MDA104 Introduction to Databases

3. Relational Model

Vlastislav Dohnal

Relational Model

- Structure of Relational Databases
- Conversion of ERD to relational model

(the following is in the next chapters)

- Query Language
  - Fundamental Relational-Algebra-Operations
  - Additional Relational-Algebra-Operations
  - Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database

Example of a Relation *Account*

account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
  - E.g., the value of an attribute can be an account number, but cannot be a set of account numbers
- Domain is said to be atomic if all its members are atomic
- The special value *null* is a member of every domain
- The null value causes complications in the definition of many operations
  - We shall ignore the effect of null values in our main presentation and consider their effect later

Relational Model

- Formally, given attributes $A_1, A_2, \dots. A_n$ and their domains $D_1, D_2, \dots. D_n$

  a **relation** r is a subset of $D_1 \times D_2 \times \dots \times D_n$

  Thus, a relation is a set of n-tuples $(a_1, a_2, \dots, a_n)$ where each $a_i \in D_i$
- **Schema** of a relation consists of
  - a list of attribute definitions
    - name
    - type/domain
  - integrity constraints

Relation Instance

- The current values (*relation instance*) of a relation are specified by a table
- An element *t* of *r* is a *tuple*
  - represented by a *row* in a table
- Order of tuples is irrelevant
  - tuples may be stored in an arbitrary order

attributes (or columns)

customer_name	*customer_street*	customer_city
Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

tuples (or rows)

customer

Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts,
  - with each relation storing one part of the information
  - E.g.:

customer : information about customers
account : information about accounts
depositor : which customer owns which account

The *customer* Relation

customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

The *Account* Relation

account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

The *depositor* Relation

customer_name	*account_number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Why Split Information Across Relations?

- Storing all information as a single relation such as

  bank(*account_number, branch_name, balance,*
  customer_name, *customer_street, customer_city*)

  results in
  - repetition of information
    - E.g., if two customers share the same account
      - What gets repeated?
  - the need for null values
    - E.g., to represent a customer without an account
- Normalization theory deals with how to design relational schemas
  - We will study in later…

Integrity Constraints: Keys

- Let $K \subseteq R$, $K \neq \emptyset$, then *K* is a **key***.*
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - by "possible *r* " we mean a relation *r* that could exist in the enterprise we are modeling.
  - Trivial superkey is R.
  - Example:
    - Relation
      customer (*customer_name, customer_street, customer_city*)
    - Keys: {*customer_name, customer_street*} and
      {*customer_name*}
      are both superkeys, if no two customers can possibly have the same name.
    - In real life, an attribute such as *customer_id* would be used instead of *customer_name*
      - to uniquely identify customers,
        - but we omit it to keep our examples small, and instead assume customer names are unique.

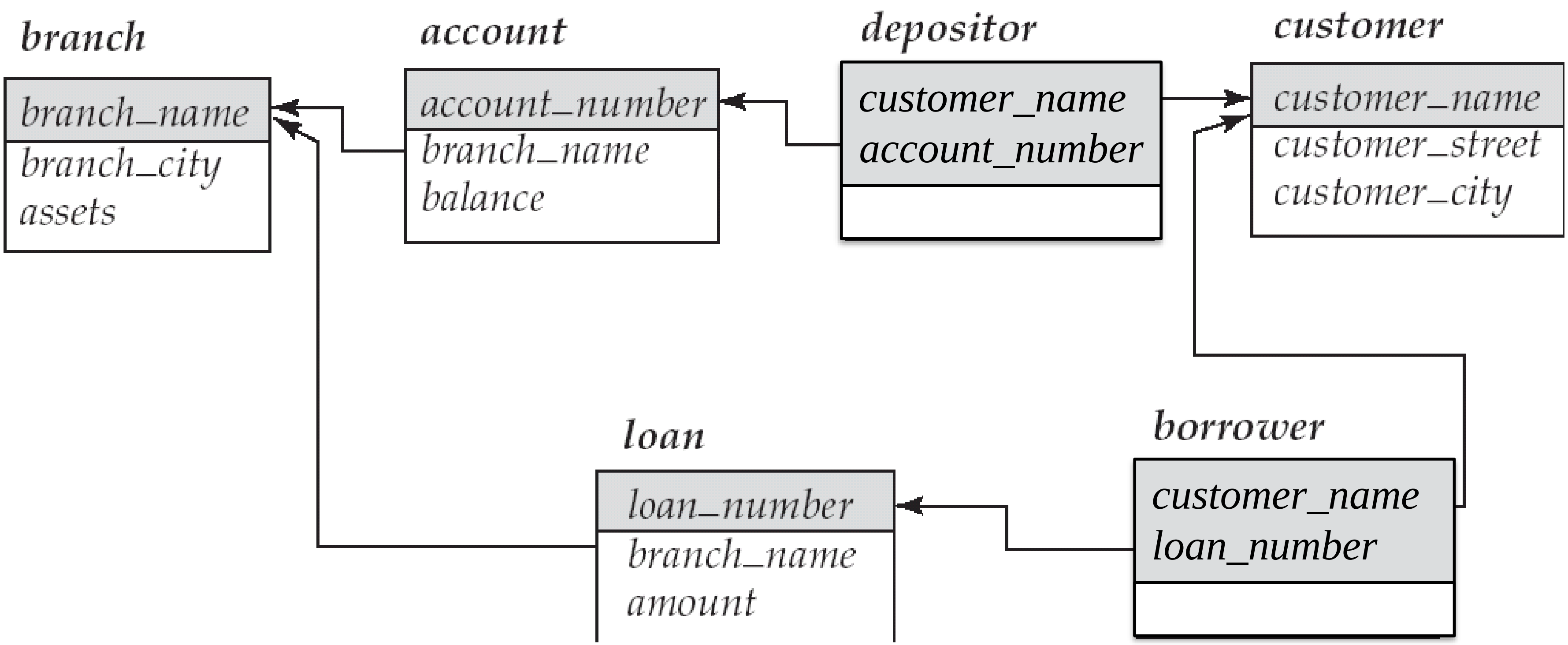
Integrity Constraints: Keys (Cont.)

- *K* is a **candidate key** if superkey *K* is minimal.
  - Example: {*customer_name*} is a candidate key for *Customer*, since it is a superkey and no subset of it is a superkey.
- **Primary key**
  - a candidate key chosen as the principal means of identifying tuples within a relation
  - Should choose an attribute whose value never, or very rarely, changes.
    - E.g. email address is unique, but may change
  - Primary key is depicted in schema using **underlined** attribute names.

Integrity Constraints: Foreign Keys

- A relation schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a **foreign key**.
  - Example:
    - Relations:
      customer (<u>*customer_name*</u>*, customer_street, customer_city*)
      account (<u>*account_number*</u>, *branch_name*, *balance*)
      depositor (<u>***customer_name***</u>, <u>***account_number***</u>)
    - Foreign keys:
      - *customer_name* and *account_number* attributes of *depositor* are **foreign keys** to *customer* and *account* respectively.
  - Only values occurring in the primary key attribute of the **referenced relation** may occur in the foreign key attribute of the **referencing relation.**
- A relation <u>may</u> have foreign keys, but it <u>should</u> have a primary key.

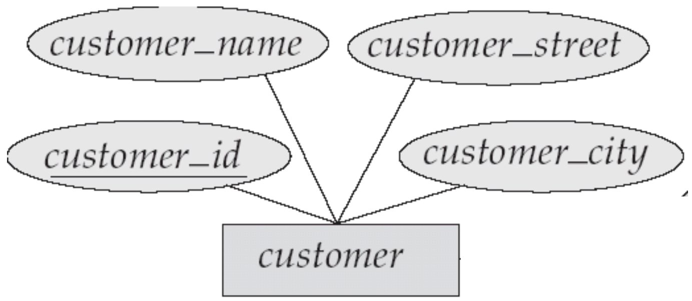
Schema Diagram



Primary keys are emphasized by grey background and foreign keys by connecting lines.

Reduction of ERD to Relational Model

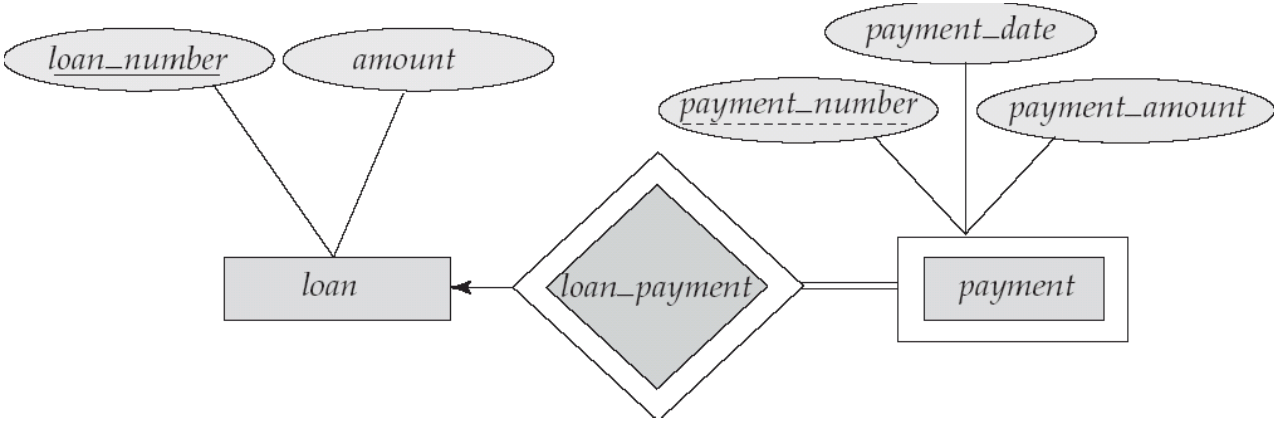
- Attributes and primary keys allow entity sets and relationship sets to be expressed uniformly as *relation schemas*
  - These schemas then represent the database schema
- For each entity set and relationship set,
  - there is a unique relation schema.
  - It is assigned the name of the corresponding 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

- An entity set reduces to a schema with the same attributes.
  - A strong entity set is handled the same way.
  - *loan* (*<u>loan_number</u>*, *amount*)

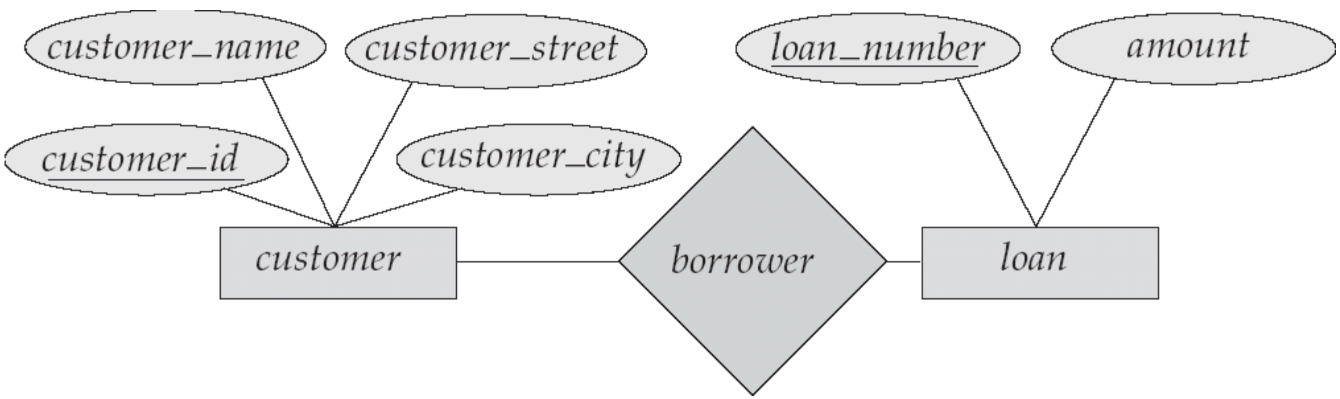
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set apart its regular attributes
  - *payment* (*<u>loan_number</u>*, *<u>payment_number</u>*, *payment_date*, *payment_amount*)



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.
  - its primary key is formed by the attributes coming from primary keys of the entity sets.
- Example: schema for relationship set borrower

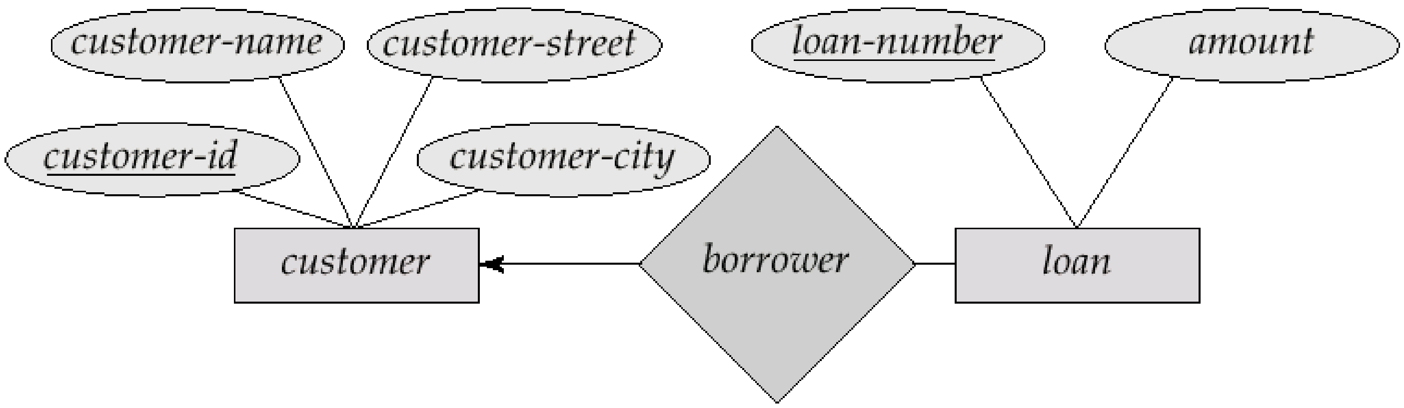
  borrower (*<u>customer_id</u>*, *<u>loan_number</u>*)



Representing Relationship Sets (cont.)

- A one-to-many relationship set
  - is represented as a schema in the same way,
  - But its primary key contains attributes from the primary of the “many” entity set.
- Example: schema for relationship set borrower

  borrower (*customer_id*, *loan_number*)



Representing Relationship Sets (cont.)

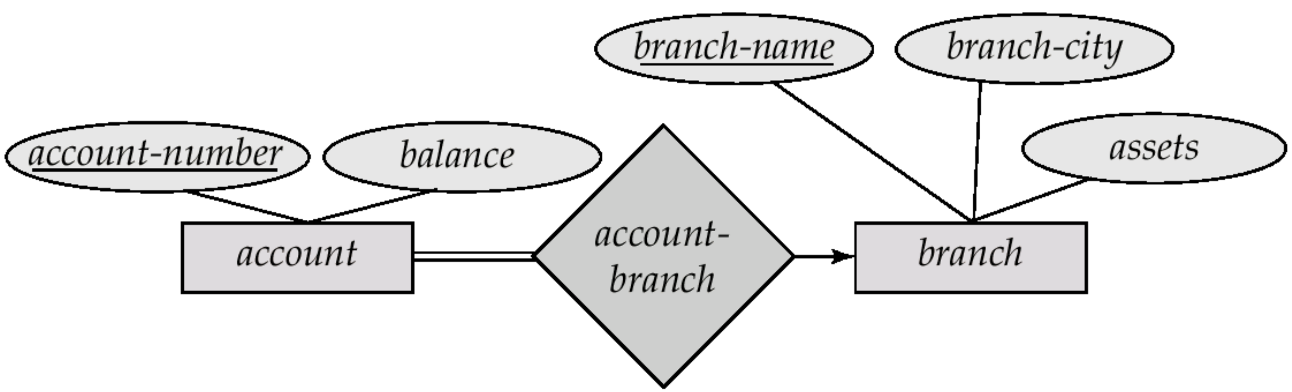
- A one-to-one relationship set
  - is represented as a schema in the same way,
  - Its primary key can be the primary of either the entity sets.
    - i.e., it behaves as in one-to-many
  - In relational model, this can be handled by adding a *unique* constraint
    - i.e., disallowing the values in the other attributes to repeat.
- Example: schema for a one-to-one relationship set *borrower*

  borrower (*customer_id, <u>loan_number</u>*)
  - The customer_id can be set unique
    - independently on the uniqueness of 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*

branch (*<u>branch_name</u>*, *branch_city*, *assets*)
account (*<u>account_number</u>*, *balance*, *branch_name*)



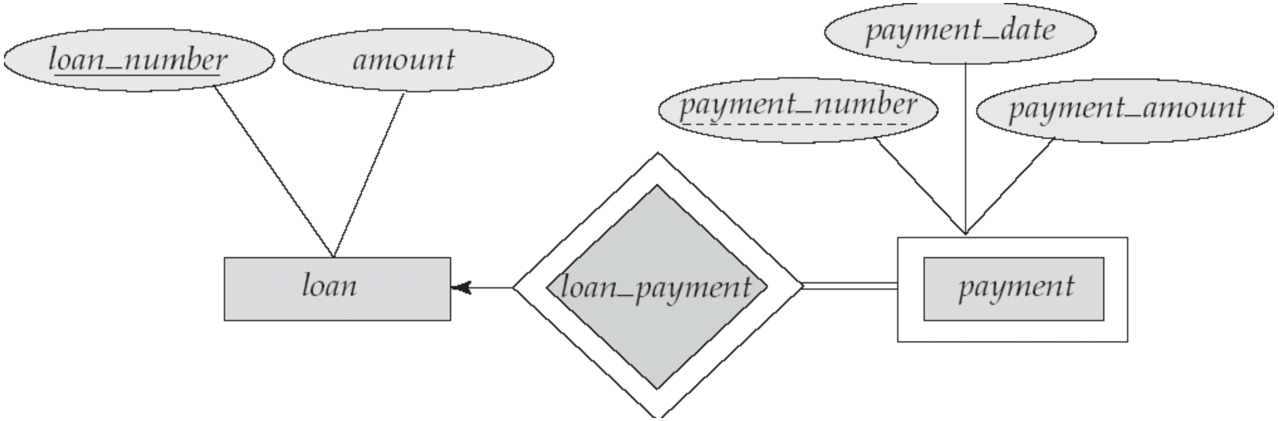
Redundancy of Schemas (Cont.)

- For total one-to-one relationship sets, either side can be chosen to act as the “many” side
  - That is, an 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.*

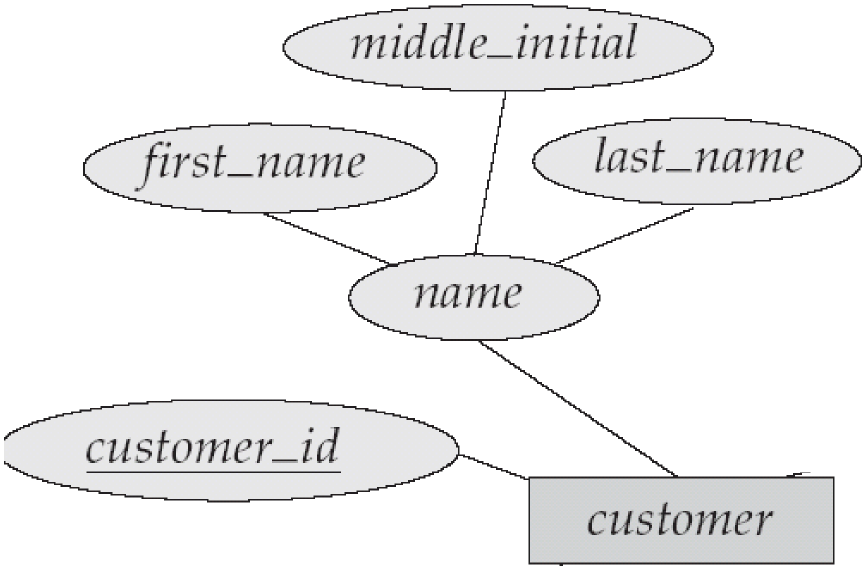
Redundancy of Schemas (Cont.)

- The schema corresponding to an *identifying relationship set*
  - linking a weak entity set to its 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*.
  - Relations
    loan (*loan_number*, *amount*)
    payment (*loan_number*, *payment_number*, *payment_date*, *payment_amount*)



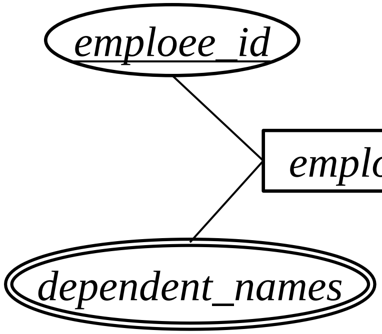
Composite Attributes

- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - Example:
    - Given entity set *customer* with a composite attribute *name* with component attributes *first_name* and *last_name*
    - The schema corresponding to the entity set has two attributes *name_first_name* and *name_last_name*



Multivalued Attributes

- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*
  - Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to the multivalued attribute *M*
  - Example:
    - Multivalued attribute *dependent_names* of *employee* is represented by a schema:
      employee_dependent_names = (*employee_id, dependent_name*)
  - Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
    - For example, an employee entity with primary key 123-45-6789 and dependents *Jack* and *Jane* maps to two tuples:

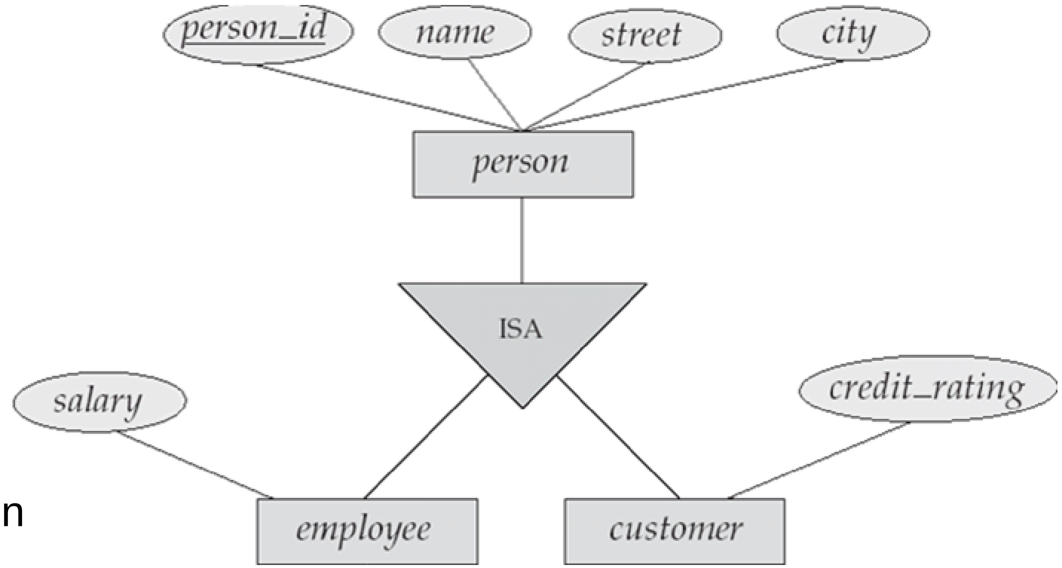


(123-45-6789, Jack)
(123-45-6789, Jane)

Representing Specialization as Schemas

- Method 1:
  - Form a schema for the higher-level entity
  - Form a schema for each lower-level entity set
    - include primary key of higher-level entity set and local attributes

Schema	**Attributes**
person	*person_id*, *name*, *street*, *city*
customer	*person_id*, *credit_rating*
employee	*person_id*, *salary*



- Drawback: getting information about, an employee requires accessing two relations

Representing Specialization as Schemas (Cont.)

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes

Schema	**Attributes**
person	*<u>person_id</u>, name, street, city*
customer	*<u>person_id</u>, name, street, city, credit_rating*
employee	*<u>person_id</u>, name, street, city, salary*

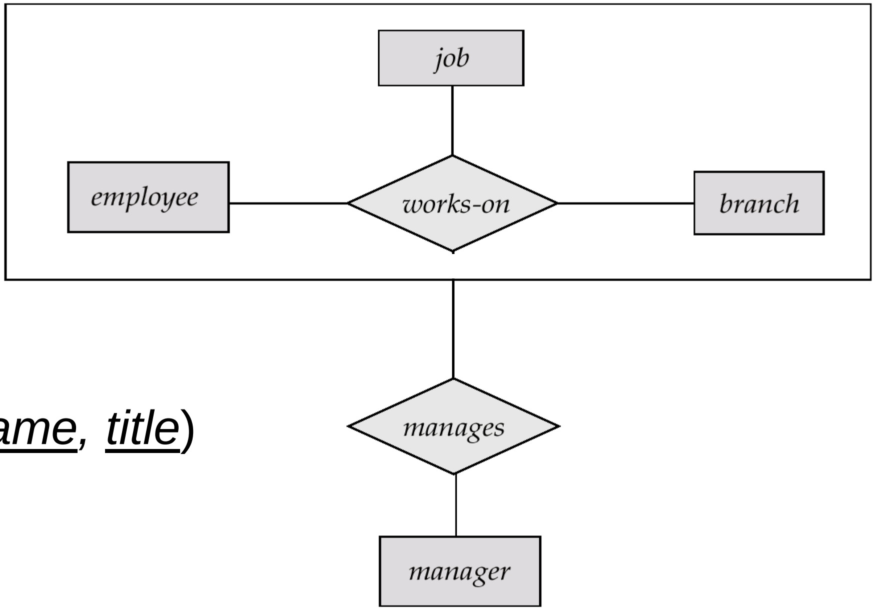
  - If specialization is total, the schema for the generalized entity set (*person*) is not required to store information
    - Can be defined as a "view" relation containing union of specialization relations
    - But explicit schema may still be needed for foreign key constraints
  - Drawback:
    - *street* and *city* may be stored redundantly for people who are both customers and employees

Schemas Corresponding to Aggregation

- Aggregation itself is not represented by any schema, but rather its corresponding relationship set is used.

- To represent a relationship set between an aggregation and an entity set, create a schema containing
  - The primary key of the aggregated relationship,
  - The primary key of the associated entity set, and
  - Any descriptive attributes

Schemas Corresponding to Aggregation (Cont.)



- Example
  employee(*employee_id*, *name*)
  branch(*branch_name*, *address*)
  job(*title*, *description*)
  works_on(*employee_id*, *branch_name*, *title*)
  manager(*manager_id*, *name*)
- To represent relationship *manages* between the aggregation *works_on* and the entity set *manager*,
  - create a schema
    manages (*employee_id*, *branch_name*, *title*, *manager_id*)
- Schema *works_on* is redundant provided we are willing to store null values for attribute *manager_id* in relation *manages*

Summary (Takeaways)

- Terminology understanding
  - Relation (table)
  - Schema of relation
  - Database
  - Schema of database
- Integrity constraints
  - What it does
  - Primary key, candidate key, super key,
    - and their mutual relationship
- Reduction of ERD to relational model (tables)