CS 360
Database Systems

Hasan M. Jamil

Department of Computer Science
University of Idaho

Outline

- Databases.
  - DBMS – goals and advantages, DB systems architecture.
  - ER model.
  - Relational model: concepts, formal/ commercial query languages, views.
  - Conceptual DB design: functional dependencies, normalization.

- Files.
  - Basic file structures and access methods.
  - B^+ trees.
  - Inverted, multi-list organizations.
  - External sorting.
  - Dynamic hashing.
- Current research issues.
  - New database models – Deductive, object-oriented, deductive object-oriented, etc.
  - Data mining, data warehousing, OLAP, DSS.

CS 360
Database Systems

Part I: Database Foundations

DBMS – Goals and Advantages, DB Systems Architecture

DB = collection of related data.

– collection should be logically coherent and have some inherent meaning.

– it may not be a random pile of data.

– typically, it is a collection of data about an enterprise.

– a certain kind of end users intended implicitly.

Example:

– list of names, addresses, and phone numbers of your friends.

– info about employees, departments, salaries, managers, etc. of a company.

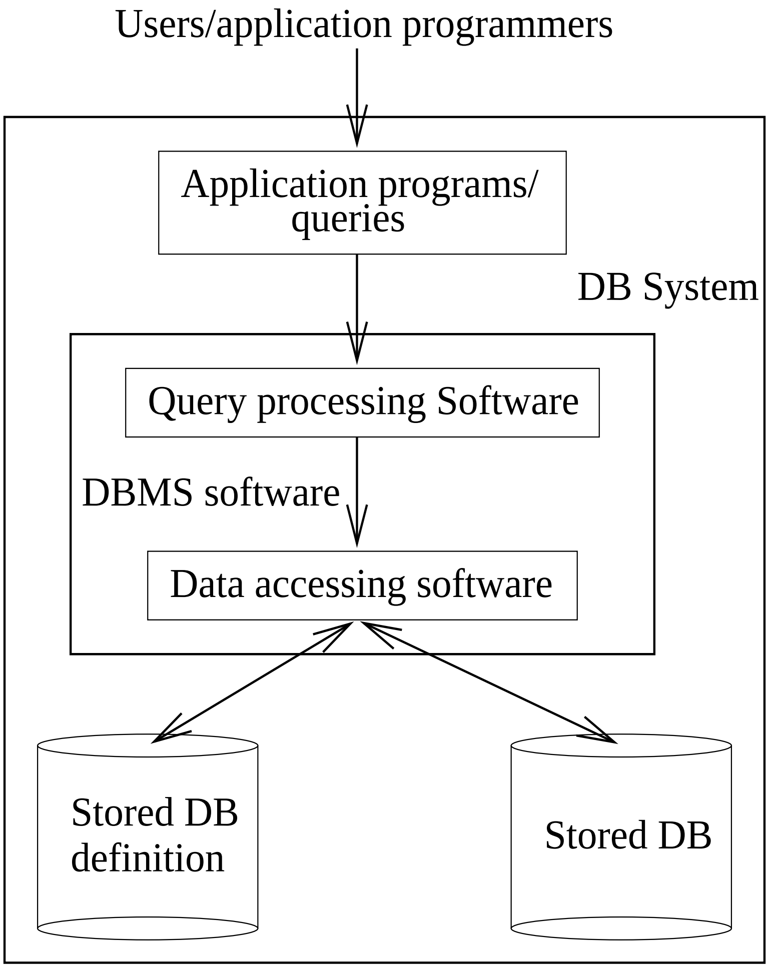
– info about students, courses, grades, professors, etc. in a university.

– info about catalogs, users, etc. in a library.

DBMS = software managing data in a DB (i.e., reading, writing, adding, updating, locating, etc.)

users see data in a DB through the DBMS (an intermediary software)

DB System = DB + DBMS


Users/application programmers
Application programs/ queries
DB System
Query processing Software
DBMS software
Data accessing software
Stored DB definition
Stored DB

Examples:

Application programs

Library DB – application program for sign out of books.

⟶ changing the status of the book from *on shelf* to *loaned out*, assigning the book to user, assigning proper date to due date, etc.

Queries

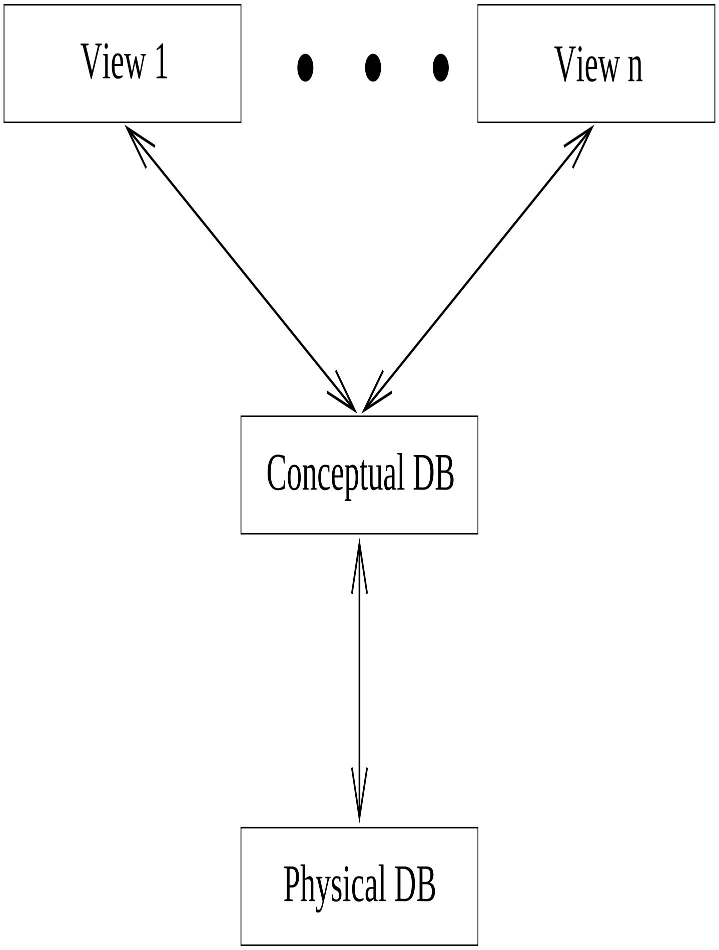
"Print the set of books in the library by Alan Turing between the years 1915 and 1945".

⟶ expressed in a suitable query language.

Goals and Advantages of DBMS

- minimizing data redundancy and avoiding inconsistency.
- concurrent access to multiple users (improves overall utilization and performance).
- centralized control over data management.
- security and authorization.
- integrity.
- reliability.
- data abstraction and independence.

Data Abstraction



View 1 • • • View n

Very high level perception of DB for user groups

Conceptual DB

Abstract view of DB (DBA)

Physical DB

Implementation for maximum efficiency

Example: An employee database

Conceptual level:

```
type emp = record
    num : integer;
    name : string;
    dob : date;
    salary : real;
    dept : string;
end
```

View level:

```
view1: (emp.name, emp.dept)
view2: (emp.name, emp.age)
```

Physical level:

A block of consecutive bytes actually holding the above info.

Data Independence

Physical $\longrightarrow$ changes in implementation strategies need not distort the conceptual perception of the DB.

Logical $\longrightarrow$ changes in conceptual DB need not affect the user views.

Schemes versus Instances.

Example:

– changing file structure from sequential to direct access (physical independence).

– adding new fields to a record or changing the type of a field (logical independence).

Instances change over time while schemes are invariant.

Scheme = (emp.name, dept, # dependents).

Instance = (John, sales, 4).

Data Definition Language (DDL)

– definition of conceptual scheme and mapping between conceptual and physical schemes.

– definition of views (external schemes) and mapping between conceptual schemes and views.

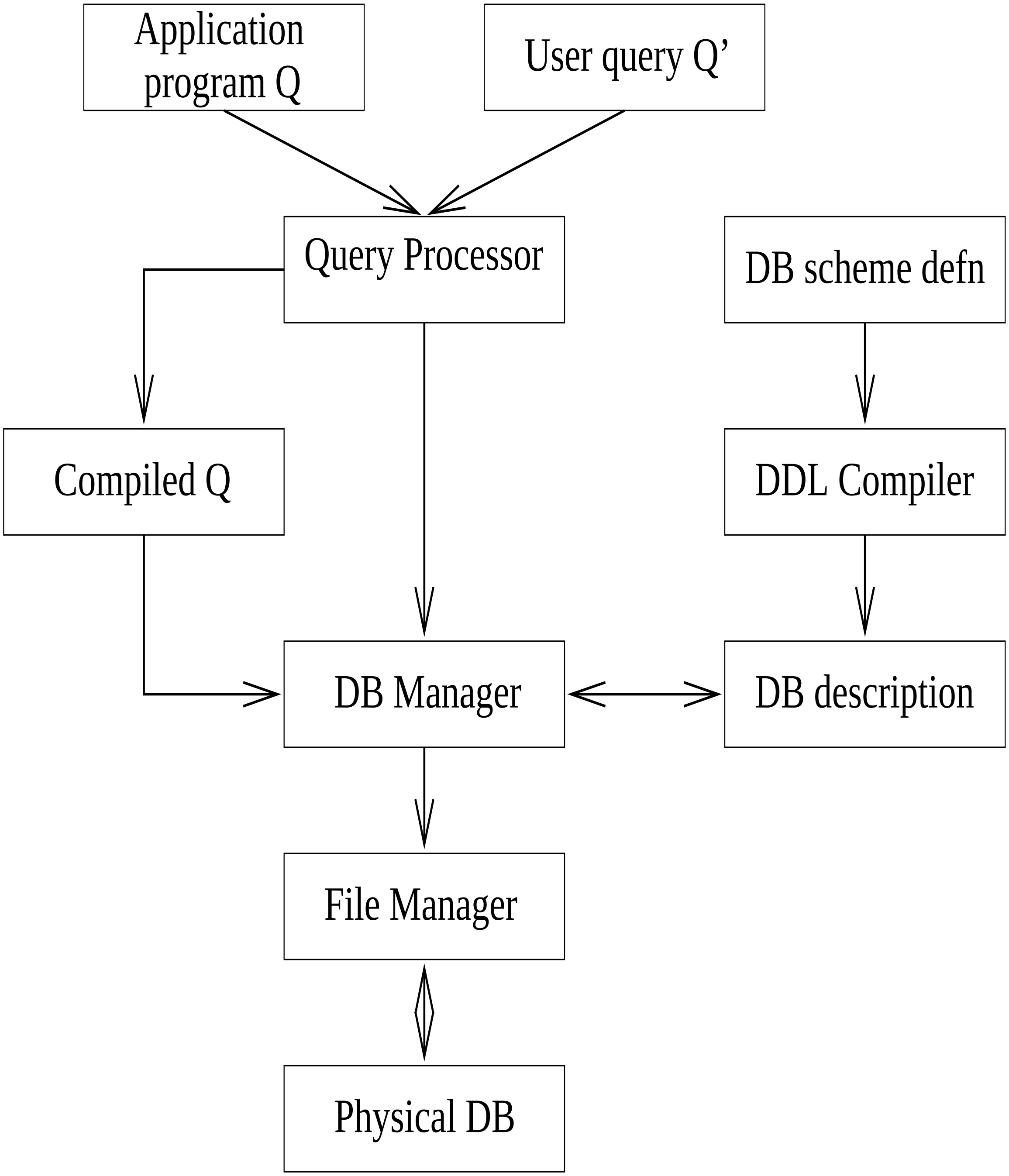
Data Manipulation Language (DML)

Querying and updating (insert, delete and modify operations) DB.

Typically, query langauge is separate.
DML - embedded in a host langauge like Cobol, Pascal, C, etc. Requires appropriate compilers.

DBA (usually a team of experts) coordinates various activities in the creation and maintenance of the DB systems.

DB systems structure



Entity-Relationship (ER) Model

– collection of abstraction/modeling primitives.

– help model real world objects or enterprises in an abstract way.

Entity and Entity Sets

– an entity is a distinguishable object that exists, e.g., the person *John*, the *book* by Alan Turing, etc.

– an entity set is a set of entities of the same type, e.g., set of all students in Concordia, set of books in a library, etc.

Entity sets need not be disjoint.

– an entity is represented by a set of attributes.

– an attribute is a function:

$$\text{attribute} : \text{entity set} \rightarrow \text{domain.}$$

Example

customer entity set - {(*name*: string), (*social-security*: integer), (*city*: string)}

A *customer* entity - {(*name*, John), (*social-security*, 123-456-789), (*city*, Montreal)}

Entity set → type definition.
Entity → variable of some type.

Database - includes a collection of entity sets and a corresponding set of entities (may be empty).

Relationships and Relationship Sets

– relationships are associations among entities.

– relationship set is a set of relationships of the same type.

If $E_1, \ldots, E_n$ are entity sets, and R is a relationship set, then

$$R \subseteq E_1 \times \ldots \times E_n$$

– most database relationships are binary.

– some may be n-ary, where $n \geq 2$.

Example

Customer entity - {(*name*, John), (*social-security*, 123-456-789), (*city*, Montreal)}

Account entity - {(*account#*, 507), (*balance*, 20K)}

Relationship CustAcc - {(*name*, John), (*social-security*, 123-456-789), (*city*, Montreal), (*account#*, 507), (*balance*, 20K)}

Attributes or Entities?

<u>Example</u>

Person entity - {(*name*, John), (*city*, Montreal), (*phone#*, 848-3033)}

<u>Example</u>

Person entity - {(*name*, John), (*city*, Montreal)}

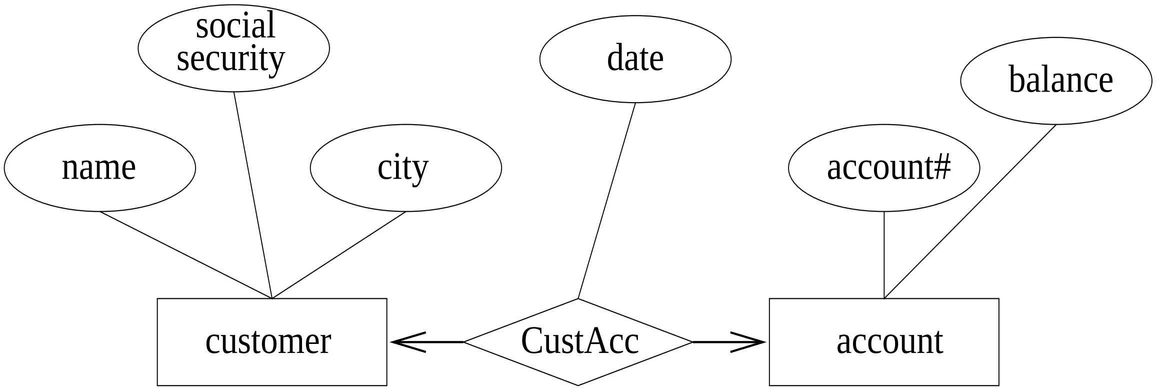
Phone entity set -
{{(*phone#*, 848-3033), (*location*, H-901-2)},
{(*phone#*, 848-3041), (*location*, H-947)}}

Relationship set -
{{(*name*, John), (*city*, Montreal), (*phone#*, 848-3033), (*location*, H-901-2)},
{(*name*, John), (*city*, Montreal), (*phone#*, 848-3041), (*location*, H-947)}}

Mapping Constraints

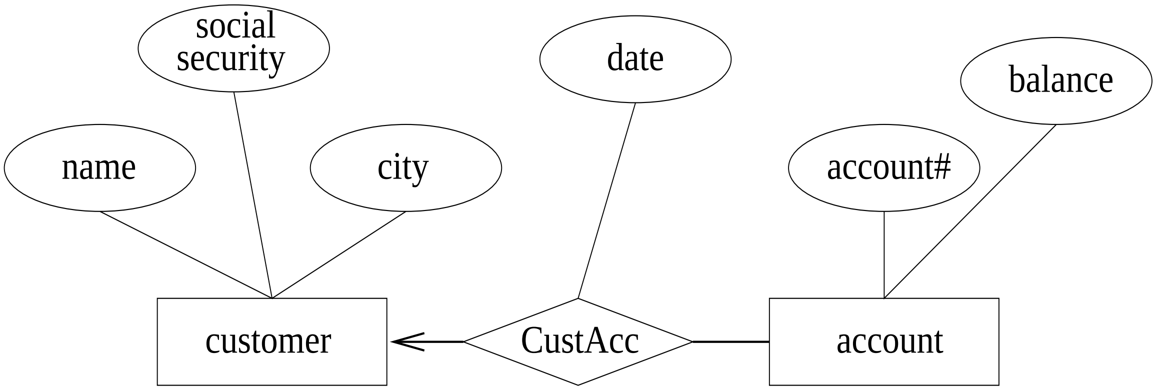
One-to-one: an entity in a set A is associated with at most one entity in another set B.

> In a one-to-one relationship between *customer* and *account*, one customer may have only one unique account. No one else have the same account.

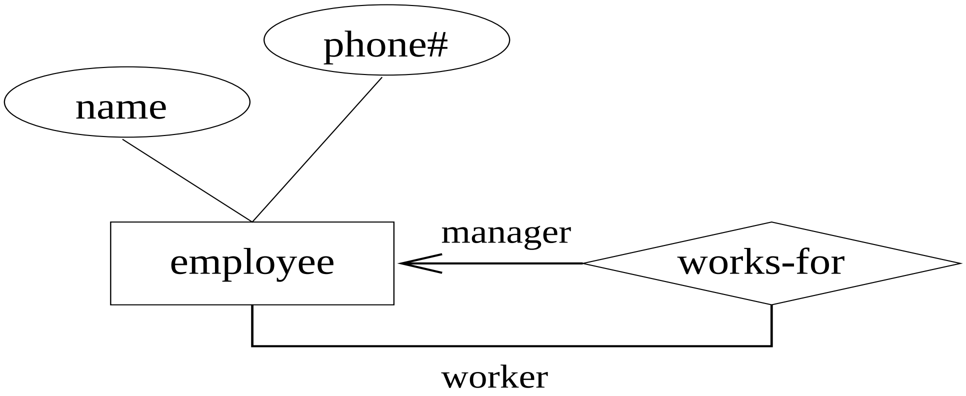


One-to many: an entity in a set A is associated with any number of entities in another set B. But the reverse is not true.

> In a one-to-many relationship between *customer* and *account*, one customer may have several unique accounts. No one else have these accounts.

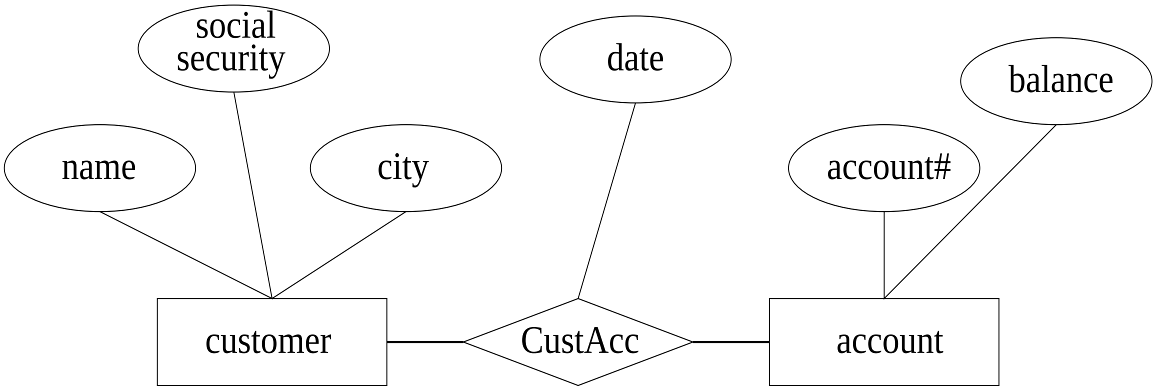


One-to-many with roles



Many-to many: an entity in a set A is associated with any number of entities in another set B, and vice-versa.

> In a many-to-many relationship between *customer* and *account*, one customer may have several accounts, and these accounts may be shared by any number of other customers.

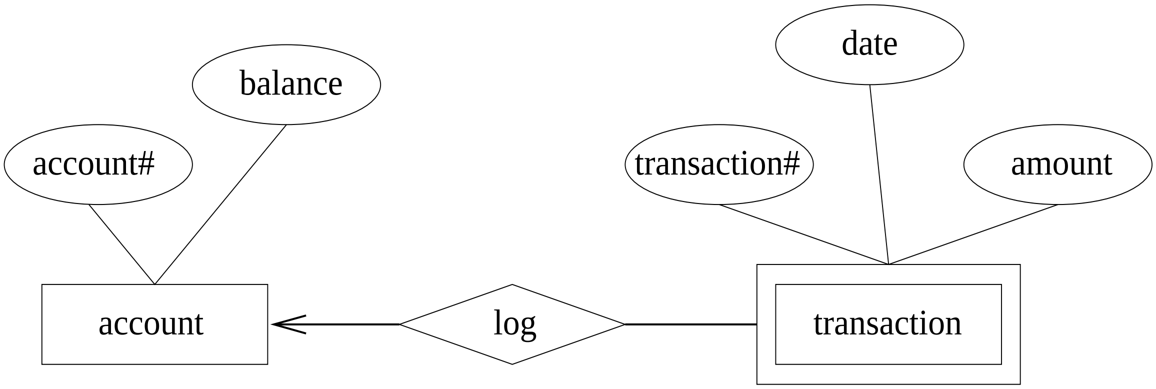


Existence dependencies - dominant entity and subordinate entity.

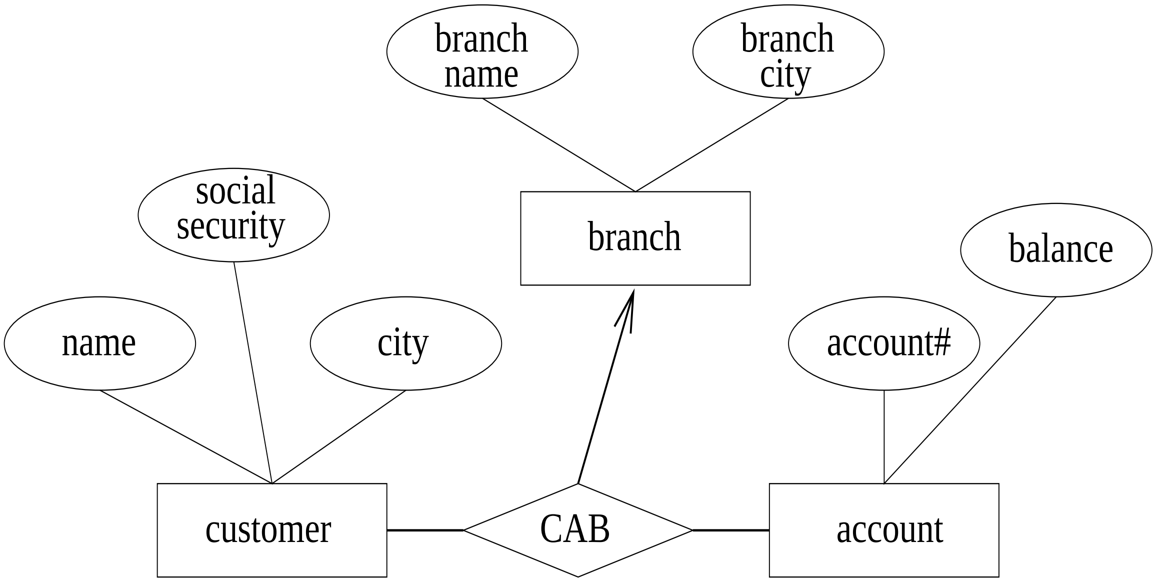
<u>Example</u>

Entity sets – *account* (dominant entity), and *transactions* (subordinate entity).

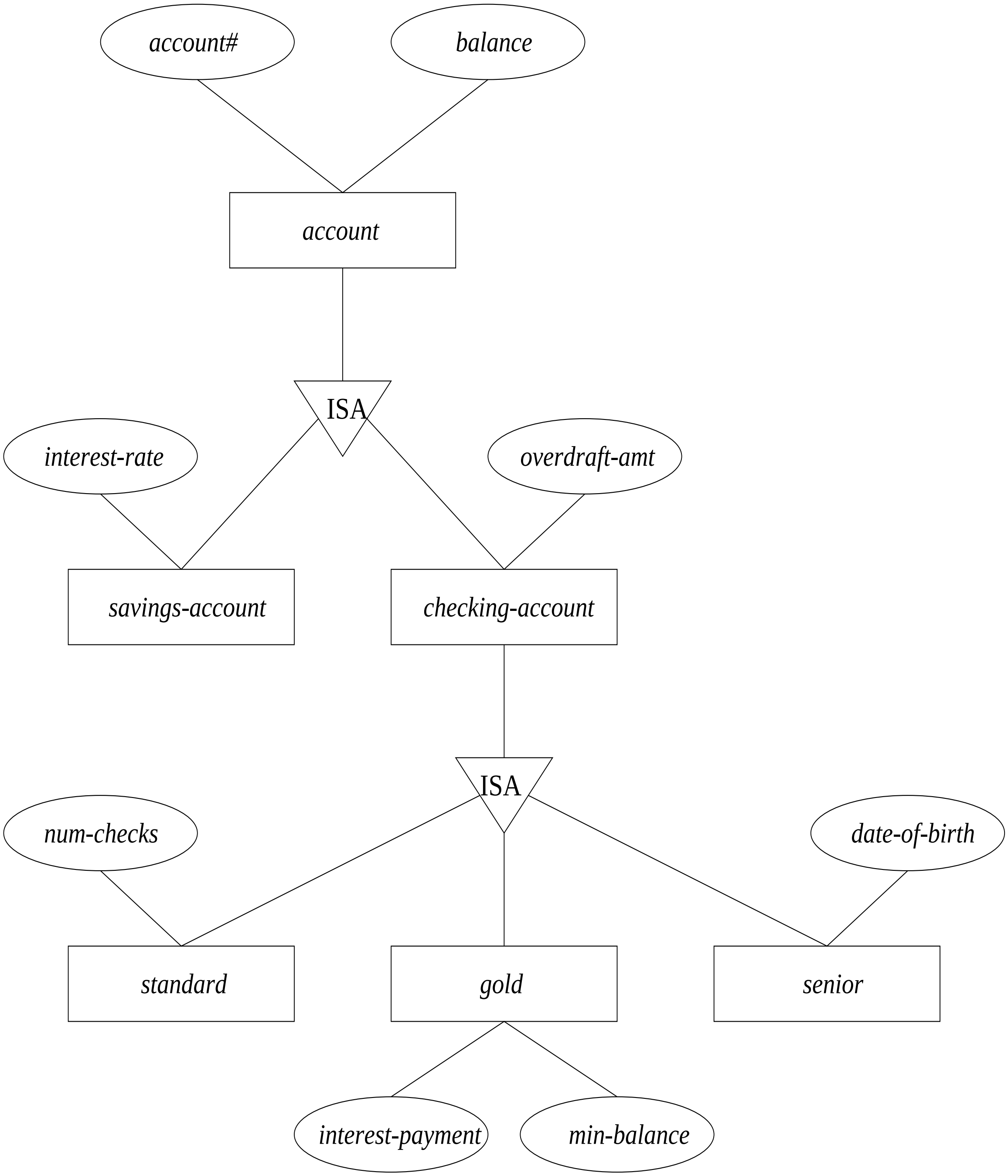
Relationship set – *log*.



Ternary relationship



Generalization/Specialization



Keys

– uniquely identifies an entity in a set of entities.

– help distinguish between entities and relationships.

Superkeys: a set of attributes of an entity set, uniquely identifies an entity in the set, e.g., *customer-name* and *social-security* in the *customer* entity set, and so is *social-security*.

Candidate keys: A candidate key is a superkey for which no proper subset is a superkey, e.g., *social-security* in *customer* entity set. May be more than one.

Primary keys: One of the candidate keys chosen by the designer.

Weak entity set: does not have sufficient attributes to form a primary key. E.g., the *transaction* entity set. Should be a part of one-to-many relationship (with no descriptive attributes) with a strong entity set.

Strong entity set: always has a primary key. E.g., the *customer* entity set.

Discriminator: The discriminator of a weak entity set is a set of attributes that distinguishes among the entities corresponding to a strong entity. E.g., *transaction#* in *transaction*.

Primary key of weak entity sets: Primary key of the strong entity + discriminator of the weak entity. E.g., *account#, transaction#*.

Attributes of relationship sets

Let R be the relationship set involving $E_1, \ldots, E_n$. Then the set of attributes of R is given by

$$\begin{aligned} attribute(R) = primary - key(E_1) \cup \cdots \cup \\ primary - key(E_n) \cup \{a_1, \ldots, a_m\} \end{aligned}$$

where $primary - key(E_i)$ = primary key of E_i, $2 \leq i \leq n$, and a_j, $0 \leq j \leq m$ are the descriptive attributes of R.

Keys of relationship sets

Let R be the relationship set involving $E_1, \ldots, E_n$, with descriptive attributes $\{a_1, \ldots, a_m\}$.

Superkey: $\{primary - key(E_1), \ldots, primary - key(E_n)\}$, if $m = 0$.

Primary key: $\{primary{-}key(E_1), \ldots, primary{-}key(E_n)\}$ is the primary key if it is many-to-many.

$\{primary - key(E_1)\}$ is the primary key if it is many-to one from E_1 to E_2 (assuming $n = 2$).

It is either of E_1 or E_2 if it is one-to-one.

If $m > 0$, depending on the semantics, a subset of a_j may be in the primary key of R.