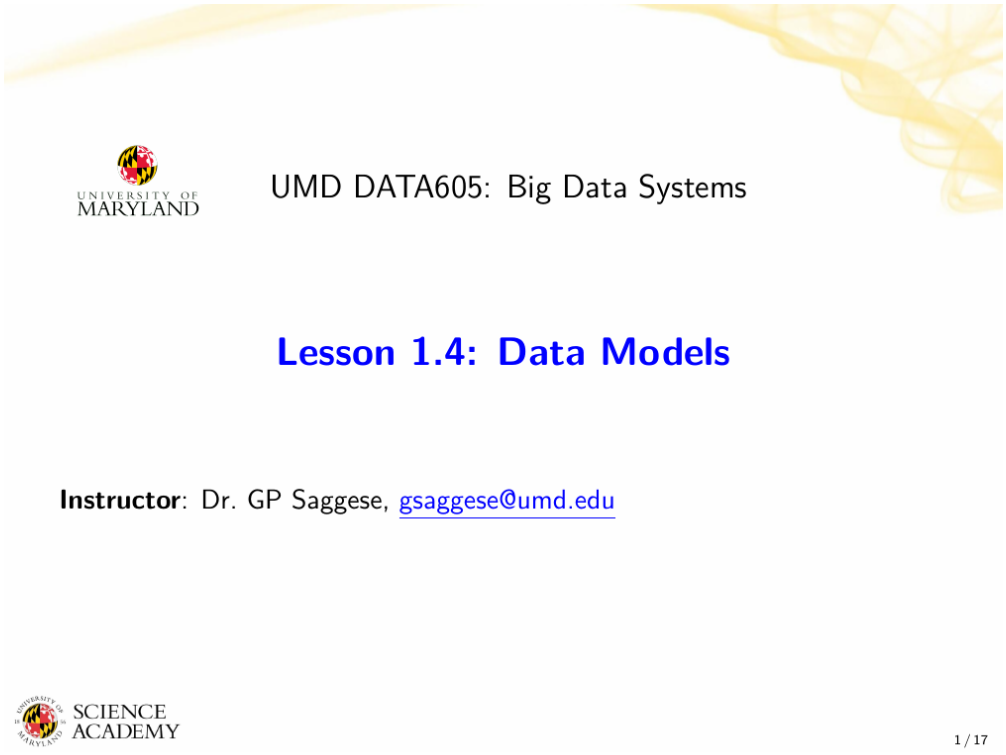
Lesson 1.4: Data Models

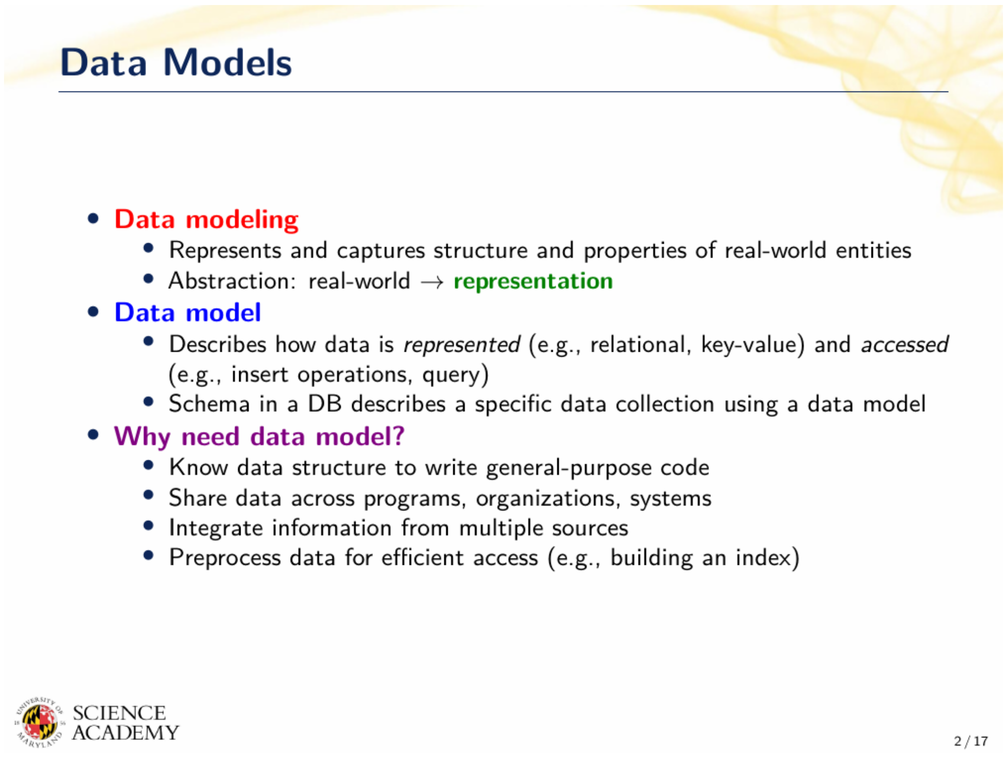


UMD DATA605: Big Data Systems

Lesson 1.4: Data Models

Instructor: Dr. GP Saggese, gsaggese@umd.edu

2 / 17: Data Models



Data Models

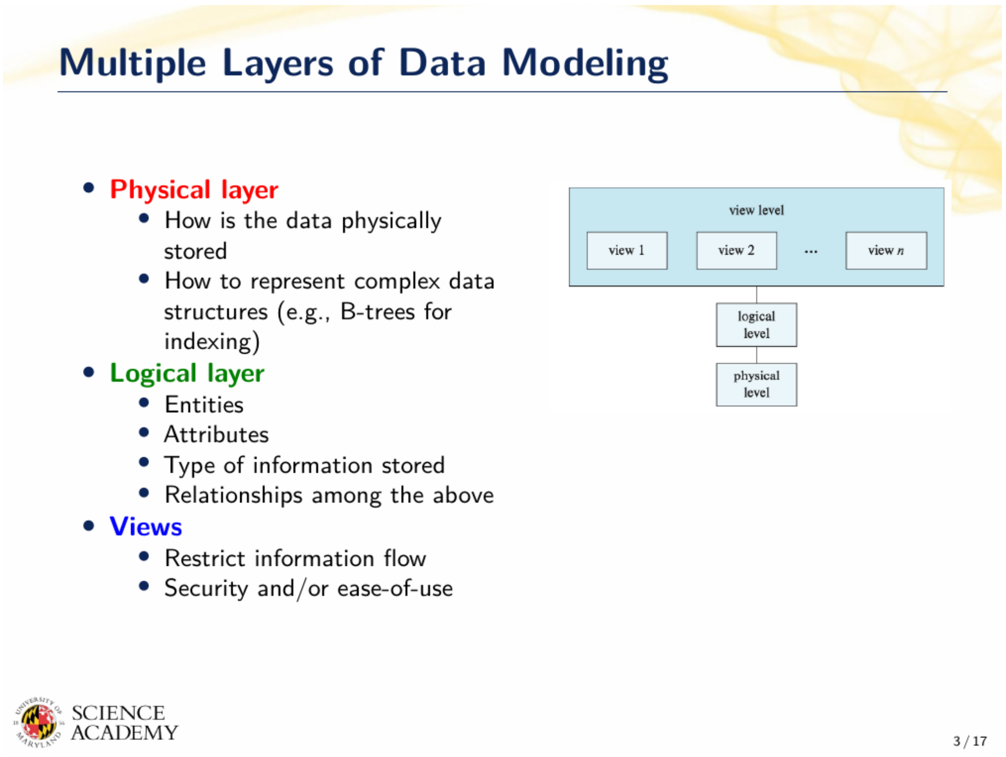
- **Data modeling**
  - Represents and captures structure and properties of real-world entities
  - Abstraction: real-world → **representation**
- **Data model**
  - Describes how data is *represented* (e.g., relational, key-value) and *accessed* (e.g., insert operations, query)
  - Schema in a DB describes a specific data collection using a data model
- **Why need data model?**
  - Know data structure to write general-purpose code
  - Share data across programs, organizations, systems
  - Integrate information from multiple sources
  - Preprocess data for efficient access (e.g., building an index)

SCIENCE ACADEMY

2 / 17

- **Data modeling**
  - *Data modeling* is the process of creating a visual representation of the structure and properties of real-world entities. Think of it as creating a blueprint that helps us understand and organize data. This process involves taking complex real-world scenarios and simplifying them into a format that computers can work with, which is known as *abstraction*. Essentially, we are translating the real world into a digital format that can be easily managed and manipulated.
- **Data model**
  - A *data model* is a framework that defines how data is structured and accessed. For example, in a relational data model, data is organized into tables, while in a key-value model, data is stored as pairs. The data model also dictates how data can be inserted, queried, and updated. A *schema* in a database is a specific instance of a data model that describes the organization of a particular data collection.
- **Why need data model?**
  - Having a data model is crucial because it provides a clear structure for data, which allows developers to write code that can handle various data scenarios. It also facilitates data sharing across different programs, organizations, and systems, ensuring consistency and compatibility. Moreover, data models enable the integration of information from multiple sources, making it easier to combine and analyze data. Additionally, they help in preprocessing data for efficient access, such as by building an index, which speeds up data retrieval and improves performance.

3 / 17: Multiple Layers of Data Modeling



Multiple Layers of Data Modeling

- Physical layer
  - How is the data physically stored
  - How to represent complex data structures (e.g., B-trees for indexing)
- Logical layer
  - Entities
  - Attributes
  - Type of information stored
  - Relationships among the above
- Views
  - Restrict information flow
  - Security and/or ease-of-use

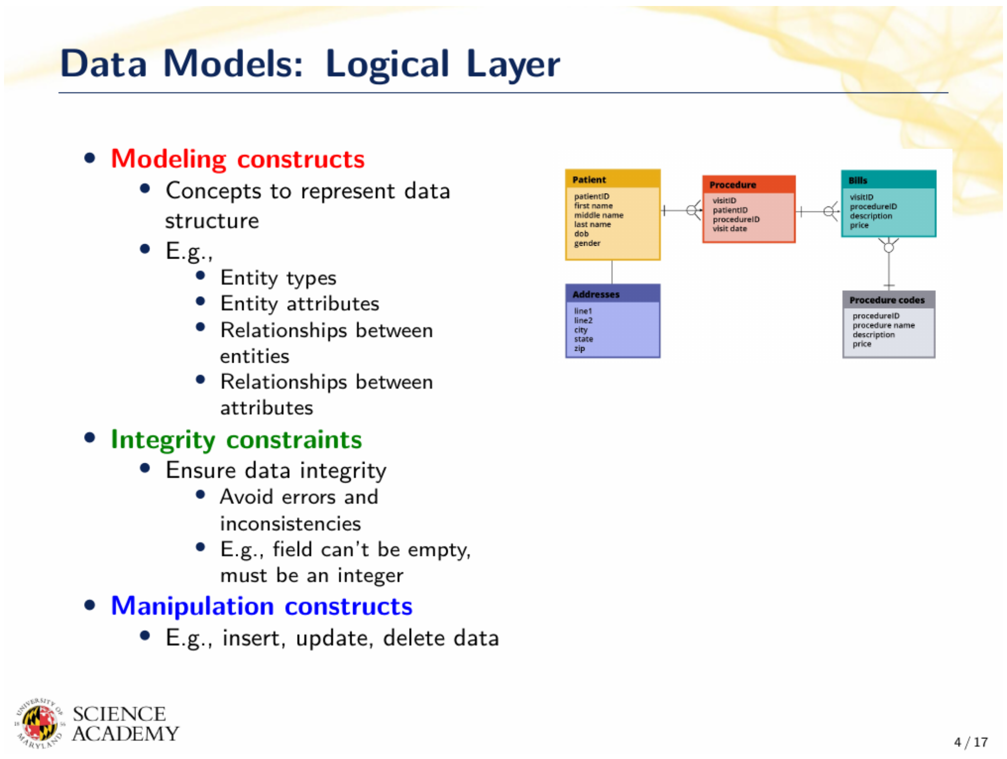
view level
view 1
view 2
...
view n
logical level
physical level

SCIENCE ACADEMY

3 / 17

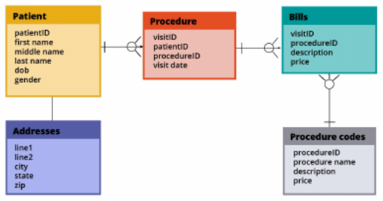
- **Physical layer**
  - This layer deals with the actual storage of data on hardware. It involves understanding how data is organized on physical media like hard drives or SSDs. For example, using *B-trees* is a method to efficiently index and retrieve data, which is crucial for performance in large databases.
  - The focus here is on optimizing storage and retrieval processes, ensuring that data can be accessed quickly and efficiently.
- **Logical layer**
  - This layer is about the abstract representation of data. It defines the *entities* (like tables in a database) and their *attributes* (the columns in those tables).
  - It also describes the *type of information* stored, such as integers, strings, or dates, and the *relationships* between different entities, like how a customer might be linked to their orders.
  - This layer is crucial for designing databases that accurately reflect the real-world scenarios they are meant to model.
- **Views**
  - Views are a way to present data to users in a specific format. They can be used to *restrict information flow*, ensuring that users only see the data they are authorized to access.
  - This is important for *security* and can also make the system easier to use by simplifying complex data structures into more understandable formats for end-users.

4 / 17: Data Models: Logical Layer



Data Models: Logical Layer

- Modeling constructs
  - Concepts to represent data structure
  - E.g.,
    - Entity types
    - Entity attributes
    - Relationships between entities
    - Relationships between attributes
- Integrity constraints
  - Ensure data integrity
    - Avoid errors and inconsistencies
    - E.g., field can't be empty, must be an integer
- Manipulation constructs
  - E.g., insert, update, delete data

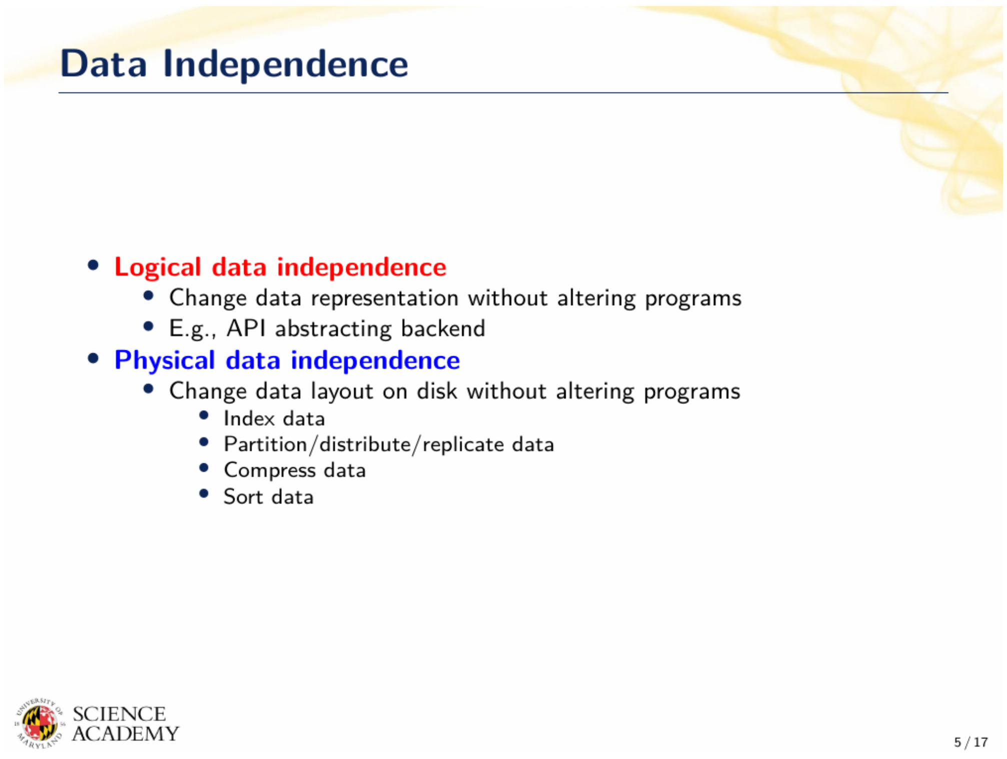


SCIENCE ACADEMY

4 / 17

- **Modeling constructs**
  - These are the tools and concepts used to define how data is structured within a database. They help in organizing data in a way that makes it easy to understand and manipulate.
  - *Entity types* refer to the different categories or classes of data, like "Customer" or "Product."
  - *Entity attributes* are the properties or details about an entity, such as a customer's name or a product's price.
  - *Relationships between entities* describe how different entities are connected, like a customer placing an order.
  - *Relationships between attributes* define how attributes within an entity relate to each other, ensuring consistency and logical connections.
- **Integrity constraints**
  - These are rules that ensure the accuracy and consistency of data within the database.
  - They help prevent errors, such as entering incorrect data types or leaving important fields empty.
  - For example, a constraint might require that a phone number field must contain only numbers and not be left blank.
- **Manipulation constructs**
  - These are the operations that allow users to interact with the data, such as adding new data (insert), changing existing data (update), or removing data (delete).
  - These constructs are essential for maintaining and updating the database as new information becomes available or as old information becomes obsolete.

5 / 17: Data Independence



Data Independence

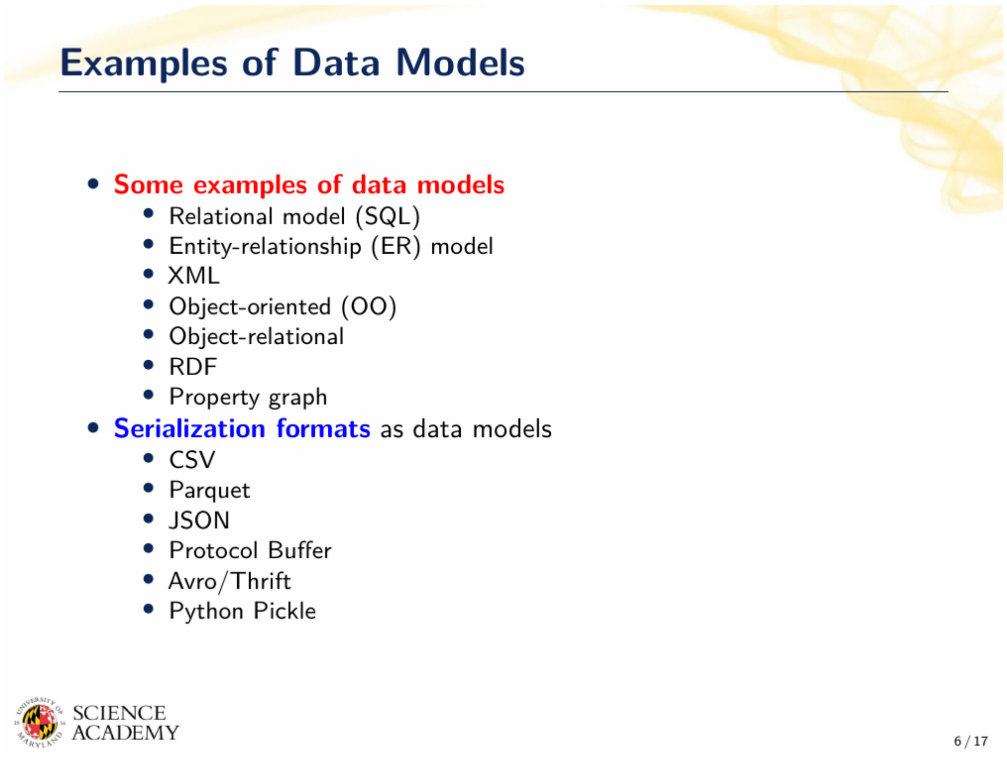
- **Logical data independence**
  - Change data representation without altering programs
  - E.g., API abstracting backend
- **Physical data independence**
  - Change data layout on disk without altering programs
    - Index data
    - Partition/distribute/replicate data
    - Compress data
    - Sort data

SCIENCE ACADEMY

5 / 17

- **Data Independence**
  - **Logical data independence**
    * This concept refers to the ability to change the way data is represented without needing to modify the programs that use this data. For example, if you have an application that accesses a database through an API, you can change how the data is structured or stored in the backend without needing to change the application itself. This is important because it allows developers to improve or optimize the data model without disrupting the applications that rely on it.
  - **Physical data independence**
    * This involves changing how data is physically stored on disk without affecting the programs that access it. Techniques like indexing, partitioning, distributing, replicating, compressing, and sorting data can be used to optimize performance or storage efficiency. For instance, you might decide to index certain columns to speed up queries or compress data to save space. These changes can be made without altering the application code, which means that applications remain stable and unaffected by these optimizations. This flexibility is crucial for maintaining and scaling large systems efficiently.

6 / 17: Examples of Data Models



Examples of Data Models

- Some examples of data models
  - Relational model (SQL)
  - Entity-relationship (ER) model
  - XML
  - Object-oriented (OO)
  - Object-relational
  - RDF
  - Property graph
- Serialization formats as data models
  - CSV
  - Parquet
  - JSON
  - Protocol Buffer
  - Avro/Thrift
  - Python Pickle

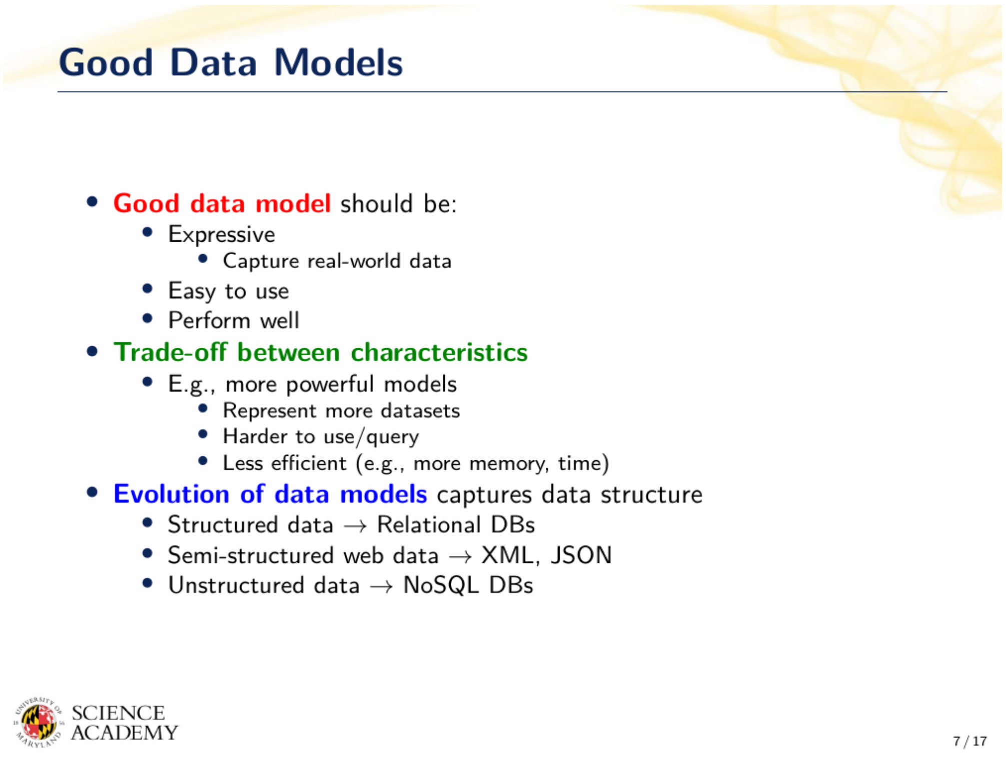
SCIENCE ACADEMY

6 / 17

- **Some examples of data models**
  - **Relational model (SQL)**: This is one of the most common data models used in databases. It organizes data into tables (or relations) with rows and columns. Each table represents a different entity, and SQL (Structured Query Language) is used to manage and query the data.
  - **Entity-relationship (ER) model**: This model is used to visually represent the data and its relationships. It uses entities (things we want to store information about) and relationships (how these entities are related) to design a database.
  - **XML**: Extensible Markup Language is a flexible text format used to store and transport data. It is both human-readable and machine-readable, making it useful for data interchange between systems.
  - **Object-oriented (OO)**: This model organizes data as objects, similar to how object-oriented programming languages like Java or C++ work. It allows for more complex data structures and relationships.
  - **Object-relational**: This combines features of both relational and object-oriented models, allowing for complex data types and relationships while still using a relational database.
  - **RDF**: Resource Description Framework is used to represent information about resources on the web. It is a standard model for data interchange on the web, often used in semantic web applications.
  - **Property graph**: This model represents data as nodes, edges, and properties. It is used in graph databases, which are useful for applications that involve complex relationships, like social networks.
- **Serialization formats as data models**

- **CSV**: Comma-Separated Values is a simple format for storing tabular data in plain text. Each line of the file is a data record, and each record consists of fields separated by commas.
- **Parquet**: This is a columnar storage file format optimized for use with big data processing frameworks like Apache Hadoop and Apache Spark. It is efficient for both storage and query performance.
- **JSON**: JavaScript Object Notation is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is widely used in web applications.
- **Protocol Buffer**: Developed by Google, this is a method for serializing structured data. It is more efficient than XML and JSON in terms of size and speed, making it suitable for communication protocols and data storage.
- **Avro/Thrift**: These are serialization frameworks that provide rich data structures and a compact binary format. They are often used in distributed systems for data exchange.
- **Python Pickle**: This is a module in Python used to serialize and deserialize Python objects. It is useful for saving the state of an object to a file or sending it over a network.

7 / 17: Good Data Models



Good Data Models

- Good data model should be:
  - Expressive
    - Capture real-world data
  - Easy to use
  - Perform well
- Trade-off between characteristics
  - E.g., more powerful models
    - Represent more datasets
    - Harder to use/query
    - Less efficient (e.g., more memory, time)
- Evolution of data models captures data structure
  - Structured data → Relational DBs
  - Semi-structured web data → XML, JSON
  - Unstructured data → NoSQL DBs

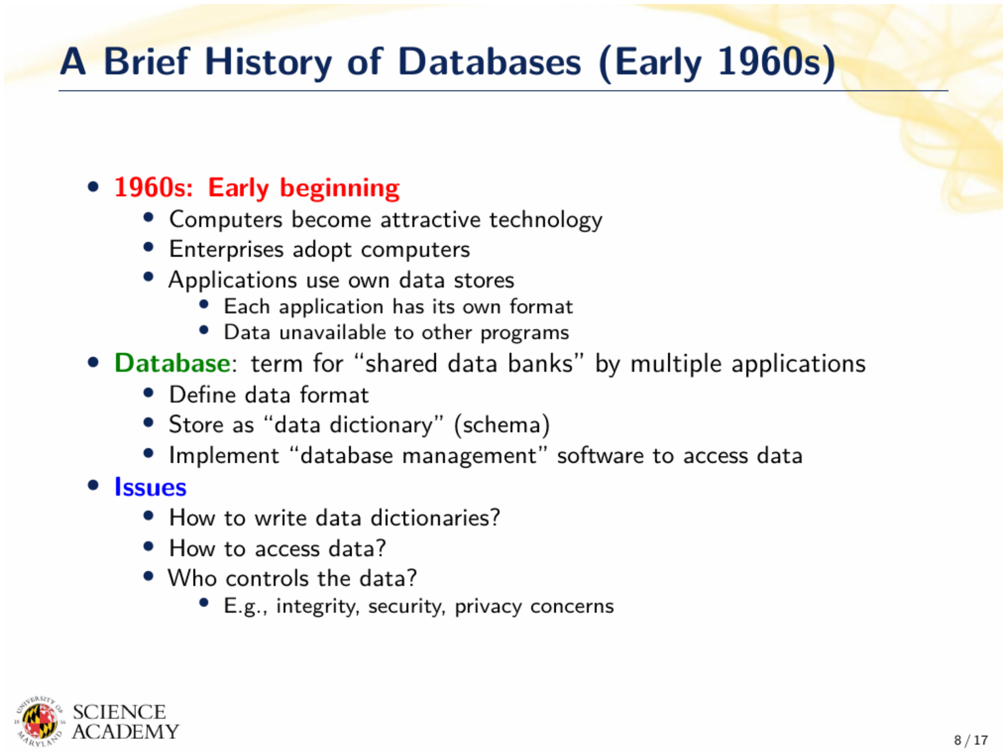
SCIENCE ACADEMY

7 / 17

- **Good data model** should be:
  - **Expressive**: A good data model needs to be able to accurately represent the complexities of real-world data. This means it should be capable of capturing all the necessary details and relationships within the data, allowing for a comprehensive understanding and analysis.
  - **Easy to use**: It's important that the data model is user-friendly. This means that both technical and non-technical users should be able to interact with it without excessive difficulty. An easy-to-use model facilitates better data management and accessibility.
  - **Perform well**: Performance is key in data models. They should be efficient in terms of speed and resource usage, ensuring that data can be processed and retrieved quickly without consuming excessive computational resources.
- **Trade-off between characteristics**:
  - There is often a balance to be struck between the expressiveness, usability, and performance of a data model. For example, more powerful models can represent a wider variety of datasets, which is beneficial for capturing complex data. However, these models can be more challenging to use and query, and they may require more memory and processing time, which can impact performance.
- **Evolution of data models** captures data structure:
  - Over time, data models have evolved to better handle different types of data structures. Initially, structured data was managed using Relational Databases (Relational DBs), which are well-suited for organized data with clear relationships.
  - As the web introduced more semi-structured data, formats like XML and JSON became popular for their flexibility in representing data that doesn't fit neatly into tables.
  - With the rise of unstructured data, such as text and multimedia, NoSQL databases

emerged to provide more scalable and flexible solutions for storing and querying this type of data.

8 / 17: A Brief History of Databases (Early 1960s)



A Brief History of Databases (Early 1960s)

- 1960s: Early beginning
  - Computers become attractive technology
  - Enterprises adopt computers
  - Applications use own data stores
    - Each application has its own format
    - Data unavailable to other programs
- Database: term for "shared data banks" by multiple applications
  - Define data format
  - Store as "data dictionary" (schema)
  - Implement "database management" software to access data
- Issues
  - How to write data dictionaries?
  - How to access data?
  - Who controls the data?
    - E.g., integrity, security, privacy concerns

SCIENCE ACADEMY

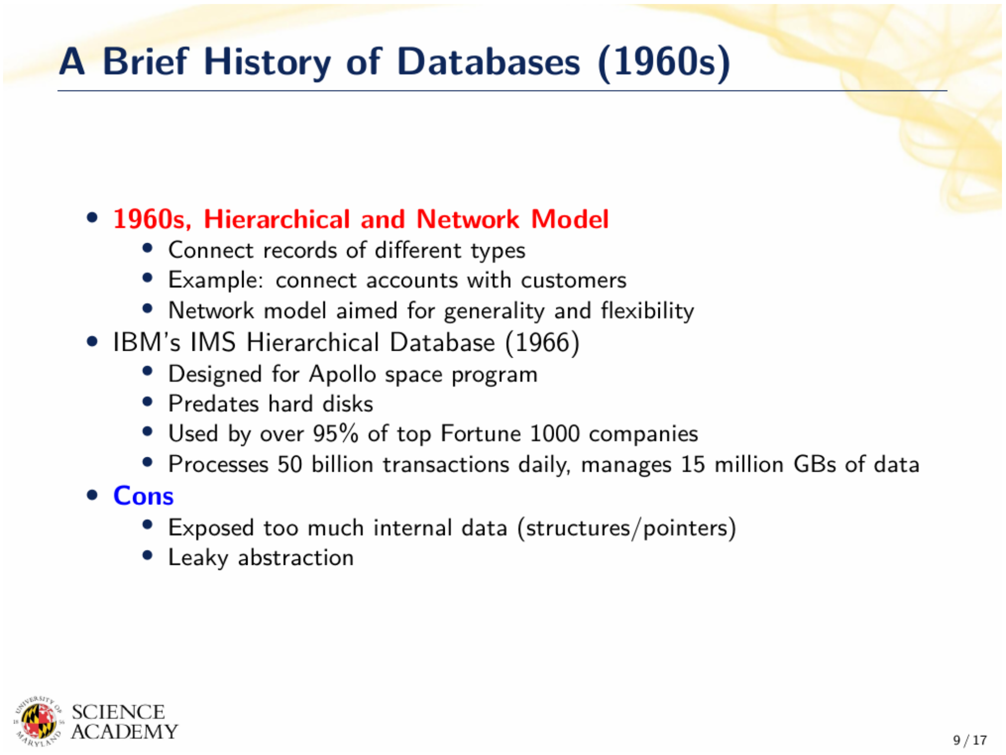
8 / 17

- A Brief History of Databases (Early 1960s)
- **1960s: Early beginning**
  - During the 1960s, computers started to become a popular and attractive technology for businesses. This was a time when companies began to see the potential of using computers to improve their operations.
  - As a result, many enterprises started adopting computers to help with various tasks. However, each application or program that a company used had its own way of storing data. This meant that the data was often in a unique format specific to that application.
  - Because of this, the data used by one application was not easily accessible or usable by other programs. This created a challenge for businesses that wanted to integrate and share data across different applications.
- **Database**: term for "shared data banks" by multiple applications
  - The concept of a "database" emerged as a solution to these challenges. A database was envisioned as a shared data bank that multiple applications could use.
  - To make this work, it was necessary to define a standard format for the data. This standard format was known as a "data dictionary" or schema, which described how the data was organized.
  - Additionally, software known as "database management" systems (DBMS) was developed to help access and manage the data stored in these databases.
- **Issues**
  - There were several issues that needed to be addressed with the introduction of databases.

One of the main questions was how to write and maintain these data dictionaries or schemas.

- Another challenge was figuring out the best way to access the data stored in the databases. This involved developing methods and tools to efficiently retrieve and manipulate the data.
- Control over the data was also a significant concern. Questions arose about who should have the authority to manage the data, ensuring its integrity, security, and privacy. These concerns were crucial as they impacted how data was protected and who could access it.

9 / 17: A Brief History of Databases (1960s)



A Brief History of Databases (1960s)

- 1960s, Hierarchical and Network Model
  - Connect records of different types
  - Example: connect accounts with customers
  - Network model aimed for generality and flexibility
- IBM's IMS Hierarchical Database (1966)
  - Designed for Apollo space program
  - Predates hard disks
  - Used by over 95% of top Fortune 1000 companies
  - Processes 50 billion transactions daily, manages 15 million GBs of data
- Cons
  - Exposed too much internal data (structures/pointers)
  - Leaky abstraction

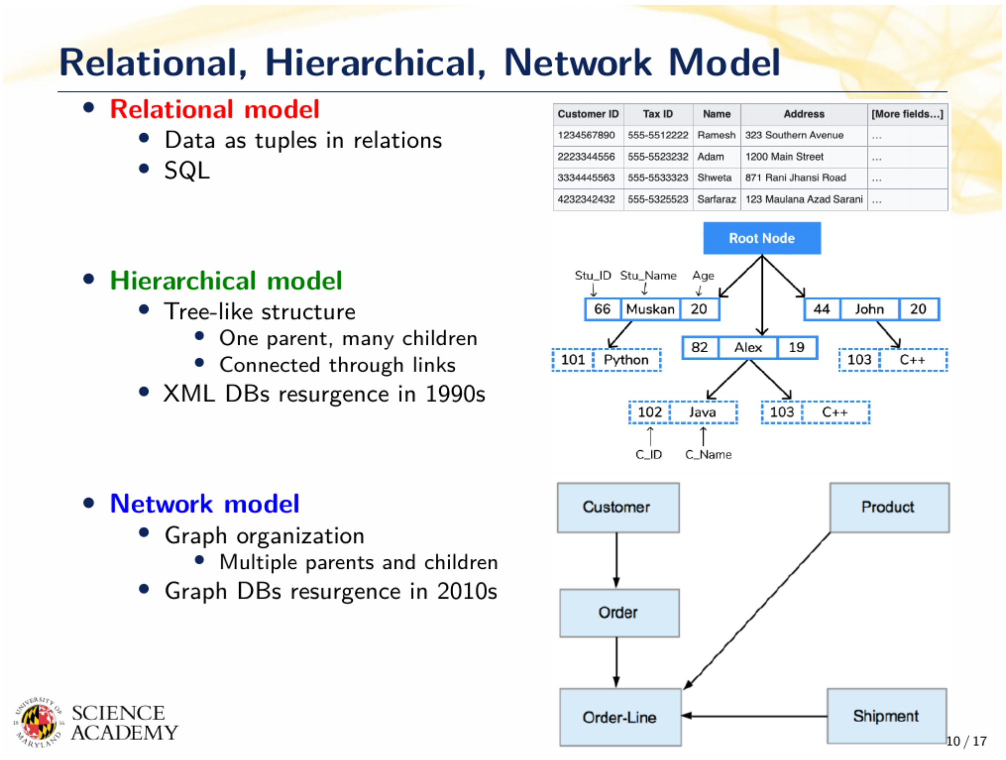
SCIENCE ACADEMY

9 / 17

- A Brief History of Databases (1960s)
- **1960s, Hierarchical and Network Model**
  - In the 1960s, databases were designed to connect different types of records, much like a family tree or a network of roads. This was known as the hierarchical and network model.
  - For example, in a bank, you could connect customer records with their account records, allowing for a structured way to manage relationships between data.
  - The network model was developed to be more general and flexible, allowing for more complex relationships between data, which was a significant advancement at the time.
- **IBM's IMS Hierarchical Database (1966)**
  - IBM developed the IMS (Information Management System) database in 1966 specifically for the Apollo space program, which required a robust system to manage vast amounts of data.
  - This database was created before the invention of hard disks, showcasing its pioneering nature in data management.
  - IMS became incredibly popular, with over 95% of the top Fortune 1000 companies using it, processing an enormous volume of transactions and managing vast amounts of data daily.
- **Cons**
  - One major downside of these early database models was that they exposed too much of the internal workings, such as data structures and pointers, to the users.

- This exposure led to what is known as a "leaky abstraction," where the complexity of the system was not hidden, making it difficult for users to interact with the database without understanding its intricate details.

10 / 17: Relational, Hierarchical, Network Model



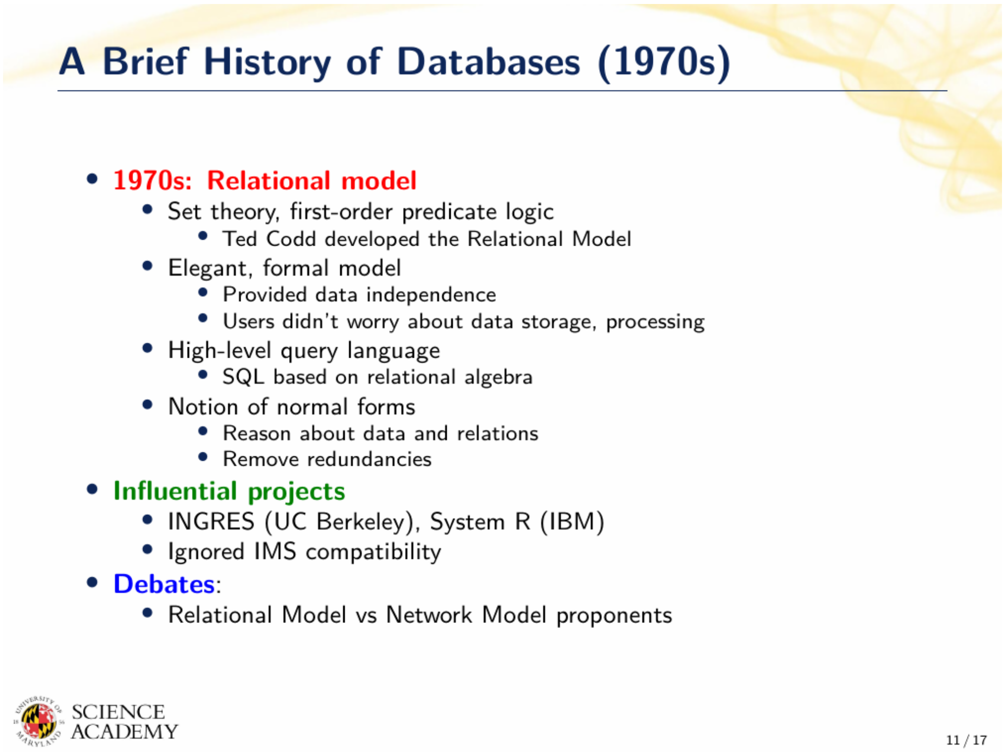
Relational, Hierarchical, Network Model

- Relational model
  - Data as tuples in relations
  - SQL
- Hierarchical model
  - Tree-like structure
    - One parent, many children
    - Connected through links
  - XML DBs resurgence in 1990s
- Network model
  - Graph organization
    - Multiple parents and children
  - Graph DBs resurgence in 2010s

Customer ID	Tax ID	Name	Address	[More fields...]
1234567890	555-5512222	Ramesh	323 Southern Avenue	...
2223344556	555-5523232	Adam	1200 Main Street	...
3334445563	555-5533323	Shweta	871 Rani Jhansi Road	...
4232342432	555-5325523	Sarfaraz	123 Maulana Azad Sarani	...

- **Relational model**
  - This model organizes data into tables, which are technically called *relations*. Each table consists of rows, known as *tuples*, and columns, which represent different data attributes. This model is widely used because it allows for easy data manipulation and querying using a language called SQL (Structured Query Language). SQL is a powerful tool for managing and retrieving data from relational databases, making this model very popular in various applications.
- **Hierarchical model**
  - In this model, data is organized in a tree-like structure. Each record has a single parent and can have multiple children, similar to a family tree. This structure is connected through links, making it efficient for certain types of data retrieval. The hierarchical model saw a resurgence in the 1990s with the rise of XML databases, which use a similar tree structure to store and manage data.
- **Network model**
  - The network model uses a graph-based organization, allowing for more complex relationships between data. Unlike the hierarchical model, a record in the network model can have multiple parents and children, providing greater flexibility. This model became popular again in the 2010s with the emergence of graph databases, which are particularly useful for applications involving complex relationships, such as social networks or recommendation systems.

11 / 17: A Brief History of Databases (1970s)



A Brief History of Databases (1970s)

- 1970s: Relational model
  - Set theory, first-order predicate logic
    - Ted Codd developed the Relational Model
  - Elegant, formal model
    - Provided data independence
    - Users didn't worry about data storage, processing
  - High-level query language
    - SQL based on relational algebra
  - Notion of normal forms
    - Reason about data and relations
    - Remove redundancies
- Influential projects
  - INGRES (UC Berkeley), System R (IBM)
  - Ignored IMS compatibility
- Debates:
  - Relational Model vs Network Model proponents

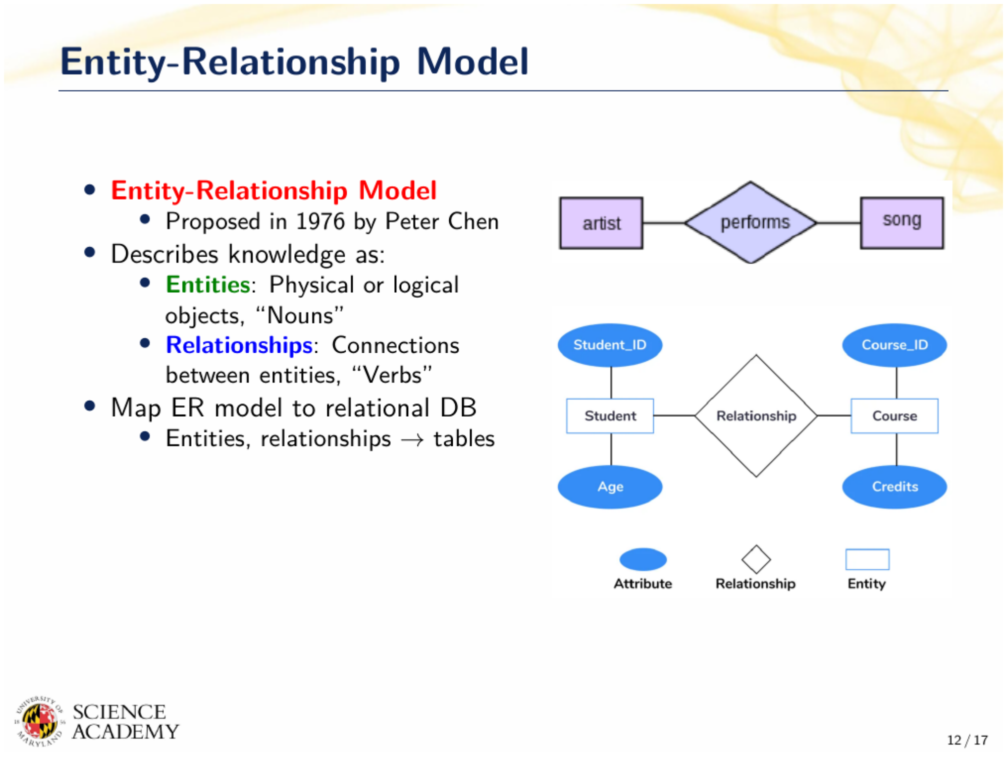
SCIENCE ACADEMY

11 / 17

- **1970s: Relational model**
  - **Set theory, first-order predicate logic**
    - In the 1970s, Ted Codd introduced the Relational Model, which was a groundbreaking way to organize and manage data. It was based on *set theory* and *first-order predicate logic*, which are mathematical concepts that help in structuring data logically.
  - **Elegant, formal model**
    - This model was considered elegant because it provided a clear and formal way to handle data. One of its key benefits was *data independence*, meaning users didn't need to worry about how data was stored or processed. They could focus on what data they needed, not how to get it.
  - **High-level query language**
    - SQL, or Structured Query Language, was developed based on relational algebra, allowing users to interact with databases using a high-level language that was easier to understand and use.
  - **Notion of normal forms**
    - Normal forms were introduced to help organize data efficiently. They provided a way to think about data and its relationships, aiming to eliminate redundancies and ensure data integrity.
- **Influential projects**
  - Projects like INGRES from UC Berkeley and System R from IBM were pivotal in demonstrating the practical applications of the Relational Model. These projects ignored compatibility with existing systems like IMS, focusing instead on the new relational approach.
- **Debates**:

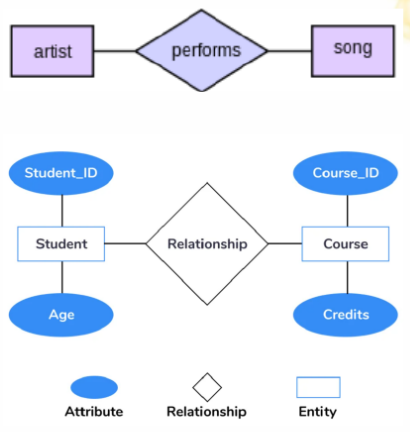
- During this time, there was a significant debate between proponents of the Relational Model and those who supported the Network Model. The Network Model was another way to organize data, but it was more complex and less flexible compared to the relational approach. This debate highlighted the shift towards more user-friendly and efficient database systems.

12 / 17: Entity-Relationship Model



Entity-Relationship Model

- **Entity-Relationship Model**
  - Proposed in 1976 by Peter Chen
- Describes knowledge as:
  - **Entities**: Physical or logical objects, "Nouns"
  - **Relationships**: Connections between entities, "Verbs"
- Map ER model to relational DB
  - Entities, relationships → tables

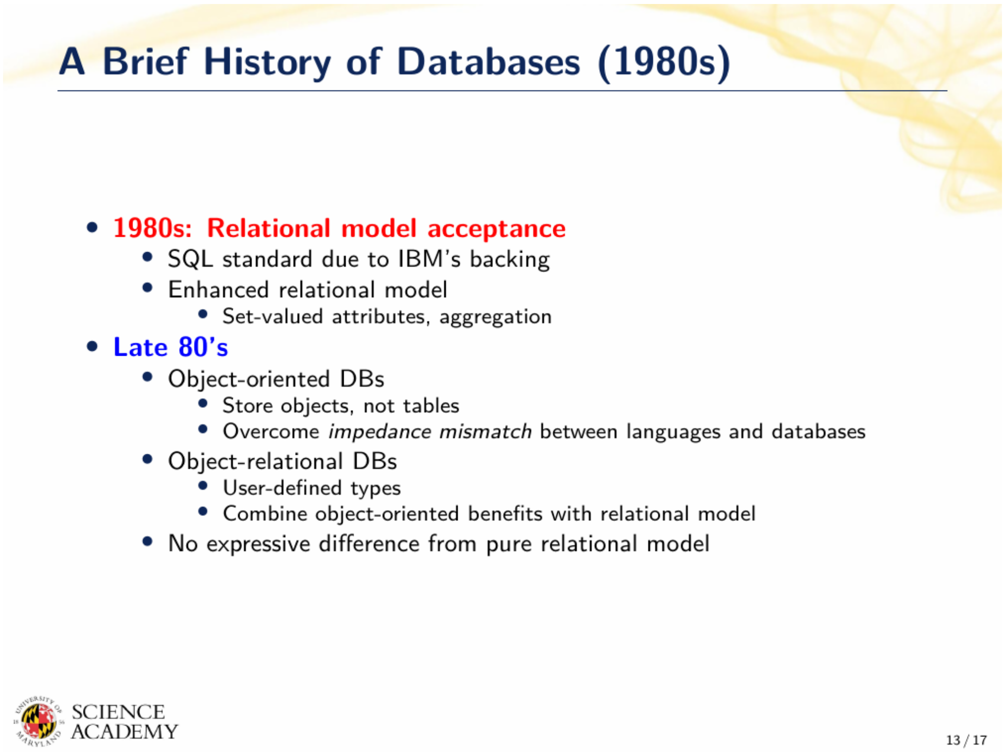


SCIENCE ACADEMY

12 / 17

- **Entity-Relationship Model**
  - The Entity-Relationship (ER) Model was introduced by Peter Chen in 1976. This model is a way to visually represent the data and its structure in a database. It helps in organizing and understanding the data by breaking it down into entities and relationships.
- **Describes knowledge as:**
  - **Entities**: These are the main components of the ER model. Think of entities as *nouns*; they represent physical or logical objects. For example, in a university database, entities could be "Student," "Course," or "Professor."
  - **Relationships**: These describe how entities are connected to each other, acting as *verbs*. For instance, a "Student" *enrolls in* a "Course," or a "Professor" *teaches* a "Course."
- **Map ER model to relational DB**
  - The ER model is a foundational step in designing a relational database. Entities and relationships are translated into tables. Each entity becomes a table, and relationships often become additional tables or foreign keys that link tables together. This mapping is crucial for creating a structured and efficient database that reflects the real-world scenario it is meant to model.

13 / 17: A Brief History of Databases (1980s)



A Brief History of Databases (1980s)

- 1980s: Relational model acceptance
  - SQL standard due to IBM's backing
  - Enhanced relational model
    - Set-valued attributes, aggregation
- Late 80's
  - Object-oriented DBs
    - Store objects, not tables
    - Overcome *impedance mismatch* between languages and databases
  - Object-relational DBs
    - User-defined types
    - Combine object-oriented benefits with relational model
  - No expressive difference from pure relational model

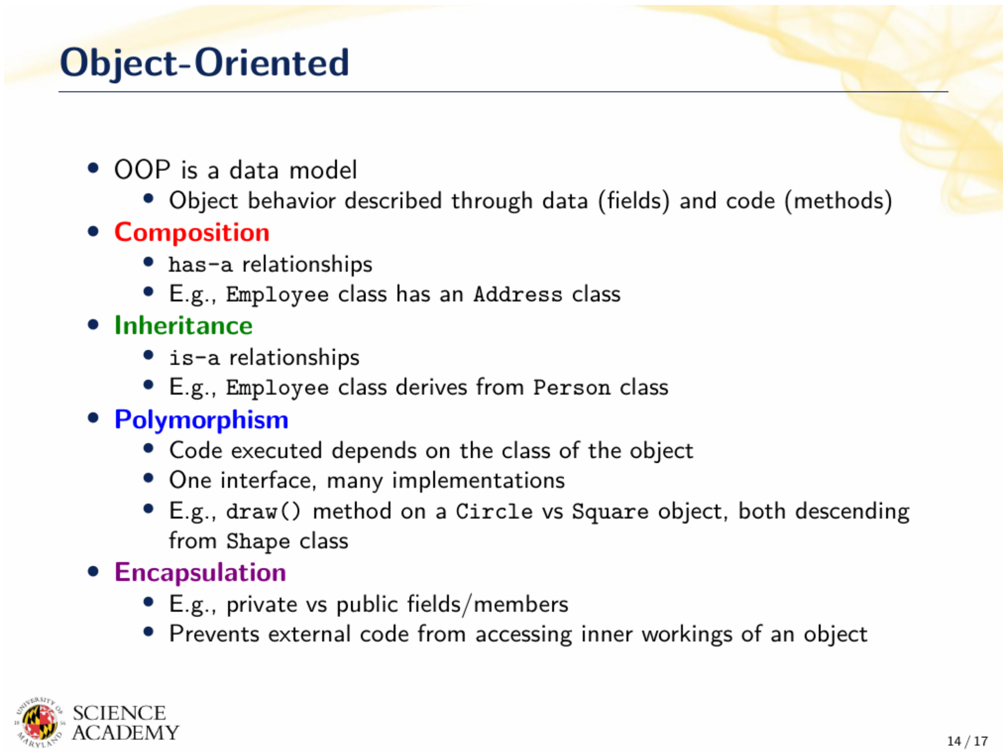
SCIENCE ACADEMY

13 / 17

- A Brief History of Databases (1980s)
- **1980s: Relational model acceptance**
  - During the 1980s, the relational database model gained widespread acceptance. This was largely due to the support from IBM, a major player in the tech industry. IBM's backing helped establish SQL (Structured Query Language) as the standard language for managing and querying relational databases. This standardization was crucial because it allowed different database systems to communicate using a common language.
  - The relational model was enhanced during this time to include more complex features. For example, set-valued attributes allowed for more flexible data structures, and aggregation functions enabled more sophisticated data analysis directly within the database.
- **Late 80's**
  - In the late 1980s, object-oriented databases emerged. These databases were designed to store data as objects rather than in traditional tables. This approach aimed to address the *impedance mismatch* problem, which is the disconnect between how data is represented in programming languages (as objects) and how it is stored in relational databases (as tables).
  - Object-relational databases were developed to combine the benefits of object-oriented databases with the relational model. They introduced features like user-defined types, allowing for more customized data structures while maintaining the relational framework.
  - Despite these advancements, there was no significant expressive difference between object-relational databases and the pure relational model. This means that while the new models offered additional features, they did not fundamentally change the way

data could be expressed or queried compared to traditional relational databases.

14 / 17: Object-Oriented


Object-Oriented

- OOP is a data model
  - Object behavior described through data (fields) and code (methods)
- Composition
  - `has-a` relationships
  - E.g., `Employee` class has an `Address` class
- Inheritance
  - `is-a` relationships
  - E.g., `Employee` class derives from `Person` class
- Polymorphism
  - Code executed depends on the class of the object
  - One interface, many implementations
  - E.g., `draw()` method on a `Circle` vs `Square` object, both descending from `Shape` class
- Encapsulation
  - E.g., private vs public fields/members
  - Prevents external code from accessing inner workings of an object

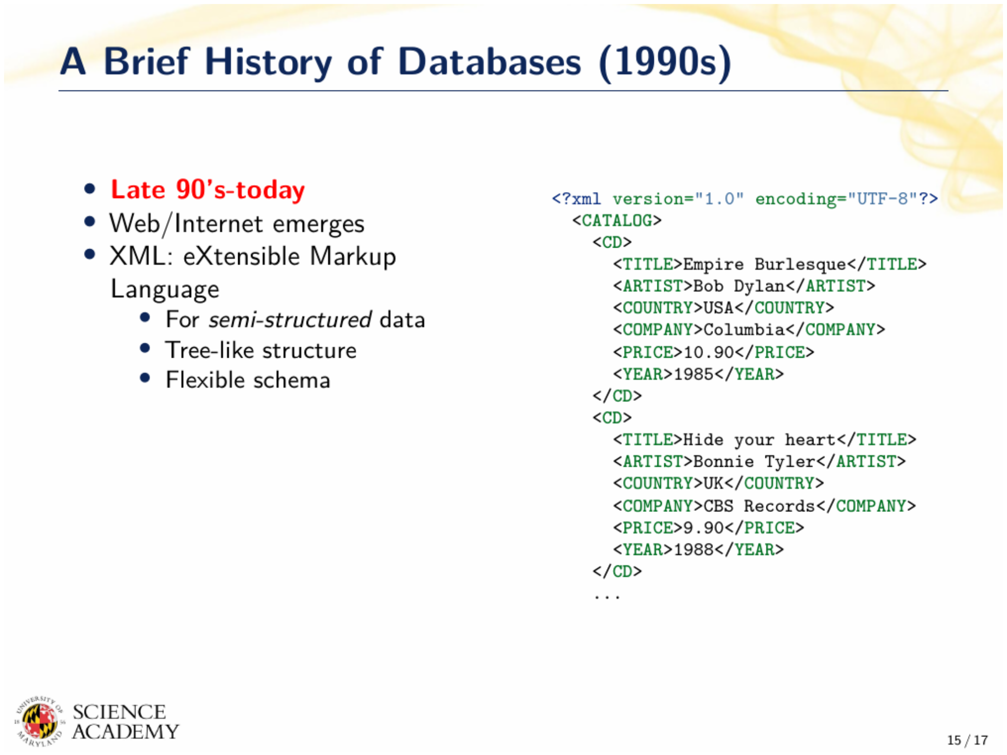
SCIENCE ACADEMY

14 / 17

- **Object-Oriented**
  - Object-Oriented Programming (OOP) is a way to model data and behavior in programming. It uses *objects* to represent real-world entities. Each object contains data, known as fields, and code, known as methods, which define its behavior. This approach helps in organizing complex programs by breaking them down into smaller, manageable parts.
- **Composition**
  - Composition is about creating complex types by combining objects. It describes a `has-a` relationship, meaning one object contains or is composed of another. For example, an `Employee` class might have an `Address` class as part of its structure. This allows for building more complex data structures by reusing existing classes.
- **Inheritance**
  - Inheritance is a mechanism where a new class, known as a subclass, is derived from an existing class, known as a superclass. This creates an `is-a` relationship. For instance, an `Employee` class might inherit from a `Person` class, meaning an employee *is a* person. This allows the subclass to inherit fields and methods from the superclass, promoting code reuse.
- **Polymorphism**
  - Polymorphism allows objects to be treated as instances of their parent class, even though they might be instances of a derived class. This means the same operation can behave differently on different classes. For example, a `draw()` method might be implemented differently in a `Circle` class and a `Square` class, both of which are derived from a `Shape` class. This concept allows for flexibility and the ability to extend code easily.
- **Encapsulation**
  - Encapsulation is about restricting access to certain components of an object, typically

by using private and public fields or members. This means that the internal state of an object is hidden from the outside, and can only be accessed or modified through specific methods. This helps in protecting the integrity of the data and prevents external code from interfering with the internal workings of an object.

15 / 17: A Brief History of Databases (1990s)



A Brief History of Databases (1990s)

- Late 90's-today
- Web/Internet emerges
- XML: eXtensible Markup Language
  - For *semi-structured* data
  - Tree-like structure
  - Flexible schema

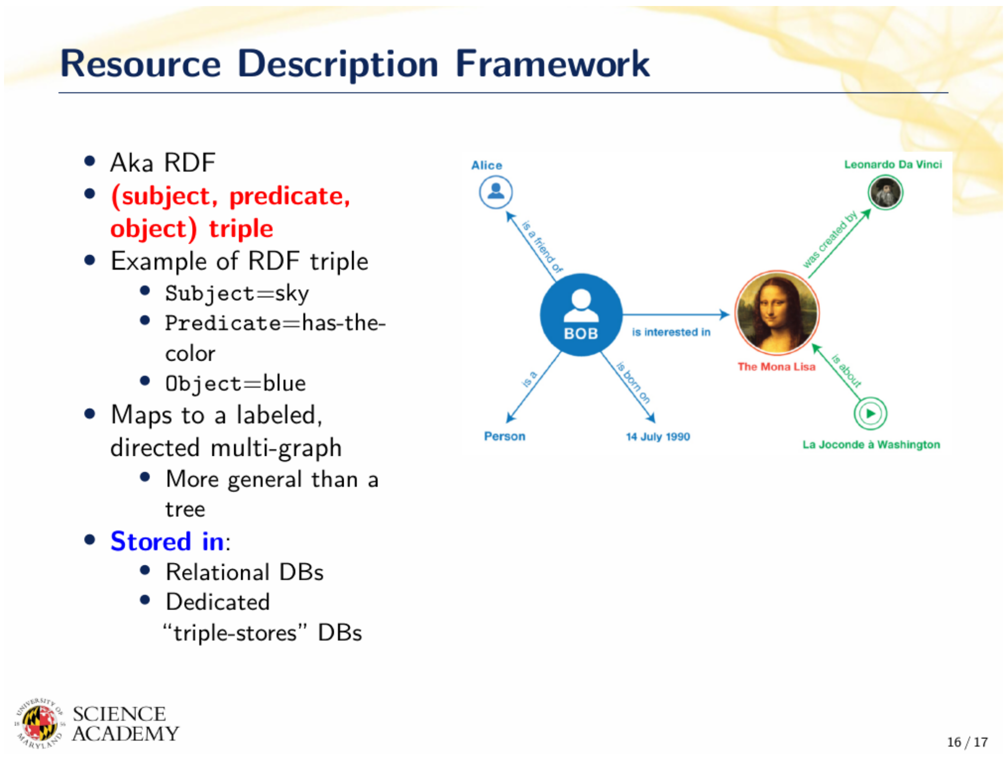
```
<?xml version="1.0" encoding="UTF-8"?>
  <CATALOG>
    <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
    </CD>
    <CD>
      <TITLE>Hide your heart</TITLE>
      <ARTIST>Bonnie Tyler</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS Records</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1988</YEAR>
    </CD>
    ...
```

SCIENCE ACADEMY

15 / 17

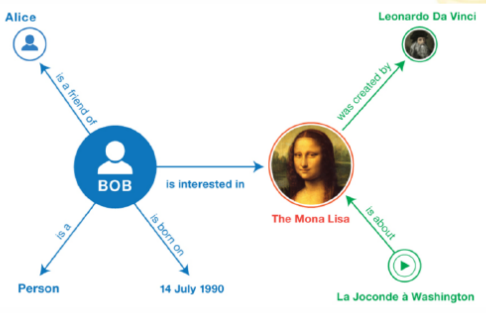
- **Late 90's-today**
  - During the late 1990s, the internet began to become widely accessible, leading to a significant shift in how data was stored and accessed. This era marked the beginning of the digital age, where information could be shared globally at unprecedented speeds. The rise of the web and internet technologies necessitated new ways to handle data that was not strictly structured.
- **Web/Internet emerges**
  - The emergence of the web and the internet revolutionized how businesses and individuals interacted with data. It allowed for the creation of dynamic websites and online services that could serve millions of users. This period saw a move from traditional databases to more flexible systems that could handle the diverse and growing amount of data generated online.
- **XML: eXtensible Markup Language**
  - XML was introduced as a way to manage *semi-structured* data, which is data that doesn't fit neatly into tables like traditional databases. It uses a tree-like structure, making it ideal for representing complex data hierarchies. XML's flexible schema means that it can adapt to different types of data without requiring a fixed structure, which was crucial for the evolving needs of web applications.
- **XML Example**
  - The example provided shows a simple XML document representing a catalog of CDs. Each CD entry includes details like the title, artist, country, company, price, and year. This format allows for easy data exchange between systems, as XML is both human-readable and machine-readable. The flexibility of XML made it a popular choice for data interchange on the web during this time.

16 / 17: Resource Description Framework



Resource Description Framework

- Aka RDF
- **(subject, predicate, object) triple**
- Example of RDF triple
  - `Subject`=sky
  - `Predicate`=has-the-color
  - `Object`=blue
- Maps to a labeled, directed multi-graph
  - More general than a tree
- **Stored in**:
  - Relational DBs
  - Dedicated "triple-stores" DBs

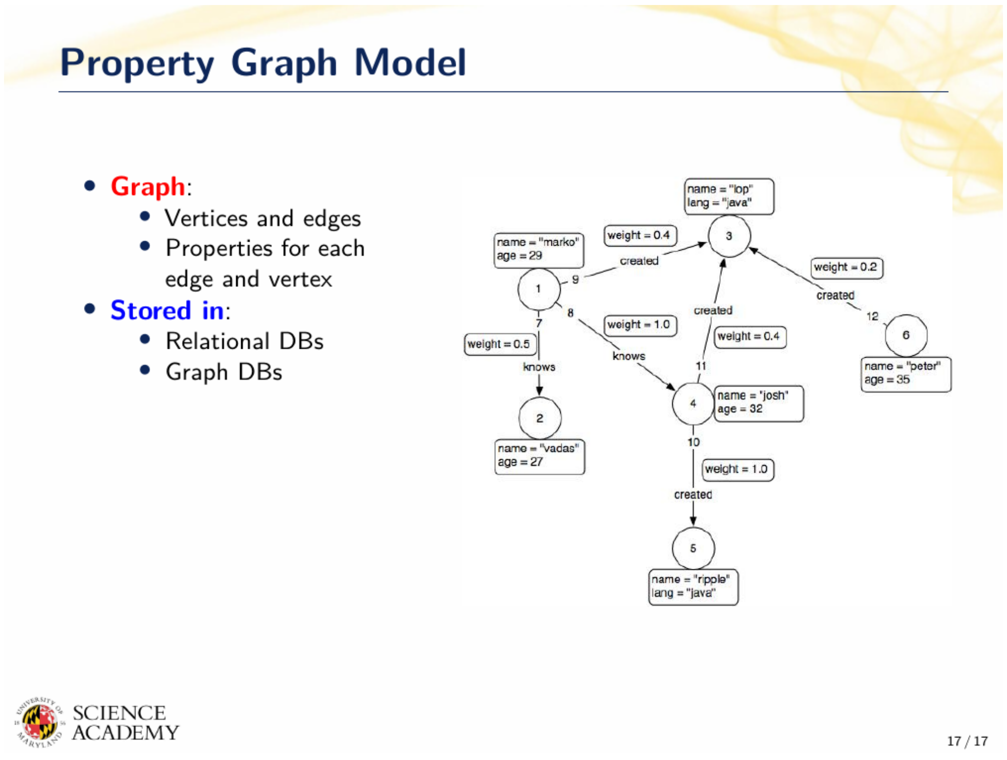


SCIENCE ACADEMY

16 / 17

- **Resource Description Framework (RDF)**
  - RDF is a framework used to represent information about resources on the web. It's a standard model for data interchange, which means it helps different systems understand and use data consistently.
- **(subject, predicate, object) triple**
  - RDF uses a simple structure called a "triple" to describe data. Each triple consists of three parts: a *subject*, a *predicate*, and an *object*. This structure is similar to a simple sentence where the subject is the thing being described, the predicate is the property or relationship, and the object is the value or another resource.
- **Example of RDF triple**
  - In the example given, the *subject* is "sky," the *predicate* is "has-the-color," and the *object* is "blue." This triple states that the sky has the color blue.
- **Maps to a labeled, directed multi-graph**
  - RDF triples can be visualized as a graph where nodes represent subjects and objects, and edges represent predicates. This graph is more flexible than a tree because it allows multiple connections between nodes, making it suitable for complex data relationships.
- **Stored in**:
  - RDF data can be stored in traditional relational databases or specialized databases known as "triple-stores," which are optimized for handling RDF triples. These storage methods help manage and query RDF data efficiently.

17 / 17: Property Graph Model



Property Graph Model

- **Graph**:
  - Vertices and edges
  - Properties for each edge and vertex
- **Stored in**:
  - Relational DBs
  - Graph DBs

SCIENCE ACADEMY

17 / 17

- **Graph**:
  - *Vertices and edges*: In the property graph model, a graph is made up of two main components: vertices (also known as nodes) and edges. Vertices represent entities or objects, while edges represent the relationships or connections between these entities. For example, in a social network, vertices could represent people, and edges could represent friendships.
  - *Properties for each edge and vertex*: Each vertex and edge can have properties, which are key-value pairs that store additional information. For instance, a vertex representing a person might have properties like “name” and “age,” while an edge representing a friendship might have a property like “since” to indicate when the friendship started. This allows for rich, detailed data representation.
- **Stored in**:
  - *Relational DBs*: Although relational databases are traditionally used for structured data in tables, they can also store graph data. However, this often requires complex queries and joins to represent relationships, which can be less efficient.
  - *Graph DBs*: Graph databases are specifically designed to store and query graph data. They naturally represent vertices and edges, making it easier and faster to perform operations like finding connections or traversing paths. Examples include Neo4j and Amazon Neptune.