Chapter 2: Relational Model

- Structure of Relational Databases
- Relational Algebra
- *Tuple Relational Calculus
- *Domain Relational Calculus
- *Extended Relational-Algebra-Operations
- Modification of the Database
- *Views

2.1 What is relational model

- The relational model is very simple and elegant.
- A **relational database** is a collection of one or more **relations,** which are based on relational model.
- A **relation** is a *table* with *rows* and *columns*.
- The major advantages of the relational model are its *simple data representation* and *the ease* with which even complex queries can be expressed.
- Owing to the great language SQL, the most widely used language for creating, manipulating, and querying relational database.

Example of a Relation

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

A relation for *instructor*

Cf. **A relationship (联系): an association among several entities . A relation(关系): is the mathematical concept, referred to a table**

Entity set and relationship set ←→ real world

Relation - table, tuple - row ←→ machine world

2.2 Basic Structure

- Formally, given sets D_1, D_2, …, D_n, $(D_i = a_{ij}|_{j=1\ldots k})$ a relation r is a subset of $D_1 \times D_2 \times \ldots \times D_n$

  - a Cartesian product (笛卡儿积) of a list of domain D_i.
- Thus a relation is a set of n-tuples $(a_{1j}, a_{2j}, \ldots, a_{nj})$ where each $a_{ij} \in D_i$.
- 例如：

张清玫教授，计算机，李勇

张清玫教授，计算机，刘晨

刘逸教授，信息，王名

A relation for sup-spec-stud

Example of Cartesian product

例如 D_1= 导师集合 = { 张清玫，刘逸 }，

D_2 = 专业集合 = { 计算机，信息 }，

D_3 = 研究生集合 ={ 李勇，刘晨，王名 }

则 $D_1 \times D_2 \times D_3$ ={(张清玫，计算机，李勇) ,
(张清玫，计算机，刘晨) ,
(张清玫，计算机，王名) ,
(张清玫，信 息，李勇) ,
(张清玫，信 息，刘晨) ,
(张清玫，信 息，王名) ,
(刘 逸，计算机，李勇) ,
(刘 逸，计算机，刘晨) ,
(刘 逸，计算机，王名) ,
(刘 逸，信 息，李勇) ,
(刘 逸，信 息，刘晨) ,
(刘 逸，信 息，王名) },

共 12 个元组。

	D1	D2	D3
√	张清玫	计算机	李勇
√	张清玫	计算机	刘晨
	张清玫	计算机	王名
	张清玫	信 息	李勇
	张清玫	信 息	刘晨
	张清玫	信 息	王名
	刘逸	计算机	李勇
	刘逸	计算机	刘晨
	刘逸	计算机	王名
	刘逸	信 息	李勇
	刘逸	信 息	刘晨
√	刘逸	信 息	王名

笛卡儿积可用一张二维表表示

sup-spec-stud

张清玫	计算机	李勇
张清玫	计算机	刘晨
刘逸	信息	

- **Example: if**

  dept_name **= {Biology,Finance,History,Music}**

  building **= {Watson,Painter,Packard}**

  budget **= {50000,80000,90000,120000}**

Then ***r*** **= {(Biology,Watson,90000),**
(Finance,Painter,120000),
(History,Painter,50000),
(Music,Packard,80000)}

is a relation over ***dept_name*** ***x*** ***building*** ***x*** ***budget*** **. (total 48 tuples)**

(1) Attribute Types

- Each attribute of a relation has a name.
- The set of allowed values for each attribute is called the domain (域) of the attribute.
- Attribute values are (normally) required to be atomic, that is, indivisible. (1st NF, 第一范式)
  - E.g. multivalued attribute values are not atomic
  - E.g. composite attribute values are not atomic
- The special value *null* is a member of every domain.
- The null value causes complications in the definition of many operations.
  - we shall ignore the effect of null values for the moment and consider their effect later.

(2) Concepts about relation

- ***A Relation*** **is concerned with two concepts : relation schema and relation instance.**
- **The relation schema describe the structure of the relation.**

  E.g. *Instructor-schema = (ID*: string, *name*: string, *dept_name*: string, *salary*: int)

  or *Instructor-schema = (ID, name, dept_name, salary*)
- **The relation instance corresponds to the snapshot of the data in the relation at a given instant in time.**
- **C.f.: Database schema and database instance.**

- Variable ←→ relation
- Variable type ←→ relation schema
- Variable value ←→ relation instance

(2-a) Relation Schema

- A_1, A_2, …, A_n are ***attributes***
- **Formally expressed :**

  $R = (A_1, A_2, \ldots, A_n)$ is a ***relation schema***

E.g. Instructor-schema = (*ID, name, dept_name, salary*)

- $r(R)$ is a ***relation*** on the ***relation schema R***

E.g. *instructor(Instrutcor-schema)= instructor*(*ID, name, dept_name, salary*)

(2-b) Relation Instance

- **The current values (*relation instance*) of a relation are specified by a table.**
- **An element *t* of *r* is a *tuple*, represented by a *row* in a table.**

attributes (or columns)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

tuples (or rows)

- **Let a *tuple variable t* stands for a tuple. Then**

  ***t [name]* denotes the value of *t* on the *name* attribute.**

(3) Relations are Unordered

- **Order of tuples is irrelevant (tuples may be stored in an arbitrary order), and tuples in a relation are no duplicate.**
- **E.g. *department*(*dep_name*, *building*, *budget*) relation with unordered tuples.**

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

(4) Keys (码、键)

- **Let K $\subseteq$ R**
- ***K* is a *superkey* (超码) of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)***
  - **Example: {*instructor-ID, instructor-name*} and {*instructor-ID*} are both superkeys of *instructor*.**
- ***K* is a *candidate key* (候选码) if *K* is minimal superkey. Example: {*instructor-ID*} is a candidate key for *instructor*, since it is a superkey, and no subset of it is a superkey.**
- **K is a primary key (主码), if k is a candidate key and been defined by user explicitly. Primary key is usually marked by underline.**

(5) Foreign key（外键，外码）

- **Assume there exists relation *r* and *s*: r(<u>A</u>, *B*, C), s(<u>*B*</u>, D), we can say that attribute B in relation *r* is foreign key referencing s, and *r* is a referencing relation（参照关系）, and s is a referenced relation（被参照关系）.**

e.g. ◆ 学生（<u>学号</u>，姓名，性别，专业号，年龄）- *参照关系*

专业（<u>专业号</u>，专业名称）- *被参照关系 （目标关系）*

其中属性*专业号*称为关系学生的*外码*。

选修（<u>学号，课程号</u>，成绩） 课程（<u>课程号</u>，课程名，学分，先修课号）

◆ ***instructor*(<u>*ID*</u>,*name*,*dept_name*,*salary*) --- referencing relation**

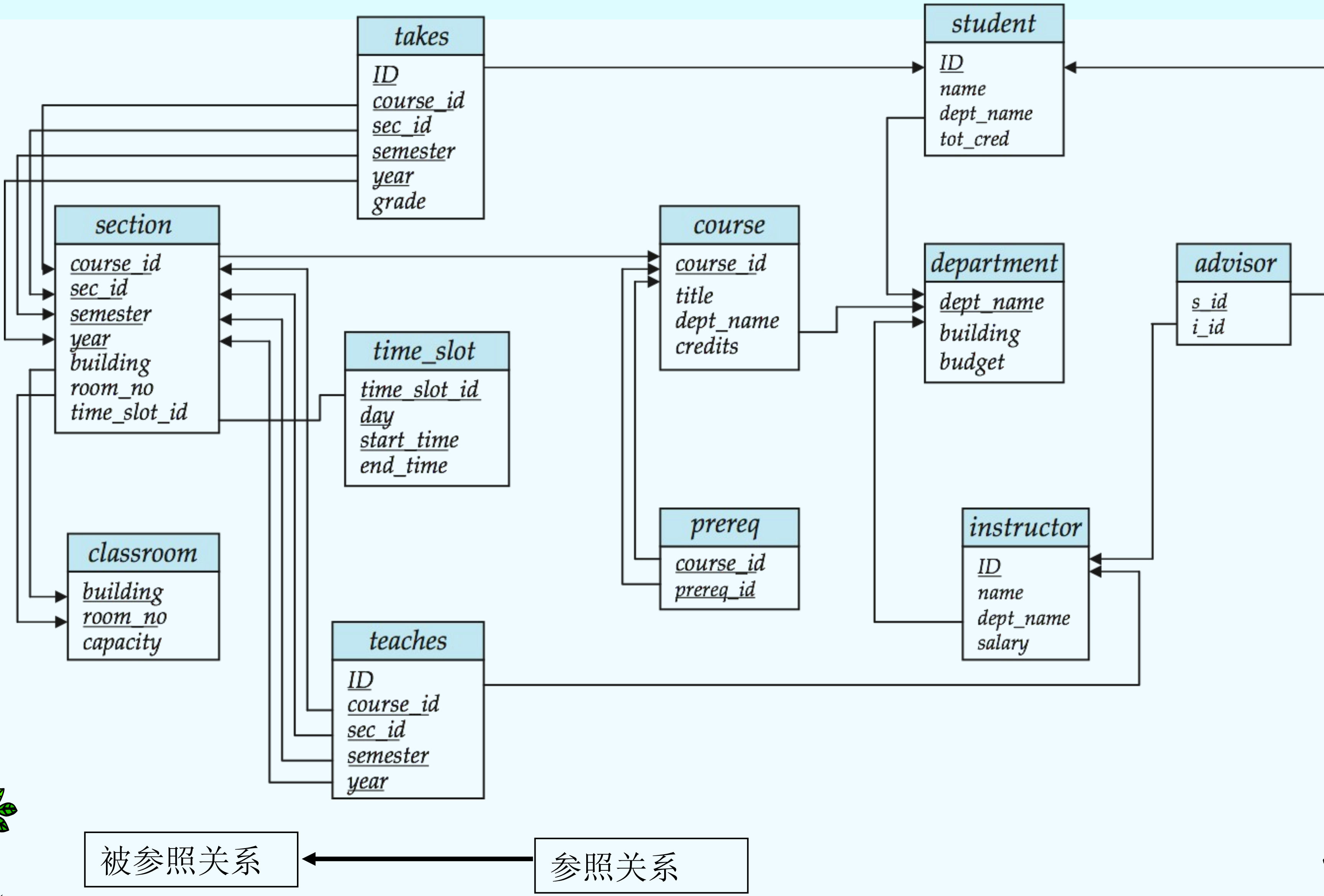
***department*(<u>*dept_name*</u>,*building*,*budget*) --- referenced relation**

参照关系中外码的值必须在被参照关系中实际存在或为 null

(6) Schema of the University Database

- ***classroom(building,room_number,capacity)***
- ***department(dept_name,building,budget)***
- ***course(course_id,title,dept_name,credits)***
- ***instructor(ID,name,dept_name,salary)***
- ***section(course_id,sec_id,semester,year,building,room_number,time_slot_id)***
- ***teaches(ID,course_id,sec_id,semester,year)***
- ***student(ID,name,dept_name,tot_cred)***
- ***takes(ID,course_id,sec_id,semester,year,grade)***
- ***advisor(s_ID,i_ID)***
- ***time_slot(time_slot_id,day,start_time,end_time)***
- ***prereq(course_id,prereq_id)***

(6) Schema Diagram（模式图）for the University Database



(7) Query Languages

- **Language in which user requests information from the database.**
- **"Pure" languages:**
  - **Relational Algebra - the basis of SQL**
  - **Tuple Relational Calculus (元组关系演算)**
  - **Domain Relational Calculus - (域关系演算) QBE**
- **Pure languages form underlying basis of query languages that people use, e.g. SQL.**

2.3 Relational Algebra

- **Procedural language （in some extent).**
- **Six basic operators**
  - **Select** 选择
  - **Project** 投影
  - **Union** 并
  - **set difference** 差（集合差）
  - **Cartesian product** 笛卡儿积
  - **Rename** 改名（重命名）
- **The operators take one or two relations as inputs and give a new relation as a result.**
- **Additional operations**
  - **Set intersection** 交
  - **Natural join** 然连接
  - **Division** 除
  - **Assignment** 赋值

(1) Select Operation – Example

- Relation r =

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

注：执行选择时，选择条件必须是针对同一元组中的相应属性值代入进行比较。

$\forall\ \sigma_{A=\beta\ \wedge\ D>5}(r)$

A	B	C	D
β	β	23	10

(1) Select Operation (cont.)

- **Notation:** $\sigma_p(r)$, σ **is pronounced as sigma**
- ***p*** **is called the selection predicate**
- **Defined as:** $\sigma_p(r) = \{ t \mid t \in r \text{ and } p(t) \}$

Where *p* is a formula in propositional calculus consisting of terms connected by: $\wedge$ **(and),** $\vee$ **(or),** $\neg$ **(not)**

Each term is one of:

<attribute>*op* <attribute> or <constant>

where *op* is one of: $=, \neq, >, \geq, <, \leq$

- **Example of selection:**

$$\sigma_{dept\_name = \text{'Finance'}}(department)$$

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

$$\sigma_{dept\_name = 'Finance'}(\textbf{\textit{department}})$$

(2) Project Operation – Example

- **Relation *r* :**

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=>

A	C
α	1
β	1
β	2

(2) Project Operation (cont.)

- **Notation:**

  $$\prod_{A1,\ A2,\ \ldots,\ Ak} (r),\ \prod \text{ is pronounced as pi}$$

  where A_1, … A_k are attribute names and *r* is a relation name

- **The result is defined as the relation of *k* columns obtained by erasing the columns that are not listed**
- **Duplicate rows removed from result, since relations are sets.**
- **E.g. To eliminate the *building* attribute of *department***

  $$\prod_{building}(department)$$

(3) Union Operation – Example

- **Relations *r, s:***

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r ∪ s:

A	B
α	1
α	2
β	1
β	3

(3) Union Operation (cont.)

- **Notation:** $r \cup s$
- **Defined as:** $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$
- **For $r \cup s$ to be valid：**

  1. *r, s* must have the same *arity* (等目 , 同元 ,same number of attributes)

  2. The attribute domains must be *compatible* (e.g., 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

- **E.g. to find all instructors and students**

$$\boxed{\Pi_{name}(instructor) \cup \Pi_{name}(student)}$$

(4) Set Difference Operation – Example

- **Relations *r, s:***

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

***r* – *s*:**

A	B
α	1
β	1

(4) Set Difference Operation (cont.)

- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \text{ and } t \notin s\}$$

- Set differences must be taken between ***compatible*** relations.
  - *r* and *s* must have the ***same arity***
  - attribute domains of *r* and *s* must **be compatible**

(5) Cartesian-Product Operation-Example (广义笛卡儿积)

Relations *r*, *s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

r* x *s:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

(5) Cartesian-Product Operation (cont.)

- **Notation** $r \times s$
- **Defined as:**

$$r \times s = \{ \{t\, q\} \mid t \in r \text{ and } q \in s\}$$

- **Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).**
- **If attributes of *r(R)* and *s(S)* are not disjoint, then renaming for attributes must be used.**

A	B
α	1
α	2
β	1

r

B	C
k	2
d	3

s

r x *s* =

A	r.B	s.B	C
α	1	k	2
α	2	k	2
β	1	k	2
α	1	d	3
α	2	d	3
β	1	d	3

(6) Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$

r

A	B
α	1
β	2

s

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

***r* x *s*=**

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

$\sigma_{A=C}(r \times s)$ =

A	B	C	D	E
α	1	α	10	a
β	2	β	20	a
β	2	β	20	b

(7) Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions. (procedural)
- Allows us to refer to a relation by more than one name.

Example: $\rho_{x}(E)$, ρ is pronounced as rho

returns the expression E under the name X

If a relational-algebra expression E has arity n, then

$$\rho_{x\ (A1,\ A2,\ \ldots,\ An)}(E)$$

(对 relation E 及其 attributes 都重命名）

returns the result of expression E under the name X, and with the attributes renamed to $A1, A2, \ldots, An$.

Banking Example

- ***branch** (__branch-name__, branch-city, assets)*
- ***customer** (__customer-name__, customer-street, customer-city)*
- ***account** (__account-number__, branch-name, balance)*
- ***loan** (__loan-number__, branch-name, amount)*
- ***depositor** (__customer-name__, __account-number__)*
- ***borrower** (__customer-name__, __loan-number__)*

Example Queries

- **Example 1: Find all loans of over $1200**

$$\sigma_{amount\ >\ 1200}\ (loan)$$

- **Example 2: Find the loan number for each loan of an amount greater than $1200**

$$\Pi_{loan\text{-}number}\ (\sigma_{amount\ >\ 1200}\ (loan))$$

loan (loan-number, branch-name, amount)

Example Queries

- **Example 3: Find the names of all customers who have a loan, or an account, or both, from the bank**

$$\Pi_{customer\text{-}name}\,(borrower) \cup \Pi_{customer\text{-}name}\,(depositor)$$

- **Example 4: Find the names of all customers who at least have a loan and an account at bank.**

$$\Pi_{customer\text{-}name}\,(borrower) \cap \Pi_{customer\text{-}name}\,(depositor)$$

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Example Queries

- **Example 5: Find the names of all customers who have a loan at the Perryridge branch.**

Query1: $\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{“Perryridge”}}(\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}(borrower \times loan)))$

Query2: $\Pi_{customer\text{-}name}(\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}(borrower \times (\sigma_{branch\text{-}name=\text{“Perryridge”}}(loan))))$

Query2 is better.

loan (loan-number, branch-name, amount)
borrower (customer-name, loan-number)

Example Queries

- **Example 6: Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.**

Query1: $\Pi_{customer\text{-}name}\,(\sigma_{branch\text{-}name\,=\,\text{“Perryridge”}}$

$(\sigma_{borrower.loan\text{-}number\,=\,loan.loan\text{-}number}(\text{borrower x loan}))) \;-$

$\Pi_{customer\text{-}name}(\text{depositor})$

Query2: $\Pi_{customer\text{-}name}\,(\sigma_{borrower.loan\text{-}number\,=\,loan.loan\text{-}number}$

$(borrower\ x\ (\sigma_{branch\text{-}name=\text{“Perryridge”}}(loan)))) \;-$

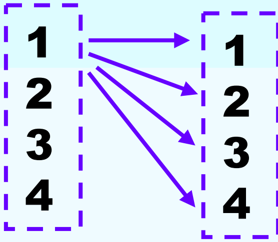
$\Pi_{customer\text{-}name}\,(\text{depositor})$

loan (loan-number, branch-name, amount)

borrower (customer-name, loan-number)

depositor (customer-name, account-number)

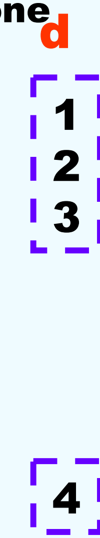
Example Queries



- **Example 7: Find the largest account balance.**

  (须进行自比较**)**

  - **Rename *account* relation as *d***
  - **Step 1: find the relation that contains all balances except the largest one**



$$\Pi_{account.balance}$$

$$(\sigma_{account.balance\ <\ d.balance}\ (account \times \rho_d\ (account)))$$

Step 2: find the largest account balance.

$$\Pi_{balance}(account) - \Pi_{account.balance}$$

$$(\sigma_{account.balance\ <\ d.balance}\ (account \times \rho_d\ (account)))$$

Additional Operations

- We define additional operations that do not add any power to the relational algebra, but that simplify common queries.
- Set intersection
- Natural join
- Division
- Assignment

(8) Set-Intersection Operation - Example

- **Relation r, s:**

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- **r ∩ s**

A	B
α	2

(8) Set-Intersection Operation

- **Notation:** $r \cap s$
- **Defined as:**
- $r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$
- **Assume:**
  - *r*, *s* have the *same arity*
  - attributes of r and s are compatible
- **Note:** $r \cap s = r - (r - s)$

(9) Natural-Join Operation

- Notation: $r \bowtie s$
- **Example:**

  R **= (*A*, *B*, *C*, *D*);** ***S*** **= (*E*, *B*, *D*)**

  - **Result schema of natural-join of *r* and *s* = (*A*, *B*, *C*, *D*, *E*) ;**
  - $r \bowtie s = \Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s))$

- **Let *r* and *s* be relations on schemas *R* and *S* respectively.**
  Then, $r \bowtie s$ is a relation on schema $R \cup S$ obtained as follows:
  - **Consider each pair of tuples t_r from *r* and t_s from *s*.**
  - **If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where**
    - ***t* has the same value as t_r on *r***
    - ***t* has the same value as t_s on *s***

Natural Join Operation – Example

- **Relations r, s:**

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
δ	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
δ	1	γ	a	α
δ	1	γ	a	γ
δ	2	β	b	δ

$r \bowtie s$

注：

(1)r,s 必须含有共同属性 （名，域对应相同），

(2) 连接二个关系中同名属性值相等的元组

(3) 结果属性是二者属性集的并集，但消去重名属性。

- **Theta join: $r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$**

  θ is the predicate on attributes in the schema.

- **Theta Join is the extension to the Nature Join.**

(10) Division Operation

$$r \div s$$

- **Suited to queries that include the phrase "for all".**

例：查询选修了所有课程的学生的学号。

enrolled

Sno	Cno	Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

÷

course

Cno
1
2
3

=

Sno
95001

$$\Pi_{Sno,\ Cno}\ (\text{enrolled}) \div \Pi_{Cno}\ (\text{course})$$

(10) Division Operation

$$r \div s$$

- Let *r* and *s* be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of r ÷ s is a relation on schema

  $R - S = (A_1, \ldots, A_m)$

$$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \land \forall u \in s \, (tu \in r) \}$$

注：商来自于$\Pi_{R-S}(r)$，并且其元组 **t** 与 **s** 所有元组的拼接被 **r** 覆盖。

Division Operation – Example

Relations *r, s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1
β	2

r

B
1
2

s

$$(r \div s = \{\ t \mid t \in \prod_{R-S}(r) \wedge [\ \forall\ u \in s\ (\ tu \in r\ )]\ \})$$

***r* ÷ *s* =**

A
α
β

Another Division Example

Relations *r, s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	1
α	b	γ	a	1
α	b	γ	b	1
β	a	γ	a	1
γ	a	γ	c	2
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	c	β	b	1

r

D	*E*
a	1
b	1

s

	A	*B*	*C*	*D*	*E*	
1	α	a	α	a	1	
2	α	b	γ	a	1	√
	α	b	γ	b	1	
3	β	a	γ	a	1	
	β	a	γ	b	3	
4	γ	a	γ	c	2	
	γ	a	γ	a	1	√
	γ	a	γ	b	1	
5	γ	c	β	b	1	

To group all tuples in *'r '* on the values of (A,B,C). For each group, if the set under D, E covers *'s'*, then the group value should be added to the answer.

$r \div s$:

A	*B*	*C*
α	b	γ
γ	a	γ

例：　　求 Q=R ÷ S

R

A	B	C
a1	b1	c1
a2	b1	c1
a1	b2	c1
a1	b2	c2
a2	b1	c2
a1	b2	c3
a1	b2	c4
a1	b1	c5

S

C
c1

Q

A	B
a1	b1
a2	b1
a1	b2

S

C
c1
c4
c2
c3

Q

A	B
a1	b2

S

B	C	D
b1	c1	d1
b2	c1	d2

Q

A
a1

S

C
c1
c2

Q

A	B
a1	b2
a2	b1

例：从 SC 表中查询至少选修 1 号课程和 3 号课程的学生号码。

SC

Sno	Cno	Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

÷

临时表 K

Cno
1
3

=

Sno
95001

$\Pi_{Sno,\ Cno}$ (SC) ÷K

Division Operation (Cont.)

- **Property**
  - **Let** $q = r \div s$
  - **Then** q **is the largest relation satisfying** $q \times s \subseteq r$
- **Definition in terms of the basic algebra operation Let** $r(R)$ **and** $s(S)$ **be relations, and let** $S \subseteq R$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

To see why ,

- $\Pi_{R-S,S}(r)$ **simply reorders attributes of** r
- $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ **gives those tuples t in**

  $\Pi_{R-S}(r)$ **such that for some tuple** $u \in s$, $tu \notin r$.

(11) Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a **temporary** relation variable.
- Example: Write $r \div s$ as

$$temp1 \leftarrow \Pi_{R-S}(r)$$

$$temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$$

$$result = temp1 - temp2$$

  - The result to the **right** of the $\leftarrow$ **is assigned to** the relation variable on the **left** of the $\leftarrow$.
  - May use variable in subsequent expressions.

Example Queries

- **Example 1: Find all customers who have an account from at least the “Downtown” and the “Uptown” branches.**

Query 1

$$\Pi_{CN}(\sigma_{BN=\text{“Downtown”}}(depositor \bowtie account)) \cap$$
$$\Pi_{CN}(\sigma_{BN=\text{“Uptown”}}(depositor \bowtie account))$$

where *CN* denotes customer-name and *BN* denotes *branch-name*.

Query 2

$$\Pi_{customer\text{-}name,\ branch\text{-}name}(depositor \bowtie account)$$
$$\div\ \rho_{temp(branch\text{-}name)}(\{(\text{“Downtown”}),$$
$$(\text{“Uptown”})\})$$

- **Example 2: Find all customers who have an account at all branches located in Brooklyn city.**

$$\Pi_{customer\text{-}name,\ branch\text{-}name}\,(depositor \bowtie account)$$
$$\div\ \Pi_{branch\text{-}name}\,(\sigma_{branch\text{-}city\ =\ \text{“Brooklyn”}}\,(branch))$$

- **example 3:** 查询选修了**全部**课程的学生**学号和姓名**。
  - 涉及表：课程信息 course(cno, cname,pre-cno,score)，选课信息 sc(sno,cno, grade)，学生信息 student(sno, sname, sex, age)
  - 当涉及到求“全部”之类的查询，常用“除法”
  - 找出全部课程号： Π_{Cno} (Course)
  - 找出选修了全部课程的学生的学号：

    $\Pi_{Sno,\ Cno}$ (SC) $\div$ Π_{cno} (Course)
  - 与 student 表自然连接（连接条件 Sno ）获得学号、姓名

    ($\Pi_{Sno,\ Cno}$ (SC) $\div$ Π_{cno} (Course)) $\bowtie$ $\Pi_{Sno,Sname}$(student)

summary

- **Union， set difference， Set intersection** 为双目、等元运算
- **Cartesian product， Natural join ， Division** 双目运算
- **Project, select** 为单运算对象
- **Priority(** 关系运算的优先级 **)：**
  - **Project**
  - **Select**
  - **Cartesian Product(times)**
  - **Join, division**
  - **Intersection**
  - **Union, difference**

2.4 Extended Relational-Algebra-Operations

- **Generalized Projection (** 广义投影 **)**
- **Aggregate Functions (** 聚集函数 **)**
- **Outer Join (** 外连接 **)**

(1) Generalized Projection

- **Extends the projection operation by allowing arithmetic functions to be used in the projection list.**

$$\Pi_{F1,\, F2,\, \ldots,\, Fn}(E)$$

- ***E* is any relational-algebra expression**
- **Each of F_1, F_2, …, F_n are arithmetic expressions involving constants and attributes in the schema of *E*.**
- **Given relation *credit-info(customer-name, limit, credit-balance),* find how much more each person can spend:**

$$\Pi_{\textit{customer-name},\ \textit{limit} - \textit{credit-balance}}\ (\textit{credit-info})$$

(2) Aggregate Functions and Operations

- **Aggregation function** takes **a collection of values** and returns **a single value** as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

  $g_{\text{avg (balance)}}$ **(account)** (求平均存款余额)

- **Aggregate operation** in relational algebra

$$_{G1,\ G2,\ \ldots,\ Gn}\ g\ _{F1(A1),\ F2(A2),\ldots,\ Fn(An)}\ (E)$$

  - *E* is any relational-algebra expression
  - $G_1, G_2 \ldots, G_n$ is **a list of attributes on which to group** (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

(2) Aggregate Operation – Example

- **Relation *r* :**

A	B	C
α	α	7
α	β	7
β	β	8
β	α	14

$\mathcal{G}_{\text{avg(c)}}(r)$

avg-C
9

${}_{A}\mathcal{G}_{\text{sum(c)}}(r)$

A	sum-c
α	14
β	22

${}_{B}\mathcal{G}_{\text{avg(c)}}(r)$

B	avg-c
α	10.5
β	7.5

Aggregate Operation – Example

- **Relation *account* grouped by *branch-name*:**

branch-name	*account-number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$_{branch\text{-}name}\, \mathcal{G}\, _{sum(balance)}\ (account)$

Sum-balance

branch-name	
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Functions (cont.)

- **Result of aggregation does NOT have a name**
  - **Can use rename operation to give it a name**
  - **For convenience, we permit renaming as part of aggregate operation**

$$_{\textit{branch-name}}\, g\, _{\textit{sum(balance)} \textit{ as sum-balance}} (\textit{account})$$

(3) Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) false by definition.
    - Will study precise meaning of comparisons with nulls later

loan_number	*branch_name*	*amount*
L_170	Downtown	3000
L_230	Redwood	4000
L_260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	**L_170**
Smith	**L_230**
Hayes	**L_155**

borrower

假设由于某种原因造成帐目不符

Inner Join
loan $\bowtie$ ***borrower***

loan_number	*branch_name*	*amount*	*customer_name*
L_170	Downtown	3000	Jones
L_230	Redwood	4000	Smith

Left Out Join
loan ⟕ ***borrower***

loan_number	*branch_name*	*amount*	*customer_name*
L_170	Downtown	3000	Jones
L_230	Redwood	4000	Smith
L_260	Perryridge	1700	***null***

Right Out Join
loan ⟖ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L_170	Downtown	3000	Jones
L_230	Redwood	4000	Smith
L_155	***null***	***null***	Hayes

The concept of Join types

Full Outer Join

loan ⟗ ***borrower***

loan_number	*branch_name*	*amount*	*customer_name*
L_170	Downtown	3000	Jones
L_230	Redwood	4000	Smith
L_260	Perryridge	1700	*null*
L_155	***null***	***null***	Hayes

(4) Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does NOT exist.
- The result of any arithmetic expression involving *null* is *null*.
- Aggregate functions simply ignore null values
  - Is an arbitrary decision. Could have returned null as result instead.
  - We follow the semantics of SQL in its handling of null values
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same
  - Alternative: assume each null is different from each other
  - Both are arbitrary decisions, so we simply follow SQL

Null Values

- **Comparisons with null values return the special truth value *unknown***
  - If *false* was used instead of *unknown*, then *not (A < 5)*
    would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* or *true*) = *true*,
    (*unknown* or *false*) = *unknown*
    (*unknown* or *unknown*) = *unknown*
  - AND: (*true* and *unknown*) = *unknown*,
    (*false* and *unknown*) = *false*,
    (*unknown* and *unknown*) = *unknown*
  - NOT: (not *unknown*) = *unknown*
  - In SQL "*P* is unknown" evaluates to true if predicate *P* evaluates to *unknown*
- **Result of select predicate is treated as *false* if it evaluates to *unknown***

(5) Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only **whole** tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.

Deletion Examples

- Delete all **account** records in the **Perryridge** branch.

$$account \leftarrow account - \sigma_{\,branch\text{-}name\, =\, \text{“Perryridge”}}\,(account)$$

- Delete all **loan** records with **amount** in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{\,amount\, \geq\, 0 \text{ and } amount\, \leq\, 50}\,(loan)$$

- Delete all **accounts** at branches located in **Needham**.

$$r_1 \leftarrow \sigma_{\,branch\text{-}city\, =\, \text{“Needham”}}\,(account \bowtie branch)$$

$$r_2 \leftarrow \Pi_{branch\text{-}name,\, account\text{-}number,\, balance}\,(r_1)$$

$$r_3 \leftarrow \Pi_{\,customer\text{-}name,\, account\text{-}number}\,(r_2 \bowtie \text{depositor})$$

$$account \leftarrow \text{account} - r_2$$

$$depositor \leftarrow \text{depositor} - r_3$$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- in relational algebra, an insertion is expressed by:

  $r \leftarrow r \cup E$

  where *r* is a relation and *E* is a relational algebra expression.
- The insertion of a **single** tuple is expressed by letting *E* be a **constant relation containing one tuple**.

Insertion Examples

- Insert information in the database specifying that Smith has \$1200 in account A-973 at the Perryridge branch.

$$\textit{account} \leftarrow \textit{account} \cup \{(\text{"Perryridge"}, \text{A-973}, 1200)\}$$

$$\text{depositor} \leftarrow \textit{depositor} \cup \{(\text{"Smith"}, \text{A-973})\}$$

- Provide as a gift for all **loan** customers in the **Perryridge** branch, a \$200 savings account. Let the loan number serve as the account number for the new savings account.

$$r_1 \leftarrow (\sigma_{\textit{branch-name} = \text{"Perryridge"}}(\textit{borrower} \bowtie \text{loan}))$$

$$\textit{account} \leftarrow \textit{account} \cup \Pi_{\textit{branch-name}, \textit{loan-number}, 200}(r_1)$$

$$\text{depositor} \leftarrow \textit{depositor} \cup \Pi_{\textit{customer-name}, \textit{loan-number}}(r_1)$$

Updating

- A mechanism to change a value in a tuple without changing *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F_1, F_2, \ldots, F_l,} (r)$$

- Each F_i is either
  - the *i*th attribute of *r*, if the *i*th attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of *r*, which gives the new value for the attribute

Update Examples

- **Make interest payments by increasing all balances by 5 percent.**

  $$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.05}(account)$$

  where *AN*, *BN* and *BAL* stand for *account-number*, *branch-name* and *balance*, respectively.

- Pay all accounts with balances over $10,000 6 percent interest

  and pay all others 5 percent

  $$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.06}(\sigma_{BAL\ >\ 10000}(account)) \cup \Pi_{AN,\ BN,\ BAL\ *\ 1.05}(\sigma_{BAL\ \leq\ 10000}(account))$$

End of Chapter 2

7th Edition: 2.6; 2.7; 2.14; 2.15