROM capitals
NATURAL JOIN population;
```

```
+---------+---------+---------+----------------+
| country | capital | country | population_mil |
+---------+---------+---------+----------------+
| Russia  | Moscow  | Russia  | 143            |
+---------+---------+---------+----------------+
| Spain   | Madrid  | Spain   | 48             |
+---------+---------+---------+----------------+
```

Note that only rows that had the same `country` attribute in both tables were included in the result.

We can perform a similar query using the `JOIN` clause with the `USING` keyword:

```
SELECT *
FROM capitals
JOIN population
USING (country);
```

```
+---------+---------+----------------+
| country | capital | population_mil |
+---------+---------+----------------+
| Russia  | Moscow  | 143            |
+---------+---------+----------------+
| Spain   | Madrid  | 48             |
+---------+---------+----------------+
```

Note that in this case the same-named columns from different tables will "collapse" into one.

`SEMI JOIN` and `ANTI JOIN` — semi-joins and anti-joins

Semi joins return rows from the left table that have at least one match in the right table.

Anti-joins return rows from the left table that have no matches in the right table.

When using semi-joins or anti-joins, the result will never contain more rows than the left table. Semi-joins provide the same logic as the `IN` operator. Anti-joins provide the same logic as the `NOT IN` operator, except that anti-joins ignore `NULL` values from the right table.

Example of a semi-join

Let’s derive a list of such country-capital pairs from the `capitals` table for which the country name is present in the `population` table:

```
SELECT *
FROM capitals
SEMI JOIN population
    USING (country);
```

```
+---------+---------+
| country | capital |
+---------+---------+
| Russia  | Moscow  |
+---------+---------+
| Spain   | Madrid  |
+---------+---------+
```

This request is equivalent to the following:

```
SELECT *
FROM capitals
WHERE country IN (SELECT country FROM population);
```

```
+---------+---------+
| country | capital |
+---------+---------+
| Russia  | Moscow  |
+---------+---------+
| Spain   | Madrid  |
+---------+---------+
```

Example of an anti-coupling

Let's output a list of such country-capital pairs from the `capitals` table for which the country name is not in the `population` table:

```
SELECT *
FROM capitals
ANTI JOIN population
    USING (country);
```

```
+---------+---------+
| country | capital |
+---------+---------+
| Italy   | Rome    |
+---------+---------+
| France  | Paris   |
+---------+---------+
```

This request is equivalent to the following:

```
SELECT *
FROM capitals
WHERE country NOT IN
    (SELECT country FROM population WHERE country IS NOT NULL);
```

```
+---------+---------+
| country | capital |
+---------+---------+
| Italy   | Rome    |
+---------+---------+
| France  | Paris   |
+---------+---------+
```

Closed connection (Self-Join)

Tengri allows you to use a closed join (joining a table to itself) for all types of joins. Note that tables must be specified via aliases, using the same table name without aliases will cause an error:

```
CREATE TABLE t (x INTEGER);
SELECT * FROM t JOIN t USING(x);
```

```
Binder Error:
Duplicate alias "t" in query!
```

Adding aliases allows the request to be processed successfully:

```
CREATE TABLE t (num INTEGER);
SELECT * FROM t t1 JOIN t t2 USING(num);
```

```
+-----+
| num |
+-----+
```

`UNION`

The `UNION` clause appends the results of one query to the results of another. This removes duplicate rows from the result unless the `ALL` operator is added.

Syntax

```
SELECT ...
UNION [ALL]
SELECT ...
[UNION [ALL]
SELECT ...
...
];
```

Examples

- Let's output a list of all countries that appear in the `country` columns in two tables — capitals and population:

```
CREATE TABLE capitals (country VARCHAR, capital VARCHAR);
CREATE TABLE population (country VARCHAR, population_mil BIGINT);
INSERT INTO capitals VALUES
    ('Russia', 'Moscow'),
    ('Italy', 'Rome'),
    ('Spain', 'Madrid'),
    ('France', 'Paris');
INSERT INTO population VALUES
    ('Russia', 143),
    ('Spain', 48),
    ('Brazil', 211);

SELECT country FROM capitals
UNION
SELECT country FROM population;
```

```
+---------+
| country |
+---------+
| Russia  |
+---------+
| Italy   |
+---------+
| France  |
+---------+
| Spain   |
+---------+
| Brazil  |
+---------+
```

- Now, for the same two tables, let's output the list of countries that occur in the `country` columns, but without abbreviating the repeated occurrences. To do this, we use the `ALL` operator:

```
SELECT country FROM capitals
```

```
UNION ALL
SELECT country FROM population;
```

```
+---------+
| country |
+---------+
| Russia |
+---------+
| Italy |
+---------+
| Spain |
+---------+
| France |
+---------+
| Russia |
+---------+
| Spain |
+---------+
| Brazil |
+---------+
```

WHERE

The `WHERE` clause defines the filters that will be applied to the data. This allows you to select only the part of the data you are interested in. Logically, the `WHERE` clause is applied immediately after the `FROM` clause.

Syntax

```
...
WHERE <predicate>
[ ... ]
```

Parameters

- `<predicate>`
  Boolean expression. The expression can contain logical operators such as `AND`, `OR` and `NOT`.

Examples

- Select rows from the table of capitals where the value of `country` is `'Italy'`:

```
SELECT *
FROM capitals
WHERE country = 'Italy';
```

```
+---------+---------+
| country | capital |
+---------+---------+
| Italy   | Rome    |
+---------+---------+
```

- Let's select from the table of days of the week the rows that match the given case-sensitive `LIKE` clause:

```
SELECT *
FROM weekdays
WHERE name LIKE '%S%';
```

```
+--------+----------+
| number |   name   |
+--------+----------+
| 6      | Saturday |
+--------+----------+
| 7      | Sunday   |
+--------+----------+
```

- Let's select from the table of days of the week the rows that match the given `ILIKE` clause, which is case insensitive:

```
SELECT *
FROM weekdays
WHERE name ILIKE '%S%';
```

```
+--------+-----------+
| number |    name   |
+--------+-----------+
| 2      | Tuesday   |
+--------+-----------+
| 3      | Wednesday |
+--------+-----------+
| 4      | Thursday  |
+--------+-----------+
| 6      | Saturday  |
+--------+-----------+
| 7      | Sunday    |
+--------+-----------+
```

- Let's select all rows corresponding to the given compound expression:

```
SELECT *
FROM weekdays
```

```
WHERE number > 5 OR number = 3;
```

```
+--------+-----------+
| number | name |
+--------+-----------+
| 3 | Wednesday |
+--------+-----------+
| 6 | Saturday |
+--------+-----------+
| 7 | Sunday |
+--------+-----------+
```

GROUP BY

The `GROUP BY` clause specifies which columns should be used for grouping when performing any aggregations in a `SELECT` clause. If the `GROUP BY` clause is specified, the query is always an aggregation, even if there is no explicitly specified aggregation in the `SELECT` clause.

If the `GROUP BY` condition is specified, all tuples that have matching data in the grouping columns (i.e., all tuples belonging to the same group) will be aggregated. The values of the grouping columns themselves are not changed, and any other columns can be combined using an aggregate function (e.g. `count`, `sum`, `avg`, etc.).

GROUP BY ALL

Use `GROUP BY ALL` to group all columns in a `SELECT` statement that are not wrapped in aggregate functions. This simplifies the syntax by allowing you to keep the list of columns in one place, and prevents errors by keeping the `SELECT` granularity consistent with the `GROUP BY` granularity (e.g., preventing duplication).

Syntax

The following syntax variants may be used:

```
SELECT ...
  FROM ...
  [ ... ]
  GROUP BY groupItem [ , groupItem [ , ... ] ]
  [ ... ]
```

```
SELECT ...
  FROM ...
  [ ... ]
  GROUP BY ALL
  [ ... ]
```

Where:

```
groupItem ::= { <column_alias> | <position> | <expr> }
```

Parameters

- `<column_alias>`
  The alias of the column specified in the `SELECT` clause.

- `<position>`
  The position of the expression in the `SELECT` clause.

- `<expr>`
  Any expression defined on tables in the current scope.

- `GROUP BY ALL`
  Specifies that all items in a `SELECT` clause that do not use aggregate functions should be used for grouping.

Examples

- Let's calculate the number of records in the `employees` table belonging to each department:

```
SELECT department, count(*)
    FROM employees
    GROUP BY department;
```

- Calculate the average salary for each department of each division:

```
SELECT division, department, avg(salary)
    FROM employees
    GROUP BY division, department;
```

- Group the data by department and division to see all unique department — division pairs:

```
SELECT division, department
    FROM employees
    GROUP BY ALL;
```

HAVING

The `HAVING` clause can be used after the `GROUP BY` clause to specify filter criteria after grouping is complete. In syntax, the `HAVING` clause is identical to the `WHERE` clause, but while the `WHERE` clause is used before

grouping, the `HAVING` clause is used after it.

Syntax

```
SELECT ...
FROM ...
GROUP BY ...
HAVING <predicate>
[ ... ]
```

Parameters

- `<predicate>`
  Boolean expression.

Examples

- Let's count the number of records in the `employees` table that contain each of the different departments, discarding departments with less than 10:

  ```
  SELECT department, count(*)
  FROM employees
  GROUP BY department
  HAVING count(*) >= 10;
  ```

- Calculate the average salary for each department of each division , but only for those departments where the average salary is greater than the median salary:

  ```
  SELECT division, department, avg(salary)
  FROM employees
  GROUP BY division, department
  HAVING avg(salary) > median(salary);
  ```

`QUALIFY`

The `QUALIFY` clause is used to filter the results of window functions. This filtering of results is similar to how the `HAVING` clause filters the results of aggregate functions used in queries with the `GROUP BY` clause.

The `QUALIFY` clause avoids the need for a subquery or expression with `WITH` to perform this filtering (just as `HAVING` avoids subqueries).

Syntax

```
QUALIFY <predicate>
```

The general form of expressions with `QUALIFY` usually looks like this (some variations in order are allowed):

```
SELECT <column_list>
  FROM <data_source>
  [GROUP BY ...]
  [HAVING ...]
  QUALIFY <predicate>
  [ ... ]
```

Parameters

- `<column_list>`
  List of `SELECT` clause.

- `<data_source>`
  The data source is usually a table, but it can be another data source similar to a table, such as a view, a custom table function, etc.

- `<predicate>`
  A predicate is an expression that filters the result after evaluating aggregate and window functions. A predicate is similar to the `HAVING` clause, but without the `HAVING` keyword itself. In addition, the predicate may contain window functions.

Examples

Let's create and fill the table:

```
CREATE TABLE qt (i INTEGER, p CHAR(1), o INTEGER);
INSERT INTO qt (i, p, o) VALUES
    (1, 'A', 1),
    (2, 'A', 2),
    (3, 'B', 1),
    (4, 'B', 2);
```

This query uses a nested structure rather than `QUALIFY`:

```
SELECT *
    FROM (
         SELECT i, p, o,
                ROW_NUMBER() OVER (PARTITION BY p ORDER BY o) AS row_num
            FROM qt
        )
    WHERE row_num = 1;
```

```
+---+---+---+---------+
| I | P | O | ROW_NUM |
|---+---+---+---------|
| 1 | A | 1 |       1 |
| 3 | B | 1 |       1 |
+---+---+---+---------+
```

And this query uses `QUALIFY`:

```
SELECT i, p, o
    FROM qt
    QUALIFY ROW_NUMBER() OVER (PARTITION BY p ORDER BY o) = 1;
```

```
+---+---+---+
| I | P | O |
|---+---+---|
| 1 | A | 1 |
| 3 | B | 1 |
+---+---+---+
```

We use `QUALIFY` to refer to window functions that are in the `SELECT` clause:

```
SELECT i, p, o, ROW_NUMBER() OVER (PARTITION BY p ORDER BY o) AS row_num
    FROM qt
    QUALIFY row_num = 1;
```

```
+---+---+---+---------+
| I | P | O | ROW_NUM |
|---+---+---+---------|
| 1 | A | 1 |       1 |
| 3 | B | 1 |       1 |
+---+---+---+---------+
```

A `QUALIFY` clause can also be combined with aggregate functions and may contain subqueries:

```
SELECT c2, SUM(c3) OVER (PARTITION BY c2) AS r
    FROM t1
    WHERE c3 < 4
    GROUP BY c2, c3
    HAVING sum(c1) > 3
    QUALIFY r IN (
        SELECT min(c1)
        FROM test
        GROUP BY c2
        HAVING min(c1) > 3);
```

ORDER BY

The ORDER BY clause is an output modifier. It is logically applied at the very end of the query (just before LIMIT, if present). The ORDER BY clause sorts rows by sort criteria in ascending or descending order. In addition, in each ORDER BY clause, you can specify whether to move NULL values to the beginning or the end.

A ORDER BY clause may contain one or more comma-separated expressions. If there are no expressions, an error will be generated. Expressions can begin with either an arbitrary scalar expression (which can be a column name), a column position number (where indexing begins with 1), or the ALL keyword. Each expression may be followed by an order modifier (ASC or DESC, default is ASC), and/or a NULL order modifier (NULLS FIRST or NULLS LAST, default is NULLS LAST).

ORDER BY ALL

The ALL keyword specifies that the output should be sorted by each column in order from left to right. The sort direction can be changed using ORDER BY ALL ASC or ORDER BY ALL DESC and/or NULLS FIRST or NULLS LAST. Note that ALL cannot be used in combination with other expressions in a ORDER BY clause - it must be by itself.

The NULL value order modifier

By default, sorting is done with the ASC and NULLS LAST parameters, that is, the values are sorted in ascending order, with the NULL values placed last. This is identical to the default sort order in PostgreSQL. The default sort order can be changed using the following configuration parameters.

Use the default_null_order parameter to change the default NULL sort order to one of the following options:

- NULLS_FIRST.
- NULLS_LAST
- NULLS_FIRST_ON_ASC_LAST_ON_DESC
- NULLS_LAST_ON_ASC_FASC_FIRST_ON_DESC:

For example:

```
SET default_null_order = 'NULLS_FIRST';
```

Use the default_order parameter to change the default sort direction to one of the following options:

- DESC.
- ASC

For example:

```
SET default_order = 'DESC';
```

Collations (matching schemes)

Text is sorted by default using binary comparison collation. This means that values are sorted by their binary representations in UTF-8. While this works well for ASCII-text (for example, for data in English), the sort order may not be correct for other languages. Collations are provided for this purpose.

Syntax

```
SELECT ...
FROM ...
ORDER BY orderItem [ , orderItem , ... ]
[ ... ]
```

Where:

```
orderItem ::= { <column_alias> | <position> | <expr> }
    [ { ASC | DESC } ] [ NULLS { FIRST | LAST } ]
```

Parameters

- `<column_alias>`
  The alias of the column specified in the `SELECT` list.

- `<position>`
  The position of the expression in the `SELECT` list.

- `<expr>`
  Any expression defined on tables in the current scope.

- `{ ASC | DESC }`
  Optionally returns the sort key values in ascending (smallest to largest) or descending (largest to smallest) order.
  Default: `ASC`

- `NULLS { FIRST | LAST }`
  Optionally specifies whether `NULL` values are returned before/after values other than `NULL`, depending on the sort order (`ASC` or `DESC`).
  Default: depends on the sort order (`ASC` or `DESC`), see The `NULL` value order modifier.

Examples

- Output the days of the week, ordered by their name, using the default sort order and the standard order for `NULL` values:

```
SELECT *
FROM weekdays
ORDER BY name;
```

```
+--------+-----------+
| number |    name   |
+--------+-----------+
| 5      | Friday    |
+--------+-----------+
| 1      | Monday    |
+--------+-----------+
| 6      | Saturday  |
+--------+-----------+
| 7      | Sunday    |
+--------+-----------+
| 4      | Thursday  |
+--------+-----------+
| 2      | Tuesday   |
+--------+-----------+
| 3      | Wednesday |
+--------+-----------+
| 8      | null      |
+--------+-----------+
```

- Let's output the days of the week ordered by their name in descending order with `NULL` values at the beginning:

```
SELECT *
FROM weekdays
ORDER BY name DESC NULLS FIRST;
```

```
+--------+-----------+
| number |    name   |
+--------+-----------+
| 8      | null      |
+--------+-----------+
| 3      | Wednesday |
+--------+-----------+
| 2      | Tuesday   |
+--------+-----------+
| 4      | Thursday  |
+--------+-----------+
| 7      | Sunday    |
```

```
+--------+-----------+
| 6      | Saturday  |
+--------+-----------+
| 1      | Monday    |
+--------+-----------+
| 5      | Friday    |
+--------+-----------+
```

- Now let's consider a situation when our table has the days of the week numbered starting from Sunday (as is done in some calendar systems). To bring it back to the usual form, we order the days of the week first by type (`weekend` or not) and then by number:

```
SELECT *
FROM weekdays
ORDER BY weekend, number;
```

```
+--------+-----------+---------+
| number |    name   | weekend |
+--------+-----------+---------+
| 2      | Monday    | false   |
+--------+-----------+---------+
| 3      | Tuesday   | false   |
+--------+-----------+---------+
| 4      | Wednesday | false   |
+--------+-----------+---------+
| 5      | Thursday  | false   |
+--------+-----------+---------+
| 6      | Friday    | false   |
+--------+-----------+---------+
| 1      | Sunday    | true    |
+--------+-----------+---------+
| 7      | Saturday  | true    |
+--------+-----------+---------+
```

- Let's show the difference in sorting using different collations. Let's take the names of two Finnish cities in their Swedish and Finnish variants and sort them by their Swedish names, using the collation rules for English first:

```
SELECT *
FROM finnish_cities
ORDER BY swed_name COLLATE EN
```

```
+-------------+-----------+
|  swed_name  |  fin_name |
+-------------+-----------+
| Åbo         | Turku     |
+-------------+-----------+
| Helsingfors | Helsinki  |
```

```
+-------------+-----------+
```

Now let's do the same thing, but use the collation rules for Swedish:

```
SELECT *
FROM finnish_cities
ORDER BY swed_name COLLATE SV
```

```
+-------------+-----------+
|  swed_name  |  fin_name |
+-------------+-----------+
| Helsingfors | Helsinki  |
+-------------+-----------+
| Åbo         | Turku     |
+-------------+-----------+
```

In the Swedish alphabet, the letter å comes at the end, while in the English collation rules it comes at the beginning. Hence we get the difference in the sorting order.

`LIMIT`

`LIMIT` clause

The `LIMIT` clause is an output modifier. It is logically applied at the very end of the query. The `LIMIT` clause limits the number of output lines.

Note that although `LIMIT` can be used without the `ORDER BY` condition, in that case the results may not be deterministic. Nevertheless, this can be useful, for example when you want to get a quick slice of data.

`OFFSET` clause

The `OFFSET` clause specifies from which position to start reading values, i.e. the first `OFFSET` values are ignored.

Syntax

```
SELECT ...
FROM ...
[ ORDER BY ... ]
LIMIT <count> [ OFFSET <start> ]
[ ... ]
```

Parameters

- `<count>`
  The number of rows to return. Must be a non-negative integer value.
  The value `NULL` is also accepted and treated as unrestricted.

- `OFFSET <start>`
  The line number after which restricted/extracted lines are returned. Must be a non-negative integer value.
  If `OFFSET` is omitted, output starts at the first line in the result set.
  The value `NULL` is also accepted and treated as unrestricted (i.e. no rows will be skipped).

Examples

- Let's select the first 5 days from the table of days of the week:

```
SELECT *
FROM weekdays
LIMIT 5;
```

```
+--------+-----------+
| number |    name   |
+--------+-----------+
| 1      | Monday    |
+--------+-----------+
| 2      | Tuesday   |
+--------+-----------+
| 3      | Wednesday |
+--------+-----------+
| 4      | Thursday  |
+--------+-----------+
| 5      | Friday    |
+--------+-----------+
```

- Let's select 5 rows from the table of days of the week, starting from position 1 (i.e. ignoring the first row):

```
SELECT *
FROM weekdays
LIMIT 5
OFFSET 1;
```

```
+--------+-----------+
| number |    name   |
+--------+-----------+
```

```
| 2      | Tuesday   |
+--------+-----------+
| 3      | Wednesday |
+--------+-----------+
| 4      | Thursday  |
+--------+-----------+
| 5      | Friday    |
+--------+-----------+
| 6      | Saturday  |
+--------+-----------+
```

`LIKE`

Description

An expression with the `LIKE` operator returns `TRUE` if the text string matches the given pattern.

If the pattern does not contain percent signs or underscores, it is interpreted literally, in which case `LIKE` acts as an equality operator:

```
SELECT
    'Tengri' LIKE 'Tengri' AS result;
```

```
+--------+
| result |
+--------+
| true   |
+--------+
```

If a pattern has special characters, it is interpreted as a regular expression rather than literally:

- An underscore `_` in the pattern corresponds to any single character.
- The percent sign `%` in the pattern corresponds to any sequence of zero or more characters.

The `LIKE` pattern match is always applied to the entire string. Therefore, if you want to match a sequence anywhere in the string (cover a substring with the pattern), the pattern must begin and end with the `%` percent sign.

You can also use the opposite expressions `<string> NOT LIKE <pattern>` and `NOT string LIKE <pattern>`:

```
SELECT
    NOT 'Tengri' LIKE 'Tengri' AS result_1,
    'Tengri' NOT LIKE 'Tengri' AS result_2;
```

```
+----------+----------+
| result_1 | result_2 |
```

```
+----------+----------+
| false    | false    |
+----------+----------+
```

`ILIKE` operator

The `ILIKE` operator can be used instead of `LIKE` to make matching case insensitive:

```
SELECT
    'Tengri' ILIKE 'tengri' AS result_1,
    'Tengri' ILIKE '%NGRi'  AS result_2;
```

```
+----------+----------+
| result_1 | result_2 |
+----------+----------+
| true     | true     |
+----------+----------+
```

Examples

- Let's show some examples of how `LIKE` templates work:

  ```
  SELECT
      'Tengri' LIKE 'TNGRi'  AS result_1,
      'Tengri' LIKE 'T%'     AS result_2,
      'TNGRi'  LIKE 'T%'     AS result_3,
      'Tengri' LIKE 'T_____' AS result_4;
  ```

  ```
  +----------+----------+----------+----------+
  | result_1 | result_2 | result_3 | result_4 |
  +----------+----------+----------+----------+
  | false    | true     | true     | true     |
  +----------+----------+----------+----------+
  ```

- Let us select from the table of capitals the countries in which the capital begins with `M`:

  ```
  CREATE TABLE capitals (country VARCHAR, capital VARCHAR);
  INSERT INTO capitals VALUES
      ('Russia', 'Moscow'),
      ('Italy', 'Rome'),
      ('Spain', 'Madrid'),
      ('France', 'Paris');

  SELECT country FROM capitals WHERE capital LIKE 'M%';
  ```

```
+---------+
| country |
+---------+
| Russia  |
+---------+
| Spain   |
+---------+
```

See also

- `SIMILAR TO`
- Text functions
- Functions for regular expressions

`SIMILAR TO`

Description

An expression with the `SIMILAR TO` operator returns `TRUE` or `FALSE` depending on whether its pattern overlaps with the given text string.

This operator is similar to the `LIKE` operator, but in this case the pattern is interpreted as a regular expression. As with `LIKE`, an expression with `SIMILAR TO` is checked successfully only if the pattern is applied to the entire string. This differs from the usual behaviour of regular expressions, where the pattern can be applied to any part of the string.

Regular expressions use the syntax RE2.

You can also use opposing expressions with opposite meanings
`<string> NOT SIMILAR TO <pattern>` and `NOT string SIMILAR TO <pattern>`.

Expressions with the `SIMILAR TO` operator are completely synonymous with expressions with the `~` operator.

Examples

```
SELECT
    'Tengri' SIMILAR TO 'TNGRi'  AS result_1, -- false expected
    'Tengri' SIMILAR TO 'T.*'    AS result_2,
    'Tengri' SIMILAR TO 'T.+i'   AS result_3,
    'TNGRi'  SIMILAR TO 'T.+i'   AS result_4,
    'T.+i'   SIMILAR TO 'T\.\+i' AS result_5;
```

```
+----------+----------+----------+----------+----------+
| result_1 | result_2 | result_3 | result_4 | result_5 |
+----------+----------+----------+----------+----------+
| false    | true     | true     | true     | true     |
```

```
+----------+----------+----------+----------+----------+
```

```
SELECT
    'Tengri' NOT SIMILAR TO 'TNGRi'  AS result_1,
    NOT 'T.{3}i' SIMILAR TO 'T.{3}i' AS result_2;
```

```
+----------+----------+
| result_1 | result_2 |
+----------+----------+
| true     | true     |
+----------+----------+
```

See also

- Operator `~`
- `LIKE`
- Text functions
- Functions for regular expressions

`AS`

The `AS` operator in SQL-queries can be used in several different functions.

Specifying names for columns in a query

The `AS` operator can be used to specify names (aliases) for columns in a table that will be output as the result of a query or used in the query itself:

```
SELECT
    name AS Student
    FROM students
```

If the given name contains a space, it must be enclosed in inverted commas:

```
SELECT
    name AS "Student Name"
    FROM students
```

Specifying names for tables in a query

The `AS` operator can be used to specify names (aliases) for tables that are used in a query:

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
    FROM
        Customers AS c,
        Orders AS o
    WHERE
        c.CustomerName="Tengri" AND c.CustomerID=o.CustomerID;
```

Here in the query, the aliases `c` and `o` for the tables `Customers` and `Orders` that are queried are used for brevity.

Specifying a query when creating a table

When creating a table, the `AS` operator can be used to specify a data query, the result of which will be written to the table being created:

```
CREATE TABLE films_recent AS
    SELECT * FROM films WHERE date_prod >= '2025-01-01';
```

In this case, you can also use FROM-first syntax and discard `SELECT *`. Then the query (equivalent to the previous one) will look like this:

```
CREATE TABLE films_recent AS
    FROM films WHERE date_prod &gt;= '2025-01-01';
```

Functions

Aggregate functions

- `any_value`
- `array_agg`
- `avg`
- `count`
- `count(argument)`
- `count_if`
- `max`
- `min`
- `median`
- `sum`

Numerical functions

- `abs`
- `add`
- `ceil`
- `cos`
- `divide`
- `even`
- `exp`
- `floor`
- `fmod`
- `gcd`
- `greatest`
- `isfinite`
- `isinf`
- `isnan`
- `lcm`
- `least`
- `lgamma`
- `ln`
- `log`
- `log2`
- `multiply`
- `nextafter`
- `pi`
- `pow`
- `radians`
- `random`
- `round_even`
- `round`
- `setseed`
- `sign`
- `signbit`
- `sin`
- `sqrt`
- `subtract`

- `trunc`
- Оператор `+`
- Оператор `-`
- Оператор `*`
- Оператор `/`
- Оператор `%`
- Оператор `^`

Text functions

- `contains`
- `length`
- `strlen`
- `trim`
- `ltrim`
- `rtrim`
- `lower`
- `upper`
- `split`
- `chr`

Functions for regular expressions

- `regexp_extract`
- `regexp_extract_all`
- `regexp_full_match`
- `regexp_matches`
- `regexp_replace`
- `regexp_split_to_array`
- `regexp_split_to_table`
- Оператор `~`

Functions for date and time

- `current_time`
- `now`
- `datepart`
- `date_diff`

- `date_trunc`
- `strptime`

Functions for JSON

- `json_array_length`
- `json_contains`
- `json_exists`
- `json_extract`
- `json_extract_string`
- `json_group_array`
- `json_group_object`
- `json_keys`
- `json_transform`
- `json_transform_strict`
- `json_valid`
- `json_value`
- `json`
- `read_json`

Functions for binary data

- `concat`
- `md5`
- `sha1`
- `sha256`
- Оператор `||`

Window functions

- `cume_dist`
- `dense_rank`
- `first_value`
- `lag`
- `last_value`
- `lead`
- `nth_value`
- `ntile`

- `percent_rank`
- `rank`
- `row_number`

Utilities

- `coalesce`
- `generate_series`
- `hash`
- `unnest`

Python module `tngri` functions

- `tngri.sql`
- `tngri.upload_df`
- `tngri.upload_file`
- `tngri.upload_s3`

Aggregate functions

Aggregate functions are functions that combine values from multiple rows into one.

Aggregate functions differ from scalar functions and window functions in that they change the cardinality of the result. That is why they can be used in queries only in expressions `SELECT` and `HAVING`.

`DISTINCT` expression in aggregate functions

When the expression `DISTINCT` is used, only unique values are considered when calculating the value of an aggregate function. Often this expression is used in conjunction with an aggregate function `count()` to get the number of unique items, but it can be used with other aggregate functions as well.

Example

```
CREATE TABLE cities(city_name VARCHAR);

INSERT INTO cities VALUES
    ('Moscow'),
    ('Moscow'),
    ('Paris'),
    ('Madrid');
```

```
SELECT
    count(DISTINCT city_name) AS distinct_cities_num,
    count(city_name) AS cities_num
FROM cities;
```

```
+--------------------+------------+
| distinct_cities_num | cities_num |
+--------------------+------------+
| 3                  | 4          |
+--------------------+------------+
```

Some aggregate functions are insensitive to repeated values (e.g, `min()` и `max()`), and for them the `DISTINCT` expression is ignored.

`any_value()`

Description Returns the first value from `argument` other than `NULL`.

Usage `any_value(argument)`

This function is affected by the sort order in the table.

▼ *See example*

```
CREATE TABLE numbers(number_asc BIGINT, number_desc BIGINT);

INSERT INTO numbers VALUES
    (NULL,  3),
    (1,     2),
    (2,     1),
    (3,     NULL);

SELECT
    any_value(number_asc) AS "any value from number asc",
    any_value(number_desc) AS "any value from number desc"
FROM numbers;
```

```
+--------------------------+---------------------------+
| any value from number asc | any value from number desc |
+--------------------------+---------------------------+
| 1                        | 3                         |
+--------------------------+---------------------------+
```

`array_agg()`

Description Returns a list containing all the values of a column.

Usage `array_agg(argument)`

Aliases `list()`

This function is affected by the sort order in the table.

▼ *See example*

```
CREATE TABLE numbers(number BIGINT);

INSERT INTO numbers VALUES
    (1),
    (2),
    (3),
    (NULL);

SELECT
    array_agg(number) AS array_agg,
    list(number) AS list
FROM numbers;
```

```
+--------------+--------------+
|   array_agg  |     list     |
+--------------+--------------+
| {1,2,3,None} | {1,2,3,None} |
+--------------+--------------+
```

avg()

Description Calculates the average of all non-empty values in `argument`.

Usage `avg(argument)`

Aliases `mean()`

▼ *See example*

```
CREATE TABLE numbers(number BIGINT);

INSERT INTO numbers VALUES
    (1),
    (2),
    (3),
    (NULL);

SELECT
    avg(number) AS average,
    mean(number) AS mean
FROM numbers;
```

```
+---------+------+
```

```
| average | mean |
+---------+------+
| 2       | 2    |
+---------+------+
```

`count()`

Description Calculates the number of rows in the group.

Usage `count()`

Aliases `count(*)`

▼ *See example*

```sql
CREATE TABLE numbers(number BIGINT);

INSERT INTO numbers VALUES
    (1),
    (2),
    (3),
    (NULL);

SELECT
    count() AS rows_count,
    count(*) AS rows_count_star
FROM numbers;
```

```
+------------+-----------------+
| rows_count | rows_count_star |
+------------+-----------------+
| 4          | 4               |
+------------+-----------------+
```

`count(argument)`

Description Calculates the number of non-empty values in `argument`.

Usage `count(argument)`

▼ *See example*

```sql
CREATE TABLE numbers(number BIGINT);

INSERT INTO numbers VALUES
    (1),
    (2),
    (3),
    (NULL);
```

```
SELECT
    count() AS rows_count,
    count(number) AS values_count
FROM numbers;
```

```
+------------+--------------+
| rows_count | values_count |
+------------+--------------+
| 4          | 3            |
+------------+--------------+
```

count_if()

Description Returns the number of records that satisfy the condition, or `NULL` if no records satisfy the condition.

Usage `count_if(<condition>)`

▼ *See example*

```
CREATE TABLE text_table(text_data VARCHAR);

INSERT INTO text_table VALUES
('Tengri'),
('Tengri'),
('TNGRi'),
(NULL);

SELECT
    COUNT_IF(TRUE) AS row_number,
    COUNT_IF(text_data = 'Tengri') AS tengri_number
FROM text_table;
```

```
+------------+---------------+
| row_number | tengri_number |
+------------+---------------+
| 4          | 2             |
+------------+---------------+
```

max()

Description Returns the maximum value available in `argument`.

Usage `max(argument)`

▼ *See example*

```
CREATE TABLE numbers(number BIGINT);
```

```
INSERT INTO numbers VALUES
    (1),
    (2),
    (3),
    (NULL);

SELECT
    max(number) AS max,
    min(number) AS min
FROM numbers;
```

```
+-----+-----+
| max | min |
+-----+-----+
| 3   | 1   |
+-----+-----+
```

`min()`

Description Returns the minimum value present in `argument`.

Usage `min(argument)`

▼ *See example*

```
CREATE TABLE numbers(number BIGINT);

INSERT INTO numbers VALUES
    (1),
    (2),
    (3),
    (NULL);

SELECT
    max(number) AS max,
    min(number) AS min
FROM numbers;
```

```
+-----+-----+
| max | min |
+-----+-----+
| 3   | 1   |
+-----+-----+
```

`median()`

Description Returns the median value of all non-empty values in `argument`.

Usage `median(argument)`

In case of an even number of values, the average between the two centre values is taken.

▼ *See example*

Let's show the difference between the median value and the mean value (`avg()`) on examples of even (`_even`) and odd (`_odd`) number sets containing empty values.

```
CREATE TABLE numbers(
    number_even BIGINT,
    number_odd BIGINT);

INSERT INTO numbers VALUES
    (1,    1),
    (2,    2),
    (3,    10),
    (10,   NULL),
    (NULL, NULL);

SELECT
    median(number_even) AS median_even,
    avg(number_even) AS avg_even,
    median(number_odd) AS median_odd,
    avg(number_odd) AS avg_odd
FROM numbers;
```

```
+-------------+----------+------------+-------------------+
| median_even | avg_even | median_odd |      avg_odd      |
+-------------+----------+------------+-------------------+
| 2.5         | 4        | 2          | 4.333333333333333 |
+-------------+----------+------------+-------------------+
```

`sum()`

Description Calculates the sum of all non-empty values in `argument`.

Usage `sum(argument)`

In case of boolean values, counts the number of `True` values.

▼ *See example*

```
CREATE TABLE numbers(
    number BIGINT,
    boolean BOOL);

INSERT INTO numbers VALUES
    (1,    True),
    (2,    False),
    (3,    False),
    (NULL, NULL);
```

```
SELECT
    sum(number) AS sum_number,
    sum(boolean) AS sum_boolean
FROM numbers;
```

```
+------------+-------------+
| sum_number | sum_boolean |
+------------+-------------+
| 6          | 1           |
+------------+-------------+
```

Numerical functions

Numeric functions — these are functions for working with data of numeric types: `BIGINT`, `NUMERIC` and `DOUBLE`.

`abs()`

Description Calculates the modulus of a number.

Usage `abs(num)`.

▼ *See examples*

```
SELECT
    abs(-1)  AS result_1,
    abs(0)   AS result_2,
    abs(1.1) AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1        | 0        | 1.1      |
+----------+----------+----------+
```

`add()`

Description adds numbers together.

Usage `add(num, num)`

See also Operator `+`.

▼ *See examples*

```
SELECT
    add(1, 1)      AS result_1,
    add(-1.1, 2.1) AS result_2,
```

```
    add(1)        AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 2        | 1.0      | 1        |
+----------+----------+----------+
```

`ceil()`

Description rounds a number to the higher side.

Usage `ceil(num)`

Aliases `ceiling()`

▼ *See examples*

```
SELECT
    ceil(0.1)  AS result_1,
    ceil(-0.1) AS result_2,
    ceiling(1) AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1        | 0        | 1        |
+----------+----------+----------+
```

`cos()`

Description Calculates the cosine of an angle given in radians.

Usage `cos(num)`

▼ *See examples*

```
SELECT
    cos(0)      AS result_1,
    cos(pi())   AS result_2,
    cos(pi()/3) AS result_3;
```

```
+----------+----------+--------------------+
| result_1 | result_2 | result_3           |
+----------+----------+--------------------+
| 1        | -1       | 0.5000000000000001 |
+----------+----------+--------------------+
```

`divide()`

Description Returns the result of division as an integer.

Usage `divide(num, num)`

▼ *See examples*

```
SELECT
    divide(7, 2)  AS result_1,
    divide(7, -2) AS result_2,
    divide(7, 0)  AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 3        | -3       | null     |
+----------+----------+----------+
```

`even()`

Description Rounds to the nearest even number away from zero.

Usage `even(num)`

▼ *See examples*

```
SELECT
    even(2.1)  AS result_1,
    even(-2.1) AS result_2,
    even(0)    AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 4        | -4       | 0        |
+----------+----------+----------+
```

`exp()`

Description Calculates the exponent of a number.

Usage `exp(num)`.

Calculates the exponential value of a number: `e^x`.

▼ *See examples*

```
SELECT
```

```
    exp(0)  AS result_1,
    exp(1)  AS result_2,
    exp(-1) AS result_3;
```

```
+----------+-------------------+---------------------+
| result_1 | result_2          | result_3            |
+----------+-------------------+---------------------+
| 1        | 2.718281828459045 | 0.36787944117144233 |
+----------+-------------------+---------------------+
```

`floor()`

Description rounds a number to the smaller side.

Usage `floor(num)`

▼ *See examples*

```
SELECT
    floor(0.9)  AS result_1,
    floor(-0.9) AS result_2,
    floor(1)    AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 0        | -1       | 1        |
+----------+----------+----------+
```

`fmod()`

Description Returns the remainder of dividing the first argument by the second argument.

Usage `fmod(num, num)`

▼ *See examples*

```
SELECT
    fmod(3, 2)   AS result_1,
    fmod(3.1, 2) AS result_2,
    fmod(-10, 4) AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1        | 1.1      | 2        |
+----------+----------+----------+
```

`gcd()`

Description Calculates the greatest common divisor of two numbers.

Usage `gcd(num, num)`

Aliases `greatest_common_divisor()`

▼ *See examples*

```
SELECT
    gcd(12, 9)  AS result_1,
    gcd(-12, 9) AS result_2,
    gcd(12, 0)  AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 3        | 3        | 12       |
+----------+----------+----------+
```

`greatest()`

Description Returns the largest number specified in the arguments.

Usage `greatest(num[, num, ...])`

▼ *See examples*

```
SELECT
    greatest(1, 2, 3, 4, 4) AS result_1,
    greatest(1, -1)         AS result_2,
    greatest(0)             AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 4        | 1        | 0        |
+----------+----------+----------+
```

`isfinite()`

Description Checks whether a number is finite.

Usage `isfinite(num)`

▼ *See examples*

```
SELECT
```

```
    isfinite(1)                  AS result_1,
    isfinite('Infinity'::DOUBLE) AS result_2,
    isfinite(NULL)               AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| true     | false    | null     |
+----------+----------+----------+
```

`isinf()`

Description Checks whether a number is infinite.

Usage `isinf(num)`

▼ *See examples*

```
SELECT
    isinf(1)                  AS result_1,
    isinf('Infinity'::DOUBLE) AS result_2,
    isinf(NULL)               AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| false    | true     | null     |
+----------+----------+----------+
```

`isnan()`

Description Checks if the argument has the value `NaN` (*Not a Number*).

Usage `isnan(num)`

▼ *See examples*

```
SELECT
    isnan('NaN'::DOUBLE) AS result_1,
    isnan(1.1)           AS result_2,
    isnan(NULL)          AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| true     | false    | null     |
+----------+----------+----------+
```

`lcm()`

Description Calculates the least common multiple of two numbers.

Usage `lcm(num, num)`

Aliases `least_common_multiple()`

▼ *See examples*

```
SELECT
    lcm(3, 7)     AS result_1,
    lcm(333, 777) AS result_2,
    lcm(37, 0)    AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 21       | 2331     | 0        |
+----------+----------+----------+
```

`least()`

Description Returns the smallest number specified in the arguments.

Usage `least(num[, num, ...])`

▼ *See examples*

```
SELECT
    least(1, 1, 2, 3, 4) AS result_1,
    least(1, -1)         AS result_2,
    least(0)             AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1        | -1       | 0        |
+----------+----------+----------+
```

`lgamma()`

Description Calculates the logarithm of the gamma function.

Usage `lgamma(num)`

▼ *See examples*

```
SELECT
```

```
    lgamma(1)   AS result_1,
    lgamma(11)  AS result_2,
    lgamma(1.1) AS result_3;
```

```
+----------+--------------------+-----------------------+
| result_1 | result_2           | result_3              |
+----------+--------------------+-----------------------+
| 0        | 15.104412573075518 | -0.049872441259839764 |
+----------+--------------------+-----------------------+
```

`ln()`

Description Calculates the natural logarithm of a number.

Usage `ln(num)`.

▼ *See examples*

```
SELECT
    ln(1)   AS result_1,
    ln(11)  AS result_2,
    ln(1.1) AS result_3;
```

```
+----------+--------------------+----------------------+
| result_1 | result_2           | result_3             |
+----------+--------------------+----------------------+
| 0        | 2.3978952727983707 | 0.09531017980432493 |
+----------+--------------------+----------------------+
```

`log()`

Description Calculates the logarithm of a number on base `10`.

Usage `log(num)`

Aliases `log10()`

▼ *See examples*

```
SELECT
    log(1)    AS result_1,
    log(100)  AS result_2,
    log(0.01) AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 0        | 2        | -2       |
```

```
+----------+----------+----------+
```

log2()

Description Calculates the logarithm of a number on base `2`.

Usage `log2(num)`

▼ *See examples*

```
SELECT
    log2(1)    AS result_1,
    log2(2)    AS result_2,
    log2(4096) AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 0        | 1        | 12       |
+----------+----------+----------+
```

multiply()

Description Multiplies two numbers.

Usage `multiply(num, num)`

See also Operator *.

▼ *See examples*

```
SELECT
    multiply(2, 2)      AS result_1,
    multiply(0, 2)      AS result_2,
    multiply(0.2, -0.2) AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 4        | 0        | -0.04    |
+----------+----------+----------+
```

nextafter()

Description Returns the next value with variable precision (of type `DOUBLE`) after the first number towards the second number.

Usage `nextafter(num, num)`

▼ *See examples*

```
SELECT
    nextafter(1::DOUBLE, 2)  AS result_1,
    nextafter(1::BIGINT, 2)  AS result_2,
    nextafter(-1::BIGINT, 0) AS result_3;
```

```
+-------------------+-------------------+--------------------+
|      result_1     |      result_2     |       result_3     |
+-------------------+-------------------+--------------------+
| 1.0000000000000002 | 1.0000000000000002 | -0.9999999999999999 |
+-------------------+-------------------+--------------------+
```

`pi()`

Description Returns the value of the number π.

Usage `pi()`

▼ *See examples*

```
SELECT
    pi()   AS result_1,
    pi()/2 AS result_2,
    2*pi() AS result_3;
```

```
+------------------+------------------+------------------+
|     result_1     |     result_2     |     result_3     |
+------------------+------------------+------------------+
| 3.141592653589793 | 1.5707963267948966 | 6.283185307179586 |
+------------------+------------------+------------------+
```

`pow()`

Description Exposes the first argument to the degree given by the second argument.

Usage `pow(num, num)`

Aliases `power()`

See also Operator `^`.

▼ *See examples*

```
SELECT
    pow(2, 5)   AS result_1,
    pow(25, -1) AS result_2,
    pow(25, 0)  AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 32       | 0.04     | 1        |
+----------+----------+----------+
```

`radians()`

Description converts degrees to radians.

Usage `radians(num)`

▼ *See examples*

```
SELECT
    radians(0)         AS result_1,
    radians(180)       AS result_2,
    radians(-180/pi()) AS result_3;
```

```
+----------+-------------------+----------+
| result_1 |      result_2     | result_3 |
+----------+-------------------+----------+
| 0        | 3.141592653589793 | -1       |
+----------+-------------------+----------+
```

`random()`

Description Returns an arbitrary number (of type `DOUBLE`) between `0` and `1`.

Usage `random()`

See also `setseed()`.

▼ *See examples*

```
SELECT
    random() AS result;
```

```
+--------------------+
|       result       |
+--------------------+
| 0.5656213557274057 |
+--------------------+
```

`round_even()`

Description Round the number from the first argument to the nearest even number with the

precision specified in the second argument.

Usage `round_even(num, num)`

Aliases `roundbankers()`

The second argument specifies the number of decimal places of precision and can be a negative number.

For more information on rounding to the nearest even number, see here.

▼ *See examples*

```
SELECT
    round_even(4.5, 0)    AS result_1,
    round_even(3.5, 0)    AS result_2,
    round_even(-4.5, 0)   AS result_3,
    round_even(-3.5, 0)   AS result_4,
    round_even(4.45, 1)   AS result_5,
    round_even(4.35, 1)   AS result_6,
    round_even(35.35, -1) AS result_7;
```

```
+----------+----------+----------+----------+----------+----------+----------+
| result_1 | result_2 | result_3 | result_4 | result_5 | result_6 | result_7 |
+----------+----------+----------+----------+----------+----------+----------+
| 4        | 4        | -4       | -4       | 4.4      | 4.4      | 40       |
+----------+----------+----------+----------+----------+----------+----------+
```

`round()`

Description `rounds the number from the first argument to the precision specified in the second argument.

Usage `round(num, num)`

The second argument specifies the number of decimal places of precision and can be a negative number.

▼ *See examples*

```
SELECT
    round(4.5, 0)   AS result_1,
    round(4.45, 1)  AS result_2,
    round(44.5, -1) AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 5        | 4.5      | 40       |
+----------+----------+----------+
```

`setseed()`

Description Fixes the initial value for the `random()` function.

Usage `setseed(num)`

See also `random()`.

▼ *See examples*

```
SELECT
    setseed(0.5) AS seed,
    random() AS random;
```

```
+------+--------------------+
| seed | random             |
+------+--------------------+
| null | 0.8511131886287325 |
+------+--------------------+
```

`sign()`

Description Returns `-1`, `1` or `0` depending on the sign of the argument.

Usage `sign(num)`

▼ *See examples*

```
SELECT
    sign(10)  AS result_1,
    sign(-10) AS result_2,
    sign(0)   AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1        | -1       | 0        |
+----------+----------+----------+
```

`signbit()`

Description Determines whether the sign bit of a real number is set.

Usage `signbit(num)`

▼ *See examples*

```
SELECT
    signbit(-1)                 AS result_1,
```

```
    signbit(-'Infinity'::DOUBLE) AS result_2,
    signbit(0)                   AS result_3,
    signbit(1)                   AS result_4,
    signbit('Infinity'::DOUBLE)  AS result_5;
```

```
+----------+----------+----------+----------+----------+
| result_1 | result_2 | result_3 | result_4 | result_5 |
+----------+----------+----------+----------+----------+
| true     | true     | false    | false    | false    |
+----------+----------+----------+----------+----------+
```

`sin()`

Description Calculates the sine of an angle given in radians.

Usage `sin(num)`

▼ *See examples*

```
SELECT
    sin(0)          AS result_1,
    sin(pi()/2)     AS result_2,
    sin((3*pi())/2) AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 0        | 1        | -1       |
+----------+----------+----------+
```

`sqrt()`

Description Calculates the square root.

Usage `sqrt(num)`

The number `num` must be non-negative.

▼ *See examples*

```
SELECT
    sqrt(4)   AS result_1,
    sqrt(144) AS result_2,
    sqrt(0)   AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
```

```
| 2        | 12       | 0        |
+----------+----------+----------+
```

`subtract()`

Description Subtracts the second argument from the first argument.

Usage `subtract(num, num)`

See also Operator `-`.

▼ *See examples*

```
SELECT
    subtract(1, 2)     AS result_1,
    subtract(1.1, 2.2) AS result_2,
    subtract(-1, -2)   AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| -1       | -1.1     | 1        |
+----------+----------+----------+
```

`trunc()`

Description Discards all characters after the decimal separator.

Usage `trunc(num)`

Not to be confused with rounding `round`.

▼ *See examples*

```
SELECT
    trunc(1.99)  AS result_1,
    trunc(-11.9) AS result_2,
    trunc(0.119) AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1        | -11      | 0        |
+----------+----------+----------+
```

Operator `+`

Description Adds the right argument to the left argument.

Usage `<num> + <num> [+ ...]` or `TIME + INTERVAL [+ ...]`.

If used with types for for date and time, adds the interval to the time value. Returns a value of type `TIME`.

See also `add()`.

▼ *See examples*

```
SELECT
    3 + 2      AS result_1,
    3 + 2 + -1 AS result_2,
    1.1 + 1.9  AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 5        | 4        | 3.0      |
+----------+----------+----------+
```

```
SELECT
    TIME '12:11:10'     + INTERVAL 3 hours AS result_time_1,
    INTERVAL '12:11:10' + TIME '1:1:1'     AS result_time_2;
```

```
+---------------+---------------+
| result_time_1 | result_time_2 |
+---------------+---------------+
| 15:11:10      | 13:12:11      |
+---------------+---------------+
```

Operator `-`

Description Subtracts the right argument from the left argument.

Usage `<num> - <num> [-...]` or `TIME - INTERVAL [-...]`.

If used with types for for date and time, subtracts the interval from the time value. Returns a value of type `TIME`.

See also `subtract()`.

▼ *See examples*

```
SELECT
    3 - 2      AS result_1,
    3 - 2 - +1 AS result_2,
    1.2 - 0.2  AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1        | 0        | 1.0      |
+----------+----------+----------+
```

```
SELECT
    TIME '12:11:10' - INTERVAL 3 HOUR AS result_time_1,
    TIME '12:11:10' - INTERVAL 3 HOUR - INTERVAL 1 HOUR AS result_time_2;
```

```
+---------------+---------------+
| result_time_1 | result_time_2 |
+---------------+---------------+
| 09:11:10      | 08:11:10      |
+---------------+---------------+
```

Operator *

Description Multiplies the arguments.

Usage `<num> * <num>[ * <num>, {⋯]`.

See also `multiply()`.

▼ *See examples*

```
SELECT
    3*2     AS result_1,
    3*+2*-2 AS result_2,
    3*0     AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 6        | -12      | 0        |
+----------+----------+----------+
```

Operator /

Description Divides the left argument by the right argument.

Usage `<num> / <num>[ / <num>,...]`

Returns the result as a number with variable precision (of type `DOUBLE`).

▼ *See examples*

```
SELECT
    3/2     AS result_1,
    3/+2/-2 AS result_2,
    3/1     AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1.5      | -0.75    | 3        |
+----------+----------+----------+
```

Operator `%`

Description Returns the remainder of the left argument divided by the right argument.

Usage `<num> % <num>[ % <num>, { {⋯]`.

▼ *See examples*

```
SELECT
    3 % 2       AS result_1,
    15 % 10 % 3 AS result_2,
    5 % 2.4     AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1        | 2        | 0.2      |
+----------+----------+----------+
```

Operator `^`

Description Elevates the left argument to the degree given by the right argument.

Usage `<num> ^ <num>[ ^ <num>, ...]`

See also `pow()`.

▼ *See examples*

```
SELECT
    2^3   AS result_1,
    2^3^2 AS result_2,
    1^0   AS result_3;
```

```
+----------+----------+----------+
```

```
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 8        | 64       | 1        |
+----------+----------+----------+
```

Text functions

Text functions are functions for working with text strings (data of type `VARCHAR`).

`concat()`

Description Concatenates multiple strings, arrays or binary values.

Usage `concat(argument1, argument2, {⋯)`.

Empty values (`NULL`) are ignored.

See also Operator `||`.

▼ *See example*

```
SELECT
    concat('\xAA'::BLOB, '\xff'::BLOB) as result_blob,
    concat('I', ' ', 'love', ' ', 'Tengri') as result_string,
    concat(['T', 'e'], ['n', 'g', 'r', 'i']) as result_array;
```

```
+-------------+---------------+---------------+
| result_blob | result_string |  result_array |
+-------------+---------------+---------------+
| \xAA\xFF    | I love Tengri | {T,e,n,g,r,i} |
+-------------+---------------+---------------+
```

`contains()`

Description Returns `true` if the specified string `string` contains the searched substring `search_string`.

Usage `contains(string, search_string)`

▼ *See example*

```
SELECT
    contains('I love Tengri', 'Tengri') AS check_name,
    contains('I love Tengri', 'TNGRi')  AS check_nickname;
```

```
+------------+----------------+
| check_name | check_nickname |
+------------+----------------+
```

```
| true        | false          |
+-------------+----------------+
```

`length()`

Description Returns the number of characters in a string.

Usage `length(string)`

Aliases `char_length()`, `character_length()`

▼ *See example*

```
SELECT
    length('I love Tengri 𐰇') AS length;
```

```
+--------+
| length |
+--------+
| 15     |
+--------+
```

`strlen()`

Description Returns the number of bytes in the string.

Usage `strlen(string)`

▼ *See example*

```
SELECT
    strlen('Tengri 𐰇') AS strlen;
```

```
+--------+
| strlen |
+--------+
| 11     |
+--------+
```

`trim()`

Description Removes all occurrences of any of the specified characters on both sides of the string.

Usage `trim(string[, characters])`

If no characters to be deleted are specified, the default character is a space.

▼ *See examples*

```
SELECT
    '"' || trim('  Tengri ') || '"' AS trim;
```

```
+----------+
|   trim   |
+----------+
| "Tengri" |
+----------+
```

```
SELECT
    trim('[Tengri]', '{([])}') AS trim_brackets;
```

```
+---------------+
| trim_brackets |
+---------------+
| Tengri        |
+---------------+
```

`ltrim()`

Description Removes all occurrences of any of the specified characters at the beginning of a string.

Usage `ltrim(string[, characters])`

If no characters to be deleted are specified, the default character is a space.

▼ *See examples*

```
SELECT
    '"' || ltrim(' Tengri ') || '"' AS ltrim;
```

```
+-----------+
|   ltrim   |
+-----------+
| "Tengri " |
+-----------+
```

```
SELECT
    ltrim('{{{[Tengri])}}', '{([])}') AS ltrim_brackets;
```

```
+----------------+
| ltrim_brackets |
+----------------+
| Tengri])}}     |
```

```
+----------------+
```

`rtrim()`

Description Removes all occurrences of any of the specified characters at the end of a string.

Usage `rtrim(string[, characters])`

If no characters to be deleted are specified, the default character is a space.

▼ *See examples*

```
SELECT
    '"' || rtrim(' Tengri ') || '"' AS ltrim;
```

```
+-----------+
|   ltrim   |
+-----------+
| " Tengri" |
+-----------+
```

```
SELECT
    rtrim('{{{[Tengri])}}}', '{([])}') AS rtrim_brackets;
```

```
+----------------+
| rtrim_brackets |
+----------------+
| {{{[Tengri     |
+----------------+
```

`lower()`

Description Converts a string to lower case.

Usage `lower(string)`

Aliases `lcase()`

▼ *See example*

```
SELECT
    lower('TNGRi') AS lower;
```

```
+-------+
| lower |
+-------+
| tngri |
```

```
+-------+
```

upper()

Description Converts a string to uppercase.

Usage `upper(string)`.

Aliases `ucase()`

▼ *See example*

```
SELECT
    upper('Tengri') AS upper;
```

```
+--------+
|  upper |
+--------+
| TENGRI |
+--------+
```

split()

Description Splits a string into two parts by the given separator.

Usage `split(string, separator)`.

Aliases `str_split`, `string_split`, `string_to_array`.

▼ *See example*

```
SELECT
    split('I love Tengri', ' ') AS words;
```

```
+-----------------+
|      words      |
+-----------------+
| {I,love,Tengri} |
+-----------------+
```

chr()

Description Returns the character corresponding to the value of the code ASCII or code Unicode, given in `argument`.

Usage `chr(argument)`

▼ *See example*

```
SELECT
    chr(84) || chr(78) || chr(71) || chr(82) || chr(105) AS chr;
```

```
+-------+
|  chr  |
+-------+
| TNGRi |
+-------+
```

`md5()`

Description Returns a hash MD5 of data from `argument` as a string (`VARCHAR`).

Usage `md5(argument)`

The `argument` can be binary data or a string.

▼ *See examples*

```
SELECT
    md5('\xAA\xFF'::BLOB) as md5_hash;
```

```
+----------------------------------+
|             md5_hash             |
+----------------------------------+
| 1fab7f7621f5ddc051ebd1f2c63c4665 |
+----------------------------------+
```

```
SELECT
    md5('Tengri') as md5_hash;
```

```
+----------------------------------+
|             md5_hash             |
+----------------------------------+
| 846b02d31131a10bd6ac0ba189c65bef |
+----------------------------------+
```

`sha1()`

Description Returns a hash SHA-1 of the data from `argument` as a string (`VARCHAR`).

Usage `sha1(argument)`

The `argument` can be binary data or a string.

▼ *See examples*

```
SELECT
    sha1('\xAA\xFF'::BLOB) as sha1_hash;
```

```
+------------------------------------------+
|                 sha1_hash                |
+------------------------------------------+
| e89b0db325637edfacde04a76005c492e2c5aeca |
+------------------------------------------+
```

```
SELECT
    sha1('Tengri') as sha1_hash;
```

```
+------------------------------------------+
|                 sha1_hash                |
+------------------------------------------+
| b514525a19995a2442d7565bfd9bb42d9dc71a13 |
+------------------------------------------+
```

sha256()

Description Returns a hash SHA-256 of the data from `argument` as a string (`VARCHAR`).

Usage `sha256(argument)`

The `argument` can be binary data or a string.

▼ *See example*

```
SELECT
    sha256('\xAA\xFF'::BLOB) as sha256_hash;
```

```
+------------------------------------------------------------------+
|                           sha256_hash                            |
+------------------------------------------------------------------+
| 768318522cac43261e8ef4946c2296a3643d523a8d5bda8ff5b82aa64470421a |
+------------------------------------------------------------------+
```

```
SELECT
    sha256('Tengri') as sha256_hash;
```

```
+------------------------------------------------------------------+
|                           sha256_hash                            |
+------------------------------------------------------------------+
| 8aaacef66663b14ee7c5a03dbaec7b40f0f3bf17bd12d2ed4f9aaad0e10a0d77 |
```

```
+-----------------------------------------------------------------+
```

Operator ||

Description	Concatenates multiple strings, arrays or binary values.
Usage	`argument1 || argument2 || argument2 || ...`.

Empty values (`NULL`) are ignored.

See also concat().

▼ *See examples*

```
SELECT
    '\xAA'::BLOB || '\xff'::BLOB as result_blob,
    'I' || ' ' || 'love' || ' ' || 'Tengri' as result_string,
    ['T', 'e'] || ['n', 'g', 'r', 'i'] as result_array;
```

```
+-------------+---------------+---------------+
| result_blob | result_string |  result_array |
+-------------+---------------+---------------+
|             | I love Tengri | {T,e,n,g,r,i} |
+-------------+---------------+---------------+
```

Note that the values in the `result_blob` column are not displayed in the output (because they are of type `BLOB`).

Using the `DESCRIBE` expression, output the data types for all columns in the table created in the same way as in the previous example:

```
CREATE TABLE concat AS

SELECT
    '\xAA'::BLOB || '\xff'::BLOB as result_blob,
    'I' || ' ' || 'love' || ' ' || 'Tengri' as result_string,
    ['T', 'e'] || ['n', 'g', 'r', 'i'] as result_array;

DESCRIBE TABLE concat;
```

```
+---------------+-------------+------+------+---------+-------+
|  column_name  | column_type | null |  key | default | extra |
+---------------+-------------+------+------+---------+-------+
| result_blob   | BLOB        | YES  | null | null    | null  |
+---------------+-------------+------+------+---------+-------+
| result_string | VARCHAR     | YES  | null | null    | null  |
+---------------+-------------+------+------+---------+-------+
| result_array  | VARCHAR[]   | YES  | null | null    | null  |
```

```
+--------------+------------+------+------+---------+-------+
```

Functions for regular expressions

Functions for working with text strings using regular expressions. Regular expressions use the syntax RE2.

`regexp_extract()`

Description Extracts a substring from a string using the given regular expression.

Usage `regexp_extract(string, regexp)`

If the string `string` contains a substring overlaid by the regular expression pattern `regexp`, returns that substring. If there are several substrings covered by the pattern, it returns the first one. If no such substring is found, returns an empty string.

▼ *See examples*

```
SELECT
    regexp_extract('Tengri', '..') AS result_1,
    regexp_extract('Tengri', 'n.*') AS result_2,
    regexp_extract('Tengri', '.*') AS result_3,
    regexp_extract('Tengri', '^.{5}') AS result_4,
    regexp_extract('Tengri', '[TNGRi]$') AS result_5,
    regexp_extract('Tengri', '.{7}') AS result_empty;
```

```
+----------+----------+----------+----------+----------+--------------+
| result_1 | result_2 | result_3 | result_4 | result_5 | result_empty |
+----------+----------+----------+----------+----------+--------------+
| Te       | ngri     | Tengri   | Tengr    | i        |              |
+----------+----------+----------+----------+----------+--------------+
```

`regexp_extract_all()`

Description Extracts all non-overlapping substrings from a string using the given regular expression. Returns an array of substrings.

Usage `regexp_extract_all(string, regexp[, <num>])`

If no substrings in `string` covered by the `regexp` pattern are found, returns an empty list.

Parameters

- `<num>` — the number of the group within the pattern to return (for each substring). By default (if no parameter is given), the entire substring is returned. Group numbering starts with `1`. Groups in the template are highlighted with parentheses.

  ▼ *More details about the parameter on examples*

Let's extract from the text all combinations of letters with a dot after them:

```
SELECT
    regexp_extract_all('My name is Tengri. My nickname is TNGRi.',
                       '(\w+)(\.)')
AS result;
```

```
+------------------+
|      result      |
+------------------+
| {Tengri.,TNGRi.} |
+------------------+
```

Now let's extract from the same text using the same regular expression the combinations of letters before the dot, but without the dot itself. For this purpose, let's set the group number `1`:

```
SELECT
    regexp_extract_all('My name is Tengri. My nickname is TNGRi.',
                       '(\w+)(\.)',
                       1)
AS result;
```

```
+----------------+
|     result     |
+----------------+
| {Tengri,TNGRi} |
+----------------+
```

▼ *See more examples*

```
SELECT
    regexp_extract_all('My name is Tengri. My nickname is TNGRi.', '\w+')
AS words;
```

```
+----------------------------------------+
|                 words                  |
+----------------------------------------+
| {My,name,is,Tengri,My,nickname,is,TNGRi} |
+----------------------------------------+
```

We use a combination of the `regexp_extract_all` and `unnest` functions to extract data from structured text:

```
CREATE TABLE text_table(text_data VARCHAR);
```

```
INSERT INTO text_table VALUES
('Name: "Tengri", Nickname:"TNGRi"'),
('Country: "Russia", Capital:"Moscow"');

SELECT
    unnest(
        regexp_extract_all(text_data, '(\w+):\s*"(.*?)"', 1)
          ) AS key,
    unnest(
        regexp_extract_all(text_data, '(\w+):\s*"(.*?)"', 2)
          ) AS value
FROM text_table;
```

```
+----------+--------+
|    key   | value  |
+----------+--------+
| Name     | Tengri |
+----------+--------+
| Nickname | TNGRi  |
+----------+--------+
| Country  | Russia |
+----------+--------+
| Capital  | Moscow |
+----------+--------+
```

`regexp_full_match()`

Description Checks whether a regular expression overlaps a string completely.

Usage `regexp_full_match(string, regexp)`

If the regular expression pattern `regexp` completely overlaps the string `string`, returns `true`, otherwise — `false`.

▼ *See examples*

```
SELECT
    regexp_full_match('Tengri', 'gri$')     AS result_1, -- false expected
    regexp_full_match('Tengri', '.')        AS result_2, -- false expected
    regexp_full_match('Tengri', '.*')       AS result_3, -- true expected
    regexp_full_match('Tengri', '^\w+gri$') AS result_4; -- true expected
```

```
+----------+----------+----------+----------+
| result_1 | result_2 | result_3 | result_4 |
+----------+----------+----------+----------+
| false    | false    | true     | true     |
+----------+----------+----------+----------+
```

`regexp_matches()`

Description Checks if a regular expression is contained within a string.

Usage `regexp_matches(string, regexp)`

If at least one substring covered by the `regexp` pattern is found inside the string `string`, returns `true`, otherwise — `false`.

▼ *See examples*

```
SELECT
    regexp_matches('Tengri', '.+T') AS result_1, -- false expected
    regexp_matches('Tengri', '.*T') AS result_2; -- true expected
```

```
+----------+----------+
| result_1 | result_2 |
+----------+----------+
| false    | true     |
+----------+----------+
```

`regexp_replace()`

Description Replaces a substring covered by a regular expression with the specified string.

Usage `regexp_replace(string, regexp, target)`

If a substring covered by the `regexp` pattern is found inside the string `string`, it is replaced by the string `target`. If several such substrings are found, only the first one is replaced. If no substring is found, the original string `string` is returned.

▼ *See examples*

```
SELECT
    regexp_replace('Tengri', '.', 't') AS result_1,
    regexp_replace('Tengri', '.*', 't') AS result_2,
    regexp_replace('Tengri', 'e.*r', 'NGR') AS result_3,
    regexp_replace('Tengri', 'a', 't') AS result_4;
```

```
+----------+----------+----------+----------+
| result_1 | result_2 | result_3 | result_4 |
+----------+----------+----------+----------+
| tengri   | t        | TNGRi    | Tengri   |
+----------+----------+----------+----------+
```

`regexp_split_to_array()`

Description Splits a string into parts, separated by a regular expression, and returns the parts as an array.

Usage	`regexp_split_to_array(string, regexp)`

Aliases `string_split_regex()`

If substrings overlapped by the `regexp` pattern are found in the string `string`, then the parts of the original string not overlapped by the pattern are returned as an array. If the found substrings are at the beginning or at the end of the source string, the resulting array will contain empty strings for the beginning and the end of the string. If no substrings are found, an array of one source string will be returned.

▼ *See examples*

```
SELECT
    string_split_regex('My name is Tengri. My nickname is TNGRi.', '\.\s')
AS sentences,
    string_split_regex('My name is Tengri. My nickname is TNGRi.', '[\.\s]+')
AS words;
```

```
+-----------------------------------------+-----------------------------------------+
|                 sentences               |                  words                  |
+-----------------------------------------+-----------------------------------------+
| {My name is Tengri,My nickname is TNGRi.} | {My,name,is,Tengri,My,nickname,is,TNGRi,} |
+-----------------------------------------+-----------------------------------------+
```

`regexp_split_to_table()`

Description Splits a string into parts separated by a regular expression, and returns the parts as strings.

Usage `regexp_split_to_table(string, regexp)`

If substrings overlapped by the `regexp` pattern are found in the string `string`, then the parts of the original string not overlapped by the pattern are returned as a column, with each part written to a different cell. If the found substrings are at the beginning or end of the source string, the result will include empty strings for the beginning and end of the string. If no substrings are found, a column with one cell in which the original string is written will be returned.

▼ *See examples*

```
SELECT
    regexp_split_to_table('My name is Tengri. My nickname is TNGRi.', '\.\s')
AS sentences,
    regexp_split_to_table('My name is Tengri. My nickname is TNGRi.', '[\.\s]+')
AS words;
```

```
+----------------------+----------+
|       sentences      |   words  |
+----------------------+----------+
| My name is Tengri    | My       |
+----------------------+----------+
| My nickname is TNGRi. | name     |
+----------------------+----------+
```

```
| null                  | is       |
+-----------------------+----------+
| null                  | Tengri   |
+-----------------------+----------+
| null                  | My       |
+-----------------------+----------+
| null                  | nickname |
+-----------------------+----------+
| null                  | is       |
+-----------------------+----------+
| null                  | TNGRi    |
+-----------------------+----------+
| null                  |          |
+-----------------------+----------+
```

Operator `~`

Description Checks if the regular expression covers the string completely.

Usage `string ~ regexp`.

Can be used with the negation operator `!`: `string !~ regexp`.

Fully synonymous with the `SIMILAR TO` operator.

▼ *See examples*

```
SELECT
    'Tengri' !~ 'TNGRi' AS result_1,
    'Tengri' ~ 'T.*'    AS result_2,
    'Tengri' ~ 'T.+i'   AS result_3,
    'TNGRi'  ~ 'T.+i'   AS result_4,
    'T.+i'   ~ 'T.+i'   AS result_5;
```

```
+----------+----------+----------+----------+----------+
| result_1 | result_2 | result_3 | result_4 | result_5 |
+----------+----------+----------+----------+----------+
| true     | true     | true     | true     | true     |
+----------+----------+----------+----------+----------+
```

Functions for date and time

Functions for date and time are functions for working with data of types `DATE`, `TIME`, `TIMESTAMP` and `TIMESTAMPTZ`.

`current_time()`

Description Returns the current time as a value of type `TIME`.

Usage `current_time` or `current_time()`

Aliases `get_current_time()`

▼ *See example*

```
SELECT
    current_time       AS cur_time_1,
    current_time()     AS cur_time_2,
    get_current_time() AS cur_time_3;
```

```
+-----------------+-----------------+-----------------+
|    cur_time_1   |    cur_time_2   |    cur_time_3   |
+-----------------+-----------------+-----------------+
| 10:33:24.016000 | 10:33:24.016000 | 10:33:24.016000 |
+-----------------+-----------------+-----------------+
```

`now()`

Description Returns the current time as a value of type `TIMESTAMPTZ`

Usage `now()`

▼ *See example*

```
SELECT
    now() AS cur_time;
```

```
+---------------------------------+
| cur_time                        |
+---------------------------------+
| 2025-08-26 13:38:58.461000+00:00 |
+---------------------------------+
```

`datepart()`

Description Returns the specified part of the date or time value as a value of type `BIGINT`.

Usage `datepart('<part>', (TIME | DATE | ...) <date_time>)`.

Aliases `date_part()`

Arguments can be values of types: `TIME`, `DATE`, `TIMESTAMP`, or `TIMESTAMPTZ`.

▼ *Parts can be specified using the following literals*

- `century`
- `day`
- `decade`

- `hour`
- `microseconds`
- `millennium`
- `milliseconds`
- `minute`
- `month`
- `quarter`
- `second`
- `year`

▼ *See examples*

```
SELECT
    datepart('milliseconds', TIMESTAMP '2025-02-25 00:00:00.1') AS milliseconds,
    datepart('hour',         TIME '2025-02-25 00:00:00')        AS hour,
    datepart('millennium',   DATE '2025-02-25')                 AS millennium;
```

```
+--------------+------+------------+
| milliseconds | hour | millennium |
+--------------+------+------------+
| 100          | 0    | 3          |
+--------------+------+------------+
```

`date_diff()`

Description Returns the number of time units between two points in time as a value of type `BIGINT`.

Usage `date_diff('<part>', start, end)`

Arguments can be values of type: `TIME`, `DATE`, `TIMESTAMP`, or `TIMESTAMPTZ`.

▼ *Units can be specified using the following literals*

- `century`
- `day`
- `decade`
- `hour`
- `microseconds`
- `millennium`
- `milliseconds`
- `minute`
- `month`
- `quarter`

- `second`
- `year`

▼ *See examples*

```
SELECT
    date_diff('second', TIME '01:02:03', TIME '03:02:01') AS diff_in_seconds,
    date_diff('minute', TIME '01:02:03', TIME '03:02:01') AS diff_in_minutes,
    date_diff('hour',   TIME '01:02:03', TIME '03:02:01') AS diff_in_hours,
    date_diff('day',    TIMESTAMP '2025-02-25 01:02:03', TIMESTAMP '2025-02-26 03:02:01')
AS diff_in_days,
    date_diff('day',    TIMESTAMP '2025-02-26 01:02:03', TIMESTAMP '2025-02-25 03:02:01')
AS diff_in_days,
    date_diff('day',    DATE '2024-02-27', DATE '2025-02-27') AS diff_in_days;
```

```
+-----------------+-----------------+--------------+-------------+--------------
+--------------+
| diff_in_seconds | diff_in_minutes | diff_in_hours | diff_in_days | diff_in_days |
diff_in_days |
+-----------------+-----------------+--------------+-------------+--------------
+--------------+
| 7198            | 120             | 2             | 1            | -1           | 366
|
+-----------------+-----------------+--------------+-------------+--------------
+--------------+
```

`date_trunc()`

Description	Reduces a moment in time to the specified precision.
Usage	`date_trunc('<part>', <time_stamp>)`

Reduces the `TIMESTAMP` time moment to the specified precision unit and returns the initial time moment for that unit as a value of type `TIMESTAMP` or `DATE`.

Some usage examples are described in this script.

▼ *Units can be specified using the following literals*

- `century`
- `day`
- `decade`
- `hour`
- `microseconds`
- `millennium`
- `milliseconds`
- `minute`

- `month`
- `quarter`
- `second`
- `year`

▼ *See examples*

```
SELECT
    date_trunc('minute',    TIMESTAMP '2025-02-25 01:02:03') AS minute,
    date_trunc('hour',      TIMESTAMP '2025-02-25 01:02:03') AS hour,
    date_trunc('day',       TIMESTAMP '2025-02-25 01:02:03') AS day,
    date_trunc('month',     TIMESTAMP '2025-02-25 01:02:03') AS month,
    date_trunc('quarter',   TIMESTAMP '2025-02-25 01:02:03') AS quarter,
    date_trunc('year',      TIMESTAMP '2025-02-25 01:02:03') AS year;
```

```
+---------------------+---------------------+------------+------------+------------
+------------+
| minute              | hour                | day        | month      | quarter    | year
|
+---------------------+---------------------+------------+------------+------------
+------------+
| 2025-02-25 01:02:00 | 2025-02-25 01:00:00 | 2025-02-25 | 2025-02-01 | 2025-01-01 | 2025-
01-01 |
+---------------------+---------------------+------------+------------+------------
+------------+
```

`strptime()`

Description Converts text to a point in time using the specified format.

Usage `strptime(<string>, <format>)`

Converts text to the point in time `TIMESTAMP` using the specified format. If the conversion fails, it generates an error.

Some usage examples are described in this script.

▼ *Format can be specified using the following expressions*

Expression	Description	Example
`%a`	Abbreviated weekday name.	*Sun, Mon, ...*
`%A`	Full weekday name.	*Sunday, Monday, ...*
`%b`	Abbreviated month name.	*Jan, Feb, ..., Dec*
`%B`	Full month name.	*January, February, ...*
`%c`	ISO date and time representation	*1992-03-02 10:30:20*
`%d`	Day of the month as a zero-padded decimal.	*01, 02, ..., 31*

Expression	Description	Example
`%-d`	Day of the month as a decimal number.	*1, 2, ..., 30*
`%f`	Microsecond as a decimal number, zero-padded on the left.	*000000 - 999999*
`%g`	Millisecond as a decimal number, zero-padded on the left.	*000 - 999*
`%G`	ISO 8601 year with century representing the year that contains the greater part of the ISO week (see `%V`).	*0001, 0002, ..., 2013, 2014, ..., 9998, 9999*
`%H`	Hour (24-hour clock) as a zero-padded decimal number.	*00, 01, ..., 23*
`%-H`	Hour (24-hour clock) as a decimal number.	*0, 1, ..., 23*
`%I`	Hour (12-hour clock) as a zero-padded decimal number.	*01, 02, ..., 12*
`%-I`	Hour (12-hour clock) as a decimal number.	*1, 2, ... 12*
`%j`	Day of the year as a zero-padded decimal number.	*001, 002, ..., 366*
`%-j`	Day of the year as a decimal number.	*1, 2, ..., 366*
`%m`	Month as a zero-padded decimal number.	*01, 02, ..., 12*
`%-m`	Month as a decimal number.	*1, 2, ..., 12*
`%M`	Minute as a zero-padded decimal number.	*00, 01, ..., 59*
`%-M`	Minute as a decimal number.	*0, 1, ..., 59*
`%n`	Nanosecond as a decimal number, zero-padded on the left.	*000000000 - 999999999*
`%p`	Locale's AM or PM.	*AM, PM*
`%S`	Second as a zero-padded decimal number.	*00, 01, ..., 59*
`%-S`	Second as a decimal number.	*0, 1, ..., 59*
`%u`	ISO 8601 weekday as a decimal number where 1 is Monday.	*1, 2, ..., 7*
`%U`	Week number of the year. Week 01 starts on the first Sunday of the year, so there can be week 00. Note that this is not compliant with the week date standard in ISO-8601.	*00, 01, ..., 53*
`%V`	ISO 8601 week as a decimal number with Monday as the first day of the week. Week 01 is the week containing Jan 4. Note that `%V` is incompatible with year directive `%Y`. Use the ISO year `%G` instead.	*01, ..., 53*
`%w`	Weekday as a decimal number.	*0, 1, ..., 6*
`%W`	Week number of the year. Week 01 starts on the first Monday of the year, so there can be week 00. Note that this is not compliant with the week date standard in ISO-8601.	*00, 01, ..., 53*
`%x`	ISO date representation	*1992-03-02*
`%X`	ISO time representation	*10:30:20*
`%y`	Year without century as a zero-padded decimal number.	*00, 01, ..., 99*
`%-y`	Year without century as a decimal number.	*0, 1, ..., 99*
`%Y`	Year with century as a decimal number.	*2013, 2019 etc.*
`%z`	Time offset from UTC in the form ±HH:MM, ±HHMM, or ±HH.	*-0700*

Expression	Description	Example
`%Z`	Time zone name.	*Europe/Amsterdam*
`%%`	A literal `%` character.	*%*

▼ *See examples*

```
SELECT
    strptime('Feb 25, 2025, 01:02:03 AM', '%b %d, %Y, %H:%M:%S %p') AS timestamp_1,
    strptime('2025-02-25, 01:02:03',      '%x, %X')                 AS timestamp_2,
    strptime('01:02, 01.02.99',           '%H:%M, %d.%m.%y')        AS timestamp_3,
    strptime('01:02%PM--01.02.99',        '%H:%M%%p--%d.%m.%y')     AS timestamp_4,
    strptime('1:1, 1.1.1',                '%-H:%-M, %-d.%-m.%-y')   AS timestamp_5,
    strptime('1',                         '%Y')                     AS timestamp_6;
```

```
+---------------------+---------------------+---------------------+---------------------
+---------------------+---------------------+
| timestamp_1         | timestamp_2         | timestamp_3         | timestamp_4         |
timestamp_5         | timestamp_6         |
+---------------------+---------------------+---------------------+---------------------
+---------------------+---------------------+
| 2025-02-25 01:02:03 | 2025-02-25 01:02:03 | 1999-02-01 01:02:00 | 1999-02-01 13:02:00 |
2001-01-01 01:01:00 | 0001-01-01 00:00:00 |
+---------------------+---------------------+---------------------+---------------------
+---------------------+---------------------+
```

Operator `+`

Description Adds the right argument to the left argument.

Usage `<num> + <num> [+ ...]` or `TIME + INTERVAL [+ ...]`.

If used with types for for date and time, adds the interval to the time value. Returns a value of type `TIME`.

See also `add()`.

▼ *See examples*

```
SELECT
    3 + 2      AS result_1,
    3 + 2 + -1 AS result_2,
    1.1 + 1.9  AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 5        | 4        | 3.0      |
+----------+----------+----------+
```

```
SELECT
    TIME '12:11:10'     + INTERVAL 3 hours AS result_time_1,
    INTERVAL '12:11:10' + TIME '1:1:1'     AS result_time_2;
```

```
+---------------+---------------+
| result_time_1 | result_time_2 |
+---------------+---------------+
| 15:11:10      | 13:12:11      |
+---------------+---------------+
```

Operator -

Description Subtracts the right argument from the left argument.

Usage `<num> - <num> [-...]` or `TIME - INTERVAL [-...]`.

If used with types for for date and time, subtracts the interval from the time value. Returns a value of type `TIME`.

See also `subtract()`.

▼ *See examples*

```
SELECT
    3 - 2      AS result_1,
    3 - 2 - +1 AS result_2,
    1.2 - 0.2  AS result_3;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
+----------+----------+----------+
| 1        | 0        | 1.0      |
+----------+----------+----------+
```

```
SELECT
    TIME '12:11:10' - INTERVAL 3 HOUR AS result_time_1,
    TIME '12:11:10' - INTERVAL 3 HOUR - INTERVAL 1 HOUR AS result_time_2;
```

```
+---------------+---------------+
| result_time_1 | result_time_2 |
+---------------+---------------+
| 09:11:10      | 08:11:10      |
+---------------+---------------+
```

Functions for JSON

Functions for working with `.json` extension files and with data type `JSON`.

Path inside JSON structure

Many functions for JSON use the path inside the JSON structure as one of the arguments. The path can be specified in either of two notations according to the following standards:

- JSONPath
  - `$.key1.key2` — accessing the value of key `key2`
  - `$.key1.key2[i]` — accessing the `i`-th element of the list in the value of the `key2` key
- JSON Pointer
  - `/key1/key2` — accessing the value of key `key2`
  - `/key1/key2/i` — accessing the `i`-th element of the list at the value of key `key2`

The numbering of list items in JSON structure starts with `0`.

The examples below use the JSON structure from this example:

```
CREATE TABLE js_table(js_data JSON);

INSERT INTO js_table VALUES
('{
  "first_name": "John",
  "last_name": "Smith",
  "is_alive": true,
  "age": 27,
  "address": {
    "street_address": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postal_code": "10021-3100"
  },
  "phone_numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
```

```
    "Thomas",
    "Trevor"
  ],
  "spouse": null
}');
```

`json_array_length()`

Description Returns the number of elements in the array at the specified path in JSON, or `0` if the specified path is not an array.

Usage `json_array_length(json[, path])`

If a list of paths is given in the second argument, the result is a list of array lengths on the specified paths.

▼ *See examples*

```
SELECT
    json_array_length(js_data, '$.phone_numbers')                AS phone_numbers,
    json_array_length(js_data, '$.children')                     AS children,
    json_array_length(js_data, ['$.phone_numbers', '$.children']) AS lists,
    json_array_length(js_data)                                   AS js_data,
    json_array_length(js_data, '$.first_name')                   AS first_name,
FROM js_table;
```

```
+---------------+----------+-------+---------+------------+
| phone_numbers | children | lists | js_data | first_name |
+---------------+----------+-------+---------+------------+
| 2             | 3        | {2,3} | 0       | 0          |
+---------------+----------+-------+---------+------------+
```

`json_contains()`

Description Checks if the JSON structure contains the JSON substructure specified in the second argument.

Usage `json_contains(json, json)`

Both arguments must be of type `JSON`. The second argument can be a numeric value or a text string, and the text string must be enclosed in double quotes.

▼ *See examples*

```
SELECT
    json_contains(js_data, '27')                    AS result_1,
    json_contains(js_data, '"John"')                AS result_2,
    json_contains(js_data, '{"first_name": "John"}')  AS result_3,
    json_contains(js_data, '{"first_name": "Smith"}') AS result_4, -- false expected
FROM js_table;
```

```
+----------+----------+----------+----------+
| result_1 | result_2 | result_3 | result_4 |
+----------+----------+----------+----------+
| true     | true     | true     | false    |
+----------+----------+----------+----------+
```

`json_exists()`

Description Checks if the JSON structure contains the specified path.

Usage `json_exists(json, path)`

Returns `BOOL` or an array of `BOOL[]` for cases where path points to list items in the JSON structure.

▼ *See examples*

```
SELECT
    json_exists(js_data, 'first_name')              AS result_1,
    json_exists(js_data, '$.first_name')            AS result_2,
    json_exists(js_data, '/first_name')             AS result_3,
    json_exists(js_data, '$.address.street_address') AS result_4,
    json_exists(js_data, '/address/street_address')  AS result_5,
    json_exists(js_data, '$.street_address')         AS result_6, -- false expected
    json_exists(js_data, '$.phone_numbers[*].type')  AS result_7,
    json_exists(js_data, '$..street_address')        AS result_8,
FROM js_table;
```

```
+----------+----------+----------+----------+----------+----------+-------------+----------
+
| result_1 | result_2 | result_3 | result_4 | result_5 | result_6 | result_7    | result_8
|
+----------+----------+----------+----------+----------+----------+-------------+----------
+
| true     | true     | true     | true     | true     | false    | {True,True} | {True}
|
+----------+----------+----------+----------+----------+----------+-------------+----------
+
```

`json_extract()`

Description Extracts data from a JSON structure at the specified path. Returns the data as `JSON`.

Usage `json_extract(json, path)` or `json_extract(json, [path1, path2, {⋯])`.

Aliases `json_extract_path`.

If a list of paths is given as the second argument, returns a list of values.

▼ *See examples*

```
SELECT
    json_extract(js_data, 'first_name')                AS first_name,
    json_extract(js_data, ['first_name', 'last_name']) AS all_names,
    json_extract(js_data, '$.address.street_address')  AS street_address,
    json_extract(js_data, '$.phone_numbers[*].number') AS phone_numbers,
    json_extract(js_data, '$.children[0]')             AS child_1,
FROM js_table;
```

```
+------------+------------------+---------------+--------------------------------
+-----------+
| first_name | all_names        | street_address | phone_numbers                   |
child_1   |
+------------+------------------+---------------+--------------------------------
+-----------+
| John       | {"John","Smith"} | 21 2nd Street  | {"212 555-1234","646 555-4567"} |
Catherine |
+------------+------------------+---------------+--------------------------------
+-----------+
```

`json_extract_string()`

Description Extracts data from a JSON structure at the specified path. Returns the data in `VARCHAR` form.

Usage `json_extract_string(json, path)` or `json_extract_string(json, [path1, path2, ...])`.

Aliases `json_extract_path_text`.

If a list of paths is given as the second argument, returns a list of values.

▼ *See examples*

```
SELECT
    json_extract_string(js_data, 'first_name')                AS first_name,
    json_extract_string(js_data, ['first_name', 'last_name']) AS all_names,
    json_extract_string(js_data, '$.address.street_address')  AS street_address,
    json_extract_string(js_data, '$.phone_numbers[*].number') AS phone_numbers,
    json_extract_string(js_data, '$.children[0]')             AS child_1,
FROM js_table;
```

```
+------------+-------------+---------------+-----------------------------+-----------+
| first_name | all_names    | street_address | phone_numbers               | child_1   |
+------------+-------------+---------------+-----------------------------+-----------+
| John       | {John,Smith} | 21 2nd Street  | {212 555-1234,646 555-4567} | Catherine |
+------------+-------------+---------------+-----------------------------+-----------+
```

`json_group_array()`

Description Returns a JSON list containing all the values of a column.

Usage `json_group_array(argument)`

The function changes the cardinality of the data.

▼ *See examples*

```
CREATE TABLE example (name VARCHAR);
INSERT INTO example VALUES ('Tengri'), ('TNGRi');

SELECT json_group_array(name) AS tengti_names
FROM example;
```

```
+--------------------+
| tengti_names       |
+--------------------+
| ["Tengri","TNGRi"] |
+--------------------+
```

`json_group_object()`

Description Returns a JSON structure containing all `key-value` pairs from the columns specified in the arguments.

Usage `json_group_object(argument1, argument2)`

The function modifies the cardinality of the data.

▼ *See examples*

```
CREATE TABLE example (name VARCHAR, letters_num BIGINT);
INSERT INTO example VALUES
    ('Tengri', 6),
    ('TNGRi', 5);

SELECT json_group_object(name, letters_num) AS result
FROM example;
```

```
+-----------------------+
| result                |
+-----------------------+
| {"Tengri":6,"TNGRi":5} |
+-----------------------+
```

json_keys()

Description Returns all keys from the specified JSON structure as `VARCHAR[]`.

Usage `json_keys(json[, path])`

If path is specified in the second argument, returns the keys of the JSON structure at the specified path. If a list of paths is specified, the result is a list of lists of keys.

▼ *See examples*

```
SELECT
    json_keys(js_data) AS all_keys,
FROM js_table;
```

```
+-------------------------------------------------------------------------+
| all_keys                                                                |
+-------------------------------------------------------------------------+
| {first_name,last_name,is_alive,age,address,phone_numbers,children,spouse} |
+-------------------------------------------------------------------------+
```

```
SELECT
    json_keys(js_data, '$.address') AS address_keys,
FROM js_table;
```

```
+---------------------------------------+
| address_keys                          |
+---------------------------------------+
| {street_address,city,state,postal_code} |
+---------------------------------------+
```

```
SELECT
    json_keys(js_data, ['$.address',
                        '$.phone_numbers[0]'])
    AS address_and_phone_keys,
FROM js_table;
```

```
+-----------------------------------------------------------------------------+
| address_and_phone_keys                                                      |
+-----------------------------------------------------------------------------+
| {['street_address', 'city', 'state', 'postal_code'],['type', 'number']} |
+-----------------------------------------------------------------------------+
```

`json_transform()`

Description Transforms the JSON structure according to the specified structure.

Usage `json_transform(json, string)`

Aliases `from_json()`

The target structure is specified as a text string in the second argument.

In cases of missing values in the source JSON structure, `NULL` values are put into the result.

In cases of impossibility to convert data types in the source structure to the specified ones, `NULL` value is set.

▼ *See examples*

```
CREATE TABLE example (js_data JSON);
INSERT INTO example VALUES
    ('{
        "first_name": "John",
        "last_name": "Smith",
        "age": 27
      }'),
    ('{
        "first_name": "John",
        "is_alive": true,
        "age": 28
      }');

SELECT
    json_transform(js_data,
        '{
        "first_name": "VARCHAR",
        "last_name": "VARCHAR",
        "is_alive": "BOOL",
        "age": "BIGINT"
        }')
    AS result
FROM example;
```

```
+-------------------------------------------------------------------------+
| result                                                                  |
+-------------------------------------------------------------------------+
| {"first_name": "John", "last_name": "Smith", "is_alive": null, "age": 27} |
+-------------------------------------------------------------------------+
| {"first_name": "John", "last_name": null, "is_alive": true, "age": 28}    |
+-------------------------------------------------------------------------+
```

`json_transform_strict()`

Description Transforms a JSON structure to match the specified structure. Issues an error if

structures or types do not match.

Usage	`json_transform_strict(json, string)`.
Aliases	`from_json_strict()`

The target structure is specified as a text string in the second argument.

In cases where there are no values from the specified structure in the source JSON structure, produces an error.

If it is impossible to convert data types in the source structure to the specified ones, it generates an error.

▼ *See examples*

```
CREATE TABLE example (js_data JSON);
INSERT INTO example VALUES
    ('{
        "first_name": "John",
        "last_name": "Smith",
        "age": 27
      }'),
    ('{
        "first_name": "John",
        "is_alive": true,
        "age": 28
      }');
SELECT
    json_transform_strict(js_data,
        '{"first_name": "VARCHAR", "age": "BIGINT"}')
    AS result
FROM example;
```

```
+----------------------------------+
| result                           |
+----------------------------------+
| {"first_name": "John", "age": 27} |
+----------------------------------+
| {"first_name": "John", "age": 28} |
+----------------------------------+
```

```
CREATE TABLE example (js_data JSON);
INSERT INTO example VALUES
    ('{
        "first_name": "John",
        "last_name": "Smith",
        "age": 27
      }'),
    ('{
        "first_name": "John",
        "is_alive": true,
        "age": 28
      }');
```

```
SELECT
    json_transform_strict(js_data,
        '{"first_name": "BIGINT", "age": "BIGINT"}') -- error expected
    AS result
FROM example;
```

```
ERROR: InvalidInputException: Invalid Input Error:
Failed to cast value to numerical: "John"
```

```
CREATE TABLE example (js_data JSON);
INSERT INTO example VALUES
    ('{
        "first_name": "John",
        "last_name": "Smith",
        "age": 27
      }'),
    ('{
        "first_name": "John",
        "is_alive": true,
        "age": 28
      }');

SELECT
    json_transform_strict(js_data,
        '{"first_name": "VARCHAR", "last_name": "VARCHAR"}') -- error expected
    AS result
FROM example;
```

```
ERROR: InvalidInputException: Invalid Input Error:
Object {"first_name":"John","is_alive":true,"age":28} does not have key "last_name"
```

`json_valid()`

Description Checks if the argument is a valid JSON structure.

Usage `json_valid(json)`

▼ *See examples*

```
SELECT
    json_valid(js_data)                  AS result_1,
    json_valid('{"first_name": "John"}') AS result_2,
    json_valid('{"first_name"}')         AS result_3, -- false expected
FROM js_table;
```

```
+----------+----------+----------+
| result_1 | result_2 | result_3 |
```

```
+----------+----------+----------+
| true     | true     | false    |
+----------+----------+----------+
```

`json_value()`

Description Retrieves values from a JSON structure at the specified path.

Usage `json_value(json, path)`

If the value is not scalar (list or nested structure) at the specified path, returns `NULL`.

▼ *See examples*

```
SELECT
    json_value(js_data, 'first_name')      AS first_name,
    json_value(js_data, '$.phone_numbers[*].number') AS phone_numbers,
    json_value(js_data, '$.children[0]')   AS child_1,
    json_value(js_data, '$.phone_numbers') AS phone_numbers, -- NULL expected
    json_value(js_data, '$.address')       AS address,       -- NULL expected
FROM js_table;
```

```
+------------+---------------------------------+-----------+---------------+---------+
| first_name | phone_numbers                   | child_1   | phone_numbers | address |
+------------+---------------------------------+-----------+---------------+---------+
| John       | {"212 555-1234","646 555-4567"} | Catherine | null          | null    |
+------------+---------------------------------+-----------+---------------+---------+
```

`json()`

Description Shortens the JSON structure record (removes spaces and line breaks).

Usage `json(json)`.

▼ *See examples*

```
SELECT
    json('{
            "first_name": "John",
            "last_name": "Smith"
          }
        ') AS result_1,
    json(js_data) AS result_2,
FROM js_table;
```

```
+-----------------------------------------------------
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------------
------------------------+
| result_1                                              | result_2
|
+-----------------------------------------------------
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------+
| "{""first_name"":""John"",""last_name"":""Smith""}" |
"{""first_name"":""John"",""last_name"":""Smith"",""is_alive"":true,""age"":27,""address"":
{""street_address"":""21 2nd Street"",""city"":""New
York"",""state"":""NY"",""postal_code"":""10021-
3100""},""phone_numbers"":[{""type"":""home"",""number"":""212 555-
1234""},{""type"":""office"",""number"":""646 555-
4567""}],""children"":[""Catherine"",""Thomas"",""Trevor""],""spouse"":null}" |
+-----------------------------------------------------
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------+
```

`read_json()`

Description	reads a `.json` file and writes the read data to a table.
Usage	`read_json(filename[, columns = {column_name: 'column_type', ...}])`
Aliases	`read_json_auto()`

The data types for the columns are automatically determined.

Parameters

- `columns` — optional parameter, where you can specify column names and types. In this case, only the specified columns will be added to the resulting table.

More details on loading data into Tengri are described on the page Data loading.

The examples use file `tengri_data_types.json` with data types Tengri and their brief descriptions and file `json_example_from_wikipedia.json` with this example.

▼ *See examples*

Let's read the file `tengri_data_types.json` and output the first five rows of the table:

```
SELECT *
FROM read_json(
    '<path>/tengri_data_types.json'
```

```
)
LIMIT 5
```

```
+----------+-----------+----------+--------------------------+
| name     | type      | category | description              |
+----------+-----------+----------+--------------------------+
| BIGINT   | data type | numeric  | Целые числа.              |
+----------+-----------+----------+--------------------------+
| BIGINT[] | data type | array    | Массивы целых чисел.      |
+----------+-----------+----------+--------------------------+
| BLOB     | data type | blob     | Двоичные объекты.         |
+----------+-----------+----------+--------------------------+
| BOOL     | data type | boolean  | Булевы значения.          |
+----------+-----------+----------+--------------------------+
| BOOL[]   | data type | array    | Массивы булевых значений.  |
+----------+-----------+----------+--------------------------+
```

Let's read the file `tengri_data_types.json`, set only the required columns (the keys of the source file) and output the first five rows of the table:

```
SELECT *
FROM read_json(
    '<path>/tengri_data_types.json',
    columns = {name: 'VARCHAR', category: 'VARCHAR'})
LIMIT 5
```

```
+----------+----------+
| name     | category |
+----------+----------+
| BIGINT   | numeric  |
+----------+----------+
| BIGINT[] | array    |
+----------+----------+
| BLOB     | blob     |
+----------+----------+
| BOOL     | boolean  |
+----------+----------+
| BOOL[]   | array    |
+----------+----------+
```

Let's read the file `json_example_from_wikipedia.json` and output the whole table built on it. Note that nested JSON structures are written to the table cells and are not expanded in any way (columns `address`, `phone_numbers`, `children`). To expand nested structures, you can use the function `unnest`.

```
SELECT *
FROM read_json(
    '<path>/json_example_from_wikipedia.json'
```

```
)
```

```
+------------+-----------+----------+-----
+-----------------------------------------------------------------------------------------
-----------+
-----------------------------------------------------------------------------------------
+--------------------------+--------+
| first_name | last_name | is_alive | age | address
| phone_numbers
| children                  | spouse |
+------------+-----------+----------+-----
+-----------------------------------------------------------------------------------------
-----------+
-----------------------------------------------------------------------------------------
+--------------------------+--------+
| John       | Smith     | true     | 27  | {"street_address": "21 2nd Street", "city":
"New York", "state": "NY", "postal_code": "10021-3100"} | {{'type': 'home', 'number': '212
555-1234'},{'type': 'office', 'number': '646 555-4567'}} | {Catherine,Thomas,Trevor} | null
|
+------------+-----------+----------+-----
+-----------------------------------------------------------------------------------------
-----------+
-----------------------------------------------------------------------------------------
+--------------------------+--------+
```

Functions for binary data

Functions for binary data are functions for working with values of binary type (`BLOB`).

`concat()`

Description Concatenates multiple strings, arrays or binary values.

Usage `concat(argument1, argument2, {⋯})`.

Empty values (`NULL`) are ignored.

See also Operator `||`.

▼ *See example*

```
SELECT
    concat('\xAA'::BLOB, '\xff'::BLOB) as result_blob,
    concat('I', ' ', 'love', ' ', 'Tengri') as result_string,
    concat(['T', 'e'], ['n', 'g', 'r', 'i']) as result_array;
```

```
+-------------+--------------+---------------+
| result_blob | result_string |  result_array |
+-------------+--------------+---------------+
```

```
| \xAA\xFF    | I love Tengri | {T,e,n,g,r,i} |
+-------------+---------------+---------------+
```

`md5()`

Description Returns a hash MD5 of data from `argument` as a string (`VARCHAR`).

Usage `md5(argument)`

The `argument` can be binary data or a string.

▼ *See examples*

```
SELECT
    md5('\xAA\xFF'::BLOB) as md5_hash;
```

```
+----------------------------------+
|             md5_hash             |
+----------------------------------+
| 1fab7f7621f5ddc051ebd1f2c63c4665 |
+----------------------------------+
```

```
SELECT
    md5('Tengri') as md5_hash;
```

```
+----------------------------------+
|             md5_hash             |
+----------------------------------+
| 846b02d31131a10bd6ac0ba189c65bef |
+----------------------------------+
```

`sha1()`

Description Returns a hash SHA-1 of the data from `argument` as a string (`VARCHAR`).

Usage `sha1(argument)`

The `argument` can be binary data or a string.

▼ *See examples*

```
SELECT
    sha1('\xAA\xFF'::BLOB) as sha1_hash;
```

```
+------------------------------------------+
|                 sha1_hash                |
+------------------------------------------+
```

```
| e89b0db325637edfacde04a76005c492e2c5aeca |
+------------------------------------------+
```

```
SELECT
    sha1('Tengri') as sha1_hash;
```

```
+------------------------------------------+
|                 sha1_hash                |
+------------------------------------------+
| b514525a19995a2442d7565bfd9bb42d9dc71a13 |
+------------------------------------------+
```

sha256()

Description Returns a hash SHA-256 of the data from `argument` as a string (`VARCHAR`).

Usage `sha256(argument)`

The `argument` can be binary data or a string.

▼ *See example*

```
SELECT
    sha256('\xAA\xFF'::BLOB) as sha256_hash;
```

```
+------------------------------------------------------------------+
|                           sha256_hash                            |
+------------------------------------------------------------------+
| 768318522cac43261e8ef4946c2296a3643d523a8d5bda8ff5b82aa64470421a |
+------------------------------------------------------------------+
```

```
SELECT
    sha256('Tengri') as sha256_hash;
```

```
+------------------------------------------------------------------+
|                           sha256_hash                            |
+------------------------------------------------------------------+
| 8aaacef66663b14ee7c5a03dbaec7b40f0f3bf17bd12d2ed4f9aaad0e10a0d77 |
+------------------------------------------------------------------+
```

Operator ||

Description Concatenates multiple strings, arrays or binary values.

Usage	`argument1 || argument2 || argument2 || ...`.

Empty values (`NULL`) are ignored.

See also concat().

▼ *See examples*

```
SELECT
    '\xAA'::BLOB || '\xff'::BLOB as result_blob,
    'I' || ' ' || 'love' || ' ' || 'Tengri' as result_string,
    ['T', 'e'] || ['n', 'g', 'r', 'i'] as result_array;
```

```
+-------------+---------------+---------------+
| result_blob | result_string |  result_array |
+-------------+---------------+---------------+
|             | I love Tengri | {T,e,n,g,r,i} |
+-------------+---------------+---------------+
```

Note that the values in the `result_blob` column are not displayed in the output (because they are of type `BLOB`).

Using the `DESCRIBE` expression, output the data types for all columns in the table created in the same way as in the previous example:

```
CREATE TABLE concat AS

SELECT
    '\xAA'::BLOB || '\xff'::BLOB as result_blob,
    'I' || ' ' || 'love' || ' ' || 'Tengri' as result_string,
    ['T', 'e'] || ['n', 'g', 'r', 'i'] as result_array;

DESCRIBE TABLE concat;
```

```
+---------------+-------------+------+------+---------+-------+
|  column_name  | column_type | null |  key | default | extra |
+---------------+-------------+------+------+---------+-------+
| result_blob   | BLOB        | YES  | null | null    | null  |
+---------------+-------------+------+------+---------+-------+
| result_string | VARCHAR     | YES  | null | null    | null  |
+---------------+-------------+------+------+---------+-------+
| result_array  | VARCHAR[]   | YES  | null | null    | null  |
+---------------+-------------+------+------+---------+-------+
```

Functions for geodata

Functions for geospatial data and GIS tools.

ST_Point()

Description Creates a point of type `GEOMETRY`.

Usage `ST_Point(num, num)`

Creates a point of type `GEOMETRY` from two numbers of type `DOUBLE`. The arguments can be, for example, geographic coordinates given in degrees as a decimal fraction.

Some usage examples are described in this script.

▼ *See example*

Create a table with names of organisations and coordinates of their offices. Calculate the distance from zero kilometre of Moscow (`55.75579845052788, 37.617679973467204`) to the offices of these organisations and display this information in the form of a table:

```
CREATE TABLE demo.table (organization VARCHAR,
                         latitude DOUBLE,
                         longitude DOUBLE);

INSERT INTO demo.table VALUES
    ('Postgres Professional LLC', 55.69189394353437, 37.564623398131985),
    ('Oracle Corporation', 30.243622717202587, -97.72199761339736);

SELECT
    organization,
    round(ST_Distance_Sphere(ST_Point(latitude, longitude),
                             ST_Point(55.75579845052788, 37.617679973467204)
                            )/1000)
    AS "distance from moscow, km"
FROM demo.table;
```

```
+--------------------------+--------------------------+
| organisation | distance from moscow, km |
+--------------------------+--------------------------+
| Postgres Professional LLC | 8 |
+--------------------------+--------------------------+
| Oracle Corporation | 9557 |
+--------------------------+--------------------------+
```

ST_Distance()

Description Calculates the distance between two points in the plane.

Usage `ST_Distance(GEOMETRY, GEOMETRY)`

Calculates the distance in the plane between two points of type `GEOMETRY`.

▼ *See example*

```
SELECT
    ST_Distance('POINT (0 0)'::GEOMETRY, 'POINT (5 12)'::GEOMETRY)
        AS distance;
```

```
+----------+
| distance |
+----------+
| 13       |
+----------+
```

ST_Distance_Sphere()

Description Calculates the distance between two points on the sphere.

Usage `ST_Distance_Sphere(GEOMETRY, GEOMETRY)`

Calculates the haversine (great circle of the sphere) distance between two points of type `GEOMETRY`.

Returns the distance in metres. Input data must be in coordinates given in degrees as a decimal fraction (WGS84 format, EPSG:4326) with axis order: latitude, longitude.

Some usage examples are described in this script.

▼ *See example*

Create a table with names of organisations and coordinates of their offices. Calculate the distance from zero kilometre of Moscow (`55.75579845052788, 37.617679973467204`) to the offices of these organisations and display this information in the form of a table:

```
CREATE TABLE demo.table (organization VARCHAR,
                         latitude DOUBLE,
                         longitude DOUBLE);

INSERT INTO demo.table VALUES
    ('Postgres Professional LLC', 55.69189394353437, 37.564623398131985),
    ('Oracle Corporation', 30.243622717202587, -97.72199761339736);

SELECT
    organization,
    round(ST_Distance_Sphere(ST_Point(latitude, longitude),
                             ST_Point(55.75579845052788, 37.617679973467204)
                            )/1000)
    AS "distance from moscow, km"
FROM demo.table;
```

```
+--------------------------+--------------------------+
| organisation | distance from moscow, km |
+--------------------------+--------------------------+
| Postgres Professional LLC | 8 |
```

```
+--------------------------+-------------------------+
| Oracle Corporation | 9557 |
+--------------------------+-------------------------+
```

See also

- Type for geodata

Window functions

Window functions are functions that perform calculations for a set of rows that are related to the current row in some way. Unlike aggregate functions, values calculated by window functions can be inserted into each row of the source table.

A window function call always contains an `OVER` expression following the window function name and arguments. The `OVER` expression specifies exactly how the rows of the data set are to be divided for processing by the window function.

The `OVER` expression may be supplemented by the `PARTITION BY` expression, which divides the rows of the data set into groups, combining the rows into groups by matching the values of the specified columns. The value of the window function is calculated by the rows that fall into the same group as the current row.

`cume_dist()`

Description Cumulative allocation.

Usage `cume_dist([ORDER BY column])`

Cumulative distribution: the number of rows in the group preceding the current row or with the same value as the current row, divided by the total number of rows in the group.

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

▼ *See example*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 2),
    ('A', 3),
    ('B', 4);

SELECT gr, num,
    cume_dist(ORDER BY num) OVER (PARTITION BY gr) AS cume_dist
    FROM t ORDER BY num;
```

```
+-------+--------+-----------+
```

```
| gr    | num    | cume_dist |
+-------+--------+-----------+
| A     | 1      | 0.25      |
+-------+--------+-----------+
| A     | 2      | 0.75      |
+-------+--------+-----------+
| A     | 2      | 0.75      |
+-------+--------+-----------+
| A     | 3      | 1         |
+-------+--------+-----------+
| B     | 4      | 1         |
+-------+--------+-----------+
```

dense_rank()

Description The rank of the current string within the group without skips.

Usage `dense_rank()`

Aliases `rank_dense()`

Returns the rank of the current string within a group without skips. Essentially this function counts groups of rows with matching values on a given column and assigns them numbers within the group.

▼ *See example*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 2),
    ('A', 5),
    ('B', 4);

SELECT gr, num,
    dense_rank() OVER (PARTITION BY gr ORDER BY num) AS dense_rank
    FROM t ORDER BY gr, num;
```

```
+----+--------+------------+
| gr | num    | dense_rank |
+----+--------+------------+
| A  | 1      | 1          |
+----+--------+------------+
| A  | 2      | 2          |
+----+--------+------------+
| A  | 2      | 2          |
+----+--------+------------+
| A  | 5      | 3          |
+----+--------+------------+
| B  | 4      | 1          |
```

```
+----+--------+-----------+
```

`first_value()`

Description Returns the value calculated using the specified expression for the first row of the group.

Usage `first_value(expression[ ORDER BY column][ IGNORE NULLS])`.

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

Parameters

- `IGNORE NULLS` — the value is calculated excluding rows with `NULL` value.

▼ *See example*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 3),
    ('A', NULL),
    ('B', 4);

SELECT gr, num,
    first_value(num) OVER (PARTITION BY gr) AS first_value
    FROM t ORDER BY gr, num;
```

```
+----+------+-------------+
| gr | num  | first_value |
+----+------+-------------+
| A  | 1    | 1           |
+----+------+-------------+
| A  | 2    | 1           |
+----+------+-------------+
| A  | 3    | 1           |
+----+------+-------------+
| A  | null | 1           |
+----+------+-------------+
| B  | 4    | 4           |
+----+------+-------------+
```

`lag()`

Description Returns the value calculated for the string shifted by `offset` rows from the current to the beginning of the group.

Usage `lag(expression[, offset[, default]][ ORDER BY column][ IGNORE NULLS])`.

If no such row exists, the value `default` is returned (it must be of a compatible type).

Both arguments, `offset` and `default`, are optional. If they are not specified, the following default values are used:

- `offset: 1`
- `default: NULL`

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

Parameters

- `IGNORE NULLS` — the value is calculated excluding rows with `NULL` value.

▼ *See examples*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 3),
    ('B', 4),
    ('B', 5);

SELECT gr, num,
    lag(num) OVER (PARTITION BY gr) AS lag
    FROM t ORDER BY gr, num;
```

```
+----+-----+------+
| gr | num | lag  |
+----+-----+------+
| A  | 1   | null |
+----+-----+------+
| A  | 2   | 1    |
+----+-----+------+
| A  | 3   | 2    |
+----+-----+------+
| B  | 4   | null |
+----+-----+------+
| B  | 5   | 4    |
+----+-----+------+
```

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
```

```
    ('A', 2),
    ('A', 3),
    ('B', 4),
    ('B', 5);

SELECT gr, num,
    lag(num, 1, -1) OVER (PARTITION BY gr) AS lag
    FROM t ORDER BY gr, num;
```

```
+----+-----+-----+
| gr | num | lag |
+----+-----+-----+
| A  | 1   | -1  |
+----+-----+-----+
| A  | 2   | 1   |
+----+-----+-----+
| A  | 3   | 2   |
+----+-----+-----+
| B  | 4   | -1  |
+----+-----+-----+
| B  | 5   | 4   |
+----+-----+-----+
```

`last_value()`

Description	Returns the value calculated by the specified expression for the last row of the group.
Usage	`last_value(expression[ ORDER BY column][ IGNORE NULLS])`

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

Parameters

- `IGNORE NULLS` — the value is calculated excluding rows with `NULL` value.

▼ *See example*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 3),
    ('A', NULL),
    ('B', 4),
    ('B', 5);

SELECT gr, num,
    last_value(num IGNORE NULLS) OVER (PARTITION BY gr) AS last_value
```

```
    FROM t ORDER BY gr, num;
```

```
+----+------+------------+
| gr | num  | last_value |
+----+------+------------+
| A  | 1    | 3          |
+----+------+------------+
| A  | 2    | 3          |
+----+------+------------+
| A  | 3    | 3          |
+----+------+------------+
| A  | null | 3          |
+----+------+------------+
| B  | 4    | 5          |
+----+------+------------+
| B  | 5    | 5          |
+----+------+------------+
```

`lead()`

Description	Returns the value calculated for the row shifted by `offset` rows from the current row to the end of the group.
Usage	`lead(expression[, offset[, default]][ ORDER BY column][ IGNORE NULLS])`.

If no such row exists, the value of `default` is returned (it must be type-compatible).

Both arguments, `offset` and `default`, are optional. If they are not specified, the following default values are used:

- `offset: 1`
- `default: NULL`

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

Parameters

- `IGNORE NULLS` — the value is calculated excluding rows with `NULL` value.

▼ *See examples*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 3),
    ('B', 4),
    ('B', 5);
```

```
SELECT gr, num,
    lead(num) OVER (PARTITION BY gr) AS lead
    FROM t ORDER BY gr, num;
```

```
+----+-----+------+
| gr | num | lead |
+----+-----+------+
| A  | 1   | 2    |
+----+-----+------+
| A  | 2   | 3    |
+----+-----+------+
| A  | 3   | null |
+----+-----+------+
| B  | 4   | 5    |
+----+-----+------+
| B  | 5   | null |
+----+-----+------+
```

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 3),
    ('B', 4),
    ('B', 5);

SELECT gr, num,
    lead(num, 1, -1) OVER (PARTITION BY gr) AS lead
    FROM t ORDER BY gr, num;
```

```
+----+-----+------+
| gr | num | lead |
+----+-----+------+
| A  | 1   | 2    |
+----+-----+------+
| A  | 2   | 3    |
+----+-----+------+
| A  | 3   | -1   |
+----+-----+------+
| B  | 4   | 5    |
+----+-----+------+
| B  | 5   | -1   |
+----+-----+------+
```

`nth_value()`

Description Returns the value calculated for the nth row within a group (counting from 1).

Usage	`nth_value(expression, n[ ORDER BY column][ IGNORE NULLS])`.

If there is no such row, the value `NULL` is returned.

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

Parameters

- `IGNORE NULLS` — the value is calculated excluding rows with `NULL` value.

▼ *See examples*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 3),
    ('B', 4),
    ('B', 5);

SELECT gr, num,
    nth_value(num, 3) OVER (PARTITION BY gr) AS third_value
    FROM t ORDER BY gr, num;
```

```
+----+-----+-------------+
| gr | num | third_value |
+----+-----+-------------+
| A  | 1   | 3           |
+----+-----+-------------+
| A  | 2   | 3           |
+----+-----+-------------+
| A  | 3   | 3           |
+----+-----+-------------+
| B  | 4   | null        |
+----+-----+-------------+
| B  | 5   | null        |
+----+-----+-------------+
```

`ntile()`

Description	Splits each group into `num` subgroups of equal (as large as possible) size and returns the subgroup number.
Usage	`ntile(num[ ORDER BY ordering])`

The partitioning is done so that the sizes of the subgroups are as close as possible.

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

▼ *See example*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 3),
    ('B', 4),
    ('B', 5);

SELECT gr, num,
    ntile(2) OVER (PARTITION BY gr) AS group_number
    FROM t ORDER BY gr, num;
```

```
+----+-----+--------------+
| gr | num | group_number |
+----+-----+--------------+
| A  | 1   | 1            |
+----+-----+--------------+
| A  | 2   | 1            |
+----+-----+--------------+
| A  | 3   | 2            |
+----+-----+--------------+
| B  | 4   | 1            |
+----+-----+--------------+
| B  | 5   | 2            |
+----+-----+--------------+
```

`percent_rank()`

Description Calculates the relative rank of the current row within a group.

Usage `percent_rank([ORDER BY column])`

Returns a value of type `DOUBLE` from `0` to `1` inclusive. The relative rank is calculated using the formula `(rank - 1) / (total number of rows in the group - 1)`.

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

▼ *See example*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 3),
    ('B', 4),
    ('B', 5);
```

```
SELECT gr, num,
    percent_rank(ORDER BY num) OVER (PARTITION BY gr) AS percent_rank,
    FROM t ORDER BY gr, num;
```

```
+----+-----+--------------+
| gr | num | percent_rank |
+----+-----+--------------+
| A  | 1   | 0            |
+----+-----+--------------+
| A  | 2   | 0.5          |
+----+-----+--------------+
| A  | 3   | 1            |
+----+-----+--------------+
| B  | 4   | 0            |
+----+-----+--------------+
| B  | 5   | 1            |
+----+-----+--------------+
```

`rank()`

Description Returns the rank (with skips) of the current row within a group.

Usage `rank([ORDER BY column])`

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

▼ *See example*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 2),
    ('A', 3),
    ('B', 4),
    ('B', 5);

SELECT gr, num,
    rank(ORDER BY num) OVER (PARTITION BY gr) AS rank,
    FROM t ORDER BY gr, num;
```

```
+----+-----+------+
| gr | num | rank |
+----+-----+------+
| A  | 1   | 1    |
+----+-----+------+
| A  | 2   | 2    |
+----+-----+------+
| A  | 2   | 2    |
```

```
+----+-----+------+
| A  | 3   | 4    |
+----+-----+------+
| B  | 4   | 1    |
+----+-----+------+
| B  | 5   | 2    |
+----+-----+------+
```

`row_number()`

Description Returns the number of the current row in its group (counting from `1`).

Usage `row_number([ORDER BY column])`

If the `ORDER BY` condition is specified, the value is calculated using the specified order.

▼ *See example*

```
CREATE TABLE t(gr VARCHAR, num BIGINT);

INSERT INTO t VALUES
    ('A', 1),
    ('A', 2),
    ('A', 2),
    ('A', 3),
    ('B', 4),
    ('B', 5);

SELECT gr, num,
    row_number() OVER (PARTITION BY gr) AS row_number,
    FROM t ORDER BY gr, num;
```

```
+----+-----+------------+
| gr | num | row_number |
+----+-----+------------+
| A  | 1   | 1          |
+----+-----+------------+
| A  | 2   | 2          |
+----+-----+------------+
| A  | 2   | 3          |
+----+-----+------------+
| A  | 3   | 4          |
+----+-----+------------+
| B  | 4   | 1          |
+----+-----+------------+
| B  | 5   | 2          |
+----+-----+------------+
```

Utilities

Utilities are functions for working with data of various types that are difficult to categorise. Their descriptions are collected in this section.

`coalesce()`

Description Returns the first value other than `NULL` from the list of argument values.

Usage `coalesce(argument1[, argument2, {⋯])`

If the only argument has the value `NULL`, then `NULL` is returned.

▼ *See example*

```
SELECT
    coalesce(NULL, 'Tengri', NULL) AS result_1,
    coalesce(NULL, '', NULL) AS result_2,
    coalesce('Tengri') AS result_3,
    coalesce(NULL) AS result_4;
```

```
+----------+----------+----------+----------+
| result_1 | result_2 | result_3 | result_4 |
+----------+----------+----------+----------+
| Tengri   |          | Tengri   | null     |
+----------+----------+----------+----------+
```

`generate_series()`

Description Generates a list of values in the range between `start` and `stop`.

Usage `generate_series([start,] stop[, step])`

The parameters `start` and `stop` are treated as "inclusive".
The default value for `start` is `0`, and for `step` is `1`.

▼ *See example*

```
SELECT
    generate_series(10) AS stop,
    generate_series(5, 10) AS start_stop,
    generate_series(5, 10, 2) AS start_stop_step;
```

```
+-------------------------+----------------+-----------------+
| stop                    | start_stop     | start_stop_step |
+-------------------------+----------------+-----------------+
| {0,1,2,3,4,5,6,7,8,9,10} | {5,6,7,8,9,10} | {5,7,9}         |
+-------------------------+----------------+-----------------+
```

hash()

Description Returns a hash of the data from `argument` as a number.

Usage `hash(argument)`

▼ *See example*

```
SELECT
    hash('Tengri') AS hash;
```

```
+----------------------+
|         hash         |
+----------------------+
| 15418814193266442000 |
+----------------------+
```

unnest()

Description Expands lists or structures from `argument` into a set of distinct values.

Usage `unnest(argument) [, recursive := true] [, max_depth := <num>]`

Applying the function to a list yields one line for each element in the list. The usual scalar expressions in the same `SELECT` expression are repeated for each row output.

When multiple lists are expanded in the same `SELECT` expression, they are expanded each into a separate column. If one list is longer than another, the shorter list is filled with `NULL` values.

The function changes the cardinality of the data.

Parameters

- `recursive := true`
  Enables recursive mode. If this mode is enabled (value `true`), the function fully expands lists and then fully expands nested structures. This can be useful for fully "flattening" columns that contain lists within lists or structures within lists. Note that lists within structures are not expanded.

  ▼ *For more information about the parameter, see examples*

  Let's show how this parameter works on two examples with the same data with and without the parameter enabled:

  ```
  SELECT
      unnest([[1, 2, 3], [4, 5]], recursive := true) AS result;
  ```

  ```
  +--------+
  | result |
  ```

```
+--------+
| 1      |
+--------+
| 2      |
+--------+
| 3      |
+--------+
| 4      |
+--------+
| 5      |
+--------+
```

```
SELECT
    unnest([[1, 2, 3], [4, 5]]) AS result;
```

```
+---------+
|  result |
+---------+
| {1,2,3} |
+---------+
| {4,5}   |
+---------+
```

- `max_depth := <num>`
  The `max_depth` parameter allows you to limit the maximum depth of recursive deployment. Recursive mode is automatically enabled if the maximum depth is specified.

  ▼ *For more information about the parameter, see examples*

  Let's show how this parameter works using three examples with the same data: with default deployment depth (`1`), with deployment depth `2` and with deployment depth `3`:

```
SELECT
    unnest([[['T', 'e'], ['n', 'g']], [['r', 'i'], []], [['!', '!', '!']]])
AS result;
```

```
+-------------------------+
|          result         |
+-------------------------+
| {['T', 'e'],['n', 'g']} |
+-------------------------+
| {['r', 'i'],[]}         |
+-------------------------+
| {['!', '!', '!']}       |
+-------------------------+
```

```
SELECT
    unnest([[['T', 'e'], ['n', 'g']], [['r', 'i'], []], [['!', '!', '!']]],
    max_depth := 2)
AS result;
```

```
+---------+
|  result |
+---------+
| {T,e}   |
+---------+
| {n,g}   |
+---------+
| {r,i}   |
+---------+
| {}      |
+---------+
| {!,!,!} |
+---------+
```

```
SELECT
    unnest([[['T', 'e'], ['n', 'g']], [['r', 'i'], []], [['!', '!', '!']]],
    max_depth := 3)
AS result;
```

```
+--------+
| result |
+--------+
| T      |
+--------+
| e      |
+--------+
| n      |
+--------+
| g      |
+--------+
| r      |
+--------+
| i      |
+--------+
| !      |
+--------+
| !      |
+--------+
| !      |
+--------+
```

▼ *See more examples*

```
SELECT
    unnest([1,2,3])       AS numbers,
    unnest(['a','b','c']) AS letters;
```

```
+---------+---------+
| numbers | letters |
+---------+---------+
| 1       | a       |
+---------+---------+
| 2       | b       |
+---------+---------+
| 3       | c       |
+---------+---------+
```

```
SELECT
    unnest([1,2,3])   AS numbers,
    unnest(['a','b']) AS letters;
```

```
+---------+---------+
| numbers | letters |
+---------+---------+
| 1       | a       |
+---------+---------+
| 2       | b       |
+---------+---------+
| 3       | null    |
+---------+---------+
```

```
SELECT
    unnest([{'column_a': 1, 'column_b': 84},
            {'column_a': 100, 'column_b': NULL, 'column_c':22}],
            recursive := true);
```

```
+----------+----------+----------+
| column_a | column_b | column_c |
+----------+----------+----------+
| 1        | 84       | null     |
+----------+----------+----------+
| 100      | null     | 22       |
+----------+----------+----------+
```

```
SELECT
    unnest([{'column_a': 1, 'column_b': 84},
            {'column_a': 100, 'column_b': NULL, 'column_c':22}])
```

```
AS result;
```

```
+-----------------------------------------------------+
|                        result                       |
+-----------------------------------------------------+
| {"column_a": 1, "column_b": 84, "column_c": null}   |
+-----------------------------------------------------+
| {"column_a": 100, "column_b": null, "column_c": 22} |
+-----------------------------------------------------+
```

Python module `tngri` functions

The Python module `tngri` was created for convenient work with data in cells of type Python. The functions described on this page are available in it.

The `tngri` module functions are available only in cells of Python type and for their operation it is necessary to import the module: `import tngri`.

`tngri.sql()`

Description Executes the specified query SQL inside a cell of type Python.

Usage `tngri.sql('<SQL_query>')`

This function is useful when you need to execute any SQL queries directly inside a cell of type Python, such as inside a loop or other complex constructs, without creating a separate cell of type SQL and using any local variables and functions Python inside the query text SQL.

One usage scenario is described here.

▼ *See examples*

Example 1

Let's create a table with a name from the `table_name` variable and write to it in a loop:

- index (starting with `1`)
- word from the phrase specified in the `test_phrase` variable
- the result of applying the given function `length_in_chars` to this word

In each iteration of the loop we will output the index value, the added word and the result of the query SQL with the current number of rows in the table being created.

```
import tngri

def length_in_chars(text):
    if len(text) == 1:
        return '1 character'
```

```
    else:
        return f'{len(text)} characters'

table_name = 'my_table'
test_phrase = 'I love Tengri'

tngri.sql(f'CREATE OR REPLACE TABLE {table_name} \
          (index INT, word VARCHAR, length VARCHAR)'
         )

ind = 0
for word in test_phrase.split(' '):
    ind += 1
    tngri.sql(f"INSERT INTO {table_name} VALUES \
              ({ind}, '{word}', '{length_in_chars(word)}')"
             )
    print(f'Added word: "{word}"')
    print(tngri.sql(f'SELECT count(*) FROM {table_name}'))
```

```
Step: 1
Added word: "I"
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 1        |
+----------+
Step: 2
Added word: "love"
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 2        |
+----------+
Step: 3
Added word: "Tengri"
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 3        |
+----------+
```

Now in a cell of type SQL we will display the created table ordered by index:

```
SELECT * FROM my_table
    ORDER BY index
```

```
+-------+--------+--------------+
| index | word   | length       |
+-------+--------+--------------+
| 1     | I      | 1 character  |
+-------+--------+--------------+
| 2     | love   | 4 characters |
+-------+--------+--------------+
| 3     | Tengri | 6 characters |
+-------+--------+--------------+
```

Example 2

Let's iteratively load data from `.parquet` files from S3 storage into a table by file path mask.

In each iteration of the loop we will output the result of the query with the number of rows in the created table.

```
import tngri

for i in range(1,4):
    file_name = f"s3://prostore/Stage/<lake_path>/{i}.parquet"
    tngri.sql(f"INSERT INTO raw_dyntest SELECT * FROM read_parquet('{file_name}')")
    print(tngri.sql("SELECT count(*) FROM raw_dyntest"))
```

```
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 10000000 |
+----------+
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 20000000 |
+----------+
shape: (1, 1)
+----------+
| column_0 |
| ---      |
```

```
 | i64      |
+----------+
 | 30000000 |
+----------+
```

`tngri.upload_df()`

Description Uploads data from DataFrame to Tengri.

Usage `tngri.upload_df(DataFrame)`

Uploads data from the specified DataFrame to Tengri (in the S3) storage.
Returns a string containing the name of the `.parquet` file to which the data was uploaded.

Loading data into Tengri using Python is described in more detail at here.

▼ *See examples*

- Let's create a DataFrame, and load it into Tengri:

```
import tngri
import pandas

my_df = pandas.DataFrame(range(100))

tngri.upload_df(my_df)
```

```
UploadedFile(s3_path='s3://prostore/Stage/ijwsajclddxw.parquet', _client=None)
```

- Let's create a DataFrame, load it into Tengri and write the file name `.parquet` to the `file_name` variable:

```
import tngri
import pandas

my_df = pandas.DataFrame(range(100))
file_name = tngri.upload_df(my_df)

print(my_df)
print(file_name)
```

```
0
0    0
1    1
2    2
3    3
4    4
..  ..
95  95
```

```
96  96
97  97
98  98
99  99

[100 rows x 1 columns]
tdiiuetablyx.parquet
```

tngri.upload_file()

Description Uploads data from a file to Tengri.

Usage `tngri.upload_file(FilePath[, FileName])`

- `FilePath` — path to the file to upload to Tengri
- `FileName` — the name for the file in the Tengri

Loads the data from the file at the specified path into Tengri (in the S3) repository.
Returns a string containing the path and filename of the file within the S3 storage to which the data was loaded.

One of the usage scenarios is described at here.

▼ *See examples*

- Let's load data into Tengri from the `.json` file available at URL:

```
import tngri
import urllib.request

urllib.request.urlretrieve(

'https://tngri.postgrespro.ru/documentation/ru/stable/_attachments/tengri_data_types.json',
    'my_file.json'
    )

tngri.upload_file('my_file.json')
```

```
UploadedFile(s3_path='s3://prostore/Stage/pxfihzbonctd.json', _client=None)
```

Let's output the first 5 rows of the table by reading it from the loaded file:

```
SELECT * FROM read_json("pxfihzbonctd.json")
    LIMIT 5
```

```
+----------+-----------+----------+--------------------------+
```

```
| name     | type      | category | description               |
+----------+-----------+----------+---------------------------+
| BIGINT   | data type | numeric  | Целые числа.              |
+----------+-----------+----------+---------------------------+
| BIGINT[] | data type | array    | Массивы целых чисел.      |
+----------+-----------+----------+---------------------------+
| BLOB     | data type | blob     | Двоичные объекты.         |
+----------+-----------+----------+---------------------------+
| BOOL     | data type | boolean  | Булевы значения.          |
+----------+-----------+----------+---------------------------+
| BOOL[]   | data type | array    | Массивы булевых значений. |
+----------+-----------+----------+---------------------------+
```

- Let's load data into Tengri from the `.json` file available at URL, and save the name of the loaded file to a variable:

```
import tngri
import urllib.request

urllib.request.urlretrieve(

'https://tngri.postgrespro.ru/documentation/ru/stable/_attachments/tengri_data_types.jso
n',
    'my_file.json'
    )

file_name = tngri.upload_file('my_file.json')
print(file_name)
```

```
pxfihzbonctd.json
```

Let's output the first 5 rows of the table by reading it from the loaded file:

```
SELECT * FROM read_json("pxfihzbonctd.json")
    LIMIT 5
```

```
+----------+-----------+----------+---------------------------+
| name     | type      | category | description               |
+----------+-----------+----------+---------------------------+
| BIGINT   | data type | numeric  | Целые числа.              |
+----------+-----------+----------+---------------------------+
| BIGINT[] | data type | array    | Массивы целых чисел.      |
+----------+-----------+----------+---------------------------+
| BLOB     | data type | blob     | Двоичные объекты.         |
+----------+-----------+----------+---------------------------+
| BOOL     | data type | boolean  | Булевы значения.          |
+----------+-----------+----------+---------------------------+
| BOOL[]   | data type | array    | Массивы булевых значений. |
```

```
+---------+----------+---------+-------------------------+
```

`tngri.upload_s3()`

Description Uploads a file from the specified bucket S3 to Tengri.

Usage

```
tngri.upload_s3(
    object = 's3://<file_path>.parquet',
    access_key = '<access_key>',
    secret_key = '<secret_key>',
    [filename = '<new_name>.parquet']
    )
```

- `object` — path to file in S3 storage .
- `access_key` — access key for S3.
- `secret_key` — secret key for S3.

The file extension can be anything. It will remain the same as it was in the initial file. If necessary, the path and name of the uploaded file can be specified via the optional `filename` parameter.

An example of use is described at here.

Data types

	Type name	Aliases	
Numeric types	`BIGINT`.	`INTEGER`	Integers
	`NUMERIC`	`DECIMAL`	Real numbers with specified precision
	`DOUBLE`	`FLOAT`	Real numbers with variable precision
Text type	`VARCHAR`	`CHAR`, `BPCHAR`, `STRING`, `TEXT`	Text strings
Types for date and time	`DATE`		Dates
	`TIME`		Time
	`TIMESTAMP`.		Time stamps
	`TIMESTAMPTZ`		Time stamps with time zone
JSON type	`JSON'		Data in JSON
Types for arrays	`BIGINT[/]`	`INTEGER[]`	Arrays of integers
	`VARCHAR[/]`	`CHAR[]`, `BPCHAR[]`, `STRING[]`, `TEXT[]`	Arrays of text strings
	`BOOL[/]`	`BOOLEAN[]`	Arrays of Boolean values

	Type name	Aliases	
Binary type	`BLOB'	``BYTEA''	Binary objects
Logical type	`BOOL	`BOOLEAN	Boolean values
Type for geodata	GEOMETRY	GEOMETRY	Geospatial data

Numeric types

Name	Alias	Range	Description
BIGINT	INTEGER	- 2^63... 2^63 - 1	Integers
NUMERIC	DECIMAL.	WIDTH: 37	Real numbers with a given precision
DOUBLE	FLOAT	1E-307... 1E+308	Real numbers with variable precision

Integer type

- BIGINT.
  - Alias: INTEGER

The BIGINT data type stores integers, that is, numbers without fractional components, of various ranges. Attempts to store values outside the valid range will result in an error.

The range of possible values for the BIGINT type is from - 2^63 to 2^63 - 1.

Type with specified precision

- NUMERIC.
  - Alias: DECIMAL

The NUMERIC (WIDTH, SCALE) data type represents an exact decimal value with a fixed decimal separator — a dot.

When creating a value of type NUMERIC, you can specify the parameters WIDTH and SCALE.

- The WIDTH parameter defines the maximum total number of digits.
- The SCALE parameter defines the number of digits after the decimal point.

For example, the NUMERIC(3, 2) type can contain the value 1.23, but cannot contain the value 12.3 or 1.234. If the WIDTH and SCALE parameters are not specified explicitly, the default values are used: NUMERIC(37,18).

Adding, subtracting and multiplying two fixed-point decimal numbers returns another fixed-point decimal number with the required WIDTH and SCALE or gives an error if the required WIDTH exceeds the maximum supported WIDTH, which is 37.

If you need to store numbers with a known number of decimal places, as well as precise addition, subtraction and multiplication operations (e.g. for **monetary amounts**), then you

> should use the `NUMERIC` precision data type.

Variable precision type

- `DOUBLE`.
  - Alias: `FLOAT`

The `DOUBLE` data type is a numeric type with variable precision. This type stores a double precision floating decimal point number (8 bytes).

The range of possible values for the `DOUBLE` type is from `1E-307` to `1E+308`.

As with the data type `NUMERIC`, when converting literals or converting other data types to a type with variable precision, input data that cannot be represented exactly is stored as approximate values.

While multiplication, addition, and subtraction for type `NUMERIC` are exact operations, the same operations for a type with variable precision are only approximate.

> If you need to perform fast or complex calculations, a variable-precision data type may be more appropriate than a type of `NUMERIC`. However, if you use the results of these calculations to make important decisions, you should carefully analyse the implementation of these calculations in borderline cases (ranges, infinite values, anti-overflows, invalid operations, etc.). They may be handled differently than you expect.

Examples

Create a table `numbers` with columns having numeric types `BIGINT`, `NUMERIC(4,3)`, `DOUBLE` and fill one row with values `2^62`, `1.1` and `3.14159265358979323846`:

```
CREATE TABLE numbers(
    bigint BIGINT,
    numeric NUMERIC(4,3),
    double DOUBLE);

INSERT INTO numbers VALUES
    (2^62, 1.1, 3.14159265358979323846);

SELECT * FROM numbers;
```

```
+--------------------+---------+-------------------+
|        bigint      | numeric |       double      |
+--------------------+---------+-------------------+
| 4611686018427388000 | 1.100   | 3.141592653589793 |
+--------------------+---------+-------------------+
```

Note that in the program output in the `numeric` column the original number is displayed with the specified precision (3 digits after the decimal separator), and in the `double` column the original number is displayed

with the highest possible precision, so the number of digits after the separator is less than in the entered number.

See also

- Numerical functions

Text type

- `VARCHAR`.
  - Aliases: `CHAR`, `BPCHAR`, `STRING`, `TEXT`

Description

The `VARCHAR` type is used to store text strings. This type allows storing characters Unicode. The data is encoded in the format UTF-8.

To specify the value of a text string in the expression `INSERT`, use single quotes.

Example

Create a `varchar_table` table and fill two cells with text strings:

```
CREATE TABLE varchar_table(text_column VARCHAR);

INSERT INTO varchar_table VALUES
    ('My name is Tengri'),
    ('My nickname is TNGRi');

SELECT * FROM varchar_table;
```

```
+----------------------+
|      text_column     |
+----------------------+
| My name is Tengri    |
+----------------------+
| My nickname is TNGRi |
+----------------------+
```

Note that in programme output, text strings are displayed without inverted commas.

See also

- `LIKE`
- Text functions

- Functions for regular expressions

Types for date and time

Name	Format	Description
`DATE`	`YYYYYY-MM-DD`	Dates
`TIME`	`[YYYYY-MM-DD ]HH:MM[:SS][.MS]`	Time
`TIMESTAMP`	`YYYYYY-MM-DD hh:mm[:ss][.zzzzzzzz]`.	Time moments
`TIMESTAMPTZ`.	`YYYYYY-MM-DD hh:mm[:ss][.zzzzzzzz][+-TT[:tt] | ZONE]`	Time moments with time zone

Dates

- `DATE`

The `DATE` type is used to store a date (a combination of year, month and day). Dates are counted according to the Gregorian calendar, even for times before its introduction.

Data for this type must be formatted according to the standard ISO 8601:
`YYYYYY-MM-DD`.

Time

- `TIME`

The `TIME` type is used to store time (hours, minutes, seconds and microseconds within a day).

Data for this type must be formatted according to the standard ISO 8601:
`[YYYYY-MM-DD ]HH:MM:SS[.MS]`.

The `TIME` type should only be used in rare cases where the date portion of the timestamp may be ignored. In most cases, the type should be used to represent specific points in time `TIMESTAMP`.

Moments in time

- `TIMESTAMP`
- `TIMESTAMPTZ`

The `TIMESTAMP` and `TIMESTAMPTZ` types are used to store timestamps — specific moments in time. They combine date (type `DATE`) and time (type `TIME`) information.

For the `TIMESTAMP` type, the data must be formatted according to the ISO 8601 standard:
`YYYYYY-MM-DD hh:mm:ss[.zzzzzzzz]`.

For the `TIMESTAMPTZ` (*TIMESTAMP TIME ZONE*) type, the data shall be formatted in the same way, but it is

allowed to add a time zone:
`YYYYYY-MM-DD hh:mm:ss[.zzzzzzzz][+-TT[:tt] | ZONE]`.

Decimal places beyond the supported precision are ignored and do not cause an error.

Time zones can be specified either via an indentation in hours from the specified time (`+01`, `-01`) or by explicitly specifying an identifier from list of time zones `tz_database`.

Examples

Let's create a table `dates` with columns of types `DATE`, `TIME` and `TIMESTAMP` and write different valid values into them:

```
CREATE TABLE dates(
    date DATE,
    time TIME,
    timestamp TIMESTAMP);

INSERT INTO dates VALUES
    ('2025-01-02', '1:1', '2025-01-02 1:1:00'),
    ('2025-01-02', '1:01:11', '2025-01-02 1:1:11.111'),
    ('2025-01-02', '01:01:11.1234567', '2025-01-02 1:1:11.1234567');

SELECT * FROM dates;
```

```
+------------+-----------------+----------------------------+
|    date    |       time      |          timestamp         |
+------------+-----------------+----------------------------+
| 2025-01-02 | 01:01:00        | 2025-01-02 01:01:00        |
+------------+-----------------+----------------------------+
| 2025-01-02 | 01:01:11        | 2025-01-02 01:01:11.111000 |
+------------+-----------------+----------------------------+
| 2025-01-02 | 01:01:11.123456 | 2025-01-02 01:01:11.123456 |
+------------+-----------------+----------------------------+
```

Note that in the programme output, the `time` column shows the time in the standard form and with the mandatory value of seconds (despite the different input methods). In the `timestamp` column, the time is also displayed in standard form. Microseconds are only displayed if they have been set and with an accuracy of 6 digits.

Let's create a table `time_stamps` with columns of types `TIMESTAMP` and `TIMESTAMPTZ` and write in them the same pairs of values — with and without time zone:

```
CREATE TABLE time_stamps(
    timestamp TIMESTAMP,
    timestamptz TIMESTAMPTZ);
```

```
INSERT INTO time_stamps VALUES
    ('2025-01-02 0:0:0',       '2025-01-02 0:0:0'),
    ('2025-01-02 0:0:0+01:00', '2025-01-02 0:0:0+01:00');

SELECT * FROM time_stamps;
```

```
+---------------------+---------------------------+
|      timestamp      |        timestamptz        |
+---------------------+---------------------------+
| 2025-01-02 00:00:00 | 2025-01-02 00:00:00+00:00 |
+---------------------+---------------------------+
| 2025-01-01 23:00:00 | 2025-01-01 23:00:00+00:00 |
+---------------------+---------------------------+
```

Note that in both cases the time zone indication is handled correctly, but in the case of `TIMESTAMPTZ` it is stored in the table, while in the case of `TIMESTAMP` type it is not.

To demonstrate working with time zones, let's insert into the `time_stamps` table the time values with different time zone indications. For clarity, we will insert the same values in two columns with types `VARCHAR` and `TIMESTAMPTZ`:

```
CREATE TABLE time_stamps(
    timestampt_text VARCHAR,
    timestamptz TIMESTAMPTZ);

INSERT INTO time_stamps VALUES
    ('2025-01-01 0:0:0 UTC', '2025-01-01 0:0:0 UTC'),
    ('2025-01-01 0:0:0 CET', '2025-01-01 0:0:0 CET'),
    ('2025-01-01 0:0:0+01' , '2025-01-01 0:0:0+01' ),
    ('2025-01-01 0:0:0-01' , '2025-01-01 0:0:0-01' ),
    ('2025-01-01 0:0:0+25' , '2025-01-01 0:0:0+25' ),
    ('2025-01-01 0:0:0-25' , '2025-01-01 0:0:0-25' );

SELECT * FROM time_stamps;
```

```
+----------------------+---------------------------+
|    timestampt_text   |        timestamptz        |
+----------------------+---------------------------+
| 2025-01-01 0:0:0 UTC | 2025-01-01 00:00:00+00:00 |
+----------------------+---------------------------+
| 2025-01-01 0:0:0 CET | 2024-12-31 23:00:00+00:00 |
+----------------------+---------------------------+
| 2025-01-01 0:0:0+01  | 2024-12-31 23:00:00+00:00 |
+----------------------+---------------------------+
| 2025-01-01 0:0:0-01  | 2025-01-01 01:00:00+00:00 |
+----------------------+---------------------------+
| 2025-01-01 0:0:0+25  | 2024-12-30 23:00:00+00:00 |
```

```
+---------------------+--------------------------+
| 2025-01-01 0:0:0-25  | 2025-01-02 01:00:00+00:00 |
+---------------------+--------------------------+
```

Let's try to enter invalid data into the `dates` table:

```
CREATE TABLE dates(
    date DATE);

INSERT INTO dates VALUES
    ('2025-12-13'),
    ('2025-13-13');
```

```
ERROR: ConversionException: Conversion Error: date field value out of range:
"2025-13-13"
```

The data cannot be entered because an invalid month number — `13` — is specified.

```
CREATE TABLE dates(
    time TIME);

INSERT INTO dates VALUES
    ('1:1'),
    ('24:1');
```

```
ERROR: ConversionException: Conversion Error: time field value out of range:
"24:1", expected format is ([YYYY-MM-DD ]HH:MM:SS[.MS])
```

Data cannot be entered because a invalid hour value — `24` — is specified.

```
CREATE TABLE dates(
    timestamp TIMESTAMP);

INSERT INTO dates VALUES
    ('2025-01-01 1');
```

```
ERROR: ConversionException: Conversion Error: invalid timestamp field format:
"2025-01-01 1", expected format is (YYYY-MM-DD HH:MM:SS[.US][±HH:MM| ZONE])
```

Data cannot be entered because a non-valid time value — `1` — is set.

```
CREATE TABLE dates(
    timestamptz TIMESTAMPTZ);

INSERT INTO dates VALUES
    ('2025-01-01 0:0:0 Europe/Moscow'),
    ('2025-01-01 0:0:0 Asia/Srednekolymsk'),
    ('2025-01-01 0:0:0 MSK');
```

```
ERROR: NotImplementedException: Not implemented Error: Unknown TimeZone 'MSK'
```

The data cannot be entered because a invalid time zone value `MSK` is set. The values `Europe/Moscow` and `Asia/Srednekolymsk` are valid because they are included in list of time zone identifiers `tz_database`.

See also

- Functions for date and time

JSON type

- `JSON`

Description

The `JSON` type is used to store JSON data according to the standard syntax described by in the specification.

For storing such data it is possible to use `VARCHAR` type, but when using `JSON` type the data will be checked for conformity of input values to JSON format, so it is more convenient to use special `JSON` type in such cases. Besides, special functions are available for the `JSON` type, which allow you to directly access data in the JSON structure.

Inverted commas inside the text in JSON format should be double. And the quotes framing this text inside the `INSERT` expression must be single (see the example below).

Examples

Let's create a `js_table` table and insert the data in JSON format from this example into the `js_data` column:

```
CREATE TABLE js_table(name VARCHAR, js_data JSON);

INSERT INTO js_table VALUES
('John Smith',
'{
  "first_name": "John",
  "last_name": "Smith",
```

```
  "is_alive": true,
  "age": 27,
  "address": {
    "street_address": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postal_code": "10021-3100"
  },
  "phone_numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Thomas",
    "Trevor"
  ],
  "spouse": null
}');
```

Derive the top-level field names from the loaded JSON data using the `json_keys` function:

```
SELECT
    json_keys(js_data) AS json_fields
FROM js_table;
```

```
+-----------------------------------------------------------------------+
|                              json_fields                              |
+-----------------------------------------------------------------------+
| {first_name,last_name,is_alive,age,address,phone_numbers,children,spouse} |
+-----------------------------------------------------------------------+
```

Let's extract the full name from the `name` text column into separate columns, and take the age `age` and the number of children from the JSON data — the length of the array in the `children` field. To do this, we use the `json_extract` and `json_array_length` functions.

```
SELECT
    name,
    json_extract(js_data, 'age') AS age,
    json_array_length(js_data, 'children') AS children_num
FROM js_table;
```

```
+------------+-----+--------------+
|    name    | age | children_num |
+------------+-----+--------------+
| John Smith | 27  | 3            |
+------------+-----+--------------+
```

See also

- Functions for JSON

Types for arrays

- `BIGINT[]`
  - Alias: `INTEGER[]`
- `VARCHAR[]`
  - Aliases: `CHAR[]`, `BPCHAR[]`, `STRING[]`, `TEXT[]`
- `BOOL[]`
  - Aliases: `BOOLEAN[]`

Description

The `BIGINT[]`, `VARCHAR[]`, and `BOOL[]` types are used to store arrays of integers, arrays of strings, and arrays of boolean values, respectively.

The length of arrays in the same column of the table may not be the same.

Arrays can be used to store vectors such as embeddings (vector representations) of texts or images.

Examples

Create a table `array_table` and insert values of type `VARCHAR` and type `VARCHAR[]` into it:

```
CREATE TABLE array_table(text_column VARCHAR, array_column VARCHAR[]);

INSERT INTO array_table VALUES
    ('I love Tengri', ['I','love', 'Tengri']),
    ('I adore Tengri', ['I','adore', 'Tengri']);

SELECT * FROM array_table;
```

```
+---------------+-----------------+
|  text_column  |   array_column  |
+---------------+-----------------+
```

```
| I love Tengri  | {I,love,Tengri}  |
+----------------+------------------+
| I adore Tengri | {I,adore,Tengri} |
+----------------+------------------+
```

Note that in program output, arrays are displayed with curly braces `{}` and text values in them — without quotes.

Now let's create an `array_table` table with columns of type `BIGINT` and `BIGINT[]` and insert numeric values and arrays of numbers into it:

```
CREATE TABLE array_table(number_column BIGINT, array_column BIGINT[]);

INSERT INTO array_table VALUES
    (2025, [2,0,2,5]),
    (0025, [0,0,2,5]);

SELECT * FROM array_table;
```

```
+---------------+--------------+
| number_column | array_column |
+---------------+--------------+
| 2025          | {2,0,2,5}    |
+---------------+--------------+
| 25            | {0,0,2,5}    |
+---------------+--------------+
```

Note that the numeric value `0025` in the programme output is cast to the normal form `25`, and arrays in the programme output are shown through curly braces `{}`.

Binary type

- `BLOB`
  - Alias: `BYTEA`

Description

The `BLOB` data type is designed to store arbitrary binary objects. It can contain any binary data without any restrictions. What this data actually represents remains unknown to Tengri.

`BLOB` is typically used to store non-text objects whose type is not explicitly supported, such as images. Although the `BLOB` type can store objects up to 4 GB in size, it is not recommended to store very large objects in Tengri. It is usually more convenient to store large files on the file system, and store file paths in Tengri as the type `VARCHAR`.

See also

- Functions for binary data

Logical type

- `BOOL`
  - Alias: `BOOLEAN`

Description

The `BOOL` type represents a truth statement and can take the values:
`true`, `false` (boolean values) or `NULL`.

Boolean values can be explicitly specified using `true` and `false` literals. Most often, however, they are created as a result of a comparison. For example, the comparison `i > 10` returns a Boolean value.

Boolean values can be used in the expressions `WHERE` and `HAVING` to filter results. In this case, expressions whose result evaluates to `true` will pass the filter, and expressions whose result evaluates to `false` or `NULL` will be discarded.

Logical operators

The logical operators `AND` and `OR` can be used to combine Boolean values.

Table 1. Truth table for the `AND` operator

X	X AND true	X AND false	X AND NULL
`true`	`true`	`false`	`NULL`
`false`	`false`	`false`	`false`
`NULL`	`NULL`	`false`	`NULL`

Table 2. Truth table for the `OR` operator

X
X OR true
X OR false
X OR NULL
`true`
`true`
`true`
`true`
`false`
`true`
`false`
`NULL`
`NULL`
`true`
`NULL`
`NULL`

Type for geodata

- `GEOMETRY`

Description

Type for reading and writing geospatial data, converting between coordinate systems and working with GIS tools.

Examples

Let's calculate the distance in the plane between two points given in integer coordinates.

```
SELECT
    ST_Distance('POINT (0 0)'::GEOMETRY, 'POINT (5 12)'::GEOMETRY)
        AS distance;
```

```
+----------+
| distance |
+----------+
| 13       |
+----------+
```

Let's calculate the distance between the offices of Postgress Pro and Oracle in kilometres.

```
SELECT
    round(ST_Distance_Sphere('POINT (55.69189394353437, 37.564623398131985)'::GEOMETRY,
                             'POINT (30.243622717202587, -97.72199761339736)'::GEOMETRY
                             )/1000)
    AS "distance between postgress and oracle, km";
```

```
+-----------------------------------------+
| distance between postgress and oracle, km |
+-----------------------------------------+
| 9561                                    |
+-----------------------------------------+
```

See also

- Functions for geodata

Scenarios

Initial system setup

Suppose we need to make initial system settings to run one user who will have one role. The user will work on one compute pool inside one circuit.

In order for the actions described below to be possible, they must be performed either by an administrator or by a user with privileges to create other users, roles and compute pools.

1. Create the `analyst` role:

```
CREATE ROLE analyst;
```

2. Create the `compute_s` compute pool and set the size of the `S` computes for it:

```
CREATE WORKER POOL compute_s WORKER SIZE S;
```

3. Create user `tengri_demo_user`, set its password for identification and assign the default compute pool `compute_s` to it:

```
CREATE USER tengri_demo_user
    IDENTIFIED BY PASSWORD '***'
    DEFAULT WORKER POOL compute_s;
```

4. Assign the `tengri_demo_user` user the `analyst` role:

```
GRANT ROLE analyst TO tengri_demo_user;
```

5. Grant the `analyst` role the `USAGE` and `MONITOR` privileges on the `compute_s` compute pool:

```
GRANT USAGE, MONITOR ON WORKER POOL compute_s TO ROLE analyst;
```

Next:

a. if the system already has an `analytical_sandbox` schema and our user will only work in it, then grant him the privilege to work with all tables within that schema, both existing and those to be created in the future:

```
GRANT ALL ON SCHEMA analytical_sandbox TO ROLE analyst;
```

b. if the user will be working in a new schema, then create this `analytical_sandbox_new` schema and grant him the privilege to work with all tables within this schema:

```
CREATE SCHEMA analytical_sandbox_new;

GRANT ALL ON SCHEMA analytical_sandbox_new TO ROLE analyst;
```

6. If the user `tengri_demo_user` is expected to create new schemas, then grant him the privilege to create a schema within the catalogue:

```
GRANT CREATE SCHEMA ON CATALOG TO ROLE analyst;
```

The `tengri_demo_user` user will then be able to:

- Create tables within the `analytical_sandbox` (or `analytical_sandbox_new`) schema
- Read tables inside the `analytical_sandbox` (or `analytical_sandbox_new`) schema

With these settings the user `tengri_demo_user` will be able to:

- Delete tables within a `analytical_sandbox` (or `analytical_sandbox_new`) schema
- Delete the `analytical_sandbox` (or `analytical_sandbox_new`) schema

Data loading

You can upload data to Tengri in the following ways:

- Using the upload wizard — button **[Upload file]**.
- Via the Python `tngri` module.

Uploading data from a local file using the upload wizard

Supported file extensions for uploading:

- `.csv`
- `.json`

- `.parquet`

To load data from a file:

1. Click **[Upload file]** in the Tengri interface .
2. Select or move the file to the opened upload area.
3. In the **Parse file** window that opens, check if the parsed data is correct.
   If necessary, select the required recognition settings and press **[Next]**.
4. You will then see a box with the code for downloading the file to be inserted into the notebook.
   For the file `file_name.csv` and for the user `user_name` it will look like this:

   ```
   select *
   from read_csv(
       "user_name/<id>_file_name.csv"
   )
   ```

   Press **[Add cell]** to add this cell to your notebook.

Once the cell is added to the notebook, the data from the file will be available for work.

Loading data from a file via the Python `tngri` module

To load data from non-local files, you can use the Python `tngri` module.

Loading data from a file via URL

Sample code at Python to load data from the `iris.csv` file located at the specified URL into Tengri:

```
import polars ①
import tngri

df = polars.read_csv(
    "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv" ②
)

tngri.upload_df(df) ③
```

① Import the required modules Python

② Read the file at the specified URL and write it to the `df` variable using the `polars` module

③ Upload data from `df` to Tengri using the function `tngri.upload_df`.

In addition to `.csv` files, this method can be used for other extensions — `.json`, `.xlsx` and others (see Polars data upload documentation for details).

After that, a message like this will appear in the output cell:

```
UploadedFile(s3_path='s3://<path>/<file_id>.parquet')
```

Now the data from the file is available for work. To work with them, you need to use the `read_parquet` function and specify the id of the loaded `.parquet` file from the previous step:

```
SELECT * FROM read_parquet('<file_id>.parquet');
```

The `.parquet` extension will be the extension of the file loaded in this way in any case, regardless of the extension of the initial file.

Loading data from a file saved in S3

Example code at Python to load into Tengri data from the `my_file.parquet` file located in your bucket S3:

```
import tngri ①

tngri.upload_s3(
    object="s3://my_folders/my_file.parquet", ②
    access_key="***",
    secret_key="***"
)
```

① Import module Python `tngri`.

② Set the parameters of your bucket S3 (file path and access keys)

The function `tngri.upload_s3` uploads the file from your S3 bucket to Tengri.

The data from the file will then be available for work.

The file extension of the file can be anything. It will remain the same as it was in the initial file. To work with different extensions you should use different functions.

To work with data from a file in our example, use the `read_parquet` function:

```
SELECT * FROM read_parquet('my_file.parquet');
```

If necessary, you can specify a path and name for the uploaded file inside Tengri via the optional `filename` parameter of the `tngri.upload_s3` function:
`filename="new_path/new_name.parquet"`.

Working with data from downloaded files

- Check that the loaded `.csv` file is available:

```
SELECT * FROM read_csv('customer_country.csv');
```

- Create a table with data from the loaded `.csv` file:

```
CREATE OR REPLACE TABLE customer_country AS
    SELECT * FROM read_csv('customer_country.csv');
```

- Load data from the `.csv` file into an existing table:

```
INSERT INTO customer_country
    SELECT * FROM read_csv('customer_country.csv');
```

Working with data from downloaded files of different extensions

To work with loaded files of other extensions, you need to use the corresponding functions.

- Create a table with data from loaded file `.parquet`:

```
CREATE OR REPLACE TABLE customer_country AS
    SELECT * FROM read_parquet('customer_country.parquet');
```

- Create a table with data from the loaded `.json` file:

```
CREATE OR REPLACE TABLE customer_country AS
    SELECT * FROM read_json('customer_country.json');
```

- Create a table with data from the loaded `.xlsx` file:

```
CREATE OR REPLACE TABLE customer_country AS
    SELECT * FROM read_xlsx('customer_country.json');
```

See also

- Function `read_json`

Automating data loading from multiple files

In this example, we will show how to load into Tengri data that is stored as multiple `.parquet` files in the storage of S3.

Using the Python `boto3` module, we will output the paths of all `.parquet` files in a particular storage:

```
import boto3
import boto3.s3.transfer

path = 'Stage/<lake_path>/DataLake/2049/'
s3_client = boto3.client('s3',
                         endpoint_url='***',
                         aws_access_key_id='***',
                         aws_secret_access_key='***',
                         region_name='us-east-1')

response = s3_client.list_buckets()

for bucket in response['Buckets']:
    print(f'Bucket: {bucket["Name"]}')
    response = s3_client.list_objects_v2(
    Bucket=bucket["Name"],
    Prefix='Stage')

    for content in response.get('Contents', []):
        file_path = str(content['Key'])
        if path in file_path:
            print(file_path)
```

```
Bucket: prostore
Stage/<lake_path>/DataLake/2049/1.parquet
Stage/<lake_path>/DataLake/2049/2.parquet
Stage/<lake_path>/DataLake/2049/3.parquet
Stage/<lake_path>/DataLake/2049/4.parquet
Stage/<lake_path>/DataLake/2049/5.parquet
Bucket: s3bench-asusnode21
```

Let's try querying SQL for data from one of these files and make sure that the query is processed and the data is displayed in the output:

```
SELECT *
    FROM read_parquet('s3://prostore/Stage/<lake_path>/DataLake/2049/1.parquet')
LIMIT 5;
```

```
+-------------------------------------------------------------------------------------------
---------------------------------+-------+---------------------+---------------------------+
----------+----------+----------------------------+------------------------+
| f1
| f2    | f3                  | f4                          | jd1      | jd2      |
command_load_datetime      | command_load_session_id |
+-------------------------------------------------------------------------------------------
---------------------------------+-------+---------------------+---------------------------+
```

```
----------+----------+---------------------------+------------------------+
| {"id": 673, "age": 77, "name": "d1b7d724ef3a8451189fc47da62b9888", "email":
"b32c9c5d3b1768e7955144103d72c593@example.com"} | false | 0001-01-02 06:44:06 | <memory at
0x7f0bb8523880> | 7000-1-1 | 7000-2-1 | 2025-06-23 08:05:23.636063 | 195
|
+-----------------------------------------------------------------------------------------------
---------------------------------+-------+--------------------+--------------------------+
----------+----------+---------------------------+------------------------+
| {"id": 221, "age": 34, "name": "4de35c863a2df55018eca39f130c0738", "email":
"3535ac8c3b04c33ac67ee70693707bdb@example.com"} | true  | 0001-01-02 02:59:53 | <memory at
0x7f0bb85219c0> | 7000-1-1 | 7000-2-1 | 2025-06-23 08:05:23.636063 | 195
|
+-----------------------------------------------------------------------------------------------
---------------------------------+-------+--------------------+--------------------------+
----------+----------+---------------------------+------------------------+
| {"id": 612, "age": 27, "name": "94a027942dffccb4abe3923914592ba5", "email":
"10178786f4378e98f62be257c4215056@example.com"} | false | 0001-01-02 11:53:53 | <memory at
0x7f0bb8522c80> | 7000-1-1 | 7000-2-1 | 2025-06-23 08:05:23.636063 | 195
|
+-----------------------------------------------------------------------------------------------
---------------------------------+-------+--------------------+--------------------------+
----------+----------+---------------------------+------------------------+
| {"id": 136, "age": 73, "name": "500522adeb682c1efcdce8af83610a27", "email":
"de507ff06310d2b35ea65c40f26f6a94@example.com"} | false | 0001-01-02 00:56:43 | <memory at
0x7f0bb8521fc0> | 7000-1-1 | 7000-2-1 | 2025-06-23 08:05:23.636063 | 195
|
+-----------------------------------------------------------------------------------------------
---------------------------------+-------+--------------------+--------------------------+
----------+----------+---------------------------+------------------------+
| {"id": 150, "age": 67, "name": "7d283d7c78b5692f6162d8b65baee1ed", "email":
"df78cebca3c40d06e8b43ad8a7e98f56@example.com"} | true  | 0001-01-02 13:46:41 | <memory at
0x7f0bb8521600> | 7000-1-1 | 7000-2-1 | 2025-06-23 08:05:23.636063 | 195
|
+-----------------------------------------------------------------------------------------------
---------------------------------+-------+--------------------+--------------------------+
----------+----------+---------------------------+------------------------+
```

Now we need to load the data from all the `.parquet` files into one table.

To do this, first initialise this table — create it, but do not write data to it yet, so that we can do it in the next step with a loop.

To initialise the table, execute this command with the `SELECT *` subquery for one of these files and the `LIMIT` — `0` parameter:

```
CREATE OR REPLACE TABLE raw_dyntest AS
    SELECT *
        FROM read_parquet('s3://prostore/Stage/<lake_path>/DataLake/2049/1.parquet')
    LIMIT 0;
```

```
Done in 22.2 sec

+--------+
| status |
+--------+
| CREATE |
+--------+
```

This query creates a table with the required set of columns and their types, now we can load data into it in a loop.

For this purpose, in the cell of type Python we will describe a loop by the names of all files and in this loop using the function `tngri.sql` we will iteratively load data from each file `.parquet` into the initialised table `raw_dyntest`.

In each iteration we will output the number of rows in this table that should increment.

```
import tngri

for i in range(1,6):
    file_name = f"s3://prostore/Stage/<lake_path>/DataLake/2049/{i}.parquet"
    insert_sql = f"INSERT INTO raw_dyntest SELECT * FROM read_parquet('{file_name}')"
    print(insert_sql)
    tngri.sql(insert_sql)

    print(tngri.sql("SELECT count(*) FROM raw_dyntest"))
```

```
insert into raw_dyntest select * from
read_parquet('s3://prostore/Stage/<lake_path>/DataLake/2049/1.parquet')
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 10000000 |
+----------+
insert into raw_dyntest select * from
read_parquet('s3://prostore/Stage/<lake_path>/DataLake/2049/2.parquet')
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 20000000 |
+----------+
insert into raw_dyntest select * from
read_parquet('s3://prostore/Stage/<lake_path>/DataLake/2049/3.parquet')
shape: (1, 1)
```

```
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 30000000 |
+----------+
insert into raw_dyntest select * from
read_parquet('s3://prostore/Stage/<lake_path>/DataLake/2049/4.parquet')
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 40000000 |
+----------+
insert into raw_dyntest select * from
read_parquet('s3://prostore/Stage/<lake_path>/DataLake/2049/5.parquet')
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 50000000 |
+----------+
```

Now let's display the number of rows in the table filled in this way:

```
SELECT count(*) AS row_count
    FROM raw_dyntest;
```

```
+-----------+
| row_count |
+-----------+
| 50000000  |
+-----------+
```

To check that queries to this data on such a volume (50 million rows) work properly, let's display the distribution of `true` and `false` values in the `f2` column:

```
SELECT f2, count(*) AS f2_cnt
    FROM raw_dyntest_ng
    GROUP BY f2;
```

```
+-------+----------+
| f2    | f2_cnt   |
```

```
+-------+----------+
| false | 24998577 |
+-------+----------+
| true  | 25001423 |
+-------+----------+
```

One of the columns of the table contains data in the form of JSON structure, and one of the keys of this structure is `age`. Let's build a table of age values distribution. To do this, use the function `json_extract`.

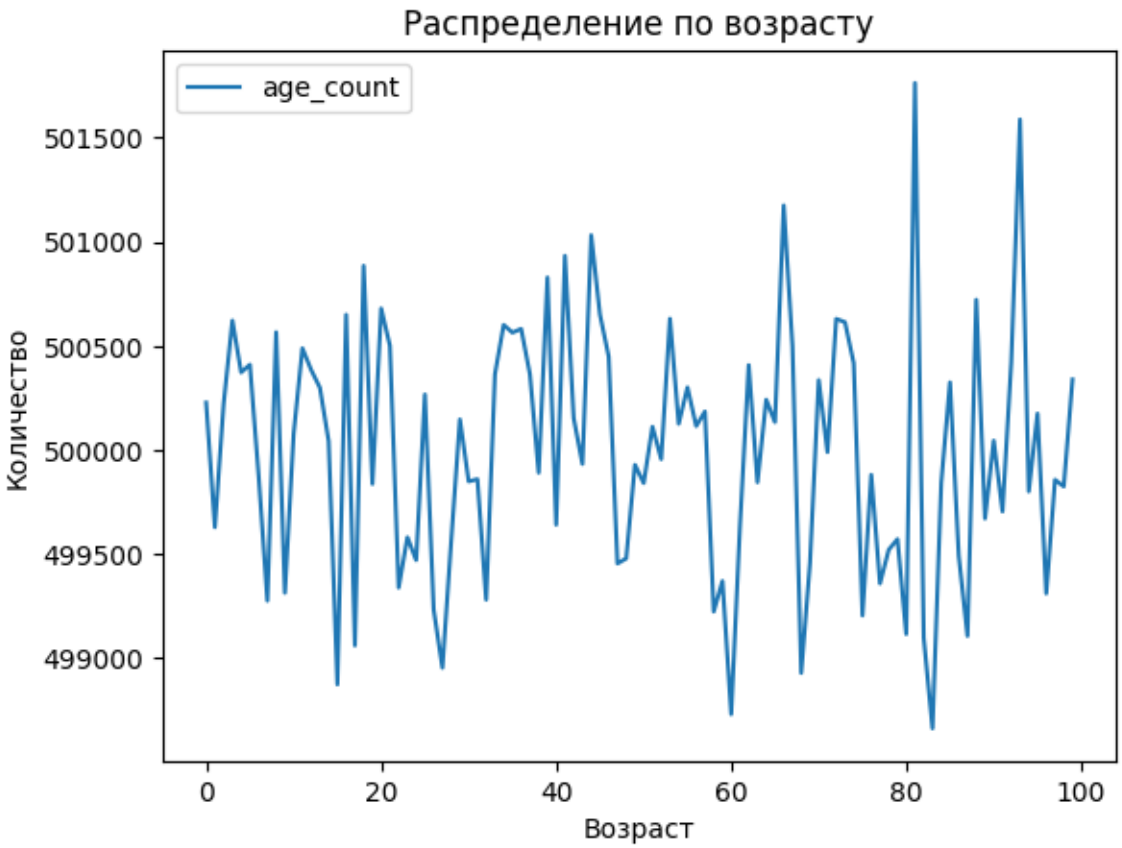
```
SELECT
    json_extract(f1, ['age'])[1]::INT AS age,
    count(*) AS age_count
    FROM raw_dyntest
    GROUP BY 1 ORDER BY 1;
```

```
Done in 9.9 sec

+-----+-----------+
| age | age_count |
+-----+-----------+
| 0   | 500228    |
+-----+-----------+
| 1   | 499629    |
+-----+-----------+
| 2   | 500222    |
+-----+-----------+
| 3   | 500622    |
+-----+-----------+
| 4   | 500373    |
+-----+-----------+
| 5   | 500410    |
+-----+-----------+
| 6   | 499876    |
+-----+-----------+
| 7   | 499276    |
+-----+-----------+
| 8   | 500566    |
+-----+-----------+
| 9   | 499314    |
+-----+-----------+
| ... | ...       |
+-----+-----------+
```

Based on this table, plot the distribution of values by age:

```
import matplotlib

plt = cell_output.plot(
    title='Распределение по возрасту',
```

```
    y='age_count',
    ylabel='Количество',
    x='age',
    xlabel='Возраст',
)

plt.get_figure()
```



Accessing Iceberg storage directly using Python

In this example, we will show you how to access the Iceberg repository directly using Python. Using this method, you will be able to read and edit existing tables, create new tables, including those based on data from existing tables, and perform other operations directly in Iceberg.

First of all, in a cell of the SQL type, we will create a test table with one column and write 1 million numbers starting from `1` into the rows. To do this, we will use the `unnest` and `generate_series` functions.

```
CREATE OR REPLACE TABLE demo.iceberg_test (numbers BIGINT);

INSERT INTO demo.iceberg_test (numbers)
    SELECT unnest(generate_series(1,1000000));
```

```
Done in 2.9 sec
```

```
+---------+
| Count   |
+---------+
| 1000000 |
+---------+
```

Let's display the resulting table:

```
SELECT * FROM demo.iceberg_test
    ORDER BY numbers
```

```
Done in 2.1 sec

+---------+
| numbers |
+---------+
| 1       |
+---------+
| 2       |
+---------+
| 3       |
+---------+
| 4       |
+---------+
| 5       |
+---------+
| ...     |
+---------+
999+ rows
```

Suppose we need to create a new table based on this table, which should contain all columns from the original table and one new column. In the new column we should write the data calculated using the function described at Python, passing the values from the columns of the original table as arguments.

As such a test function, let's describe the `check_odd` function, which returns the `odd` string for odd numbers and the `even` string for even numbers.

We will do all this by accessing Iceberg directly using Python:

```
import tngri
import pyarrow
import pyiceberg
from contextlib import suppress
import pandas


# Задаем имя исходной таблицы
source_table_name = 'demo.iceberg_test'
```

```python
# Задаем имя целевой таблицы
target_table_name = 'demo.iceberg_test_target'

# Задаем имя колонки в целевой таблице для записи новых данных
target_column = 'odd_or_even'

print(f'Source table: {source_table_name}')
print(f'Target table: {target_table_name}')

# Загружаем исходную таблицу
source_table = catalog.load_table(source_table_name)

# Создаем целевую таблицу, копируем схему из исходной
with suppress(Exception):
    catalog.drop_table(target_table_name)
sink = catalog.create_table(target_table_name, source_table.schema())

# Добавляем колону для записи новых данных (обязательно указать тип данных) ①
with sink.update_schema() as tx:
    tx.add_column(target_column, pyiceberg.schema.StringType())

# Тестовая функция для вычисления новых данных
def check_odd(num):
    if num % 2:
        return 'odd'
    else:
        return 'even'

# Делим исходную таблицу на батчи, чтобы не загружать в память целиком
table_batches = source_table.scan().to_arrow_batch_reader()

# Записываем новые данные по батчам
step = 0
for batch in table_batches:
    batch: pyarrow.RecordBatch

    step += 1
    print(f'Step: {step}')

    # Преобразуем батч в DataFrame
    part_df = batch.to_pandas()

    # Записываем новые данные в новую колонку DataFrame с тем же именем, что в таблице
    part_df[target_column] = part_df['numbers'].apply(lambda num: check_odd(num))

    # Выводим размер DataFrame с новыми данными
    print(f'Result dataframe shape: {part_df.shape}')

    # Преобразуем DataFrame обратно в таблицу
    sink_part = pyarrow.Table.from_pandas(part_df)

    # Аппендим данный батч в целевую таблицу
    sink.append(sink_part)
```

```
# Выводим длину целевой таблицы через scan().count()
print(f'Result table length: {sink.scan().count()}')
# Выводим длину целевой таблицы через tngri.sql
print(tngri.sql(f'SELECT count(*) FROM {target_table_name}')) ②
```

① Learn more about data types in pyiceberg here

② Detailed description of the function: `tngri.sql`

```
Done in 11.9 sec

Source table: demo.iceberg_test
Target table: demo.iceberg_test_target
Step: 1
Result dataframe shape: (16960, 2)
Step: 2
Result dataframe shape: (122880, 2)
Step: 3
Result dataframe shape: (122880, 2)
Step: 4
Result dataframe shape: (122880, 2)
Step: 5
Result dataframe shape: (122880, 2)
Step: 6
Result dataframe shape: (122880, 2)
Step: 7
Result dataframe shape: (122880, 2)
Step: 8
Result dataframe shape: (122880, 2)
Step: 9
Result dataframe shape: (122880, 2)
Result table length: 1000000
shape: (1, 1)
+----------+
| column_0 |
| ---      |
| i64      |
+----------+
| 1000000  |
+----------+
```

Let's check the target table. To do this, in a cell of type SQL we will display its first five rows, ordering the rows by the `numbers` column:

```
SELECT * FROM demo.iceberg_test_target
    ORDER BY numbers
    LIMIT 5
```

```
Done in 2.1 sec
```

```
+---------+-------------+
| numbers | odd_or_even |
+---------+-------------+
| 1 | odd |
+---------+-------------+
| 2 | even |
+---------+-------------+
| 3 | odd |
+---------+-------------+
| 4 | even |
+---------+-------------+
| 5 | odd |
+---------+-------------+
```

Analytical scenarios

- Analysing data from Instagram* posts
- Analysing geodata from sports trackers

Analysing data from Instagram* posts

* *Instagram belongs to Meta, a company recognised as extremist and banned in the Russian Federation.*

In this example, we will demonstrate how we can analyse data from the "instagram ads" dataset at Kaggle. The data in this dataset are posts from the social network Instagram with all sorts of attributes (comments, likes, audio and video information, etc.).

Downloading of source data

First of all, let's load the data. This can be done either manually via Upload Wizard, or by using, for example, this code at Python.

Here we set the URL of the dataset page to Kaggle, check that the downloaded file is a `.zip` file, and if it is, unzip its contents to a temporary directory, and then upload all the unzipped files to Tengri using the function `tngri.upload_file`. Output their information via `print` to get the names of the uploaded files for later use.

```
import urllib.request
import os
import zipfile
import tngri

zip_url = 'https://www.kaggle.com/api/v1/datasets/download/geraygench/instagram-ads'

urllib.request.urlretrieve(zip_url, 'data.zip')
temp_dir = 'temp'
try:
```

```
    with zipfile.ZipFile('data.zip', 'r') as zObject:
        zObject.extractall(path=temp_dir)
    for file in os.listdir(temp_dir):
        print(tngri.upload_file(os.path.join(temp_dir, file)))
except:
    print('Not a valid zip file')
```

```
<file_id>.json
```

Now let's copy the name of the saved `.json` file to paste it into the next cell as an argument to the `read_json` function.

Let's display the first 5 lines of the downloaded data to visually check that the download was successful. To do this, paste the file name copied in the previous step.

```
SELECT * FROM read_json("<file_id>.json")
    LIMIT 5
```

```
Done in 1.1 sec
+-----------------------------------------------+--------------------+---------
+-------------+-----+
| inputUrl                                      | id                 | type    | shortCode
| ... |
+-----------------------------------------------+--------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3351813129396710000 | Video   | C6ECEhTLlj7
| ... |
+-----------------------------------------------+--------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3335943316421300000 | Video   | C5LpsWOIUc8
| ... |
+-----------------------------------------------+--------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3319217982042080000 | Sidecar | C4QOysvtPUU
| ... |
+-----------------------------------------------+--------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3372181927023210000 | Sidecar | C7MZZiWNffn
| ... |
+-----------------------------------------------+--------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3369947094264730000 | Video   | C7EdQcJgkwr
| ... |
+-----------------------------------------------+--------------------+---------
+-------------+-----+
```

Now let's create a `instagram_post` table and load this data into it.

For those columns in which we found a nested JSON structure, we will specify the `JSON` type. For some of the columns, we will specify new names and data types that are convenient for us (for example, `TIMESTAMP`). Let's set a restriction that the value of the identifier `ig_post_id` must not be empty to reject potential invalid data fragments.

```
CREATE OR REPLACE TABLE raw_instagram_post AS
SELECT
    inputUrl,
    id AS ig_post_id,
    type,
    shortCode, caption, hashtags, mentions, url,
    dimensionsHeight, dimensionsWidth,
    commentsCount , firstComment,
    displayUrl, images, videoUrl,
    alt, likesCount,
    videoViewCount, videoPlayCount,
    timestamp::TIMESTAMP AS post_ts,
    ownerFullName, ownerUsername, ownerId,
    productType,
    videoDuration,
    isSponsored, isPinned,
    locationName, locationId,
    error, description, paidPartnership,
    sponsors::JSON AS sponsors,
    taggedUsers::JSON AS taggedUsers,
    musicInfo::JSON AS musicInfo,
    coauthorProducers::JSON AS coauthorProducers,
    latestComments::JSON AS latestComments,
    childPosts::JSON AS childPosts
FROM read_json("<file_id>.json")
    WHERE ig_post_id IS NOT NULL;
```

```
Done in 1.7 sec
+--------+
| status |
+--------+
| CREATE |
+--------+
```

Let's display the first 5 rows of the table to visually check that the data in it corresponds to our expectations.

```
SELECT * FROM raw_instagram_post
    LIMIT 5;
```

```
Done in 1.2 sec
+-----------------------------------------------+--------------------+---------
+-------------+-----+
```

```
| inputurl                                      | ig_post_id          | type    | shortcode
| ... |
+-----------------------------------------------+---------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3351813129396710000 | Video   | C6ECEhTLlj7
| ... |
+-----------------------------------------------+---------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3335943316421300000 | Video   | C5LpsWOIUc8
| ... |
+-----------------------------------------------+---------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3319217982042080000 | Sidecar | C4QOysvtPUU
| ... |
+-----------------------------------------------+---------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3372181927023210000 | Sidecar | C7MZZiWNffn
| ... |
+-----------------------------------------------+---------------------+---------
+-------------+-----+
| https://www.instagram.com/fjallravenofficial/ | 3369947094264730000 | Video   | C7EdQcJgkwr
| ... |
+-----------------------------------------------+---------------------+---------
+-------------+-----+
```

Data preparation

Now let's analyse the nested JSON structure in the `musicinfo` column using the `unnest` and `json_keys` functions:

```
SELECT unnest(keys) AS musicinfo_keys, keys_count FROM
    (SELECT json_keys(musicinfo) AS keys, count(*) keys_count
        FROM raw_instagram_post WHERE musicinfo IS NOT NULL GROUP BY 1) t ;
```

```
Done in 1.1 sec
+--------------------------+------------+
| musicinfo_keys           | keys_count |
+--------------------------+------------+
| artist_name              | 1176       |
+--------------------------+------------+
| song_name                | 1176       |
+--------------------------+------------+
| uses_original_audio      | 1176       |
+--------------------------+------------+
| should_mute_audio        | 1176       |
+--------------------------+------------+
| should_mute_audio_reason | 1176       |
+--------------------------+------------+
| audio_id                 | 1176       |
```

```
+-------------------------+------------+
```

Let's extract the data from the JSON structure using the function `json_extract` and write it into a new table `silver_instagram_musicinfo`, the columns of which are the keys from the original JSON structure (inside `musicinfo`) that we saw in the previous step, and in the column `ig_post_id` we will write the id from the original table `instagram_post`.

Note that we use the `json_extract` function here no more than once for one column of the source table. This saves time and computational power, because after a single extraction of this data, it will be accessed almost instantly.

```
CREATE OR REPLACE TABLE silver_instagram_musicinfo AS
SELECT DISTINCT ig_post_id,
    arr[1] AS 'artist_name',
    arr[2] AS 'song_name',
    arr[3]::bool AS 'uses_original_audio',
    arr[4]::bool AS 'should_mute_audio',
    arr[5] AS 'should_mute_audio_reason',
    arr[6]::bigint AS 'audio_id'
FROM (SELECT ig_post_id, musicinfo,
    json_extract(musicinfo, ['artist_name',
                             'song_name',
                             'uses_original_audio',
                             'should_mute_audio',
                             'should_mute_audio_reason',
                             'audio_id'])
                             arr
    FROM raw_instagram_post WHERE musicinfo IS NOT NULL
) e;
```

```
Done in 1.1 sec
+--------+
| status |
+--------+
| CREATE |
+--------+
```

Let's display the first five rows from the resulting table to visually check the data in it.

```
SELECT * FROM silver_instagram_musicinfo
    LIMIT 5;
```

```
Done in 1 sec
+--------------------+----------------------+--------------------------
+--------------------+------------------+--------------------------+------------------+
| ig_post_id         | artist_name          | song_name                |
uses_original_audio | should_mute_audio | should_mute_audio_reason | audio_id         |
```

```
+--------------------+---------------------+--------------------------
+--------------------+-------------------+--------------------------+------------------+
| 3361389062866375005 | "fjallravenofficial" | "Original audio"          | true
| false             | ""                    | 1091589181917200 |
+--------------------+---------------------+--------------------------
+--------------------+-------------------+--------------------------+------------------+
| 3362166784222944940 | "Lyle Workman"       | "Sun Beyond the Darkness" | false
| false             | ""                    | 700744903595866  |
+--------------------+---------------------+--------------------------
+--------------------+-------------------+--------------------------+------------------+
| 3370702174072339576 | "victoriassecret"    | "Original audio"          | true
| false             | ""                    | 969258887793092  |
+--------------------+---------------------+--------------------------
+--------------------+-------------------+--------------------------+------------------+
| 3365659145996406849 | "victoriassecret"    | "Original audio"          | true
| false             | ""                    | 1856092344873079 |
+--------------------+---------------------+--------------------------
+--------------------+-------------------+--------------------------+------------------+
| 3367926439711140761 | "asos"               | "Original audio"          | true
| false             | ""                    | 1428332318047391 |
+--------------------+---------------------+--------------------------
+--------------------+-------------------+--------------------------+------------------+
```

Now let's look at the column of the source table with information about tagged users `taggedUsers`. Note that the data in it is represented as arrays. Using the functions `unnest` and `json_extract`, create the table `silver_instagram_taggeduser`:

```
CREATE OR REPLACE TABLE silver_instagram_taggeduser AS
    SELECT DISTINCT ig_post_id,
        arr[1]::varchar AS 'full_name',
        arr[2]::bigint AS 'user_id',
        arr[3]::bool AS 'is_verified',
        arr[4] AS 'profile_pic_url',
        arr[5] AS 'username'
    FROM (
        SELECT ig_post_id,
        json_extract(users, ['full_name',
                             'id','is_verified',
                             'profile_pic_url',
                             'username'])
        AS arr FROM
            (SELECT ig_post_id, unnest(json_extract(taggedUsers, '[*]'))
            AS users FROM
                raw_instagram_post WHERE taggedUsers IS NOT NULL
            ) un
        ) t;
```

```
Done in 1 sec
+--------+
| status |
```

```
+--------+
| CREATE |
+--------+
```

Let's display the first 5 rows of the created table `silver_instagram_taggeduser` and visually check the data in it.

```
SELECT * FROM silver_instagram_taggeduser
    LIMIT 5;
```

```
Done in 1 sec
+--------------------+------------------+----------+-------------
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--------------------------+--------------------+
| ig_post_id          | full_name         | user_id   | is_verified | profile_pic_url
| username           |
+--------------------+------------------+----------+-------------
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--------------------------+--------------------+
| 3368467612563246843 | "Candice"         | 43114731  | true        |
"https://scontent.cdninstagram.com/v/t51.2885-
19/436426783_716232097252223_2080875243016028171_n.jpg?stp=dst-
jpg_s150x150&_nc_ht=scontent.cdninstagram.com&_nc_cat=1&_nc_ohc=VZz3Y6G6kCgQ7kNvgEfJgX3&edm=A
Ps17CUBAAAA&ccb=7-5&oh=00_AYDeE3YwxCQxVEK-
3DLy0AK8cSmaFu3SVuwsUluh0pQ2Hg&oe=66526CB7&_nc_sid=10d13b"   | "candiceswanepoel" |
+--------------------+------------------+----------+-------------
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--------------------------+--------------------+
| 3371457031159064630 | "Brittany Xavier" | 739181077 | true        |
"https://scontent.cdninstagram.com/v/t51.2885-
19/299498992_480162533527636_8112607220645623732_n.jpg?stp=dst-
jpg_s150x150&_nc_ht=scontent.cdninstagram.com&_nc_cat=104&_nc_ohc=D83lh8W9o-
EQ7kNvgEAZo1_&edm=APs17CUBAAAA&ccb=7-5&oh=00_AYCI_FerVMUvMEltLhAaANlfFdtezWw90dIOCff-
oNuiLQ&oe=66526DD9&_nc_sid=10d13b" | "brittanyxavier"   |
+--------------------+------------------+----------+-------------
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--------------------------+--------------------+
| 3350535229969664632 | "dev scudds"      | 200671402 | false       |
"https://scontent.cdninstagram.com/v/t51.2885-
19/316332449_539815587596795_7700370950656909913_n.jpg?stp=dst-
jpg_s150x150&_nc_ht=scontent.cdninstagram.com&_nc_cat=111&_nc_ohc=yt51b9U7YQcQ7kNvgGr0PqA&edm
=APs17CUBAAAA&ccb=7-
```

```
5&oh=00_AYC_DbQKZ56A_C_kK4oCamtB10dpgXWxNr9dupqkTHIKZA&oe=665259D4&_nc_sid=10d13b" |
"devscudds"        |
+--------------------+------------------+-----------+-------------
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------------+--------------------+
| 3355576548572106408 | "Roxie Nafousi"   | 182064334 | true        |
"https://scontent.cdninstagram.com/v/t51.2885-
19/432688376_299228906523099_7608649815950644359_n.jpg?stp=dst-
jpg_s150x150&_nc_ht=scontent.cdninstagram.com&_nc_cat=1&_nc_ohc=mNDTJLE0sakQ7kNvgFoinRd&edm=A
Ps17CUBAAAA&ccb=7-5&oh=00_AYDyLDXyYkSqF7jeJ7SDKO-
ATM7WbbH8OYqhw_GOjiGygQ&oe=665269218_nc_sid=10d13b"   | "roxienafousi"     |
+--------------------+------------------+-----------+-------------
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------------+--------------------+
| 3369046462609537802 | "ASOS MAN"        | 640874843 | true        |
"https://scontent.cdninstagram.com/v/t51.2885-
19/244277018_902025494057602_3929487210907869376_n.jpg?stp=dst-
jpg_s150x150&_nc_ht=scontent.cdninstagram.com&_nc_cat=110&_nc_ohc=5aRpL0ZaRW8Q7kNvgFG0_9g&edm
=APs17CUBAAAA&ccb=7-5&oh=00_AYBmZ6qVQtwRziMSsl8tGtlX7u4-RnYXa-
MxQIaIEKyc7Q&oe=66527A51&_nc_sid=10d13b" | "asos_man"         |
+--------------------+------------------+-----------+-------------
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------------+--------------------+
```

Now let's look at the column of the source table with information about comments `latestComments`. In it, the data is represented as an array of comments. To find out what keys occur inside these arrays, let's use the functions `unnest`, `json_keys` and `json_extract` and execute this query:

```
SELECT unnest(keys) AS comment_keys FROM
    (SELECT json_keys(comments) AS keys, count(*) FROM
        (SELECT ig_post_id, unnest(json_extract(latestComments, '[*]'))
        AS comments FROM
            raw_instagram_post WHERE latestComments IS NOT NULL) un GROUP BY 1) t;
```

```
Done in 1 sec
+--------------------+
| comment_keys       |
+--------------------+
| id                 |
+--------------------+
| text               |
+--------------------+
| ownerUsername      |
+--------------------+
```

```
| ownerProfilePicUrl |
+--------------------+
| timestamp          |
+--------------------+
| likesCount         |
+--------------------+
| repliesCount       |
+--------------------+
| replies            |
+--------------------+
```

Let's extract the comments into the `silver_instagram_comments` table in the same way as we did above for users and artists.

Note that in this table `likesCount` and `repliesCount` are the number of likes and replies to this comment (not to the original post).

```
CREATE OR REPLACE TABLE silver_instagram_comments as
    SELECT DISTINCT ig_post_id,
        arr[1]::bigint AS comment_id,
        arr[2] AS 'text',
        arr[3] AS 'ownerUsername',
        arr[4] AS 'ownerProfilePicUrl',
        arr[5]::timestamp AS 'comment_ts',
        arr[6]::int AS 'likesCount',
        arr[7]::int AS 'repliesCount',
        arr[8] AS 'replies'
FROM (
SELECT ig_post_id,
    json_extract(comments, ['id','text','ownerUsername','ownerProfilePicUrl',
    'timestamp','likesCount','repliesCount','replies']) AS arr
    FROM (SELECT ig_post_id, unnest(json_extract(latestComments, '[*]')) AS comments
        FROM raw_instagram_post WHERE latestComments IS NOT NULL) un ) t;
```

```
Done in 1 sec
+--------+
| status |
+--------+
| CREATE |
+--------+
```

Let's check the resulting table:

```
SELECT * FROM silver_instagram_comments
    LIMIT 5;
```

```
Done in 1 sec
+--------------------+------------------
```

```
+--------------------------------------------------------------------
+----------------------------------+-----+
| ig_post_id          | comment_id        | text
| ownerusername                    | ... |
+---------------------+------------------
+--------------------------------------------------------------------
+----------------------------------+-----+
| 3351813129396705531 | 18025641284105656 | "Your work apron is cool..\nDo Fjalraven sell
them to the public?" | "dai_stick"                      | ... |
+---------------------+------------------
+--------------------------------------------------------------------
+----------------------------------+-----+
| 3367108632665594085 | 18010752380180096 | "Well done "
| "alpencore"                      | ... |
+---------------------+------------------
+--------------------------------------------------------------------
+----------------------------------+-----+
| 3362086677098626407 | 18041534590828292 | "Still made in cheap labour countries"
| "leineleineleineleineleineleine" | ... |
+---------------------+------------------
+--------------------------------------------------------------------
+----------------------------------+-----+
| 3362086677098626407 | 18028710974031364 | ""
| "arifwidie26"                    | ... |
+---------------------+------------------
+--------------------------------------------------------------------
+----------------------------------+-----+
| 3361389062866375005 | 18012200105461376 | "Classic description of type 2 fun"
| "itskate_g"                      | ... |
+---------------------+------------------
+--------------------------------------------------------------------
+----------------------------------+-----+
```

Result 1: Popularity of performers

We now use the data collected above to construct a table in which each row corresponds to one artist, and the columns represent different information that somehow indicates the popularity of that artist:

- `songs` — the number of songs by this artist
- `ig_posts` — the number of posts with songs by this artist
- `comments` — number of comments in posts with songs by this artist
- `tagged_users` — number of tagged users in posts with songs by this artist
- `likesCount` — number of likes in posts with songs by this artist
- `repliesCount` — number of replies in posts with songs by this artist

To do this, we use the expression `LEFT JOIN`.

Organise the final table by the third column (`ig_posts`) in descending order.

```
SELECT artist_name,
    count(distinct song_name) AS songs,
    count(distinct im.ig_post_id) AS ig_posts,
    count(distinct comment_id) AS comments,
    count(distinct user_id) AS tagged_users,
    sum(likesCount) AS likesCount,
    sum(repliesCount) AS repliesCount
FROM silver_instagram_musicinfo im
    LEFT JOIN silver_instagram_taggeduser tu ON tu.ig_post_id=im.ig_post_id
    LEFT JOIN silver_instagram_comments ic ON ic.ig_post_id =im.ig_post_id
GROUP BY 1 ORDER BY 3 DESC;
```

```
Done in 1.6 sec
+------------------+-------+----------+----------+-------------+-----------+-------------
+
| artist_name      | songs | ig_posts | comments | tagged_users | likescount | repliescount
|
+------------------+-------+----------+----------+-------------+-----------+-------------
+
| "telfarglobal"   | 9     | 136      | 985      | 120          | 634        | 156
|
+------------------+-------+----------+----------+-------------+-----------+-------------
+
| "everlane"       | 1     | 95       | 532      | 6            | 513        | 65
|
+------------------+-------+----------+----------+-------------+-----------+-------------
+
| "asos"           | 3     | 93       | 503      | 60           | 583        | 85
|
+------------------+-------+----------+----------+-------------+-----------+-------------
+
| "victoriassecret" | 1     | 83       | 686      | 49           | 185        | 16
|
+------------------+-------+----------+----------+-------------+-----------+-------------
+
| "summerfridays"  | 2     | 65       | 574      | 48           | 173        | 18
|
+------------------+-------+----------+----------+-------------+-----------+-------------
+
| ...              | ...   | ...      | ...      | ...          | ...        | ...
|
+------------------+-------+----------+----------+-------------+-----------+-------------
+
```

This table can be saved in `.xlsx` format by clicking the **[Download output as XLSX]** button on this cell. After that, in any application for viewing `.xlsx' files, you can organise the collected data by any column, apply additional filters, copy the data or a part of it for inserting into analytical reports.

The file saved in this way can be download here.

Result 2: Distribution of comments and events in them over time

Let’s analyse the distribution of comments and events in them (likes and repins) depending on the time elapsed from the publication of a post to the publication of a given comment on it — the "freshness" of the comment in relation to the post to which it refers.

To do this, let’s build a table in which each row will correspond to all such pairs "post — comment to it" that 0 days, 1 day and so on up to 9 days have passed between them. And for all such pairs we will output them in columns:

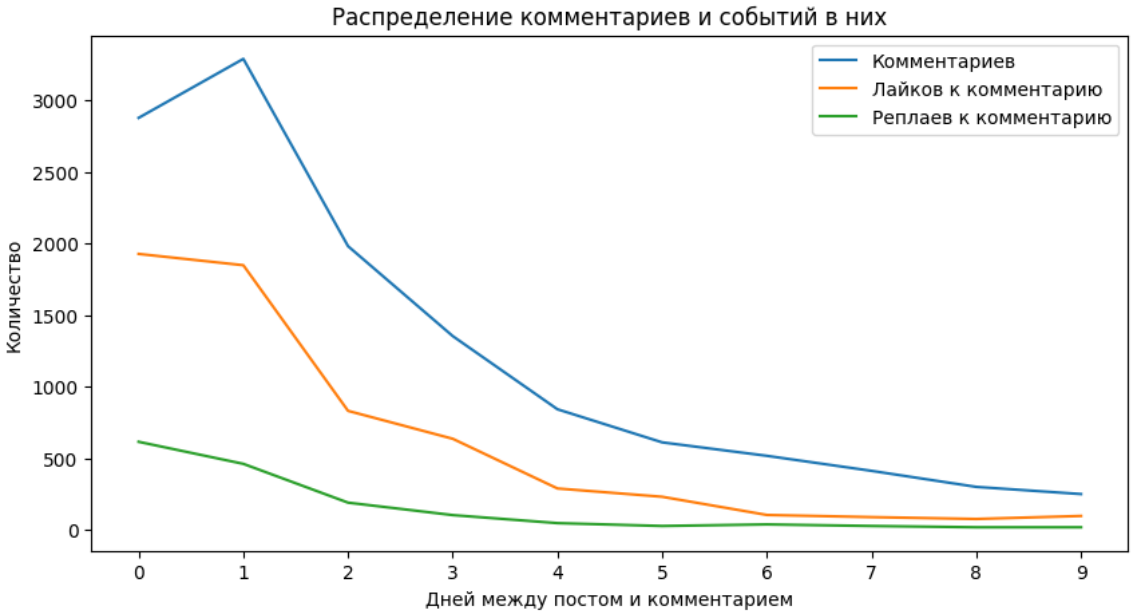
- `comments` — the number of such comments
- `likescount` — the number of likes of such comments.
- `repliescount` — the number of replays for such comments

```
SELECT
    (comment_ts::DATE - post_ts::DATE)::INT AS days_till_comment,
    count(DISTINCT comment_id) AS comments,
    sum(c.likesCount)::int AS likesCount,
    sum(c.repliesCount)::int AS repliesCount
FROM raw_instagram_post p
LEFT JOIN silver_instagram_comments c  on p.ig_post_id=c.ig_post_id
GROUP BY 1 ORDER BY 1 LIMIT 10;
```

```
Done in 1.7 sec
+-------------------+----------+------------+--------------+
| days_till_comment | comments | likescount | repliescount |
+-------------------+----------+------------+--------------+
| 0                 | 2876     | 1926       | 614          |
+-------------------+----------+------------+--------------+
| 1                 | 3288     | 1847       | 460          |
+-------------------+----------+------------+--------------+
| 2                 | 1980     | 830        | 189          |
+-------------------+----------+------------+--------------+
| 3                 | 1352     | 635        | 102          |
+-------------------+----------+------------+--------------+
| 4                 | 841      | 288        | 46           |
+-------------------+----------+------------+--------------+
| 5                 | 610      | 230        | 26           |
+-------------------+----------+------------+--------------+
| 6                 | 516      | 103        | 37           |
+-------------------+----------+------------+--------------+
| 7                 | 411      | 88         | 26           |
+-------------------+----------+------------+--------------+
| 8                 | 299      | 75         | 17           |
+-------------------+----------+------------+--------------+
| 9                 | 249      | 96         | 17           |
+-------------------+----------+------------+--------------+
```

Now in the cell of type Python we will plot the graph using the data from this table. To do this, we will use the Python `matplotlib` module. To take data from the previous cell, we use the `cell_output` variable.

```
import matplotlib

plt = cell_output.plot(
    title='Распределение комментариев и событий в них',
    y=[
        'comments',
        'likescount',
        'repliescount',
    ],
    label = ['Комментариев', 'Лайков к комментарию', 'Реплаев к комментарию'],
    ylabel='Количество',
    x='days_till_comment',
    xlabel = 'Дней между постом и комментарием',
    figsize=(10,5),
)
plt.set_xticks(cell_output['days_till_comment'])
plt.get_figure()
```



The graph clearly shows that the number of comments to a post increases on the first day and then gradually decreases. You can also see how the activity within the comments themselves (likes and repins to comments) decreases depending on the obsolescence of the comment relative to the date of the post.

Analysing geodata from sports trackers

In this example we will show how Tengri can be used to analyse geodata from sports trackers such as Garmin, Strava and others.

The most popular formats for recording and storing geodata in sports trackers are `.gpx` and `.fit`. In addition to GPS-coordinates of points (longitude, latitude, altitude), these formats also store the time the athlete travelled to a given point and other information about the athlete and his equipment (heart rate,

power, pedal speed, speed, temperature, etc.).

As a dataset for this example we will take an upload of the archive of all data from one Strava profile as of 9 September 2025. To learn how to do such an upload, see here.

The Strava data upload is an archive containing, among other things, an `activities` folder with all activities of a given user in the form of `.gpx` or `.fit` files (depending on the devices used for track recording) and an `activities.csv` file, which contains additional information from Strava service (activity name, activity type, comments, various calculated indicators, etc.).

In total, there are 2190 activities (tracks) in our dataset, some of which are represented as `.gpx` files, and some as `.fit` files.

Loading raw data

First, let's load data. To do this in a cell of type Python let's do the following:

1. Upload a file from the file sharing service
2. Unzip the `.zip` archive and the `.gz` files inside it
3. Set the column names in the two tables to be created — for all tracks and for all points from all tracks
4. describe the auxiliary functions
5. We loop through all the activity files and do parsing of data from them, and parsing will be different for `.gpx` and `.fit` files. For parsing we will use the handy modules Python `gpxpy` and `fitdecode`.
6. We write the generated DataFrames into tables using the function `tngri.sql`

```
import os
import zipfile
import tngri
import gdown
import gpxpy
import pandas as pd
import gzip
import shutil
import fitdecode
import datetime
from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="tengri")

start = datetime.datetime.now()

①
# Upload the source files
google_drive_url = 'https://drive.google.com/uc?id=1yFycFAGv00WPPGvr65pCRSeHDJM9wol6'
output = 'data.zip'
gdown.download(google_drive_url, output, quiet=False)

②
# Unzip the .zip
temp_dir = 'temp'
```

```
try:
    with zipfile.ZipFile('data.zip', 'r') as zObject:
        zObject.extractall(path=temp_dir)
except Exception as e:
    print(f'Error: {e}')

downloaded = len(os.listdir(temp_dir))
print(f'Downloaded files total: {downloaded}')

# Unpack the .gz
unpacked = 0
for file in os.listdir(temp_dir):
    file_path = os.path.join(temp_dir, file)
    if file.endswith('.gz'):
        unpacked += 1
        with gzip.open(file_path, 'rb') as f_in:
            with open(file_path[:-3], 'wb') as f_out:
                shutil.copyfileobj(f_in, f_out)

print(f'Unpacked .gz files total: {unpacked}')
print(f'Time from start: {str(datetime.datetime.now() - start)}')

③
# Columns for the track table
tracks_dict = {
    'track_id':[],
    'track_name':[],
    'track_length_km':[],
    'track_first_coord_lat':[],
    'track_first_coord_lon':[],
    'country':[]
    }
# Columns for the points table
points_dict = {
    'track_id':[],
    'latitude':[],
    'longitude':[],
    'elevation':[],
    'heart_rate':[],
    'temperature':[],
    'cadence':[],
    'power':[],
    'time':[],
    'speed':[]
    }

④
# Get extension from point for .gpx
def ext_from_point(point, type):
    res = pd.NA
    for ext in point.extensions:
        ext_list = [el.text for el in ext if type in el.tag]
        if len(ext_list) == 1:
```

```
            res = ext_list[0]
    return res

# Get power from point for .gpx
def power_from_point(point):
    res = pd.NA
    for ext in point.extensions:
        if ext.tag == 'power':
            res = ext.text
    return res

# Coordinate conversion for .fit
def convert_coord(coord):
    if pd.isna(coord):
        return coord
    else:
        return coord/((2**32)/360)

# Convert speed from m/s to km/h
def convert_speed(speed):
    if pd.isna(speed):
        return speed
    else:
        return speed*3.6

# Determine country by coordinates
def get_country(coords):
    location = geolocator.reverse(f'{coords[0]}, {coords[1]}')
    if not location is None:
        return location.raw['address']['country']
    else:
        return 'Unknown'

⑤
# Start parsing gpx and fit
files_done = 0
for file in os.listdir(temp_dir):
    file_path = os.path.join(temp_dir, file)

    if file.endswith('.gpx'):
        files_done += 1
        with open(file_path) as f:
            gpx = gpxpy.parse(f)
            if not len(gpx.tracks) == 1:
                print(f'Not 1 track in file: {file_path}')

            cur_length_2d_km = gpx.tracks[0].segments[0].length_2d()/1000
            track_first_lat = gpx.tracks[0].segments[0].points[0].latitude
            track_first_lon = gpx.tracks[0].segments[0].points[0].longitude

            tracks_dict['track_first_coord_lat'].append(track_first_lat)
            tracks_dict['track_first_coord_lon'].append(track_first_lon)
            tracks_dict['track_id'].append(file)
```

```
            tracks_dict['track_name'].append(gpx.tracks[0].name)
            tracks_dict['track_length_km'].append(round(cur_length_2d_km, 2))
            tracks_dict['country'].append(get_country((track_first_lat, track_first_lon)))

            for point in gpx.tracks[0].segments[0].points:
                points_dict['track_id'].append(file)
                points_dict['latitude'].append(point.latitude)
                points_dict['longitude'].append(point.longitude)
                points_dict['elevation'].append(point.elevation)
                points_dict['heart_rate'].append(ext_from_point(point, 'hr'))
                points_dict['temperature'].append(ext_from_point(point, 'atemp'))
                points_dict['cadence'].append(ext_from_point(point, 'cad'))
                points_dict['power'].append(power_from_point(point))
                points_dict['time'].append(point.time)
                # There is no instantaneous speed in gpx
                points_dict['speed'].append(pd.NA)

    elif file.endswith('.fit'):
        files_done += 1
        track_first_lat = 0
        track_first_lon = 0
        with fitdecode.FitReader(file_path) as fit_file:
            tracks_dict['track_id'].append(file)
            tracks_dict['track_name'].append(pd.NA)
            distance = 0
            for frame in fit_file:
                if not isinstance(frame, fitdecode.records.FitDataMessage):
                    continue
                if not frame.name == 'record':
                    continue
                if not (frame.has_field('position_lat') and
frame.has_field('position_long')):
                    continue
                cur_lat = convert_coord(frame.get_value('position_lat', fallback=pd.NA))
                cur_lon = convert_coord(frame.get_value('position_long', fallback=pd.NA))

                points_dict['track_id'].append(file)
                points_dict['latitude'].append(cur_lat)
                points_dict['longitude'].append(cur_lon)
                points_dict['elevation'].append(frame.get_value('altitude', fallback=pd.NA))
                #points_dict['heart_rate'].append(frame.get_value('heart_rate',
fallback=pd.NA))
                points_dict['heart_rate'].append(pd.NA)
                points_dict['temperature'].append(frame.get_value('temperature',
fallback=pd.NA))
                #points_dict['cadence'].append(frame.get_value('cadence', fallback=pd.NA))
                points_dict['cadence'].append(pd.NA)
                points_dict['power'].append(frame.get_value('power', fallback=pd.NA))
                points_dict['time'].append(frame.get_value('timestamp'))
                points_dict['speed'].append(convert_speed(frame.get_value('enhanced_speed',
fallback=pd.NA)))

                cur_dist = frame.get_value('distance', fallback=pd.NA)
```

```
                if type(cur_dist) == float:
                    distance = round((cur_dist/1000), 2)
                if track_first_lat == 0 and not pd.isna(cur_lat):
                    track_first_lat = cur_lat
                if track_first_lon == 0 and not pd.isna(cur_lon):
                    track_first_lon = cur_lon
            tracks_dict['track_length_km'].append(distance)
            tracks_dict['track_first_coord_lat'].append(track_first_lat)
            tracks_dict['track_first_coord_lon'].append(track_first_lon)
            tracks_dict['country'].append(get_country((track_first_lat, track_first_lon)))

    if files_done > 0 and files_done % 100 == 0 and file.endswith(('.fit', '.gpx')):
        print(f'Files done: {files_done}/{downloaded} \
        Points: {len(points_dict['track_id'])} \
        Time from start: {str(datetime.datetime.now() - start)}')

# Write the collected dictionaries to the DataFrame
df_tracks = pd.DataFrame.from_dict(tracks_dict)
df_points = pd.DataFrame.from_dict(points_dict)

df_points['time_text'] = df_points['time'].astype(str)
df_points = df_points.drop(columns=['time'])

# Table names to write in
table_names = ['demo_geo_tracks', 'demo_geo_points']

⑥
# Write the generated DataFrames to the tables
for frame, table_name in zip([df_tracks, df_points], table_names):
    print(f'Creating table: {table_name}')
    print(f'Size: {frame.shape}')

    file_name = tngri.upload_df(frame)
    tngri.sql(f"CREATE OR REPLACE TABLE {table_name} AS SELECT * FROM
read_parquet('{file_name}')")

print(f'Finished. Time from start: {str(datetime.datetime.now() - start)}')
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1yFycFAGv00WPPGvr65pCRSeHDJM9wol6
From (redirected):
https://drive.google.com/uc?id=1yFycFAGv00WPPGvr65pCRSeHDJM9wol6&confirm=t&uuid=4e852907-
5eeb-4ff6-ab4e-d166d6d45ecd
To: /home/python/data.zip

  0%|          | 0.00/243M [00:00<?, ?B/s]
  0%|          | 1.05M/243M [00:00<00:23, 10.5MB/s]
  2%|▏         | 4.72M/243M [00:00<00:09, 25.4MB/s]
  4%|▍         | 10.5M/243M [00:00<00:05, 38.9MB/s]
  6%|▌         | 14.7M/243M [00:00<00:07, 29.6MB/s]
  8%|▊         | 19.9M/243M [00:00<00:06, 35.6MB/s]
```

```
 10%|▉         | 24.1M/243M [00:00<00:05, 37.3MB/s]
 12%|▉▉        | 28.8M/243M [00:00<00:05, 39.1MB/s]
 14%|▉▉        | 33.6M/243M [00:00<00:05, 40.5MB/s]
 16%|▉▉        | 37.7M/243M [00:01<00:05, 39.9MB/s]
 17%|▉▉        | 41.9M/243M [00:01<00:04, 40.2MB/s]
 19%|▉▉        | 47.2M/243M [00:01<00:04, 43.5MB/s]
 21%|▉▉▉       | 51.9M/243M [00:01<00:04, 43.6MB/s]
 23%|▉▉▉       | 56.6M/243M [00:01<00:04, 43.3MB/s]
 25%|▉▉▉       | 61.3M/243M [00:01<00:04, 43.5MB/s]
 28%|▉▉▉       | 67.1M/243M [00:01<00:03, 46.3MB/s]
 30%|▉▉▉       | 71.8M/243M [00:01<00:03, 46.1MB/s]
 32%|▉▉▉▉      | 78.6M/243M [00:01<00:03, 48.7MB/s]
 35%|▉▉▉▉      | 84.4M/243M [00:02<00:03, 50.9MB/s]
 37%|▉▉▉▉      | 89.7M/243M [00:02<00:03, 49.9MB/s]
 39%|▉▉▉▉      | 94.9M/243M [00:02<00:02, 50.6MB/s]
 41%|▉▉▉▉      | 100M/243M [00:02<00:02, 49.2MB/s]
 43%|▉▉▉▉▉     | 105M/243M [00:02<00:02, 47.4MB/s]
 46%|▉▉▉▉▉     | 111M/243M [00:02<00:02, 48.9MB/s]
 48%|▉▉▉▉▉     | 116M/243M [00:02<00:02, 47.3MB/s]
 50%|▉▉▉▉▉     | 122M/243M [00:02<00:02, 46.5MB/s]
 52%|▉▉▉▉▉▉    | 126M/243M [00:02<00:02, 43.4MB/s]
 54%|▉▉▉▉▉▉    | 132M/243M [00:03<00:02, 43.6MB/s]
 57%|▉▉▉▉▉▉    | 137M/243M [00:03<00:02, 45.7MB/s]
 58%|▉▉▉▉▉▉    | 142M/243M [00:03<00:02, 43.9MB/s]
 60%|▉▉▉▉▉▉    | 147M/243M [00:03<00:02, 40.4MB/s]
 62%|▉▉▉▉▉▉▉   | 152M/243M [00:03<00:02, 42.0MB/s]
 64%|▉▉▉▉▉▉▉   | 156M/243M [00:03<00:02, 40.9MB/s]
 66%|▉▉▉▉▉▉▉   | 160M/243M [00:03<00:02, 40.4MB/s]
 68%|▉▉▉▉▉▉▉   | 165M/243M [00:03<00:01, 39.9MB/s]
 69%|▉▉▉▉▉▉▉   | 169M/243M [00:03<00:01, 39.6MB/s]
 71%|▉▉▉▉▉▉▉   | 173M/243M [00:04<00:01, 36.9MB/s]
 73%|▉▉▉▉▉▉▉▉  | 177M/243M [00:04<00:02, 29.7MB/s]
 75%|▉▉▉▉▉▉▉▉  | 181M/243M [00:04<00:01, 31.9MB/s]
 76%|▉▉▉▉▉▉▉▉  | 186M/243M [00:04<00:01, 33.8MB/s]
 78%|▉▉▉▉▉▉▉▉  | 190M/243M [00:04<00:01, 35.4MB/s]
 80%|▉▉▉▉▉▉▉▉  | 194M/243M [00:04<00:01, 36.6MB/s]
 82%|▉▉▉▉▉▉▉▉▉ | 198M/243M [00:04<00:01, 37.3MB/s]
 83%|▉▉▉▉▉▉▉▉▉ | 202M/243M [00:04<00:01, 38.0MB/s]
 85%|▉▉▉▉▉▉▉▉▉ | 207M/243M [00:05<00:00, 38.5MB/s]
 87%|▉▉▉▉▉▉▉▉▉ | 211M/243M [00:05<00:00, 38.8MB/s]
 88%|▉▉▉▉▉▉▉▉▉ | 215M/243M [00:05<00:00, 38.9MB/s]
 90%|▉▉▉▉▉▉▉▉▉ | 219M/243M [00:05<00:00, 39.1MB/s]
 92%|▉▉▉▉▉▉▉▉▉▉| 223M/243M [00:05<00:00, 39.2MB/s]
 94%|▉▉▉▉▉▉▉▉▉▉| 228M/243M [00:05<00:00, 40.1MB/s]
 96%|▉▉▉▉▉▉▉▉▉▉| 233M/243M [00:05<00:00, 42.2MB/s]
 98%|▉▉▉▉▉▉▉▉▉▉| 239M/243M [00:05<00:00, 46.5MB/s]
100%|▉▉▉▉▉▉▉▉▉▉| 243M/243M [00:05<00:00, 41.4MB/s]

Downloaded files total: 2190
Unpacked .gz files total: 1765
Time from start: 0:00:11.331007
Files done: 100/2190         Points: 561928         Time from start: 0:01:57.949657
Files done: 200/2190         Points: 1055973         Time from start: 0:03:48.225765
```

```
Files done: 300/2190         Points: 1525239         Time from start: 0:05:38.334613
Files done: 400/2190         Points: 2125124         Time from start: 0:07:28.989251
Files done: 500/2190         Points: 2677727         Time from start: 0:09:25.163544
Files done: 600/2190         Points: 3190749         Time from start: 0:11:12.439539
Files done: 700/2190         Points: 3682379         Time from start: 0:13:02.326586
Files done: 800/2190         Points: 4241999         Time from start: 0:14:49.970186
Files done: 900/2190         Points: 4767488         Time from start: 0:16:37.493898
Files done: 1000/2190         Points: 5252788         Time from start: 0:18:24.054901
Files done: 1100/2190         Points: 5782689         Time from start: 0:20:14.398042
Files done: 1200/2190         Points: 6333129         Time from start: 0:22:03.400943
Files done: 1300/2190         Points: 6862458         Time from start: 0:23:51.433175
Files done: 1400/2190         Points: 7413209         Time from start: 0:25:41.723606
Files done: 1500/2190         Points: 7888312         Time from start: 0:27:28.314530
Files done: 1600/2190         Points: 8469405         Time from start: 0:29:22.235098
Files done: 1700/2190         Points: 8945913         Time from start: 0:31:10.916823
Files done: 1800/2190         Points: 9444734         Time from start: 0:33:01.630686
Files done: 1900/2190         Points: 9931590         Time from start: 0:34:49.403797
Files done: 2000/2190         Points: 10481654         Time from start: 0:36:37.313329
Files done: 2100/2190         Points: 11042971         Time from start: 0:38:29.835817
Creating table: demo_geo_tracks
Size: (2190, 6)
Creating table: demo_geo_points
Size: (11473061, 10)
Finished. Time from start: 0:40:59.713264
```

Note that we record data in two tables, which are linked through track ID (activity file name). And for convenience of further work with data, in the track table we immediately record the coordinates of the first point of the track (columns `track_first_coord_lat` and `track_first_coord_lon`) and the country calculated from these coordinates.

The execution of this cell takes about 40 minutes. But about half of that time is spent accessing the `geopy` service to determine the country from the coordinates of the track start point. If there's a need to speed up the initial download. it is useful to disable this service.

Checking downloaded data

Check the track table:

```
SELECT count(*) AS "Количество треков"
    FROM demo_geo_tracks
```

```
+-------------------+
| Количество треков |
+-------------------+
| 2190              |
+-------------------+
```

```
SELECT *
    FROM demo_geo_tracks
    LIMIT 10
```

```
Done in 8.2 sec.

+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| track_id        | track_name | track_length_km | track_first_coord_lat |
track_first_coord_lon | country       | speed              | time_stamp          |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 8205566762.fit  | null       | 51.04           | 52.47870256192982     | 13.336647050455213
| Deutschland   | 36.619200000000006 | 2020-06-26 13:44:05 |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 5758691664.fit  | null       | 3.92            | 55.7587356492877      | 37.745437659323215
| Россия        | 36.781200000000005 | 2020-06-26 13:44:06 |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 7562241822.fit  | null       | 21.69           | 36.594818104058504    | 30.56384850293398
| Türkiye       | 37.0512            | 2020-06-26 13:44:07 |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 5752363946.fit  | null       | 62.85           | 55.764476750046015    | 37.71281814202666
| Россия        | 37.3068            | 2020-06-26 13:44:08 |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 5774225138.fit  | null       | 114.24          | 55.99621960893273     | 36.27057650126517
| Россия        | 37.4652            | 2020-06-26 13:44:09 |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 3602018438.fit  | null       | 34.76           | 55.76445009559393     | 37.712875390425324
| Россия        | 37.7856            | 2020-06-26 13:44:10 |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 2665481253.fit  | null       | 43.88           | 57.06134635023773     | 23.311248868703842
| Latvija       | 37.980000000000004 | 2020-06-26 13:44:11 |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 16185006845.fit | null       | 66.47           | 36.71585462987423     |
-4.2837147787213326   | España        | 38.037600000000005 | 2020-06-26 13:44:12 |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 5061483340.fit  | null       | 16.87           | 40.78166037797928     | -73.96295428276062
| United States | 38.037600000000005 | 2020-06-26 13:44:13 |
+-----------------+------------+-----------------+-----------------------
+----------------------+---------------+--------------------+---------------------+
| 1842519266.fit  | null       | 71.94           | 55.73788298293948     | 37.65193585306406
| Россия        | 38.419200000000004 | 2020-06-26 13:44:14 |
+-----------------+------------+-----------------+-----------------------
```

```
+----------------------+--------------+--------------------+--------------------+
```

Checking the point table:

```
SELECT count(*) AS "Количество точек"
    FROM demo_geo_points
```

```
+------------------+
| Количество точек |
+------------------+
| 11473061         |
+------------------+
```

```
SELECT *
    FROM demo_geo_points
    LIMIT 10
```

```
Done in 8.3 sec.

+-----------------+--------------------+--------------------+-------------------+------------
+-------------+---------+-------+--------------------+---------------------------+
| track_id        | latitude           | longitude          | elevation         | heart_rate
| temperature | cadence | power | speed              | time_text                 |
+-----------------+--------------------+--------------------+-------------------+------------
+-------------+---------+-------+--------------------+---------------------------+
| 11078368333.fit | 36.822746274992824 | -4.126659873872995 | 64.39999999999998 | null
| 18          | null    | 290   | 17.9784            | 2023-12-08 15:17:16+00:00 |
+-----------------+--------------------+--------------------+-------------------+------------
+-------------+---------+-------+--------------------+---------------------------+
| 11078368333.fit | 36.82279204018414  | -4.126656521111727 | 64.79999999999995 | null
| 18          | null    | 227   | 18.576             | 2023-12-08 15:17:17+00:00 |
+-----------------+--------------------+--------------------+-------------------+------------
+-------------+---------+-------+--------------------+---------------------------+
| 11078368333.fit | 36.822841661050916 | -4.126651743426919 | 65                | null
| 18          | null    | 148   | 18.4356            | 2023-12-08 15:17:18+00:00 |
+-----------------+--------------------+--------------------+-------------------+------------
+-------------+---------+-------+--------------------+---------------------------+
| 11078368333.fit | 36.82288742624223  | -4.126646546646953 | 65.20000000000005 | null
| 18          | null    | 204   | 18.511200000000002 | 2023-12-08 15:17:19+00:00 |
+-----------------+--------------------+--------------------+-------------------+------------
+-------------+---------+-------+--------------------+---------------------------+
| 11078368333.fit | 36.82293319143355  | -4.126641768962145 | 65.39999999999998 | null
| 18          | null    | 222   | 18.8244            | 2023-12-08 15:17:20+00:00 |
+-----------------+--------------------+--------------------+-------------------+------------
+-------------+---------+-------+--------------------+---------------------------+
| 11078368333.fit | 36.822978956624866 | -4.126635566353798 | 65.60000000000002 | null
| 18          | null    | 213   | 18.9216            | 2023-12-08 15:17:21+00:00 |
+-----------------+--------------------+--------------------+-------------------+------------
```

```
+------------+---------+-------+--------------------+--------------------------+
| 11078368333.fit | 36.82302857749164  | -4.126623664051294 | 65.79999999999995 | null
| 18          | null    | 219   | 19.040399999999998 | 2023-12-08 15:17:22+00:00 |
+-----------------+--------------------+--------------------+-------------------+-----------
+------------+---------+-------+--------------------+--------------------------+
| 11078368333.fit | 36.823078114539385 | -4.126606900244951 | 66                | null
| 18          | null    | 223   | 19.1484            | 2023-12-08 15:17:23+00:00 |
+-----------------+--------------------+--------------------+-------------------+-----------
+------------+---------+-------+--------------------+--------------------------+
| 11078368333.fit | 36.82312773540616  | -4.126583598554134 | 66.20000000000005 | null
| 18          | null    | 223   | 19.332             | 2023-12-08 15:17:24+00:00 |
+-----------------+--------------------+--------------------+-------------------+-----------
+------------+---------+-------+--------------------+--------------------------+
| 11078368333.fit | 36.82317350059748  | -4.126557866111398 | 66.39999999999998 | null
| 18          | null    | 216   | 19.6452            | 2023-12-08 15:17:25+00:00 |
+-----------------+--------------------+--------------------+-------------------+-----------
+------------+---------+-------+--------------------+--------------------------+
```

We see that the row counts in the track and point tables obtained via SQL queries match the data from the log of our first cell — 2,190 tracks and 11,473,061 points.

Now let's download the file `strava_activities.csv` via Download Wizard and check the data in the table generated from it.

```
CREATE OR REPLACE TABLE demo_geo_strava_activities AS
    SELECT *
        FROM read_csv(
            'abogdanov/5fe14b4c4110_strava_activities.csv'
        );

SELECT *
    FROM demo_geo_strava_activities
    LIMIT 10
    ORDER BY distance DESC;
```

```
Done in 9 sec.

+------------+--------------------------+----------------------------+--------------
+-------------------------------------------------------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+----------------------+--------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+-----------------------+-----------------------+-------------
+-----------------+-----------------+--------------------+-----------+--------------
+-----------+-----------------+--------------------+-------------------+------------
+----------------+-------------+---------------+------------+--------------------+------
+------------+-------------------------+-------------+---------------------------
+--------------------------+-----------+--------------------+-------------
+------------------------+--------------------------+-------------------
```

```
+--------------------+--------------------+----------+----------+-----------------
+------------+-----------+--------------+------------------------+-------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+----------------------+---------------
+------------------------+-------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+-----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------+
| activity id | activity date            | activity name               | activity type |
activity description                                                          | elapsed
time | distance | max heart rate | relative effort | commute | activity private note |
activity gear        | filename                     | athlete weight | bike weight | elapsed
time_1 | moving time | distance_1 | max speed | average speed | elevation gain | elevation
loss | elevation low | elevation high | max grade | average grade | average positive grade |
average negative grade | max cadence | average cadence | max heart rate_1 | average heart
rate | max watts | average watts | calories | max temperature | average temperature |
relative effort_1 | total work | number of runs | uphill time | downhill time | other time |
perceived exertion | type | start time | weighted average power | power count | prefer
perceived exertion | perceived relative effort | commute_1 | total weight lifted | from
upload | grade adjusted distance | weather observation time | weather condition | weather
temperature | apparent temperature | dewpoint | humidity | weather pressure | wind speed |
wind gust | wind bearing | precipitation intensity | sunrise time | sunset time | moon phase
| bike     | gear | precipitation probability | precipitation type | cloud cover | weather
visibility | uv index | weather ozone | jump count | total grit | average flow | flagged |
average elapsed speed | dirt distance | newly explored distance | newly explored dirt
distance | activity count | total steps | carbon saved | pool length | training load |
intensity | average grade adjusted pace | timer time | total cycles | recovery | with pet |
competition | long run | for a cause | media
|
+-------------+-------------------------+---------------------------+--------------
+-----------------------------------------------------------------------------------
+--------------+----------+---------------+-----------------+---------
+----------------------+---------------------+-------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+-----------------------+-----------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+---------------------+-------------------+------------
+----------------+-------------+---------------+------------+--------------------+------
+------------+-----------------------+-------------+---------------------------
+--------------------------+-----------+---------------------+-------------
+------------------------+--------------------------+------------------
+---------------------+---------------------+----------+----------+-----------------
+------------+-----------+-------------+-------------------------+--------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+--------------
+------------+------------+--------------+---------+----------------------+--------------
```

```
+------------------------+-----------------------------+---------------+------------
+-------------+-------------+---------------+-----------+----------------------------
+------------+-------------+----------+----------+-------------+----------+------------
+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------+
| 8735873891  | Mar 18, 2023, 7:08:04 AM | La Primavera with RCC       | Ride          | null
| 31281        | 200.77   | 179            | 194             | false   | null
| Giant TCR Advanced 1 | activities/9376119823.fit.gz  | null           | 8           | 31281
| 23034       | 200777.4   | 16.188    | 8.717         | 295            | 372            |
26.8          | 71.6           | 47.4      | 0             | null                    | null
| 172         | 74              | null             | 131                | null      | 146
| 3248     | null            | 14                  | 194               | 3246159    | null
| null        | null          | null       | null               | null | null       | 176
| 26231       | 0                         | null                       | 0         | null
| 1           | null                     | 1679122816               | 3                  | 7.5
| 5.6                  | 4        | 0.78     | 1011.5           | 2.9        | 6.7       |
184          | 0                        | 1679116544   | 1679159808  | 0.89       | 4194513  |
null | 0                          | null               | 1           | 16093              | 0
| 340.5         | null       | null       | null         | 0       | 6.419                |
1138.9        | null                     | null                          | null           |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
media/4cda8a70-bd40-4e47-9f16-f8cce644e23d.jpg|media/5e549f56-1cc5-414e-b1f6-
70923b28c261.jpg|media/aaefa66b-e6e3-425a-bf68-265c1d9f6355.mp4|media/bbf17907-6990-458b-
8c41-ccba2dd463e4.jpg|media/d90e8b35-f65d-4649-8fcc-37214ba3b428.jpg|media/054f30b2-d198-
4ecd-b520-2338c670d509.jpg
|
+------------+-------------------------+----------------------------+--------------
+-----------------------------------------------------------------------------------
+--------------+----------+---------------+----------------+---------
+----------------------+---------------------+--------------------------------
+---------------+------------+---------------+------------+-----------+-----------
+---------------+---------------+---------------+--------------+----------------
+-----------+--------------+----------------------+-----------------------+------------
+-----------------+------------------+--------------------+-----------+---------------
+----------+-----------------+--------------------+--------------------+------------
+----------------+------------+--------------+------------+--------------------+------
+------------+-----------------------+-------------+---------------------------
+--------------------------+-----------+--------------------+-------------
+------------------------+-------------------------+------------------
+---------------------+---------------------+----------+----------+-----------------
+------------+-----------+--------------+-------------------------+--------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+----------------------+--------------
+--------------------------+-----------------------------+---------------+-------------
+--------------+-------------+---------------+-----------+----------------------------
+------------+-------------+----------+----------+-------------+----------+------------
+-----------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
-----------------------------------------------------+
| 2556833948  | Jul 23, 2019, 6:00:52 AM | Giro delle Dolomiti Stage 3 | Ride          | null
| 37818        | 180.47   | 177            | 184             | false   | null
| Giant TCR Advanced 1 | activities/2713670395.fit.gz  | 83             | 8           | 37818
| 28854       | 180471     | 19.4      | null          | 3151           | 3175           |
225           | 2256           | 47.1      | 0             | null                   | null
| 251         | 71              | 177             | 121                | null      | 149
| 5603     | null            | 29                  | 184               | null       | null
| null        | null          | null       | null               | null | null       | null
| null        | null                        | null                      | null      | null
| null        | null                      | null                     | null              | null
| null                 | null     | null     | null             | null       | null      |
null         | null                     | null         | null        | null       | 4194513  |
null | null                       | null               | null        | null               |
null     | null          | null       | null       | null         | null    | null
| null          | null                     | null                          | null           |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
media/c21cda2a-7c14-47f7-a840-24c4cb412025.jpg|media/ae791d8b-e9a1-497c-9309-c49bbc318702.jpg
|
+-------------+--------------------------+-----------------------------+--------------
+----------------------------------------------------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+----------------------+-------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+------------------------+-------------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+---------------------+-------------------+------------
+----------------+-------------+---------------+------------+--------------------+------
+------------+-------------------------+-------------+----------------------------
+--------------------------+-----------+--------------------+-------------
+------------------------+--------------------------+-------------------
+---------------------+---------------------+----------+----------+-----------------
+------------+-----------+--------------+------------------------+--------------
+-------------+------------+----------+------+---------------------------
+--------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+-----------------------+---------------
+-------------------------+-------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
-----------------------------------------------------+
| 4242331242  | Oct 25, 2020, 5:34:56 AM | Alanya Camp Day 6           | Ride          |
Queen stage                                                                    | 33077
```

```
| 180.46   | 157           | 198             | false   | null                  | Giant TCR
Advanced 1 | activities/4538350403.fit.gz  | null           | 8           | 33077          |
25687       | 180461.1   | 22.8      | 7.025         | 3110           | 3119           | 22
| 1060.4         | 33.3      | 0             | null                  | null
| 108         | 73              | null             | 128               | null      | 190
| 4997     | null            | 27                  | 198               | null       | null
| null        | null          | null       | null               | null | null       | null
| null        | 0                        | null                      | 0         | null
| 1           | null                    | 1603602048               | 1                  | 25.2
| 25.2                | 6.6      | 0.3      | 1013.4           | 1.6        | 1.9       | 40
| 0                      | 1603598976   | 1603638272  | 0.32       | 4194513  | null | 0
| null               | 0           | 16093              | 0        | 264.3         | null
| null       | null         | null    | null                  | null          | null
| null                           | null           | null        | null         | null        |
null          | null      | null                         | null       | null         | null
| null     | null        | null     | null        | media/574b5612-10c1-49b3-b3e1-
bc97896153b0.jpg|media/1944d2ba-2e4a-4d15-b7ae-56ede81c8db2.jpg|media/363ad0a2-1fb7-4552-
ae66-a53ec25e6bea.jpg|media/fc1cbf56-0bc7-4a26-88d1-c2558db2f934.jpg|media/d3a84a2d-0d73-
4452-9be3-e53ff1d952f8.jpg|media/6013cbeb-193b-46ba-873d-5120c26a7535.jpg
|
+-------------+--------------------------+----------------------------+--------------
+-------------------------------------------------------------------------------
+--------------+----------+---------------+-----------------+---------
+-----------------------+----------------------+-------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+------------------------+-------------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+----------------------+--------------------+------------
+----------------+-------------+---------------+------------+--------------------+------
+------------+-------------------------+-------------+----------------------------
+---------------------------+-----------+----------------------+-------------
+-------------------------+--------------------------+-------------------
+---------------------+----------------------+----------+----------+-----------------
+------------+-----------+--------------+-------------------------+--------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+----------------------+--------------
+-------------------------+--------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------+
| 5109982540  | Apr 11, 2021, 7:09:45 AM | Desperados camp Day 8       | Ride          |
Queen stage                                                                  | 33941
| 179.42   | 156           | 232             | false   | null                  | Giant TCR
Advanced 1 | activities/5447314715.fit.gz  | null           | 8           | 33941          |
26284       | 179424.8   | 19.7      | 6.826         | 3192           | 3196           | -16
| 1030.8         | 27        | 0             | null                  | null
```

```
| 107         | 73              | null            | 128                | null      | 184
| 5125     | null            | 17                  | 232               | null       | null
| null        | null          | null       | null               | null | null       | null
| null        | 0                         | null                       | 0         | null
| 1           | null                     | 1618124416               | 1                 | 12.6
| 12.6                | -1.3     | 0.38     | 1015.5           | 2.1        | 2.3       |
151          | 0                       | 1618111488   | 1618158208  | 0.99       | 4194513  |
null | 0                         | null               | 0           | 16093              | 3
| 374.3         | null       | null       | null         | null    | null                 |
null          | null                    | null                          | null           |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
media/7cab3cd2-33d2-4c90-a3a7-9fe1ab10717a.jpg|media/6ed80c53-4d61-4ed7-b86c-
08f85e81ef3d.jpg|media/90ab131a-463e-412c-89c8-cbf489112f6a.jpg|media/90caba00-d1d4-4b58-
80f5-672797383a42.jpg|media/4486c339-d16d-4c33-bac4-c790117bb62b.jpg|media/b633df5d-3cb5-
42cd-9ce8-166382ab7d08.jpg|media/24da2f65-2a38-49e6-aaad-4e1637465eee.jpg|media/72c68f98-
acbd-4ba8-9f3c-1841147ee3ec.jpg
|
+-------------+-------------------------+----------------------------+--------------
+-------------------------------------------------------------------------------
+--------------+----------+---------------+-----------------+---------
+----------------------+---------------------+------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+-----------------------+-------------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+---------------------+-------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
+------------+------------------------+-------------+---------------------------
+---------------------------+-----------+---------------------+-------------
+-------------------------+--------------------------+-------------------
+---------------------+---------------------+----------+----------+-----------------
+------------+-----------+--------------+-------------------------+--------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+--------------
+------------+------------+-------------+---------+----------------------+--------------
+-------------------------+-------------------------------+---------------+-------------
+--------------+-------------+--------------+-----------+---------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-----------------------------------------------------+
| 9196092834  | Jun 3, 2023, 8:53:43 AM  | To Chartres and back        | Ride          | Эпик
соло                                                                    | 31048
| 178.89   | null           | null            | false   | null                 | Giant TCR
Advanced 1 | activities/9866271692.fit.gz  | null           | 8           | 31048          |
24765       | 178891.4   | 16.196    | 7.224         | 1273           | 1269           | 60.8
| 179.8          | 35.1      | 0             | null                     | null
| 158         | 77              | null            | null              | null      | 139
| 3394     | null            | 24                  | null             | 3393782    | null
```

```
| null        | null          | null      | null                | null | null      | 164
| 27934       | 0                       | null                      | 0         | null
| 1           | null                    | 1685779200                | 1                   | 16.4
| 15.8                 | 9.8      | 0.65      | 1018.3            | 3.5        | 7.3        | 30
| 0                       | 1685764224   | 1685821696  | 0.5         | 4194513  | null | 0
| 1                  | 0           | 27927.8           | 3        | null          | null
| null       | null          | 0        | 5.762                 | 162.6         | null
| null                            | null            | null         | null          | null         |
null           | null      | null                        | null       | null          | null
| null     | null         | null     | null          | media/6767f454-38a4-4e8f-9744-
0224aa404404.jpg|media/9710109e-2167-4787-af1e-9bc0d48d8b37.jpg|media/f8bfef10-0af0-4f26-
884d-2c39ba52e84b.jpg|media/734bba38-006a-46d8-8a3a-821a6664b02c.jpg|media/efe18288-537a-
45af-9c8f-0753faee311b.jpg|media/44862277-2c3c-4076-a6c7-c0808d078bcb.jpg|media/7e33db5c-
3484-4baf-bcaf-fe45b519cbef.jpg|media/ba09fc14-0114-4089-9d30-
97a717a98f97.jpg|media/994a029a-6360-43a3-b2b1-f86bf4926cf0.jpg|media/197cd6db-63da-4dde-
bd3b-f66ff65553c5.jpg|media/a69ff087-470a-43e1-aaf9-6514757a67f9.jpg |
+-------------+--------------------------+-----------------------------+---------------
+-------------------------------------------------------------------------------------
+--------------+----------+----------------+-----------------+---------
+-----------------------+----------------------+-------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+------------------------+------------------------+-------------
+-----------------+------------------+--------------------+-----------+---------------
+----------+-----------------+---------------------+-------------------+------------
+----------------+-------------+---------------+------------+--------------------+------
+------------+------------------------+-------------+---------------------------
+---------------------------+-----------+---------------------+-------------
+-------------------------+--------------------------+-------------------
+---------------------+----------------------+----------+----------+------------------
+------------+-----------+--------------+-------------------------+--------------
+-------------+------------+----------+------+---------------------------
+--------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+-----------------------+---------------
+-------------------------+------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+-----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+--------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
------------------------------------------------------+
| 13705580758 | Feb 23, 2025, 8:14:47 AM | Morning Drop Ride          | Ride           | Was
dropped on the way back and had to do 80km ITT                                | 21668
| 168.59   | 180             | 424              | false   | null                 | Rose xlite
04        | activities/14626116429.fit.gz | null           | 7.8          | 21668           |
19470       | 168594.1   | 19.88      | 8.659          | 1288            | 1299            | 31.4
| 184.4           | 12.9      | 0              | null                    | null
| 164         | 84               | 180               | 149               | null      | 193
| 3718     | null              | 18                  | 424                | 3711913    | null
| null        | null          | null      | null                | null | null      | 222
| 20512       | 0                       | null                      | 0         | null
```

```
| 1           | null                    | 1740297600               | 1                 | 12.6
| 13.3                 | 7        | 0.69     | 1027             | 2.4        | 3.8       |
339          | 0                       | 1740293760   | 1740333952  | 0.875      | 13283968 |
null | 0                         | 1                  | 0           | 34184              | 0
| null          | null       | null       | null         | 0       | 7.781                |
0             | null                    | null                           | null           |
null        | null         | null        | 293           | 72        | null
| null       | null         | null     | null     | null        | null     | null        |
media/ca9d74d0-33fc-4db2-bb39-2ffdaaf08d67.jpg|media/584a4f49-c9f9-4560-807d-72752f676e75.jpg
|
+-------------+--------------------------+----------------------------+--------------
+-------------------------------------------------------------------------------
+--------------+----------+---------------+----------------+---------
+----------------------+----------------------+------------------------------
+---------------+-------------+---------------+-------------+-----------+-----------
+---------------+---------------+---------------+--------------+----------------
+-----------+--------------+-----------------------+-------------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+--------------------+------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
+------------+------------------------+-------------+---------------------------
+--------------------------+-----------+---------------------+-------------
+--------------------------+--------------------------+-------------------
+----------------------+----------------------+----------+----------+-----------------
+------------+-----------+--------------+------------------------+--------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+-----------------------+---------------
+--------------------------+--------------------------------+----------------+-------------
+--------------+-------------+--------------+-----------+-----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------+
| 5084894611  | Apr 7, 2021, 7:07:29 AM  | Desperados camp Day 4       | Ride          | Pre-
epic                                                                                  | 25473
| 164.2    | 165            | 248             | false   | null                 | Giant TCR
Advanced 1 | activities/5421087598.fit.gz  | null           | 8           | 25473          |
19934       | 164206.4   | 21.2      | 8.238         | 1736           | 1729           |
-36.8         | 572.4          | 21.3      | 0             | null                    | null
| 108         | 79              | null             | 135                | null      | 206
| 4253     | null            | 24                  | 248               | null       | null
| null        | null          | null       | null               | null | null       | null
| null        | 0                         | null                      | 0         | null
| 1           | null                    | 1617778816               | 1                 | 20.1
| 20.1                 | 11.1     | 0.56     | 1011.8           | 1.7        | 2.9       | 86
| 0                       | 1617766144   | 1617812352  | 0.86       | 4194513  | null | 0
| null               | 0.11        | 16093              | 4        | 345.5         | null
| null       | null         | null    | null                  | null          | null
| null                         | null           | null        | null         | null       |
```

```
null          | null      | null                        | null       | null         | null
| null     | null        | null     | null        | media/68f7fee3-f6ed-4285-9bd5-
a64271b3876e.jpg
|
+-------------+--------------------------+----------------------------+--------------
+-------------------------------------------------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+----------------------+------------------------------
+-----------------+-------------+-----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+-----------------------+-------------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+--------------------+-------------------+------------
+----------------+-------------+---------------+------------+--------------------+------
+------------+------------------------+-------------+--------------------------
+--------------------------+-----------+---------------------+-------------
+------------------------+--------------------------+-------------------
+---------------------+---------------------+----------+----------+-----------------
+------------+-----------+--------------+-------------------------+--------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+--------------
+------------+------------+--------------+---------+-----------------------+--------------
+-------------------------+-------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+-----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------------------------------------------+
| 6178861995  | Oct 28, 2021, 6:09:52 AM | Cyprus Camp Day 5           | Ride          | null
| 30438        | 161.65   | 159            | 114             | false   | null
| Giant TCR Advanced 1 | activities/6566187110.fit.gz  | null           | 8           | 30438
| 20881       | 161657.4   | 17.227    | 7.742         | 1484           | 1502           | -4
| 181.6          | 27.7      | 0             | null                    | null
| 142         | 76              | null             | 121                | null      | 145
| 2956     | null            | 20                 | 114               | 2953912    | null
| null        | null          | null       | null               | null | null       | 178
| 22443       | 0                         | null                       | 0         | null
| 1           | null                     | 1635400832               | 2                 | 22.3
| 22.3                | 14.2     | 0.6      | 1016.9           | 1.8        | 1.9       | 73
| 0                      | 1635393920   | 1635433088  | 0.75       | 4194513  | null | 0
| null               | 0.7         | 16093              | 1        | 285.2         | null
| null       | null         | 0       | 5.311                | 102.7         | null
| null                           | null           | null        | null         | null        |
null          | null      | null                          | null       | null         | null
| null     | null        | null     | null        | media/a74a5532-85bc-40fe-9a4c-
8645bca02811.jpg|media/c335908c-72d8-4a09-ac2f-1719bb665973.jpg
|
+-------------+--------------------------+----------------------------+--------------
+-------------------------------------------------------------------------------
+--------------+----------+----------------+-----------------+---------
```

```
+----------------------+---------------------+-----------------------------
+---------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+----------------------+----------------------+-------------
+----------------+-----------------+--------------------+-----------+--------------
+----------+----------------+---------------------+-------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
+------------+-------------------------+-------------+--------------------------
+--------------------------+-----------+---------------------+-------------
+-------------------------+--------------------------+------------------
+---------------------+---------------------+----------+----------+-----------------
+------------+-----------+--------------+-------------------------+-------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+--------------
+------------+------------+--------------+---------+----------------------+--------------
+-------------------------+--------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+----------------------------
+------------+--------------+----------+----------+------------+----------+------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
-----------------------------------------------------+
| 9325531707  | Jun 24, 2023, 5:04:12 AM | Coup d'etat Ride            | Ride          |
Wonderful ride with nice guys! Thank you [strava://athletes/71275936/10]Luis Chase | 31007
| 159.99   | 179            | 332             | false   | null                  | Giant TCR
Advanced 1 | activities/10003255376.fit.gz | null           | null        | null           |
null        | null       | null      | null          | null           | null           | null
| null           | null      | null          | null                    | null
| null        | null            | null             | null               | null      | null
| null     | null            | null               | null              | null       | null
| null        | null          | null       | null               | null | null       | null
| null        | null                        | null                      | null      | null
| null        | null                     | null                     | null              | null
| null                 | null     | null     | null             | null       | null      |
null         | null                     | null         | null        | null       | null     |
null | null                      | null               | null        | null               |
null     | null          | null       | null       | null         | null    | null
| null          | null                     | null                           | null           |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
media/0df135af-edfe-4148-af00-1d09287558b1.jpg|media/50663e37-4f88-407d-9fe2-
b5c1187a2f69.jpg|media/c196fee3-6fa1-419d-aa9b-95050bead0b9.mp4|media/1e674155-6607-49fd-
b28c-5bea672a8cd9.jpg|media/addc9782-4c14-43d5-a978-0c728442fc7c.jpg
|
+-------------+--------------------------+---------------------------+--------------
+-------------------------------------------------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+----------------------+-----------------------------
+---------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+----------------------+----------------------+-------------
```

```
+----------------+-----------------+--------------------+----------+--------------
+----------+----------------+--------------------+-------------------+------------
+---------------+------------+--------------+-----------+--------------------+------
+-----------+----------------------+------------+--------------------------
+-------------------------+----------+--------------------+------------
+------------------------+-------------------------+------------------
+--------------------+---------------------+----------+----------+-----------------
+-----------+----------+-------------+-----------------------+-------------
+-------------+------------+----------+------+--------------------------
+--------------------+------------+--------------------+----------+--------------
+-----------+-----------+-------------+---------+----------------------+--------------
+------------------------+-------------------------------+---------------+------------
+-------------+------------+--------------+-----------+--------------------------
+-----------+-------------+----------+----------+------------+----------+------------
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
-----------------------------------------------------+
| 11993808841 | Jul 27, 2024, 7:16:28 AM | Morning Ride                | Ride          | null
| 22167        | 157.8    | 177            | 262             | false   | null
| Rose xlite 04        | activities/12786638321.fit.gz | null           | 7.8         | 22167
| 18925       | 157800.3   | 18.886    | 8.338         | 1451           | 1489           |
-6.8          | 624.2          | 23.4      | 0             | null                   | null
| 93          | 68              | 177              | 137                | null      | 0
| 4130     | null            | 28                  | 262               | 3204       | null
| null        | null          | null       | null               | null | null       | 5
| 19179       | 0                         | null                       | 0         | null
| 1           | null                     | 1722063616               | 2                  | 23
| 25.6                 | 20.7     | 0.87     | 1012.7           | 0.7        | 1.6       | 24
| 0                      | 1722057600   | 1722108416  | 0.625      | 13283968 | null | 0
| 1                  | 0.41        | 20127              | 1        | null          | null
| null       | null         | 0       | 7.119                 | 0             | null
| null                          | null           | null        | null         | null        |
0             | 2         | null                          | null       | null         | null
| null     | null        | null     | null        | media/db7a0b8e-b71f-4a38-902e-
3435ae00f790.jpg
|
+-------------+--------------------------+----------------------------+--------------
+-----------------------------------------------------------------------------------
+--------------+----------+---------------+----------------+---------
+----------------------+---------------------+--------------------------------
+---------------+-------------+---------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+-----------------------+-----------------------+-------------
+----------------+-----------------+--------------------+-----------+--------------
+----------+----------------+--------------------+-------------------+------------
+---------------+------------+--------------+-----------+--------------------+------
+------------+----------------------+------------+--------------------------
+--------------------------+----------+--------------------+------------
+------------------------+-------------------------+------------------
+--------------------+---------------------+----------+----------+-----------------
```

```
+-----------+----------+-------------+-----------------------+-------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+--------------
+------------+------------+-------------+---------+----------------------+---------------
+-------------------------+-------------------------------+---------------+-------------
+-------------+-------------+--------------+-----------+----------------------------
+-----------+-------------+----------+----------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------+
```

In this table we can see all activity parameters that are downloaded from the Strava service itself (not from track files). There are various parameters, including such exotic ones as `Moon Phase` and `With Pet` .

Note that all velocities are in m/s, time is in seconds, and distance is in kilometres.

Data cleaning and preparation

Let's create a silver layer table `demo_geo_tracks_silver`. To do this, convert the time from the `activity date` column to the `TIMESTAMP` type using the `strptime` function, which allows you to specify a specific pattern of text description of a moment in time. And merge the track table obtained from the activity files with the table obtained from the `strava_activities.csv` file, checking that `track_id` is contained in `filename` using the `contains` function:

```
CREATE OR REPLACE TABLE demo_geo_tracks_silver AS
    SELECT *,
        strptime("activity date", '%b %d, %Y, %H:%M:%S %p') as time_stamp
        FROM demo_geo_tracks
        JOIN demo_geo_strava_activities ON (contains(filename, track_id));

SELECT * FROM demo_geo_tracks_silver
    LIMIT 10;
```

```
Done in 9.4 sec.

+-----------------+------------+-----------------+-----------------------
+-----------------------+---------------+-------------+--------------------------
+-------------------------------+---------------+-----------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+----------------------+--------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+-----------------------+-----------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+----------------+--------------------+-------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
```

```
+-----------+-----------------------+------------+--------------------------
+--------------------------+----------+--------------------+------------
+------------------------+-------------------------+-------------------
+--------------------+---------------------+----------+----------+-----------------
+-----------+----------+-------------+------------------------+-------------
+------------+-----------+----------+------+--------------------------
+-------------------+------------+--------------------+----------+--------------
+-----------+-----------+-------------+---------+----------------------+--------------
+------------------------+-----------------------------+---------------+------------
+-------------+------------+--------------+----------+---------------------------
+-----------+-------------+----------+----------+------------+----------+------------
+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----+--------------------+
| track_id        | track_name | track_length_km | track_first_coord_lat |
track_first_coord_lon | country       | activity id | activity date            | activity
name                  | activity type | activity description                   | elapsed
time | distance | max heart rate | relative effort | commute | activity private note |
activity gear        | filename                     | athlete weight | bike weight | elapsed
time_1 | moving time | distance_1 | max speed | average speed | elevation gain | elevation
loss | elevation low | elevation high | max grade | average grade | average positive grade |
average negative grade | max cadence | average cadence | max heart rate_1 | average heart
rate | max watts | average watts | calories | max temperature | average temperature |
relative effort_1 | total work | number of runs | uphill time | downhill time | other time |
perceived exertion | type | start time | weighted average power | power count | prefer
perceived exertion | perceived relative effort | commute_1 | total weight lifted | from
upload | grade adjusted distance | weather observation time | weather condition | weather
temperature | apparent temperature | dewpoint | humidity | weather pressure | wind speed |
wind gust | wind bearing | precipitation intensity | sunrise time | sunset time | moon phase
| bike     | gear | precipitation probability | precipitation type | cloud cover | weather
visibility | uv index | weather ozone | jump count | total grit | average flow | flagged |
average elapsed speed | dirt distance | newly explored distance | newly explored dirt
distance | activity count | total steps | carbon saved | pool length | training load |
intensity | average grade adjusted pace | timer time | total cycles | recovery | with pet |
competition | long run | for a cause | media
| time_stamp          |
+-----------------+------------+----------------+----------------------
+----------------------+--------------+-------------+--------------------------
+---------------------------------+---------------+-----------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+---------------------+--------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+-----------------------+-------------------------+-------------
+-----------------+-----------------+--------------------+-----------+---------------
+----------+-----------------+---------------------+-------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
+------------+-----------------------+-------------+--------------------------
+--------------------------+-----------+--------------------+-------------
+------------------------+-------------------------+------------------
+--------------------+--------------------+----------+----------+----------------
+------------+-----------+-------------+-------------------------+--------------
```

```
+------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+--------------
+------------+------------+-------------+---------+----------------------+--------------
+------------------------+------------------------------+---------------+-------------
+-------------+-------------+--------------+-----------+---------------------------
+------------+-------------+----------+----------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
-----+--------------------+
| 8205566762.fit  | null       | 51.04           | 52.47870256192982     | 13.336647050455213
| Deutschland   | 7688162488  | Aug 23, 2022, 4:39:47 PM | Berlin Recon                   |
Ride          | null                                    | 7740         | 51.04    | 174
| 73              | false   | null                  | Giant TCR Advanced 1 |
activities/8205566762.fit.gz  | null           | 8           | 7740           | 6518        |
51044.1    | 16.012    | 7.831         | 174            | 188            | 18.6          |
68.4           | 9.3       | 0             | null                    | null
| 116         | 73              | null             | 134                | null      | 168
| 1083     | null            | 22                  | 73                | 1082634    | null
| null        | null          | null       | null               | null | null       | 202
| 7738        | 0                         | null                      | 0         | null
| 1           | null                     | 1661270400               | 2                  | 24.5
| 24.6                | 15.6     | 0.58     | 1018.7           | 3.5        | 3.8       | 60
| 0                     | 1661227392   | 1661278592  | 0.89       | 4194513  | null | 0
| null               | 0.34        | 16093              | 1        | 301.9         | null
| null       | null         | 0       | 6.595                | 332.6         | null
| null                          | null           | null        | null         | null        |
null          | null      | null                         | null       | null         | null
| null     | null        | null     | null        | media/f6509beb-c34d-4eb4-97c1-
9c80b92308ec.jpg|media/f2fe2880-42a1-4794-bef0-b5d563437aca.jpg
| 2022-08-23 16:39:47 |
+-----------------+------------+-----------------+----------------------
+----------------------+--------------+-------------+--------------------------
+------------------------------+--------------+---------------------------------------
+--------------+----------+---------------+-----------------+---------
+----------------------+---------------------+------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+---------------+---------------+--------------+----------------
+-----------+---------------+-----------------------+-------------------------+-------------
+-----------------+------------------+--------------------+----------+--------------
+----------+-----------------+--------------------+-------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
+------------+-------------------------+-------------+---------------------------
+--------------------------+-----------+--------------------+-------------
+------------------------+-------------------------+-------------------
+---------------------+----------------------+----------+----------+-----------------
+------------+-----------+-------------+------------------------+-------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+--------------
+------------+------------+-------------+---------+----------------------+--------------
+------------------------+------------------------------+---------------+-------------
+-------------+-------------+--------------+-----------+---------------------------
+------------+-------------+----------+----------+-------------+----------+-------------
```

```
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-----+--------------------+
| 5758691664.fit  | null       | 3.92            | 55.7587356492877      | 37.745437659323215
| Россия        | 5408223435  | Jun 3, 2021, 6:35:24 PM  | Ночной заезд                    |
Ride          | null                                    | 554          | 3.91     | 143
| 2               | false   | null                  | Giant TCR Advanced 1 |
activities/5758691664.fit.gz  | null           | 8           | 554            | 554         |
3918.7     | 11.3      | 7.073         | 15             | 20             | 110.2         |
130.8          | 4.7       | -0.2          | null                    | null
| 92          | 67              | null             | 122                | null      | 127
| 68       | null            | 18                  | 2                 | 68242      | null
| null        | null          | null       | null               | null | null       | 141
| 555         | null                        | null                       | 0         | null
| 1           | null                     | 1622743168               | 1                  | 17.6
| 17.6                 | 6.2      | 0.47     | 1024.7           | 3.5        | 8.7       | 42
| 0                       | 1622681472   | 1622743552  | 0.8        | 4194513  | null | 0
| null               | 0.28        | 16093              | 0        | 339           | null
| null       | null         | null    | null                  | null          | null
| null                          | null           | null        | null         | null        |
null          | null      | null                          | null       | null         | null
| null     | null        | null     | null        | null
| 2021-06-03 18:35:24 |
+-----------------+------------+-----------------+----------------------
+----------------------+---------------+-------------+--------------------------
+--------------------------------+---------------+-----------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+----------------------+-------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+---------------+----------------
+-----------+---------------+----------------------+-------------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+---------------------+-------------------+------------
+----------------+-------------+---------------+------------+--------------------+------
+------------+-----------------------+-------------+---------------------------
+--------------------------+-----------+---------------------+-------------
+------------------------+-------------------------+------------------
+---------------------+---------------------+----------+----------+------------------
+------------+-----------+--------------+-------------------------+--------------
+-------------+------------+----------+------+---------------------------
+--------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+----------------------+---------------
+------------------------+-------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-----+--------------------+
| 7562241822.fit  | null       | 21.69           | 36.594818104058504    | 30.56384850293398
| Türkiye       | 7103414491  | May 7, 2022, 3:04:49 PM  | Вечерний велозаезд            |
Ride          | Завтра будем страдать в длинную гору    | 3044         | 21.69    | 148
```

```
| 10              | false   | null                  | Giant TCR Advanced 1 |
activities/7562241822.fit.gz  | null           | 8           | 3044           | 2786        |
21692.4    | 16.224    | 7.786         | 105            | 119            | 2             | 45
| 8.1       | 0             | null                  | null                    | 129         |
66              | null            | 108               | null      | 126           | 345
| null            | 21                  | 10                | 343778     | null          |
null        | null          | null       | null               | null | null       | 175
| 3045        | 0                         | null                      | 0         | null
| 1           | null                    | 1651935616              | 2                 | 22.5
| 22.5                | 9.7      | 0.44     | 1013            | 3.5        | 4.6       |
196          | 0                       | 1651892352   | 1651942272  | 0.21       | 4194513  |
null | 0                         | null               | 0.45        | 16093              | 1
| 352.8         | null       | null       | null         | 0       | 7.126                |
645           | null                    | null                          | null          |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
media/318cbe19-4ae0-4800-a873-349417c4df90.jpg
| 2022-05-07 15:04:49 |
+-----------------+------------+-----------------+----------------------
+----------------------+--------------+-------------+-------------------------
+-------------------------------+--------------+---------------------------------------
+-------------+----------+---------------+-----------------+---------
+----------------------+---------------------+------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+---------------+----------------+--------------+----------------
+-----------+--------------+----------------------+-----------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+---------------------+--------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
+------------+-----------------------+-------------+--------------------------
+--------------------------+-----------+---------------------+-------------
+------------------------+-------------------------+------------------
+--------------------+---------------------+----------+----------+-----------------
+------------+-----------+-------------+-----------------------+--------------
+-------------+------------+----------+------+-------------------------
+--------------------+-------------+-------------------+----------+--------------
+------------+------------+--------------+---------+----------------------+---------------
+-----------------------+-------------------------------+---------------+------------
+--------------+-------------+--------------+-----------+-----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
-----+--------------------+
| 5752363946.fit  | null       | 62.85           | 55.764476750046015    | 37.71281814202666
| Россия        | 5402167712  | Jun 2, 2021, 3:55:33 PM  | Вечерний велозаезд             |
Ride          | null                                    | 8485         | 62.85    | 169
| 125             | false   | null                  | Giant TCR Advanced 1 |
activities/5752363946.fit.gz  | null           | 8           | 8485           | 7359        |
62849.2    | 14.5      | 8.54          | 220            | 255            | 154.2         |
226.2          | 10.7      | 0             | null                    | null
| 168         | 81              | null             | 143               | null      | 208
| 1527     | null            | 16                  | 125               | 1530481    | null
```

```
| null        | null          | null       | null               | null | null       | 224
| 7711        | null                       | null                      | 0         | null
| 1           | null                    | 1622646016              | 2                 | 19.7
| 19.7                | 5.6      | 0.4      | 1023.4           | 4.3        | 6.3       | 38
| 0                     | 1622595200   | 1622657152  | 0.76       | 4194513  | null | 0
| null               | 0.68        | 16093              | 1        | 360           | null
| null       | null         | null    | null                  | null          | null
| null                         | null           | null        | null         | null        |
null          | null      | null                         | null       | null         | null
| null     | null        | null     | null        | null
| 2021-06-02 15:55:33 |
+-----------------+------------+-----------------+----------------------
+----------------------+--------------+-------------+--------------------------
+--------------------------------+---------------+-----------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+----------------------+-------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+--------------+----------------
+-----------+---------------+-----------------------+-----------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+--------------------+-------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
+------------+-------------------------+-------------+---------------------------
+--------------------------+-----------+----------------------+-------------
+-------------------------+-------------------------+-------------------
+---------------------+----------------------+----------+----------+-----------------
+------------+-----------+--------------+------------------------+--------------
+-------------+------------+----------+------+--------------------------
+---------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+-----------------------+---------------
+--------------------------+--------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
-----+---------------------+
| 5774225138.fit  | null       | 114.24          | 55.99621960893273     | 36.27057650126517
| Россия        | 5423038640  | Jun 6, 2021, 8:24:11 AM  | Granfondo Волоколамск Группа Б |
Ride          | 6 место в личном зачете и 1 в командном | 10566        | 114.24   | 189
| 284             | false   | null                  | Giant TCR Advanced 1 |
activities/5774225138.fit.gz  | null           | 8           | 10566          | 10496       |
114243.9   | 22.1      | 10.885        | 917            | 894            | 152.8         |
262.8          | 9.6       | 0             | null                    | null
| 173         | 80              | null            | 154               | null      | 199
| 2027     | null            | 26                 | 284               | 2026580    | null
| null        | null          | null       | null               | null | null       | 242
| 10567       | 0                         | null                      | 0         | null
| 1           | null                     | 1622966400              | 2                 | 22.1
| 22.1                | 10.2     | 0.47     | 1014.9           | 3.5        | 4         | 71
| 0.1                   | 1622940800   | 1623003392  | 0.89       | 4194513  | null | 0.01
| 5                 | 0.45        | 16093              | 5        | 341.8         | null
| null       | null         | null    | null                  | null          | null
```

```
| null                          | null           | null        | null         | null        |
null          | null      | null                          | null       | null         | null
| null     | null        | null     | null        | media/946f24a2-cd5f-4b18-ae4b-
822158a40244.jpg|media/7d11f14c-7845-489a-bb34-f71b882c47e6.jpg|media/08fe3d9d-2f68-4319-
b5ce-00c6a6282a49.jpg|media/679c1351-1ac7-4a72-a525-c3bc1766c202.jpg|media/ba7f9fd0-a470-
45d9-8083-00ab445251cd.jpg|media/df115e87-def4-46ee-9f7e-899fd370903e.jpg | 2021-06-06
08:24:11 |
+----------------+-----------+----------------+----------------------
+----------------------+---------------+-------------+--------------------------
+--------------------------------+--------------+-----------------------------------------
+--------------+----------+---------------+-----------------+---------
+----------------------+----------------------+------------------------------
+----------------+-------------+----------------+-------------+-----------+----------
+---------------+----------------+----------------+--------------+----------------
+-----------+---------------+----------------------+----------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+--------------------+------------------+------------
+----------------+-------------+--------------+-----------+--------------------+------
+------------+------------------------+-------------+--------------------------
+--------------------------+-----------+--------------------+-------------
+--------------------------+--------------------------+------------------
+---------------------+---------------------+----------+----------+-----------------
+------------+-----------+-------------+------------------------+-------------
+-------------+------------+----------+------+--------------------------
+---------------------+-------------+--------------------+----------+--------------
+------------+------------+-------------+---------+----------------------+--------------
+--------------------------+--------------------------------+---------------+-------------
+--------------+-------------+--------------+-----------+----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+--------------------+
| 3602018438.fit  | null       | 34.76           | 55.76445009559393     | 37.712875390425324
| Россия        | 3370915442  | Apr 29, 2020, 2:03:19 PM | Дневной велозаезд              |
Ride          | null                                    | 4075         | 34.76    | 174
| 127             | false   | null                  | Giant TCR Advanced 1 |
activities/3602018438.fit.gz  | 83             | 8           | 4075           | 3946        |
34762.6    | 15        | 8.801         | 140            | 146            | 132           |
162.6          | 17.6      | 0             | null                    | null
| 104         | 86              | 174             | 159                | null      | 191
| 1088     | null            | 11                 | 127               | null       | null
| null        | null          | null       | null               | null | null       | null
| null        | null                        | null                       | 0         | null
| 1           | null                     | 1588168832               | 2                 | 11.5
| 11.5                | 0        | 0.45     | 1000.3           | 7.4        | 12.1      |
301          | 0                       | 1588125056   | 1588179840  | 0.21       | 4194513  |
null | 0                          | null              | 0.53        | 16093              | 1
| 367.2         | null       | null       | null         | null    | null                  |
null          | null                    | null                           | null          |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
null
```

```
| 2020-04-29 14:03:19 |
+-----------------+------------+-----------------+----------------------
+----------------------+--------------+-------------+--------------------------
+------------------------------+--------------+----------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+---------------------+-------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+--------------+----------------
+-----------+---------------+----------------------+----------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+--------------------+-------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
+------------+-----------------------+-------------+---------------------------
+--------------------------+-----------+---------------------+-------------
+-------------------------+-----------------------+-------------------
+---------------------+----------------------+----------+----------+-----------------
+------------+-----------+--------------+------------------------+--------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+----------------------+---------------
+-------------------------+-------------------------------+----------------+-------------
+--------------+-------------+--------------+-----------+----------------------------
+------------+-------------+----------+----------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
-----+---------------------+
| 2665481253.fit  | null       | 43.88           | 57.06134635023773     | 23.311248868703842
| Latvija       | 2509891301  | Jul 6, 2019, 6:12:44 PM  | Night ride                    |
Ride          | null                                    | 4958         | 43.88    | 163
| 85              | false   | null                  | Giant TCR Advanced 1 |
activities/2665481253.fit.gz  | 83             | 8           | 4958           | 4958        |
43880      | 12.2      | null          | 51             | 43             | -1            | 6
| 7.7       | 0             | null                    | null                    | 101         |
89              | 163              | 143                | null      | 170           | 1395
| null            | 12                  | 85                | null       | null           |
null        | null          | null       | null               | null | null       | null
| null        | null                        | null                       | null      | null
| null        | null                      | null                       | null              | null
| null                 | null     | null     | null             | null       | null      |
null         | null                      | null         | null        | null       | 4194513  |
null | null                        | null               | null        | null               |
null     | null          | null       | null       | null         | null    | null
| null          | null                      | null                           | null           |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
null
| 2019-07-06 18:12:44 |
+-----------------+------------+-----------------+----------------------
+----------------------+--------------+-------------+--------------------------
+------------------------------+--------------+----------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+---------------------+-------------------------------
```

```
+---------------+------------+---------------+------------+-----------+-----------
+--------------+---------------+----------------+--------------+---------------
+----------+--------------+----------------------+----------------------+------------
+----------------+-----------------+--------------------+----------+--------------
+----------+----------------+--------------------+------------------+------------
+---------------+------------+--------------+-----------+--------------------+------
+-----------+-----------------------+-------------+--------------------------
+-------------------------+----------+--------------------+------------
+-------------------------+-------------------------+------------------
+--------------------+--------------------+----------+----------+-----------------
+-----------+-----------+--------------+-------------------------+-------------
+------------+------------+----------+------+--------------------------
+--------------------+------------+--------------------+----------+--------------
+-----------+-----------+--------------+---------+----------------------+--------------
+-------------------------+-------------------------------+---------------+------------
+--------------+------------+--------------+-----------+---------------------------
+-----------+------------+--------------+----------+----------+----------+------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----+--------------------+
| 5061483340.fit  | null       | 16.87           | 40.78166037797928     | -73.96295428276062
| United States | 4741945778  | Feb 6, 2021, 7:18:45 AM  | NYC                            |
Virtual Ride  | null                                    | 1976         | 16.87    | 157
| 16              | false   | null                  | Giant TCR Advanced 1 |
activities/5061483340.fit.gz  | null           | 8           | 1976           | 1953        |
16870.5    | 15.2      | 8.638         | 176            | 0              | 17            |
46.2           | 12.4      | -0.1          | null                    | null
| 103         | 89              | null             | 133                | null      | 169
| 313      | null            | null               | 16                | null       | null
| null        | null          | null       | null               | null | null       | 175
| 1977        | null                        | null                       | 0         | null
| 1           | null                     | null                     | null              | null
| null                 | null     | null     | null             | null       | null      |
null         | null                     | null         | null        | null       | 4194513  |
null | null                         | null               | null        | null               |
null     | null          | null       | null       | null         | null    | null
| null          | null                     | null                          | null          |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
media/9218c649-fc79-4378-a133-5d691bd50c38.jpg
| 2021-02-06 07:18:45 |
+-----------------+------------+----------------+----------------------
+----------------------+--------------+-------------+-------------------------
+--------------------------------+--------------+----------------------------------------
+--------------+----------+----------------+-----------------+---------
+----------------------+---------------------+-------------------------------
+----------------+-------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+--------------+---------------
+-----------+---------------+-----------------------+-----------------------+-------------
+-----------------+------------------+--------------------+-----------+---------------
+----------+-----------------+--------------------+-------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
```

```
+-----------+-----------------------+-------------+--------------------------
+--------------------------+-----------+---------------------+-------------
+------------------------+--------------------------+-------------------
+---------------------+---------------------+----------+----------+-----------------
+------------+-----------+-------------+-------------------------+-------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+--------------
+------------+------------+-------------+---------+----------------------+--------------
+------------------------+-------------------------------+----------------+-------------
+-------------+-------------+--------------+-----------+----------------------------
+------------+-------------+----------+---------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
-----+--------------------+
| 1842519266.fit  | null       | 71.94           | 55.73788298293948     | 37.65193585306406
| Россия        | 1717234129  | Jul 21, 2018, 9:51:37 AM | Noon ride                      |
Ride          | null                                    | 9707         | 71.94    | 170
| 186             | false   | null                  | Giant TCR Advanced 1 |
activities/1842519266.fit.gz  | 83             | 8           | 9707           | 8353        |
71942      | 13.6      | null          | 313            | 299            | 110           |
193            | 33.3      | 0             | null                    | null
| 101         | 86              | 170              | 148                | null      | 178
| 2298     | null            | 27                  | 186               | null       | null
| null        | null          | null       | null               | null | null       | null
| null        | null                        | null                       | 0         | null
| null        | null                      | null                      | null              | null
| null                 | null     | null     | null             | null       | null      |
null         | null                     | null         | null        | null       | 4194513  |
null | null                        | null               | null        | null               |
null     | null          | null       | null       | null         | null    | null
| null          | null                      | null                          | null           |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
null
| 2018-07-21 09:51:37 |
+-----------------+------------+-----------------+----------------------
+----------------------+--------------+------------+--------------------------
+--------------------------------+---------------+----------------------------------------
+--------------+----------+---------------+-----------------+---------
+----------------------+----------------------+-------------------------------
+----------------+------------+----------------+-------------+------------+-----------
+---------------+----------------+----------------+--------------+----------------
+-----------+---------------+-----------------------+-------------------------+-------------
+-----------------+------------------+--------------------+-----------+--------------
+----------+-----------------+--------------------+-------------------+------------
+----------------+------------+--------------+------------+--------------------+------
+------------+-------------------------+-------------+---------------------------
+--------------------------+-----------+---------------------+-------------
+------------------------+--------------------------+-------------------
+---------------------+---------------------+----------+----------+-----------------
+------------+-----------+-------------+-------------------------+-------------
+-------------+------------+----------+------+--------------------------
```

```
+--------------------+------------+--------------------+----------+---------------
+------------+------------+--------------+---------+-----------------------+---------------
+------------------------+-------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+-----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----+--------------------+
| 11417255201.fit | null       | 30.91           | 36.71511836349964     |
-4.2892365250736475   | España        | 10674837301 | Feb 1, 2024, 2:24:07 PM  | Afternoon
Ride                  | Ride          | From new apartment                        | 4793
| 30.91    | 166            | 67              | false   | null                   | Rose xlite
04        | activities/11417255201.fit.gz | null           | 7.8         | 4793           |
4571        | 30913.6    | 15.33     | 6.763         | 658            | 676            | 8.8
| 509.8          | 20.3      | 0             | null                      | null
| 118         | 74              | null             | 137                | null      | 188
| 850      | null            | 15                  | 67                | 849803     | null
| null        | null          | null       | null               | null | null       | 227
| 4794        | 0                         | null                        | 0         | null
| 1           | null                     | 1706796032               | 1                  | 16.9
| 16.4                 | 10.4     | 0.65     | 1033             | 2.7        | 5.9       |
128          | 0                        | 1706771968   | 1706809344  | 0.625      | 13283968 |
null | 0                         | 1                 | 0.29        | 26463              | 2
| null          | null       | null       | null         | 0       | 6.45                  |
109.7         | null                     | null                           | null           |
null        | null         | null        | null          | null      | null
| null       | null         | null     | null     | null        | null     | null        |
media/da2f0e86-c77e-42c2-b7c5-5d08d0ac7acf.jpg
| 2024-02-01 14:24:07 |
+-----------------+------------+----------------+----------------------
+----------------------+--------------+-------------+-------------------------
+------------------------------+--------------+---------------------------------------
+--------------+----------+---------------+-----------------+---------
+----------------------+----------------------+-----------------------------
+----------------+------------+---------------+-------------+-----------+----------
+---------------+---------------+---------------+--------------+----------------
+----------+--------------+----------------------+----------------------+------------
+-----------------+------------------+--------------------+-----------+---------------
+----------+-----------------+--------------------+-------------------+------------
+----------------+-------------+--------------+------------+--------------------+------
+------------+----------------------+-------------+--------------------------
+--------------------------+-----------+--------------------+-------------
+------------------------+--------------------------+------------------
+---------------------+----------------------+----------+----------+-----------------
+------------+-----------+--------------+-------------------------+-------------
+-------------+------------+----------+------+--------------------------
+--------------------+-------------+--------------------+----------+---------------
+------------+------------+--------------+---------+----------------------+---------------
+-------------------------+-----------------------------------+----------------+-------------
+--------------+-------------+---------------+-----------+-----------------------------
+------------+--------------+----------+----------+-------------+----------+-------------
+-----------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----+--------------------+
```

Now let's create a table of silver layer points — leave only the necessary columns in it, convert the data in the `time_text` column to the `TIMESTAMP` type (using the `strptime` and `regexp_extract` functions), and filter out all points where the `latitude` or `longitude` value is empty:

```
CREATE OR REPLACE TABLE demo_geo_points_silver AS
    SELECT
        track_id,
        latitude,
        longitude,
        elevation,
        temperature,
        power,
        speed,
        strptime(regexp_extract(time_text, '^.{19}'), '%c') as time_stamp,
    FROM demo_geo_points
    WHERE latitude IS NOT NULL AND longitude IS NOT NULL;

SELECT *
    FROM demo_geo_points_silver
    LIMIT 10;
```

```
Done in 11.6 sec.

+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+---------------------+
| track_id       | latitude           | longitude          | elevation          | temperature
| power | speed              | time_stamp          |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+---------------------+
| 3922792040.fit | 55.84256364963949  | 37.73899080231786  | 167.60000000000002 | 24
| null  | 36.619200000000006 | 2020-06-26 13:44:05 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+---------------------+
| 3922792040.fit | 55.842567421495914 | 37.73883054032922  | 167.60000000000002 | 24
| null  | 36.781200000000005 | 2020-06-26 13:44:06 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+---------------------+
| 3922792040.fit | 55.842567421495914 | 37.73866650648415  | 167.60000000000002 | 24
| null  | 37.0512            | 2020-06-26 13:44:07 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+---------------------+
| 3922792040.fit | 55.842567421495914 | 37.73849870078266  | 167.60000000000002 | 24
| null  | 37.3068            | 2020-06-26 13:44:08 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+---------------------+
| 3922792040.fit | 55.84256364963949  | 37.73833081126213  | 167.60000000000002 | 24
```

```
| null  | 37.4652            | 2020-06-26 13:44:09 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+--------------------+
| 3922792040.fit | 55.8425560221076   | 37.738166777417064 | 167.60000000000002 | 24
| null  | 37.7856            | 2020-06-26 13:44:10 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+--------------------+
| 3922792040.fit | 55.842548394575715 | 37.73799511604011  | 167.39999999999998 | 24
| null  | 37.980000000000004 | 2020-06-26 13:44:11 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+--------------------+
| 3922792040.fit | 55.842544538900256 | 37.7378349378705   | 167.20000000000005 | 24
| null  | 38.037600000000005 | 2020-06-26 13:44:12 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+--------------------+
| 3922792040.fit | 55.84254076704383  | 37.737670904025435 | 167                | 24
| null  | 38.037600000000005 | 2020-06-26 13:44:13 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+--------------------+
| 3922792040.fit | 55.84253691136837  | 37.73749924264848  | 166.79999999999995 | 24
| null  | 38.419200000000004 | 2020-06-26 13:44:14 |
+---------------+--------------------+--------------------+--------------------
+-------------+-------+--------------------+--------------------+
```

Analyses

A wide variety of statistics can be derived from the data generated above. We will limit ourselves here to just a few examples.

Statistics by country

Let's display the total mileage in kilometres for all tracks with the start in a given country, excluding virtual races. Order the result in descending order of total mileage. Use functions `round` and `sum`.

```
SELECT
    country AS "Country",
    round(sum(distance)) AS "Total distance, km",
FROM demo_geo_tracks_silver
WHERE NOT "Activity Type" = 'Virtual Ride'
GROUP BY 1
ORDER BY 2 DESC
```

```
+------------------------------+--------------------+
| Country                      | Total distance, km |
+------------------------------+--------------------+
| Россия                       | 36947              |
+------------------------------+--------------------+
| España                       | 30437              |
+------------------------------+--------------------+
```

```
| Türkiye                         | 8255               |
+---------------------------------+--------------------+
| Deutschland                     | 6033               |
+---------------------------------+--------------------+
| Italia                          | 2681               |
+---------------------------------+--------------------+
| France                          | 1833               |
+---------------------------------+--------------------+
| K□προς - Kıbrıs                 | 1452               |
+---------------------------------+--------------------+
| Latvija                         | 1191               |
+---------------------------------+--------------------+
| Schweiz/Suisse/Svizzera/Svizra | 708                |
+---------------------------------+--------------------+
| Sverige                         | 615                |
+---------------------------------+--------------------+
| □□□□□□□□                       | 595                |
+---------------------------------+--------------------+
| Magyarország                    | 421                |
+---------------------------------+--------------------+
| Nederland                       | 408                |
+---------------------------------+--------------------+
| Österreich                      | 352                |
+---------------------------------+--------------------+
| Slovensko                       | 223                |
+---------------------------------+--------------------+
| Unknown                         | 56                 |
+---------------------------------+--------------------+
| United Kingdom                  | 48                 |
+---------------------------------+--------------------+
```

We obtained a table ranking countries according to the total mileage in each country.

Analysing several indicators within one activity

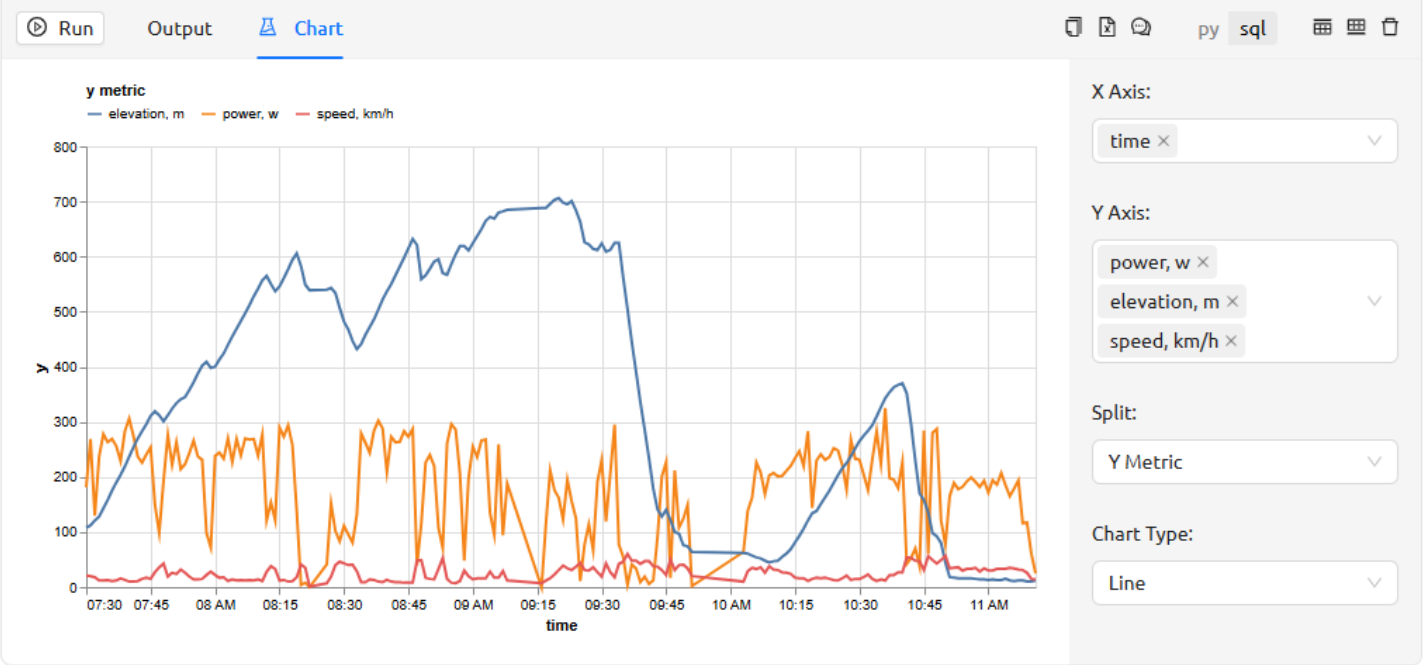
Let's generate a graph that simultaneously shows speed, altitude and power inside one given activity with minute-by-minute segmentation and an additional start time constraint. We use the functions `avg`, `round` and `date_trunc` with the parameter `minute`.

```
SELECT
    round(avg(power)) AS "power, w",
    round(avg(elevation)) AS "elevation, m",
    round(avg(speed)) AS "speed, km/h",
    date_trunc('minute', time_stamp) as time,
FROM demo_geo_points_silver
WHERE track_id = '16627535673.fit' and time_stamp < (timestamp '2025-08-24 07:30:00')
GROUP BY TIME
ORDER BY TIME
```

```
Done in 8.5 sec.
```

```
+----------+--------------+-------------+---------------------+
| power, w | elevation, m | speed, km/h | time                |
+----------+--------------+-------------+---------------------+
| 181      | 108          | 21          | 2025-08-24 07:30:00 |
+----------+--------------+-------------+---------------------+
| 269      | 112          | 20          | 2025-08-24 07:31:00 |
+----------+--------------+-------------+---------------------+
| 130      | 121          | 18          | 2025-08-24 07:32:00 |
+----------+--------------+-------------+---------------------+
| 237      | 128          | 12          | 2025-08-24 07:33:00 |
+----------+--------------+-------------+---------------------+
| 278      | 143          | 12          | 2025-08-24 07:34:00 |
+----------+--------------+-------------+---------------------+
| ...      | ...          | ...         | ...                 |
+----------+--------------+-------------+---------------------+
201 rows
```

To display the chart, let's use the **Chart** tab in the cell output interface:



We have a graph that simultaneously displays several different metrics for a given activity — speed, altitude, and power. Hovering the mouse over a line on the graph shows the value at that point. In these graphs, users can examine the metrics as they relate to each other. For example, in this case, you can see that speed increases on downhills and power decreases on uphills, while the opposite is true on uphills.

Power distribution within an activity

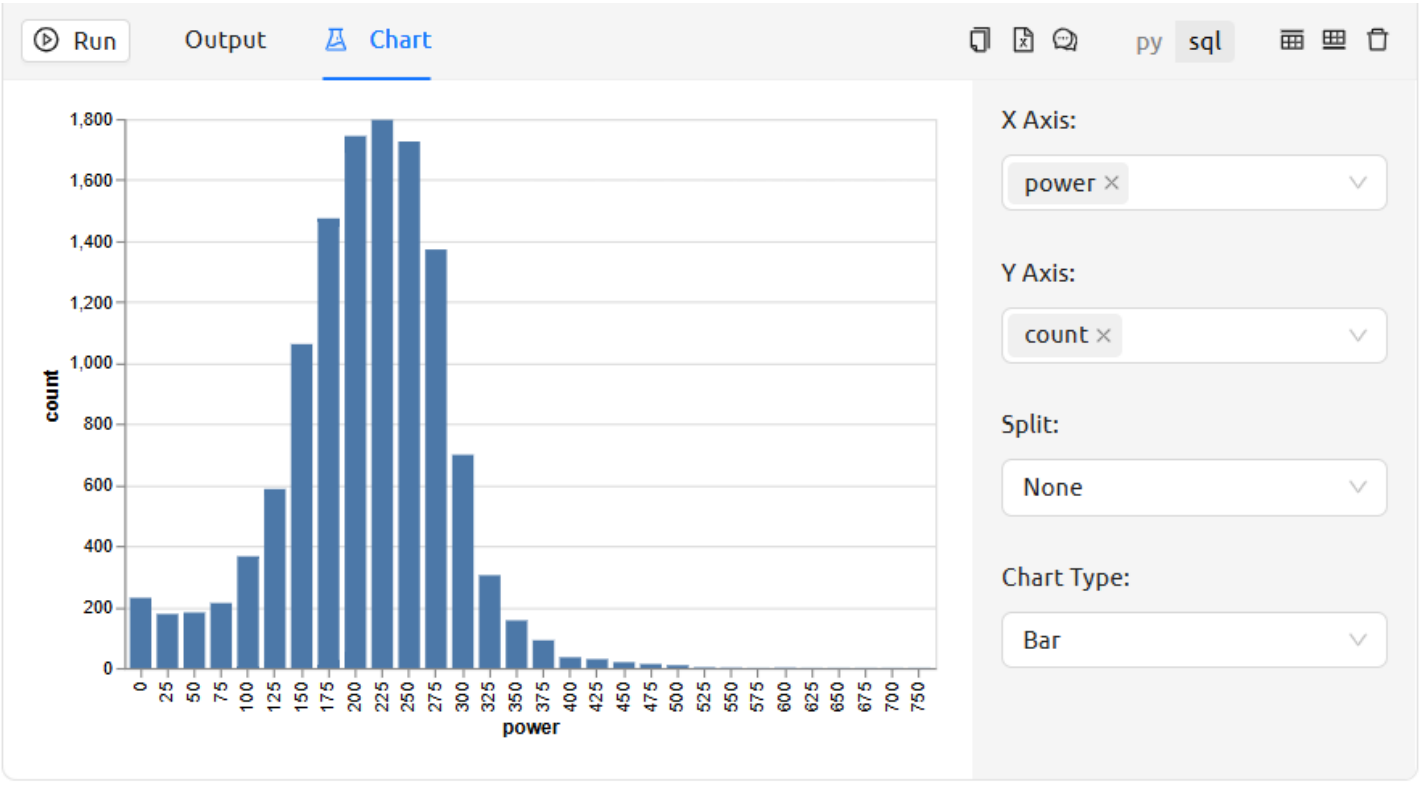
Let's plot the power distribution within the same activity in 25 watt increments. To do this, let's use the function `floor`.

```
SELECT
    floor(power/25)*25 AS power,
    count(*) AS count
```

```
FROM demo_geo_points_silver
WHERE track_id = '16627535673.fit' AND power > 0
GROUP BY 1
ORDER BY 1
```

```
Done in 8.2 sec.

+-------+-------+
| power | count |
+-------+-------+
| 0     | 230   |
+-------+-------+
| 25    | 177   |
+-------+-------+
| 50    | 182   |
+-------+-------+
| 75    | 213   |
+-------+-------+
| 100   | 366   |
+-------+-------+
| ...   | ...   |
+-------+-------+
30 rows
```

To display the chart, let's use the **Chart** tab in the cell output interface:



We have obtained a graph of power distribution within one activity in 25 watt increments.

Statistics by calendar periods

To display detailed statistics by calendar periods (months, days of the week, hours within a day, etc.), let's create a new table based on `demo_geo_tracks_silver`, in which we will add columns corresponding to different time periods. Use the `concat` and `regexp_extract` functions for this purpose.

```
CREATE OR REPLACE TABLE demo_geo_tracks_silver_time AS
SELECT
    track_id,
    hour(time_stamp) AS hour,
    regexp_extract(string(time_stamp), '.{5} ') as day_of_year,
    month(time_stamp) AS month,
    monthname(time_stamp) AS month_name,
    dayname(time_stamp) AS day_name,
    concat(regexp_extract(string(day_of_year), '.{2} $') , ' ', month_name) as text_date,
FROM demo_geo_tracks_silver;

SELECT *
    FROM demo_geo_tracks_silver_time
    LIMIT 10;
```

```
+----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| track_id       | hour | day_of_year | month_year | month | year | month_name | day_name  |
text_date   |
+----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| 8205566762.fit | 16   | 08-23       | 2022-08    | 8     | 2022 | August     | Tuesday   |
23 August   |
+----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| 5758691664.fit | 18   | 06-03       | 2021-06    | 6     | 2021 | June       | Thursday  |
03 June     |
+----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| 7562241822.fit | 15   | 05-07       | 2022-05    | 5     | 2022 | May        | Saturday  |
07 May      |
+----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| 5752363946.fit | 15   | 06-02       | 2021-06    | 6     | 2021 | June       | Wednesday |
02 June     |
+----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| 5774225138.fit | 8    | 06-06       | 2021-06    | 6     | 2021 | June       | Sunday    |
06 June     |
+----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| 3602018438.fit | 14   | 04-29       | 2020-04    | 4     | 2020 | April      | Wednesday |
29 April    |
+----------------+------+-------------+------------+-------+------+------------+-----------
```

```
+-------------+
| 2665481253.fit  | 18   | 07-06       | 2019-07    | 7     | 2019 | July       | Saturday  |
06 July     |
+-----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| 5061483340.fit  | 7    | 02-06       | 2021-02    | 2     | 2021 | February   | Saturday  |
06 February |
+-----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| 1842519266.fit  | 9    | 07-21       | 2018-07    | 7     | 2018 | July       | Saturday  |
21 July     |
+-----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
| 11417255201.fit | 14   | 02-01       | 2024-02    | 2     | 2024 | February   | Thursday  |
01 February |
+-----------------+------+-------------+------------+-------+------+------------+-----------
+-------------+
```

Let's make a graph that shows the total distance in kilometres over the entire time in activities starting at each hour of the day. To join the `demo_geo_tracks_silver_time` and `demo_geo_tracks_silver` tables by the `track_id` column, we will use the expression `JOIN` with the `USING` parameter.

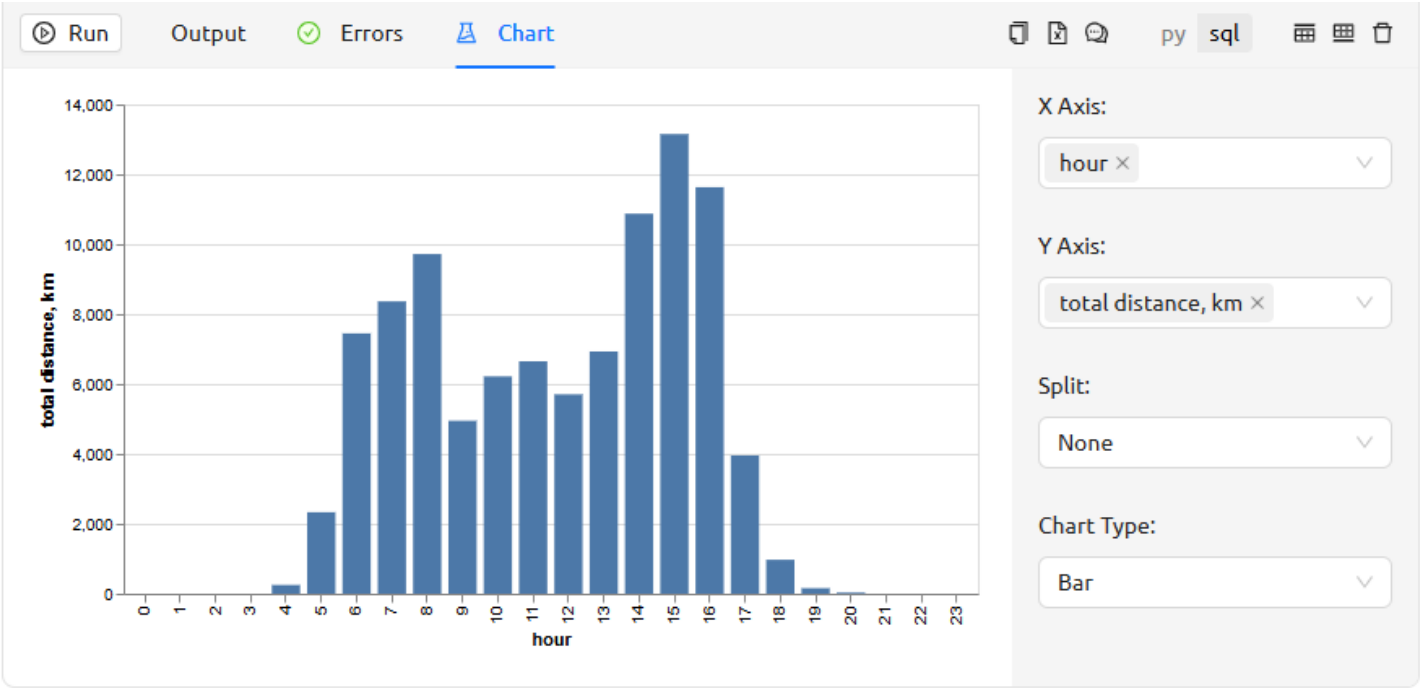
To fill the rows with empty hours we use the functions `unnest` and `generate_series`.

```
CREATE OR REPLACE TABLE demo_geo_tracks_hour AS
SELECT
    distance,
    hour,
    time_stamp
FROM demo_geo_tracks_silver_time
    JOIN demo_geo_tracks_silver
        USING (track_id);

-- Добавляем пустые часы
INSERT INTO demo_geo_tracks_hour (hour, distance)
    SELECT unnest(generate_series(23)), 0;

SELECT
    round(sum(distance)) AS "Total distance, km",
    hour AS Hour
FROM demo_geo_tracks_hour
GROUP BY 2
ORDER BY 2;
```

```
Done in 9.5 sec.

+--------------------+------+
| Total distance, km | Hour |
+--------------------+------+
| 0                  | 0    |
```

```
+--------------------+------+
| 0                  | 1    |
+--------------------+------+
| 0                  | 2    |
+--------------------+------+
| 0                  | 3    |
+--------------------+------+
| 250                | 4    |
+--------------------+------+
| 2332               | 5    |
+--------------------+------+
| 7448               | 6    |
+--------------------+------+
| 8365               | 7    |
+--------------------+------+
| 9719               | 8    |
+--------------------+------+
| 4949               | 9    |
+--------------------+------+
| ...                | ...  |
+--------------------+------+
24 rows
```

To display the chart, let's use the **Chart** tab in the cell output interface:



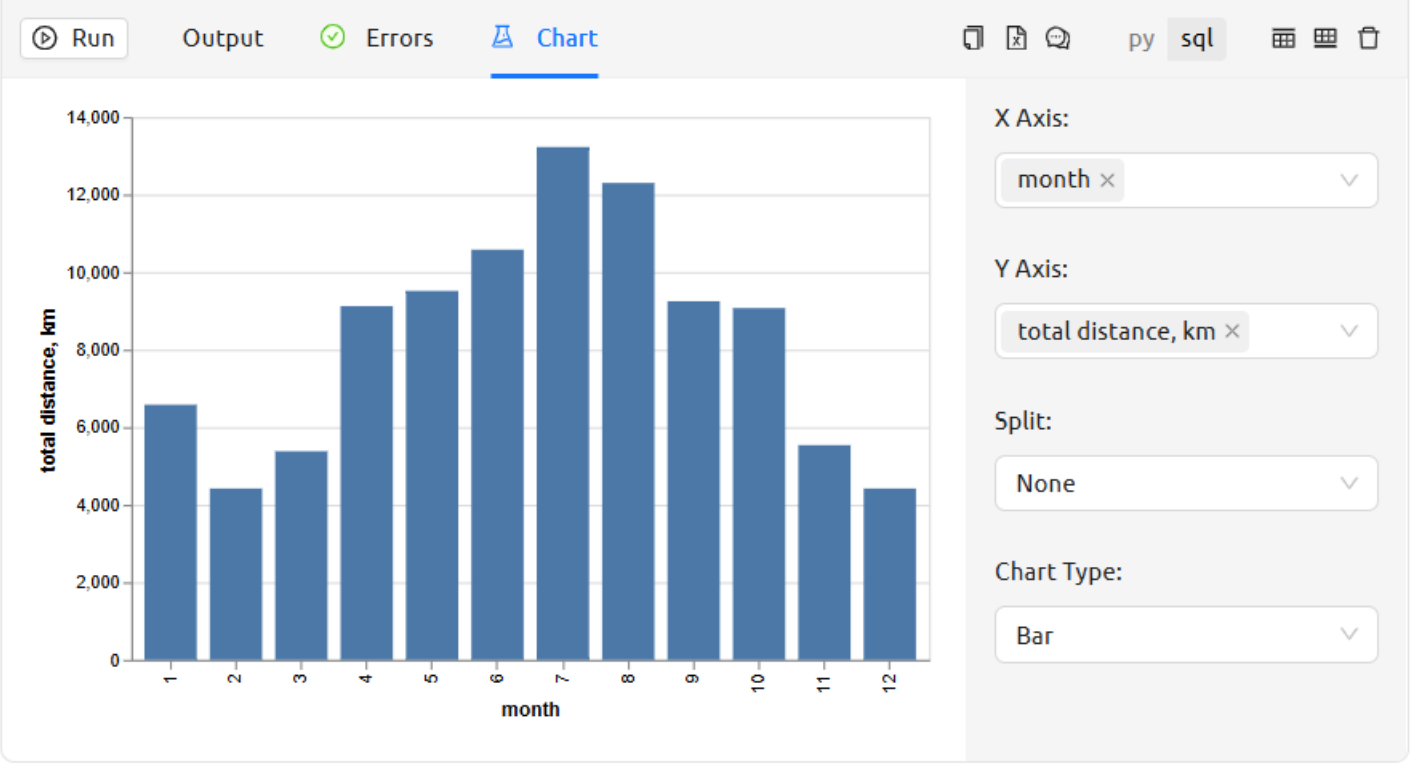
Now plot a graph to show the total distance in kilometres for each month for the whole time.

```
SELECT
    round(sum(distance)) AS "Total distance, km",
    month,
FROM demo_geo_tracks_silver_time
    JOIN demo_geo_tracks_silver
        USING (track_id)
```

```
GROUP BY 2
ORDER BY 2
```

```
Done in 8.2 sec.

+--------------------+-------+
| total distance, km | month |
+--------------------+-------+
| 6580               | 1     |
+--------------------+-------+
| 4419               | 2     |
+--------------------+-------+
| 5380               | 3     |
+--------------------+-------+
| 9122               | 4     |
+--------------------+-------+
| 9514               | 5     |
+--------------------+-------+
| 10578              | 6     |
+--------------------+-------+
| 13225              | 7     |
+--------------------+-------+
| 12298              | 8     |
+--------------------+-------+
| 9250               | 9     |
+--------------------+-------+
| 9072               | 10    |
+--------------------+-------+
| 5537               | 11    |
+--------------------+-------+
| 4419               | 12    |
+--------------------+-------+
```

To display the chart, let's use the **Chart** tab in the cell output interface:



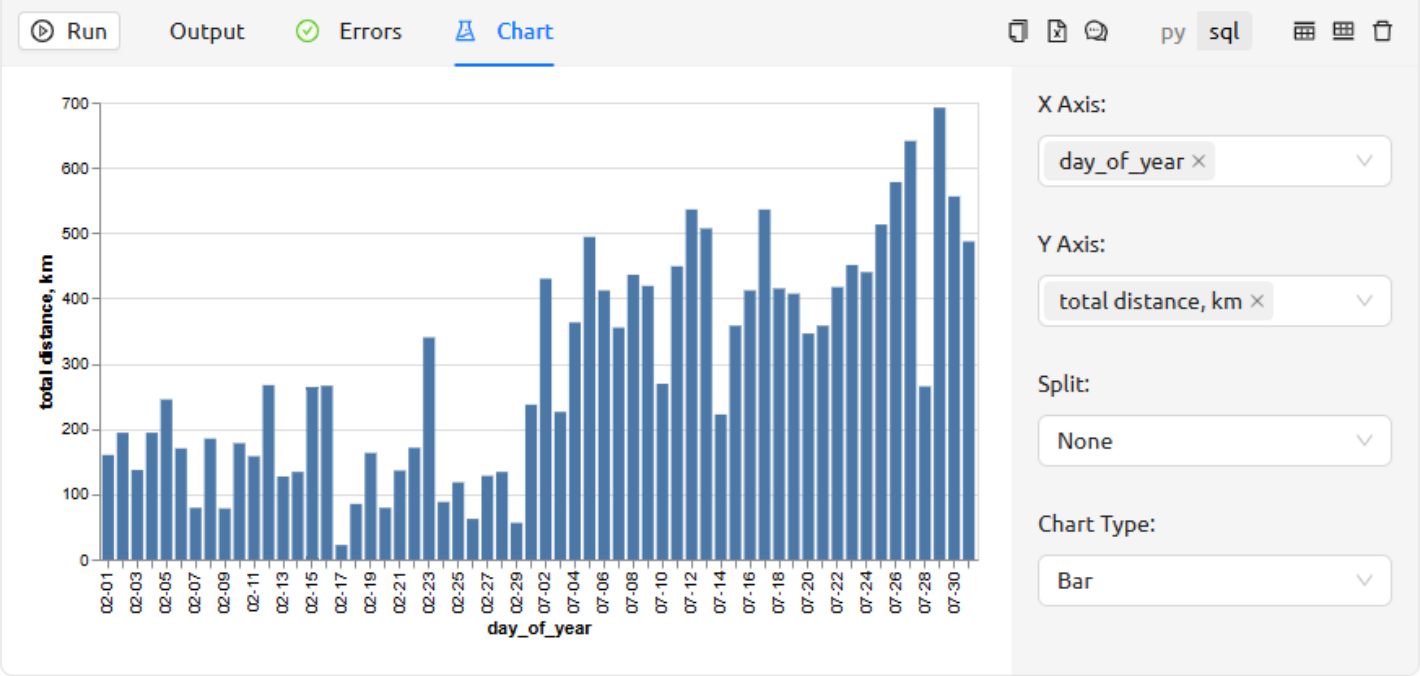
Expectedly, in the winter months, the total distance is significantly less than in the summer months.

Let us now plot the total distances in kilometres for each date in the given months on a single graph. For this example, let's take February and July as the months with the lowest and highest total distance.

```
SELECT
    round(sum(distance)) AS "Total distance, km",
    day_of_year,
FROM demo_geo_tracks_silver_time
    JOIN demo_geo_tracks_silver
        USING (track_id)
WHERE month IN [2,7]
GROUP BY 2
ORDER BY 2
```

```
Done in 8.1 sec.

+--------------------+-------------+
| total distance, km | day_of_year |
+--------------------+-------------+
| 160                | 02-01       |
+--------------------+-------------+
| 194                | 02-02       |
+--------------------+-------------+
| 137                | 02-03       |
+--------------------+-------------+
| 194                | 02-04       |
+--------------------+-------------+
| 245                | 02-05       |
+--------------------+-------------+
```

```
| ...                | ...         |
+--------------------+-------------+
60 rows
```



Now let's rank the days of the week — order the days of the week by total distance in kilometres for the whole time, starting with the maximum.

```
SELECT
    round(sum(distance)) AS "Total distance, km",
    day_name as "Day of week",
FROM demo_geo_tracks_silver_time
    JOIN demo_geo_tracks_silver
        USING track_id
GROUP BY 2
ORDER BY 1 DESC
```

```
Done in 8.4 sec.

+--------------------+-------------+
| total distance, km | day of week |
+--------------------+-------------+
| 21636              | Saturday    |
+--------------------+-------------+
| 19006              | Sunday      |
+--------------------+-------------+
| 13817              | Thursday    |
+--------------------+-------------+
| 12453              | Tuesday     |
+--------------------+-------------+
| 12400              | Wednesday   |
+--------------------+-------------+
| 10696              | Friday      |
```

```
+--------------------+-------------+
| 9386               | Monday      |
+--------------------+-------------+
```

Changes in indicators for the whole period

Now let's display on one graph the changes in activity indicators over time with segmentation by quarters. To do this, we will use the `date_trunc` function with the `quarter` parameter. Let's take the following indicators:

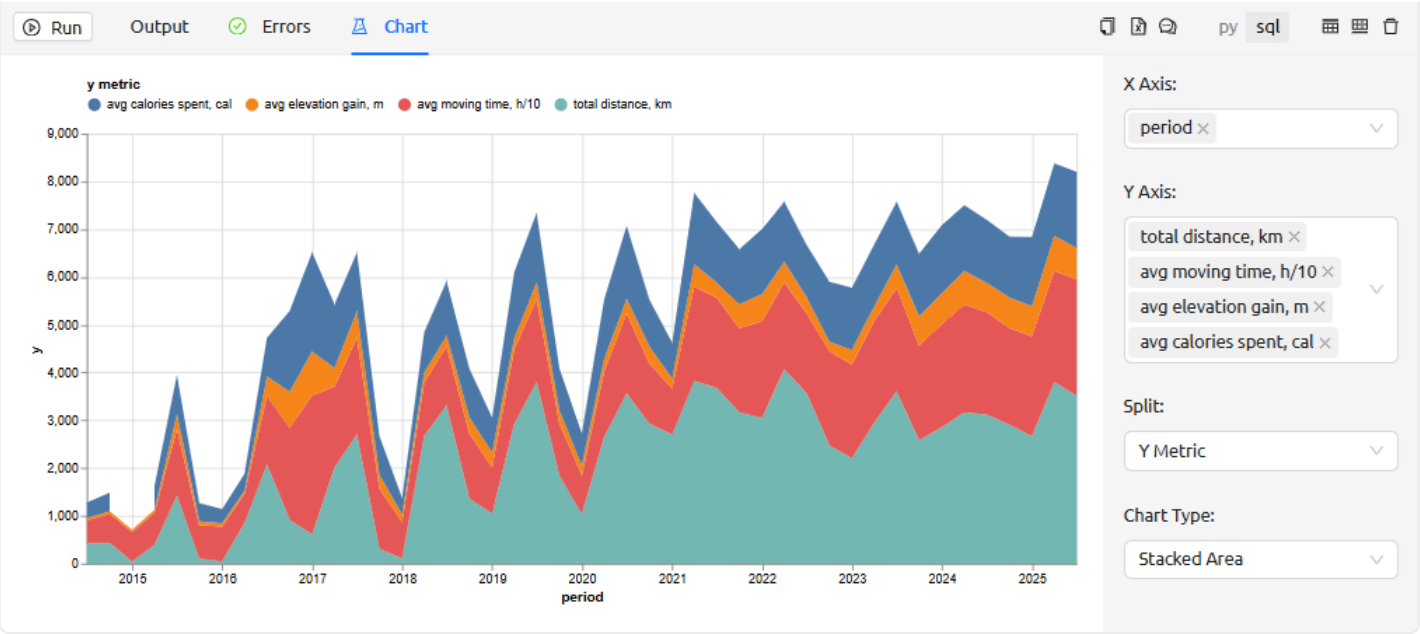
- total distance
- average time per activity
- average altitude gain per activity
- average number of calories consumed per activity

```
SELECT
    round(avg("Elevation Gain")) AS "avg elevation gain, m",
    round(avg("Moving Time")/3.6) AS "avg moving time, h/10",
    round(avg(Calories)) AS "avg calories spent, cal",
    round(sum(distance)) AS "total distance, km",
    period
FROM (SELECT
        distance,
        "Elevation Gain",
        "Moving Time",
        Calories,
        date_trunc('quarter', time_stamp) AS period
        FROM demo_geo_tracks_silver
            JOIN demo_geo_tracks_silver_time
                USING (track_id)
        )
GROUP BY period
ORDER BY period
```

```
Done in 8.3 sec.

+----------------------+----------------------+-------------------------
+--------------------+------------+
| avg elevation gain, m | avg moving time, h/10 | avg calories spent, cal | total distance,
km | period     |
+----------------------+----------------------+-------------------------
+--------------------+------------+
| 52                   | 485                  | 322                     | 423
| 2014-07-01 |
+----------------------+----------------------+-------------------------
+--------------------+------------+
| 60                   | 597                  | 389                     | 438
| 2014-10-01 |
```

```
+----------------------+----------------------+-------------------------
+--------------------+------------+
| 46                   | 623                  | null                    | 41
| 2015-01-01 |
+----------------------+----------------------+-------------------------
+--------------------+------------+
| 60                   | 676                  | 502                     | 390
| 2015-04-01 |
+----------------------+----------------------+-------------------------
+--------------------+------------+
| 310                  | 1394                 | 816                     | 1426
| 2015-07-01 |
+----------------------+----------------------+-------------------------
+--------------------+------------+
| ...                  | ...                  | ...                     | ...
| ...        |
+----------------------+----------------------+-------------------------
+--------------------+------------+
45 rows
```

Since changes in all these indicators generally indicate an improvement in the athlete's training level, we will choose the **Stacked Area** chart type, in which the values of the indicators are added together:



This graph shows, firstly, an increase in all the indicators and, secondly, a decrease in the seasonal factor over time.

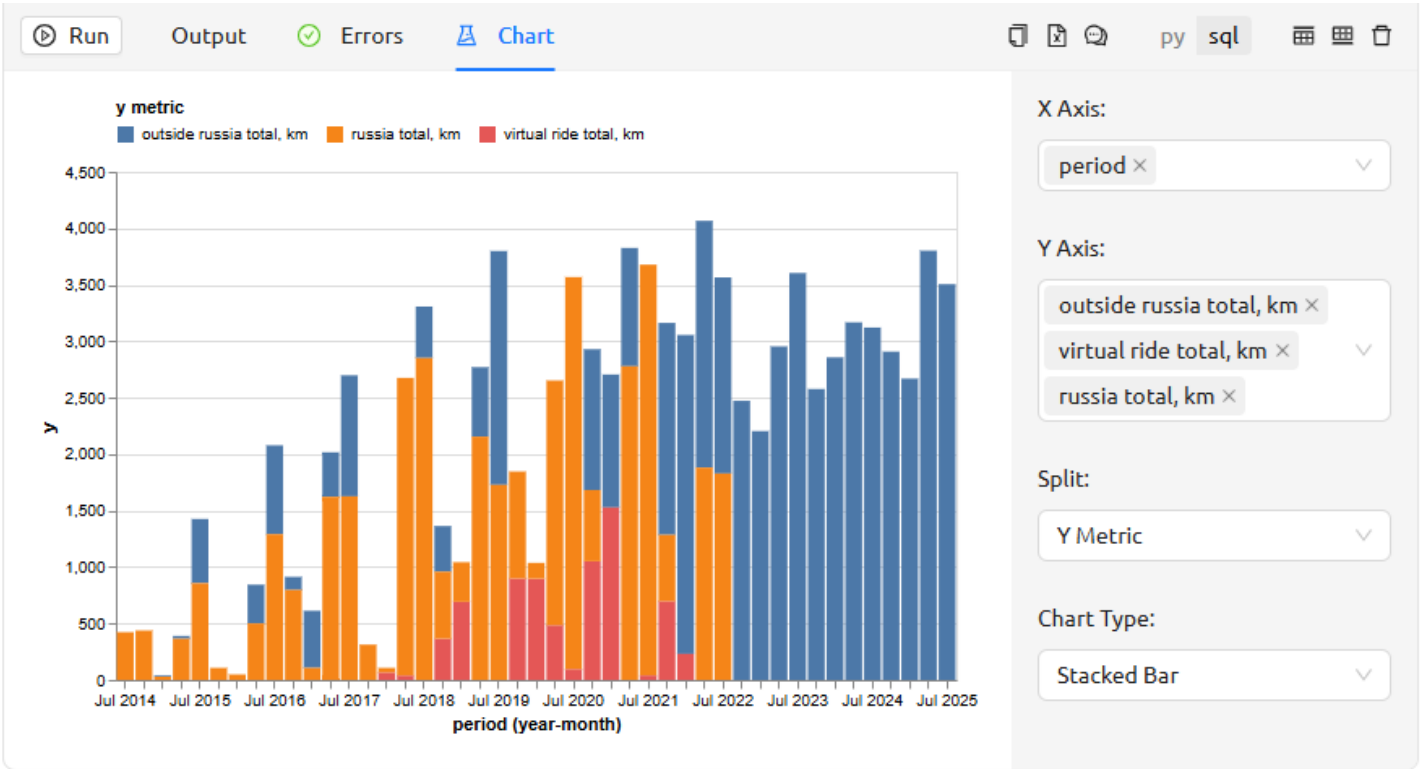
Let's display statistics on total mileage in Russia / outside Russia and in virtual races with segmentation by quarters in one graph.

```
SELECT
    round(sum(CASE WHEN country == 'Россия' AND NOT "Activity Type" = 'Virtual Ride'
        THEN distance END))
    AS "Russia total, km",
    round(sum(CASE WHEN NOT country == 'Россия' AND NOT "Activity Type" = 'Virtual Ride'
```

```
        THEN distance END))
    AS "Outside Russia total, km",
    round(sum(CASE WHEN "Activity Type" = 'Virtual Ride' THEN distance END))
    AS "Virtual Ride total, km",
    date_trunc('quarter', time_stamp) AS period
FROM demo.geo_tracks_silver
GROUP BY period
ORDER BY period
```

```
Done in 8.7 sec.

+------------------+---------------------------+------------------------+------------+
| russia total, km | outside russia total, km | virtual ride total, km | period     |
+------------------+---------------------------+------------------------+------------+
| 423              | null                      | null                   | 2014-07-01 |
+------------------+---------------------------+------------------------+------------+
| 438              | null                      | null                   | 2014-10-01 |
+------------------+---------------------------+------------------------+------------+
| 29               | 12                        | null                   | 2015-01-01 |
+------------------+---------------------------+------------------------+------------+
| 365              | 25                        | null                   | 2015-04-01 |
+------------------+---------------------------+------------------------+------------+
| 857              | 569                       | null                   | 2015-07-01 |
+------------------+---------------------------+------------------------+------------+
| ...              | ...                       | ...                    | ...        |
+------------------+---------------------------+------------------------+------------+
45 rows
```

The **Stacked Bar** type would work well for this type of chart:



In this graph you can see different dependencies: the seasonal factor affects the number of virtual rides — in winter there are more of them; activities outside Russia are first only in summer periods, and later the activities outside Russia remain the only ones. We can also see a general decrease in the seasonal factor, as in the previous graph.

Analysing points by their distance from a given point

We know that the user whose activity archive we are studying lived in Moscow, so let's try to analyse his activity depending on the distance from Moscow. To do this, let's take the coordinates zero kilometre of Moscow: `55.75579845052788, 37.617679973467204` and create a table where for each activity we will calculate the distance of its starting point from Moscow in kilometres:

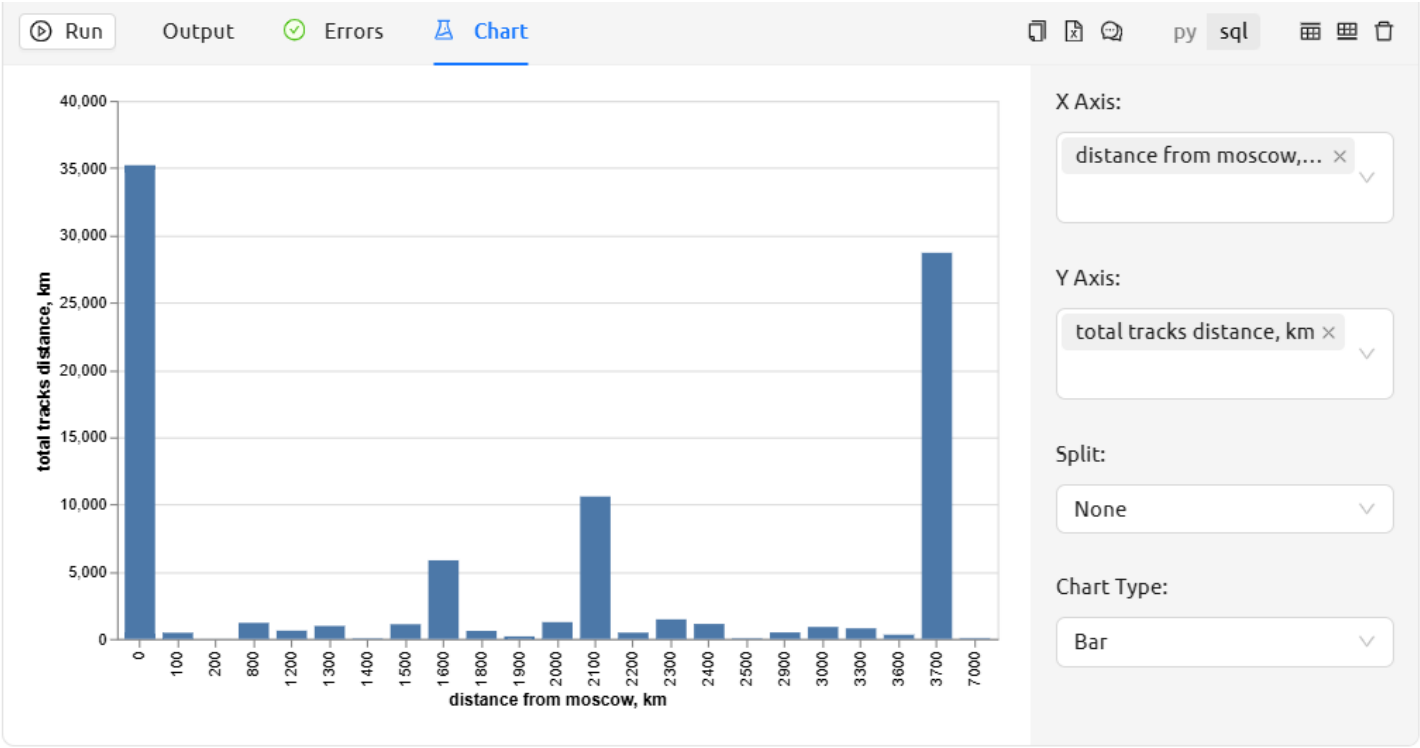
```
CREATE OR REPLACE TABLE demo_geo_tracks_by_moscow AS
SELECT
    ST_Distance_Sphere(ST_Point(track_first_coord_lat, track_first_coord_lon),
                       ST_Point(55.75579845052788, 37.617679973467204)
                       )/1000
    AS km_from_moscow,
    track_id,
FROM demo_geo_tracks_silver;

SELECT * FROM demo_geo_tracks_by_moscow
    LIMIT 10;
```

```
Done in 9.3 sec.

+--------------------+-----------------+
| km_from_moscow     | track_id        |
+--------------------+-----------------+
| 1614.8913738847054 | 8205566762.fit  |
+--------------------+-----------------+
| 8.000389153235107  | 5758691664.fit  |
+--------------------+-----------------+
| 2196.013494904038  | 7562241822.fit  |
+--------------------+-----------------+
| 6.0300128287340815 | 5752363946.fit  |
+--------------------+-----------------+
| 88.17899082303168  | 5774225138.fit  |
+--------------------+-----------------+
| 6.033077121995654  | 3602018438.fit  |
+--------------------+-----------------+
| 890.3163236663786  | 2665481253.fit  |
+--------------------+-----------------+
| 7501.918475014584  | 5061483340.fit  |
+--------------------+-----------------+
| 2.926600115019104  | 1842519266.fit  |
+--------------------+-----------------+
| 3770.6424243313804 | 11417255201.fit |
+--------------------+-----------------+
```

Let’s look at the distribution of the total length of the tracks depending on the distance of the track start from Moscow in steps of 100 kilometres. For this purpose we will use the function `floor`.

```
SELECT
    floor(km_from_moscow/100)*100 AS "Distance from Moscow, km",
    round(sum(distance)) AS "Total tracks distance, km"
FROM demo_geo_tracks_by_moscow
    JOIN demo_geo_tracks_silver
        USING (track_id)
    WHERE NOT "Activity Type" = 'Virtual Ride'
GROUP BY 1
ORDER BY 1;
```

```
Done in 8.3 sec.

+--------------------------+---------------------------+
| distance from moscow, km | total tracks distance, km |
+--------------------------+---------------------------+
| 0                        | 35205                     |
+--------------------------+---------------------------+
| 100                      | 457                       |
+--------------------------+---------------------------+
| 200                      | 5                         |
+--------------------------+---------------------------+
| 800                      | 1191                      |
+--------------------------+---------------------------+
| 1200                     | 613                       |
+--------------------------+---------------------------+
| ...                      | ...                       |
+--------------------------+---------------------------+
29 rows
```



On the graph we can see that there are several columns noticeably higher than the neighbouring ones. They correspond to the different cities from which the user made regular workouts. Apart from Moscow, these are three cities:

- Malaga — `36.71011042348239, -4.429510437480414`
- Berlin — `52.51622694136956, 13.377732359062023`
- Alanya — `36.544934513586334, 31.993451012738028`

Let's calculate for all activities the distances to these cities:

```
CREATE OR REPLACE TABLE demo_geo_tracks_by_cities AS
SELECT
    track_id,
    distance,
    ST_Distance_Sphere(ST_Point(track_first_coord_lat, track_first_coord_lon),
                       ST_Point(55.75579845052788, 37.617679973467204)
                       )/1000
        AS km_from_moscow,
    ST_Distance_Sphere(ST_Point(track_first_coord_lat, track_first_coord_lon),
                       ST_Point(36.71011042348239, -4.429510437480414)
                       )/1000
        AS km_from_malaga,
    ST_Distance_Sphere(ST_Point(track_first_coord_lat, track_first_coord_lon),
                       ST_Point(52.51622694136956, 13.377732359062023)
                       )/1000
        AS km_from_berlin,
    ST_Distance_Sphere(ST_Point(track_first_coord_lat, track_first_coord_lon),
                       ST_Point(36.544934513586334, 31.993451012738028)
                       )/1000
        AS km_from_alanya,
FROM demo_geo_tracks_silver;
```

```
SELECT * FROM demo_geo_tracks_by_cities
    LIMIT 10;
```

```
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| track_id       | distance | km_from_moscow     | km_from_malaga     | km_from_berlin     |
km_from_alanya     |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 8205566762.fit  | 51.04    | 1614.8913738847054 | 2235.249319891205  | 5.0145197840760245 |
2294.093997594629  |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 5758691664.fit  | 3.91     | 8.000389153235107  | 3787.2509319742358 | 1618.720214954235  |
2180.111417985302  |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 7562241822.fit  | 21.69    | 2196.013494904038  | 3104.0466510427395 | 2221.451446627775  |
127.78858146201004 |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 5752363946.fit  | 62.85    | 6.0300128287340815 | 3785.4665468428007 | 1616.7118606890685 |
2180.242964284234  |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 5774225138.fit  | 114.24   | 88.17899082303168  | 3707.0000971542977 | 1528.712391035889  |
2186.697705370458  |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 3602018438.fit  | 34.76    | 6.033077121995654  | 3785.4691666595877 | 1616.7152967306572 |
2180.24094049402   |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 2665481253.fit  | 43.88    | 890.3163236663786  | 3051.4212916480724 | 811.7057240823138  |
2370.646300384682  |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 5061483340.fit  | 16.87    | 7501.918475014584  | 5886.25300373306   | 6375.612772054101  |
8583.236350749687  |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 1842519266.fit  | 71.94    | 2.926600115019104  | 3780.985953704283  | 1612.7664280078745 |
2176.465565328427  |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
| 11417255201.fit | 30.91    | 3770.6424243313804 | 12.516233931245793 | 2232.9498881694144 |
3217.9897501282717 |
+----------------+----------+--------------------+--------------------+--------------------
+--------------------+
```

Let's derive statistics on the total distance of tracks with start in different cities. As a threshold value of distance we take `100 km`:

```
SELECT
    round(sum(CASE WHEN km_from_moscow < 100 THEN distance END))
        AS "Moscow total, km",
    round(sum(CASE WHEN km_from_malaga < 100 THEN distance END))
        AS "Malaga total, km",
    round(sum(CASE WHEN km_from_alanya < 100 THEN distance END))
        AS "Alanya total, km",
    round(sum(CASE WHEN km_from_berlin < 100 THEN distance END))
        AS "Berlin total, km",
    round(sum(CASE WHEN
        (
            km_from_moscow >= 100 AND
            km_from_malaga >= 100 AND
            km_from_alanya >= 100 AND
            km_from_berlin >= 100
        )
        THEN distance END))
        AS "Rest total, km",
    round(sum(distance)) AS "Total, km"
FROM demo_geo_tracks_by_cities
```

```
+------------------+------------------+------------------+------------------+----------------
+-----------+
| moscow total, km | malaga total, km | alanya total, km | berlin total, km | rest total, km
| total, km |
+------------------+------------------+------------------+------------------+----------------
+-----------+
| 35155            | 28969            | 8136             | 5841             | 21242
| 99343     |
+------------------+------------------+------------------+------------------+----------------
+-----------+
```

Displaying multiple points on the map

To demonstrate the display of points on the map, let's create a separate table with points of one activity within Moscow with the maximum distance for the whole time.

```
CREATE OR REPLACE TABLE demo_geo_points_for_map AS
    SELECT * FROM demo_geo_points_silver
        WHERE
        (track_id IN (
        SELECT track_id
            from demo_geo_tracks_by_distance
                JOIN demo_geo_tracks_silver
                    USING (track_id)
            WHERE km_from_moscow < 100
```

```
            ORDER BY distance DESC
            LIMIT 1
            )
    );

SELECT count(*) AS "Points for map total"
    FROM demo_geo_points_for_map;
```

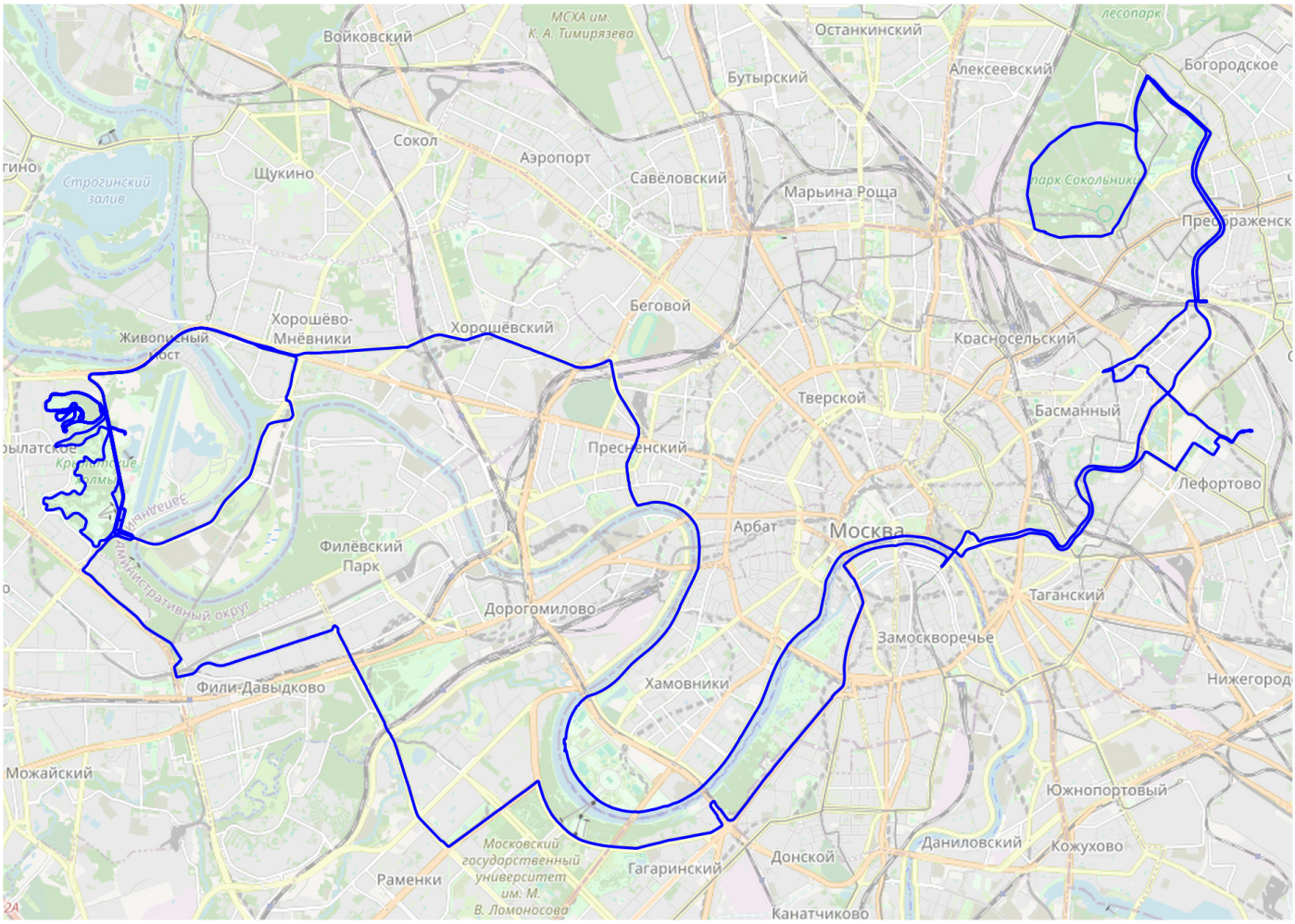
```
Done in 9.7 sec.

+----------------------+
| points for map total |
+----------------------+
| 16458                |
+----------------------+
```

Now in a cell of type Python we will write the code that can be used to access the created table with filtered points and display them on the map OSM using the module Python TileMapBase:

```
import tngri
import pyarrow
import pyiceberg
from pyiceberg.catalog import load_catalog
from contextlib import suppress
import pandas as pd
import matplotlib.pyplot as plt
import tilemapbase
import warnings
warnings.filterwarnings("ignore")

def print_map(source_table_name):

    # Load the source table
    source_table = catalog.load_table(source_table_name)

    # Convert to DataFrame
    trip_df = source_table.scan().to_pandas()
    print(f'Source table size: {trip_df.shape}')

    tilemapbase.init(create=True)

    # Size of map margins around the extreme points of the track
    expand=0.01
    extent = tilemapbase.Extent.from_lonlat(
        trip_df.longitude.min() - expand,
        trip_df.longitude.max() + expand,
        trip_df.latitude.min() - expand,
        trip_df.latitude.max() + expand,
    )
```

```
    track_names = trip_df['track_id'].unique()
    print(f 'Number of tracks: {len(track_names)}')

    # Print the total distance of the tracks in the source table
    print(tngri.sql('''
    select round(sum(distance)) as "Total distance, km"
    from demo_geo_tracks_silver
    where track_id in (
            select track_id
            from demo_geo_points_for_map)
    '''))

    track_list = []
    for track_name in track_names:
        track = trip_df.loc[trip_df['track_id'] == track_name].apply(
            lambda x: tilemapbase.project(x.longitude, x.latitude), axis=1
        ).apply(pd.Series)
        track.columns = ["x", "y"]
        track_list.append(track)

    tiles = tilemapbase.tiles.build_OSM()

    fig, ax = plt.subplots(figsize=(8, 8), dpi=300)

    # height -- map detail
    plotter = tilemapbase.Plotter(extent, tiles, height=600)

    # alpha -- map clutter
    plotter.plot(ax, tiles, alpha=0.6)

    # If there are several tracks, make the line thickness smaller
    if len(track_names) == 1:
        width = 1
    else:
        width = 0.5
    for track in track_list:
        ax.plot(track.x, track.y, color='blue', linewidth=width)

    plt.axis('off')
    fig.get_figure()

print_map('demo_geo_points_for_map')
```

```
Done in 30 sec.

Размер исходной таблицы: (16458, 8)
Количество треков: 1
Суммарная дистанция треков: 135.0 км
```



Note that to output the total distance of the displayed tracks, we use the `SELECT` query call in cell Python via the `tngri.sql` function.

Now let's overwrite the table with the coordinates of the points to be displayed on the map, selecting all activities with a start in Berlin to display them all on the same map.

```
CREATE OR REPLACE TABLE demo_geo_points_for_map AS
    SELECT
        latitude,
        longitude,
        track_id
    FROM demo_geo_points_silver
    WHERE
    (track_id IN (
        SELECT track_id
            FROM demo_geo_tracks_by_distance
            WHERE km_from_berlin < 100
        )
    );

SELECT * FROM demo_geo_points_for_map
    LIMIT 10;
```

```
Done in 9.6 sec.
```

```
+--------------------+--------------------+----------------+
| latitude           | longitude          | track_id       |
+--------------------+--------------------+----------------+
| 52.529693618416786 | 13.444848032668233 | 9440992383.fit |
+--------------------+--------------------+----------------+
| 52.52970124594867  | 13.44483470544219  | 9440992383.fit |
+--------------------+--------------------+----------------+
| 52.52970124594867  | 13.44481467269361  | 9440992383.fit |
+--------------------+--------------------+----------------+
| 52.52969739027321  | 13.444789862260222 | 9440992383.fit |
+--------------------+--------------------+----------------+
| 52.5296859908849   | 13.444773685187101 | 9440992383.fit |
+--------------------+--------------------+----------------+
| 52.52967827953398  | 13.4447565022856   | 9440992383.fit |
+--------------------+--------------------+----------------+
| 52.529674507677555 | 13.444727919995785 | 9440992383.fit |
+--------------------+--------------------+----------------+
| 52.529674507677555 | 13.44469740986824  | 9440992383.fit |
+--------------------+--------------------+----------------+
| 52.529670652002096 | 13.444674527272582 | 9440992383.fit |
+--------------------+--------------------+----------------+
| 52.529670652002096 | 13.444655416533351 | 9440992383.fit |
+--------------------+--------------------+----------------+
```

Let's output the number of points in the created table:

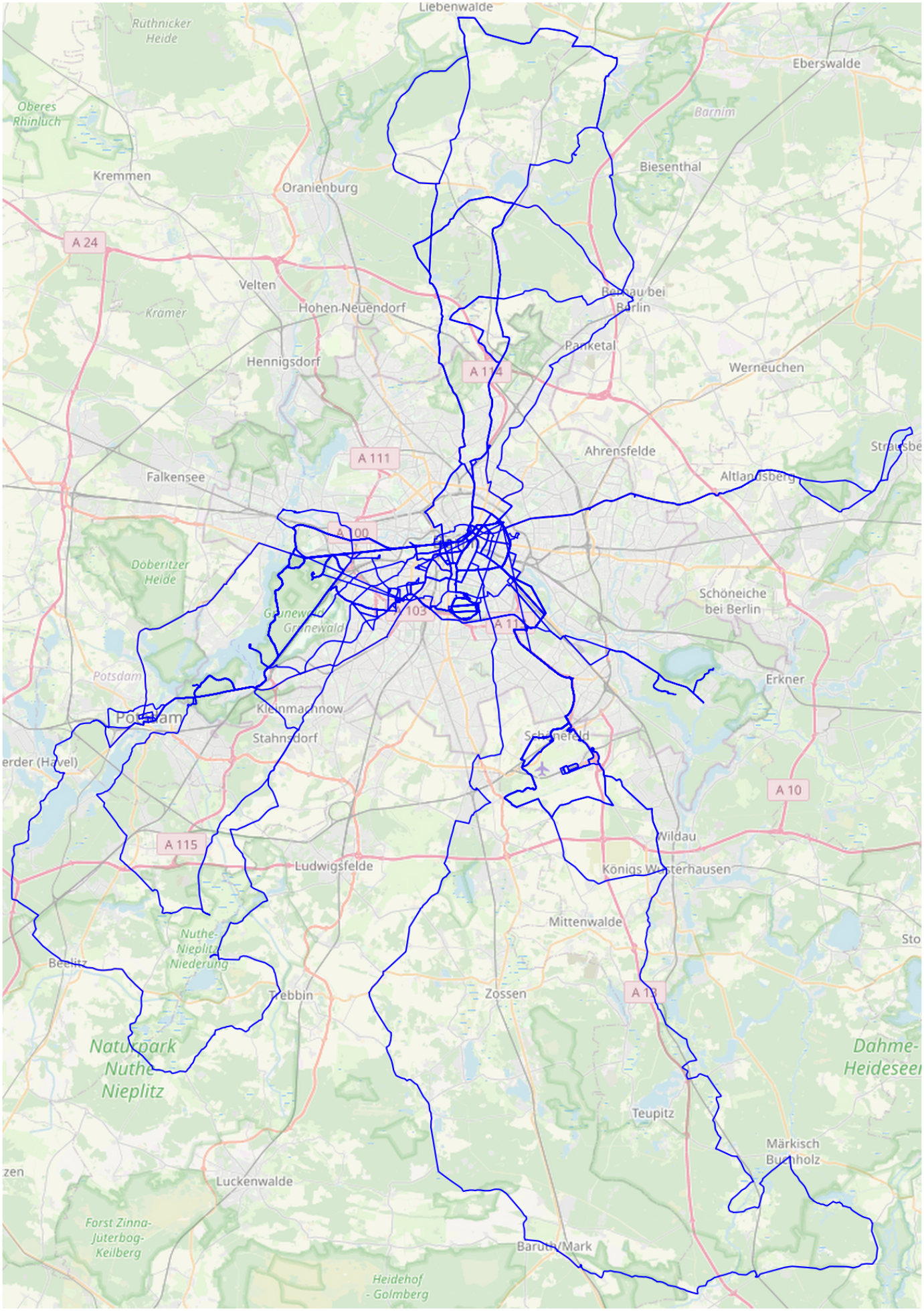
```
SELECT count(*) AS "Points for map total"
    FROM demo_geo_points_for_map
```

```
+----------------------+
| points for map total |
+----------------------+
| 708616               |
+----------------------+
```

Run the previously created `print_map` function in cell Python for the updated table with the same name and display a map with all the activities in Berlin:

```
print_map('demo_geo_points_for_map')
```

```
Done in 1 min. 6 sec.

Размер исходной таблицы: (708616, 3)
Количество треков: 108
Суммарная дистанция треков: 5841.0 км
```



Now let’s map all the tracks with the start at Velotrek in Moscow:

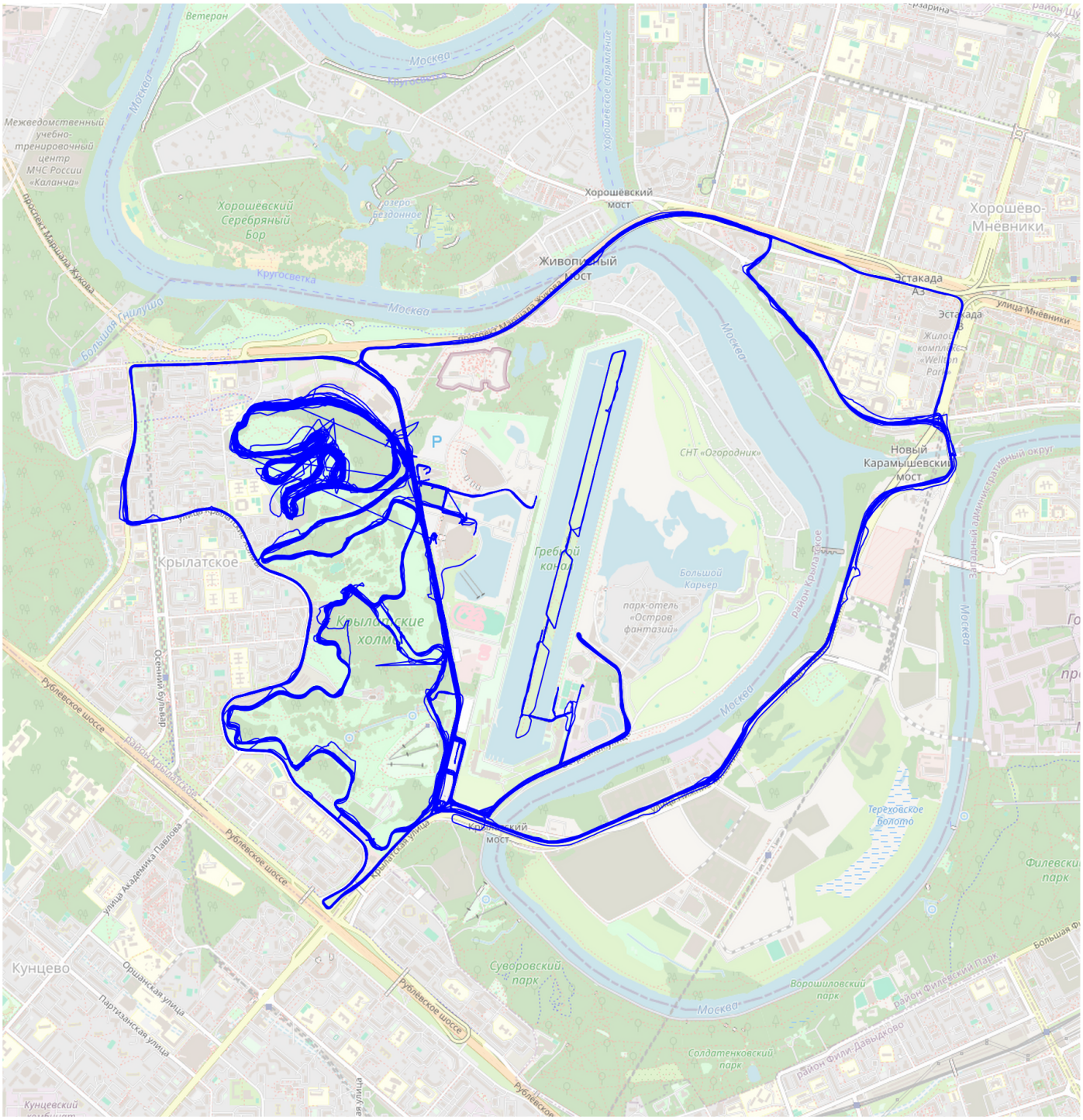
```
CREATE OR REPLACE TABLE demo.geo_points_for_map AS
SELECT
    latitude,
    longitude,
    track_id
FROM demo.geo_points_silver
WHERE track_id NOT IN
    (SELECT DISTINCT track_id
    FROM demo.geo_points_silver
    WHERE ST_Distance_Sphere(ST_Point(latitude, longitude),
                            ST_Point(55.76314006886495, 37.43303308055759)
                            )/1000 > 3);

SELECT count(*) AS "Points for map total"
    FROM demo.geo_points_for_map;
```

```
+----------------------+
| points for map total |
+----------------------+
| 215466               |
+----------------------+
```

Let's start the map display:

```
print_map('demo_geo_points_for_map')
```

```
Done in 39 sec.

Размер исходной таблицы: (215466, 3)
Количество треков: 43
Суммарная дистанция треков: 2063.0 км
```


Ветеран
Москва
Межведомственный учебно-тренировочный центр МЧС России «Каланча»
Хорошевский Серебряный Бор
Хорошевское спрямление
Хорошёвский мост
Хорошёво-Мнёвники
Живописный мост
Эстакада
Улица Мнёвники
Кругосветка
Большая Грузинская
Крылатское
Крылатские холмы
Осенний бульвар
Рублёвское шоссе
Крылатская улица
Крылатский мост
Гребной канал
Большой Карьер
парк-отель «Остров фантазий»
СНТ «Огородник»
Новый Карамышевский мост
Западный административный округ
район Крылатское
Терехово
Терехово болото
Филевский парк
Кунцево
Улица Академика Павлова
Оршанская улица
Партизанская улица
Суворовский парк
Ворошиловский парк
Район Филевский Парк
Район Филёвский Парк
Район Фили-Давыдково
Солдатенковский парк
Кунцевский

Alphabetical Index

General list of expressions SQL, functions and data types in alphabetical order.

`abs`	Function	Calculates the modulus of a number.
`add`	Function	adds numbers together.
`ALTER ROLE`	SQL expression	Change role attributes.
`ALTER SCHEMA`	SQL expression	Change schema attributes.
`ALTER USER`	SQL expression	Change user attributes.
`ALTER WORKER POOL`	SQL expression	Changing the attributes of a worker pool.
`ALTER WORKER POOL SET DEFAULT`	SQL expression	Set default attributes of the worker pool.
`any_value`	Function	Returns the first value from `argument` other than `NULL`.
`array_agg`	Function	Returns a list containing all the values of a column.
`AS`	SQL expression	Элемент запроса `SELECT`.
`avg`	Function	Calculates the average of all non-empty values in `argument`.
`BIGINT`	Data type	Integers.
`BIGINT[]`	Data type	Arrays of integers.
`BLOB`	Data type	Binary objects.
`BOOL`	Data type	Boolean values.
`BOOL[]`	Data type	Arrays of Boolean values.
`BOOLEAN`	Data type	Псевдоним типа данных `BOOL`.
`BOOLEAN[]`	Data type	Псевдоним типа данных `BOOL[]`.
`BPCHAR`	Data type	Псевдоним типа данных `VARCHAR`.
`BPCHAR[]`	Data type	Псевдоним типа данных `VARCHAR[]`.
`BYTEA`	Data type	Псевдоним типа данных `BLOB`.
`ceil`	Function	rounds a number to the higher side.
`ceiling`	Function	Псевдоним функции `ceil`.
`CHAR`	Data type	Псевдоним типа данных `VARCHAR`.
`CHAR[]`	Data type	Псевдоним типа данных `VARCHAR[]`.
`char_length`	Function	Псевдоним функции `length`.
`character_length`	Function	Псевдоним функции `length`.
`chr`	Function	Returns the character corresponding to the value of the code ASCII or code Unicode, given in `argument`.
`coalesce`	Function	Returns the first value other than `NULL` from the list of argument values.
`concat`	Function	Concatenates multiple strings, arrays or binary values.

`contains`	Function	Returns `true` if the specified string `string` contains the searched substring `search_string`.
`cos`	Function	Calculates the cosine of an angle given in radians.
`count`	Function	Calculates the number of rows in the group.
`count(argument)`	Function	Calculates the number of non-empty values in `argument`.
`count_if`	Function	Returns the number of records that satisfy the condition, or `NULL` if no records satisfy the condition.
`CREATE ROLE`	SQL expression	Create a new role.
`CREATE SCHEMA`	SQL expression	Create a new schema.
`CREATE TABLE`	SQL expression	Create a new table.
`CREATE USER`	SQL expression	Create a new user.
`CREATE VIEW`	SQL expression	Create a new view.
`CREATE WORKER POOL`	SQL expression	Create a worker pool.
`cume_dist`	Function	Cumulative allocation.
`current_time`	Function	Returns the current time as a value of type `TIME`.
`DATE`	Data type	Dates.
`date_diff`	Function	Returns the number of time units between two points in time as a value of type `BIGINT`.
`date_part`	Function	Псевдоним функции `datepart`.
`date_trunc`	Function	Reduces a moment in time to the specified precision.
`datepart`	Function	Returns the specified part of the date or time value as a value of type `BIGINT`.
`DECIMAL`	Data type	Псевдоним типа данных `NUMERIC`.
`dense_rank`	Function	The rank of the current string within the group without skips.
`DESCRIBE TABLE`	SQL expression	Display information about the table.
`DESCRIBE USER`	SQL expression	Display information about the user.
`divide`	Function	Returns the result of division as an integer.
`DOUBLE`	Data type	Real numbers with variable precision.
`DROP ROLE`	SQL expression	Reset a role.
`DROP SCHEMA`	SQL expression	Deleting a schema.
`DROP TABLE`	SQL expression	Deleting a table.
`DROP USER`	SQL expression	Reset user.
`DROP VIEW`	SQL expression	Deleting a view.
`DROP WORKER POOL`	SQL expression	Reset a worker pool.
`even`	Function	Rounds to the nearest even number away from zero.
`exp`	Function	Calculates the exponent of a number.

`first_value`	Function	Returns the value calculated using the specified expression for the first row of the group.
`FLOAT`	Data type	Псевдоним типа данных `DOUBLE`.
`floor`	Function	rounds a number to the smaller side.
`fmod`	Function	Returns the remainder of dividing the first argument by the second argument.
`FROM`	SQL expression	Элемент запроса `SELECT`.
`from_json`	Function	Псевдоним функции `json_transform`.
`from_json_strict`	Function	Псевдоним функции `json_transform_strict`.
`gcd`	Function	Calculates the greatest common divisor of two numbers.
`generate_series`	Function	Generates a list of values in the range between `start` and `stop`.
`GEOMETRY`	Data type	Geospatial data.
`get_current_time`	Function	Псевдоним функции `current_time`.
`GRANT`	SQL expression	Grant privileges.
`greatest`	Function	Returns the largest number specified in the arguments.
`greatest_common_divisor`	Function	Псевдоним функции `gcd`.
`GROUP BY`	SQL expression	Элемент запроса `SELECT`.
`hash`	Function	Returns a hash of the data from `argument` as a number.
`HAVING`	SQL expression	Элемент запроса `SELECT`.
`ILIKE`	SQL expression	Элемент запроса `SELECT`.
`INSERT`	SQL expression	Add data to a table.
`INTEGER`	Data type	Псевдоним типа данных `BIGINT`.
`INTEGER[]`	Data type	Псевдоним типа данных `BIGINT[]`.
`isfinite`	Function	Checks whether a number is finite.
`isinf`	Function	Checks whether a number is infinite.
`isnan`	Function	Checks if the argument has the value `NaN` (*Not a Number*).
`JOIN`	SQL expression	Элемент запроса `SELECT`.
`JSON`	Data type	Data in JSON.
`json`	Function	Shortens the JSON structure record (removes spaces and line breaks).
`json_array_length`	Function	Returns the number of elements in the array at the specified path in JSON, or `0` if the specified path is not an array.
`json_contains`	Function	Checks if the JSON structure contains the JSON substructure specified in the second argument.
`json_exists`	Function	Checks if the JSON structure contains the specified path.
`json_extract`	Function	Extracts data from a JSON structure at the specified path. Returns the data as `JSON`.

`json_extract_path`	Function	Псевдоним функции `json_extract`.
`json_extract_string`	Function	Extracts data from a JSON structure at the specified path. Returns the data in `VARCHAR` form.
`json_extract_string_path`	Function	Псевдоним функции `json_extract_string`.
`json_group_array`	Function	Returns a JSON list containing all the values of a column.
`json_group_object`	Function	Returns a JSON structure containing all `key-value` pairs from the columns specified in the arguments.
`json_keys`	Function	Returns all keys from the specified JSON structure as `VARCHAR[]`.
`json_transform`	Function	Transforms the JSON structure according to the specified structure.
`json_transform_strict`	Function	Transforms a JSON structure to match the specified structure. Issues an error if structures or types do not match.
`json_valid`	Function	Checks if the argument is a valid JSON structure.
`json_value`	Function	Retrieves values from a JSON structure at the specified path.
`lag`	Function	Returns the value calculated for the string shifted by `offset` rows from the current to the beginning of the group.
`last_value`	Function	Returns the value calculated by the specified expression for the last row of the group.
`lcase`	Function	Псевдоним функции `lower`.
`lcm`	Function	Calculates the least common multiple of two numbers.
`lead`	Function	Returns the value calculated for the row shifted by `offset` rows from the current row to the end of the group.
`least`	Function	Returns the smallest number specified in the arguments.
`least_common_multiple`	Function	Псевдоним функции `lcm`.
`length`	Function	Returns the number of characters in a string.
`lgamma`	Function	Calculates the logarithm of the gamma function.
`LIKE`	SQL expression	Элемент запроса `SELECT`.
`LIMIT`	SQL expression	Элемент запроса `SELECT`.
`list`	Function	Псевдоним функции `array_agg`.
`ln`	Function	Calculates the natural logarithm of a number.
`log`	Function	Calculates the logarithm of a number on base `10`.
`log10`	Function	Псевдоним функции `log`.
`log2`	Function	Calculates the logarithm of a number on base `2`.
`lower`	Function	Converts a string to lower case.
`ltrim`	Function	Removes all occurrences of any of the specified characters at the beginning of a string.
`max`	Function	Returns the maximum value available in `argument`.

md5	Function	Returns a hash MD5 of data from argument as a string (VARCHAR).
mean	Function	Псевдоним функции avg.
median	Function	Returns the median value of all non-empty values in argument.
min	Function	Returns the minimum value present in argument.
multiply	Function	Multiplies two numbers.
nextafter	Function	Returns the next value with variable precision (of type DOUBLE) after the first number towards the second number.
now	Function	Returns the current time as a value of type TIMESTAMPTZ.
nth_value	Function	Returns the value calculated for the nth row within a group (counting from 1).
ntile	Function	Splits each group into num subgroups of equal (as large as possible) size and returns the subgroup number.
NUMERIC	Data type	Real numbers with specified precision.
OFFSET	SQL expression	Элемент запроса SELECT.
ORDER BY	SQL expression	Элемент запроса SELECT.
percent_rank	Function	Calculates the relative rank of the current row within a group.
pi	Function	Returns the value of the number π.
pow	Function	Exposes the first argument to the degree given by the second argument.
power	Function	Псевдоним функции pow.
QUALIFY	SQL expression	Элемент запроса SELECT.
radians	Function	converts degrees to radians.
random	Function	Returns an arbitrary number (of type DOUBLE) between 0 and 1.
rank	Function	Returns the rank (with skips) of the current row within a group.
rank_dense	Function	Псевдоним функции dense_rank.
read_json	Function	reads a .json file and writes the read data to a table.
read_json_auto	Function	Псевдоним функции read_json.
regexp_extract	Function	Extracts a substring from a string using the given regular expression.
regexp_extract_all	Function	Extracts all non-overlapping substrings from a string using the given regular expression. Returns an array of substrings.
regexp_full_match	Function	Checks whether a regular expression overlaps a string completely.
regexp_matches	Function	Checks if a regular expression is contained within a string.
regexp_replace	Function	Replaces a substring covered by a regular expression with the specified string.
regexp_split_to_array	Function	Splits a string into parts, separated by a regular expression, and returns the parts as an array.

regexp_split_to_table	Function	Splits a string into parts separated by a regular expression, and returns the parts as strings.
REVOKE	SQL expression	Revoke privileges.
round	Function	`rounds the number from the first argument to the precision specified in the second argument.
round_even	Function	Round the number from the first argument to the nearest even number with the precision specified in the second argument.
roundbankers	Function	Псевдоним функции round_even.
row_number	Function	Returns the number of the current row in its group (counting from 1).
rtrim	Function	Removes all occurrences of any of the specified characters at the end of a string.
SELECT	SQL expression	Запрос SELECT.
setseed	Function	Fixes the initial value for the random() function.
sha1	Function	Returns a hash SHA-1 of the data from argument as a string (VARCHAR).
sha256	Function	Returns a hash SHA-256 of the data from argument as a string (VARCHAR).
SHOW ROLES	SQL expression	Display a list of all roles.
SHOW TABLES	SQL expression	Display a list of all tables.
SHOW USERS	SQL expression	Display a list of all users.
SHOW WORKER POOL	SQL expression	Display the current worker pool.
SHOW WORKER POOLS	SQL expression	Display a list of all worker pools.
sign	Function	Returns -1, 1 or 0 depending on the sign of the argument.
signbit	Function	Determines whether the sign bit of a real number is set.
SIMILAR TO	SQL expression	Элемент запроса SELECT.
sin	Function	Calculates the sine of an angle given in radians.
split	Function	Splits a string into two parts by the given separator.
sqrt	Function	Calculates the square root.
ST_Distance	Function	Calculates the distance between two points in the plane.
ST_Distance_Sphere	Function	Calculates the distance between two points on the sphere.
ST_Point	Function	Creates a point of type GEOMETRY.
str_split	Function	Псевдоним функции split.
STRING	Data type	Псевдоним типа данных VARCHAR.
STRING[]	Data type	Псевдоним типа данных VARCHAR[].
string_split	Function	Псевдоним функции split.
string_split_regex	Function	Псевдоним функции regexp_split_to_array.

`string_to_array`	Function	Псевдоним функции `split`.
`strlen`	Function	Returns the number of bytes in the string.
`strptime`	Function	Converts text to a point in time using the specified format.
`subtract`	Function	Subtracts the second argument from the first argument.
`sum`	Function	Calculates the sum of all non-empty values in `argument`.
`TEXT`	Data type	Псевдоним типа данных `VARCHAR`.
`TEXT[]`	Data type	Псевдоним типа данных `VARCHAR[]`.
`TIME`	Data type	Time.
`TIMESTAMP`	Data type	Time stamps.
`TIMESTAMPTZ`	Data type	Time stamps with time zone.
`tngri.sql`	Python function	Executes the specified query SQL inside a cell of type Python.
`tngri.upload_df`	Python function	Uploads data from DataFrame to Tengri.
`tngri.upload_file`	Python function	Uploads data from a file to Tengri.
`tngri.upload_s3`	Python function	Uploads a file from the specified bucket S3 to Tengri.
`trim`	Function	Removes all occurrences of any of the specified characters on both sides of the string.
`trunc`	Function	Discards all characters after the decimal separator.
`ucase`	Function	Псевдоним функции `upper`.
`UNION`	SQL expression	Элемент запроса `SELECT`.
`unnest`	Function	Expands lists or structures from `argument` into a set of distinct values.
`upper`	Function	Converts a string to uppercase.
`USE ROLE`	SQL expression	Set the active role.
`USE WORKER POOL`	SQL expression	Set the current compute pool.
`VARCHAR`	Data type	Text strings.
`VARCHAR[]`	Data type	Arrays of text strings.
`WHERE`	SQL expression	Элемент запроса `SELECT`.
`WITH`	SQL expression	Элемент запроса `SELECT`.
`%`	Operator	Returns the remainder of the left argument divided by the right argument.
`*`	Operator	Multiplies the arguments.
`+`	Operator	Adds the right argument to the left argument.
`<<functions-numeric::minus_operator,->>`	Operator	Subtracts the right argument from the left argument.
`/`	Operator	Divides the left argument by the right argument.
`^`	Operator	Elevates the left argument to the degree given by the right argument.

||	Operator	Concatenates multiple strings, arrays or binary values.
~	Operator	Checks if the regular expression covers the string completely.

Objects total: 215