COP 4710: Database Systems
Spring 2006

Chapter 2 – Introduction to Data Modeling
Part 2

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cop4710/spr2006

School of Electrical Engineering and Computer Science
University of Central Florida

Extensions of the E-R Model

- Some features of a real world situation can be difficult to model using only the features of the E-R model that we have seen so far.
- Some quite common concepts require extending the E-R model to incorporate mechanisms for modeling these features. Again, we won't look at all of them, but rather an overview of some of the more important extensions.

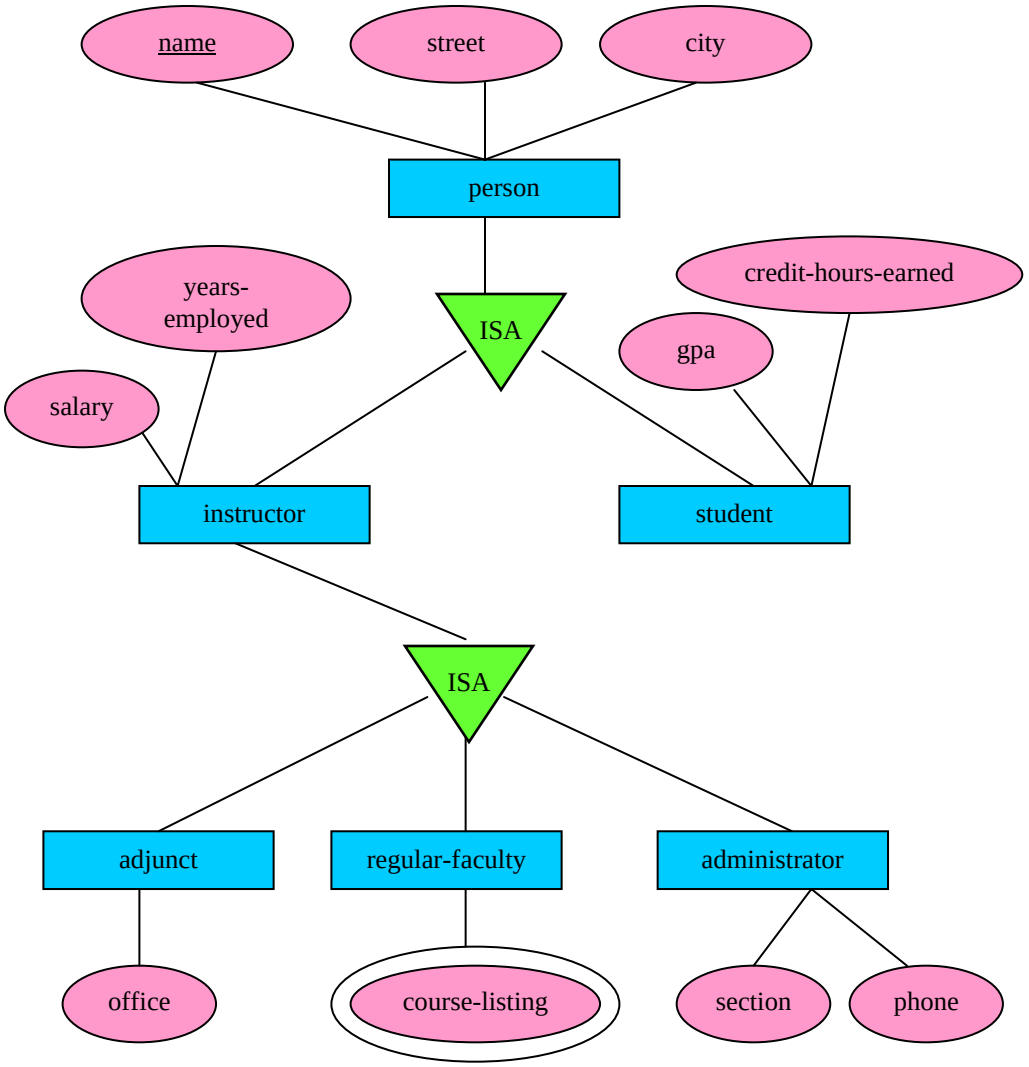
Specialization

- An entity set may include sub-groupings of entities that are distinct in some way from other entities in the set. For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the set.
  - As an example, consider the entity set *person*, with attributes *name*, *street*, and *city*. A person could further be classified as one of the following: *student* or *instructor*. Each of these person types is described by a set of attributes that includes all of the attributes of the entity set *person*, plus possibly some additional attributes. For example, *student* entities may be further described by the attributes *gpa*, and *credit-hours-earned*, whereas, *instructor* entities are not characterized by these attributes, but rather a different set such as, *salary*, and *years-employed*.
- The process of designating sub-groupings within an entity set is called *specialization*.

Specialization (cont.)

- The specialization of *person* allows us to distinguish among persons according to whether they are students or instructors.
- Specialization can be repeatedly applied so that there may be specializations within specializations.
- In terms of an E-R diagram, specialization is depicted by a triangle shaped component which is labeled ISA, which is a shorthand form of the "is-a" superclass-subclass relationship.
- The ISA relationship is illustrated in the diagram in the next slide.

Specialization (cont.)



Generalization

- The refinement from an initial entity set into successive levels of entity sub-groupings represents a top-down design approach in which distinctions are made explicit.
- This same design process could also proceed in a bottom-up approach, in which multiple entity sets are synthesized into a higher-level entity on the basis of common attributes. In other words, we might have first identified the entity set *students(name, address, city, gpa, credit-hours-earned)* and an entity set *instructors(name, address, city, salary, years-employed).*
- This commonality of attributes is expressed by *generalization*, which is a containment relationship that exists between a higher-level entity set and one or more lower level entity sets.

Generalization (cont.)

- In our example, *person* is the higher-level entity set and *instructor* and *student* are the lower-level entity sets.
- The higher-level entity set represents the *superclass* and the lower-level entity represents the *subclass*. Thus, *person* is the superclass of the *instructor* and *student* subclasses.
- For all practical purposes, generalization is just the inverse of specialization and both processes can be applied (almost interchangeably) in designing the schema for some real-world scenario. Notice in the E-R diagram on page 5 that there is no difference specified between generalization and specialization other that how you view the picture (reading from the top down or from the bottom up).

Specialization vs. Generalization

- Differences in the two approaches are normally characterized by their starting points and overall goal:
- Specialization arises from a single entity set; it emphasizes differences among the entities within the set by creating distinct lower-level entity sets. These lower-level entity sets may have attributes or participate in relationships, that do not apply to all the entities in the higher-level entity set.
- In fact, the reason that a designer may need to use specialization is to represent such distinctive features of the real world scenario.
  - For example, if *instructor* and *student* neither have attributes that *person* entities do not have nor participate in relationships different than those in which *person* entities participate, there would be no need to specialize the *person* entity set.

Specialization vs. Generalization (cont.)

- Generalization arises from the recognition that a number of entity sets share some common characteristics (namely, they are described by the same attributes and participate in the same relationship sets).
- On the basis of these commonalities, generalization synthesizes these entity sets into a single, higher-level entity set.
- Generalization is used to emphasize the similarities among lower-level entity sets and to hide the differences. It also permits an economy of representation in that the shared attributes are not replicated.

Attribute Inheritance

- A crucial property of the higher and lower level entities that are created by specialization and generalization is *attribute inheritance*.
- The attributes of the higher-level entity sets are said to be *inherited* by the lower-level entity sets.
  - In our example above, *instructor* and *student* both inherit all the attributes of *person* (recall that *person* is the superclass for both *instructor* and *student*).
- A lower-level entity set (or subclass) also inherits participation in the relationship sets in which its higher-level entity set (its superclass) participates.
- A lower-level entity (subclass) inherits all attributes and relationships which belong to the higher-level entity set (superclass) which defines it.

Attribute Inheritance (cont.)

- Higher-level entity sets do not inherit any attribute or relationship which is defined within the lower-level entity set.
- Typically, what is developed will be a *hierarchy* of entity sets in which the highest-level entity appears at the top of the hierarchy.
- If, in such a hierarchy, a given entity set may be involved as a lower-level entity set in only one ISA relationship, then the inheritance is said to be *single-inheritance*.
- If, on the other hand, a given entity set is involved as a lower-level entity set in more than one ISA relationship, then the inheritance is said to be *multiple-inheritance* (then the resulting structure is called a *lattice*).

Constraints on Generalization

- In order to more accurately model a real-world situation, a data designer may choose to place constraints on a generalization (or specialization).

- The first type of constraint involves determining which entities can be members of a given lower-level entity set. This membership can be defined in one of the following two ways:

Predicate-defined: In predicate-defined lower-level entity sets, membership is evaluated on the basis of whether or not an entity satisfies an explicit predicate (a condition).

  – For example, assume that the higher-level entity set *account* has the attribute *account-type*. All account entities are evaluated on the defining *account-type* attribute. Only those entities that satisfy the predicate *account-type* = "*savings account*" would be allowed to belong to the lower-level entity set *savings-account*. Since all the lower-level entities are evaluated on the basis of the same attribute, this type of generalization is said to be *attribute-defined*.

Constraints on Generalization (cont.)

User-defined: User-defined lower-level entity sets are not constrained by a membership condition; rather, the database user assigns entities to a given entity set.

– For instance, suppose that after working 3 months at a bank, the employee is assigned to one of five different work groups. The teams would be represented as five lower-level entity sets of the higher-level entity set *employee*. A given employee is not assigned to a specific work group automatically on the basis of an explicit defining condition. Instead, the user responsible for making the group assignment does so on an individual basis, which may be arbitrary.

Constraints on Generalization (cont.)

- A second type of generalization constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization. The lower-level entity sets may be one of the following:

Disjoint: A disjointness constraint requires that an entity belong to no more than one lower-level entity set. In the example from above, an *account* entity can satisfy only one condition for the *account-type* attribute at any given time.

  – For example, an account-type might be either a checking account or a savings account, but it cannot be both.

Constraints on Generalization (cont.)

Overlapping: In *overlapping generalizations*, the same entity may belong to more than one lower-level entity set within a single generalization. For example, consider the banking work group from the previous section. Suppose that certain managers may participate in more than one work team. A given employee (a manager) may therefore appear in more than one of the group entity sets that are lower-level entity sets of *employee*.

- Note: lower-level entity overlap is the default case; a disjointness constraint must be placed explicitly on a generalization (or specialization). Within the E-R model a disjointness constraint is modeled by placing the word "disjoint" next to the triangle symbol as shown in the example below. The meaning of this diagram should now be clear: employees and customers are specializations of the set persons and the disjointness constraint implies that an employee is not also a customer. If the disjoint constraint is removed, then it is possible for an employee to also be a customer (or viewed from the other direction, it is possible for a person to be both a customer as well as an employee).

Constraints on Generalization (cont.)

- A final type of constraint, the *completeness constraint* on a generalization or specialization, specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization. This type of constraint can assume one of the following two forms:

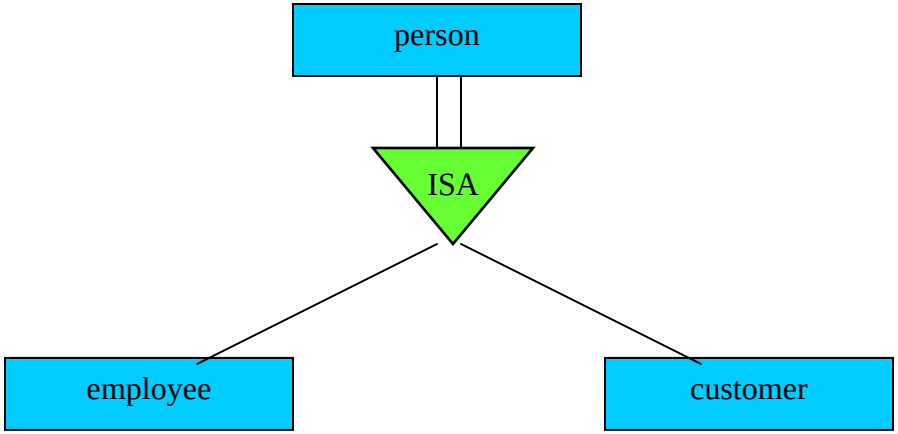
Total generalization/specialization: Each higher-level entity must belong to a lower-level entity.

Partial generalization/specialization: Some higher-level entities may not belong to any lower-level entity set.

  - Partial generalization is the default case. (Recall that total participation in a relationship is represented in the E-R model by a double line – so too will it be used to represent a total generalization. In the example shown below the generalization is total and overlapping which means that every person must appear as either an employee or a customer and it is possible for a person to be both.

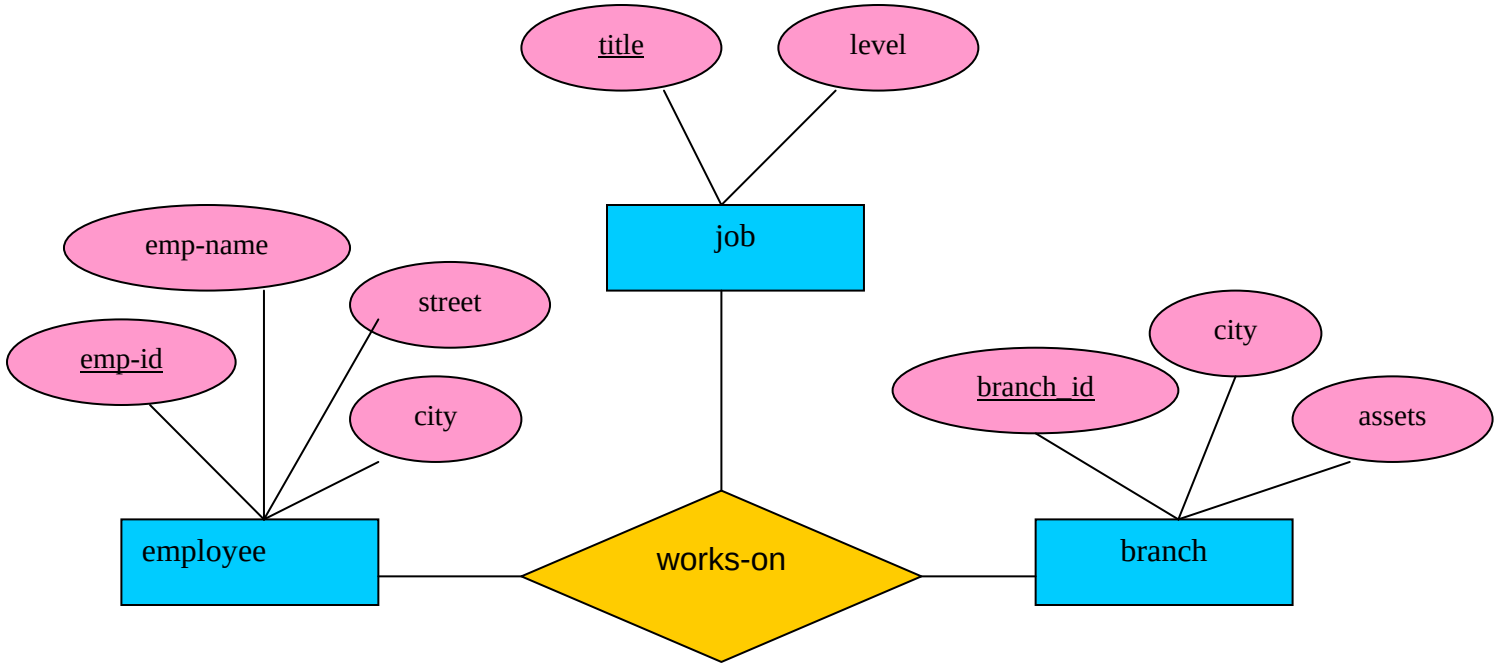
Example ERDs with Constraints

A total overlapping generalization/specialization



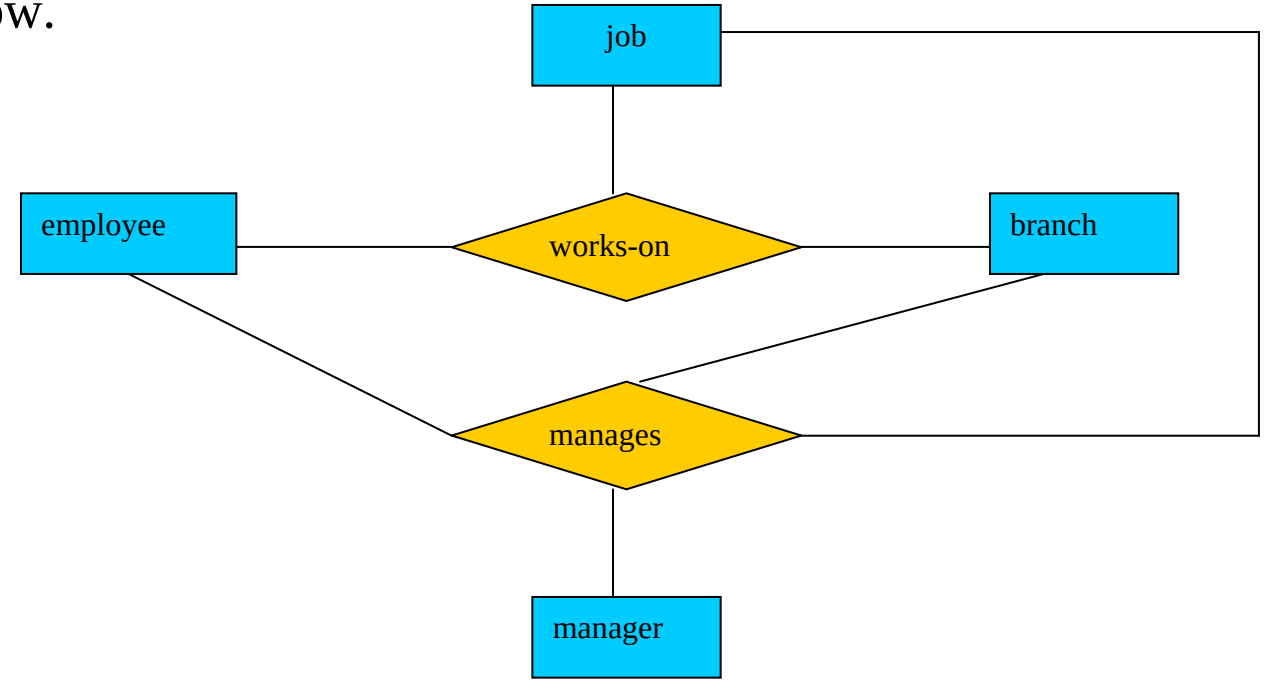
Aggregation

- One of the limitations of the E-R model is that it cannot express relationships among relationships. To understand why this is important consider the ternary relationship (3-way relationship) *works-on* between *employee*, *branch*, and *job* shown in the following E-R diagram.



Aggregation (cont.)

- Given this scenario, now suppose that we want to record the managers for tasks performed by an employee at a branch office; that is, we want to keep track of managers for *(employee*, *branch*, *job)* combinations. Let's assume that there is an entity set *manager*.
- One way to handle this is to create a quaternary relationship as shown below.



Aggregation (cont.)

Question: Why wouldn't a binary relationship between *manager* and *employee* work?

Answer:

A binary relationship would not permit us to represent which (*branch*, *job*) combinations of an employee are managed by which manager.

Aggregation (cont.)

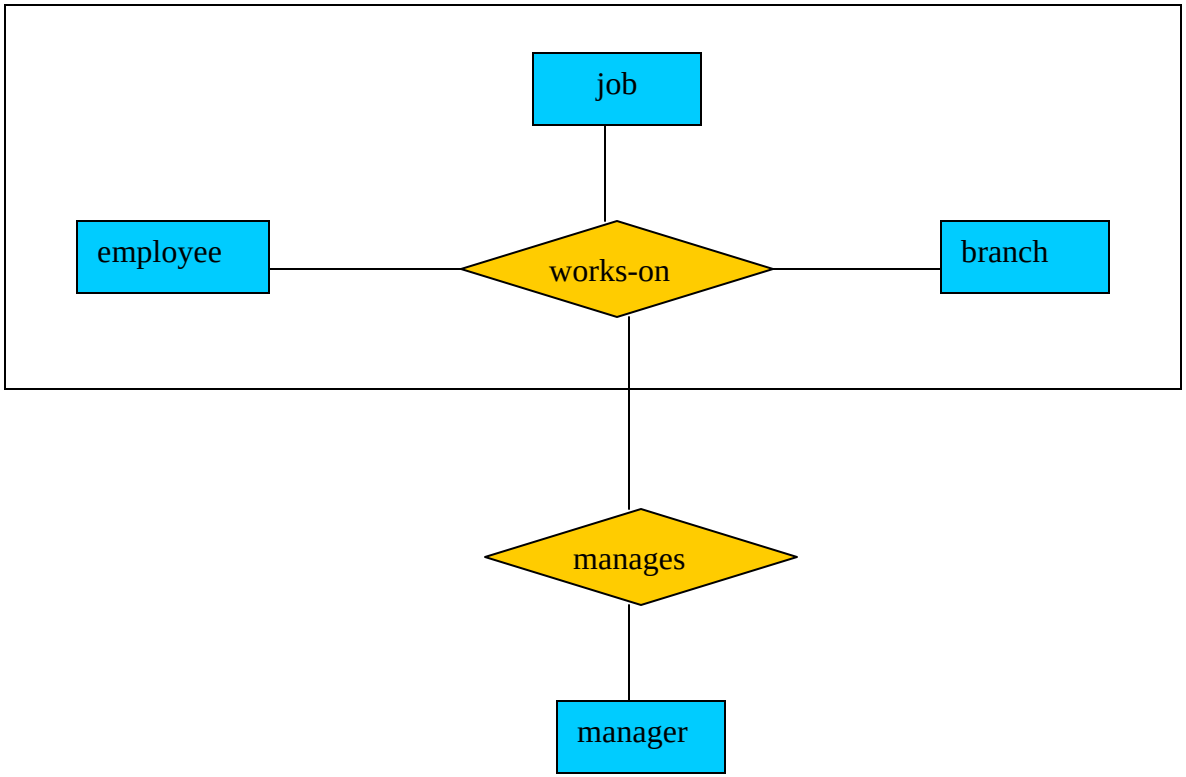
- When you look at the E-R diagram which models this situation, it would appear that the relationships sets *works-on* and *manages* could be combined into a single relationship set. However, we cannot do this since some *employee*, *branch*, *job* combinations may not have a manager.

- There is clearly redundant information in this figure, however, since every *employee*, *branch*, *job* combination in *manages* is also in *works-on*. If the manager were a value rather than an entity, we could make *manager* a multi-valued attribute of the relationship *works-on*. However, doing this would make it more difficult (both logically as well as in execution cost) to find, for example, employee-branch-job triples for which the manager is responsible. However, this option is not available in any case since the manager is a *manager* entity.

Aggregation (cont.)

- The best way to model this type of situation is to use *aggregation*.
- Aggregation is an abstraction through which relationships are treated as higher-level entities.
- Thus, in our example, we would regard the relationship set *works-on* (relating the entity sets *employee*, *branch*, and *job*) as a higher-level entity set called *works-on*. Such an entity set is treated in the same manner as any other entity set. We can then create a binary relationship *manages* between *works-on* and *manager* to represent who manages what tasks.
- The E-R diagram in the next slide illustrates how aggregation is represented in the E-R model.

Aggregation (cont.)

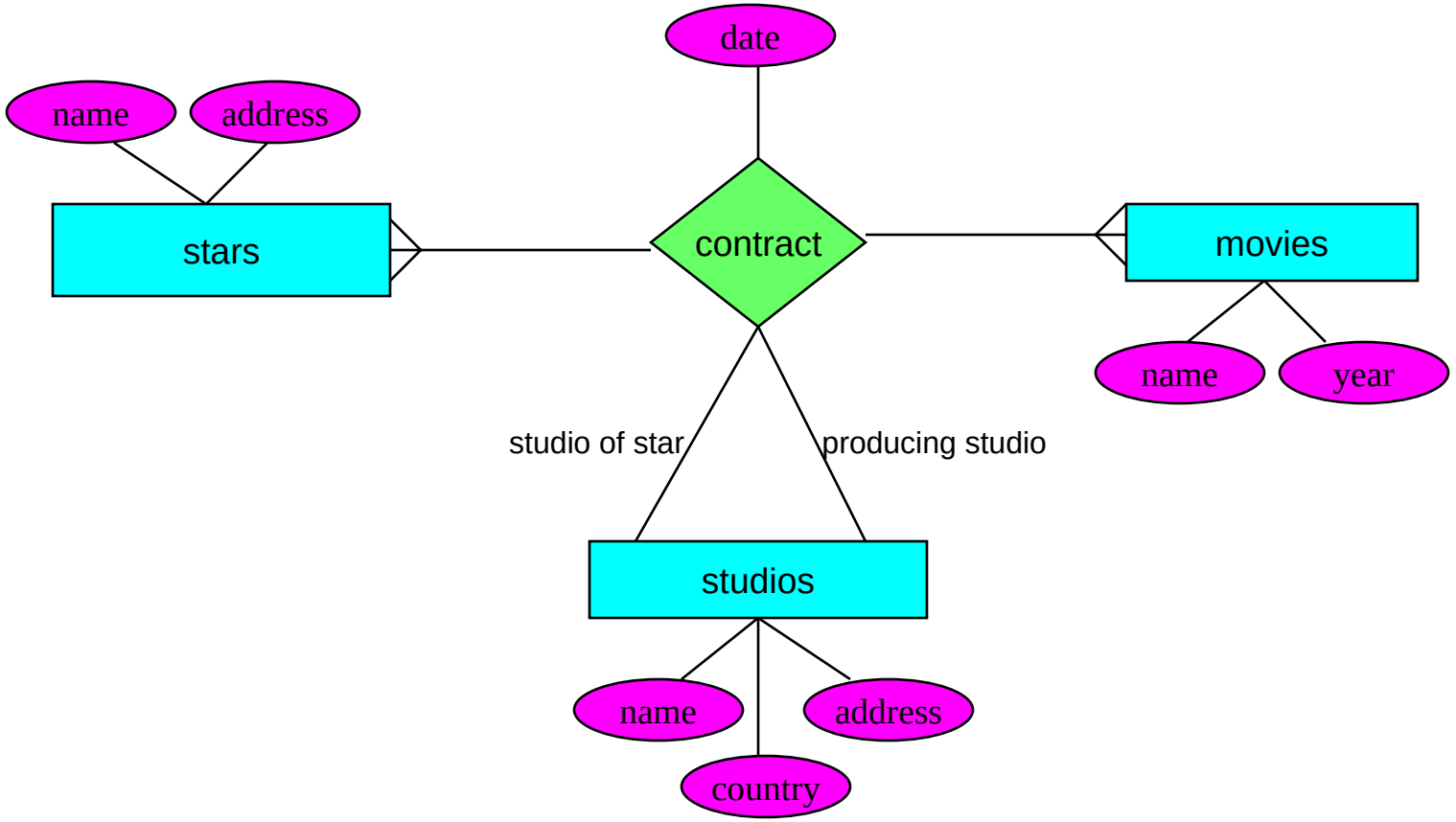


ERD illustrating aggregation

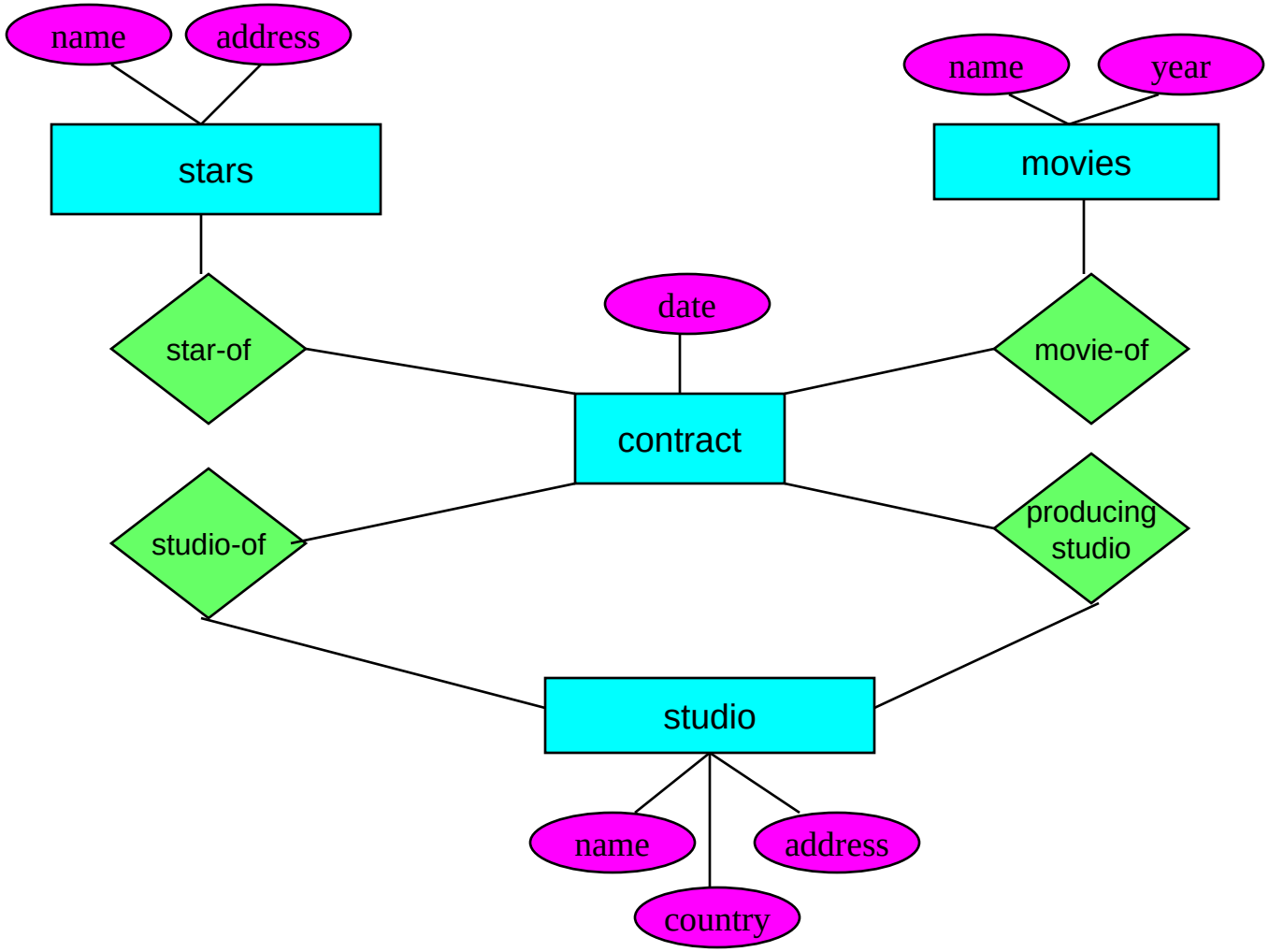
Multiway Relationships

- Most of the relationships that we have examined so far have been binary relationships, i.e., those relationships involving two entity sets.
- Any relationship involving more than two entity sets can be converted to a collection of binary, many-to-one relationships.
  - This is useful because, while the E-R model does not limit relationships to binary, many data models do, such as the Object Definition Language.
- To illustrate the conversion of a multiway relationship into a collection of binary relationships, consider the example E-R diagram on the next page.

Multiway Relationships (cont.)

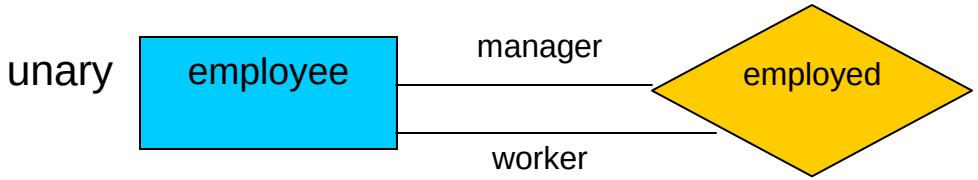


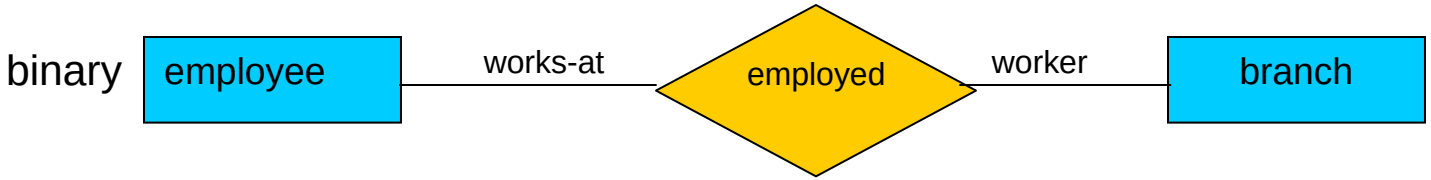
Multiway Relationship Converted to a Collection of Binary Relationships



E-R Diagrams with Role Indicators

- Roles in an E-R diagram are indicated by labeling the lines that connect entity sets to relationship sets.
- Roles can be identified for unary (recursive), binary, and nonbinary relationships.



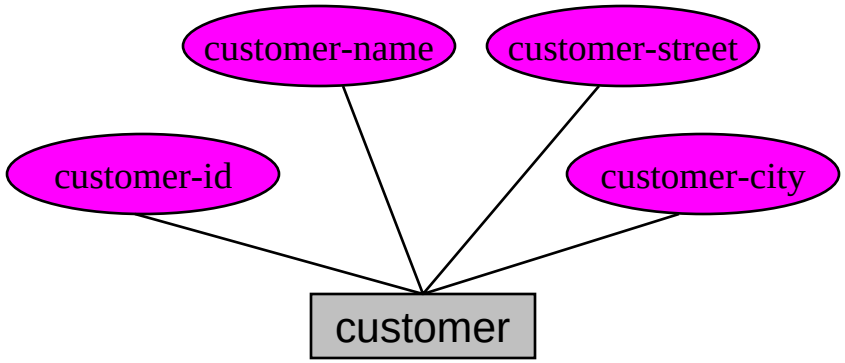


The Unified Modeling Language (UML) (cont.)

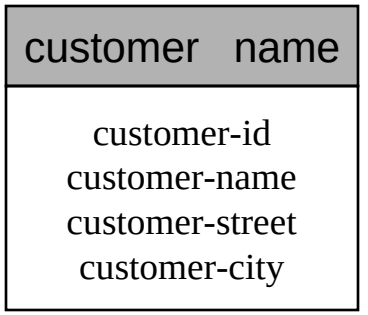
- Some of the parts of UML are:
  1. Class diagram. A class diagram is similar to an E-R diagram. We'll see the correspondence between them shortly.
  2. Use case diagram. Use case diagrams show the interaction between users and the system, in particular the steps of tasks that users perform (such as withdrawing money from a bank account or registering for a course).
  3. Activity diagram. Activity diagrams depict the flow of tasks between various components of the system.
  4. Implementation diagram. Implementation diagrams show the system components and their interconnections, both at the software component level and the hardware component level.

Correspondence of E-R & UML Class Diagrams

Entity sets and attributes



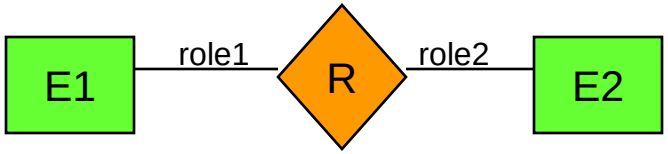
E-R Diagram

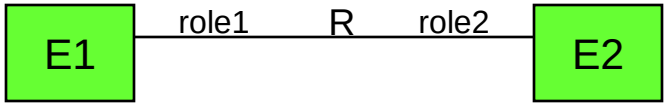


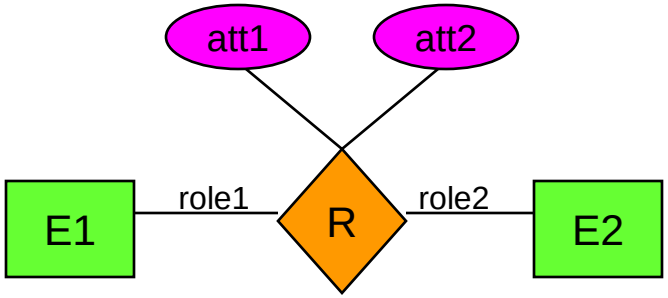
UML Class Diagram

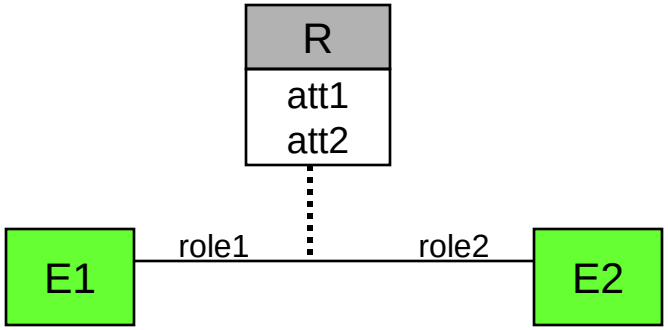
Correspondence of E-R & UML Class Diagrams (cont.)

Relationships







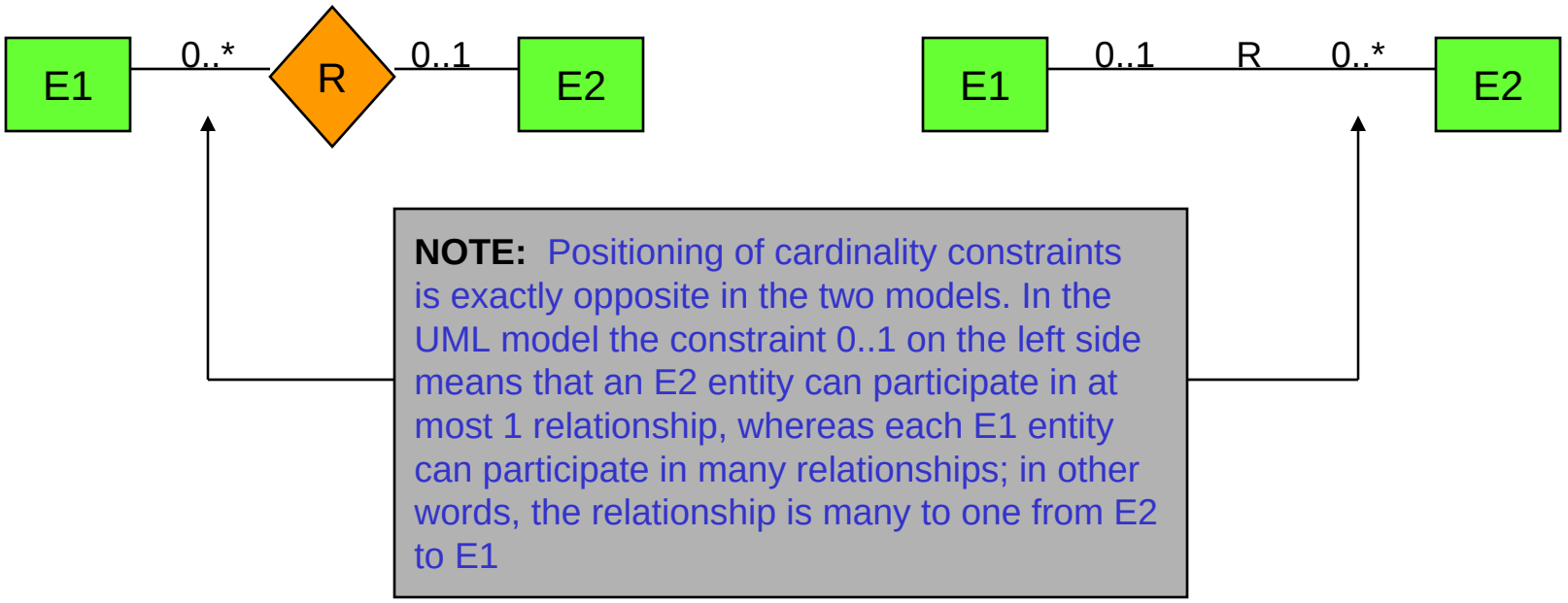


E-R Diagrams

UML Class Diagrams

Correspondence of E-R & UML Diagrams (cont.)

Cardinality Constraints

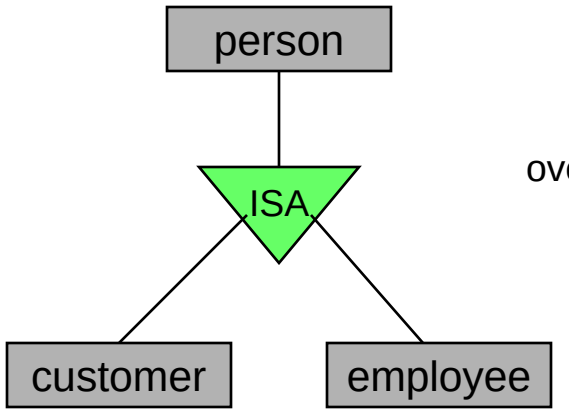


E-R Diagrams

UML Diagrams

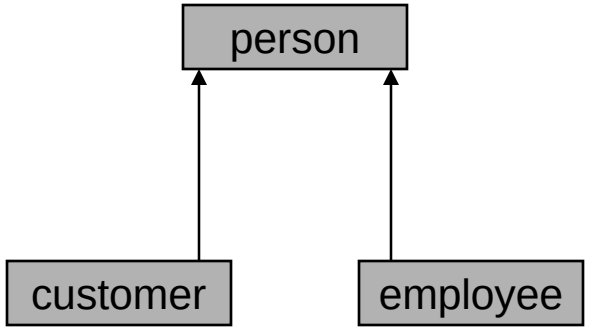
Correspondence of E-R & UML Class Diagrams (cont.)

Generalization & Specialization



E-R Diagrams

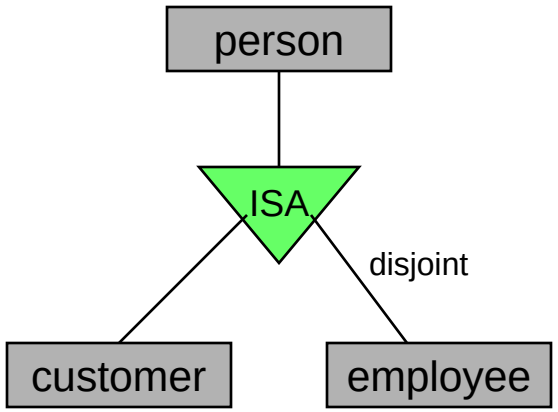
overlapping generalization



UML Class Diagrams

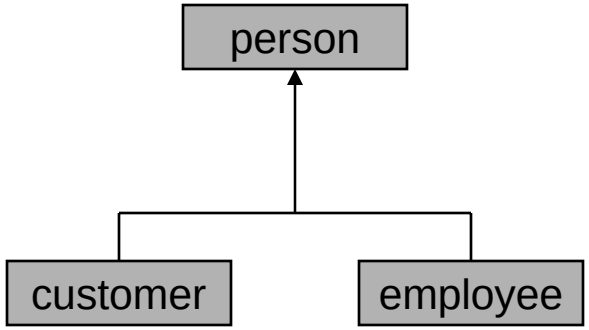
Correspondence of E-R & UML Class Diagrams (cont.)

Generalization & Specialization



E-R Diagrams

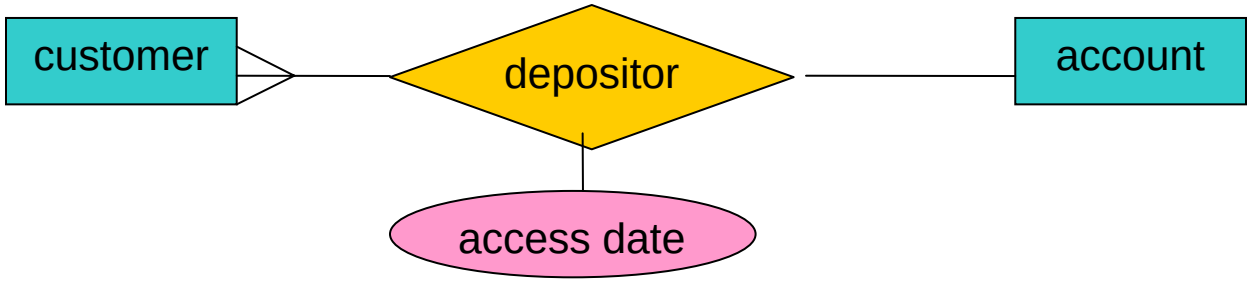
disjoint generalization



UML Class Diagrams

Referential Integrity Constraints

- Referential integrity constraints can be as simple as asserting that a given attribute have a non-null, single value. However, referential integrity constraints most commonly refer to the relationships among entity sets.
- Let's again consider our banking example and the many-to-one relationship between customer and account as shown below:



Referential Integrity Constraints (cont.)

- The many-to-one relationship depositor simply says that no account can be deposited into by more than one customer (and also that a customer can deposit into many different accounts).
- More importantly, it does **not** say that an account must be deposited into by a customer, nor does it say that a customer must make a deposit into an account. Further, it does not say that if an account is deposited into by a customer that the customer be present in the database!
- A referential integrity constraint requires that each entity "referenced" by the relationship must exist in the database.
- There are several methods which can be used to enforce referential integrity constraints:

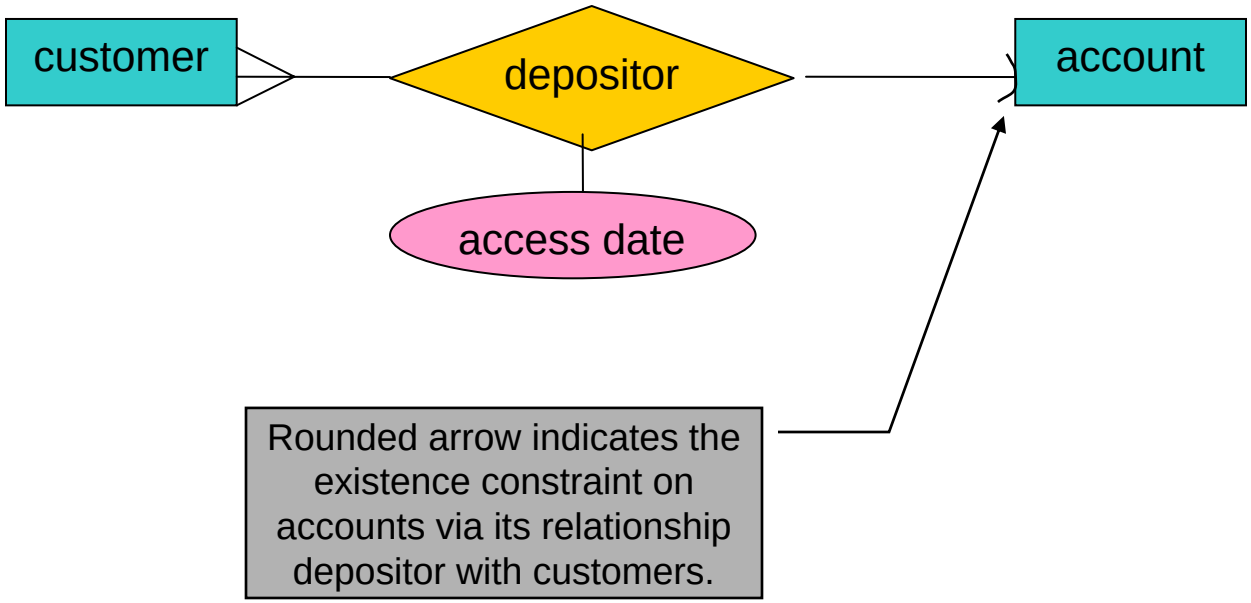
Referential Integrity Constraints (cont.)

1. Deletion of a referenced entity is not allowed. In other words, if Kristi makes a deposit into account number 456, then subsequently we cannot delete either the information concerning either Kristi or account 456.

2. If a referenced entity is deleted, then all entries that reference the deleted entity also be deleted. In other words, if we delete the information on Kristi, then we must delete all account information for accounts that she (alone) has deposited into. Notice in the specific example we are considering, that the relationship is M:1 which means that if Kristi has deposited into an account, she will be the only customer to do so. This will not be the case for a M:M relationship however.

- Referential integrity constraints can be modeled in the E-R model. Typically, they are depicted with a curved arrow as shown on the next page.

Referential Integrity Constraints (cont.)



The Relational Data Model

- The relational data model is based on the concept of mathematical relations.
- Codd (the guy who proposed the relational model) was a trained mathematician and he used terminology taken from this discipline, primarily set theory and predicate logic.

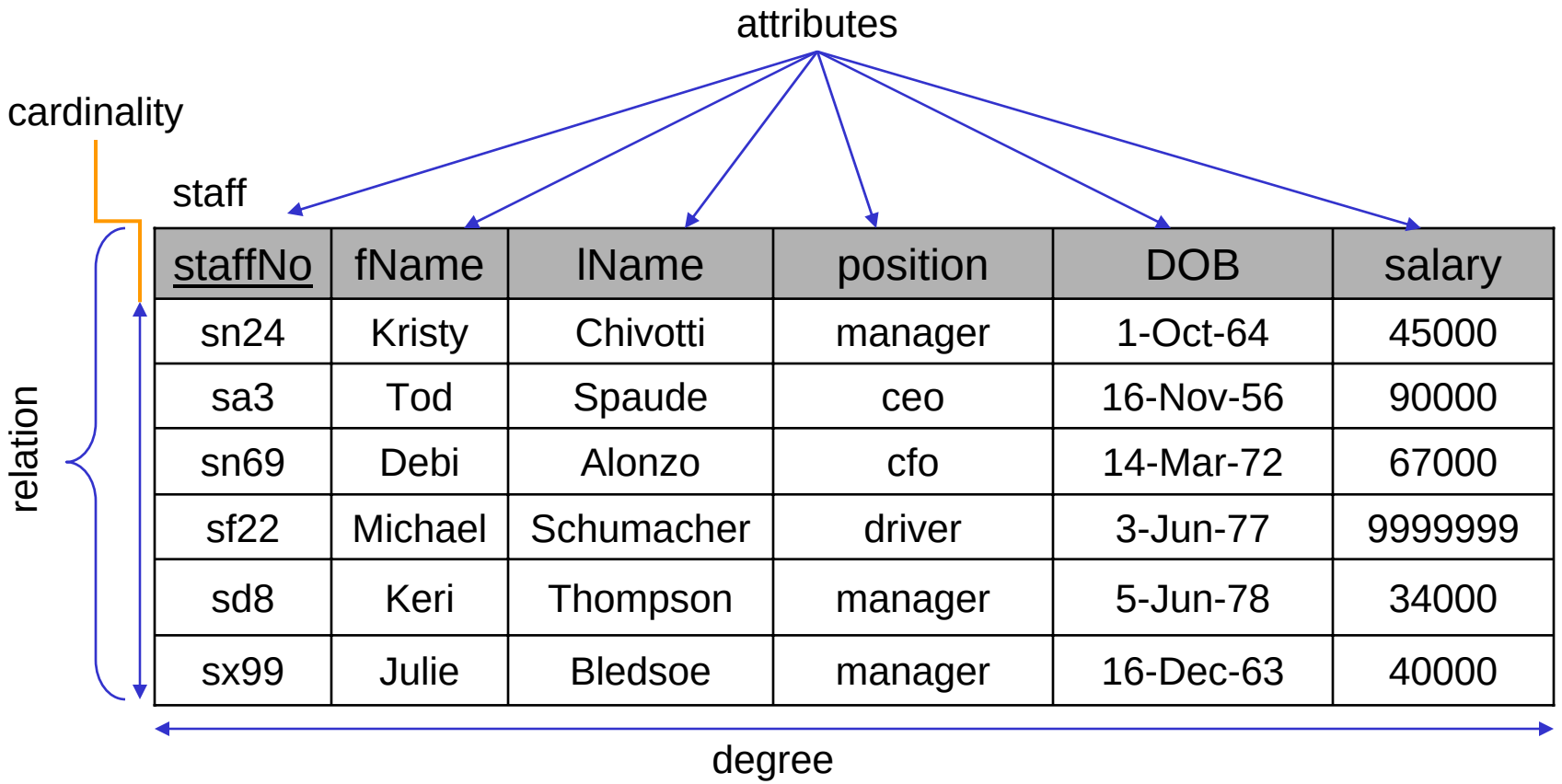
The Relational Data Model (cont.)

- Relation: A relation is a table (matrix) with rows and columns. Relations hold information about the objects modeled in the db.
- Attribute: An attribute is a named column of a relation. An attribute is some characteristic of an entity (or relationship) that is modeled in the database. Attributes can appear in any order in a relation.
- Domain: A domain is the set of allowable values for one or more attributes. Every attribute is defined on some domain. Domains may be distinct for each attribute, or two or more attributes may be defined on the same domain.

The Relational Data Model (cont.)

- Tuple: A tuple is a row of a relation. Tuples can appear in any order in a relation and the relation will remain the same, and therefore convey the same meaning.
- Degree: The degree of a relation is the number of attributes it contains.
- Cardinality: The cardinality of a relation is the number of tuples it contains.
- Relational database: A collection of normalized relations with distinct relation names.

An Example Relation



staff

staffNo	fName	lName	position	DOB	salary
sn24	Kristy	Chivotti	manager	1-Oct-64	45000
sa3	Tod	Spaude	ceo	16-Nov-56	90000
sn69	Debi	Alonzo	cfo	14-Mar-72	67000
sf22	Michael	Schumacher	driver	3-Jun-77	9999999
sd8	Keri	Thompson	manager	5-Jun-78	34000
sx99	Julie	Bledsoe	manager	16-Dec-63	40000

Example Domain Definitions

Attribute	Domain Name	Meaning	Domain Definition
staffNo	staffnumbers	set of all possible staff numbers	character: size 4, must begin with letter s.
fName, lName	name	set of all possible person names	character: size 20
DOB	date	date person was born	date: range from 1-Jan-20, format: dd-mmm-yy
salary	salaries	possible values of staff salaries	monetary: 7 digits, range 10,000-9,999,999
position	alljobs	set of all possible positions	select one from set: {ceo, cfo, coo,manager, asst. manager, driver, secretary}

Alternate Terminology for Relational Model

Formal Term	Alternative 1	Alternative 2
relation	table	file
tuple	row	record
attribute	column	field

What is a Relation

- To understand the true meaning of the term relation, we need to review some basic math concepts:
- Given two sets D_1 and D_2 where

  $D_1 = \{2, 4\}$ and $D_2 = \{1, 3, 5\}$
- The Cartesian product of these two sets, written $D_1 \times D_2$, is the set of all ordered pairs such that the first element is a member of D_1 and the second element is a member of D_2.
  - $D_1 \times D_2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$
- Any subset of this Cartesian product is a relation.
  - Thus, we could produce relation R such that: $R = \{(2, 3), (4, 3)\}$
- We can specify some condition which will select elements from $D_1 \times D_2$ to be included in R, such as:
  - $R = \{(x, y) \mid x \in D_1, y \in D_2, \text{and } y = 3\}$

What is a Relation (cont.)

- Given three sets D_1, D_2, and D_3 where

  $D_1 = \{2, 4\}$, $D_2 = \{1, 3\}$, and $D_3 = \{3, 6\}$

- The Cartesian product of three sets, written $D_1 \times D_2 \times D_3$, is the set of all ordered triples such that the first element is a member of D_1, the second element is a member of D_2, and the third element is a member of D_3.

  – $D_1 \times D_2 \times D_3 = \{(2, 1, 3), (2, 1, 6), (2, 3, 3), (2, 3, 6)$

  $(4, 1, 3), (4, 1, 6), (4, 3, 3), (4, 3, 6)\}$

- Any subset of this Cartesian product is a relation.

- In general, if D_1, D_2, ... D_n are *n* sets. Their Cartesian product is defined as: $D_1 \times D_2 \times \cdots \times D_n = \{(d_1, d_2, \cdots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \cdots, d_n \in D_n\}$

  and generally written as: $\mathop{X}_{i=1}^{n} D_i$

What is a Relation (cont.)

- A relational schema is a named relation defined by a set of attribute and domain name pairs.
  - $R_i = \{A_1:d_1, A_2:d_2, ..., A_n:d_n \mid d_1 \in D_1, d_2 \in D_2, ..., d_n \in D_n\}$
- A relational database schema is a set of relation schemas, each with a distinct name.
  - $R = \{R_1, R_2, ..., R_n\}$

What is a Relation (cont.)

A relation has the following properties:

1. The relation has a name that is distinct from all other relation names in the relational schema.
2. Each cell (attribute) contains exactly one atomic value.
3. Each attribute has a distinct name.
4. The values of an attribute are all from the same domain.
5. Each tuple is distinct; there are no duplicate tuples.
6. The order of the attributes has no significance.
7. The order of the tuples has not significance, theoretically. (However, in practice, the order may affect the efficiency of accessing tuples. Much more on this later.)

Relation Schemas vs. Relation Instances

- There is an important distinction to be made between a relation schema and a relation instance.
- The schema is the name and attributes for the relation and is relatively immutable.
- An instance is a set of tuples for that relation, and the instance may change frequently. Indeed most updates and certainly every insert and deletion will change the instance.
  - A snapshot database models the current "state" of the real world which is captured in the database. At any given moment in time it is modeling the current "instance" of the real world. If the real world state changes, so too must the database to maintain the representation of the current real world instance.

Equivalent Relations

A	B	C
1	2	3
3	2	1
4	4	1
2	1	3

a relation instance

B	C	A
2	3	1
2	1	3
4	1	4
1	3	2

a relation instance

A	B	C
4	4	1
3	2	1
1	2	3
2	1	3

a relation instance

equivalent relation instances

A	B	C
4	4	1
3	2	1
1	2	4
2	1	3

a relation instance

this relation instance is not equivalent to any of the other three