Participation of an Entity Set in a Relationship Set

- Total participation (double line): every entity in the entity set participates in at least one relationship in the relationship set
  - E.g., participation of section in sec_course is total
    - every section must have an associated course
  - Partial participation: some entities may not participate in any relationship in the relationship set
  - Example: participation of course in section is partial
Cardinality Constraints on Ternary Relationship

- We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint.
- E.g., an arrow from `proj_guide` to `instructor` indicates each student has at most one guide for a project.
- If there is more than one arrow, there are two ways of defining the meaning.
  - E.g., a ternary relationship $R$ between $A$, $B$ and $C$ with arrows to $B$ and $C$ could mean
    1. each $A$ entity is associated with a unique entity from $B$ and $C$ or
    2. each pair of entities from $(A, B)$ is associated with a unique $C$ entity, and each pair $(A, C)$ is associated with a unique $B$
  - Each alternative has been used in different formalisms
  - To avoid confusion we outlaw more than one arrow.
Weak Entity Sets

- An entity set that does not have a primary key is referred to as a **weak entity set**.
- The existence of a weak entity set depends on the existence of a **identifying entity set**
  - It must relate to the identifying entity set via a one-to-many relationship set from the identifying to the weak entity set
  - Weak side of relationship set must be *total*
  - **Identifying relationship** depicted using a double diamond
- The **discriminator** (or *partial key*) of a weak entity set is the set of attributes that *help* distinguish among all the entities of a weak entity set.
- The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set’s discriminator.

Weak Entity Sets (Cont.)

- We underline the discriminator of a weak entity set with a dashed line.
- We put the identifying relationship of a weak entity in a double diamond.
- Primary key for **section** – *(course_id, sec_id, semester, year)*
Weak Entity Sets (Cont.)

- Note: the primary key of the strong entity set is not explicitly stored with the weak entity set, since it is implicit in the identifying relationship.

- If course_id were explicitly stored, section could be made a strong entity, but then the relationship between section and course would be duplicated by an implicit relationship defined by the attribute course_id common to course and section.

Symbols Used in E-R Notation

- Entity set (E)
- Relationship set (R)
- Identifying relationship set for weak entity set
- Total participation of entity set in relationship
- Attributes:
  - simple (A1)
  - composite (A2)
  - multivalued (A3)
  - derived (A4)
- Primary key
- Discriminating attribute of weak entity set
Symbols Used in E-R Notation (Cont.)

- \( R \) many-to-many relationship
- \( R \) many-to-one relationship
- \( R \) one-to-one relationship
- \( R \) cardinality limits
- \( R \) role indicator

E-R Diagram for a University Enterprise
Redundant Attributes

• Suppose we have entity sets
  • instructor, with attributes including dept_name
department
  and a relationship
  • inst_dept relating instructor and department

• Attribute dept_name in entity instructor is redundant since there is an explicit relationship inst_dept which relates instructors to departments
  • The attribute replicates information present in the relationship, and should be removed from instructor
  • BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see.

Reduction to Relational Schemas
Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as relation schemas that represent the contents of the database.

- A database which conforms to an E-R diagram can be represented by a collection of schemas.

- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.

- Each schema has a number of columns (generally corresponding to attributes), which have unique names.

Representing Entity Sets With Simple Attributes

- A strong entity set reduces to a schema with the same attributes `student(course_ID, title, credits)`

- A weak entity set becomes a table that includes a foreign key for the primary key of the identifying strong entity set `section ( course_id, sec_id, semester, year )`
Representing Relationship Sets

- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set \(\text{advisor}\)

\[
\text{advisor} = (s\_id, i\_id)
\]

Redundancy of Schemas

- Many-to-one and one-to-many relationship sets that are \textit{total} on the many-side can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side
- Example: Instead of creating a schema for relationship set \(\text{inst\_dept}\), add an attribute \(\text{dept\_name}\) to the schema arising from entity set \(\text{instructor}\)
Redundancy of Schemas (Cont.)

- For one-to-one relationship sets, either side can be chosen to act as the “many” side
  - That is, extra attribute can be added to either of the tables corresponding to the two entity sets

Instructor

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
</table>

department

| dept_name | building | budget |

- Instructor(ID, dept_name, name, salary)

  or

- department(dept_name, ID, building, budget)

Redundancy of Schemas (Cont.)

- If participation is partial on the “many” side, replacing a relationship schema by an extra attribute in the schema corresponding to the “many” side could result in null values (generally avoided)
  - i.e. the approach in the previous slides does not work
  - need to represent relationship as a separate table

- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.
  - Example: The section schema already contains the attributes that would appear in the sec_course schema
Composite Attributes

- Composite attributes flattened out
  - Example: given entity set *instructor*
    - with composite attribute *name*
    - with component attributes *first_name* and *last_name*
    - replace with *name_first_name* and *name_last_name*
      - Prefix omitted if there is no ambiguity
- Ignoring multivalued attributes, extended instructor schema is
  - *instructor*(ID, 
    first_name, 
    middle_initial, 
    last_name, 
    street, 
    street_number, 
    street_name, 
    apt_number, 
    city, 
    state, 
    zip, 
    phone_number, 
    date_of_birth, 
    age ( )

Multivalued Attributes

- Multivalued attribute *M* of entity *E* represented by a separate schema *EM*
  - Schema *EM* includes *E*’s primary key and attribute corresponding to *M*
  - Example: Multivalued attribute *phone_number* of *instructor*:
    - ```
      inst_phone= ( ID, phone_number )
    ```
  - Each value of the multivalued attribute maps to separate tuple of *EM*:
    - *instructor* entity with primary key 22222 and numbers 456-7890 and 123-4567 maps to:
      - (22222, 456-7890)
      - (22222, 123-4567)
Multivalued Attributes (Cont.)

- Special case: entity *section* has only one attribute other than the primary-key attributes, and that attribute is multi-valued
  - Optimization: Don’t create the relation corresponding to the entity, just create the one corresponding to the multivalued attribute
  - \textit{time_slot}(\textit{time_slot_id}, \textit{day}, \textit{start_time}, \textit{end_time})

ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now
- Schema per relationship set

 lots of foreign key dependences (weak, relationships..)
ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now

- Schema per relationship set

lots of foreign key dependences (weak, relationships..)
ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now

- Schema per relationship set

lots of foreign key dependences (weak, relationships..)
ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now

- Schema per relationship set

lots of foreign key dependencies (weak, relationships..)
ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now

- Schema per relationship set

lots of foreign key dependences (weak, relationships...)

lots of foreign key dependences (weak, relationships...)
Design Issues

• **Binary versus n-ary relationship sets**
  Although it is possible to replace any nonbinary (n-ary, for \( n > 2 \)) relationship set by a number of distinct binary relationship sets, a \( n \)-ary relationship set shows more clearly that several entities participate in a single relationship.

• **Placement of relationship attributes**
  e.g., attribute *date* as attribute of *advisor* or as attribute of *student*
Binary Vs. Non-Binary Relationships

- Some relationships that appear to be non-binary may be better represented using binary relationships
  - E.g., A ternary relationship *parents*, relating a child to his/her father and mother, is best replaced by two binary relationships, *father* and *mother*
    - Using two binary relationships allows partial information (e.g., only mother being know)
  - But there are some relationships that are naturally non-binary
    - Example: *proj_guide*, with several project members

Converting Non-Binary Relationships to Binary Form

- In general, any non-binary relationship can be represented using binary relationships by creating an artificial entity set.
  - Replace $R$ between entity sets $A$, $B$ and $C$ by an entity set $E$, and $R_A$, $R_B$, $R_C$, relating $E$ with $A$, $B$, and $C$
  - Create a special identifying attribute for $E$
  - Add any attributes of $R$ to $E$
  - For each relationship $(a_i, b_i, c_i)$ in $R$
    - create a new entity $e_i$ in the entity set $E$
    - add $(e_i, a_i)$ to $R_A$, etc.

![Diagram of converting non-binary relationships to binary form.](image-url)
Where did we come up with the schema that we used?
- E.g. why not store the actor names with movies?

If from an E-R diagram, then:
- Did we make the right decisions with the E-R diagram?

Goals:
- Formal definition of what it means to be a “good” schema.
- How to achieve it.
Relational Schemas and Redundancy

- movies(name, year, len)
- stars(name, addr, gender, birthdate)
- execs(name, cert#)
- studios(stud_name, address)

- in(star_name, movie_name, movie_year)
- made_by(movie_name, movie_year)
- produced_by(movie_name, movie_year)
- helmed_by(cert#)

![Diagram of relational schemas and redundancy]
Relational Schemas and Redundancy

• movies(name, year, len)
• stars(name, addr, gender, birthdate)
• execs(name, cert#)
• studios(stud_name, address, pres#)

• in(star_name, movie_name, movie_year)
• made_by(movie_name, movie_year)
• produced_by(movie_name, movie_year)
Relational Schemas and Redundancy

- movies(name, year, len, prod#)
- stars(name, addr, gender, birthdate)
- execs(name, cert#)
- studios(stud_name, address, pres#)

- in(star_name, movie_name, movie_year)
- made_by(movie_name, movie_year)

Relational Schemas and Redundancy

- movies(name, year, len, prod#, studio_name)
- stars(name, addr, gender, birthdate)
- execs(name, cert#)
- studios(stud_name, address, pres#)

- in(star_name, movie_name, movie_year)
- made_by(movie_name, movie_year)
Relational Schemas and Redundancy

- movies(name, year, len, prod#, studio_name)
- stars(name, addr, gender, birthdate)
- execs(name, cert#)
- studios(stud_name, address, pres#)

- in(star_name, movie_name, movie_year)

Is this a good idea???
Relational Database Design

or

“Troubles With Schemas“

Movie(title, year, length, inColor, studioName, producerC#, starName)

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Hamill</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Fisher</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>H. Ford</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>187</td>
<td>Yes</td>
<td>Universal</td>
<td>150</td>
<td>Watts</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>no</td>
<td>RKO</td>
<td>20</td>
<td>Fay</td>
</tr>
</tbody>
</table>

Issues:

1. Redundancy ➔ higher storage, inconsistencies ("anomalies")
   - update anomalies, insertion anomalies

2. Need nulls
   - Unable to represent some information without using nulls
   - How to store movies w/o actors (pre-productions etc)?
Movie(title, year, length, inColor, studioName, producerC#, starNames)

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarNames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>{Hamill, Fisher, H. Ford}</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>187</td>
<td>Yes</td>
<td>Universal</td>
<td>150</td>
<td>Watts</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>no</td>
<td>RKO</td>
<td>20</td>
<td>Fay</td>
</tr>
</tbody>
</table>

Issues:
3. Avoid sets
   - Hard to represent
   - Hard to query

Smaller schemas always good ????

Split Studio(name, address, presC#) into:
   Studio1 (name, presC#),
   Studio2(name, address)???

<table>
<thead>
<tr>
<th>Name</th>
<th>presC#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox</td>
<td>101</td>
</tr>
<tr>
<td>Studio2</td>
<td>101</td>
</tr>
<tr>
<td>Universial</td>
<td>102</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox</td>
<td>Address1</td>
</tr>
<tr>
<td>Studio2</td>
<td>Address1</td>
</tr>
<tr>
<td>Universial</td>
<td>Address2</td>
</tr>
</tbody>
</table>

This process is also called “decomposition”

Issues:
4. Requires more joins (w/o any obvious benefits)
5. Hard to check for some dependencies
   What if the “address” is actually the presC#’s address?
   No easy way to ensure that constraint (w/o a join).
Smaller schemas always good ????

Decompose StarsIn(movieTitle, movieYear, starName) into:

StarsIn1(movieTitle, movieYear)
StarsIn2(movieTitle, starName) ??

<table>
<thead>
<tr>
<th>movieTitle</th>
<th>movieYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>movieTitle</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>Hamill</td>
</tr>
<tr>
<td>King Kong</td>
<td>Watts</td>
</tr>
<tr>
<td>King Kong</td>
<td>Faye</td>
</tr>
</tbody>
</table>

Issues:
6. "joining" them back results in more tuples than what we started with
   (King Kong, 1933, Watts) & (King Kong, 2005, Faye)

This is a "lossy" decomposition
   We lost some constraints/information

The previous example was a "lossless" decomposition.

Desiderata

- No sets
- Correct and faithful to the original design
  - Avoid lossy decompositions
- As little redundancy as possible
  - To avoid potential anomalies
- No "inability to represent information"
  - Nulls shouldn’t be required to store information
- Dependency preservation
  - Should be possible to check for constraints

Not always possible.
We sometimes relax these for:

simpler schemas, and fewer joins during queries.