E-R Model ⟹ The Entity-Relationship Model.

ER diagram

 ⟹ Entity

 ⟹ attribute

 ⟹ primary ~~primary~~ key

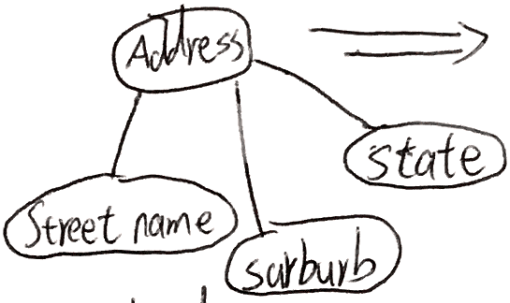
 ⟹ derived attribute.

eg: if we know birthday attribute.

The age attribute is derivod attribute

Type of Null Values

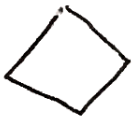
- Not Applicable.
- Unknown
- Missing

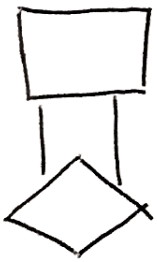


⟹ Composite Attribute.

 ⟹ Multivalued

eg: Degree = {BSc, MInfTech}

 ⟹ relationship

 ⟹ recursive relationship.

it means that an entity can be in relationship with itself.

A relationship can be between two entities, this is a binary relationship.

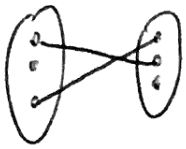
A relationship can be between three entities, this is a Ternary Relationship

same as N-ary Relationship

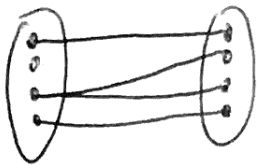
Binary Relationship

Ternary Relationship

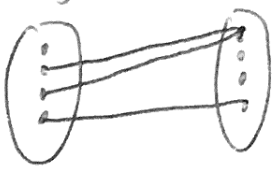
- One to One



One to Many



Many to One



key Constraints.

if we want to insert a row of Movies.

we should be insert a row of MovieStar first.

participation Constraints ==

eg: Every movie must have a director

Each account is created and maintained by a single branch

Each account is assigned to one or more customers

A loan is always assigned to a single customer

weak ~~Enti~~ Entities 

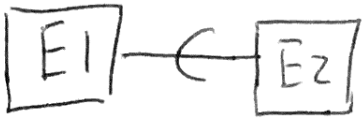
weak entities have a partial key 

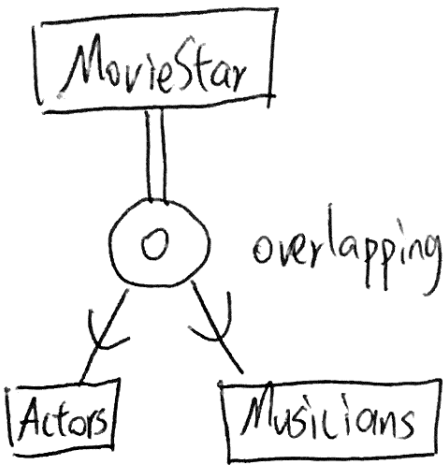
Owner entities and weak entities must participate in a One to One or One to many.

Weak entity and their identifying relationship sets are shown with double lines

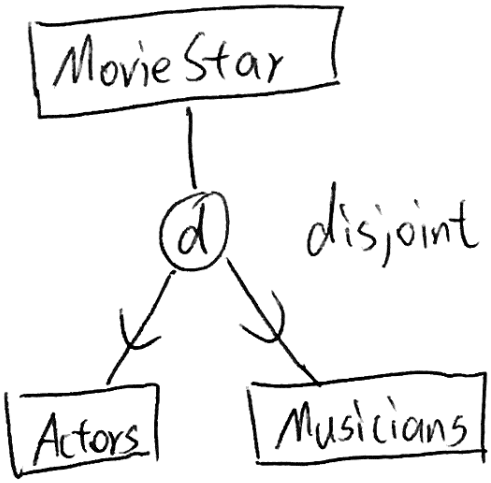
- sub classes (E1) are specializations of superclass (E2)

Superclass is generalization of subclasses (E1)





mean: Every movie star has to be either an actor or a musician.
A movie star can be both an actor and musician.



mean: A movie Stan ~~can't~~ cannot be both an ~~at~~ actor and musican

---

The number of attributes in a relation R is called the degree of R

Each Tuple t is an ordered list of n values:

eg: $t = \langle v_1, v_2, \cdots, v_n \rangle$ t is called an n-tuple

eg: R

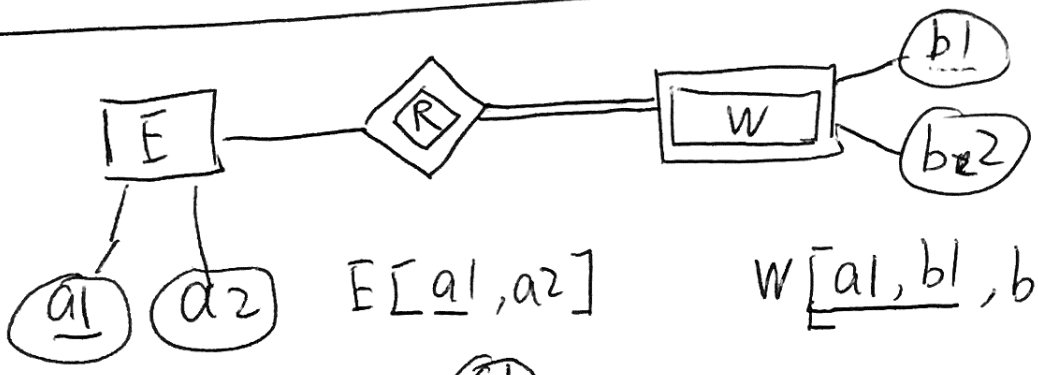
Relation Instance

A	B	C
0	1	2
3	4	5
6	7	8
9	10	11

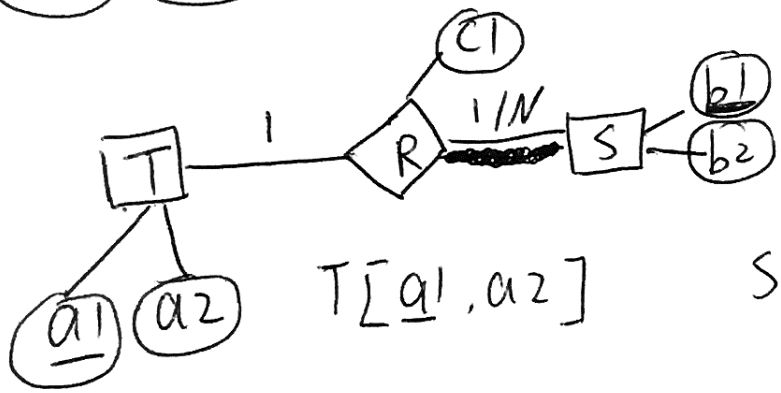
R has four tuple.
(6,7,8) is a tuple of R
The schema of R is R(A, B, C)
degree = 3

Integrity Constraint Types

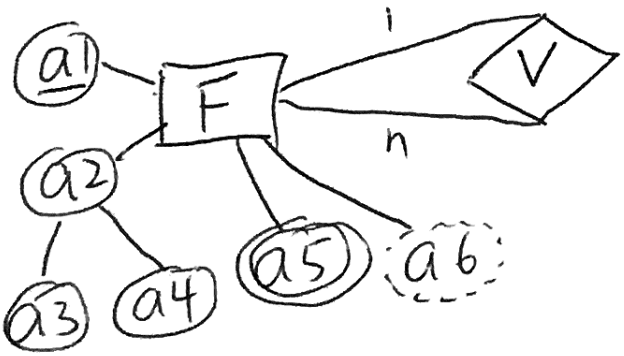
- Domain constraints : Gender only has {Male, Female}
- Key constraints : unique ~~and not null~~, Key is a minimal Superkey
- Entity constraints : ~~unique~~ No primary key can be null
- Referential integrity constraints : All foreign keys reference existing entities
- User-defined constraints



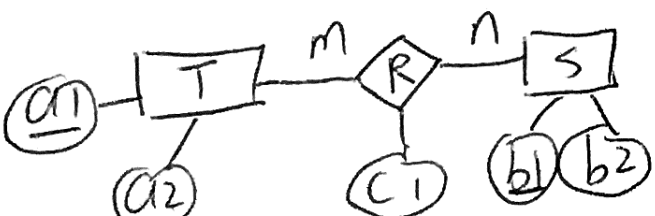
E[a1, a2] W[a1, b1, b2]



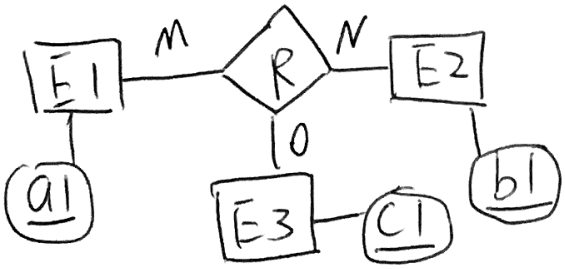
T[a1, a2] S[b1, b2, a1, C1]



F[a1, a3, a4, V_a1]
F2[a1, a5]



T[a1, a2]
S[b1, b2]
R[a1, b1, C1]

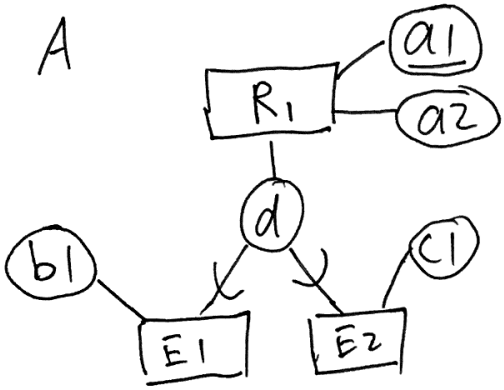


E1[a1]
E2[b1]
E3[c1]

R[a1, b1, c1]

---

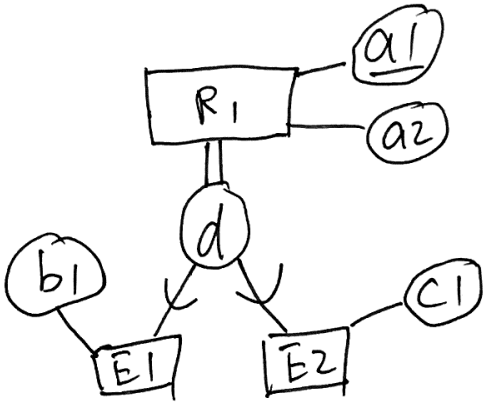
option A



R1(a1, a2)
E1(a1 b1)
E2(a1 C1)

option B

E1(a1, a2, b1)
E2(a1, a2, C1)

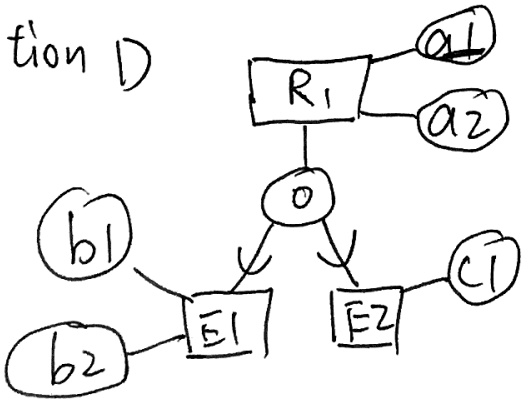


option C

a single relational table for all subclasses and the superclass

R1(a1, a2, b1, C1, T)

option D



R1(a1, a2, b1, b2, C1, T)

This is not a good design ~~right~~, because we get:

modification anomaly, deletion anomaly and insertion anomaly.

if $t1.X = t2.X \Rightarrow t1.Y = t2.Y$. which means given two tuples in R. if the X values agree. then the Y values must also agree: $X \rightarrow Y$ (A functional dependency)

~~Given a table with a million rows can you tell if a~~

Given a table with a million rows we cannot tell if a functional dependency exists from data.

Give atable we can check if a functional dependency doesn't hold from data.

Functional dependencies come from the universal discourse excatly

---

A superkey for a relation uniquely identifies the relation, but does not have to be minimal

key is a subset of superkey

A key is a minimal set of attributes that uniquely identity a relation.

$B \rightarrow C$: Attribute B determines C

explicit functional dependency: S_ID → Name.

implicit functional dependency: {{S_ID, Gender}} → Name

- Reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$
- augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$
- Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$ (implicit FD)
- Union if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Decomposition if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Closure of F: the set of all FDs implied by F.

- ~~Key~~ Superkey: A set of attributes such that no two tuples have the same values for these attributes
- Key: A minimal Superkey, called a Candidate Key if more than one:
  - Primary key: A selected candidate key
  - Secandary key: Remaining candidate keys.

- Prime Attribute : An attribute that is a member of any candidate key.
- Non-prime attribute : An attribute that is not a member of any candidate key

---

First Normal Form (1NF)

Domains of attributes include only atomic values ~~and the~~

Second Normal Form (2NF)

Every FD $X \rightarrow Y$, where X is a minimal key and Y is a non-prime attribute, then no proper subset of X determines Y.

Every FD $X \rightarrow Y$, X is a part of the key, and Y is a non-prime attribute.

eg: S-L-C (Sno, Sdept, Sloc, Cno, Grade) so it's not in 2NF

student – location – course

{Sno, Cno} → Grade

Sno → Sdept.

Sdept → Sloc

Third Normal Form (3NF)
- A relation R is in 3NF if for all non-trivial dependencies in R:
if X → b. then X is a super key for R or b is a prime attribute of R

S-L-C ∉ 2NF ⇒ { S-C {Sno, Cno, Grade}, S-L {Sno, Sdept, Sloc} } ∈ 2NF

S-L ∈ 3NF ⇒ { S-L {Sno, Cno, Grade}, S-D {Sno, Sdept}, D-L {Sdept, Sloc} } ∈ 3NF

S-L ∉ 3NF

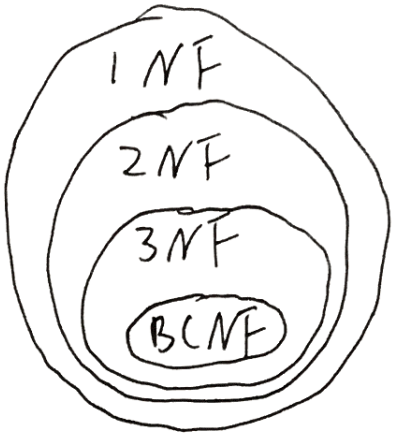
BCNF

A relation R is in BCNF: if $X \to b$, then X is a superkey for R.

eg: R(A, B, C, D, E)

$E \to C$

$E \to D$



1. Add implicit FDs to your list of FDs.

$\begin{cases} E \to C \\ E \to D \end{cases}$

2. Pick any $f \in FD$ that violates BCNF of the form $X \to b$

$E \to C \Rightarrow$ E is not a superkey for R.

3. Decompose R into two relations: $R_1(All - b)$ & $R_2(X \cup b)$

$R_1(A, B, D, E)$ $R_2(C, E)$

4. Recurse on R_1 and R_2 using FD.

① $\{E \to D$ in R_1 $E \to C$ in R_2

② $E \to D \Rightarrow$ E is not a super key for R_1

③ $R_1(A, B, E)$ $R_2(C, E)$, $R_3(D, E)$.

$\Rightarrow R_1(A, B, E)$ $R_2(D, C, E)$

Data Definition Language (DDL) { create, alter, drop

create table <table name>
(<column name> <column type> [<attribute constraint>]{,
<column name> <column type> [<attribute constraint>]},
[<table constraint>{,<table constraint>}]);

column type { integer, char(20)

constraint { primary key (<column name>)
foreign key (<column name>)
references <table>

<foreign key> —reference→ <primary key>

create table B (....
foreign key .... references A <table> } => if the row in A is deleted, then the row in B is automatically deleted.
on delete cascade);

on delete/update set null / set default / cascade.

---

alter table <table name> add <column name> <column type>;

alter table <table name> drop <column name> [cascade];

alter table <table name> alter <column name> <column-options>;

alter table <table name> add <constraint name> <constraint-options>;

alter table <table name> drop <constraint name> [cascade];

drop table [if exists] <table name> [restrict | ~~casda~~ cascade];

Data Manipulation Language (DML) { select, insert, update, delete

select [distinct] (<attribute list>/*) from <table list> [where <condition>];

eg: Find events that have occurred before 1943.

Query: select * from events where date < 19430000;
└ type: 1941-05-25

eg: Find the title of all of the movies that contain "sin"

Query: select * from movies where title like "%sin%";

Like is used for stand for string matching:

- '%' stands for 0 or more arbitrary characters.
- '_' stands for any one character.

- like eg: ... where adress like '% st Lucia%';
  ... where strDate like '__/05/__';
- in eg: ... where LName in ('Jones', 'Wong', 'Harrison')

- is eg: ... where DNo is null ;

- between eg: ... where Salary between 1000 and 3000 ;

⇒ 1000 <= Salary <= 3000

eg: ... where LName between "A" and "K" ;

⇒ ["A" , "K")

"<>" == 'not equals'

---

order by <column name> ase ; (for asending order (default) , eg: 1,2,3)

desc ; (for descending order eg: z, y, x ..)

union A ∪ B intersection A ∩ B except A − B

select .....

union / intersection / except

select .....;

R

ID	A	B
1	2	1
2	2	0
3	3	1
4	3	0
5	4	1

6

select * A from R where B=1 or B=0 or A=4;

A
2
3
4
2
3
4

same as: select A from R where B=1 union all
select A from R where B=0 union all
select A from R where A=4 ;

select A from R where B=1 union
select A from R where B=0 union.
select A from R where A=4 ;

A
2
3
4

Find A of R who has been in B in 0 and 1

wrong: ✗ select A from where B=0 and B=1;

correct: select A from where B=0 intersection
select A from where B=1;

Find A of R who B equal to 1 but not equal to 0

select A from where B=1 except.
select A from where B=0;

Aggregation-Function.

- sum/avg ([distinct] <column name>)
- count ([distinct] <column name> / *)
- max/min (<column name>).

When group by is used in an SQL statement, any attribute appeared in SELECT Clause must also appeared in a aggregation function or in Group By Clause

eg: select B, sum(A) from R group by B;

Find A of R who has not been in R with B = 0

wrong: ✗ select A from R where B <> 0;

correct ①: select A from R where B <> 0 except
select A from R where B = 0;

②: select A from R where
A not in (select A from R where B=0);

---

cName	state
Stanford	CA
Berkeley	CA
MIT	MA
Cornell	NY

For each college, check if there is another college in the same state.

```
select cName, state from college C1
where exists (select * from college C2
              where C2.state = C1.state and
              C2.cName <> C1.cName);
```

- ANY: Evaluates to true if one comparison is true
- ALL: Evaluates to true if all comparisons are true.

(in) = (= any)

(not in) = (<> all)

- use joins when you are displaying results from multiple tables
- use sub-queries when you need to compare aggregates to other values

Find the IDs of movie stars who have played in all of the movies

- ~~Move~~ Movie (MovieID, Title, Year)
- StarsIn (MovieID, starID, Role)
- MovieStar (StarID, Name, Gender)

A

Sno	Pno
S1	P1
S1	P2
S1	P3
S2	P1
S2	P2
S3	P2
S4	P2
S4	P3

B

Pno
P2
P3

A / B ⇒

Sno
S1
S4

select startid, movieid from starsin

/ select movieid from movies

⇓

~~select~~ not exists (select ~~not~~ M.movieid from Movie M where not exists (select ~~S.movid~~ S.movieid from Starsin S where S.movieid = M.movieid))

⇒ select MS.starID from MovieStar Ms where not exists (select M.movieid from Movie M where not exists (select S.movieid from Starsin S where S.movieid = M.movieid and S.starID = MS.starID))

create view <view name> (<column name>, ---) as
<select statement>;

drop view [if exists] <view_name> [restrict | cascade];

- restrict: drop the table, unless there is a view on it.

- cascade: drop the table, and recursively drops any view referencing it

~~inster~~ insert into <table name> [(<column name> {, <column name>})]
values (<constant value>);

delete from <table name> [where <select condition>];

update <table name> set <column name> = <value expression>
[where <select condition>];

---

create table student
(
...
primary key (snum),
check (age >=10 and age <100),
check (NOT EXISTS (select starID From MovieStar
where starID not in (select starID
from starsIn)));

---

create trigger youngStudentUpdate after insert on Student
referncing new table NewStudent for each statement insert into
Youngstudent (snum, sname, age) select snum, sname, age FROM NewStudent
where NewStudent.age <= 18;

$T(n)$

$O(c)$

$O(\log n)$

$O(\log^k n)$ $(k>1)$

$O(n^c)$ $(0<c\leq 1)$

$O(n \log n)$

$O(n^k)$ $(k>1)$

$O(c^n)$

selection Sort

```
a=[ .... ]
for i in range( len(a) ):
    index = i
    for j in range(i+1, len(a)):
        if a[j] < a[index]:
            index = j
    a[i], a[index] = a[index], a[i]
```

insertion sort

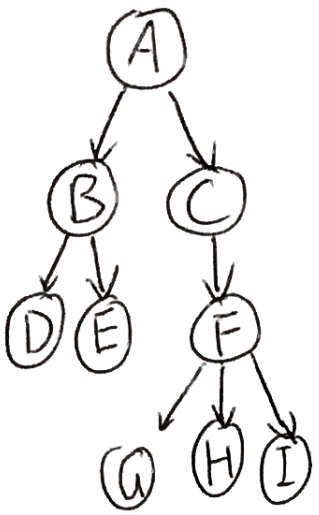
```
for i in range(1, len(a));
    index = i
    min = a[i]
    while index >0 and min < a[index-1];
        a[index] = a[index-1]
        index -=1
    a[index] = min
```

insertion sort

```
for i in range(1, len(a)):
    for j in range (i):
        if a[i-j] < a[i-j-1];
            a[i-j], a[i-j-1] = a[i-j-1], a[i-j]
        else:
            break
```

Merge Sort.

```
def merge_sort(a):
    if n > 1:
        mid = n//2
        A1 = a[:mid]
        A2 = a[mid:]
        merge_sort(A1)
        merge_sort(A2)
        j=0
        i=0
        while i<mid and mid+j<n:
            if A1[i] < A2[j]:
                a[i+j] = A1[i]
                i+=1
            else:
                a[i+j] = A2[j]
                j+=1
        while i< mid:
            a[i+j] = A1[i]
            i+=1
        while mid+j<n
            a[i+j] = A2[j]
            j+=1
```



A is a root

D is a leaf

B and C are the child of A

the parent of B is A

B is the sibling of C

C and A are the ancestor of F

F and C are the descendent of A

is a subtree

the depth of E is 2

the height of tree is 3

the degree of C is 1, the degree of A is 2

the branching factor of tree is 3.

the branching factor of 2 is called binary

the branching factor of n is called n-ary

Relation Integrity Constraint 790/ midterm exam

- Domain constraints

  eg: Violates Domain constraint, because the specification says the "date opened" attribute is automatically populated, therefore it cannot be null

- Key constraints

  eg: violates key constraint, because there already exists a DEPARTMENT tuple with DNUMBER=4.

- Entity constraints

  eg: Violates entity integrity, because PNO, which is part of the primary key of WORKS_ON, is null

- Referential integrity constraints

  eg: Deleting DNUM=2 will leave a dangling reference in the child table foreign table.

  eg: Violates referential integrity because DNUM=2 and there is no tuple in the DEPARTMENT relation with DNUMBER=2.

① $A \to C$ ② $C \to D$

a insertion anomaly

A	B	C	D
1	3	2	5
1	4	2	5
2	3	1	2
3	2	1	2

eg: INSERT $<1,3,3,5>$

INSERT $<$null$,3,3,5>$, distributed systems without knowing the value of A

a deletion anomaly

eg: DELETE $<3,2,1,2>$, we lose information $A \to C$ $(3 \to 1)$

a modification anomaly.

eg: UPDATE $<1,3,2,5>$ to $<1,3,3,5>$

creates inconsistency with $1 \to 2$ and $1 \to 3$ $(A \to C)$

R1 [A, B] FD1: $A \to B$

R2 [C, D, E] FD2: $C \to \{D, E\}$

R_3 [A, C] No non-trivial FD