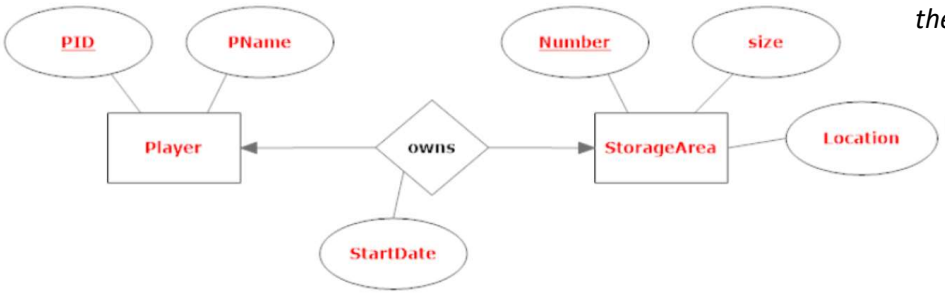
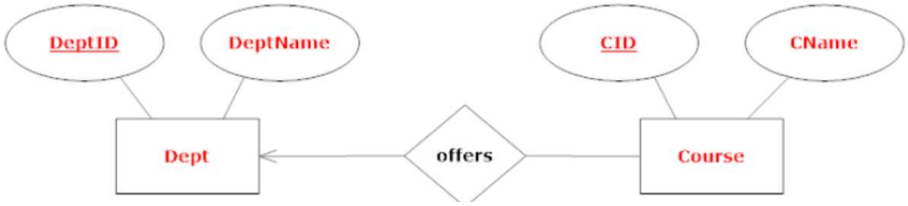
ERDs: Basic Rules



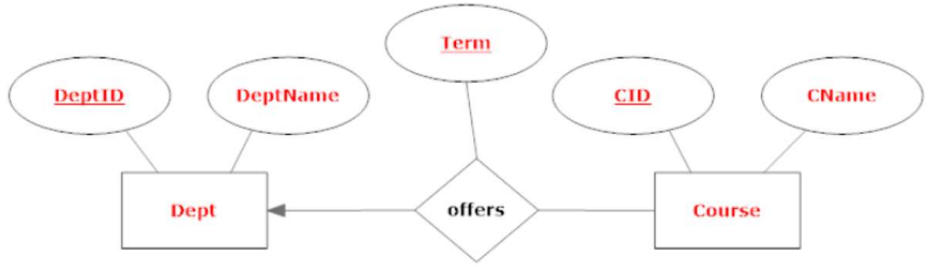


1:1 or 1:M (with no relationship keys) relationships become *part of one of the entity tables*

a. If 1:1, PK of on side is copied to the other as a FK
b. If 1:M, PK of the "one" side is copied to the "many" side as a FK
c. Any relationship attributes go on the side with the FK

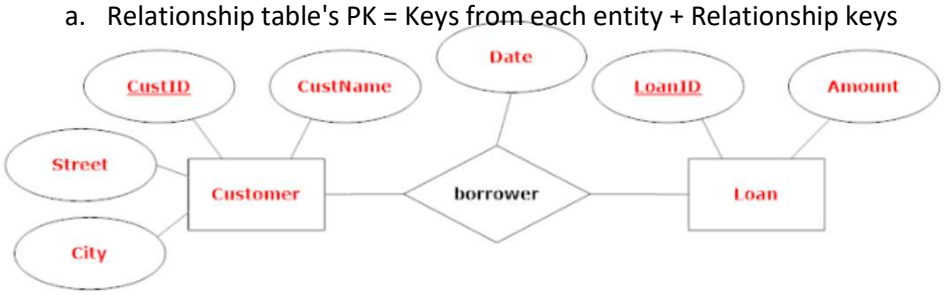
1:1 or 1:M (with relationship keys) relationships map to a *separate table*

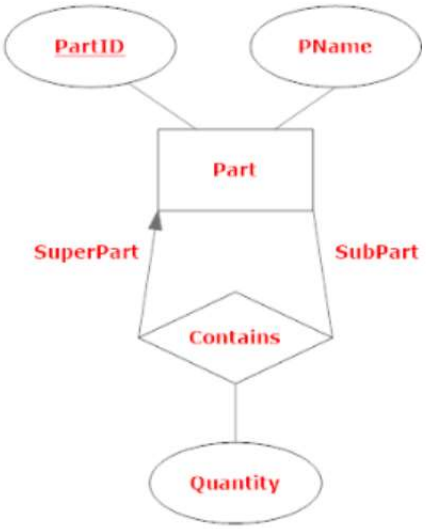
a. Relationship maps to a table with it's PK and Attributes, plus the PK from the "many" side
    i. Does NOT get the PK from the "one" side
b. "many" side has a FK that references the "one" side's PK
c. In a recursive relationship, the entity contains a FK to itself



M:M or Multi-way relationships map to a *separate table*

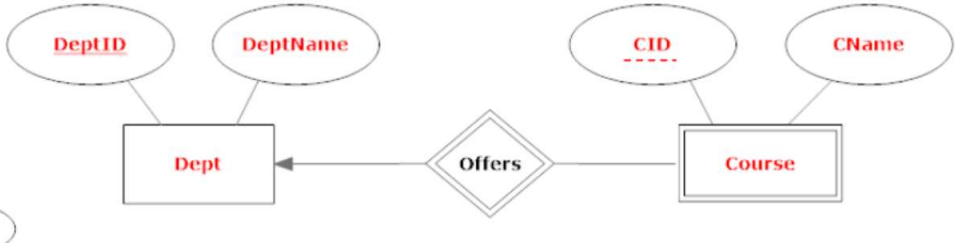
a. Relationship table's PK = Keys from each entity + Relationship keys

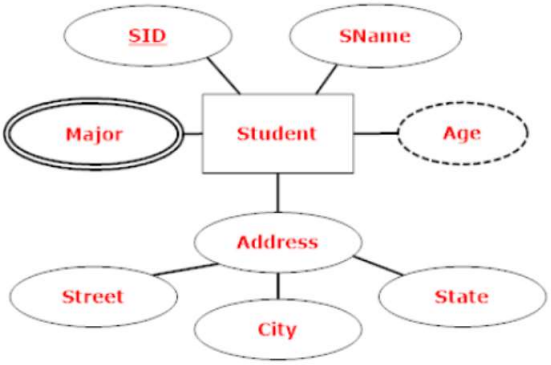




Weak Entity Sets become a *seperate table*

a. Contains all it's attributes and FKs to the PKs of identifying entity sets
b. PK = ID'ing Keys from the identifing Entity sets + Discriminator
c. The total, 1:M supporting relationship is not mapped





Composite Attributes only *use the 2nd level attributes* as columns
Derrived Attributes map *as-is* (and enforce via triggers)
Multi-Valued Attributes become a *separate table*
PK = Attribute itself + PK of main entity set

ERDs: IsA

A: Relation for each Entity Set

- Redundancy is inherint to the design (a student is in both the Students and People tables)
- Need multiple tables to complete a record for a student or employee

A1: Partial/Overlapping

Person (SSN, Name, DOB)
Employee (*SSN*, Department, Salary)
FK Employee(SSN) Ref Person(SSN)
Student (*SSN*, GPA, StartDate)
FK Students(SSN) Ref Person(SSN)

A2: Disjoint/Total

Person (SSN, Role, Name, DOB)
Unique (SSN, Role)
Role in {'Student', 'Employee'}
Employee (*SSN*, *Role*, Deparment, Salary)
Employee.Role = 'Employee'
Student (*SSN*, *Role*, GPA, StartDate
Role = 'Student'

B: One Big Table

- Full of nulls
- Less joins

B1: Partial/Overlapping

Person (SSN, Name, DOB, Department, Salary, GPA, StartDate)

B2: Disjoint/Total

- Need triggers to enforce attribute values based on role

Person (SSN, Name, DOB, Role, Department, Salary, GPA, StartDate)
Unique (SSN, Role)
Role in {'Student', 'Employee'}

C: Relations only for Specialization

- Cannot be used for Partial, but good for total
- If overlapping, need to update both tables on updates
- If disjoint, need triggers to ensure not in both tables
- Best for total disjoint

Employee (SSN, Name, DOB, Department, Salary)
Student (SSN, Name, DOB, GPA, StartDate

D: Relation for Every Combination

- Could get out of hand with lots of overlapping specializations
- Needs triggers to ensure a record is only in one table
- Good for overlapping relationships
- If Partial, need an additional table:
  Person (SSN, Name, DOB)
- Probably simpler to use A2

Employee (SSN, Name, DOB, Department, Salary)
Student (SSN, Name, DOB, GPA, StartDate)
StudentEmployee (SSN, Name, DOB, Department, Salary, GPA, StartDate)

```
create table Person (
        PID number(3),
        Role varchar2(10),
        name varchar2(30),
        DoB date,
        constraint Person_pk primary key (PID),
        constraint Person_un unique (PID, Role),
        constraint PersonRoleVal check (Role in ('Student', 'Employee'))
);

create table Student (
        PID number(3),
        Role varchar2(30) default 'Student' not null,
        GPA number(2,1),
        constraint Student_pk primary key (PID),
        constraint StudentRoleVal check (Role in ('Student')),
        constraint Student_fk foreign key (PID, Role) references Person (PID, Role)
);

create table Employee (
        PID number(3),
        Role varchar2(30) default 'Employee' not null,
        Salary number(6),
        constraint Employee_pk primary key (PID),
        constraint EmployeeRoleVal check (Role in ('Employee')),
        constraint Employee_fk foreign key (PID, Role) references Person (PID, Role)
);
```