Today's Lecture

- Database design through ER diagrams
- Creating and modifying relations, specifying integrity constraints using SQL
- Translate ER diagrams to relations
- A little on views

Main steps in database design

- Conceptual Design
  - Typically, entity-relationship (ER) model is used at this stage to model enterprise data
- Schema Refinement or Normalization
  - The ER model is translated into relation schema and refined. E.g., break down some relation schema into finer components to avoid redundancy
- Physical Database Design and Tuning
  - Fine tune database design by considering workloads and access patterns
  - Indexes may be created and additional relations may be maintained
ER Model

- ER model allows us to describe data in the real-world in terms of entities and their relationships.
- What are the entities and relationships in the enterprise?
- What information about these entities and relationships should we store in the database?
- What are the integrity constraints or business rules that should hold?
  - E.g., ssn is an identifier for employees in the company.
- An ER Model can be represented pictorially, called ER diagram.
- Can map an ER diagram into a relational schema.

ER Model Basics

- **Entity**: Real-world object distinguishable from other objects. An entity is described (in DB) using a set of attributes.
- **Entity Set**: A collection of similar entities. E.g., all employees.
- All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)
  - Each entity set has a key.
  - Each attribute has a domain.
**Entity set, Entity Example**

Employees

- ssn: 12-34, Name: “John”, Lot: 50
- ssn: 11-13, Name: “Gary”, Lot: 40
- ssn: 31-17, Name: “Ann”, Lot: 60

**ER Model Basics**

- **Relationship**: Association among two or more entities. E.g., Attishoo works in Pharmacy department.
- **Relationship Set**: Collection of similar relationships.
  - An \( n \)-ary relationship set \( R \) relates \( n \) entity sets \( E_1, \ldots, E_n \); each relationship in \( R \) involves entities \( e_i \in E_i, \ldots, e_n \in E_n \)
  - Same entity set could participate in different relationship sets, or in different “roles” in same set
- **Descriptive attribute, Role, Role indicator**
Relationship set, Relationship Example

Employees

Works_In

Departments

- ssn: 12-34, did: 7, since: 99
- ssn: 11-13, did: 7, since: 00
- ssn: 11-13, did: 8, since: 98
- ssn: 31-17, did: 8, since: 00
- ssn: 37-96, did: 9, since: 93

Ternary relationships

Records who works in what department at where
Key Constraints

- Consider Works_In: An employee can work in many departments; a dept can have many employees.
- Manages: each dept has at most one manager
  - Key constraint
- Similarly, for ternary relationships

Participation Constraints

- Does every department have a manager?
  - If so, this is a participation constraint: the participation of Departments in Manages is said to be total (vs. partial).
  - Every did value in Departments table must appear in a row of the Manages table (with a non-null ssn value!)
**Weak Entities**

- A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities)
  - Weak entity set must have total participation in this identifying relationship set (since a weak entity cannot exist on its own)

![Entity Relationship Diagram](image)

**ISA ("is a") Hierarchies**

As in C++, or other PLs, attributes are inherited.
If we declare A ISA B, every A entity is also considered to be a B entity.

- Overlap constraints: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (Allowed/disallowed)
- Covering constraints: Is every Employees entity either an Hourly_Emps or a Contract_Emps entity? (Yes/no)
- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.
Aggregation

- Used when we have to model a relationship involving (entity sets and) a relationship set.
  - Aggregation allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.

Aggregation vs. ternary relationship:
- Monitors is a distinct relationship, with a descriptive attribute.
- Also, can say that each sponsorship is monitored by at most one employee.

Conceptual Design Using the ER Model

- Design choices:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary? Aggregation?

- Constraints in the ER Model:
  - A lot of data semantics can (and should) be captured.
  - But some constraints cannot be captured in ER diagrams.
Entity vs. Attribute

- Should *address* be an attribute of *Employees* or an entity (connected to *Employees* by a relationship)?
- Depends upon the use we want to make of address information, and the semantics of the data:
  - If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

Entity vs. Attribute (Contd.)

- *Works_In4* does not allow an employee to work in a department for two or more periods.
- Similar to the problem of wanting to record several addresses for an employee: We want to record several values of the descriptive attributes for each instance of this relationship. Accomplished by introducing new entity set, *Duration*.
**Entity vs. Relationship**

- First ER diagram OK if a manager gets a separate discretionary budget for each dept.
- What if a manager gets a discretionary budget that covers all managed depts?
  - Redundancy: `dbudget` stored for each dept managed by manager.
  - Misleading: Suggests `dbudget` associated with department-mgr combination.

**Binary vs. Ternary Relationships**

- If each policy is owned by just 1 employee, and each dependent is tied to the covering policy, first diagram is inaccurate.
- What are the additional constraints in the 2nd diagram?
Binary vs. Ternary Relationships (Contd.)

- Previous example illustrated a case when two binary relationships were better than one ternary relationship.

- An example in the other direction: a ternary relation $\text{Contracts}$ relates entity sets $\text{Parts}$, $\text{Departments}$ and $\text{Suppliers}$, and has descriptive attribute $\text{qty}$. No combination of binary relationships is an adequate substitute:
  - S “can-supply” P, D “needs” P, and D “deals-with” S does not imply that D has agreed to buy P from S.
  - How do we record $\text{qty}$?

Summary of Conceptual Design

- Conceptual design follows requirements analysis,
  - Yields a high-level description of data to be stored

- ER model popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.

- Basic constructs: entities, relationships, and attributes (of entities and relationships).

- Some additional constructs: weak entities, ISA hierarchies, and aggregation.

- Note: There are many variations on ER model.
• Several kinds of integrity constraints can be expressed in the ER model: key constraints, participation constraints, and overlap/covering constraints for ISA hierarchies. Some foreign key constraints are also implicit in the definition of a relationship set.
  – Some constraints (notably, functional dependencies) cannot be expressed in the ER model.
  – Constraints play an important role in determining the best database design for an enterprise.

Summary of ER (Contd.)

• ER design is subjective. There are often many ways to model a given scenario! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  – Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use ISA hierarchies, and whether or not to use aggregation.
• Ensuring good database design: resulting relational schema should be analyzed and refined further. FD information and normalization techniques are especially useful.
Relational Database: Definitions

- A relational database consists of a set of relations
- A relation consists of two parts:
  - Schema: Students(name:string, age:int, GPA:real)
  - Instance: a table, with rows and columns.

### Students

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>19</td>
<td>3.5</td>
</tr>
<tr>
<td>Ann</td>
<td>20</td>
<td>3.2</td>
</tr>
<tr>
<td>Joe</td>
<td>23</td>
<td>3.3</td>
</tr>
</tbody>
</table>

- Can think of a relation as a set of rows or tuples (i.e., all rows are distinct).

Another Example

- Students(sid: int, name: string, login: string, age:int, GPA:real)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

- Cardinality = 3, degree = 5, all rows must be distinct
- Do all columns in a relation instance have to be distinct?
- How can we create this relation?
SQL - Structured Query Language

- Developed by IBM (System R) in the 1970s
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extensions, current standard)
- DDL – Data Definition Language
  - for creating database types or schemas in terms of a particular data model e.g., create table students(name: string, age: int);
- DML – Data Manipulation Language
  - for managing data (e.g., insert, delete, update tuples)

Creating Relations in SQL

- Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.
- As another example, the Enrolled table holds information about courses that students take.

```sql
CREATE TABLE Students
(sid: CHAR(20),
 name: CHAR(20),
 login: CHAR(10),
 age: INTEGER,
 gpa: REAL)

CREATE TABLE Enrolled
(sid: CHAR(20),
 cid: CHAR(20),
 grade: CHAR(2))
```
Destroying and Altering Relations

**DROP TABLE**  Students

- Destroys the relation Students. The schema information and the tuples are deleted.

**ALTER TABLE**  Students
  ADD COLUMN  firstYear: integer

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Adding and Deleting Tuples

- Can insert a single tuple using:

  **INSERT INTO**  Students (sid, name, login, age, gpa)
  **VALUES**  (53688, 'Smith', 'smith@ee', 18, 3.2)

- Can delete all tuples satisfying some condition (e.g., name = Smith):

  **DELETE**
  **FROM**  Students S
  **WHERE**  S.name = 'Smith'
Integrity Constraints (ICs)

- IC: condition that must be true for any instance of the database; e.g., domain constraints.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A legal instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Primary Key Constraints

- A set of fields is a key for a relation if no two distinct tuples can have same values in all key fields, and
- Minimal key – smallest possible set of fields that is the key (usually we use the word key to mean minimal key)
- Otherwise, it is called a superkey
- If there’s >1 key for a relation, one of the keys is chosen (by DBA) to be the primary key. The others are called candidate keys
- sid is a key for Students. (What about name?) The set {sid, gpa} is a superkey.
- Usually the key is underlined.
  - Student(sid, name, login, age, GPA)
  - Enrolled(sid, cid, grade)
Foreign Keys, Referential Integrity

- **Foreign key**: Set of fields in one relation that is used to `refer` to a tuple in another relation.
  - Like a `logical pointer`
  - Must correspond to the key of the second relation. Why?
- E.g. Below, *sid* is a foreign key referring to Students:
  - Enrolled(sid: string, cid: string, grade: string)
  - If all foreign key constraints are enforced, referential integrity is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?
  - Links in HTML!

Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

```sql
CREATE TABLE Enrolled
(sid CHAR(20), cid CHAR(20), grade CHAR(2),
 PRIMARY KEY (sid,cid),
 FOREIGN KEY (sid) REFERENCES Students)
```

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666 Carnatic101 C</td>
<td>53666 Jones jones@cs 18 3.4</td>
</tr>
<tr>
<td>53666 Reggae203 B</td>
<td>53688 Smith smith@eeecs 18 3.2</td>
</tr>
<tr>
<td>53650 Topology112 A</td>
<td>53650 Smith smith@math 19 3.8</td>
</tr>
<tr>
<td>53666 History105 B</td>
<td></td>
</tr>
</tbody>
</table>
Enforcing Referential Integrity

• There are many ways to enforce referential integrity
• Consider Students and Enrolled; sid in Enrolled is a foreign key that references Students.
• What should be done if an Enrolled tuple with a non-existent student id is inserted? (Reject it!)
• What should be done if a Students tuple is deleted?
  1. Also delete all Enrolled tuples that refer to it.
  2. Disallow deletion of a Students tuple that is referred to.
  3. Set sid in Enrolled tuples that refer to it to a default sid.
  4. (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value null, denoting 'unknown' or 'inapplicable'.)
• Similar if primary key of Students tuple is updated.

Referential Integrity in SQL

• SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  – Default is NO ACTION (delete/update is rejected)
  – CASCADE (also delete all tuples that refer to deleted tuple)
  – SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid)
   REFERENCES Students
   ON DELETE CASCADE
   ON UPDATE SET DEFAULT)

Winter 2003
Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know \textit{name} is not a key, but the assertion that \textit{sid} is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.

ICs (cont’d)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>John</td>
<td>john@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

- Given that \textit{sid} is a key for \textit{Students} relation, we know that
  - for any legal instance of \textit{students} relation, no two distinct student tuples share the same \textit{sid} value
  - Alternatively, if two student tuples have the same \textit{sid} value, we know they must be the same tuple
- Is \textit{name} a key for \textit{Students} relation?
Logical DB Design: ER to Relational

- Entity sets to tables (obviously!)

CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn))

Relationship Sets to Tables

- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
  - This set of attributes forms a superkey for the relation.
  - All descriptive attributes.

CREATE TABLE Works_In(
 ssn CHAR(11),
 did INTEGER,
 since DATE,
 PRIMARY KEY (ssn, did),
 FOREIGN KEY (ssn) REFERENCES Employees,
 FOREIGN KEY (did) REFERENCES Departments)
Translating ER Diagrams with Key Constraints

- Each dept has at most one manager, according to the key constraint on Manages.

CREATE TABLE Manages(
  did INTEGER,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments
)

CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees
)

Map relationship to a table:
- Note that did is the key now!
- Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.

Winter 2003
Review: Participation Constraints

- Does every department have a manager?
  - If so, this is a participation constraint: the participation of Departments in Manages is said to be total (vs. partial).
- Every did value in Departments table must appear in a row of the Manages table (with a non-null ssn value!)

Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```sql
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11) NOT NULL,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  ON DELETE NO ACTION)
```

An Employees tuple cannot be deleted while it is referenced by some Dept_Mgr tuple
Review: Weak Entities

- The key of a weak entity consists of its partial key as well as that of its owner entity.
- Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
- Weak entity set must have total participation in this identifying relationship set.

![Diagram showing employees, policy, and dependents entities with relationships]

Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
- When the owner entity is deleted, all owned weak entities must also be deleted.

```sql
CREATE TABLE Dep_Policy (  
  pname CHAR(20),  
  age INTEGER,  
  cost REAL,  
  ssn CHAR(11) NOT NULL,  
  PRIMARY KEY (pname, ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees,  
  ON DELETE CASCADE)
```

Information about an employee's policy and dependent is deleted if the corresponding employee is deleted.
Others (see textbook)

- Translating ISA hierarchies to relations
- Translating Aggregation to relations

Views

- A view is a relation, defined in terms of other relations, but its tuples are not stored.

```
CREATE VIEW YoungActiveStudents (name, grade) AS
    SELECT S.name, E.grade
    FROM   Students S, Enrolled E
    WHERE  S.sid = E.sid and S.age<21
```

- Views can be dropped using the DROP VIEW command. Similarly, tables can be dropped with DROP TABLE command
  - What happens if DROP TABLE is performed and there is a view defined on the table?
  - DROP TABLE command has options to let the user specify this.
Views and Security

- Views can be used to present only the desired information, while hiding details in underlying relation(s).
  - Given YoungStudents, but not Students or Enrolled, we can find students who have are enrolled, but not the cid's of the courses they are enrolled in.
- Define views and allow only a group of users to access to it

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - In addition, we always have domain constraints.
- Powerful and natural query language exist.
- Rules to translate ER to relational model
Recommended Reading

• Chapter 2
  – Section 2.1 – 2.5
• Chapter 3
  – Section 3.1 – 3.5, 3.6.1, 3.7