21121350
Database System

Lecture 2: Relational Model

Lu Chen (陈璐)
College of Computer Science
Zhejiang University
Spring & Summer 2023
luchen@zju.edu.cn/18868818726

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 the relational model.
- A relation is a table with rows and columns.
- The major advantages of the relational model are its straightforward 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

A relation for **instructor**

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

- Cf.: relationship and relation
  - A relationship is an association among several entities.
  - A relation is the mathematical concept, referred to as a table.

Entity set and relationship set ←→ real world

Relation --- table,   tuple --- row ←→ machine world

Outline

- Structure of Relational Databases
- Fundamental Relational-Algebra Operations
- Additional Relational-Algebra Operations
- Extended Relational-Algebra Operations
- Modification of the Database

Basic Structure

- Formally, given sets $D_1, D_2, \ldots, D_n$ $(D_i = a_{ij} \mid_{j=1\ldots k})$,
  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 $a_{ij} \in D_i$ $(i \in [1, n])$.
- E.g.,

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 = 学生集合 = {李勇, 刘晨, 王名}

则 D_1 X D_2 X 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),

	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

反映了特定意义的子集：

sup-spec-stud

张清玫	计算机	李勇
张清玫	计算机	刘晨
刘逸	信息	王名

Example of Relation

- If

  customer-name = {Jones, Smith, Curry, Lindsay}

  customer-street = {Main, North, Park}

  customer-city = {Harrison, Rye, Pittsfield}

  Then *r* = {(Jones, Main, Harrison),
  (Smith, North, Rye),
  (Curry, North, Rye),
  (Lindsay, Park, Pittsfield)}

  is a relation over *customer-name* x *customer-street* x *customer-city.*

  (total 36 tuples)

A Cartesian product

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, i.e., indivisible (--- 1st NF, 关系理论第一范式)
  - 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 now ignore the effect of null values, and consider their effect later.

Concepts about Relation

- A relation is concerned with two concepts: relation schema and relation instance.

- The relation schema describes the structure of the relation.
  - E.g., *Student-schema* = (*sid*: string, *name*: string, *sex*: string, *age:* int, *dept*: string) or
    Student-schema = (*sid*, *name*, *sex*, *age*, *dept*)
- 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

Relation Schema

- Assume $A_1, A_2, \ldots, A_n$ are attributes
- Formally expressed:

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

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

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

  E.g., *instructor*(*instructor-schema*)

  = *instructor*(*ID*, *name*, *dept_name*, *salary*)

Relation Instance

- The current values (i.e., *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)

instructor

ID	*name*	*dept_name*	*salary*
12121	Wu	Finance	90000
22222	Einstein	Physics	95000
33456	Gold	Physics	87000
83821	Brandt	Comp. Sci.	92000

tuples
(or rows)

- Let a tuple variable *t* be a tuple, then *t*[*name*] denotes the value of *t* on the *name* attribute.

The Properties of Relation

- The order of tuples is irrelevant (i.e., tuples may be stored in an arbitrary).
- No duplicated tuples in a relation.
- Attribute values are atomic.

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

Database

- A database consists of multiple relations

- Information about an enterprise (e.g., university) is broken up into parts
  - Instructor
  - Student
  - Advisor

- Bad design:

  univ (*instructor -ID, name, dept_name, salary, student_Id*, ..)

  results in
  - Repetition of information (e.g., two students have the same instructor)
  - The need for null values (e.g., represent an student with no advisor)

- Normalization theory (Chapter 7) deals with how to design "good" relational schemas

Key (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 $r(R)$
  - E.g., both {*ID*} and {*ID*, *name*} are superkeys of the relation ***instructor***.
- K is a candidate key (候选码) if K is minimal superkey.
  - E.g., {*ID*} is a candidate key for the relation ***instructor***, since it is a superkey and no any subset.
- K is a primary key (主码), if K is a candidate key and is defined by user explicitly.
  - Primary key is usually marked by underline.

Foreign Key (外键/外码)

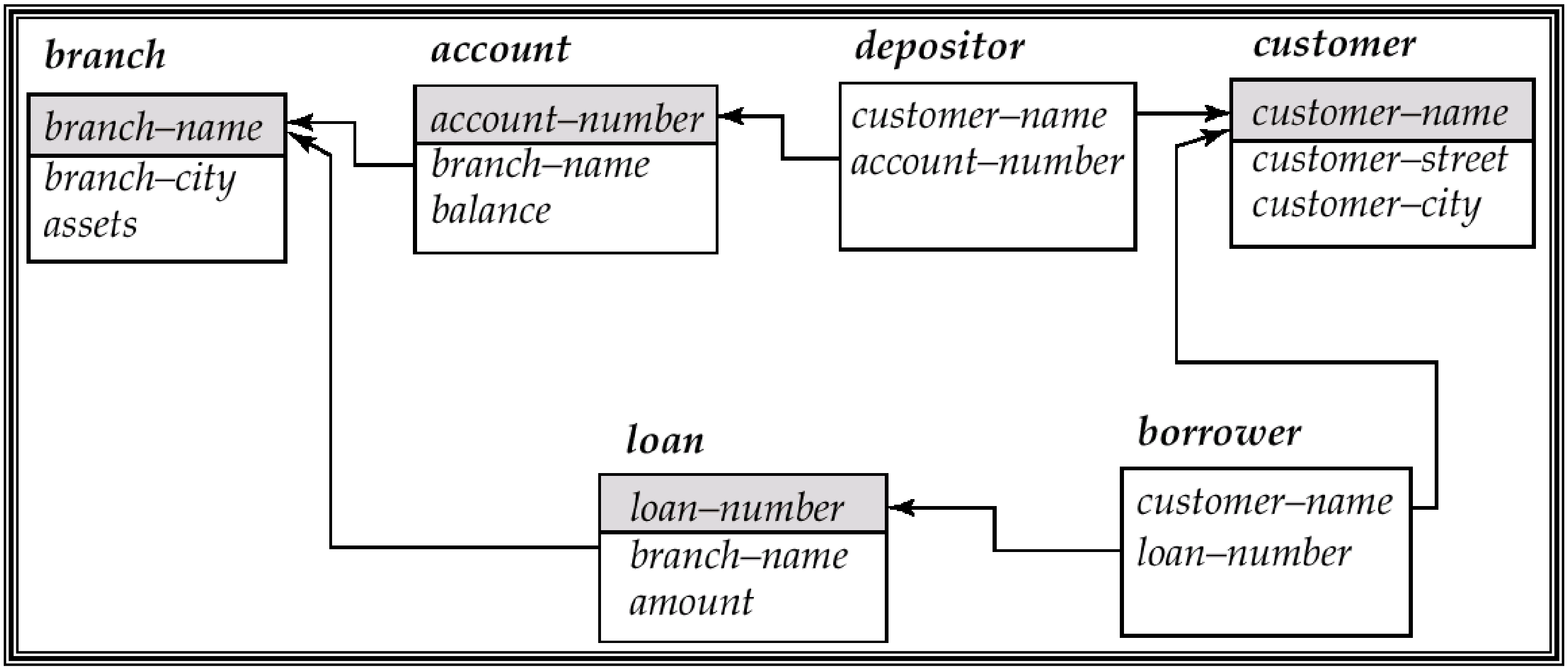
- Assume there exists relations *r* and *s*: *r*(<u>*A*</u>, *B*, *C*), *s*(<u>*B*</u>, *D*), we can say that attribute *B* in relation *r* is foreign key referencing *s*, and *r* is a referencing relation (参照关系), and *s* is a referenced relation (被参照关系).
  - E.g.1: 学生(<u>学号</u>, 姓名, 性别, 专业号, 年龄) --- 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.
  - E.g.2: Account(<u>account-number</u>, branch-name, balance) --- referenced relation
    depositor (<u>customer-name</u>, account-number) --- referencing 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**。

- Primary key and foreign key are integrated constraints.

Schema Diagram

- Schema diagram for the banking enterprise



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

  Not required!!

- Pure languages form underlying basis of query languages that people use.

Outline

- Structure of Relational Databases
- Fundamental Relational-Algebra Operations
- Additional Relational-Algebra Operations
- Extended Relational-Algebra Operations
- Modification of the Database

Fundamental Relational-Algebra Operations

- 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 return a new relation as a result.

Example of Select Operation

Relation *r* =

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

$\sigma_{A = B}(r) = ?$

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

A	*B*	*C*	*D*
β	β	23	10

Note that, the selection conditions need to aim at the attribute values of the same tuple, when we conduct section operation.

Select Operation Formalization

- Notation: $\sigma_p(r)$
- 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: $=$, $\neq$, $>$, $\geq$, $<$, $\leq$
- E.g.,

  $$\sigma_{\textit{branch-name}=\text{‘}\textit{Perryridge}\text{’}}(\textit{account})$$

Example of Project Operation

Relation r =

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

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

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

A	C
α	1
α	1
β	1
β	2

$\Rightarrow$

A	C
α	1
β	1
β	2

Project Operation Formalization

- Notation:

  $$\Pi_{A1,\, A2,\, \ldots,\, Ak}(r)$$

  where $A_1, \ldots A_k$ 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 *branch-name* attribute of *account*

  $$\Pi_{account\text{-}number,\ balance}\ (account)$$

Example of Union Operation

Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cup s$:

A	B
α	1
α	2
β	1
β	3

Union Operation Formalization

- 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：
  - *r* and *s* must have the same arity (i.e., the same number of attributes)
  - The attribute domains must be compatible
- E.g., Find all customers with either an account or a loan

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

Example of Set Difference Operation

Relations *r*, *s*:

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

r – *s*:

A	B
α	1
β	1

Set Difference Operation Formalization

- 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.

Example of Cartesian-Product Operation

Relations *r*, *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

Cartesian-Product Operation Formalization

- 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 (i.e., $R \cap S = \emptyset$).
- If attributes of $r(R)$ and $s(S)$ are not disjoint, then renaming for attributes must be used.
- E.g.,

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

Composition of Operations

- Can build expressions using multiple operations.
- E.g., $\sigma_{A = C}(r \times s)$

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

$r \times s =$

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

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.
- E.g.,

$$\rho_x(E)$$

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*

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*)

The *loan* Relation

loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The *borrower* Relation

customer-name	*loan-number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

The *account* Relation

account-number	*branch-name*	*balance*
A-101	Downtown	500
A-215	Mianus	700
A-102	Perryridge	400
A-305	Round Hill	350
A-201	Brighton	900
A-222	Redwood	700
A-217	Brighton	750

The *depositor* Relation

customer-name	*account-number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

customer

customer-name	*customer-street*	*customer-city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

branch

branch-name	*branch-city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

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 (Cont.)

- 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 (Cont.)

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

Query 1: $\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{'Perryridge'}}(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times loan)))$

Query 2: $\Pi_{customer\text{-}name}(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times (\sigma_{branch\text{-}name=\text{'Perryridge'}}(loan))))$

Query 2 is better.

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

Example Queries (Cont.)

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

Query 1: $\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{'Perryridge'}}$
$(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times loan))) -$
$\Pi_{customer\text{-}name}(depositor)$

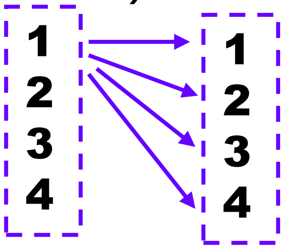
Query 2: $\Pi_{customer\text{-}name}(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}$
$(borrower \times (\sigma_{branch\text{-}name=\text{'Perryridge'}}(loan)))) -$
$\Pi_{customer\text{-}name}(depositor)$

Query 2 is better.

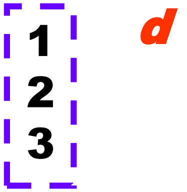
loan(*loan-number*, *branch-name*, *amount*)
borrower(*customer-name*, *loan-number*)
depositor(*customer-name*, *account-number*)

Example Queries (Cont.)

- Example 7: Find the largest account balance (i.e., self-comparison).
  - Step 1: Rename *account* relation as *d*.
  - Step 2: Find the relation including all balances except the largest one.

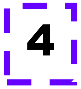


$$\Pi_{account.balance}(\sigma_{account.balance<d.balance}(account \times \rho_d(account)))$$



  - Step 3: Find the largest account balance.

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



account(*account-number*, *branch-name*, *balance*)

Outline

- Structure of Relational Databases
- Fundamental Relational-Algebra Operations
- Additional Relational-Algebra Operations
- Extended Relational-Algebra Operations
- Modification of the Database

Additional Relational-Algebra Operations

- Four basic operators
  - Set intersection　　交
  - Natural join　　自然连接
  - Division　　除
  - Assignment　　赋值

- We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

Example of Set-Intersection Operation

Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cap s =$

A	B
α	2

Set-Intersection Operation Formalization

- Notation: $r \cap s$
- Defined as:

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

- Assume:
  - r and s have the same arity.
  - attributes of r and s are compatible.
- Note: $r \cap s = r - (r - s)$

Natural Join Operation Formalization

- Notation: $r \bowtie s$
- Example: $R = (A, B, C, D)$, $S = (B, D, E)$
  - Result schema of the 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.

Example of Natural Join Operation

Relations *r*, *s*:

r

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

s

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

$r \bowtie s =$

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

注意：(1) *r*, *s*必须含有共同属性(名和域都对应相同); (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 Operation Formalization

- Notation: $r \bowtie_{\theta} s$
  where θ is the predicate on attributes in the schema.
- Theta join: $r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$
- Theta Join is the extension to the Nature Join.

Division Operation

- Suited to queries that include the phrase “for all”.
- In fact, it determines whether a collection contains another collection.

例：查询选修了所有课程的学生的学号。

enrolled

Sno	*Cno*
95001	1
95001	2
95001	3
95002	2
95002	3

÷

course

Cno
1
2
3

=

Sno
95001

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

Enrolled(*sno*, *cno*, *grade*)
Course(*cno*, *cname*, *credits*)

Division Operation Formalization

- Notation: $r \div s$

- Let r and s be relations on schemas R and S, respectively, where $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $S = (B_1, \ldots, B_n)$. Then, the result of $r \div s$ is a relation on the schema $R - S = (A_1, \ldots, A_m)$ and $r \div s = \{t \mid t \in \prod_{R-S}(r) \wedge \forall\, u \in s(tu \in r)\}$.

- Note that $\prod_{R-S}(r)$ encloses the result of $r \div s$, and meanwhile, the union of the tuple(s) t and all the tuples in s is covered by r (i.e., 商来自于$\prod_{R-S}(r)$, 并且其元组t与s所有元组的拼接被r覆盖).

Example of Division Operation

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

$r \div s =$

A
α
β

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

Another Example of Division Operation

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

r :

	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	

$r \div s =$

A	B	C
α	b	γ
γ	a	γ

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

Example 1: Compute $Q = R \div S$

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

Example 2: 从SC表中查询至少选修1号课程和3号课程的所有学生号码。

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

$$\Pi_{Sno,\ Cno}(sc) \div K$$

Division Operation Characteristic

- Property/Characteristic
  - 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 $S \subseteq R$, then $r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$.
- To see why
  - $\Pi_{R-S,S}(r)$ only 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 $u \in s$, $tu \notin r$.

r

A	B
a1	b1
a2	b1
a1	b2
a2	b2
a3	**b1**

s

B
b1
b2

q

A
a1
a2

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

$$
\begin{aligned}
&temp1 \leftarrow \Pi_{R-S}(r) \\
&temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r)) \\
&result = temp1 - temp2
\end{aligned}
$$

  - 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: $\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{'Downtown'}}(depositor \bowtie account)) \cap$
$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{'Uptown'}}(depositor \bowtie account))$

Query 2: $\Pi_{customer\text{-}name,\ branch\text{-}name}(depositor \bowtie account) \div$
$\rho_{temp(branch\text{-}name)}(\{(\text{'Downtown'}), (\text{'Uptown'})\})$

depositor(*customer-name*, *account-number*)
account(*account-number*, *branch-name*, *balance*)

Example Queries (Cont.)

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

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

branch(*branch-name*, *branch-city*, *assets*)
depositor(*customer-name*, *account-number*)
account(*account-number*, *branch-name*, *balance*)

- 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*, *credits*), 选课信息 *enrolled*(*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：$\Pi_{Cno}(Course)$
  - 找出选修了全部课程的学生的学号：$\Pi_{\text{Sno, Cno}}(enrolled) \div \Pi_{\text{cno}}(Course)$
  - 与*student*表自然连接（连接条件*Sno*）获得学号、姓名：$(\Pi_{\text{Sno, Cno}}(enrolled) \div \Pi_{\text{cno}}(Course)) \bowtie \Pi_{\text{Sno,Sname}}(student)$

Summary

- Union, set difference, Set intersection 为双目、等元运算
- Cartesian product, Natural join, Division 为双目运算
- Project, select 为单运算对象 (i.e., 单目运算)

- The priority of operations is as follows:
  - Project
  - Select
  - Cartesian Product (times)
  - Join, division
  - Intersection
  - Union, difference

Outline

- Structure of Relational Databases
- Fundamental Relational-Algebra Operations
- Additional Relational-Algebra Operations
- Extended Relational-Algebra Operations
- Modification of the Database

Extended Relational-Algebra Operations

- Extended relational-algebra operators
  - Generalized Projection
  - Aggregate Functions
  - Outer Join

Generalized Projection

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

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

  where *E* is any relational-algebra expression, and each of F_1, F_2, …, F_n are arithmetic expressions involving constants and attributes in the schema of *E*.

- Given a relation *credit-info*(*customer-name*, *limit*, *credit-balance*), find how much more each person can spend:

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

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

E.g., 求平均存款余额

$$\mathcal{g}_{\text{avg}(\textit{balance})}(\textit{account})$$

- Aggregate operation in relational algebra

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

where E is any relational-algebra expression, G_1, G_2 …, G_n is a list of attributes on which to group (can be empty), each F_i is an aggregate function, and each A_i is an attribute name.

Example of Aggregate Operation

Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	8
β	α	14

r

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

avg-c
9

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

A	*sum-c*
α	14
β	22

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

B	*avg-c*
α	10.5
β	7.5

Example of Aggregate Operation (Cont.)

- 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

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

Sum-balance

branch-name	
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Function

- 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

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

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.
    - We shall study precise meaning of comparisons with nulls later

Outer Join – Example

- Relation *instructor1*

ID	*name*	*dept_name*
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches1*

ID	*course_id*
10101	CS-101
12121	FIN-201
76766	BIO-101

Outer Join – Example

- Join

instructor $\bowtie$ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- Left Outer Join

instructor $⟕$ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

Outer Join – Example

- Right Outer Join
  instructor ⟖ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

- Full Outer Join
  instructor ⟗ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

Outer Join using Joins

- Outer join can be expressed using basic operations
  - e.g. $r ⟕ s$ can be written as

    $(r \bowtie s)\ \cup\ (r - \Pi_R(r \bowtie s)\ \times \{(null, \ldots, null)\}$

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
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same

Null Values

- Comparisons with null values return the special truth value: *unknown*

Not(A $>$ 5)

A $\leq$ 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*

Outline

- Structure of Relational Databases
- Fundamental Relational-Algebra Operations
- Additional Relational-Algebra Operations
- Extended Relational-Algebra Operations
- Modification of the Database

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.
- It can delete only whole tuples; cannot delete values on some 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

- E.g.1: Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\text{-}name\ =\ \text{'Perryridge'}}(account)$$

- E.g.2: Delete all loan records with amount in the range of 0 to 50.

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

- E.g.3: Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{branch\text{-}city\ =\ \text{'Needham'}}(account \bowtie branch)$$
$$r_2 \leftarrow \Pi_{branch\text{-}name,\ account\text{-}number,\ balance}(r_1)$$
$$r_3 \leftarrow \Pi_{customer\text{-}name,\ account\text{-}number}(r_2 \bowtie depositor)$$
$$account \leftarrow account - r_2$$
$$depositor \leftarrow depositor - r_3$$

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

- E.g.1: Insert information in the database specifying that Smith has \$1200 in account A-973 at the Perryridge branch.

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

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

- E.g.2: 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.

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

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

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

Update

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

$$r \leftarrow \prod_{F1,\ F2,\ \ldots,\ Fl}(r)$$

where each F_i is either the ith attribute of r, if the ith 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

- E.g.1: Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \Pi_{account\text{-}number,\ branch\text{-}name,\ balance\ *\ 1.05}(account)$$

- E.g.2: Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent.

$$account \leftarrow \Pi_{account\text{-}number,\ branch\text{-}name,\ balance\ *\ 1.06}(\sigma_{balance\ >\ 10000}(account))$$
$$\cup\ \Pi_{account\text{-}number,\ branch\text{-}name,\ balance\ *\ 1.05}(\sigma_{balance\ \leq\ 10000}(account))$$

Q & A

Your questions and suggestions are expected for me.

Thanks a lot!

Questions?

Exercises: 2.12, 2.14, and 2.15 (see Page 62)