Welcome to CSCI E-66!

- This is a course on databases, but it's also more than that.
- We'll look at different ways of storing/managing data.
- Key lesson: there are multiple approaches to data-management problems.
  - one size doesn't fit all!
- Key goal: to be able to choose the right solution for a given problem.
Data, Data Everywhere!

- financial data
- commercial data
- scientific data
- socioeconomic data
- etc.

Databases and DBMSs

- A database is a collection of related data.
  - refers to the data itself, not the program

- Managed by some type of database management system (DBMS)
The Conventional Approach

- Use a DBMS that employs the relational model
  - RDBMS = relational database management system
  - use the SQL query language
- Examples: IBM DB2, Oracle, Microsoft SQL Server, MySQL
- Typically follow a client-server model
  - the database server manages the data
  - applications act as clients
- Extremely powerful
  - SQL allows for more or less arbitrary queries
  - support transactions and the associated guarantees

Transactions

- A transaction is a sequence of operations that is treated as a single logical operation.
- Example: a balance transfer
  

  \[
  \text{transaction } T1 \\
  \text{read balance1} \\
  \text{write(balance1 - 500)} \\
  \text{read balance2} \\
  \text{write(balance2 + 500)} \\
  \]

- Other examples:
  - making a flight reservation
    select flight, reserve seat, make payment
  - making an online purchase
  - making an ATM withdrawal
- Transactions are all-or-nothing: all of a transaction’s changes take effect or none of them do.
Why Do We Need Transactions?

• To prevent problems stemming from system failures.
  
  • example 1:
    
    transaction
    
    ```
    read balance1
    write(balance1 - 500)
    CRASH
    read balance2
    write(balance2 + 500)
    ```
    
    • what should happen?
  
  • example 2:
    
    transaction
    
    ```
    read balance1
    write(balance1 - 500)
    read balance2
    write(balance2 + 500)
    user told "transfer done"
    CRASH
    ```
    
    • what should happen?

Why Do We Need Transactions? (cont.)

• To ensure that operations performed by different users don’t overlap in problematic ways.

  • example: what’s wrong with the following?

    user 1’s transaction
    
    ```
    read balance1
    write(balance1 - 500)
    read balance2
    ```
    
    user 2’s transaction
    
    ```
    read balance1
    read balance2
    if (balance1 + balance2 < min)
    write(balance1 - fee)
    write(balance2 + 500)
    ```
    
    • how could we prevent this?
Limitations of the Conventional Approach

- Can be overkill for applications that don't need all the features
- Can be hard / expensive to setup / maintain / tune
- May not provide the necessary functionality
- Footprint may be too large
  - example: can't put a conventional RDBMS on a small embedded system
- May be unnecessarily slow for some tasks
  - overhead of IPC, query processing, etc.
- Does not scale well to large clusters

Example Problem I: User Accounts

- Database of user information for email, groups, etc.
- Used to authenticate users and manage their preferences
- Needs to be extremely fast and robust
- Don't need SQL. Why?
- Possible solution: use a key-value store
  - key = user id
  - value = password and other user information
  - less overhead and easier to manage than an RDBMS
  - still very powerful: transactions, recovery, replication, etc.
Example Problem II: Web Services

- Services provided or hosted by Google, Amazon, etc.
  - Google Analytics, Earth, Maps, Gmail, etc.
  - Netflix, Pinterest, Reddit, Flipboard, GitHub, etc.

- Can involve huge amounts of data / traffic

- Scalability is crucial
  - Load can increase rapidly and unpredictably
  - Use large clusters of commodity machines

- Conventional RDBMSs don't scale well in this way.

- Solution: some flavor of noSQL

What Other Options Are There?

- View a DBMS as being composed of two layers.

- At the bottom is the storage layer or storage engine.
  - Stores and manages the data

- Above that is the logical layer.
  - Provides an abstract representation of the data
  - Based on some data model
  - Includes some query language, tool, or API for accessing and modifying the data

- To get other approaches, choose different options for the layers.
Options for the Logical Layer (partial list)

- relational model + SQL
- object-oriented model + associated query language
- XML + XPath or XQuery
- JSON + associated API
- key-value pairs + associated API
- graph-based model + associated API/query language
- comma-delimited or tab-delimited text + tool for text search

Options for the Storage Layer (partial list)

- transactional storage engine
  - supports transactions, recovery, etc.
- a non-transactional engine that stores data on disk
- an engine that stores data in memory
- a column store that stores columns separately from each other
  - vs. a traditional row-oriented approach
  - beneficial for things like analytical-processing workloads
Course Overview

- data models/representations (logical layer), including:
  - entity-relationship (ER): used in database design
  - relational (including SQL)
  - object-oriented and object-relational
  - semistructured: XML, JSON
  - noSQL variants

- implementation issues (storage layer), including:
  - storage and indexing structures
  - transactions
  - concurrency control
  - logging and recovery
  - distributed databases and replication

Course Components

- Lectures and weekly sections
  - sections: optional but recommended; start next week; times and locations TBA
  - also available by streaming and recorded video

- Five problem sets
  - several will involve programming in Java
  - all will include written questions
  - grad-credit students will complete extra problems
  - must be your own work
    - see syllabus or website for the collaboration policy

- Midterm exam

- Final exam
Prerequisites

- A good working knowledge of Java
- A course at the level of CSCI E-22
- Experience with fairly large software systems is helpful.

Course Materials

- Lecture notes will be the primary resource.
- Other options:
  - *Database Management Systems* by Ramakrishnan and Gehrke (McGraw-Hill)
  - *Database System Concepts* by Silberschatz et al. (McGraw-Hill)
Additional Administrivia

• Instructor: Dave Sullivan
• TAs: Cody Doucette, Eli Saracino
• Office hours and contact info. will be available on the Web: http://sites.fas.harvard.edu/~cscie66
• For questions on content, homework, etc.:
  • use Piazza
  • send e-mail to cscie66@fas.harvard.edu

Database Design

• In database design, we determine:
  • which pieces of data to include
  • how they are related
  • how they should be grouped/decomposed
• End result: a logical schema for the database
  • describes the contents and structure of the database
ER Models

- An *entity-relationship (ER) model* is a tool for database design.
  - graphical
  - implementation-neutral

- ER models specify:
  - the relevant entities (“things”) in a given domain
  - the relationships between them

Sample Domain: A University

- Want to store data about:
  - employees
  - students
  - courses
  - departments

- How many tables do you think we’ll need?
  - can be hard to tell before doing the design
  - in particular, hard to determine which tables are needed to encode relationships between data items
Entities: the “Things”

- Represented using rectangles.

- Examples:

  ![Entity Diagram]

- Strictly speaking, each rectangle represents an *entity set*, which is a collection of individual entities.

Attributes

- Associated with entities are *attributes* that describe them.
  - represented as ovals connected to the entity by a line
  - double oval = attribute that can have multiple values
  - dotted oval = attribute that can be derived from other attributes

![Attribute Diagram]
Keys

- A key is an attribute or collection of attributes that can be used to uniquely identify each entity in an entity set.
- An entity set may have more than one possible key.
  - example:

  ![Diagram of Person entity with attributes id, name, address, email, age]

  - possible keys include:

Candidate Key

- A candidate key is a minimal collection of attributes that is a key.
  - minimal = no unnecessary attributes are included
    - not the same as minimum
  - Example: assume (name, address, age) is a key for Person
    - it is a minimal key because we lose uniqueness if we remove any of the three attributes:
      - (name, address) may not be unique
        - e.g., a father and son with the same name and address
      - (name, age) may not be unique
      - (address, age) may not be unique
  - Example: (id, email) is a key for Person
    - it is not minimal, because just one of these attributes is sufficient for uniqueness
    - therefore, it is not a candidate key
Key vs. Candidate Key

- Consider an entity set for books:

  ![ER Diagram for Books]

  - isbn
  - author_id
  - title

  **Question:** Is isbn a primary key or a candidate key?

  **Answer:** Yes, isbn is a primary key.

  - author_id, title
  - author_id, isbn
  - author_id

  **Question:** What are the candidate keys?

  **Answer:** The candidate keys are:
  - author_id, title
  - author_id, isbn
  - author_id

Primary Key

- We typically choose one of the candidate keys as the **primary key**.
- In an ER diagram, we underline the primary key attribute(s).
Relationships Between Entities

- Relationships between entities are represented using diamonds that are connected to the relevant entity sets.
- For example: students are enrolled in courses

- Another example: courses meet in rooms

Relationships Between Entities (cont.)

- Strictly speaking, each diamond represents a relationship set, which is a collection of relationships between individual entities.

- In a given set of relationships:
  - an individual entity may appear 0, 1, or multiple times
  - a given combination of entities may appear at most once
    - example: the combination (CS 105, CAS 315) may appear at most once
Attributes of Relationships

- A relationship set can also have attributes.
  - they specify info. associated with the relationships in the set

- Example:
  
  ![Diagram of Person, Enrolled, and Course entities with credit status attribute]

Key of a Relationship Set

- A key of a relationship set can be formed by taking the union of the primary keys of its participating entities.
  - example: (person.id, course.name) is a key of enrolled

- The resulting key may or may not be a primary key. Why?
Degree of a Relationship Set

- "enrolled" is a binary relationship set: it connects two entity sets.
  - degree = 2

- It's also possible to have higher-degree relationship sets.
- A ternary relationship set connects three entity sets.
  - degree = 3

Relationships with Role Indicators

- It's possible for a relationship set to involve more than one entity from the same entity set.

- For example: every student has a faculty advisor, where students and faculty members are both members of the Person entity set.

- In such cases, we use role indicators (labels on the lines) to distinguish the roles of the entities in the relationship.

- Relationships like this one are referred to as recursive relationships.
Cardinality (or Key) Constraints

- A *cardinality constraint* (or *key constraint*) limits the number of times that a given entity can appear in a relationship set.

- Example: each course meets in *at most one* (i.e., 0 or 1) room

```
Course -- Meet In --> Room
```

- A key constraint specifies a functional mapping from one entity set to another.
  - each course is mapped to at most one room (course → room)
  - as a result, each course appears in at most one relationship in the *meets in* relationship set

- The arrow in the ER diagram has same direction as the mapping.
  - note: the R&G book uses a different convention for the arrows

Cardinality Constraints (cont.)

- The presence or absence of cardinality constraints divides relationships into three types:
  - many-to-one
  - one-to-one
  - many-to-many

- We'll now look at each type of relationship.
Many-to-One Relationships

- Meets In is an example of a *many-to-one* relationship.
- We need to specify a *direction* for this type of relationship.
  - example: Meets In is many-to-one from *Course* to *Room*

In general, in a many-to-one relationship from A to B:

- an entity in A can be related to *at most one* entity in B
- an entity in B can be related to an arbitrary number of entities in A (0 or more)

Picturing a Many-to-One Relationship

- Each course participates in at most one relationship, because it can meet in at most one room.
- Because the constraint only specifies a maximum (*at most one*), it's possible for a course to not meet in any room (e.g., CS 610).
Another Example of a Many-to-One Relationship

- The diagram above says that:
  - a given book can be borrowed by at most one person
  - a given person can borrow an arbitrary number of books
- Borrows is a many-to-one relationship from Book to Person.
- We could also say that Borrows is a one-to-many relationship from Person to Book.
  - one-to-many is the same thing as many-to-one, but the direction is reversed

One-to-One Relationships

- In a one-to-one relationship involving A and B: [not from A to B]
  - an entity in A can be related to at most one entity in B
  - an entity in B can be related to at most one entity in A
- We indicate a one-to-one relationship by putting an arrow on both sides of the relationship:
- Example: each department has at most one chairperson, and each person chairs at most one department.
Many-to-Many Relationships

- In a *many-to-many relationship* involving A and B:
  - an entity in A can be related to an arbitrary number of entities in B (0 or more)
  - an entity in B can be related to an arbitrary number of entities in A (0 or more)

- If a relationship has no cardinality constraints specified (i.e., if there are no arrows on the connecting lines), it is assumed to be many-to-many.

How can we indicate that each student has at most one major?

- *Majors In* is what type of relationship in this case?
What if each student can have more than one major?

- *Majors In* is what type of relationship in this case?

Another Example

- How can we indicate that each student has at most one advisor?

- *Advises* is what type of relationship?
Cardinality Constraints and Ternary Relationship Sets

- The arrow into "study group" encodes the following constraint: "a person studies in at most one study group for a given course."

- In other words, a given (person, course) combination is mapped to at most one study group.
  - A given person or course can itself appear in multiple studies-in relationships

Other Details of Cardinality Constraints

- For relationship sets of degree >= 3, we use at most one arrow, since otherwise the meaning can be ambiguous.

- It is unclear whether this diagram means that:
  1) each person is mapped to at most one (course, study group) combo
  2) each (person, course) combo is mapped to at most one study group
  and
  each (person, study group) combo is mapped to at most one course
Participation Constraints

- Cardinality constraints allow us to specify that each entity will appear *at most* once in a given relationship set.

- Participation constraints allow us to specify that each entity will appear *at least* once (i.e., 1 or more times).
  - indicate using a thick line (or double line)

- Example: each department must have at least one chairperson.

- We say Department has *total participation* in Chairs.
  - by contrast, Person has *partial participation*

Participation Constraints (cont.)

- We can combine cardinality and participation constraints.

- a person chairs at most one department
  - specified by which arrow?
- a department has ____________ person as a chair
The Relational Model: A Brief History


- Earlier data models were closely tied to the physical representation of the data.

- The model was revolutionary because it provided data independence – separating the logical model of the data from its underlying physical representation.

- Allows users to access the data without understanding how it is stored on disk.

- Codd won the Turing Award (computer science’s Nobel Prize) in 1981 for his work.

The Relational Model: Basic Concepts

- A database consists of a collection of tables.

- Example of a table:

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>class</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>Jill Jones</td>
<td>Canaday C-54</td>
<td>2011</td>
<td>3/10/85</td>
</tr>
<tr>
<td>25252525</td>
<td>Alan Turing</td>
<td>Lowell House F-51</td>
<td>2008</td>
<td>2/7/88</td>
</tr>
<tr>
<td>33566891</td>
<td>Audrey Chu</td>
<td>Pfoho, Moors 212</td>
<td>2009</td>
<td>10/2/86</td>
</tr>
<tr>
<td>45678900</td>
<td>Jose Delgado</td>
<td>Eliot E-21</td>
<td>2009</td>
<td>7/13/88</td>
</tr>
<tr>
<td>66666666</td>
<td>Count Dracula</td>
<td>The Dungeon</td>
<td>2007</td>
<td>11/1431</td>
</tr>
</tbody>
</table>

- Each row in a table holds data that describes either:
  - an entity
  - a relationship between two or more entities

- Each column in a table represents one attribute of an entity.
  - each column has a domain of possible values
Relational Model: Terminology

- Two sets of terminology:
  
  - table = relation
  - row = tuple
  - column = attribute

- We'll use both sets of terms.

Requirements of a Relation

- Each column must have a unique name.

- The values in a column must be of the same type (i.e., must come from the same domain).
  
  - integers, real numbers, dates, strings, etc.

- Each cell must contain a single value.
  
  - example: we can't do something like this:
    
    | id     | name      | ... | phones             |
    |--------|-----------|-----|--------------------|
    | 12345678 | Jill Jones | ... | 123-456-5678, 234-666-7890 |
    | 25252525 | Alan Turing | ... | 777-777-7777, 111-111-1111 |
    | ...     | ...       |     | ...                |

- No two rows can be identical.
  
  - identical rows are known as duplicates
Null Values

- By default, the domains of most columns include a special value called null.

- Null values can be used to indicate that:
  - the value of an attribute is unknown for a particular tuple
  - the attribute doesn't apply to a particular tuple. example:

```
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>...</th>
<th>major</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>Jill Jones</td>
<td>...</td>
<td>computer science</td>
</tr>
<tr>
<td>25252525</td>
<td>Alan Turing</td>
<td>...</td>
<td>mathematics</td>
</tr>
<tr>
<td>33333333</td>
<td>Dan Dabbler</td>
<td>...</td>
<td>null</td>
</tr>
</tbody>
</table>
```

Relational Schema

- The schema of a relation consists of:
  - the name of the relation
  - the names of its attributes
  - the attributes' domains (although we'll ignore them for now)

- Example:
  
```
Student(id, name, address, email, phone)
```

- The schema of a relational database consists of the schema of all of the relations in the database.
ER Diagram to Relational Database Schema

• Basic process:
  • entity set → a relation with the same attributes
  • relationship set → a relation whose attributes are:
    • the primary keys of the connected entity sets
    • the attributes of the relationship set

• Example of converting a relationship set:

  in addition, we would create a relation for each entity set

Renaming Attributes

• When converting a relationship set to a relation, there may be multiple attributes with the same name.
  • need to rename them

• Example:

  We may also choose to rename attributes for the sake of clarity.
Special Case: Many-to-One Relationship Sets

• Ordinarily, a binary relationship set will produce three relations:
  • one for the relationship set
  • one for each of the connected entity sets

• Example:
  
  MeetsIn(course_name, room_name)
  Course(name, start_time, end_time)
  Room(name, capacity)

Special Case: Many-to-One Relationship Sets (cont.)

• However, if a relationship set is many-to-one, we often:
  • eliminate the relation for the relationship set
  • capture the relationship set in the relation used for the entity set on the many side of the relationship

  MeetsIn(course_name, room_name)
  Course(name, start_time, end_time, room_name)
  Room(name, capacity)
Special Case: Many-to-One Relationship Sets (cont.)

- Advantages of this approach:
  - makes some types of queries more efficient to execute
  - uses less space

<table>
<thead>
<tr>
<th>name</th>
<th>course_name</th>
<th>room_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cscie50b</td>
<td>cscie50b</td>
<td>Sci Ctr B</td>
</tr>
<tr>
<td>cscie119</td>
<td>cscie119</td>
<td>Sever 213</td>
</tr>
<tr>
<td>cscie160</td>
<td>cscie160</td>
<td>Sci Ctr A</td>
</tr>
<tr>
<td>cscie268</td>
<td>cscie268</td>
<td>Sci Ctr A</td>
</tr>
</tbody>
</table>

- If one or more entities don't participate in the relationship, there will be null attributes for the fields that capture the relationship:

<table>
<thead>
<tr>
<th>name</th>
<th>room_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cscie50b</td>
<td>Sci Ctr B</td>
</tr>
<tr>
<td>cscie119</td>
<td>Sever 213</td>
</tr>
<tr>
<td>cscie160</td>
<td>Sci Ctr A</td>
</tr>
<tr>
<td>cscie268</td>
<td>Sci Ctr A</td>
</tr>
<tr>
<td>cscie160</td>
<td>NULL</td>
</tr>
</tbody>
</table>

- If a large number of entities don't participate in the relationship, it may be better to use a separate relation.
Special Case: One-to-One Relationship Sets

- Here again, we're able to have only two relations – one for each of the entity sets.

- In this case, we can capture the relationship set in the relation used for either of the entity sets.

- Example:

  ![Diagram of one-to-one relationship]

  \[
  \text{Person}(id, \text{name, chaired_dept}) \quad \text{OR} \quad \text{Person}(\text{name, id})
  \]

  \[
  \text{Department}(\text{name, office}) \quad \text{OR} \quad \text{Department}(\text{name, office, chair_id})
  \]

  - which of these would probably make more sense?

Many-to-Many Relationship Sets

- For many-to-many relationship sets, we need to use a separate relation for the relationship set.

- Example:

  ![Diagram of many-to-many relationship]

  - can't capture the relationships in the Student table
    - a given student can be enrolled in multiple courses
  
  - can't capture the relationships in the Course table
    - a given course can have multiple students enrolled in it
  
  - need to use a separate table:

    \[
    \text{Enrolled}(\text{student_id, course_name, credit_status})
    \]
Recall: Primary Key

- We typically choose one of the candidate keys as the primary key.
- In an ER diagram, we underline the primary key attribute(s).
  
  ![ER Diagram of Person Entity Set]

- In the relational model, we also designate a primary key by underlying it.
  
  \[ \text{Person}(\text{id, name, address, \ldots}) \]

- A relational DBMS will ensure that no two rows have the same value / combination of values for the primary key.
  - example: it won't let us add two people with the same id

Primary Keys of Relations for Entity Sets

- When translating an entity set to a relation, the relation gets the same primary key as the entity set.
  
  ![Diagram showing translation from entity set to relation]

  \[ \text{Student}(\text{id, \ldots}) \]

  \[ \text{Course}(\text{name, \ldots}) \]
When translating a relationship set to a relation, the primary key depends on the cardinality constraints.

For a many-to-many relationship set, we take the union of the primary keys of the connected entity sets.

![Diagram](image)

$\rightarrow$ Enrolled($student\_id$, $course\_name$, $credit\_status$)

- doing so prevents a given combination of entities from appearing more than once in the relation
- it still allows a single entity (e.g., a single student or course) to appear multiple times, as part of different combinations

For a many-to-one relationship set, if we decide to use a separate relation for it, what should that relation's primary key include?

![Diagram](image)

$\rightarrow$ Borrows($person\_id$, $isbn$)
Primary Keys of Relations for Relationship Sets (cont.)

- For a *many-to-one* relationship set, if we decide to use a separate relation for it, what should that relation's primary key include?

  \[ \text{Borrows(person\_id, isbn)} \]

- limiting the primary key enforces the cardinality constraint
  - in this example, the DBMS will ensure that a given book is borrowed by at most once person
- how else could we capture this relationship set?

Primary Keys of Relations for Relationship Sets (cont.)

- For a *one-to-one* relationship set, what should the primary key of the resulting relation be?

  \[ \text{Chairs(person\_id, department\_name)} \]
Foreign Keys

• A foreign key is attribute(s) in one relation that take on values from the primary-key attribute(s) of another (foreign) relation

• Example: MajorsIn has two foreign keys

![Diagram of foreign keys in a relational database](image)

• We use foreign keys to capture relationships between entities.