CS240: Database Systems

Lecture 08:

Review for Final Exam

Final Exam

- Closed-book Exam
- Choice Question
- Questions:
  - Write down the answers

Content

- Lecture 1 – Introduction
- Lecture 2 – Entity-Relation diagram
- Lecture 3 – Relational Model
- Lecture 4 – Relational Algebra
- Lecture 5 – SQL
- Lecture 6 – Functional Dependencies and Normalization
- Lecture 7 – Transaction

Lecture 1 – Introduction

- What is a Database?
- What is a DBMS?
- Data models, data abstraction
- People who Deal With Databases

What is a Database?

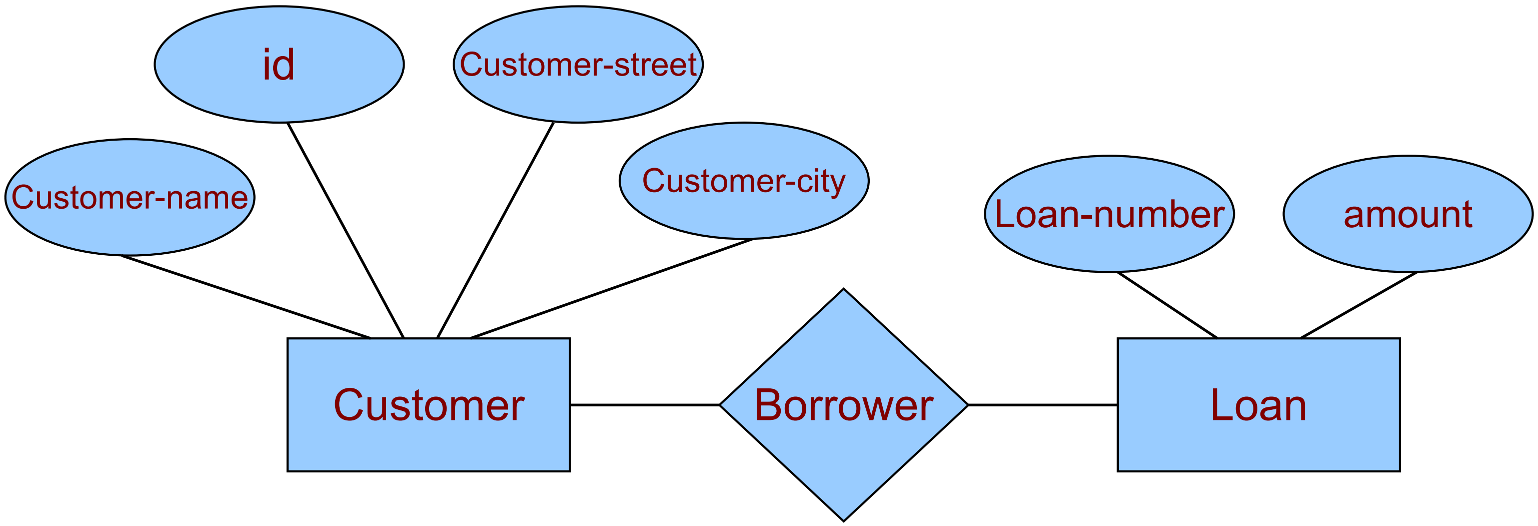
- A collection of data, typically describing the activities of one or more related organizations.
- Models real-world enterprise
  - Entities
    - e.g. students, professors, courses, classrooms
  - Relationships between entities
    - e.g. student's enrollment in courses, professor teaching courses, and use of room for courses.

What is a DBMS?

- **DBMS – Database Management System**
- A DBMS is a collection of ***software programs*** to enable users to ***create***, ***maintain*** and ***utilize*** a ***database***.

Data Models

- A data model is a collection of tools or concepts for describing data, the meaning of data, data relationships and data constraints.
  - **Entity-Relationship Model (ER Model)**



Data Models

- **Relational Model**

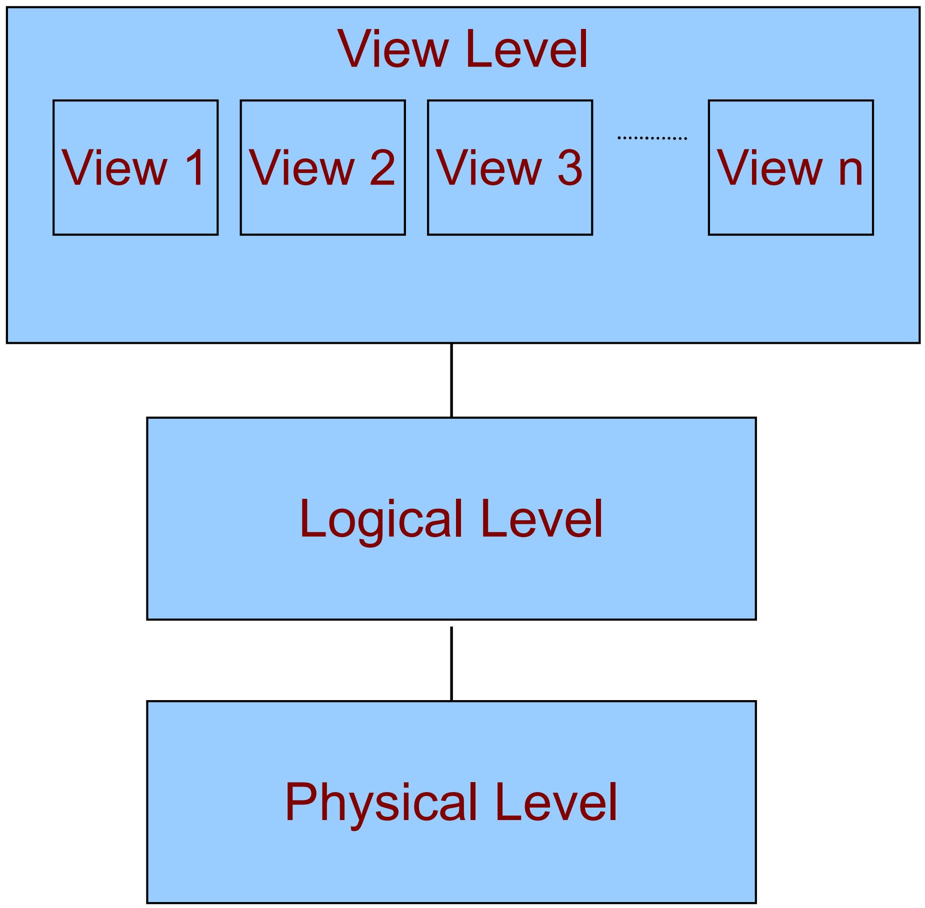
Customer-Name	ID	customer-street	customer-city

  - Main concept: **relation**, basically a table with rows and columns. A column is also called a **field** or **attribute**
  - Other models such as the Network Model, Hierarchical Model, Objected-relational Model

We will focus on the dominant ***Relational model***.

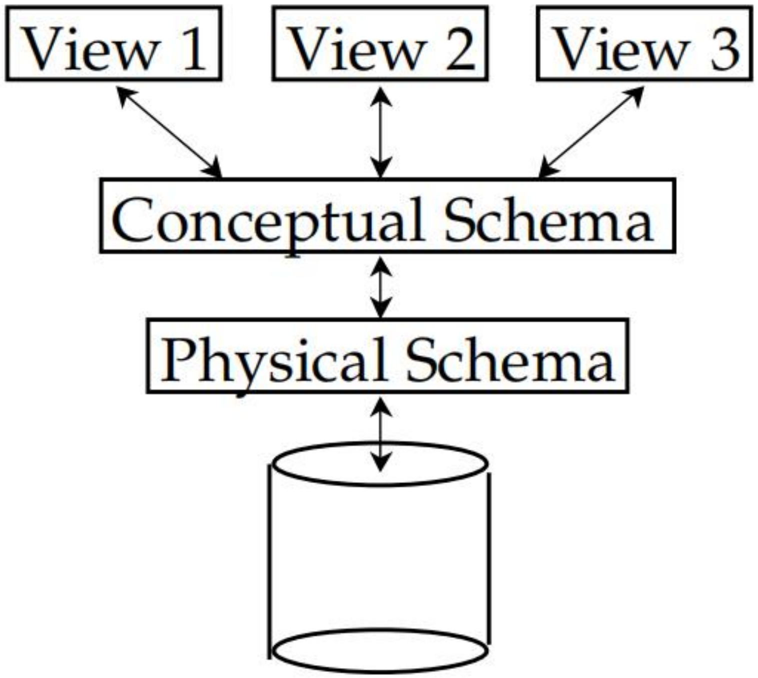
A description of data in terms of a data model is called a **schema**.

Data Abstraction



Levels of Abstraction

- Many views, single conceptual (logical) schema and physical schema.
  - Views (External schema) describe how users see the data.
  - Conceptual schema defines logical structure
  - Physical schema describes the files and indexes used.



People who Deal With Databases

- **Database Administrator (DBA):** Person(s) who has central control over the database and is responsible for the following tasks:
  - Schema definition/modification
  - Storage structure definition/modification
  - Authorization of data access
  - Integrity constraint specification
  - Monitoring performance
  - Responding to changes in requirements

People who Deal With Databases

- **Application Programmers**
  - Embed DML calls in program written in a host language (e.g., Cobol, C, Java). (DML stands for data manipulation language)
  - e.g., programs that generates payroll checks, transfer funds between accounts
- **Sophisticated Users**
  - Form request in database query language
- **Naive users**
  - Invokes one of the permanent application programs that have been written previously
  - e.g. transfer – transfer fund between accounts

Lecture 2 – Entity-Relation diagram

- Database Design

1. Requirement analysis
2. Conceptual database design – ER
3. Logical database design – relational schema
4. Schema refinement
5. Physical database design
6. Application and security design

Lecture 2 – Entity-Relation diagram

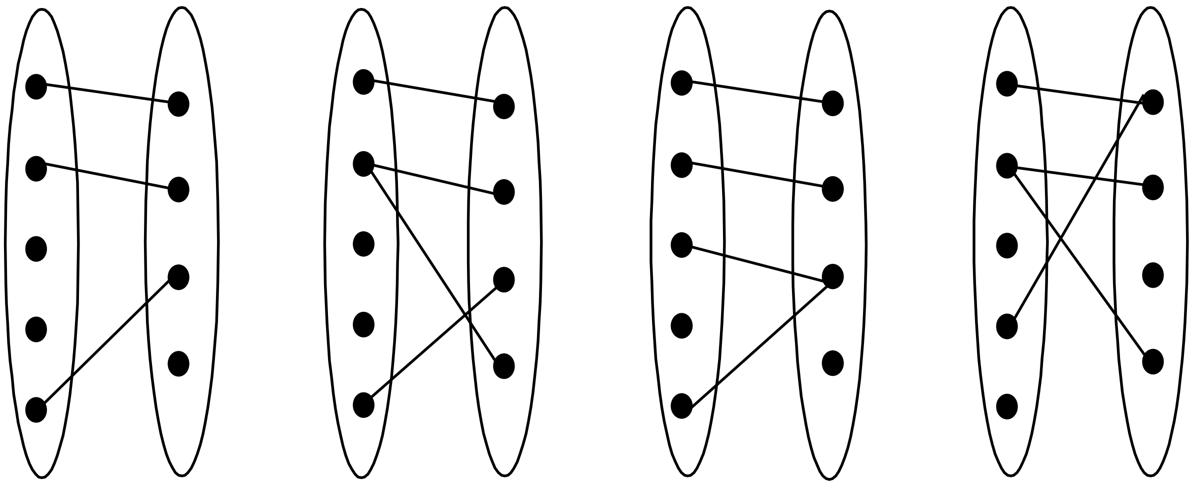
- **Entity-Relationship (ER) model** is a popular conceptual data model.
- This model is used in the design of database applications.
- E-R model views the real world as a collection of **entities** and **relationships** among entities.
- Elements of ER model
  - Entities, Entity Sets, Attributes
  - Relationships **(!= relations!)**

Summary: Key

- ***Super Key***: A key that can be uniquely used to identify an entity, that may contain extra attributes that are not necessary to uniquely identify entities.
- ***Candidate Key***: A candidate key can be uniquely used to identify an entity without any extraneous data. It is a minimal super-key. Often abbreviate the ***Candidate Key*** to just ***Key***.
- ***Primary Key***: It is one of the candidate keys.

Constraints

- The model describes data to be stored and the **constraints** over the data
- The type of association of a binary relationship can be classified into the following cases:



1-to-1
1:1

1-to-Many
1:N

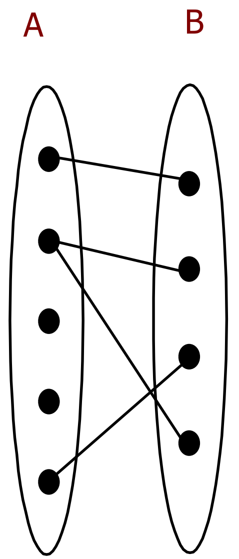
Many-to-1
N:1

Many-to-Many
N:M

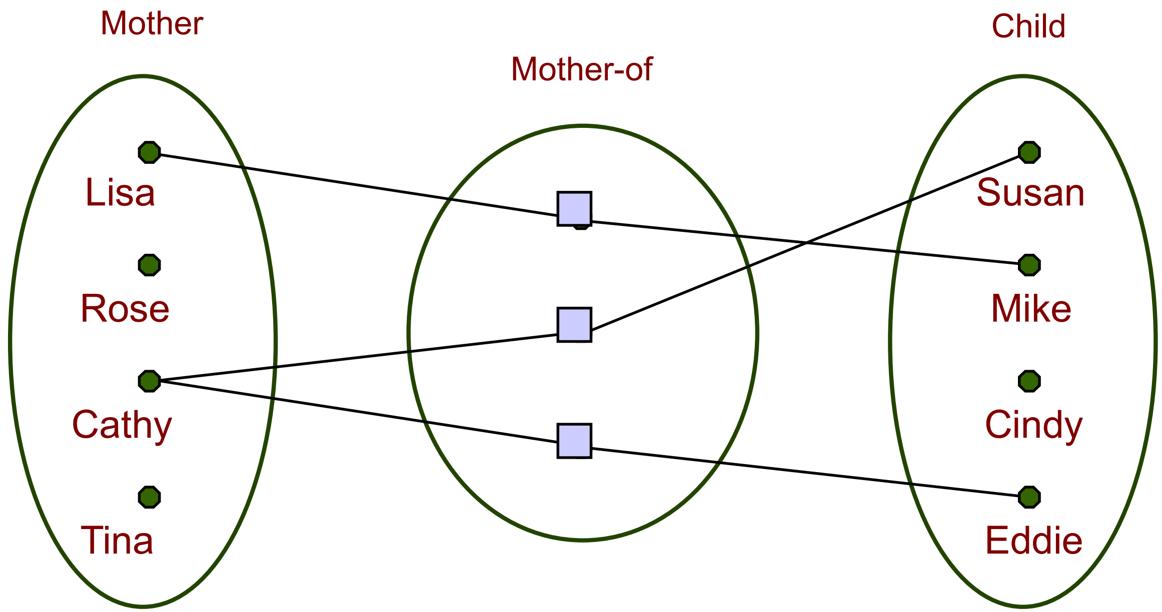
The preceding discussion identified each relationship in both directions; that is, relationship are ***bidirectional***.

One-to-many relationship

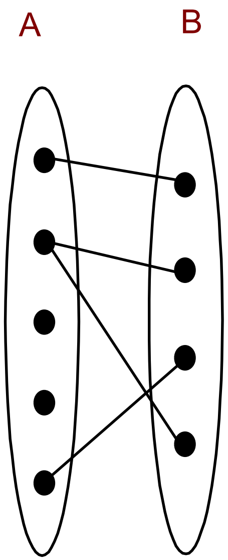
- An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.
- Example: each child can appear in at most one mother-child relationship.



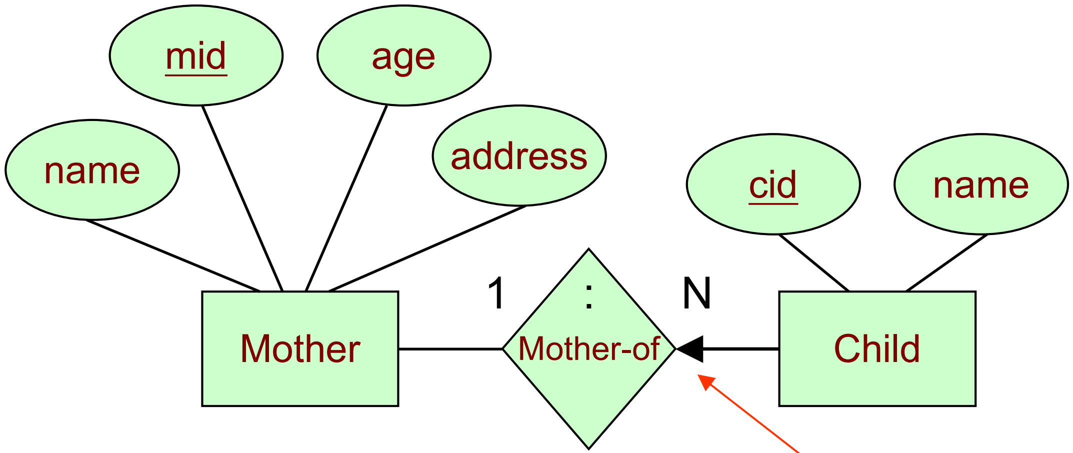
1-to-Many
1:N



- Child has a ***key constraint*** in the mother-of relationship set.
- This restriction can be indicated by an arrow in the E-R diagram.
  - **A** Child is associated with at most **one** Mother via Mother-of
  - **A** Mother is associated with **several** (including 0) Children via Mother-of



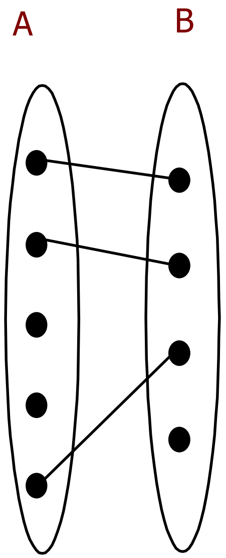
1-to-Many
1:N



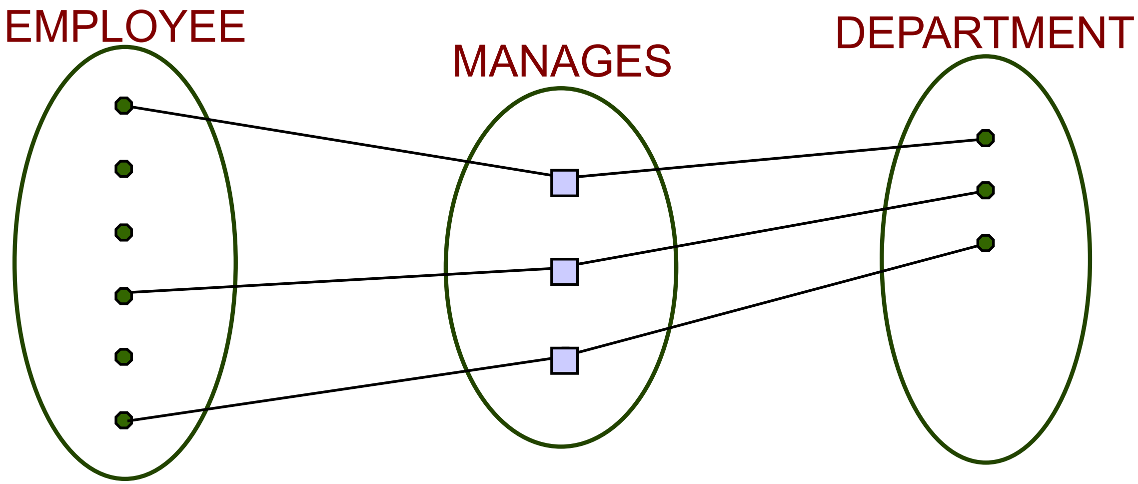
Intuitively, the arrow states that given a child entity, we can uniquely determine the mother-of relationship.

One-to-one relationship

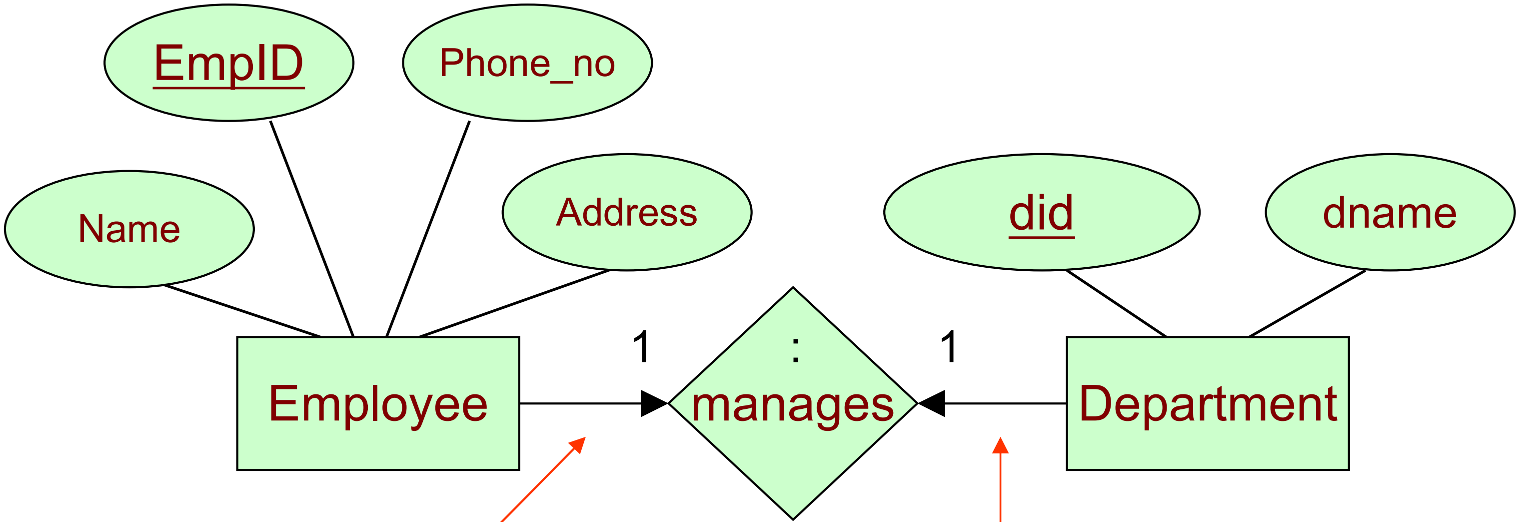
- If the relationship between A and B satisfies the one-to-one mapping constraint from A to B, then
- **An** entity in A is related to **at most one** entity in B, and
- **An** entity in B is related to **at most one** entity in A.

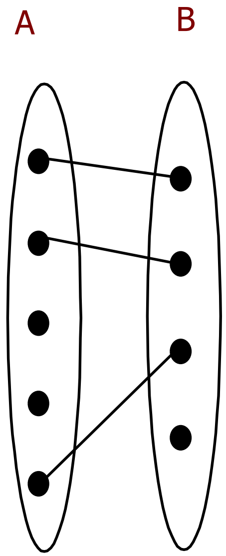


1-to-1
1:1



- A Employee is associated with at most one Department via the relationship manages
- A Department is associated with at most one Employee via manages





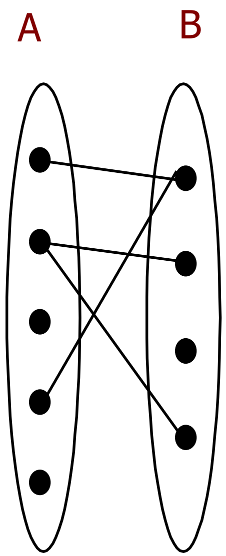
An employee can associate with at most one department via the Relationship “manages”.

A department can associate with at most one employee via the relationship “manages”

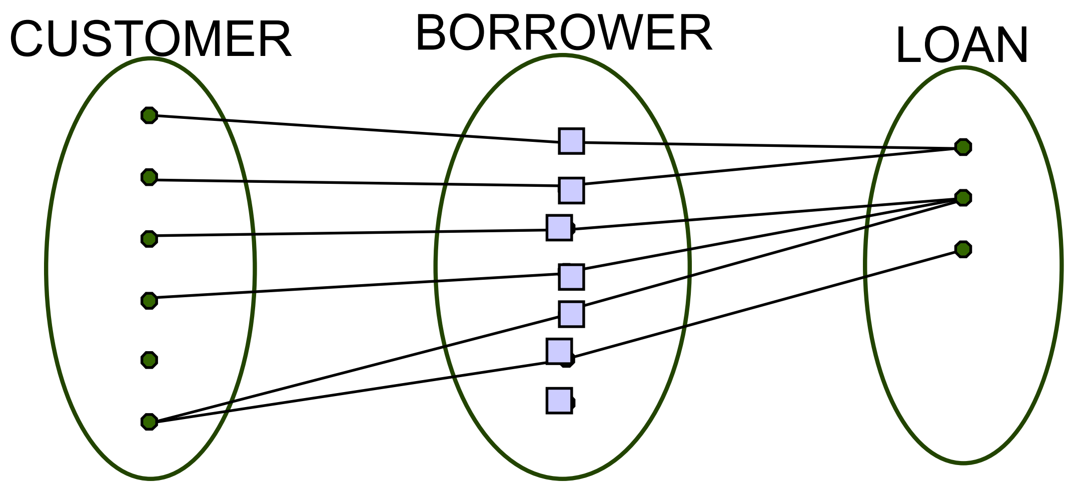
1-to-1
1:1

Many-to-many Relationship

- **An** entity in A is associated with **any number** of entities in B
- **An** entity in B is associated with **any number** of entities in A
- In fact, there is no restriction in the mapping.

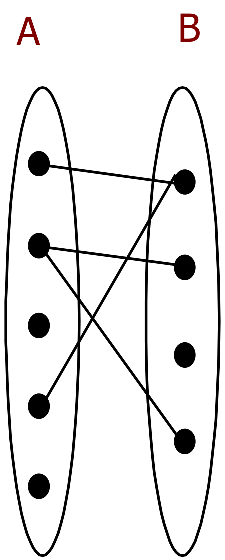


Many-to-Many
N:M

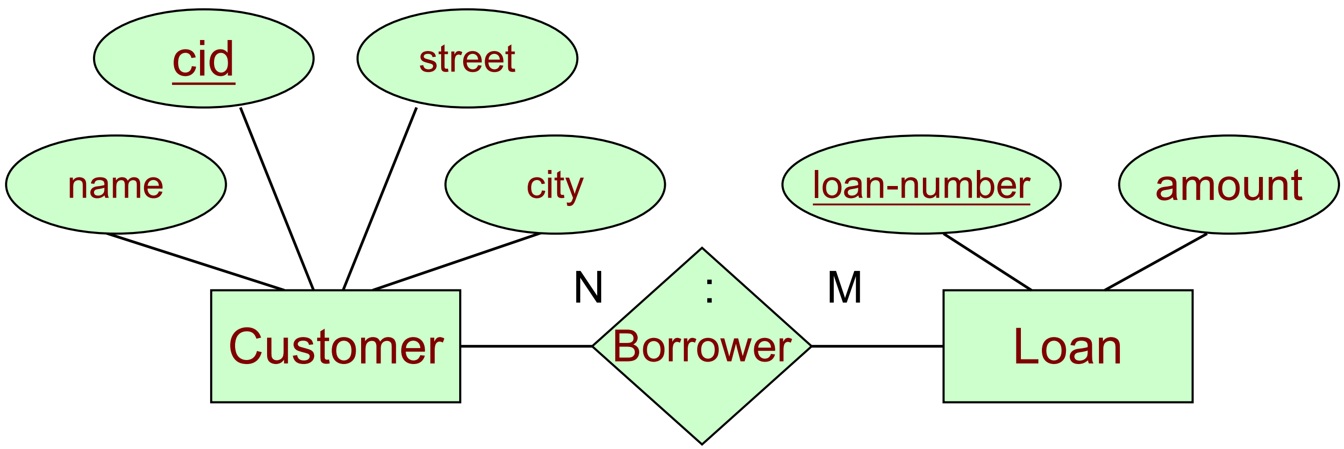


Many-to-many Relationship

- A customer can associate with several loans (possibly 0) via Borrower
- A loan can associate with several customers (possibly 0) via Borrower

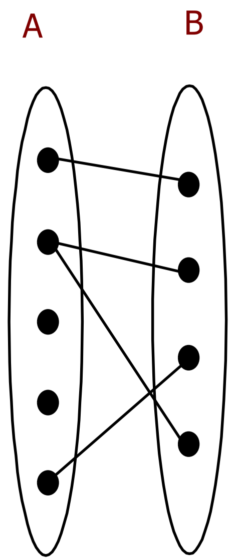


Many-to-Many
N:M



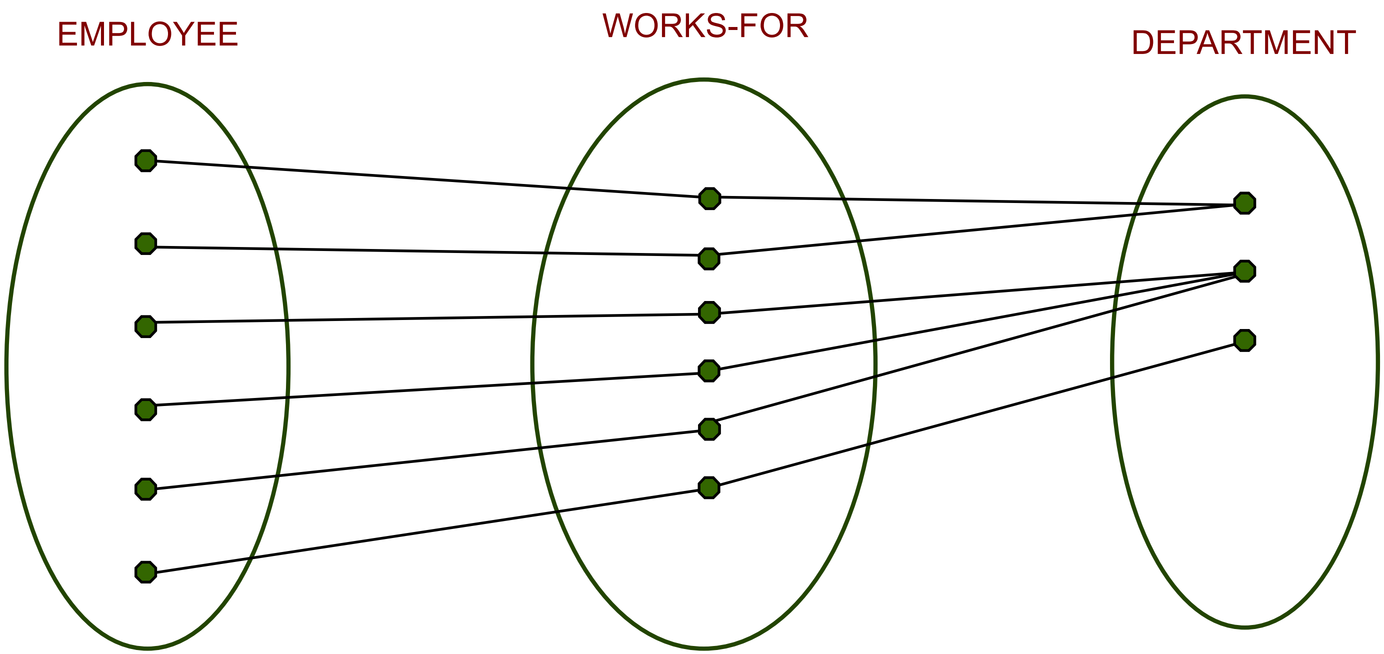
Participation Constraint

- We can classify participation in relationships as follows.
- **Total**
  - Each entity in the entity set must be associated in at least one relationship
- **Partial**
  - Each entity in the entity set may (or may not) be associated in a relationship

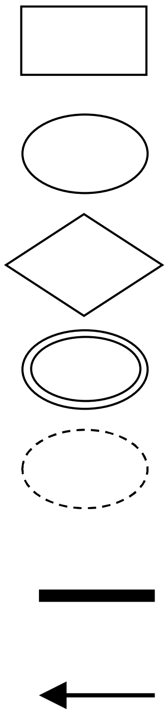


Participation Constraint

- For example
  - ***Every employee*** must work for some department.
  - The participation of EMPLOYEE in WORKS-FOR is ***total participation***



Lecture 2 – Entity-Relation diagram



- Rectangle – entity set
- Ellipse – attribute
- Diamond – relationship set
- Double ellipse – multi-valued attribute
- Dashed ellipse – derived attribute
- Thick line – total participation
- Arrow – key constraint

Steps when you draw an ER diagram

1. Look at the question description carefully
2. Identify the entities (including all the attributes)
   - How to represent a key in an ER diagram?
3. Identify the relationships between entities
4. The participation constraint and the key constraint
   - Total participation or partial participation?
   - One to one, one to many or many to many?

Exercise 1

- A university registrar's office maintains data about the following entities:
  a) **course**, including number, title, credits, syllabus, and prerequisites;
  b) **course offering**, including course number, year, semester, section number, instructor(s), timings, and classroom; (course offering depends on the course that university provided)
  c) **student**, including student-id, name, and program;
  d) **instructor**, including identification number, name, department, and title.
  - Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled.
- Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the cardinality constraints and participation constraints.

Lecture 3 – Relational Model

- Concepts:
  - Relation
  - Attribute
  - Tuple
  - Attribute value
  - Domain

Relation Name/Table Name

Attributes/Columns (collectively as a schema)

STUDENT			
Name	**Student-id**	**Age**	**GPA**
Chan Kin Ho	99223367	23	11.19
Lam Wai Kin	96882145	17	10.89
Man Ko Yee	96452165	22	8.75
Lee Chin Cheung	96154292	16	10.98
Alvin Lam	96520934	15	9.65

Tuples/Rows

Schema

- The relation schema is

  Student(Name, Student-id, Age, GPA)

  OR

Name	Student-id	Age	GPA

- The primary key is underlined in the above

Schema Definition in SQL

- The relation schema is

  Customer(<u>customer-name</u>, customer-street, customer-city)

  or

Customer	<u>customer-name</u>	customer-street	customer-city

- The primary key is underlined in the above

```
CREATE TABLE Customer
(
    customer-name     CHAR(20)   NOT NULL,
    customer-street   CHAR(30),
    customer-city     CHAR(30),
    PRIMARY KEY (customer-name)
)
```

Schema Definition in SQL

- To remove a relation from an SQL database, we use the **drop table** command:

  drop table r

- We use the alter table command to add or delete attributes to an existing relation
- All records in the relation are assigned null for a new attribute.

  alter table customer **add** phone char(10)

  alter table customer **drop** phone

Summary

- The mapping relations of relation data model

Informal Terms	Formal Terms
Table	Relation
Column	Attribute
All possible Column Values	Domain
Row	Tuple
Table Cell Value	Attribute Value

Relational Integrity Constraints (IC)

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are some *main types* of constraints in the relational model:
  - **Domain** constraint
    - Every value in a tuple must be from the *domain of its attribute* (or it could be **NULL**, if allowed for that attribute)
  - **Key** constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints
    - A **foreign key** is a set of attributes in one relation *R1* that is used to refer to a tuple in another relation *R2*
- ICs are specified when the schema is defined
- ICs are checked when relations are modified

Populated Database State

- Basic operations for changing the database:
  - INSERT a new tuple in a relation
  - DELETE an existing tuple from a relation
  - MODIFY an attribute of an existing tuple

Possible Violations for INSERT

- Domain constraint:
  - if one of the attribute values provided for the new tuple is not of the specified attribute domain
- Key constraint:
  - if the value of a key attribute in the new tuple already exists in another tuple in the relation
- Referential integrity:
  - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
- Entity integrity:
  - if the primary key value is null in the new tuple

Possible Violations for DELETE

- Referential constraints
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
- Options of remedies:
  - **RESTRICT**: reject the deletion
  - **CASCADE**: delete the record in the referencing table
  - **SET NULL**: set the foreign keys of the referencing tuples to NULL
  - **SET DEFAULT**: set the foreign keys of the referencing tuples to default value

Possible Violations for UPDATE

- Constraint violations depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
  - Updating a foreign key (FK):
    - May violate referential integrity
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain and business rules constraints

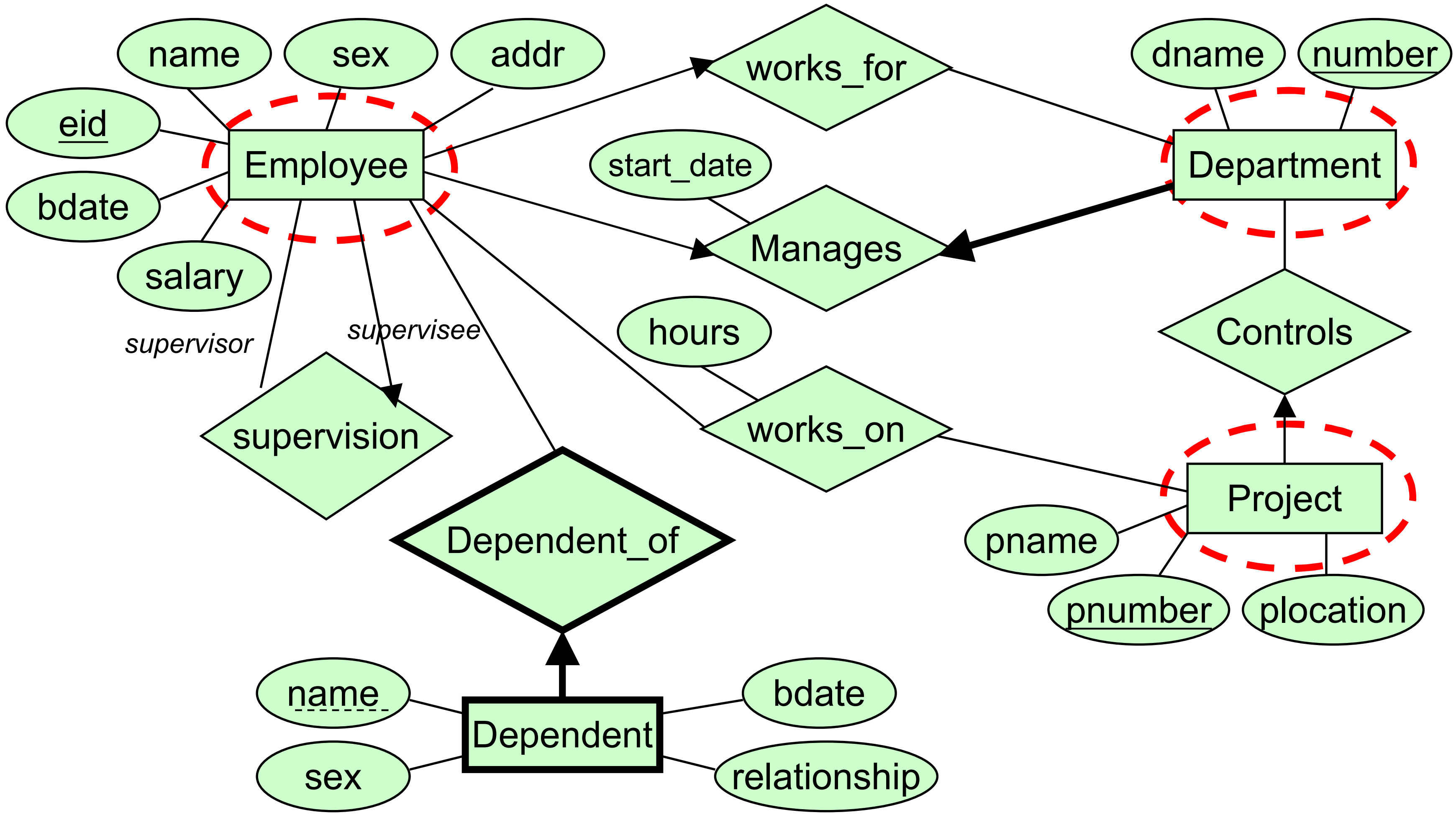
ER to Relational Model

1. Convert entities first
   - From Strong Entity to Weak Entity
2. Convert relationships
   - 1-to-1,
   - 1-to-many (many-to-1)
   - many-to-many
   - From binary relation to non-binary relation

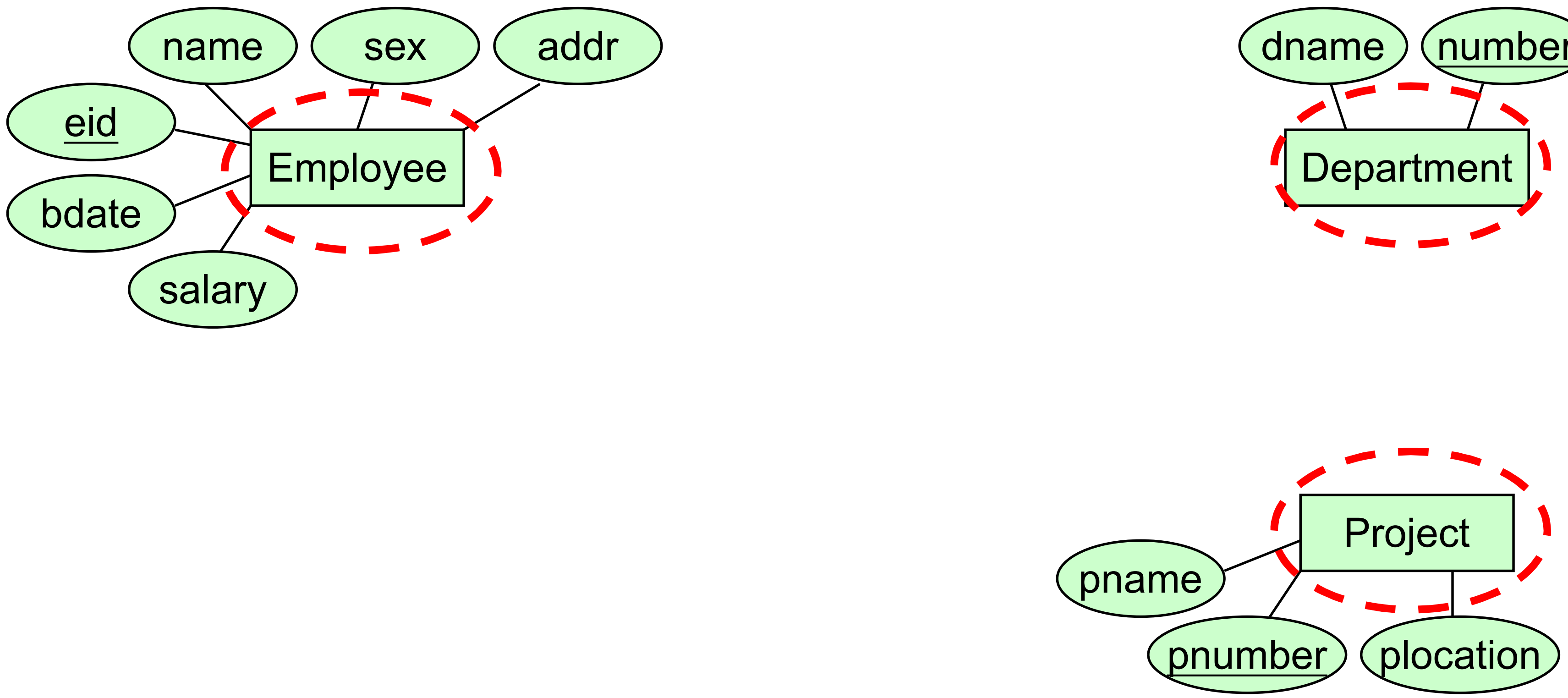
Steps

- **Step 1** (Strong Entity Set)
- **Step 2** (Weak Entity Set)
- **Step 3** (1-to-1 Relationship)
- **Step 4** (1-to-many Relationship)
- **Step 5** (Many-to-many Relationship)
- **Step 6** (Non-binary Relationship)

Step 1



name
sex
addr
eid
Employee
bdate
salary
supervisor
supervisee
supervision
works_for
start_date
Manages
hours
works_on
dname
number
Department
Controls
Project
pname
pnumber
plocation
Dependent_of
name
sex
Dependent
bdate
relationship


name
sex
addr
eid
Employee
bdate
salary
dname
number
Department
Project
pname
pnumber
plocation

Step 1 (Strong Entity)

- Example
- We create the relation schemas EMPLOYEE, DEPARTMENT and PROJECT.

EMPLOYEE

name	eid	bdate	addr	sex	salary

DEPARTMENT

dname	dnumber

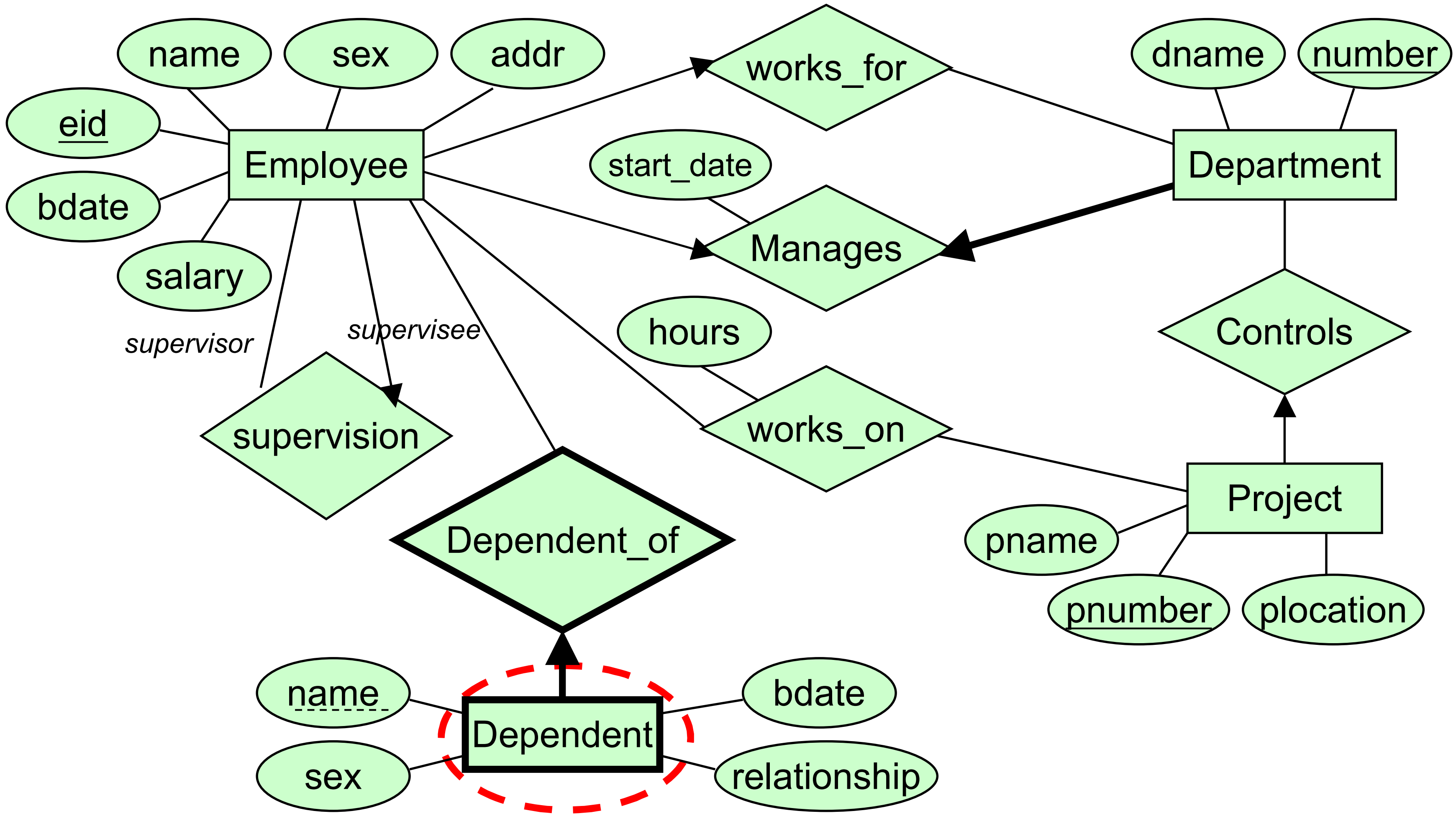
PROJECT

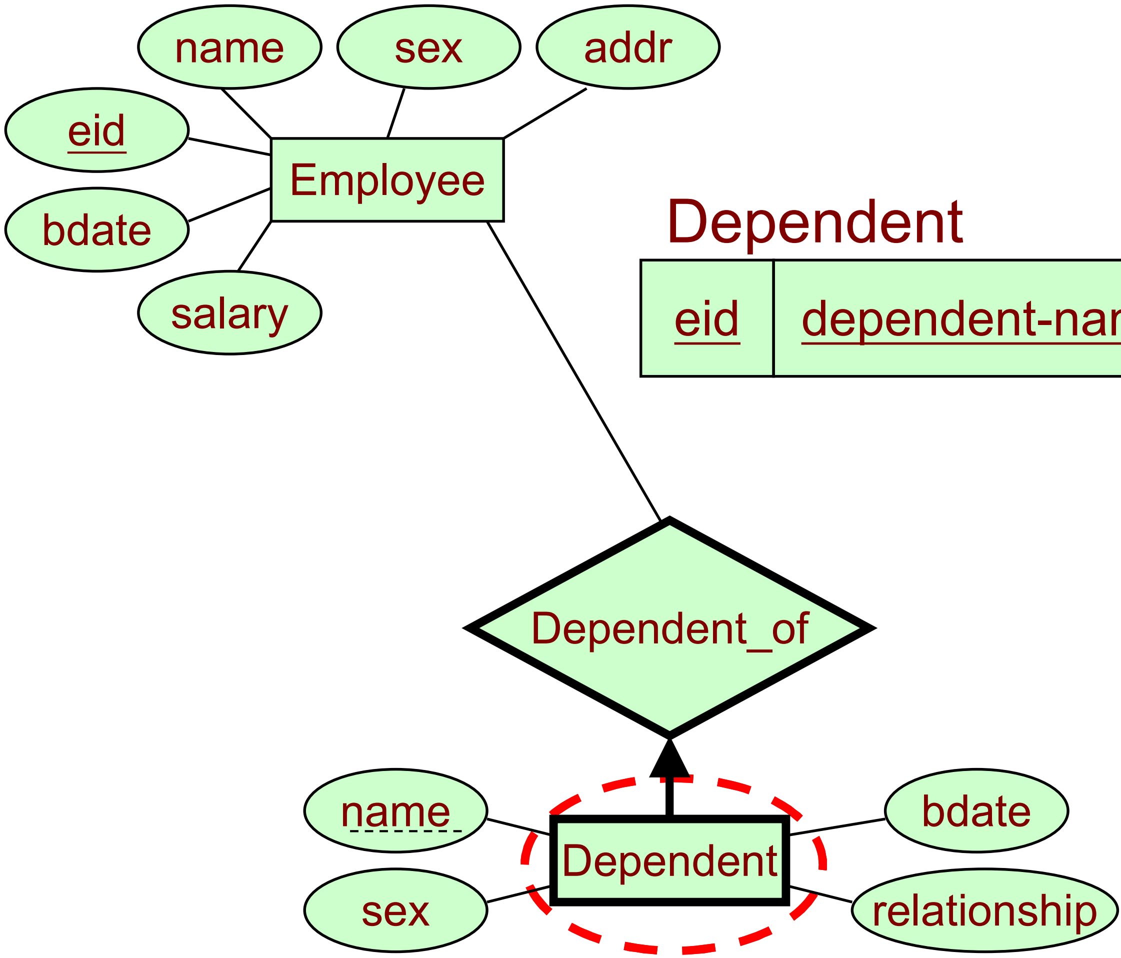
pname	pnumber	plocation

Step 2 (Weak Entity)

- For each **weak entity set** W in the ER model,
  - Create a relation schema R, and include all attributes
  - In addition, include the primary key(s) of the owner(s)
  - The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity set W

Step 2 (Weak Entity)



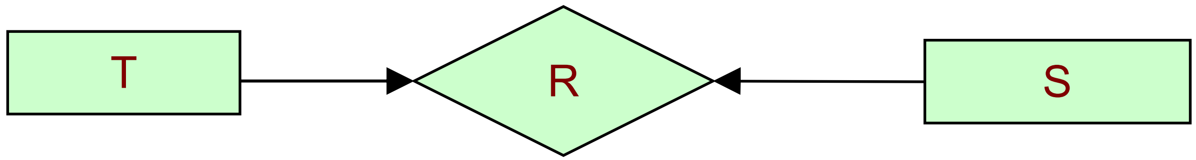


Dependent

eid	dependent-name	sex	bdate	relationship

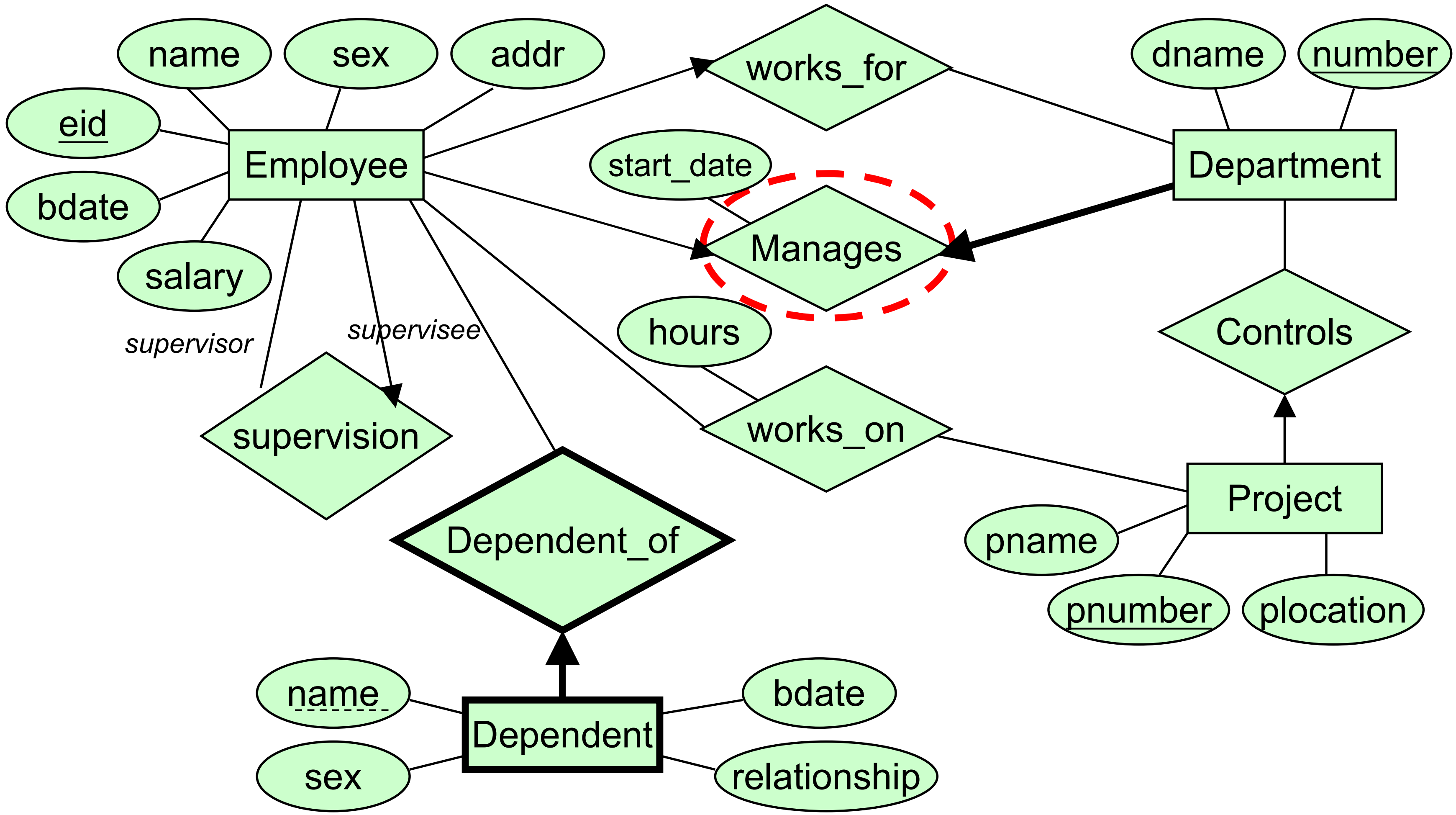
Step 3 (1-to-1 Relationship)

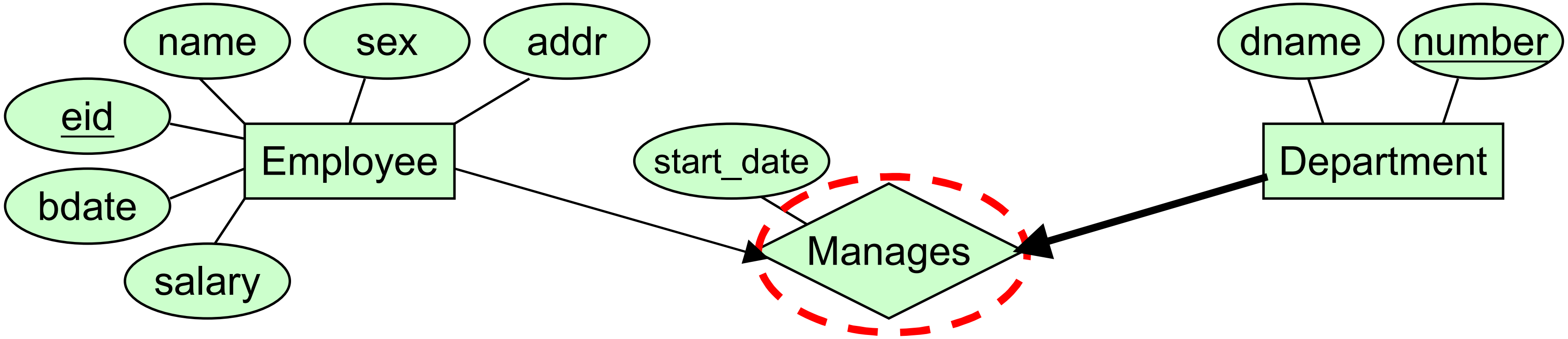
- For each binary one-to-one (1:1) relationship set R



  - Choose one of the 2 relation schemas, say S,
    - get primary key of T,
    - include it as foreign keys in S
    - Include the attributes of the relationship set R as attributes of S
  - Better if S has total participation in R

Step 3 (1-to-1 Relationship)





Choose the one with total participation

We include the primary key of EMPLOYEE as foreign key in DEPARTMENT and rename it mgr_id.

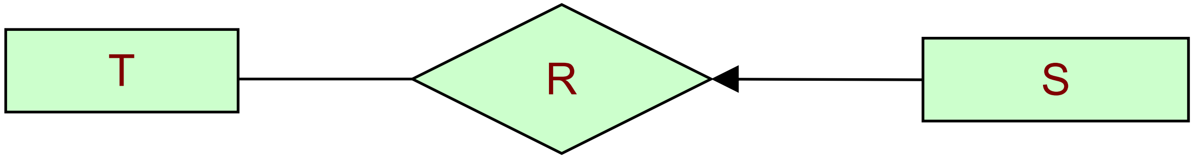
We include the attribute startdate of MANAGES and rename it mgr_start_date.

DEPARTMENT

dname	dnumber	mgr_id	mgr_start_date

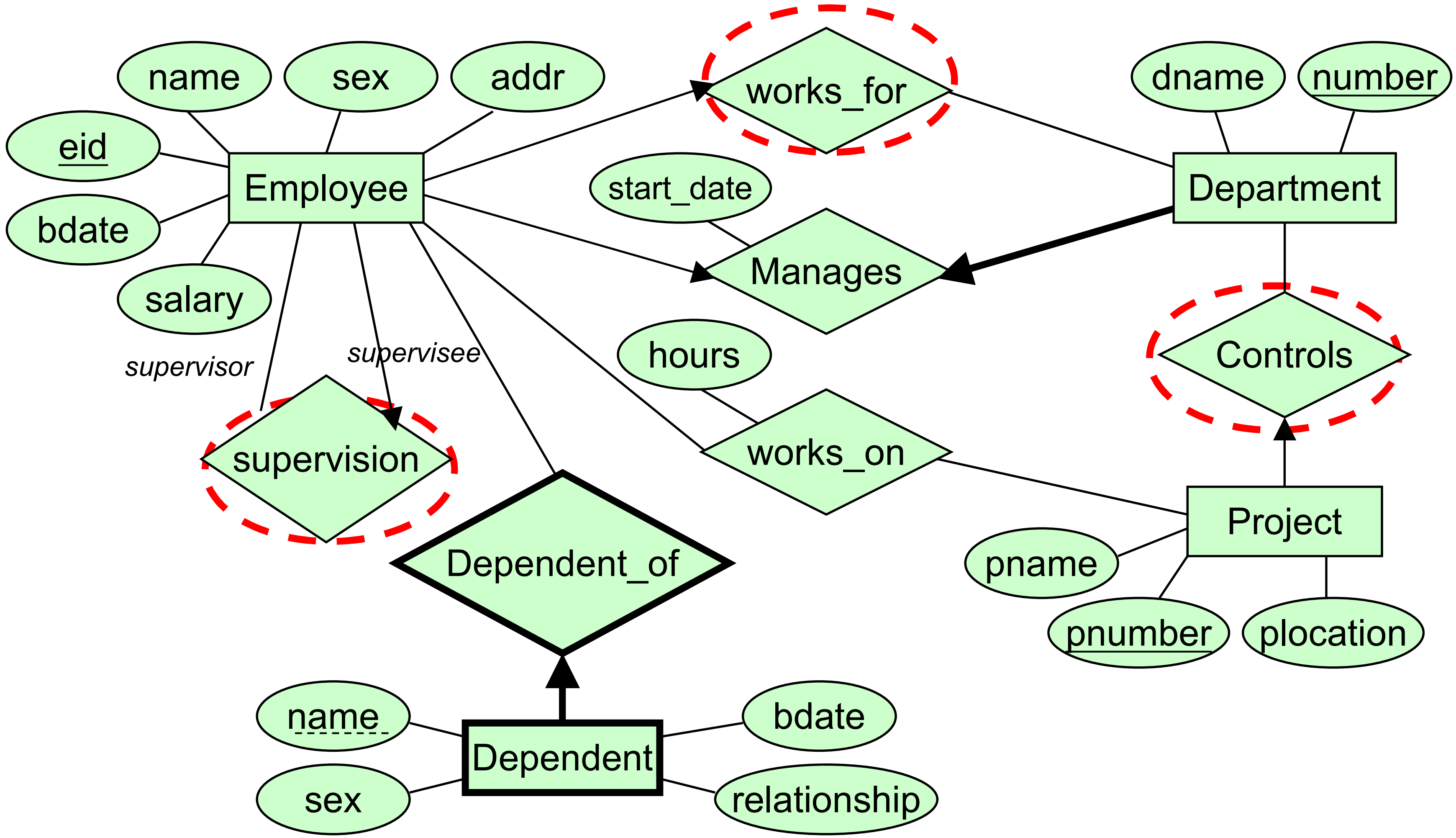
Step 4 (1-to-many Relationship)

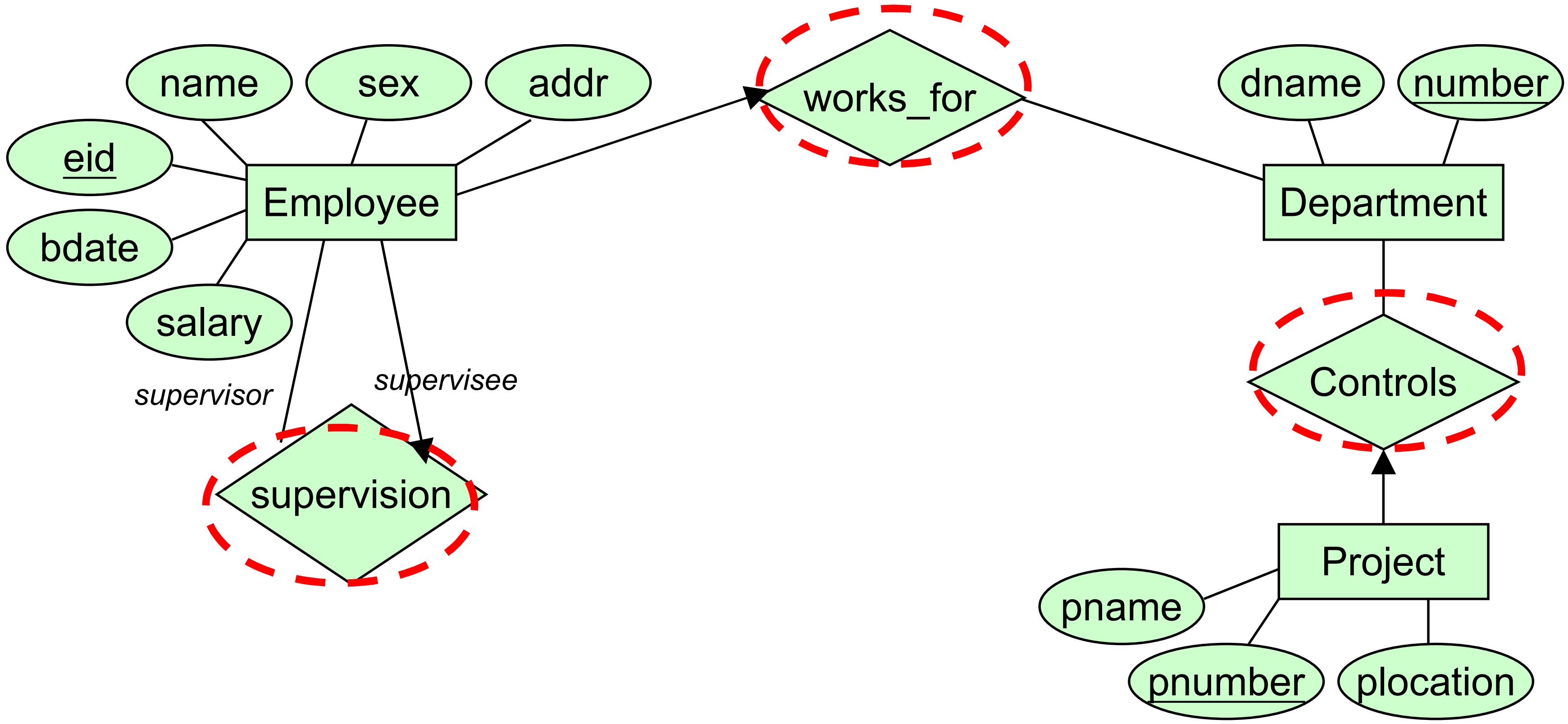
- For each **binary one-to-many** relationship set



  - Include as foreign key in S the primary key that represents the other entity set T participating in R
  - Include any attributes of the one-to-many relationship set as attributes of S
  - <u>In other words, we always choose the many side as S</u>

Step 4 (1-to-many Relationship)




name
sex
addr
eid
Employee
bdate
salary
supervisor
supervisee
supervision
works_for
dname
number
Department
Controls
Project
pname
pnumber
plocation

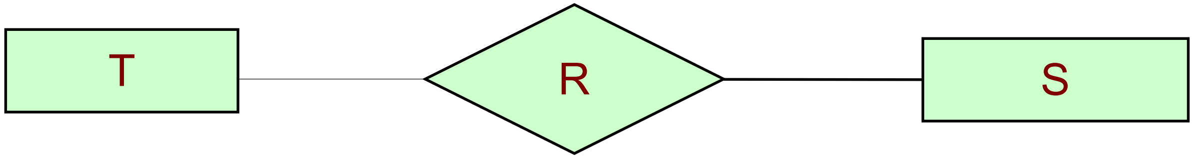
- The primary key dnumber of the DEPARTMENT relation schema is included as foreign key in the EMPLOYEE relation schema.
- We rename it as dno. (The renaming is not necessary but makes the name more meaningful.)

EMPLOYEE

name	eid	bdate	addr	sex	salary	dno

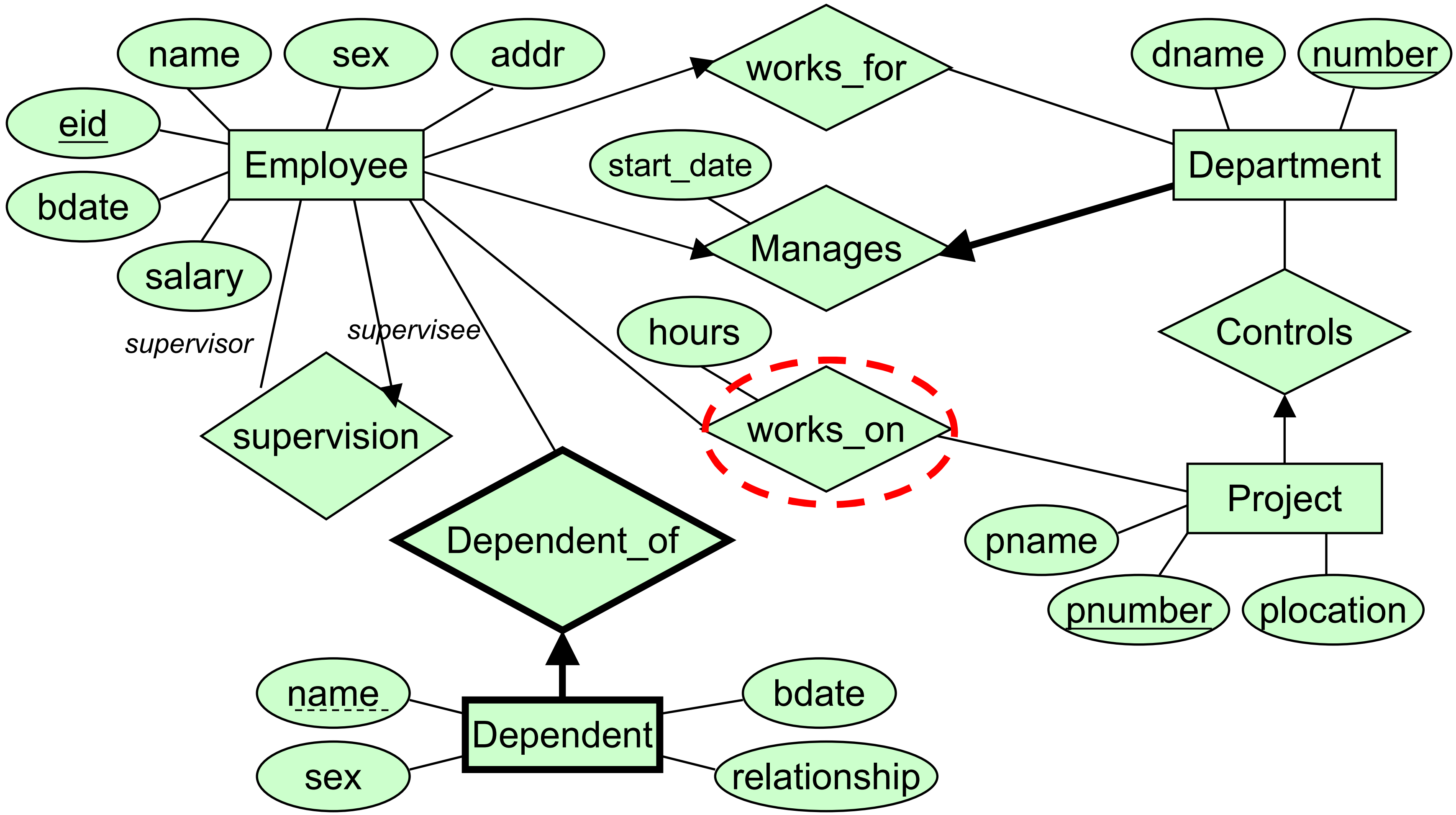
Step 5 (Many-to-many Relationship)

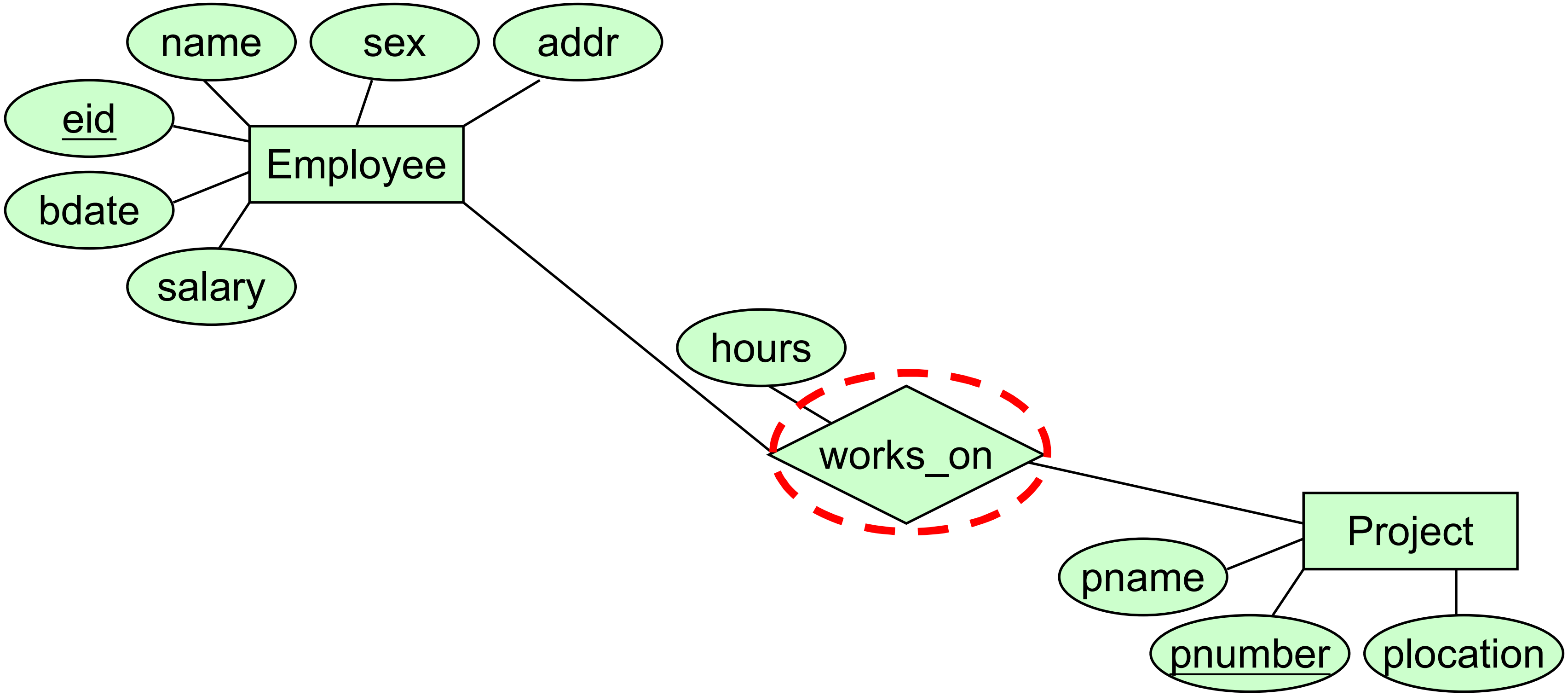
- For each binary many-to-many relationship set R



  - Create a new relation schema P to represent R
  - Include as foreign key attributes in P the primary keys of the relation schemas for the participating entity sets in R
  - Their combination will form the primary key of P
  - Also include attributes of the many-to-many relationship set as attributes of P

Step 5 (Many-to-Many Relationship)




name
sex
addr
eid
Employee
bdate
salary
hours
works_on
Project
pname
pnumber
plocation

- **create a new** relation schema S to represent R.

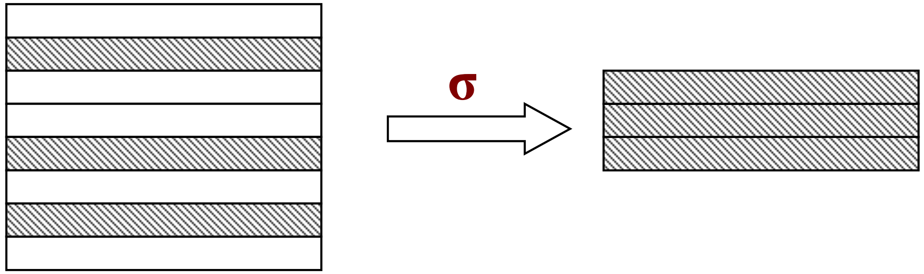
WORKS_ON

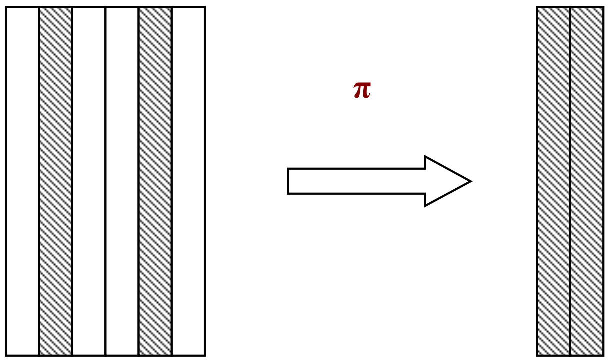
eid	pnumber	hours

- Map the many-to-many relationship sets
- WORKS_ON by creating the relation schema WORKS_ON.
- Include the primary keys of PROJECT and EMPLOYEE as foreign keys.

Lecture 4 – Relational Algebra

- Basic operators in relation algebra:
  - select (σ)
  - project (Π)
  - union (U)
  - set different ($-$)
  - Cartesian product (x)
  - rename (ρ)
  - Intersect($\cap$)
- Combinatory operators
  - Join ($\bowtie$)
  - Division (/)





- PROJECT creates a vertical partitioning

Lecture 5 – SQL

- Introduction
- Basic SQL
- Union, Intersect, and Except
- Nested SQL
- Division
- Aggregate Operators
- Group by and Having

Introduction

- The SQL (Structured Query Language) language has several aspects to it
  - The Data Manipulation Language (DML)
    - The subset of SQL that allows users to pose queries and to insert, delete, and modify rows.
  - The Data Definition Language (DDL)
    - The subset of SQL that supports the creation, deletion, and modification to definitions for tables and views.
  - The Data Control Language (DCL)
    - The subset of SQL that are used for access control and permission management for users in the database. DCL commands are GRANT and REVOKE.

SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

SELECT	<attribute list>
FROM	<table list>
[**WHERE**	<condition>]
[**GROUP BY**	<grouping attribute(s)>]
[**HAVING**	<group condition>]
[**ORDER BY**	<attribute list>]

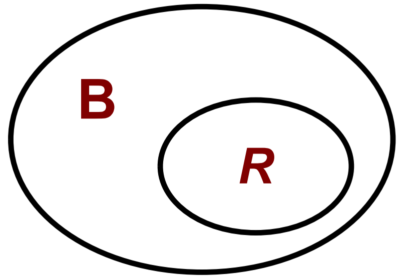
Union, Intersect, and Except

- SQL provides three set-manipulation constructs that extend the basic query form presented earlier.
  - Union ($\cup$)
  - Intersection ($\cap$)
  - Except ($-$)
- Many systems recognize the keyword MINUS for EXCEPT
- By default, duplicates are eliminated
- Sets must be union-compatible

Nested Queries

- A nested query is a query that has another query embedded within it.
- The embedded query is called a subquery.
- The embedded query can be a nested query itself.
  - Queries may have very deeply nested structures.

Division in SQL



Find sailors who've reserved all boats.

NOT EXISTS (B – R)

```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS
          ((SELECT  B.bid
            FROM  Boats B)
           EXCEPT
           (SELECT  R.bid
            FROM  Reserves R
            WHERE  R.sid=S.sid))
```

All boats

All boats reserved by S

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

Note that this query is correlated – for each sailor S, we check to see that the set of boats reserved by S includes every boat.

Logic: there does not exist any boat that is not reserved by S

An Example

Sailors

sid	sname	rating	age
s01	Tom	10.2	26
s02	James	20.3	38

Reserves

sid	bid	day
s01	b01	2003
s01	b02	2004
s02	b01	2002
s02	b02	2003
s02	b03	2004

Boats

bid	bname	color
b01	Dragon	red
b02	Ocean	blue
b03	Roto	green

```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS
            ((SELECT  B.bid
              FROM  Boats B)
             EXCEPT
             (SELECT  R.bid
              FROM  Reserves R
              WHERE  R.sid=S.sid))
```

All boats {b01, b02, b03}

When sid=s01, then all the boats reserved by s01: {b01, b02}

When sid=s02, then all the boats reserved by s01: {b01, b02, b03}

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

An Alternative way to write the previous query without using EXCEPT

```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS  (SELECT  B.bid
                    FROM  Boats B
                    WHERE  NOT EXISTS  (SELECT  R.bid
                                        FROM  Reserves R
                                        WHERE  R.bid=B.bid
                                               AND R.sid=S.sid))
```

Intuitively, for each sailor we check that there is no boat that has not been reserved by this sailor.

Aggregate Operators

- SQL allows the use of arithmetic expressions.
- SQL supports five aggregate operations, which can be applied on any column of a relation.

COUNT([DISTINCT] A)	The number of (unique) values in the A column.
SUM ([DISTINCT] A)	The sum of all (unique) values in the A column.
AVG ([DISTINCT A)	The average of all (unique) values in the A column.
MAX (A)	The maximum value in the A column.
MIN (A)	The minimum value in the A column.

Views in SQL

- A view is a “virtual” table that is derived from other tables
- Allows for limited update operations
- Since the table may not physically be stored
- Allows full query operations
- A convenience for expressing certain operations
- Provide a mechanism to hide certain data from the view of certain users

Views

List (sid, bid) for sailors and boats they have reserved.

```
CREATE VIEW  SB (sid, bid)
  AS SELECT S.sid, R.bid
     FROM Reserves R NATURAL RIGHT OUTER JOIN Sailors S
```

sid	bid
22	101
31	NULL
58	103

View SB is a table which is not explicitly stored in the database.

A view can be used just like a base table.

Lecture 6 – FDs and Normalization

- Functional Dependencies (FDs)
- Normal Forms Based on Keys
- Decomposition
- BCNF Decomposition Algorith

Functional Dependencies

- A functional dependency (FD) is a kind of Integrity Constraint that generalizes the concept of a key

Let R be a relation schema and let $X \subset R$ and $Y \subset R$ be nonempty sets of attributes in R. An instance (relation) r of R satisfies the FD $X \rightarrow Y$. If the following holds for every pair of tuples t_1 and t_2 in r

If $t_1.X = t_2.X$, then $t_1.Y = t_2.Y$

The notation $t_1.X$ refers to the projection of tuple t_1 onto the attributes in X

- An FD $X \rightarrow Y$ essentially says that if two tuples agree on the values in attributes X, they must also agree on the values in attributes Y.

Information Deduced from FDs

- α is a superkey for R iff $\alpha \rightarrow R$
  - where R is taken as the schema for relation R
- α is a candidate key for R iff
  - $\alpha \rightarrow R$, and
  - for no γ that is a proper subset of α, $\gamma \rightarrow R$

Closure of a set of FDs

- The set of all FDs implied by a given set F of FDs is called the closure of F, denoted as F^+
- Armstrong's Axioms, can be applied repeatedly to infer all FDs implied by a set of FDs

Suppose X,Y, and Z are sets of attributes over a relation.

Armstrong's Axioms

- Reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$
  e.g., $\{ A, C \} \subseteq \{ A, B, C \} \Rightarrow ABC \rightarrow AC$
- Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$
  e.g., $A \rightarrow B \Rightarrow CA \rightarrow CB$
- Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
  e.g., $A \rightarrow BD$, $BD \rightarrow C \Rightarrow A \rightarrow C$

Additional Rules

- Sometimes, it is convenient to use some additional rules while reasoning about F^+

Suppose X,Y, and Z are sets of attributes over a relation.

- Union: if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Decomposition: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

- These additional rules are not essential in the sense that their soundness can be proved using Armstrong's Axioms

Attribute Closure

- Computing the closure of a set of FDs can be expensive
- In many cases, we just want to check if a given FD

  $X \rightarrow$ Y is in F^+

  X - a set of attributes

  F - a set of functional dependencies
- Attribute Closure X^+ with respect to F, is the set of attributes A such that $X \rightarrow$ A can be infered using the Armstrong's Axioms.

Given a set of attributes $A_1, \ldots, A_n$

The **closure**, $\{A_1, \ldots, A_n\}^+ = \{B \in \text{Atts} \mid A_1, \ldots, A_n \rightarrow B\}$

Algorithm: Attribute Closure

- α – a set of attributes
- α^+ – closure of α under F
- Algorithm to compute α^+:

closure := α

repeat until (there is no change) do

    for each $\beta \to \gamma$ do

        if $\beta \subseteq$ closure then closure := closure $\cup\ \gamma$

    end for

Example: Attribute Closure

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \}$
- To compute AG^+:

  closure = AG

  closure = ABG ($A \rightarrow B$)

  closure = ABCG ($A \rightarrow C$)

  closure = ABCGH ($CG \rightarrow H$)

  closure = ABCGHI ($CG \rightarrow I$)

Is AG a candidate key?

$AG \rightarrow R$

$A \rightarrow R$?

$G \rightarrow R$?

- To check if a given FD $AG \rightarrow HI$ is in F^+, Check if HI is in AG^+

Normal Forms

1^{st} Normal Form	No repeating data groups
2^{nd} Normal Form	No partial key dependency
3^{rd} Normal Form	No transitive dependency
Boyce-Codd Normal Form	Reduce keys dependency
4^{th} Normal Form	No multi-valued dependency
5^{th} Normal Form	No join dependency

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$

Boyce-Codd Normal Form (BCNF)

- R – a relation schema
- F – a set of functional dependencies on R
- R is in BCNF if for any $\alpha \rightarrow A$ in F
  - $\alpha \rightarrow A$ is trivial ($A \in \alpha$), *or*
  - α is a key for R

Third Normal Form (3NF)

- R – a relation schema
- F – a set of functional dependencies on R
- A – a single attribute in R
- R is in 3NF if for any $\alpha \to A$ in F
  - $\alpha \to A$ is trivial ($A \in \alpha$), *or*
  - α is a key for R, *or*
  - A is *part of some key(s)* for R
    - *part of some key(s)*: AB is a key for R
      - then A is a part of the key
      - and B is also a part of the key
- R is in BCNF $\Rightarrow$ R is in 3NF

Lossless Join Decompositions

A decomposition is *lossless* if we can recover:

R($\underline{A}$,B,C)

Decompose

R1($\underline{A}$,B) $\bowtie$ R2($\underline{A}$,C)

Recover

R'($\underline{A}$,B,C) should be = R($\underline{A}$,B,C)

R' is in general larger than R. Must ensure R' = R

Theorem: Lossless Join Decomposition

- R – a relation schema
- F – a set of functional dependencies on R
- { R_1, R_2 } is a lossless-join decomposition of R

$\Leftrightarrow$ $(R_1 \cap R_2) \rightarrow R_1 \in F^+$ or $(R_1 \cap R_2) \rightarrow R_2 \in F^+$

$\Leftrightarrow$ $(R_1 \cap R_2)$ is a *key* for R_1 or R_2

- (the attributes common to R_1 and R_2 must contain a key for either R_1 or R_2)

Dependency Preservation

- R – a relation schema
- F – a set of functional dependencies on R
- { R_1, R_2 } is a decomposition of R
- F_i – a subset of F with only attributes in R_i

  (i.e. the *projection* of F on R_i)
- Dependency is *preserved* if $\boxed{(F_1 \cup F_2)^+ = F^+}$

  e.g.: R = (A, B, C), F = {A→B, B→C}

  R_1 = (A, B) and R_2 = (B, C)

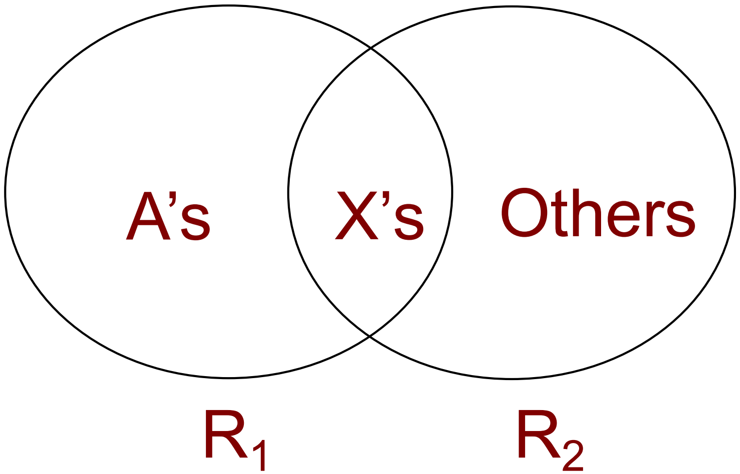
  F_1 = {A→B} and F_2 = {B→C}

Algorithm: BCNF Decomposition

Suppose R is not in BCNF, A is an attribute, and $X \to A$ ($X_1, \ldots, X_m \rightarrow A_1, \ldots, A_n$) is a FD that *violates* the BCNF condition.

1. Remove A from R
2. Decompose R into XA and R-A

   $R_1(X_1, \ldots, X_m, A_1, \ldots, A_n)$ and $R_2(X_1, \ldots, X_m, [others])$
3. Repeat this process until all the relations become BCNF



Exercise

Given

R = (A, B, C, D, E)

F = {A→BC, CD→E, B→D, E→A}

(1) Compute A^+ and B^+

(2) List all candidate keys of R

(3) Is R in 3NF, why?

(4) If R is not in BCNF, decompose it into BCNF.

Lecture 7 – Transaction

- Transaction
- ACID Property
- Serial Schedule
- Conflict Serializability

Transactions

- A collection of several operations on the database appears to be a single unit from the point of view of the database user. Within the database system, however, it consists of several operations.
- Collections of operations that form a single logical unit of work are called ***transactions***.

ACID Property

- Atomicity
  - In a transaction, either all operations are carried out or none are.
- Consistency
  - Regardless of other transactions, each transaction must preserve the consistency of the database
- Isolation
  - User can understand a transaction without considering the effect of other transactions
- Durability
  - The effect of transaction should persist forever whenever the transaction is completed/committed.

Serial Schedules

- Serial schedule
  - A schedule which the operations belonging to one single transaction appear together
  - E.g. H_1 is a serial schedule

    H_1: T_1T_2

    H_2 and H_3 are not serial schedule
- Serializable schedules
  - Equivalent to some serial schedule
  - E.g. H_1 and H_2 are serializable schedules (to T_1T_2)
  - H_3 is a serializable schedule (to T_2T_1).

H_1:

T_1:	R(A),	W(A),		
T_2:			W(B),	R(A)

H_2:

T_1:	R(A),		W(A),	
T_2:		W(B),		R(A)

H_3:

T_1:		R(A),		W(A)
T_2:	W(B),		R(A),	

Conflict Serializability

- Two operations are **conflict** if
  1. They are operations of different transactions on the **same** data object
  2. At least one of them is a **Write** operation
  - E.g.

T_i:	W(X),	
T_j:		R(X)

T_i:	R(X),	
T_j:		W(X)

T_i:	W (X),	
T_j:		W(X)

- Two operations are **non-conflict**
  - E.g.

T_i:	R(X),	
T_j:		R(X)

T_i:	W(X),	
T_j:		R(Y)

T_i:	R(X),	
T_j:		W(Y)

T_i:	W (X),	
T_j:		W(Y)

Conflict Equivalent

- Two schedules S_1 and S_2 are **conflict equivalent** if
  - S_1 and S_2 involve the *same operations* of the *same* transaction
  - Every pair of conflicting *operations* is *ordered* in the *same* way in S_1 and S_2

Conflict Serializability

- S is **conflict serializable** if it is conflict equivalent to a serial schedule
- E.g.

H_8:

T_1:	W (X),		R(Y),	
T_2:		R(Y),		R(X)

H_9:

T_1:	W (X),	R(Y),		
T_2:			R(Y),	R(X)

- H_8 and H_9 are conflict equivalent
- H_9 is a serial schedule
- H_8 is conflict serializable

Conflict-serializability - Both schedules have the same sets of respective chronologically ordered pairs of conflicting operations.

Precedence Graph

- Test for conflict serializability
- A directed graph G=(V,E), where
  - V includes all transactions involved in the schedule
  - E consists of all edges $T_i \rightarrow T_j$ for which one of three conditions holds:

Conflict Operations

    - T_i executes write(X) before T_j executes read(X)
    - T_i executes read(X) before T_j executes write(X)
    - T_i executes write(X) before T_j executes write(X)

T_i:	W(X),	
T_j:		R(X)

T_i:	R(X),	
T_j:		W(X)

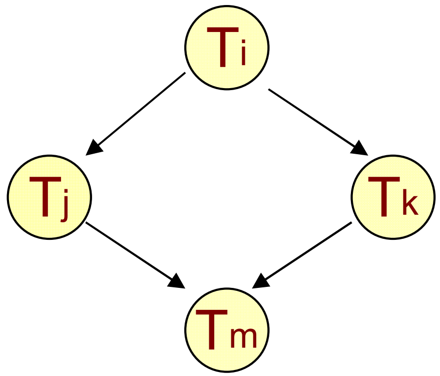
T_i:	W (X),	
T_j:		W(X)

Precedence Graph

- ***Theorem:*** A schedule S is conflict serializable iff G(S) is **acyclic** **(i.e. no cycle)**
- The serialization order can be obtained through ***topological sorting***, which determines a linear order consistent with the partial order of the dependence graph

Precedence Graph

- $w_i[a]\ r_k[a]\ r_i[b]\ w_j[b]\ w_k[c]\ r_m[c]\ w_j[d]\ r_m[d]$



Precedence graph

Two possible serialization orders