Data visualization with D3.js

Introduction to D3.js

- D3.js is basically the **JavaScript library that turns raw data into visual, interactive graphics in the browser**.

- D3.js stands for **Data-Driven Documents**. It helps developer **bind data to HTML/SVG elements** and then control how those elements look, move, and update.

- **Coding Example**

Advantages of using D3.js

- **Super powerful**: You can build literally any type of chart or animation.

- **Customizable**: Unlike chart libraries, supports multiple flexi-styles.

- **Data-binding system**: Automatically creates, updates, or removes elements based on data.

- **Great animations**: Smooth transitions when data changes.

- **Uses web standards**: SVG, HTML, CSS.

Object Model of ODMG

- The **ODMG Object Model** is a **standard way of defining, storing, and manipulating objects in object-oriented databases**.(eg MongoDB)

- ODMG was developed by the Object Data Management Group to make object databases interoperable and uniform across vendors.
- Core Components of the ODMG Object Model include: Objects, Object Identifier, Literals, Types, Interfaces and Classes, Inheritance, Relationships, Extents, Object Lifecycle, Persistence.

Object Model of ODMG

- ODMG relates to Object Definition Language (ODL) and Object Query Language(OQL).
- Object Definition Language (ODL) is used for defining objects, classes, relationships.
- Object Query Language(OQL) is used for SQL-like query language for objects.
- ODMG model is important for standardizing object-oriented databases, help bridges programming languages with database concepts, the model has influenced multiple NoSQL databases(eg. MongoDB, Cassandra, DynamoDB etc.)

Object Model of ODMG

- **Core Components**

➢Objects

- Every entity stored is an object.

- Each object has:

State (attributes/values)
Behavior (methods)
Identity (OID – Object Identifier, unique across DB)

Object Model of ODMG

➢Object Identifier (OID)

- Unique, system-generated.
- Stays constant even if data changes.
- Allows sharing and linking of objects efficiently.

➢Literals

- Basic immutable values.
- Example: integers, strings, boolean, date, float.
- Literals have no identity (unlike objects).

Object Model of ODMG

➢ODMG Types

- Atomic types (int, float, bool)
- Collection types:

Set<T>

Bag<T>

List<T>

Array<T>

Dictionary<Key,Value>

- Object types (user-defined)

Object Model of ODMG

➢Interfaces & Classes

- An interface defines the structure (attributes + methods).
- A class implements/interfaces.
- Classes have:

Attributes

Relationships/associations

Methods (behavior)

Object Model of ODMG

➢ **Inheritance**

- Supports multiple inheritance.
- Subclasses inherit attributes and methods from super-classes.
- Polymorphism also supported.

➢ **Relationships**

- Defines how objects are connected.
- Can be:
  1-to-1
  1-to-many
  many-to-many

- Often represented with inverse relationships so both ends stay consistent.

Object Model of ODMG

➢ **Extents**

- Think of extents as “tables” in the object world.
- An extent stores all objects of a particular class.
- Useful for querying and navigation.

➢ **Object Lifecycle**

- Object creation
- Activation (loaded from DB)
- Deactivation (removed from cache)
- Deletion

➢ **Persistence**

- ODMG defines how objects are:
  - Made persistent
  - Stored in the object database
  - Retrieved using OQL (Object Query Language)

Object Definition Language (ODL)

- A schema definition language used to describe the **types**, **classes**, **interfaces**, **attributes**, **relationships**, and **extents** in an object database.
- Based on Interface Definition Language(IDL), extended for databases.
- Generally used to define:

Interfaces / Classes
Attributes (including collection types like set, bag, list, array)
Relationships (possibly bidirectional)
Inheritance (extends another interface/class)
Extent (the persistent set of instances of a class)

Object Definition Language (ODL)

- Coding Example showcasing turning of ODL schema into data for D3.js

Object Query Language

- OQL is the **query language** of ODMG.
- Looks like SQL but supports object paths, collections, methods.
- Navigating inside objects

```
SELECT a.city
FROM Person p, p.addresses a
WHERE p.name = "Arjun";
```

Object Query Language

- Method calls supported

```
SELECT p.getSalary()
FROM Employee p
WHERE p.department.name = "HR";
```

- Developer can use D3.js visualization of OQL – Coding Example

Object Database Conceptual Design

- This is the systematic process used for database design.

Layer	Meaning	Example
ODL	Defines object schema	interface Student { … }
Objects	Instances stored in database	JSON-like objects
OQL	Query objects	SELECT s.name FROM Student s
D3.js	Visualize results	Graphs, charts, networks

Object Database Conceptual Design (ODL→OQL→D3)

➢**Schema layer (ODL)**

- Define the database schema in ODL. The object DB uses this to define classes, extents, relationships.

➢**Data + Queries (OQL)**

- Data lives in object DB. Run OQL queries to retrieve results.

➢**Backend adapter**

- A small server executes OQL (e.g. Node.js). Converts results to JSON suitable for visualization.

➢**Frontend**

- Browser fetches JSON results. Uses D3 to build charts / graphs from that data. (e.g. directed graph, bar chart, scatterplot, timeline graphs)

Distributed Database Concepts

- A **Distributed Database System (DDBMS)** is a database stored across multiple sites but functioning as a single logical database.

❑Key Concepts

➢ **Distribution Transparency** →The ability of a distributed database to **hide the complexities of data distribution** from the users.

- **Location transparency**

Users do **not need to know where (at which site/server)** the data is stored.

- **Fragmentation transparency**

The system hides the fact that a table may be **horizontally or vertically fragmented** and stored across multiple sites.

- **Replication transparency**

Users are unaware that **multiple copies** of the same data exist across different sites.

Distributed Database Concepts

- **Fragmentation** → Fragmentation is the process of **splitting a table into smaller pieces** and distributing them across different sites for performance, availability, and parallelism.

- **Horizontal (rows distributed)**

Table is divided **by rows** based on a condition.

- **Vertical (columns distributed)**

Table is divided **by columns**.

- **Hybrid**

A combination of **horizontal + vertical fragmentation**.

- **Replication**

Replication means **storing full or partial copies of data across multiple sites** to increase availability, reliability, and read performance.

Distributed Database Concepts

➢ **Concurrency Control** →Ensures that **simultaneous distributed transactions** execute correctly.

- **Two-phase commit**

A coordinator ensures all sites either **commit or abort together.**

- **Distributed locking**

Locks (read/write) are managed across multiple sites so no conflicting updates occur.

- **Timestamp ordering**

Each transaction gets a **global timestamp**.

Distributed Database Concepts

- **Distributed Query Processing**

- **Query decomposition**

Break SQL/OQL query into **simpler algebraic operations** and check correctness, minimizing unnecessary data.

- **Localization**

Transform the global query into **operations on local fragments**, deciding which sites execute what.

- **Global optimization**

Choose the **lowest-cost distributed execution plan**, minimizing communication cost, number of site visits, intermediate site visits

- **Failures**

- **Site failure**

A site (node/server) crashes; local transactions fail but others continue.

- **Communication failure**

Messages are lost, delayed, or corrupted between sites; transactions may wait or timeout.

- **Network partitioning**

The network splits into isolated segments; each partition continues working but cannot communicate with others.

Distributed Database Concepts

➢ **Advantages**

- Reliability
- Scalability
- Local autonomy

➢ **Challenges**

- Higher complexity
- Synchronization
- Deadlocks

Types of Distributed Database Systems

Type of DDS	Description (Crisp)	Examples / Notes
Homogeneous DDS	All sites use the **same DBMS**, same schemas, same data models; easy communication and consistency.	Distributed Oracle, Distributed PostgreSQL
Heterogeneous DDS	Sites use **different DBMSs**, schemas, or data models; requires translation & middleware.	Oracle + MySQL + SQL Server combined
Federated / Multidatabase System	Autonomous databases that cooperate but maintain local control.	Local DBs connected via global schema
Client–Server DDS	Clients send requests; servers handle DB processing.	Web apps with remote DB servers
Peer-to-Peer DDS	All nodes act as **peers**—each can store data and process queries.	Blockchain-style systems, P2P DBs
Distributed Relational DB	Traditional RDBMS distributed across sites with SQL support.	Distributed MySQL or PostgreSQL clusters
Distributed Object DB	Object-oriented model distributed across sites.	ObjectDB, db4o
Cloud-based DDS	Data distributed across cloud regions/zones.	AWS Aurora Global DB, Google Spanner
Distributed NoSQL	NoSQL datasets distributed across nodes for scale and availability.	Cassandra, MongoDB Sharded Cluster

Query Processing in Distributed Databases

- Query processing in a distributed database involves executing a user query efficiently across multiple geographically separated sites while minimizing communication cost, ensuring correctness, and optimizing performance.

➢**Query Decomposition**

- Transforms a high-level query (OQL) into an **algebraic representation**.
- Performs **validation**, **normalization**, and **semantic checking**.
- Breaks the query into **smaller operations**.
- Ensures equivalence to the original query while preparing it for distribution.

Query Processing in Distributed Databases

➢ **Data Localization**

- Maps the decomposed query to the **physical location of data**.
- Converts logical fragments → physical fragments.
- Produces a **localized query plan** that accesses the correct sites.

➢ **Global Optimization**

- Finds the **best distributed execution strategy**.
- Minimizes: **Communication cost** (dominant factor),Local computation, Data transfer size.

Query Processing in Distributed Databases

➢ **Distributed Execution**

- Executes the optimized plan across all sites.
- Coordinates local operations and merges results.
- Ensures consistency, correct ordering, and fault tolerance.

Overview of Concurrency Control and recovery techniques in Distributed Databases.

➢ **Two-Phase Commit (2PC)**

- A protocol ensuring all sites either **commit** or **abort** a distributed transaction.
- **Phase 1:** Prepare → each site votes “commit/abort”.
- **Phase 2:** Commit/Abort → decision is broadcast by coordinator.

➢ **Distributed Locking**

- Extends two-phase locking (2PL) across sites.
- Locks on data items/fragments are coordinated globally to preserve serializability.
- A lock manager (centralized, primary-copy, or distributed) manages lock requests.

➢ **Timestamp Ordering**

- Each transaction gets a global timestamp.
- Operations are ordered based on timestamps to avoid conflicts.
- Ensures serializability without locks—useful in loosely coupled distributed systems.

Overview of Concurrency Control and recovery techniques in Distributed Databases.

➢ **Recovery Techniques**

- **Distributed Checkpointing**

Sites take coordinated or uncoordinated snapshots of system state.

Helps rollback to a consistent global state after a failure.

- **Distributed Logging**

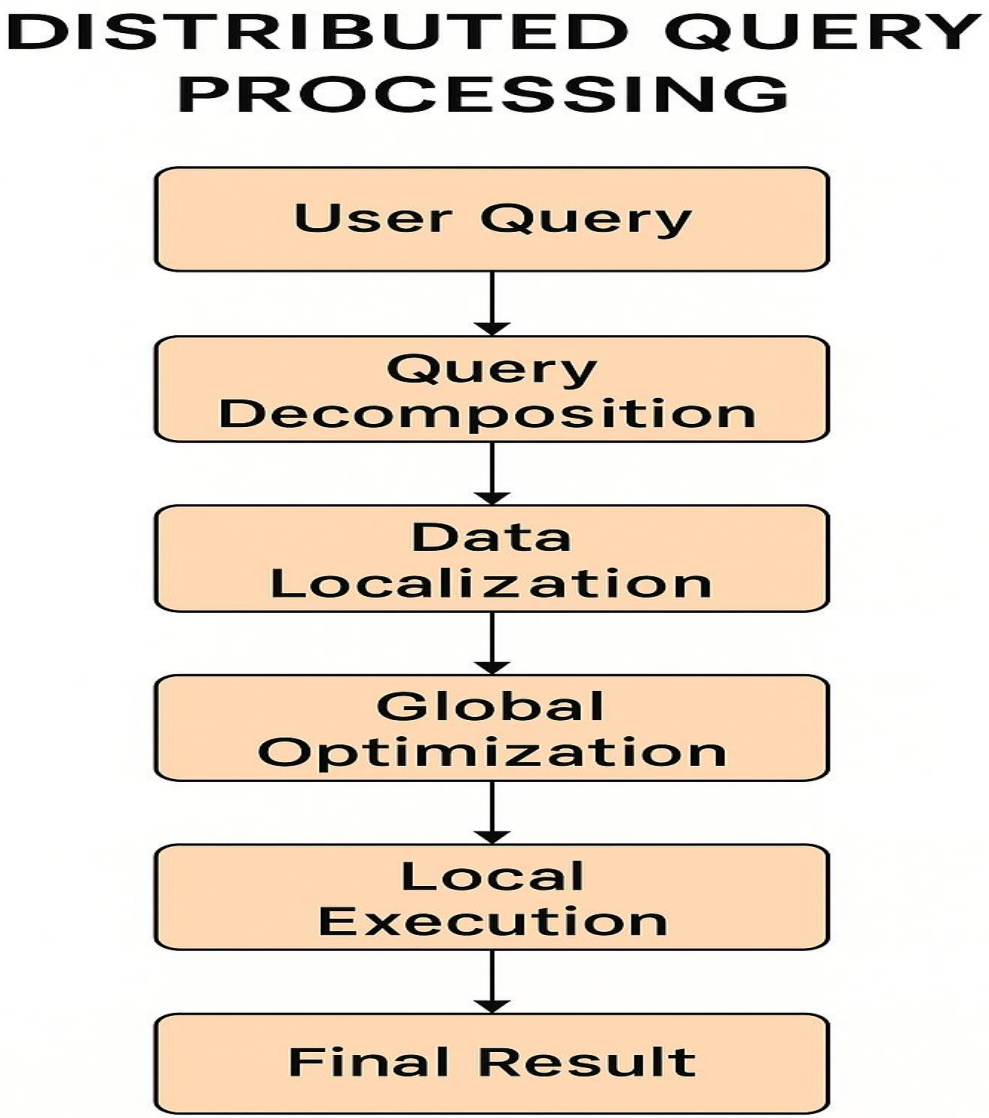
Local logs maintain before-image/after-image of data.

When transactions span sites, logs ensure atomicity during recovery.

- **Failure Recovery**

When a site recovers, logs + checkpoints are used to redo committed transactions and undo uncommitted ones.

Overview of Concurrency Control and recovery techniques in Distributed Databases.



THANK YOU