Introduction to Relational Databases

Peter Bazeley

BIPG 540/740
Overview

- DBMS vs. Flat Files
- The Relational Model
- Relations
- Schemas
- Primary Keys
- Relationships
- Relationships Examples
- Derived Attributes, Views
Database Management Systems (DBMSs) are software systems that facilitate management and access of data.

A relational DBMS (RDBMS) is a database system that uses the relational data model.

Other data models include hierarchical, network, object-oriented, and object-relational.

The relational model is the most popular.
DBMS vs. Flat Files

- Why use a DBMS rather than storing everything in flat files?

- Ultimately, it depends on the task at a hand

- DBMSs take care of data storage and access details

- Is this useful or an inconvenience?
Flat Files: Cons

- Must write a custom program **every time** a new search is needed
  - Searches are limited by structure of files
  - Alternatively, could write code library of access routines, but this is more work and flexibility must be considered

- Need to consider concurrent access details
  - Multiple people editing records
  - Accessing a record that is being deleted by another person

- Need to consider access/security issues
  - Who can access which parts of the database
  - How will access be managed?
DBMS: Pros

- Data storage/access abstraction
  - Don’t have to worry about how/where data stored
  - Implementation of low-level access routines not required

- Efficient searching/updating
  - DBMSs use sophisticated, semi-optimized access routines
  - Further optimization available

- Data integrity check mechanisms available
  - e.g. to avoid adding a record that already exists
  - Or to make sure data entered conforms to certain specifications
DBMS: Pros Cont’d

- Access/security management built-in
- Concurrent access details taken care of
- Reduced application development time
- Convenient, powerful stand-alone access tool
- Uniform, consistent access methods
DBMS: Cons

- In certain cases, data access can be slower
  - It’s faster to read from disk than a DBMS
  - Highly specialized searches may be completed more quickly by custom programs
  - Data manipulation facilities may be inconvenient

- May need to retrieve data in a way not supported
  - e.g. complex text manipulation
  - Operation that works on multiple rows

- Still limited by structure of database
  - Must conform data/tasks to database structure
  - How long will it take to conform your data for loading?
Flat Files vs. DBMS

- Depends on task
  - How long will database be used?
  - Who needs to be able to access data, and how?
  - How complex is data?
  - How complex are searches?

- Both types of databases will require sufficient planning for future needs
The Relational Model

- The central concept in the relational model is the relation.

- Think of a relation as a collection of “things”:
  - collection of students
  - collection of genes

- These “things” are called records or tuples.
The Relation

- Each relation has one or more characteristics, known as *attributes* or *fields*

  - **Student**: address, GPA, phone number, …
  - **Gene**: sequence, function, chromosome, …

- Each record in a relation has these attributes
  - They have different values, of course
  - Attribute values can be missing/empty (more on this in later lectures)
Schemas

- A description of a relation is called a **schema**

Schemas consist of:

- The **name** of the relation
- A **list** of the attributes in a relation
- The **type** or **domain** of each attribute
  - Number (integer, real, etc.)
  - Character (single character, string of characters, etc.)
  - Logical (True/False)
Student Relation Schema

- Relation: Student

- Attributes:
  - Name - character string
  - Age - integer
  - Phone number - character string
  - G.P.A. - real number
Gene Relation Schema

- Relation: Gene

- Attributes:
  - Name - character string
  - Sequence - character string
  - Function - character string
  - Chromosome - integer
Schema Diagrams

- Often you will see schemas in box diagrams:

<table>
<thead>
<tr>
<th>GENE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gene ID - <em>serial</em></td>
</tr>
<tr>
<td>Name - <em>varchar</em></td>
</tr>
<tr>
<td>Sequence - <em>varchar</em></td>
</tr>
<tr>
<td>Function - <em>varchar</em></td>
</tr>
<tr>
<td>Chromosome - <em>integer</em></td>
</tr>
</tbody>
</table>

- Although the format may differ
- More on “Gene ID” later
Relations and Tables

- An instance of a relation is table.

- Tables have:
  - Rows
    - each row is a record
    - student 1, student 2, etc.
  - Columns
    - each column is an attribute
    - name, phone number, age, G.P.A., etc.

- Think of a relation as the abstract idea and a table as an actual set of records.
<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Phone Number</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>19</td>
<td>419-383-2879</td>
<td>3.4</td>
</tr>
<tr>
<td>Sarah Jones</td>
<td>21</td>
<td>419-383-3120</td>
<td>3.1</td>
</tr>
<tr>
<td>Tim Roberts</td>
<td>20</td>
<td>419-383-4560</td>
<td>2.5</td>
</tr>
<tr>
<td>Name</td>
<td>Sequence</td>
<td>Function</td>
<td>Chromosome</td>
</tr>
<tr>
<td>------</td>
<td>---------------</td>
<td>----------------</td>
<td>------------</td>
</tr>
<tr>
<td>abc</td>
<td>ATGGCCAA...</td>
<td>oxidize fat</td>
<td>2</td>
</tr>
<tr>
<td>efg</td>
<td>TGGACTTA...</td>
<td>transport $Ca^{2+}$</td>
<td>13</td>
</tr>
<tr>
<td>hij</td>
<td>CTAGATCA...</td>
<td>structural</td>
<td>6</td>
</tr>
</tbody>
</table>
Primary Keys

- Each record must be uniquely identifiable.

- Otherwise there is no way to differentiate records.

- A set of one or more attributes that uniquely identifies a record is called a candidate key or just key.

- If more than one key exists, one of these is chosen, and is called the primary key.
Primary Keys

- Usually, primary keys are created by adding a new attribute, which has type “serial”
  - A sequential set of unique numbers
  - 1, 2, 3, ...

- Alternatively, the primary key can be a set of existing attributes:
  - (name, age, phone number)
  - As long as the record is uniquely identified, any combination of attributes is acceptable
<table>
<thead>
<tr>
<th>Student ID</th>
<th>Name</th>
<th>Age</th>
<th>Phone Number</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>John Smith</td>
<td>19</td>
<td>419-383-2879</td>
<td>3.4</td>
</tr>
<tr>
<td>312</td>
<td>Sarah Jones</td>
<td>21</td>
<td>419-383-3120</td>
<td>3.1</td>
</tr>
<tr>
<td>057</td>
<td>Tim Roberts</td>
<td>20</td>
<td>419-383-4560</td>
<td>2.5</td>
</tr>
</tbody>
</table>
## Gene Table Primary Key

<table>
<thead>
<tr>
<th>Gene ID</th>
<th>Name</th>
<th>Sequence</th>
<th>Function</th>
<th>Chromosome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abc</td>
<td>ATGGCCAA...</td>
<td>oxidize fat</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>efg</td>
<td>TGGACTTA...</td>
<td>transport Ca2+</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>hij</td>
<td>CTAGATCA...</td>
<td>structural</td>
<td>6</td>
</tr>
</tbody>
</table>
Gene Table Schema

- In schema diagrams, the primary key is usually annotated.

**GENE**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gene ID</td>
<td>serial</td>
<td>(PK)</td>
</tr>
<tr>
<td>Name</td>
<td>varchar</td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td>varchar</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>varchar</td>
<td></td>
</tr>
<tr>
<td>Chromosome</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>
Multiple Relations

- More often than not your database will have multiple relations
  - Student, college, residence hall, course, ...
  - Gene, chromosome, genome, organism, ...

- The utility of the relational model is being able to link these various relations
Multiple Relations

- What do I mean by “linking relations”?
  => **Relationships**

- **Students**
  - Belong to a college
  - Live in a residence hall
  - Enroll in several courses

- **Genes**
  - Are located in a chromosome
  - Exist in an organism
Relationships

- Relationships can be:
  - One-to-one
    - One bed per student
    - One genome per organism
    - These could be in the same relation
  - One-to-many
    - One college for many students
    - One chromosome for many genes
  - Many-to-many
    - Many students take many courses
    - Many genes exist in many organisms
Relationships Example

- Students in a College

**STUDENT**
- Student ID - integer (PK)
- Name - varchar
- Age - integer
- Phone Number - varchar
- G.P.A. - real

**COLLEGE**
- College ID - integer (PK)
- Name - varchar
- Building Location - varchar
- Office Phone Number - varchar
Students in a College

- How do we model this relationship?

- The relations must be linked by a common attribute

- They likely don’t have any naturally common attributes
  - Students and Colleges
  - Apple and Oranges
Students in a College

- To model this relationship, we’ll put one or more attributes from one relation into the other relation
  - So, we are adding another attribute(s) to one of the relations
  - Which attribute(s)?

- We need to be able to uniquely associate the 2 relations
  - We’ll use primary keys
  - But, do we use the student’s PK or the college’s PK?
Students in a College

- Well, what kind of a relationship is this?
  - One-to-many
  - One College for many Students

- Two options:
  - Add the student’s PK to the college relation
  - Add the college’s PK to the student relation
Students in a College

- What if we store each student in the college table?

<table>
<thead>
<tr>
<th>College ID</th>
<th>Name</th>
<th>Building Location</th>
<th>Office Phone Number</th>
<th>Student ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Engineering</td>
<td>Main Street</td>
<td>419-383-1234</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Engineering</td>
<td>Main Street</td>
<td>419-383-1234</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Engineering</td>
<td>Main Street</td>
<td>419-383-1234</td>
<td>3</td>
</tr>
</tbody>
</table>
Students in a College

- Since there are many students per college, it would be cumbersome and redundant to store each student in the college table.
- Instead, we’ll store the college’s PK in the student table.

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Name</th>
<th>Age</th>
<th>Phone Number</th>
<th>GPA</th>
<th>College ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Smith</td>
<td>19</td>
<td>419-383-2879</td>
<td>3.4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Sarah Jones</td>
<td>21</td>
<td>419-383-3120</td>
<td>3.1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Tim Roberts</td>
<td>20</td>
<td>419-383-4560</td>
<td>2.5</td>
<td>5</td>
</tr>
</tbody>
</table>
Students in a College

- Storing the College ID in the Student table is much less redundant

- As an aside, redundancy can be prone to errors
  - Typing errors during data entry
  - Mistyped entries would be interpreted as distinct
To model relationships, the primary key of one relation is an attribute in another relation.
Foreign Keys

- A primary key from one relation stored in another is a **foreign key**
What about *THAT*

This symbol can be used to indicate the type of relationship

- In this case, many-to-one
- Think of the 3 legs on the left end as “many” vs. the one leg on the other end, “one”
- Accordingly, one-to-one and many-to-many symbols can also be used:
Relational Database

- A relational database is a collection of one or more relations

- Each relation is linked to the others by primary keys, directly or indirectly

- Indirectly?
  - student => college => college faculty member
  - gene => chromosome => genome
Derived vs. Stored Attributes

- So far all of the attributes we have seen are stored directly in the database.

- Can also derive attributes from others stored in database:
  - Calculate age from D.O.B.
  - Compile full name from first and last name attributes.
Derived Attributes

- Age from D.O.B.
  - Find absolute DOB age in years
  - Find today’s absolute age in years
  - Subtract the DOB age from today’s age and divide by 365.25 (accounting for leap years)

- Average gene length for a chromosome
  - Sum gene lengths for chromosome
  - Divide by total number of genes on chromosome
Views

- Views are derived relations

- A collection of attributes can be derived from one or more relations and “stored” in a view

- A view is thereafter accessible, just as you would access any other table

- Updating views may or may not be allowed, depending on the database system you are using
Views

Why use a view?

Views are persistent (until database is shutdown)

- So if you are constantly creating certain derived attributes, a view would be useful
- Alternative would be to store redundant information, which isn’t recommended
  - Added consistency task
  - Waste of space
  - Views are more flexible
Reference

What Now?

- We have a few in-class challenge problems
- Any questions before then?