CS 4604: Introduction to Database Management Systems

Entity/Relationship Models II

Virginia Tech CS 4604 Sprint 2021
Instructor: Yinlin Chen
Today’s Topics

• Design ER models
• ER to Relational
Weak Entities

- A **strong entity** set is an entity set that has a **key** attribute type.
- A **weak entity** set is an entity set whose key contains attributes from one or more other entity sets.
  - 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.
- Weak entities have a “partial key” (dashed underline)
Weak Entity Set

- **Weak entity** set is always existentially dependent from owner entity set (not vice versa!)
- Representation: a rectangle with a double border in the E/R diagram
- **Supporting relationship**: diamond with a double border

```
RNR
HNR
Beds
```

```
Room

BELONGS TO

Hotel
HNR
Hname
```

**Many – to - One**
Finding the Key for a Weak Entity Set

- Room is a weak entity set if its key consists of:
  - Zero or more of its own attributes
  - Key attributes from supporting relationships for Room

```
Room 1..1 Hotel
     \    /
      \  /
       \/
       Room
```

Many – to - One
Supporting relationship

• A relationship R from a weak entity set Room to Hotel is supporting if
  – R is a binary, many-to-one relationship from Room to Hotel
  – R has referential integrity from Hotel to Room

• How does Hotel help Room?
  – Hotel supplies its key attributes to define Room's key
  – If Hotel is itself a weak entity set, some of its key attributes come from entity sets to which Hotel is connected by supporting relationships

Many – to - One
Ternary Relationship Types Example

- Assume that we have a situation where suppliers can supply products for projects.
- A supplier can supply a particular product for multiple projects.
- A product for a particular project can be supplied by multiple suppliers.
- A project can have a particular supplier supply multiple products.
- The model must also include the quantity and due date for supplying a particular product to a particular project by a particular supplier.
Binary Relationship Types

SUPPLIER

SUPPLIES

PROJECT

CAN SUPPLY

PRODUCT

USES

PRODNR

SUPNR

PNR
Binary Relationship Types

• Say we have two projects: project 1 uses a pencil and a pen, and project 2 uses a pen
• Supplier Peters supplies the pencil for project 1 and the pen for project 2
• Supplier Johnson supplies the pen for project 1
• From the binary relationship types, it is not clear who supplies the pen for project 1!

<table>
<thead>
<tr>
<th>SUPPLIES</th>
<th>USES</th>
<th>CAN SUPPLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplier</td>
<td>Project</td>
<td>Product</td>
</tr>
<tr>
<td>Peters</td>
<td>Project 1</td>
<td>Pencil</td>
</tr>
<tr>
<td>Peters</td>
<td>Project 2</td>
<td>Pen</td>
</tr>
<tr>
<td>Johnson</td>
<td>Project 1</td>
<td>Pen</td>
</tr>
<tr>
<td>Peters</td>
<td></td>
<td>Pencil</td>
</tr>
<tr>
<td>Peters</td>
<td></td>
<td>Pen</td>
</tr>
<tr>
<td>Johnson</td>
<td></td>
<td>Pen</td>
</tr>
</tbody>
</table>
Ternary Relationship Types

SUPPLIER

SUPPLY

PROJECT

PRODUCT

SUPPLY

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Product</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peters</td>
<td>Pencil</td>
<td>Project 1</td>
</tr>
<tr>
<td>Peters</td>
<td>Pen</td>
<td>Project 2</td>
</tr>
<tr>
<td>Johnson</td>
<td>Pen</td>
<td>Project 1</td>
</tr>
</tbody>
</table>
Aggregation

- Allows relationships to have relationships
- Entity types that are related by a particular relationship type can be combined or aggregated into a higher-level aggregate entity type
- Aggregation is especially useful when the aggregate entity type has its own attribute types and/or relationship types
Limitations of the ER model

• ER model presents a temporary snapshot and cannot model temporal constraints
  – Examples: a project needs to be assigned to a department after one month, a purchase order must be assigned to a supplier after two weeks, etc.

• ER model cannot guarantee the consistency across multiple relationship types
  – Examples: an employee should work in the department that he/she manages, suppliers can only be assigned to purchase orders for products they can supply
Limitations of the ER model

• **Domains are not included** in the ER model
  – Examples: hours should be positive; prodtype must be red, white or sparkling, supstatus is an integer between 0 and 100

• **Functions are not included** in the ER model
  – Examples: calculate average number of projects an employee works on; determine which supplier charges the maximum price for a product
Examples of the ER Diagram (Model)
Examples of the ER Diagram (Model)
Conceptual Design Using the ER Model

• ER modeling can get tricky!
• Design choices:
  – Entity or attribute?
  – Entity or relationship?
  – Relationships: Binary or ternary? Aggregation?
• ER Model goals and limitations:
  – Lots of semantics can (and should) be captured.
  – Some constraints cannot be captured in ER.
    • We will refine things in our logical (relational) design
Principle 1: Avoiding Redundancy

- Redundancy occurs when we say the same thing in two different ways
- Redundancy wastes space and causes inconsistency
  - The two instances of the same fact may become inconsistent if we change one & forget to change the other
Example: Avoiding Redundancy
Principle 2: Entity vs. Attribute

• “Address”:
  – attribute of Employees?
  – Entity of its own?

• It depends! Semantics and usage.
  – Several addresses per employee?
    • must be an entity
    • atomic attribute types (no set-valued attributes!)
  – Care about structure? (city, street, etc.)
    • must be an entity! (or at least multiple attributes)
    • atomic attribute types (no tuple-valued attributes!)
Entity Sets Versus Attributes

• Rule: An entity set should satisfy at least one of the following conditions:
  – It is more than the name of something;
    • i.e., it has at least one non-key attribute.
  Or
  – It is the “many” in a many-one or many-many relationship.
Example: Entity vs. Attribute

- If we had no manufacturer address information
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 we think about applications.
  – Note: There are many variations on ER model
    • Both graphically and conceptually

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

• Some additional constructs: **weak entities** and aggregation.
Summary of ER (Cont.)

• Basic integrity constraints
  – key constraints
  – participation constraints

• Some foreign key constraints are also implicit in the definition of a relationship set.

• Many other constraints (notably, functional dependencies) cannot be expressed.

• Constraints play an important role in determining the best database design for an enterprise.
Summary of ER (Cont.)

• ER design is **subjective**. Many ways to model a given scenario!

• Analyzing alternatives can be tricky! Common choices include:
  – Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use aggregation

• For good DB design: resulting relational schema should be **analyzed** and **refined** further.
  – Functional Dependency information + normalization coming in subsequent lecture.
Guidelines

• Be faithful to the specification of the application
• Avoid redundancy
• Keep the entities and relationship simple
• Select the right relationships
• Select the right type of element
Be Faithful to the Specification

• Do not use meaningless or unnecessary attributes

• Define the multiplicity of a relationship appropriately
  – What is the multiplicity of the relationship Take between Students and Courses?
  – What is the multiplicity of the relationship Teach between Professors and Courses?
Select the Right Relationships

- Do not add unnecessary relationships
- It may be possible to deduce one relationship from another
- Do we need the relationship Instruct between Professors and Students?
Select the Right Relationships

• Do we need the relationships Take and Teach?
Select the right kind of element

- Attribute or Entity or Relationship
- Can we make Professor an attribute of Courses and remove the relationship Teach?
Select the right kind of element

- Attribute or Entity or Relationship
- What about this?
Select the right kind of element

- Attribute or Entity or Relationship
- What about this?

Research signifies a research project the student is working on with a professor
Converting an Entity Set into an Attribute

• If an entity set $E$ satisfies the following properties:
  – All relationships involving $E$ have arrows entering $E$
  – The attributes of $E$ collectively identify an entity (i.e., no attribute depends on another)
  – No relationship involves $E$ more than once

• Then we can replace $E$ as follows:
  – If there is a many-one relationship $R$ from an entity set $F$ to $E$, remove $R$ and make the attributes of $E$ be attributes of $F$
  – If there is a multiway relationship $R$ with an arrow to $E$, make $E$’s attributes be new attributes of $R$ and remove the arrow from $R$ to $E$
Recap: Types of Constraints

- **Keys** are attributes or sets of attributes that uniquely identify an entity within its entity set
- **Single-value constraints** require that a value be unique in certain contexts
- **Referential integrity constraints** require that a value referred to actually exists in the database
- **Degree constraints** specify what set of values an attribute can take
- **General constraints** are arbitrary constraints that should hold in the database
- **Constraints are part of the schema of a database**
Single Value Constraints

- There is at most one value in a given context
- Each attribute of an entity set has a single value
  - If the value is missing, we can invent a “null” value
  - E/R models cannot represent the requirement that an attribute cannot have a null value
- A many-one relationship implies a single value constraint
Referential Integrity Constraint

• **Asserts** that exactly one value exists in a given context
  – Usually used in the context of relationships
• Example: Many-one Advises relationship between Students and Professors
  – Many-one requirement says that no student may have more than one advising professor
  – Referential integrity constraint says that each student must have exactly one advising professor and that professor **must be present** in the database
• If R is a (many-to-one or one-to-one) relationship from E to F, we use a rounded arrowhead pointing to F to indicate that we require that the entity in F related by R to an entity in E **must exist**
Example: Referential Integrity Constraint

- Each department has at most one chairperson who is its head (there are times when a department may not have a chairperson)
- Each chairperson can be the head of at most one department and this department must exist in the database
Enforcing Referential Integrity Constraints

- We forbid the deletion of a referenced entity (e.g., a professor) until the professor advises no students.
- We require that if we delete a referenced entity, we delete all entities that reference it.
- When we insert a (student, professor) pair into the Advises relationship, the professor must exist in the Professors entity set.
Degree Constraints

• Indicates limits on the # of entities that can be connected
• For example,

```
Stars <=10 Stars_in Movies
```

• Limits number of stars in each move to <=10
Degree Constraints

• Indicates limits on the # of entities that can be connected
Steps in Database Design

• Requirements Analysis
  – user needs; what must database do?
• Conceptual Design
  – high level description (often done w/ER model)
  – Object-Relational Mappings (ORMs: Hibernate, Rails, Django, etc.) encourage you to program here
• Logical Design
  – translate ER into DBMS data model
  – ORMs often require you to help here too
• Schema Refinement
  – consistency, normalization
• Physical Design - indexes, disk layout
• Security Design - who accesses what, and how
Recap: Relational Model

• Built around a single concept for modelling data: the relation or table
• Supports high-level programming language (SQL)
• Has an elegant mathematical design theory
• Most current DBMS are relational
Recap: The Relation

- A relation is a two-dimensional table:
  - Relation ⇔ Table
  - Attribute ⇔ Column name
  - Tuple ⇔ Row (not the header row)
  - Database ⇔ Collection of relations

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermione Grainger</td>
<td>Potions</td>
<td>A</td>
</tr>
<tr>
<td>Draco Malfoy</td>
<td>Potions</td>
<td>B</td>
</tr>
<tr>
<td>Harry Potter</td>
<td>Potions</td>
<td>A</td>
</tr>
<tr>
<td>Ron Weasley</td>
<td>Potions</td>
<td>C</td>
</tr>
</tbody>
</table>
Recap: The Schema

• The schema of a relation is the name of the relation followed by a parenthesized list of attributes
  – CoursesTaken(Student, Course, Grade)

• A design in a relational model consists of a set of schemas.
  – Such a set of schemas is called a relational database schema
Converting ER Diagram to Relational Design

- Entity Set → Relation
  - Attribute of Entity Set → Attribute of a Relation
- Relationship → relation whose attributes are
  - Attribute of the relationship itself
  - Key attributes of the connected entity sets
- Several special cases:
  - Weak entity sets.
  - Combining relations (especially for many-one relationships)
  - ISA relationships and subclasses
Subclasses in ER Diagrams

Product
  isa Software Product
  isa Educational Product

name
category
price

platforms

Age Group
Example: ER Diagram
Schemas for Entity Sets

- For each entity set, create a relation with the same name and with the same set of attributes
  - Students (Name, Address)
  - Professors (Name, Office, Age)
  - Departments (Name)
Mapping Entity Types

EMPLOYEE(SSN, address, first name, last name)
PROJECT(PNR, pname, pduration)
Mapping Multi-Valued Attribute Types

• For each multi-valued attribute type, we create a new relation \( R \).
• We put the multi-valued attribute type in \( R \) together with a foreign key referring to the primary key of the original relation.
• Multi-valued composite attribute types are again decomposed into their components.
• The primary key can then be set based upon the assumptions.
EMPLOYEE(SSN, ename, address)
EMP-PHONE(PhoneNr, SSN)
- SSN foreign key refers to SSN in EMPLOYEE, NULL NOT ALLOWED
## Mapping Multi-Valued Attribute Types

### EMPLOYEE

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>511</td>
<td>John Smith</td>
<td>14 Avenue of the Americas, New York</td>
<td>001</td>
</tr>
<tr>
<td>289</td>
<td>Paul Barker</td>
<td>208 Market Street, San Francisco</td>
<td>001</td>
</tr>
<tr>
<td>356</td>
<td>Emma Lucas</td>
<td>432 Wacker Drive, Chicago</td>
<td>002</td>
</tr>
</tbody>
</table>

### EMP-PHONE

<table>
<thead>
<tr>
<th>PhoneNR</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>900-244-8000</td>
<td>511</td>
</tr>
<tr>
<td>900-244-8000</td>
<td>289</td>
</tr>
<tr>
<td>900-244-8002</td>
<td>289</td>
</tr>
<tr>
<td>900-246-6006</td>
<td>356</td>
</tr>
</tbody>
</table>
Composite Attributes Types

- first name
- last name
- name
- email
- SUPNR
- SUPPLIER
- address
- date of birth
- status
- Street
- Number
- ZIP
- City
- Country
Schemas for Relationships

• For each relationship, create a relation with the same name whose attributes are
  – Attributes of the relationship itself
  – Key attributes of the connected entity sets (even if they are weak)
• Take (StudentName, Address, Number, DepartmentName)
• Teach (ProfessorName, Office, Number, DepartmentName)
• Evaluation (StudentName, Address, ProfessorName, Office, Number, DepartmentName, Grade)
Combining Relations

- Consider many-one Teach relationship from Courses to Professors
- Schemas are:
  - Courses(Number, DepartmentName, CourseName, Classroom, Enrollment)
  - Professors(Name, Office, Age)
  - Teach(Number, DepartmentName, ProfessorName, Office)
- The key for Courses uniquely determines all attributes of Teach
- We can combine the relations for Courses and Teach into a single relation whose attributes are
  - All the attributes for Courses
  - Any attributes of Teach
  - The key attributes of Professors
Rules for Combining Relations

• We can combine into one new relation Q
  – The relation for an entity set E
  – all many-to-one relationships R1, R2, ..., Rk from E to other entity sets E1, E2, ..., Ek respectively

• The attributes of Q are
  – All the attributes of E
  – Any attributes of R1, R2, ..., Rk
  – The key attributes of E1, E2, ..., Ek
If an entity set E appears $k > 1$ times in a relationship $R$ (in different roles), the key attributes for E appear $k$ times in the relation for $R$, appropriately renamed

- PreReq (RequirerNumber, RequirerDeptName, RequirementNumber, RequirementDeptName)
Supporting Relationships

- **Departments**(Name)
- **Courses**(Number, DepartmentName, CourseName, Classroom, Enrollment)
- **Offer**(Name, Number, DepartmentName)
  - But Name and DepartmentName are identical, so the schema for Offer is **Offer**(Number, DepartmentName)
  - The schema for Offer is a subset of the schema for the weak entity set, so we can dispense with the relation for Offer
Rules for Supporting Relationships

• If W is a weak entity set, the relation for W has a schema whose attributes are
  – all attributes of W
  – all attributes of supporting relationships for W
  – for each supporting relationship for W to an entity set E
  – the key attributes of E

• There is no relation for any supporting relationship for W
Mapping Relationship Types

• Mapping a binary 1:1 relationship type
• Mapping a binary 1:N relationship type
• Mapping a binary N:M relationship type
• Mapping unary relationship types
• Mapping n-ary relationship types
Mapping a Binary 1:1 Relationship Type

• Create **two relations: one for each entity type** participating in the relationship type
• The connection can be made by including a foreign key in one of the relations to the primary key of the other
• In case of existence dependency (participation constraint), put the foreign key in the existence-dependent relation and declare it as NOT NULL
• The attribute types of the 1:1 relationship type can then be added to the relation with the foreign key
Mapping a Binary 1:1 Relationship Type
Mapping a Binary 1:1 Relationship Type

EMPLOYEE(SSN, ename, address)
DEPARTMENT(DNR, dname, dlocation, SSN)
- SSN foreign key refers to SSN in EMPLOYEE, NULL NOT ALLOWED
- SSN is an alternate key

<table>
<thead>
<tr>
<th>EMPLOYEE(SSN, ename, address)</th>
</tr>
</thead>
<tbody>
<tr>
<td>511</td>
</tr>
<tr>
<td>289</td>
</tr>
<tr>
<td>356</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPARTMENT(DNR, dname, dlocation, SSN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>001 Market 3rd floor 511</td>
</tr>
<tr>
<td>002 Call center 2nd floor 511</td>
</tr>
<tr>
<td>003 Finance basement 289</td>
</tr>
<tr>
<td>004 ICT 1st floor 511</td>
</tr>
</tbody>
</table>
Mapping a Binary 1:N Relationship Type

- Binary 1:N relationship types can be mapped by including a foreign key in the relation corresponding to the participating entity type at the N-side of the relationship type.
- The foreign key refers to the primary key of the relation corresponding to the entity type at the 1-side of the relationship type.
- Depending upon the minimum cardinality, the foreign key can be declared as NOT NULL or NULL ALLOWED.
- The attribute types of the 1:N relationship type can be added to the relation corresponding to the participating entity type.
Mapping a Binary 1:N Relationship Type

EMployee(SSN, ename, address, starting date, DNR)
- DNR foreign key refers to DNR in DEPARTMENT, NULL NOT ALLOWED

Department(DNR, dname, dlocation)
## Mapping a Binary 1:N Relationship Type

### Employee Relations

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Start Date</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>511</td>
<td>John Smith</td>
<td>14 Avenue of the Americas, New York</td>
<td>01/01/2000</td>
<td>Marketing</td>
</tr>
<tr>
<td>289</td>
<td>Paul Barker</td>
<td>208 Market Street, San Francisco</td>
<td>01/01/1998</td>
<td>Call center</td>
</tr>
<tr>
<td>356</td>
<td>Emma Lucas</td>
<td>432 Wacker Drive, Chicago</td>
<td>01/01/2010</td>
<td>Finance</td>
</tr>
</tbody>
</table>

### Department Relations

<table>
<thead>
<tr>
<th>DNR</th>
<th>Name</th>
<th>Location</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Marketing</td>
<td>3rd floor</td>
<td>511</td>
</tr>
<tr>
<td>002</td>
<td>Call center</td>
<td>2nd floor</td>
<td>511</td>
</tr>
<tr>
<td>003</td>
<td>Finance</td>
<td>basement</td>
<td>289</td>
</tr>
<tr>
<td>004</td>
<td>ICT</td>
<td>1st floor</td>
<td>511</td>
</tr>
</tbody>
</table>
Mapping a Binary N:M Relationship Type

• M:N relationship types are mapped by introducing a new relation R

• The primary key of R is a combination of foreign keys referring to the primary keys of the relations corresponding to the participating entity types

• The attribute types of the M:N relationship type can also be added to R
Mapping a Binary N:M Relationship Type

**EMPLOYEE(SSN, ename, address)**

**PROJECT(PNR, pname, pduration)**

**WORKS_ON(SSN, PNR, hours)**

- SSN foreign key refers to SSN in EMPLOYEE, NULL NOT ALLOWED
- PNR foreign key refers to PNR in PROJECT, NULL NOT ALLOWED
Mapping a Binary N:M Relationship Type

<table>
<thead>
<tr>
<th>EMPLOYEE (SSN, ename, address, DNR)</th>
<th>PROJECT (PNR, pname, pduration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>511 John Smith 14 Avenue of the Americas, New York 001</td>
<td></td>
</tr>
<tr>
<td>289 Paul Barker 208 Market Street, San Francisco 001</td>
<td></td>
</tr>
<tr>
<td>356 Emma Lucas 432 Wacker Drive, Chicago 002</td>
<td></td>
</tr>
<tr>
<td>1001 B2B 100</td>
<td></td>
</tr>
<tr>
<td>1002 Analytics 660</td>
<td></td>
</tr>
<tr>
<td>1003 Web site 52</td>
<td></td>
</tr>
<tr>
<td>1004 Hadoop 826</td>
<td></td>
</tr>
</tbody>
</table>

WORKS_ON(SSN, PNR, hours)

<table>
<thead>
<tr>
<th>SSN</th>
<th>PNR</th>
<th>hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>511</td>
<td>1001</td>
<td>10</td>
</tr>
<tr>
<td>289</td>
<td>1001</td>
<td>80</td>
</tr>
<tr>
<td>289</td>
<td>1003</td>
<td>50</td>
</tr>
</tbody>
</table>
Mapping Unary Relationship Types

- A recursive 1:1 or 1:N relationship type can be implemented by adding a foreign key referring to the primary key of the same relation.

- For an N:M recursive relationship type, a new relation R needs to be created with two NOT NULL foreign keys referring to the original relation.
EMPLOYEE(SSN, ename, address, supervisor)
- supervisor foreign key refers to SSN in EMPLOYEE, NULL ALLOWED
- supervisor is an alternate key
Mapping Unary Relationship Types

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>City, State</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>511</td>
<td>John Smith</td>
<td>14 Avenue of the Americas, New York</td>
<td></td>
<td>289</td>
</tr>
<tr>
<td>289</td>
<td>Paul Barker</td>
<td>208 Market Street, San Francisco</td>
<td></td>
<td>412</td>
</tr>
<tr>
<td>356</td>
<td>Emma Lucas</td>
<td>432 Wacker Drive, Chicago</td>
<td></td>
<td>289</td>
</tr>
<tr>
<td>412</td>
<td>Dan Kelly</td>
<td>668 Strip, Las Vegas</td>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>
Mapping Unary Relationship Types

EMPLOYEE(SSN, ename, address)

SUPERVISION(Supervisor, Supervisee)

- Supervisor foreign key refers to SSN in EMPLOYEE, NULL NOT ALLOWED
- Supervisee foreign key refers to SSN in EMPLOYEE, NULL NOT ALLOWED
## Mapping Unary Relationship Types

<table>
<thead>
<tr>
<th>EMPLOYEE(SSN, ename, address)</th>
<th>SUPERVISION(Supervisor, Supervisee)</th>
</tr>
</thead>
<tbody>
<tr>
<td>511  John Smith  14 Avenue of the Americas, New York</td>
<td>289  511</td>
</tr>
<tr>
<td>289  Paul Barker  208 Market Street, San Francisco</td>
<td>289  356</td>
</tr>
<tr>
<td>356  Emma Lucas  432 Wacker Drive, Chicago</td>
<td>412  289</td>
</tr>
<tr>
<td>412  Dan Kelly  668 Strip, Las Vegas</td>
<td>412  511</td>
</tr>
</tbody>
</table>
Mapping n-ary Relationship Types

• To map an n-ary relationship type, we first create relations for each participating entity type.
• We then also define one additional relation R to represent the n-ary relationship type and add foreign keys referring to the primary keys of each of the relations corresponding to the participating entity types.
• The primary key of R is the combination of all foreign keys which are all NOT NULL.
• Any attribute type of the n-ary relationship can also be added to R.
Mapping n-ary Relationship Types

TOURIST(TNR, ...)
TRAV_AGENCY(ANR, ...)
HOTEL(HNR, ...)
BOOKING(TNR, ANR, HNR, price)

- TNR foreign key refers to TNR in Tourist, NULL NOT ALLOWED
Mapping n-ary Relationship Types

INSTRUCTOR(INR, ...)
COURSE(CNR, ...)
SEMESTER(SEM-YEAR, ...)
OFFERS(INR, CNR, SEM-YEAR)
- INR foreign key refers to ...
## Mapping n-ary Relationship Types

<table>
<thead>
<tr>
<th>INSTRUCTOR( INR, iname, ...)</th>
<th>COURSE( CNR, cname, ...)</th>
<th>SEMESTER( SEM-YEAR, ...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Bart</td>
<td>100 Database Management</td>
<td>1-2015</td>
</tr>
<tr>
<td>12 Wilfried</td>
<td>110 Analytics</td>
<td>2-2015</td>
</tr>
<tr>
<td>14 Seppe</td>
<td>120 Java Programming</td>
<td>1-2016</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OFFERS( INR, CNR, SEM-YEAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 100 1-2015</td>
</tr>
<tr>
<td>12 100 1-2016</td>
</tr>
<tr>
<td>10 120 1-2015</td>
</tr>
<tr>
<td>14 120 1-2015</td>
</tr>
</tbody>
</table>
Mapping Weak Entity Sets

• A **weak entity set** should be mapped into a **relation** $R$ with all its corresponding attribute types

• A **foreign key** must be added referring to the primary key of the relation corresponding to the owner entity type

• Because of the existence dependency, the foreign key is declared as **NOT NULL**

• The primary key of $R$ is then the combination of the partial key and the foreign key
Mapping Weak Entity Sets

Hotel (HNR, Hname)
Room (RNR, HNR, beds)
- HNR foreign key refers to HNR in Hotel, NULL NOT ALLOWED
Mapping Weak Entity Sets

<table>
<thead>
<tr>
<th>ROOM (RNR, HNR, Beds)</th>
<th>HOTEL (HNR, Hname)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 101 2</td>
<td>100 Holiday Inn New York</td>
</tr>
<tr>
<td>6 101 4</td>
<td>101 Holiday Inn Chicago</td>
</tr>
<tr>
<td>8 102 2</td>
<td>102 Holiday Inn San Francisco</td>
</tr>
</tbody>
</table>
## Putting It All Together

<table>
<thead>
<tr>
<th>ER model</th>
<th>Relational model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity set</td>
<td>Relation</td>
</tr>
<tr>
<td>Weak entity set</td>
<td>Foreign key</td>
</tr>
<tr>
<td>1:1 or 1:N relationship type</td>
<td>Foreign key</td>
</tr>
<tr>
<td>M:N relationship type</td>
<td>New relation with two foreign keys</td>
</tr>
<tr>
<td>n-ary relationship type</td>
<td>New relation with n foreign keys</td>
</tr>
<tr>
<td>Simple attribute type</td>
<td>Attribute type</td>
</tr>
<tr>
<td>Composite attribute type</td>
<td>Component attribute type</td>
</tr>
<tr>
<td>Multi-valued attribute type</td>
<td>Relation and foreign key</td>
</tr>
<tr>
<td>Key attribute type</td>
<td>Primary or alternative key</td>
</tr>
</tbody>
</table>
Putting it All Together

[Database schema diagram showing relationships between EMPLOYEE, DEPARTMENT, PROJECT, WORKS IN, WORKS ON, MANAGES, IN CHARGE OF, SSN, ename, address, dnr, dname, dlocation, hours, pnr, pname, pduration]
Putting It All Together

• EMPLOYEE (SSN, ename, streetaddress, city, sex, dateofbirth, SUPERVISOR, DNR)
  - SUPERVISOR foreign key refers to SSN in EMPLOYEE, NULL ALLOWED
  - DNR foreign key refers to DNR in DEPARTMENT, NOT NULL

• DEPARTMENT (DNR, dname, dlocation, MGNR)
  - MGNR: foreign key refers to SSN in EMPLOYEE, NOT NULL
  - MGNR is an alternate key

• PROJECT (PNR, pname, pduration, DNR)
  - DNR: foreign key refers to DNR in DEPARTMENT, NOT NULL

• WORKS-ON (SSN, PNR, HOURS)
  - SSN foreign key refers to SSN in EMPLOYEE, NOT NULL
  - PNR foreign key refers to PNR in PROJECT, NOT NULL
Reading and Next Class

• Entity/Relationship Models II
  – Ch 2, 3
• Next: SQL I: Ch 5