

# LECTURE 15 OF 42

## Normal Forms Notes: E-R and Exam 1 Review

Monday, 25 February 2008

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course page: <http://snipurl.com/va60>

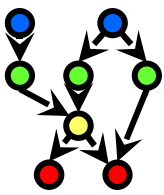
Course web site: <http://www.kddresearch.org/Courses/Spring-2008/CIS560>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

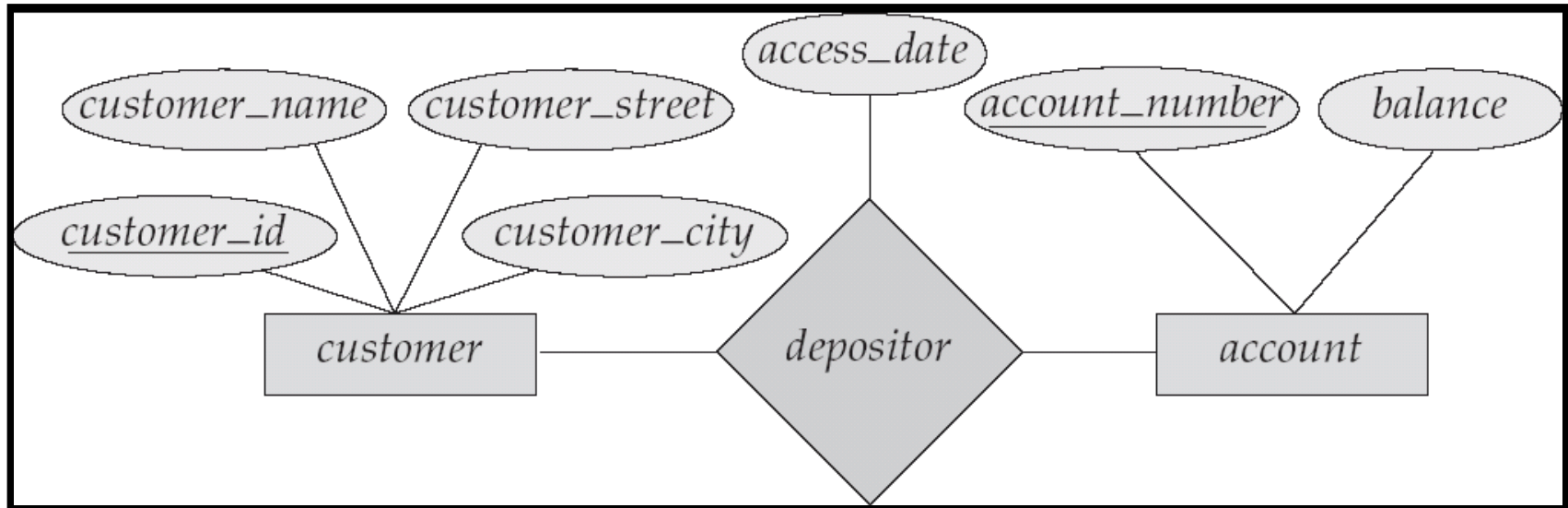
**Reading for Next Class:**

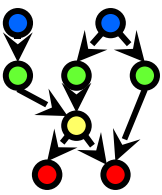
First half of Chapter 7, Silberschatz *et al.*, 5<sup>th</sup> edition





# RELATIONSHIP SETS WITH ATTRIBUTES: REVIEW

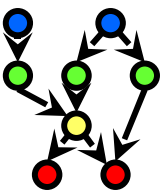




# WEAK ENTITY SETS: REVIEW

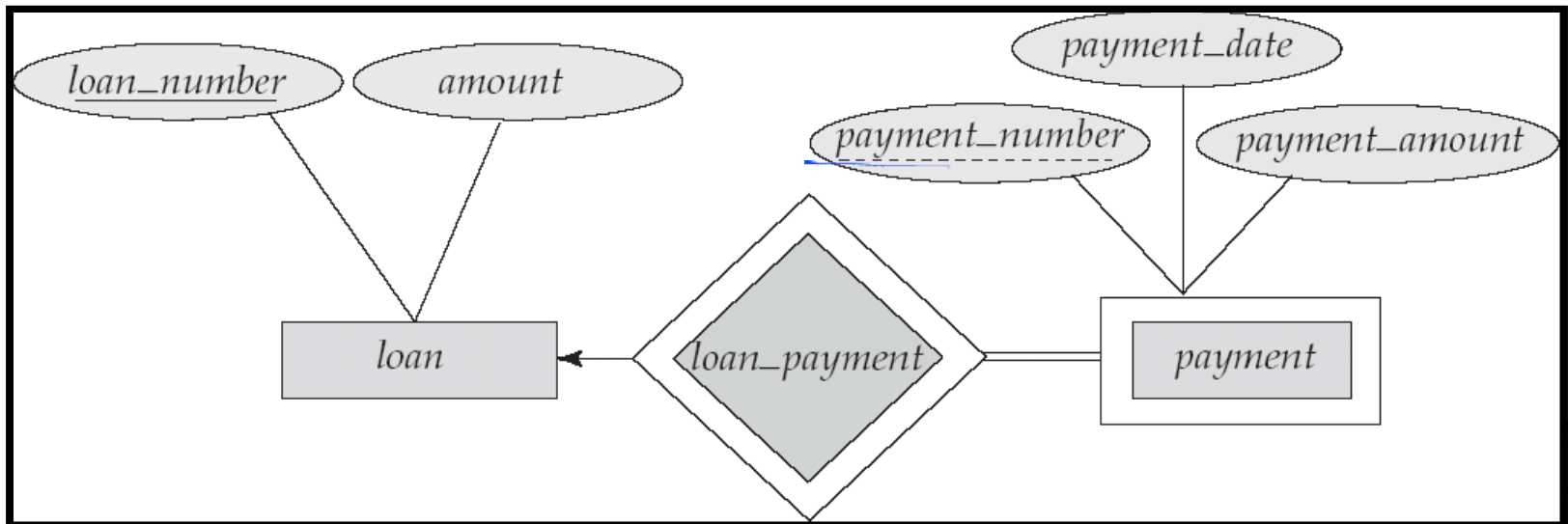
- An entity set that does not have a primary key is referred to as a **weak entity set**.
- The existence of a weak entity set depends on the existence of a **identifying entity set**
  - ★ it must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak entity set
  - ★ Identifying relationship depicted using a double diamond
- The **discriminator** (or *partial key*) of a weak entity set is the set of attributes that distinguishes 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.

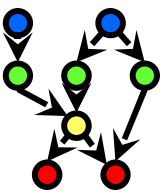




## WEAK ENTITY SETS (CONT.)

- We depict a weak entity set by double rectangles.
- We underline the discriminator of a weak entity set with a dashed line.
- `payment_number` – discriminator of the *payment* entity set
- Primary key for *payment* – (`loan_number`, `payment_number`)

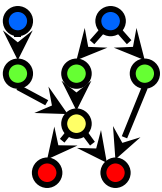




## WEAK ENTITY SETS (CONT.)

- 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 *loan\_number* were explicitly stored, *payment* could be made a strong entity, but then the relationship between *payment* and *loan* would be duplicated by an implicit relationship defined by the attribute *loan\_number* common to *payment* and *loan*



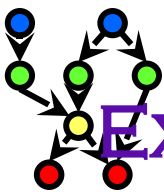


## MORE WEAK ENTITY SET EXAMPLES

- In a university, a *course* is a strong entity and a *course\_offering* can be modeled as a weak entity
- The discriminator of *course\_offering* would be *semester* (including year) and *section\_number* (if there is more than one section)
- If we model *course\_offering* as a strong entity we would model *course\_number* as an attribute.

Then the relationship with *course* would be implicit in the *course\_number* attribute

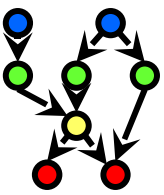




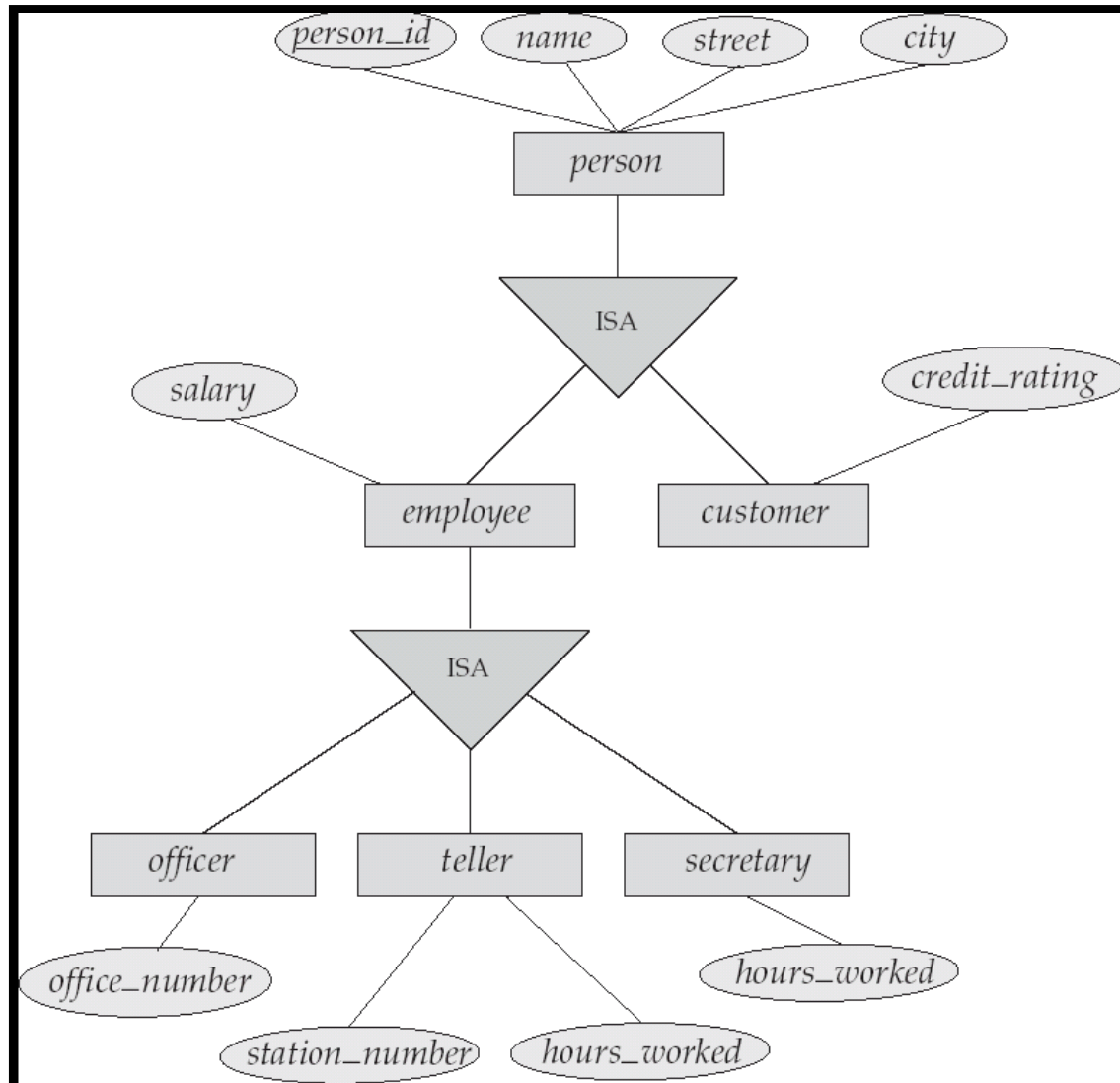
## EXTENDED E-R FEATURES: SPECIALIZATION

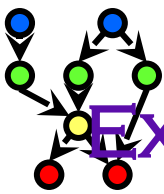
- Top-down design process; we designate subgroupings within an entity set that are distinctive from other entities in the set.
- These subgroupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (E.g. *customer* “is a” *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.





# SPECIALIZATION EXAMPLE

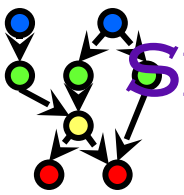




# EXTENDED ER FEATURES: GENERALIZATION

- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

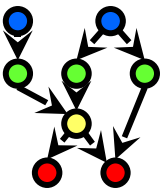




# SPECIALIZATION AND GENERALIZATION (CONT.)

- Can have multiple specializations of an entity set based on different features.
- E.g. *permanent\_employee* vs. *temporary\_employee*, in addition to *officer* vs. *secretary* vs. *teller*
- Each particular employee would be
  - ★ a member of one of *permanent\_employee* or *temporary\_employee*,
  - ★ and also a member of one of *officer*, *secretary*, or *teller*
- The ISA relationship also referred to as **superclass - subclass** relationship

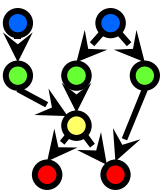




# DESIGN CONSTRAINTS ON A SPECIALIZATION/GENERALIZATION

- Constraint on which entities can be members of a given lower-level entity set.
  - ★ **condition-defined**
    - ⇒ Example: all customers over 65 years are members of *senior-citizen* entity set; *senior-citizen* ISA *person*.
  - ★ **user-defined**
- Constraint on whether or not entities may belong to more than one lower-level entity set within a single generalization.
  - ★ **Disjoint**
    - ⇒ an entity can belong to only one lower-level entity set
    - ⇒ Noted in E-R diagram by writing *disjoint* next to the ISA triangle
  - ★ **Overlapping**
    - ⇒ an entity can belong to more than one lower-level entity set

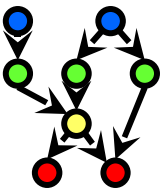




## DESIGN CONSTRAINTS ON A SPECIALIZATION/GENERALIZATION (CONT.)

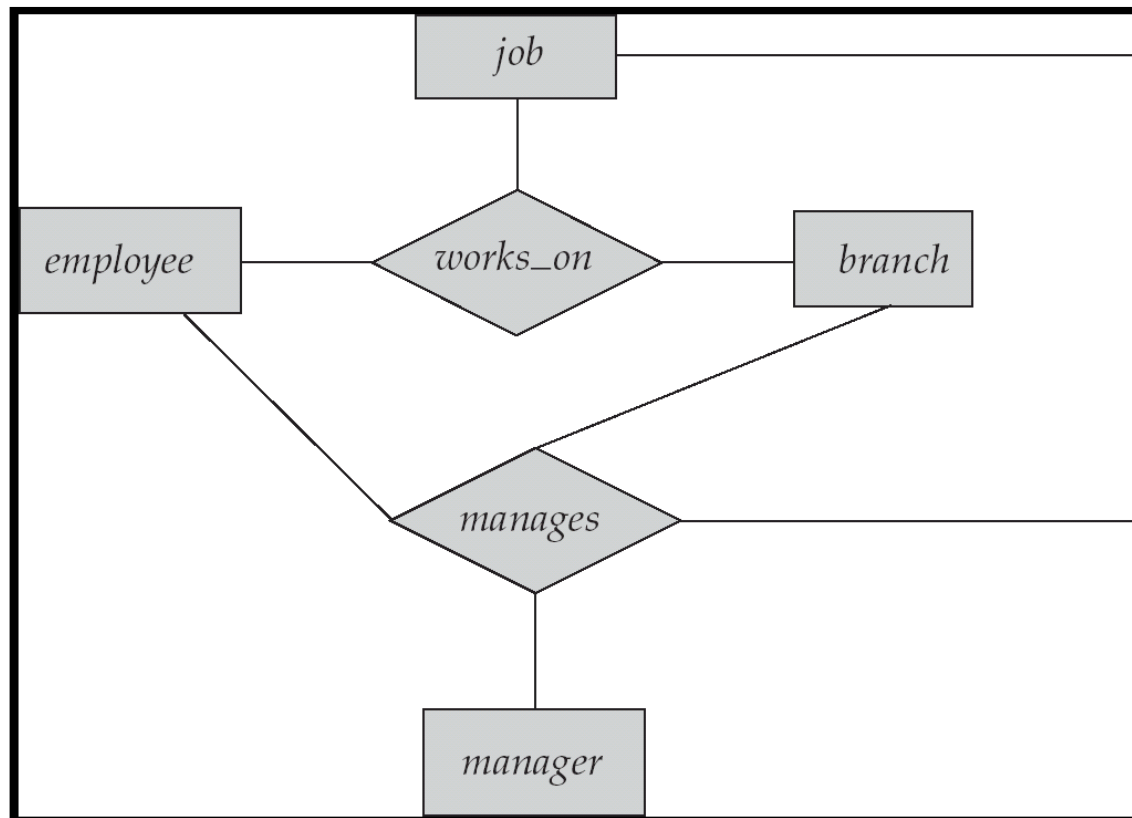
- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.
  - ★ **total** : an entity must belong to one of the lower-level entity sets
  - ★ **partial**: an entity need not belong to one of the lower-level entity sets

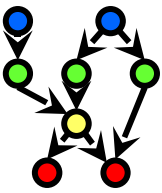




# AGGREGATION

- Consider the ternary relationship *works\_on*, which we saw earlier
- Suppose we want to record managers for tasks performed by an employee at a branch

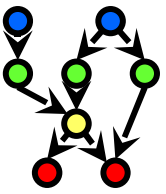




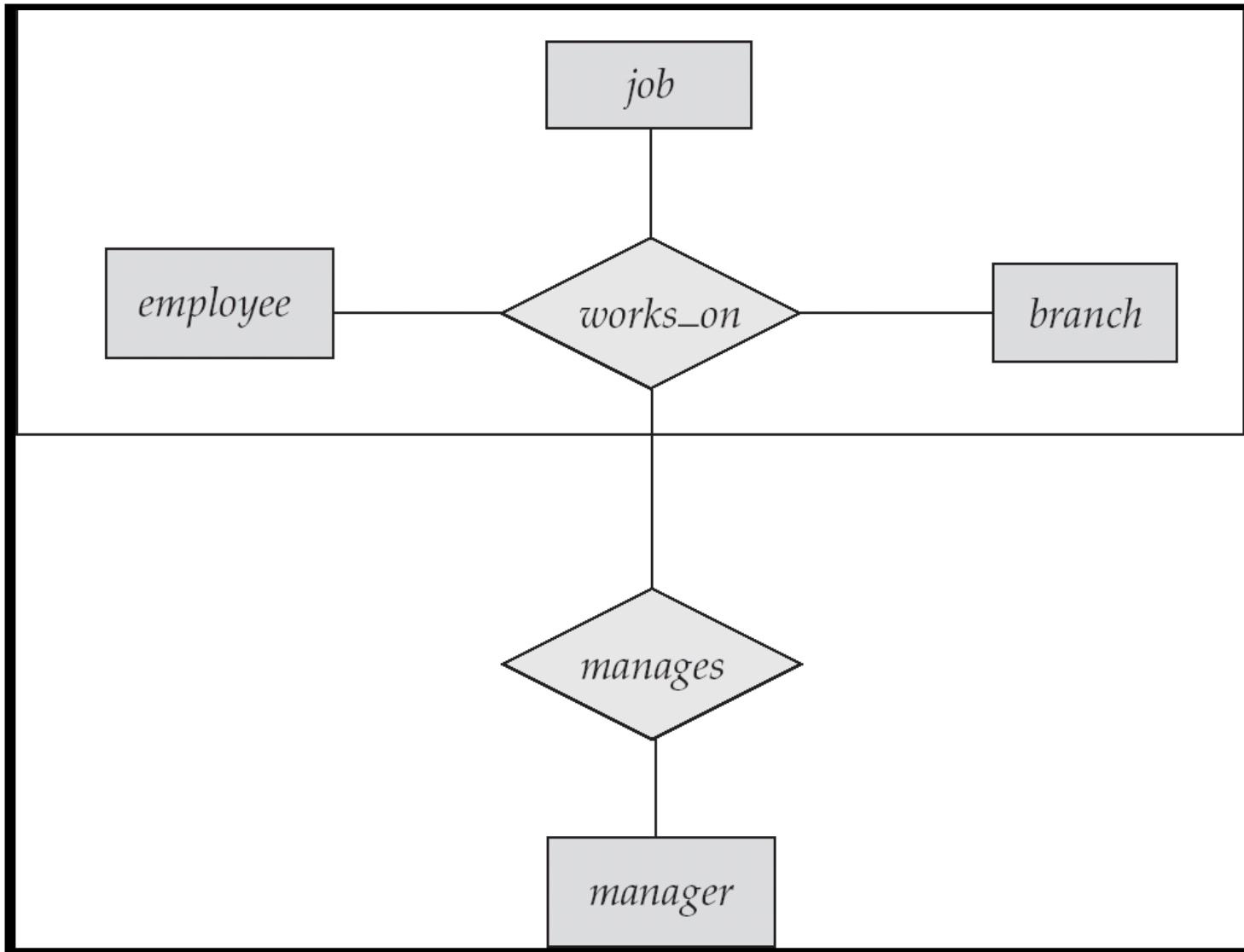
## AGGREGATION (CONT.)

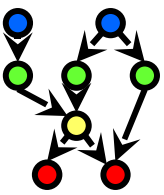
- Relationship sets *works\_on* and *manages* represent overlapping information
  - ★ Every *manages* relationship corresponds to a *works\_on* relationship
  - ★ However, some *works\_on* relationships may not correspond to any *manages* relationships
    - ⇒ So we can't discard the *works\_on* relationship
- Eliminate this redundancy via *aggregation*
  - ★ Treat relationship as an abstract entity
  - ★ Allows relationships between relationships
  - ★ Abstraction of relationship into new entity
- Without introducing redundancy, the following diagram represents:
  - ★ An employee works on a particular job at a particular branch
  - ★ An employee, branch, job combination may have an associated manager





# E-R DIAGRAM WITH AGGREGATION

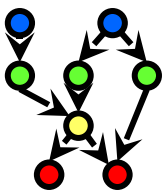




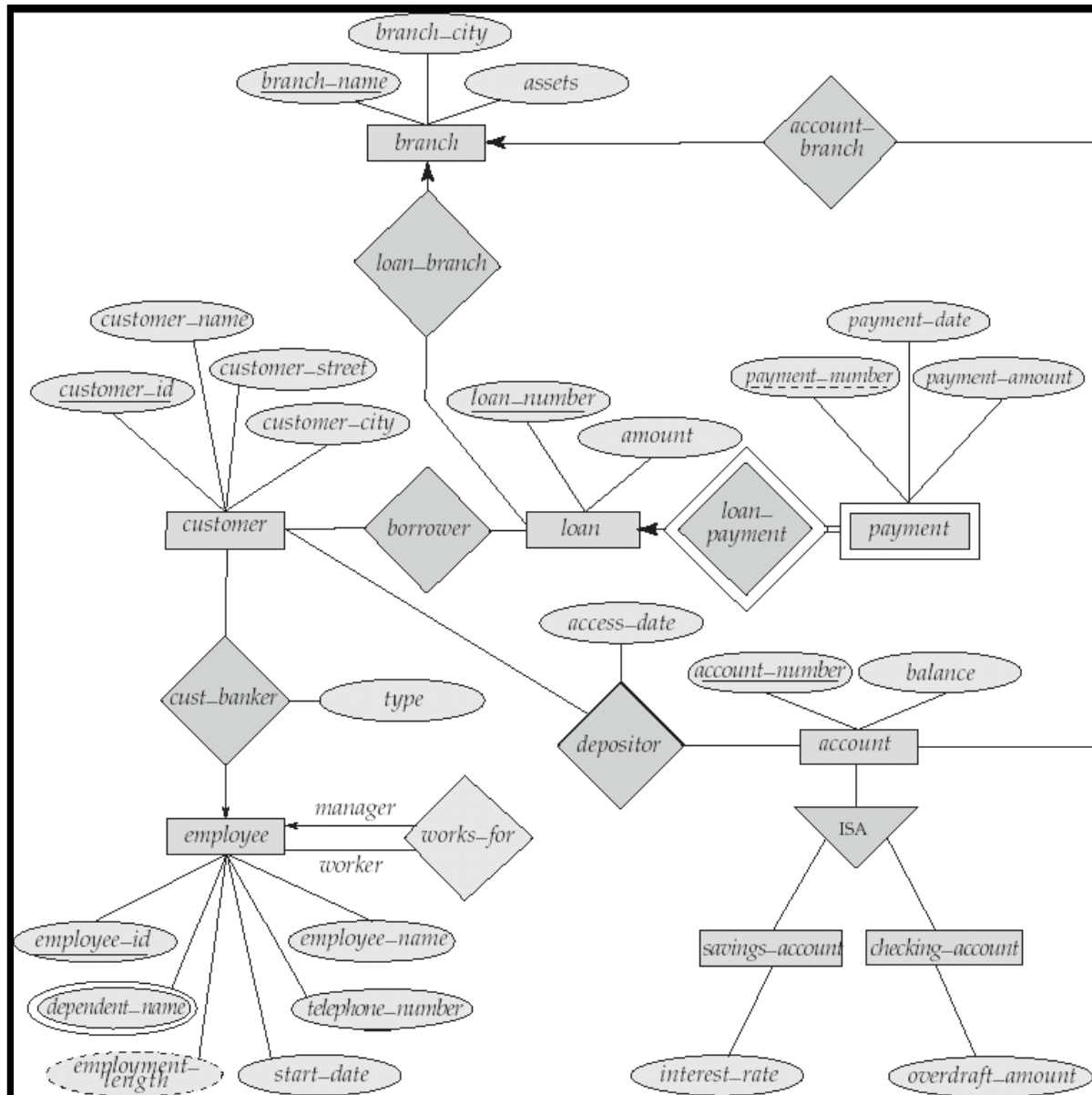
# E-R DESIGN DECISIONS

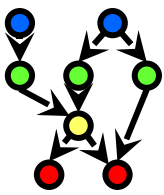
- The use of an attribute or entity set to represent an object.
- Whether a real-world concept is best expressed by an entity set or a relationship set.
- The use of a ternary relationship versus a pair of binary relationships.
- The use of a strong or weak entity set.
- The use of specialization/generalization – contributes to modularity in the design.
- The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure.



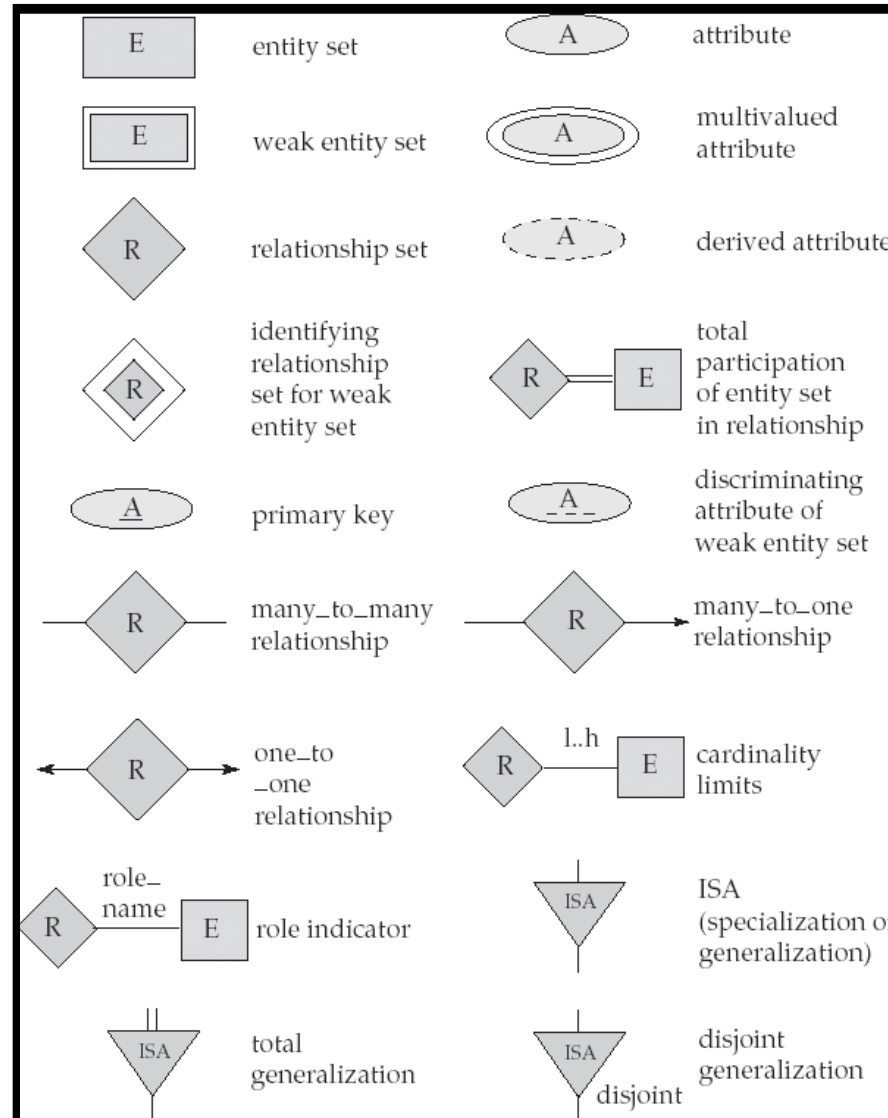


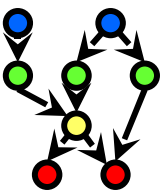
# E-R DIAGRAM FOR A BANKING ENTERPRISE



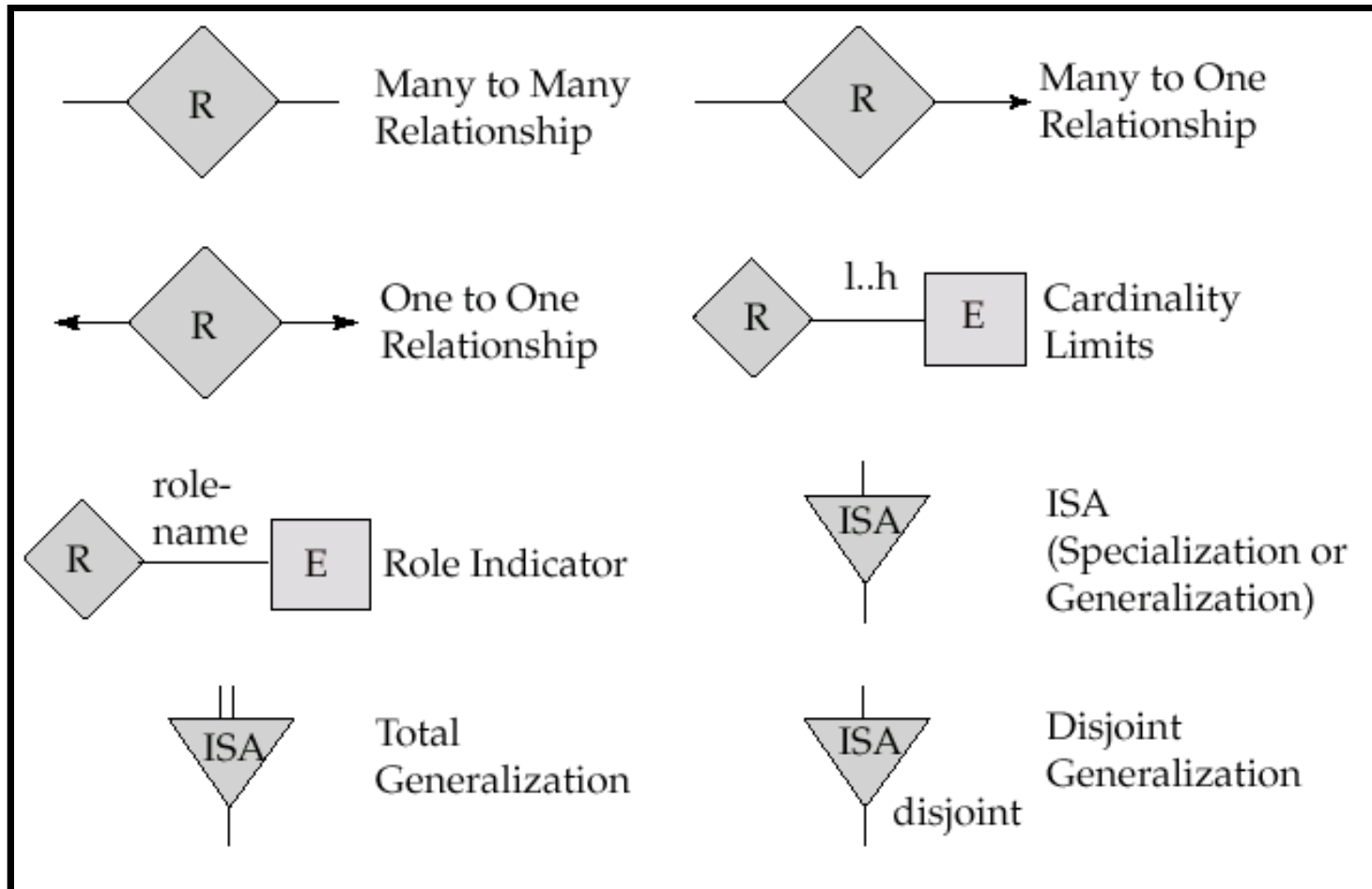


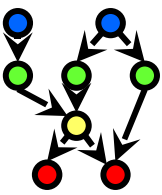
# SUMMARY OF SYMBOLS USED IN E-R NOTATION





# SUMMARY OF SYMBOLS (CONT.)

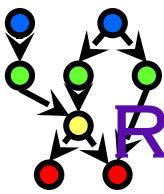




# REDUCTION TO RELATION SCHEMAS

- Primary keys allow entity sets and relationship sets to 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.





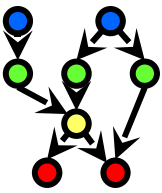
# REPRESENTING ENTITY SETS AS SCHEMAS

- A strong entity set reduces to a schema with the same attributes.
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set

*payment* =

( *loan\_number*, *payment\_number*, *payment\_date*,  
*payment\_amount* )

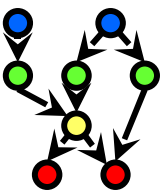




## REPRESENTING RELATIONSHIP SETS AS SCHEMAS

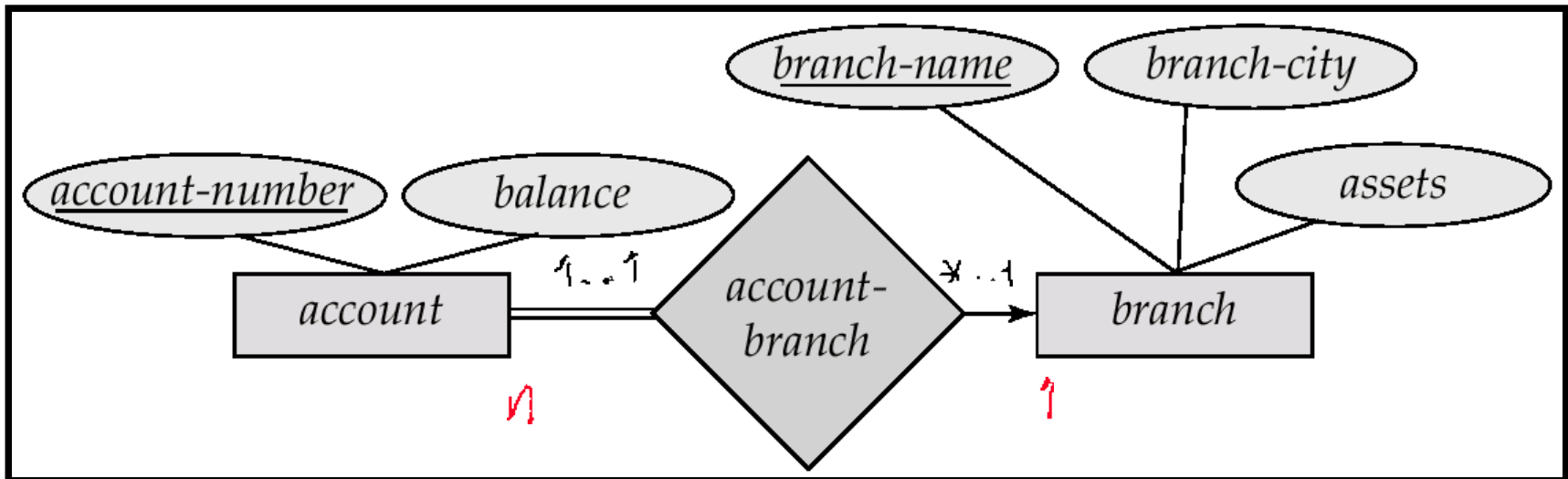
- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set borrower  
*borrower* = (customer\_id, loan\_number)

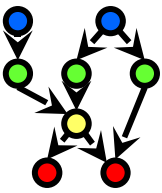




# REDUNDANCY OF SCHEMAS

- Many-to-one and one-to-many relationship sets that are 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 *account\_branch*, add an attribute *branch\_name* to the schema arising from entity set *account*

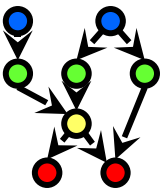




## REDUNDANCY OF SCHEMAS (CONT.)

- 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
- If participation is *partial* on the “many” side, replacing a schema by an extra attribute in the schema corresponding to the “many” side could result in null values
- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.
  - ★ Example: The *payment* schema already contains the attributes that would appear in the *loan\_payment* schema (i.e., *loan\_number* and *payment\_number*).

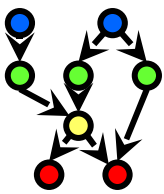




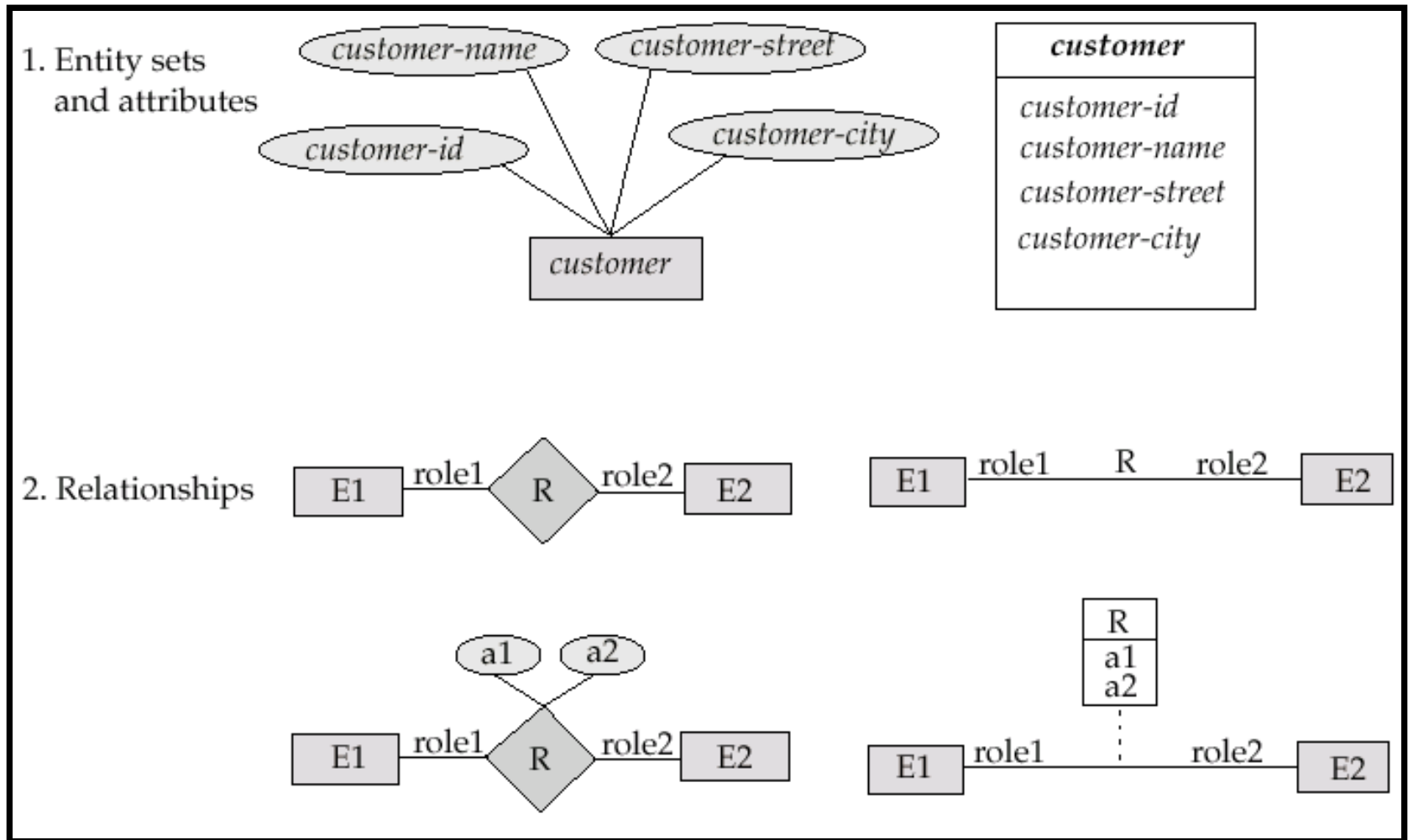
# UML

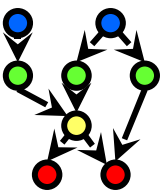
- **UML:** Unified Modeling Language
- UML has many components to graphically model different aspects of an entire software system
- UML Class Diagrams correspond to E-R Diagram, but several differences.





# SUMMARY OF UML CLASS DIAGRAM NOTATION

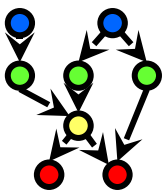




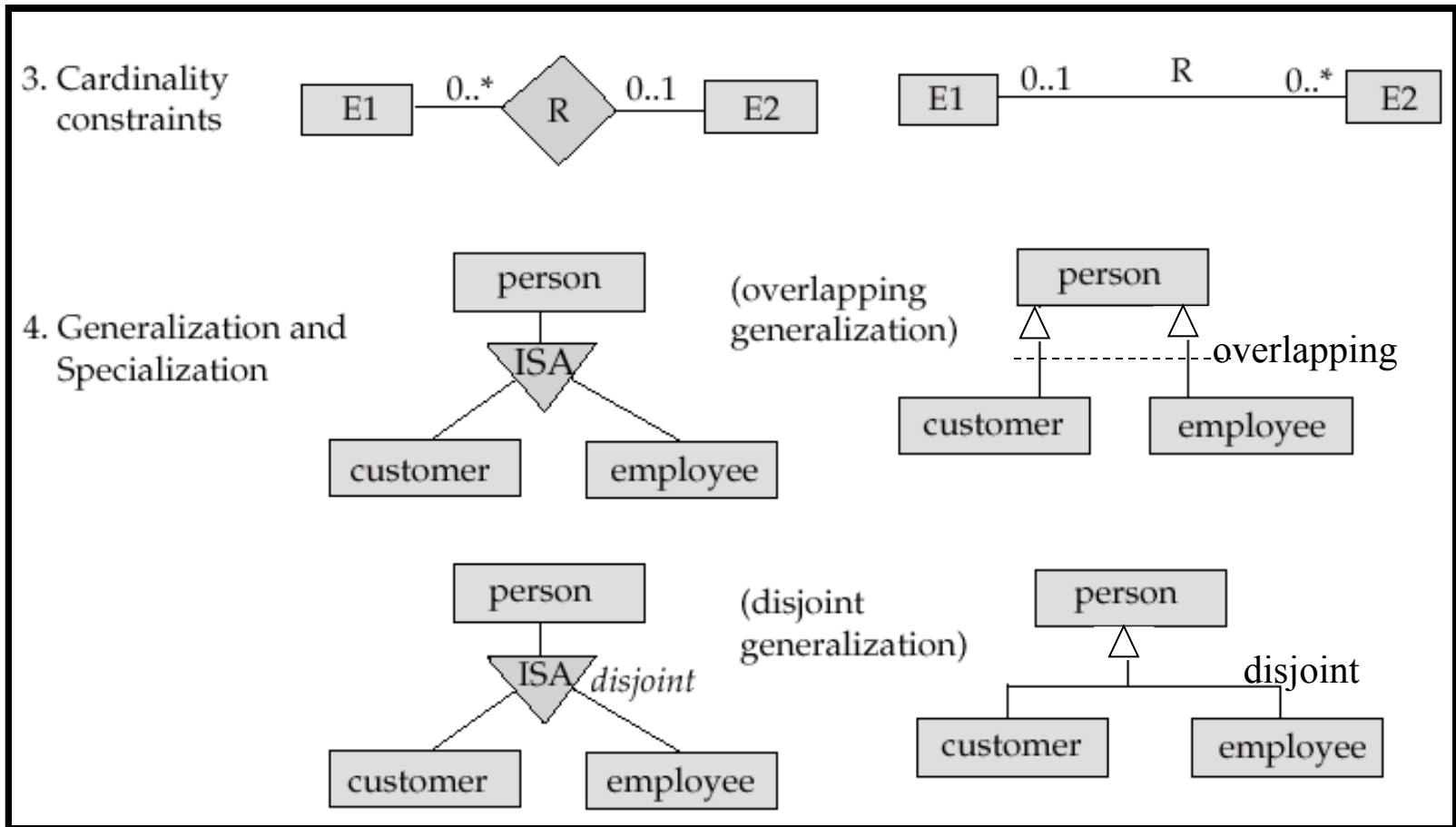
## UML CLASS DIAGRAMS (CONT.)

- Entity sets are shown as boxes, and attributes are shown within the box, rather than as separate ellipses in E-R diagrams.
- Binary relationship sets are represented in UML by just drawing a line connecting the entity sets. The relationship set name is written adjacent to the line.
- The role played by an entity set in a relationship set may also be specified by writing the role name on the line, adjacent to the entity set.
- The relationship set name may alternatively be written in a box, along with attributes of the relationship set, and the box is connected, using a dotted line, to the line depicting the relationship set.
- Non-binary relationships drawn using diamonds, just as in ER diagrams





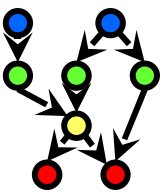
# UML CLASS DIAGRAM NOTATION (CONT.)



\*Note reversal of position in cardinality constraint depiction

\*Generalization can use merged or separate arrows independent of disjoint/overlapping

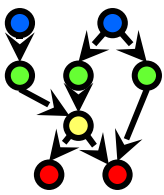




## UML CLASS DIAGRAMS (CONTD.)

- Cardinality constraints are specified in the form  $l..h$ , where  $l$  denotes the minimum and  $h$  the maximum number of relationships an entity can participate in.
- Beware: the positioning of the constraints is exactly the reverse of the positioning of constraints in E-R diagrams.
- The constraint  $0..*$  on the  $E2$  side and  $0..1$  on the  $E1$  side means that each  $E2$  entity can participate in at most one relationship, whereas each  $E1$  entity can participate in many relationships; in other words, the relationship is many to one from  $E2$  to  $E1$ .
- Single values, such as 1 or \* may be written on edges; The single value 1 on an edge is treated as equivalent to  $1..1$ , while \* is equivalent to  $0..*$ .

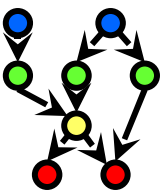




# CHAPTER 7: RELATIONAL DATABASE DESIGN

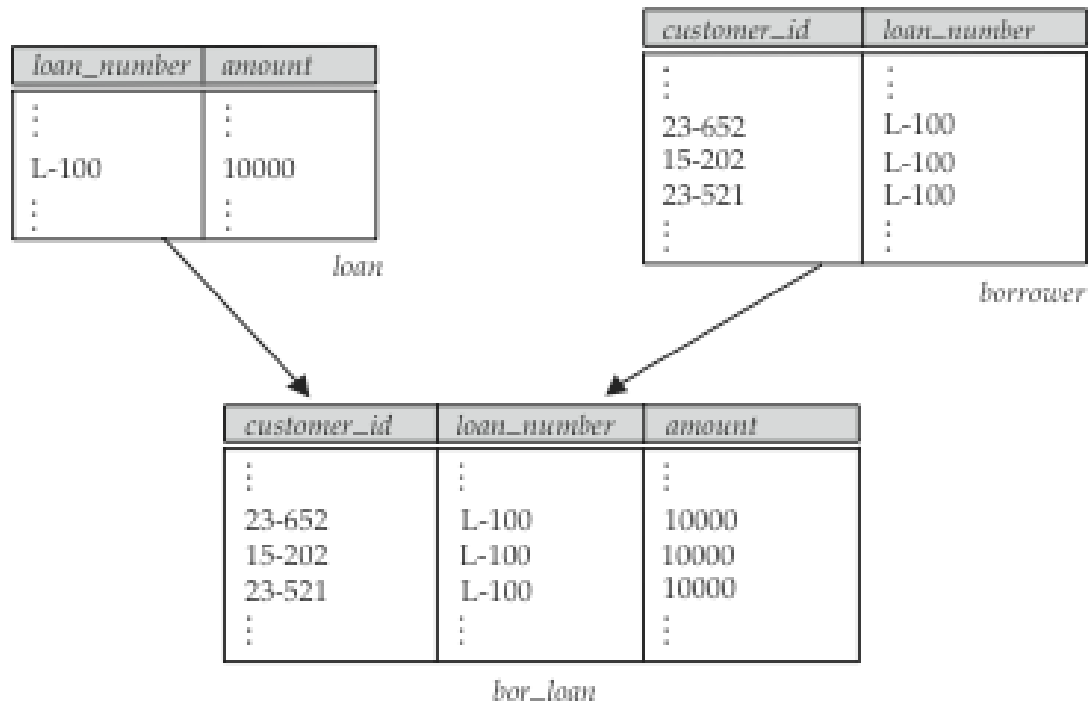
- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Algorithms for Functional Dependencies
- Decomposition Using Multivalued Dependencies
- More Normal Form
- Database-Design Process
- Modeling Temporal Data

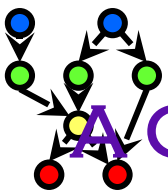




# COMBINE SCHEMAS?

- Suppose we combine *borrow* and *loan* to get  
*bor\_loan* = (*customer\_id*, *loan\_number*, *amount*)
- Result is possible repetition of information (L-100 in example below)





# A COMBINED SCHEMA WITHOUT REPETITION

- Consider combining *loan\_branch* and *loan*  
*loan\_amt\_br* = (*loan\_number*, *amount*, *branch\_name*)
- No repetition (as suggested by example below)

<i>loan_number</i>	<i>amount</i>
⋮	⋮
L-100	10000
⋮	⋮

<i>loan_number</i>	<i>branch_name</i>
⋮	⋮
L-100	Springfield
⋮	⋮

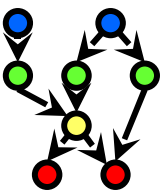
*loan*

*loan\_branch*

<i>loan_number</i>	<i>amount</i>	<i>branch_name</i>
⋮	⋮	⋮
L-100	10000	Springfield
⋮	⋮	⋮

*loan\_amt\_br*

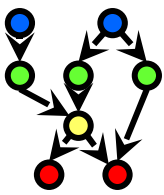




## WHAT ABOUT SMALLER SCHEMAS?

- Suppose we had started with *bor\_loan*. How would we know to split up (**decompose**) it into *borrower* and *loan*?
- Write a rule “if there were a schema (*loan\_number*, *amount*), then *loan\_number* would be a candidate key”
- Denote as a **functional dependency**:  
$$\textit{loan\_number} \rightarrow \textit{amount}$$
- In *bor\_loan*, because *loan\_number* is not a candidate key, the amount of a loan may have to be repeated. This indicates the need to decompose *bor\_loan*.
- Not all decompositions are good. Suppose we decompose *employee* into  
$$\textit{employee1} = (\textit{employee\_id}, \textit{employee\_name})$$
$$\textit{employee2} = (\textit{employee\_name}, \textit{telephone\_number}, \textit{start\_date})$$
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a lossy decomposition.





# A LOSSY DECOMPOSITION

<i>employee_id</i>	<i>employee_name</i>	<i>telephone_number</i>	<i>start_date</i>
⋮			
123-45-6789	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
⋮			

*employee*

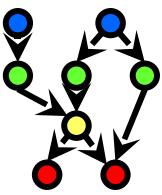
<i>employee_id</i>	<i>employee_name</i>
⋮	
123-45-6789	Kim
987-65-4321	Kim
⋮	

<i>employee_name</i>	<i>telephone_number</i>	<i>start_date</i>
⋮		
Kim	882-0000	1984-03-29
Kim	869-9999	1981-01-16
⋮		



<i>employee_id</i>	<i>employee_name</i>	<i>telephone_number</i>	<i>start_date</i>
⋮			
123-45-6789	Kim	882-0000	1984-03-29
123-45-6789	Kim	869-9999	1981-01-16
987-65-4321	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
⋮			

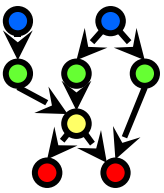




# FIRST NORMAL FORM

- Domain is atomic if its elements are considered to be indivisible units
  - ★ Examples of non-atomic domains:
    - ⇒ Set of names, composite attributes
    - ⇒ Identification numbers like CS101 that can be broken up into parts
- A relational schema  $R$  is in first normal form if the domains of all attributes of  $R$  are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - ★ Example: Set of accounts stored with each customer, and set of owners stored with each account
  - ★ We assume all relations are in first normal form (and revisit this in Chapter 9)

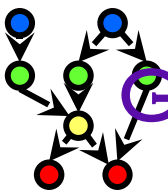




## FIRST NORMAL FORM (CONT'D)

- Atomicity is actually a property of how the elements of the domain are used.
  - ★ Example: Strings would normally be considered indivisible
  - ★ Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
  - ★ If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
  - ★ Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

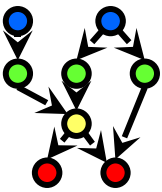




# GOAL – DEVISE A THEORY FOR THE FOLLOWING

- Decide whether a particular relation  $R$  is in “good” form.
- In the case that a relation  $R$  is not in “good” form, decompose it into a set of relations  $\{R_1, R_2, \dots, R_n\}$  such that
  - ★ each relation is in good form
  - ★ the decomposition is a lossless-join decomposition
- Our theory is based on:
  - ★ functional dependencies
  - ★ multivalued dependencies

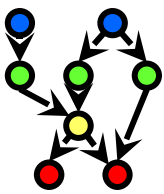




# FUNCTIONAL DEPENDENCIES

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key*.





## FUNCTIONAL DEPENDENCIES (CONT.)

- Let  $R$  be a relation schema

$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

- The functional dependency

$$\alpha \rightarrow \beta$$

holds on  $R$  if and only if for any legal relations  $r(R)$ , whenever any two tuples  $t_1$  and  $t_2$  of  $r$  agree on the attributes  $\alpha$ , they also agree on the attributes  $\beta$ . That is,

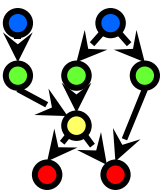
$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider  $r(A,B)$  with the following instance of  $r$ .

1	4
1	5
3	7

- On this instance,  $A \rightarrow B$  does **NOT** hold, but  $B \rightarrow A$  does hold.





## FUNCTIONAL DEPENDENCIES (CONT.)

- $K$  is a superkey for relation schema  $R$  if and only if  $K \rightarrow R$
- $K$  is a candidate key for  $R$  if and only if
  - ★  $K \rightarrow R$ , and
  - ★ for no  $\alpha \subset K$ ,  $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

*bor\_loan = (customer\_id, loan\_number, amount).*

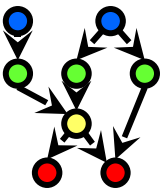
We expect this functional dependency to hold:

*loan\_number → amount*

but would not expect the following to hold:

*amount → customer\_name*





# USE OF FUNCTIONAL DEPENDENCIES

- We use functional dependencies to:
  - ★ test relations to see if they are legal under a given set of functional dependencies.
    - ⇒ If a relation  $r$  is legal under a set  $F$  of functional dependencies, we say that  $r$  satisfies  $F$ .
  - ★ specify constraints on the set of legal relations
    - ⇒ We say that  $F$  holds on  $R$  if all legal relations on  $R$  satisfy the set of functional dependencies  $F$ .
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - ★ For example, a specific instance of *loan* may, by chance, satisfy *amount* → *customer\_name*.

