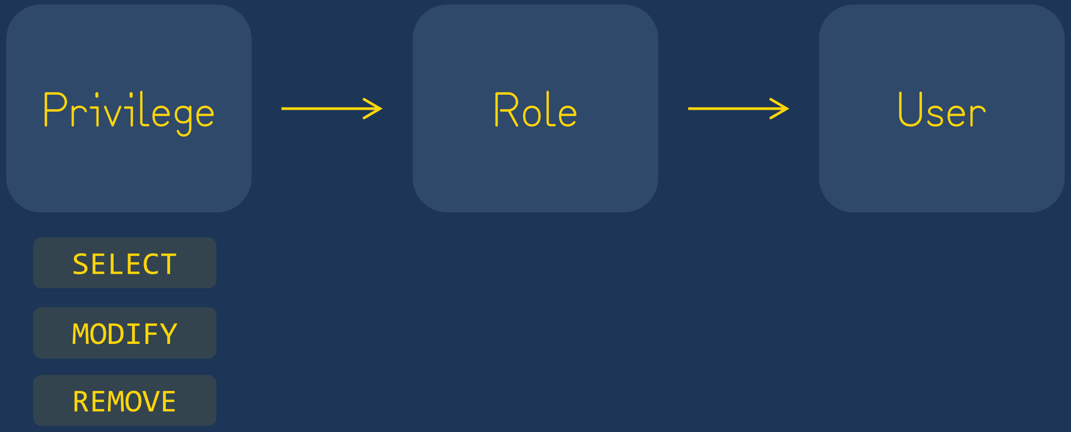
Access Control Overview

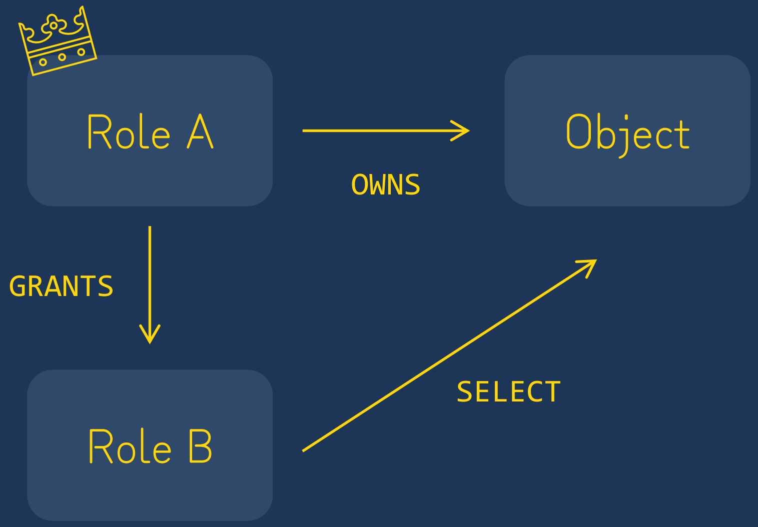
Access Control Overview

Role-based Access Control (RBAC)



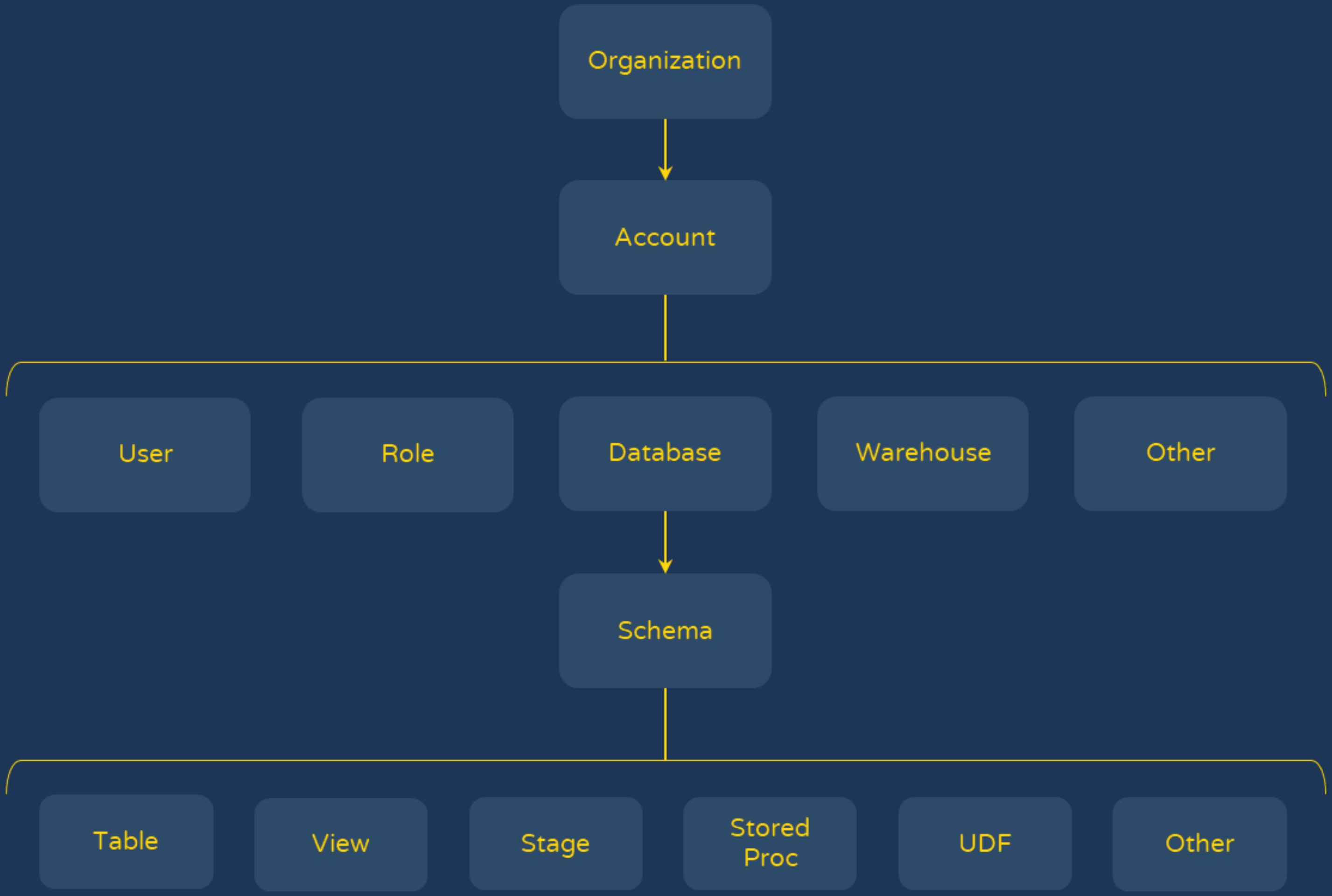
Role-based access control (RBAC) is an access control framework in which access privileges are assigned to roles and in turn assigned to users.

Discretionary Access Control (DAC)

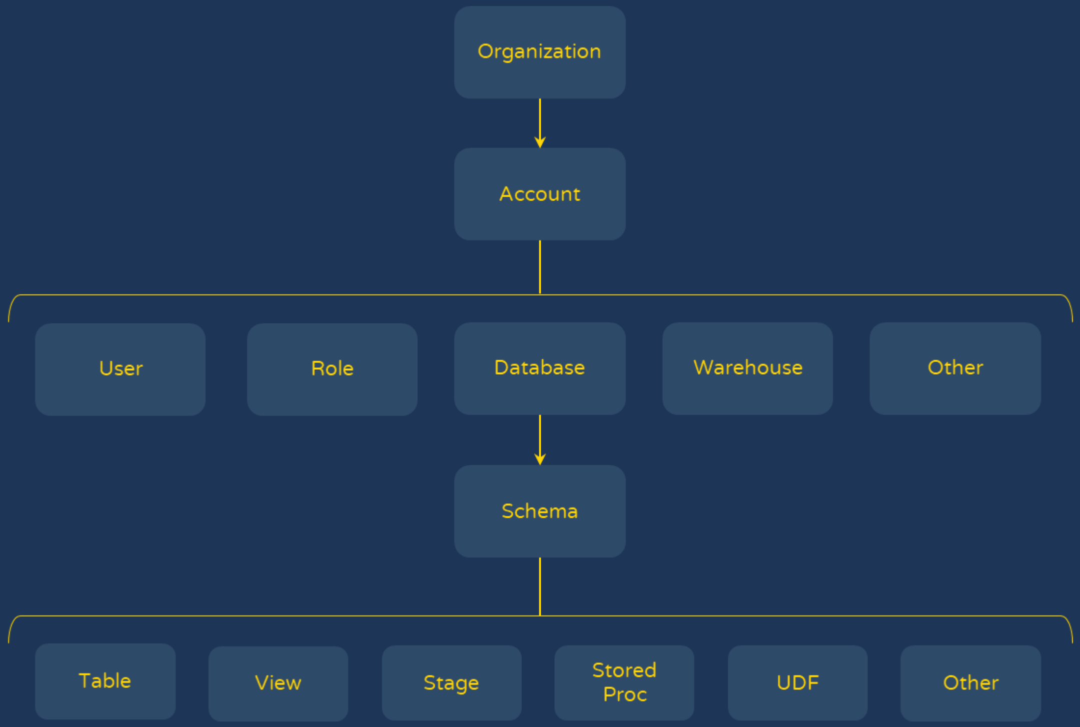


Snowflake combines RBAC with Discretionary Access Control (DAC) in which each object has an owner, who can in turn grant access to that object.

Securable Objects



Securable Objects



Every securable object is owned by a single role which can be found by executing a SHOW <object> command.

The owning role:

- Has all privileges on the object by default.
- Can grant or revoke privileges on the object to other roles.
- Transfer ownership to another role.
- Share control of an object if the owning role is shared.

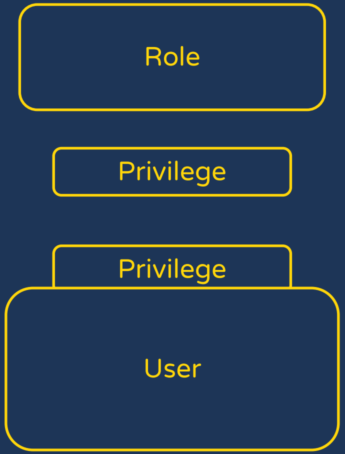
Access to objects is also defined by privileges granted to roles:

- Ability to create a Warehouse.
- Ability to list tables contained in a schema
- Ability to add data to a table

Unless allowed by a grant, access to a securable object will be denied.

Roles

Roles



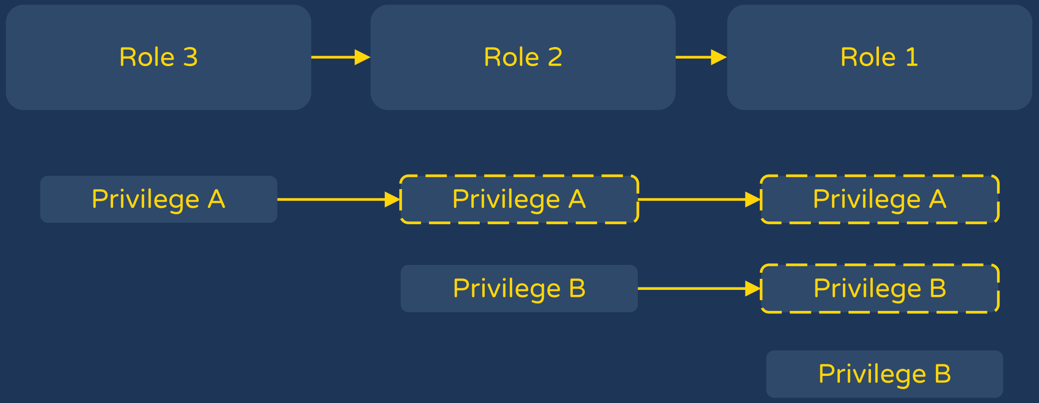
```
GRANT USAGE ON DATABASE TEST_DB TO ROLE TEST_ROLE;

GRANT SELECT ON TABLE TEST_TABLE TO ROLE TEST_ROLE;

GRANT USAGE ON SCHEMA TEST_SCHEMA TO ROLE TEST_ROLE;
GRANT ROLE TEST_ROLE TO USER ADMIN;
```

```
GRANT ROLE ROLE_3
 TO ROLE ROLE_2;
```

```
GRANT ROLE ROLE_2
 TO ROLE ROLE_1;
```



A role is an entity to which privileges on securable objects can be granted or revoked.

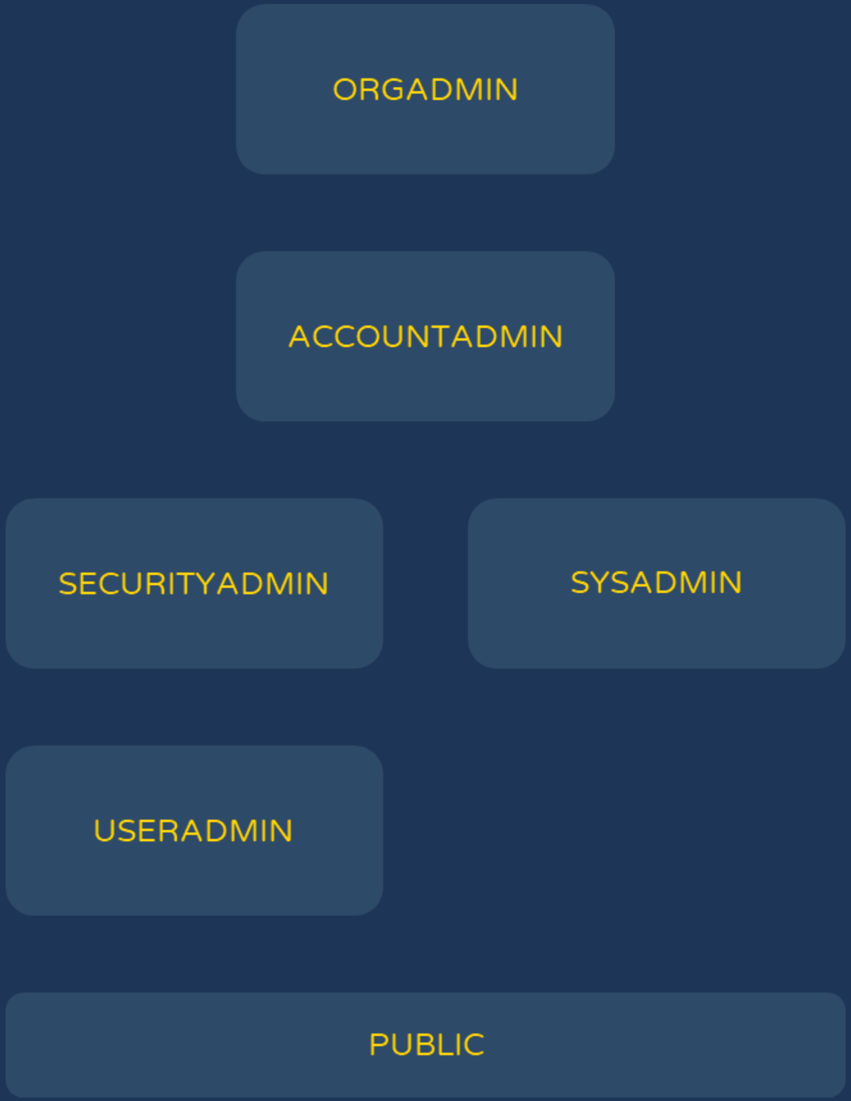
Roles are assigned to users to give them the authorization to perform actions.

A user can have multiple roles and switch between them within a Snowflake session.

Roles can be granted to other roles creating a role hierarchy.

Privileges of child roles are inherited by parent roles.

System-defined Roles



ORGADMIN

ACCOUNTADMIN

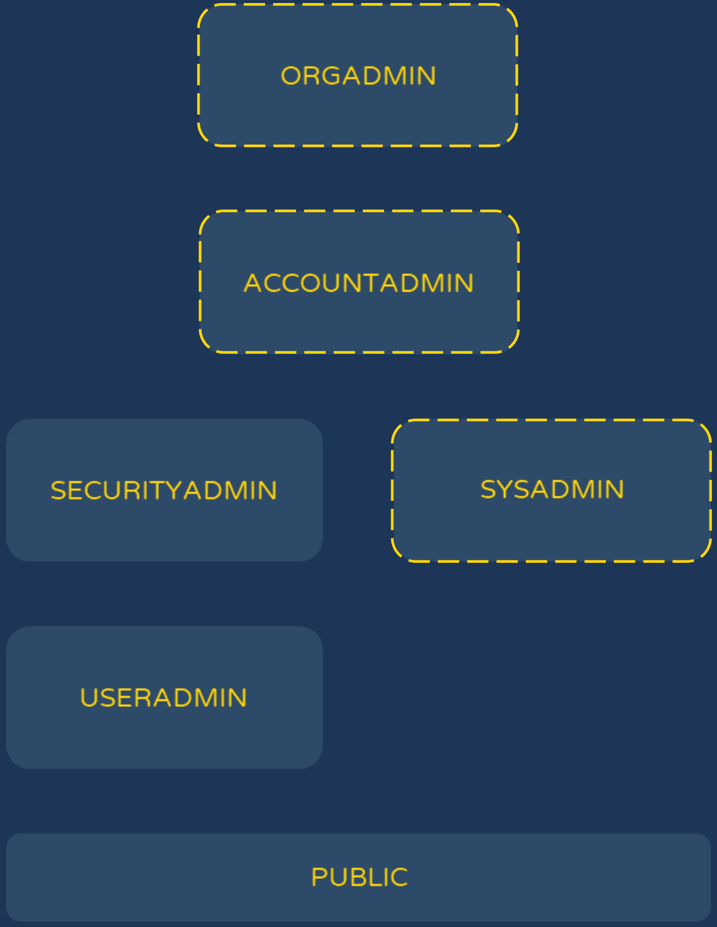
SECURITYADMIN

SYSADMIN

USERADMIN

PUBLIC

System-defined Roles



ORGADMIN

- Manages operations at organization level.
- Can create account in an organization.
- Can view all accounts in an organization.
- Can view usage information across an organization.

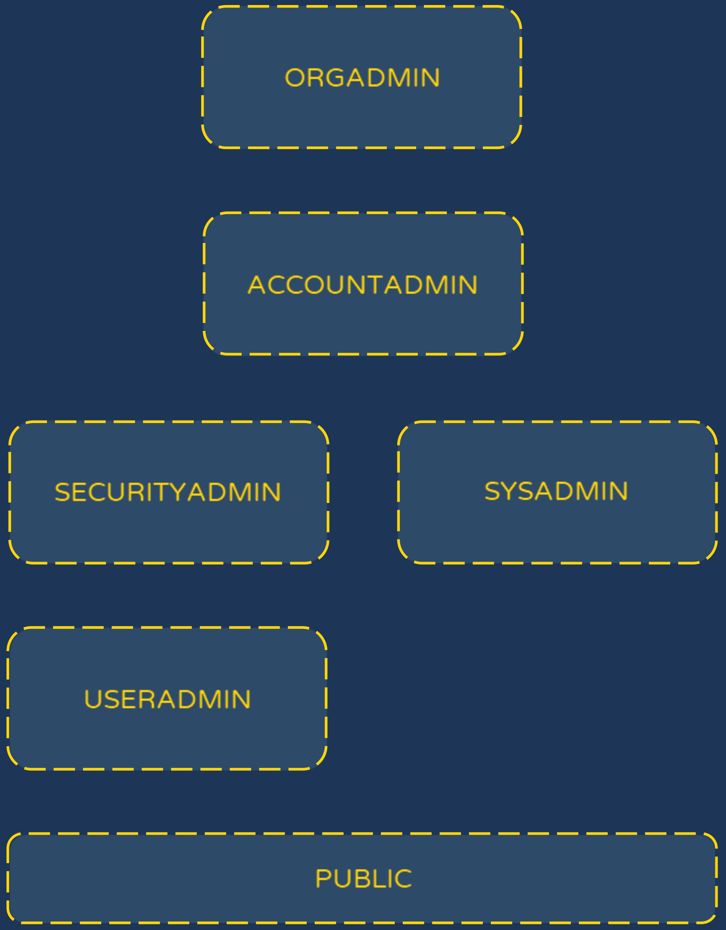
ACCOUNTADMIN

- Top-level and most powerful role for an account.
- Encapsulates SYSADMIN & SECURITYADMIN.
- Responsible for configuring account-level parameters.
- View and operate on all objects in an account.
- View and manage Snowflake billing and credit data.
- Stop any running SQL statements.

SYSADMIN

- Can create warehouses, databases, schemas and other objects in an account.

System-defined Roles



SECURITYADMIN

- Manage grants globally via the MANAGE GRANTS privilege.
- Create, monitor and manage users and roles.

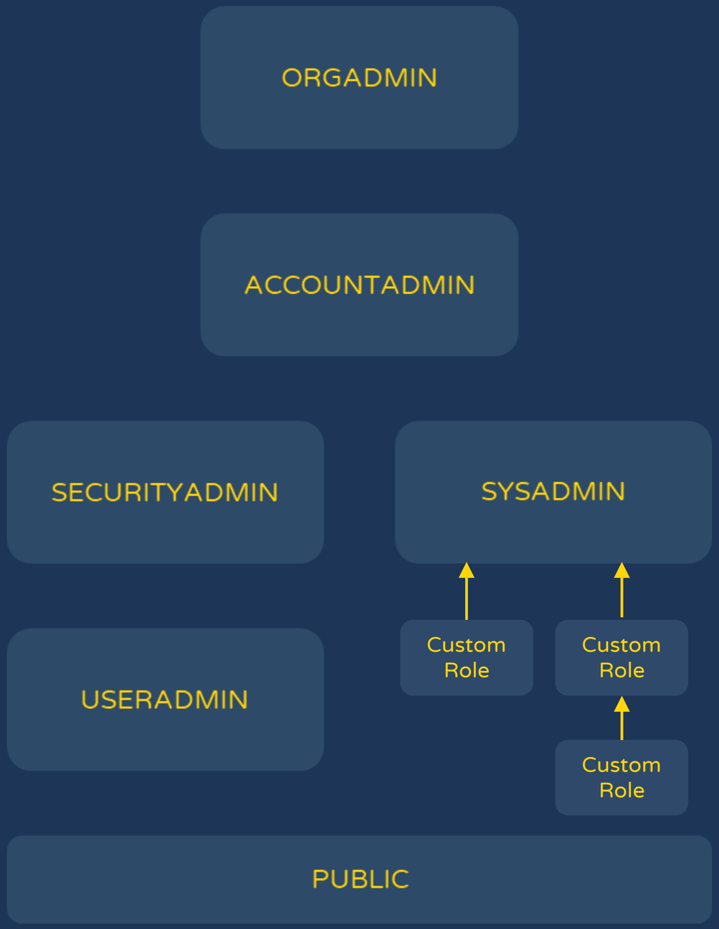
USERADMIN

- User and Role management via CREATE USER and CREATE ROLE security privileges.
- Can create users and roles in an account.

PUBLIC

- Automatically granted to every user and every role in an account.
- Can own securable objects, however objects owned by PUBLIC role are available to every other user and role in an account.

Custom Roles



Custom roles allows you to create a role with custom and fine-grained security privileges defined.

Custom roles allow administrators working with the system-defined roles to exercise the security principle of least privilege.

Custom roles can be created by the SECURITYADMIN & USERADMIN roles as well as by any role to which the CREATE ROLE privilege has been granted.

It is recommended to create a hierarchy of custom roles with the top-most custom role assigned to the SYSADMIN role.

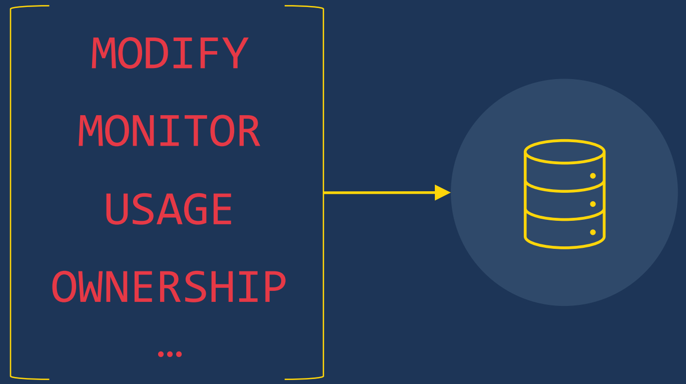
If custom roles are not assigned to the SYSADMIN role, system admins will not be able to manage the objects owned by the custom role.

Privileges

Privileges

A security privilege defines a level of access to an object.

For each object there is a set of security privileges that can be granted on it.



MODIFY

MONITOR

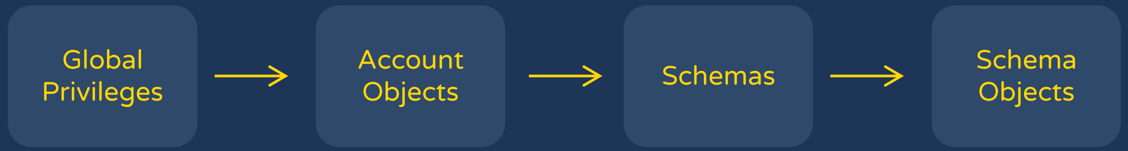
USAGE

OWNERSHIP

...

There are 4 categories of security privileges:

- Global Privileges
- Privileges for account objects
- Privileges for schemas
- Privileges for schema objects



Global Privileges → Account Objects → Schemas → Schema Objects

Privileges are managed using the GRANT and REVOKE commands.

```
GRANT USAGE ON DATABASE MY_DB TO ROLE MY_ROLE;
```

```
REVOKE USAGE ON DATABASE MY_DB TO ROLE MY_ROLE;
```

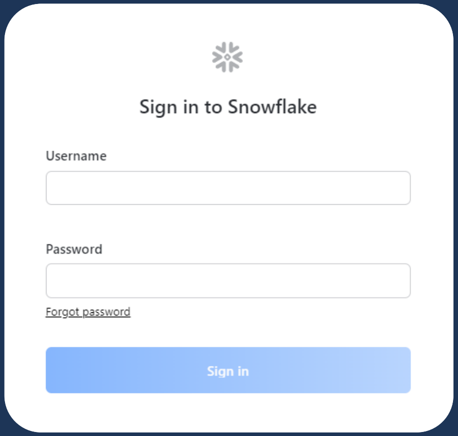
Future grants allow privileges to be defined for objects not yet created.

```
GRANT SELECT ON FUTURE TABLES IN SCHEMA MY_SCHEMA TO ROLE MY_ROLE;
```

User Authentication

User Authentication

User authentication is the process of authenticating with Snowflake via user provided username and password credentials.



User authentication is the default method of authentication.

Users with the USERADMIN role can create additional Snowflake users, which makes use of the CREATE USER privilege.

```
CREATE USER USER1
PASSWORD='ABC123’
DEFAULT_ROLE = MYROLE
MUST_CHANGE_PASSWORD = TRUE;
```

- A password can be any case-sensitive string up to 256 characters.
- Must be at least 8 characters long.
- Must contain at least 1 digit.
- Must contain at least 1 uppercase letter and 1 lowercase letter.

‘q@-*DaC2yjZoq3Re4JYX’

MFA

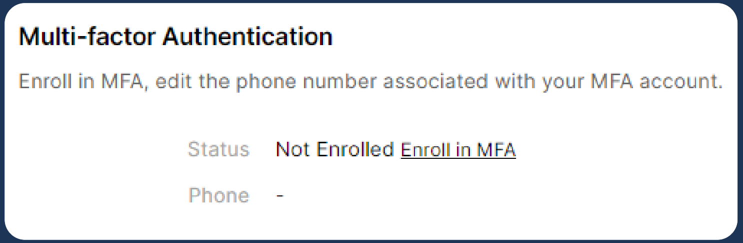
Multi-factor Authentication (MFA)

MFA is an additional layer of security, requiring the user to prove their identity not only with a password but with an additional piece of information (or factor).

x2

MFA in Snowflake is powered by a service called Duo Security.

MFA is enabled on a per-user basis & only via the UI.



Multi-factor Authentication

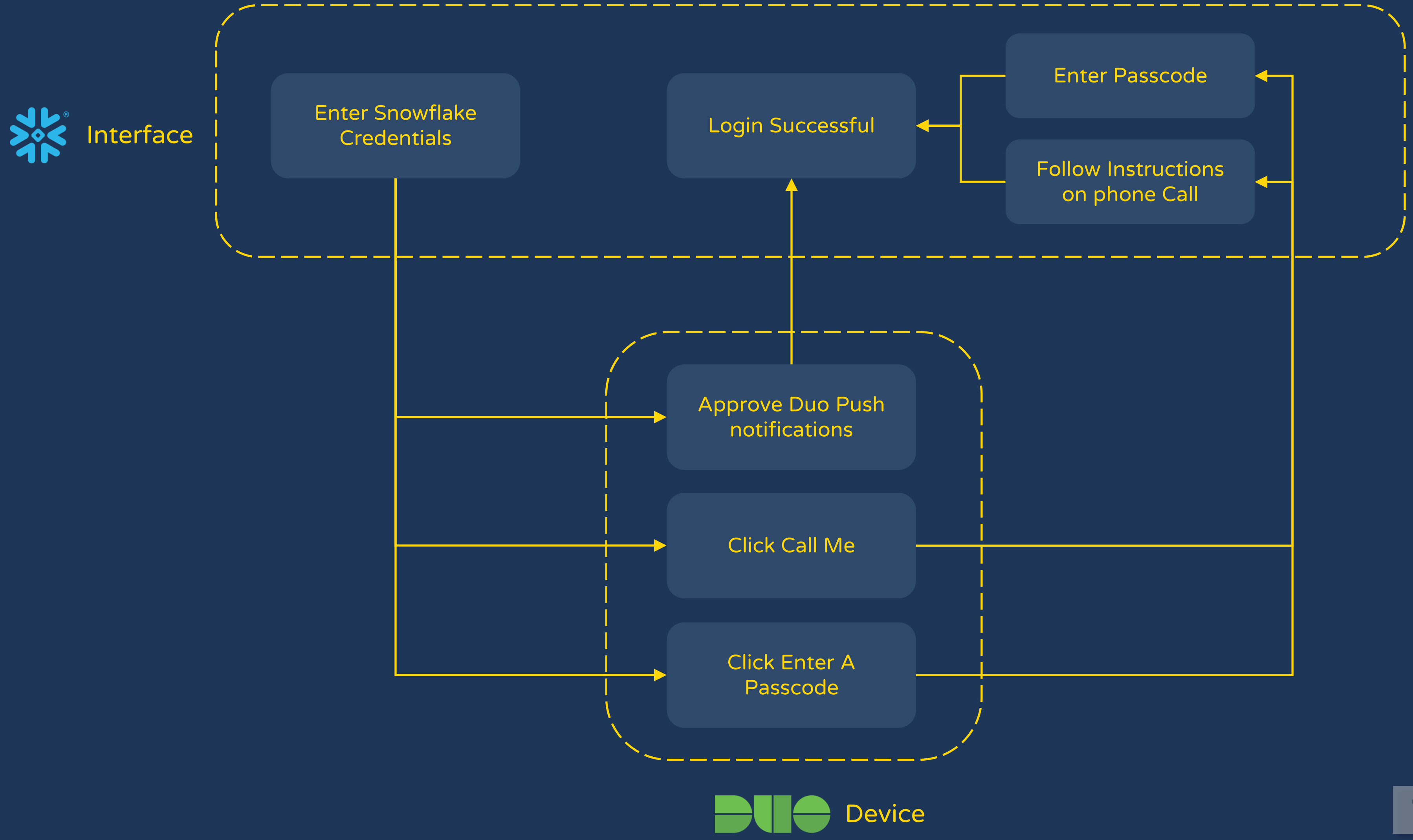
Enroll in MFA, edit the phone number associated with your MFA account.

Status	Not Enrolled Enroll in MFA
Phone	-

Snowflake recommend that all users with the ACCOUNTADMIN role be required to use MFA.

ACCOUNTADMIN

Multi-factor Authentication Flow



Interface
Enter Snowflake Credentials
Login Successful
Enter Passcode
Follow Instructions on phone Call
Approve Duo Push notifications
Click Call Me
Click Enter A Passcode
Device

MFA Properties

MINS_TO_BYPASS_MFA

```
ALTER USER USER1 SET
MINS_TO_BYPASS_MFA=10;
```

Specifies the number of minutes to temporarily disable MFA for the user so that they can log in.

DISABLE_MFA

```
ALTER USER USER1 SET
DISABLE_MFA=TRUE;
```

Disables MFA for the user, effectively cancelling their enrolment. To use MFA again, the user must re-enrol.

ALLOWS_CLIENT_MFA_CACHING

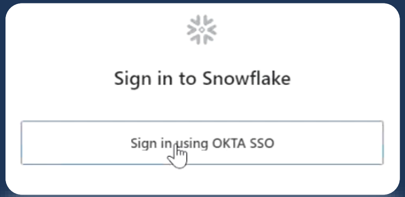
```
ALTER USER USER1 SET
ALLOWS_CLIENT_MFA_CACHING=TRUE;
```

MFA token caching reduces the number of prompts that must be acknowledged while connecting and authenticating to Snowflake.

Federated Authentication

Federated Authentication (SSO)

Federated authentication enables users to connect to Snowflake using secure SSO (single sign-on).



Sign in to Snowflake

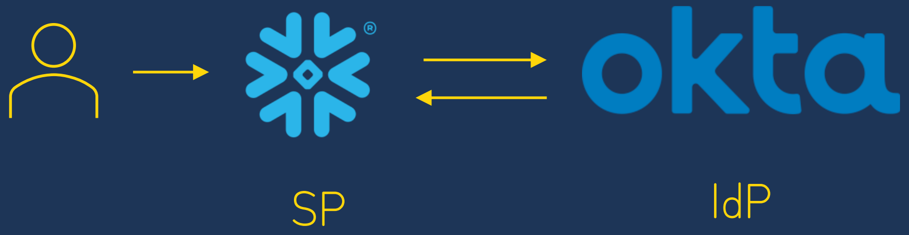
Sign in using OKTA SSO

Snowflake can delegate authentication responsibility to an SAML 2.0 compliant external identity provider (IdP) with native support for Okta and ADFS IdPs.

Microsoft

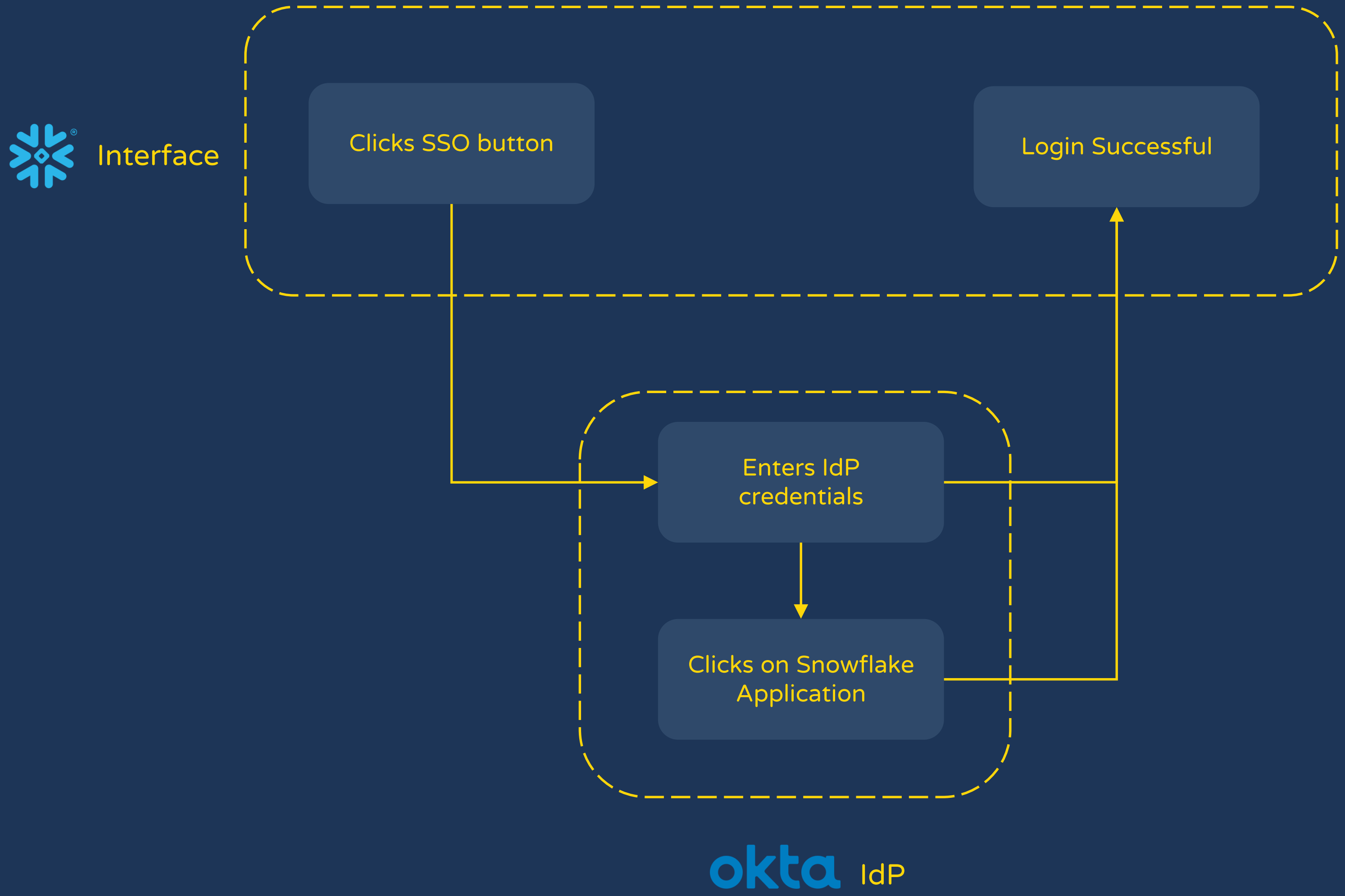
okta

An IdP is an independent service responsible for creating and maintaining user credentials as well as authenticating users for SSO access to Snowflake.



In a federated environment Snowflake is referred to as a Service Provider (SP).

Federated Authentication Login Flow



Federated Authentication Properties

SAML_IDENTITY_PROVIDER

```
ALTER ACCOUNT SET SAML_IDENTITY_PROVIDER =
'{
  "certificate": "XXXXXXXXXXXXXXXXXXX",
  "ssoUrl": "https://abccorp.testmachine.com/adfs/ls",
  "type"  : "ADFS",
  "label" : "ADFSSingleSignOn"
  }';
```

How to specify an IdP during the Snowflake setup of Federated Authentication.

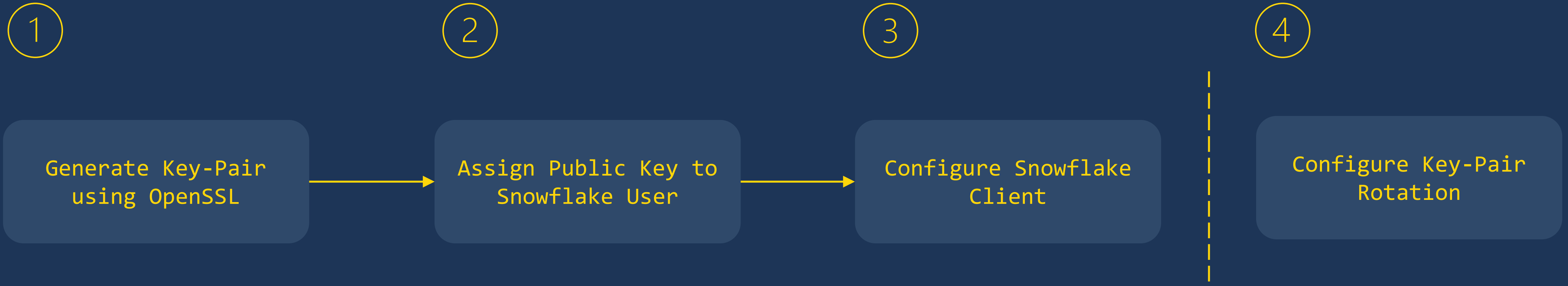
SSO_LOGIN_PAGE

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

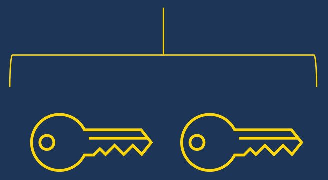
Enable button for Snowflake-initiated SSO for your identity provider (as specified in SAML_IDENTITY_PROVIDER) in the Snowflake main login page.

Key Pair Authentication, OAuth & SCIM

Key Pair Authentication


1
Generate Key-Pair using OpenSSL
2
Assign Public Key to Snowflake User
3
Configure Snowflake Client
4
Configure Key-Pair Rotation

2048-bit RSA key pair



```
ALTER USER USER1 SET
RSA_PUBLIC_KEY='MIIB%JA...';
```

- SnowSQL
- Python connector
- Spark connector
- Kafka connector
- Go driver
- JDBC driver
- ODBC driver
- .NET driver
- Node.js Driver

```
ALTER USER USER1 SET
RSA_PUBLIC_KEY_2=‘JER£E...';
```

```
ALTER USER USER1 UNSET
RSA_PUBLIC_KEY;
```

OAuth & SCIM

OAuth

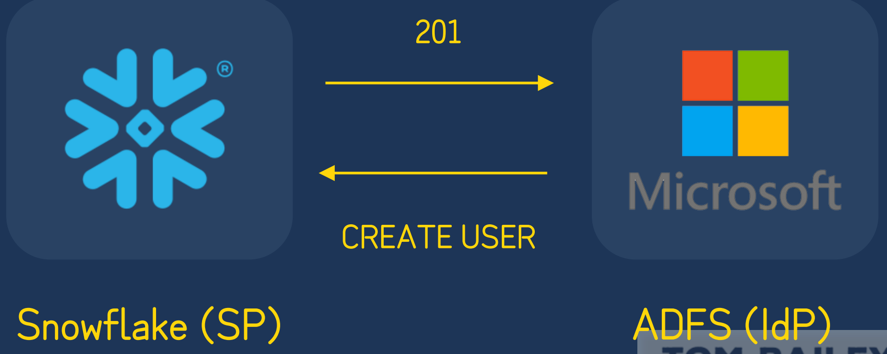
Snowflake supports the OAuth 2.0 protocol.

OAuth is an open-standard protocol that allows supported clients authorized access to Snowflake without sharing or storing user login credentials.

Snowflake offers two OAuth pathways: Snowflake OAuth and External OAuth.

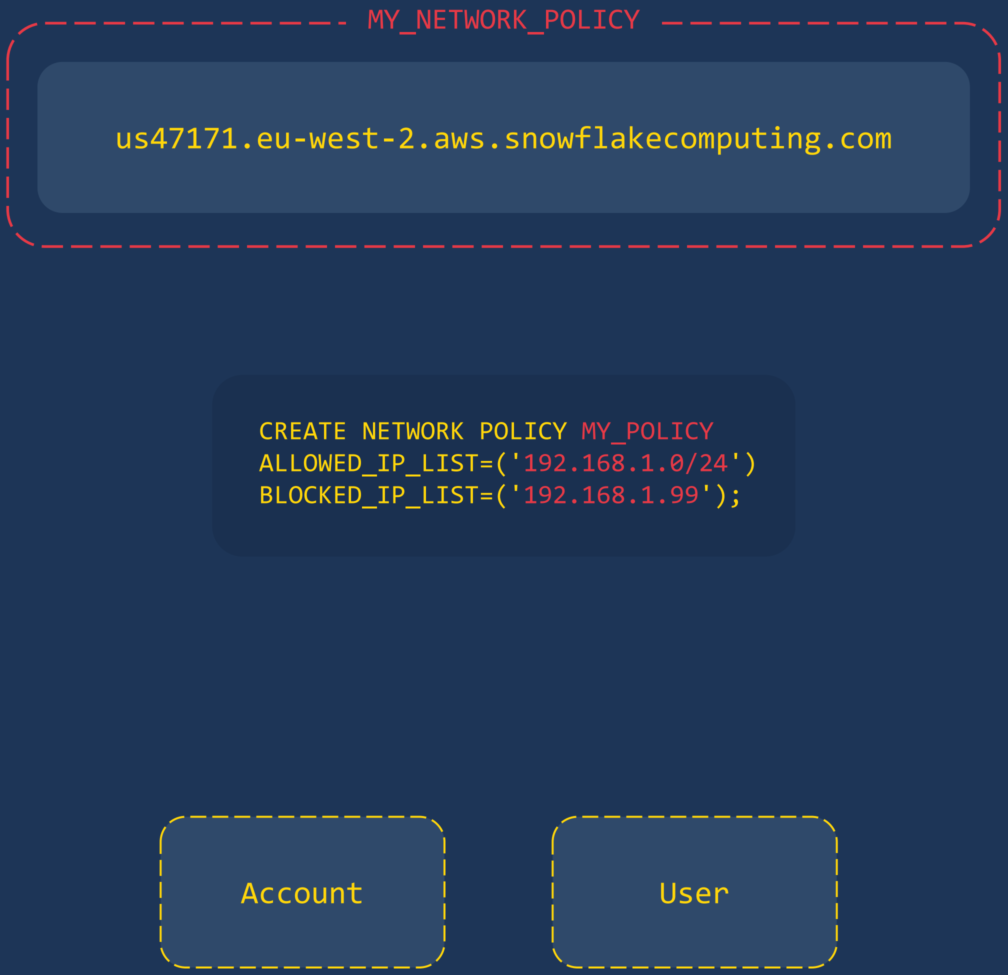
SCIM

System for Cross-domain Identity Management (SCIM) can be used to manage users and groups (Snowflake roles) in cloud applications using RESTful APIs.



Network Policies

Network Policies



```
CREATE NETWORK POLICY MY_POLICY
ALLOWED_IP_LIST=('192.168.1.0/24')
BLOCKED_IP_LIST=('192.168.1.99');
```

Network Policies provide the user with the ability to allow or deny access to their Snowflake account based on a single IP address or list of addresses.

Network Policies are composed of an allowed IP range and optionally a blocked IP range. Blocked IP ranges are applied first.

Network Policies currently support only IPv4 addresses.

Network policies use CIDR notation to express an IP subnet range.

Network Policies can be applied on the account level or to individual users.

If a user is associated to both an account-level and user-level network policy, the user-level policy takes precedence.

Network Policies

```
CREATE NETWORK POLICY MYPOLICY
ALLOWED_IP_LIST=('192.168.1.0/24')
BLOCKED_IP_LIST=('192.168.1.99');
```

```
SHOW NETWORK POLICIES;
```

ACCOUNT

Only one Network Policy can be associated with an account at any one time.

```
ALTER ACCOUNT SET NETWORK_POLICY = MYPOLICY;
```

SECURITYADMIN or ACCOUNTADMIN system roles can apply policies. Or custom role with the ATTACH POLICY global privilege.

```
SHOW PARAMETERS LIKE ‘MYPOLICY' IN ACCOUNT;
```

USER

Only one Network Policy can be associated with an user at any one time.

```
ALTER USER USER1 SET NETWORK_POLICY = MYPOLICY;
```

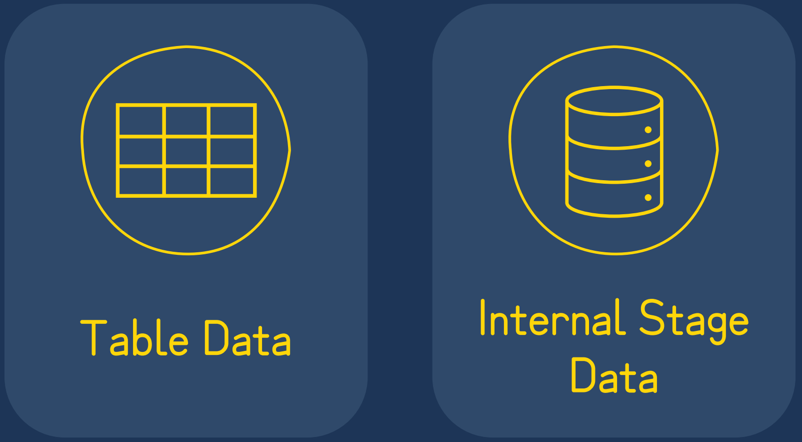
SECURITYADMIN or ACCOUNTADMIN system roles can apply policies. Or custom role with the ATTACH POLICY global privilege.

```
SHOW PARAMETERS LIKE ‘MYPOLICY' IN USER USER1;
```

Data Encryption

Data Encryption

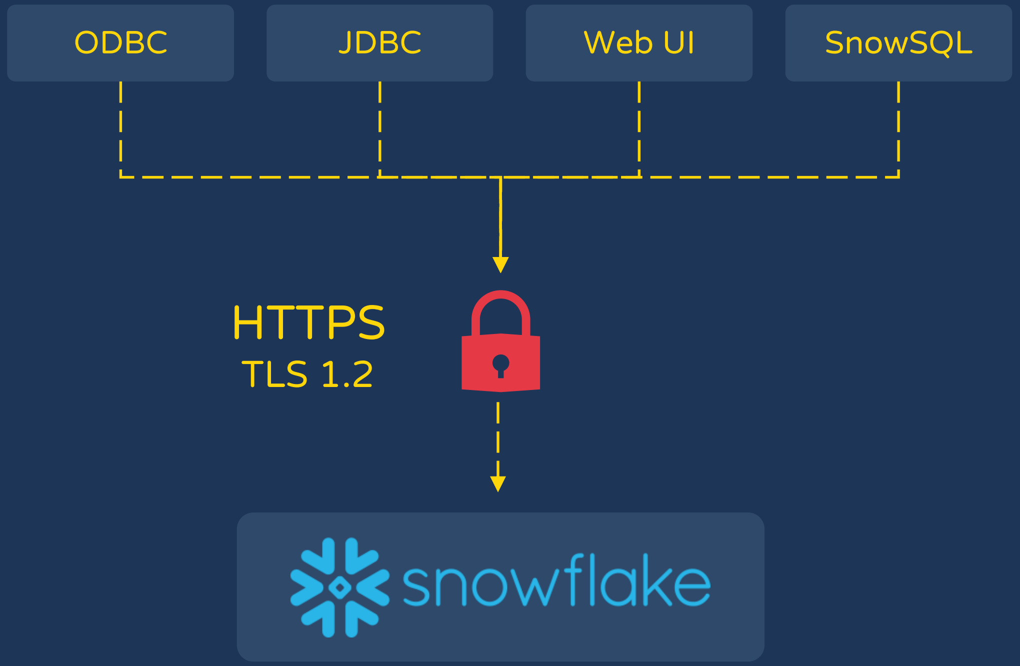
Encryption At Rest



AES-256 strong encryption

Virtual Warehouse and Query Result Caches

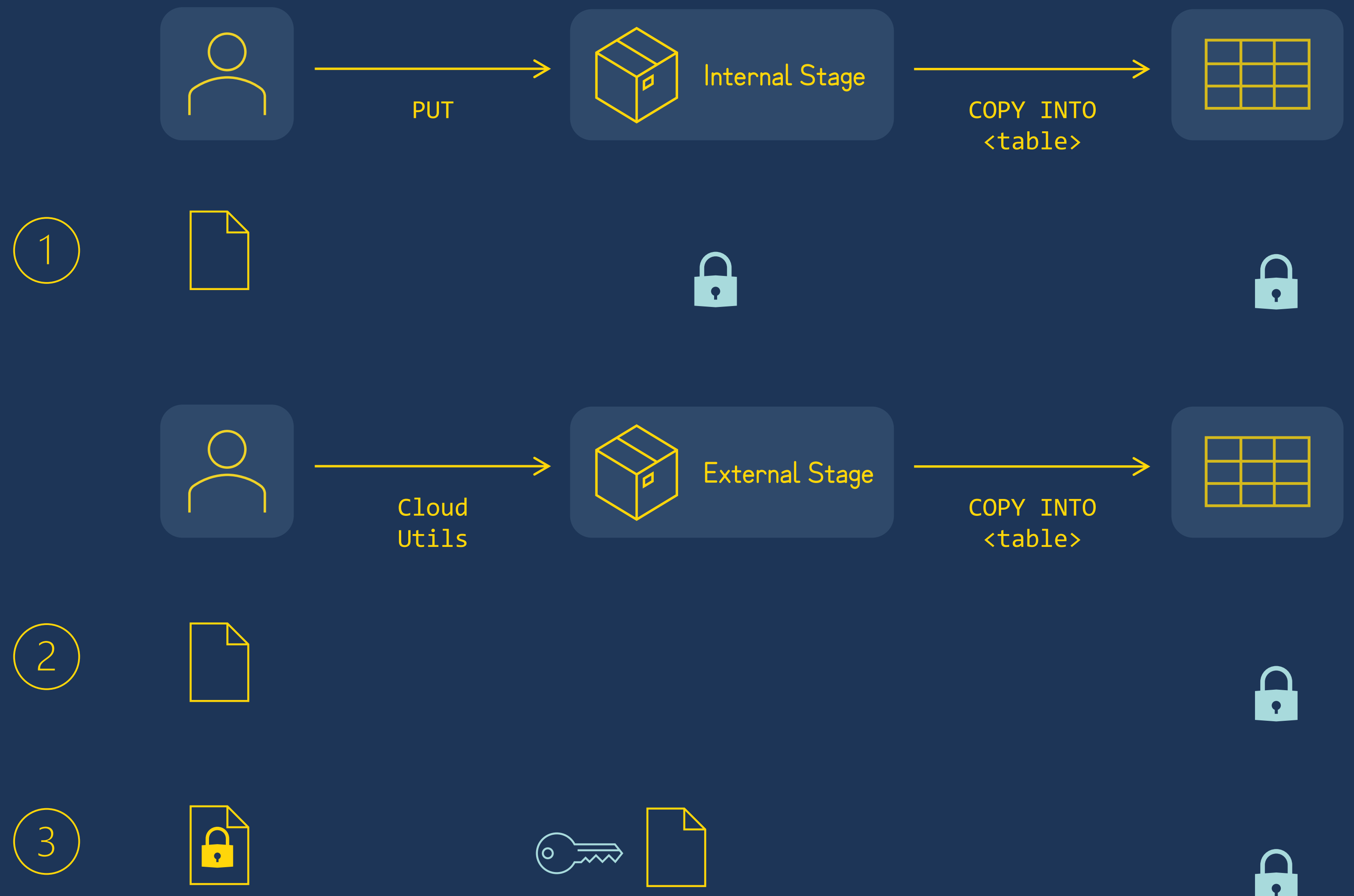
Encryption In Transit



TOM BAILEY COURSES

tombaileycourses.com

E2EE Encryption Flows



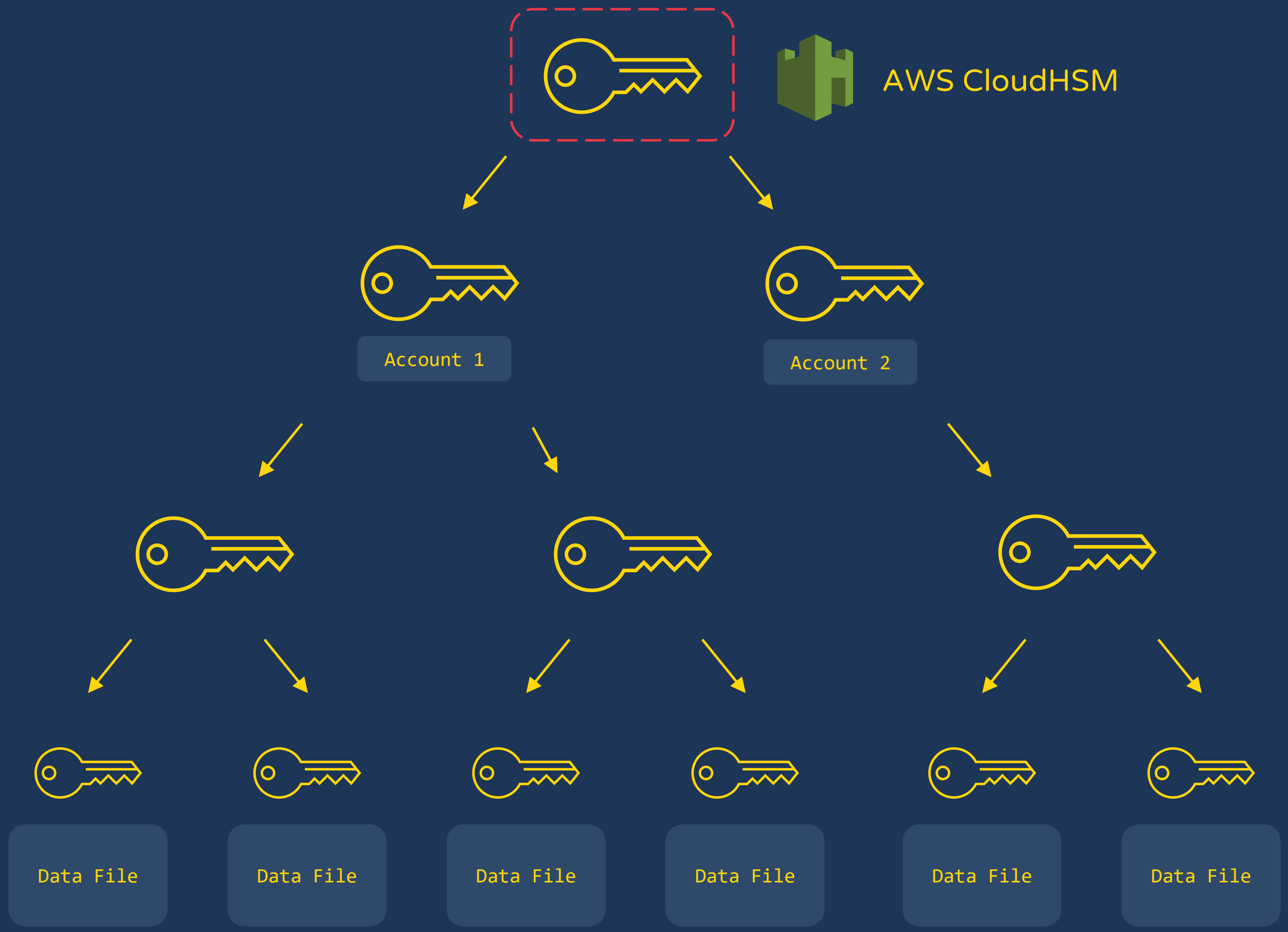
Hierarchical Key Model

Root Key

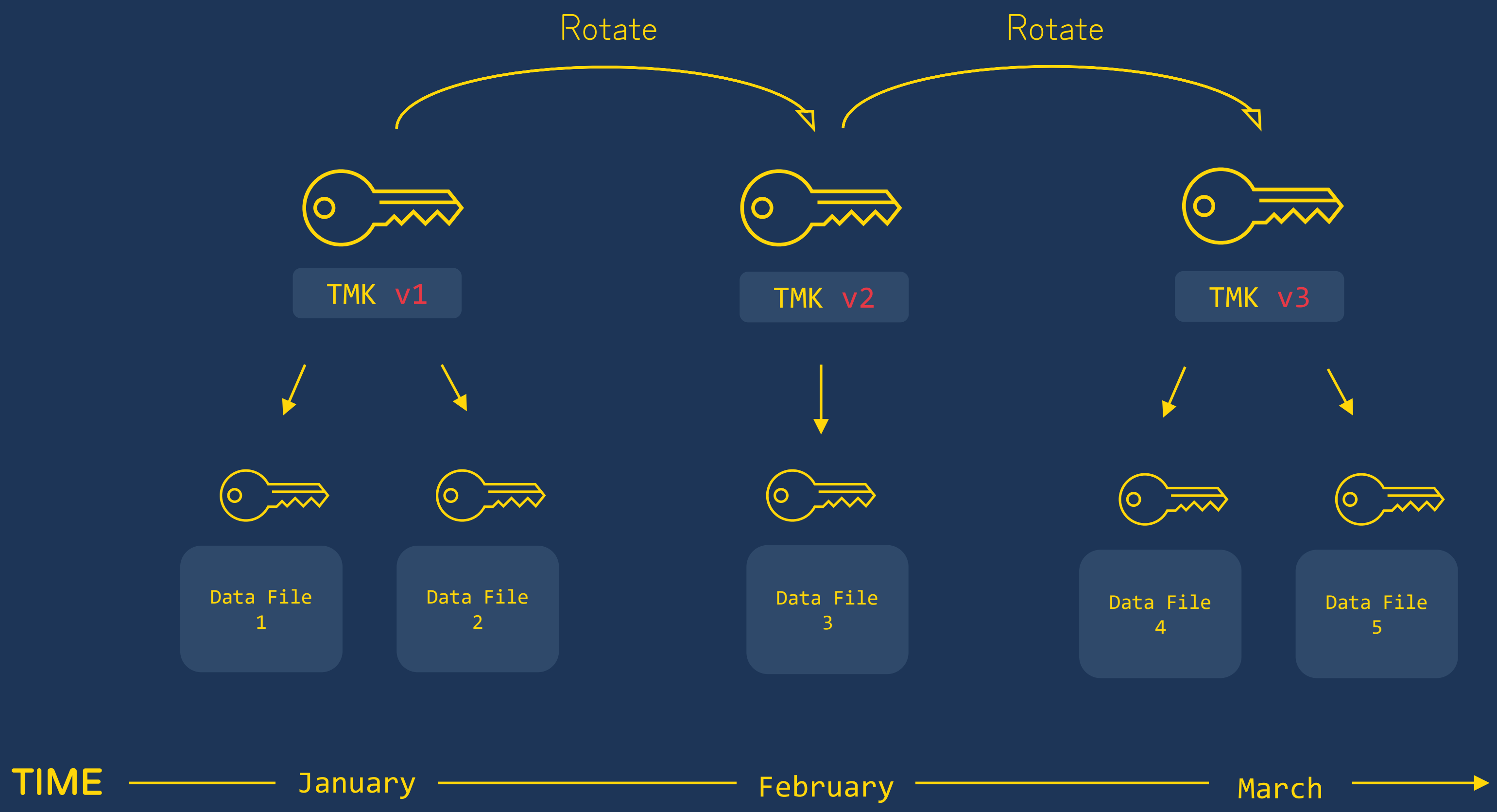
Account Master Keys

Table Master Keys

File Keys

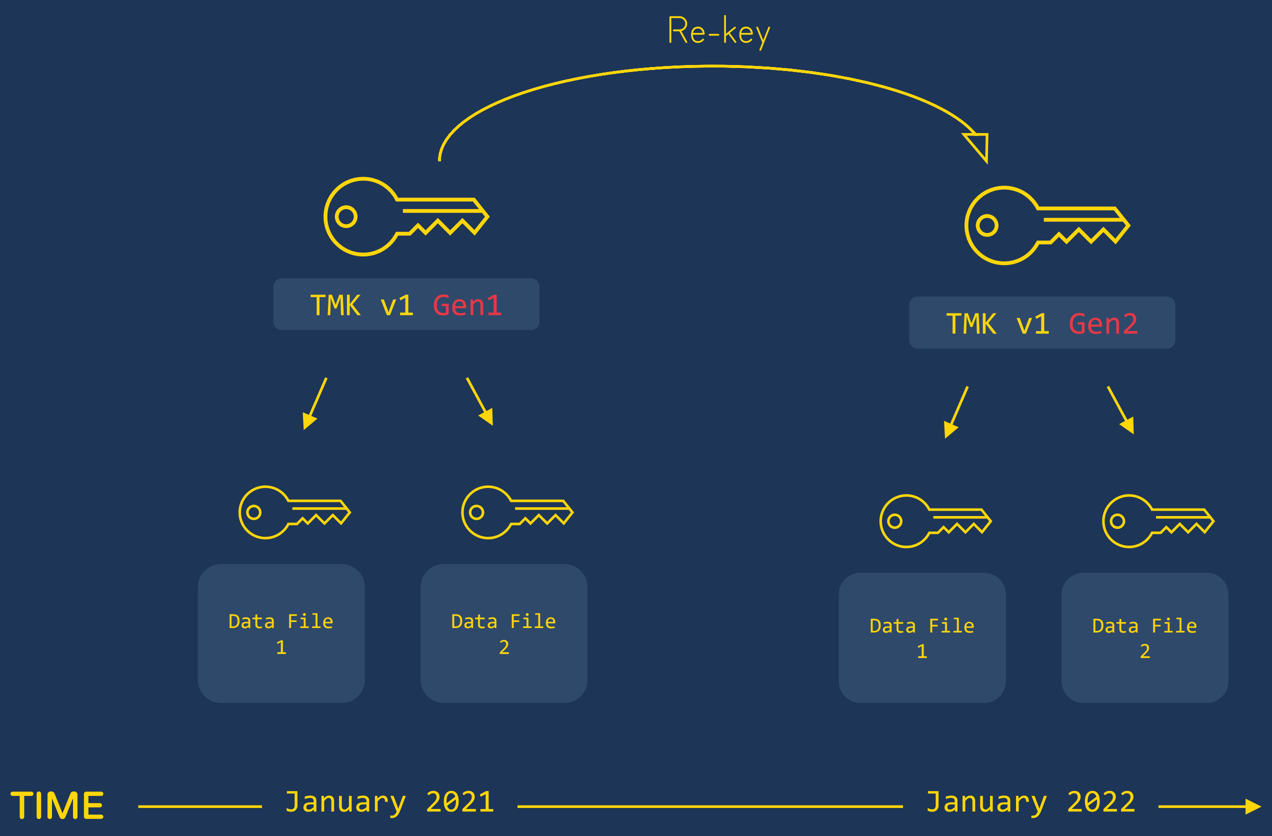


Key Rotation



Key rotation is the practise of transparently replacing existing account and table encryption keys every **30 days** with a new key.

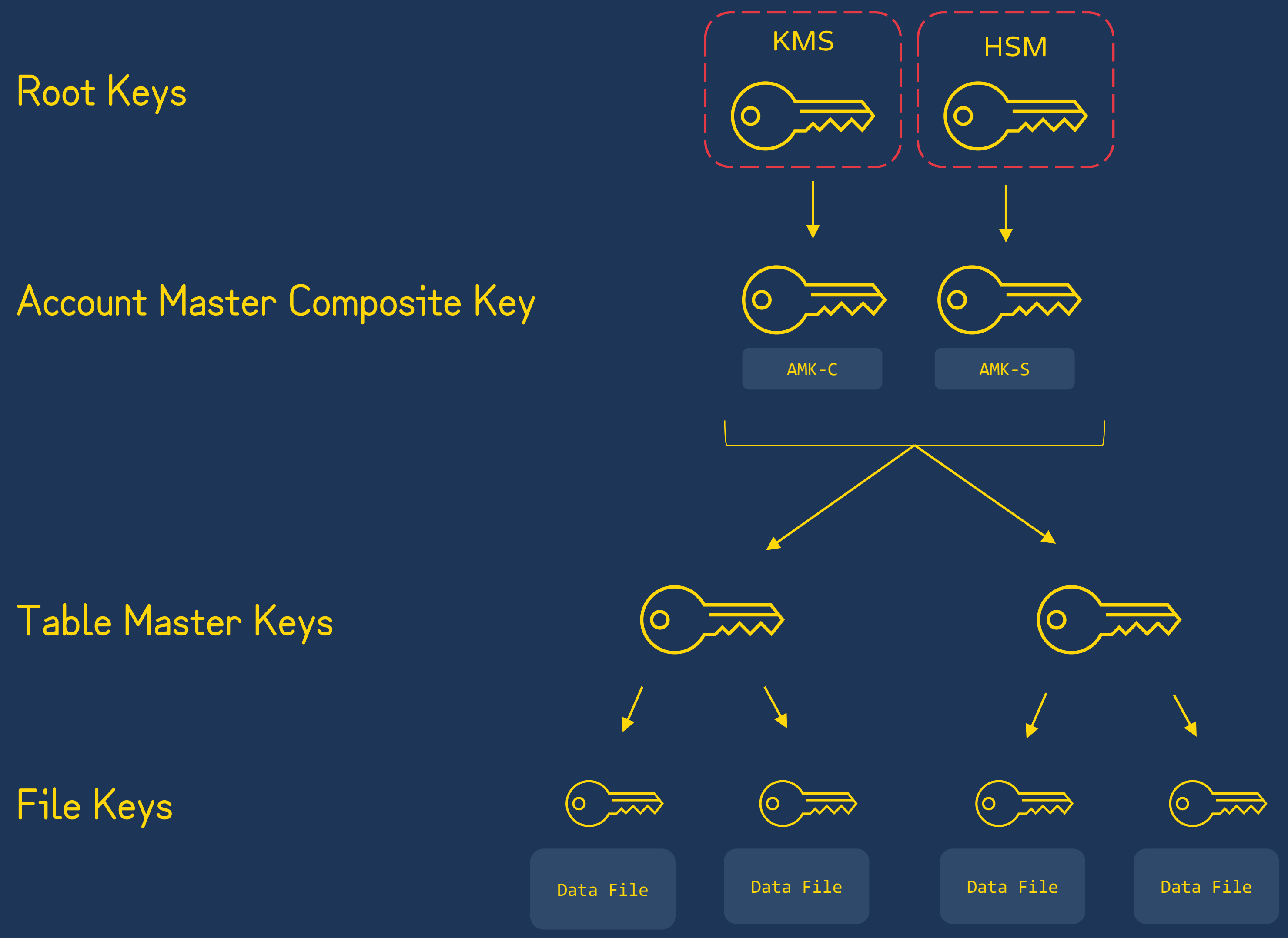
Re-Keying



Once a retired key exceeds **1 year**, Snowflake automatically creates a new encryption key and re-encrypts all data previously protected by the retired key using the new key.

```
ALTER ACCOUNT SET PERIODIC_DATA_REKEYING = TRUE;
```

Tri-secret Secure and Customer Managed Keys

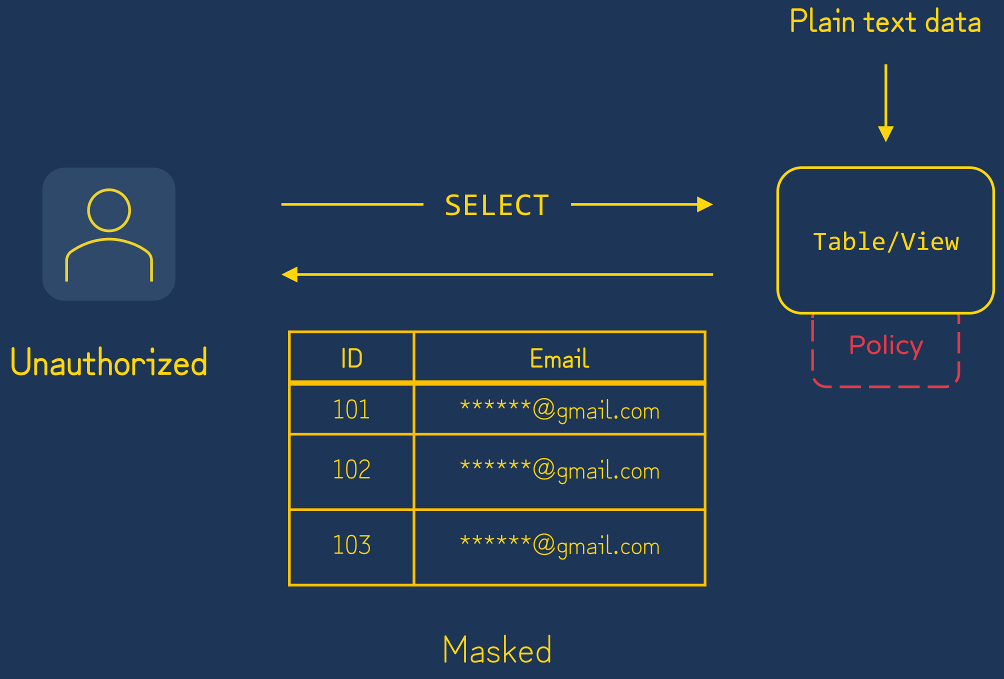


TOM BAILEY COURSES

tombaileycourses.com

Column Level security

Dynamic Data Masking

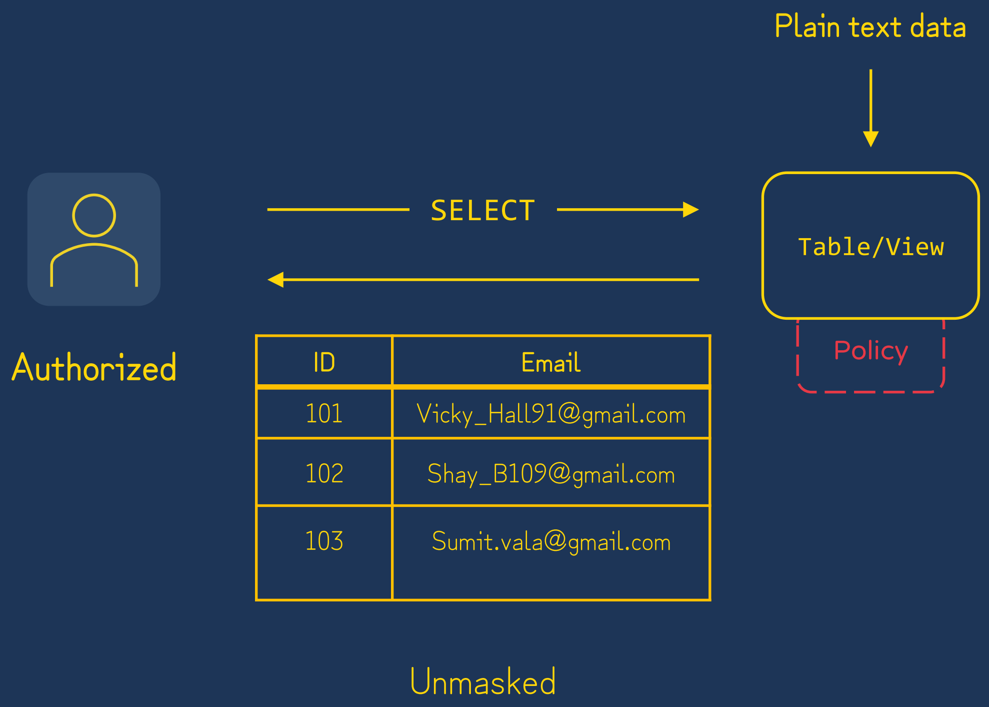


ID	Email
101	******@gmail.com
102	******@gmail.com
103	******@gmail.com

Masked

Sensitive data in plain text is loaded into Snowflake, and it is dynamically masked at the time of query for unauthorized users.

Dynamic Data Masking



ID	Email
101	Vicky_Hall91@gmail.com
102	Shay_B109@gmail.com
103	Sumit.vala@gmail.com

Sensitive data in plain text is loaded into Snowflake, and it is dynamically masked at the time of query for unauthorized users.

Masking Policies

Unmasked

Partially Masked

System Functions

User Defined Functions

Semi-structured

Fully Masked

```
CREATE MASKING POLICY EMAIL_MASK AS (VAL STRING) RETURNS STRING ->
  CASE
    WHEN CURRENT_ROLE() IN (‘SUPPORT’) THEN VAL
    WHEN CURRENT_ROLE() IN (‘ANALYST’) THEN REGEXP_REPLACE(VAL,'.+\@','*****@’)
    WHEN CURRENT_ROLE() IN (‘HR’) THEN SHA2(VAL)
    WHEN CURRENT_ROLE() IN (‘SALES’) THEN MASK_UDF(VAL)
    WHEN CURRENT_ROLE() IN (‘FINANCE’) THEN OBJECT_INSERT(VAL, 'USER_EMAIL', '*', TRUE)
  ELSE '********'
    END;
```

```
ALTER TABLE IF EXISTS EMP_INFO MODIFY COLUMN USER_EMAIL SET MASKING POLICY EMAIL_MASK;
```

TOM BAILEY COURSES

tombaileycourses.com

Masking Policies

Unmasked

Partially Masked

System Functions

User Defined Functions

Semi-structured

Fully Masked

```
CREATE MASKING POLICY EMAIL_MASK AS (VAL STRING) RETURNS STRING ->
  CASE
    WHEN CURRENT_ROLE() IN (‘SUPPORT’) THEN VAL
    WHEN CURRENT_ROLE() IN (‘ANALYST’) THEN REGEXP_REPLACE(VAL,'.+\@','*****@’)
    WHEN CURRENT_ROLE() IN (‘HR’) THEN SHA2(VAL)
    WHEN CURRENT_ROLE() IN (‘SALES’) THEN MASK_UDF(VAL)
    WHEN CURRENT_ROLE() IN (‘FINANCE’) THEN OBJECT_INSERT(VAL, 'USER_EMAIL', '*', TRUE)
  ELSE '********'
    END;
```

```
ALTER TABLE IF EXISTS EMP_INFO MODIFY COLUMN USER_EMAIL SET MASKING POLICY EMAIL_MASK;
```

Masking Policies

1

Data masking policies are schema-level objects, like tables & views.

2

Creating and applying data masking policies can be done independently of object owners.

3

Masking policies can be nested, existing in tables and views that reference those tables.

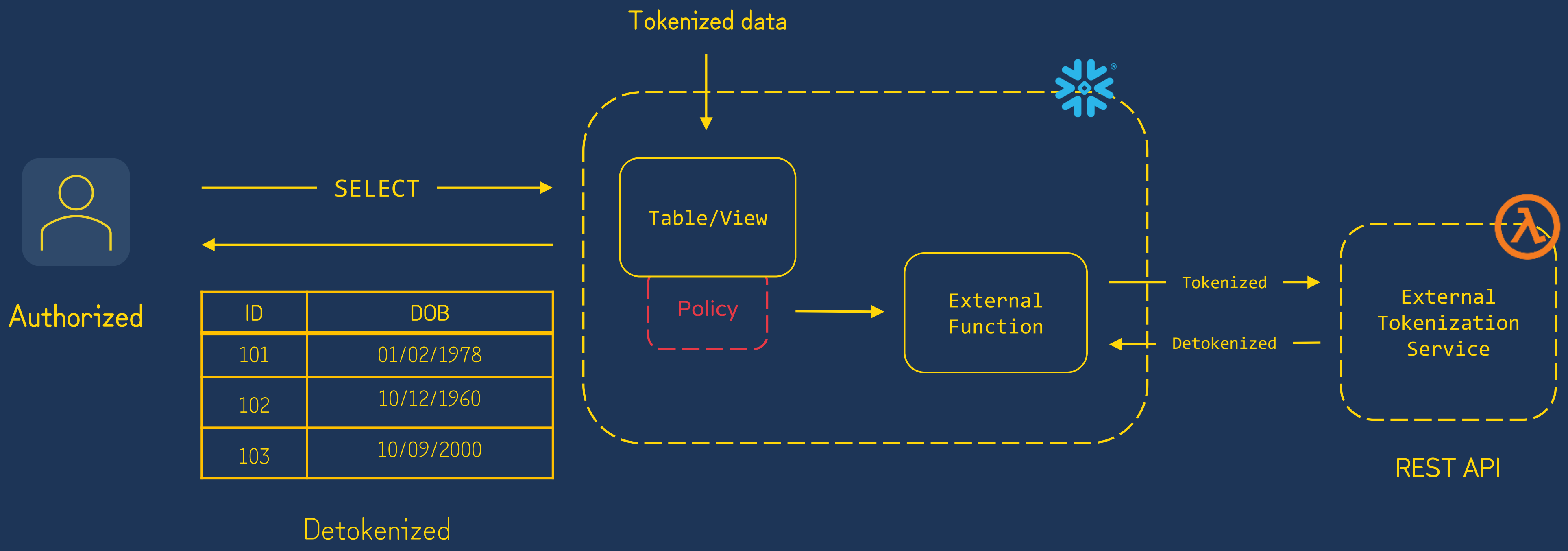
4

A masking policy is applied no matter where the column is referenced in a SQL statement.

5

A data masking policy can be applied either when the object is created or after the object is created.

External Tokenization

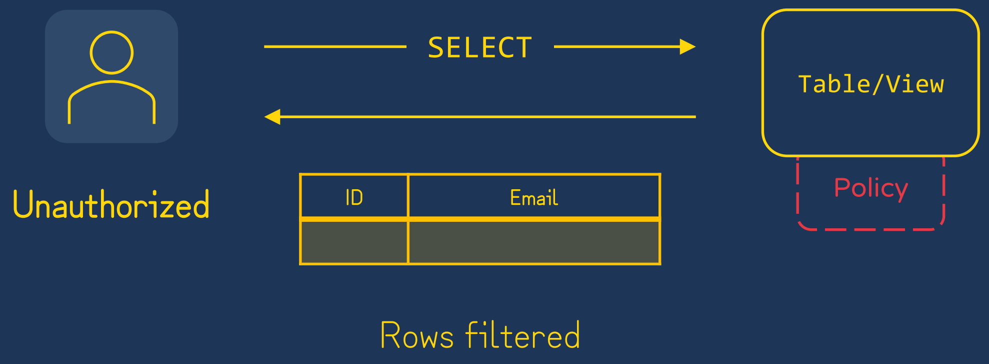


ID	DOB
101	01/02/1978
102	10/12/1960
103	10/09/2000

Tokenized data is loaded into Snowflake, which is detokenized at query run-time for authorized users via masking policies that call an external tokenization service using external functions.

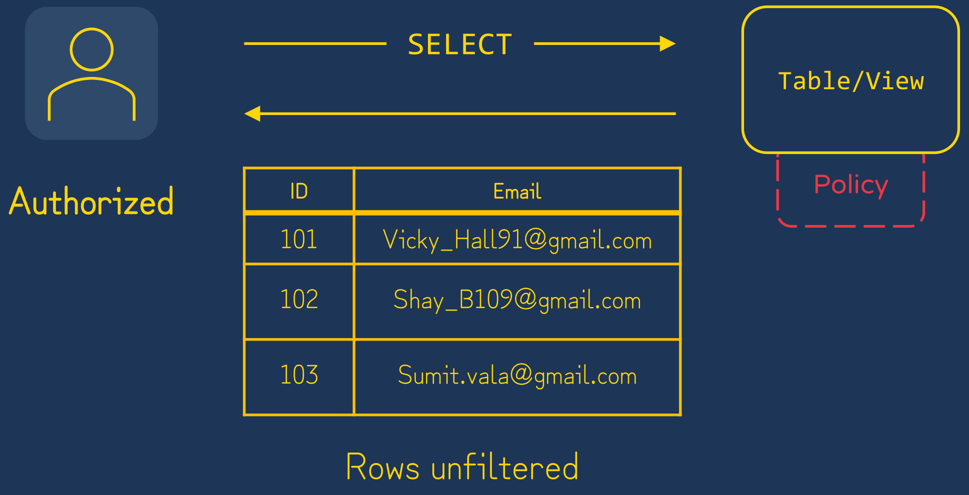
Row Level security

Row Access Policies



Row access policies enable a security team to restrict which rows are return in a query.

Row Access Policies



Authorized

SELECT

Table/View

Policy

ID	Email
101	Vicky_Hall91@gmail.com
102	Shay_B109@gmail.com
103	Sumit.vala@gmail.com

Rows unfiltered

Row access policies enable a security team to restrict which rows are return in a query.

Row Access Policies

```
CREATE OR REPLACE ROW ACCESS POLICY RAP_ID AS (ACC_ID VARCHAR) RETURNS BOOLEAN ->
  CASE
      WHEN 'ADMIN' = CURRENT_ROLE() THEN TRUE
      ELSE FALSE
  END;
```

```
ALTER TABLE ACCOUNTS ADD ROW ACCESS POLICY RAP_IT ON (ACC_ID);
```

Similarities

Schema level object

Segregation of duties

Creation and applying workflow

Nesting policies

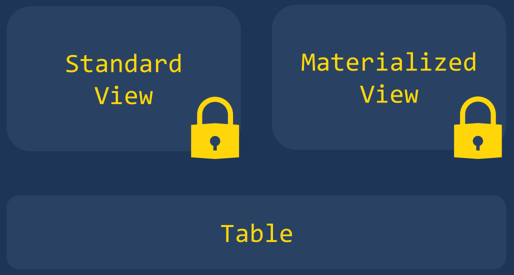
Adding a masking policy to a column fails if the column is referenced by a row access policy.

Row access policies are evaluated before data masking policies.

Secure Views

Secure Views

Secure views are a type of view designed to limit access to the underlying tables or internal structural details of a view.



Both standard and materialized views can be designated as secure.

A secure view is created by adding the keyword SECURE in the view DDL.

```
CREATE OR REPLACE SECURE VIEW MY_SEC_VIEW AS
SELECT COL1, COL2, COL3 FROM MY_TABLE;
```

The definition of a secure view is only available to the object owner.

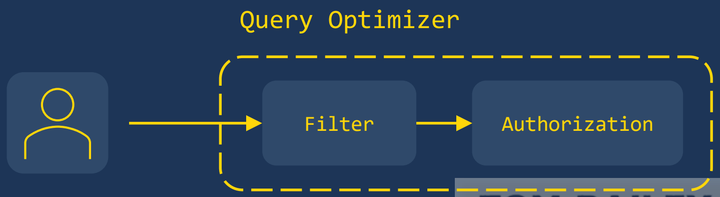
SHOW VIEWS;

GET_DDL();

Information Schema

Account Usage

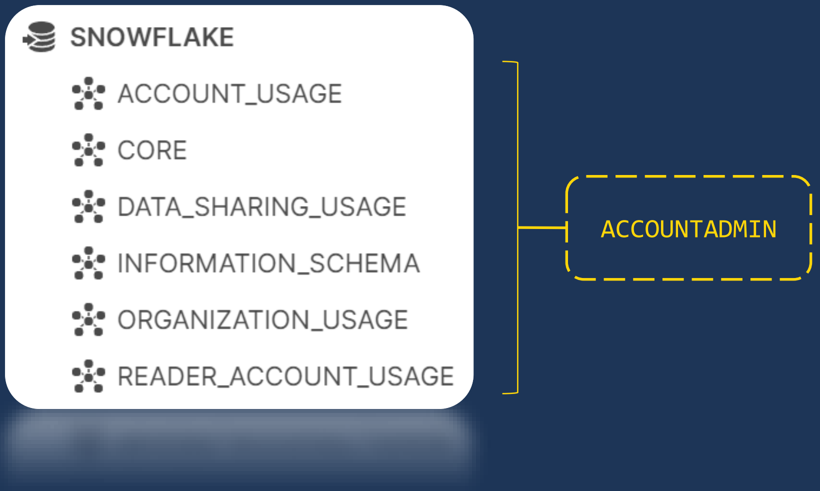
Secure views bypass query optimizations which may inadvertently expose data in the underlying table.



Account Usage and Information Schema

Account Usage

Snowflake provide a shared read-only databased called SNOWFLAKE, imported using a Share object called ACCOUNT_USAGE.



SNOWFLAKE
- ACCOUNT_USAGE
- CORE
- DATA_SHARING_USAGE
- INFORMATION_SCHEMA
- ORGANIZATION_USAGE
- READER_ACCOUNT_USAGE

ACCOUNTADMIN

It is comprised of 6 schemas, which contain many views providing fine-grained usage metrics at the account and object level.

By default, only users with the ACCOUNTADMIN role can access the SNOWFLAKE database.

Account usage views record dropped objects, not just those that are currently active.

```
SELECT * FROM "SNOWFLAKE"."ACCOUNT_USAGE"."TABLES";
```

TABLE_ID	DELETED
4	2022-12-03 09:08:35.765 -0800

There is latency between an event and when that event is recorded in an account usage view.

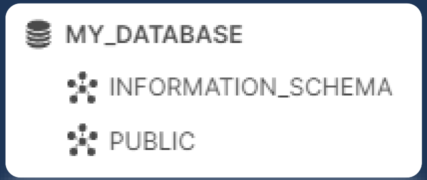
~ 2 Hours

Certain account usage views provide historical usage metrics. The retention period for these views is 1 year.

365 Days

Information Schema

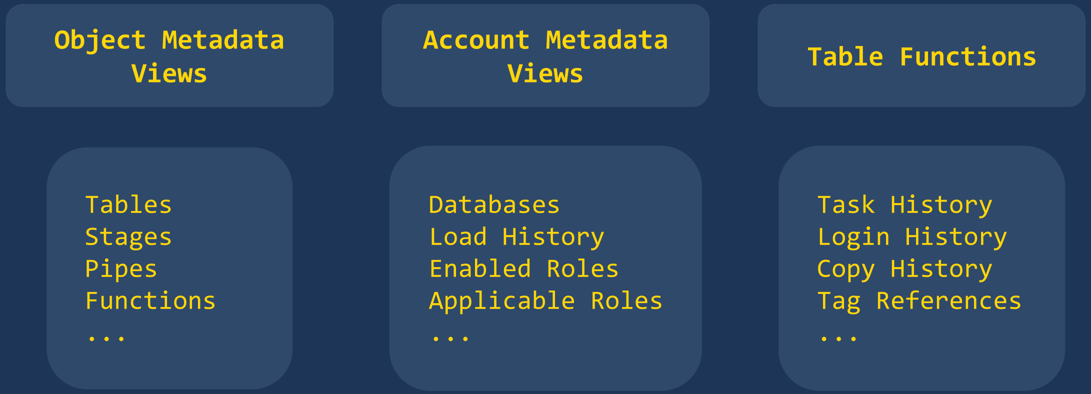
Each database created in an account automatically includes a built-in, read-only schema named INFORMATION_SCHEMA based on the SQL-92 ANSI Information Schema.



MY_DATABASE
INFORMATION_SCHEMA
PUBLIC

Each INFORMATION_SCHEMA contains:

- Views displaying metadata for all objects contained in the database.
- Views displaying metadata for account-level objects (non-database objects such as roles, warehouses and databases).
- Table functions displaying metadata for historical and usage data across an account.



Object Metadata Views	Account Metadata Views	Table Functions
Tables Stages Pipes Functions ...	Databases Load History Enabled Roles Applicable Roles ...	Task History Login History Copy History Tag References ...

The output of a view or table function depends on the privileges granted to the user’s current role.

Account Usage vs. Information Schema

Property	Account Usage	Information Schema
Includes dropped objects	Yes	No
Latency of data	From 45 minutes to 3 hours (varies by view)	None
Retention of historical data	1 Year	From 7 days to 6 months (varies by view/table function)

TOM BAILEY COURSES

tombaileycourses.com