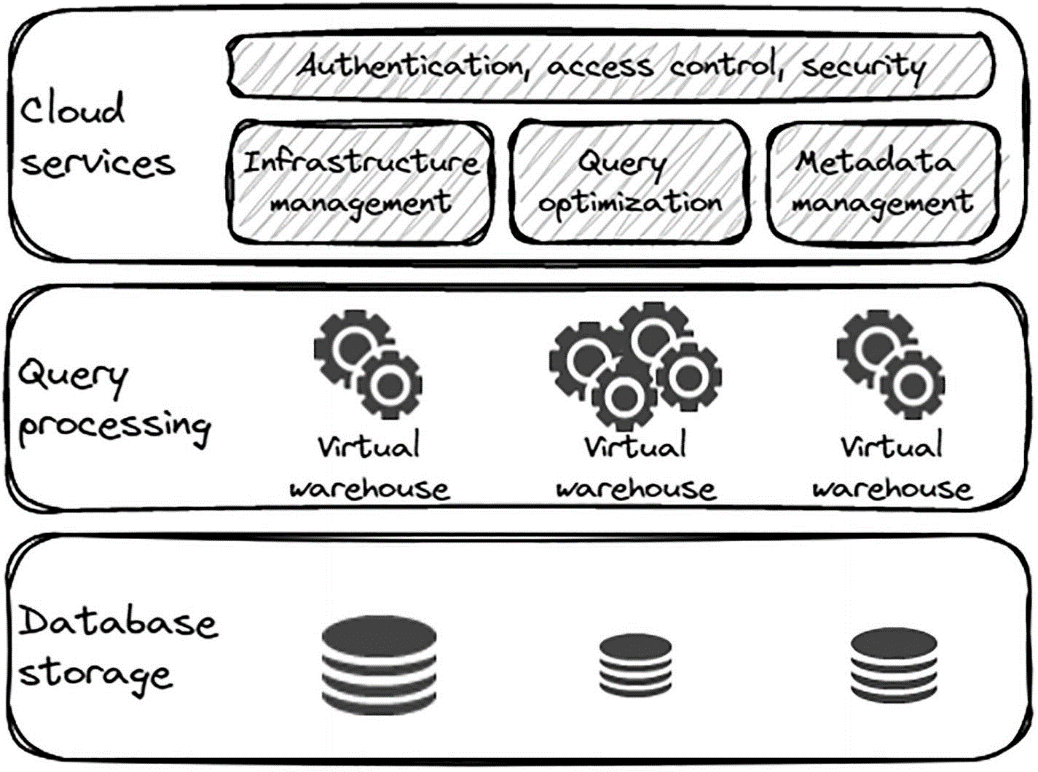
**[Chapter 3](https://learning.oreilly.com/library/view/aws-certified-cloud/9781801075930/B17124_03_Final_SK_ePub.xhtml" \l "_idTextAnchor050): Snowflake Architecture**

In this chapter we wil see layers: Database storage, Query processing, Cloud services. This equals superior flexibility and cost effectiveness.

## Architecture Layers

Snowflake stores data in a central data repository, similar to traditional shared-disk architectures. A shared-disk architecture (SD) is a distributed computing architecture in which the nodes share same disk devices but each node has its own private memory. Queries are processed using MPP (massively parallel processing) compute clusters, where each node in the cluster stores a portion of the data locally, similar to shared-nothing architectures. This combination offers the data management simplicity of a shared-disk architecture, with better performance and scalability of a shared-nothing architecture.

A shared-nothing architecture (SN) is a [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing" \o "Distributed computing) [architecture](https://en.wikipedia.org/wiki/Software_architecture" \o "Software architecture) in which each update request is satisfied by a single node (processor/memory/storage unit) in a [computer cluster](https://en.wikipedia.org/wiki/Computer_cluster" \o "Computer cluster). The intent is to eliminate contention among nodes. Nodes do not share (independently access) the same memory or storage.

One alternative architecture is shared everything, in which requests are satisfied by arbitrary combinations of nodes. This may introduce contention, as multiple nodes may seek to update the same data at the same time. It also contrasts with [shared-disk](https://en.wikipedia.org/wiki/Shared-disk_architecture" \o "Shared-disk architecture) and [shared-memory](https://en.wikipedia.org/wiki/Shared-memory_architecture" \o ") architectures.

SN eliminates [single points of failure](https://en.wikipedia.org/wiki/Single_point_of_failure), allowing the overall system to continue operating despite failures in individual nodes and allowing individual nodes to upgrade hardware or software without a system-wide shutdown.[[1]](https://en.wikipedia.org/wiki/Shared-nothing_architecture#cite_note-1)

A SN system can scale simply by adding nodes, since no central resource bottlenecks the system.[[2]](https://en.wikipedia.org/wiki/Shared-nothing_architecture#cite_note-2) In databases, a term for the part of a database on a single node is a [shard](https://en.wikipedia.org/wiki/Shard_(database_architecture)" \o "Shard (database architecture)). A SN system typically partitions its data among many nodes. A refinement is to replicate commonly used but infrequently modified data across many nodes, allowing more requests to be resolved on a single node.

Database Storage: Data is stored in the object storage of the cloud provider (AWS, Azure, or GCP), where the Snowflake account is hosted. When data is loaded into Snowflake—whether it is structured, semi-structured, or unstructured—it is reorganized into Snowflake’s internal optimized, compressed, columnar format.

## Query Processing: This is where queries are processed. Snowflake processes queries using *virtual warehouses, which* is an MPP (massively parallel processing) compute cluster composed of multiple compute nodes allocated by Snowflake from a cloud provider.

Each virtual warehouse is an independent compute cluster that does not share compute resources (has no impact on the performance) with other virtual warehouses. When more users are added, more virtual warehouses can be provisioned without degrading overall performance.

## Cloud Services: This is a collection of services that coordinate all activities across Snowflake, such as handling user logins, processing user requests, dispatching queries, and taking care of security.

* Authentication
* Infrastructure management
* Metadata management
* Query parsing and optimization
* Access control

## Snowflake Objects: All data in Snowflake is maintained in databases. Each database consists of one or more schemas, which are logical groupings of database objects, such as tables and views as well as other types of objects including functions, stored procedures, tasks, streams, sequences, and more.

Metadata about all objects in each database is stored in a dedicated schema named INFORMATION\_SCHEMA that Snowflake automatically creates in every database in an account. This schema is maintained by Snowflake and is available as read-only to the user.

* Views for all objects contained in the database
* Views for non-database objects such as roles and warehouses
* Table functions for historical usage data

Only objects for which the current role has been granted access privileges are returned.

## Tables: Snowflake supports defining permanent, temporary, or transient tables. Each of these table types is suitable for a particular use case, primarily based on how long the data must be preserved.

## Permanent Tables: This is the default table type in Snowflake. These tables remain persistent after the user session ends and are accessible to other users depending on granted privileges. The data stored in permanent tables is covered by Snowflake’s storage protection and is available for time travel and is failsafe. Snowflake supports the CREATE OR REPLACE syntax in most statements, including in the CREATE TABLE statement, allowing you to re-create any objects without having to drop them first.

**Note**: Although Snowflake supports the standard SQL syntax for creating primary keys, foreign keys, and unique constraints, these constraints are not enforced when data is inserted into tables. Only NOT NULL constraints are enforced.

## Temporary Tables (contribute to storage costs): This is to store non-permanent data that is used only in the current session. Once the session ends, data stored in temporary tables is gone for good and is not recoverable by Snowflake data protection features. Temporary tables are not visible to other users or sessions.

## Snowflake allows you to create a temporary table with the same name as an existing permanent table. The query will use the temporary table rather than the permanent table.

## Transient Tables: The key difference between transient and permanent tables is that transient tables don’t have a failsafe period. The key difference between transient and temporary tables is that data in transient tables is persisted after the user session ends and is accessible by other users depending on granted privileges. are used for non-permanent data that doesn’t need the same level of data protection and recovery as permanent tables. Transient tables are used for non-permanent data that doesn’t need the same level of data protection and recovery as permanent tables. A use case for transient tables is creating intermediate tables that are populated by ETL processes that are truncated and repopulated during each execution of the ETL process.

**Note:** In addition to transient tables, Snowflake also supports creating transient databases and transient schemas. All tables created in a transient schema, as well as all schemas created in a transient database, are transient. Once you have created a transient table, you will not be able to modify it to a permanent table and vice versa.

SHOW TABLES command is usually more efficient than querying INFORMATION\_SCHEMA because it doesn’t require a running warehouse to execute, and it limits the results to the current schema

Comparison of Table Types:

|  |  |  |  |
| --- | --- | --- | --- |
| **TABLE TYPE** | **PERSISTENCE** | **TIME TRAVEL** | **FAILSAFE** |
| Temporary | Remainder of session | 0 or 1 days | 0 days |
| Transient | Until dropped | 0 or 1 days | 0 days |
| Permanent (Std Ed.) | Until dropped | 0 or 1 days | 7 days |
| Permanent (<= Ent Ed.) | Until dropped | 0 or 90 days | 7 days |

## Views: A view is basically a named definition of a query that allows the result of the query to be used as if it were a table. The results of a view are created by executing the query at the time that the view is referenced in another query.

Views serve many purposes, such as protecting data by exposing only a subset of the data in underlying tables or simplifying complex queries by hiding part of the query logic behind a view that is then used in another query.

**Caution:** In many databases, when you create a view and don't specify the schema, the database will use the current schema. This behavior differs from Snowflake, so be sure you understand this concept thoroughly.

Views in Snowflake are read-only, which means that you cannot execute DML commands such as update a view.

## Data Types

The main data type groups in Snowflake are:

* Numeric data types
* String and binary data types
* Logical data types
* Date and time data types
* Semi-structured data types
* Geospatial data types

Data type conversion is done implicitly by Snowflake whenever possible. Users can also explicitly cast a value from one data type to another data type using any of the following functions or operators:

* The CAST function; for example CAST (amount AS INTEGER)
* The :: operator; for example amount::INTEGER
* The corresponding SQL function, if available; for example TO\_DATE('2022-07-04')

## Numeric Data Types

Precision is the number of digits in a number. Scale is the number of digits to the right of the decimal point in a number.

NUMBER(Numeric precision, Numeric scale).

Numeric precision refers to the maximum number of digits that are present in the number.

ie 1234567.89 has a precision of 9

Numeric scale refers to the maximum number of decimal places

ie 123456.789 has a scale of 3

Thus the maximum allowed value for decimal(5,2) is 999.99

NUMBER; it can store up to 38 digits, with an optional precision (the total number of digits) and scale (the number of digits to the right of the decimal point), for example NUMBER(18, 2). Synonyms for NUMBER are DECIMAL and NUMERIC.

To define integer data types, NUMBER(38, 0) can be used, which indicates zero digits to the right of the decimal point. Synonyms for integer data types include INT, INTEGER, BIGINT, SMALLINT, and others, where you don’t specify the precision and scale.

Floating point numbers can be defined as FLOAT, FLOAT4, FLOAT8, DOUBLE, REAL, and others. All of these are represented as 64-bit floating-point double precision numbers in Snowflake.

## String and Binary Data Types

Snowflake supports data type VARCHAR that stores Unicode UTF-8 characters for character strings. The VARCHAR keyword accepts an optional parameter, which represents the maximum number of characters that will be stored, for example VARCHAR(255). If you don’t specify the parameter, the maximum allowed length 16,777,216 will be used.

A VARCHAR is also limited to a maximum number of bytes, which is 16MB.

**Note:** Remember the 16MB limit. This is not only the maximum allowed length of a character string, but also the size of a Snowflake micro-partition, which limits the size of all objects and values that can be stored as a single unit in Snowflake.

Snowflake supports the BINARY data type for binary strings. The maximum length of a binary string is 8MB, regardless of the encoding. The size limit is set at 8MB so that the variable fits within 16MB when converted to a hexadecimal string.

## Logical Data Types

## Snowflake supports BOOLEAN as a logical data type with TRUE, FALSE, or NULL values. The NULL value represents an unknown value just like in other data types.

* Strings that represent TRUE values: 'true', 't', 'yes', 'y', 'on', '1' (case-insensitive)
* Strings that represent FALSE values: 'false', 'f', 'no', 'n', 'off', '0' (case-insensitive)
* Numbers that represent FALSE values: 0 (zero)
* Numbers that represent TRUE values: Any non-zero value

## Date and Time Data Types

The DATE data type is used for storing dates with no time parts. It recognizes dates in the most common formats, such as YYYY-MM-DD, DD-MON-YYYY, and more.

Snowflake supports the following data types for storing timestamps:

* TIMESTAMP\_LTZ. Internally, the value is stored as a UTC timestamp, but all operations on the value are performed using the current session time zone defined by the TIMEZONE parameter.
* TIMESTAMP\_NTZ or DATETIME. Stores the timestamp value without taking a time zone into consideration. Sometimes this is also referred to as “wallclock” time.
* TIMESTAMP\_TZ. Stores the timestamp together with a time zone offset. When a time zone is not provided, the value of the current session TIMEZONE parameter is used.

Due to limitations of the Gregorian calendar, Snowflake recommends that you avoid using years prior to 1582 in timestamp values.

The TIME data type is used for storing time values formatted as HH:MI:SS. Fractional seconds are also supported by adding an optional precision parameter, for example TIME(9), which means nanoseconds and is also the default.

## Semi-Structured Data Types

* VARIANT. Can contain any data type, including OBJECT and ARRAY. The maximum length of a VARIANT value is 16MB.
* ARRAY. Contains zero or more elements, just like arrays used in programming languages. Each element is addressed by its position in the array.
* OBJECT. Contains key/value pairs where keys are stored as VARCHAR data types, and values are stored as VARIANT data types.

A combination of VARIANT, ARRAY, and OBJECT data types enable you to represent data in semi-structured formats, such as JSON, Avro, ORC, Parquet, and XML.

## Geospatial Data Types

Snowflake provides geospatial data types including points, lines, and polygons on the Earth’s surface. The following geospatial data types are available:

* GEOGRAPHY. Uses the WGS 84 standard, where points on the Earth are represented as degrees of longitude from -180 degrees to +180 degrees and latitude from -90 to +90. Altitude is currently not supported.
* GEOMETRY. Coordinates are represented as pairs of real numbers with units that are determined by the spatial reference system (SRS).

## User-Defined Functions

A key difference between system-defined functions and UDFs is that UDFs are database objects, which means that they reside in a database and schema. When you call a UDF without fully qualifying the name, Snowflake will look for the UDF in the database and schema that is used in the current session. A system-defined function is never fully qualified and can be called from any database or schema.

A UDF can be written in one of these supported programming languages:

* SQL
* JavaScript
* Java
* Python

A UDF can return either scalar or tabular results. The properties of each are:

* Scalar UDF. A scalar function returns a single column or value. When you use a scalar function in a query, it results in one output row for each input row.
* Tabular UDF. A tabular function, also called a UDTF (user-defined table function), returns zero, one, or many rows for each input row. In a tabular UDF, the return statement contains the TABLE keyword. You must also specify the names and data types of the columns that the function will return.

UDF names may be overloaded, meaning that you can have more than one UDF with the same name, but different arguments, either by the number of arguments or the argument data types. When calling an overloaded UDF, the correct function is determined by the arguments that were provided in the function call.

Another type of a UDF is one that returns the result of a SQL query where the query expression returns at most one row.

The query expression in a UDF can reference database objects such as tables, views, and sequences. If an object in the expression is unqualified, it will be resolved in the schema that contains the UDF. In case the expression references an object in a different schema, the owner of the function must have sufficient privileges to access the object. Similarly as in views, the user calling the UDF does not need access to the objects referenced in the query expression within the UDF, but only needs the privilege to execute the UDF.

## Stored Procedures

Stored procedures allow you to write procedural code that executes SQL statements as well as other programming constructs. Some of the benefits of using stored procedures include:

* Writing procedural logic such as looping and conditional statements that SQL queries don’t support.
* Including error handling logic.
* Creating and executing SQL statements that have been constructed dynamically.
* Writing a sequence of multiple SQL statements that are performed as a single unit of code.
* Calling a function or another stored procedure for additional procedural logic.

You can write a stored procedure in one of the following languages:

* Snowflake Scripting
* JavaScript
* Java (using Snowpark)
* Scala (using Snowpark)
* Python (using Snowpark)

Stored procedures can be written so that the code in the procedure is executed either with the privileges of the role that owns the procedure or with the privileges of the role that calls the procedure. We refer to such stored procedures as either owner’s rights stored procedures or caller’s rights stored procedures. The properties of each type of stored procedure are as follows:

* **Caller’s rights stored procedure.** This type of stored procedure executes with the privileges of the role that calls the procedure. This means that the procedure caller must have sufficient privileges to execute all statements in the stored procedure. The procedure can also access the caller’s current session and use session variables if needed.
* **Owner’s rights stored procedure.** This type of stored procedure executes with the privileges of the role that owns the procedure. The reason that you might write a procedure with owner’s rights is so that the owner can enable the caller to execute specific tasks that require additional privileges without granting those privileges. In this case, the role that owns the stored procedure must have sufficient privileges to execute all statements in the stored procedure.

A stored procedure is designated as a caller’s rights stored procedure by specifying the EXECUTE AS CALLER keywords when creating the procedure. Alternatively, the EXECUTE AS OWNER keywords will designate a stored procedure as an owner’s rights stored procedure.

## Comparing UDFs and Stored Procedures

|  |  |  |
| --- | --- | --- |
| **COMPARISON ATTR** | **UDF** | **STORED PROC** |
| Purpose | To calculate and return a value. | To execute SQL statements. |
| Return value | Always returns a value explicitly by specifying an expression. | May return a value, such as an error indicator, but is not required to do so. |
| Use | Used in queries where a general expression can be used. | Called as an independent statement. |
| Execution statement | select MY\_FUNCTION(); | call MY\_PROCEDURE(); |

When to create a UDF:

* When you are migrating a function from an existing application.
* When you need a function that can be called as part of a SQL statement.
* When you require a single output value for every input row of a SQL statement.

When to create a stored procedure:

* When you are migrating a stored procedure from an existing application.
* When you want to execute SQL statements such as SELECT, UPDATE, DELETE, and so on.
* When you want to execute DDL statements to create or modify database objects.
* When you want to execute any other types of statements such as creating users, creating and granting roles, and so on.

## Streams

Streams are Snowflake’s way to support CDC (change data capture). A stream can be created on the following types of tables:

* Standard tables
* External tables
* Directory tables
* Underlying tables for a view (in case we have a stream on a view)

When a stream is created on a table, the snapshot of the table at that point in time represents the initial state of the stream. Each subsequent SQL statement that inserts, updates, or deletes data in the table creates a new version of the table. The stream points to each version of the table at a given point in time.

**Note:** Streams don’t contain any table data. When a stream is created on a table, hidden columns are added to the table that store change tracking metadata.

 There are three metadata columns:

* METADATA$ACTION. Contains the name of the DML operation that was executed (INSERT or DELETE). When an UPDATE operation is executed on the table, the stream records the change as a DELETE of the previous version of the data before it was updated and an INSERT with the new version of the data.
* METADATA$ISUPDATE. Contains a Boolean TRUE or FALSE flag, which indicates whether the operation was part of an UPDATE statement.
* METADATA$ROW\_ID. Contains the unique ID of the row.

Streams record only the difference between two versions of a table. If more than one DML operation is executed on a row, for example when a row is added and then updated, the stream will record only the difference from the previous version and the current version, which is a new row.

## Consuming Streams

To consume the contents of a stream after it has been accumulating changed data for a while, a statement that selects from the stream and writes to a target table. Just querying a stream using a SELECT statement does not consume it.

You may require more than one statement to consume a stream; for example you could write a part of the data from a stream to one target table and another part of the stream to another target table. In such situations, you must surround the INSERT statements with an explicit transaction, starting with BEGIN and ending with COMMIT.

## Types of Streams

* **Standard**. Can be created on tables, directory tables, or views. A standard stream tracks all changes to the source object, including inserts, updates, and deletes.
* **Append-only**. Can be created on standard tables, directory tables, or views. An append-only stream tracks inserts only; it does not track updates or deletes.
* **Insert-only**. Can be created on external tables. An insert-only stream tracks inserts only; it does not track deletes, such as removing a file from the cloud storage location of the external table.

## Tasks

Tasks are user-defined objects that support scheduled execution of SQL statements. A task can be executed on a schedule that is defined when the task is created. A task can also be executed by a parent task if a task dependency has been defined. Snowflake does not support executing tasks based on events.

Tasks are often combined with streams to create ETL processes for loading data on a regular basis. Streams ensure that only changed data is loaded to the target table each time the ETL process is executed and tasks ensure that loading is performed on schedule.

A task can execute different types of SQL code, such as:

* Execute a single SQL statement
* Call a stored procedure
* Execute procedural logic using Snowflake Scripting

When tasks execute, they require compute resources.

* **User-managed**. The user manages compute resources for tasks by specifying an existing virtual warehouse when creating the task.
* **Snowflake-managed**. Snowflake determines the size of compute resources for each task run based on historical runs of the same task. The user specifies a suggested warehouse size when the task is created as an initial size that is used until sufficient history is accumulated. Snowflake-managed compute resources in tasks are also referred to as serverless.

Snowflake always executes only one instance of a scheduled task at one point in time. If a task is still running when the next scheduled time of the same task occurs, that run will be skipped.

If you do have tasks scheduled during 1 AM and 3 AM on Sundays, you should check the schedule manually on Sundays twice a year when the time changes.

Note: Make sure that the statements that you provide in the body of the task execute as expected before you create the task. This will save you troubleshooting headaches later.

Every task is suspended immediately upon creation. If you want a task to run on schedule, you must resume it before it will execute. However, if you just want to check that it is performing as expected, you can execute it manually a single time using the EXECUTE TASK command

**execute task PERIODIC\_COPY\_CUSTOMER;**

To resume a task to allow it to run on schedule, use the ALTER TASK command, like this:

**alter task PERIODIC\_COPY\_CUSTOMER resume;**

EXECUTE TASK privilege. This privilege is not enabled by default and unless it has been explicitly granted, a typical user does not have it.

One way to resolve this issue is to grant the EXECUTE TASK privilege to the role that the user has set as the current role. Another way, as recommended by Snowflake, is to create a custom role and grant the EXECUTE TASK privilege to this role.

To check the status of the task and to verify whether it is executing as expected, use the TASK\_HISTORY() table function in INFORMATION\_SCHEMA with the following statement:

**select \***

**from table(INFORMATION\_SCHEMA.TASK\_HISTORY())**

**order by scheduled\_time desc;**

The output from the TASK\_HISTORY() function shows useful information, such as the next scheduled date and time when the task will run as well as a record for each of the previous executions of the task.

**Caution:** After you have completed testing the task, don’t forget to set it to suspended or delete the task altogether. If you don’t, the task will continue executing every five minutes and will needlessly consume resources.

More frequently, you'll schedule tasks to run at a given time on given days, for example, every day at 8 PM or every Saturday at midnight and so on. If you want these types of schedules, you can use a CRON expression in the SCHEDULE parameter when you create the task.

In addition to creating and scheduling individual tasks, Snowflake supports creating DAGs (directed acyclic graphs). A DAG is tree of tasks that contains one root task and one or more child tasks in a hierarchical manner. To schedule a DAG, you only schedule the root task. You don’t schedule any child tasks because they will execute in turn after their predecessor completes. A DAG can contain a maximum of 1,000 tasks including the root task. A task can have no more than 100 parent tasks and no more than 100 child tasks.

## Sequences

Sequences are used to generate unique numbers, for example to generate values for a primary key in a table or to generate unique values such as row numbering. Sequences that are generated by Snowflake consistently increase in value (or decrease in value if the increment is negative) but are not necessarily without gaps. Values generated by a sequence will always be unique, even if the same sequence is used in different SQL statements executed at the same time.

The default starting value is 1 and the default increment value is also 1.

You can retrieve the next value of the sequence using the NEXTVAL keyword and the following command:

**select CUST\_PK\_SEQ.NEXTVAL;**

Sequences can be used in tables as default column values to generate primary or unique keys.

## Summary

This chapter described the three Snowflake architecture layers (database storage, query processing, and cloud services) and their purposes. It reviewed the main object types that are stored in Snowflake, including tables, views, user-defined functions, stored procedures, streams, tasks, and sequences. It also walked through the data types that you can use in Snowflake.