SNOWFLAKE BUSINESS ANALYST WORKSHOP

State of Colorado

October 2025

Agenda

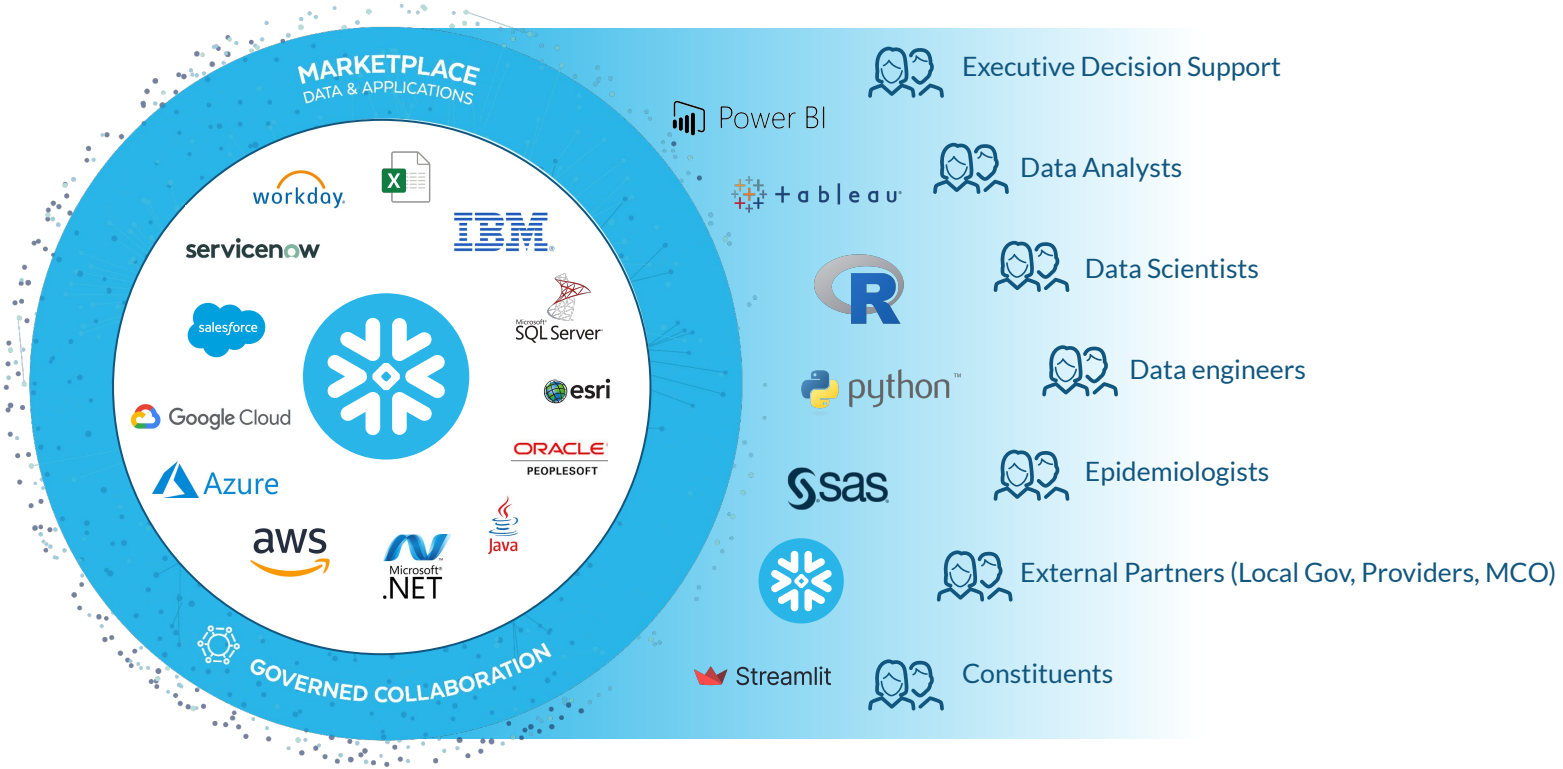
1. Snowflake Overview
2. Hands-on Lab: Data Exploration, Querying, & Analytics
3. Q&A and Wrap-up

Snowflake Overview

1:00 PM - 1:30 PM

A Single Source of Truth

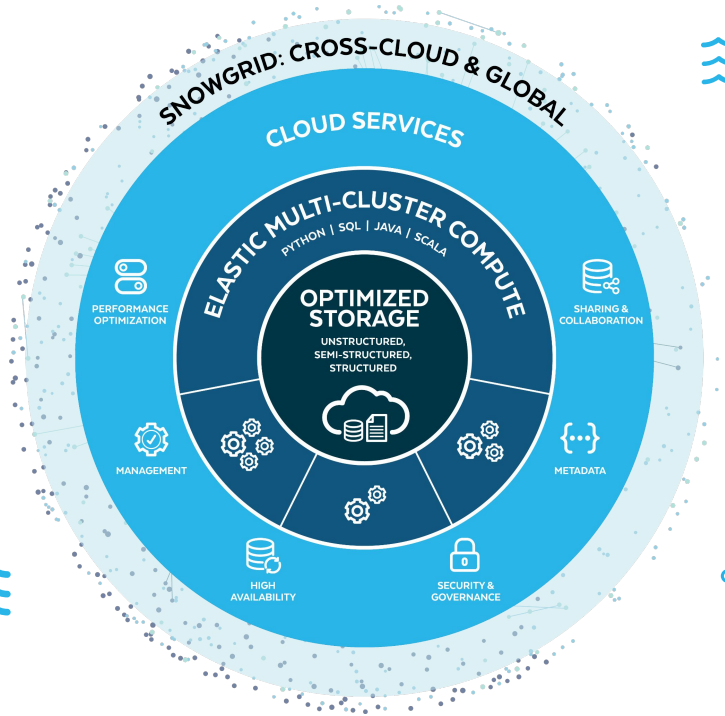
Shatter data silos and democratize insights to every team



One Platform. Any Architectural Pattern.

Data Warehouse
Centralized analytics for structured business data

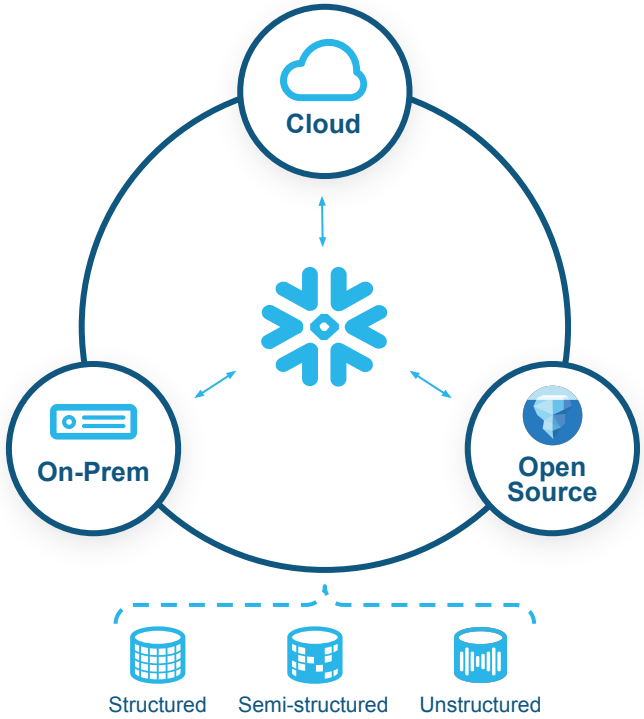
Data Lake
Unlimited storage for versatile data types and workloads



Data Lakehouse
Transactional data lake for unified analytics, AI/ML, collaborative workloads

Data Mesh / Fabric
Distributed and governed, domain-oriented collaboration

Optimized Storage



Un-siloed Access to Your Data

Unstructured, semi-structured, and structured data together with near-infinite scale.

Easily Manage Data at Scale

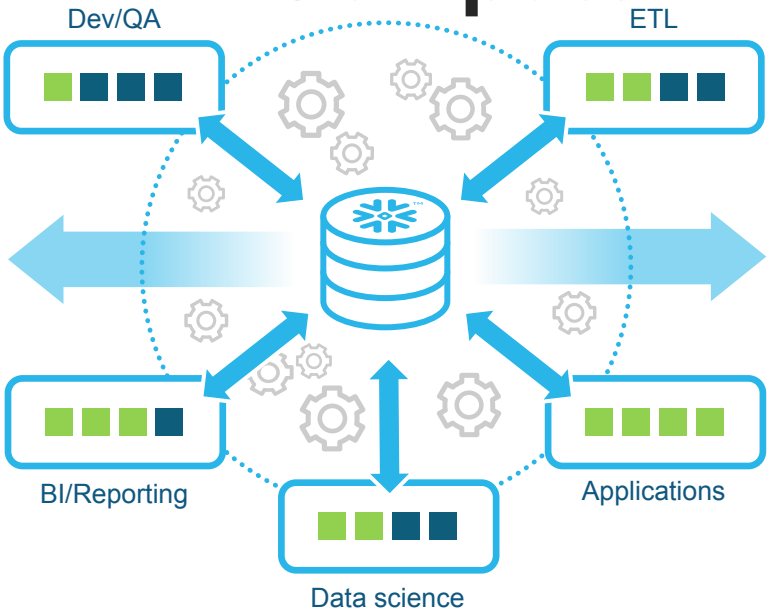
Fast and efficient access, optimized compression, and secure data - all automated.

Flexibility & Interoperability

Work with data on-premises or in open table formats* to remove lock-in and adapt to any architectural pattern.

* in public preview

Elastic Multi-Cluster Compute



One Engine for Every Workload

Power streaming pipelines, analytics, AI, interactive apps, and more through a single engine with flexible compute options including GPUs.

Leading Performance & Concurrency

Fast, reliable performance for virtually all users and jobs with no tuning or contention, delivered through isolated compute.

Familiar Languages

Work in SQL, Python, or Java. Run your preferred libraries with Snowpark. All without moving data.

Flexible Development

Use native development interfaces, bring your own IDE, or leverage popular third-party tools.

Cloud Services

Snowflake Managed

- Governance Controls
- AI Models & LLMs
- Maintenance & Tuning
- Performance Enhancements
- Networking & Encryption
- Administration & Availability

Fully Managed Service

Automate costly and complex operations to reduce overheard and improve efficiency. Transparent performance enhancements automatically applied via releases.

Unified, Built-In Governance

Snowflake Horizon delivers unified compliance, security, privacy, interoperability, and access capabilities, without additional configurations or protocols.

Accessible, Built-In AI

Snowflake Cortex* enables secure access to industry-leading AI models, LLMs, and vector search functionality, with no infrastructure to manage.

* in private preview

Near-Zero Management - It Just Works

	Task	Ordinary on-prem DW/DL	Ordinary cloud DW/DL	Snowflake
Infrastructure	Hardware	You	Cloud vendor	Built-in
	Software	Distribution	Cloud vendor	Built-in
	Hardware cluster setup	You	You	Built-in
	Software provisioning	Tools	Tools	Built-in
Data & service protection	Data protection & retention	Platform or Add-on	Platform or Add-on	Cloud Data Platform as a Service
	Node failure protection & recovery	You	You	
	Disaster recovery	You	You	
	Service monitoring & alerting	You	You	
Security	Physical security	You	Cloud vendor	
	Deployment security	You	You	
	Security monitoring	You	You	
Database management & tuning	Compute scaling	You	You	
	Index management	You	You	
	Data partitioning	You	You	
	Metadata & statistics management	You	You	
	Query optimization	You	You	

- NO Infrastructure
- NO Tuning
- NO Optimization
- NO Indexing
- NO Storage Worries
- NO Vacuuming
- NO Partitioning
- NO Required Sorting
- NO Workload Mgmt.
- NO Manual Backups

Protecting Your Data in Snowflake

End-to-End Encryption

Always-encrypted client communications, plus integration with cloud provider private networking

Fully Encrypted Storage

Data at rest is always encrypted while handled by the Snowflake drivers and systems

Strong Authentication

Built in multi-factor, integration with your federated SSO, easy user management

Full Auditing

Track every login, every transaction, every data transfer, and export to your security tools

Role-Based Access Control

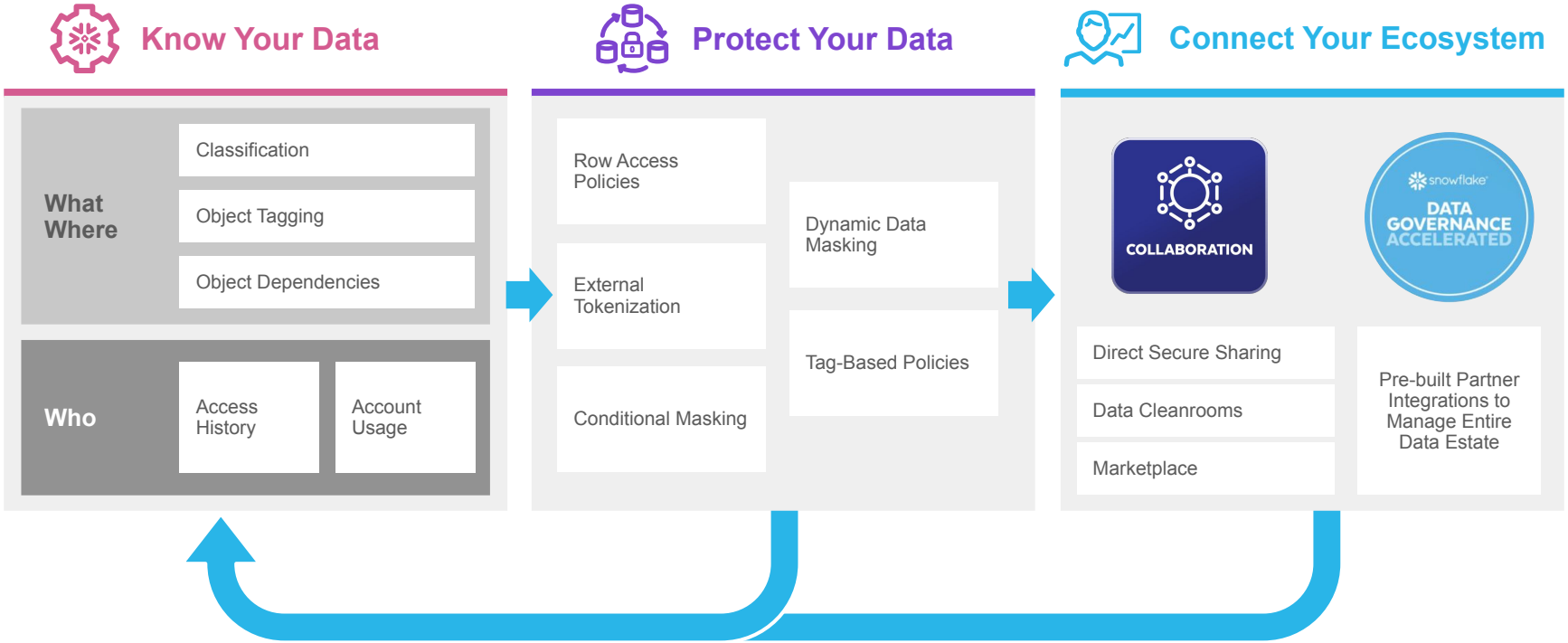
All objects, actions, and even compute usage can be controlled with roles

Recovery

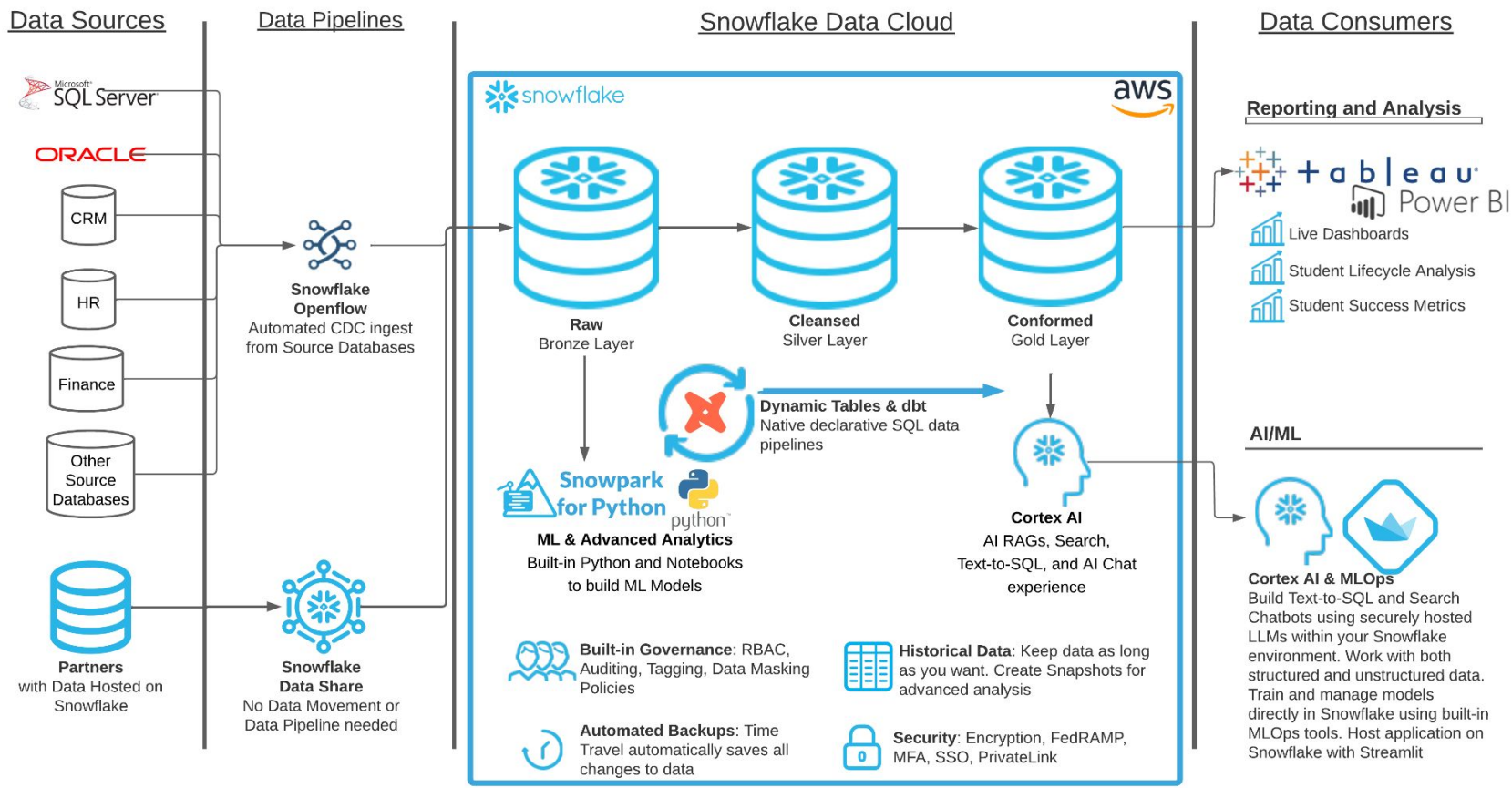
We give you options to ensure your data can be recovered in case of an accident or worse

[Snowflake Security Product Documentation]

UNIFIED GOVERNANCE



Example Data Architecture



Hands-on Lab: Data Exploration, Querying, & Analytics

1:30 PM - 3:30 PM

The Fundamentals: Getting Started with Data

What is a Relational Database?

- Collection of data that organizes data in tables and maintains relationships between the tables for information retrieval
- Enterprise RDBMS include:
  - Microsoft SQL Server
  - Oracle Database
  - Oracle MySQL
  - IBM DB2
  - SAP Sybase
  - Teradata
  - PostgreSQL
  - **SNOWFLAKE!!!**

Key Terms

- A **table** is the basic unit of data storage in a database. Data is stored in rows and columns. You can specify rules for each column of a table like establishing its data type, forcing a column to contain a value in every row, or ensuring unique records
- A **view** is a tailored presentation of data selected from one or more tables, possibly including other views. A view contains no actual data but rather derives what it shows from the tables and views on which it is based. Therefore, a view can be thought of as a virtual table
- A **schema** is like a folder within a database similar to a directory on a file system that groups tables together
- The **join** between tables is made by a **Primary Key – Foreign Key** pair, where the Foreign Key in a table is the Primary Key of another table
- A **query** is used to create, manipulate, or view database objects
- **SQL** stands for Structured Query Language as is sometimes pronounced “sequel”

SELECT

- A SELECT statement is used to fetch records from a database table in the form of a table called a result set
- One or more columns may be specified, or the asterisk (*) can be used to return all records from the table
- Use the LIMIT keyword to return only the first n rows from the result set
- The result set is not stored in the database without the use of additional functions

```
SELECT column1, column2, column3 FROM table_name
SELECT * FROM table_name
SELECT column1, column2, column3 FROM table_name LIMIT 5
```

LIMIT

- The **LIMIT** keyword can be added to a **SELECT** statement to limit the number of records that are returned
- This is a best practice when developing and testing out queries to have quicker return times and to avoid higher compute costs
- Remember to remove the **LIMIT** and try the queries completely before productionizing them

```
SELECT column1, column2, column3 FROM table_name LIMIT 100
```

WHERE

- The **WHERE** clause can be added to a **SELECT** statement to filter records that meet certain conditions, i.e. return a subset of the table
- We use expressions to define these conditions
  - Most conditional operators are self-explanatory (=, <, >, etc.)
  - A common operator for “not equals” is <> or !=

```
SELECT column1, column2, column3 FROM table_name WHERE column1 = ‘ABC’
SELECT column1, column2, column3 FROM table_name WHERE column2 > 100
SELECT column1, column2, column3 FROM table_name WHERE column3 <> 0
```

AND/OR

- An **AND/OR** operator can be added to a **WHERE** clause to filter records on based on multiple conditions
- The **AND** operator returns all records where both conditions are true
- The **OR** operator returns all records if either the first condition or the second condition are true

```
SELECT column1, column2, column3 FROM table_name
WHERE a_condition AND another_condition;
```

```
SELECT column1, column2, column3 FROM table_name
WHERE a_condition OR another_condition;
```

IN

- The **IN** operator can be used to specify multiple values in a **WHERE** clause
- This is equivalent to a series of **OR** clauses

```
SELECT column1, column2, column3 FROM table_name
WHERE column1 IN (value1, value2, …, valueN)
SELECT column1, column2, column3 FROM table_name
WHERE column1 = value1 OR column1 = value OR … OR column1 = valueN
```

BETWEEN

- The **BETWEEN** operator can be used in a **WHERE** clause to select values within a range

```
SELECT column1, column2, column3 FROM table_name
WHERE column2 BETWEEN value1 AND value2
```

LIKE

- The **LIKE** operator can be used in a **WHERE** clause to search for a specified pattern in a column
- **LIKE** supports the % wildcards

```
SELECT column1, column2, column3 FROM table_name
WHERE column2 LIKE '%value1%'
```

NULL / NOT NULL

- **NULL** means that a value has not been entered for a particular column in a row, i.e. the value is missing
- **NULL** is not numeric or any other value so it can't equal itself or anything else
- **NOT** creates the negation of an expression
- To test for a **NULL** value you will want to use the expressions **IS NULL** or **IS NOT NULL**

```
SELECT column1, column2 FROM table_name
WHERE column2 IS NULL
```

```
SELECT column1, column2 FROM table_name
WHERE column2 IS NOT NULL
```

ORDER BY

- The **ORDER BY** keyword can be added to a **SELECT** statement to sort the records
- The typical default order is ascending **(ASC)**, but you can also specify descending **(DESC)**

```
SELECT column1, column2, column3 FROM table_name
ORDER BY column2, column1, column3 DESC
```

Aggregation and Summarization: Finding the Insights

Aggregate Functions

- The following functions, including **COUNT**, calculate totals or statistics by column
- These statistics can apply to all rows or groupings of rows using **GROUP BY**
- Other SQL implementations may include additional statistics (standard deviation, variance, median, etc.)

Function	Description
COUNT()	Counts the total number of records
SUM()	Sum of numeric values
AVG()	Finds the average
MIN()	Returns the smallest number
MAX()	Returns the largest number

GROUP BY

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to aggregate by groups (similar to Excel's pivot table).

The **GROUP BY** clause follows the **WHERE** clause and precedes the **ORDER BY** clause (if present)

This clause regularly makes use of SQL's aggregate functions; when used with **GROUP BY**, each aggregate function produces a single value for each group

```
SELECT column1, column2, count(1) as ct, sum(column1) as sum1
FROM table_name
WHERE a_condition
GROUP BY column1, column2
ORDER BY column2, column1
```

HAVING

- The **HAVING** clause can be added to **GROUP BY** to filter records that meet certain conditions, i.e. return a subset of the groupings
- Operators are the same as the **WHERE** clause
-

```
SELECT column1, column2, count(1) as counts FROM table_name
GROUP BY column1, column2 HAVING counts > 5
```

```
SELECT column1, count(1) as counts, sum(column2) as sum2
FROM table_name GROUP BY column1
HAVING counts > 1 and sum2 > 10000
```

DISTINCT

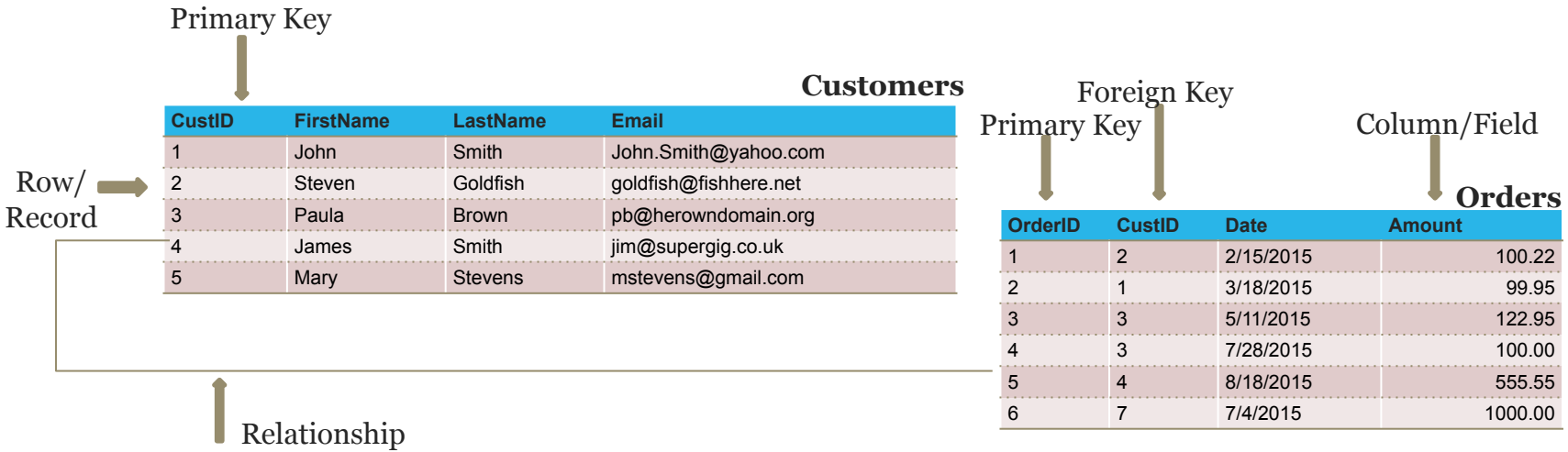
- The **DISTINCT** function returns the unique values in a column or unique records across multiple columns
- The combination of **COUNT** and **DISTINCT** returns the number of unique values

```
SELECT DISTINCT column1 FROM table_name
```

```
SELECT COUNT(DISTINCT column1) FROM table_name
```

The Relational Power: Joining Data Together

Relational Tables



CustID	FirstName	LastName	Email
1	John	Smith	John.Smith@yahoo.com
2	Steven	Goldfish	goldfish@fishhere.net
3	Paula	Brown	pb@herowndomain.org
4	James	Smith	jim@supergig.co.uk
5	Mary	Stevens	mstevens@gmail.com

OrderID	CustID	Date	Amount
1	2	2/15/2015	100.22
2	1	3/18/2015	99.95
3	3	5/11/2015	122.95
4	3	7/28/2015	100.00
5	4	8/18/2015	555.55
6	7	7/4/2015	1000.00

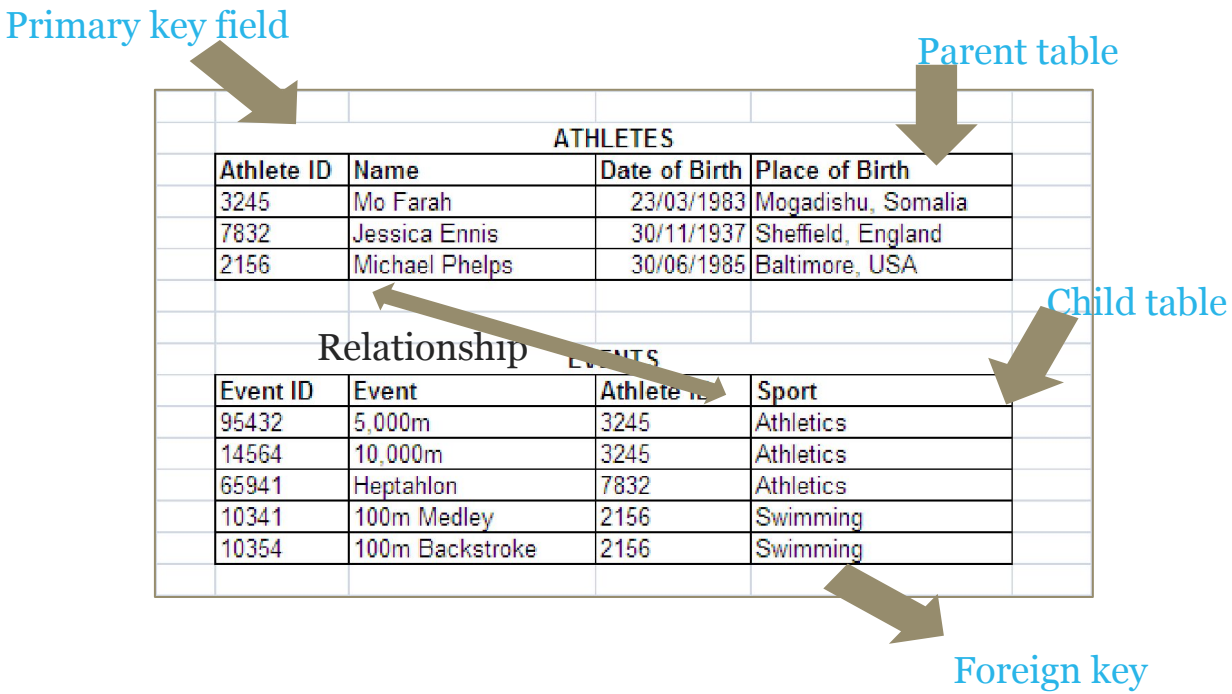
Primary Key

- Primary key is a unique identifier of records in a table
- Primary key values may be generated manually or automatically
- A primary key can consist of more than one field

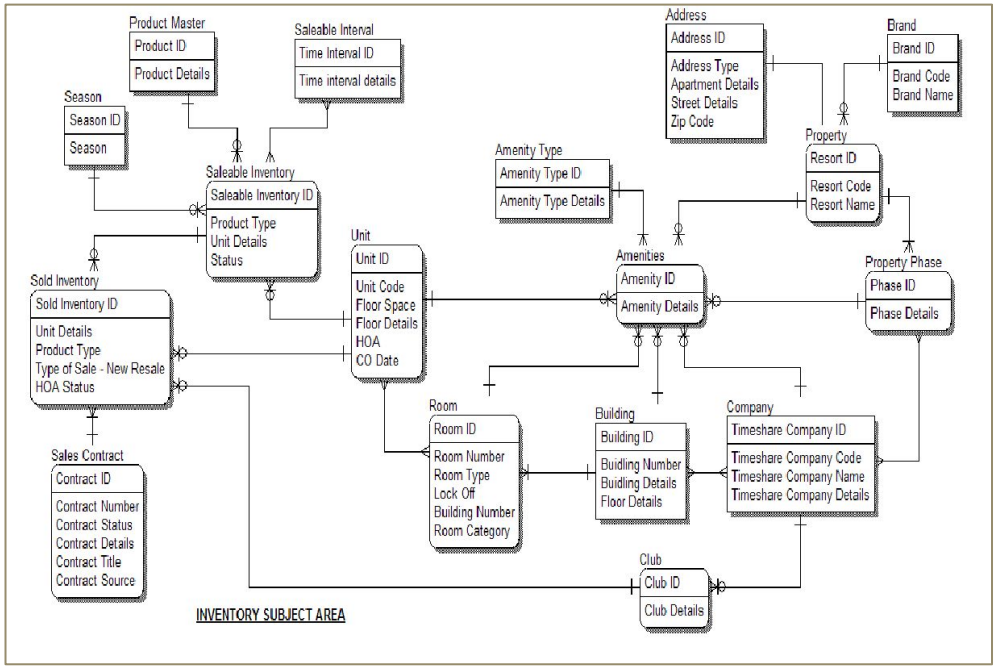
Athlete Table

ID	Name	Country	Phone Number	E-mail Address
3245	Mo Farah	UK	02890 987001	mobot@ukathletics.co.uk
4532	Usain Bolt	Jamaica	02890 945002	usain.bolt@jamaica.com
7832	Jessica Ennis	UK	02890 922003	j.ennis@ukathletics.co.uk
2156	Michael Phelps	USA	02890 931004	michael.phelps@usathletics.com
9875	Yang Sun	China	02890 969005	yang.sun@swimming.cn

Foreign Key



Entity Relationship Diagram

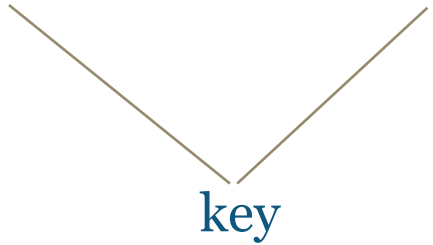


JOINS

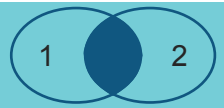
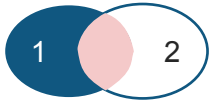
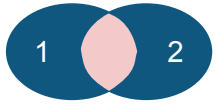
A **JOIN** combines fields from two tables using a key column shared by the tables

SELECT table_a.column1, table_a.column2, table_b.column3

FROM table_a **JOIN** table_b ON table_a.column1 = table_b.column1



Join Types

	**INNER JOIN** Selects rows from both tables as long as there is a match between the key columns in both tables
	**LEFT JOIN** Returns all rows from the left table (1), with the matching rows in the right table (2); the result is NULL in the right side when there is no match
	**RIGHT JOIN** Returns all rows from the right table (2), with the matching rows in the left table (1); the result is NULL in the left side when there is no match
	**FULL OUTER JOIN** Returns all rows from the left table (1) and from the right table (2); the result is NULL in the left or right side when there is no match

CASE STATEMENTS

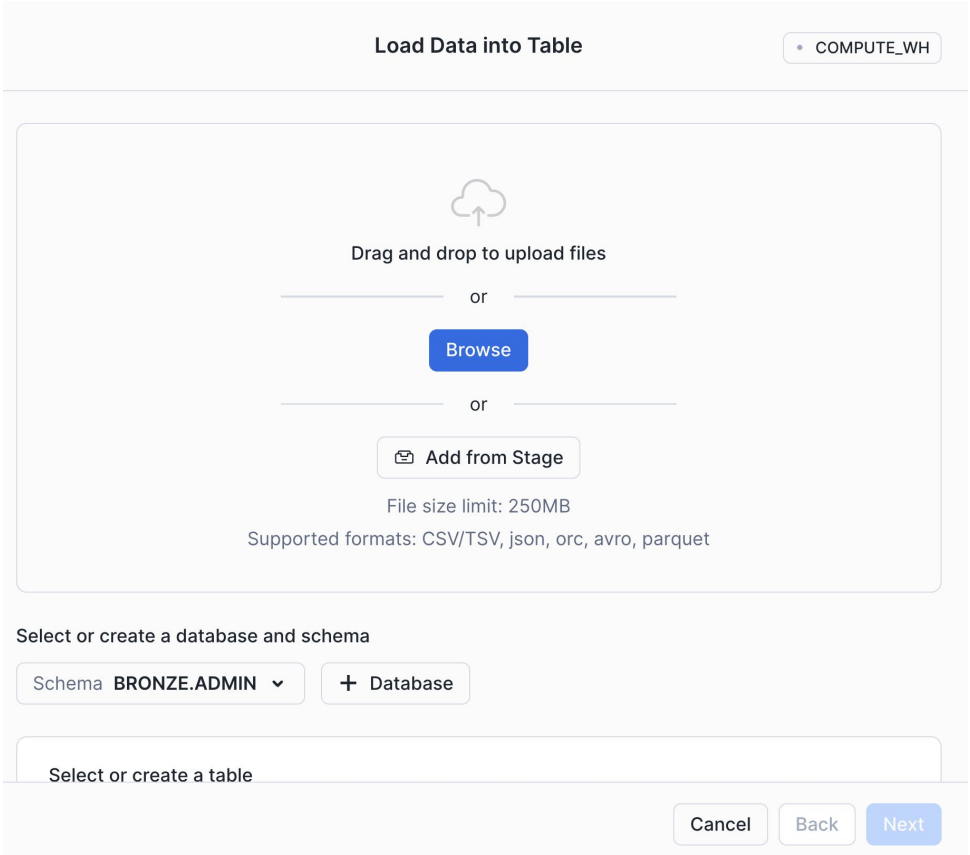
- **CASE** statements allow conditional logic in SQL queries.
- They evaluate conditions and return a result when the first condition is met

```
SELECT
        CASE
                WHEN column1 = 1 THEN ‘one’
                WHEN column1 = 2 THEN ‘two’
                WHEN column1 = 3 THEN ‘three’
                ELSE ‘zero’
        END AS  number_col

FROM table_name
```

Data Management & Reporting

Uploading Data through Snowflake UI



Views

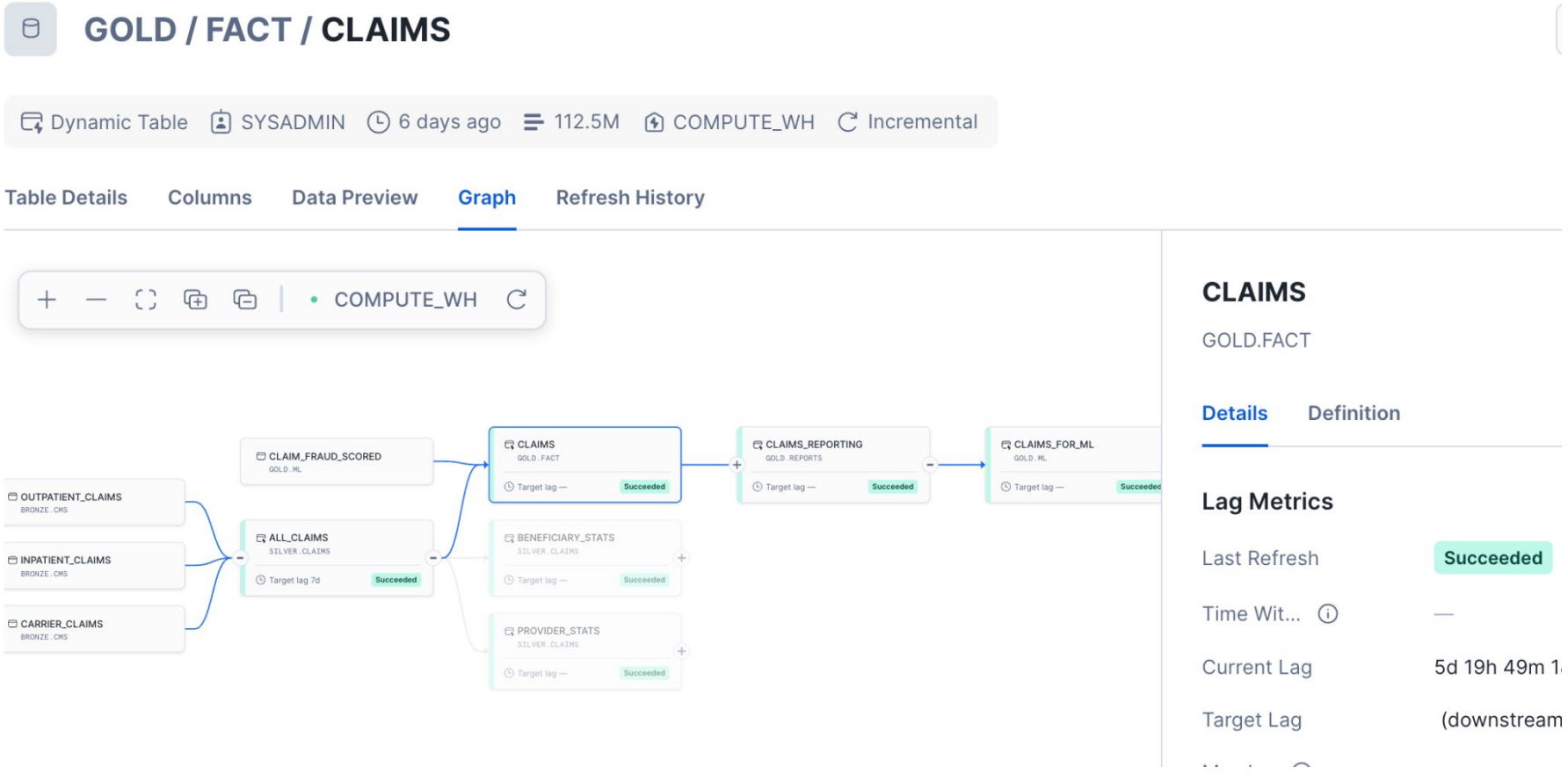
- A **VIEW** allows the result of a query to be accessed as if it were a table.
- The **SELECT** query is specified in the **CREATE VIEW** statement.

```
CREATE OR REPLACE VIEW view_name AS

SELECT
        CASE
                WHEN column1 = 1 THEN ‘one’
                WHEN column1 = 2 THEN ‘two’
                WHEN column1 = 3 THEN ‘three’
                ELSE ‘zero’
        END AS  number_col,
        column2, column3

FROM table_name
```

Snowflake Dynamic Tables

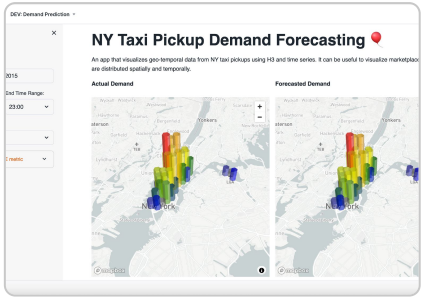


BONUS: Using AI to Build Dashboards

Streamlit-in-Snowflake

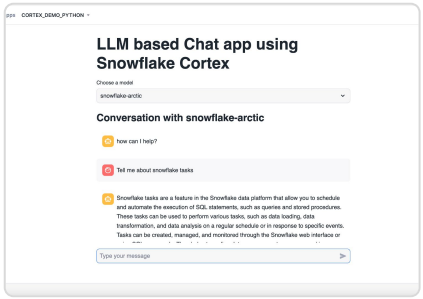
Natural Language Processing

Sentiment Analysis
NLP models that showcase text summarization, sentiment analysis, translation



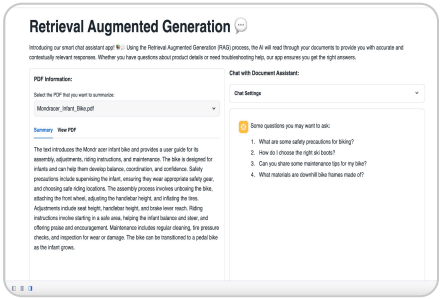
Mapping & Geospatial

Interactive Maps
Build powerful geospatial experiences using H3 and Mapbox



LLM Integration

Chat-based Applications
Applications which leverage leading LLM models from Snowflake Cortex



RAG Applications

Retrieval Augmented Gen
Ask questions of data siloed in documents using RAG with Snowflake Cortex

Streamlit