Relational Database Design

Translation of ER-diagram into Relational Schema

Dr. Sunnie S. Chung
CIS430/530
Learning Objectives

- Define each of the following database terms
  - Relation
  - Primary key
  - Foreign key
  - Referential integrity
  - Field
  - Data type
  - Null value

- Discuss the role of designing databases in the analysis and design of an information system
- Learn how to transform an entity-relationship (ER) Diagram into an equivalent set of well-structured relations
ER/EER to database schema
Process of Database Design

- Steps in translation:
  - Entity sets to tables
  - Relationships to tables
  - Constraints
  - Weak entity sets

- Logical Design
  - Based upon the conceptual data model
  - Four key steps
    1. Develop a logical data model for each known user interface for the application using normalization principles.
    2. Combine normalized data requirements from all user interfaces into one consolidated logical database model
    3. Translate the conceptual E-R data model for the application into normalized data requirements
    4. Compare the consolidated logical database design with the translated E-R model and produce one final logical database model for the application
Entity Sets to Tables

- Each attribute of the E. S. becomes an attribute of the table

Relations:
CUSTOMER(Customer_ID, Name, Address)
PRODUCT(Product_ID, Description)
ORDER(Order_Number, Customer_ID, Order_Date)
LINE ITEM(Order_Number, Product_ID, Order_Quantity)
INVOICE(Invoice_Number, Order_Number)
SHIPMENT(Invoice_Number, Product_ID, Ship_Quantity)
Relational Database Model

• Data represented as a set of related tables or relations

• Relation
  - A named, two-dimensional table of data. Each relation consists of a set of named columns and an arbitrary number of unnamed rows

• Properties
  - Entries in cells are simple
  - Entries in columns are from the same set of values
  - Each row is unique
  - The sequence of columns can be interchanged without changing the meaning or use of the relation
  - The rows may be interchanged or stored in any sequence
Relational Database Model

- **Well-Structured Relation**
  - A relation that contains a minimum amount of redundancy and allows users to insert, modify and delete the rows without errors or inconsistencies.

<table>
<thead>
<tr>
<th>Emp_ID</th>
<th>Name</th>
<th>Dept</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Margaret Simpson</td>
<td>Marketing</td>
<td>42,000</td>
</tr>
<tr>
<td>140</td>
<td>Allen Beeton</td>
<td>Accounting</td>
<td>39,000</td>
</tr>
<tr>
<td>110</td>
<td>Chris Lucero</td>
<td>Info Systems</td>
<td>41,500</td>
</tr>
<tr>
<td>190</td>
<td>Lorenzo Davis</td>
<td>Finance</td>
<td>38,000</td>
</tr>
<tr>
<td>150</td>
<td>Susan Martin</td>
<td>Marketing</td>
<td>38,500</td>
</tr>
</tbody>
</table>
Transforming E-R Diagrams into Relations

- It is useful to transform the conceptual data model into a set of normalized relations

- Steps
  1. Represent entities
  2. Represent relationships
  3. Normalize the relations
  4. Merge the relations
Refining the ER Design for the COMPANY Database

- Change attributes that represent relationships into relationship types
- Determine cardinality ratio and participation constraint of each relationship type
ER Diagrams, Naming Conventions, and Design Issues

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Figure 7.14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Entity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Weak Entity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Relationship</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Identifying Relationship</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Key Attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multivalued Attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Composite Attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Derived Attribute</td>
<td></td>
</tr>
</tbody>
</table>

- Total Participation of $E_2$ in $R$
- Cardinality Ratio: $N$ for $E_1$-$E_2$ in $R$
- Structural Constraint (min, max) on Participation of $E$ in $R$
Design Choices for ER Conceptual Design

- Model concept first as an attribute
  - Refined into a relationship if attribute is a reference to another entity type

- Attribute that exists in several entity types may be elevated to an independent entity type
  - Can also be applied in the inverse
Alternative Notations for ER Diagrams

- Specify structural constraints on Relationships
  - Replaces Cardinality ratio (1:1, 1:N, M:N) and single/double line notation for Participation constraints
  - Associate a pair of integer numbers (min, max) with each participation of an entity type $E$ in a relationship type $R$, where $0 \leq \text{min} \leq \text{max}$ and $\text{max} \geq 1$
Cardinality Ratio
(1:1, 1:N, M:N)

- 1:N: Each dept has at most one manager on Manages.

Translation to relational model?
Figure 7.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.
Transforming E-R Diagrams into Relations

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

- The primary key must satisfy the following two conditions
  a. The value of the key must uniquely identify every row in the relation
  b. The key should be nonredundant
CUSTOMER

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>Name</th>
<th>Address</th>
<th>City_State_ZIP</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1273</td>
<td>Contemporary Designs</td>
<td>123 Oak St.</td>
<td>Austin, TX 28384</td>
<td>5%</td>
</tr>
<tr>
<td>6390</td>
<td>Casual Corner</td>
<td>18 Hoosier Dr.</td>
<td>Bloomington, IN 45821</td>
<td>3%</td>
</tr>
</tbody>
</table>
Transforming E-R Diagrams into Relations

Represent Relationships

- Binary 1:N Relationships
  - Add the Primary key attribute (or attributes) of the entity on the one side of the relationship as a Foreign key in the relation on the other (N) side
  - The one side migrates to the many side
Transforming Binary 1:N Relationships into Relations

• Relationship:
  CUSTOMER Places ORDER(s)

• ORDER Table BEFORE Relationship:
  (Order_Number, Order_Date, Promised_Date)

• ORDER Table AFTER Relationship:
  (Order_Number, Order_Date, Promised_Date, Customer_ID)

CREATE TABLE ORDER(
  Order_Number CHAR(1),
  Order_Date DATE,
  Promised_Date DATE,
  Customer_ID CHAR(1),
  PRIMARY KEY
  (Order_Number),
  FOREIGN KEY (Customer_ID)
  REFERENCES CUSTOMER(Customer_ID));
Transforming E-R Diagrams into Relations

- Binary or Unary 1:1
  - Three possible options
    a. Add the primary key of A as a foreign key of B
    b. Add the primary key of B as a foreign key of A
    c. Both
Transforming E-R Diagrams into Relations

Represent Relationships

- Binary and higher $M:N$ relationships
  - Create another relation and include primary keys of all relations as primary key of new relation
Transforming Binary M:N Relationships into Relations

- Relationship Requests:
  Order Requests Products

1. Create Table ORDERLINE for Relationship Requests
2. Add PK of each side of Tables (Order_Number, Product_ID) as Foreign Keys
3. Make composite of both attributes as Primary Key of the Table ORDERLINE:
   CREATE TABLE ORDERLINE (Order_Number CHAR(10),
   Product_ID CHAR(10),
   Quantity_Ordered Integer,
   PRIMARY KEY (Order_Number, Product_ID),
   FOREIGN KEY (Order_Number) REFERENCES ORDER(Order_Number),
   FOREIGN KEY (Product_ID) REFERENCES PRODUCT(Product_ID));

   ORDERLINE(Order_Number, Product_ID, Quantity_Ordered)
**Constraints on Binary Relationship Types**

- **Cardinality ratio** for a binary relationship
  - Specifies maximum number of relationship instances that entity can participate in

- **Participation Constraint**
  - Specifies whether existence of entity depends on its being related to another entity
  - Types: *total* and *partial*
Attributes of Relationship Types

- Attributes of 1:1 or 1:N relationship types can be migrated to one entity type
- For a 1:N relationship type
  - Relationship attribute can be migrated only to entity type on N-side of relationship
- For M:N relationship types
  - Some attributes may be determined by combination of participating entities
  - Must be specified as relationship attributes
Weak Entity Types

- Do not have key attributes of their own
  - Identified by being related to specific entities from another entity type
- Identifying relationship
  - Relates a weak entity type to its owner
- Always has a total participation constraint
Transforming E-R Diagrams into Relations

- **Unary 1:N Relationships**
  - Relationship between instances of a single entity type
  - Utilize a recursive foreign key
    - A foreign key in a relation that references the primary key values of that same relation

- **Unary M:N Relationships**
  - Create a separate relation
  - Primary key of new relation is a composite of two attributes that both take their values from the same primary key
**Figure 9.13b** Two Unary Relations — Bill-of-Materials Structure (M:N)

**Figure 9.13a** Two Unary Relations — EMPLOYEE with Manages Relationship (1:N)
Transforming Unary 1:N Relationships into Relations

- Relationship:
  EMPLOYEE (as Manager) Manages EMPLOYEE

- EMPLOYEE Table
  BEFORE Relationship:
  (Emp_ID, Name, Birthday)

- EMPLOYEE Table AFTER Relationship:
  (Emp_ID, Name, Birthday, Mgr_ID)

CREATE TABLE EMPLOYEE
  (Emp_ID CHAR(1),
   Name VARCHAR(30),
   Birthday DATE,
   Mgr_ID CHAR(1),
   PRIMARY KEY (Emp_ID),
   FOREIGN KEY (Mgr_ID)
   REFERENCES EMPLOYEE (Emp_ID));
Transforming Unary M:N Relationships into Relations

- Relationship Contains:
  ITEM Contains ITEM

1. Create Table for Relationship CONTAINS

2. Add PK of each side of Tables (Containing_Item_Num, Contained_Item_Num) as Foreign Keys

3. Make composite of both attributes as Primary Key of the Table CONTAINS:

CREATE TABLE CONTAINS (Containing_Item_Num CHAR(10),
                         Contained_Item_Num CHAR(10),
                         Quantity Integer,
                         PRIMARY KEY (Containing_Item_Num,
                                       Contained_Item_Num),
                         FOREIGN KEY (Containing_Item_Num)
                         REFERENCES ITEM(Item_Number),
                         FOREIGN KEY (Contained_Item_Num)
                         REFERENCES ITEM(Item_Number));

CONTAINS (Containing_Item_Num, Contained_Item_Num, Quantity)
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 (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.
**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 (1 owner, many weak entities).
  - **Weak entity** set must have **total participation** in this identifying relationship set.
Translating weak entities

- Weak entity set and identifying relationship set are translated into a single table --- it has a (1,1) cardinality constraint.

```sql
CREATE TABLE Dep_Policy (  
dname CHAR(20),  
age INTEGER,  
cost REAL,  
parent_ssn CHAR(9) NOT NULL,  
PRIMARY KEY (dname, parent_ssn),  
FOREIGN KEY (parent_ssn) REFERENCES Employees,  
    ON DELETE CASCADE)
```

- When an owner entity is deleted all owned entity should also be deleted.
Primary Key Constraints

- A set of fields is a **key** for a relation if:
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key. - **Key is minimal.**
     - However, 2 does not hold (so false) for superkey - which is not minimal.
     - If there's more than one keys for a relation, one of the keys is chosen (by DBA) to be the **primary key**.
- E.g., customer_id is a key for Customer. (What about name?) The set \{customer_id, name\} could be a superkey.

**Primary key can not have null value**
**Domain Constraint**

- The value of each Attribute $A$ with Domain Type $D(A_i)$ must be an atomic value from the domain type $D(A_i)$. 
Definitions of Keys and Attributes Participating in Keys

• A **superkey** of a relation schema \( R = \{A_1, A_2, \ldots, A_n\} \) is a set of attributes \( S \), *subset-of* \( R \), with the property that no two tuples \( t_1 \) and \( t_2 \) in any legal relation state \( r \) of \( R \) will have \( t_1[S] = t_2[S] \).

That is, for any given two tuples \( t_1, t_2 \) in data (extensions) of Relation schema \( R \), \( t_1[S] \) is not identical to \( t_2[S] \).

• A **key** \( K \) is a superkey with the *additional property* that removal of any attribute from \( K \) will cause \( K \) not to be a superkey any more; **Key is minimal**.
Definitions of Keys and Attributes Participating in Keys

• If a relation schema has more than one key, each is called a candidate key.

• One of the candidate keys is arbitrarily designated to be the primary key, and the others are called secondary keys.

• A Prime attribute must be a member of any (candidate) key

• A Nonprime attribute is not a prime attribute—that is, it is not a member of any (candidate) key.
Foreign Keys, Referential Integrity

- **Foreign key**: Set of fields in one relation that is used to `refer` to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer`.

- E.g. `customer_id` in Order is a foreign key referring to `Customer`:
  
  \[
  \text{Order (order\_number, order\_date, promised\_date, customer\_id)}
  \]
Foreign Keys, Referential Integrity

- If all foreign key constraints are enforced, referential integrity is achieved; all foreign key values should refer to existing values, i.e., no dangling references.

- Can you name a data model w/o referential integrity?
  - Links in HTML!
Enforcing Referential Integrity

- Consider **Students** *(sid, name, gpa)* and **Enrolled** *(rid, semester, sid)*;
- *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?
  - Also delete all Enrolled tuples that refer to it.
  - Disallow deletion of a Students tuple that is referred to.
  - Set sid in Enrolled tuples that refer to it to a *default sid*.
  - (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.
Logical DB Design: ER to Relational

• Entity sets to tables.

CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))
Review: Key Constraints

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

Translation to relational model?
Transforming 1:N, M:N Relationships with Key Constraints

ER Diagram:
Translating ER Diagrams with Key Constraints

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

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

```sql
CREATE TABLE `Dept_Mgr`(
    `did` INTEGER,
    `dname` CHAR(20),
    `budget` REAL,
    `ssn` CHAR(11),
    `since` DATE,
    PRIMARY KEY (`did`),
    FOREIGN KEY (`ssn`) REFERENCES Employees
)
```
Transforming Realationship to Tables

Example E-R diagram:
Relationship Sets to Tables

- In translating a relationship Works_In (M-N) 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.

```sql
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
)
```
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!)

![Database Diagram]

- **Employees**
  - ssn
  - name
  - lot

- **Manages**
  - since
  - did
  - dname
  - budget

- **Departments**
  - since

- **Works_In**
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)
```
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 entities

- Weak entity set and identifying relationship set are translated into a single table --- it has a (1,1) cardinality constraint.

```
CREATE TABLE Dep_Policy (
    dname CHAR(20),
    age INTEGER,
    cost REAL,
    parent_ssn CHAR(9) NOT NULL,
    PRIMARY KEY (dname, parent_ssn),
    FOREIGN KEY (parent_ssn) REFERENCES Employees,
    ON DELETE CASCADE)
```

- When an owner entity is deleted all owned entity should also be deleted.
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 (  
dname CHAR(20),  
age INTEGER,  
cost REAL,  
ssn CHAR(11) NOT NULL,  
PRIMARY KEY (pname, ssn),  
FOREIGN KEY (ssn) REFERENCES Employees,  
ON DELETE CASCADE)
```
Review: Binary vs. Ternary Relationships

- 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?

Bad design

Better design
Binary vs. Ternary Relationships

(Contd.)

• The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.

• Participation constraints lead to NOT NULL constraints.

• What if Policies is a weak entity set?

PK of Policies: (policyid, ssn)

PK of Dependents: (dname, policyid, ssn)

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 (  
dname CHAR(20),  
age INTEGER,  
policyid INTEGER,  
PRIMARY KEY (dname, policyid).  
FOREIGN KEY (policyid) REFERENCES Policies  
ON DELETE CASCADE);
Translating Class Hierarchies

- Example:
  - Employees could have two subclasses:
    1. Hourly Employees (Hourly_Emps)
       - Characterized by: hourly wages and hours worked
    2. Contract Employees (Contract_Emps)
       1. Characterized by: contract id.
Translating Class Hierarchies

- Two approaches
  - Three tables: 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*);
    - We 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.
An Example

CREATE TABLE Student (  
    ID NUMBER,  
    Fname VARCHAR2(20),  
    Lname VARCHAR2(20),  
);
Constraints in Create Table

- Adding constraints to a table enables the database system to enforce data integrity.
- Different types of constraints:
  * Not Null
  * Default Values
  * Unique
  * Primary Key
  * Foreign Key
  * Check Condition
CREATE TABLE Student ( 
    ID NUMBER,
    Fname VARCHAR2(20)  NOT NULL,
    Lname VARCHAR2(20)  NOT NULL,
);


**Primary Key Constraint**

Primary Key implies: * **NOT NULL** * **UNIQUE.**

There can only be one primary key.

```
CREATE TABLE Student (
  ID NUMBER PRIMARY KEY,
  Fname VARCHAR2(20) NOT NULL,
  Lname VARCHAR2(20) NOT NULL,
);  
```
Primary Key Constraint
(Syntax 2)

CREATE TABLE Students (  
    ID       NUMBER,  
    Fname    VARCHAR2(20) NOT NULL,  
    Lname    VARCHAR2(20) NOT NULL,  
    PRIMARY KEY(ID)  
);  

Needed when the primary key is made up of two or more attributes (fields)
FOREIGN KEY Constraint

NOTE: ID must be unique (or primary key) in Students table

CREATE TABLE Studies(
    Course       NUMBER,
    Student      NUMBER,
    FOREIGN KEY (Student) REFERENCES Students(ID)
);