7BDIN006W
Big Data Theory and Practice

Lecture 2

Big Data Management. Sources and Structures of Big Data.
Relational Model

UNIVERSITY OF
WESTMINSTER

ACTIVITY: Word Cloud

Question: What coding background do you have?

Be honest, the poll is anonymous.

PollEv.com/nataliayerashenia404

Big Data Management

Big Data Management refers to the organisation, administration, and governance of large volumes of structured and unstructured data.

The aim is to ensure the data's quality, privacy, and security while making it accessible and useful.

Sources of Big Data

Social Media: Platforms like Facebook, Twitter, and Instagram generate massive amounts of user data every second.

Machine Logs: Servers, networks, and applications continuously log operational data, which is invaluable for troubleshooting and optimisation.

Internet of Things (IoT): Sensors, wearables, vehicles, and other connected devices continuously transmit data.

E-commerce: Online shopping platforms track users' behaviours, purchases, preferences, and more.

Finance & Trading: Stock exchanges, electronic trading platforms, and banks create vast quantities of transactional data.

Healthcare: Hospitals, clinics, and research centres produce large datasets from patient records, medical imaging, and genomic sequencing.

Multimedia: Videos, images, and audio files, especially from platforms like YouTube or Spotify.

Government & Administrative Records: Census data, tax records, and other public records.

Scientific Research: Data from various scientific disciplines' experiments, simulations, and field observations.

Mobile Data: Data from mobile apps, location-based services, and mobile searches.

Structures of Big Data

Data Type	Definition	Examples
Structured Data	Data that adheres to a specific format or schema. Often stored in relational databases.	Database tables with rows and columns, CSV files.
Unstructured Data	Data without a predefined structure or schema. Most of the big data is unstructured.	Text files, multimedia content, social media posts.
Semi-structured Data	Data that doesn't follow the formal structure of data models but has some organizational properties.	XML, JSON, NoSQL databases.
Time-series Data	Data collected at specific time intervals. It's ordered and can show change over time.	Stock market data, sensor data over time.
Graph Data	Data where entities and their relationships are both equally important. It's structured as nodes (entities) and edges (relationships).	Social networks, recommendation engines.

Big Data Storage Systems

Category	Description	Examples/Components
HDFS (Hadoop Distributed File System)	Part of the Hadoop framework, designed to store vast amounts of data across multiple machines. Uses block-based storage. Each block is replicated for fault tolerance.	Components: NameNode (metadata), DataNodes (data blocks).
NoSQL Databases		
- Key-Value Stores	Data is stored as key-value pairs.	Redis, DynamoDB
- Document Stores	Data is stored as documents, typically in JSON format.	MongoDB, CouchDB
- Column Stores	Data is stored in columns rather than rows.	Apache Cassandra, HBase
- Graph Databases	Designed for data whose relations are represented as a graph.	Neo4j, OrientDB
Object Storage	Ideal for vast amounts of unstructured data. Data is stored as objects in a flat namespace. Each object has a unique identifier for retrieval.	Amazon S3, Google Cloud Storage
Data Lakes	Repositories for raw data in its native format. Supports petabytes of data. Data can be loaded without a predefined schema.	Built on HDFS or cloud-based solutions
RDBMS	Traditionally for structured data; sometimes used with other storage solutions for big data.	MySQL, PostgreSQL, Oracle Database
In-memory Databases	Stores data in the system's RAM for faster access. Commonly used for real-time analytics.	Redis, Apache Spark, SAP HANA
Cloud Storage Solutions	Big data workloads are often on the cloud for scalability and cost-efficiency.	Amazon S3, Amazon Redshift, Google BigQuery, Azure Blob Storage

Relational Model

The relational model represents the database as *a collection of relations*.

A *relation* is a table of values.

Introduced by **E.F. Codd** in 1970.

Fundamental Concepts:

- **Relation:** Corresponds to a table. Each relation has a unique name.
- **Tuple:** A single row of a table representing a set of related data values.
- **Attribute:** A named column of a table representing a specific type of data.
- **Domain:** A set of atomic values that an attribute can take.
- **Degree:** The number of attributes in a relation.

Advantages of the Relational Model

- **Flexibility:** Users can construct a variety of ad-hoc queries.
- **Simplicity:** Data is perceived as tables with simple operations defined on them.
- **Data Integrity:** Ensured through integrity constraints.
- **Data Independence:** The structure can change without affecting applications or users.

RDBMS and SQL

Relational Database Management Systems (RDBMS):

Software systems that allow users to define, create, maintain, and control access to the database.

Examples: MySQL, Oracle, PostgreSQL, SQL Server

SQL (Structured Query Language):

- The standard language for querying and manipulating relational databases.
- Supports DML (Data Manipulation Language) operations like SELECT, INSERT, UPDATE, and DELETE.
- Supports DDL (Data Definition Language) operations like CREATE, ALTER, and DROP.

Fundamental Concepts of the Relational Model

Relation:

A **relation** corresponds to what many might traditionally think of as a table.

- Each relation/table has a distinct name to differentiate it from others.
- It is a set, meaning it contains no duplicate tuples (or rows).

Tuple:

- A **tuple** is equivalent to a row in a table.
- Represents a single item or record within the relation.
- For instance, in a "Student" relation, each tuple might represent one student and their associated information.

Attribute:

- An **attribute** corresponds to a column in a table.
- It has a specific name, distinguishing it from other attributes.
- It defines a certain type of data. For example, an attribute named "Age" might contain numerical values, while "FirstName" would contain strings.

Fundamental Concepts of the Relational Model

Domain:

- The **domain** of an attribute defines the set of permissible values the attribute can take.
- It ensures data integrity. For example, the domain of an "Age" attribute might be set as positive integers only.
- Each attribute in a relation has an associated domain. For instance, a "PhoneNumber" attribute might be restricted to 10-digit numbers.

Degree:

- The **degree** of a relation refers to the number of attributes it contains.
- For example, a relation with attributes "FirstName," "LastName," and "Age" would have a degree of 3.
- A relation's degree can also be referred to as its "arity."

Key Concepts

In the relational model of databases, a **key** is a set of *one or more attributes* that uniquely identifies a tuple (row) within a relation (table). Keys ensure that each record within a table can be uniquely distinguished from other records, even if the data has some similar values.

Primary Key: A chosen candidate key. Each relation/table has exactly one primary key.

Foreign Key: An attribute or set of attributes in one table that matches the primary key of another table, establishing relationships.

Example

Imagine a simple relation named **Students**:

StudentID	FirstName	LastName	Age
001	John	Doe	20
002	Jane	Smith	22

- **Relation:** Students.
- **Tuples:** There are two tuples, one for John Doe and one for Jane Smith.
- **Attributes:** StudentID, FirstName, LastName, Age.
- **Domains:**
  - StudentID: A unique identifier for each student.
  - FirstName: String values representing first names.
  - LastName: String values representing last names.
  - Age: Positive integers.
- **Degree:** The degree of the Students relation is 4 since it has four attributes.

Common Data Types and Corresponding Domains

- **INTEGER:** The domain includes all integer values, e.g., ..., -3, -2, -1, 0, 1, 2, 3, ...
- **FLOAT/DOUBLE:** The domain includes real numbers, allowing for decimal points.
- **CHAR(n) or VARCHAR(n):** Character strings of length n. VARCHAR is for variable-length strings up to n characters.
- **BOOLEAN:** The domain includes values TRUE or FALSE.
- **DATE:** Allows valid date values, typically in formats like YYYY-MM-DD.
- **TIMESTAMP:** Combines date and time, usually to the precision of fractions of a second.

Custom Domains:

In addition to predefined data types, many relational database systems allow users to define custom domains. This is particularly useful when specific constraints are needed.

Illustrative Example: Relational Model

Relations **Students** & **Majors**

StudentID	FirstName	LastName	MajorID
101	John	Doe	CS01
102	Jane	Smith	MT01
103	Alice	White	CS01

MajorID	MajorName
CS01	Comp. Sci.
MT01	Mathematics
PH01	Physics

Primary Key
Students: StudentID is the primary key.
Majors: MajorID is the primary key.

Foreign Key
In **Students**, MajorID is a foreign key referencing MajorID in **Majors**.

SQL Basic Operations: Select

SELECT (Projection): Retrieves certain columns from a table.

Retrieve only the first names of students: SELECT FirstName FROM Students

FirstName
John
Jane
Alice

SQL Basic Operations:

PROJECT (Selection): Filters rows based on specific criteria.

Find all students majoring in Comp. Sci.: SELECT * FROM Students WHERE MajorID = 'CS01';

StudentID	FirstName	LastName	MajorID
101	John	Doe	CS01
103	Alice	White	CS01

SQL Basic Operations

JOIN: Combines rows from two or more tables based on a related column.

Join **Students** and **Majors** to see students with their major names:

```
SELECT FirstName, LastName, MajorName FROM Students JOIN Majors ON Students.MajorID = Majors.MajorID;
```

FirstName	LastName	MajorName
John	Doe	Comp. Sci.
Jane	Smith	Mathematics
Alice	White	Comp. Sci.

SQL Basic Operations

SELECT (Projection): Retrieves certain columns from a table.

Retrieve only the first names of students: SELECT FirstName FROM Students

FirstName
John
Jane
Alice

SQL Basic Operations: Union & Cross Product

StudentID	StudentName
1	Alice
2	Bob
3	Charlie

MajorID	MajorName
101	Mathematics
102	Physics
103	Biology

SQL Basic Operations: Cross Product

The cross product of two tables is a combination of each row of the first table with each row of the second table.

```
SELECT StudentID, StudentName, MajorID,
MajorName
FROM Students, Majors;
```

or

```
SELECT StudentID, StudentName, MajorID,
MajorName
FROM Students
CROSS JOIN Majors;
```

StudentID	StudentName	MajorID	MajorName
1	Alice	101	Mathematics
1	Alice	102	Physics
1	Alice	103	Biology
2	Bob	101	Mathematics
2	Bob	102	Physics
2	Bob	103	Biology
3	Charlie	101	Mathematics
3	Charlie	102	Physics
3	Charlie	103	Biology

SQL Basic Operations: Union

The union operation combines the tuples of two tables into a single table. The two tables must have the same number of attributes, and the attributes must be of the same type and in the same order. The result eliminates duplicate rows.

```
SELECT StudentID as ID, StudentName as Name FROM Students
UNION
SELECT MajorID as ID, MajorName as Name FROM Majors;
```

ID	Name
1	Alice
2	Bob
3	Charlie
101	Mathematics
102	Physics
103	Biology