日期: /

Attribute (/column) Eg: name

Schema: table with list of attributes 

Tuple: (/ row) Eg: (1, James, ——)

Relation: (/ table)

Super key ⊃ Candidate key ⇒ Primary key

(uniquely identify tuples) (minimal) (one of candidate key)

Schema diagram

? Foreign key: { one to one / one to many / many to many

Relational algebra:

Relational algebra { expression: $\Pi_{name}(\sigma_{price>8.0}(Product))$ / tree: Π_{name} over $\sigma_{price>8.0}$ over product } Query

Select σ: $\sigma_{price>8.0}(Product)$ $\sigma_{(price>8.0) \vee (brand = 'DP')}(Product)$

and ∧ , or ∨ , not ¬

Project Π: $\Pi_{name}(Product)$, show 'name' attribute.

Cartesian product ×: $\sigma_{Customer.cust\text{-}id = Invoice.cust\text{-}id}(Customer \times Invoice)$

{ Union ∪: prod-id {1, 2} ∪ prod-id {4} = prod-id {1, 2, 4}, union the tuples.

Set difference −: {1, 2} − {2, 5} = {1}, tuples − tuples.

Set intersection ∩: {1, 2} ∩ {2, 5} = {2}, tuples

日期: /

Natural join ⋈: equal to $\Pi_{R\text{ attributes} \cup S\text{ attributes}}(\sigma_{R.a = S.a}(R \times S))$

Rename ρ: $\rho_{Temp}(Product)$, $Product \times \rho_{Temp}(Product)$

Assignment ←: (like variable, for simplification)

Eg: $Temp1 \leftarrow \Pi_{-}(\sigma_{(}())$

$Temp2 \leftarrow \Pi_{-}(\sigma_{e}())$

$Temp1 \cup Temp2$.

Eg: Database modification

$Project \leftarrow \Pi_{-}(\sigma_{-}(Project))$

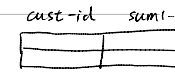
Generalized projection: Eg: $\Pi_{price * 80\% \text{ as discount price}}(Product)$

Aggregate function: (Function: sum, avg, count, min, max)

Eg: $\mathcal{G}_{max(price)}(Product)$. → 

Aggregate function + Group: first Group → Aggregate function based on group

Eg: $cust\text{-}id\ \mathcal{G}_{sum(amount)}(Invoice)$

↑ Group by ↑ Function on each group → 

Null value: 

Insertion: $Product \leftarrow Product \cup \{(5, 'soda', --)\}$

insert new tuple.

Deletion: $Product \leftarrow Product - \mathcal{G}_{p}(Product)$.

delete tuple.

日期: SQL / Clause

create table Invoice (inv_id int,
primary key (inv_id))

alter table Invoice add staff_id int
(add column)

---

① select * (all) A1, A2 from r1, r2. where P ;

<=> $\Pi_{A_1, A_2}(\sigma_P(r_1 \times r_2))$.

② select -- from -- { where price > 8.0 and price < 9.0
where price between 8.0 and 9.0

③ select I.inv_id , C.name
from Customer as C , Invoice as I
where C.cust_id = I.cust_id

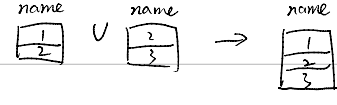
$\Pi_{...}$ (customer ⋈ Invoice)

④ select * from Product order by price asc (desc)

⑤ select max(price) from Invoice

⑥ select * from Invoice group by cust_id

⑦ (select name from A)
union (intersect, except)
(select name from B)



union all →

1
2
2
3

⑧ insert into Product values (5, 'soda' —)

日期: /

delete from Product where prod_id=3

update Product set price=price * 2 where brand = 'AB'

⑦ inner join

select * from
A inner join B on
A.id = B.id

⑧ left outer join : All atteribute in left table appear in result. fill Null.

select * from
A left outer join B on
A.id = B.id

⑨ full outer join : left and right all appear.

⑩ subqueries :

select * from Product
wher price <
(select avg(price) from Product)

⑪ 78 in 日, Eg. where price < all
80 > some 日, (select price from Product
80 > all 日. where brand = 'CO') ;

⑫ exists ↓ ∃ exists in (table) | unique ↓ no duplicates

⑬ view create view P as
select * from Project;

⑮ group by ... having ~~ ; select cust_id, sum(amount)
from ... group by ~~ ;

日期: /

⑭ create view DPC as view = temp(table)

select ~~

from ~~

group by ~~

having ~~ ;

⑮ slect * from ~ where { not exists / ~ < some / ~ < all

subquery (select ~ from ~ where ~~)

⑯ select ~ from A (only support A.~, not B.~)

inner join B on B.~ = A.~ (B must in front)

group by A.attr ; (B must in front)

having AVG(A.~) = ~ ; (A.attr must in select A.attr)

B to A: many to one Def: B ⟩ A.

B to A : is many to one. (or one to one)

C (B-PK, A-PK) ← (B-PK) superkey

schema diagram (relational schema)

if A ← C — B

then

A: PK

B: PK, A-PK (FK)

employ: ssn

work_in: ssn (FK), dno (FK)

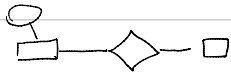
depart: dno, manage_ssn (FK)

children: name, employee_ssn (FK)

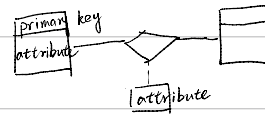
FK: Foreign key

日期: /

ER Diagram



UML notation



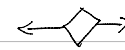
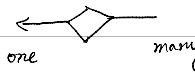
Entity set : (tuple) 

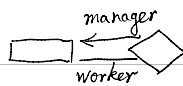
Relationship set : 

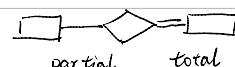
Multi-valued attributes : 

have many values

Eg: one man has 2 phones

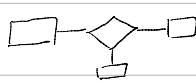
- **one to one** relationship 
- **one to many** relationship 

Role 

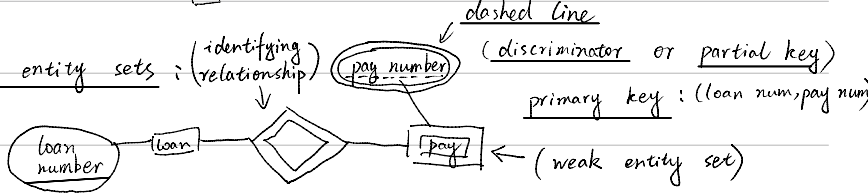
- **Total** participation : 

- **Partial**

Degree : number of entity sets

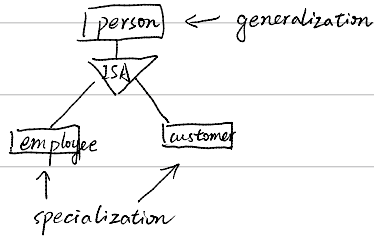


degree = 3

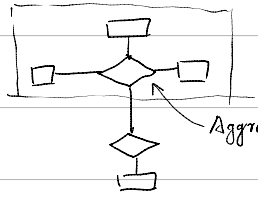
Weak entity sets : (identifying relationship) 

日期: /

specialization, generalization :



Aggregation : eliminate redundancy

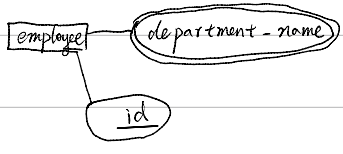


Representing entity set :

strong entity set : loan = (loan number, amount)

weak entity set : payment = (loan number, payment number)

multivalued attribute : employee - depart-name = (id, depart-name)



Specialization :

(generalization) person = (id, ——)

(specialization) customer = (id, ——) , employee = (id, ——)

日期: /

First Norm

Def:

① attribute is atomic : cannot be devided into parts
(CS101) ↛ (CS, 101)

② no duplicates

- Lossy decomposition : if decompose $R=(A,B,C) \to R_1=(A,B), R_2=(B,C)$
  r → $\Pi_{A,B}(r)$, $\Pi_{B,C}(r)$
  $\Pi_{A,B}(r) \bowtie \Pi_{B,C}(r) \neq r$.
- Lossless-join decomposition :

good form : ① lossless-join decomposition, ② $R_1, \cdots, R_n$ is good form
split $R \to \{R_1, \cdots R_n\}$, e.g. BCNF, 3NF
(Good Form)

$\alpha \to \beta$ is rule : α determines uniquely β
(well defined)
(default : surjective)

1	4
1	5
3	7
A	B

$B \to A$ ✓, $A \to B$ ✗

tuple t_1, t_2 : if $\alpha \to \beta$, $t_1(\alpha) = t_2(\alpha) \Rightarrow t_1(\beta) = t_2(\beta)$
(key)

functional dependencies : $\alpha \to \beta$

super key : $K \to R$ (relation schema R)

candidate key : $K \to R$, $\nexists\, \alpha \subset K$, s.t. $\alpha \to R$.
(ID, name) → (ID)

trivial : $\alpha \to \beta$ is trivial if $\beta \subset \alpha$. (ID, name) → (ID)

Armstrong's axioms :
① $\beta \subset \alpha \Rightarrow \alpha \to \beta$ (reflexivity)
② $\alpha \to \beta \to \gamma\alpha \to \gamma\beta$ (augmentation) + attribute $\gamma\alpha$:
③ $\alpha \to \beta, \beta \to \gamma \to \alpha \to \gamma$ (transitivity)

日期: /

△ closure of $\mathcal{F}$: $\mathcal{F}^+$ $\quad \mathcal{F} = \{ \underbrace{A \to B}_{\uparrow}, A \to C, \cdots \}$

$\mathcal{F}^+ = \text{expand}(\mathcal{F})$ $\quad$ functional dependencies

① union: $\alpha \to \beta, \alpha \to \gamma \Rightarrow \alpha \to \beta\gamma$

② decomposition: $\alpha \to \beta\gamma \Rightarrow \alpha \to \beta, \alpha \to \gamma$

③ pseudo-transitivity: $\alpha \to \beta, \gamma\beta \to \delta \Rightarrow \gamma\alpha \to \delta$.

closure of α (i.e. α^+) under $\mathcal{F}$

① result := α
while (result changed) do
for each $\beta \to \gamma$ in $\mathcal{F}$ do
if $\beta \subset$ result then
result := result $\cup \gamma$

② $\mathcal{F}^+$:
for each $\gamma \subset R$
find closure γ^+
for each $\beta \subset \gamma^+$
output $\gamma \to \beta$.
$\mathcal{F}^+ = \mathcal{F}^+ \cup (\gamma \to \beta)$

find candidate key: 1个{A}, 2个{AB}, ——.

△ canonical cover of $\mathcal{F}$ (minimal) :

$\mathcal{F}_c$ means no redundant and extraneous.

extraneous : Eg. ① $\mathcal{F} = \{A \to C, AB \to C\}$
B is extraneous in $AB \to C$

② $\mathcal{F} = \{A \to C, AB \to CD\}$ C is extraneous.
$\mathcal{F}_c = \{A \to C, AB \to D\}$

Lossless-join decomposition : $R = (R_1, R_2)$ $\quad$ r: relation R: attr

$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

△ Thm: R_1, R_2 is loss-less decomposition if $\begin{cases} R_1 \cap R_2 \to R_1 \\ \text{or} \\ R_1 \cap R_2 \to R_2 \end{cases}$

dependency preserving : ① $\mathcal{F}$ must be canonical cover. $\mathcal{F} = \mathcal{F}_c$

② E.g. $\mathcal{F} = \{A \to B, B \to C\}$ $R_1 = (A, B)$, $R_2 = (B, C)$

all functional dependency can be splitted into $R_1, \cdots$

日期: /

BCNF Normal Form : ∀ functional dependency $\alpha \to \beta \in \mathcal{F}^+$

at least one satisfies : $\alpha \in R, \beta \in R$

- $\alpha \to \beta$ is trivial (ie. $\beta \subset \alpha$)
- α is superkey for R

BCNF decomposition: each R_i is in BCNF, and decomposition is lossless-join.

(preserve dependency)

3NF : ∀ $\alpha \to \beta$ in $\mathcal{F}^+$

at least one satisfies :

- $\alpha \to \beta$ is trivial (i.e. $\beta \in \alpha$)
- α is superkey for R
- each attribute A in $\beta - \alpha$, is contained in a candidate key for R.

(maybe different candidate key)

3NF decomposition ensures : each R_i is in 3NF,

may not dependency preserving

loss-less decomposition.

3NF ⊃ BCNF

Boyce-Codd Normal Form (BCNF)

A relation schema R is in **BCNF** with respect to a set $\mathcal{F}$ of functional dependencies if for all functional dependencies in $\mathcal{F}^+$ of the form

$$\alpha \to \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \to \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for R

日期: /

BCNF Decomposition Algorithm

result := {*R* }
compute $\mathcal{F}^+$
while (some schema R_i in *result* is not in BCNF) **do**
  let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that
    holds on R_i such that $\alpha \rightarrow R_i$ is not in $\mathcal{F}^+$,
    and $\alpha \cap \beta = \varnothing$
  result := (*result* – R_i) ∪ (R_i – β) ∪ (α, β)

Note: each R_i is in BCNF, and decomposition is lossless-join

Third Normal Form (3NF)

- A relation schema *R* is in **3NF** if for all:
  $$\alpha \rightarrow \beta \text{ in } \mathcal{F}^+$$
  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for *R*
  - Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R*
    (**NOTE**: each attribute may be in a different candidate key)
- A relation is in BCNF → it is in 3NF
  - Because BCNF requires one of the first two conditions

3NF Decomposition Algorithm

let $\mathcal{F}_c$ be a canonical cover for $\mathcal{F}$
i := 0
for each functional dependency $\alpha \rightarrow \beta$ in $\mathcal{F}_c$ **do**
  if no schema R_j, $1 \le j \le i$ contains $\alpha\ \beta$ **then**
    i := *i* + 1
    R_i := $\alpha\ \beta$
  if no schema R_j, $1 \le j \le i$ contains a candidate key for *R* **then**
    i := *i* + 1
    R_i := any candidate key for *R*
/* remove redundant relations */
while (some schema R_j is contained in another schema R_k)
  R_j := R_i
  i:=*i*–*1*
return (R_1, R_2, ..., R_i)

It ensures:
- each schema R_i is in 3NF
- decomposition is dependency preserving and lossless-join

日期: /