Describe Cloud Concepts
-
AZURE FUNDAMENTALS

Introduction to Cloud technologies
& AZ-900 certification preparation

Describe Azure architecture and services

Steven VINCENT - Burgundy School of Business -
Jan. 2024-

Chapter study guide

Describe Azure architecture and services (35–40%)

Describe Azure storage services

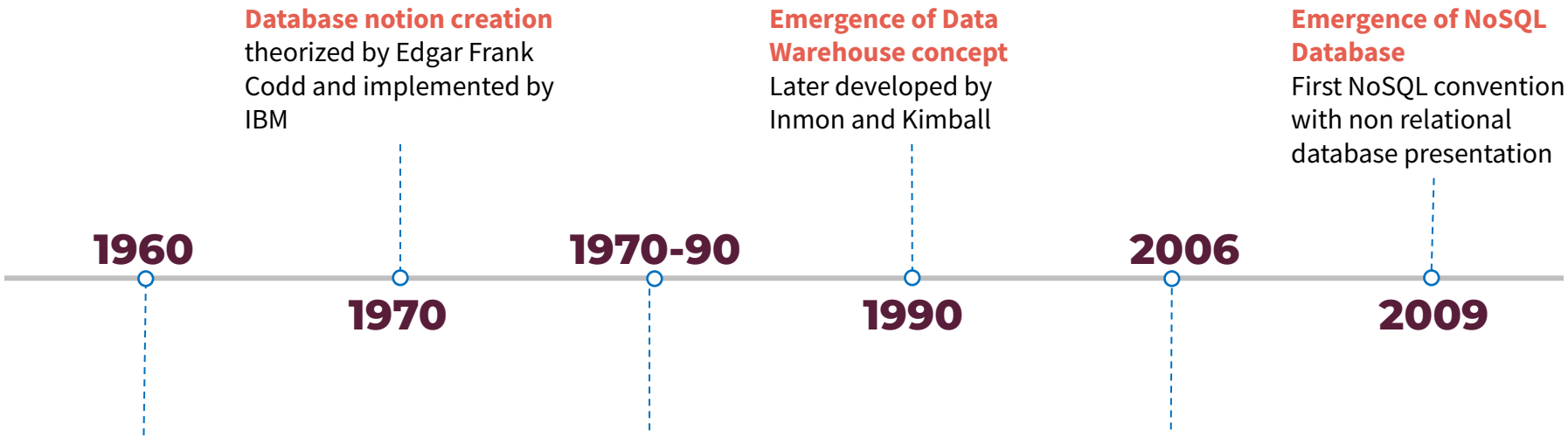
- Compare Azure storage services
- Describe storage tiers
- Describe redundancy options
- Describe storage account options and storage types
- Identify options for moving files, including AzCopy, Azure Storage Explorer, and Azure File Sync
- Describe migration options, including Azure Migrate and Azure Data Box

Describe Azure identity, access, and security

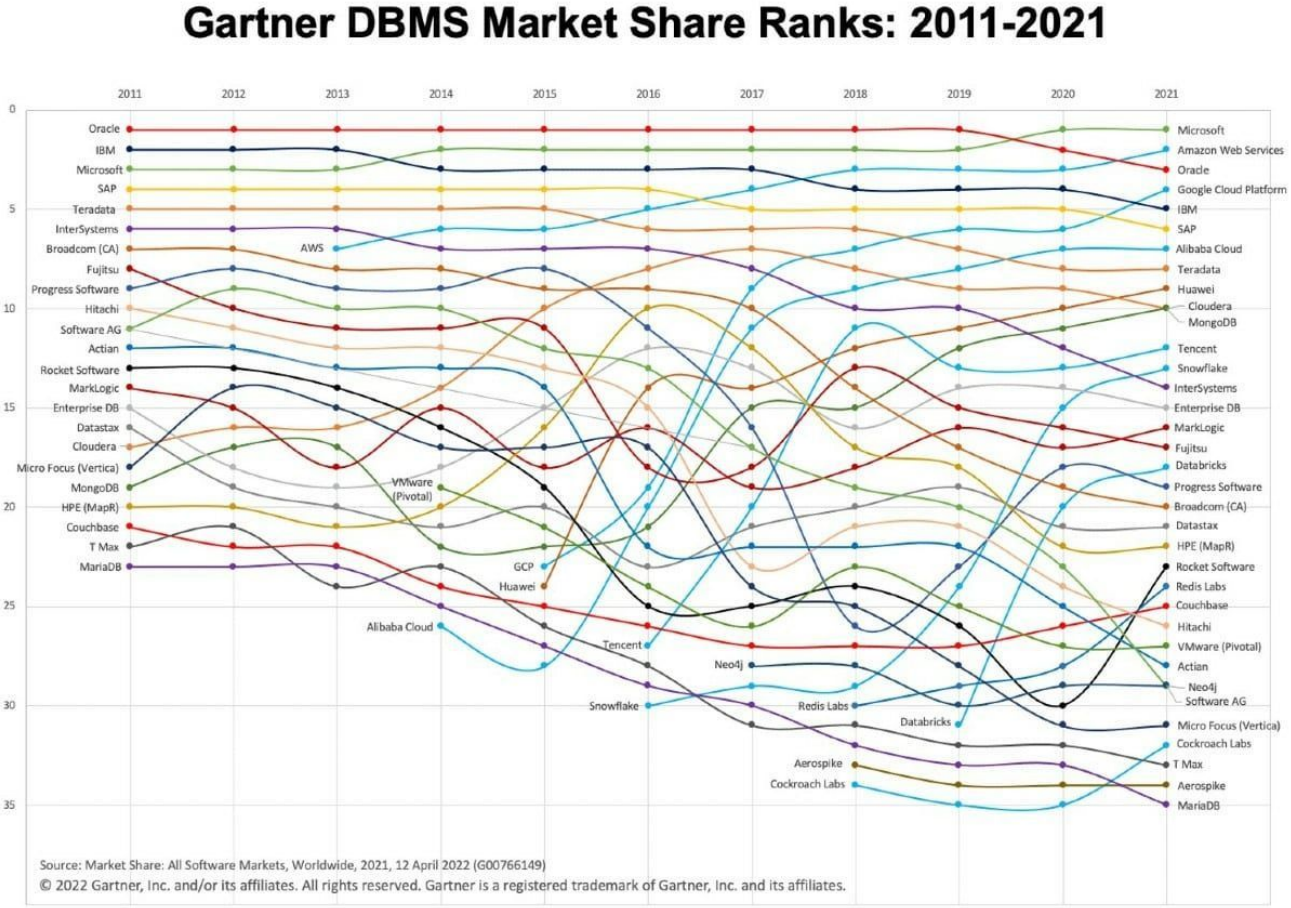
- Describe directory services in Azure, including Microsoft Azure Active Directory (Azure AD), part of Microsoft Entra and Azure Active Directory Domain Services (Azure AD DS)
- Describe authentication methods in Azure, including single sign-on (SSO), multifactor authentication, and passwordless
- Describe external identities and guest access in Azure
- Describe Conditional Access in Microsoft Azure Active Directory (Azure AD), part of Microsoft Entra
- Describe Azure role-based access control (RBAC)
- Describe the concept of Zero Trust
- Describe the purpose of the defense in depth model
- Describe the purpose of Microsoft Defender for Cloud

Introduction to data storage

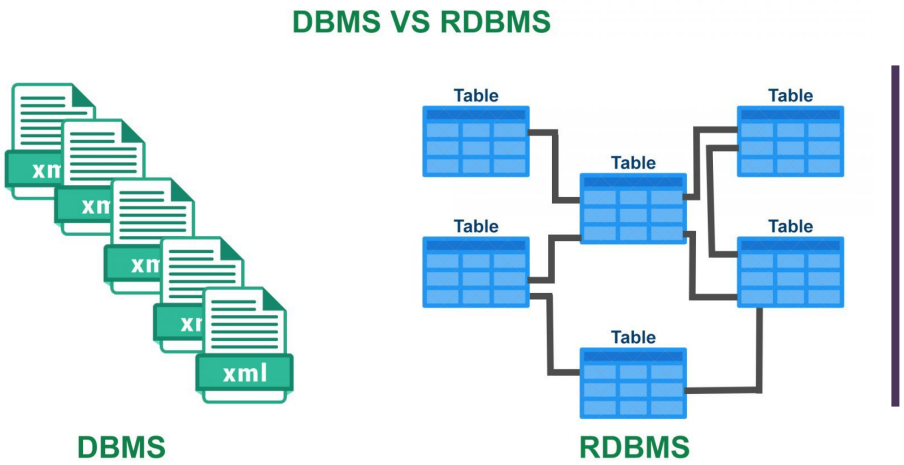
HISTORY OF DATABASE

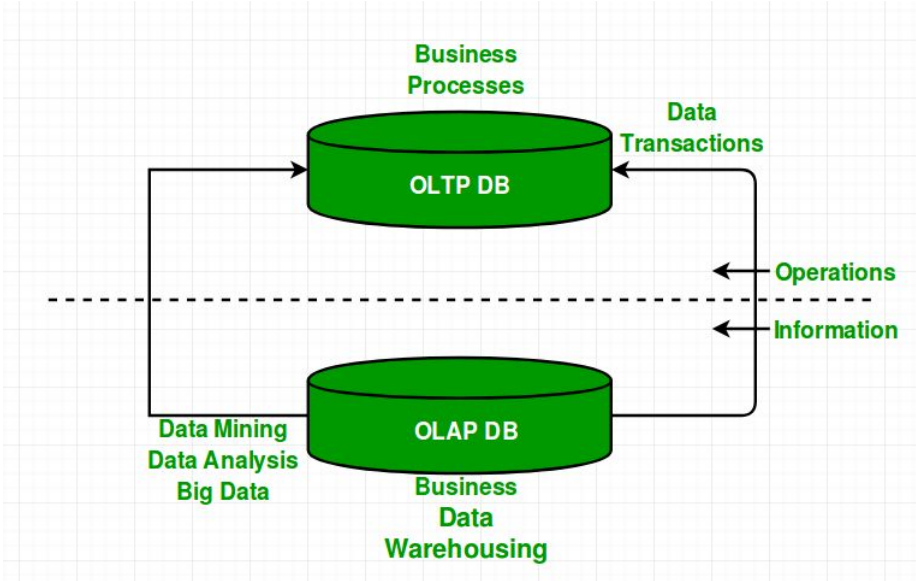


DBMS MARKET SHARE

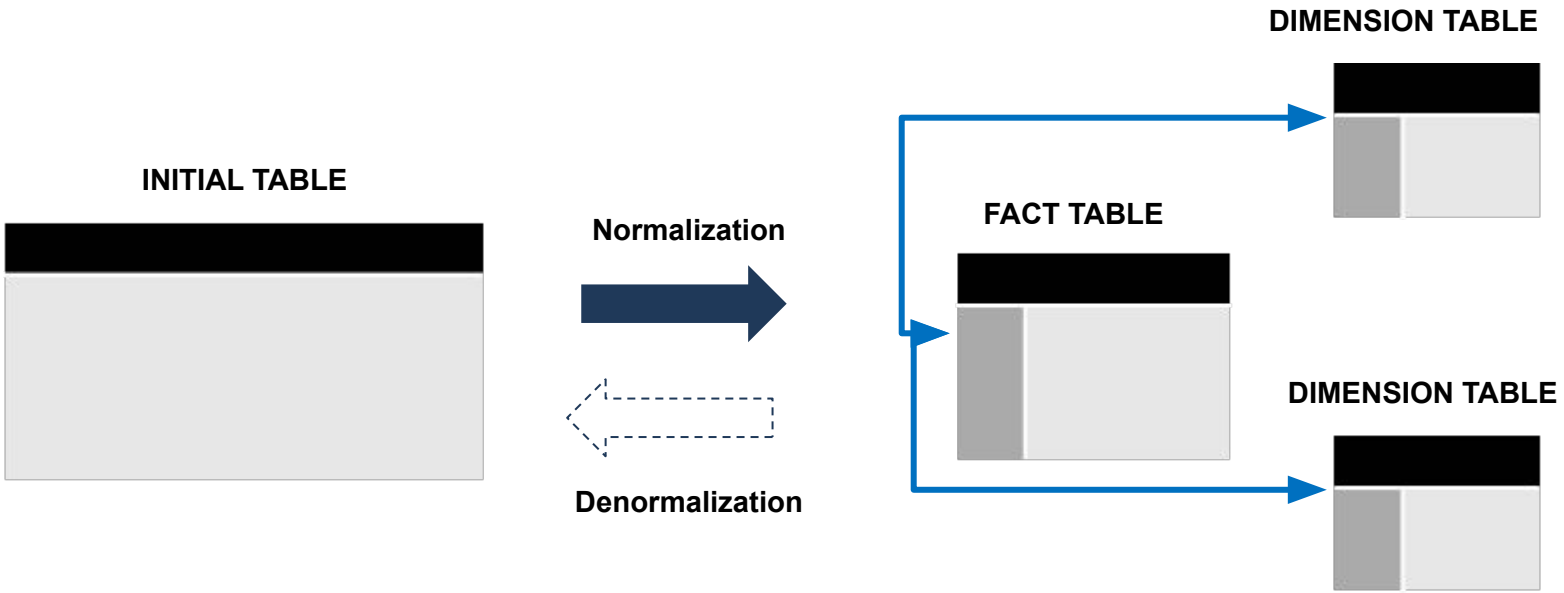


DBMS AND RDBMS / OLAP VS OLTP

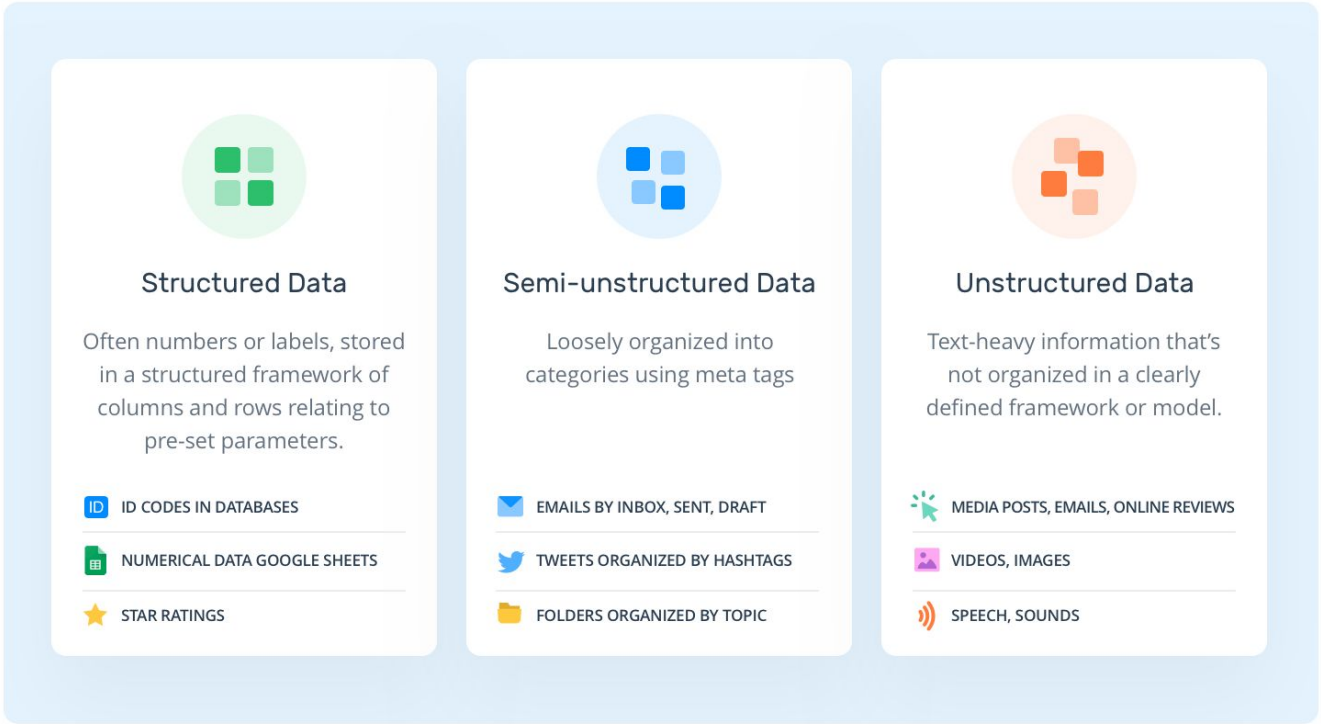




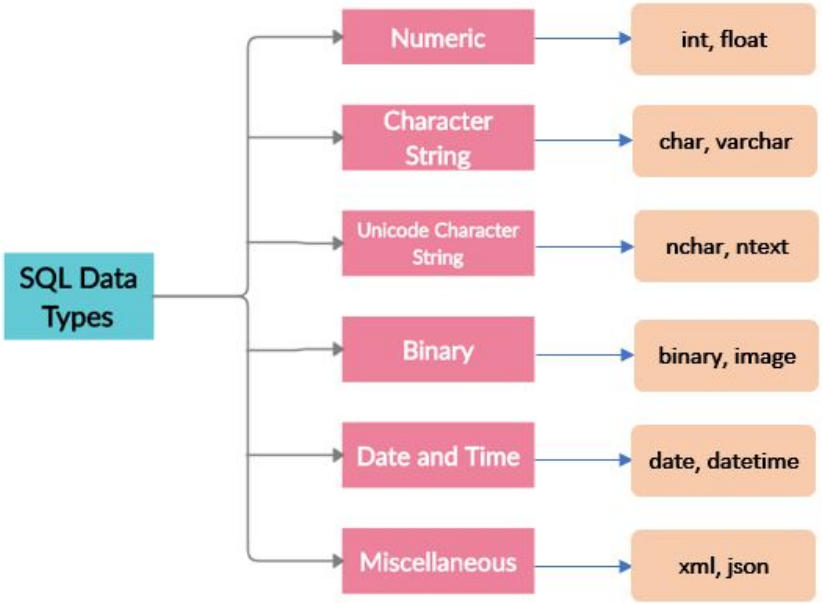
RELATIONAL DATABASE



UNSTRUCTURED VS STRUCTURED DATA

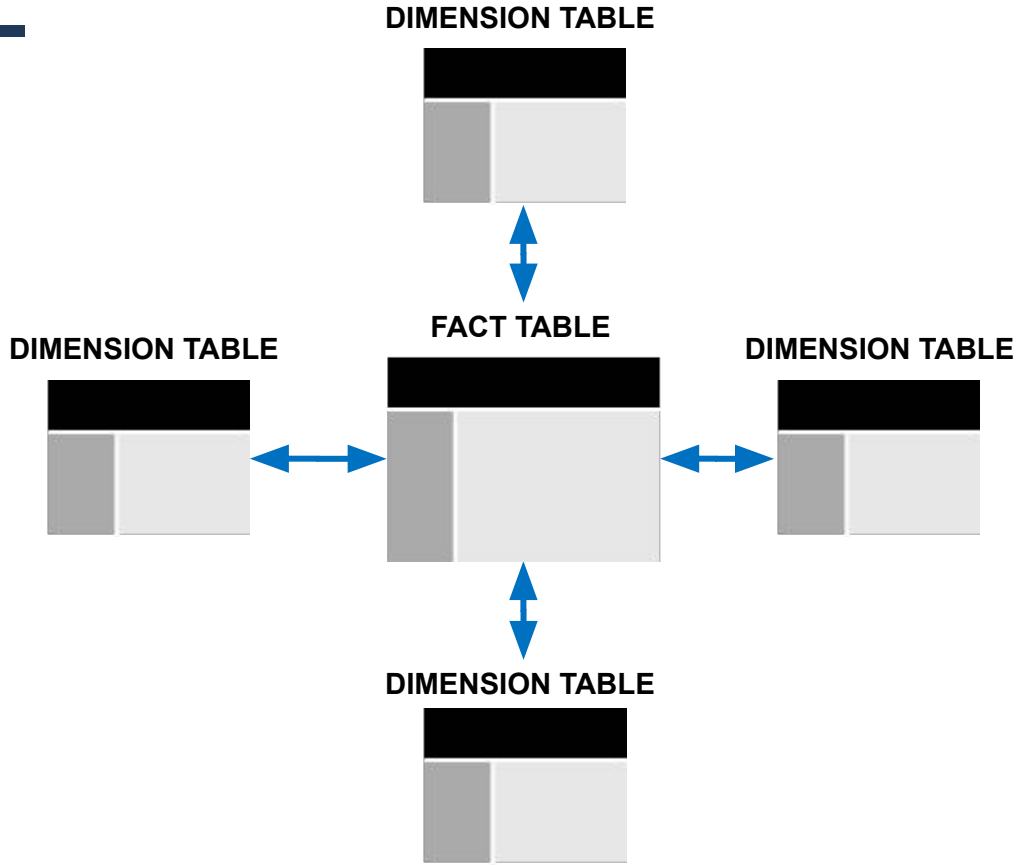


DATA TYPES IN DATABASES

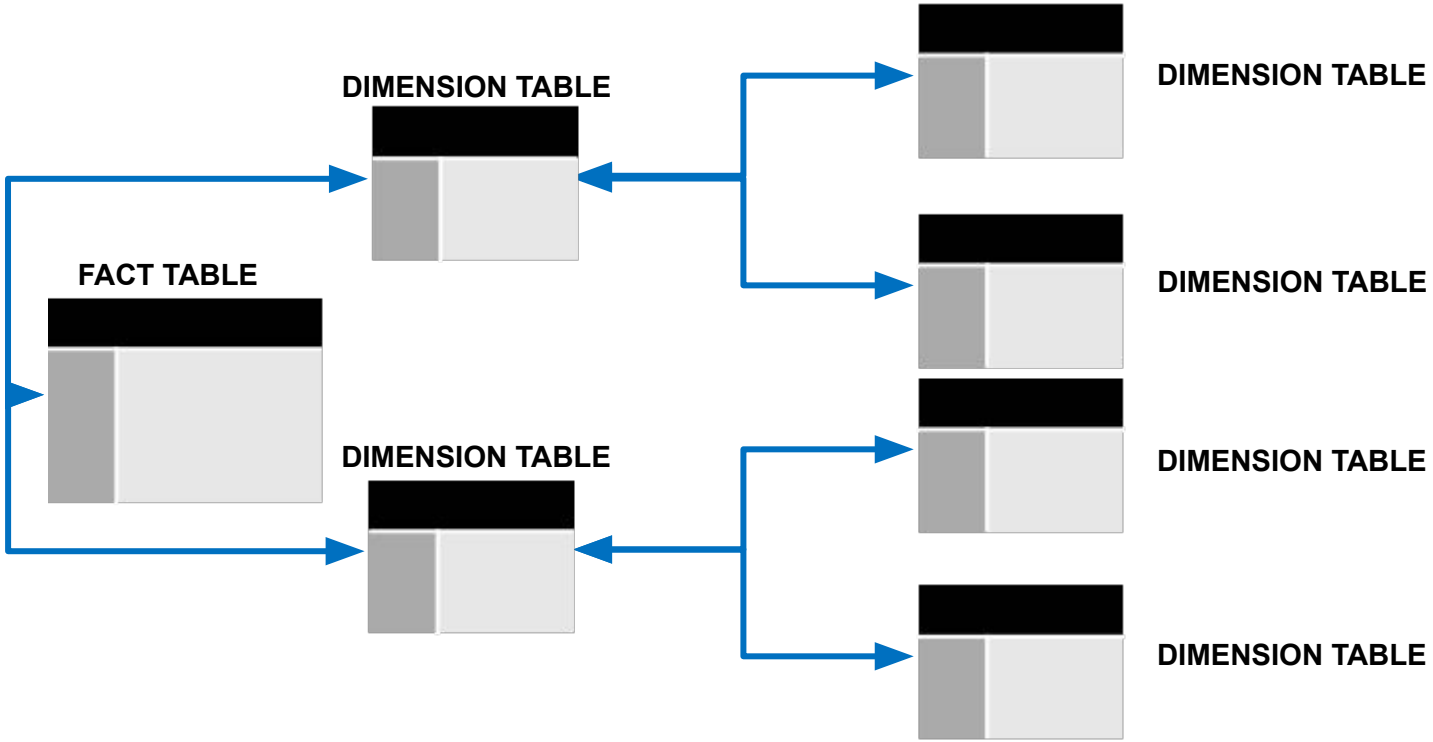



```
CREATE TABLE utilisateur
(
    id INT PRIMARY KEY NOT NULL,
    nom VARCHAR(100),
    prenom VARCHAR(100),
    email VARCHAR(255),
    date_naissance DATE,
    pays VARCHAR(255),
    ville VARCHAR(255),
    code_postal VARCHAR(5),
    nombre_achat INT
)
```

STAR SCHEMA

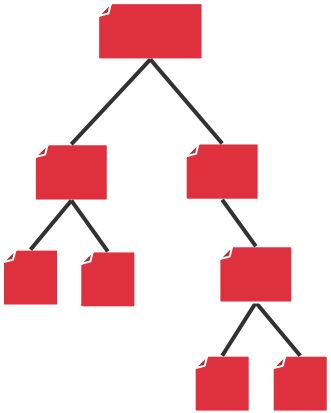

DIMENSION TABLE
DIMENSION TABLE
FACT TABLE
DIMENSION TABLE
DIMENSION TABLE

SNOWFLAKE SCHEMA

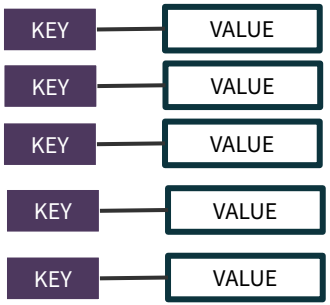


NOSQL DATABASE

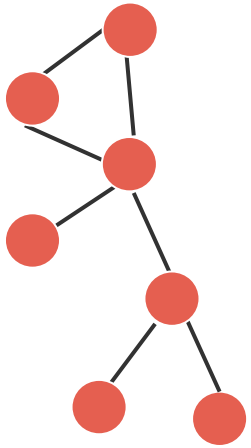
Document



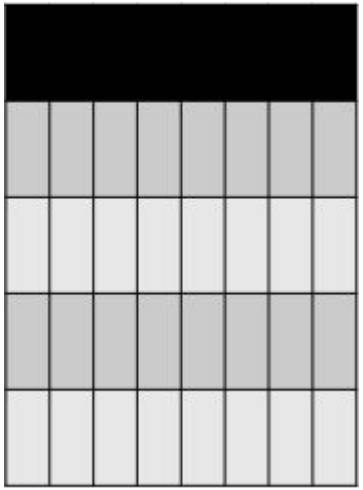
Key-Value



Graph

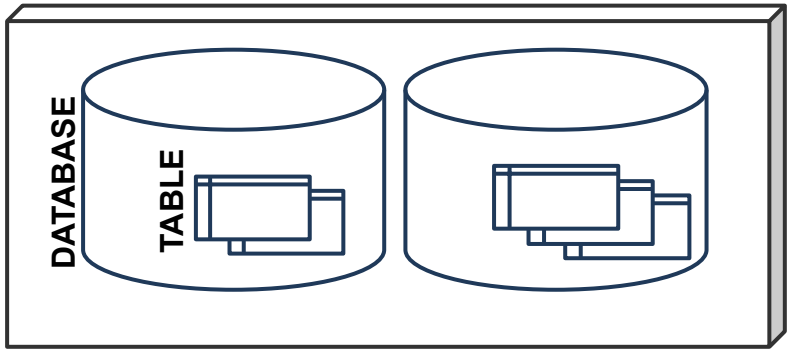


Wide-column



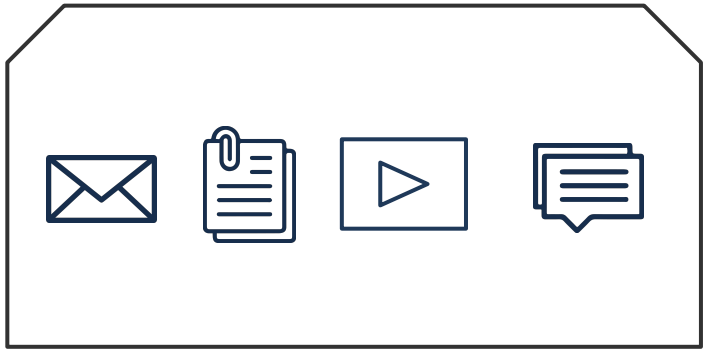
DATA WAREHOUSE VS DATA LAKE

DATA WAREHOUSE



Structured Data

DATA LAKE



Unstructured Data

Describe Azure storage services

Azure storage services Azure storage account

- Highly available
- Redundancy
- Encrypted
- Fine-grained control over data access
- Scalable
- Endpoint / Rest API

Storage accounts

Storage account provides a unique namespace for your Azure Storage data that's accessible from anywhere in the world over HTTP or HTTPS.

Data Lake Storage Gen2 *:*
https://<storage-account-name>.dfs.core.windows.net
Azure Files *:*
https://<storage-account-name>.file.core.windows.net

Azure storage redundancy

Redundancy ensures that your data is safe if transient hardware failures occur or if an unexpected outage occurs.

Locally redundant storage (LRS):
replicates your data three times within a single data center in the primary region.

Geo-redundant storage (GRS):
GRS is similar to running LRS in two regions
+ Read-access geo-redundant storage (RA-GRS)

Zone redundant storage (ZRS)
your data is still accessible for both read and write operations even if a zone becomes unavailable

Geo-zone-redundant storage (GZRS):
is similar to running ZRS in the primary region and LRS in the secondary region.
+ Read-access geo-zone-redundant storage (RA-GZRS)

Recovery Point Objective (RPO) : The interval between the most recent writes to the primary region and the last write to the secondary region due to Asynchronous replication (15 min)

Azure storage services

Container
Blob storage

Unstructured data

Can store and ingest massive amounts of data (text, binary, videos).

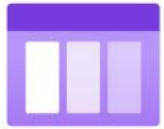
Serving images/documents directly to a browser.
Streaming video and audio.
Storing data for backup and archiving.

File share

Structured data

Managed file shares for cloud or on-premises deployments

Lift and shift
Replace on-premise file server or NAS



Queue
storage

Unstructured data

A messaging store for reliable messaging between application components.

Submit button on a website trigger a message
Backlog of work to process asynchronously

Table storage

Semi-structured data

Allows you to store structured NoSQL data in the cloud, providing a key/attribute store with a schemaless design.
User data for web applications, address books, device information

Azure disks and Database

Disks

Unstructured data

Managed or unmanaged disk emulation

Persistent storage for Virtual Machines

Azure Cosmos DB

Semi-structured data

Globally distributed NoSQL Database service (Schema-less) with multiple APIs

Highly responsive & multi-regional applications

SQL databases

Structured data

High-performance and managed relational SQL database service (schema and relationships)

Application Database

Different
- Sizes
- Types (SSD, HDD)
- Performance tiers

+ SQL
+ MongoDB
+ Cassandra

+ MySQL
+ PostgreSQL

Azure data migration & file movements

Data migration

Azure Data Box

Physical migration service that helps transfer large amounts of data in a quick, inexpensive, and reliable way

Azure Migrate

Service that helps you migrate from an on-premises environment to the cloud

File movements

Az Copy

Command-line to upload files, download files, copy files between storage accounts, and even synchronize files.

+ Move files between cloud provider

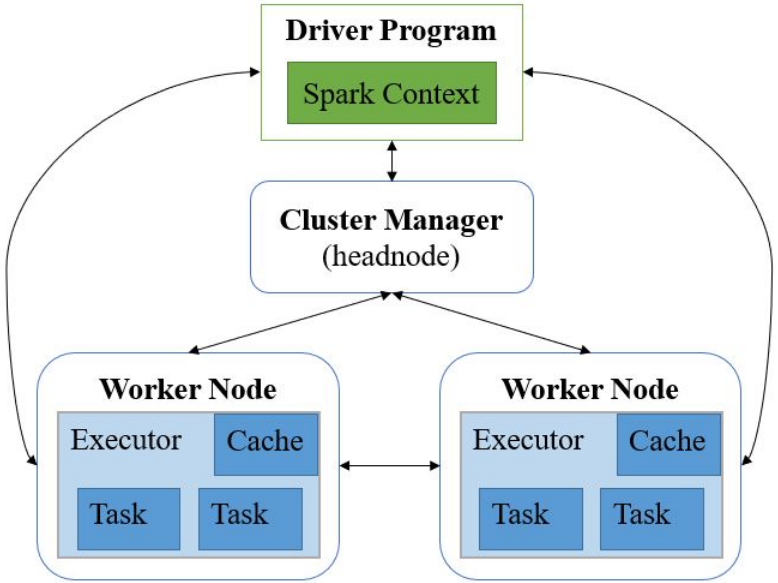
Storage Sync Services

Azure File Sync is a tool that lets you centralize your file shares (bi-directionally synced) in Azure Files and keep the flexibility, performance, and compatibility of a Windows file server.

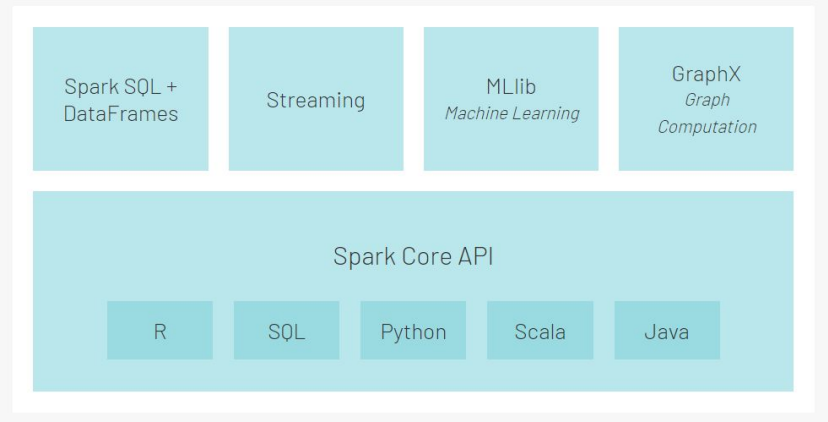
INTRODUCTION TO DATABRICKS

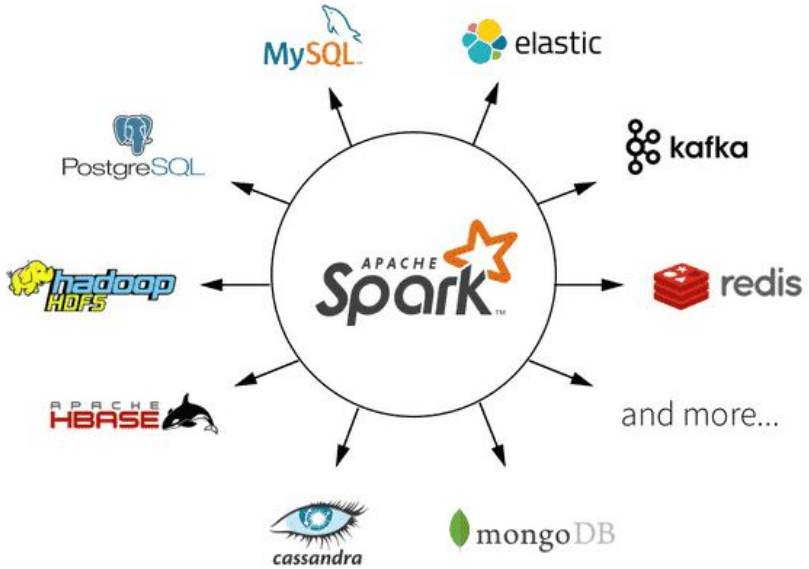
SPARK PRESENTATION

Created in 2014, Apache Spark is an open-source unified analytics engine for large-scale data processing



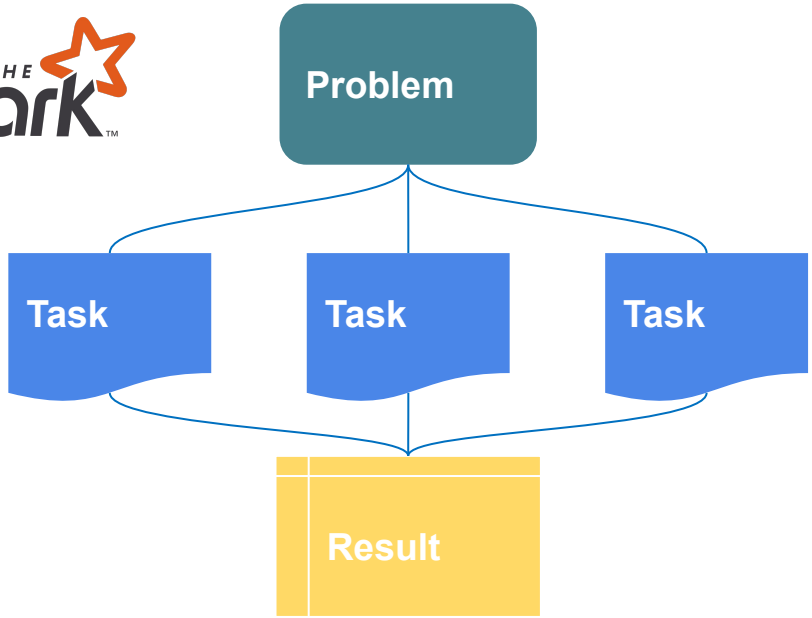
APACHE SPARK ECOSYSTEM





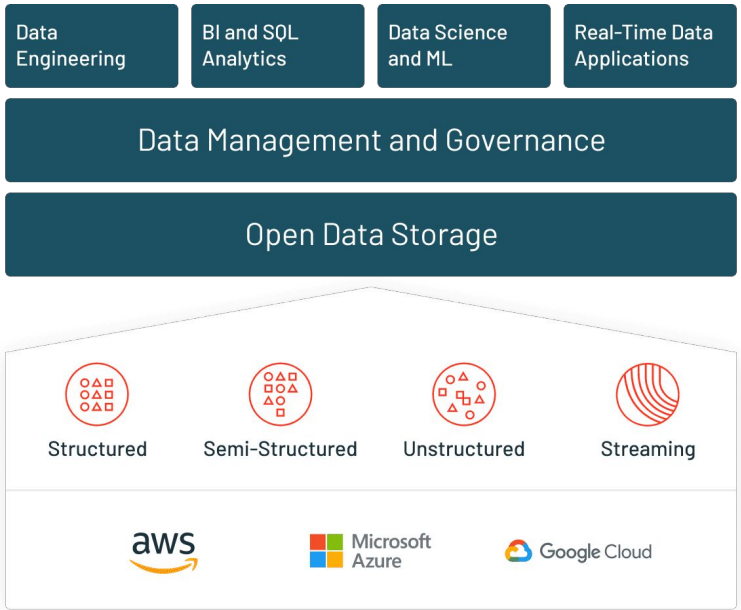
PARALLEL PROCESSING VS SEQUENTIAL PROCESSING

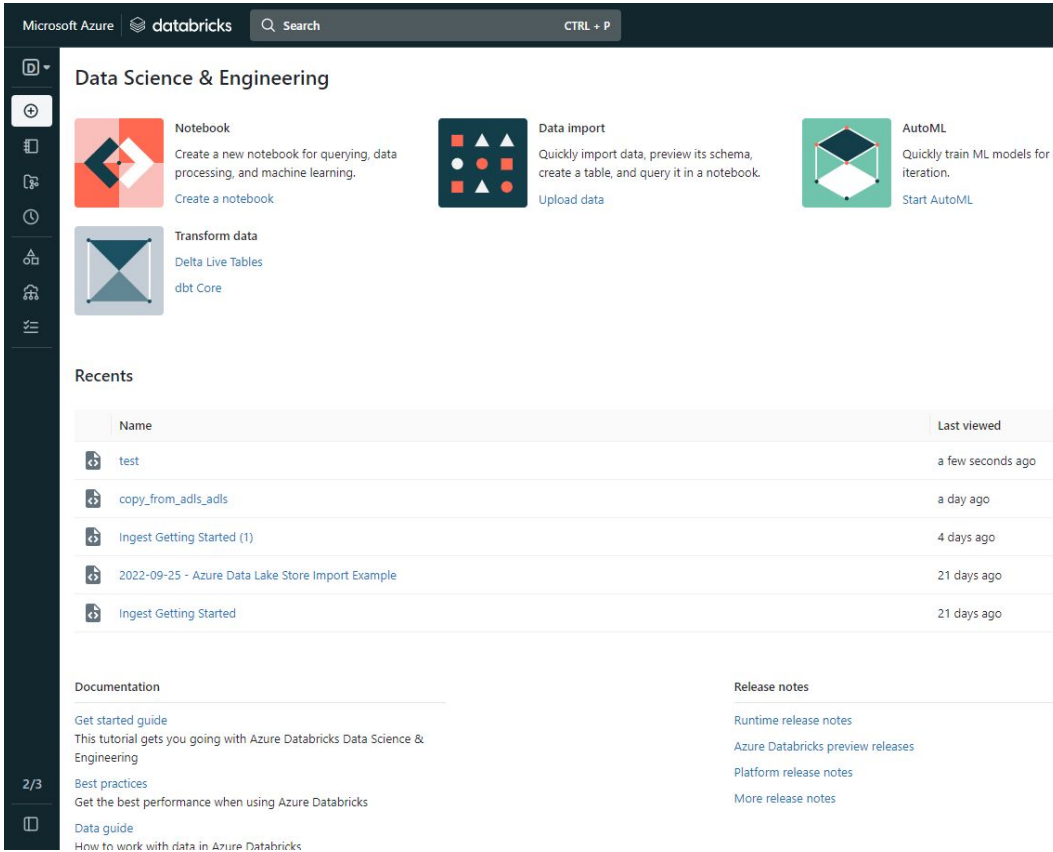
(Spark processing)



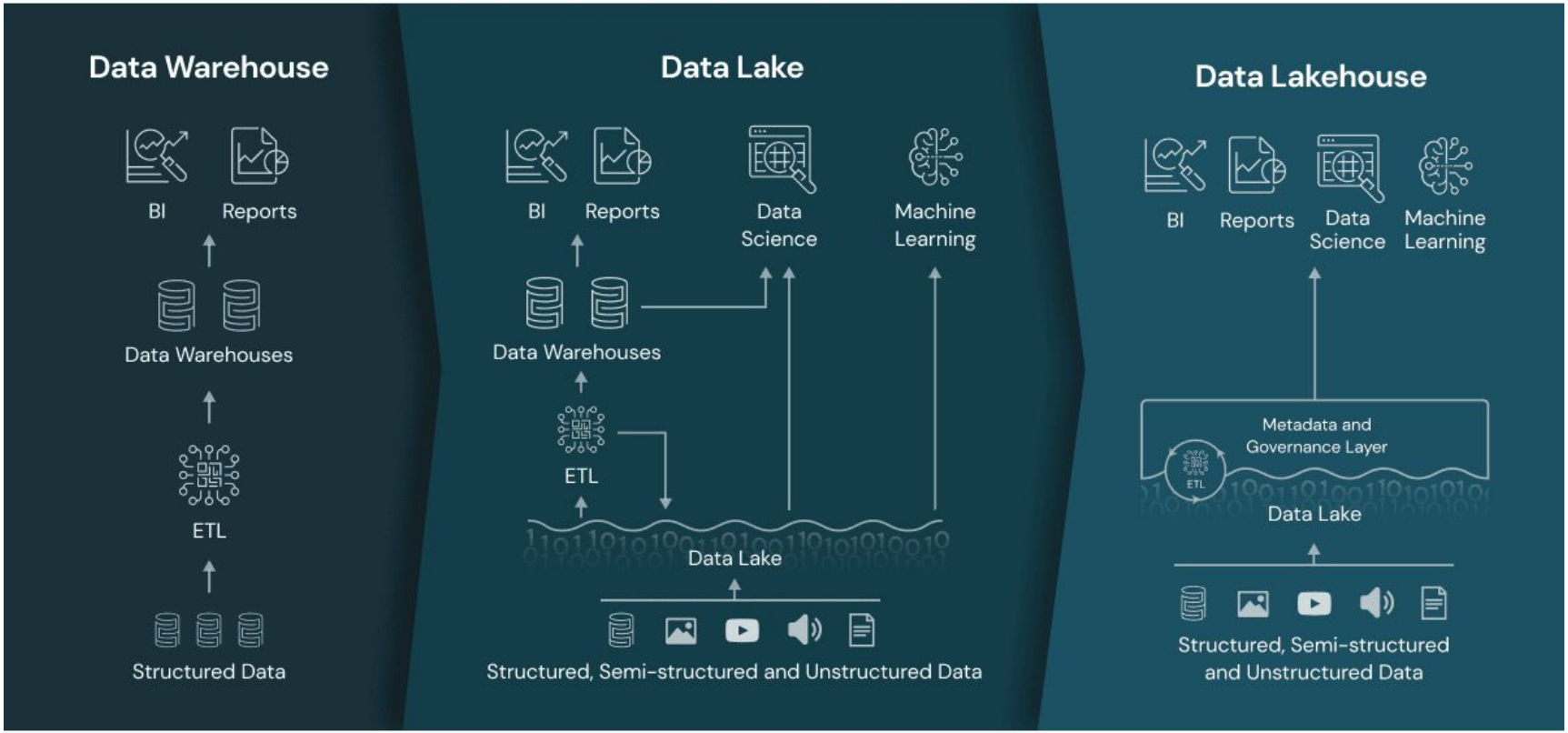
(Pandas processing)

DATABRICKS


Data Engineering
BI and SQL Analytics
Data Science and ML
Real-Time Data Applications
Data Management and Governance
Open Data Storage
Structured
Semi-Structured
Unstructured
Streaming
aws
Microsoft Azure
Google Cloud


Microsoft Azure
databricks
Search
CTRL + P
Data Science & Engineering
Notebook
Create a new notebook for querying, data processing, and machine learning.
Create a notebook
Data import
Quickly import data, preview its schema, create a table, and query it in a notebook.
Upload data
AutoML
Quickly train ML models for iteration.
Start AutoML
Transform data
Delta Live Tables
dbt Core
Recents
Name
Last viewed
test
a few seconds ago
copy_from_adls_adls
a day ago
Ingest Getting Started (1)
4 days ago
2022-09-25 - Azure Data Lake Store Import Example
21 days ago
Ingest Getting Started
21 days ago
Documentation
Get started guide
This tutorial gets you going with Azure Databricks Data Science & Engineering
Best practices
Get the best performance when using Azure Databricks
Data guide
Release notes
Runtime release notes
Azure Databricks preview releases
Platform release notes
More release notes
2/3

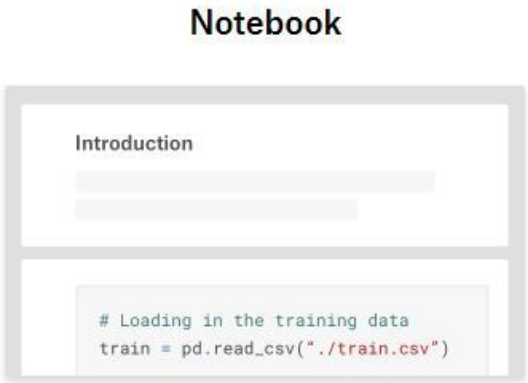
EVOLUTION OF THE DATASTACK



DATABRICKS ADVANTAGES

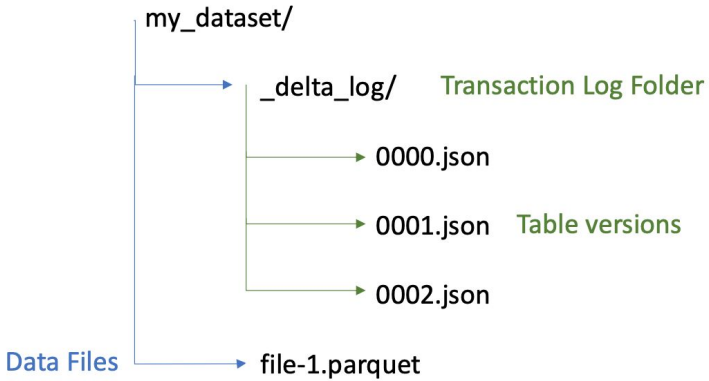
Multiple languages supported

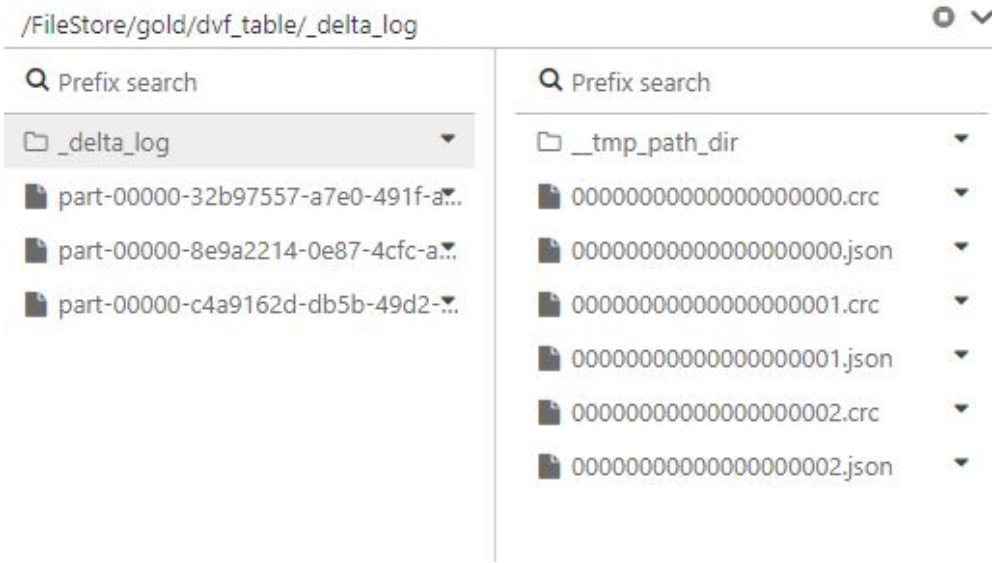
Easily explore data and document code



Versioning integration with GIT

DELTA FORMAT

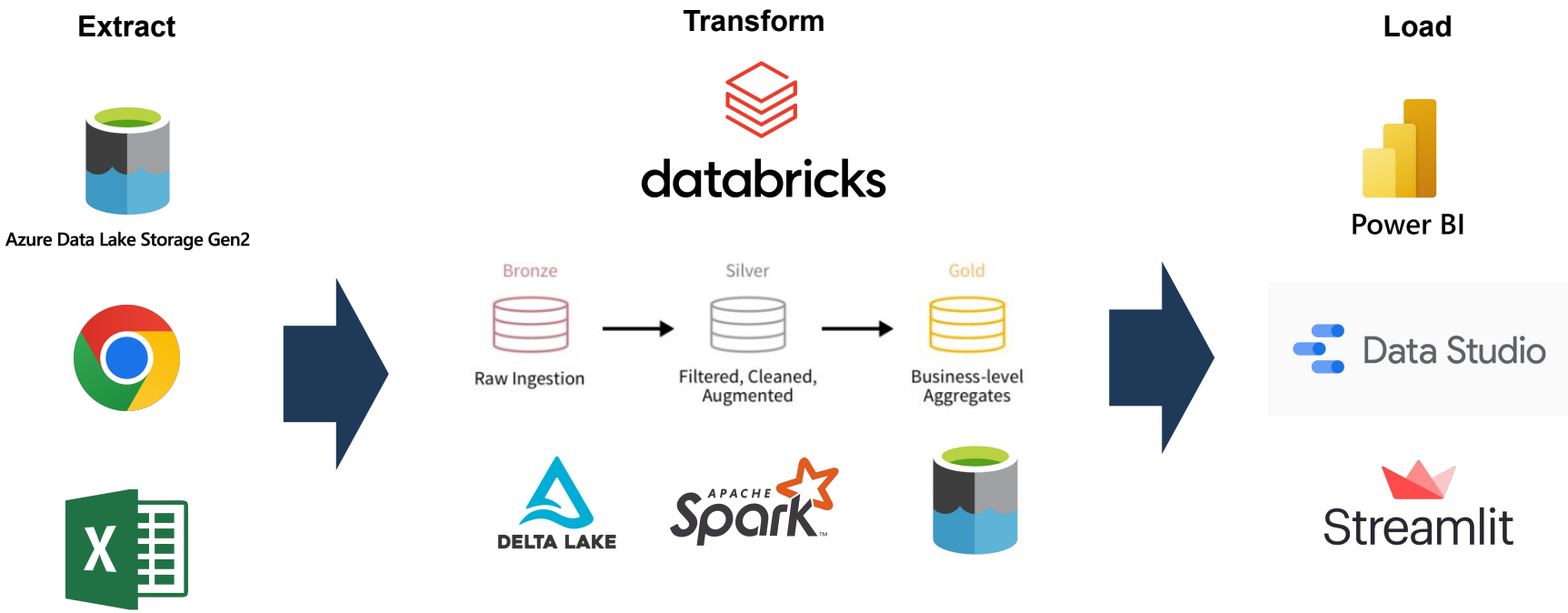




DATABRICKS MEDAILLON ARCHITECTURE

WORKSHOP

WORKSHOP ARCHITECTURE



SQL CHEAT SHEET

DELTA LAKE WITH SPARK SQL

Delta Lake is an open source storage layer that brings ACID transactions to Apache Spark™ and big data workloads.

delta.io | Documentation | GitHub | Delta Lake on Databricks

CREATE AND QUERY DELTA TABLES

Create and use managed database

```
-- Managed database is saved in the Hive metastore.
Default database is named "default".
DROP DATABASE IF EXISTS dbName;
CREATE DATABASE dbName;
USE dbName  -- This command avoids having to specify
dbName.tableName every time instead of just tableName.
```

Query Delta Lake table by table name (preferred)

```
/* You can refer to Delta Tables by table name, or by
path. Table name is the preferred way, since named tables
are managed in the Hive Metastore (i.e., when you DROP a
named table, the data is dropped also — not the case for
path-based tables.) */
SELECT * FROM [dbName.] tableName
```

Query Delta Lake table by path

```
SELECT * FROM delta.`path/to/delta_table` -- note backticks
```

Convert Parquet table to Delta Lake format in place

```
-- by table name
CONVERT TO DELTA [dbName.]tableName
[PARTITIONED BY (col_name1 col_type1, col_name2
col_type2)]

-- path-based tables
CONVERT TO DELTA parquet.`/path/to/table` -- note backticks
[PARTITIONED BY (col_name1 col_type1, col_name2 col_type2)]
```

Create Delta Lake table as SELECT * with no upfront schema definition

```
CREATE TABLE [dbName.] tableName
USING DELTA
AS SELECT * FROM tableName | parquet.`path/to/data`
[LOCATION `/path/to/table`]
-- using location = unmanaged table
```

Create table, define schema explicitly with SQL DDL

```
CREATE TABLE [dbName.] tableName (
    id INT [NOT NULL],
    name STRING,
    date DATE,
    int_rate FLOAT)
USING DELTA
[PARTITIONED BY (time, date)] -- optional
```

Copy new data into Delta Lake table (with idempotent retries)

```
COPY INTO [dbName.] targetTable
  FROM (SELECT * FROM "/path/to/table")
  FILEFORMAT = DELTA  -- or CSV, Parquet, ORC, JSON, etc.
```


DELTA LAKE DDL/DML: UPDATE, DELETE, MERGE, ALTER TABLE

Update rows that match a predicate condition

```
UPDATE tableName SET event = 'click' WHERE event = 'clk'
```

Delete rows that match a predicate condition

```
DELETE FROM tableName WHERE "date < '2017-01-01"
```

Insert values directly into table

```
INSERT INTO TABLE tableName VALUES (
    (8003, "Kim Jones", "2020-12-18", 3.875),
    (8004, "Tim Jones", "2020-12-20", 3.750)
);
-- Insert using SELECT statement
INSERT INTO tableName SELECT * FROM sourceTable
-- Atomically replace all data in table with new values
INSERT OVERWRITE loan_by_state_delta VALUES (...)
```

Upsert (update + insert) using MERGE

```
MERGE INTO target
USING updates
ON target.Id = updates.Id
WHEN MATCHED AND target.delete_flag = "true" THEN
  DELETE
WHEN MATCHED THEN
  UPDATE SET *  -- star notation means all columns
WHEN NOT MATCHED THEN
  INSERT (date, Id, data) -- or, use INSERT *
  VALUES (date, Id, data)
```

Insert with Deduplication using MERGE

```
MERGE INTO logs
USING newDedupedLogs
ON logs.uniqueId = newDedupedLogs.uniqueId
WHEN NOT MATCHED
  THEN INSERT *
```

Alter table schema — add columns

```
ALTER TABLE tableName ADD COLUMNS (
    col_name data_type
    [FIRST|AFTER colA_name])
```

Alter table — add constraint

```
-- Add "Not null" constraint:
ALTER TABLE tableName CHANGE COLUMN col_name SET NOT NULL
-- Add "Check" constraint:
ALTER TABLE tableName
ADD CONSTRAINT dateWithinRange CHECK date > "1900-01-01"
-- Drop constraint:
ALTER TABLE tableName DROP CONSTRAINT dateWithinRange
```

TIME TRAVEL

View transaction log (aka Delta Log)

```
DESCRIBE HISTORY tableName
```

Query historical versions of Delta Lake tables

```
SELECT * FROM tableName VERSION AS OF 0
SELECT * FROM tableName@v0 -- equivalent to VERSION AS OF 0
SELECT * FROM tableName TIMESTAMP AS OF "2020-12-18"
```

Find changes between 2 versions of table

```
SELECT * FROM tableName VERSION AS OF 12
EXCEPT ALL SELECT * FROM tableName VERSION AS OF 11
```

TIME TRAVEL (CONTINUED)

Rollback a table to an earlier version

```
-- RESTORE requires Delta Lake version 0.7.0+ & DBR 7.4+.
RESTORE tableName VERSION AS OF 0
RESTORE tableName TIMESTAMP AS OF "2020-12-18"
```

UTILITY METHODS

View table details

```
DESCRIBE DETAIL tableName
DESCRIBE FORMATTED tableName
```

Delete old files with Vacuum

```
VACUUM tableName [RETAIN num HOURS] [DRY RUN]
```

Clone a Delta Lake table

```
-- Deep clones copy data from source, shallow clones don't.
CREATE TABLE [dbName.] targetName
[SHALLOW | DEEP] CLONE sourceName [VERSION AS OF 0]
[LOCATION "path/to/table"]
-- specify location only for path-based tables
```

Interoperability with Python / DataFrames

```
-- Read name-based table from Hive metastore into DataFrame
df = spark.table("tableName")
-- Read path-based table into DataFrame
df = spark.read.format("delta").load("/path/to/delta_table")
```

Run SQL queries from Python

```
spark.sql("SELECT * FROM tableName")
spark.sql("SELECT * FROM delta.`/path/to/delta_table`")
```

Modify data retention settings for Delta Lake table

```
-- logRetentionDuration -> how long transaction log history
is kept, deletedFileRetentionDuration -> how long ago a file
must have been deleted before being a candidate for VACCUM.
ALTER TABLE tableName
SET TBLPROPERTIES(
    delta.logRetentionDuration = "interval 30 days",
    delta.deletedFileRetentionDuration = "interval 7 days"
);
SHOW TBLPROPERTIES tableName;
```

PERFORMANCE OPTIMIZATIONS

Compact data files with Optimize and Z-Order

```
*Databricks Delta Lake feature
OPTIMIZE tableName
[ZORDER BY (colNameA, colNameB)]
```

Auto-optimize tables

```
*Databricks Delta Lake feature
ALTER TABLE [table_name | delta.`path/to/delta_table`]
SET TBLPROPERTIES (delta.autoOptimize.optimizeWrite = true)
```

Cache frequently queried data in Delta Cache

```
*Databricks Delta Lake feature
CACHE SELECT * FROM tableName
-- or:
CACHE SELECT colA, colB FROM tableName WHERE colNameA > 0
```

databricks