Conceptual Design Using the Entity-Relationship (ER) Model

Module 5, Lectures 1 and 2
Overview of Database Design

- **Conceptual design:** (ER Model is used at this stage.)
  - 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 hold?
  - A database `schema` in the ER Model can be represented pictorially (*ER diagrams*).
  - Can map an ER diagram into a relational schema.

- **Schema Refinement:** (Normalization) Check relational schema for redundancies and related anomalies.

- **Physical Database Design and Tuning:** Consider typical workloads and further refine the database design.
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.
  - Can map entity set to a relation easily.

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

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
</tr>
</tbody>
</table>
**ER Model Basics (Contd.)**

- **Relationship**: Association among 2 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 $e_1 \in E_1, \ldots, e_n \in E_n$
  - Same entity set could participate in different relationship sets, or in different “roles” in same set.
ER Model Basics (Contd.)

- Relationship sets can also have descriptive attributes (e.g., the since attribute of Works_In).
- 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 superkey for the relation.
  - All descriptive attributes.

CREATE TABLE Works_In(
  ssn CHAR(1),
  did INTEGER,
  since DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
    REFERENCES Employees,
  FOREIGN KEY (did)
    REFERENCES Departments)

<table>
<thead>
<tr>
<th>ssn</th>
<th>did</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>51</td>
<td>1/1/91</td>
</tr>
<tr>
<td>123-22-3666</td>
<td>56</td>
<td>3/3/93</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>51</td>
<td>2/2/92</td>
</tr>
</tbody>
</table>
Key Constraints

- Consider Works_In: An employee can work in many departments; a dept can have many employees.
- In contrast, each dept has at most one manager, according to the key constraint on Manages.

Translation to relational model?
Translating ER Diagrams with Key Constraints

- 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.

CREATE TABLE Manages(
  ssn CHAR(11),
  did INTEGER,
  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)
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!)
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)
```
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 (1 owner, many weak entities).
- Weak entity set must have total participation in this identifying relationship set.
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)
```
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. (Query answers should reflect this: unlike C++!)

- **Overlap constraints**: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? *(Allowed/disallowed)*

- **Covering constraints**: Does every Employees entity also have to be 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.
Translating ISA Hierarchies to Relations

- **General approach:**
  - 3 relations: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, *ssn*); must delete Hourly_Emps tuple if referenced Employees tuple is deleted).
    - Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes.
  - Alternative: Just Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: *ssn*, *name*, *lot*, *hourly_wages*, *hours_worked*.
    - Each employee must be in one of these two subclasses.
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.
  - Monitors mapped to table like any other relationship set.

☞ 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 modelled as an entity or an attribute?
  - Should a concept be modelled 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.

- **Need for further refining the schema:**
  - Relational schema obtained from ER diagram is a good first step. But ER design subjective & can’t express certain constraints; *so this relational schema may need refinement.*
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 modelled as an entity (since attribute values are atomic).
Entity vs. Attribute (Contd.)

- Works_In2 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.
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 of \textit{dbudget}, which is stored for each dept managed by the manager.
  - Misleading: suggests \textit{dbudget} tied to managed dept.
If each policy is owned by just 1 employee:
- Key constraint on Policies would mean policy can only cover 1 dependent!

What are the additional constraints in the 2nd diagram?
**Binary vs. Ternary Relationships (Contd.)**

- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to \texttt{NOT NULL} constraints.
- What if Policies is a weak entity set?

```sql
CREATE TABLE Policies (  
  policyid INTEGER,  
  cost REAL,  
  ssn CHAR(11) NOT NULL,  
  PRIMARY KEY (policyid).  
  FOREIGN KEY (ssn) REFERENCES Employees,  
  ON DELETE CASCADE)

CREATE TABLE Dependents (  
  pname CHAR(20),  
  age INTEGER,  
  policyid INTEGER,  
  PRIMARY KEY (pname, policyid).  
  FOREIGN KEY (policyid) REFERENCES Policies,  
  ON DELETE CASCADE)
```
Previous example illustrated a case when 2 binary relationships were better than a ternary relationship.

An example in the other direction: a ternary relation Contracts relates entity sets Parts, Departments and Suppliers, and has descriptive attribute 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 qty?
Constraints Beyond the ER Model

- **Functional dependencies:**
  - *e.g.*, A dept can’t order two distinct parts from the same supplier.
    - Can’t express this wrt ternary Contracts relationship.
  - Normalization refines ER design by considering FDs.

- **Inclusion dependencies:**
  - Special case: Foreign keys (ER model can express these).
  - *e.g.*, At least 1 person must report to each manager. (Set of ssn values in Manages must be subset of supervisor_ssn values in Reports_To.) Foreign key? Expressible in ER model?

- **General constraints:**
  - *e.g.*, Manager’s discretionary budget less than 10% of the combined budget of all departments he or she manages.
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 of these constraints can be expressed in SQL only if we use general CHECK constraints or assertions.
- 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.