

EXTENDED E-R FEATURES

S.P.Khandait

EXTENSIONS TO E-R MODEL

- Basic E-R model is good for many uses
- Several extensions to E-R model for more advanced modeling
 - Generalization and specialization
 - Aggregation
- These extensions can also be converted to relational model
 - Introduce a few more design choices

SPECIALIZATION

- An entity-set might contain distinct subgroups of entities
 - Subgroups have some different attributes, not shared by entire entity-set
- E-R model provides specialization to represent such entity-sets
- Example: bank account categories
 - Checking accounts
 - Savings accounts
 - Have common features, but also unique attributes

GENERALIZATION AND SPECIALIZATION

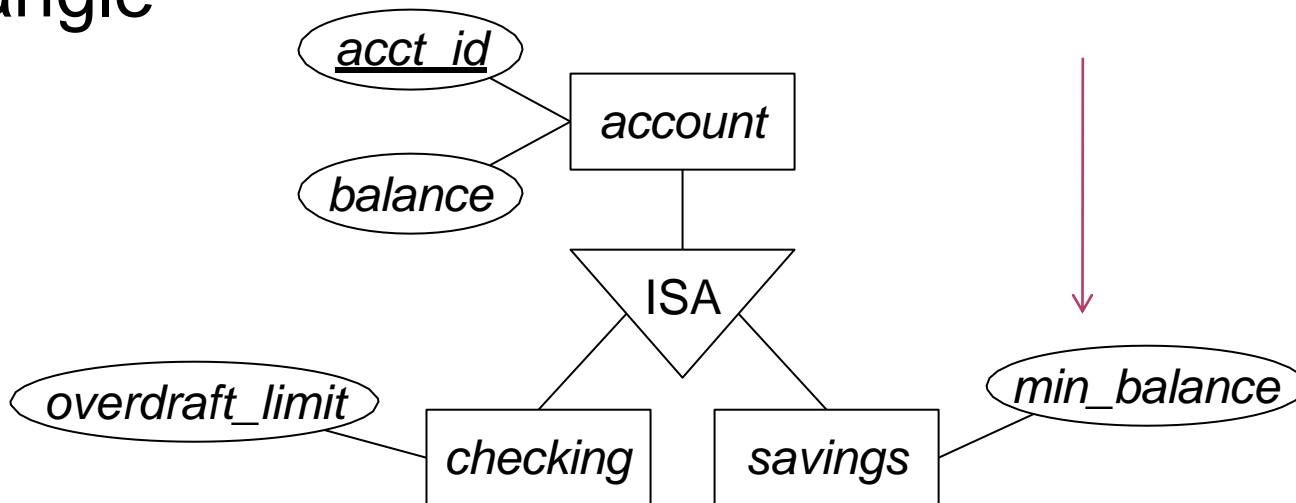
- **Generalization:** a “bottom up” approach
 - Taking similar entity-sets and unifying their common features
 - Start with specific entities, then create generalizations from them
- **Specialization:** a “top down” approach
 - Creating general purpose entity-sets, then providing specializations of the general idea
 - Start with general notion, then refine it
- **Terms are basically equivalent**
 - Book refers to generalization as overarching concept

BANK ACCOUNT EXAMPLE

- Checking and savings accounts have:
 - account number
 - balance
 - owner(s)
- Checking accounts also have:
 - overdraft limit and associated account
 - check transactions
- Savings accounts also have:
 - minimum balance

BANK ACCOUNT EXAMPLE (2)

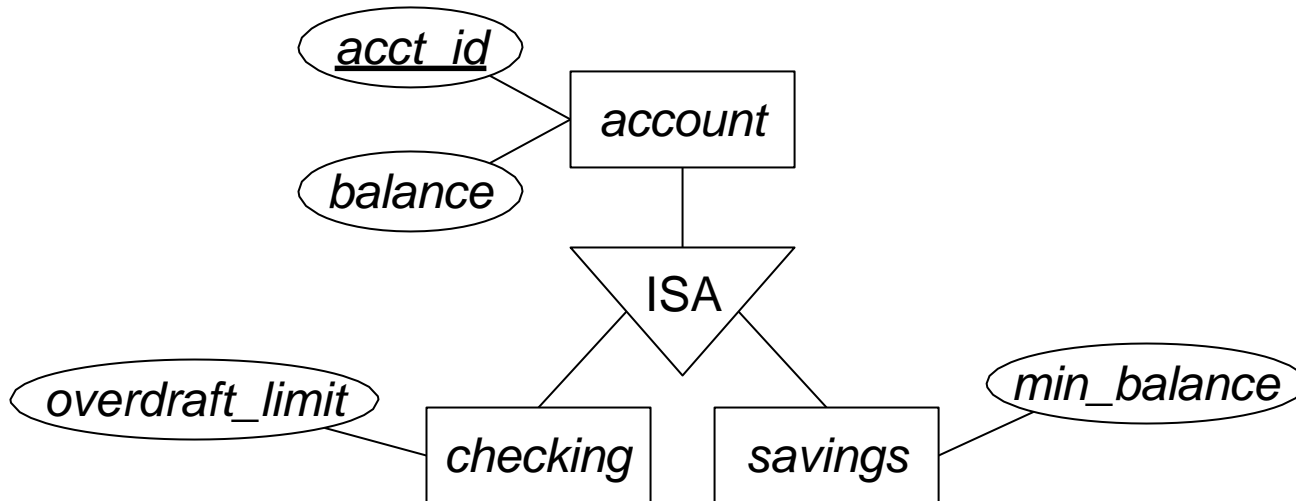
- Create entity-set to represent common attributes
 - Called the superclass, or higher-level entity-set
- Create entity-sets to represent specializations
 - Called subclasses, or lower-level entity-sets
- Join superclass to subclasses using “ISA” triangle



INHERITANCE

- Attributes of higher-level entity-sets are inherited by lower-level entity-sets
- Relationships involving higher-level entity-sets are also inherited by lower-level entity-sets!
 - A lower-level entity-set can participate in its own relationship-sets, too
- Usually, entity-sets inherit from one superclass
 - Entity-sets form a hierarchy
- Can also inherit from multiple superclasses
 - Entity-sets form a lattice
 - Introduces many subtle issues, of course

SPECIALIZATION CONSTRAINTS



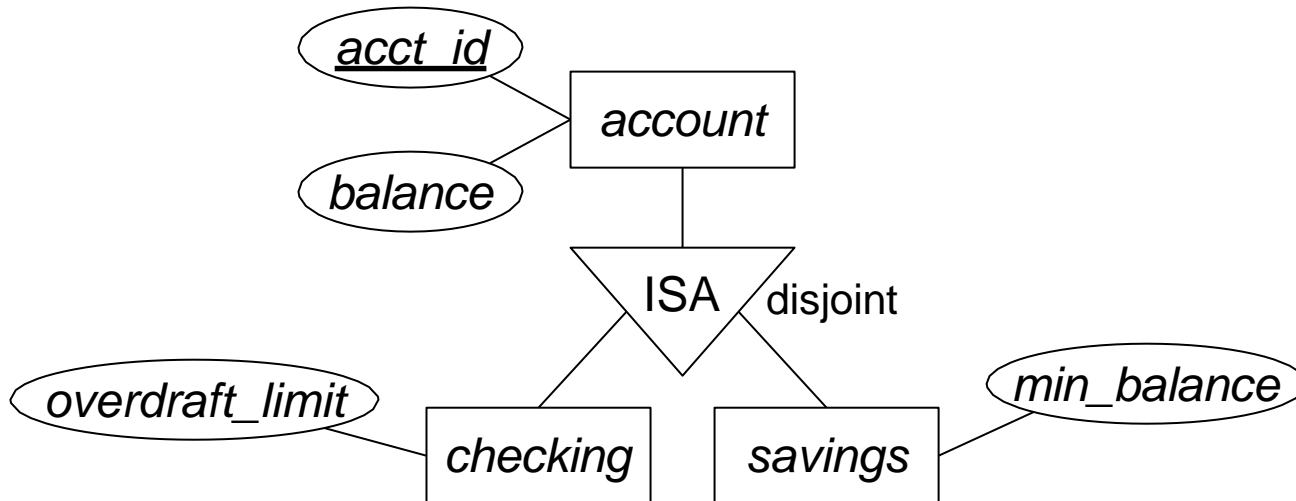
- Can an account be both a savings account and a checking account?
- Can an account be neither a savings account or a checking account?
- Can specify constraints on specialization
 - Enforce what “makes sense” for the enterprise

DISJOINTNESS CONSTRAINTS

- “An account must be either a checking account, or a savings account, but not both.”
- An entity may belong to only one of the lower-level entity-sets
 - Must be a member of *checking*, or a member of *savings*, but not both!
 - Called a “disjointness constraint”
 - A better way to state it: a disjoint specialization
- If an entity can be a member of multiple lower-level entity-sets:
 - Called an overlapping specialization

DISJOINTNESS CONSTRAINTS (2)

- Default constraint is overlapping!
- Indicate disjoint specialization with word “disjoint” next to triangle
- Updated bank account diagram:

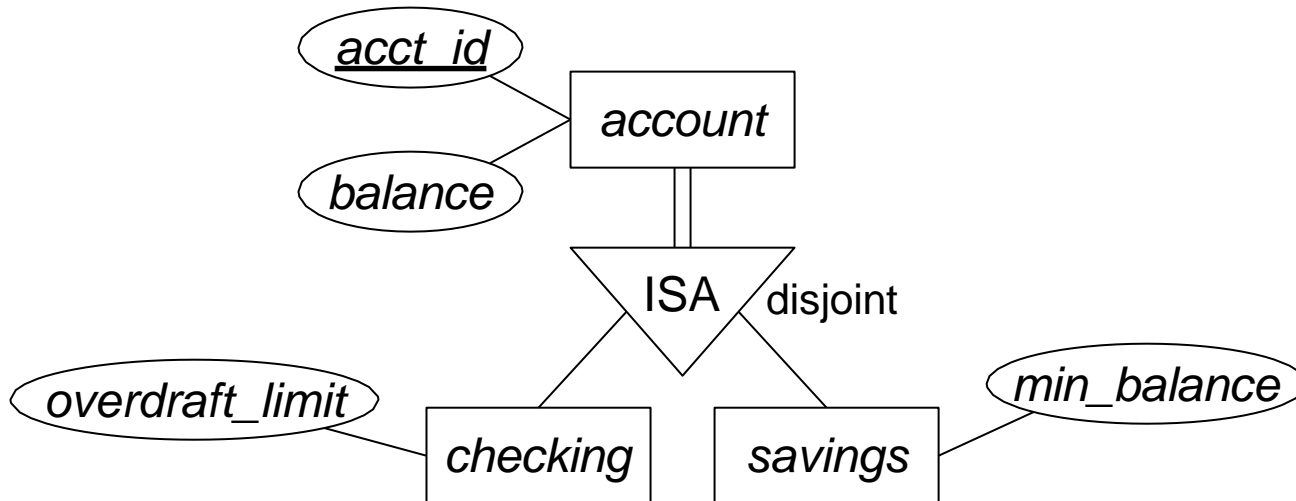


COMPLETENESS CONSTRAINTS

- “An account must be a checking account or a savings account.”
- Every entity in higher-level entity-set must also be a member of at least one lower-level entity-set
 - Called total specialization
- If entities in higher-level entity-set aren’t required to be members of lower-level entity-sets:
 - Called partial specialization
- *account* specialization is a total specialization

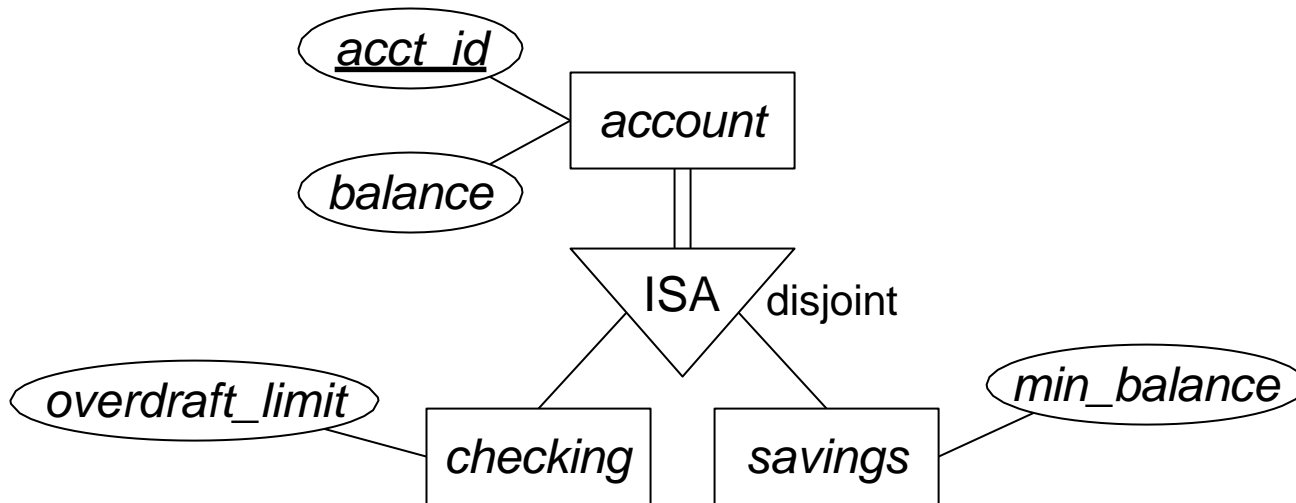
COMPLETENESS CONSTRAINTS (2)

- Default constraint is partial specialization
- Specify total specialization constraint with a double line on superclass side
- Updated bank account diagram:



ACCOUNT TYPES?

- Our bank schema so far:



- How to tell whether an account is a checking account or a savings account?
 - No attribute indicates type of account

MEMBERSHIP CONSTRAINTS

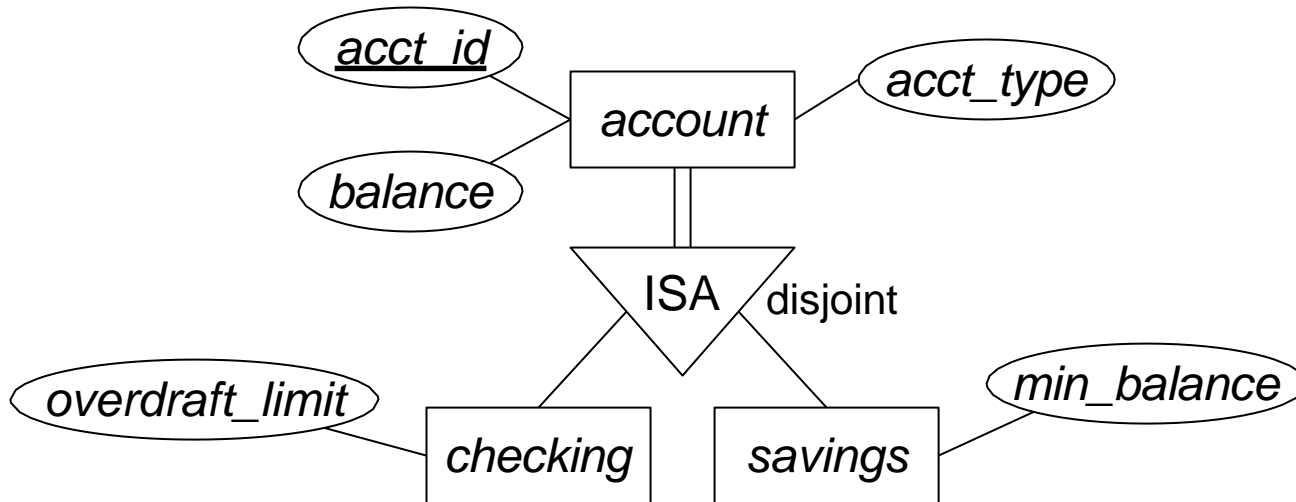
- Membership constraints specify which entities are members of lower-level entity-sets
 - e.g. which accounts are checking or savings accounts
- Condition-defined lower-level entity-sets
 - Membership is specified by a predicate
 - If an entity satisfies a lower-level entity-set's predicate then it is a member of that lower-level entity-set
 - If *all* lower-level entity-sets refer to the same attribute, this is called attribute-defined specialization
 - e.g. *account* could have an *account_type* attribute

MEMBERSHIP CONSTRAINTS (2)

- Entities may simply be assigned to lower-level entity-sets by a database user
 - No explicit predicate governs membership
 - Called user-defined membership
- Generally used when an entity's membership could change in the future
- Bank account example:
 - Accounts *could* use user-defined membership, but wouldn't make so much sense
 - Makes it harder to write queries involving only one kind of account
 - Best choice is probably attribute-defined membership

BANK ACCOUNTS

- Final bank account diagram:

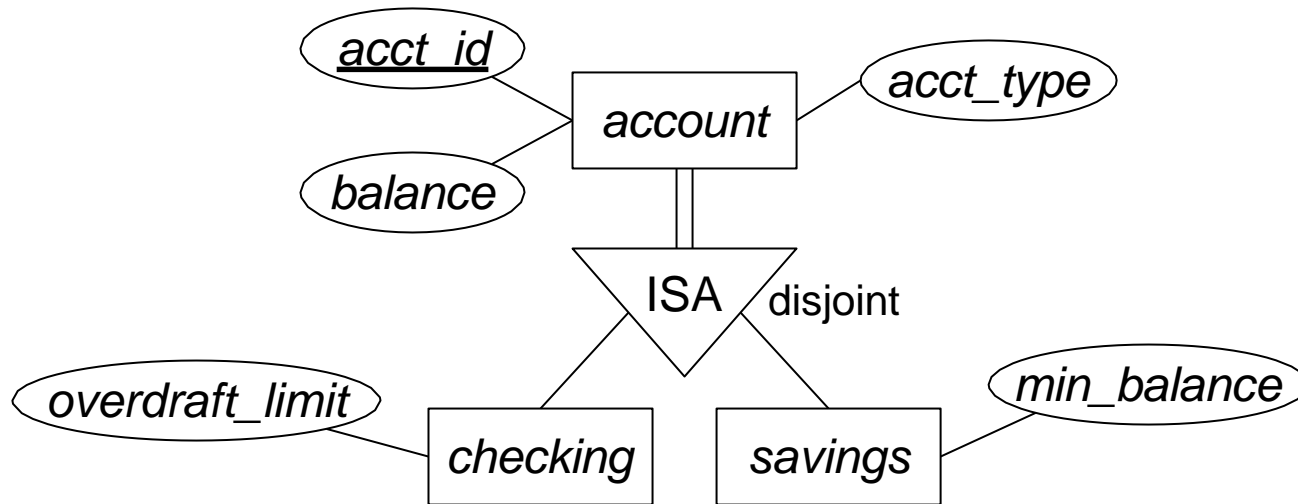


- Would also create relationship-sets against various entity-sets in hierarchy
 - associate *customer* with *account*
 - associate *check_txns* weak entity-set with *checking*

MAPPING TO RELATIONAL MODEL

- Mapping generalization/specialization to relational model is straightforward
- Create relation schema for higher-level entity-set
 - Including primary keys, etc.
- Create schemas for lower-level entity-sets
 - Subclass schemas include superclass' primary key attributes!
 - Primary key is same as superclass' primary key
 - If subclass contains its own primary key, treat as a separate candidate key
 - Foreign key reference from subclass schemas to superclass schema, on primary-key attributes

MAPPING BANK ACCOUNT SCHEMA



- Schemas:

account(*acct_id*, *acct_type*, *balance*)

checking(*acct_id*, *overdraft_limit*)

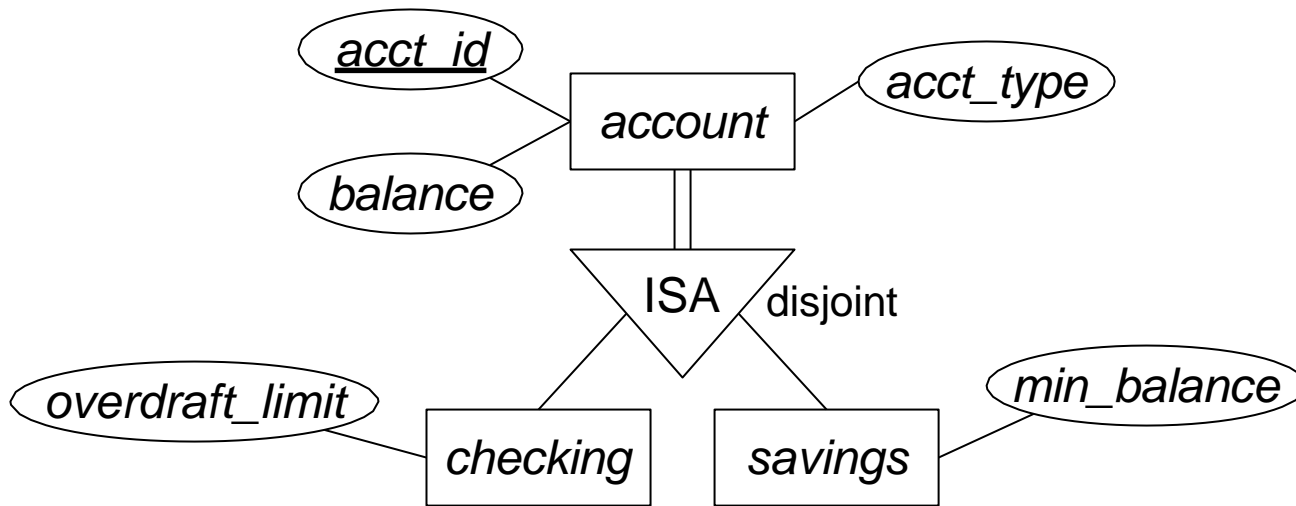
savings(*acct_id*, *min_balance*)

- Could use **CHECK** constraints SQL tables for membership constraints, other constraints

ALTERNATIVE SCHEMA MAPPING

- If specialization is disjoint and complete, can convert only lower-level entity-sets to relational schemas
 - Every entity in higher-level entity-set also appears in lower-level entity-sets
 - Every entity is a member of *exactly one* lower-level entity-set
- Each lower-level entity-set has its own relation schema
 - All attributes of superclass entity-set are included on each subclass entity-set
 - No relation schema for superclass entity-set

ALTERNATIVE ACCOUNT SCHEMA



- Schemas:

checking(acct_id, acct_type, balance, overdraft_limit)

savings(acct_id, acct_type, balance, min_balance)

ALTERNATIVE ACCOUNT SCHEMA (2)

- Alternative schemas:

checking(acct_id, acct_type, balance, overdraft_limit)

savings(acct_id, acct_type, balance, min_balance)

- Problems?

- Enforcing uniqueness of account IDs!

- Representing relationships involving general accounts

- Can solve by creating a simple relation:

account(acct_id)

- Contains *all* valid account IDs

- Relationships involving accounts can use *account*

- Need foreign key constraints again...

GENERATING PRIMARY KEYS

- *Generating* primary key values is actually the easy part
- Most databases provide sequences
 - A source of **INTEGER** or **BIGINT** values
 - Perfect for primary key values
 - Multiple tables can use a sequence for their primary keys

- PostgreSQL example:

```
CREATE SEQUENCE acct_seq;
```

```
CREATE TABLE checking (  
    acct_id INT PRIMARY KEY DEFAULT nextval('acct_seq');  
    ...  
);
```

```
CREATE TABLE savings (  
    acct_id INT PRIMARY KEY DEFAULT nextval('acct_seq');  
    ...  
);
```

ALTERNATIVE SCHEMA MAPPING

- Alternative mapping has some drawbacks
 - Doesn't actually give many benefits in general case
 - Biggest issue is managing primary keys!
- Fewer drawbacks if:
 - Total, disjoint specialization
 - No relationships against superclass entity-set
- If specialization is overlapping, some details are stored multiple times
 - Unnecessary redundancy, and consistency issues
- Also limits future schema changes

EER DESIGN

Q1) Design a generalization-specialization hierarchy for a motor-vehicle sales company. The company sells motorcycles, passenger cars, vans and buses. Justify your placement of attributes at each level of hierarchy. Explain why they should not be placed at a higher or lower level.

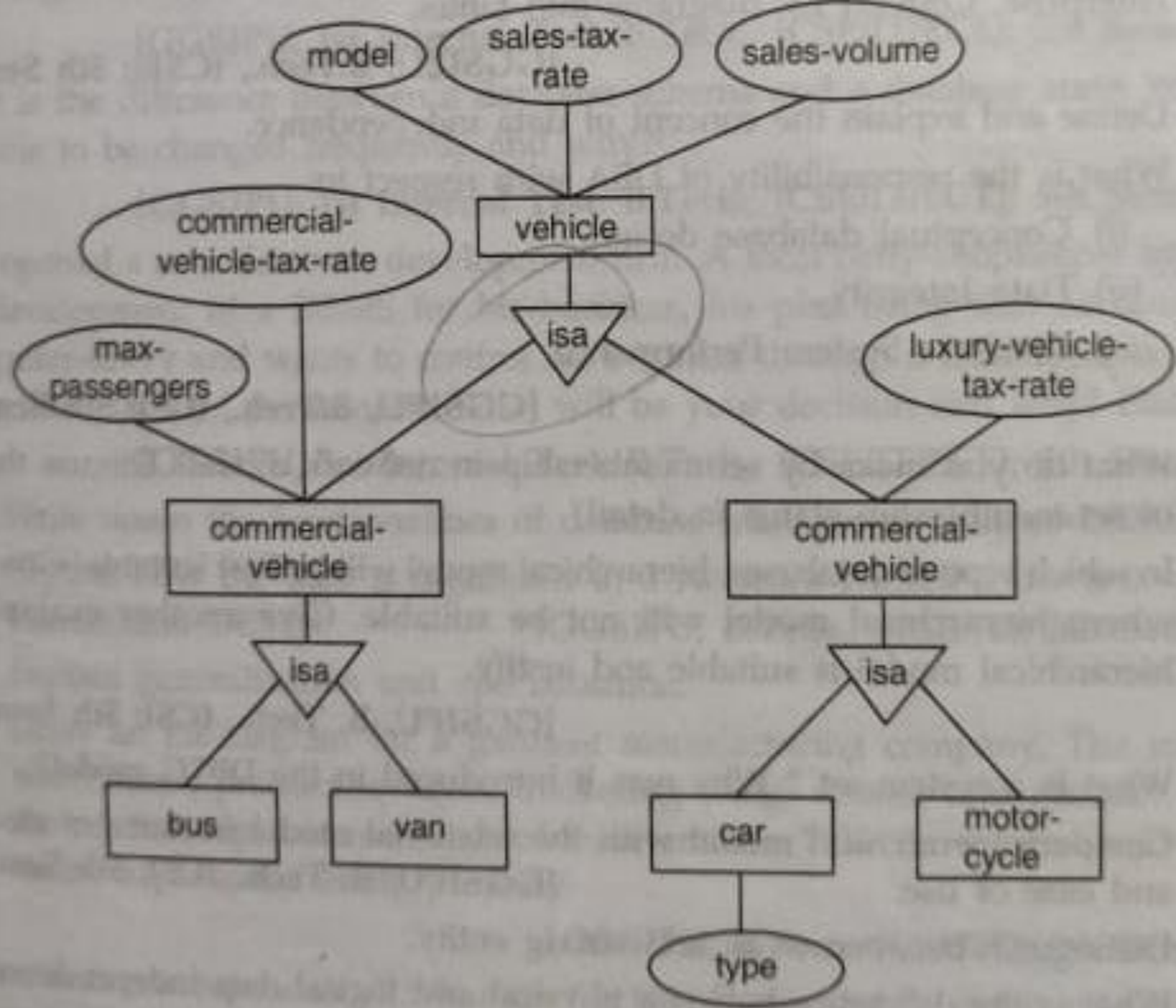
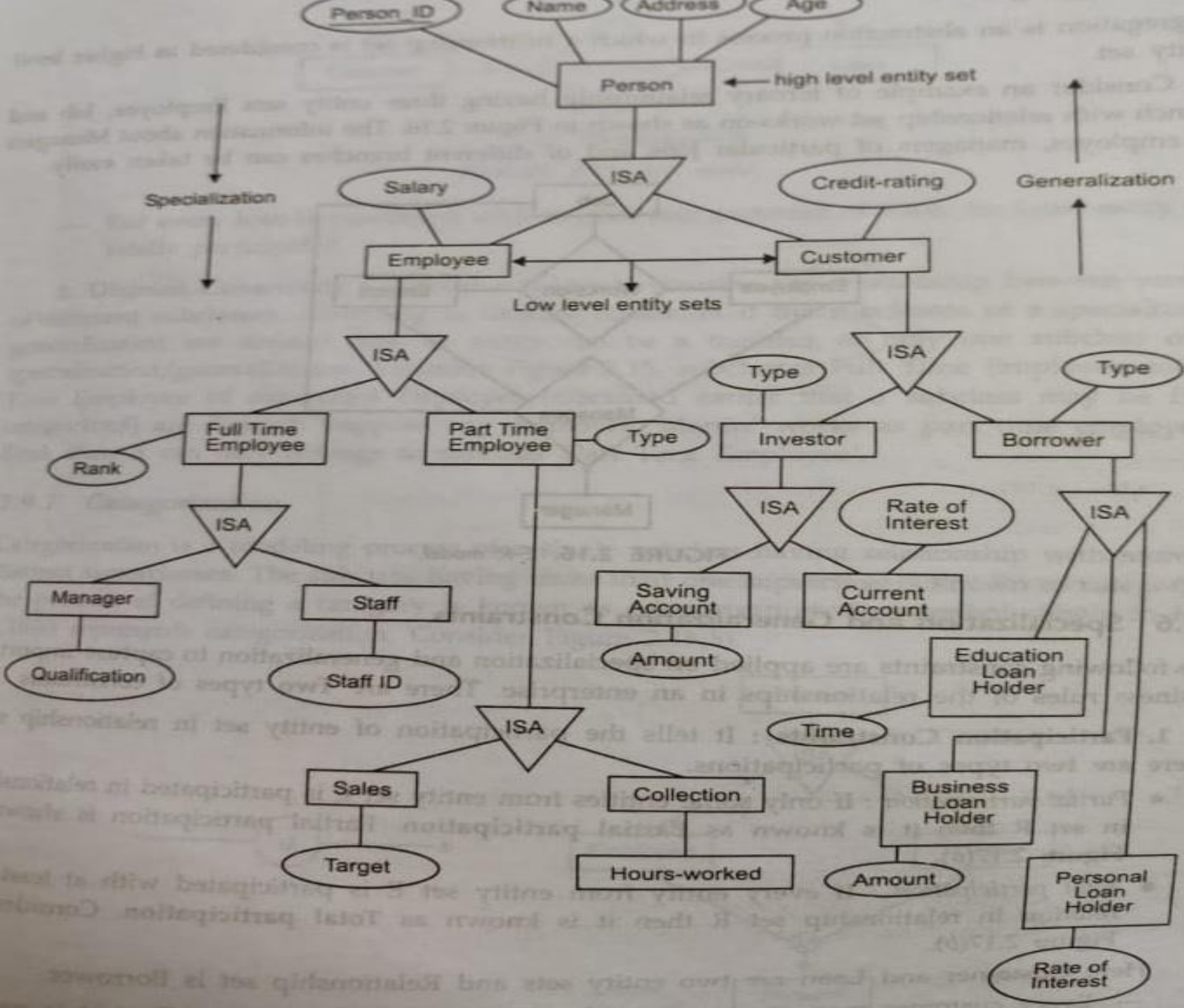


Fig. 4.51

EER DESIGN

Q2) Construct an EER Model for Banking System showing generalization-specialization



Specialization

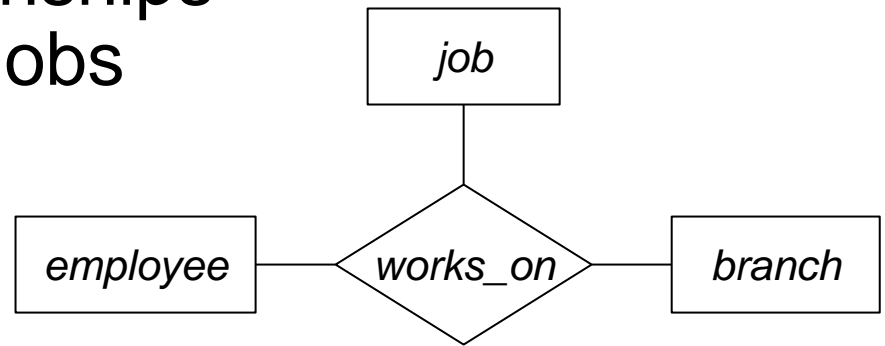
high level entity set

Generalization

Low level entity sets

RELATIONSHIPS OF RELATIONSHIPS

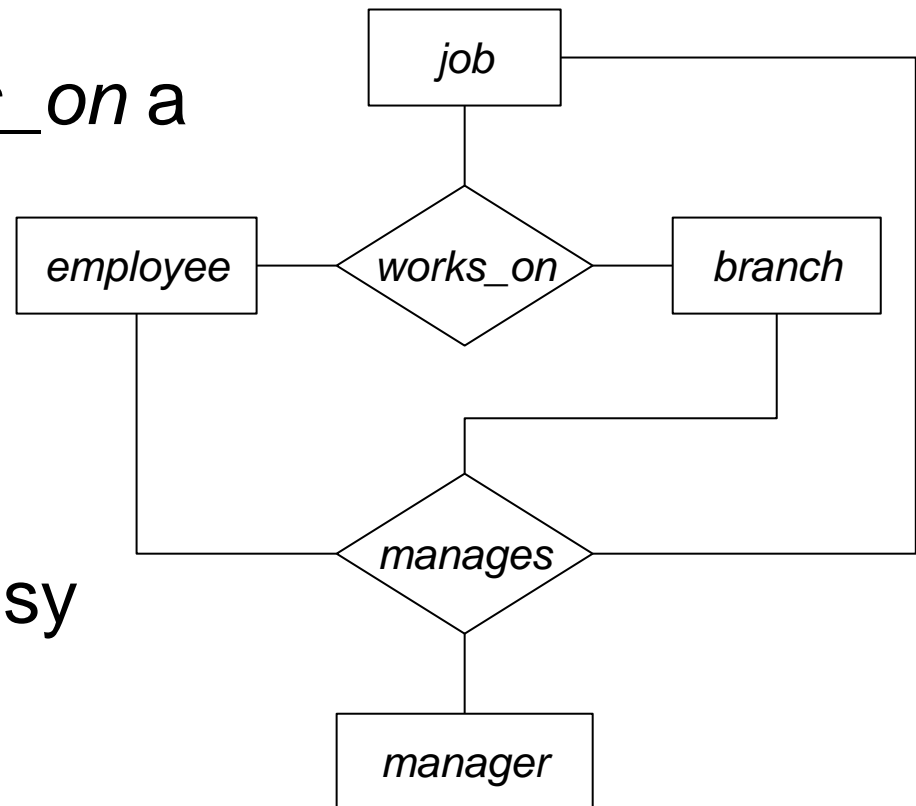
- Basic E-R model can't represent relationships involving other relationships
- Example: employee jobs



- Want to assign a manager to each (employee, branch, job) combination
 - Need a separate *manager* entity-set
 - Relationship between each manager, employee, branch, and job entity

REDUNDANT RELATIONSHIPS

- One option: a quaternary relationship
 - This option has lots of redundant information
 - Benefit is that some jobs might not require a manager
- Could also make *works_on* a quaternary relationship
 - Don't use a separate *manager* relation
 - Jobs with no manager would use *null* values instead
- These options are clumsy

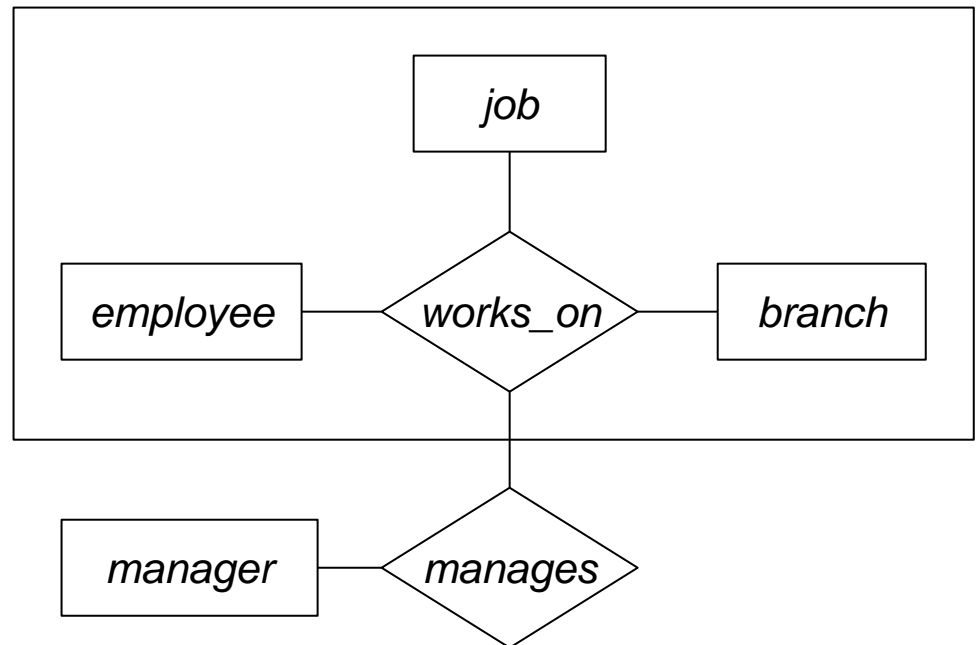


AGGREGATION

Another option is to treat *works_on* relationship as an aggregate

- Build a relationship against the aggregate
- *manages* implicitly includes set of entities participating in a *works_on* relationship instance

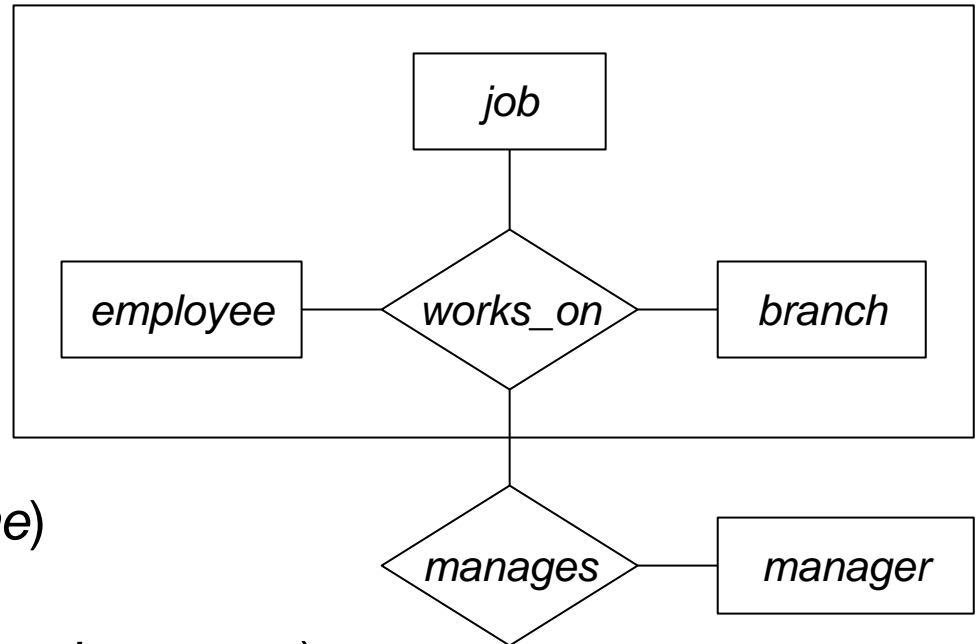
– Jobs can also have no manager



MAPPING TO RELATIONAL MODEL

- Mapping for aggregation is straightforward
- For entity-sets and relationship-set being used as an aggregate, mapping is unchanged
- Relationship-set against the aggregate:
 - Includes primary keys of participating entity-sets
 - Includes all primary key attributes of aggregated relationship-set
 - Also includes any descriptive attributes
 - Primary key of relationship-set includes all the above primary key attributes
 - Foreign key against aggregated relationship-set, as well as participating entity-sets

MANAGER EXAMPLE



- Job schemas:

employee(*emp_id*, *emp_name*)

job(*title*, *level*)

branch(*branch_name*, *branch_city*, *assets*)

works_on(*emp_id*, *branch_name*, *title*)

- Manager schemas:

manager(*mgr_id*, *mgr_name*)

manages(*mgr_id*, *emp_id*, *branch_name*, *title*)

DIFFERENCES

- Differences between version with aggregation, and version with quaternary relationship?
- Biggest difference:
 - Quaternary relationship's schema derives primary and foreign key constraints from participating entities
 - Relationship using aggregation derives primary and foreign key constraints from aggregate relationship
- A subtle difference
 - Doesn't have any significant practical impact

REVIEW

- Covered two extensions to E-R model
 - Higher level abstractions
- Generalization and specialization
 - Can specify constraints:
 - Membership constraints
 - Completeness constraints
 - Disjointedness constraints
- Aggregation
 - Can build relationships that include other relationships
- Straightforward mappings to relational model
- Next time: normal forms!