CS639: Data Management for Data Science
Lecture 3: Principles of Data Management

Theodoros Rekatsinas
Announcements

• Mix-up with due dates 😞 It should be fixed now.
  • No changes to the midterm

• Updates and hints on PA1 assignment on Piazza

• Questions?
Today’s Lecture

1. Data Management
2. Data Models
3. RDBMs and the Relational Data Model
1. Data Management
Data Management

• Data represents the traces of real-world processes.

• Data is valuable but hard and costly to manage
  • Storage, representation complexity, collection

• Data management seeks to answer two questions:
  • What operations do we want to perform on this data?
  • What functionality do we need to manage this data?
Required Functionality

• Describe real-world entities in terms of stored data
• Create & persistently store large datasets
• Efficiently query & update
  • Must handle complex questions about the data
  • Must handle sophisticated updates
  • Performance matters
• Change structure (e.g., add attributes)
• Concurrency control: enable simultaneous queries, updates etc
• Crash recovery
• Access control, security, integrity

It is difficult and costly to implement all these features!
Systems providing data management features

• Relational database management systems
• HDFS-based systems (e.g., hadoop)
• Stream management systems: Apache Kafka
• Others?
2. Data Models
What you will learn about in this section

1. Types of Data

2. Data Models
Data is highly heterogeneous

- Structured data
- Semi-structured data
- Unstructured data

Increasing amounts of data
Structured data

• Information with a high degree of organization

• All data conforms to a schema. Ex: business data

• Easy to query, search over, aggregate

• Example: tables in a database, tables in excel, etc.
Semi-structured data

• Some structure in the data but implicit and irregular

• It contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data

• Example: JSON, HTML, XML
Unstructured data

• Information that either does not have a pre-defined structure or is not organized in a pre-defined manner.

• Text, video, images, etc.

• Abundant and extremely valuable. Hard to query, aggregate, analyze, search.
Data Model

• A data model is a collection of concepts for describing data.

• A schema is a description of a particular collection of data, using the given data model.

• A data model enables users to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk.
Levels of abstraction

- **External Schema**: schema seen by apps.
- **Conceptual Schema**: a.k.a. logical schema, describes stored data in terms of data model.
- **Physical Schema**: includes storage details, file organization, indexes.
- **Disk**: Classical picture. Remember it!
Data models

- Relational
- Key/Value
- Graph
- Document
- Column-family
- Array/Matrix
- Hierarchical
- Network

Most database management systems
Data models

- Relational
- Key/Value
- Graph
- Document
- Column-family
- Array/Matrix
- Hierarchical
- Network

No SQL
Data models

- Relational
- Key/Value
- Graph
- Document
- Column-family
- **Array/Matrix**
- Hierarchical
- Network

Machine learning, Scientific applications
Data models

• Relational
• Key/Value
• Graph
• Document
• Column-family
• Array/Matrix
• Hierarchical
• Network

Obsolete / Rare
3. RDBMs and the Relational Data Model
What you will learn about in this section

1. Definition of DBMS
2. Data models & the relational data model
3. Schemas & data independence
What is a DBMS?

• A large, integrated collection of data

• Models a real-world *enterprise*
  • *Entities* (e.g., Students, Courses)
  • *Relationships* (e.g., Alice is enrolled in CS564)

A **Database Management System (DBMS)** is a piece of software designed to store and manage databases.
A Motivating, Running Example

• Consider building a course management system (**CMS**):
  
  • Students
  • Courses
  • Professors

  \[ \text{Entities} \]

• Who takes what
• Who teaches what

  \[ \text{Relationships} \]
Data models

- A **data model** is a collection of concepts for describing data
  - The relational model of data is the most widely used model today
    - Main Concept: the *relation*- essentially, a table

- A **schema** is a description of a particular collection of data, **using the given data model**
  - E.g. every *relation* in a relational data model has a *schema* describing types, etc.
Modeling the Course Management System

• **Logical Schema**
  • Students(sid: string, name: string, gpa: float)
  • Courses(cid: string, cname: string, credits: int)
  • Enrolled(sid: string, cid: string, grade: string)
Modeling the Course Management System

• Logical Schema
  • Students(sid: string, name: string, gpa: float)
  • Courses(cid: string, cname: string, credits: int)
  • Enrolled(sid: string, cid: string, grade: string)
Other Schemata...

• **Physical Schema**: describes data layout
  • Relations as unordered files
  • Some data in sorted order (index)

• **Logical Schema**: Previous slide

• **External Schema**: (Views)
  • Course_info(cid: string, enrollment: integer)
  • Derived from other tables
Data independence

**Concept:** Applications do not need to worry about *how the data is structured and stored*

**Logical data independence:**
protection from changes in the logical structure of the data

I.e. should not need to ask: *can we add a new entity or attribute without rewriting the application?*

**Physical data independence:**
protection from physical layout changes

I.e. should not need to ask: *which disks are the data stored on? Is the data indexed?*

One of the most important reasons to use a DBMS
Relational Model

- **Structure**: The definition of relations and their contents.
- **Integrity**: Ensure the database’s contents satisfy constraints.
- **Manipulation**: How to access and modify a database’s contents.
Tables in the Relational Model

A **relation** or **table** is a multiset of tuples having the attributes specified by the schema.

Let’s break this definition down.

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

**Product**

- **PName**
- **Price**
- **Manufacturer**
### Tables in the Relational Model

<table>
<thead>
<tr>
<th>Product</th>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td></td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td></td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td></td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td></td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

A **multiset** is an unordered list (or: a set with multiple duplicate instances allowed)

- List: \([1, 1, 2, 3]\)
- Set: \(\{1, 2, 3\}\)
- Multiset: \(\{1, 1, 2, 3\}\)

i.e. no `next()`, etc. methods!
## Tables in the Relational Model

An **attribute** (or **column**) is a typed data entry present in each tuple in the relation.

Attributes must have an **atomic** type, i.e. not a list, set, etc.

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
Tables in the Relational Model

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

Also referred to sometimes as a record

A tuple or row is a single entry in the table having the attributes specified by the schema.
# Tables in the Relational Model

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

The number of tuples is the **cardinality** of the relation.

The number of attributes is the **arity** of the relation.

$n$-ary Relation = Table with $n$ columns
Data Types in Relational Model

• Atomic types:
  • Characters: CHAR(20), VARCHAR(50)
  • Numbers: INT, BIGINT, SMALLINT, FLOAT
  • Others: MONEY, DATETIME, ...

• Every attribute must have an atomic type
  • Hence tables are flat
Table Schemas

• The **schema** of a table is the table name, its attributes, and their types:

```
Product(Pname: string, Price: float, Category: string, Manufacturer: string)
```

• A **key** is an attribute whose values are unique; we underline a key

```
Product(Pname: string, Price: float, Category: string, Manufacturer: string)
```
Key constraints

A **key** is a **minimal subset of attributes** that acts as a unique identifier for tuples in a relation.

- A key is an implicit constraint on which tuples can be in the relation.
- i.e. if two tuples agree on the values of the key, then they must be the same tuple!

Students(sid:string, name:string, gpa: float)

1. Which would you select as a key?
2. Is a key always guaranteed to exist?
3. Can we have more than one key?
NULL and NOT NULL

- To say “don’t know the value” we use **NULL**
- NULL has (sometimes painful) semantics, more details later

```
Students(sid:string, name:string, gpa: float)
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bob</td>
<td>3.9</td>
</tr>
<tr>
<td>143</td>
<td>Jim</td>
<td>NULL</td>
</tr>
</tbody>
</table>

*Say, Jim just enrolled in his first class.*

We may constrain a column to be NOT NULL, e.g., “name” in this table.
Foreign Key constraints

• A foreign key specifies that an attribute from one relation has to map to a tuple in another relation.
Foreign Key constraints

• Suppose we have the following schema:

Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)

• And we want to impose the following constraint:
  • ‘Only real students may enroll in courses’ i.e. a student must appear in the Students table to enroll in a class

<table>
<thead>
<tr>
<th>Students</th>
<th>Enrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>101</td>
<td>Bob</td>
</tr>
<tr>
<td>123</td>
<td>Mary</td>
</tr>
</tbody>
</table>

We say that student_id is a **foreign key** that refers to Students
Summary of Schema Information

• Schema and Constraints are how databases understand the semantics (meaning) of data

• They are also useful for optimization
DATA MANIPULATION LANGUAGES (DML)

• How to store and retrieve information from a database.

• Procedural: The query specifies the (high-level) strategy the DBMS should use to find the desired result.

• We will see SQL and Relational Algebra