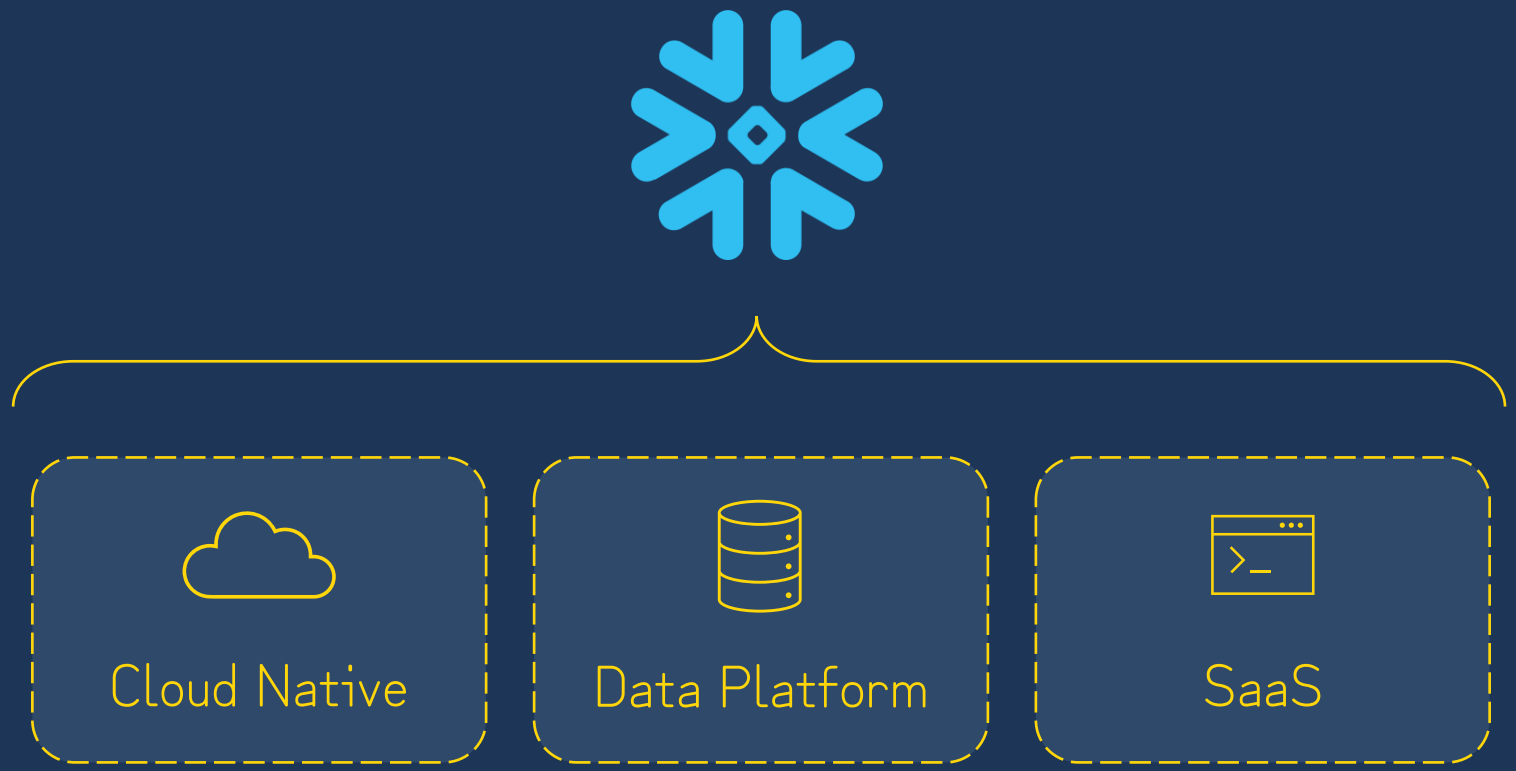
What is Snowflake?

What is Snowflake?



Data Platform

Data Warehouse

- Structured & relational data
- ANSI Standard SQL
- ACID compliant transactions
- Data stored in databases, schemas & tables

Data Lake

- Scalable storage and compute
- Schema does not need to be defined upfront
- Native processing of semi-structured data formats

Data Engineering

- COPY INTO & Snowpipe
- Separate compute clusters
- Tasks and Streams
- All data encrypted at rest and in transit.

Data Science

- Remove data management roadblocks with centralised storage
- Partner eco-system includes data science tooling:
  - Amazon SageMaker
  - DataRobot
  - Dataiku

Data Sharing

- Secure Data Sharing
- Data Marketplace
- Data Exchange
- BI with the Snowflake partner ecosystem tools

Data Applications

- Connectors and Drivers
- UDFs and Stored Procedures
- External UDFs
- Preview features such as Snowpark

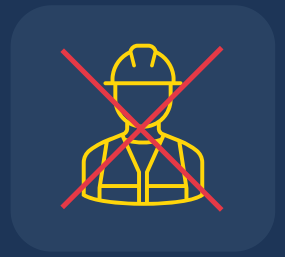
Cloud Native

Snowflake's software is purpose built for the Cloud.

All Snowflake infrastructure runs on the Cloud in either AWS, GCP or Azure.

Snowflake makes use of Cloud's elasticity, scalability, high availability, cost-efficiency & durability.

Software as a service (SaaS)



No management of hardware

Transparent updates and patches

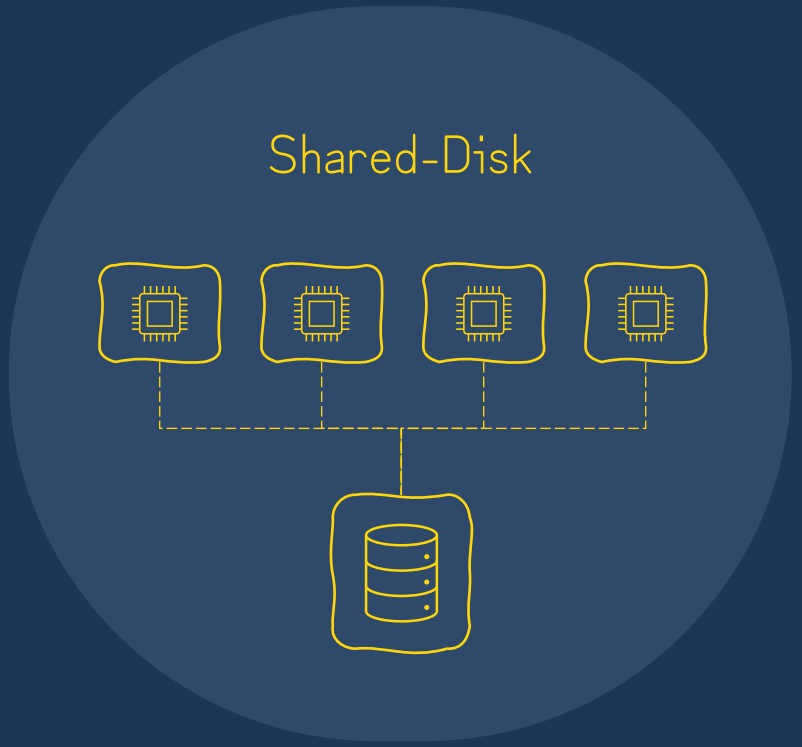
Subscription payment model

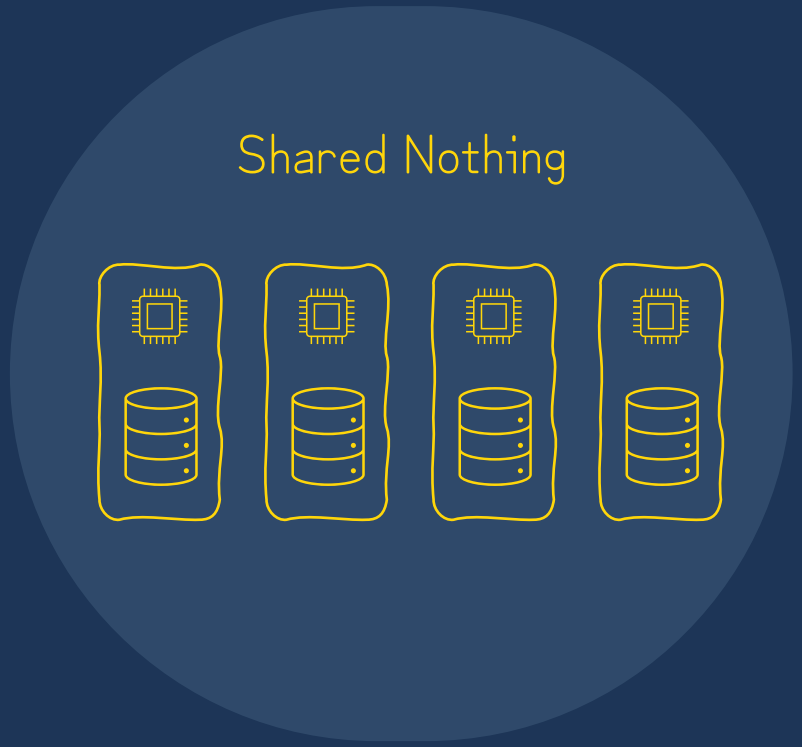
Ease of access

Automatic optimisation

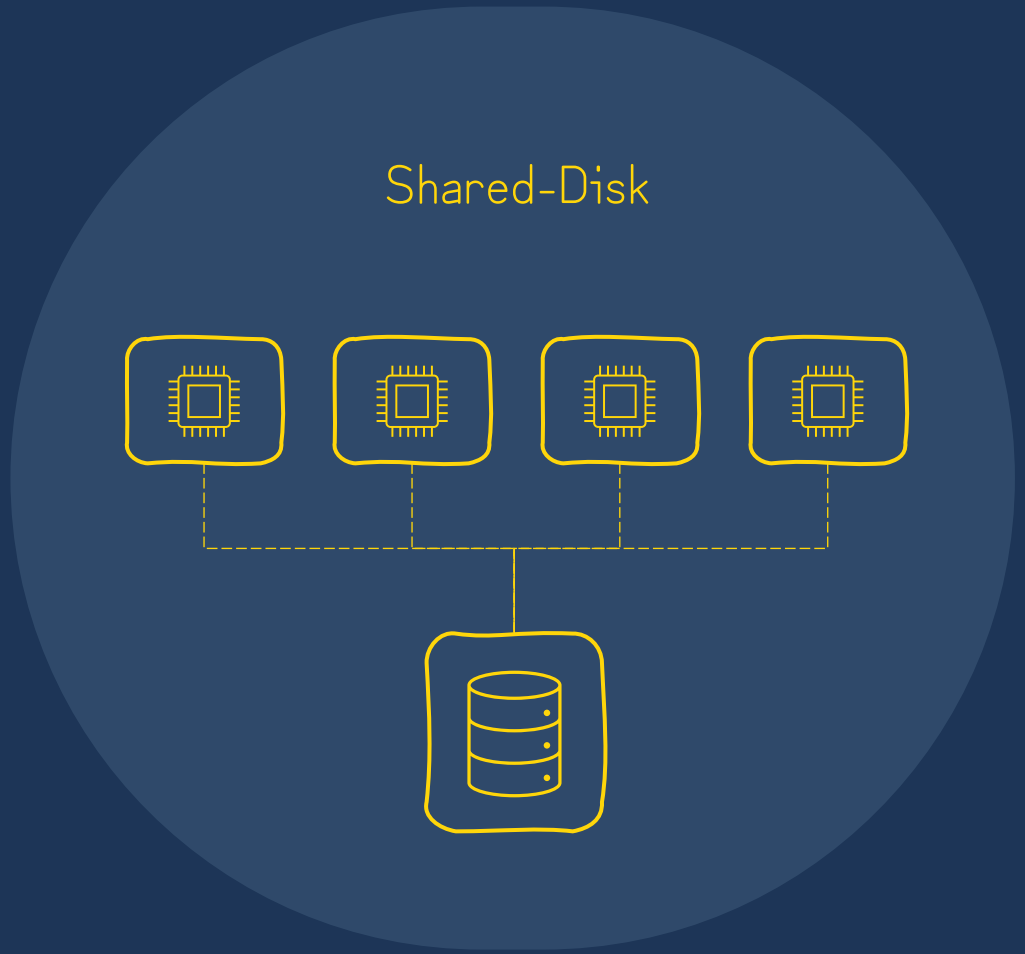
Multi-cluster Shared Data Architecture

Distributed Architectures





Shared Disk Architecture



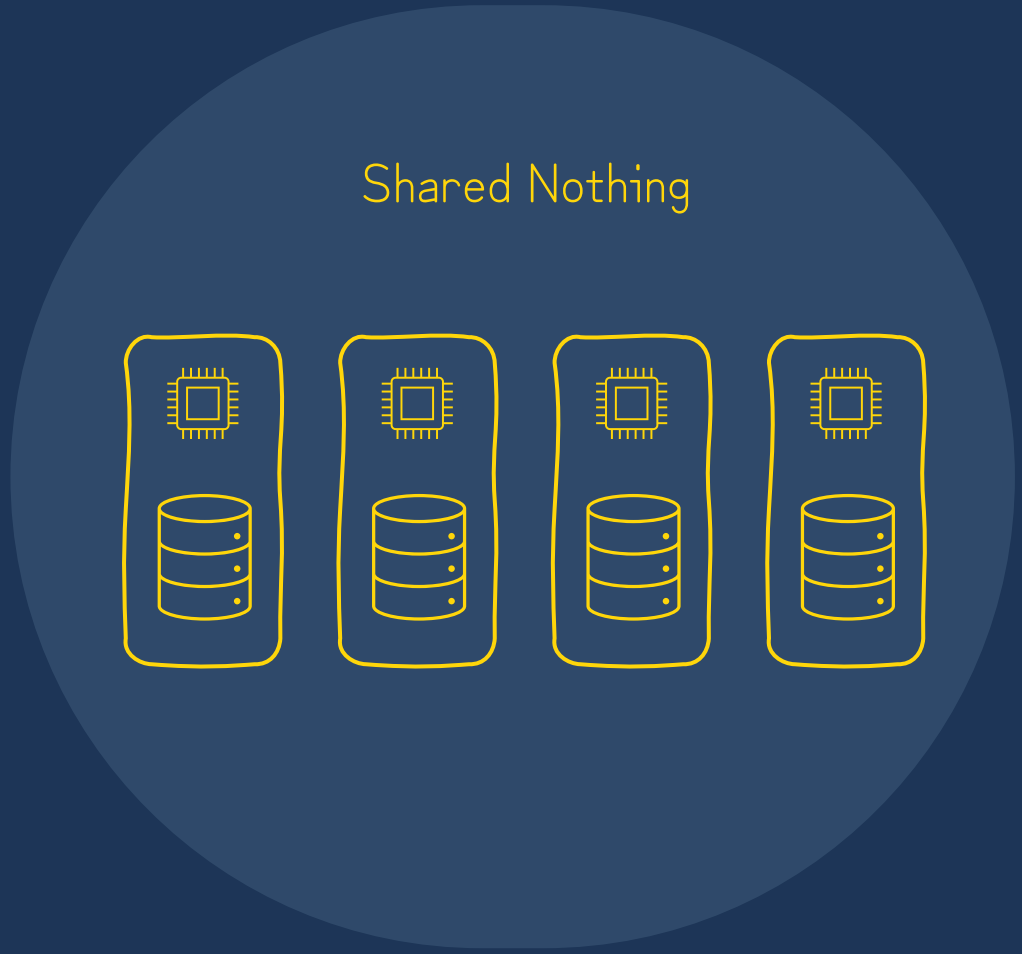
Advantages

- Relatively simple to manage
- Single source of truth

Disadvantages

- Single point of failure
- Bandwidth and network latency
- Limited scalability

Shared Nothing Architecture



Advantages

- Co-locating compute and storage avoids networking latency issues
- Generally cheaper to build & maintain
- Improved scaling over shared-disk architecture

Disadvantages

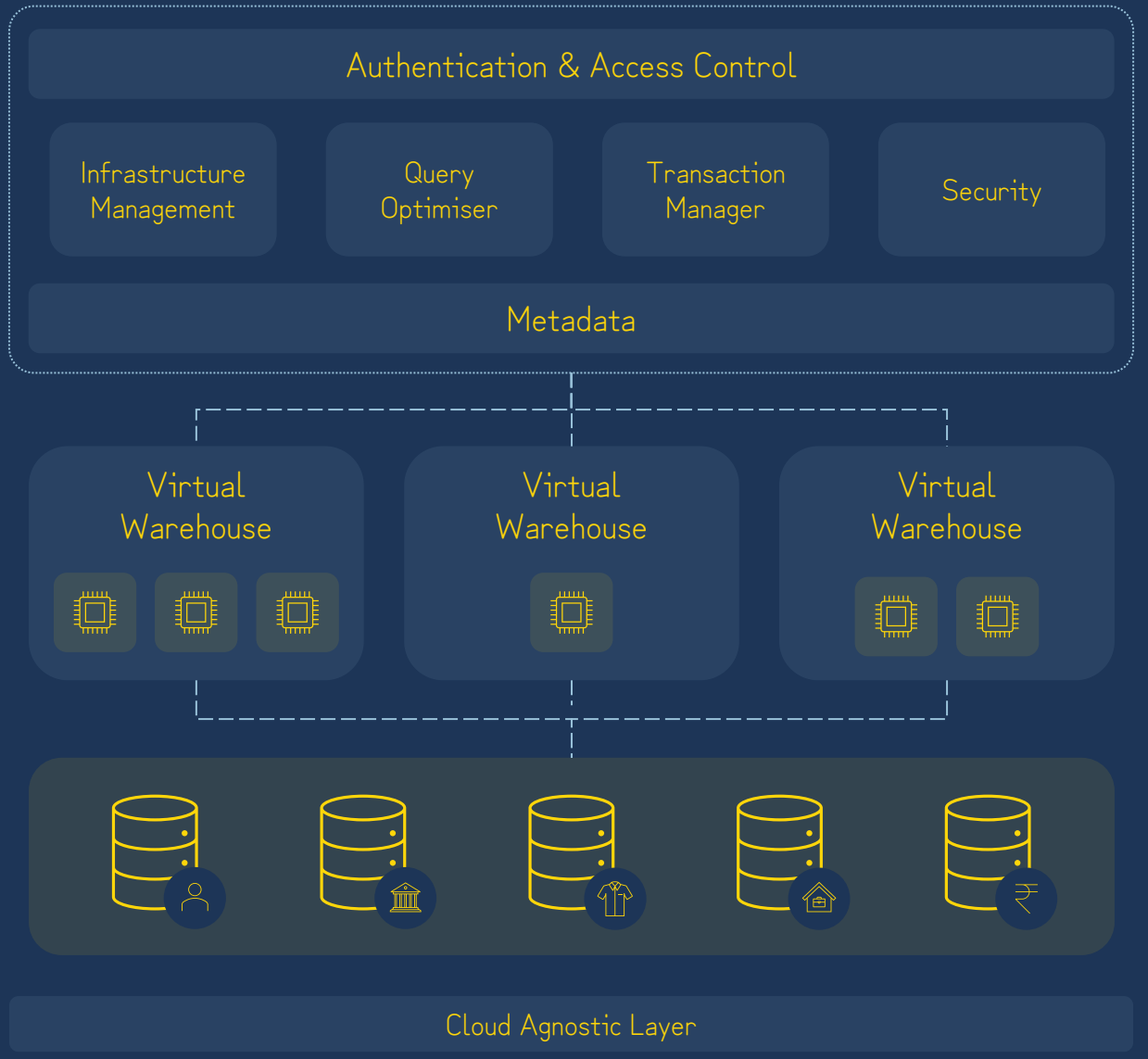
- Scaling still limited
- Storage and compute tightly coupled
- Tendency to overprovision

Multi-cluster Shared Data Architecture

Cloud Services Layer

Query Processing Layer

Data Storage Layer



- Decouple storage, compute and management services.
- Three infinitely scalable layers.
- Workload isolation with virtual warehouses.

Storage Layer

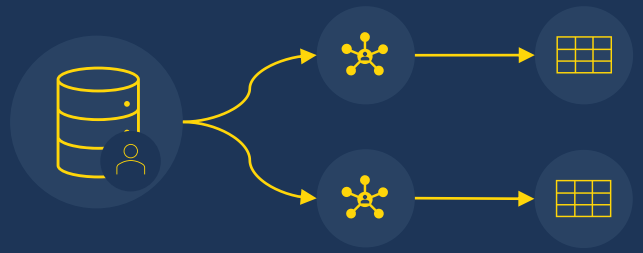
Storage Layer

Persistent and infinitely scalable cloud storage residing in cloud providers blob storage service, such as AWS S3.

Snowflake users by proxy get the availability & durability guarantees of the cloud providers blob storage.

99.999999999% durability
AWS S3

Data loaded into Snowflake is organized by databases, schemas and accessible primarily as tables.



Both structured and semi-structured data files can be loaded and stored in Snowflake.

CSV
JSON
Avro
ORC
Parquet
XML

Storage Layer

When data files are loaded or rows inserted into a table, Snowflake reorganizes the data into its proprietary compressed, columnar table file format.

The data that is loaded or inserted is also partitioned into what Snowflake call micro-partitions.

Storage is billed by how much is stored based on a flat rate per TB calculated monthly.

$42.00(TB/mo)

AWS Europe London

Data is not directly accessible in the underlying blob storage, only via SQL commands.

```
SELECT * FROM <table>;
```

Query Processing Layer

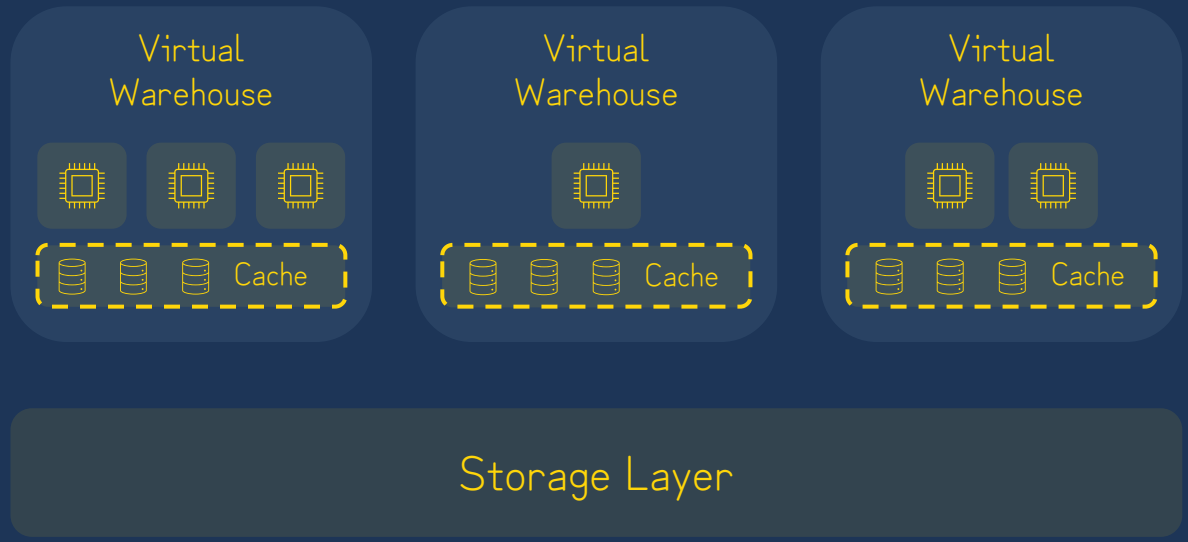
Query Processing Layer

The query processing layer consists of "Virtual Warehouses" that execute the processing tasks required to return results for most SQL statements.

A **Virtual Warehouse** is a named abstraction for a cluster of a cloud-based compute instances that Snowflake manage.

```
CREATE WAREHOUSE MY_WH WAREHOUSE_SIZE=LARGE;
```

Underlying nodes of a Virtual Warehouse cooperate in a similar way to a shared-nothing compute clusters making use of local caching.



Query Processing Layer

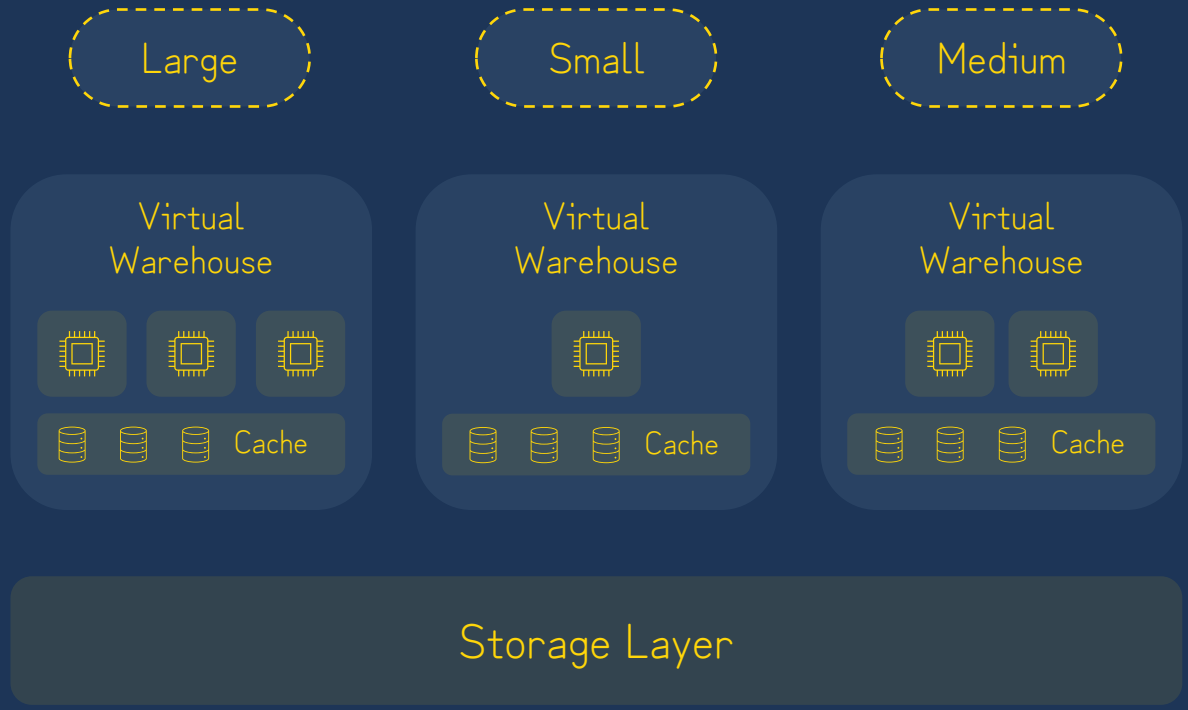
Virtual warehouses can be created or removed instantly.

Virtual warehouses can be paused or resumed.

Virtually unlimited number of virtual warehouses can be created each with it's own configuration.

Virtual warehouses come in multiple "t-shirt" sizes indicating their relative compute power.

All running virtual warehouses have consistent access to the same data in the storage layer.



Services Layer

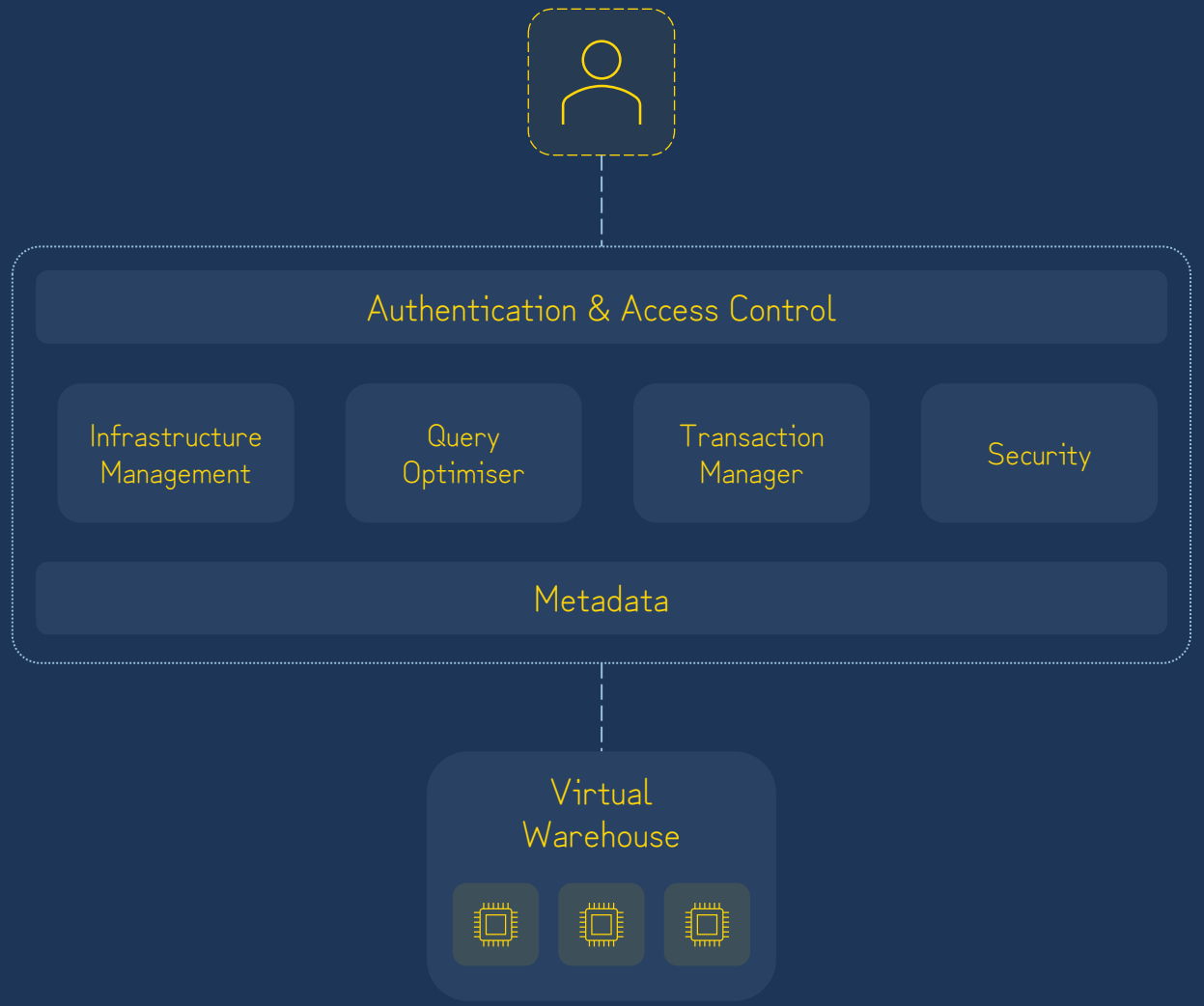
Services Layer

The services layer is a collection of highly available and scalable services that coordinate activities such as authentication and query optimization across all Snowflake accounts.

Similar to the underlying virtual warehouse resources, the services layer also runs on cloud compute instances.

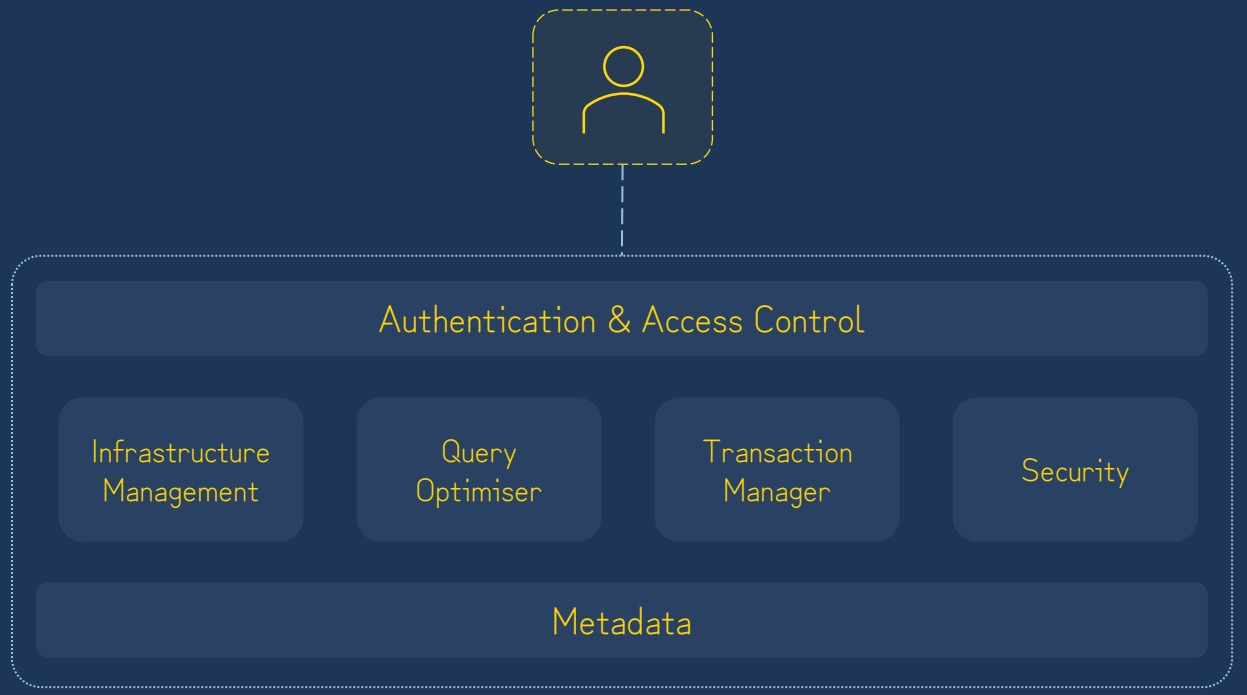
Services managed by this layer include:

- Authentication & Access Control
- Infrastructure Management
- Transaction Management
- Metadata Management
- Query parsing and optimisation
- Security



Services Layer

The services layer is a collection of highly available and scalable services that coordinate activities such as authentication and query optimization across all Snowflake accounts.



Editions & Key Features

Snowflake Editions & Key Features

Standard

Enterprise

Business Critical

Virtual Private Snowflake

SQL Support

Security, Governance, & Data Protection

Compute Resource Management

Interface & Tools

Releases

Data Import & Export

Data Replication & Failover

SQL Support

Feature	Standard	Enterprise	Business Critical	VPS
⇒ Standard SQL defined in SQL:1999	✓	✓	✓	✓
⇒ Advanced DML defined in SQL:2003	✓	✓	✓	✓
⇒ Standard data types: VARCHAR, NUMBER, TIMESTAMP etc	✓	✓	✓	✓
⇒ Semi-structured data types: VARIANT, OBJECT & ARRAY.	✓	✓	✓	✓
⇒ Multi-statement transactions	✓	✓	✓	✓

SQL Support

Feature	Standard	Enterprise	Business Critical	VPS
⇒ User Defined Functions (UDFs)	✓	✓	✓	✓
⇒ Automatic Clustering	✓	✓	✓	✓
⇒ Zero-copy Cloning	✓	✓	✓	✓
⇒ Search Optimization Service	✗	✓	✓	✓
⇒ Materialized Views	✗	✓	✓	✓

Security, Governance & Data Protection

Feature	Standard	Enterprise	Business Critical	VPS
Continuous Data Protection				
⇒ Federated authentication and SSO	✓	✓	✓	✓
⇒ Multi-factor authentication (MFA)	✓	✓	✓	✓
⇒ Time Travel & Fail-safe	✓	✓	✓	✓
⇒ Encryption at-rest and in-transit	✓	✓	✓	✓
⇒ Network Policies	✓	✓	✓	✓
⇒ Access Control Framework	✓	✓	✓	✓

Security, Governance & Data Protection

Feature	Standard	Enterprise	Business Critical	VPS
⇒ Column and row access policies	✗	✓	✓	✓
⇒ Tri-secret secure	✗	✗	✓	✓
⇒ Private Connectivity	✗	✗	✓	✓
⇒ Support for compliance regulations: PCI DSS, HIPAA, HITRUST CSF, IRAP, FedRAMP	✗	✗	✓	✓
⇒ Dedicated metastore and pool of compute resources	✗	✗	✗	✓

Compute Resource Management

Feature	Standard	Enterprise	Business Critical	VPS
⇒ Separate compute clusters (Virtual Warehouses)	✓	✓	✓	✓
⇒ Resource Monitors	✓	✓	✓	✓
⇒ Multi-cluster Virtual Warehouses	✗	✓	✓	✓

Interface & Tools

Feature	Standard	Enterprise	Business Critical	VPS
⇒ Classic UI & Snowsight UI	✓	✓	✓	✓
⇒ SnowSQL & SnowCD	✓	✓	✓	✓
⇒ Native programming interfaces	✓	✓	✓	✓
⇒ Third-party tools ecosystem	✓	✓	✓	✓
⇒ Snowflake Partner Connect	✓	✓	✓	✓

Data Import & Export

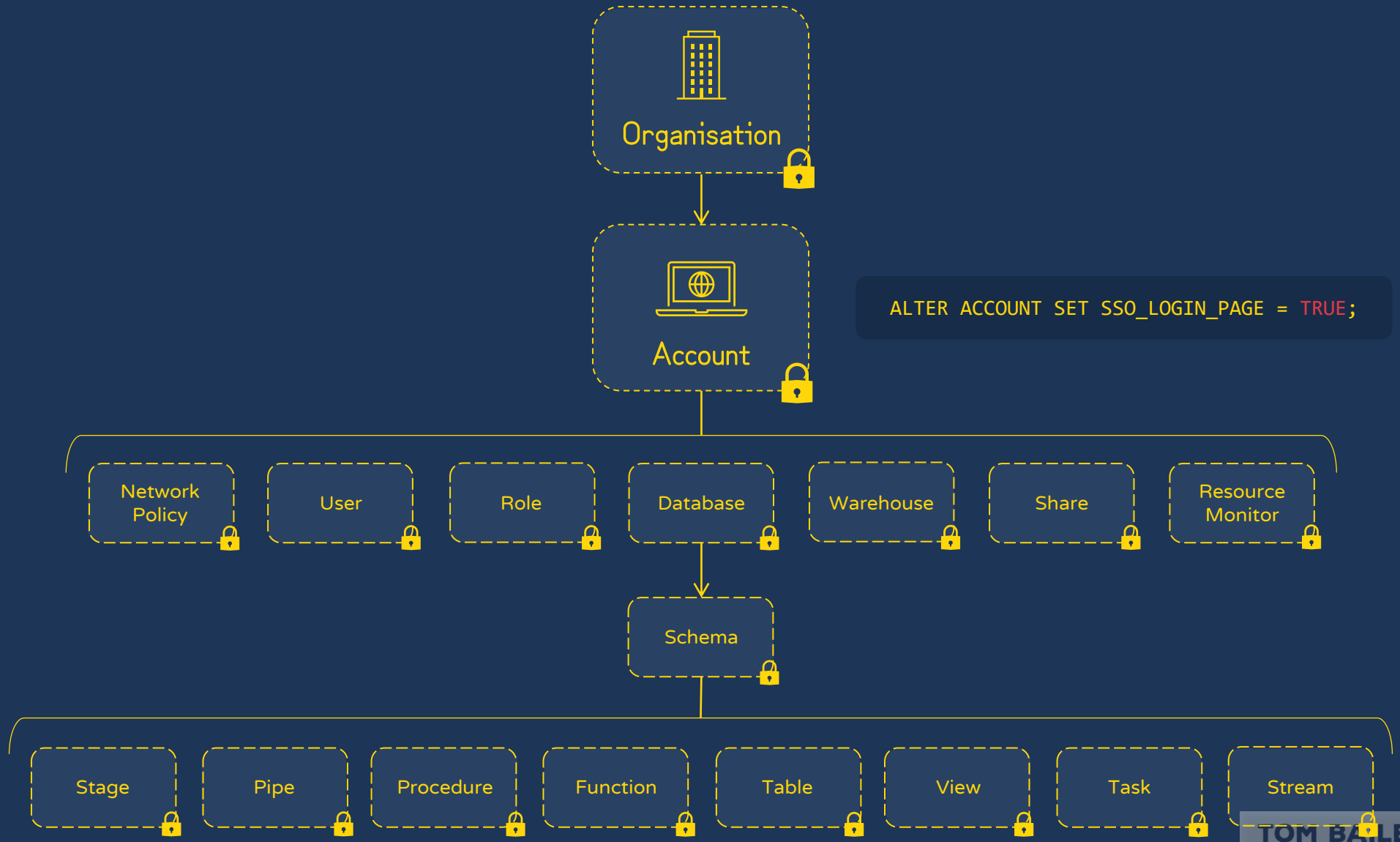
Feature	Standard	Enterprise	Business Critical	VPS
⇒ Bulk Loading	✓	✓	✓	✓
⇒ Bulk Unloading	✓	✓	✓	✓
⇒ Continuous Data Loading with Snowpipe	✓	✓	✓	✓
⇒ Snowflake Connector for Kafka	✓	✓	✓	✓

Data Replication & Failover

Feature	Standard	Enterprise	Business Critical	VPS
⇒ Database replication	✓	✓	✓	✓
⇒ Database failover and failback	✗	✓	✓	✓

Snowflake's Catalogue and Objects

Snowflake Object Model




```
ALTER ACCOUNT SET SSO_LOGIN_PAGE = TRUE;
```

Organisation, Account, Database & Schema.

Organisation Overview

1

Manage one or more Snowflake accounts.

2

Setup and administer Snowflake features which make use of multiple accounts.

3

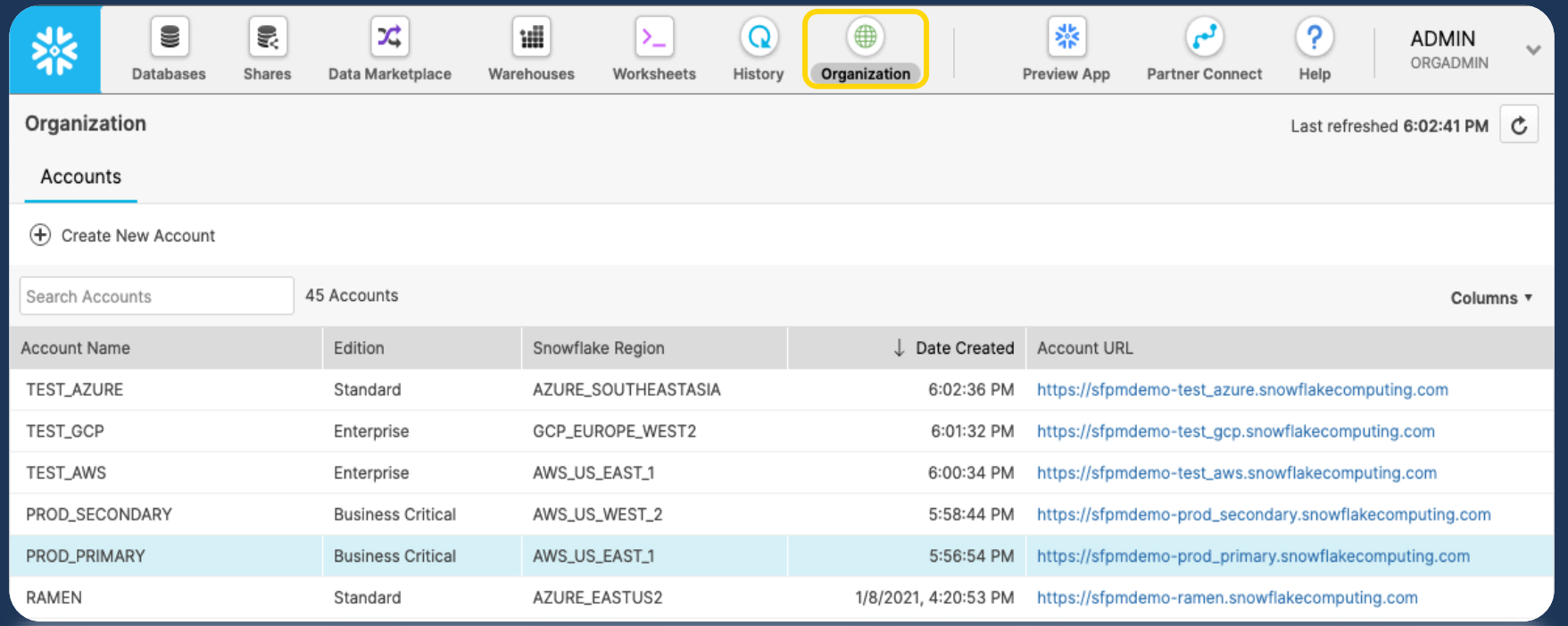
Monitoring usage across accounts.

Organisation Setup

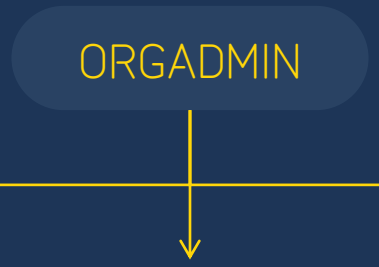
Contact Snowflake support

Provide organisation name and nominate an account

ORGADMIN role added to nominated account



ORGADMIN Role



Account Management

```
CREATE ACCOUNT MYACCOUNT1
  ADMIN_NAME = admin
  ADMIN_PASSWORD = 'Password123'
  FIRST_NAME = jane
  LAST_NAME = smith
  EMAIL = 'myemail@myorg.org'
  EDITION = enterprise
  REGION = aws_us_west_2;
```

```
SHOW ORGANIZATION ACCOUNTS;

SHOW REGIONS;
```

Enable cross-account features

```
SELECT
system$global_account_set_parameter(
  'UT677AA',
  'ENABLE_ACCOUNT_DATABASE_REPLICATION',
  'true');
```

Monitoring account usage

```
SELECT
ROUND(SUM(AVERAGE_BYTES) /POWER(1024,4),2)
FROM ORGANIZATION_USAGE.STORAGE_DAILY_HISTORY
WHERE USAGE_DATE = CURRENT_DATE();
```

Account Overview

An account is the administrative name for a collection of storage, compute and cloud services deployed and managed entirely on a selected cloud platform.

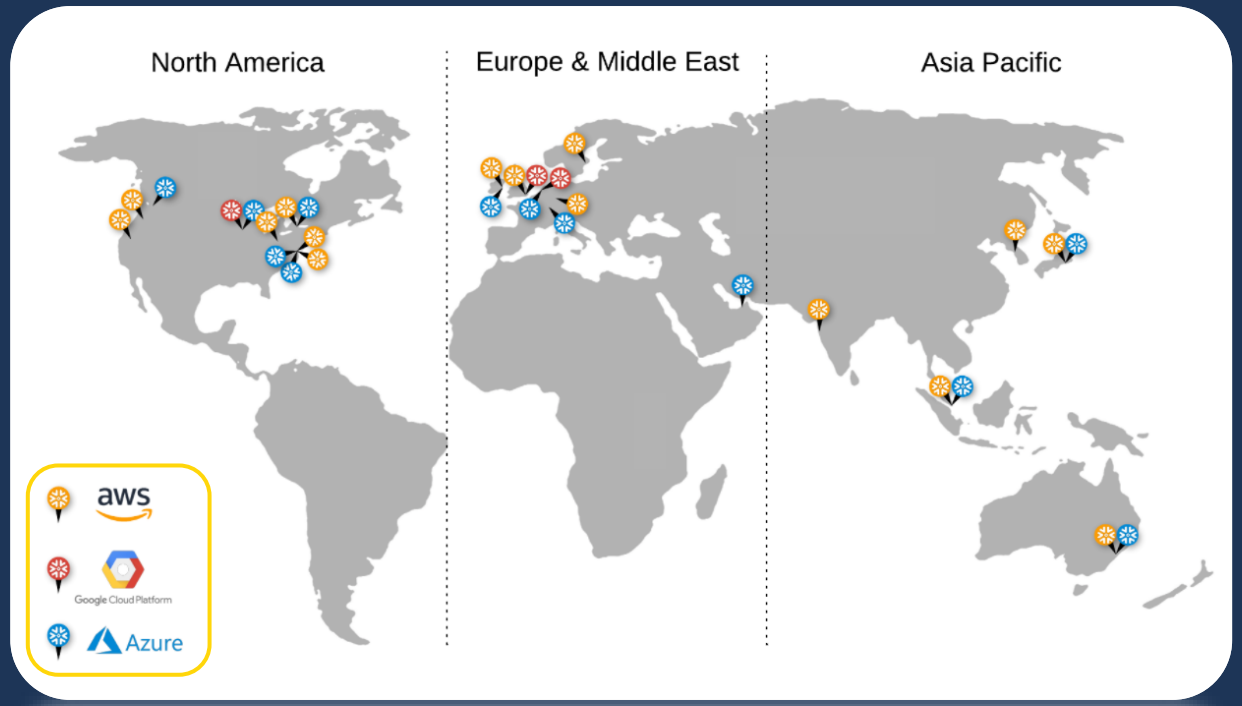
Each account is hosted on a **single cloud provider**:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure (Azure)

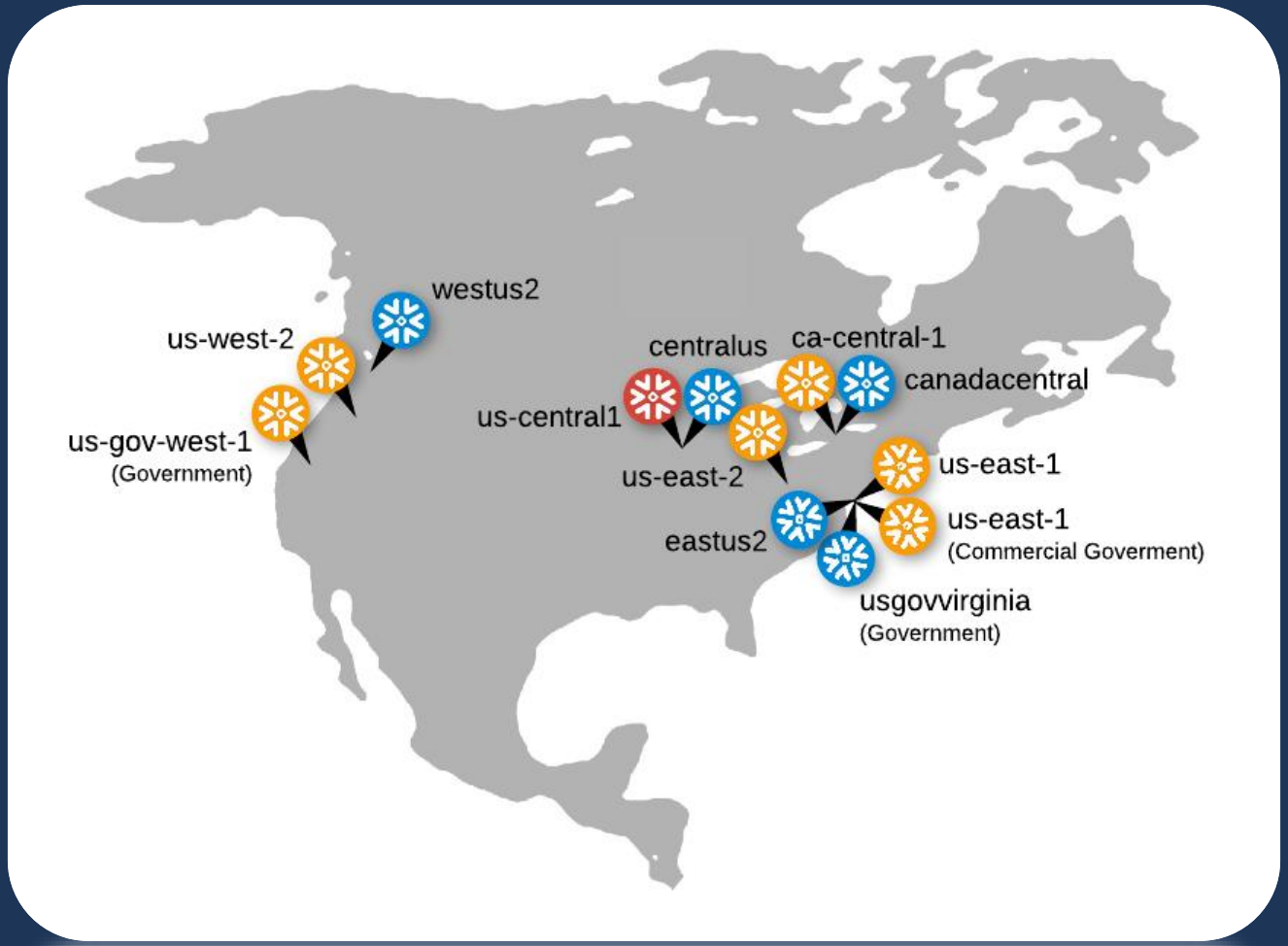
Each account is provisioned in a **single geographic region**.

Each account is created as a **single Snowflake edition**.

An account is created with the system-defined role ACCOUNTADMIN.



Account Regions



aws.us-west-2

US West (Oregon)

aws.ca-central-1

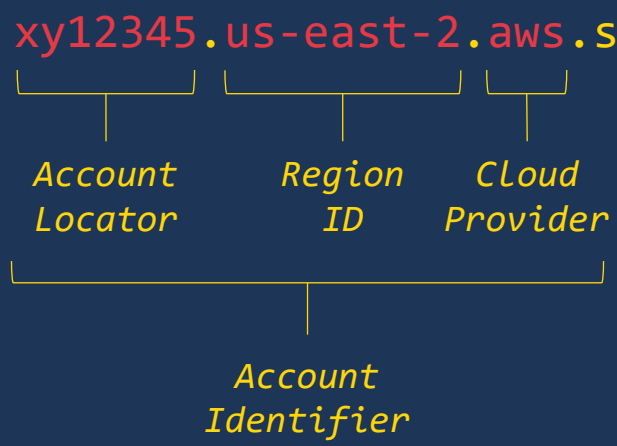
Canada (Central)

azure.westus2

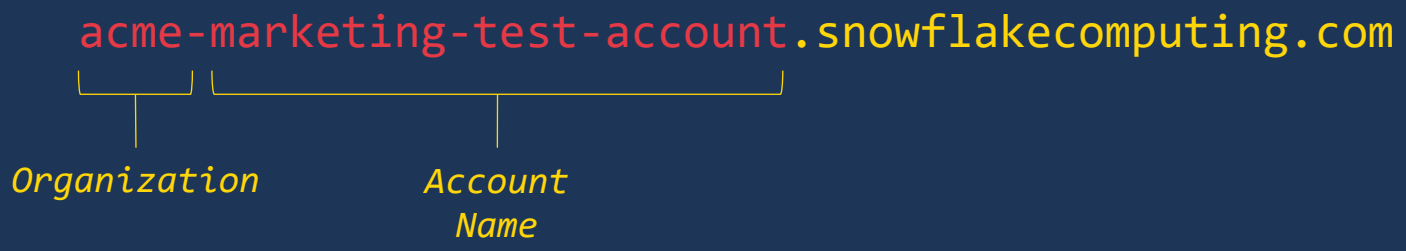
West US 2 (Washington)

Account URL

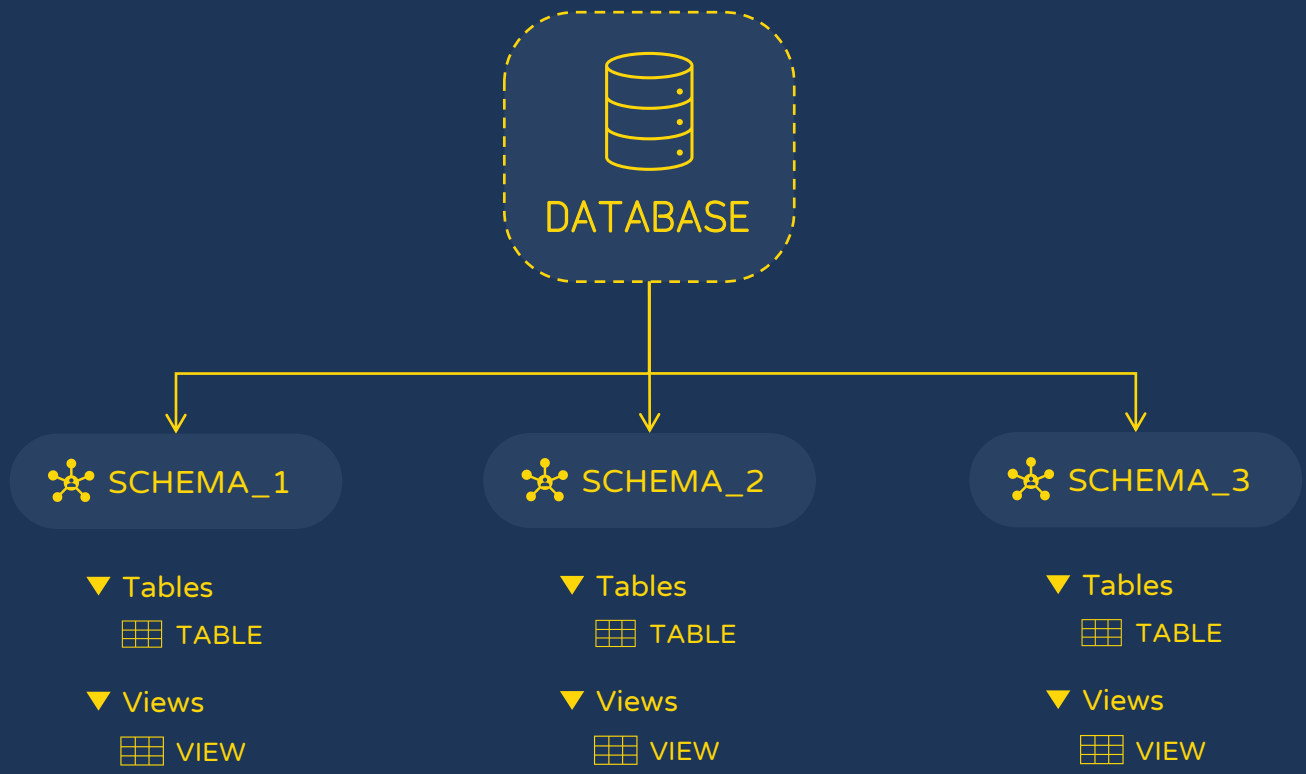
Using an Account Locator as an Identifier



Using an Organization and Account Name as Identifier



Database & Schemas



Database & Schemas

DATABASE

Databases must have a unique identifier in an account.

A database must start with an alphabetic character and cannot contain spaces or special characters unless enclosed in double quotes.

```
CREATE DATABASE MY_DATABASE;
```

```
CREATE DATABASE MY_DB_CLONE CLONE MYTESTDB;
```

```
CREATE DATABASE MYDB1
  AS REPLICA OF MYORG.ACCOUNT1.MYDB1
  DATA_RETENTION_TIME_IN_DAYS = 10;
```

```
CREATE DATABASE SHARED_DB FROM SHARE UTT783.SHARE;
```

SCHEMA

Schemas must have a unique identifier in a database.

A schema must start with an alphabetic character and cannot contain spaces or special characters unless enclosed in double quotes.

```
CREATE SCHEMA MY_SCHEMA;
```

```
CREATE SCHEMA MY_SCHEMA_CLONE CLONE MY_SCHEMA;
```

Table and View Types

Table Types

Permanent

Default table type.

Exists until explicitly dropped.

Temporary

Used for transitory data.

Persist for duration of a session.

Transient

Exists until explicitly dropped.

No fail-safe period.

External

Query data outside Snowflake.

Read-only table.

	Permanent	Temporary	Transient	External
Time Travel	✓ 90 days	✓ 1 day	✓ 1 day	✗
Fail-safe	✓	✗	✗	✗

View Types

Standard

```
CREATE VIEW MY_VIEW AS
SELECT COL1, COL2 FROM MY_TABLE;
```

Does not contribute to storage cost.

If source table is dropped, querying view returns error.

Used to restrict contents of a table.

Materialized

```
CREATE MATERILIZED VIEW MY_VIEW AS
SELECT COL1, COL2 FROM MY_TABLE;
```

Stores results of a query definition and periodically refreshes it.

Incurs cost as a serverless feature.

Used to boost performance of external tables.

Secure

```
CREATE SECURE VIEW MY_VIEW AS
SELECT COL1, COL2 FROM MY_TABLE;
```

Both standard and materialized can be secure.

Underlying query definition only visible to authorized users.

Some query optimizations bypassed to improve security.

UDFs and Stored Procedures

User Defined Functions (UDFs)

User defined functions (UDFs) are schema-level objects that enable users to write their own functions in three different languages:

- SQL
- JavaScript
- Python
- Java

```
CREATE FUNCTION AREA_OF_CIRLE(radius FLOAT)
 RETURNS FLOAT (area number)
 AS
 $$
   pi() * radius * radius
 $$;
```

UDFs accept 0 or more parameters.

UDFs can return scalar or tabular results (UDTF).

UDFs can be called as part of a SQL statement.

```
SELECT  AREA_OF_CIRCLE(col1) FROM MY_TABLE;
```

UDFs can be overloaded.

JavaScript UDF

```
CREATE FUNCTION JS_FACTORIAL(d double)
    RETURNS DOUBLE
    LANGUAGE JAVASCRIPT
    AS
    $$
    if (D <= 0) {
        return 1
    } else {
        var result = 1;
        for (var i = 2; i <= D; i++) {
            result = result * i;
        }
        return result;
    }
    $$;
```

JavaScript

JavaScript is specified with the language parameter.

Enables use of high-level programming language features.

JavaScript UDFs can refer to themselves recursively.

Snowflake data types are mapped to JavaScript data types.

Java UDF

```
CREATE FUNCTION DOUBLE(X INTEGER)
     RETURNS INTEGER
     LANGUAGE JAVA
     HANDLER='TestDoubleFunc.double'
     TARGET_PATH='@~/TestDoubleFunc.jar'
     AS
     $$
         class TestDoubleFunc {
             public static int double(int x) {
               return x * 2;
             }
         }
     $$;
```

Java

Snowflake boots up a JVM to execute function written in Java.

Snowflake currently supports writing UDFs in Java versions 8.x, 9.x, 10.x, and 11.x.

Java UDFs can specify their definition as in-line code or a pre-compiled jar file.

Java UDFs cannot be designated as secure.

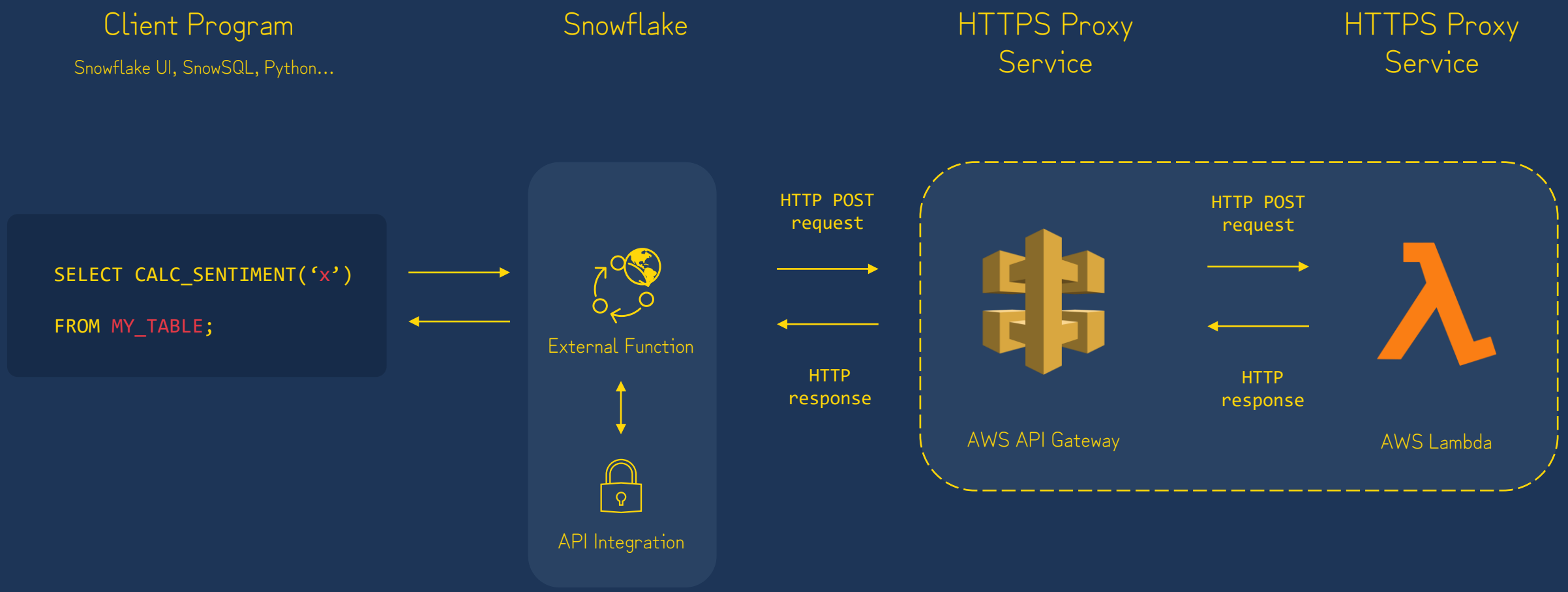
External Functions

```
CREATE OR REPLACE EXTERNAL FUNCTION CALC_SENTIMENT(STRING_COL VARCHAR)
    RETURNS VARIANT
    API_INTEGRATION = AWS_API_INTEGRATION
    AS 'https://ttu.execute-api.eu-west-2.amazonaws.com/';
```

← Function Name and Parameters
← Return Type
← Integration Object
← URL Proxy Service

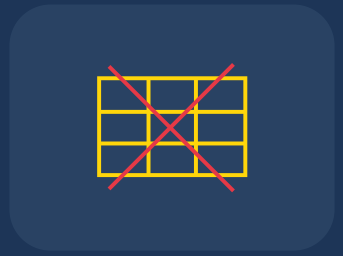
```
CREATE OR REPLACE API INTEGRATION AWS_API_INTEGRATION
    API_PROVIDER=AWS_API_GATEWAY
    API_AWS_ROLE_ARN='ARN:AWS:IAM::123456789012:ROLE/MY_CLOUD_ACCOUNT_ROLE'
    API_ALLOWED_PREFIXES=('HTTPS://XYZ.EXECUTE-API.US-WEST-2.AMAZONAWS.COM/PRODUCTION')
    ENABLED=TRUE;
```

External Function Call Lifecycle

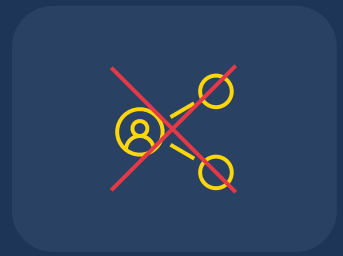


External Function Limitations

Slower



Scalar only



Not sharable

Less secure

Egress charges

Stored Procedures

In Relational Database Management Systems (RDBMS) **stored procedures** were **named collections of SQL statements** often containing procedural logic.

Database Admin (DBA)

Data Engineer (DE)

```
CREATE PROCEDURE CLEAR_EMP_TABLES
DELETE FROM EMP01 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())
AS
DELETE FROM EMP02 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())
BEGIN
DELETE FROM EMP03 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())

DELETE FROM EMP04 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())

DELETE FROM EMP05 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())



END
```

```
EXECUTE CLEAR_EMP_TABLES;
```

ⓘ SQL examples on this slide from **Microsoft SQL Server.**

Snowflake Stored Procedures

1

JavaScript

2

Snowflake Scripting

SQL

3

Snowpark

Python

Java

Scala

Stored Procedure: JavaScript

Stored procedure identifier and input parameters. →

RETURNS option mandatory. →

JAVASCRIPT, SQL, PYTHON, JAVA & SCALA. →

Stored procedures can execute with the owner's rights or caller's rights. →

Stored procedures mix JavaScript and SQL in their definition using Snowflake's JavaScript API. →

```
CALL EXAMPLE_STORED_PROCEDURE('EMP01');
```

```
CREATE PROCEDURE EXAMPLE_STORED_PROCEDURE(PARAM1 STRING)
    RETURNS STRING
    LANGUAGE JAVASCRIPT
    EXECUTE AS OWNER
    AS
    $$
        var param1 = PARAM1;
        var sql_command = "SELECT * FROM " + param1;
        snowflake.execute({sqlText: sql_command});
        return "Succeeded.";
    $$;
```

Stored Procedures & UDFs

Feature	UDF	Stored Procedure
Called as part of SQL statement	✓	✗
Ability to overload	✓	✓
0 or more input parameters	✓	✓
Use of JavaScript API	✗	✓
Return of value optional	✗	✓
Values returned usable in SQL	✓	✗
Call itself recursively	✗	✓

Functions calculate something and return a value to the user.

Stored procedures perform actions rather than return values.

Sequences

Sequences

```
CREATE SEQUENCE DEFAULT_SEQUENCE
START = 1
INCREMENT = 1;
```

```
SELECT DEFAULT_SEQUENCE.NEXTVAL;
```

NEXTVAL
1

```
SELECT DEFAULT_SEQUENCE.NEXTVAL;
```

NEXTVAL
2

```
SELECT DEFAULT_SEQUENCE.NEXTVAL;
```

NEXTVAL
3

ⓘ Values generated by a sequence are globally unique.

Sequences

```
CREATE SEQUENCE INCREMENT_SEQUENCE

START = 0

INCREMENT = 5;
```

```
SELECT INCREMENT_SEQUENCE.NEXTVAL, INCREMENT_SEQUENCE.NEXTVAL, INCREMENT_SEQUENCE.NEXTVAL, INCREMENT_SEQUENCE.NEXTVAL;
```

NEXTVAL	NEXTVAL_1	NEXTVAL_2	NEXTVAL_3
~~30~~	~~40~~	~~~~40~~~~	~~~~50~~~~

ⓘ Sequences cannot guarantee their values will be gap free.

Sequences

1. INSERT INTO TABLE.

```
CREATE SEQUENCE TRANSACTION_SEQ

START = 1001

INCREMENT = 1;
```

```
INSERT INTO TRANSACTION (ID)
VALUES (TRANSACTION_SEQ.NEXTVAL)
```

```
SELECT ID FROM TRANSACTION;
```

NEXTVAL
1001

Sequences

2 DEFAULT VALUE FOR A COLUMN TABLE.

```
CREATE TABLE TRANSACTIONS
(ID INTEGER DEFAULT TRANSACTION_SEQ.NEXTVAL,
AMOUNT DOUBLE);
```

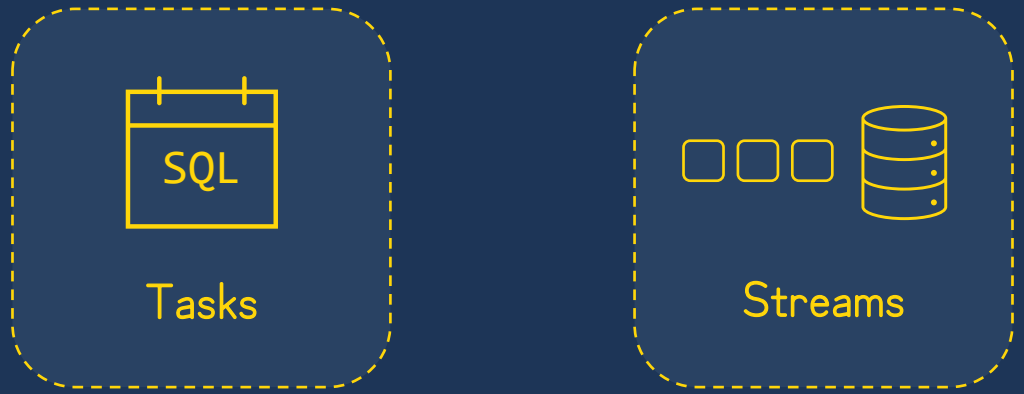
```
INSERT INTO TRANSACTION (AMOUNT) VALUES (756.00);
```

```
SELECT ID FROM TRANSACTION;
```

ID	AMOUNT
1002	756.00

Tasks & Streams

Tasks & Streams



Tasks

A task is an object used to schedule the execution of a SQL command or a stored procedure.

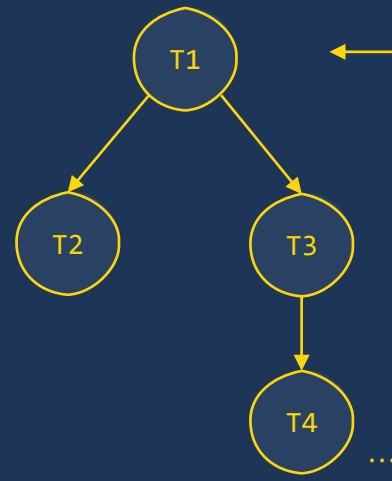
Task Workflow

1. ACCOUTNADMIN role or CREATE TASK privilege.

2.
```
CREATE TASK T1
  WAREHOUSE = MYWH
  SCHEDULE = '30 MINUTE'
AS
  COPY INTO MY_TABLE
  FROM $MY_STAGE;
```
← Task Name
← Warehouse Definition
← Triggering Mechanism
← Query Definition

3.
```
ALTER TASK MINUTE_TASK RESUME;
```
← Start task

Tree of Tasks



← Root Task
Must define a schedule.

← Child Task
Cannot define a schedule.

← Up to 1000 child tasks

```
CREATE TASK T2
  WAREHOUSE = MYWH
  AFTER T1
AS
  COPY INTO MY_TABLE FROM $MY_STAGE;
```
← Child task triggering mechanism

Streams

A stream is an object created to view & track DML changes to a source table – inserts, updates & deletes.

Create Stream

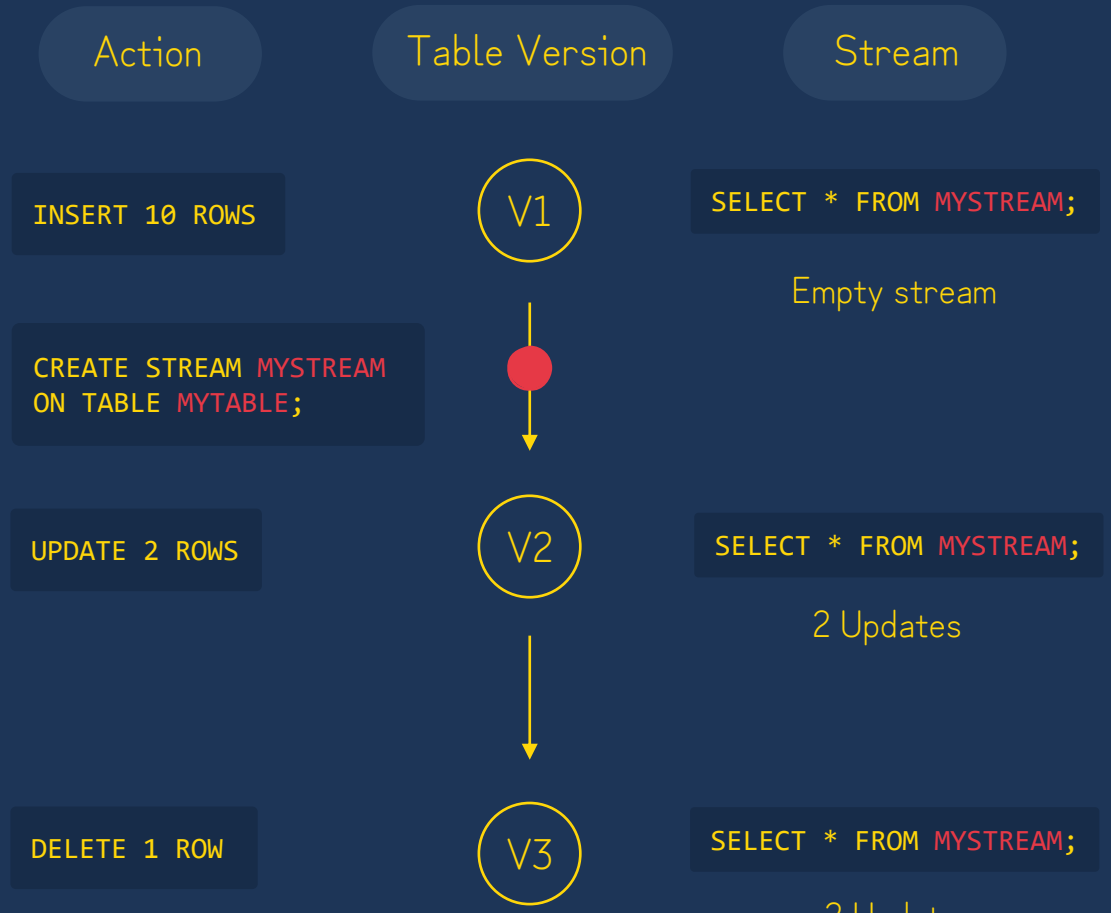
```
CREATE STREAM MY_STREAM ON TABLE MY_TABLE;
```

Query Stream

```
SELECT * FROM MY_STREAM;
```

EMP_ID	EMP_NAME	EMP_DOB	EMP_POSITION	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID
AC1IO9	Ramesh Aravind	10/09/1964	Actor	INSERT	FALSE	cc576bf4fee43b88c4fd03

Streams



Progress stream offset

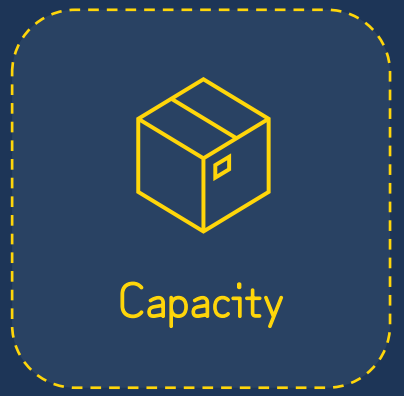
```
INSERT INTO MYTABLE2 SELECT * FROM MYSTREAM;
```

Tasks & Streams

```
CREATE TASK MYTASK1
  WAREHOUSE = MYWH
  SCHEDULE = '5 MINUTE'
WHEN
  SYSTEM$STREAM_HAS_DATA('MYSTREAM')
AS
  INSERT INTO MYTABLE1(ID,NAME) SELECT ID, NAME
  FROM MYSTREAM WHERE METADATA$ACTION = 'INSERT';
```

Billing

Billing Overview



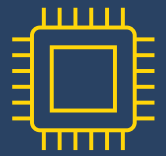
Pay for usage as you go

Pay for usage upfront

Billing Overview

Virtual Warehouse Services

Cloud Services



Serverless Services

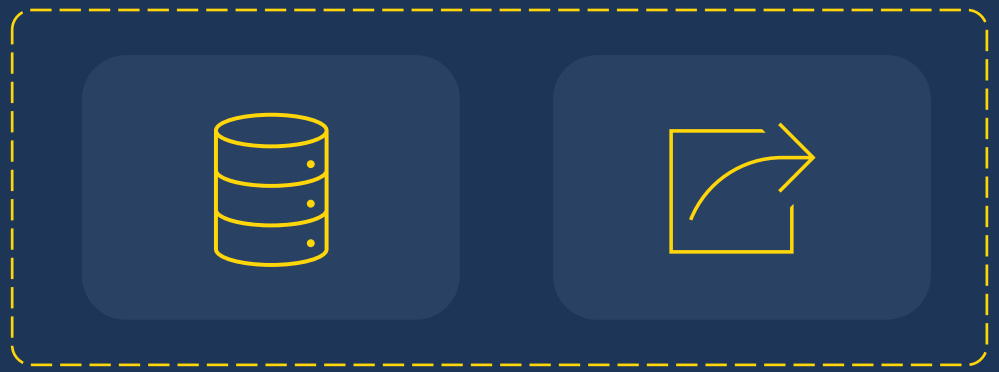
Storage

Data Transfer

Billing Overview

Credits

Dollar Value



Compute Billing Overview

Credits | Snowflake billing unit of measure for compute resource consumption.

Virtual Warehouse Services

- Credit calculated based on size of virtual warehouse.
- Credit calculated on per second basis while a virtual warehouse is in 'started' state.
- Credit calculated with a minimum of 60 seconds.

Cloud Services

- Credits calculated at a rate of 4.4 Credits per compute hour.
- Only cloud services that exceeds 10% of the daily usage of the compute resources are billed.
- This is called the Cloud Services Adjustment.

Serverless Services

- Each serverless feature has it's own credit rate per compute-hour.
- Serverless features are composed of both compute services and cloud services.
- Cloud Services Adjustment does not apply to cloud services usage when used by serverless features.

Data Storage & Transfer Billing Overview

Dollar value | Storage and Data Transfer are billed in currency.

Data Storage

- Data storage is calculated monthly based on the average number of on-disk bytes per day in the following locations:
  - **Database Tables.**
  - **Internal Stages.**
- Costs calculated based on a flat dollar value rate per terabyte (TB) based on:
  - **Capacity or On-demand.**
  - **Cloud provider.**
  - **Region.**

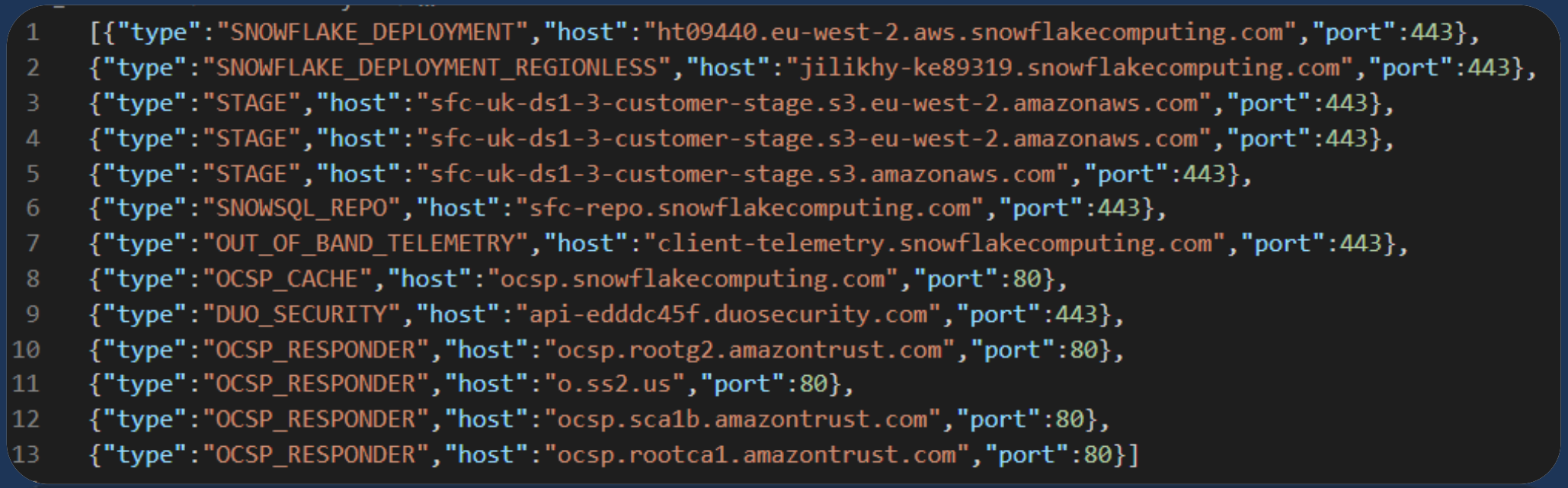
Data Transfer

- Data transfer charges apply when moving data from one region to another or from one cloud platform to another.
- Unloading data from Snowflake using `COPY INTO <location>` command.
- Replicating data to a Snowflake account in a different region or cloud platform.
- External functions transferring data out of and into Snowflake.

SnowCD

1. `SELECT SYSTEM$WHITELIST();`

Returns hostnames and port numbers.



```
[{"type":"SNOWFLAKE_DEPLOYMENT","host":"ht09440.eu-west-2.aws.snowflakecomputing.com","port":443},
{"type":"SNOWFLAKE_DEPLOYMENT_REGIONLESS","host":"jilikhy-ke89319.snowflakecomputing.com","port":443},
{"type":"STAGE","host":"sfc-uk-ds1-3-customer-stage.s3.eu-west-2.amazonaws.com","port":443},
{"type":"STAGE","host":"sfc-uk-ds1-3-customer-stage.s3-eu-west-2.amazonaws.com","port":443},
{"type":"STAGE","host":"sfc-uk-ds1-3-customer-stage.s3.amazonaws.com","port":443},
{"type":"SNOWSQL_REPO","host":"sfc-repo.snowflakecomputing.com","port":443},
{"type":"OUT_OF_BAND_TELEMETRY","host":"client-telemetry.snowflakecomputing.com","port":443},
{"type":"OCSP_CACHE","host":"ocsp.snowflakecomputing.com","port":80},
{"type":"DUO_SECURITY","host":"api-edddc45f.duosecurity.com","port":443},
{"type":"OCSP_RESPONDER","host":"ocsp.rootg2.amazontrust.com","port":80},
{"type":"OCSP_RESPONDER","host":"o.ss2.us","port":80},
{"type":"OCSP_RESPONDER","host":"ocsp.sca1b.amazontrust.com","port":80},
{"type":"OCSP_RESPONDER","host":"ocsp.rootca1.amazontrust.com","port":80}]
```

SnowCD

whitelist.json


```
[{"type":"SNOWFLAKE_DEPLOYMENT","host":"ht09440.eu-west-2.aws.snowflakecomputing.com","port":443},
{"type":"SNOWFLAKE_DEPLOYMENT_REGIONLESS","host":"jilikhy-ke89319.snowflakecomputing.com","port":443},
{"type":"STAGE","host":"sfc-uk-ds1-3-customer-stage.s3.eu-west-2.amazonaws.com","port":443},
{"type":"STAGE","host":"sfc-uk-ds1-3-customer-stage.s3-eu-west-2.amazonaws.com","port":443},
{"type":"STAGE","host":"sfc-uk-ds1-3-customer-stage.s3.amazonaws.com","port":443},
{"type":"SNOWSQL_REPO","host":"sfc-repo.snowflakecomputing.com","port":443},
{"type":"OUT_OF_BAND_TELEMETRY","host":"client-telemetry.snowflakecomputing.com","port":443},
{"type":"OCSP_CACHE","host":"ocsp.snowflakecomputing.com","port":80},
{"type":"DUO_SECURITY","host":"api-edddc45f.duosecurity.com","port":443},
{"type":"OCSP_RESPONDER","host":"ocsp.rootg2.amazontrust.com","port":80},
{"type":"OCSP_RESPONDER","host":"o.ss2.us","port":80},
{"type":"OCSP_RESPONDER","host":"ocsp.sca1b.amazontrust.com","port":80},
{"type":"OCSP_RESPONDER","host":"ocsp.rootca1.amazontrust.com","port":80}]
```


```
snowcd ~\whitelist.json
```

```
Performing 30 checks on 12 hosts
All checks passed.
```

```
Check for 5 hosts failed, display as follow:
================================================
Host: ocsp.snowflakecomputing.com
Port: 80
Type: OCSP_CACHE
Failed Check: HTTP checker
Error: Invalid http code received: 400 Bad Request
Suggestion: Check the connection to your http host or transparent Proxy
```

Connectivity: Connectors, Drivers and Partnered Tools

Connectors and Drivers

Python

Go

PHP

.NET

NodeJS

Spark

Kafka

JDBC

ODBC

Connect third-party tools

Python Connector Example

```
pip install snowflake-connector-python==2.6.2
```

```
#!/usr/bin/env python
import snowflake.connector

# Gets the version
ctx = snowflake.connector.connect(
    user='<user_name>',
    password='<password>',
    account='<account_identifier>'
    )
cs = ctx.cursor()
try:
    cs.execute("SELECT current_version()")
    one_row = cs.fetchone()
    print(one_row[0])
finally:
    cs.close()
ctx.close()
```

Snowflake Partner Tools

Business Intelligence

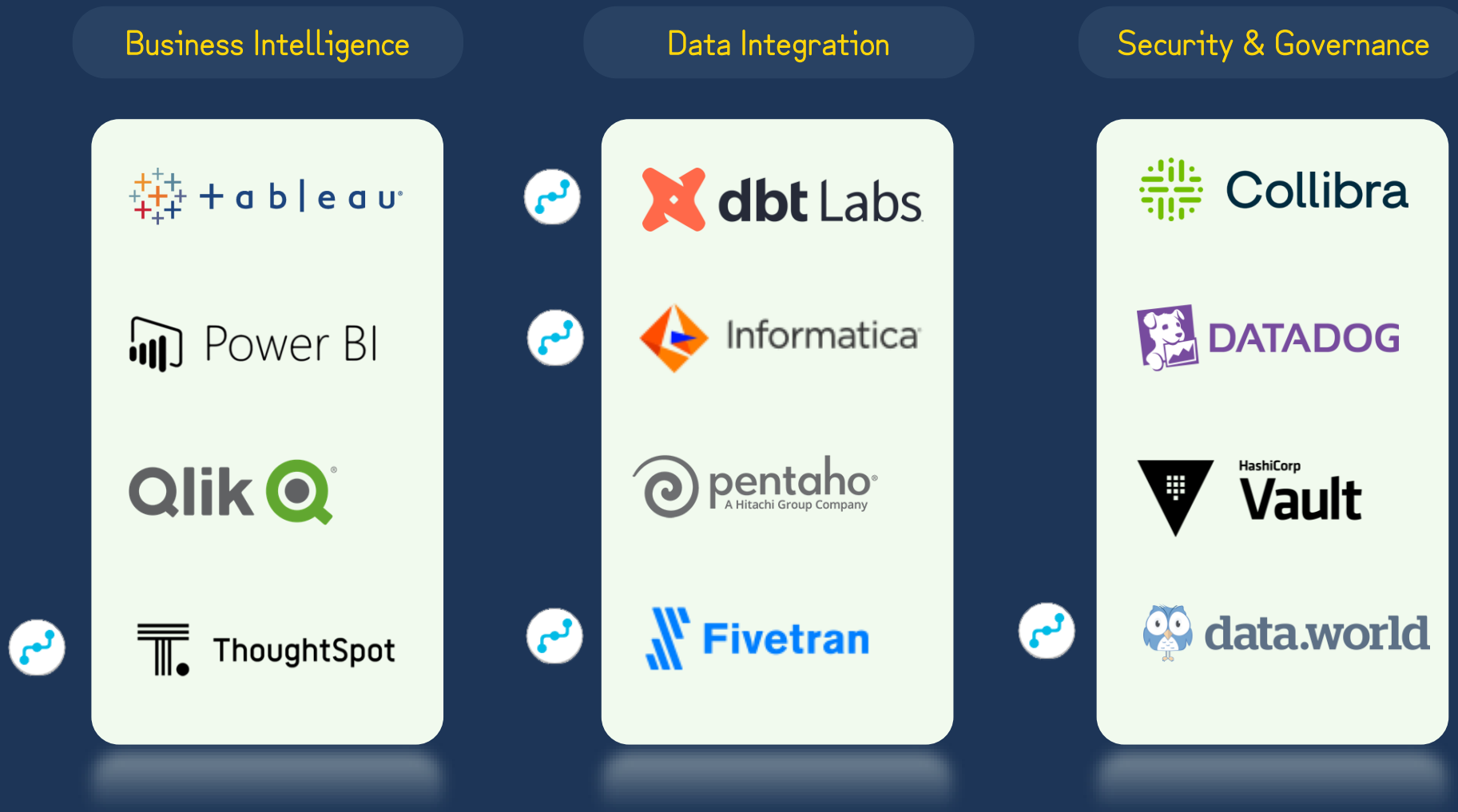
Data Integration

Security & Governance

SQL Development & Management

Machine Learning & Data Science

Snowflake Partner Tools



Snowflake Partner Connect is a feature to expedite connectivity with partnered tools.

Snowflake Partner Tools

Machine Learning & Data Science

DataRobot

data iku

Amazon SageMaker

zepl

SQL Development & Management

SqlDBM

SEEKWELL

Hackolade

Agile Data ENGINE

Snowflake Scripting

Snowflake Scripting

Snowflake Scripting is an extension to Snowflake SQL that adds support for **procedural logic.**

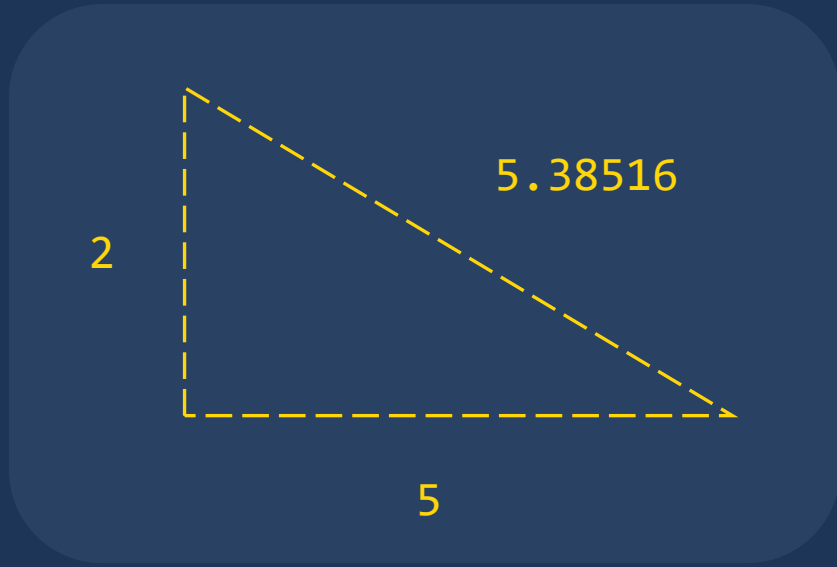
It's used to write **stored procedures** and **procedural code outside of a stored procedure.**

```
DECLARE
   (variable declarations, cursor declarations, etc.)
BEGIN
   (Snowflake Scripting and SQL statements)
EXCEPTION
   (statements for handling exceptions)
END;
```

Snowflake Scripting

```
declare
    leg_a number(38, 2);
    hypotenuse number(38,5);
begin
    leg_a := 2;
    let leg_b := 5;

    hypotenuse := sqrt(square(leg_a) + square(leg_b));
    return hypotenuse;
end;
```



anonymous block
5.38516

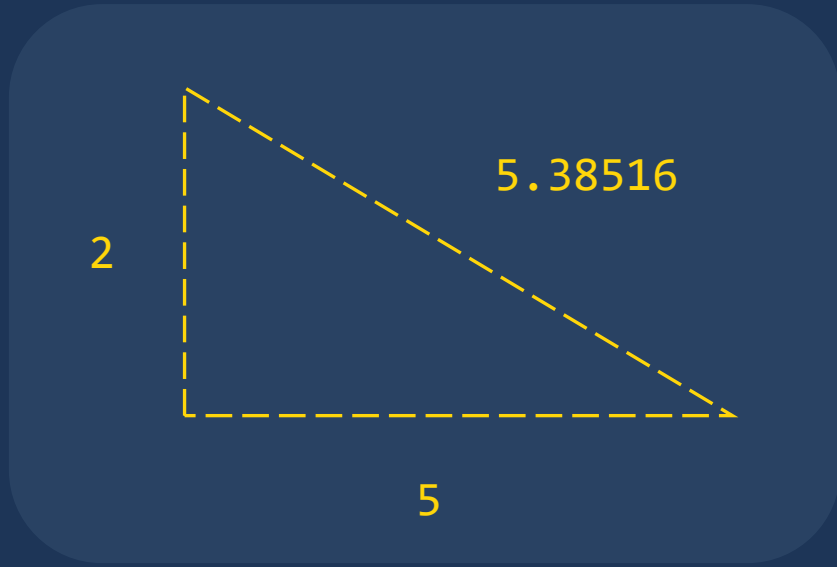
ⓘ Variables can only be used within the scope of the block.

ⓘ Variables can also be declared and assigned in the BEGIN section using the LET keyword.

Snowflake Scripting

```
CREATE PROCEDURE pythagoras()
RETURNS float
LANGUAGE sql
AS
declare
    leg_a number(38, 2);
    hypotenuse number(38,5);
begin
    leg_a := 2;
    let leg_b := 5;

    hypotenuse := sqrt(square(leg_a) + square(leg_b));
    return hypotenuse;
end;
```



ⓘ SnowSQL and the Classic Console do not correctly parse Snowflake Scripting blocks, they need to be wrapped in string constant delimiters like dollar signs.

Branching Constructs

```
begin
  let count := 4;
  if (count % 2 = 0) then
    return 'even value';
  else
    return 'odd value';
  end if;
end;
```

DECLARE or EXCEPTION sections of a block are optional.

Looping Constructs

```
declare
  total integer default 0;
  max_num integer default 10;
begin
  for i in 1 to max_num do
    total := i + total;
  end for;
  return total;
end;
```

anonymous block
55

Cursor

```
declare
  total_amount float;
  c1 cursor for select amount from transactions;
begin
  total_amount := 0.0;
  for record in c1 do
    total_amount := total_amount + record.amount;
  end for;
  return total_amount;
end;
```

anonymous block
136.78

RESULTSET

TABLE()

```
declare
    res resultset;
  begin
    res := (select amount from transactions);
    return table(res);
  end;
```

amount
101.01
24.78
10.99

RESULTSET

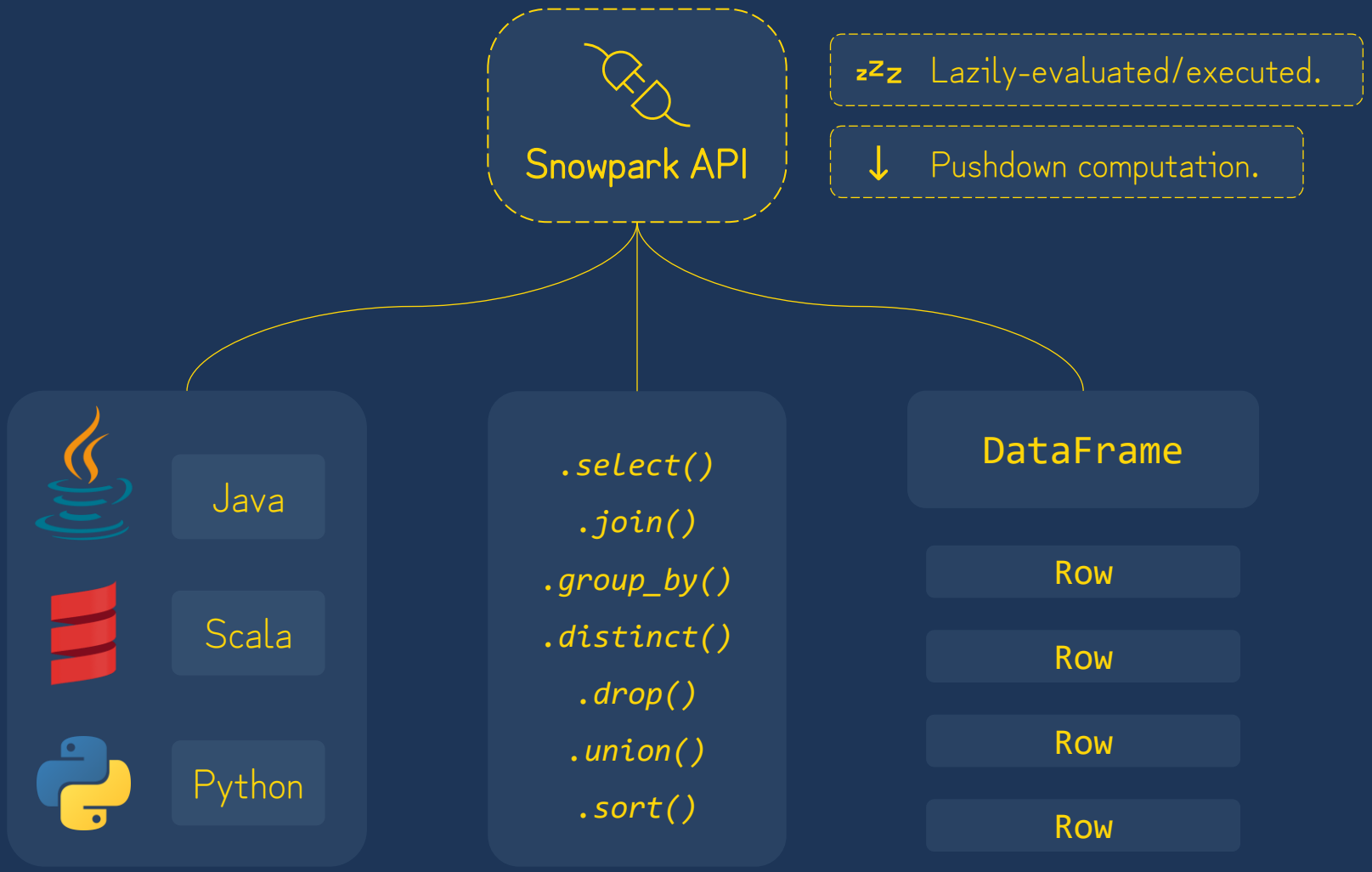
Cursor

```
declare
    total_amount float;
    res resultset default (select amount from transactions);
    c1 cursor for res;
begin
  total_amount := 0.0;
  for record in c1 do
    total_amount := total_amount + record.amount;
  end for;
  return total_amount;
end;
```

anonymous block
136.78

Snowpark

Snowpark


Snowpark API
zzz Lazily-evaluated/executed.
↓ Pushdown computation.
Java
Scala
Python
.select()
.join()
.group_by()
.distinct()
.drop()
.union()
.sort()
DataFrame
Row
Row
Row
Row

Snowpark API: Python

```python
import os
from snowflake.snowpark import Session
from snowflake.snowpark.functions import col
```

```python
connection_parameters = {
    "account": os.environ["snowflake_account"],
    "user": os.environ["snowflake_user"],
    "password": os.environ["snowflake_password"],
    "role": os.environ["snowflake_user_role"],
    "warehouse": os.environ["snowflake_warehouse"],
    "database": os.environ["snowflake_database"],
    "schema": os.environ["snowflake_schema"]
  }
```

Snowpark API: Python

```
14    session = Session.builder.configs(connection_parameters).create()

15    transactions_df = session.table("transactions")

16    print(transactions_df.collect())
```

Console output:

```
[Row(ACCOUNT_ID=8764442, AMOUNT=12.99),
Row(ACCOUNT_ID=8764442, AMOUNT=50.0),
Row(ACCOUNT_ID=8764442, AMOUNT=1100.0),
Row(ACCOUNT_ID=8764443, AMOUNT=110.0),
Row(ACCOUNT_ID=8764443, AMOUNT=2766.0),
Row(ACCOUNT_ID=8764443, AMOUNT=1010.0),
Row(ACCOUNT_ID=8764443, AMOUNT=3022.23),
Row(ACCOUNT_ID=8764444, AMOUNT=6986.0),
Row(ACCOUNT_ID=8764444, AMOUNT=1500.0)]
```

Snowpark API: Python

```
18  transactions_df_filtered = transactions_df.filter(col("amount") >= 1000.00)

19  transaction_counts_df = transactions_df_filtered.group_by("account_id").count()

20  flagged_transactions_df = transaction_counts_df.filter(col("count") >= 2).rename(col("count"), "flagged_count")

21  flagged_transactions_df.write.save_as_table("flagged_transactions", mode="append")

22  print(flagged_transactions_df.show())

23  session.close()
```

Console output:

```
-----------------------------------
|“ACCOUNT_ID”  |“FLAGGED_COUNT”  |
-----------------------------------
|8764443       |3                |
|8764444       |2                |
-----------------------------------
```