Chapter 4
Logical Database Design and
the Relational Model

Objectives

• Define terms for the relational data model
• Brief introduction to SQL
• Transform E-R diagrams to relations
• Create tables with entity and relational integrity constraints
Advantages of Relational Model

- Can represent all kinds of information
- Based on Math (relations)
- Natural to people
- Relatively simple
- We know how to implement it fast
Motivating Example

• Make a list of students in the class, keeping their ID, name and phone number

• You’d probably come up with something like this:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>Mike</td>
<td>111</td>
</tr>
<tr>
<td>yy</td>
<td>Elisa</td>
<td>222</td>
</tr>
</tbody>
</table>

• This is the basic structure of the relational model, a table or relation
Extra Assumptions

• You would not repeat the same row twice
• No two rows have the same ID, but they may have the same name and phone number

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>Mike</td>
<td>111</td>
</tr>
<tr>
<td>yy</td>
<td>Elisa</td>
<td>222</td>
</tr>
</tbody>
</table>

• ID would be the \textit{PRIMARY KEY (PK)}.  

Now add emails … (many!)

• Now you need to add the emails of each student, but you do not know how many emails
• Can you come up with a solution? Try it …
Many Fields

• Could come up with something like this

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Phone</th>
<th>Email1</th>
<th>Email2</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>Mike</td>
<td>111</td>
<td>bad</td>
<td>idea 😉</td>
</tr>
<tr>
<td>yy</td>
<td>Elisa</td>
<td>222</td>
<td>bad</td>
<td>idea 😞</td>
</tr>
</tbody>
</table>

• Above would not work very well. How many fields?
  – Wasted space
  – What if a student has more emails?
  – How to process it in my program?

Un-normalized

• Could also try this:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Phone</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>Mike</td>
<td>111</td>
<td><a href="mailto:mk@ad.com">mk@ad.com</a></td>
</tr>
<tr>
<td>xx</td>
<td>Mike</td>
<td>111</td>
<td><a href="mailto:mk@vu.edu">mk@vu.edu</a></td>
</tr>
<tr>
<td>yy</td>
<td>Elisa</td>
<td>222</td>
<td><a href="mailto:eli@vu.edu">eli@vu.edu</a></td>
</tr>
</tbody>
</table>

• Problem is duplication, we are repeating the name and phone number in the second row
  – What if Mike changes his phone?

• Later we will study normalization to solve this.
Now add emails … (many!)

• A much better way:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>Mike</td>
<td>111</td>
</tr>
<tr>
<td>yy</td>
<td>Elisa</td>
<td>222</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td><a href="mailto:mk@ad.com">mk@ad.com</a></td>
</tr>
<tr>
<td>xx</td>
<td><a href="mailto:mk@vu.edu">mk@vu.edu</a></td>
</tr>
<tr>
<td>yy</td>
<td><a href="mailto:eli@vu.edu">eli@vu.edu</a></td>
</tr>
</tbody>
</table>

• Every StudentID on the second table needs a matching ID on the first table: StudentID is a **FOREIGN KEY**

• In a way, StudentID in the second table is a *pointer* or *reference* to the first table

---

**Formalizing: Relations**

• Definition: A *relation* is a **named table** of data
  – Table is made up of rows (records or tuples), and columns (attributes or fields)

• Not all tables qualify as relations. Requirements:
  1. Every relation has a unique name.
  2. Every attribute value is atomic (not multivalued, not composite)
  3. Every row is unique (can’t have two rows with exactly the same values for all their fields)
  4. Attributes (columns) in tables have unique names
  5. The order of the columns is irrelevant
  6. The order of the rows is irrelevant

By definition, all relations are in **1st Normal Form (1NF)**.
Correspondence with ER Model

- Relations (tables) correspond to entity types and to many-to-many relationship types
- Rows correspond to entity instances and to many-to-many relationship instances
- Columns correspond to attributes

- NOTE: The word *relation* (in relational database) is NOT the same as the word *relationship* (in ER model)

Formalizing Key Fields

- **Primary key (PK)**
  - Minimal set of attributes that uniquely identifies a row, chosen for referencing
  - This is how we can guarantee that all rows are unique

- **Foreign key (FK)**
  - Set of attributes in a table that serves as a reference to the primary key of another table
Constraints Implied by the Keys

• Entity Integrity Constraint
  – No attribute of the PK may be null

• Referential Integrity Constraint
  – For a FK, either all attributes are null, or the values appear in the PK of a row of the referred table

Well-Structured Relations

• Bad designs have redundancy
• Well-structured relation: minimal redundancy
• Anomaly: error or inconsistency arising from bad design when changing (Insert, Update, Delete) data
• We eliminate anomalies through Normalization
Is Student a Well-Structured Relation?

Student

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Phone</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>Mike</td>
<td>111</td>
<td><a href="mailto:mk@ad.com">mk@ad.com</a></td>
</tr>
<tr>
<td>xx</td>
<td>Mike</td>
<td>111</td>
<td><a href="mailto:mk@vu.edu">mk@vu.edu</a></td>
</tr>
<tr>
<td>yy</td>
<td>Elisa</td>
<td>222</td>
<td><a href="mailto:eli@vu.edu">eli@vu.edu</a></td>
</tr>
</tbody>
</table>

Review

- Conceptual Model
- Relational Model
- Relation
- Well-Structured Relation
- Primary Key
- Foreign Key
- Entity Integrity Constraint
- Referential Integrity Constraint
Transforming E-R Into Relations

- Use a rectangle for each entity (table), with attributes inside rectangles, too
  - Can be vertical or horizontal
  - Primary key is underlined

- Use arrows from Foreign key to Primary key

E-R vs. Relational

- Entities are represented by tables
  - But tables may also represent relationships, or multivalued attributes

- Foreign Keys used to relate table rows
  - Similar to relationships in E-R, but lower level

- Relational model is more concrete, lower level
  - Usually many more tables than entities
  - Harder to understand by non-technical people
  - Directly implementable
Brief Intro to SQL

SQL

- Structured Query Language
- Standard Language for Relational Databases
- Every RDBMS implements SQL, with slight variations, including
  - Special data types
  - Stored procedures
  - Kinds of indexes, file organization ...
Basics

- Data Definition Language (DDL)
  - Create the database schema
  - CREATE / DROP / ALTER, …

- Data Manipulation Language (DML)
  - Manipulate the populated data
  - INSERT, UPDATE, DELETE, SELECT, …

- Data Control Language (DCL)
  - Manage users, permissions and such

Conventions

- -- comments until end of line
- /* can also use C-style comments */
- SQL is case insensitive (except for data)
- But we usually type reserved words in ALL CAPS
- Use single quotes for ‘character constants’
CREATE TABLE

CREATE TABLE table_name (  
    field type constraints,  
    field2 type2 ,  
    CONSTRAINT name ...,  
    ...  
);  

CREATE TABLE Book (  
    ISBN CHAR(9) PRIMARY KEY,  
    Title VARCHAR(20) UNIQUE NOT NULL,  
    Pages Integer  
);  

Common Datatypes

• CHAR(n)  
  – fixed length strings, padded with spaces at end

• VARCHAR(n)  
  – variable length strings, but no longer than n

• NUMERIC(prec, dec)  
  – fixed precision numbers (not floats)  
  – prec(ision) is total number of digits  
  – dec is how many after the decimal point  
  – NUMERIC(3,2) max value is 9.99

• DATE, TIMESTAMP  
  – Represent dates, or specific points in time
Common Constraints

• PRIMARY KEY
• NOT NULL
• UNIQUE
• REFERENCES (foreign key)
  – After REFERENCES put name of table, then field in parenthesis
  – StudentId REFERENCES Student(Id)
• CHECK
  – Allows to specify a condition or predicate
  – CHECK(age>20)

CREATE TABLE Example

CREATE TABLE Student {
  Id CHAR(3) PRIMARY KEY,
  Name VARCHAR(20) NOT NULL,
  Age INT DEFAULT 20 CHECK(Age>0 AND AGE<100),
  Gender CHAR NOT NULL,
  Deg_code CHAR(2) NOT NULL REFERENCES Degree(code),
  Major CHAR(3),
  Credits INTEGER
};
INSERT

- INSERT INTO table (fields)
  VALUES (values)

- Character constants have single quotes ‘a’

    INSERT INTO Student (Id,Name,Major,Age)
    VALUES (1, 'Amy Cremer','CS',21);

SELECT (Retrieving Data)

- SELECT fields
  FROM table
  WHERE conditions

  - Can use fields or expressions (a+3), * for all fields
  - Conditions use normal operators (=,>) and are combined with AND, OR, NOT

    SELECT *  SELECT Id,Name
    FROM Student FROM Student
    WHERE Major='CS'
More Examples

```
SELECT Id, Name, Age+5
FROM Student
WHERE Major='CS' AND Gender='F'
```

```
SELECT Id, Name, Age+5 AS AgeIn5
FROM Student
WHERE Age>=20 OR Age <=10
```

DELETE

• DELETE FROM table
  
WHERE conditions

```
DELETE
FROM Student
WHERE Id=1
```

• If no conditions, delete all data

• Does NOT delete the meta-data, use DROP TABLE for that
UPDATE (Change Data)

- UPDATE table
  SET field=value
  WHERE conditions

```sql
UPDATE Student
SET Name='Alfredo', Age=25
WHERE Id=1
;

UPDATE Student
SET Age=Age+1
WHERE Id=1
;
```

- Write CREATE TABLE statement

**Complete Example**

![Database Diagram](image)

- Write CREATE TABLE statement
Now Insert Statements

INSERT INTO Major(Id, Name) VALUES ('CS', 'Computer Science');

INSERT INTO Student(Id, Name, MajorId) VALUES (1, 'Paul Gordon', 'CS');

• Notice that field names and values match
• Notice that value of Student.MajorId needs to match a value of Major.Id
Now You Try

- Insert major for SWE, IT
- Insert students majoring in SWE, IT
Basic SELECTS

- Show all students
  
  ```
  SELECT * FROM Student;
  ```

- Now you do all majors
  
  ```
  SELECT * FROM Major;
  ```

- And then Id and Name of all students majoring in CS
  
  ```
  SELECT Id, Name
  FROM Student
  WHERE Student.MajorId = 'CS'
  ```

Updates

- Change the student with Id 2 to be CS instead of SWE
  
  ```
  UPDATE Student
  SET MajorId = 'CS'
  WHERE Id=2
  ```

- Change the name of the student with Id=3 to be Rich
Deletes

• Delete any student majoring in IT

    DELETE  
    FROM Student  
    WHERE MajorId = 'IT'  
    ;

• Now you do – delete the IT major (from the table)

From E-R to Tables
Six Cases of Transforming E-R Diagrams into Relations

1. Map Regular Entities
2. Map Binary Relationships
3. Map Weak Entities
4. Map Associative Entities
5. Map Unary Relationships
6. Map Ternary (and n-ary) Relationships

1. Mapping Regular Entities

- Create a new table for each entity
- Remember to underline the identifier (Fig. 4-8)
- For composite attributes, map only the basic pieces (Fig. 4-9)
- Derived attributes disappear
- For multivalued attributes we need a new table (Fig. 4-10)
- We may need to create several tables for independent multivalued attributes
You Try …

**BOOK**

| ISBN | Title | {Authors} | Format (Binding, NumPages, Dimensions, [Weight]) |

Corresponding SQL
Six Cases of Transforming E-R Diagrams into Relations

1. Map Regular Entities
2. Map Binary Relationships
3. Map Weak Entities
4. Map Associative Entities
5. Map Unary Relationships
6. Map Ternary (and n-ary) Relationships

Mapping Binary Relationships

- One-to-Many
  - Primary key on the one side becomes a foreign key on the many side (Fig. 4-12).
- One-to-One
  - Primary key on the mandatory side becomes a foreign key on the optional side (Fig. 4-14).
- Many-to-Many
  - Create a new relation with the primary keys of the two entities as its primary key (Fig. 4-13).
Fig. 4-12: Example of mapping a 1:M relationship
(a) Relationship between customers and orders

[Diagram showing relationship between customers and orders]

Note the mandatory one

Fig. 4-12: (b) Mapping the relationship

[Diagram showing mapping of relationship]

[Primary key on the one side becomes a foreign key on the many side]

Again, no null value in the foreign key…this is because of the mandatory minimum cardinality

One-to-Many Relationship

CREATE TABLE Program (  
  ID INT PRIMARY KEY,  
  Name VARCHAR(20)  
);  

CREATE TABLE Student (  
  ID INT PRIMARY KEY,  
  Name VARCHAR(50),  
  Major INT REFERENCES Program(ID)  
);  

• For one-to-many (or one-to-one) relationship, put a foreign key on the side that relates to one entity.
• Put any relationship attributes on that table, too.
Figure 4-14 Example of mapping a binary 1:1 relationship
a) In_charge relationship (1:1)

Often in 1:1 relationships, one direction is optional.

Fig. 4-14: (b) Resulting relations

[Primary key on the mandatory side becomes a foreign key on the optional side]
Many-to-Many Relationship

For a many-to-many, we need a new table representing the relationship.
This table has Foreign Keys to both entities.

Figure 4-13 Example of mapping an M:N relationship

a) Completes relationship (M:N)

The Completes relationship will need to become a separate relation.
b) Three resulting relations

**EMPLOYEE**

- EmployeeID
- EmployeeName
- EmployeeBirthDate

**CERTIFICATE**

- EmployeeID
- CourseID
- DateCompleted

**COURSE**

- CourseID
- CourseTitle

**You Try It …**

**PERSON**

- ID
- Name

**COUNTRY**

- ID
- Name

Composite primary key (CPK)

Foreign key

Foreign key
Six Cases of Transforming E-R Diagrams into Relations

1. Map Regular Entities
2. Map Binary Relationships
3. Map Weak Entities
4. Map Associative Entities
5. Map Unary Relationships
6. Map Ternary (and n-ary) Relationships

3. Mapping Weak Entities

- A weak entity becomes a separate relation with a foreign key taken from the strong entity
- Primary key composed of:
  - Partial identifier of weak entity
  - Primary key of identifying relation (strong entity)
    (Fig. 4-11)
Weak Entities

- Transform the strong entity normally
- For the weak entity, the PK becomes the identifier, plus the PK of the identifying entity

You Try It …

- Transform the strong entity normally
- For the weak entity, the PK becomes the identifier, plus the PK of the identifying entity
Six Cases of Transforming E-R Diagrams into Relations

1. Map Regular Entities
2. Map Binary Relationships
3. Map Weak Entities
4. Map Associative Entities
5. Map Unary Relationships
6. Map Ternary (and n-ary) Relationships

4. Mapping Associative Entities

• Identifier Not Assigned
  – Default primary key for the association relation is composed of the primary keys of the two entities (as in M:N relationship) (Fig. 4-15)

• Identifier Assigned
  – It is natural and familiar to end-users.
  – Default identifier may not be unique. (Fig. 4-16).
Figure 4-16: Mapping an associative entity

(a) Associative entity (SHIPMENT)

(b) Three resulting relations

Primary key differs from foreign keys

You Try It …
Six Cases of Transforming E-R Diagrams into Relations

1. Map Regular Entities
2. Map Binary Relationships
3. Map Weak Entities
4. Map Associative Entities
5. Map Unary Relationships
6. Map Ternary (and n-ary) Relationships

5. Mapping Unary Relationships

• Same as other relationships, except that the FK may go to the same table.
• For one-to-many, the table has a reference to other rows of the same table (Fig. 4-17).
• For many-to-many, an extra table has two FKs, both to the same table (Fig. 4-18).
**Figure 4-17** Mapping a unary 1:N relationship

(a) EMPLOYEE entity with unary relationship

A recursive FK is a FK in a relation that references the PK values of that same relation.

(b) EMPLOYEE relation with recursive foreign key

**Figure 4-18:** Mapping a unary M:N relationship

(a) Bill-of-materials relationships (M:N)

One table for the entity type.

(b) ITEM and COMPONENT relations

One table for an associative relation in which the primary key has two attributes, both taken from the primary key of the entity.
You Try It …

EMPLOYEE
SSN
Name

Boss

Worker

Supervises

You Try It …

COURSE
ID
Name

IsCorequisite

Requirer

Required

Pre

Post

IsPrerequisite

Requirer

Required

Pre

Post
Six Cases of Transforming E-R Diagrams into Relations

1. Map Regular Entities
2. Map Binary Relationships
3. Map Weak Entities
4. Map Associative Entities
5. Map Unary Relationships
6. Map Ternary (and n-ary) Relationships

6. Mapping Ternary Relationships

• One relation for each entity and one for the associative entity.
• Associative entity has foreign keys to each entity in the relationship (Fig. 4-19).
Figure 4-19 Mapping a ternary relationship

a) PATIENT TREATMENT Ternary relationship with associative entity

b) Mapping the ternary relationship PATIENT TREATMENT

A patient may receive a treatment once in the morning, then the same treatment in the afternoon.
b) Mapping the ternary relationship PATIENT TREATMENT

Remember that the primary key MUST be unique.

This is why treatment date and time are included in the composite primary key.

But this makes a very cumbersome key...

It would be better to create a surrogate key like Treatment#.

Mixed Exercises
Next Topic

• Next topic is the most important topic (theory) in this database management class.
• What is it?
• **Normalization**