Topic1

An overview of database and DBMS

From

Chapters 1 and 2 of Fundamentals of Database Systems, Authors: Elmasri and Navathe

Publisher: Addison Wesley -Pearson

By: A. Abhari

CPS510

Ryerson University

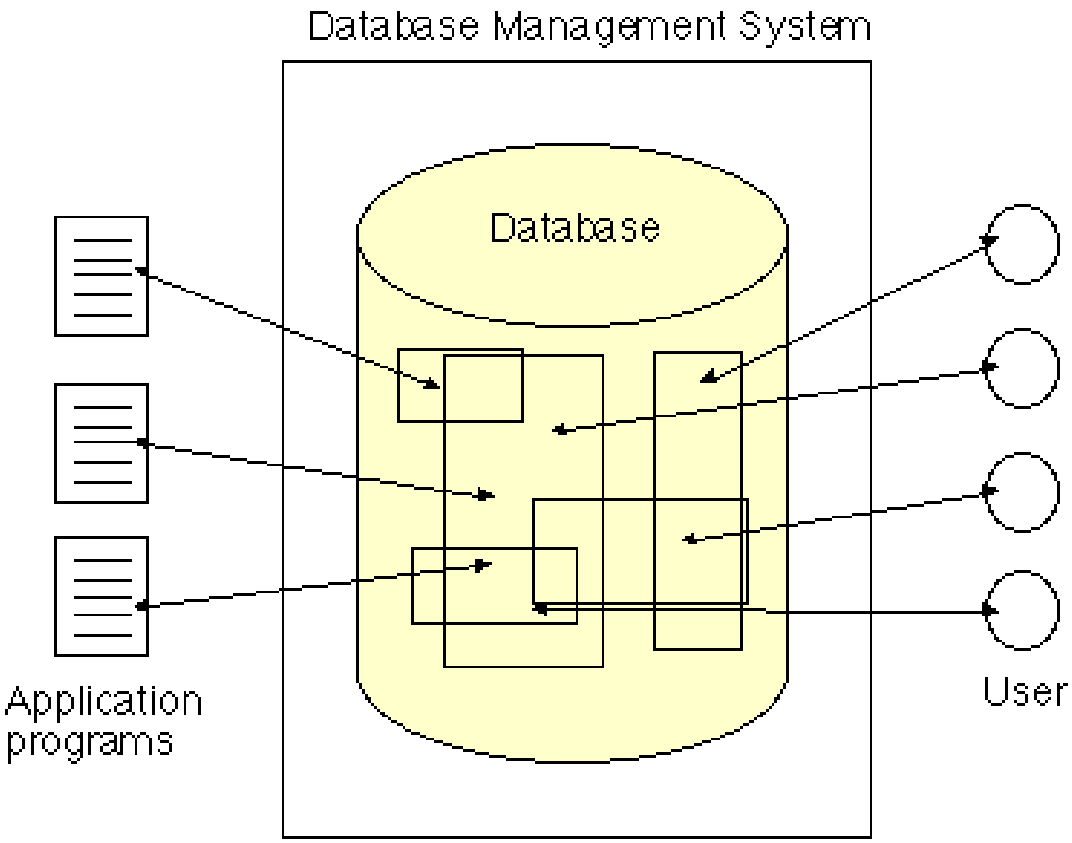
Topics in this Section

- What is a Database System?
- What is a Database?
- Why Database?
- Data Independence
- Relational Systems, and Others

Database System

- Computerized record-keeping system
- A collection of programs to create and maintain database
- Supports operations
  - Add or delete files to the database
  - Insert, retrieve, remove, or change data in database
- Components
  - Data, hardware, software, users

A Simplified Database System



Database System - Data

- Database system may support single user (for small machines) or many users
- When there are many users in organizations:
  - Data is integrated: database is unification of distinct files. Any redundancy among these files partly or wholly is eliminated.
  - Data is shared: Different users can have access to the same data
- Different users will require different views

Database System - Data

- For example a given database can have EMPLOYEE file that shows the information of employees. Also this database can contain an ENROLLMENT file that shows the enrollment of employees in training courses.
- Personnel department uses EMPLOYEE and educational department uses ENROLLMENT files.

EMPLOYEE

NAME	ADDRESS	DEPARTMENT	SALARY	…

ENROLLMENT

NAME	COURSE	…

Database System - Data

- ENROLMENT file does not need the department of employees who took a course because it will be redundant information (integrity).
- DEPARTMENT of employees can be used by the users in personnel and education departments (sharing)
- Although users in personnel and education departments share DEPARTMENT portion but they have different views on database.

Database System - Hardware

- The hardware components of database system consist of the disks in which data are stored thus database system can perform
  - Direct access to subset portions of data
  - Rapid I/O

- Data operated on in the main memory

Database System - Software

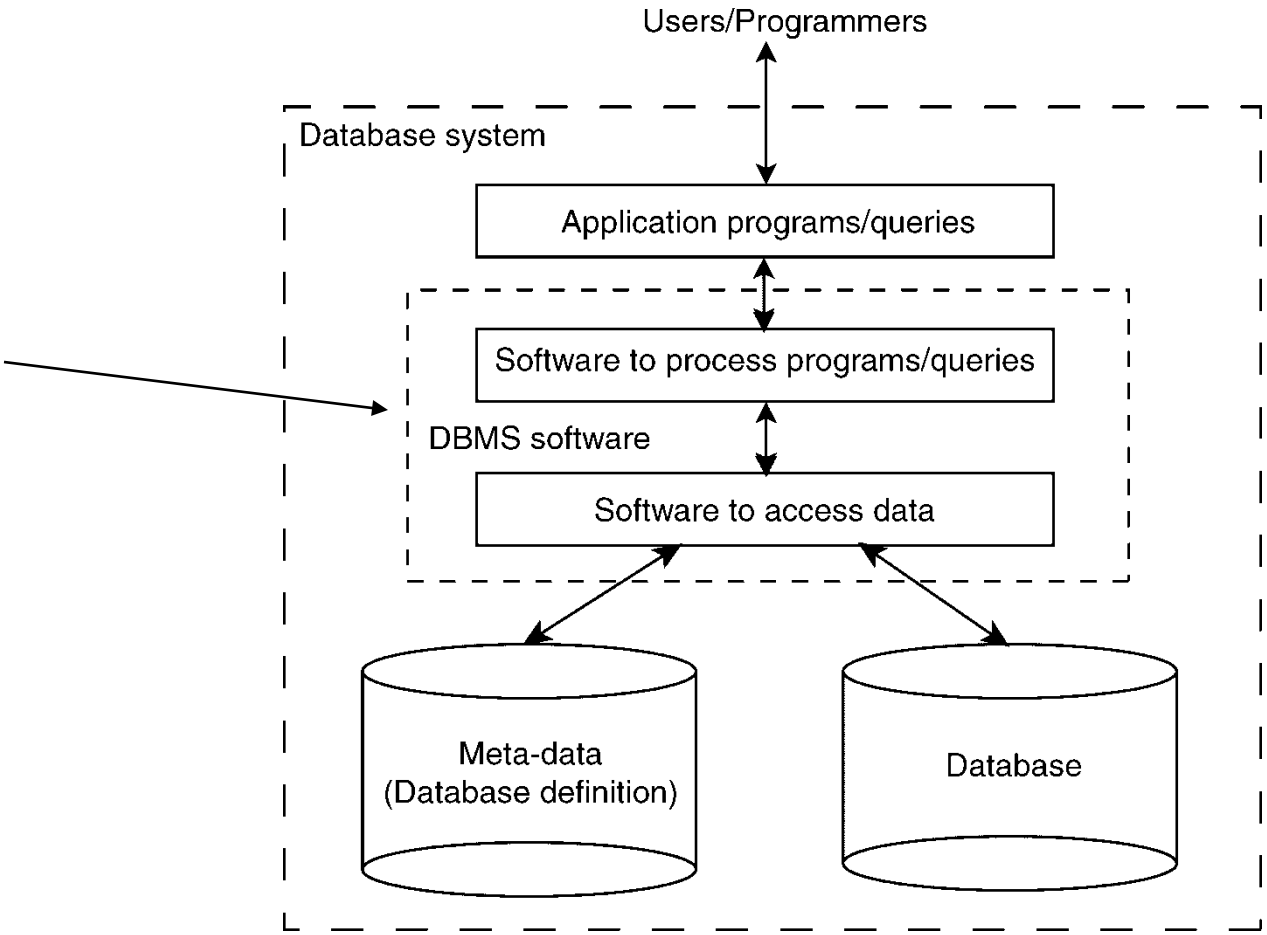
Between physically stored data and users of the systems there is a layer of software referred to as:

- Database manager
- Database server
- Database management system (DBMS)

DBMS shields database users from hardware details

- Note that DBMS is also referred to some products provided by specific vendor. For example DB2

DBMS in a Database System Environment



Database System - Software

- DBMS is not (but may come with)
  - Application Development Tools
  - Application Software
  - Transaction Manager (TP Monitor)
  - Report Writer
  - System utilities
- Note that people often use the term *database* when they really mean *DBMS*. For example "Vendor X's database" is wrong, it should be "Vendor X's DBMS"

Database System - Users

- Application programmers
- End users
- Database Administrators (DBA). DBA is a person or team of IT professional/s whose job is to create the database and put in place the technical controls needed to enforce the various policy decisions made by data administrator. Note that DBA is different from Data administrator (DA). DA's job is to decide what data should be stored and who can perform what operations on data (i.e., data security)

What is a Database?

- Collection of persistent data that is used by the application systems of some given enterprise (enterprise is a self contained commercial, scientific, technical or other organization).
- Collection of true propositions: For example the fact “Supplier S1 is located in London” might be such a true proposition
- Made up of entities, relationships, properties

  (we will talk about it later)

What is a Database? (persistent data)

- Persistent data is different from the data that last for a short time. For example the intermediate results are transient data that last for a short time.
- When persistent data has been accepted by DBMS for entry in database it can be removed from database only by some explicit request to DBMS.
- The earlier term for persistent data was operational data which reflected the original emphasis in database systems for production or operational databases. However databases are now increasingly used for other kind of applications too. For example database can offers decision support via operational data and data warehouse (i.e., summary information)

What is a Database?

- For example here are the example of data that are used in database for following enterprises
  - Student data (for a university)
  - Patient data (for a hospital)
  - Product data (for a company)
- Data can be
  - Static (e.g., part#, SIN)
  - Dynamic (e.g., quantity, balance)
  - Quasi-static (e.g., salary)

Why Database?

- **Shared data**: Not only for existing applications but also new ones can be developed to operate against the same data.
- **Reduced redundancy**: If we need to keep some redundancies it should be controlled. For example the updates should be propagated to all redundant data

Why Database?

- Example of a situation in which redundancy is not completely eliminated

Saving account

account#	customer name	address	balance

Chequeing account

account#	customer name	address	balance

Why Database?

- **Reduced inconsistent data**: Inconsistency happens when one of the redundant data has been updated and the other has not. Propagating updates is used to avoid inconsistency
- **Transaction support**: By putting several update operations in one transaction we can guarantee that either all of them are done or none of them are done. For example the operations for transferring cache from account A to account B can be put in one transaction.

Why Database?

- **Support for data integrity**: Ensures that the data in database is accurate.

For example:

- We shouldn’t have an employee working in non exiting department.
- We shouldn’t have number of hours entered as 400 instead of 40
- Inconsistency can lead to the lack of integrity.

Why Database?

Security enforcement: Ensuring that the only means of access to a database is through proper channels: By:

- Restricting unauthorized data
- Different checks (security constraints) can be established for each type of access (retrieve, insert, delete, etc.)
  - Example: Course marks database
    - A student can have access to his/her own mark
      - Should not be able to see other student's marks
    - TA might have access to enter marks for the current assignment only
      - Should not be allowed to change marks for the other assignments/tests
    - Instructor can have full access to the course database

Why Database?

- **Support for standards**
  - Due to central control of database.
    - » Example
      - – Address must be two lines
        - ✈Each line 40 characters (maximum)

Why Database?

- **Conflicting requirements can be met**
  - Knowledge of overall enterprise requirements as opposed to individual requirements
    - » System can be designed to provide overall service that is best for the enterprise
    - » Data representation can be selected that is good for most important applications (at the cost of some other applications).

Data Independence

- Data independence
  - Traditional file processing is data-dependent
    - Knowledge of data organization and access technique is built into application logic and code
- **Examples of situations in which the stored representation might be subject to change:**
    - An application program written to search a student file in which records are sorted in ascending order by **`student#`** fails if the sort order is reversed
    - Representation of numeric data
      - binary or decimal or BCD
      - fixed or floating point
      - real or complex

Data Independence

- Representation of characters
  - ASCII (uses 1 byte)
  - Unicode (uses 2 bytes)
    - ✈used in JAVA
  - Universal character set (UCS)
    - ✈UCS-2 (uses 2 bytes – essentially Unicode)
    - ✈UCS-4 (uses 4 bytes)
- Unit for numeric data
  - inches or centimeters
  - pounds or kilograms
- Data encoding

  Red = 1, Blue = 2, ...

  == changed to ==> Red = 0, Blue = 1, ...

Data Independence

- In database systems DBMS immune applications to such changes
- In database systems the logical and physical representation of data are separated
- Using database allows changes to application programs without changing the structure of the underlying data and *vice versa*. So the database can grow without impairing existing applications. For example without requiring any changes to the existing applications a “unit cost” field can be added to the “part” stored record in the “parts” stored file of the database shown in Fig 1.7 of the text book.

Relational Systems

- Introduction of relational model in 1969-70 was the most important innovation in database history
- Relational systems are based on logic and mathematics. *Relation* is basically a mathematical term for a table.
- In relational systems data is perceived as tables, only and operators derive new tables from existing
- Relational systems are not pointer based (to the user). Although they may use pointers at the level of physical implementation.

Relational Systems-SUPPLIER Table

supplier#	supplier_name	city
1	Acme Supplies	Toronto
2	Sona Systems	Ottawa
3	AtoZ Systems	New York
4	Quality Supplies	London
5	Best Supplies	Saskatoon

Relational Systems-PART Table

part#	part_name	weight
1	Tower case	2.5
2	Sony display	4.5
3	Mother board	0.6
4	Yamaha speakers	2
5	Power supply	3

Relational Systems-PART-SUPPLIER Table

part#	supplier#	quantity
1	1	20
1	2	34
2	2	23
3	3	33
3	5	43
4	5	45
5	4	22

Relational products

They appeared in the market in late 70s and mostly support SQL. Names of some of these products which are based on the relational system are:

- DB2 from IBM Corp.
- Ingres II from Computer Associate International Inc.
- Informix from Informix Software Inc.
- Microsoft SQL Server from Microsoft Corp.
- Oracle 11g from Oracle Corp.
- Sybase Adaptive Server from Sybase Corp.

Not Relational Systems

- Hierarchic
- Network
- Inverted List
- Object
- Object/Relational
- Multi-dimensional

We will talk about them later

Topic 2

Database System Architecture

From

Chapters 1 and 2 of Fundamentals of Database Systems, Authors: Elmasri and Navathe

Publisher: Addison Wesley - Pearson

By: Abdolreza Abhari

CPS510

Ryerson University

Topics in this Section

- Three levels of architecture
- Mappings
- Database Administrator (DBA)
- Database Management System (DBMS)
- Database Communications
- Client/Server Architecture
- Utilities
- Distributed Processing

Data Modeling: Schemas and Instances

- Before start to talk about database architecture note that in any data model, it is important to distinguish between
  - » description of the database (database ***schema***)
  - » database itself (***instance*** of a database)
- Database schema
  - * Describes the database
  - * Specified during the database design phase
    - » Not expected to change frequently
  - * Most data models have a notation for graphical representation of schema

Data Modeling: Schemas and Instances

Example schema: SUPPLIER-PARTS database

SUPPLIER

supplier#	supplier_name	city

PART

part#	part_name	weight

PARTS_SUPPLIER

part#	supplier#	quantity

Data Modeling: Schemas and Instances

- Database instance
  - » Refers to the data in the database at a particular moment in time
  - » Many database instances can correspond to a particular schema
  - » Every time we insert, delete, update the value of a data item, we change one instance of database to another
  - » DBMS is partially responsible for ensuring that every instance satisfies
    - – Structure and constraints specified in the database schema
  - » See the example instance of SUPPLIER-PARTS database shown before

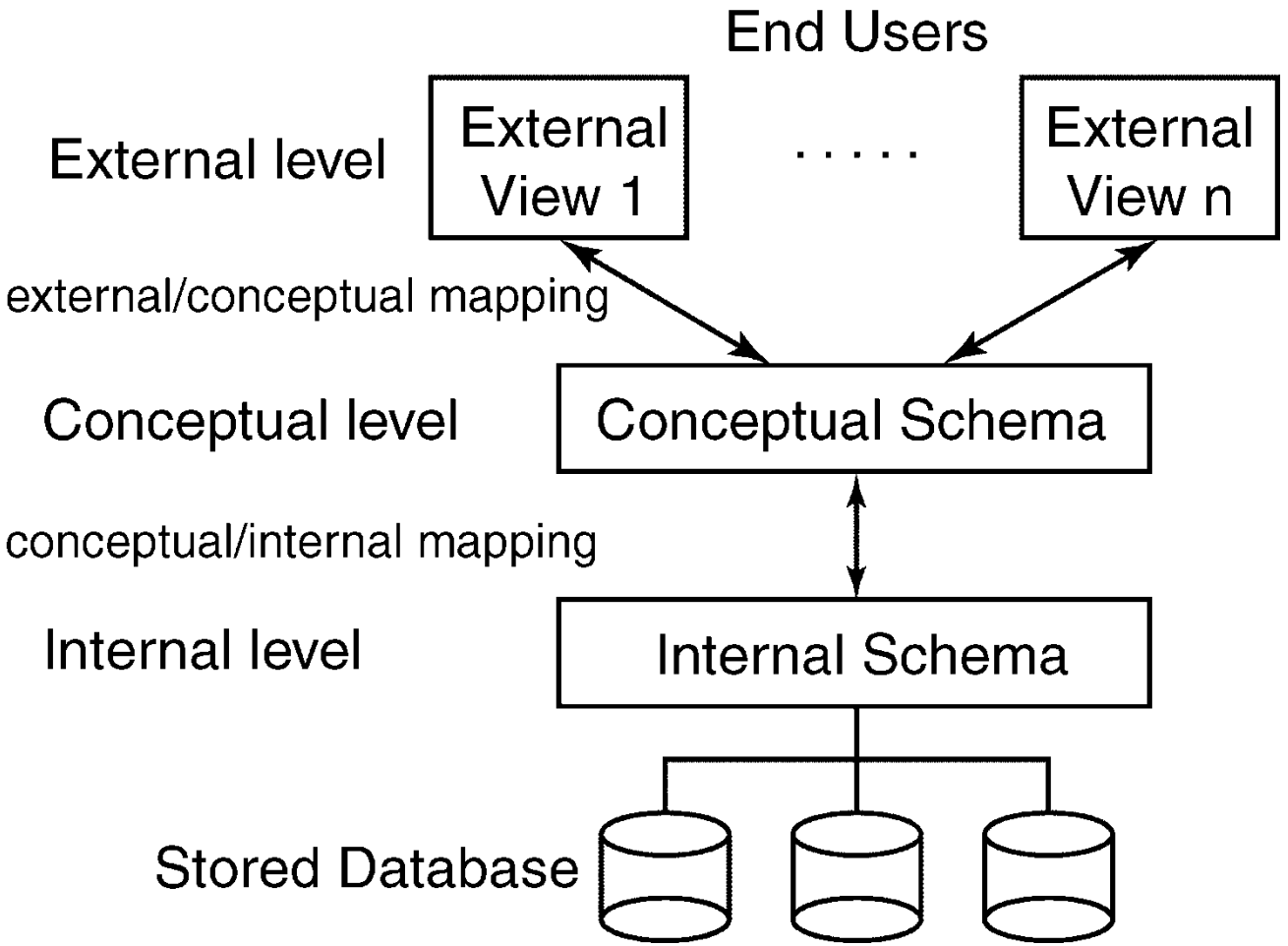
Three Levels of Architecture

- Three level architecture is also called ANSI/SPARC architecture or three schema architecture
- This framework is used for describing the structure of specific database systems (small systems may not support all aspects of the architecture)
- In this architecture the database schemas can be defined at three levels explained in next slide

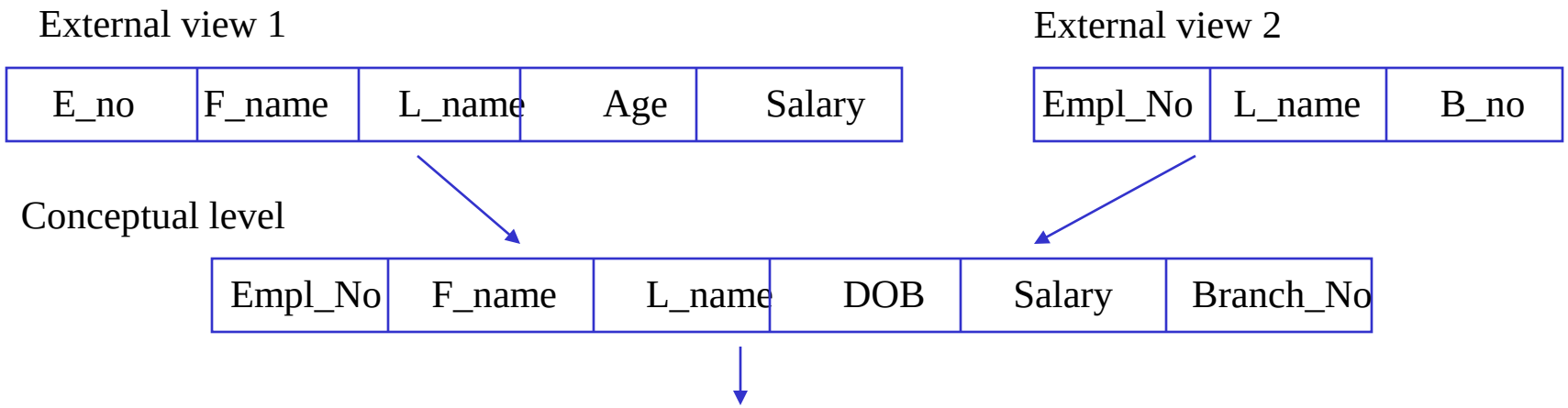
Three Levels of Architecture

- Internal level: Shows how data are stored inside the system. It is the closest level to the physical storage. This level talks about database implementation and describes such things as file organization and access paths. Note that relational model has **nothing** explicit to say regarding the internal level
- Conceptual level: Deals with the modeling of the whole database. The conceptual schema of database is defined in this level
- External level: This level models a user oriented description of part of the database. The views for individual users are defined by means of external schemas in this level

Three Levels of Architecture



Three Levels of Architecture-Example



Internal level

```
struct EMPLOYEE {
    int Empl_No;
    int Branch_No;
    char F_name [15];
    char L_name [15];
    struct date Date_of_Birth;
    float Salary;
    struct EMPLOYEE *next;       //pointer to next employee record
    };   index Empl_No; index Branch_No;      //define indexes for  employees
```

Mapping

- Mapping is the key for providing data independence. Here is more explanation on data independence based on three-level architecture.
- Data independence is the capacity to change the schema at one level without having to change the schema at the next higher level
- Two types of data independence are
  - Logical data independence
  - Physical data independence

Mapping - Data Independence

- Logical data independence (provided by external/ conceptual mapping)
  - Ability to modify conceptual schema without changing
    - External views
    - Application programs
  - Changes to conceptual schema may be necessary
    - Whenever the logical structure of the database changes
      - ✈Due to changed objectives
  - Examples
    - » Adding a data item to schema
      - Adding price of a part to PART table
    - » Adding PROJECT table to the SUPPLIER-PARTS database

Mapping - Data Independence

- Physical data independence (provided by conceptual/internal mapping)
  - Ability to modify internal or physical schema without changing
    - Conceptual or view level schema
    - Application programs
  - Changes to physical schema may be necessary to
    - Improve performance of retrieval or update
      - Example: Adding a new index structure on ***`city`***
- Achieving logical data independence is more difficult than physical data independence
  - Because application programs heavily rely on the logical structure of the data they access

Database Administrator

- Participates in conceptual database design
- Determines how to implement conceptual schema
- Teach users, and help them report
- Implement security and integrity
- Implement unload/reload utilities
- Monitor and tune database performance

DBMS (Languages)

- Users interact with database with data sublanguage (embedded within a host language) which consists of at least two types of languages
  - DDL: To define the database
    - Used to define the database
      - ✈ For defining schemas at various levels
    - Required in building databases
    - DBA and database designers are typical users
    - Commonly referred to as ***data definition language***
  - DML: To manipulate data
    - Used to construct and use the database
      - ✈ Facilitates retrieval, insertion, deletion and updates
    - Typical users are “End Users”
    - Referred to as ***data manipulation language***

Database Management System

- DDL processor / compiler
- DML processor / compiler
- Handle scheduled and *ad hoc* queries
- Optimizer and run-time manager
- Security and integrity
- Recovery and concurrency
- Data dictionary :The **data dictionary** is a system database that contains "data about the data“. That is *definitions* of other objects in the system, also known as *metadata*
- Performance tuning utilities

Support for System Processes

- Data Communications interface
- Client Server Architecture
- External tool support: query, reports, graphics, spreadsheets, statistics
- Utilities: unload/reload, stats, re-org
- Distributed processing

Topic 3

ER Model and Deriving Relational Schema

From

Chapters 3 and 4 of Fundamentals of Database Systems, Authors: Elmasri and Navathe

Publisher: Addison Wesley -Pearson

By: Abdolreza Abhari

CPS510

Ryerson University

Topics in this Section

- Design of the conceptual schema
- Entity-relationship (ER) model
  - Entities
  - Relationships
  - Attributes
- ER diagrams
- Deriving relational schema from ER model

Design of the Conceptual Schema

- *Stage one:* Choice of model: User requirements and real world concepts should go into the design model. If using E-R model, E-R diagram and relational schemas are the results of this stage.

Design of the Conceptual Schema

- *Stage one:* Choice of model: User requirements and real world concepts should go into the design model. If using E-R model, E-R diagram and relational schemas are the results of this stage.
- *Stage two:* Normalization: Conceptual schema are then used in normalization. This further leads to adjusted diagrams and a normalized relational model. (will be discussed later)

Design of the Conceptual Schema

- *Stage one:* Choice of model: User requirements and real world concepts should go into the design model. If using E-R model, E-R diagram and relational schemas are the results of this stage.
- *Stage two:* Normalization: Conceptual schema are then used in normalization. This further leads to adjusted diagrams and a normalized relational model. (will be discussed later)
- *Stage three:* Optimization: (will be discussed later). The outcome of this stage is the data dictionary and the database description.

Entity-Relationship Model

- Entity-relationship (ER) model
  - Introduced by Peter Chen in 1976
- ER model consists of
    - Entities
    - Relationships
    - Attributes
- No single, generally accepted notation
  - Note that
    - You may find variants of the notation used here
    - You may also find symbols different from the ones we use

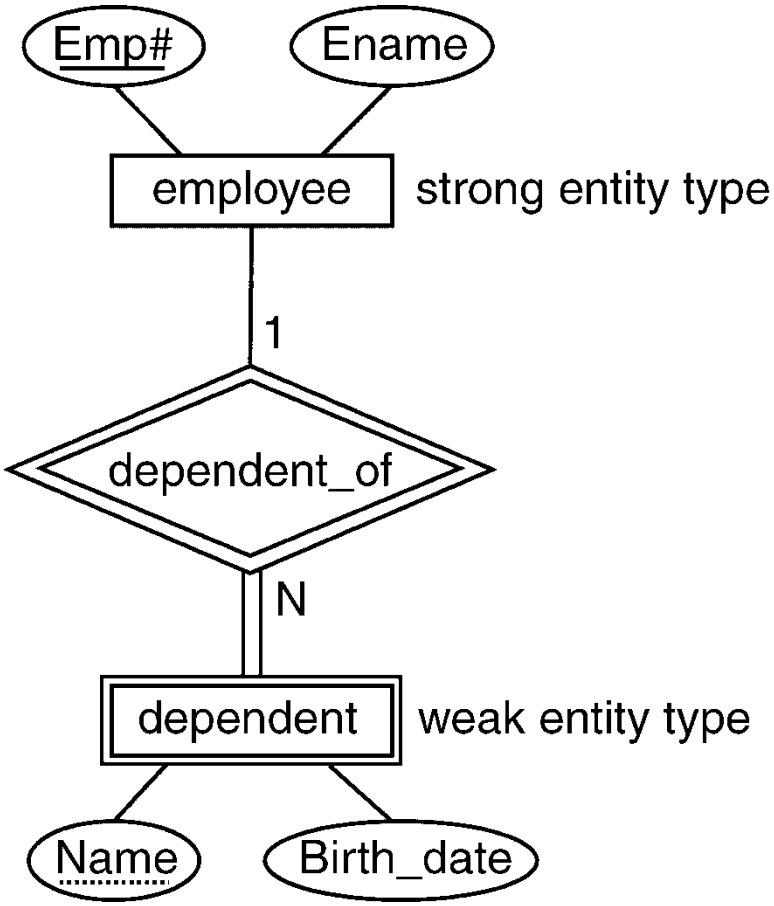
Entities

- Entity is a “thing in the real world with an independent existence”
  - Two types of entities
    - Strong/regular entity (simply called *entity*)
    - Weak entity
  - Strong entity types are called *owner* or *dominant* entity types
    - Exist on their own
  - Weak entity types are called *dependent* or *subordinate* entity types
    - Existence of weak entity depends on the existence of another strong entity

Entities (cont'd)

Weak Entity Example

- Every dependent must be associated with an employee
- If an employee is deleted from the database, dependent must also be deleted
- We can delete a dependent without affecting employee
- Weak entity type is indicated by double outlined box



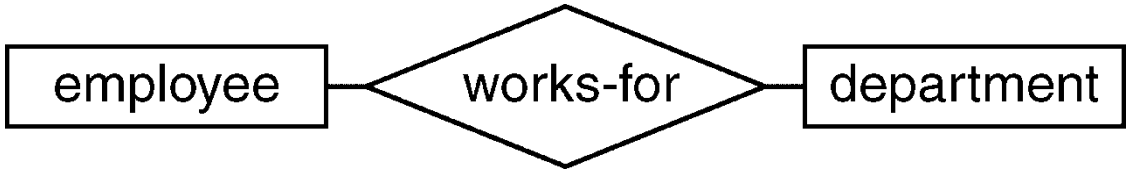
Relationships

A relationship associates entities with one another

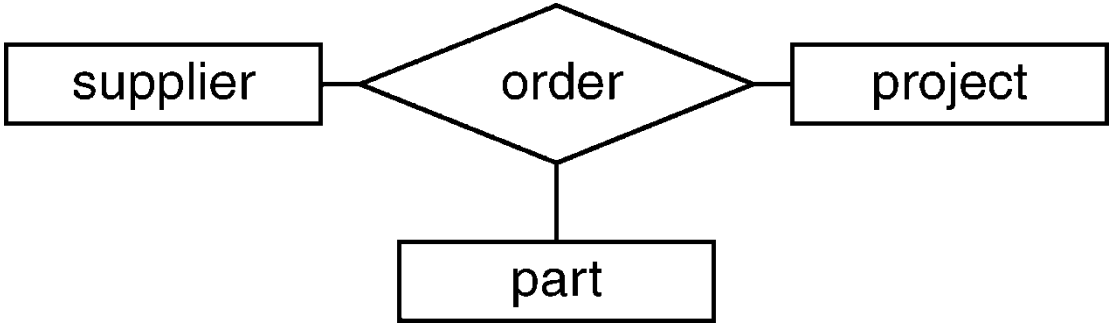
Degree of a relationship type is the number of participating entity types

Examples

- Binary relationship



- Ternary relationship



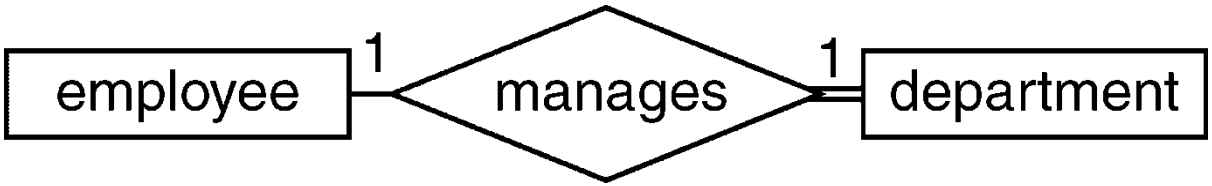
Relationships (cont'd)

- Mapping constraints
  - one-to-one (1:1)
  - one-to-many (1:N)
  - many-to-many (M:N)

Examples

one-to-one

- A manager can manage only one department
- Each department can have only one manager

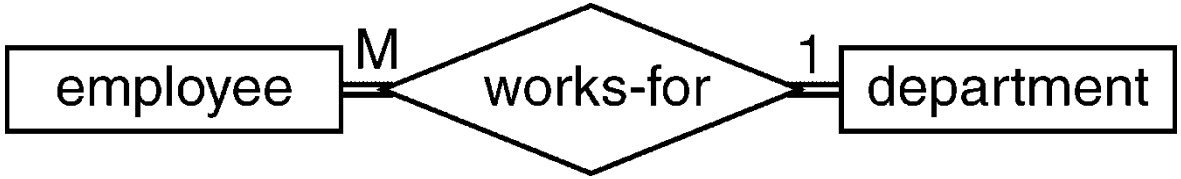


Relationships (cont'd)

one-to-many

» A department can have several employees

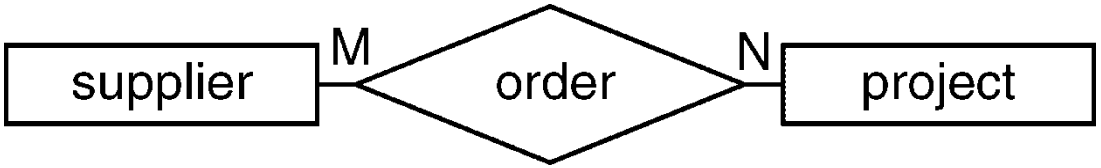
» Each employee may work in only one department



many-to-many

•A supplier can supply parts to several projects

•A project can receive parts from several suppliers

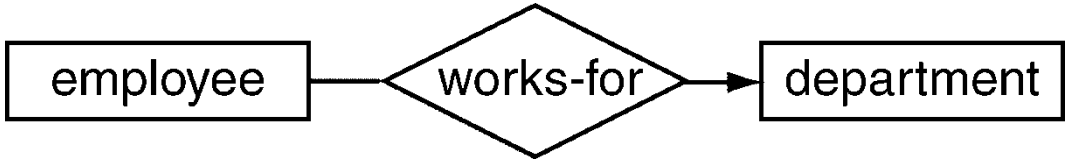


Relationships (cont'd)

- Alternative notation for mapping constraints
  - Uses directed line to represent one-to-many or one-to-one mapping

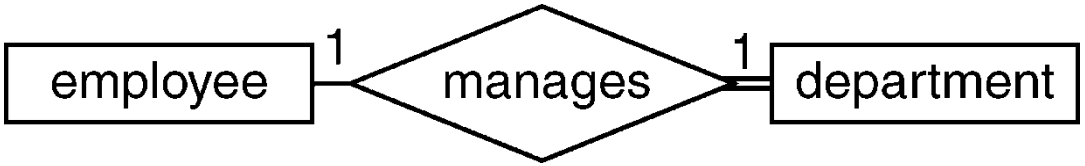
•one-to-one example

•one-to-many example



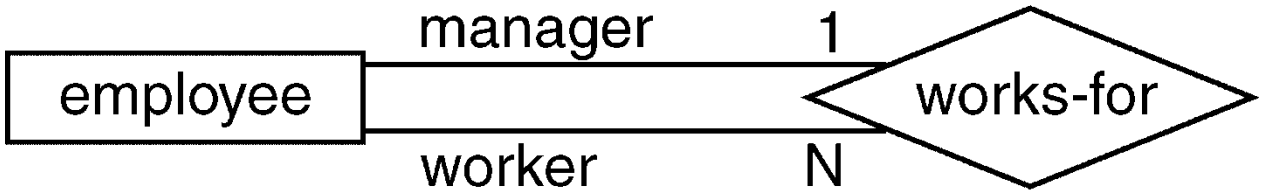
Relationships (cont'd)

- Participation constraints
  - Two types
    - » Total participation (indicated by double lines)
      - – Every department must be managed by a manager
      - – **`department`** participation in **`manages`** relationship is total
    - » Partial participation (indicated by a single line)
      - – Not every employee manages a **`department`**
      - – **`employee`** participation in **`manages`** relationship is partial



Relationships (cont'd)

- Recursive relationships
  - Each entity in a relationship plays a *role*
  - In a recursive relationship
    - An entity participates more than once in different roles
- Example
  - 1:N recursive relationship on **`employee`** entity
    - Each manager manages several workers
    - Each worker may have only one manager



Attributes/Properties

- Describe properties of entities and relationships
  - » Both entities and relationships can have attributes
- Types of attributes
  - » Simple or composite
  - » Single-valued or multi-valued
  - » Stored/based or derived
  - * An attribute can be a
    - – Key or non-key
  - * An attribute can have a
    - – Null value

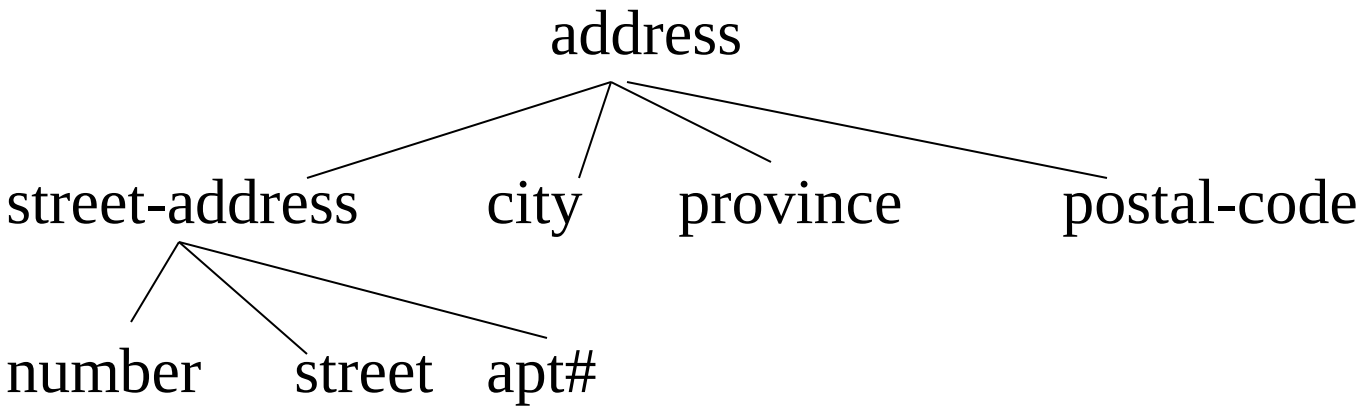
    in some circumstances

Attributes (cont'd)

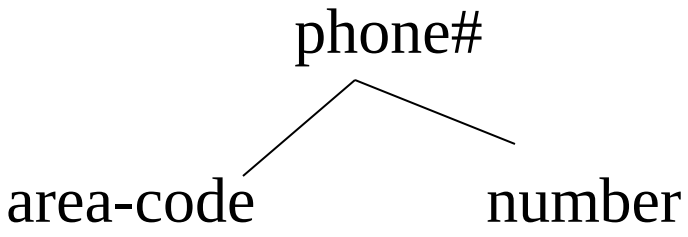
- Simple versus composite attributes
  - Simple attributes
    - Not divisible
      - called *atomic* attributes
    - Examples
      - **`part#, weight`**
  - Composite attributes
    - Consists of several simple attributes
    - Useful if the user refers it
      - sometimes as a unit
      - other times as individual components

Attributes (cont'd)

Example: Need a list of all suppliers located on Yonge street



Example: Require a list of all customers in 416 area code



Attributes (cont'd)

- Single- versus multi-valued attributes
  - Single-valued
    - Examples: **`SIN`**, **`part-weight`**
  - Multi-valued
    - Examples: **`college-degrees`**, **`skills`**

- Stored versus derived attributes
  - Stored attribute
    - Example: **`date-of-birth`**
  - Derived attribute
    - Example: **`age`**

Attributes (cont'd)

- Key or non-key attributes
  - Key attribute
    - An attribute that is unique
      - distinct for each individual entity instance
      - Examples: **`emp#, SIN, student#`**
      - Can used to identify an entity
  - Key attributes are shown underlined in the ER diagram
    - A key attribute may not be a single attribute
      - All attributes that form the key are shown underlined
      - We show only one key attribute
        - ✈Different notation is used in the text (not recommended)

Attributes (cont'd)

Keys and Identifiers

- Each entity in an entity type needs to be identified uniquely
  - Sometimes artificial attributes are created to facilitate
    - E.g. **`student#, employee#`**
  - One or more attributes can be used as an entity identifier
    - For **`marks`** entity type, **`student#`** and **`course#`** are required to find the **`grade`**

Attributes (cont'd)

- Candidate key
  - » Minimal subset of attributes that uniquely identifies an entity
    - – Example: **employee#**
      SIN
- Primary key
  - » The candidate key chosen by the designer to access each entity
    - – Example: **employee#**
  - » Can be defined for strong entities
  - » Weak entities may not have primary keys associated with them
  - » Note:
    - – Strong and weak only from a particular application point of view
      - ✈Not inherent in the physical world

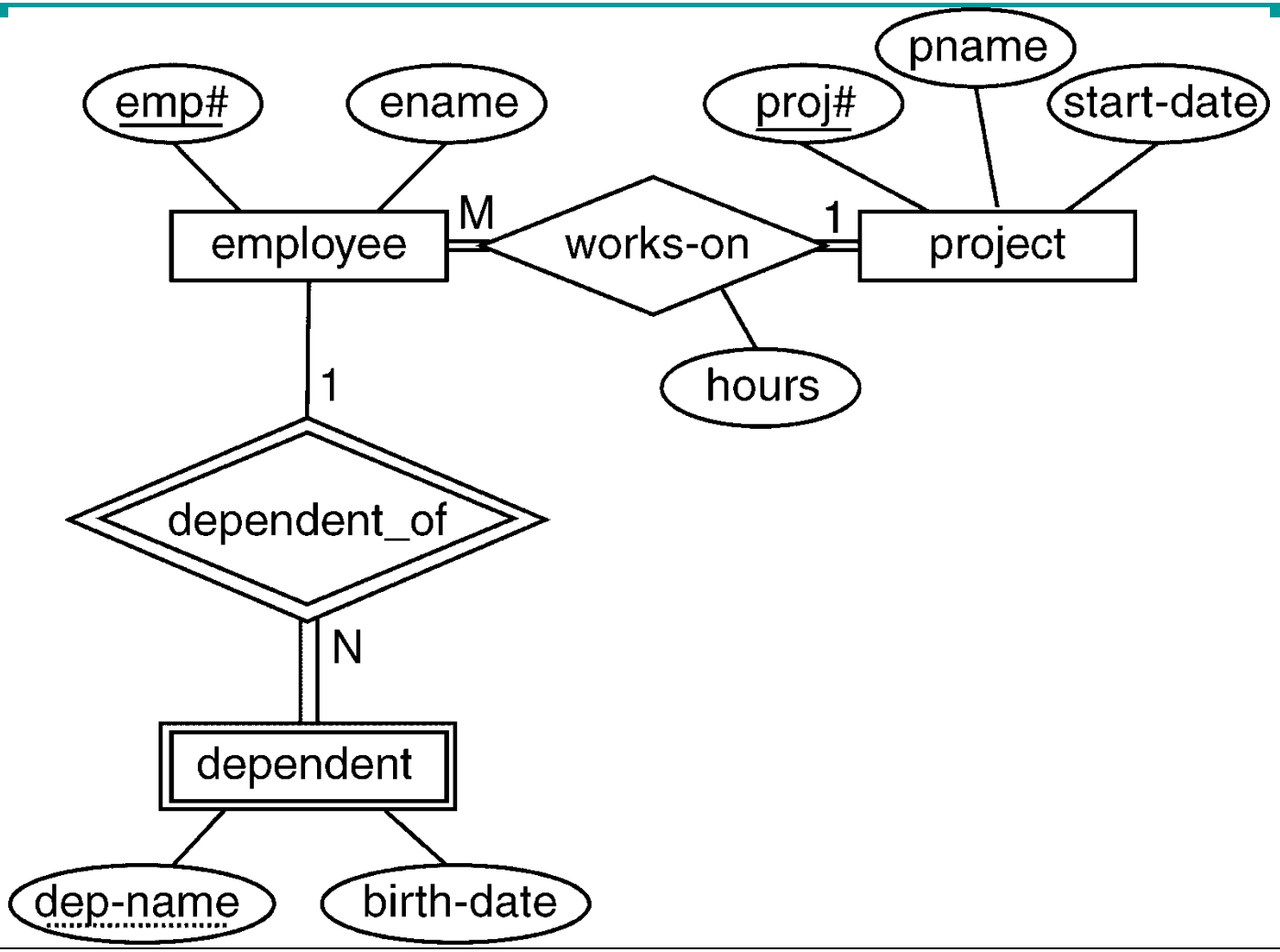
Attributes (cont'd)

- Primary key for weak entity types
  - The entity **`dependent`** cannot be identified uniquely
    - Several people may have the same name
    - We need to identify different dependents of a particular employee
  - Primary key of a weak entity type is formed by the primary key of the associated strong entity plus the weak entity discriminator
  - Example
    - **`Emp#, dep-name`** may serve as a primary key for the weak entity type **`dependent`**

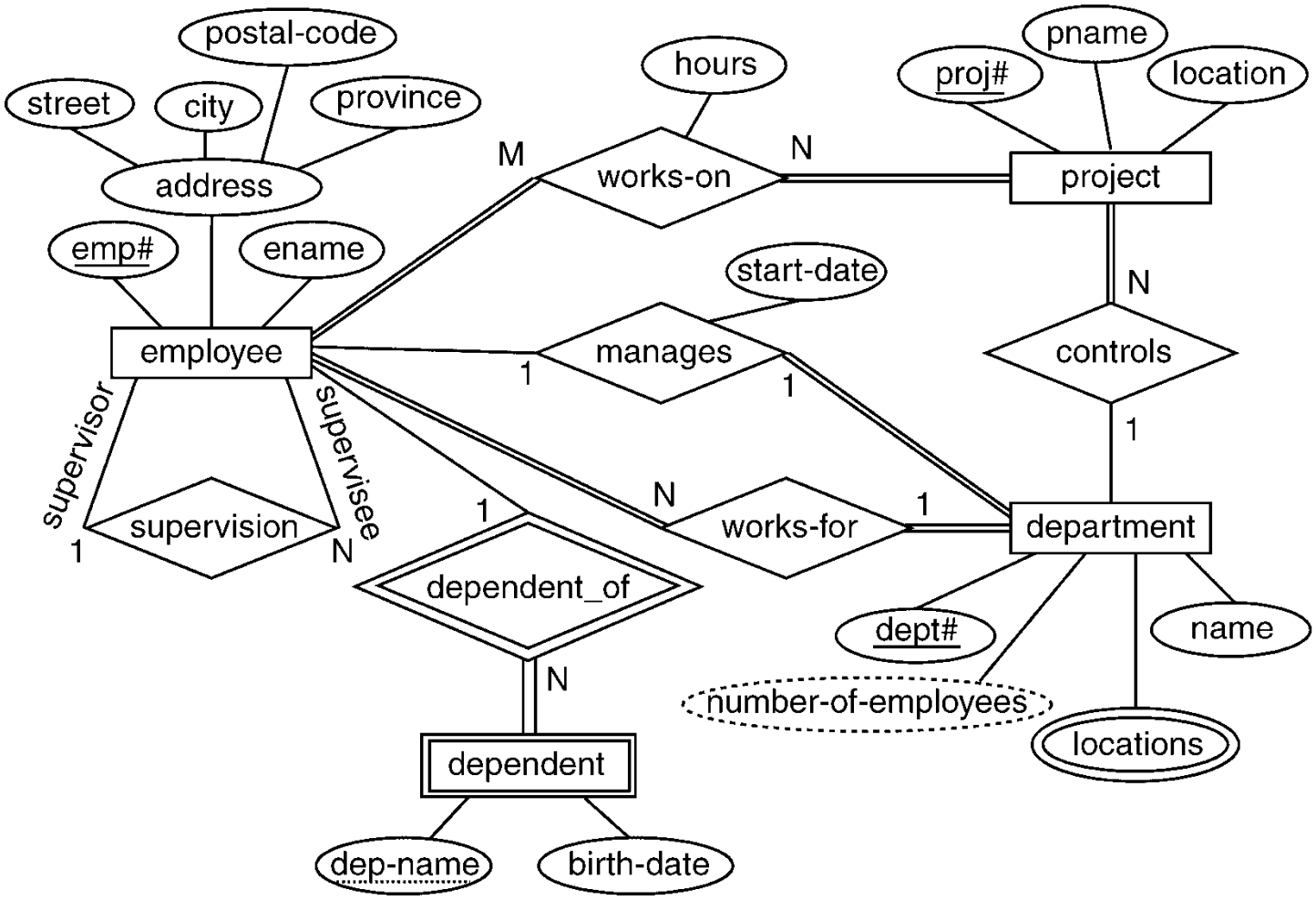
Attributes (cont'd)

- Null values
  - A special attribute value NULL is created to represent various things
    - » Not applicable
      - – A single-family home may not have **`apt#`** attribute
    - » Unknown
      - – missing information
        - ✈Not known at this time
        - ✈Examples: **`citizenship`**, **`grade`**
      - – not known
        - ✈We don't know if the attribute value exists
        - ✈Example: **`email-address`**

ER Diagram Example - 1



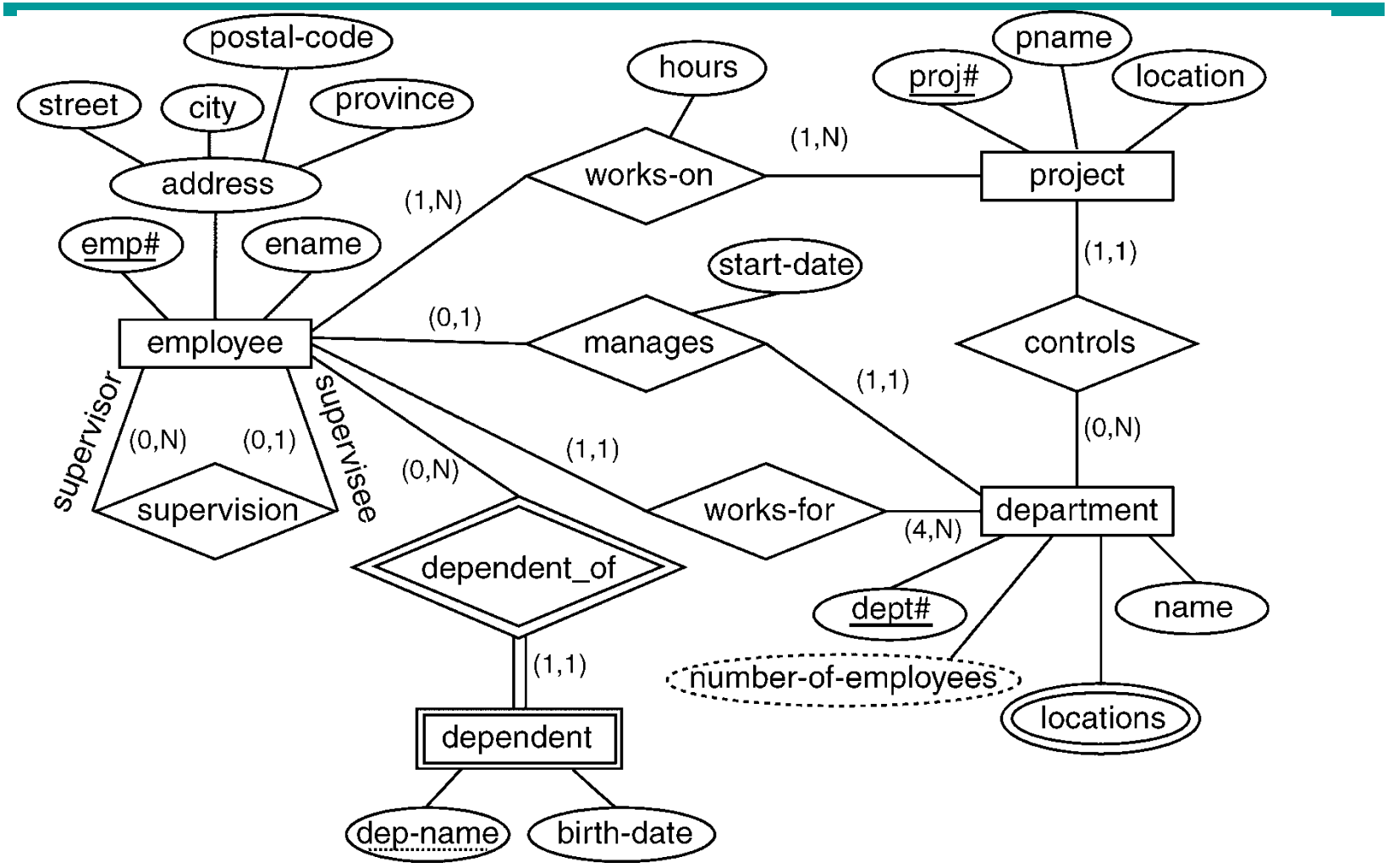
ER Diagram Example - 2



Alternative Notation for Structural Constraints

- Associate a pair of integer numbers
  - (**`min`**, **`max`**) where $0 \leq$ **`min`** $\leq$ **`max`**
  - Each entity must participate in at least **`min`** at most **`max`** relationship instance *at all times*
- More flexible mapping constraints than the three types described before
  - Can easily be applied to relationships of any degree
- Participation constraints can also be specified
  - **`min`** = 0 implies partial participation
  - **`min`** > 0 implies total participation

ER Diagram Example with (min, max)

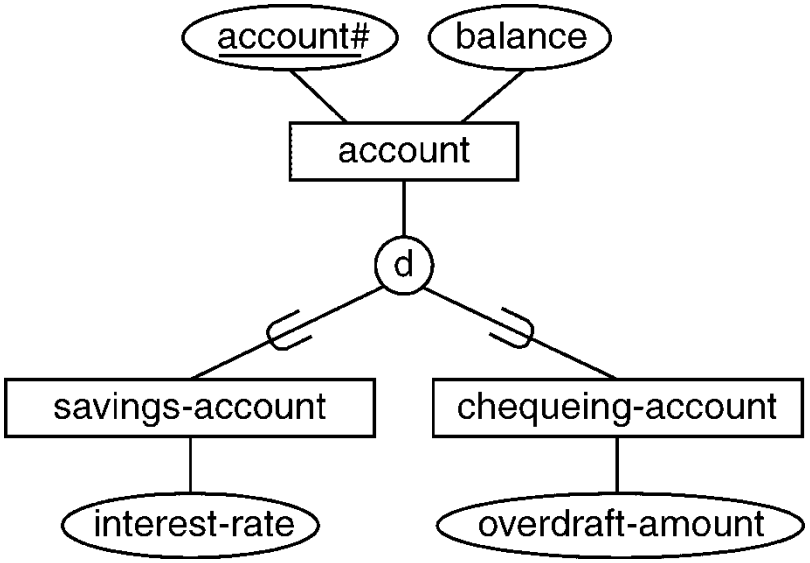


Subclass and Superclass

- Subclass(Subtypes) and superclass(Supertypes)
  - Subclass allows sub-groupings of entities
  - **`student`** entity type can have **`part-time`** and **`full-time`** student subclasses
  - **`student`** is said to be *superclass*
  - Attribute inheritance
    - » Member of a subclass inherits all the attribute of its superclass
    - » Each subclass can have its own attributes
      - – in addition to the inherited attributes

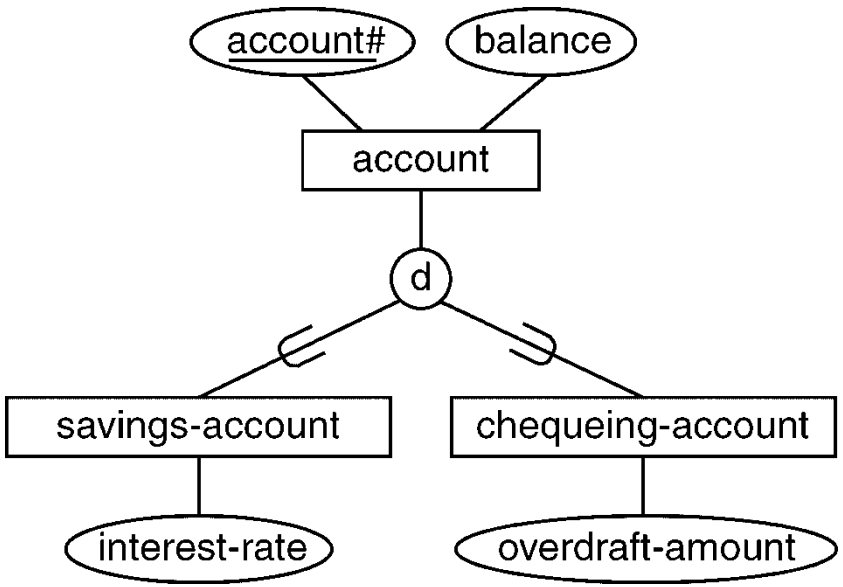
Specialization

- Specialization
  - Process of defining a set of subclasses of an entity type
    - » Usually based on some distinguishing characteristic of the entity type
    - » Multiple specializations can be defined on a single entity type
  - Example
    - » **account** can be specialized into **savings-account** and **chequeing-account**

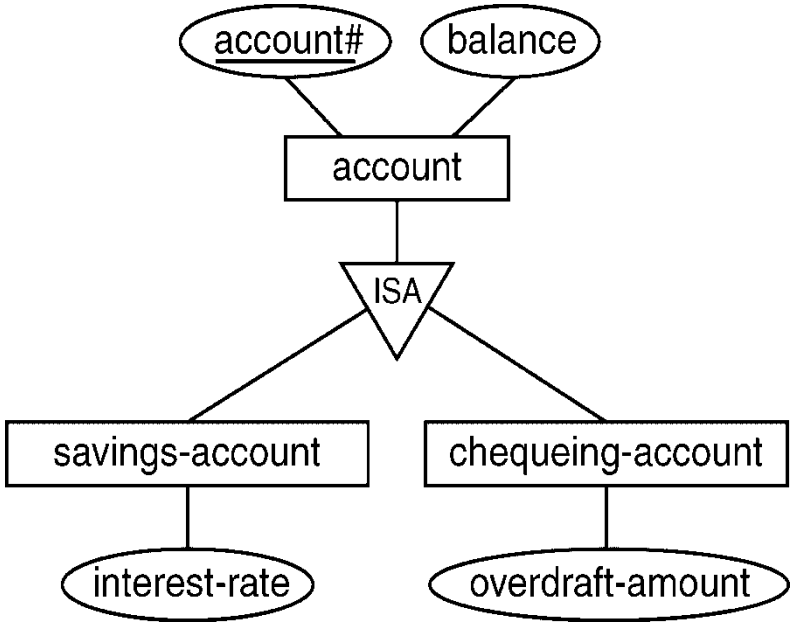


Specialization (cont'd)

Our notation



ISA notation



Specialization (cont'd)

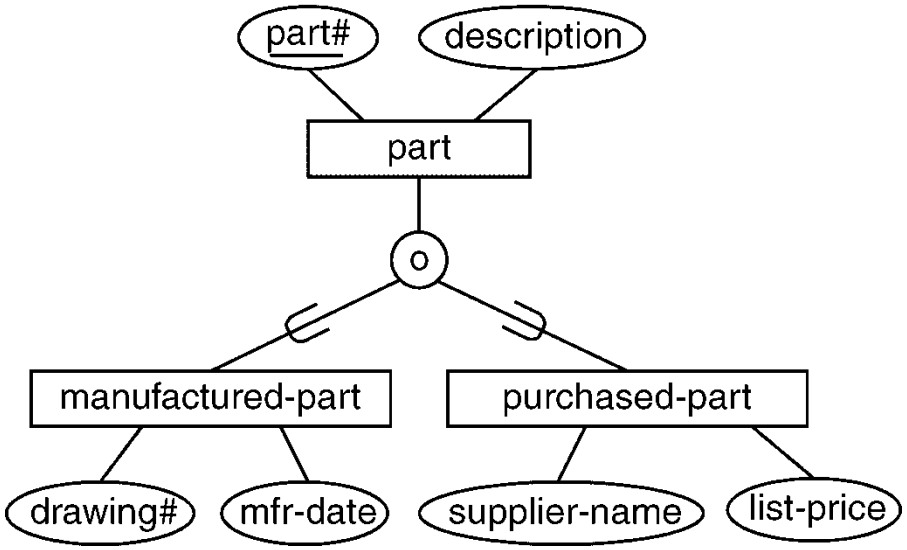
- Two constraints
  - » Disjointness constraint
  - » Completeness constraint
- Disjointness constraint
  - * Disjoint
    - » An entity can be a member of at most one of the subclasses of the specialization
    - » We use "d" in ER diagrams to represent disjoint constraint
  - * Overlapping
    - » The same entity can be a member of more than one subclass of the specialization
    - » We use "o" in ER diagrams to represent overlapping constraint

Specialization (cont'd)

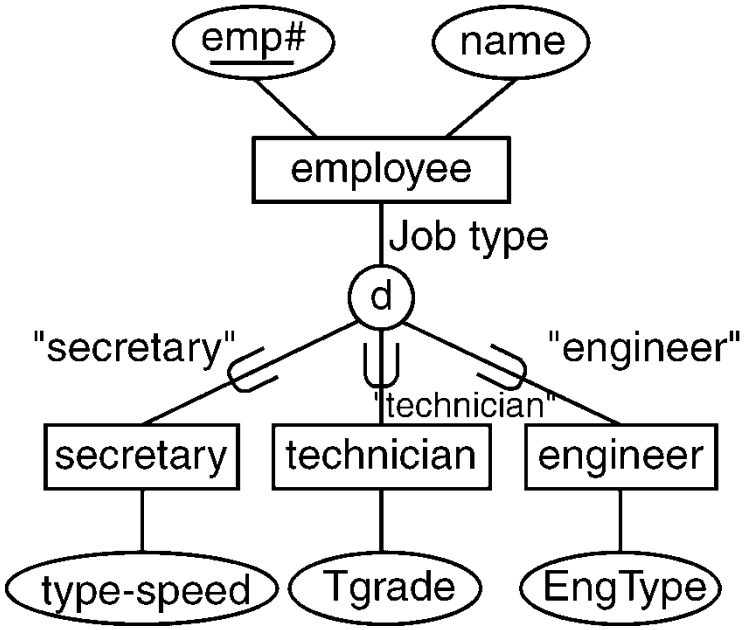
- Completeness constraint
  - Total
    - Every entity in the superclass must be a member of some subclass in the specialization
  - Partial
    - An entity may not belong to any of the subclasses in the specialization
- This leads to four types of specialization
    - disjoint, total
    - disjoint, partial
    - overlapping, total
    - overlapping, partial

Specialization (cont'd)

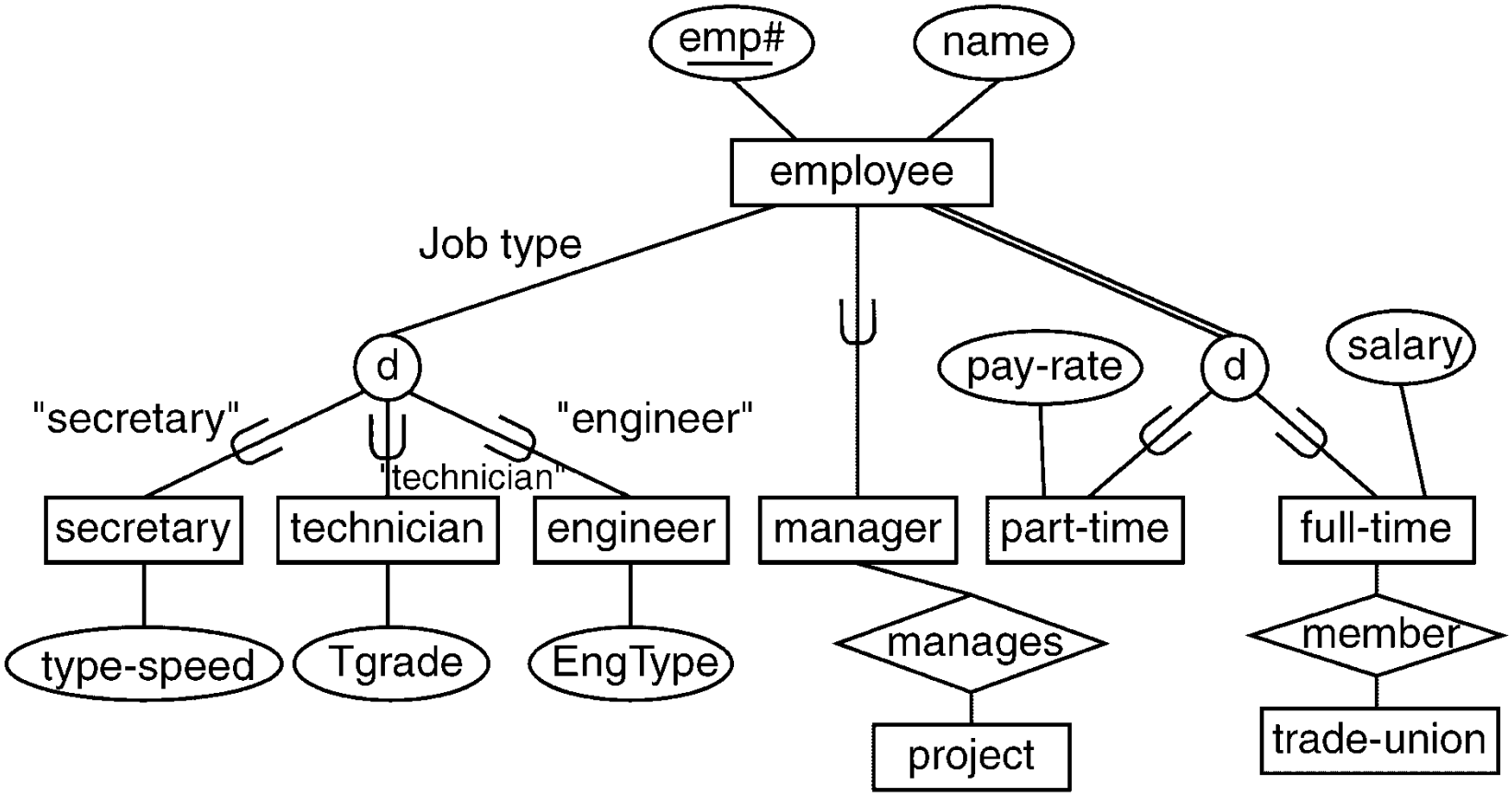
Specialization with overlapping subclasses



Attribute-defined specialization



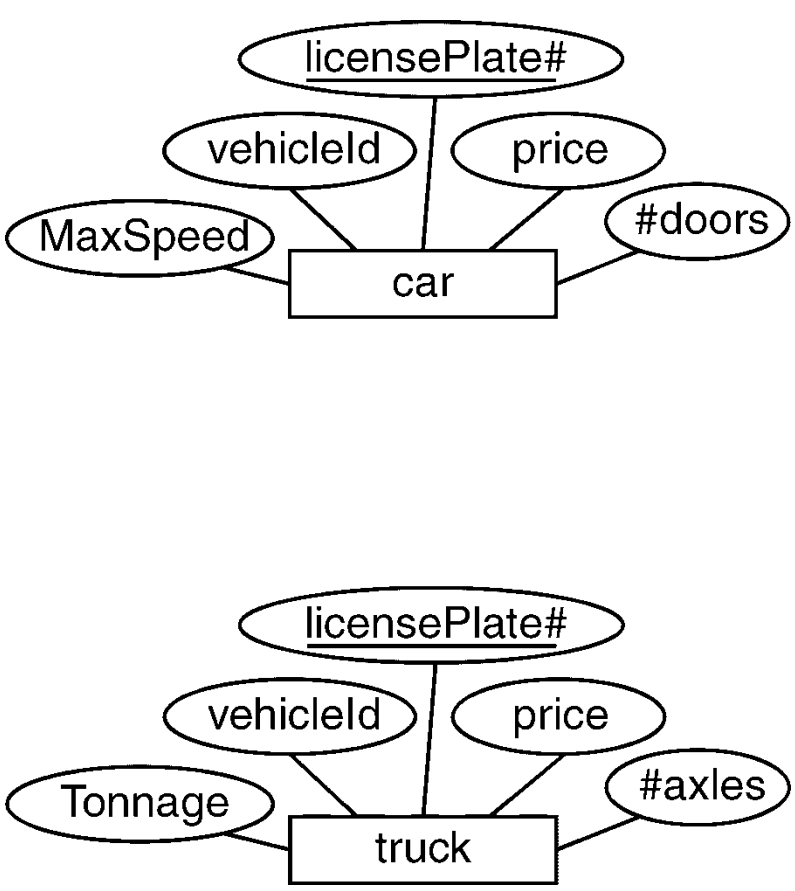
Specialization (cont'd)

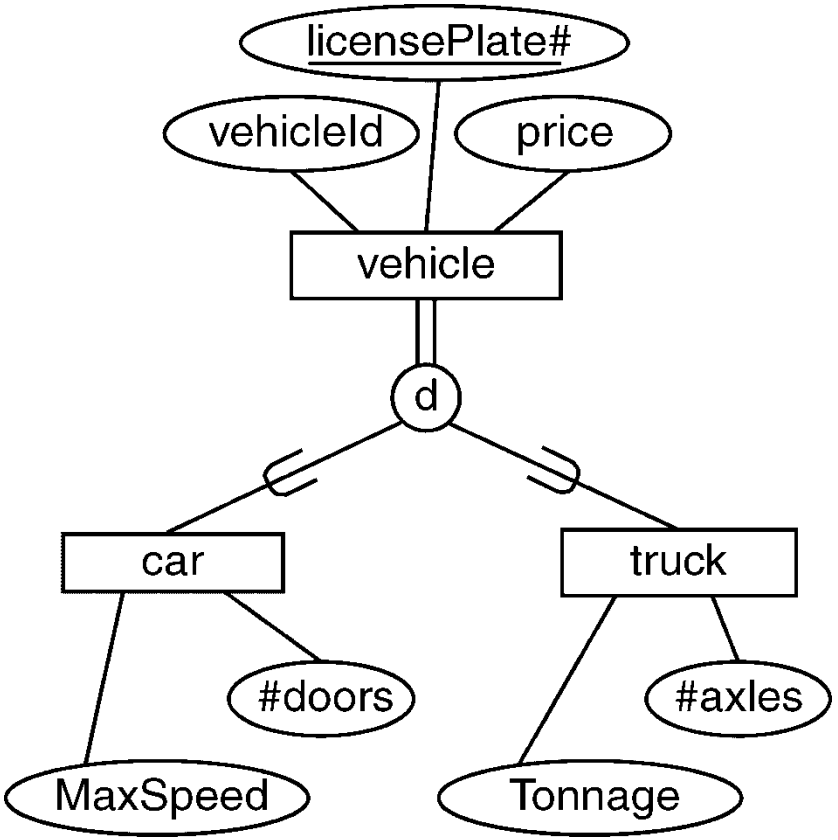


Generalization

- Generalization
  - Result of taking the union of two or more lower-level entity types to produce a higher-level entity type
    - The original entity types are special subclasses and the new higher-level entity type is the superclass
    - Functionally the inverse of the specialization process
  - We don’t use any special notation for generalization
  - The original entities that are used in generalization are special subclasses.
  - In other words in generalization every higher-level entity must also be a lower-level entity but specialization does not have this constraint.

Generalization





Deriving Relational Schema

- Fairly straightforward to derive relational schema from the ER diagrams

Strong Entity

* An entity type E with attributes A_1, A_2, …, A_K is represented as a k-degree relation

  $E(A_1, A_2, \ldots, A_K)$

  » Each tuple of the relation represents one entity in the entity type

  » Include only simple components of a composite attribute

Deriving Relational Schema (cont'd)

Relationship

- A relationship R among entity types E_1, E_2, …, E_K
  - Let P_1, P_2, …, P_K be the primary keys of the entity sets E_1, E_2, …, E_K respectively
- Relationship R has attributes A_1, A_2, …, A_R
- The relationship R is represented as a (k+r)-degree relation

$$R(P_1, P_2, \ldots, P_K, A_1, A_2, \ldots, A_R)$$

Deriving Relational Schema (cont'd)

Weak entity

- A weak entity type W has attributes A_1, A_2, …, A_W
- Depends on strong entity type S with primary key P_S
- The weak entity is represented as

$$W(P_S, A_1, A_2, \ldots, A_W)$$

Multi-valued attribute

- A multi-valued attribute A_M of entity type E (or relationship type R) with primary key A_K is represented by

$$M(A_K, A_M)$$

- A_K and A_M together form the primary key to M

Deriving Relational Schema (cont'd)

Example

* The project-employee ER diagram (Example 1) is converted to the following five relations:

EMPLOYEE (emp#, ename)

PROJECT (proj#, pname, start-date)

WORKS-ON (emp#, proj#, hours)

DEPENDENT (emp#, dep-name, birth-date)

DEPENDENT-OF (emp#, dep-name)

* Primary key shown underlined
* The last relation is redundant

Deriving Relational Schema (cont'd)

Example (cont'd)

- Problems in representing the weak entity type
  - Using dep-name as the key means if two dependents of the same employee have the same name we have duplicated keys.
  - Multiple occurrences of a dependent may be avoided by giving the dependent its own unique identifier
  - The modified schema is

EMPLOYEE (emp#, ename)

PROJECT (proj#, pname, start-date)

WORKS-ON (emp#, proj#, hours)

DEPENDENT (dep-id, dep-name, birth-date)

DEPENDENT-OF (emp#, dep-id)

Deriving Relational Schema (cont'd)

For 1:1 and 1:M Relations

* We can avoid a separate relation by adding attributes to the associated entity
  - » Reduces redundancy
* Example revisited
  - » The revised schema is

EMPLOYEE (emp#, proj#, ename, hours)

PROJECT (proj#, pname, start-date)

~~**WORKS-ON (emp#, proj#, hours)**~~

DEPENDENT (dep-id, dep-name, birth-date)

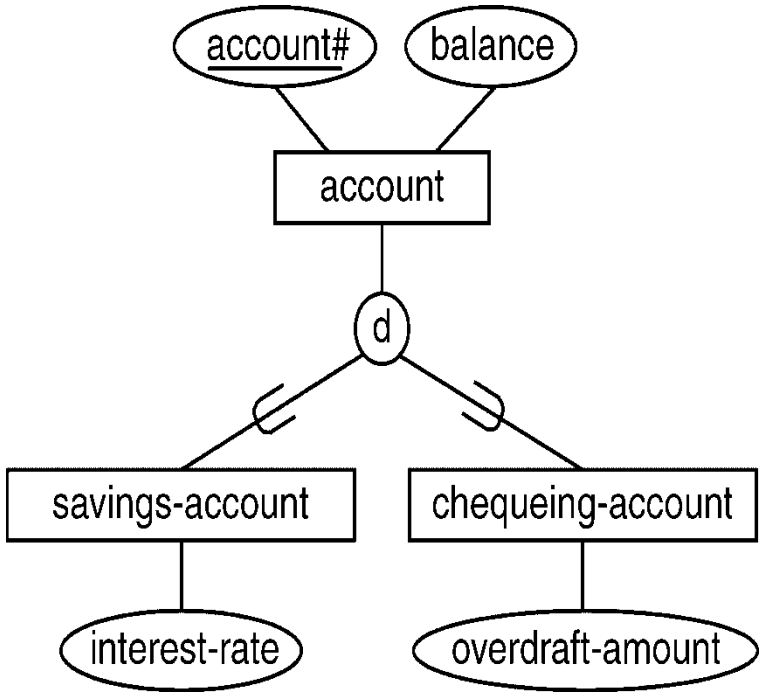
DEPENDENT-OF (emp#, dep-id)

Deriving Relational Schema (cont'd)

- Two methods for deriving relational schema from an ER diagram with specialization/generalization
  - Method 1
    - Create a table for the higher-level entity
    - For each lower-level entity, create a table which includes a column for each of its attributes plus for primary key of the higher-level entity
  - Method 2
    - Do not create a table for the higher-level entity
    - For each lower-level entity, create a table which includes a column for each of its attributes plus a column for each attribute of the higher-level entity

Deriving Relational Schema (cont'd)

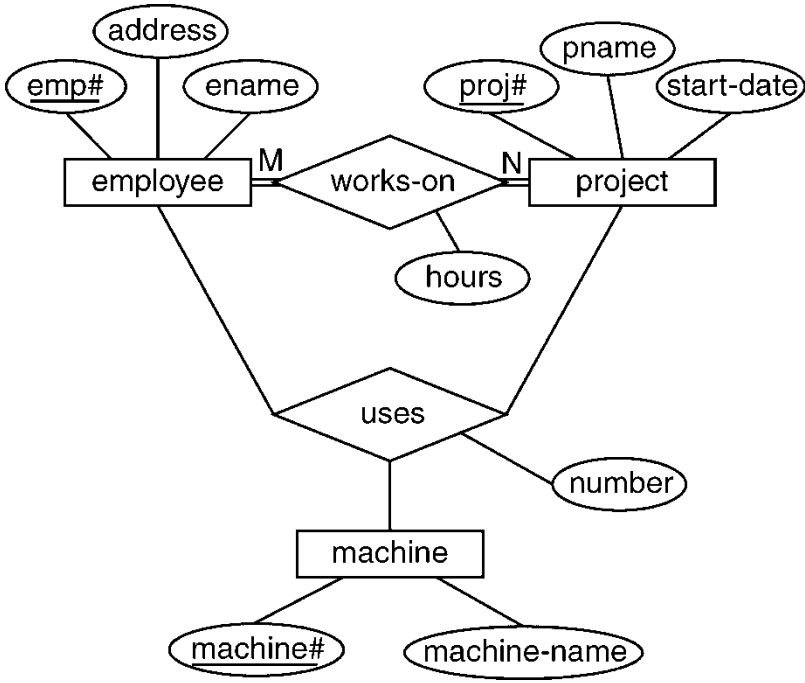
- Method 1
  - **account (<u>account#</u>, balance)**
  - **savings-account (<u>account#</u>,**
  - **interest-rate)**
  - **chequeing-account (<u>account#</u>,**
  - **overdraft-amount)**



- Method 2
  - **savings-account (<u>account#</u>,**
  - **balance, interest-rate)**
  - **chequeing-account (<u>account#</u>,**
  - **balance, overdraft-amount)**

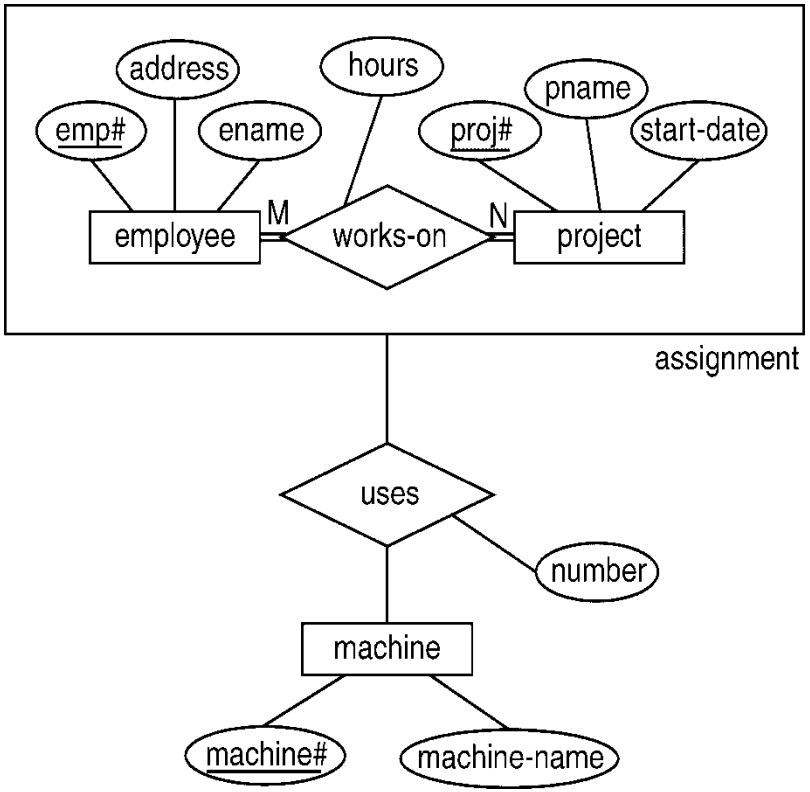
Aggregation

- Motivation
  - A limitation of the ER model
    - Not possible to express relationship among relationships
  - We may have to use two or more relationships
    - **works-on** and **uses** relationships are independent
    - But it is complicated because we just wanted to show that when employee works on a project, he/she uses a machine…



Aggregation (cont'd)

- Aggregation is an abstraction through which relationships are treated as higher-level entities
- Example
  - We create a new higher-level entity called **assignment**
  - Now we can establish relationships by treating this new entity as a regular entity



Aggregation (cont'd)

- Deriving relational schema
  - Transform the higher-level entity
    - Use the procedure described before
  - Transform the aggregate relationship
    - Entity types participating in the higher-level entity H: E_1, E_2, …, E_{K-1}
    - Let P_1, P_2, …, P_K be the primary keys of E_1, E_2, …, E_K respectively
    - Attributes of relationship R between entity types H and E_K: A_1, A_2, …, A_R
    - The relationship is represented by

$$R(P_1, P_2, \ldots, P_{K-1}, P_K , A_1, A_2, \ldots, A_R)$$

Aggregation (cont'd)

Example

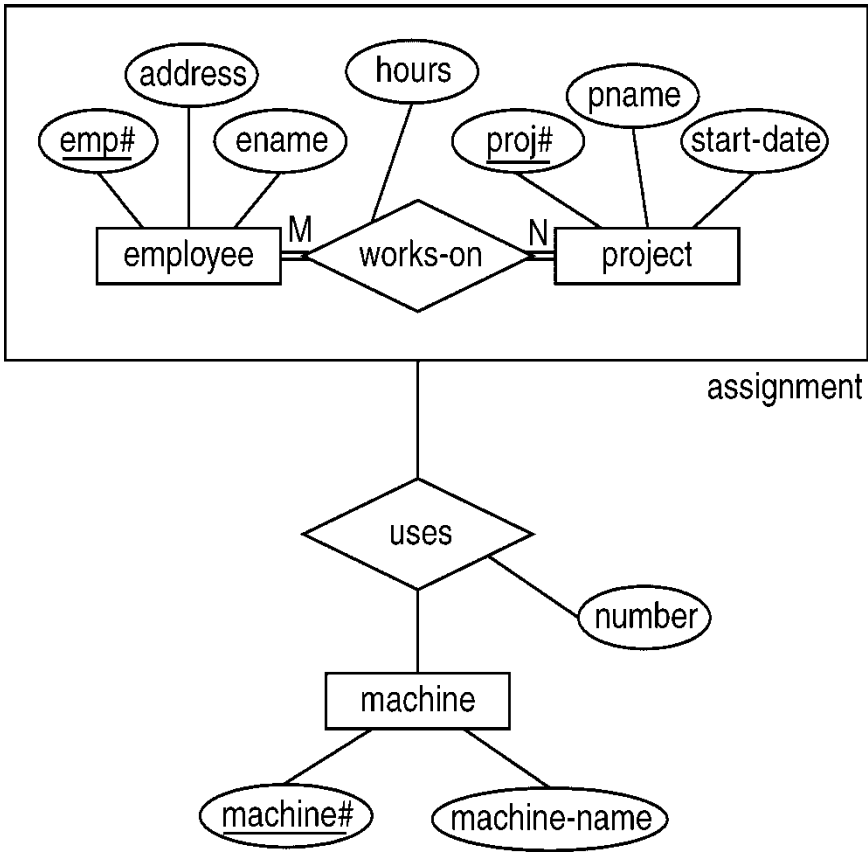
EMPLOYEE(emp#, ename, adddress)

PROJECT(proj#, pname, start-date)

WORKS-ON(emp#,proj#, hours)

MACHINE(machine#, machine-name)

USES(emp#,proj#, machine#, number)



E/R Diagram and Data Dictionary

- As mentioned before, data dictionary is the database designer's database
- The results of E/R diagram can be used to identify the kinds of objects the dictionary needs to support
- For example a weak or strong entity, total or partial participation in a relationship and a supertype or subtype entity and etc., all can be explained in a data dictionary.

Project: University Database

Consider the following requirements for a university database

- The university keeps track of each student's name, address, student number, social insurance number, and the courses they have registered.
- In addition, for undergraduate and graduate students the degree program (BA, BCS, MSc, PhD) they are in is also maintained. (For other students such as special students, exchange students etc. this information is not needed.)

University Database

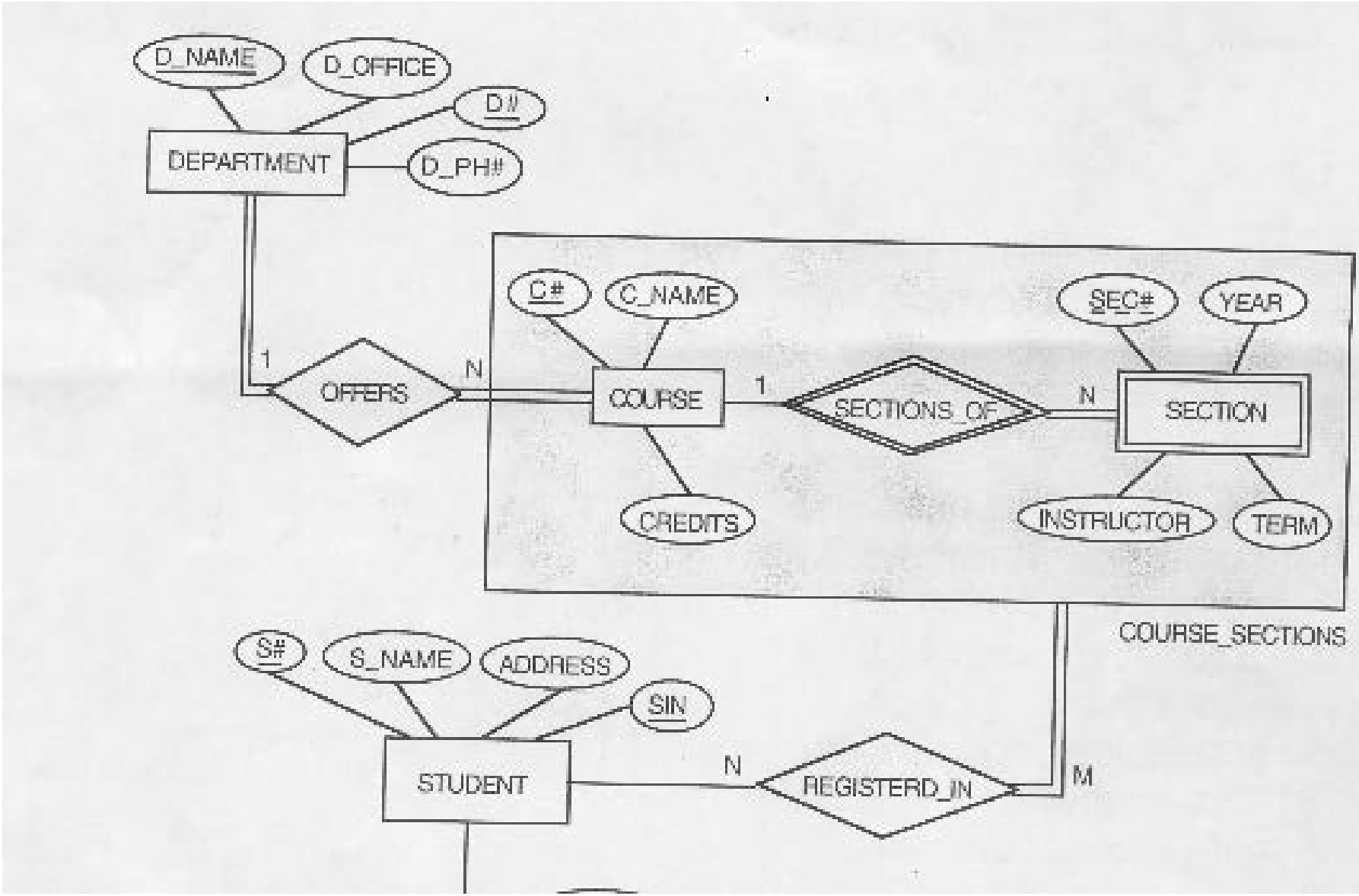
- For graduate students, a list of degrees held (degree, university, and the year degree was awarded) and their office in the department and phone number are included in the database.
- All graduate students are financially supported either by a teaching assistantship (TA) or by a research assistantship (RA). For the TAs we would like to keep the number of hours per week they are working and for the RAs the research project they are associated with (just research project name).

University Database

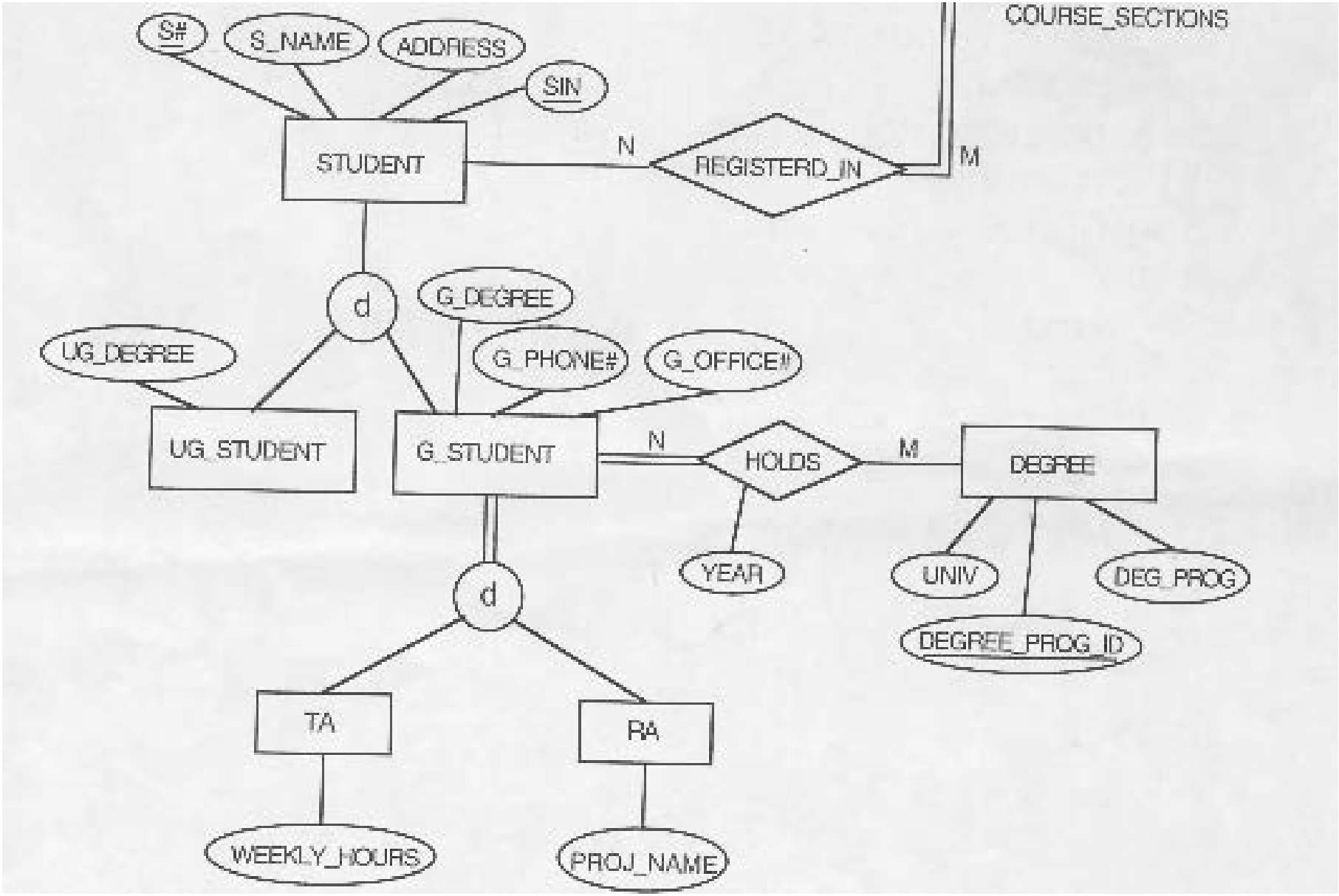
- Each department is represented. The data about departments are its name, department code, office number, and office phone. Both name and code have unique values for each department.
- Each course has a course number, course name, number of credits and the offering department. The value of course number is unique for each course.

University Database

- Each section has an instructor, term, year, course, and section number. The section number distinguishes different sections of the same course that are taught during the same year; its values are 1, 2, 3,..., up to the number of sections taught during each year.
- The ER diagram is shown in the next two slides
- Specify a preliminary relational database schema for ER diagram



CONTINUES IN THE NEXT SLIDE


COURSE_SECTIONS
S#
S_NAME
ADDRESS
SIN
STUDENT
N
REGISTERD_IN
M
d
G_DEGREE
UG_DEGREE
G_PHONE#
G_OFFICE#
UG_STUDENT
G_STUDENT
N
HOLDS
M
DEGREE
YEAR
UNIV
DEG_PROG
DEGREE_PROG_ID
d
TA
RA
WEEKLY_HOURS
PROJ_NAME

Bank Database