Lecture 11: Transforming Conceptual Schema to Logical Schema

CSX3006 DATABASE SYSTEMS

ITX3006 DATABASE MANAGEMENT SYSTEMS

Outline

- Why multiple database design steps?
- Transformation of ER model to relational schema
  - Representing Strong Entity Set
  - Representing Composite Attributes
  - Representing Multi-valued attributes
  - Representing Weak Entity Sets
  - Representing 1:1 Relationship Sets
  - Representing 1:M Relationship Set
  - Representing M:M Relationship Sets
  - Relationship Representation and Enforcement of Constraints
  - Representing N-ary Relationships Sets
  - Representing Unary Relationship Sets
  - Representing Inheritance Hierarchy
  - Representing Aggregation

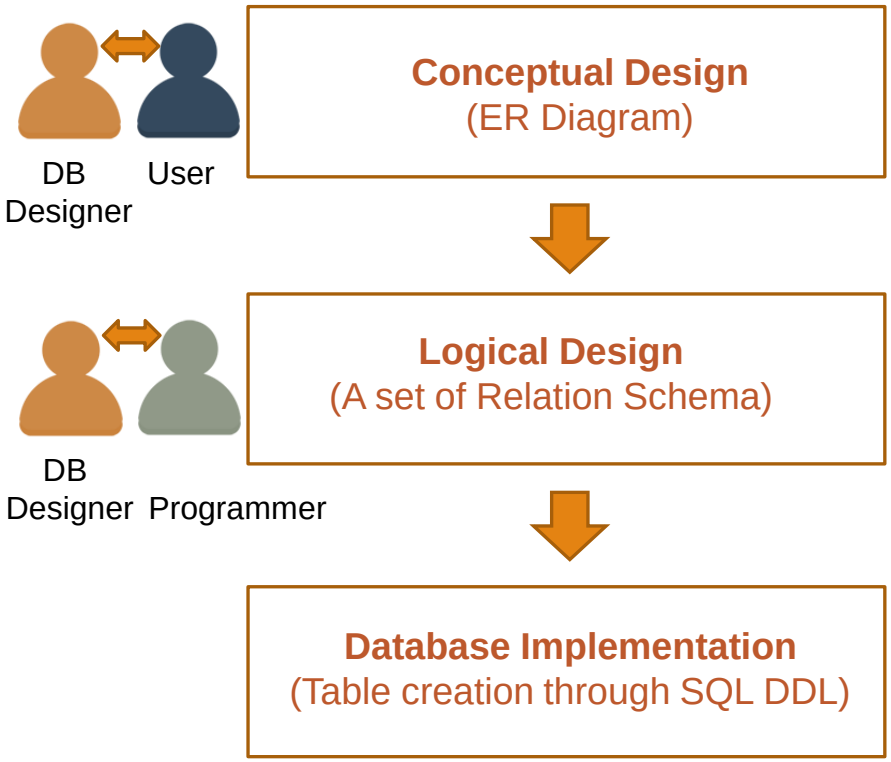
Conceptual Schema vs. Logical Schema

- Conceptual Schema
  - High level abstractions and constructs
    - More natural expressions of the data requirements and business rules
    - Captures the *semantics of data* in an enterprise being modelled
    - Captures some of the *business rules* in the enterprise
  - ER model is mostly adapted for designing the conceptual schema

Conceptual Schema vs. Logical Schema

- Logical Schema
  - 'Middle-level' abstractions and constructs
    - Captures the *data requirements of applications* in a format that is *independent of underlying physical file structures, data structure and organizations.*
  - Relational model is mostly adapted for designing the logical schema

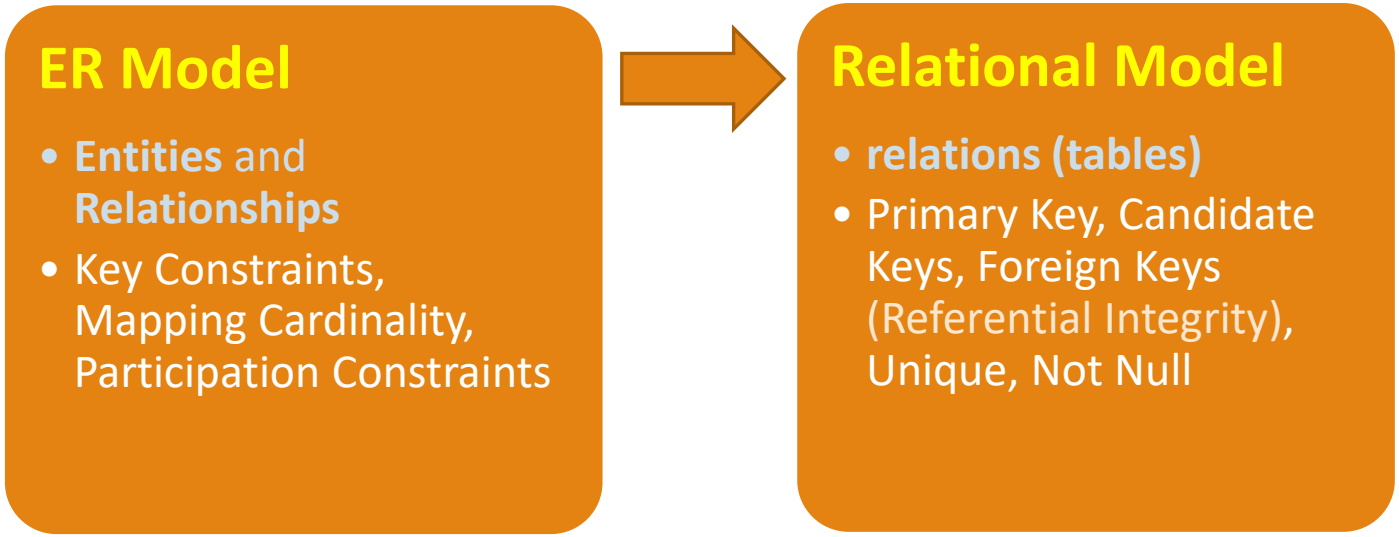
Why multiple design steps?



- No need to worry about performance aspects or limitations of a specific DBMS;
  - Focus is on producing a correct model of the real world
  - No DBMS can directly support the high level conceptual schema **efficiently**
  - Logical schema are more appropriate for *expressing application logic* by application programmers.
- Note: the conceptual design is not invalided if a different DBMS is used later

So what's required?

- Transform the conceptual schema into the logical schema supported by a chosen DBMS.



Transformation of ER model to relational schema

- How to represent Strong Entity Sets in relational schema
- How to represent Composite Attributes in relational schema
- How to represent Multi-valued Attributes in relational schema
- How to represent Weak Entity Sets in relational schema
- How to represent Relationship Sets in relational schema
  - Effects of Cardinality Mappings in the transformation process
  - Effect of Participation Constraints in the transformation process
  - Transformation of ER level Constraints into Relational level constraints
- Representing Generalization/Specialization Hierarchy
- Representing Aggregation

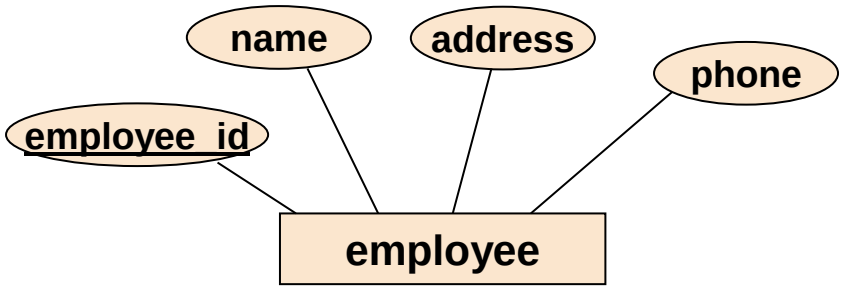
Representing Strong Entity Set - 1

- A **Strong Entity Set** is represented as a **relation** (a **table**) at logical level

1. **Create a table** with the name of the **strong entity set** being transformed
2. Every attribute is transformed into a **column** of the table
3. The **key attribute** (or a set of attributes in the case of composite key) is made **primary key** of the table

ER Model		Relational Model
Strong entity set		Table
Simple attribute		Column
Key attribute		Primary key

Representing Strong Entity Set - 2



ER Model

Employee = (employee_id, name, address, phone)

Relational Model

```
create table employee (
    employee_id          char(10),
     name                char(25),
     address             char(30),
     phone               char(30),
    primary key (employee_id)
);
```

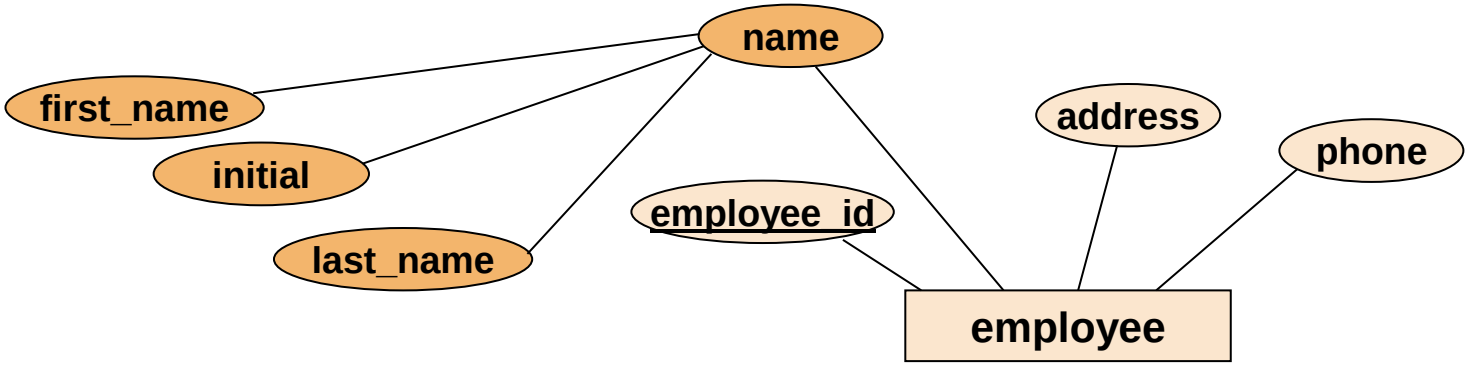
SQL DDL

Representing Composite Attributes - 1

- **Composite attributes** are flattened out by creating **a *separate* attributes *for each*** component attribute

ER Model	Relational Model
Composite attributes 	Separated attributes

Representing Composite Attributes - 2



Employee = (employee_id, first_name, initial, last_name, address, phone)

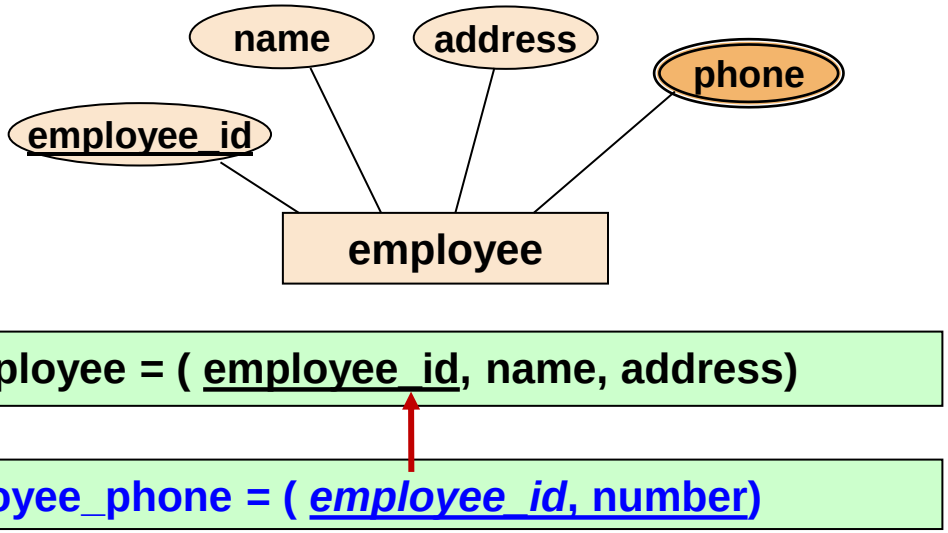
```
create table employee (
     employee_id                    char(10),
      first_name                    char(15),
      initial                       char(1),
      last_name                     char(15),
      address                       char(30),
      phone                         char(10),
     primary key (employee_id)
);
```

Representing Multi-valued attributes - 1

- **A multi-valued attribute** M of entity set E is represented by **a separate relation** EM
  - Schema *EM* has the following attributes:
    - Attributes: the **PK of *E*, multi-valued attribute *M***
    - **Primary key (PK)**
      - *The **PK** of **E** + the **multi-valued attribute M***
    - **Foreign key (FK)**
      - ***Reference to the PK of the E.***
      - ***This FK** CANNOT BE NULL*

ER Model	Relational Model
Multi-valued attribute M Of entity set E -- e.g., E = (K, M, N, O) K is PK.	1. Revised relation E, and 2. A separate relation EM, with FK -- e.g., $E_{revised}$ = (K, N, O) EM = (K, M)

Representing Multi-valued attributes - 2




```
create table employee (
        employee_id             char(10),
        name                    char(15),
        address                 char(30),
        primary key (employee_id)
);
```

```
create table employee_phone (
        employee_id             char(10),
        number                  char(10),
        primary key (employee_id, number),
        foreign key (employee_id) references
                employee(employee_id)
                on delete cascade
                on update cascade
);
```

Representing Multi-valued attributes - 3

Employee = (employee_id, name, address)

Employee_phone = (*employee_id*, number)

employee_id	name	address
101	John	2/51
~~102~~	~~Sara~~	~~35/8~~
103	Pete	1/20

employee_id	number
101	0811111111
101	0822222222
102	083333333

- What happens when **an employee is deleted** from the Employee table?

```
create table employee (
    employee_id          char(10),
    name                 char(15),
    address              char(30),
    primary key (employee_id)
);
```

```
create table employee_phone (
    employee_id          char(10),
    number               char(10),
    primary key (employee_id, number),
    foreign key (employee_id) references
        employee(employee_id)
        on delete cascade
        on update cascade
);
```

Representing Multi-valued attributes - 4

Employee = (employee_id, name, address)

Employee_phone = (*employee_id*, number)

employee_id	name	address
101	John	2/51
~~102~~	~~Sara~~	~~35/8~~
103	Pete	1/20

employee_id	number
101	0811111111
101	0822222222
~~102~~	~~083333333~~

- What happens when **an employee is deleted** from the Employee table?
  - **All corresponding tuples from the employee_phone must be deleted**

```
create table employee (
        employee_id             char(10),
        name                    char(15),
        address                 char(30),
        primary key (employee_id)
);
```

```
create table employee_phone (
        employee_id             char(10),
        number                  char(10),
        primary key (employee_id, number),
        foreign key (employee_id) references
                employee(employee_id)
                on delete cascade
                on update cascade
);
```

Representing Multi-valued attributes - 5

Employee = (employee_id, name, address)

Employee_phone = (*employee_id*, number)

employee_id	name	address
101	John	2/51
~~102~~ 105	Sara	35/8
103	Pete	1/20

employee_id	number
101	0811111111
101	0822222222
102	083333333

- What happens when **an employee's id** of the Employee table is **updated**?

```
create table employee (
    employee_id          char(10),
    name                 char(15),
    address              char(30),
    primary key (employee_id)
);
```

```
create table employee_phone (
    employee_id          char(10),
    number               char(10),
    primary key (employee_id, number),
    foreign key (employee_id) references
        employee(employee_id)
        on delete cascade
        on update cascade
);
```

Representing Multi-valued attributes - 6

Employee = (employee_id, name, address)

Employee_phone = (*employee_id*, number)

employee_id	name	address
101	John	2/51
~~102~~ 105	Sara	35/8
103	Pete	1/20

employee_id	number
101	0811111111
101	0822222222
~~102~~ 105	083333333

- What happens when **an employee's id** of the Employee table is **updated**?
  - **All corresponding tuples from the Employee_phone must be updated**

```
create table employee (
    employee_id          char(10),
    name                 char(15),
    address              char(30),
    primary key (employee_id)
);
```

```
create table employee_phone (
    employee_id          char(10),
    number               char(10),
    primary key (employee_id, number),
    foreign key (employee_id) references
        employee(employee_id)
        on delete cascade
        on update cascade
);
```

Representing Multi-valued attributes - 7

Employee = (employee_id, name, address)

Employee_phone = (*employee_id*, number)

employee_id	name	address
101	John	2/51
102	Sara	35/8
103	Pete	1/20

employee_id	number
101	0811111111
101	0822222222
102	083333333
109	**0899999999**

- Can we **insert a tuple** to Employee_phone WITHOUT having a corresponding Employee?

```
create table employee (
    employee_id          char(10),
    name                 char(15),
    address              char(30),
    primary key (employee_id)
);
```

```
create table employee_phone (
    employee_id          char(10),
    number               char(10),
    primary key (employee_id, number),
    foreign key (employee_id) references
        employee(employee_id)
        on delete cascade
        on update cascade
);
```

Representing Multi-valued attributes - 8

Employee = (employee_id, name, address)

Employee_phone = (*employee_id*, number)

FK: NOT NULL

employee_id	name	address
101	John	2/51
102	Sara	35/8
103	Pete	1/20

employee_id	number
101	0811111111
101	0822222222
102	083333333
~~109~~	~~0899999999~~

Cannot!!

- Can we **insert a tuple** to Employee_phone WITHOUT having a corresponding Employee?
  - Cannot. (**This rule** can be enforced by making the **foreign key** to be **NOT NULL**)

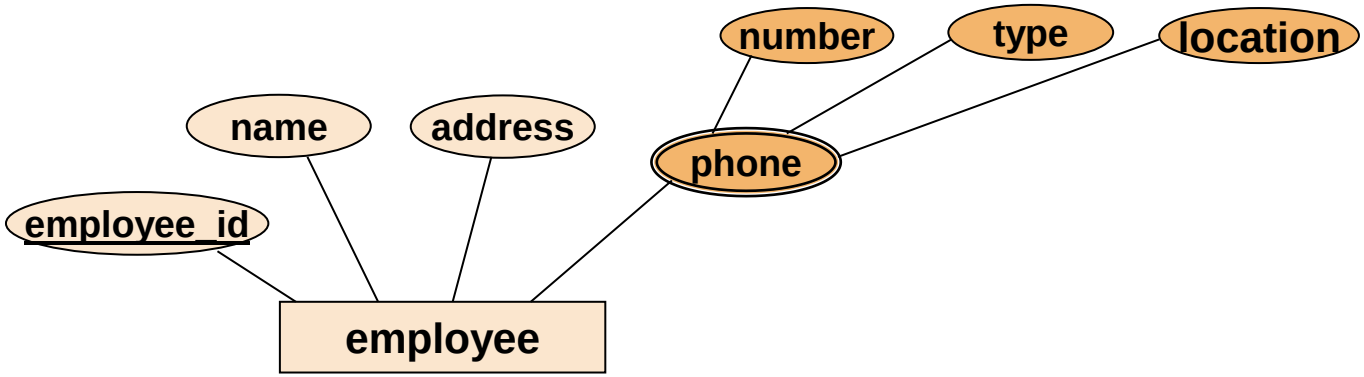
Implies UNIQUE NOT NULL, but

'number' alone needs not be unique

```
create table employee (
        employee_id             char(10),
        name                    char(15),
        address                 char(30),
        primary key (employee_id)
);
```

```
ate table employee_phone (
        employee_id             char(10),
        number                  char(10),
        primary key (employee_id, number),
        foreign key (employee_id) references
                employee(employee_id)
                on delete cascade
                on update cascade
);
```

Representing Multi-valued Composite attributes -1



- What should be the **primary key** of the **employee_phone table**?

Employee = (employee_id, name, address)

Employee_phone = (*employee_id*, number, type, location)

Representing Multi-valued Composite attributes -2

- Depends on the business logic!
  - **Case 1:** an employee can have *multiple phone numbers, but a number is guaranteed to be unique* within the scope of an employee.
    - **PK = (employee_id, number)**
  - **Case 2:** an employee may have *two phones in two different locations, but* they happen to have the *same number*!
    - **PK = (employee_id, number, location)**

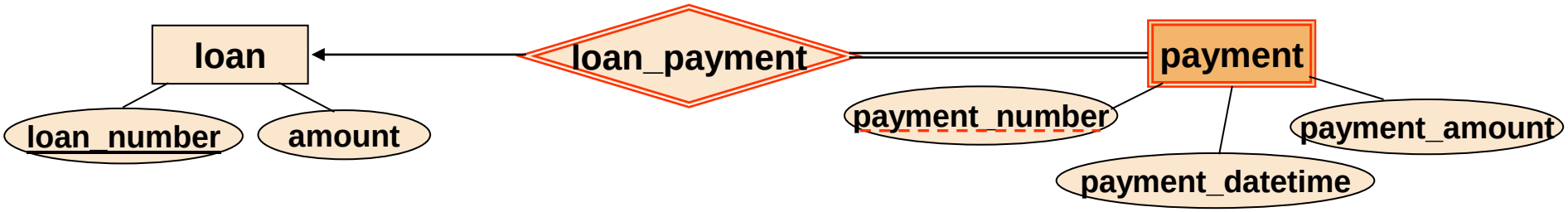
- **Think of Referential Integrity Constraints Settings Required!**

Representing Weak Entity Sets - 1

- A weak entity set is made into a single relation (table) and the identifying relationship set is captured as a foreign key *referring the primary key of owner set*.
  - Two tables linked by *a **foreign key** reflecting the **identifying set*** ← weak entity set
    - **The foreign key cannot be null**
      - When the (strong) identifying entity is deleted (or updated), all weak entities depending on the strong entity must be deleted (or updated)

ER Model	Relational Model
O: owner entity set, W: weak entity set with identifying relationship set I O = (A, B) I = (A, X) W = (X, Y)	A relation W with FK from the owner entity set O = (A, B) W = (A, X, Y)

Representing Weak Entity Sets - 2



Loan = (loan_number, amount)

Payment = (*loan_number*, pay_number, pay_amount, pay_datetime)

FK: NOT NULL

Representing Weak Entity Sets - 3

Loan = (loan_number, amount)

Payment = (*loan_number*, pay_number, pay_amount, pay_datetime)

FK: NOT NULL

```
create table loan (
      loan_number        char(10),
       amount            numeric(9,2),
       primary key (loan_number)
);
```

```
create table payment (
      loan_number                char(10),
       pay_number                numeric(4),
       pay_amount                numeric(9,2),
       pay_datetime              timestamp,
       primary key (loan_number, pay_number),
       foreign key (loan_number) references
                loan(loan_number)
                on delete cascade
                on update cascade
);
```

- **loan_number can't be null** since it's part of the primary key
  - **Implicitly NOT NULL**

Remark: constraints of FK

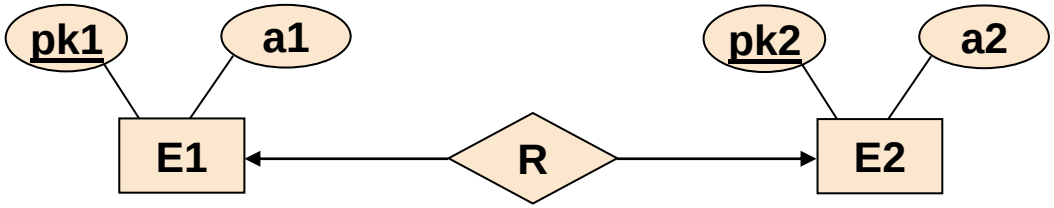
- We must consider if we can specify the min. and max. cardinalities on each FK.
  - Min. cardinality = 1 → specify by using 'NOT NULL' constraint
  - Max. cardinality = 1 → specify by using 'UNIQUE' constraint
  - Min. cardinality = 0 → no need to specify
  - Max. cardinality = n → no need to specify
    - However, for Max. cardinality = 1, if the FK is also PK of the table, it automatically implies 'UNIQUE'.

FK { Min. cardinality = ?
Max. cardinality = ?

Contract_ID	Job	Start_Date	End_Date	Staff_ID
100	Programmer	1 Jan. 2019	2 Jan. 2020	201901
101	IT Support	1 Jan. 2020	2 Jan. 2021	202001
102	Programmer	1 Feb. 2020	1 Feb. 2021	202002

Representing 1:1 Relationship Sets – 1

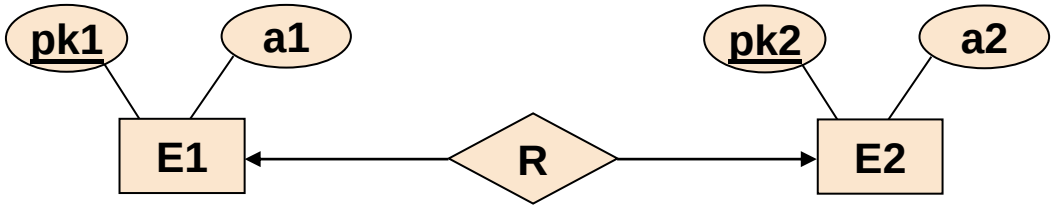
One-to-One Relationship Sets



- An entity on the left can be associated with at most one entity on the right *and vice versa*.

Representing 1:1 Relationship Sets – 2

One-to-One Relationship Sets



Alternative 1

E12 = (pk1, a1, pk2, a2)

Alternative 2

E1 = (pk1, a1)

E2 = (pk2, a2, *fk1*)

Alternative 3

E1 = (pk1, a1, *fk2*)

E2 = (pk2, a2)

Alternative 4

E1 = (pk1, a1)

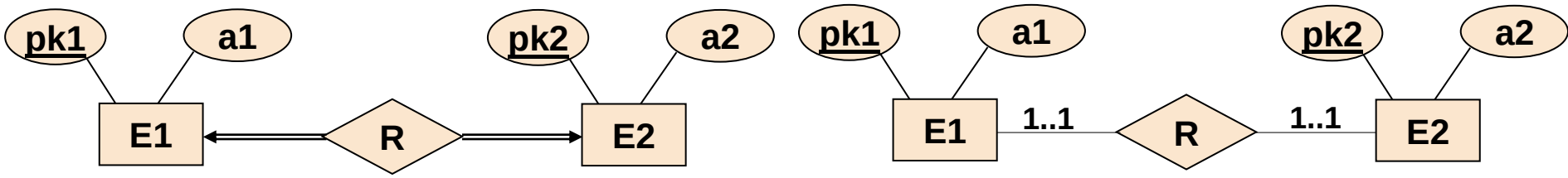
R = (*fk1*, *fk2*)

E2 = (pk2, a2)

- Total participation on both sides (1..1 ; 1..1)
- Total-Partial; Partial-Total (1..1 ; 0..1) (0..1 ; 1..1)
- Partial on both sides (0..1 ; 0..1)

Representing 1:1 Relationship Sets – 1

Total Participation at BOTH sides



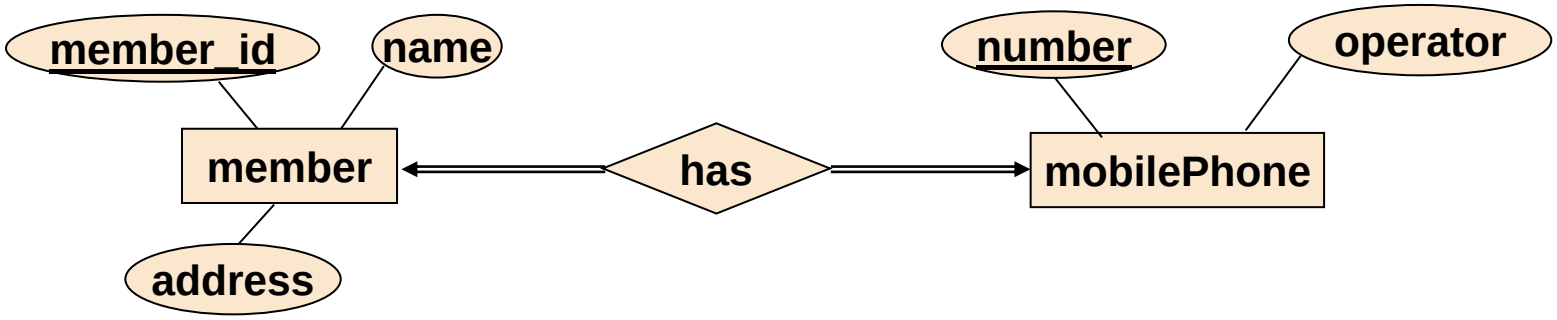
- It is SOMETIMES possible to **subsume** one entity set into the other
  - The choice *depends on* ***which is the most important entity set*** (more attributes, better key, semantic nature of them).

E12 = (pk1, a1, $\overline{\text{pk2}}$, a2)

  - **The PK of the new combined entity** => the ***original more important entity set***
  - **The candidate key** *=> the* ***key of the subsumed entity set***
  - If there are any attributes in common, the duplicates are removed.

Representing 1:1 Relationship Sets – 2

Total Participation at BOTH sides



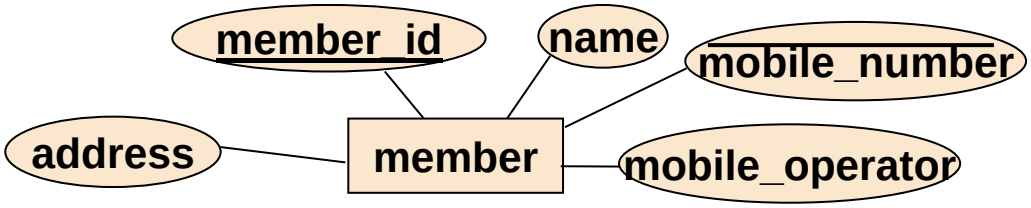
- **Example:** One-to-One between member and mobilePhone BOTH total participation
  - Each member is *required* to have *one and only one* mobile number.
  - Each mobile number is *required* to belong to *one and only one* member.

member = (member_id, name, address, mobile_number, mobile_operator)

Representing 1:1 Relationship Sets – 3

Total Participation at BOTH sides

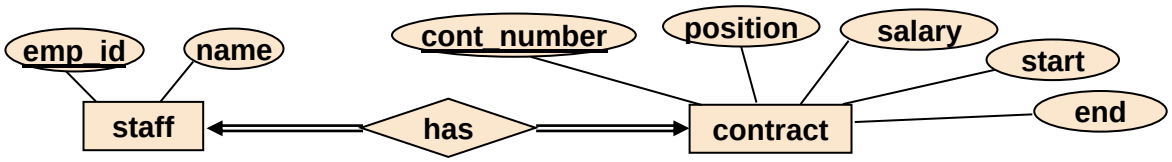
- **So here is another design alternative even at ER level**
  - Pros and cons of each alternatives?
    - WHEN should NOT Combine?



member = (member_id, name, address, mobile_number, mobile_operator)

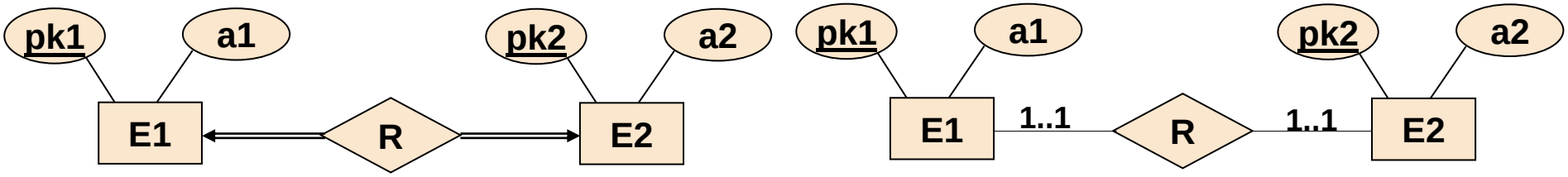
Representing 1:1 Relationship Sets – 4
Total Participation at BOTH sides

- **When not to combine**
  - the two entity types ***represent different entities*** in the `real world'.
  - the entities ***participate in*** very **different relationships *with other* entities.**
  - efficiency considerations:
    - fast responses are required
    - different patterns of updating occur to the two different entity types.
      - *E.g., Yearly contract staff* *(sign a new contract in yearly basis)*



Representing 1:1 Relationship Sets – 5

Total Participation at BOTH sides



- **If NOT COMBINED into a single entity set,**
  - The PK of one entity set becomes the FK in the other.
  - Note: **should NOT have a FK in BOTH entity**.
  - **Referential Integrity on the FK**
    - **must NOT be NULL** → enforces the **MINimum cardinality of 1**
    - **must be UNIQUE** → enforces the **MAXimum cardinality of 1**

E1 = (pk1, a1)

E2 = (pk2, a2, *fk1*)

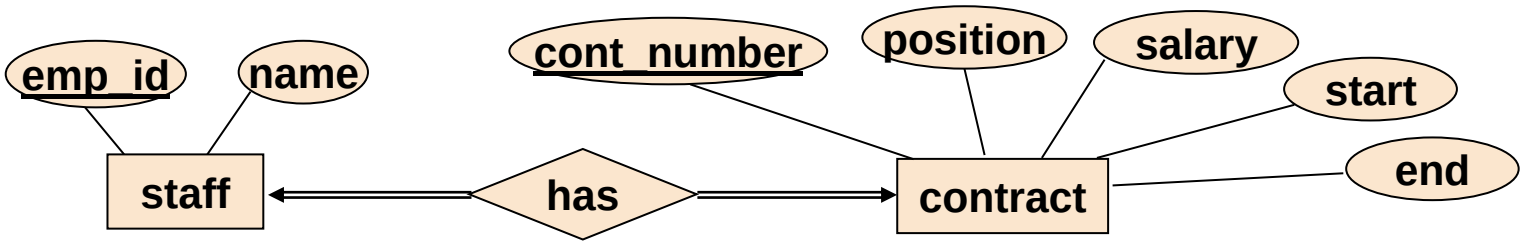
OR

E1 = (pk1, a1, *fk2*)

E2 = (pk2, a2)

Representing 1:1 Relationship Sets – 6

Total Participation at BOTH sides



Choice 1 (combine entity set)

Staff = (emp_id, name, cont_number, position, salary, start, end)

Side Note: Integrity Constraints - 1

Choice 2

Staff = (emp_id, name, *cont_number*)

Contract = (cont_number, position, salary, start, end)

Referential Integrity on the FK

NOT NULL, UNIQUE

ON UPDATE CASCADE, ON DELETE RESTRICT

- **CAN**
  - Insert a new contract *without* a corresponding staff member
    - Need application logic to make sure that there will be a staff member which will be associated with this new contract.
  - Delete a staff member; consequently, the associated contract will *not* have the corresponding staff member anymore
    - Need application logic to ensure that the contract is deleted
- **CANNOT**
  - Insert a staff member which does not have a contract
  - Insert a staff member having the same contract number as another existing staff member (must be one staff per contract).

Side Note: Integrity Constraints - 2

Choice 3

Referential Integrity on the FK

NOT NULL, UNIQUE

ON UPDATE CASCADE, ON DELETE CASCADE

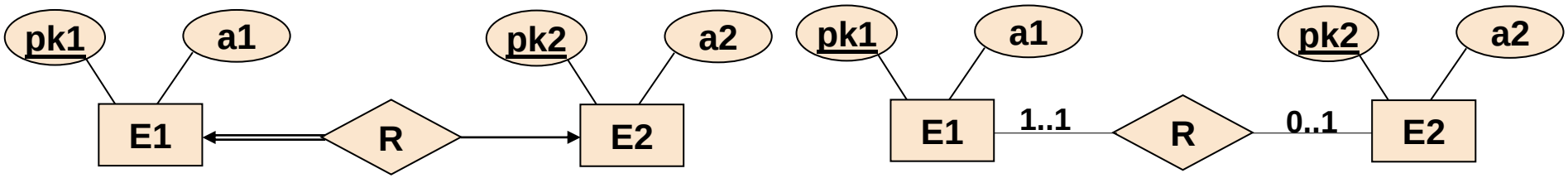
Staff = (emp_id, name)

Contract = (cont_number, position, salary, start, end, *emp_id*)

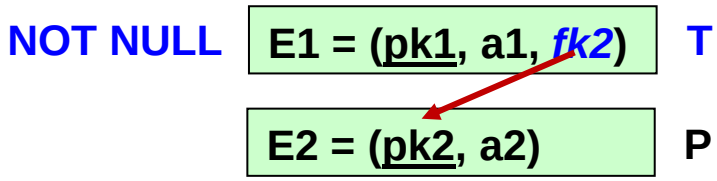
- **CAN**
  - Insert a staff member *without* a contract
  - Delete a contract; consequently, a staff member will become contract-less!
- **CANNOT**
  - Insert a contract which is not associated with a staff member
  - Insert a contract associated with the same staff as an existing contract
    - Each staff is associated with maximum of one contract

Representing 1:1 Relationship Sets – 1

One-to-One with one Partial participation

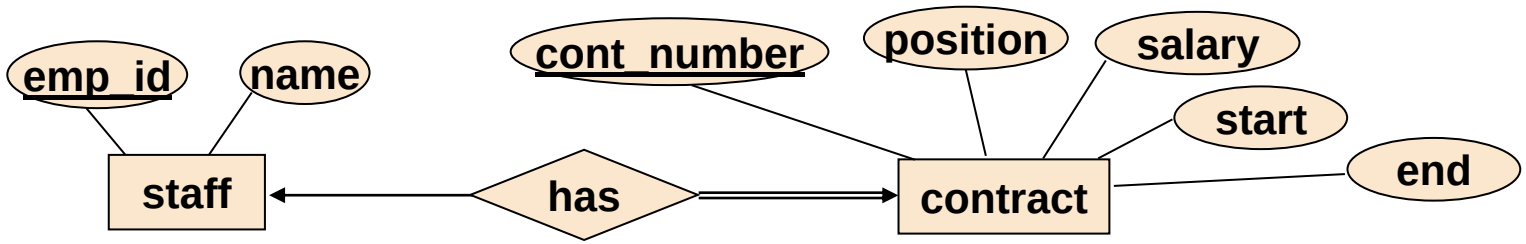


- Two entities sets are kept in *separate tables*, and
- The *association* between them are *made through a **FK** **from** the **total** side **to** the **partial** side*
  - Avoid null value in FK.



Representing 1:1 Relationship Sets – 2

One-to-One with one Partial participation



Note: *what to do on delete and on update depends on application!*

Referential Integrity on the FK

NOT NULL, UNIQUE

ON UPDATE CASCADE, ON DELETE CASCADE

Staff = (emp_id, name)

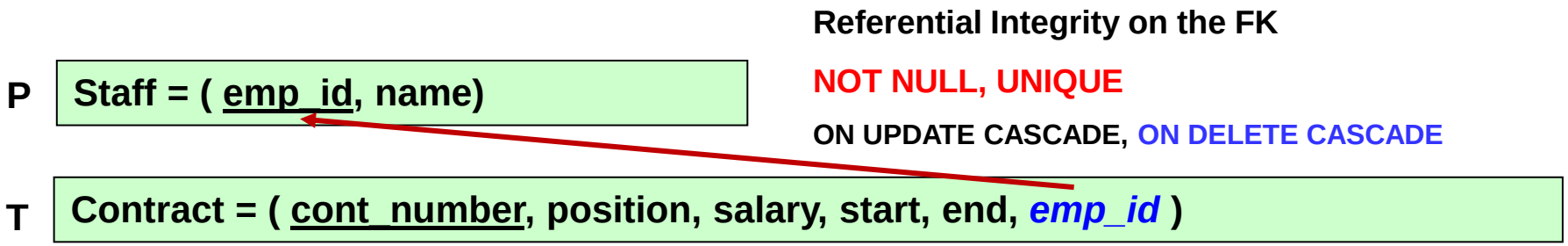
Contract = (cont_number, position, salary, start, end, *emp_id*)

Cont_ number	position	salary	start	end	Emp_id (FK)
C111	…	…	…	…	101
c222	…	…	…	…	102
c333	…	….	…	…	105

Emp_id	name	Cont_number (FK)
101	**A**	C111
102	**B**	c222
103	C	null

Not good!

Side Note: Integrity Constraints

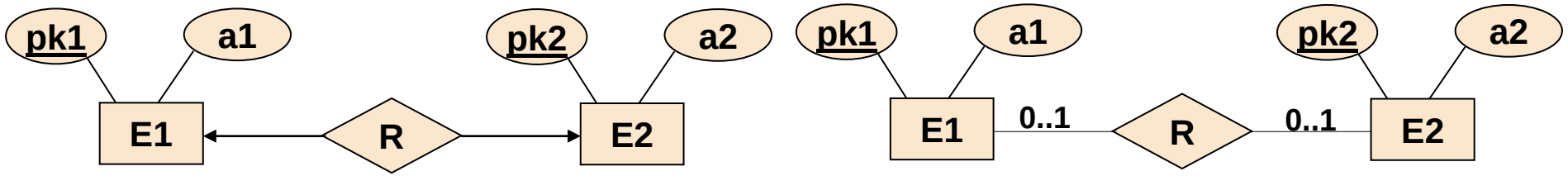


- **NOT NULL on the FK -> Cannot insert** a contract without a staff member
- **UNIQUE on the FK ->** A staff member can be associated with only one contract
- **ON UPDATE (DELETE) CASCADE**-> Updating (deleting) an emp_id in Staff table will result in updating (deleting) also in the Contract table.

- **What if a staff member is inserted without a contract?** ok
- **What if a contract is deleted? What happens to the Staff member?**
  - No need for any special action → since **Staff is in the partial participation**

Representing 1:1 Relationship Sets - 1

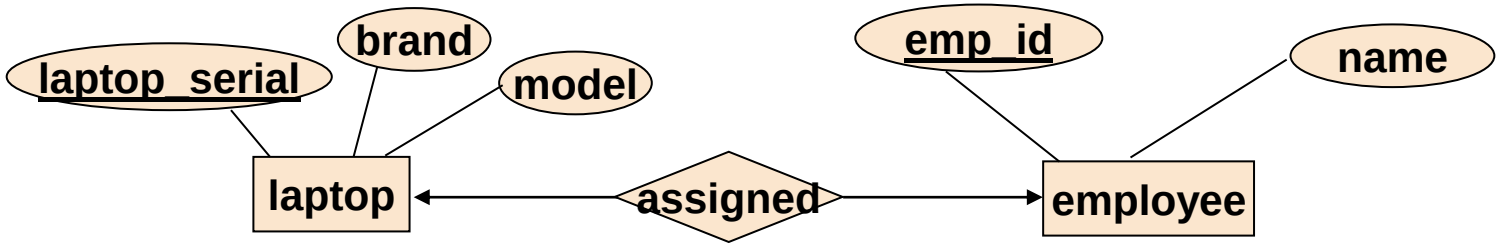
One-to-One; Partial participation on BOTH sides



- **Choice 1: (2 Tables)**
  - Two entities sets are kept in ***separate tables***, and
  - The *association* between them are *made through* ***a FK***
  - The PK of one entity set becomes the FK in the other *but not* vice versa.
    - The FK must be Allowed to be NULL (why? Partial participation!)
- **Choice 2: (3 Tables)**
  - **Create a relation** (table) for the relationship, and
  - associate the entities through the FKs in the newly created table
    - What should be the primary key of the third table?

Representing 1:1 Relationship Sets – 2

One-to-One; Partial participation on BOTH sides



Choice 1a:

employee = (emp_id, name, *laptop_serial*)

laptop = (laptop_serial, brand, model)

Referential Integrity on the foreign key

UNIQUE, but NULL must be allowed

ON UPDATE CASCADE, ON DELETE SET NULL

Choice 1b:

employee = (emp_id, name)

laptop = (laptop_serial, brand, model, *emp_id*)

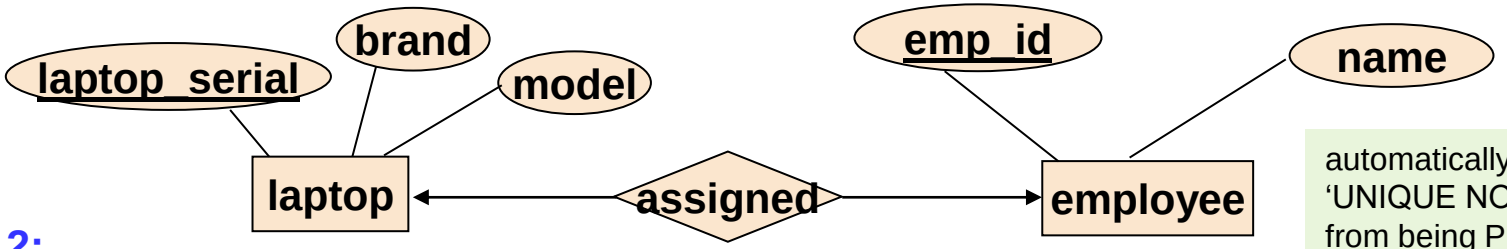
Referential Integrity on the foreign key

UNIQUE, but NULL must be allowed

ON UPDATE CASCADE, ON DELETE SET NULL

Representing 1:1 Relationship Sets – 3

One-to-One; Partial participation on BOTH sides



automatically implied 'UNIQUE NOT NULL' from being PK of the table.

Choice 2:

employee = (emp_id, name)

emp_laptop = (*emp_id*, *laptop_serial*)

laptop = (laptop_serial, brand, model)

Referential Integrity on each of the FKs

UNIQUE and NOT NULL

ON UPDATE CASCADE, ON DELETE CASCADE

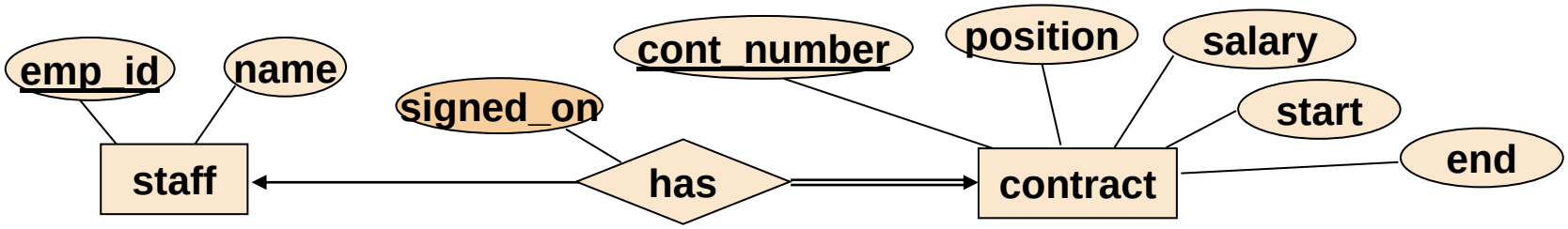
(Note: '*UNIQUE' implies max. cardinality = 1,*
'NOT NULL' implies min. cardinality = 1)

```
create table emp_laptop (
    emp_id              char(10),
    laptop_serial       char(15) unique not null
    primary key (emp_id),
    foreign key (emp_id) references
            employee(emp_id)
            on delete cascade
            on update cascade,
    foreign key (laptop_serial) references
            laptop(laptop_serial)
            on delete cascade
            on update cascade
);
```

Representing 1:1 Relationship Sets - 1
Descriptive Attributes

- Reposition descriptive attributes to **either** entity set
  - But place them with the **table having the FK**

- Example 1:

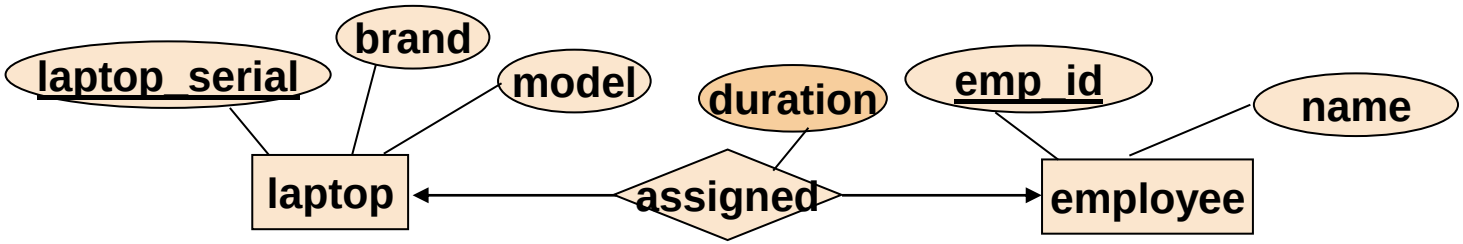


Staff = (emp_id, name)

Contract = (cont_number, position, salary, start, end, *emp_id* , signed_on)

Representing 1:1 Relationship Sets - 2
Descriptive Attributes

- Example 2



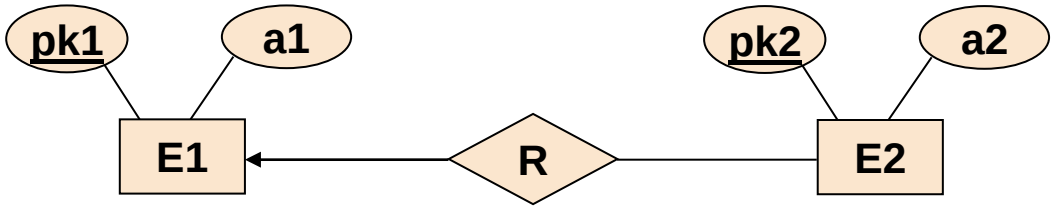
employee = (emp_id, name)

emp_laptop = (*emp_id*, *laptop_serial*, duration)

laptop = (laptop_serial, brand, model)

Representing 1:M Relationship Set

One-to-Many Relationship Sets

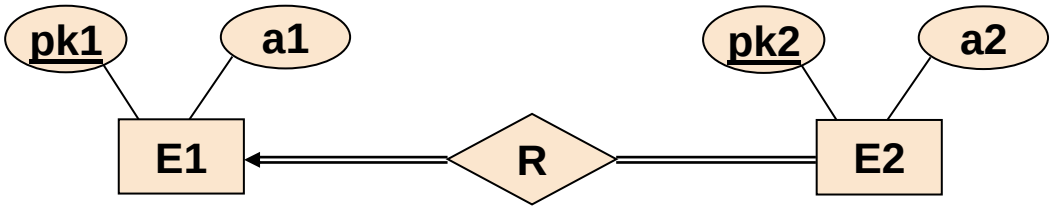


- An entity on the left can be associated with **many** entities on the right
- An entity on the right can be associated with **at most one** entity on the left

- The methods of representing **One-to-Many** Relationship sets equally apply for representing **Many-to-One**

Representing 1:M Relationship Set - 1

Total Participation on BOTH sides



- **The two entity sets are represented by *two separate tables*, and**
- **The *association* are made through a *FK***
  - **A FK is created on the <u>MANY side table</u>** *referencing the PK of the ONE side table*
  - Referential Integrity on the FK
    - **must NOT be NULL** → **enforces the total participation**

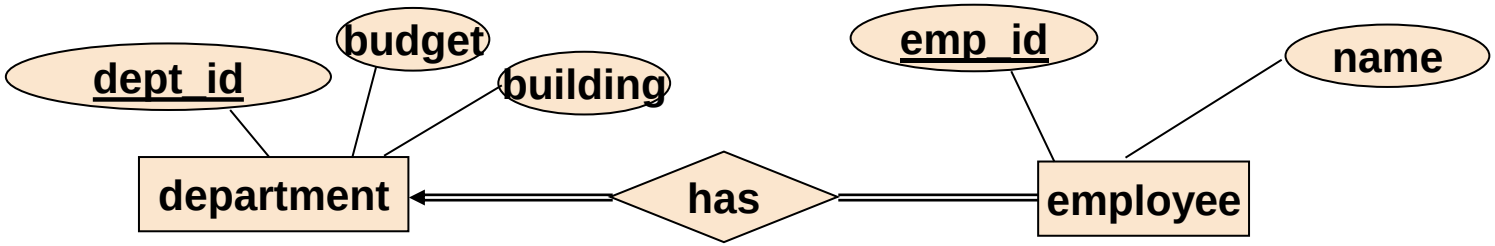
      → The minimum cardinality = 1
  - The maximum cardinality of 1 is enforced by the fact that PK2 is the **primary key** *(automatically implies 'UNIQUE')*;
    - Each E2 entity instance is associated with only one E1 instance

E1 = (<u>pk1</u>, a1)

E2 = (<u>pk2</u>, a2, *fk1*)

Representing 1:M Relationship Set - 2

Total Participation on BOTH sides



department = (dept_id, budget, building)

employee = (emp_id, name, *dept_id*)

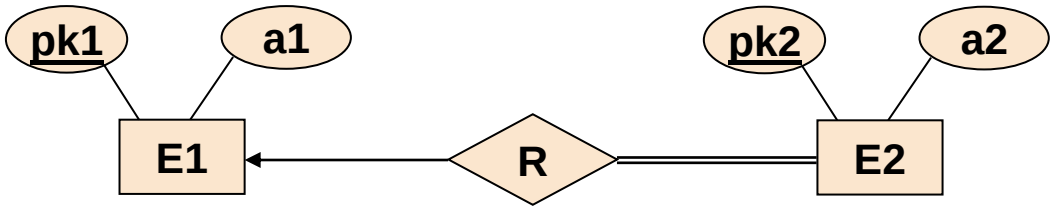
Referential Integrity on the FK

NOT NULL

ON UPDATE CASCADE, ON DELETE ???????

- An employee CANNOT be inserted without being associated with a department
  - Enforced by NOT NULL on the FK
- An employee CANNOT be associated with two or more departments
  - Enforced by emp_id being the PK
- Does not automatically ensure that a department must have one or more employees
  - Must be regulated through application logic

Representing 1:M Relationship Set - 1
Total on Many side and Partial on One side



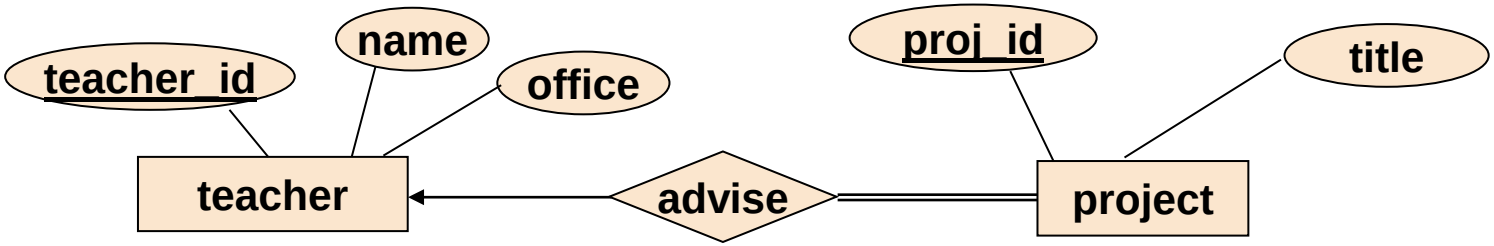
- **The two entity sets are represented by *two separate tables,* and**
- **The *association* are made through a *FK***
  - **A FK is created on the MANY side table** *referencing the PK of the ONE side table*
  - Referential Integrity on the FK
    - **must NOT be NULL** → **enforces the total participation**
  - The maximum cardinality of 1 is enforced by the fact that PK2 is the primary key;
    - Each E2 entity instance is associated with only one E1 instance

E1 = (pk1, a1)

E2 = (pk2, a2, *fk1*)

Representing 1:M Relationship Set – 2

Total on Many side and Partial on One side



teacher = (teacher_id, name, office)

project = (proj_id, title, *teacher_id*)

Referential Integrity on the FK

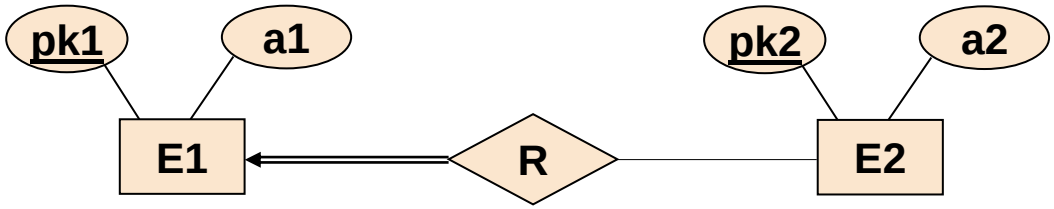
NOT NULL

ON UPDATE CASCADE, ON DELETE ???????

- A project CANNOT be inserted ***without*** being associated with a teacher
  - Enforced by **NOT NULL** on the FK
- A project CANNOT be associated with ***two or more*** teachers
  - Enforced by **proj_id being the PK**
- The one side is partial (Teacher may or may not look after any project)
  - There is **no need to put any other constraints**

Representing 1:M Relationship Set - 1

Partial on Many side and Total on One side (choice 1)



Choice 1 (2 Tables)

E1 = (pk1, a1)

E2 = (pk2, a2, *fk1*)

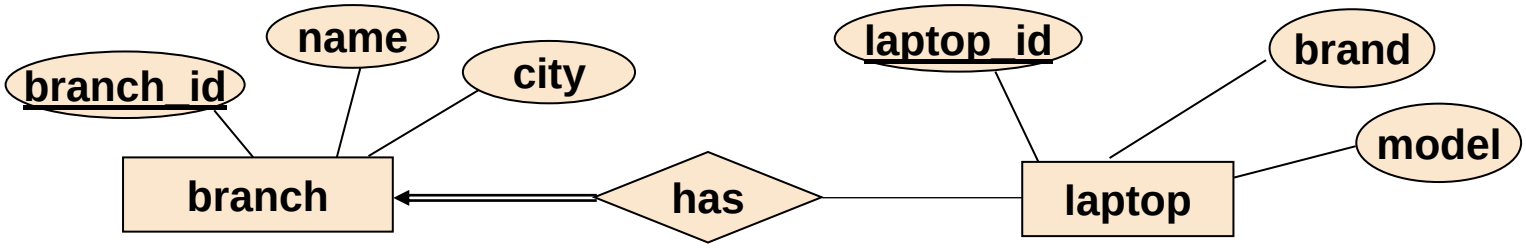
Not a good choice!!

Referential Integrity on the FK

- **NULL Allowed** – since **the many side** is **partial** participation; there may be entities in E2 which are not associated with any entities in E1
- **UNIQUE NOT Possible** – different E2s can be associated with the same E1
- maximum cardinality of 1 on the E1 is enforced by pk2 being the primary key of E2
- However, **the total participation of the E1 (one side) is not enforced automatically**
  - Must be regulated through application logic

Representing 1:M Relationship Set - 2

Partial on Many side and Total on One side (choice 1)



branch = (branch_id, name, city)

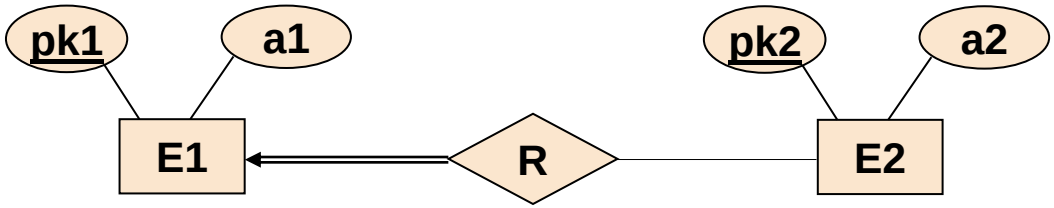
laptop = (laptop_id, brand, model, branch_id)

Not a good choice!!

FK
- Null allowed
- UNIQUE NOT possible

Representing 1:M Relationship Set - 3

Partial on Many side and Total on One side (choice 2)



- Note: the PK of R is the PK of the MANY side

- **Referential Integrity on fk2**
  - **UNIQUE, NOT NULL**
    - implied automatically since it's the primary key of R
- **Referential Integrity on fk1**
  - **NOT NULL**

Choice 2 (3 Tables)

E1 = (pk1, a1)

R = (*fk1*, *fk2*)

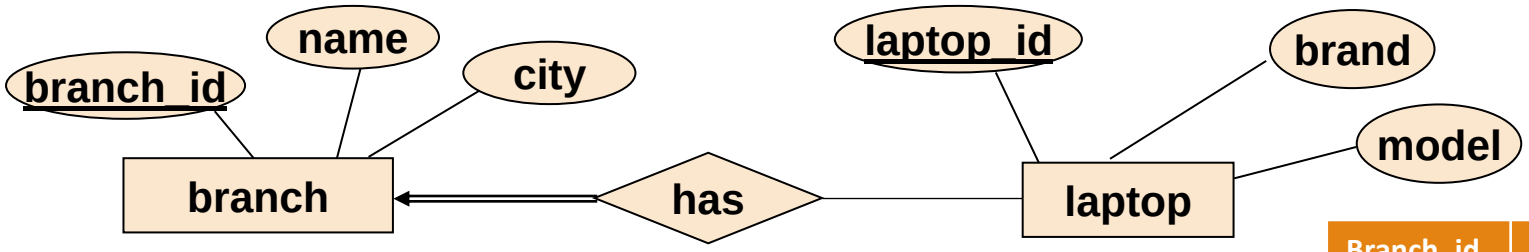
E2 = (pk2, a2)

R:

fk1	fk2
1	A
1	B
1	C

Representing 1:M Relationship Set - 3

Partial on Many side and Total on One side (choice 2)



Branch_id	Laptop_id
A	L1
A	L2
B	L3

- Branch MUST have one or more laptops
- Laptop MAY be owned by a branch, but not more than one branch

Choice 2

branch = (branch_id, name, city)

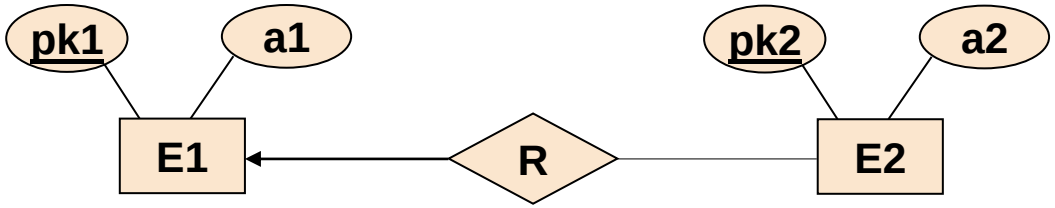
branchlaptop = (*branch_id*, *laptop_id*)

laptop = (laptop_id, brand, model)

```
create table branchlaptop (
    laptop_id       char(10)
    branch_id       char(15) not null
    primary key (laptop_id),
    foreign key (branch_id) references
            branch(branch_id),
    foreign key (laptop_id) references
            laptop(laptop_id)
);
```

Representing 1:M Relationship Set - 1

Partial Participation on BOTH sides (Choice 1)



Not a good choice!!

pk2	Fk1 (Pk1)
A	1
B	1
C	2
D	Null
E	Null
F	Null

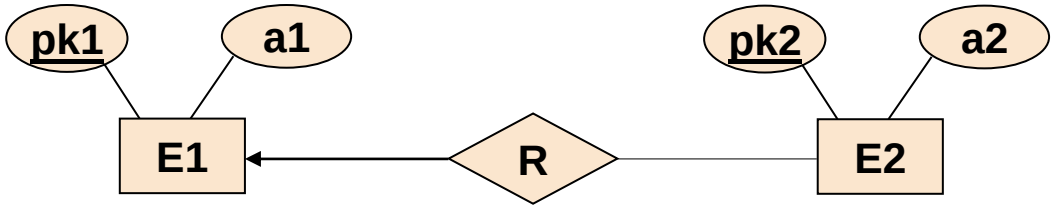
- Choice 1 (2 Tables)
  - Referential Integrity on the FK
    - **NULL Allowed** – since the many side is partial participation; there may be entities in E2 which are not associated with any entities in E1
    - **UNIQUE NOT Possible** – different E2s can be associated with the same E1
    - maximum cardinality of 1 on the E1 is enforced by pk2 being the primary key of E2

E1 = (pk1, a1)

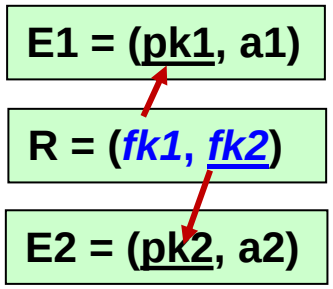
E2 = (pk2, a2, *fk1*)

Representing 1:M Relationship Set - 2

Partial Participation on BOTH sides (Choice 2)

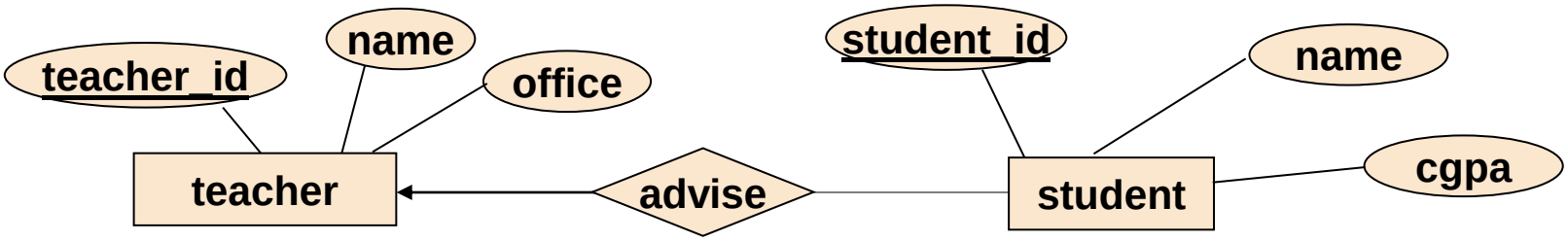


- Choice 2 (3 Tables)
  - Note: the PK of R is the PK of the MANY side
  - Referential Integrity on fk2
    - **UNIQUE, NOT NULL**
      - implied automatically since it's the primary key of R
  - Referential Integrity on fk1
    - **NOT NULL**



Representing 1:M Relationship Set - 3

Partial Participation on BOTH sides (Choice 2)



teacher = (teacher_id, name, office)

advising = (*teacher_id*, *student_id*)

student = (student_id, name, cgpa)

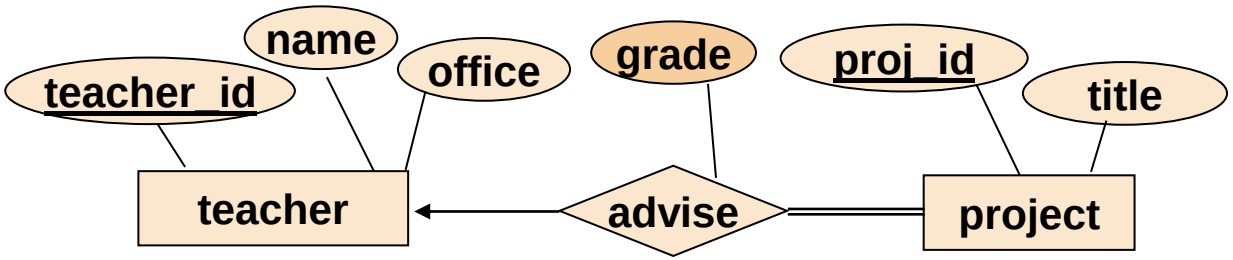
```
create table advising (
    student_id          char(7)
    teacher_id          char(6) not null
    primary key (student_id),
    foreign key (student_id) references
            student(student_id),
    foreign key (teacher_id) references
            teacher(teacher_id)
);
```

Teacher_id = {L1, L2, L3, L4}
Student_id = {1,2,3,4,5}

Teacher_id	Student_id
L1	1
L2	2
L2	3

Representing 1:M Relationship Set - 1
Descriptive Attributes

- Reposition descriptive attributes **to the MANY side**
  - place them with the table having the FK
- Example 1

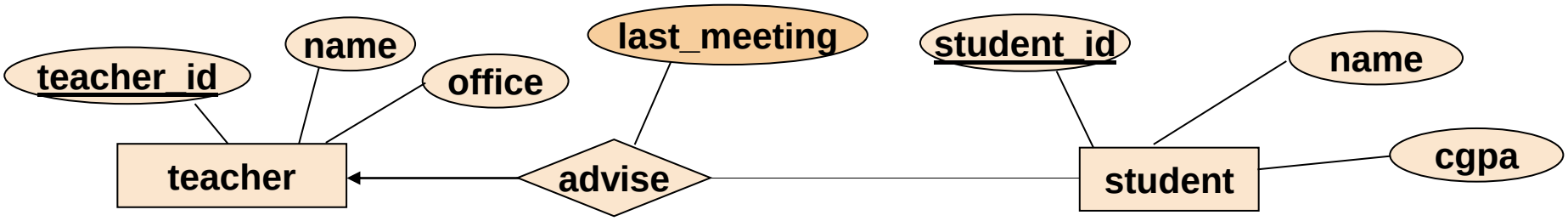


teacher = (teacher_id, name, office)

project = (proj_id, titile, *teacher_id*, grade)

Representing 1:M Relationship Set - 2
Descriptive Attributes

- Example 2

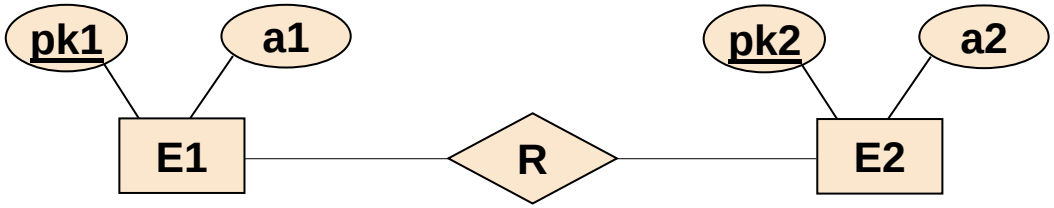


teacher = (teacher_id, name, office)

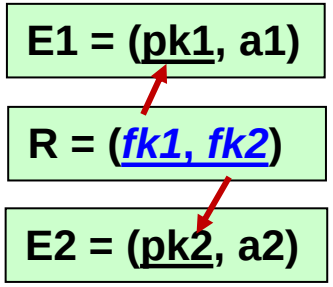
advising = (*teacher_id*, *student_id*, last_meeting)

student = (student_id, name, cgpa)

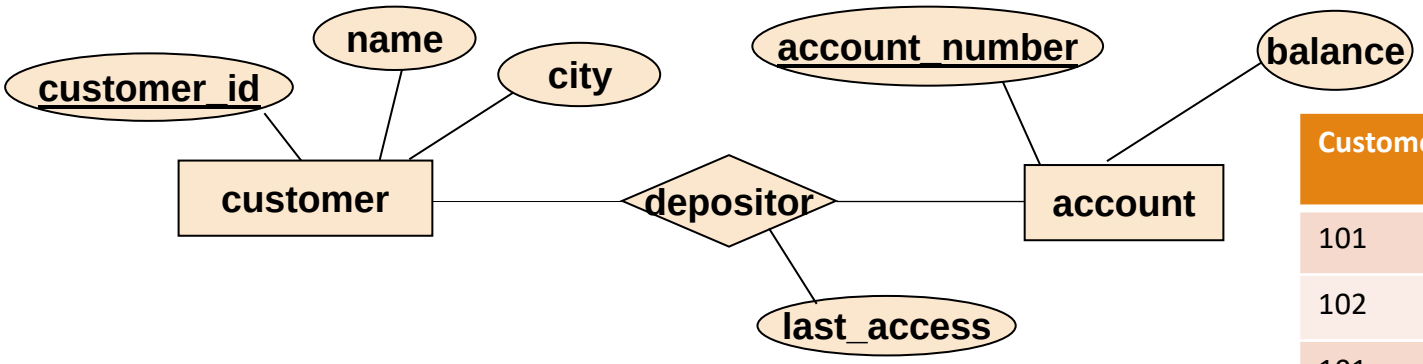
Representing M:M Relationship Sets - 1



- **Always create a separate table for the relationship**
  - The table will contain the PKs of the entity sets participating in the relationship ***plus*** any descriptive attributes of the relationship set
  - **The PK of the table will be the UNION of the PKs of the entity sets**
  - Each individual FK **CANNOT be unique** (Many-to-Many)



Representing M:M Relationship Sets - 2



Customer_id	Account_ number	Last_access
101	A_111	Yesterday
102	A_111	Yesterday
101	A_222	Today
102	A_222	Today

customer = (customer_id, name, city)

depositor = (*customer_id, account_number*, last_access)

account = (account_number, balance)

```
create table depositor (
    customer_id         char(7),
    account_number      char(6),
    last_access         date,
    primary key (customer_id, account_number),
    foreign key (customer_id) references
            customer(customer_id),
    foreign key (account_number) references
            account(account_number)
);
```

Relationship Representation and Enforcement of Constraints - 1

- **One-to-One**
  - Sometimes can be merged into a single table one when Total-Total
  - Two tables with a FK from one table to the other
    - Total-Total; Total-Partial; Partial-Total
  - Three tables with FK to the other two tables
    - Especially for Partial-Partial or when the relationship has attributes
- **One-to-Many (Many-to-One)**
  - Two tables with a FK from MANY side to the ONE side
    - Total-Total; Many (Total) - One (Partial)
  - Three tables with FKs to the other two tables
    - One (Total) – Many (Partial) ; Partial-Partial, and when relationship has attributes

Relationship Representation and Enforcement of Constraints - 2

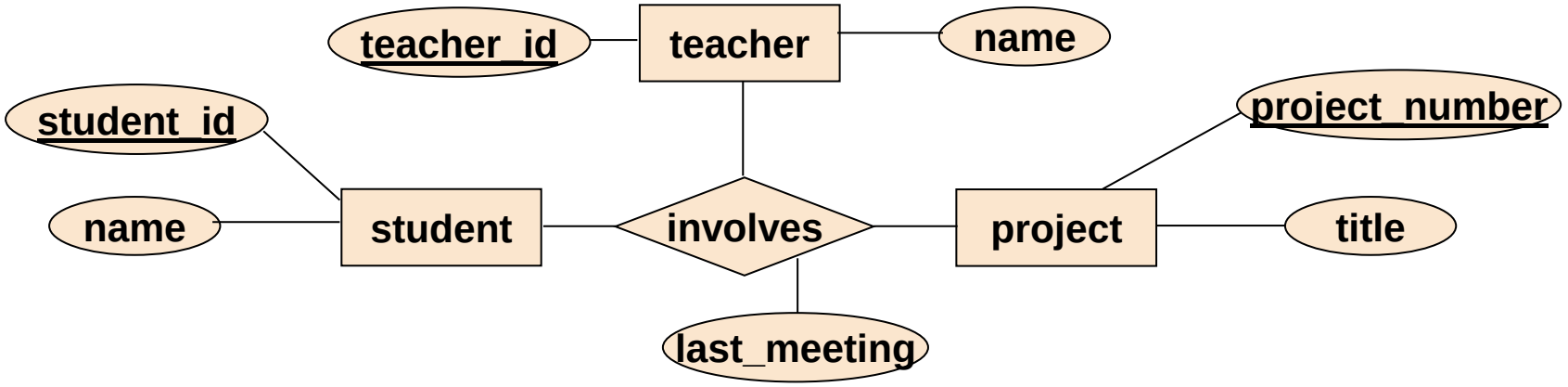
- **Many-to-Many**
  - Always three tables with FKs to the other two tables

- Realize that NOT all kinds of Cardinality Mapping and Participation Constraints can be automatically enforced
  - Total Participation (or minimum cardinality of 1) may not be enforceable in certain cases
  - Specific Cardinality mappings like (2..8) cannot be automatically enforced
    - Need to use other techniques like check, trigger, stored procedures, or application logic

Representing N-ary Relationships Sets - 1

- **Steps of transforming relationship sets having degree > 2**
  1. Create a table for the relationship
  2. Include the PKs of all the entity sets involved and any other descriptive attributes of the relationship set as the attributes of the table
  3. Create FKs referencing the PK of each entity set participating in the relationship
  4. The PK of the table will be the UNION of the PK of the entity sets involved in the relationship.
     - Except when there is a single arrow
       - The PK of the table will be the UNION of the PKs of entity sets not on the arrow side.
- But first, re-evaluate your ER design to see if you really need the N-ary relationship.

Representing N-ary Relationships Sets - 2



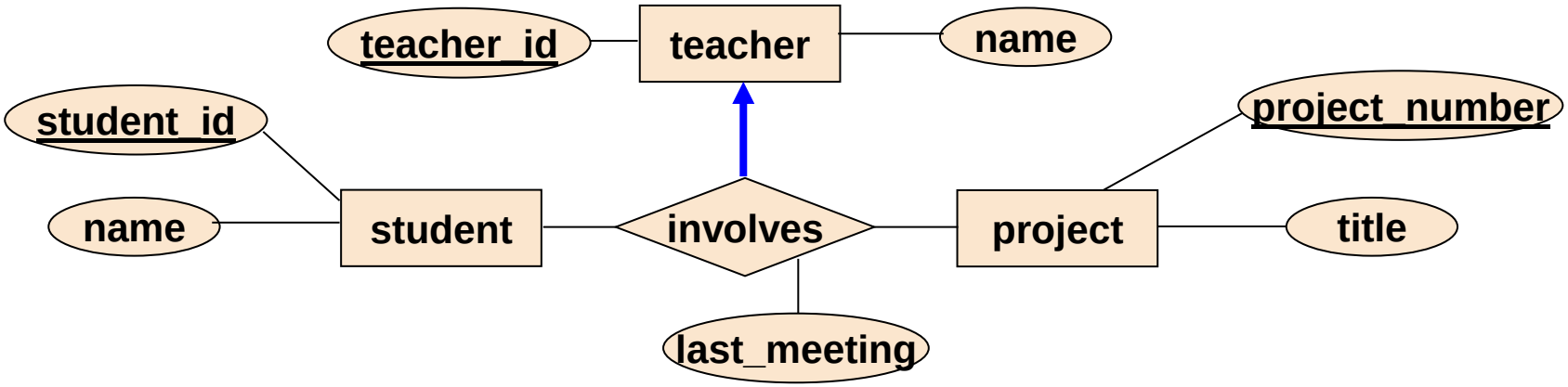
student = (student_id, name)

teacher = (teacher_id, name)

project = (project_number, title)

stp = (*student_id, teacher_id, project_number*, last_meeting)

Representing N-ary Relationships Sets - 3



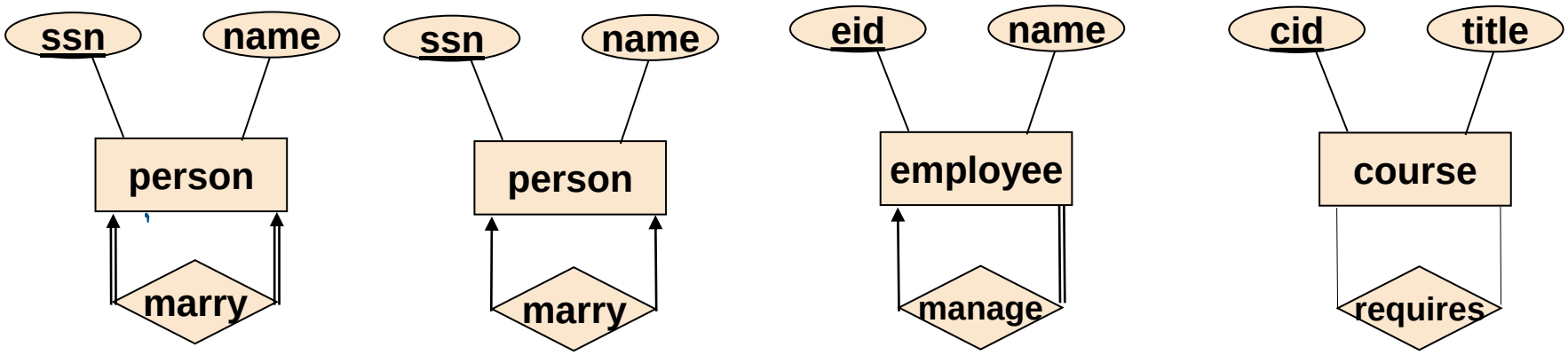
student = (student_id, name)

teacher = (teacher_id, name)

project = (project_number, title)

stp = (*student_id*, *teacher_id*, *project_number*, last_meeting)

Representing Unary Relationship Sets - 1

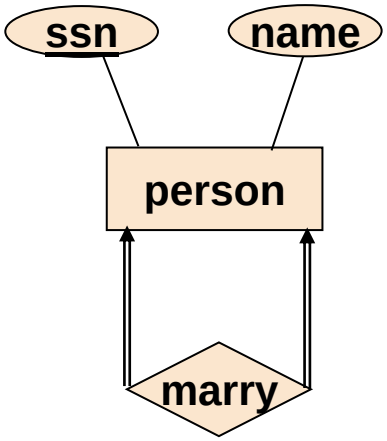


- **Same methods as those of transforming Binary Relationship Sets**
  - Unary Relationship Sets = binary relationship set, *where*
    - *an entity set participates in the same relationship set twice, each with different roles.*

Transforming Binary Relationship Sets

1. **Determine the Cardinality Mappings**
   - One-To-One, One-to-Many, and Many-to-One
2. **Determine the Participation Constraints**
   - Total vs. Partial (Or Minimum Cardinality)
3. **Choose the methods for the required choice**
   - Single table with a FK referring back to its own PK
   - Extra table for the relationship having FK to the entity table

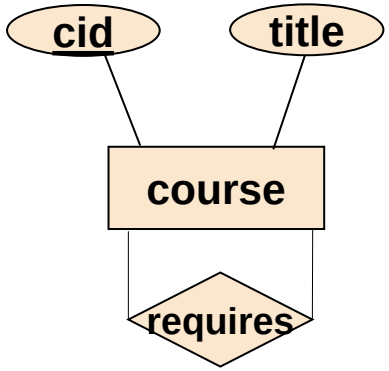
Representing Unary Relationship Sets - 2



person = (ssn, name, *spouse*)

```
create table person (
    ssn                     char(15),
    name                    char(6),
    spouse   char(15) unique not null,
    primary key (ssn),
    foreign key (spouse)  references person(ssn)
);
```

Representing Unary Relationship Sets - 3



course = (cid, title)

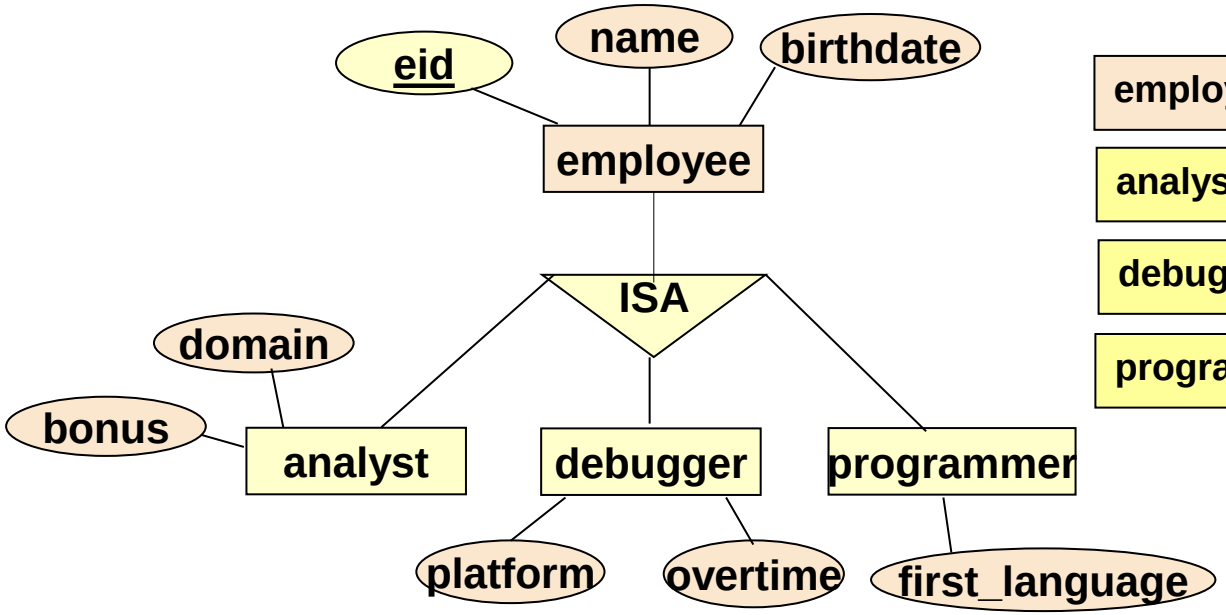
prerequisite = (*cid, pre*)

```
create table course (
        cid                     char(6),
        title                   char(50),
        primary key (cid)
);
create table prerequisite (
      cid                       char(6),
      pre                       char(6),
      primary key (cid, pre),
      foreign key (cid) references course(cid),
      foreign key (pre) references course(cid)
);
```

Representing Inheritance Hierarchy – 1

Strategy 1 (The most close to the EER)

- **The PK of the super-class is mapped into each subclass and becomes the PK of the sub-classes.**
  - The PK of each sub-class is **also made a FK**



employee = (e_id, name, birthdate)

analyst = (*e_id*, bonus, domain)

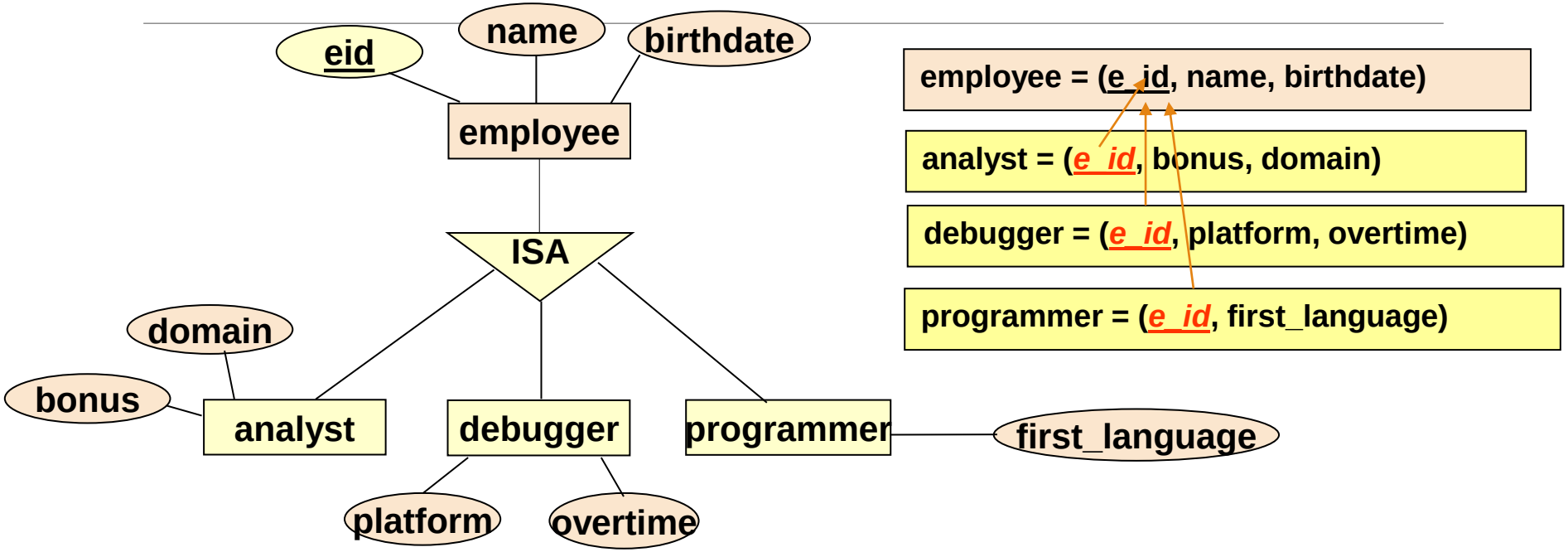
debugger = (*e_id*, platform, overtime)

programmer = (*e_id*, first_language)

Note: the enforcement of constraints (overlapping, disjoint, total, partial) need to be done through application logic

Representing Inheritance Hierarchy – 2

Strategy 1 (The most close to the EER)

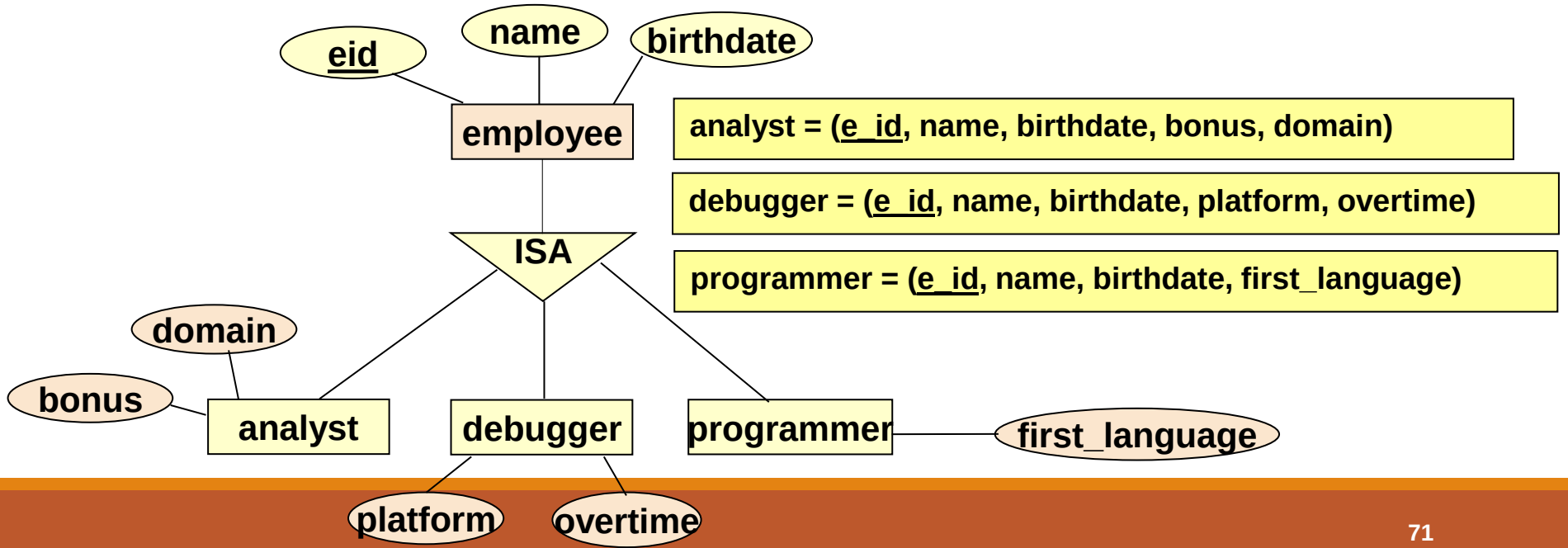


- Referential Integrity Constraints on the FKs
  - **UNIQUE, NOT NULL**
- Enforcement of Total, Partial, Disjoint, Overlapping rules need to be enforced through check, triggers, or stored procedures.
- What should be actions on delete and on update?

Representing Inheritance Hierarchy – 3

Strategy 2 (Disregard the hierarchy and represent as separate entity sets)

- **OK when the *hierarchy is disjoint and total*** and **when the superclass is not participating any other relationship** with other entity sets..
  - If overlapping, then result in data redundancy → Data Inconsistency
  - **Not possible to define a single FK constraints on the super-class** if the super-class is participating in other relationships with other entity sets.

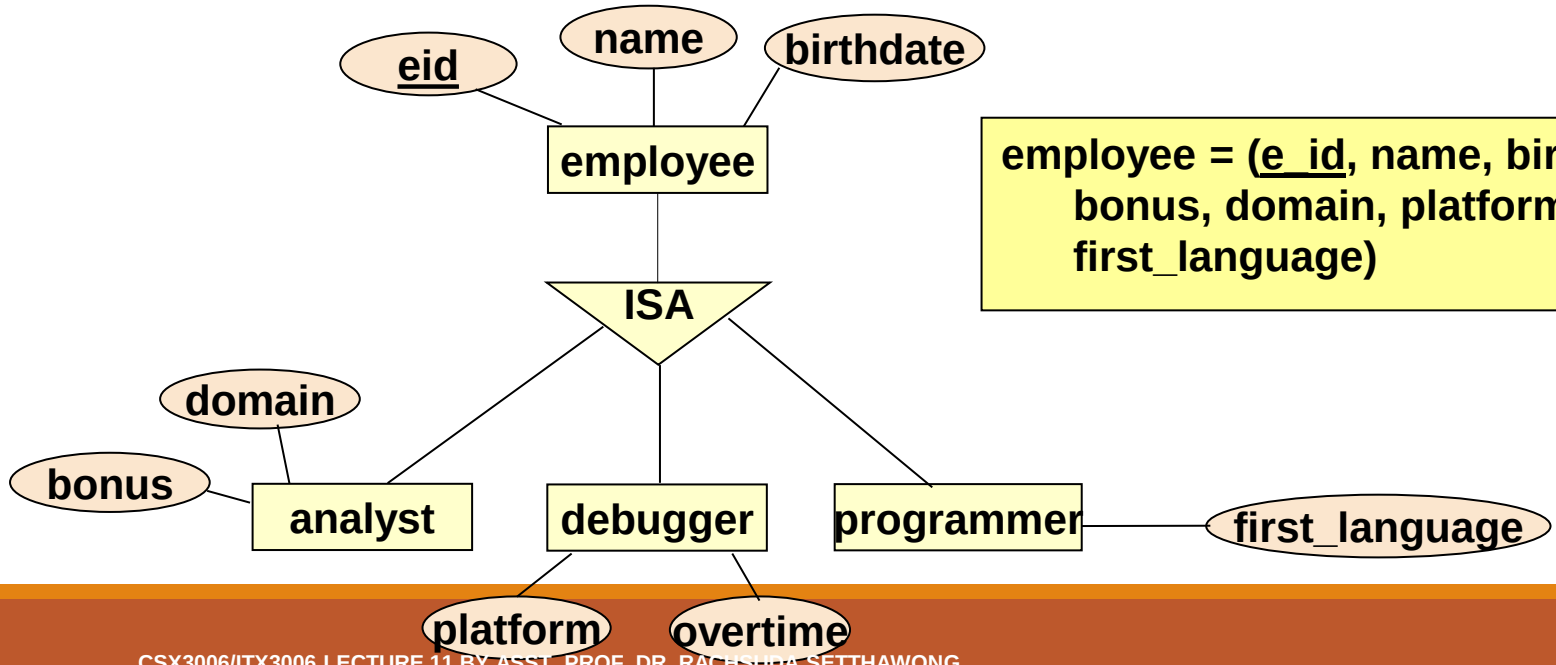


Representing Inheritance Hierarchy – 4

Strategy 3 (Disregard the hierarchy and represent as a single entity set)

- **Not really a good choice** as there will be many null values
- Furthermore, **Cannot handle situations when the sub-classes are engaged in specific relationships with other entity sets.**
- Only good point is that it avoids joins to get additional information about each subclass

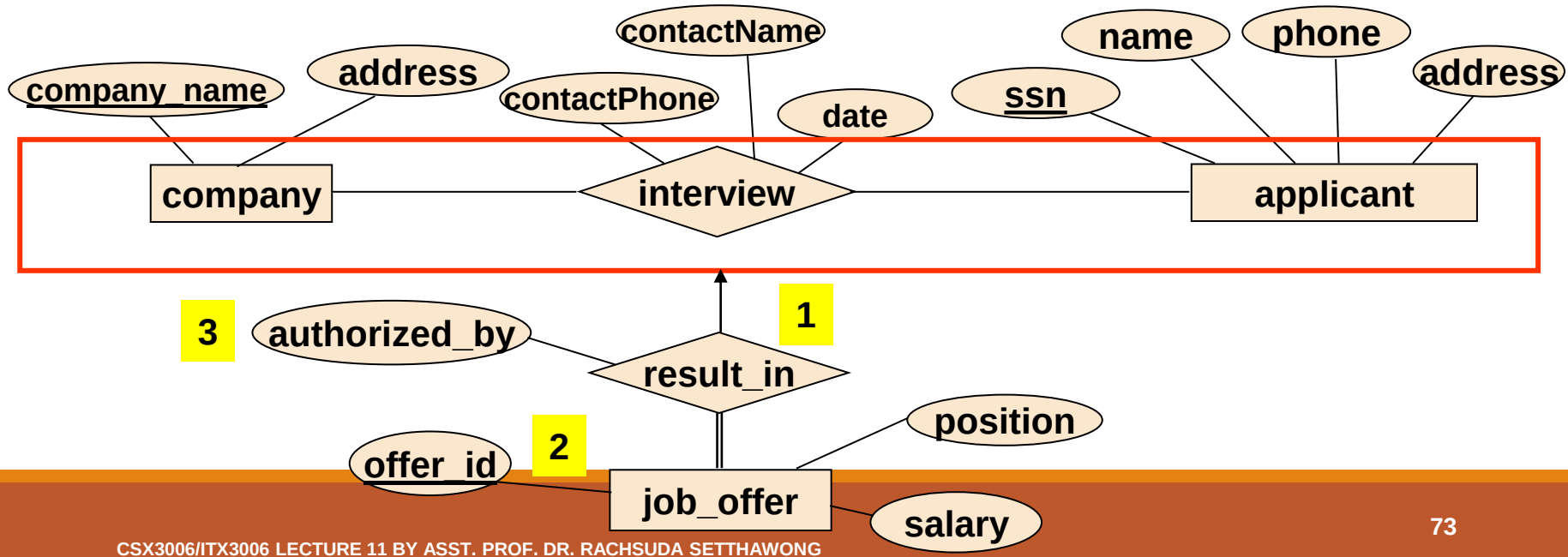
- **Do NOT use this strategy unless there is huge penalty in the performance caused by join operations required by strategy #1**



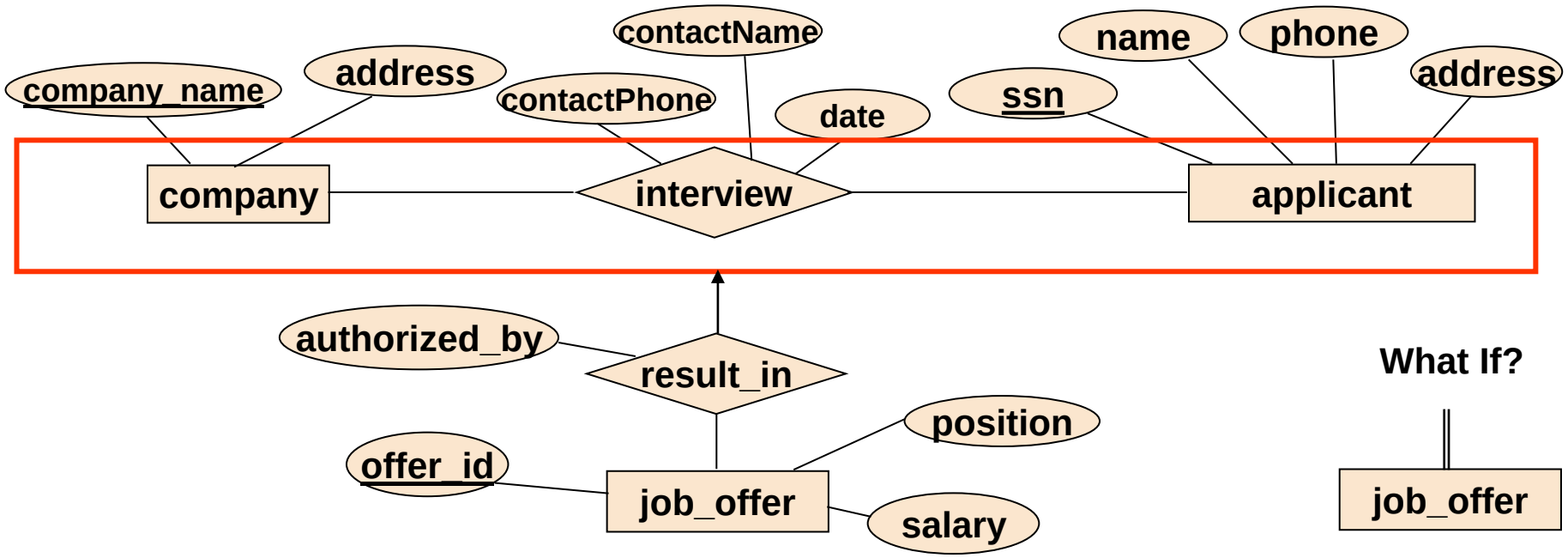
employee = (e_id, name, birthdate, bonus, domain, platform, overtime, first_language)

Representing Aggregation

- Create a table containing
  1. **PK of the aggregated relationship,**
  2. **the PK of the associated entity set**
  3. **any descriptive attributes**
- Note: the **whole aggregation** is *represented as a table for the relationship* linking the 'aggregated entity set' to the other entity set (e.g. result_in)



Representing Aggregation - 2



company = (company_name, address)

applicant = (ssn, name, phone, address)

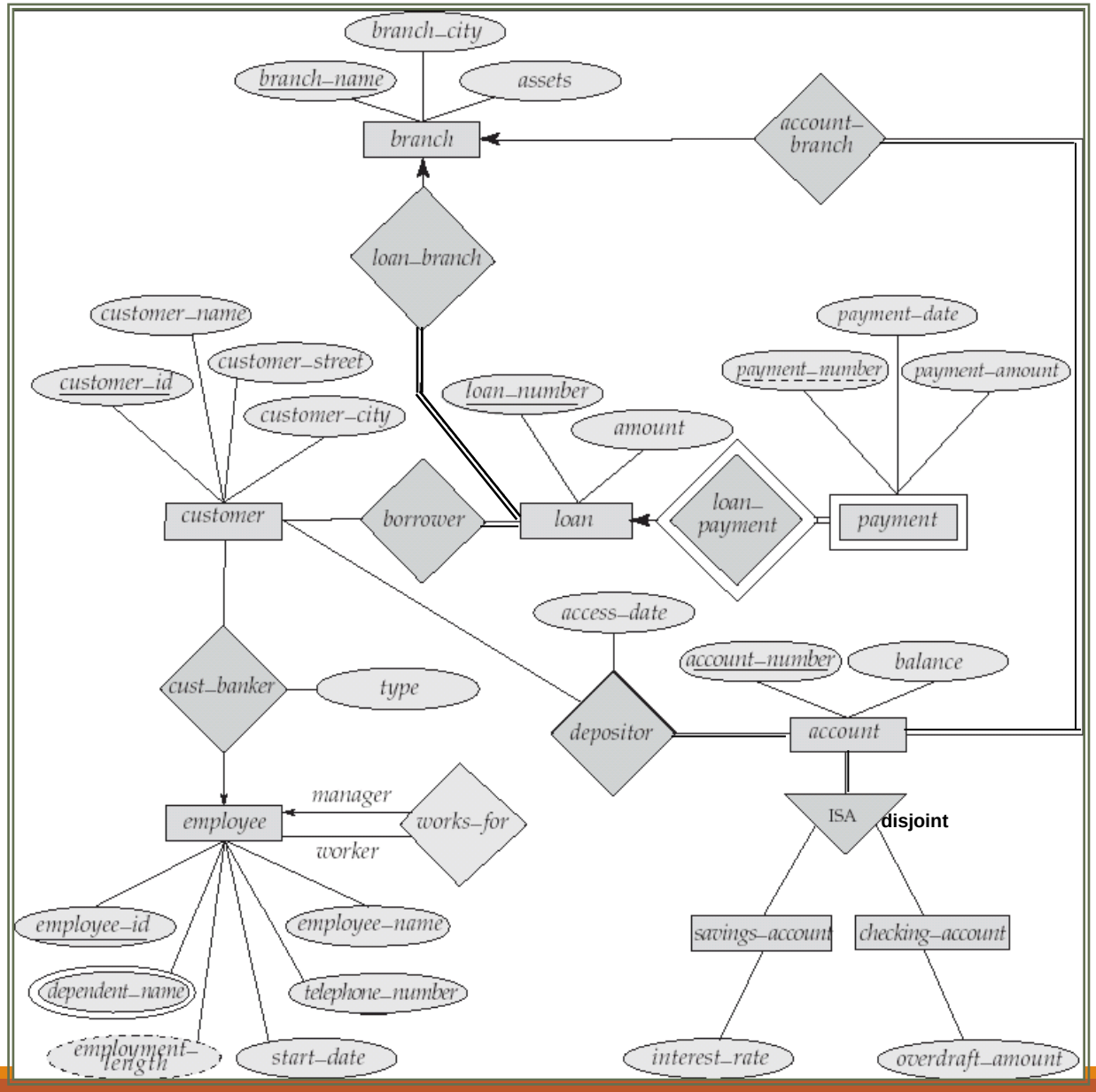
interview = (*company_name, ssn*, contactName, contactPhone, date)

result = (*company_name, ssn*, *offer_id*, authorized_by)

job_offer = (offer_id, position, salary)

Offer_id is the PK of result table since it's a Many-to-One relationship from job_offer to interview

Self-Exercise #1



- Transform into a set of relational schema
- Define SQL DDL to create the tables
- Carefully consider the integrity constraints specification in DDL.

Self-Exercise #2

- Modify the ER diagram shown in the previous slide to support the banking model where the bank keeps track of deposits and withdrawals from savings and checking accounts.
  - Assume the bank does not care who made the deposits or withdrawals.
  - Assume the bank does not care about methods of deposits or withdrawals, e.g.) Cash Withdrawal, withdrawal from ATM machine 178, online transfer
  - It should be very similar how loan payments are tracked