Relational Design from ER model

ISYS2120 Data and Information Management

Prof Alan Fekete

University of Sydney

Acknowledge: slides from Uwe Roehm and Alan Fekete, and from the materials associated with reference books (c) McGraw-Hill, Pearson

Choosing Relations based on E-R Diagram

- Schema design approach for
  - Strong Entities
  - Weak Entities
  - Relationships
    - Many-to-many, Many-to-one
    - Unary Relationships
    - Ternary Relationships
  - IsA Hierarchy

- We will concentrate in the lecture on typical examples…

Producing a relational schema from conceptual design

- A conceptual design (eg an E-R diagram) shows what facts about the real world should be kept, to support the organizational needs (including operational changes and analysis for reporting)
- To actually keep that information, the organisation will have a relational dbms which stores data structured in a relational schema
- It is essential, that *all* the necessary facts about *any* valid state of the world can be represented as an instance of the schema
- It is very desirable, that the schema has constraints that *enforce* properties that always hold in the domain
  - That is, one cannot represent in database, facts that could not possibly hold in the domain

Value of constraints

- Example: suppose we know that each supplier has a different SuppID
  - We want the dbms to reject any change that would include two suppliers with the same SuppID
- Example: suppose we know that each product comes from at most one supplier
  - We want dbms to reject any change that would give a second supplier for a given product
- Constraints in the schema prevent (some cases of) loss of data quality
  - They help make the database contents matching the real world

Overview of the process

- Look at entity sets and their attributes in ER diagram, and introduce tables that correspond
- Look at relationship sets in ER diagrams, and sometimes introduce extra tables to capture this information; in other cases, add extra columns to tables that were already introduced
- Try to make sure each entity set key information is reflected in primary key constraints
- Try to make sure each cardinality constraint from relationship set, is reflected in schema (sometimes by foreign key constraint, sometimes just in table structure with primary key information)
- Try to make sure each participation constraint from relationship set, is reflected in schema, usually by NOT NULL constraint

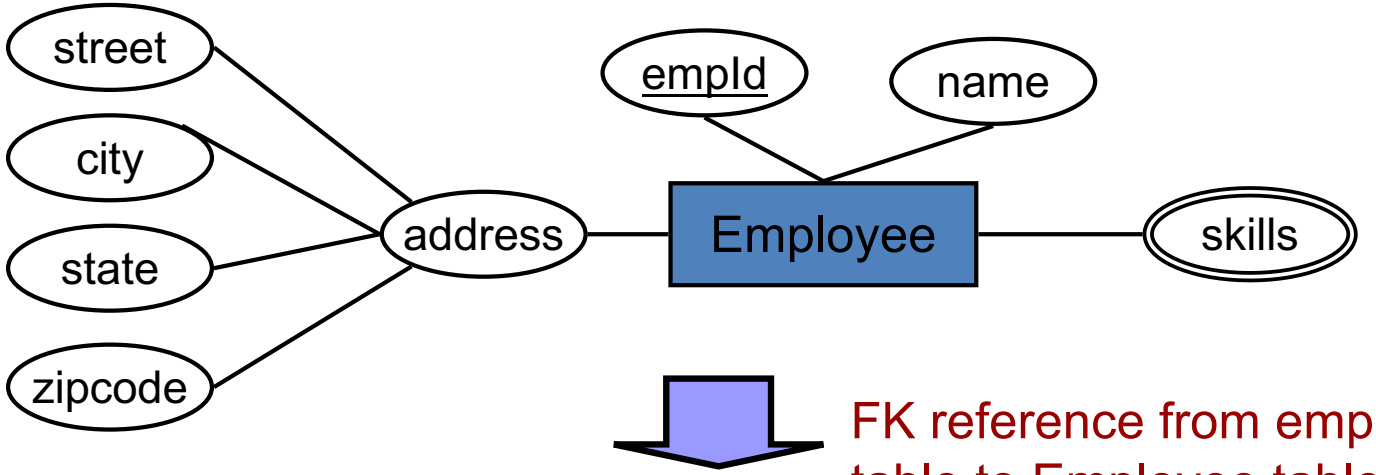
Mapping usual (strong) Entity Set to Tables

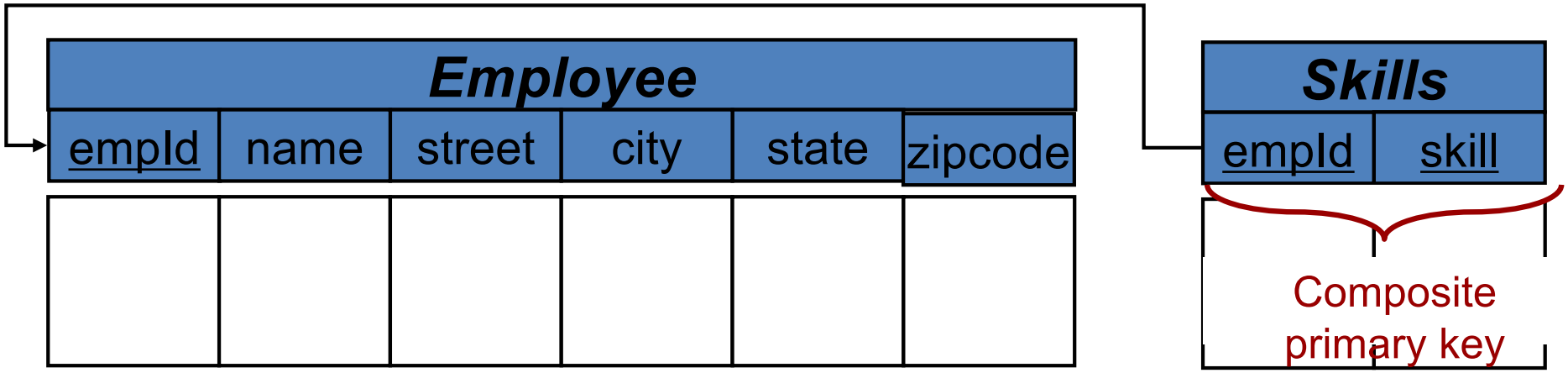
- Each **entity type** is captured in schema by creating a corresponding table, usually given the same name as the Entity Type
  - **Simple attributes**
    E-R simple attributes map directly onto columns in the table
    - Primary key of the table, is column(s) that correspond to primary key of the entity set
  - **Composite attributes**
    Composite attributes are flattened out by creating a separate column for each component attribute
    => We use only their simple, component attributes
  - **Multi-valued attribute**

    Do not include these in the table that corresponds to the entity set!
  - Instead, introduce an additional table, with one column for holding individual single values from the multiple collection, and another column which is a foreign key referencing the entity's primary key
    - The primary key of this table, is the combination of all the columns!
    - Eg When employee 57 has skills {coding, testing, carpentry}, there will be three rows for this in the additional table: (57, coding), (57, testing), (57, carpentry).

Example: Mapping Entity Types

- Employee entity type with composite/multi-valued attributes



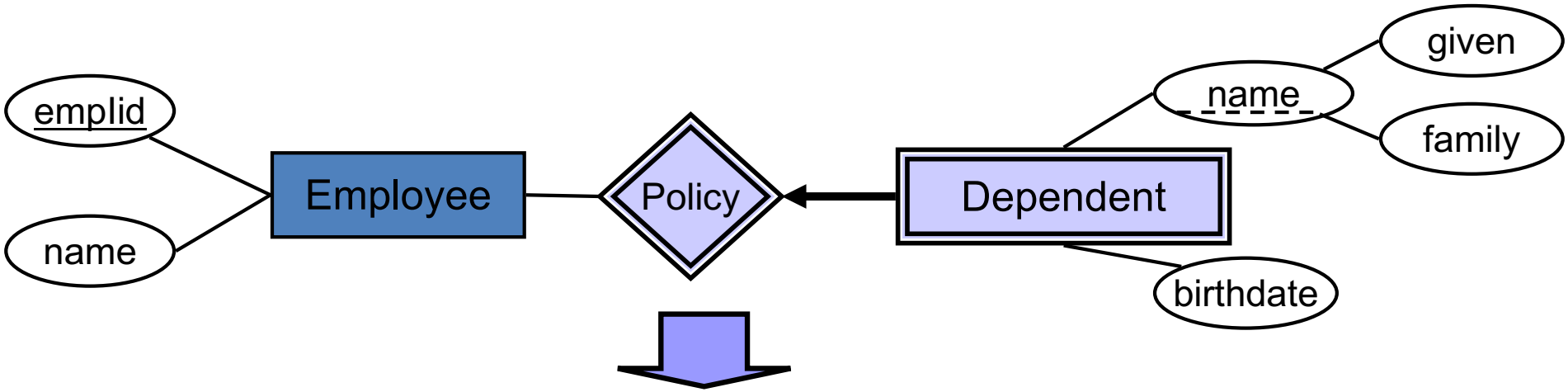


Mapping of Weak Entity Types

- Each **weak entity type** is captured in schema by a separate table
  - introduce a separate table with columns for the attributes (as for strong entity set) and *also* a column which is a foreign key taken from the owning entity
  - primary key of this table is composite: the combination with both
    - Discriminator of weak entity
    - Foreign key column(s) which references the primary key of relation from owning entity

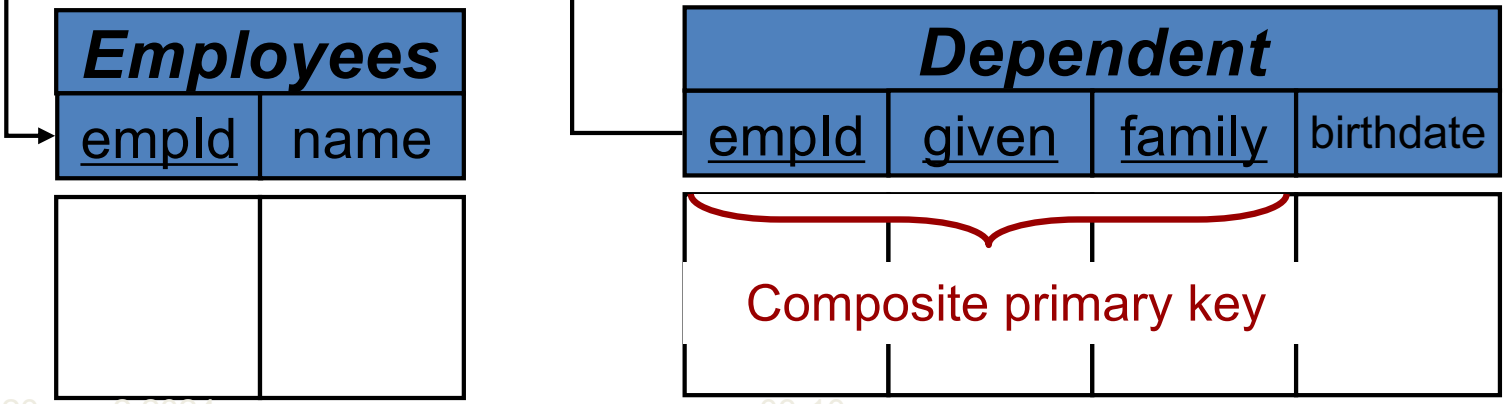
Example: Mapping of Weak Entity

- Weak entity set ‘Dependent’ with composite partial key



Foreign key

NOTE: the domain constraint for the foreign key of a weak entity should **not** allow *NULL*



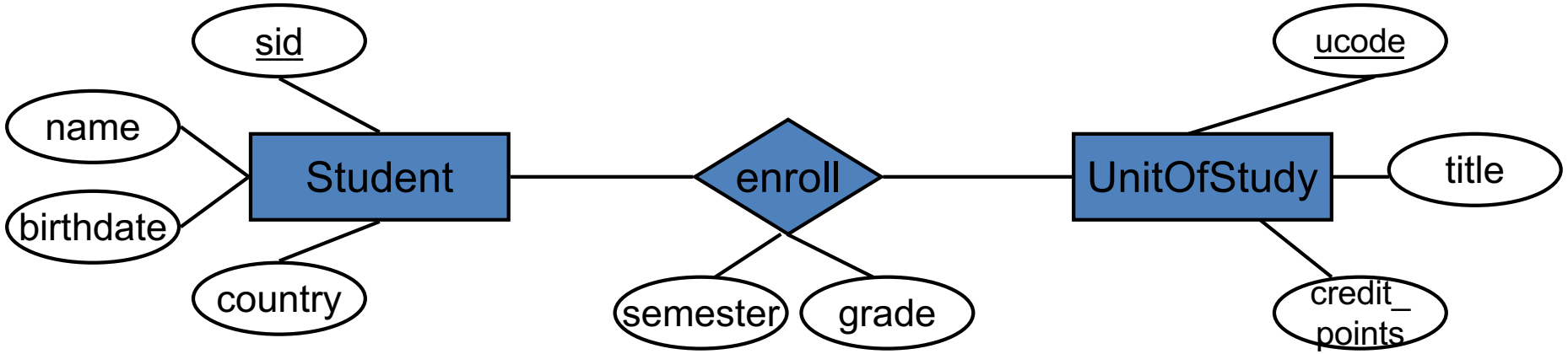
Mapping of Relationship Types

- There is a general technique (can be used for any relationship set)
- Introduce in schema an additional table (sometimes called a join-table) with columns that are non-null foreign keys (referencing the primary keys of the entity types involved), and also columns for any non-multivalued attributes on the relationship set itself
  - Name of join table can be name of relationship type, or sometimes, schema uses the combination of entity set names
  - Constraints in table depend on those in relationship type
  - Eg, if an entity type has cardinality that an entity is involved in at most one relationship, this means the primary key of new table is just the foreign key for that entity type
    - Otherwise (none of the entities are limited to being in “at most one” relationship), the primary key of new table will be the combination of all the columns which are foreign keys referencing involved entity types

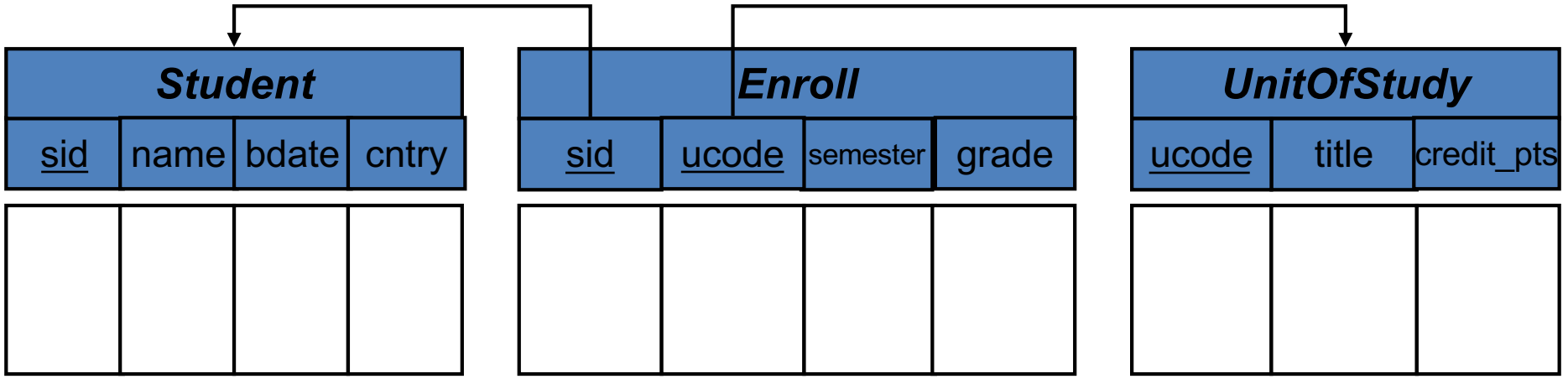
Multivalued attribute of the relationship type, needs yet another separate table, with non-null foreign keys to each involved entity’s table

Mapping of Many-to-Many Relationship Type

- Many-to-many relationship between Student & UnitOfStudy



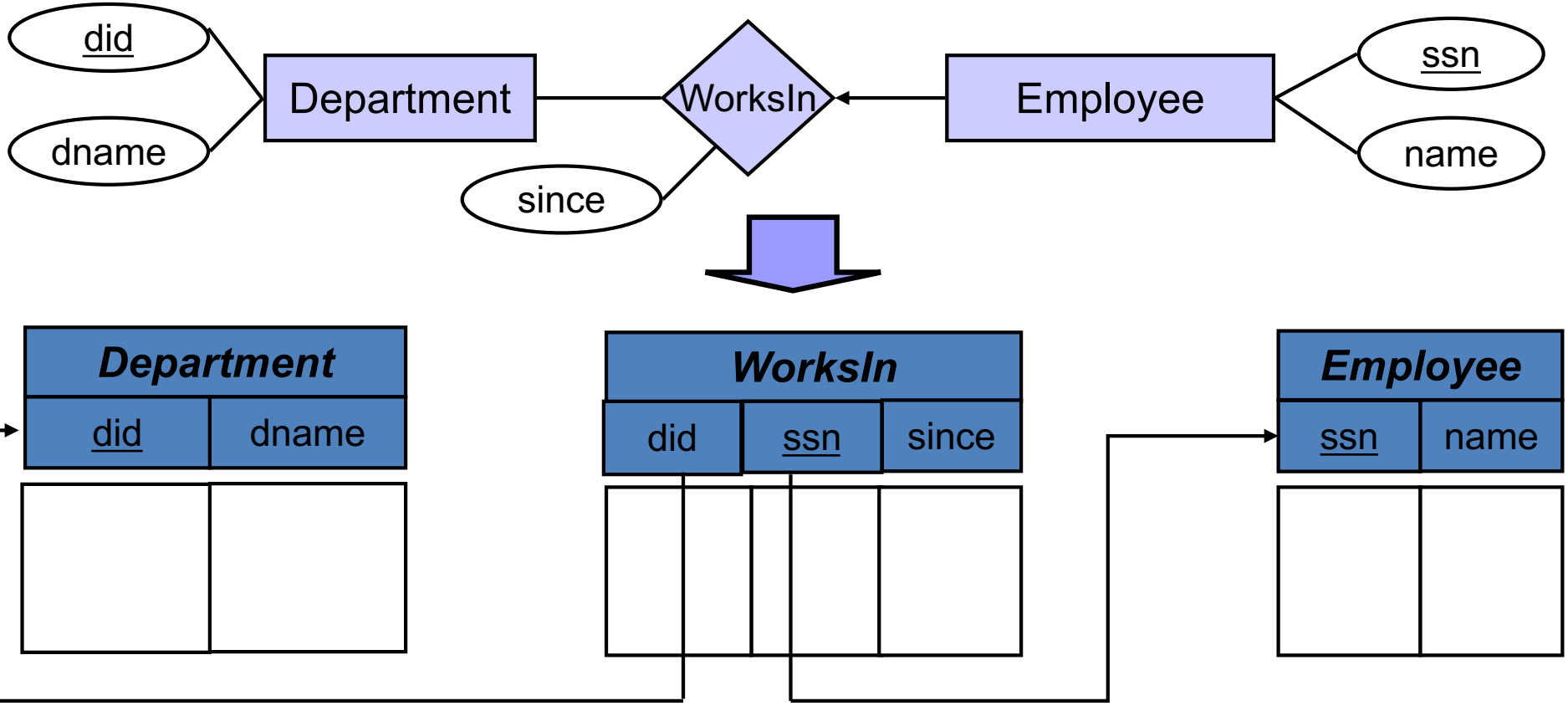
Note that primary key for Enroll table is composite (sid,ucode)



Alternative name for "join table" could be Student_Unit

Mapping of Many-to-One Relationship

- Many Employees to One Department



Note that ssn in WorksIn table is both primary key, and foreign key to Employee

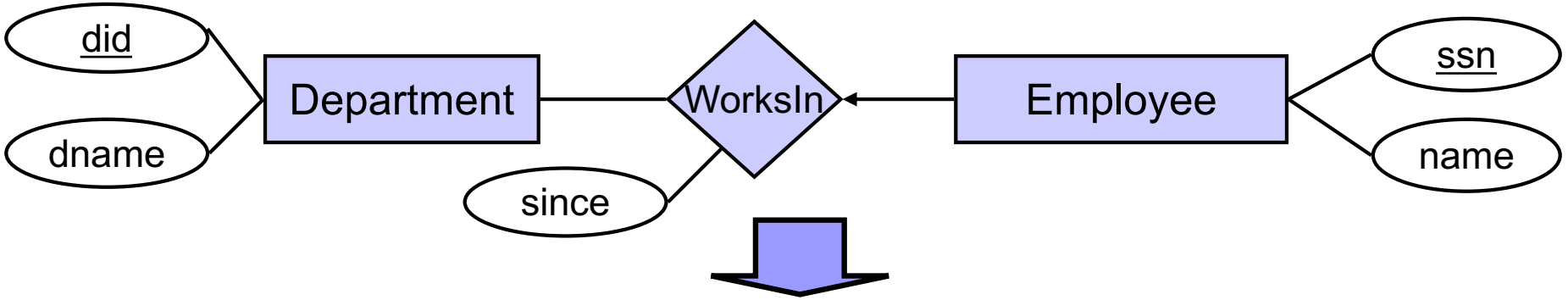
Alternative name for “join table” could be Employee_Department

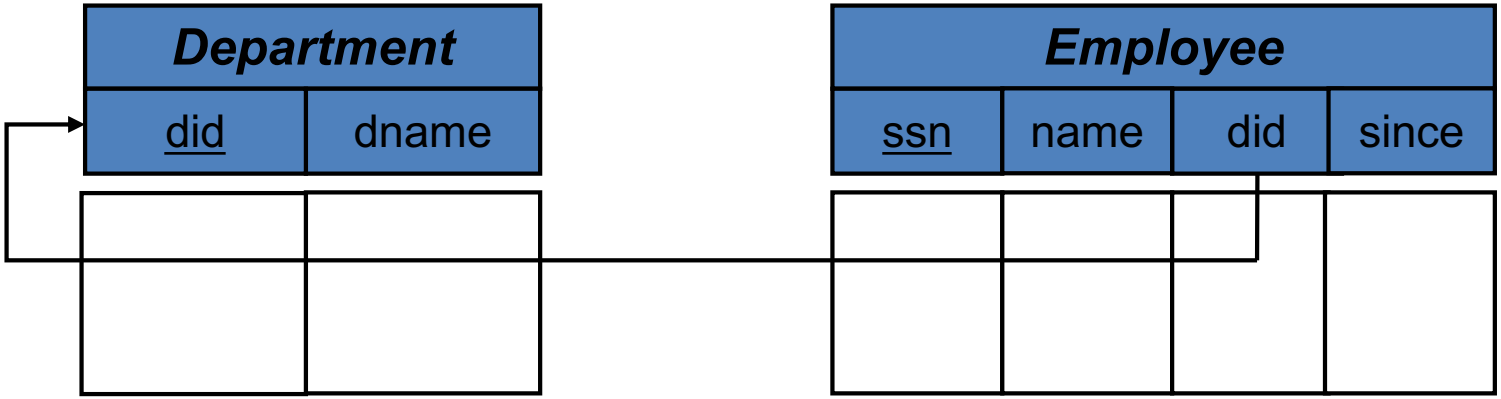
Alternative Mapping of Relationship Types

- In cases of many-to-one relationship, there is a different way to map this to the relational schema
- Not creating a join table, *instead* we might modify the schema of the table that corresponds to the many side, to introduce one or more extra columns
  - foreign key referencing the “one” side, and (if needed) other columns for the attributes of the relationship set
- Primary key of this table stays as before
- If there is also total participation constraint for many side: foreign key of other side is declared NOT NULL

Alternative Mapping of Many-to-One Relationship

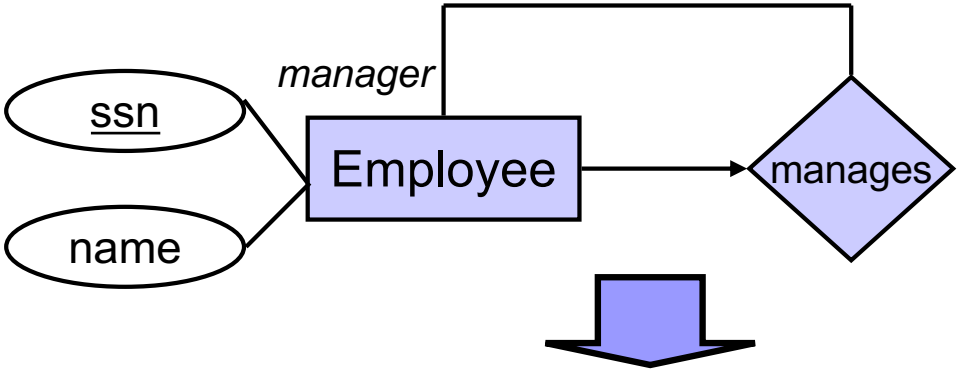
- Many Employees to One Department

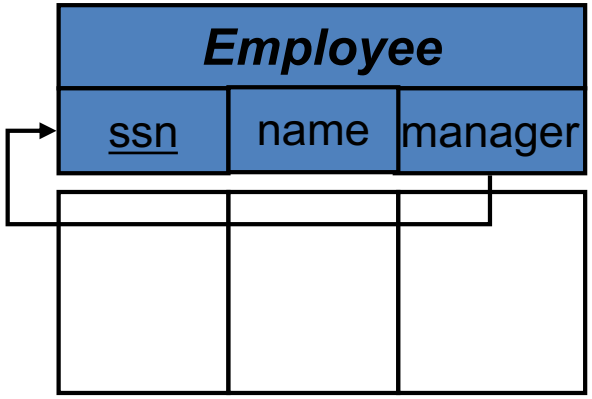




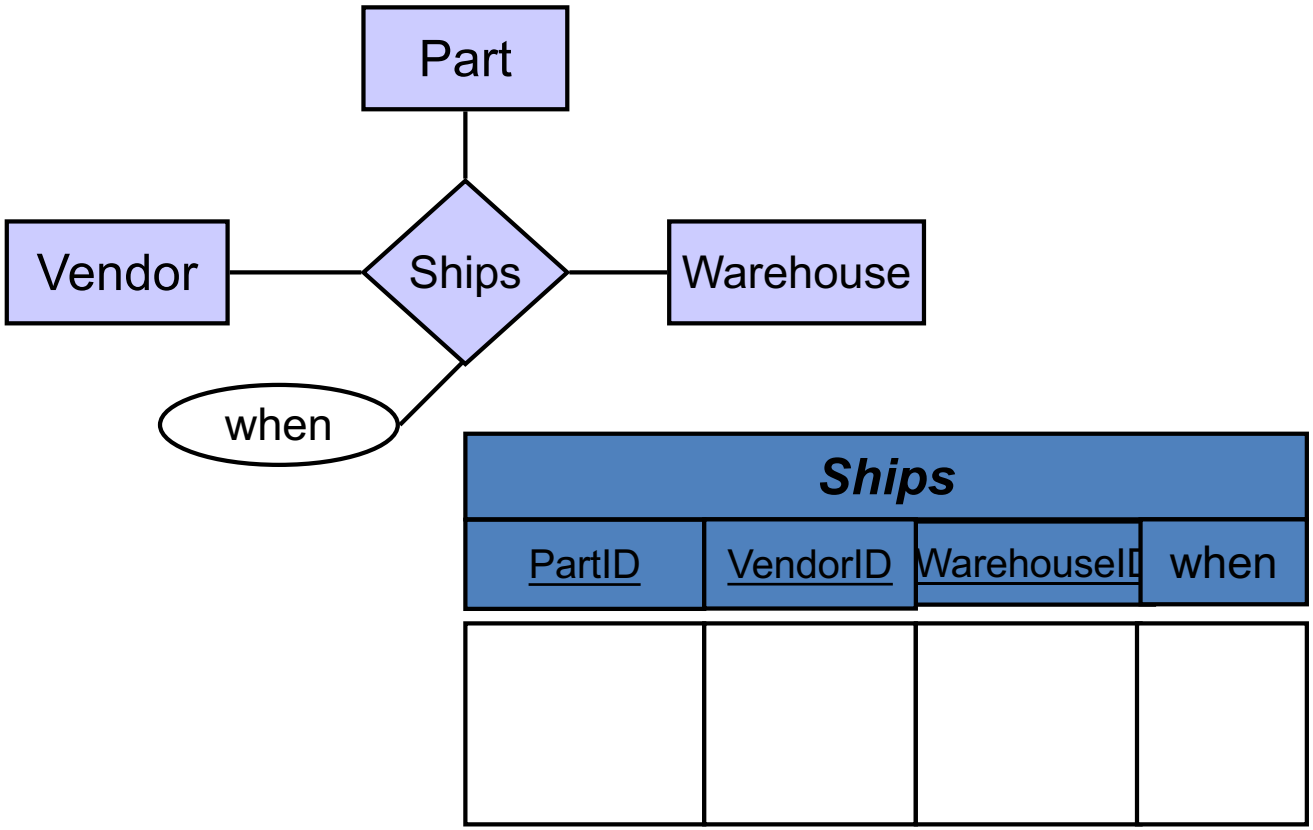
Mapping of recursive relationship

- Many Employees to One Department





Mapping of ternary relationship

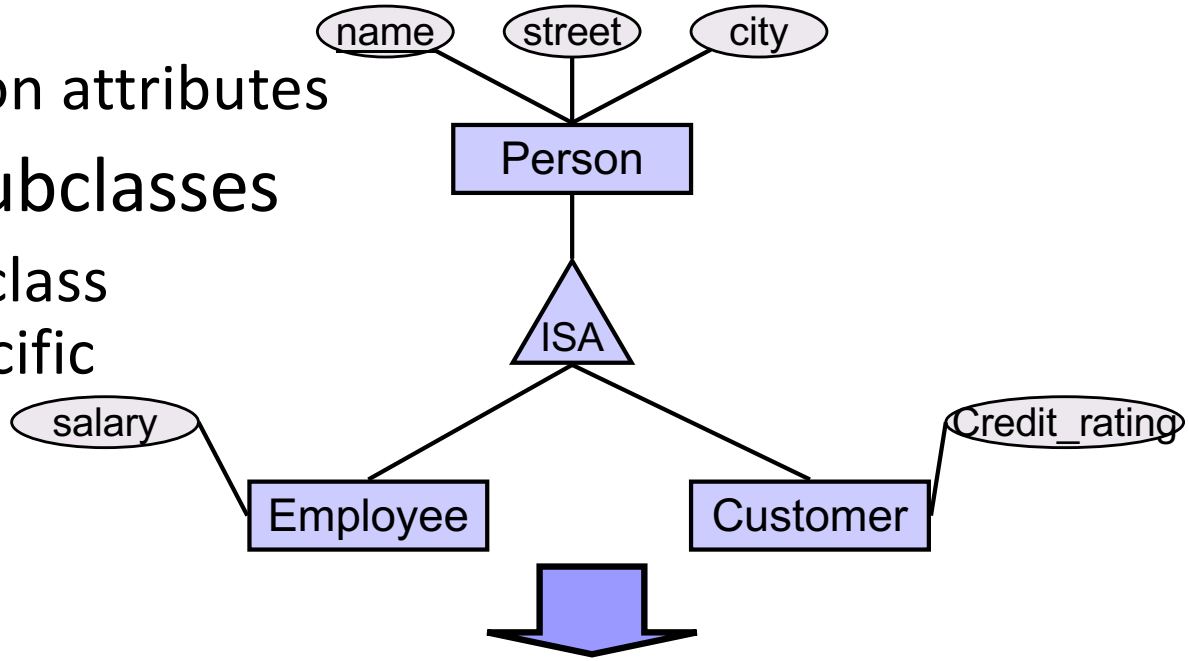


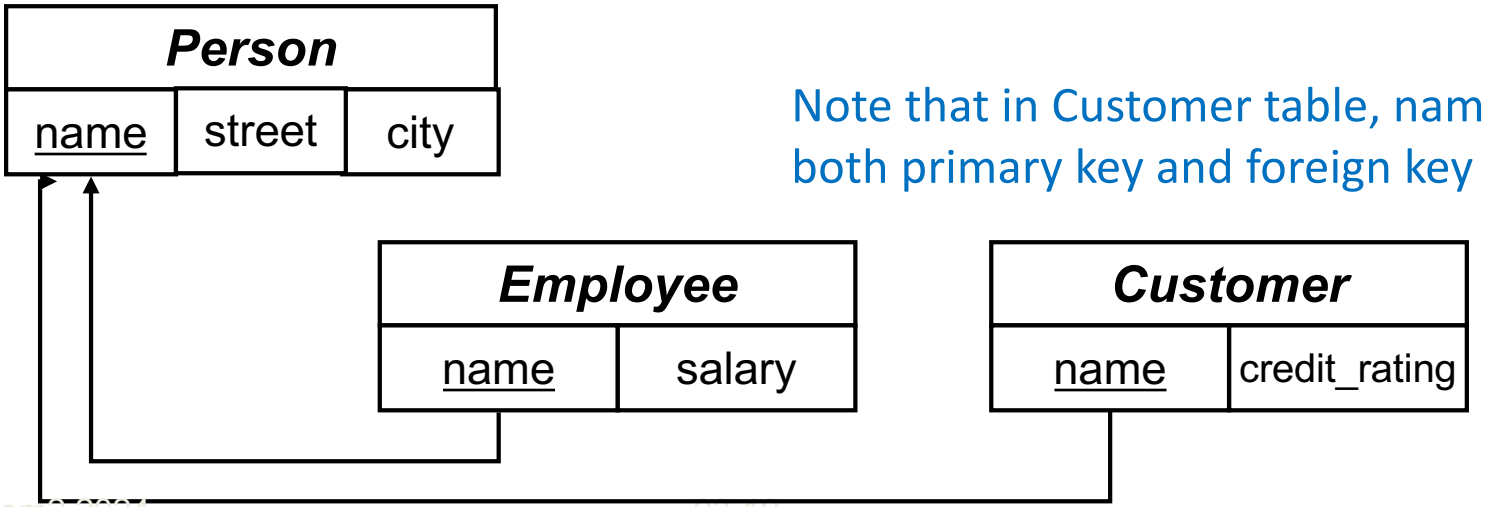
Mapping of ISA-Hierarchies

- Standard way (works always):
  - Distinct relations: one for the superclass and one for each subclass
  - Superclass attributes (including key and possible relationships) go into superclass relation
  - Subclass attributes go into the appropriate sub-relation; primary key of superclass relation is also included as primary key of subclass relation
  - primary key of a subclass relation is also a non-null foreign key referencing the superclass relation

Mapping of ISA-Hierarchy

- Table for superclass
  - Primary key + common attributes
- Separate tables for subclasses
  - Primary key of superclass (==foreign key) + specific attributes





Note that in Customer table, name is both primary key and foreign key to Person

Mapping of ISA-Hierarchies

- Alternative design (which can be used in presence of a **total** covering constraint):
  - Distinct relations for each subclass
    - Its attributes are all those in subclass and also all attributes inherited from higher sets
    - primary key of table is primary key from superclass
  - No foreign-keys needed to capture inheritance

Mapping of ISA-Hierarchies

- Another alternative design (can be used where subclasses are disjoint)
  - One relation for the superclass
  - All superclass and all subclass attributes are included in this table, and also an attribute to hold the string name of the subclass appropriate for this row
    - NULL used for any subclass attribute, in row for entity of a different subclass!

Summary: Schema Correspondence with E-R Model

- Tables correspond to entity types (entity sets) and to many-to-many relationship types/sets and to muti-valued attributes
  - And sometimes to other relationship types/sets,

- A single row correspond with an entity, or a relationship

- Columns correspond with attributes or many-to-one relationships.
- Primary key constraints reflect primary key (identifier) information in entity sets (for a table corresponding to the entity set)
  - In a join table corresponding to many-many relationship, the primary key combines the keys of the entity sets involved

Take care

- The word relation (in relational database) is NOT the same as the word relationship (in E-R model)
- In ER diagram, no entity set should ever have attributes that are keys of other entity sets
  - These connections should be shown in separate relationship sets
  - But in relational schema, tables can (and often do) include columns which are foreign keys referencing other tables

Checking a relational design

- Take some example state-of-domain
  - Invent it (but make sure ER constraints hold) if you don’t have real information
- Work out how to represent this in the relations
- Check each schema IC makes sense based on the ER diagram and constraints
  - Especially, check each column for NULL or NOT NULL; check every tables PK
- Also, see whether each of the constraints of ER diagram are enforced in schema

References

- Silberschatz/Korth/Sudarshan(7ed)
  - Chapter 6.7, 6.8.6, 6.9

Also

- Kifer/Bernstein/Lewis(complete version, 2ed)
  - Chapter 4.5
- Ramakrishnam/Gehrke(3ed)
  - Chapter 3.5
- Garcia-Molina/Ullman/Widom(complete book, 2ed)
  - Chapter 4.5

Summary

- Mapping approach for
  - Strong Entities
  - Weak Entities
  - Relationships
    - Many-to-many, Many-to-one
    - Unary Relationships
    - Ternary Relationships
  - IsA Hierarchy

Summary

- How information systems are built today: a DBMS is used to maintain and query large datasets, that are shared among many application programs
  - Benefits include recovery from system crashes, concurrent access, quick application development, data integrity and security
  - Data independence
- Overview of the relational model
  - Most important ideas and terminology
- The main roles of professionals who need to know about DBMS
- Some big ideas
  - Store the schema
  - Declarative queries
  - System Administration