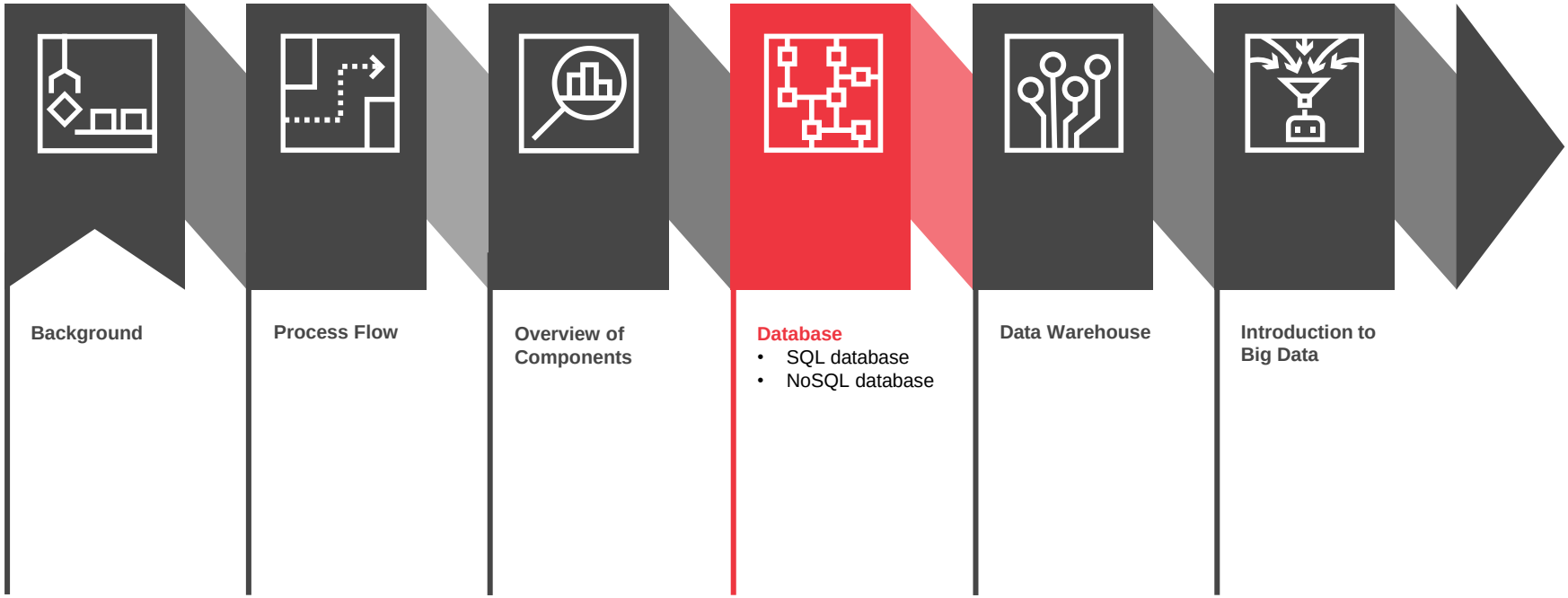
Maltem Consulting Group

Data Engineering and Big Data

Agenda

Date: Tuesday, 28 September 2021		Time: 9:00 am to 05:00am	ITE Campus	
	Agenda	Presenter	Time	Duration (mins)
1	MongoDB Installtion	Ben	9:00 AM 10:00 AM	60
2	Catch up	Mahesh	10:00 AM 10:30 AM	30
3	Break	Ben	10:15 AM 10:45 AM	30
4	SQL DataBase	Mahesh	10:45 AM 12:15 PM	90
5	Lunch Break	Mahesh	12:15 PM 1:15 PM	60
6	SQL Lab Work	Ben	1:15 PM 2:15 PM	60
7	NOSQL Database	Mahesh	2:15 PM 2:45 PM	30
8	Break	Ben	2:45 PM 3:15 PM	30
9	Introduction to Big Data	Mahesh	3:15 PM 3:45 PM	30
10	NOSQL Lab Work	Mahesh	3:45 PM 4:45 PM	60

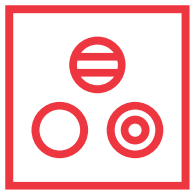
Day 2


Background
Process Flow
Overview of Components
Database
• SQL database
• NoSQL database
Data Warehouse
Introduction to Big Data

Types of Databases

There are two type of database normally data engineers deals with:

Relational / SQL databases



Distributed / NoSQL databases

SQL Databases

SQL databases have a **structure** that allows us to identify and access **data in a relation** to another part of data in the database. Often, data is organized into **tables, row** and **columns**

Advantages:

- **Easy to use:** have rows and columns like Excel
- **Portable**: the same query can run on different flavours of SQL DB with minimal changes
- **Have well defined standards:** ANSI (American National Standard Institutes)
- Intuitive query language

Disadvantages:

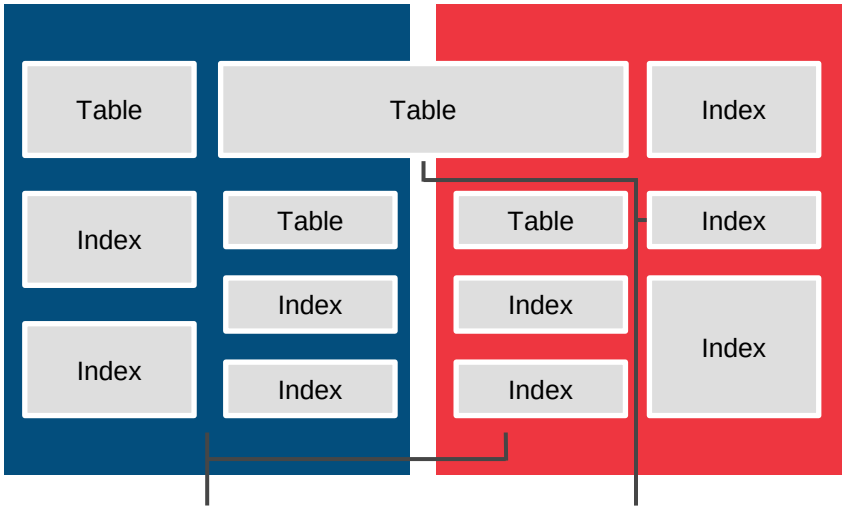
- **Performance:** vertical scaling
- Not suitable for modern applications e.g. IoT, Big Data

Some familiar SQL databases:

Relational database terminologies

- **Data file**: **Physical file** on **disk** created by the database application which contains the data
- **Schema**: **Collection of objects** in a database, including logical structures such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links
- **Table**: **Basic unit** of data storage. Data in tables is stored in **rows** and **columns** just like a tab in Excel
- **View**: A custom-tailored **presentation** of the data from **one** or **more tables**. Views do **not** actually store **data**, but derive it from the tables referenced in the view definition
- **Index**: An object that is used for **fast** and **efficient access** to stored information. Much like a table of contents in a book or a hyperlink

Tablespace
(one or more datafiles)

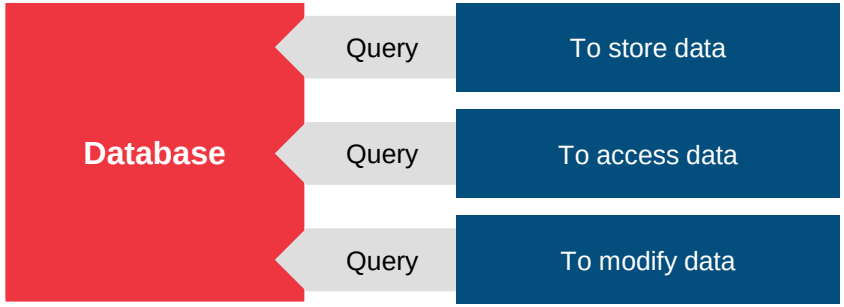


Datafiles
(physical structures associated with only one tablescape)

Objects
(stored in tablespaces-may span several datafiles)

SQL Language

- SQL stands for **Structured Query Language**
- SQL is a standard language for **accessing** and **manipulating** databases
- Most database systems use SQL, some also have their own **additional proprietary extensions**
- SQL allows a **less-technical user** to use "**English-like**" language to "query" databases



Type of SQL statement

- **Data Definition Language (DDL)** – CREATE, DROP, RENAME, TRUNCATE
- **Data Manipulation Language (DML)** – SELECT, INSERT, UPDATE, DELETE
- **Data Control Language (DCL)** – GRANT, REVOKE
- **Transaction Control Language (TCL)** – COMMIT, ROLLBACK

SQL CRUD Operations

Maltem Consulting Group

As a data engineer, you will be **frequently performing operations** that requires you to **read** from the database, **write** into the database, **update** the set of records in the tables and **delete** the records/transactions which are not required from the database.

CRUD is the acronym for the four basic commands in SQL -

C- Create command to create tables.

R- Read command to read data from tables.

U- Update command to update values in tables.

D- Delete command to delete rows from tables.

SQL Create

Maltem Consulting Group

Create command is used to **create a new table** in the specified **database**.

There are multiple ways to create a table.

Syntax:

```
CREATE TABLE tableName (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
)
```

```
SELECT column1, column2, column3, ...
INTO newtable
FROM oldtable
WHERE condition;
```

Example:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

```
SELECT AddressID,  AddressLine1,  AddressLine2, City,
PostalCode INTO  Address
FROM OLTP_Dataset.Person.Address
WHERE AddressID < 100;
```

Each **row** in a table represents an **entity** and each **column** stores the **attributes defining an entity**.
The above commands creates a tables named ‘Persons’ and ‘Address’, which stores data related to an entity in each row

SQL Read

Maltem Consulting Group

Read operation in SQL is done using **SELECT** statements

Syntax:

```
SELECT [DISTINCT] column
FROM mytable [JOIN another_table
ON mytable.column = another_table.column
WHERE constraint_expression]
[GROUP BY column]
[ORDER BY column ASC/DESC]
[LIMIT count OFFSET COUNT];
```

```
//select all the columns/fields from table 'Address' in 'Person' database
SELECT * FROM Person.Address;
```

Output:

Results | Spatial results | Messages

	AddressID	AddressLine1	AddressLin...	City	StateProvinc...	PostalCo...	SpatialLocation
1	1	1970 Napa Ct.	NULL	Bothell	79	98011	0xE6100000010CAE8BFC28BCE4474067A8918989
2	2	9833 Mt. Dias Blv.	NULL	Bothell	79	98011	0xE6100000010CD6FA851AE6D74740BC262A0A0
3	3	7484 Roundtree Drive	NULL	Bothell	79	98011	0xE6100000010C18E304C4ADE14740DA930C789
4	4	9539 Glenside Dr	NULL	Bothell	79	98011	0xE6100000010C813A0D5F9FDE474011A5C28A7
5	5	1226 Shoe St.	NULL	Bothell	79	98011	0xE6100000010C61C64D8ABBD94740C460EA3FD

SQL Read contd.

Maltem Consulting Group

Using WHERE clause example

```
//Get all the employee who are 'Design Engineer'
SELECT BusinessEntityID, JobTitle, LoginID FROM HumanResources.Employee WHERE JobTitle = 'Design Engineer';

//Get all the employee who are not 'Design Engineer'
SELECT BusinessEntityID, JobTitle, LoginID FROM HumanResources.Employee WHERE JobTitle <> 'Design Engineer';

//Display details of the persons whose record modified in data range
SELECT BusinessEntityID, FirstName, MiddleName, LastName, ModifiedDate FROM Person.Person
WHERE ModifiedDate BETWEEN '2009-01-01' AND '2013-12-31';

//Display Product Id and Its Name where name of the product has 'Bike' string
SELECT ProductID, Name FROM Production.Product WHERE Name LIKE '%Bike%';
```

Using GROUP BY clause example

```
//Get the total number of item ordered for each product
SELECT SUM(OrderQty) AS Total, ProductID FROM Sales.SalesOrderDetail GROUP BY ProductID;

//Display count of orders placed by year for each customer
SELECT CustomerID, COUNT(*) AS  SalesCount, YEAR(OrderDate) AS OrderYear FROM Sales.SalesOrderHeader GROUP BY CustomerID,
YEAR(OrderDate);

//Get the product Id ordered more than 5000 times
SELECT SUM(OrderQty) AS TotalOrdered, ProductID FROM Sales.SalesOrderDetail GROUP BY ProductID HAVING SUM(OrderQty) > 5000 ;
```

SQL Read contd.

Maltem Consulting Group

Using ORDER BY examples

```
//Sort the record by last name Ascending order
SELECT BusinessEntityID, LastName, FirstName, MiddleName FROM Person.Person
ORDER BY LastName ASC

//Sort the record by last name Decending order
SELECT BusinessEntityID, LastName, FirstName, MiddleName FROM Person.Person
ORDER BY LastName DESC
```

Using JOIN examples

To **Join** any **two or more tables** there should be **relationship** between them which is defined by below keys

- **Primary Key**- A field in the table that **uniquely identify** a record in the table.
- **Foreign Key**- A field in the table that is **primary key in another table**.

```
//Get personal information of the person working in a company as an employee
SELECT JobTitle, BirthDate, FirstName, LastName FROM HumanResources.Employee AS E
INNER JOIN Person.Person AS P ON E.BusinessEntityID = P.BusinessEntityID;

//Displays all the products along with the SalesOrderID even if an order has never been placed for that product
SELECT SalesOrderID, P.ProductID, P.Name FROM Production.Product AS P LEFT JOIN Sales.SalesOrderDetail AS SO ON P.ProductID =
SO.ProductID;
```

SQL Read contd.

Maltem Consulting Group

Using all the clauses we learnt..

Select city, postal code and modified date from table ‘Address’ and AddressTypeID from table ‘BusinessEntityAddress’, where AddressTypeID is equal to 3, grouping the result by city, postal code, modified date, address type id. The final result is sorted by postal code in descending order.

```
SELECT A.City, A.PostalCode, A.ModifiedDate, B.AddressTypeID
FROM Person.Address as A
JOIN  Person.BusinessEntityAddress as B
ON A.AddressID=B.AddressID
WHERE B.AddressTypeID=3
GROUP BY A.City, A.PostalCode, A.ModifiedDate, B.AddressTypeID
ORDER BY A.PostalCode DESC;
```

SQL Update

SQL **UPDATE** command **modifies** the **existing one** or **more records** in a table based on condition stated in WHERE clause.

Syntax:

```
UPDATE table_name SET column1 = value1, column2 = value2...., columnN = valueN WHERE [condition];
```

Example:

Update the column named ‘City’ for ‘AddressID’ = 2

Results | Spatial results | Messages

	AddressID	AddressLine1	AddressLin...	City	StateProvinc...	PostalCo...	SpatialLocation
1	1	1970 Napa Ct.	NULL	Bothell	79	98011	0xE6100000010
2	2	9833 Mt. Dias Blv.	NULL	Bothell	79	98011	0xE6100000010

```
UPDATE Person.Address SET City='Boston' WHERE AddressID=2;
```

Messages

```
(1 row(s) affected)
```

Results | Spatial results | Messages

	Address...	AddressLine1	AddressLin...	City	StateProvinc...	PostalCo...	SpatialLocat
1	1	1970 Napa Ct.	NULL	Bothell	79	98011	0xE610000(
2	2	9833 Mt. Dias Blv.	NULL	Boston	79	98011	0xE610000(
3	3	7484 Roundtree Drive	NULL	Bothell	79	98011	0xE610000(

SQL Delete

The SQL **DELETE** command is used to **delete the existing one or more records** from a table based on a condition.

Syntax:

```
DELETE FROM table_name WHERE [condition];
```

Example:

```
SELECT * FROM Person.Address;
```

Results | Spatial results | Messages

	Address...	AddressLine1	AddressLin...	City	StateProvinc...	PostalCo...	SpatialLocation
1	1	1970 Napa Ct.	NULL	Bothell	79	98011	0xE6100000010CAE8BFC28E
2	2	9833 Mt. Dias Blv.	NULL	Boston	79	98011	0xE6100000010CD6FA851AE
3	3	7484 Roundtree Drive	NULL	Bothell	79	98011	0xE6100000010C18E304C4A
4	4	9539 Glenside Dr	NULL	Bothell	79	98011	0xE6100000010C813A0D5F9

```
DELETE FROM Person.Address WHERE City='Boston';
```

Results | Spatial results | Messages

	Address...	AddressLine1	AddressLin...	City	StateProvinc...	PostalCo...	SpatialLocation
1	1	1970 Napa Ct.	NULL	Bothell	79	98011	0xE6100000010CAE8BF
2	3	7484 Roundtree Drive	NULL	Bothell	79	98011	0xE6100000010C18E30
3	4	9539 Glenside Dr	NULL	Bothell	79	98011	0xE6100000010C813A0
4	5	1226 Shoe St.	NULL	Bothell	79	98011	0xE6100000010C61C64

Workshop on Relational Databases

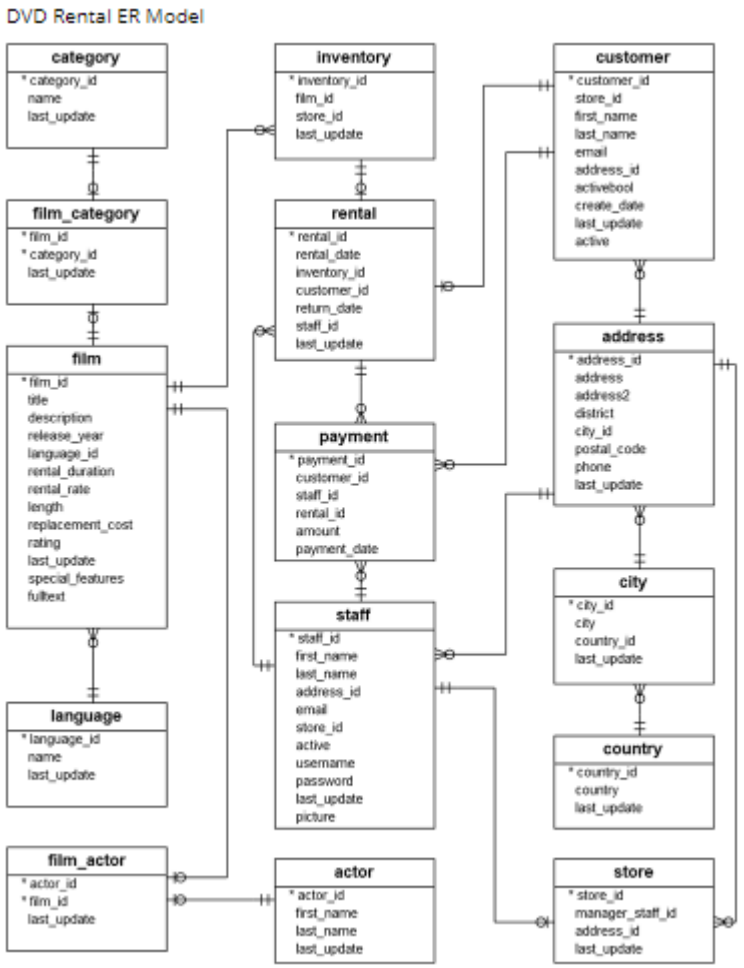
Maltem Consulting Group

Case Scenario:

The database DvdRental has 15 tables. Below are the different tables and a brief description of them.

Questions:

- What are the top and least rented (in-demand) genres and what are their total sales?
- Can we know how many distinct users have rented each genre?
- What is the average rental rate for each genre? (from the highest to the lowest)
- How many rented films were returned late, early, and on time?
- In which countries does Rent A Film have a presence and what is the customer base in each country? What are the total sales in each country? (from most to least)
- Who are the top 5 customers per total sales and can we get their details just in case Rent A Film wants to reward them?



NoSQL Databases

Maltem Consulting Group

Are **non-relational** databases with **specific data models, flexible schemas** and **scale horizontally**

Advantages:

- **Schema-less**: provide a higher level of flexibility with newer data models
- **Open Source & Low-Cost**: go-to solution for organizations with limited budgets
- **Elastic scalability**: NoSQL databases are designed to function on **full throttle even with low-cost hardware**
- **Less development time**: create a database without needing to develop a **detailed** (fine-grained) **database model**

Disadvantages:

✓ community-driven, with enterprise support

✓ **Lack of standardization** like SQL

✓ Different database for different business cases

Familiar NoSQL Databases:

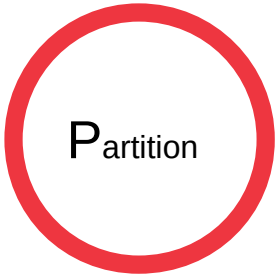
CAP Theorem

Maltem Consulting Group

CAP theorem states **3 basic requirements** when designing application in distributed architecture

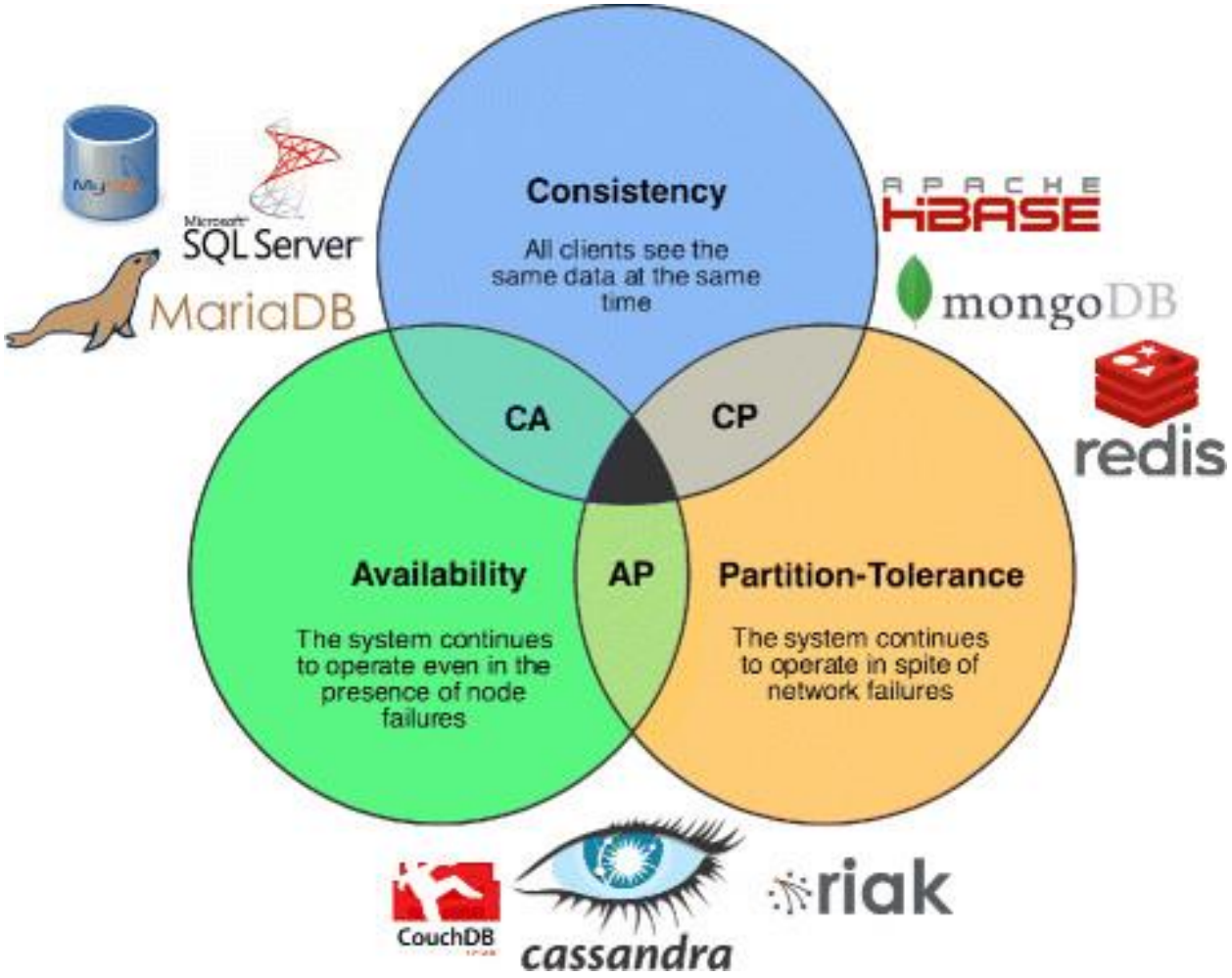
Data in the database will **remain consistent** after the execution of an **operation**. E.g. All client sees the same data after an update operation is performed

No downtime. This means the system will be always available to server the request from the client applications



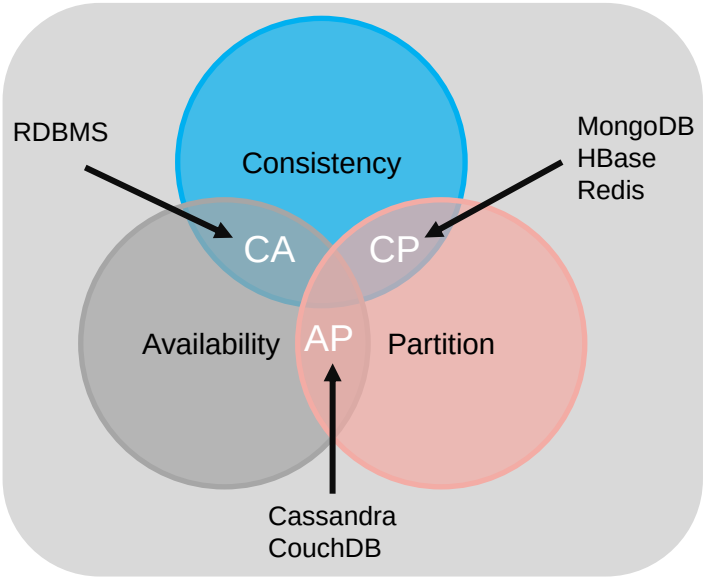
Data will **be partitioned into multiple group** and **replicated across multiple system**. System **continues to function** even though any one of the system is unreliable or loss communication among system

CAP Theorem contd.



CAP Theorem contd.

- CAP provides basic requirement to distributed system to **follow 2 of the 3 requirement**
- Theoretically, its **impossible** to follow all **3 requirement**
- Different NoSQL databases follow different combinations of the C, A, P from CAP theorem

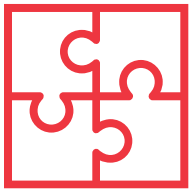


- **CA : Single system**, all the nodes are in always contact. System gets blocked when partition occurs
- **CP :** Accessing data is not possible when **master goes down**. But data will be **consistence and accurate**
- **AP :** System will be **available after partition**, but some **data** returned may **not be consistent**

Types of NoSQL Databases

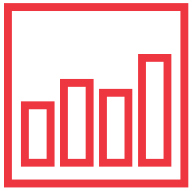
Maltem Consulting Group

NoSQL databases are often categorised in to four main types depending on the way data is stored

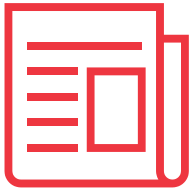


Key-Vale
A big hash table of key and values

Graph
Uses vertices and edges to store data



Columnar
Storage block contains data from one column



Document
Stores documents which are made up of tagged elements

Key-Value NoSQL Database

Maltem Consulting Group

- Uses an **associative array** (such as a map) where each key is associated with one and only one value in a collection
- In a Key-Value pair, each key value is represented as an arbitrary string such as a **hash value**
- Very **low latency** for read/writes and value stored as blob with **no upfront schema**
- Key-value stores **do not** have any **query language**. They only allow to store, retrieve and update data using **simple get, put** and **delete commands** and the data can be retrieved by making a direct request to the object in memory or on disk

Use Cases:

- Used for application which **maintains session information** and management at high scale
- Product recommendation - **latest product viewed** on a **website drive** newer product recommendation
- Store ads, coupons to push to customer in real-time

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

ORACLE BERKELEY DB

redis

Couchbase

riak

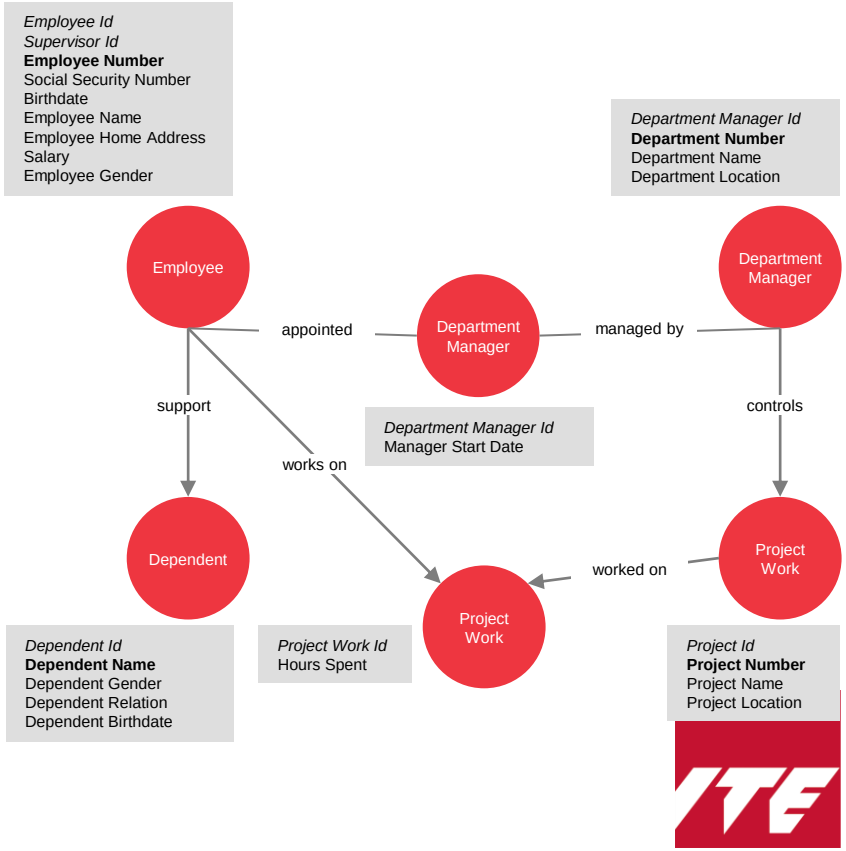
Graph NoSQL Database

- Flexible and scalable
- Stores data in the form of **vertices** and **edges**
- Vertices are instance of an object in an application having attributes defining its properties
- Relationships which are represented in the form of edges can be **unidirectional** or **bi-directional**, having one to one, one to many, many to one and many to many relationships
- Queries in graph database are **answered** on the basis of **relationships** and its **properties**, it is important to choose the relationship properly
- Easy to model and define relationship
- Performance of **relationship traversal** remains **constant** with data size growth
- Queries are shorter and more readable

JanusGraph TITAN neo4j

Use Cases:

- Fraud detection: determine relationship in transaction
- Social network: recommendations



Columnar NoSQL Database

- Stores data in **column families** as **rows** which contain the group of **associated data** that can be **accessed together**
- All columns are treated individually
- Each column stored **separately** on the **disk location**
- Stores data efficiently through data **compression** and **partitioning**.
- Supports higher writes compared to reads.
- Great when we need data **aggregations** (such as SUM, COUNT, AVG etc).
- Fast ad-hoc queries

Use Cases:

- Transaction analysis - Purchase pattern, movie location and watched history
- Time series data analysis
- Weather service analysis
- Internet of things and event analysis

Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Row layout

a1	b1	c1	a2	b2	c2	a3	b3	c3	a4	b4	c4	a5	b5	c5

Column layout

a1	a2	a3	a4	a5	b1	b2	b3	b4	b5	c1	c2	c3	c4	c5

Document NoSQL Database

- Similar to key-values databases, but the only difference is that it stores the values in the form **JSON** (Javascript object Notation), **BSON** (Binary encoding of JSON objects)
- We can store complex data like trees, collections, and dictionaries
- It does **not support relations**. Each document is standalone. It can refer to other documents by storing their key, corresponding to the particular document
- Joins are not support so it overcomes the problem of sharing the data across multiple nodes

Use Cases:

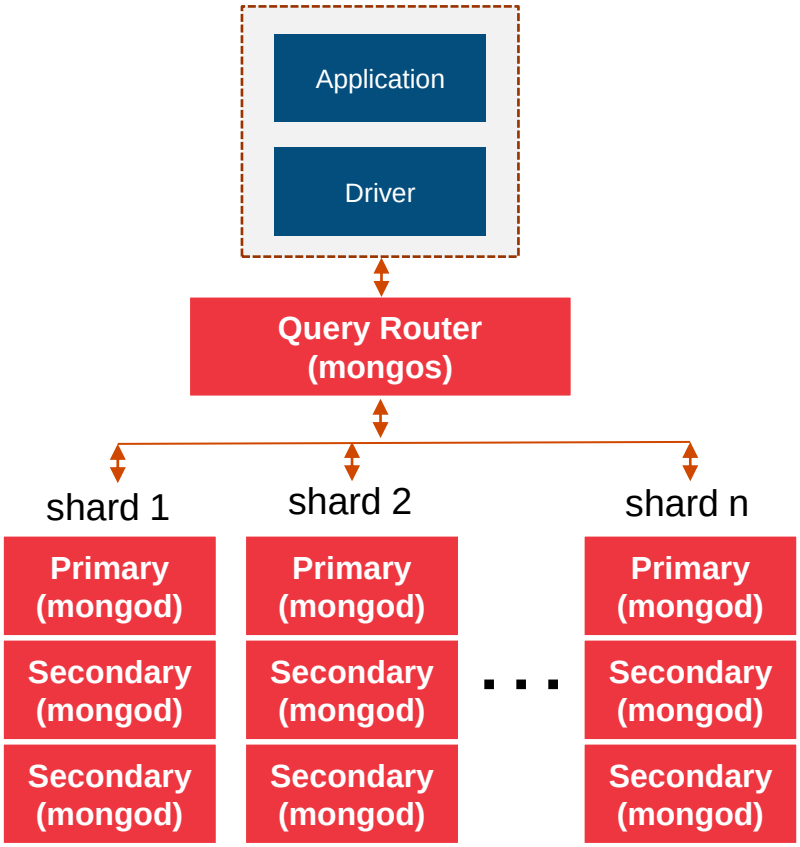
- Web pages - stores photos and video with social features of tagging, likes and comments
- Medical records– stores patient records with images of scans and reports
- Email – best suited to store unstructured data
- Research – Formulas and charts

Deciding factors to choose SQL vs NoSQL Databases

Maltem Consulting Group

Parameters	SQL	NoSQL
Definition	Relation database	Non-relational or distributed database
Type	Table	document, key-value pair, graph, columnar
Schema	Pre-defined Schema	Dynamic schema, schema-less
Ability to Scale	Vertically scalable	Horizontally scalable
Query Language	SQL	UnQL- unstructured query language varies from DB to DB
Standards	Emphasis on ACID (Atomicity, Consistency, Isolation, Durability)	Follow CAP theorem (Consistency, Availability, Partition)
Storage type	High available storage	Commodity drive storage
Open-source	Mix	All
Application/Use cases	• Online Banking • Order and sales management • Data warehousing	• Web applications • Social networks application • Internet of Things

MongoDB and its architecture



Driver: Drivers are the client libraries programs used to connect to mongodb instances

Mongos: It's a mongodb shard utility which acts a controller and query router for sharded clusters. Sharding means partition the data set into discrete parts

Mongod: This is a primary demon process which handles data requests, manages the data access and perform data management operations

Shard: Shard means partition the data set into discrete parts and store in multiple system which are interconnected

Features:

- Support ad hoc queries - Search by field, range query and regular expression searches.
- Indexing - we can index any field in a document.
- Replication – Master-Slave replication.
- Load balancing - Automatic load balancing configuration because of data placed in shards.
- Provides high performance.

Operations in MongoDB

Maltem Consulting Group

Similar to RDBMS, MongoDB support CRUD operations to create, read, update and delete the data.

Data in MongoDB is stored in BSON(Binary encoded JSON) document which provides flexibility in '**schema-less**' data model

Below table outlines common terminologies and concepts across RDBMS and MongoDB

RDBMS	MongoDB
Support ACID Transactions ACID stands for Atomicity, consistency, Isolation, Durability	Support ACID Transactions
Table	Collection
Row	Document
Column	Fields
JOINs	Embedded documents
GROUPBY	Aggregate pipeline

Create operation

MongoDB has two operations to **create or insert** the document to collection. If **collection doesn't exist**, it will **create the collection**

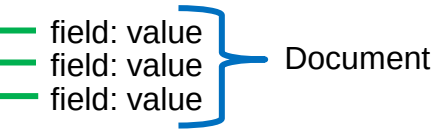
- insertOne()
- insertMany()

Syntax

```
db.collection.insertOne(<document>)
```

```
db.collection.insertMany([ <document 1> , <document 2>, ... ])
```

```
db.products.insertOne(
   {
      item: "mobile",
      qty: 5,
      price: 100
   }
)
```



Examples

```
//Create the collection and insert a document
db.products.insertOne( { item: "card", qty: 15 } );

//Insert one more document with different schema
db.products.insertOne( { item: "bike", qty: 5, price: 259 } );

//Insert document with _id
db.products.insertOne( { _id: 100, item: "mobile", qty: 10, price: 5000 } );
```

```
//Insert more than one document
db.products.insertMany( [
     { item: "charger", qty: 200 },
     { item: "head phone", qty: 50 },
     { item: "wireless charger", qty: 100 }
  ] );

//Insert multiple document with _id
db.products.insertMany( [
     { _id: 101, item: "envelopes", qty: 60 },
     { _id: 102 item: "stamps", qty: 110 },
     { _id: 103, item: "packing tape", qty: 38 }
  ] );
```

Update operation

Maltem Consulting Group

To modify the existing document in a collection we use update operations

- db.collection.updateOne()
- db.collection.updateMany()

Syntax

```
db.collection.updateOne(
   <filter>,
   <update>,
   {
      upsert: <boolean>
   }
)
```

```
db.collection.updateMany(
   <filter>,
   <update>>,
   {
      upsert: <boolean>
   }
)
```

Examples

```
//Update the quantity of the item 'envelopes' to 100
db.products.updateOne(
      { "item" : "envelopes" },
      { $set: { "qty" : 100 } }
   );

//Insert a new document if condition fails with upsert option
db.products.updateOne(
  { "charger" : { $eq: 0} },
  { $set: { "item": "charger", "sold" : true } },
  { upsert: true }
);
```

```
//check for the quantity greater than 100 and update the stock availability
db.products.updateMany(
  { qty: { $gt: 100 } },
  { $set: { "stock" : "available" } }
);

//Insert a new document if condition fails with upsert option
db.products.updateMany(
  { qty: { $gt: 100 }, "item" : "bottle" },
  { $set: { "stock" : "available" } },
  { upsert: true }
);
```

Read operation

Maltem Consulting Group

Read operation retrieves documents from a collection

- db.collection.find()

Syntax

```
db.collection.find(
   <query>,
   <projection>
)
```

```
//count the number of documents
db.biodata.count()

//returns at most 5 documents:
db.biodata.find().limit(5).pretty()

//skips the first 5 documents
db.biodata.find().skip(5).pretty()

//Return document where _id equals 5
db.biodata.find({ _id: 5 }).pretty()

//Get documents where the field last in the name embedded document equals "Hopper"
db.biodata.find({ "name.last": "Hopper" }).pretty()

//$in operator to return documents where _id equals either 1 or 5
db.biodata.find({ _id: { $in: [ 1,5 ] }}).pretty()

//get document where birth is greater than new Date('1950-01-01')
db.biodata.find({ birth: { $gt: new Date('1950-01-01')}}).pretty()
```

Select operation contd.

```
//Display result wher name.last field starts with the letter N (or is "LIKE N%")
db.biodata.find({ "name.last": { $regex: /^N/ } }).pretty()

//returns only the name field, contribs field and _id field(Projection)
db.biodata.find( { }, { name: 1, contribs: 1 } ).pretty()

//returns all fields except the first field in the name embedded document and the birth field
db.biodata.find({ contribs: 'OOP' },{ 'name.first': 0, birth: 0 }).pretty()

//returns all the documents where birth field is greater than new Date('1950-01-01') and death field does not exists
db.biodata.find( {birth: { $gt: new Date('1920-01-01') },death: { $exists: false }} ).pretty()

//returns documents name is exactly { first: "Yukihiro", last: "Matsumoto" }
db.biodata.find({"name.first": "Yukihiro", "name.last": "Matsumoto" }).pretty()

//returns documents where the array field contribs contains the element "ALGOL" or "Lisp"
db.biodata.find( { contribs: { $in: [ "ALGOL", "Lisp" ]} } ).pretty()

//return documents where contribs size equal to 3
db.biodata.find({contribs: {$size: 3}}).pretty()

//return documents where contribs toward “OOPS: and “Simula”
db.biodata.find({contribs: {$all: ["OOP", "Simula"]}}).pretty()

//returns documents where the awards array contains at least one element with both the award field equals "Turing Award" and the year field greater than 1980
db.biodata.find({ awards: { $elemMatch: { award: "Turing Award", year: { $gt: 1980 } } } }).pretty()

//return documents which match both the condition
db.biodata.find( { $and: [ { "name.first" : "John" }, { "awards.award" : "National Medal of Science" } ] } ).pretty()

//retrun document which match either of the condition
db.biodata.find( { $or: [ { "name.first" : "John" }, { "awards.award" : "National Medal of Science" } ] } ).pretty()
```

Delete operation

Maltem Consulting Group

To remove the document from a collection we use delete operations

- db.collection.deleteOne()
- db.collection.deleteMany()

Syntax

```
db.collection.deleteOne(
   <filter>
)
```

```
db.collection.deleteMany(
   <filter>
)
```

Examples

```
//Delete item
db.products.deleteOne(
      { "item" : "envelopes" }
);

//Delete using _id
db.products.deleteOne(
      { "_id" : "102" }
);
```

```
//delete all the item whose quantity greater than 100
db.products.deleteMany(
      { "qty" : { $gt : 100 } }
);
```

Workshop on NoSQL Databases

Maltem Consulting Group

Case Scenario:

We have a video database which as a collection ‘movies’ containing information about the name of the movie, which year it was produced, length of the movie, director, rating, cast and genres. We will load the data set into MongoDB Cloud (Atlas) and create and access the database using MongoDB Compass and Mongo Shell to answer below questions.

Questions:

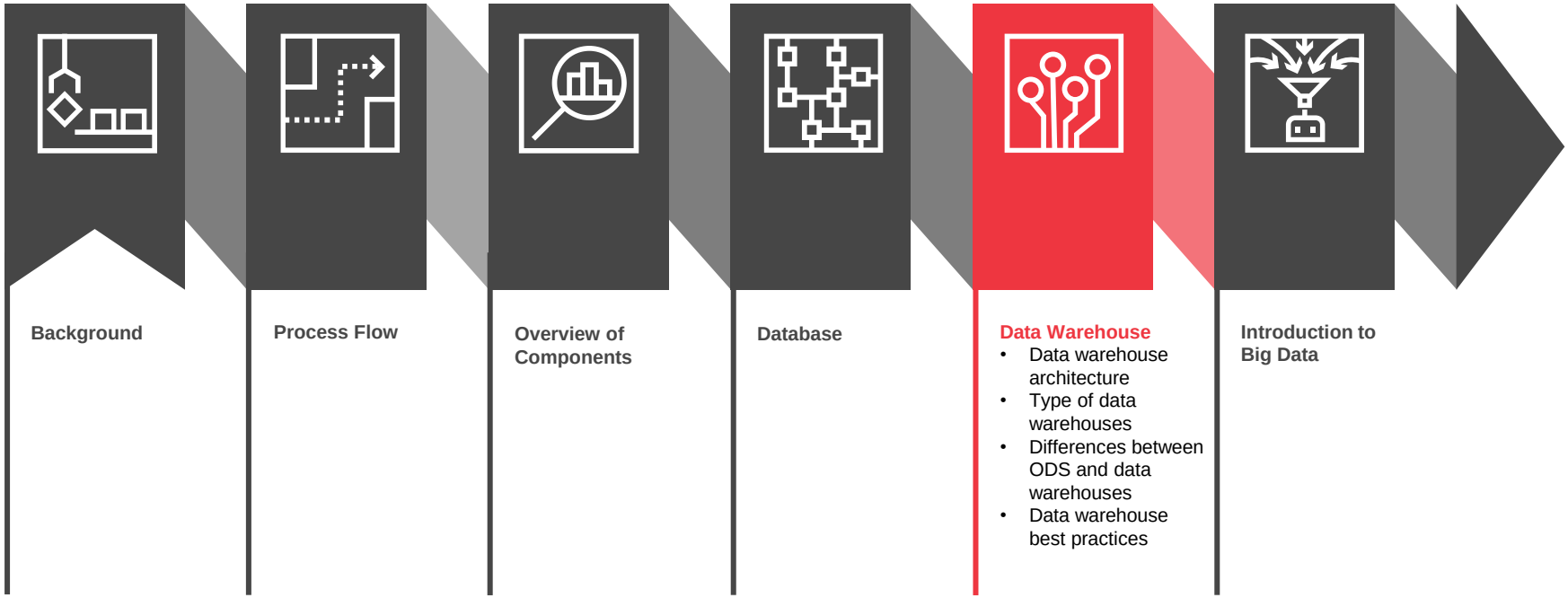
- How many documents lists whose runtime is greater then 90?
- Get movie list whose runtime is greater then 90 and less than 120 and only display title and runtime?
- How many movies in the movie collection are rated G, PG, PG-13 and display title, rated without _id?
- How many movies whose tomato meter greater than 95 and “metacritic” greater than 88?
- How many documents are there which has “mppaRating”?
- How many documents are there which doesn't have “mppaRating”?
- Get the documents where movie released in only one country?
- List all the movies which exactly match genres Comedy, Crime and Drama?
- Get the list of movies which as won Oscars?

Quick Quiz

Please describe a circumstance in which an organization would choose a non-relational database over a relational database

What are some advantages of SQL databases?

Day 2



Background

Process Flow

Overview of Components

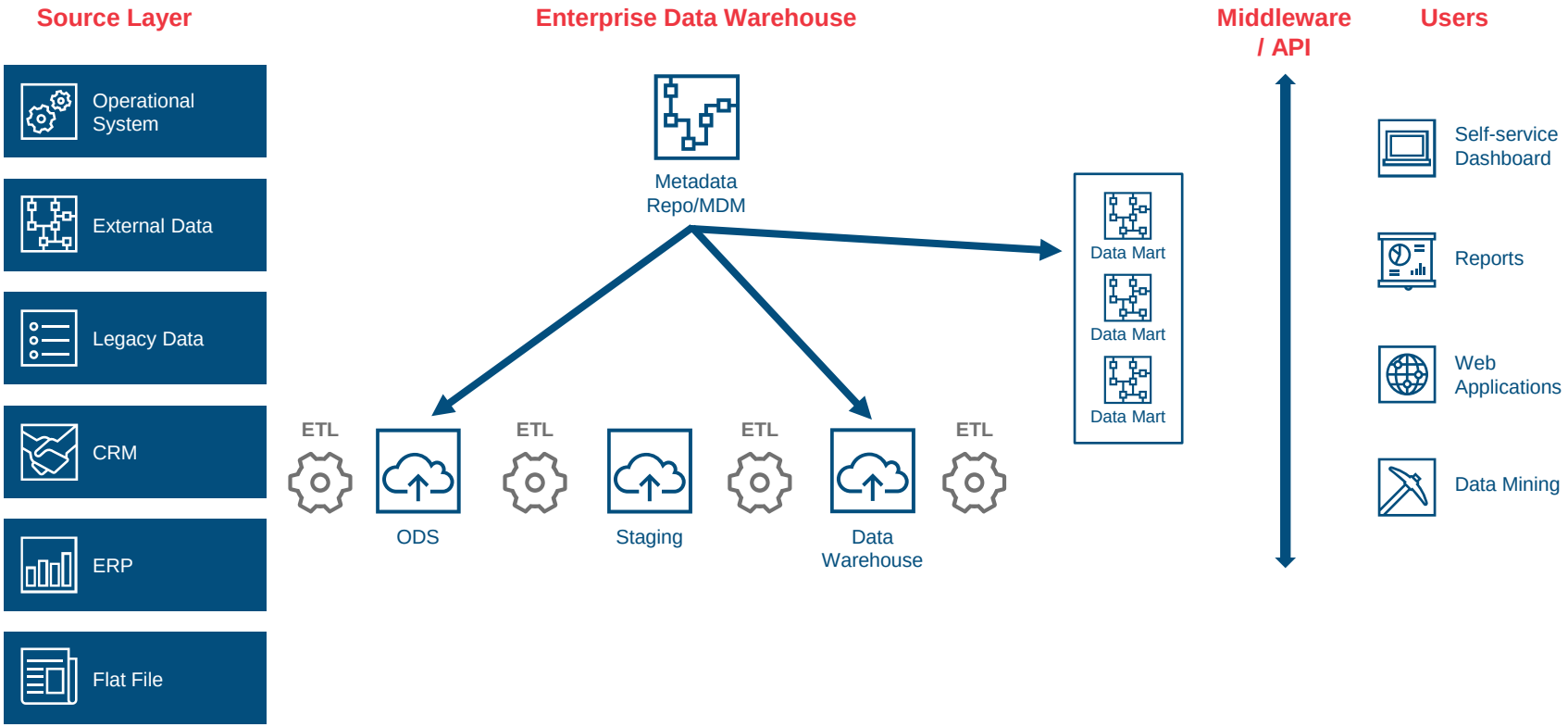
Database

Data Warehouse

- Data warehouse architecture
- Type of data warehouses
- Differences between ODS and data warehouses
- Data warehouse best practices

Introduction to Big Data

Data Warehouse architecture



Types of data warehouses

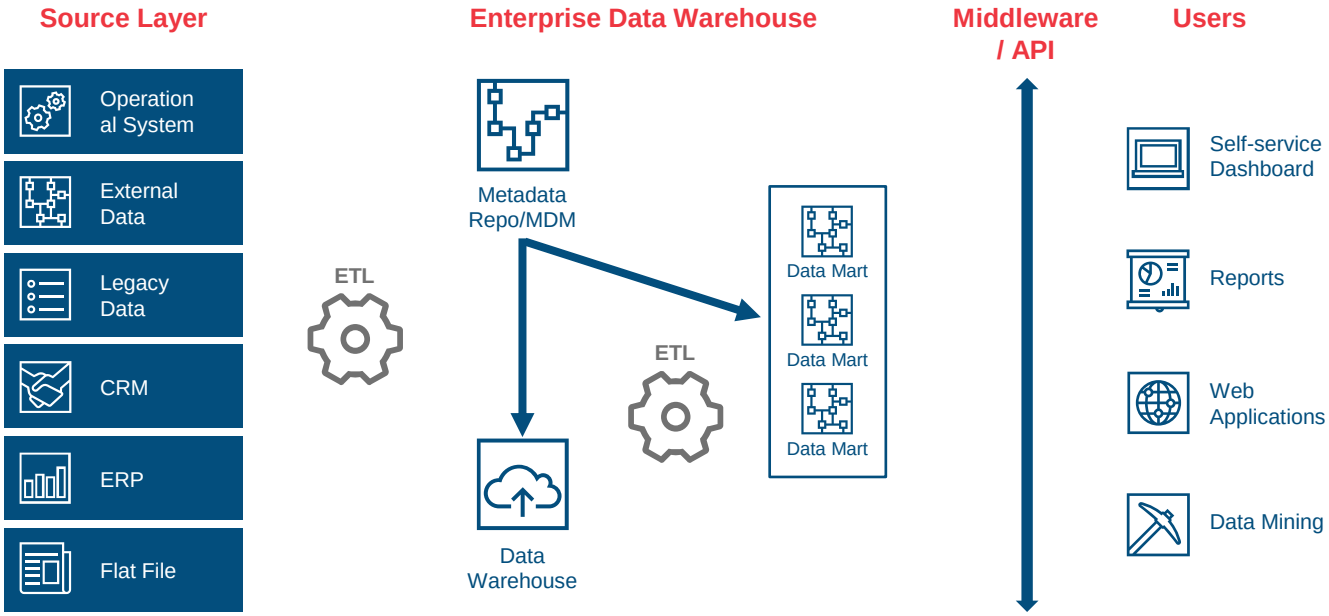
Centralized data warehouse
Data is stored in one centrally located data warehouse

Federated data warehouse
Data is stored in separate physical databases

Multi-tiered / Hybrid warehouse
Combination of centralized and federated data warehouses

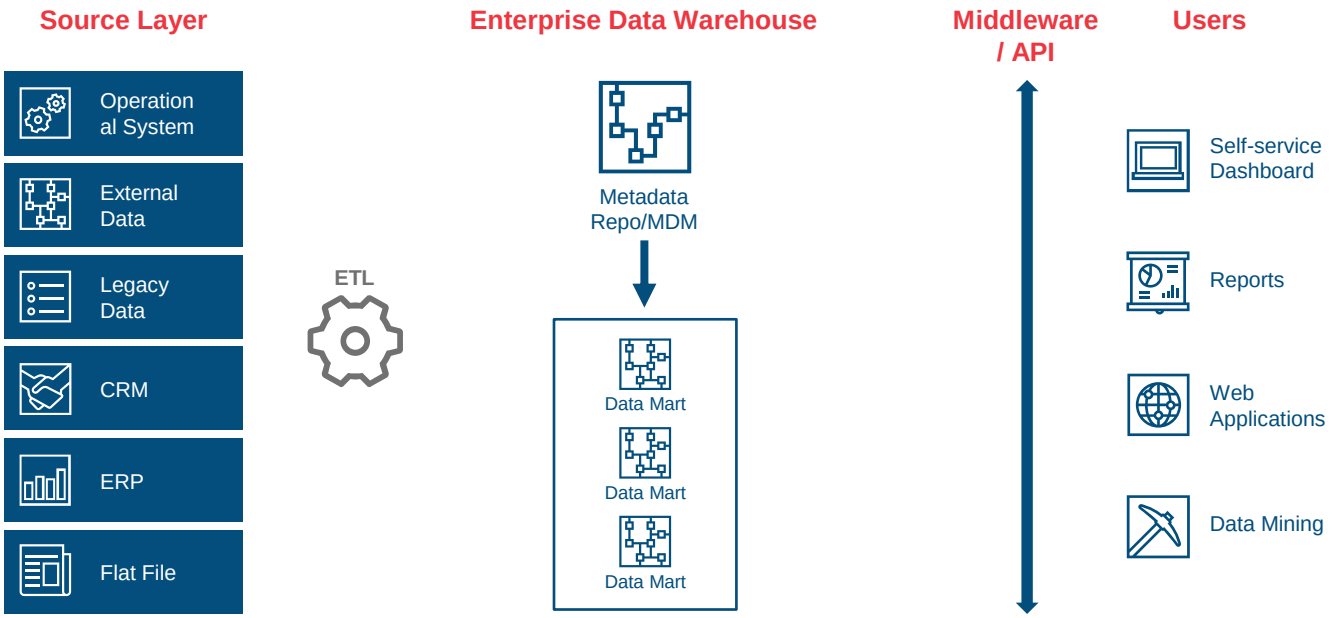
Centralized data warehouse

- Also known as an **Enterprise data warehouse**
- **Only one centrally located data repository**
- Data is **readily available** for **consumptions** due to **data preparation processes**
- **Faster to develop** new solutions for the business and merging different data sources is easier



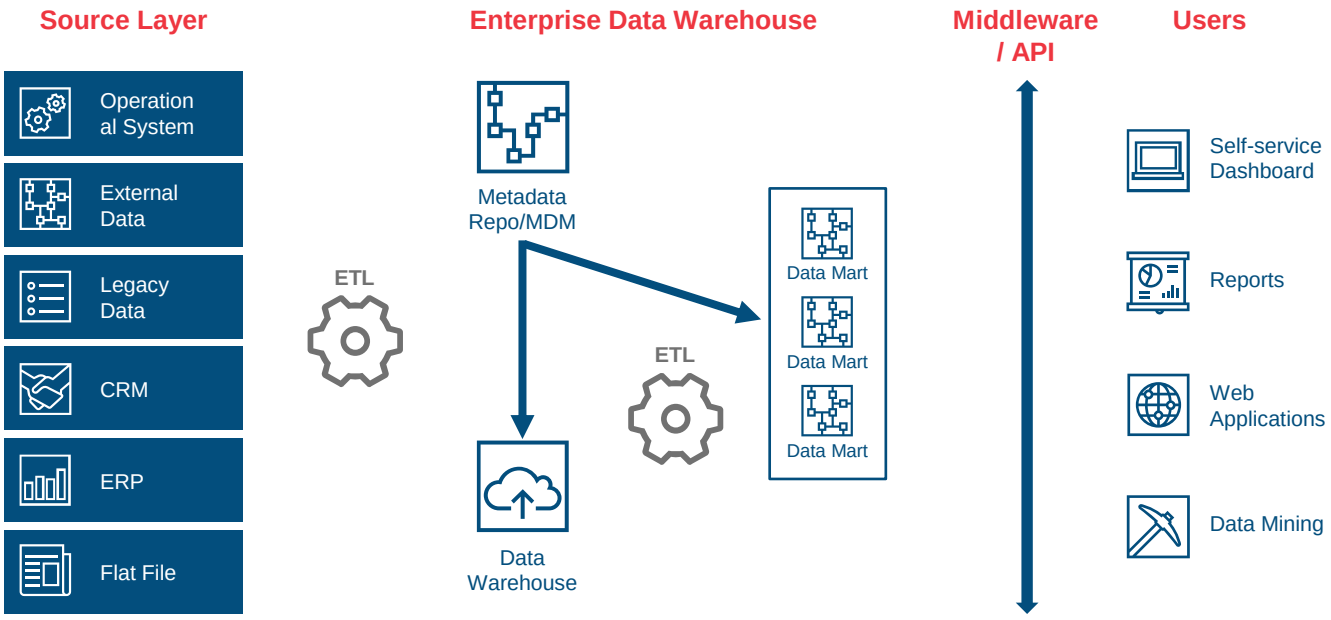
Federated data warehouse

- Also know as a **Single Department** data mart architecture
- Data is **logically consolidated** but stored in a **separate physical databases**
- Each database has **different server name, IP address, version** and can be at different locations
- Each **data mart** will store only **relevant information to a department/BUs**
- **Limited data size** compared to centralized architecture



Multi-tiered/Hybrid warehouse

- A distributed data approach
- A **combination of centralized** and **federated approach**
- Data from **different departments** can be used to **derive data marts**
- Data marts can be created for more specific areas like food, toys



Data warehousing best practices

Maltem Consulting Group

Define standards before starting – Templates for mapping, coding guidelines, documentation should be decided ahead. Timelines for status reports, release deliverables should be decided upfront before the development starts

Data Governance – Ensure data integrity and quality before the data is pushed into the data warehouse

User need based design – Primary focus of the data warehouse is to provide trusted information to the customers and address issues which are not fully articulated by the customers, this results in increased business

Faster delivery – Markets change rapidly so requirements gathering and delivery of the product should be done in an agile manner

Performance – Storage optimization and query performance tuning to account for the rapid growth of data

Adopt **agile data warehouse methodology** which breaks projects into smaller, faster deliverables

Automation – Automate the data pipeline/ETL process to leverage IT resource fully and iterate faster project execution

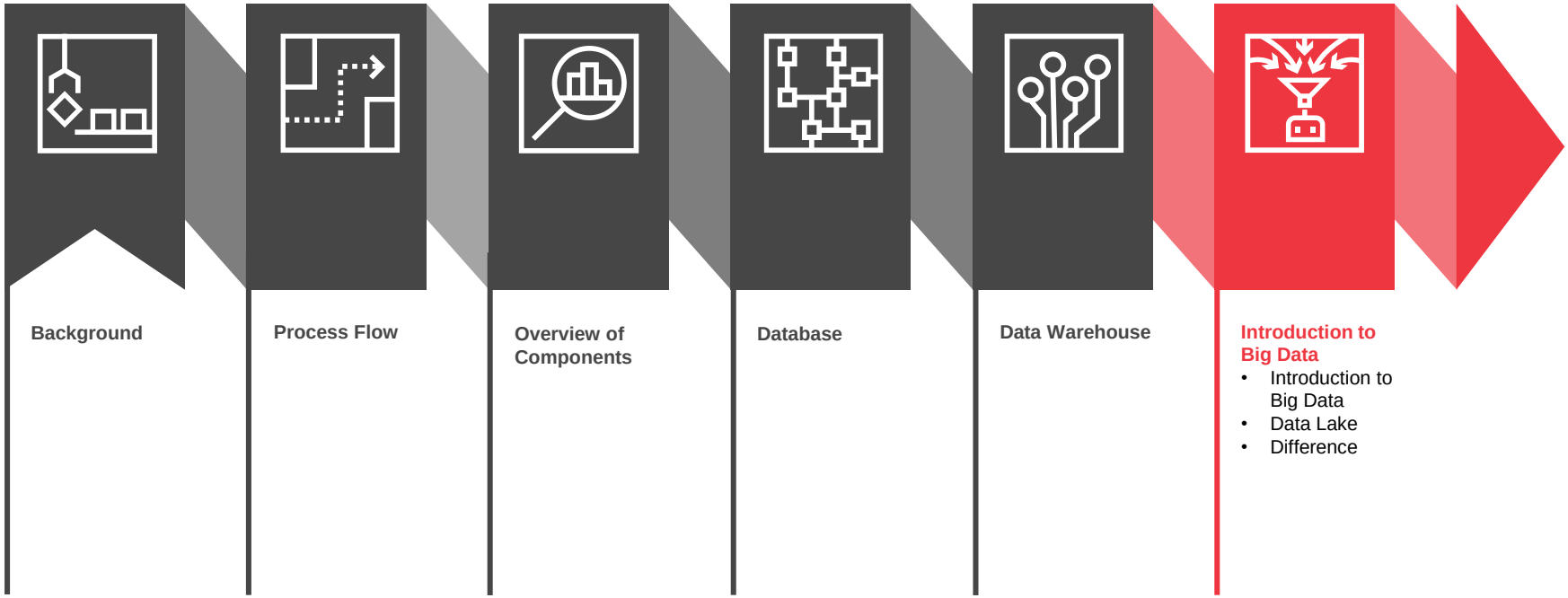
Quick Quiz

Maltem Consulting Group

Difference between centralized and federated data warehouse types

List the advantage and disadvantage of each type of database

Day 2



Background

Process Flow

Overview of Components

Database

Data Warehouse

Introduction to Big Data

- Introduction to Big Data
- Data Lake
- Difference

Introduction to Big Data

Maltem Consulting Group

Big Data is **high-volume, high-velocity** and/or **high-variety** information assets that demand **cost-effective, innovative** forms of information processing that enable enhanced insight, decision making, and process automation.
Source: Gartner

Is a 100GB data set a "Big Data"?

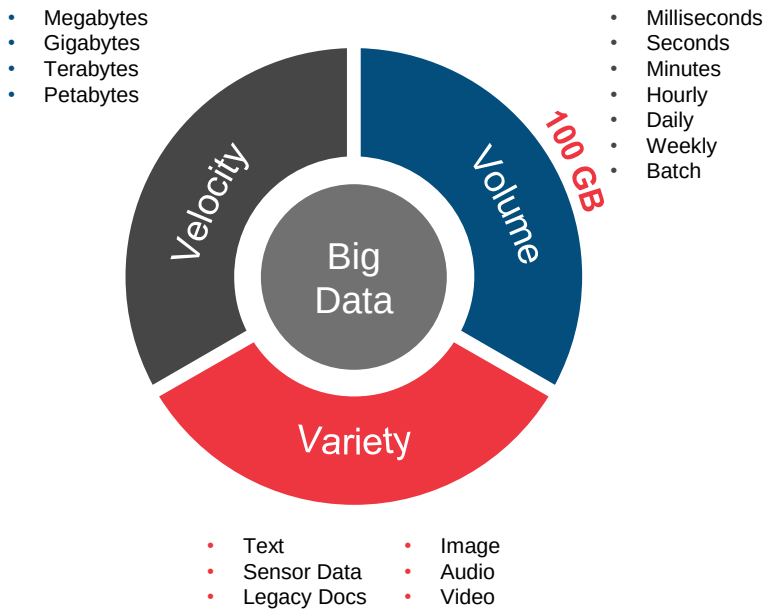
If we quickly analyse the definition, we will see 3 characteristic

- Volume
- Velocity
- Variety

How to do it?

- 100GB of structured data is easily managed by traditional data storage, so it would not be considered a **Big Data Volume**. When we start talking about 100TB or PB, it becomes Big Data
- Processing 100GB of data per minute (**Velocity**), or analysing 100GB of unstructured (social media, image, etc.) data (**Variety**) at the same speed could be Big Data
- **Cost-effective solution** for the problem created by a **combination of 3Vs**
- Requires **innovative thinking** and **use of innovative tools** and **techniques**.

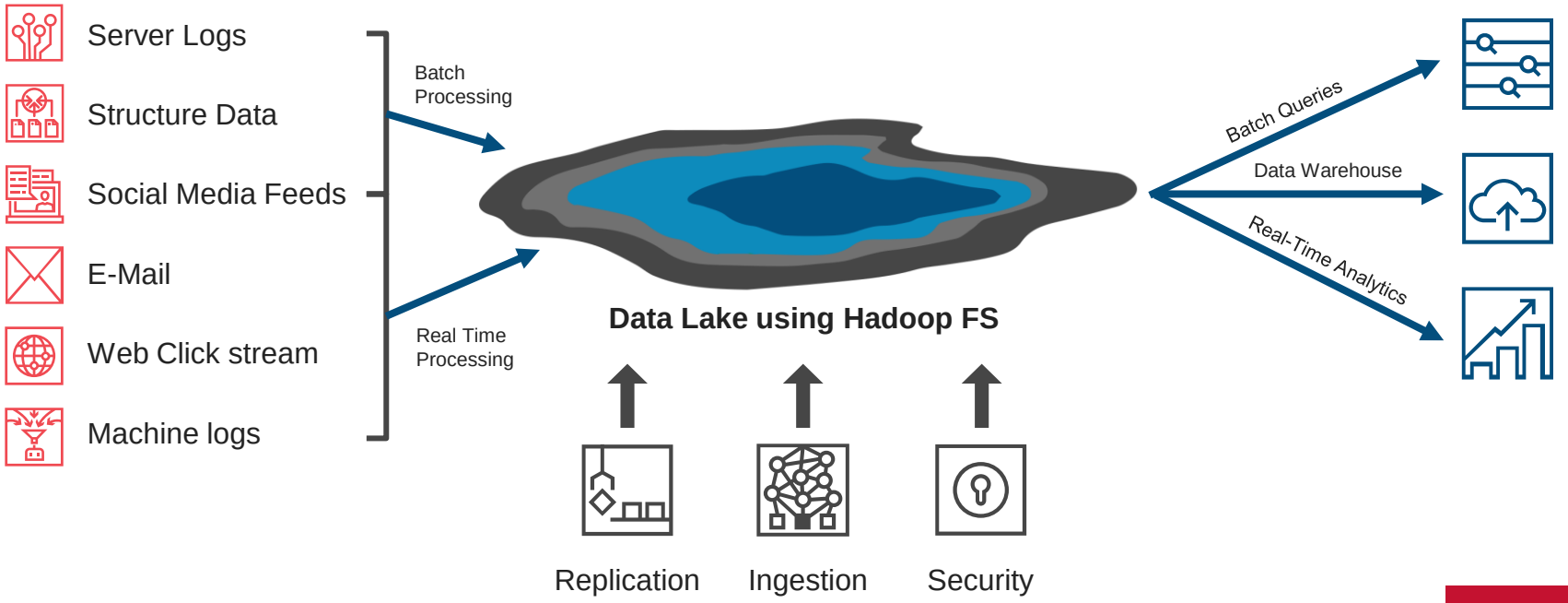
If your question fits into this definition, you have a **big data problem**



What is a Data Lake?

A data lake is **a centralized repository** that allows you to store all our **structured and unstructured data at any scale**. It stores data in a raw format, and runs different types of analytics; from dashboards and visualizations to **big data processing, real-time analytics,** and **machine learning** to guide better decisions.

Source: AWS



Difference between data warehouse and data lake

Maltem Consulting Group

Basis of Differences	Data Warehouse	Data Lake
Types of data	Stores data in the tables, row and columns	Stores raw data (Structured/Unstructured/ Semi-Structured) in its native format.
User	Business professionals	Data scientists
Processing	Schema-on-write, cleansed data, structured	Schema-on-Read, raw data in native format
Agility	Less agile, fixed configuration	Configuration and reconfiguration are done when required, highly agile
Reporting and analysis	Slow and expensive	Low storage, economical
Cost	Expensive storage	Low-cost storage
Security	Mature	Maturing

Maltem
Consulting Group

End of Day 2