IntelliPaat

Designing and Implementing an Azure Data Solution

DP 200 and DP 201

Non-Relational Data Stores & Azure Data Lake Storage

Agenda

Introduction to Non-relational Data Stores

Non-relational Database and NoSQL

01

Non-relational database is a database that does not use the tabular schema of rows and columns found in most traditional database systems

02

It uses a storage model that is optimized for the specific requirements of the type of data being stored

03

In it, data may be stored as simple key/value pairs, as JSON documents, or as a graph consisting of edges and vertices

Non-relational Database and NoSQL

influxdb

TIMESCALE

Prometheus

Time series data stores are optimized for queries over time-based sequences of data

Graph data stores are optimized for exploring weighted relationships between entities

ArangoDB

JanusGraph

neo4j

Neither formats would generalize well to the task of managing transactional data

Non-relational Database and NoSQL

The term **'NoSQL'** refers to data stores that do not use SQL for queries; instead, they use other programming languages and constructs to query data

'NoSQL' means non-relational databases, though many of these databases do support SQL-compatible queries

However, the underlying query execution strategy is very different from the way a traditional RDBMS executes queries with SQL

Non-relational Database and NoSQL

Some of the major categories of non-relational (NoSQL) databases are:

01 Document Data Stores

02 Columnar Data Stores

03 Key/Value Data Stores

04 Graph Data Stores

05 Time Series Data Stores

06 Object Data Stores

07 External Index Data Stores

Document Data Stores

Document Data Stores

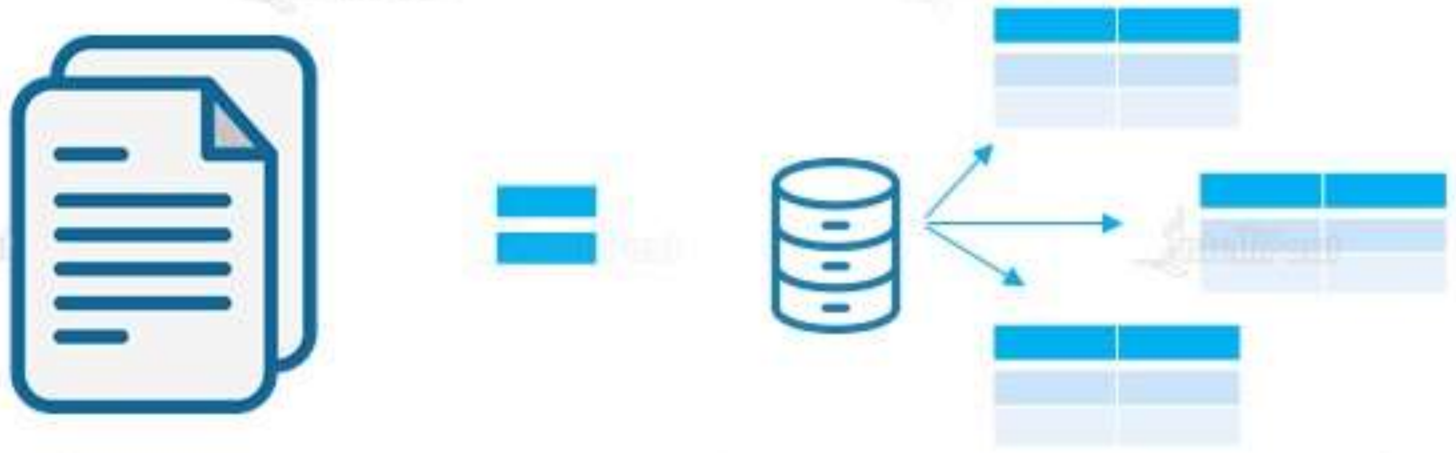
- A document data store manages a set of named string fields and object data values in an entity referred to as a **document**
- These data stores typically store data in the form of JSON documents

Document Data Stores

- The data in the fields of a document can be encoded in a variety of formats, such as, XML, YAML, JSON, BSON, or even it can be stored as plain text
- The fields within the document are exposed to the storage management system, enabling an application to query and filter data by using the values in these fields

Document Data Stores

- A single document might contain information that would be spread across several relational tables in a relational database management system (RDBMS)
- A document data store does not require all documents to have the same structure



Document Data Stores

For example, applications can store different data in documents in response to a change in business requirements

Key	Document

1001	{
	"CustomerID":99,
	"OrderItems":[
	{ "ProductID":2010,
	"Quantity":2,
	"Cost":520
	},
	{ "ProductID":4365,
	"Quantity":1,
	"Cost":18
	}],
	"OrderDate": "22/11/2019"
	}

1002	{
	"CustomerID":220,
	"OrderItems":[
	{ "ProductID":1285,
	"Quantity":1,
	"Cost":120
	}],
	"OrderDate": "25/11/2019"
	}

Document Data Stores

- Applications can retrieve documents by using the **document key**
- This is a unique identifier for each document, which is often hashed to help distribute data evenly
- Applications can also query documents based on the value of one or more fields

Key	Document
1001	`{ "CustomerID":99, "OrderItems":[ { "ProductID":2010, "Quantity":2, "Cost":520 }, { "ProductID":4365, "Quantity":1, "Cost":18 }], "OrderDate": "22/11/2019" }`
1002	`{ "CustomerID":220, "OrderItems":[ { "ProductID":1285, "Quantity":1, "Cost":120 }], "OrderDate": "25/11/2019" }`

Document Data Stores

Requirements for Document Data Stores

Requirement	Document Data
Normalization	Denormalized
Schema	Schema on read
Consistency (across concurrent transactions)	Guarantees tunable consistency at the document-level
Atomicity (transaction scope)	Collection
Locking Strategy	Optimistic (lock free)
Access Pattern	Random access

Document Data Stores

Requirements for Document Data Stores

Requirement	Document data
Indexing	Primary and secondary indexes
Data Shape	Document
Sparse	Yes
Wide (lots of columns/attributes)	Yes
Data Size	Small (KBs) to medium (low MBs)
Overall Maximum Scale	Very large (PBs)

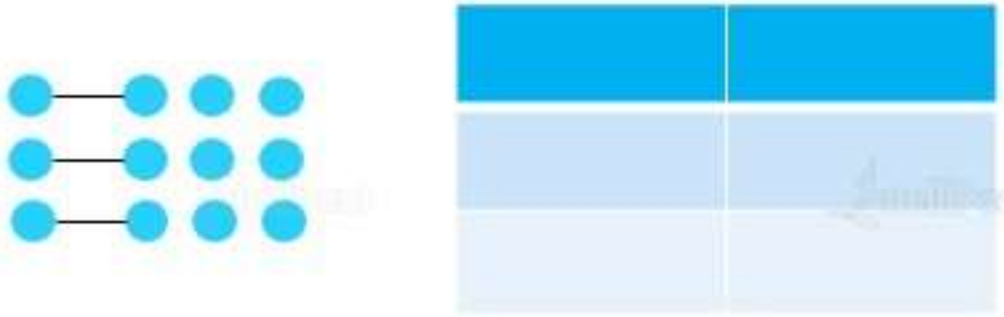
Document Data Stores

Relevant Azure Service: Azure Cosmos DB

Columnar Data Stores

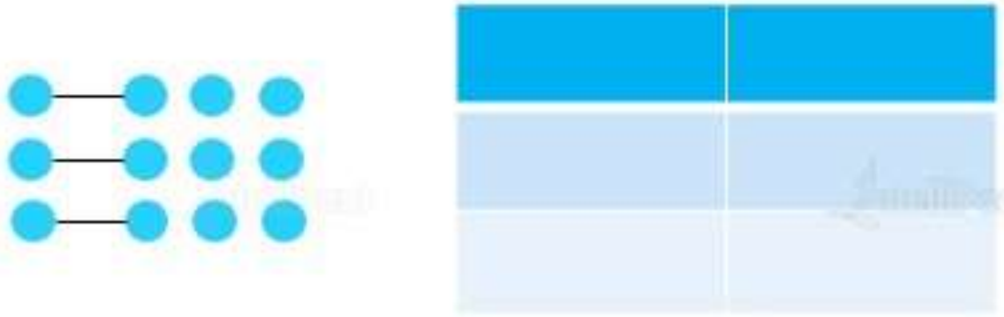
Columnar Data Stores

- A columnar or column-family data store organizes data into columns and rows
- In its simplest form, a column-family data store can appear very similar to a relational database, at least conceptually
- The real power of a column-family database lies in its denormalized approach in structuring the sparse data, which stems from the column-oriented approach in storing data



Columnar Data Stores

- Columns are divided into groups known as column families
- Each column family holds a set of columns, which are logically related and typically retrieved or manipulated as a unit
- Within a column family, new columns can be added dynamically, and here rows are sparse, i.e., a row doesn't need to have a value for every column



Columnar Data Stores

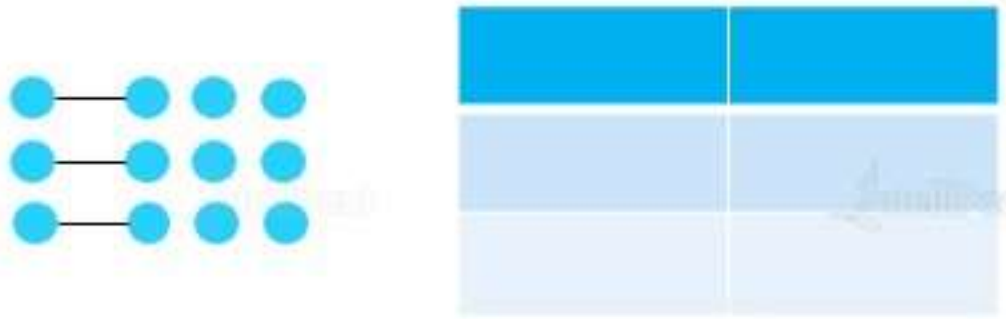
Below is an example of two column families, **Identity** and **Contact Info**

CustomerID	Column Family: Identity
001	**First name:** Satya Narayana **Last name:** Nadella
002	**First name:** Harish **Last name:** Kumar **Suffix:** Jr.
003	**First name:** Peter **Last name:** Weller **Title:** Dr.

CustomerID	Column Family: Contact Info
001	**Phone number:** 222-3600 **Email:** satyanarayana@gmail.com
002	**Email:** harishkumar@yahoo.com
003	**phone number:** 888-0120

Columnar Data Stores

- Unlike a key/value data store or a document data store, most column-family databases physically store data in the key order, rather than by computing a hash
- The row key is considered as the primary index and enables a key-based access via a specific key or a range of keys



Columnar Data Stores

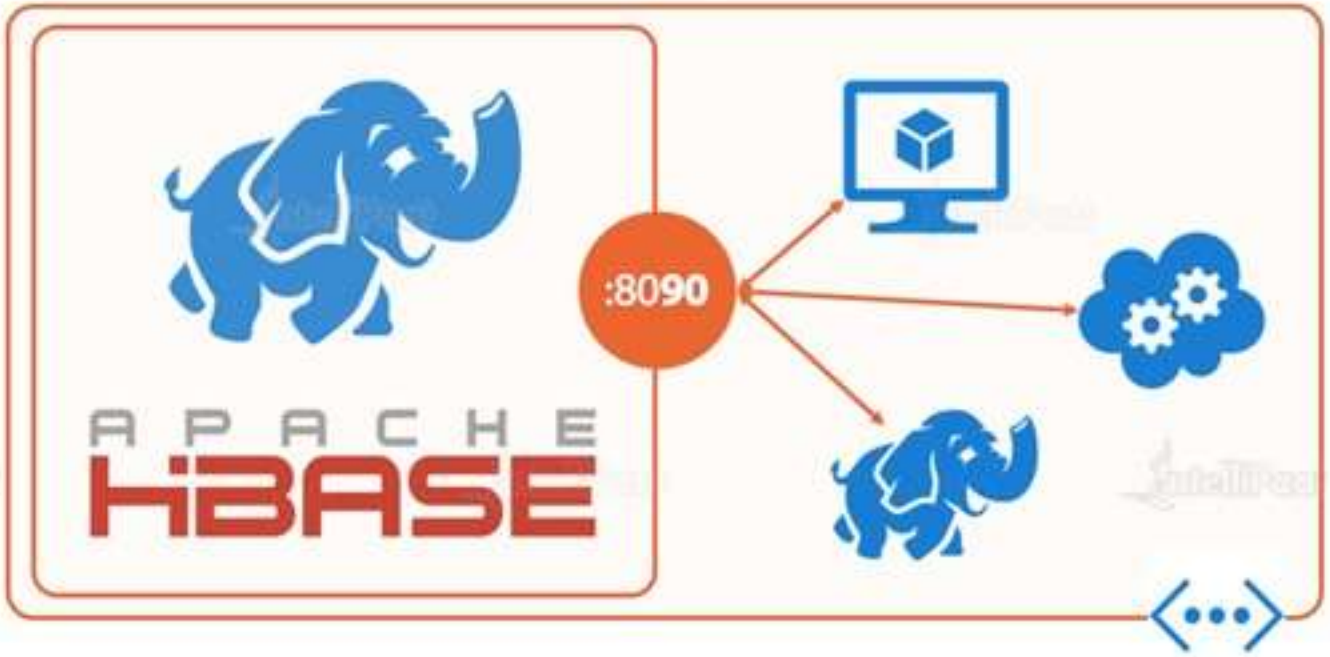
On the disk, all columns within a column family are stored together in the same file, with a certain number of rows in each column family

For large datasets, this approach offers a performance benefit by reducing the amount of data needs to be read from the disk when only a few columns are queried together at a time

Read and write operations for a row are typically atomic within a single column family, although some implementations provide atomicity across the entire row, spanning multiple column families

Columnar Data Stores

Relevant Azure Service HBase in HDInsight



Columnar Data Stores

Requirements for Columnar Data Stores

Requirement	Column-family Data
Normalization	Denormalized
Schema	Column families are defined on write and column schema on read
Consistency (across concurrent transactions)	Column-family-level guarantee
Atomicity (transaction scope)	Table
Locking Strategy	Pessimistic (row locks)
Access Pattern	Aggregates on tall/wide data

Columnar Data Stores

Requirements for Columnar Data Stores

Requirement	Column-family Data
Indexing	Primary and secondary indexes
Data Shape	Tabular with column families containing columns
Sparse	Yes
Wide (lots of columns/attributes)	Yes
Data Size	Medium (MBs) to large (low GBs)
Overall Maximum Scale	Very large (PBs)

Key/Value Data Stores

Key/Value Data Stores

- A key/value store is essentially a large hash table
- Each data value is associated with a unique key, and the key/value store uses this key to store data by using an appropriate hashing function
- The hashing function is selected to provide an even distribution of hashed keys across the data storage

Key	Value
AAAAA	1101001001111010100110101111...
AABAB	1001100001011001101011110111...
DFA766	0000000000001010101010101010...
FABCC4	1110110110101010010110101101...

Key/Value Data Stores

- Most key/value stores only support simple queries, and insert and delete operations
- To modify a value, an application must overwrite the existing data for the entire value
- In most implementations, reading or writing a single value is an atomic operation. However, if the value is large, writing may take some time

Key	Value
AAAAA	1101001001111010100110101111...
AABAB	1001100001011001101011110111...
DFA766	0000000000001010101010101010...
FABCC4	1110110110101010010110101101...

Key/Value Data Stores

- The stored values are opaque to the storage system software
- Any schema information must be provided and interpreted by the application
- Essentially, values are blobs, and the key/value store simply retrieves or stores a value by its key

Key	Value
AAAAA	1101001001111010100110101111...
AABAB	1001100001011001101011110111...
DFA766	0000000000001010101010101010...
FABCC4	1110110110101010010110101101...

Opaque to the data store

Key/Value Data Stores

- Key/value stores are highly optimized for applications performing simple lookups using the value of a key or using a range of keys
- They are less suitable for systems that need to query data across different tables of keys/values, such as joining data across multiple tables

Key	Value
AAAAA	1101001001111010100110101111...
AABAB	1001100001011001101011110111...
DFA766	0000000000001010101010101010...
FABCC4	1110110110101010010110101101...

Key/Value Data Stores

- A single key/value store can be extremely scalable as it can easily distribute data across multiple nodes on separate machines
- **Relevant Azure Services**
  - Azure Cosmos DB Table API
  - Azure Cache for Redis
  - Azure Table Storage

Key/Value Data Stores

Requirements for Key/Value Data Stores

Requirement	Key/Value Data
Normalization	Denormalized
Schema	Schema on read
Consistency (across concurrent transactions)	Key-level guarantee
Atomicity (transaction scope)	Table
Locking Strategy	Optimistic (ETag)
Access Pattern	Random access

Key/Value Data Stores

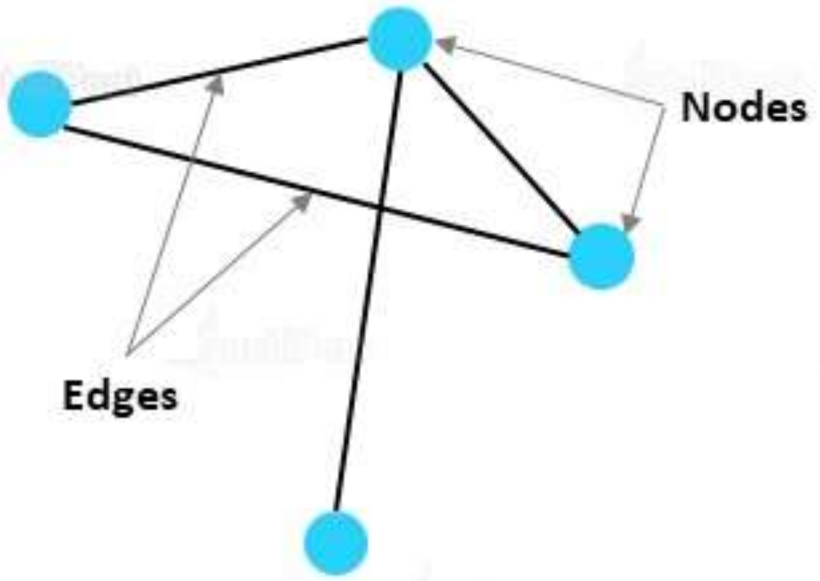
Requirements for Key/Value Data Stores

Requirement	Key/Value Data
Indexing	Primary index only
Data Shape	Key and value
Sparse	Yes
Wide (lots of columns/attributes)	No
Data Size	Small (KBs)
Overall Maximum Scale	Very large (PBs)

Graph Data Stores

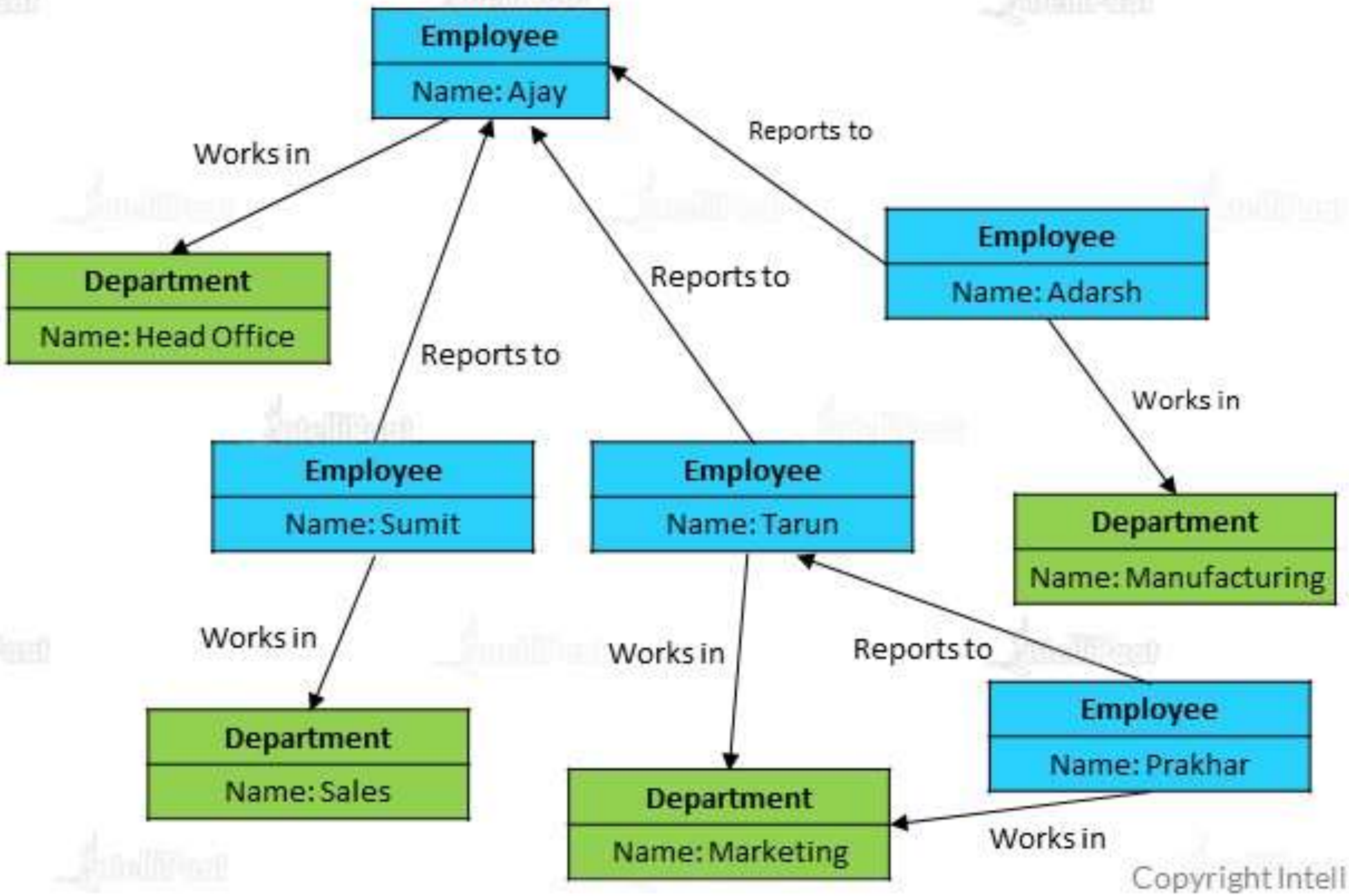
Graph Data Stores

- A graph data store manages two types of information, **nodes** and **edges**
- Nodes represent entities, and edges specify the relationships between these entities
- The purpose of a graph data store is to allow an application to efficiently perform queries, which traverse the network of nodes and edges, and to analyze the relationships between entities

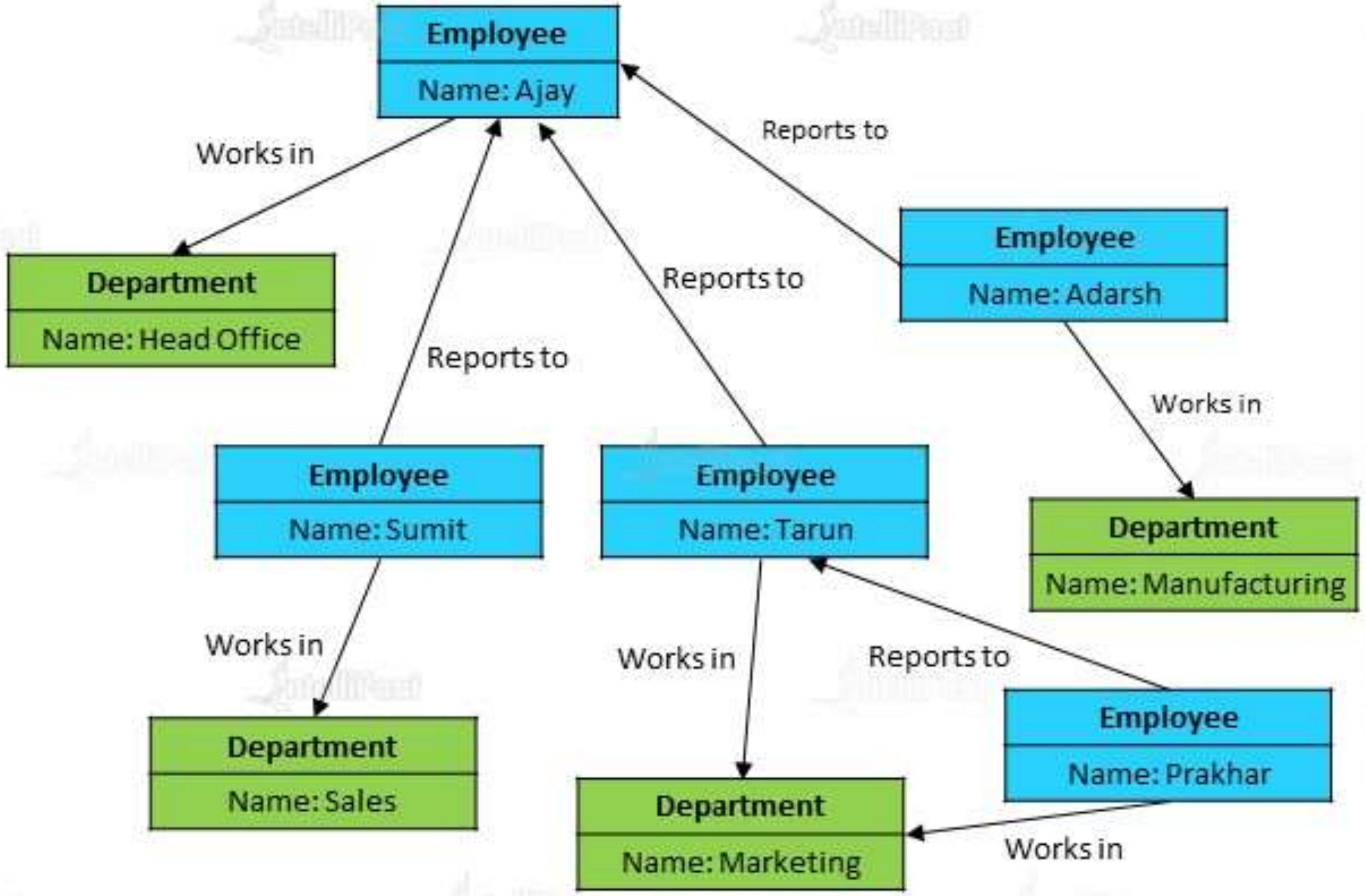


Graph Data Stores

The following diagram shows an organization's personal data structured as a graph



Graph Data Stores



- Entities are employees and departments
- Edges indicate reporting relationships and the department in which employees work
- In this graph, the arrows on the edges show the direction of the relationships

Graph Data Stores

Requirements for Graph Data Stores

Requirement	Graph Data
Normalization	Normalized
Schema	Schema on read
Consistency (across concurrent transactions)	Graph-level guarantee
Atomicity (transaction scope)	Graph-level guarantee
Locking Strategy	NA
Access Pattern	Random access

Graph Data Stores

Requirements for Graph Data Stores

Requirement	Graph Data
Indexing	Primary and secondary indexes
Data Shape	A graph containing edges and vertices
Sparse	No
Wide (lots of columns/attributes)	No
Data Size	Small (KBs)
Overall Maximum Scale	Large (TBs)

Graph Data Stores

Relevant Azure Service: Azure Cosmos DB Graph API

Time Series Data Stores

Time Series Data Stores

- Time series data is a set of values organized by time, and a time series data store is optimized for this type of data
- These data stores must support a very high number of writes as they typically collect large amounts of data in real time from a large number of sources
- Time series data stores are optimized for storing telemetry data

timestamp	deviceid	value
2019-01-05T 08:00:00.123	1	90.0
2019-01-05T 08:00:01.225	2	75.0
2019-01-05T 08:01:01.525	2	78.0

Time Series Data Stores

- Scenarios include IoT sensors and application/system counters
- Updates are rare; whereas, deletes are often done in bulk
- Although the records written to a time series database are generally small, they store often a large number of records, and the total data size can thus grow rapidly

Time Series Data Stores

★ **Relevant Azure Services**

- Azure Time Series Insights
- OpenTSDB with HBase on HDInsight

Time Series Data Stores

Requirements for Time Series Data Stores

Requirement	Time Series Data
Normalization	Normalized
Schema	Schema on read
Consistency (across concurrent transactions)	NA
Atomicity (transaction scope)	NA
Locking Strategy	NA
Access Pattern	Random access and aggregation

Time Series Data Stores

Requirements for Time Series Data Stores

Requirement	Time Series Data
Indexing	Primary and secondary indexes
Data Shape	Tabular
Sparse	No
Wide (lots of columns/attributes)	No
Data Size	Small (KBs)
Overall Maximum Scale	Large (low TBs)

Object Data Stores

Object Data Stores

- ★ Object data stores are optimized for storing and retrieving large binary objects or blobs, such as images, text files, video and audio streams, large application data objects and documents, and virtual machine disk images

Object Data Stores

- An object consists of the stored data, some metadata, and a unique ID for accessing the object
- Object stores are designed to support files that are individually very large, and they provide large amounts of total storage to manage all files

Path	Blob	Metadata
/delays/2019/11/01/flights.csv	0XAABBCCDDEEF…	{created: 2019-11-02}
/delays/2019/11/02/flights.csv	0XAADDCCDDEEF…	{created:2019-11-03}
/delays/2019/11/03/flights.csv	0XAEBBDEDDEEF…	{created: 2019-11-03}

Object Data Stores

- One special benefit of object data stores is its network file share
- Using this enables files to be accessed across a network using the standard networking protocols like server message block (SMB)

Object Data Stores

★ **Relevant Azure Services**

- Azure Blob Storage
- Azure Data Lake
- Azure File Storage

Object Data Stores

Requirements for Object Data Stores

Requirement	Object Data
Normalization	Denormalized
Schema	Schema on read
Consistency (across concurrent transactions)	NA
Atomicity (transaction scope)	Object
Locking Strategy	Pessimistic (blob locks)
Access Pattern	Sequential access

Object Data Stores

Requirements for Object Data Stores

Requirement	Object Data
Indexing	Primary index only
Data Shape	Blob and metadata
Sparse	NA
Wide (lots of columns/attributes)	Yes
Data Size	Large (GBs) to very large (TBs)
Overall Maximum Scale	Very large (PBs)

External Index Data Stores

External Index Data Stores

- External index data stores allows us to search for information held in other data stores and services
- An external index acts as a secondary index for any data store. It can be used to index massive volumes of data and provide near real-time access to these indexes

External Index Data Stores

- An external index lets us create secondary search indexes and then quickly finds the path to the files that match our criteria
- We can build a secondary index based on the values in the data and quickly look up for the key that uniquely identifies each matched item

id	search-document
233358	{"name": "Pacific Crest National Scenic Trail", "country": "San Diego", "elevation": 1294, "location":{"type":"Point","coordinates":[-120.802102,49.00021]}}

External Index Data Stores

- Indexes are created by running an indexing process
- This can be performed using a pull model, triggered by the data store, or using a push model, initiated by the application code
- The indexes can be multidimensional and may support free-text searches across large volumes of text data

id	search-document
233358	{"name": "Pacific Crest National Scenic Trail", "country": "San Diego", "elevation": 1294, "location":{"type":"Point","coordinates":[-120.802102,49.00021]}}

External Index Data Stores

1

External index data stores are often used to support full-text and web-based search

2

In these cases, searching can be exact or fuzzy

3

A fuzzy search finds documents that match a set of terms and calculates how closely they match

External Index Data Stores

Relevant Azure Service: Azure Search

External Index Data Stores

Requirements for External Index Data Stores

Requirement	External Index Data
Normalization	Denormalized
Schema	Schema on write
Consistency (across concurrent transactions)	NA
Atomicity (transaction scope)	NA
Locking Strategy	NA
Access Pattern	Random access

External Index Data Stores

Requirements for External Index Data Stores

Requirement	External Index Data
Indexing	NA
Data Shape	Document
Sparse	No
Wide (lots of columns/attributes)	Yes
Data Size	Small (KBs)
Overall Maximum Scale	Large (low TBs)

Why NoSQL or Non-relational DB?

Why NoSQL or Non-relational DB?

NoSQL databases help IT pros and developers manage the new challenges of ever-expanding diversity of data types and models

They are highly effective at handling unpredictable data, often with blazing-fast query speeds

They also provide a smooth database migration to the cloud for the existing NoSQL workloads

Why NoSQL or Non-relational DB?

Develop with Agility

- With the ability to respond to unplanned situations, NoSQL DBs cater to frequent software release cycles and are suitable for faster and more agile app development

Why NoSQL or Non-relational DB?

Handle Data with Flexibility

- NoSQL gives developers more freedom, speed, and flexibility to change both schema and queries to adapt to data requirements
- Information stored as an aggregate makes it easier for quick iterative improvements—without having to do any up-front schema design

Why NoSQL or Non-relational DB?

Operate at Any Scale

- NoSQL DBs can provide compelling operational advantages and savings with the ability to 'scale out' horizontally—or add less expensive servers without having to upgrade
- They can scale to handle more data or hold a single, large database within highly distributable clusters of servers

When to Choose a NoSQL or Non-relational DB?

Best Uses

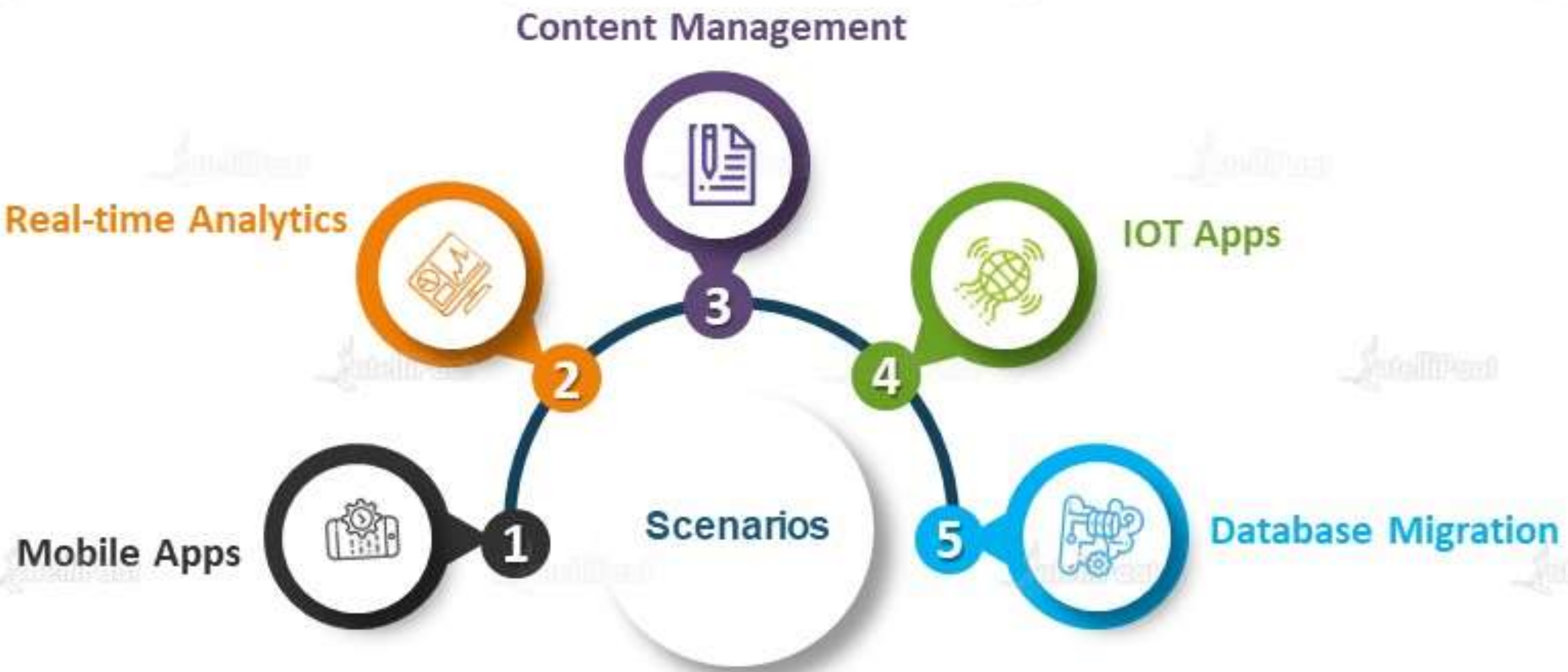
Always-on apps that serve users around the world

Handling large, unrelated, indeterminate, or rapidly changing data

Apps where performance and availability are more important than strong consistency

Schema-agnostic data or schema dictated by the app

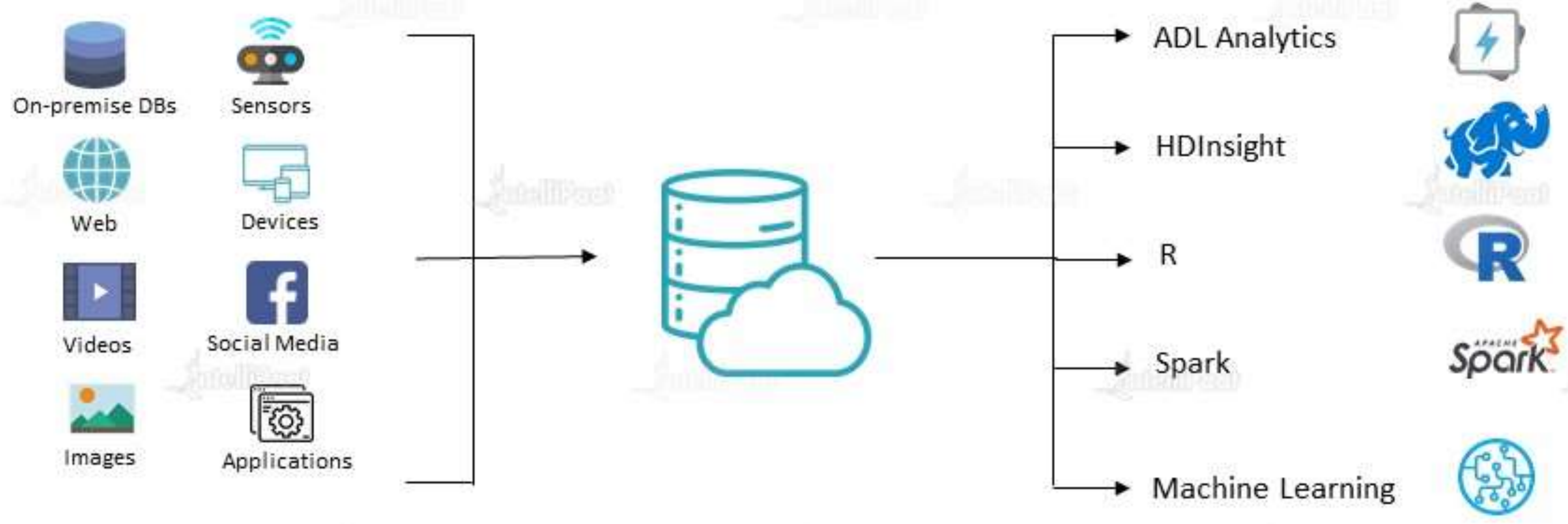
Scenarios


Content Management
Real-time Analytics
IOT Apps
3
2
4
1
5
Scenarios
Mobile Apps
Database Migration

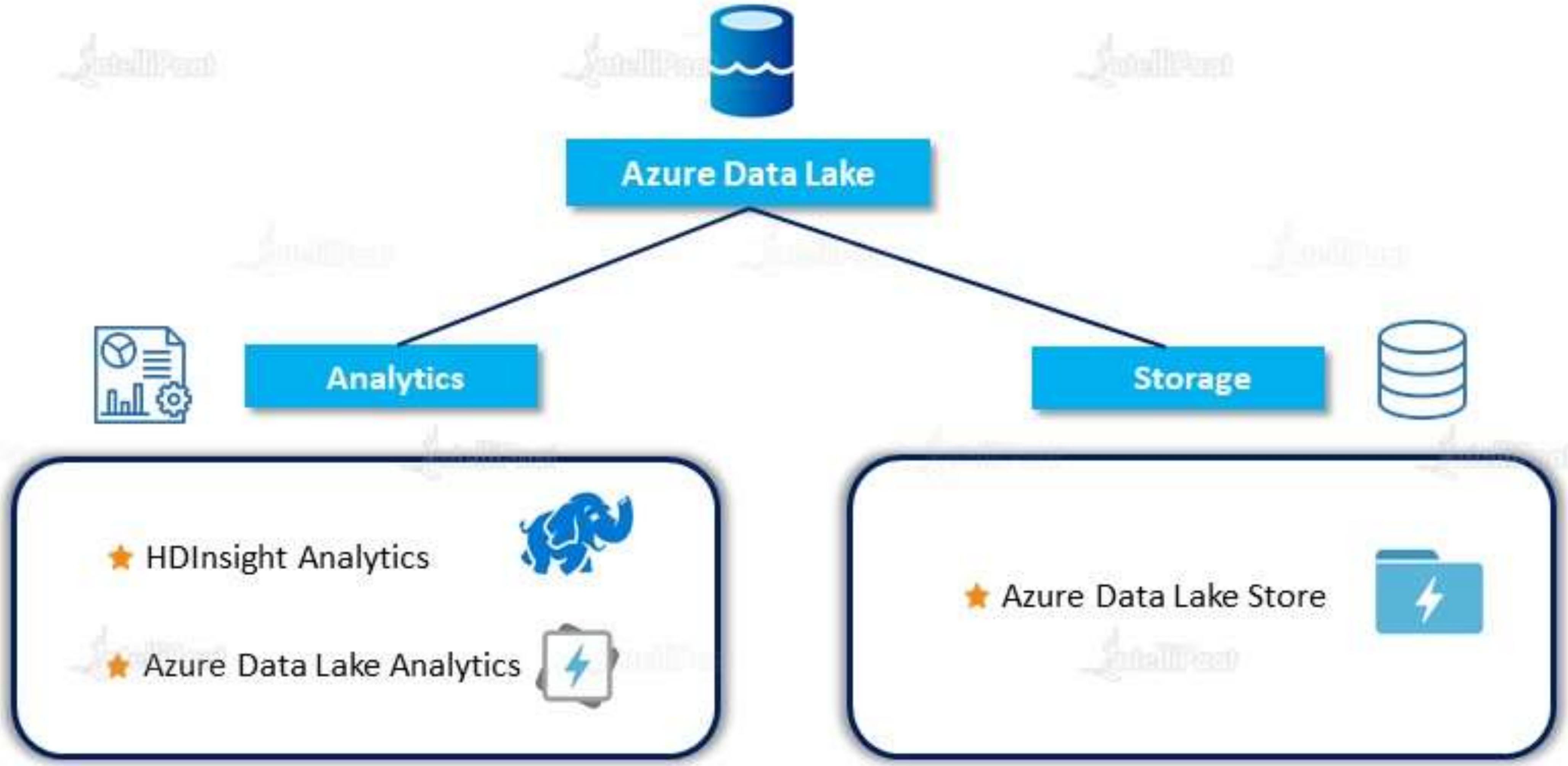
Azure Data Lake Storage

Azure Data Lake Storage

A highly scalable, distributed, parallel file system in cloud specifically designed to work with multiple frameworks



Azure Data Lake: Key Components



Azure Data Lake

Any Data

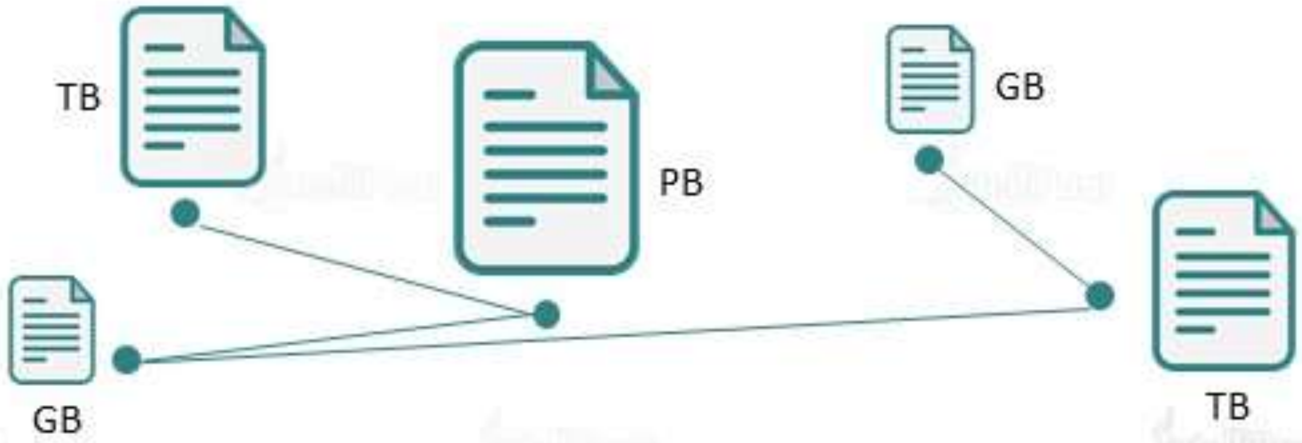
It can store any data without restriction and prior understanding

- **Structured Data:** Fixed fields and fixed schema
- **Semi-structured Data:** Self-describing data, no formal structure
- **Unstructured Data:** No predefined structure and not organized

Azure Data Lake

Any Size

There is no limit



- ★ No limit to scale
- ★ No limit on an account
- ★ Single data file size ranges from Gigabytes to Petabytes

Azure Data Lake

How does it store data?

- Azure Data Lake Store uses **WebHDFS** for storing data
- It supports **parallel reads and writes**
- If possible files are split into **2 GB chunks** called **extents**
- Just like Hadoop, **extents** are replicated three times for **availability** and **reliability**
- **Vertices** (when processing with **U-SQL**) are created based on extents

Azure Data Lake Storage Gen2

- A set of capabilities dedicated to Big Data Analytics, built on Azure Blob Storage
- The result of converging the capabilities of two existing storage services, **Azure Blob Storage** and **Azure Data Lake Storage Gen1**
- Features from Azure Data Lake Storage Gen1 are combined with low-cost, tiered storage, high-availability/disaster recovery capabilities from Azure Blob Storage

Why Data Lake?

Why Data Lake?

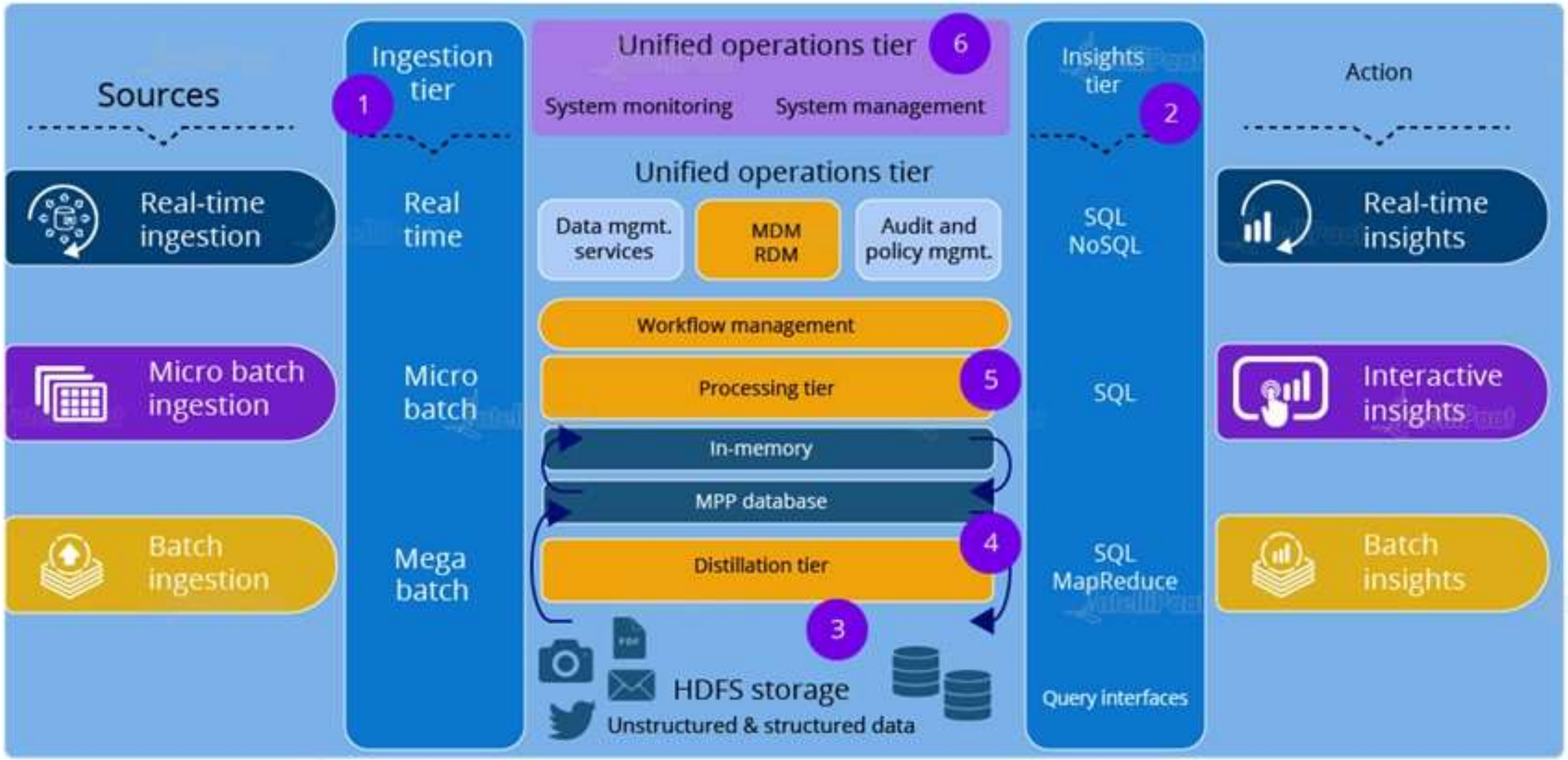
- The main objective of building a data lake is to offer an unrefined view of data to Data Scientists
- With the increase in data volume, data quality, and metadata, the quality of analyses also increases
- Data Lake offers business agility

Why Data Lake?

- Machine Learning and Artificial Intelligence can be used to make profitable predictions
- Data Lake gives a 360-degree view of customers and makes analysis more robust

Data Lake Architecture

Data Lake Architecture



Important Tiers in Data Lake Architecture

- **Ingestion Tier**: The tiers on the left side depict the data sources. The data could be loaded into Data Lake in batches or in real time
- **Insights Tier**: The tiers on the right represent the research side where insights from the system are used. SQL, NoSQL queries, or even Excel could be used for data analysis
- **HDFS**: It is a cost-effective solution for both structured and unstructured data. It is a landing zone for all data that is at rest in the system

Important Tiers in Data Lake Architecture

- **Distillation Tier** takes data from the storage tire and converts it into structured data for easier analysis
- **Processing Tier** runs analytical algorithms and user queries with varying Quality of Service(real time, interactive, batch) to generate structured data for easier analysis by downstream applications
- **Unified Operations Tier** governs system management and monitoring. It includes audit and proficiency management, data management, and workflow management

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor