数据库原理

Chp 2
Introduction to Relational Model

王朝坤
清华大学软件学院
2025年/春

关系数据库

- **Relational Databases**
  - **A relational database consists of a collection of tables/relations**

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

(a) The *instructor* table

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

(b) The *department* table

Department Table

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

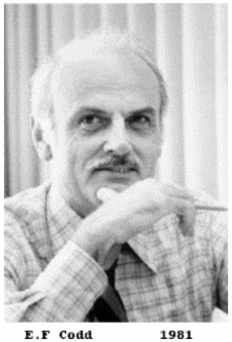
How to model the table with mathematics?

Relation

关系数据模型—E.F. Codd发明

- All the data is stored in relations.
- 关系数据模型 = 关系 + 关系代数 + 约束

Columns 列



1981年获得 ACM图灵奖

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

Rows 行

(a) The *instructor* table

Main Contents

1. **关系数据模型***
2. 关系代数基础*
3. 元组关系演算
4. 域关系演算
5. 小结

群聊: 数据库原理2025春

该二维码7天内(3月4日前)有效，重新进入将更新

Relation

- Domain
  - the set of permitted values with the same type
  - The special value **null(空值)** is a member of every **domain**, means “unknown” or “does not exist ”
- given sets D_1, D_2, …. $D_{n,}$ a **relation** r is a subset of $D_1 \times D_2 \times \ldots \times D_n$
  - Thus a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where $a_i \in D_i$
  - n is called **arity** (中文 “元”),

    n-tuple 即 n **元组**

Example of Relation

3 domains:

dept-name = {Biology, Finance, History, Music, Physics}

building = {Watson, Painter, Packard}

budget = {90000, 120000, 70000, 80000}

1 relation over ***dept-name* x *building* x *budget***:

Then *r* = { (Biology, Watson, 90000),
(Finance, Painter, 120000),
(Music, Packard, 80000),
(Physics, Watson, 70000)}

Tuple & Tuple Variable

- An element of relation *r* is a **tuple,** represents a *row* in a table
- **A tuple variable** *t* stands for a tuple
  - ***t[dept-name]*** denotes the value of t on the ***dept-name*** attribute
  - **t[1]** denotes value the first attribute of t(viz. positional notation)
- A **relation** is a set of **tuples** with the same type

Relations are Unordered

tuples are irrelevant

Tuples may be stored in an arbitrary order

E.g. *account* relation with unordered tuples

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

=

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

Relation Schema

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

  E.g. ***Department-schema*** – 表的名字

  (*dept-name,*

  building,

  budget)

- ***r*(*R*)** is a **relation instance** of *relation schema R*

  E.g. *deptartment* (***Department-schema***)

Schema and Instance

- **Relation Schema** 《——》 **Variable name**
  - **account(a-number, b-name, balance)** ***length***

- **Relation Instance** 《——》 **Variable Value** ***123***

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

The value of a given variable may change with time; similarly the contents of a relation instance may change with time as the relation is updated.

Superkey/超键

- Let $K \subseteq R$
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of ***each possible*** relation $r(R)$
  - **所有可能/合法关系实例**

- Example: {*customer-name, customer-street*} and {*customer-name*}
  are both superkeys of *Customer*, if no two customers can possibly have the same name

题 2.4 - 书中第60页

In the instance of instructor shown in Figure 2.1, no two instructors have the same name. From this, can we conclude that name can be used as a superkey (or primary key) of 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

Figure 2.1 The *instructor* relation.

Candidate Key & Primary Key

- *K* is a **candidate key/候选键** if *K* is minimal
  - {*customer-name*} is a candidate key for *Customer*

- Although several candidate keys may exist, one of the candidate keys is selected to be the **primary key/主键**
  - The attributes of the primary key is called **prime attributes(主属性)**
  - **主键用下划线标识**

Account Table

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

假设上面这个表是所有可能实例的全集，那么它的Superkey，Candidate Key ， Primary Key?

Foreign Key/外键

- Let r1(R1) and r2(R2) be relations，and R1 **with primary key K1**，
  - The subset α of R2 is a foreign key referencing K1 in relation r1. For every t2 in r2 there must be a tuple t1 in r1 such that t1[K1] = t2[α]

  – r2 is called **the referencing relation/施引关系** of the foreign key dependency, and r1 is called **the referenced relation/被引关系** of the foreign key

  – 用箭头⟶标识，从**施引关系到被引关系**

Foreign Key Example

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

***instructor*(*<u>ID</u>*, *name*, *dept_name*, *salary*)**

- *dept_name* in ***instructor*** is a foreign key from *instructor* referencing *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

Figure 2.5 The *department* 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

Figure 2.1 The *instructor* relation.

Which is the referenced relation/被引关系 of this foreign key ?

判断对错

section **(*course_id, sec_id, semester, year,* *building, room number, time slot id*)**

teaches **(*ID, course_id, sec_id, semester, year*)**

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 2.6 The *section* relation.

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Figure 2.7 The *teaches* relation.

(course_id, sec_id, semester, year) in *section*

is a foreign key from *section* referencing *teaches* ？

反过来呢?

Relational Database

- A **database schema/数据库模式** consists of a set of relation schemas
- A **database instance/数据库实例** consists of a set of relations
- In relational database, information about an enterprise is broken up into parts, with each relation **storing** one part of the information

  E.g.

  account : stores information about accounts

  depositor : stores information about which customer owns which account

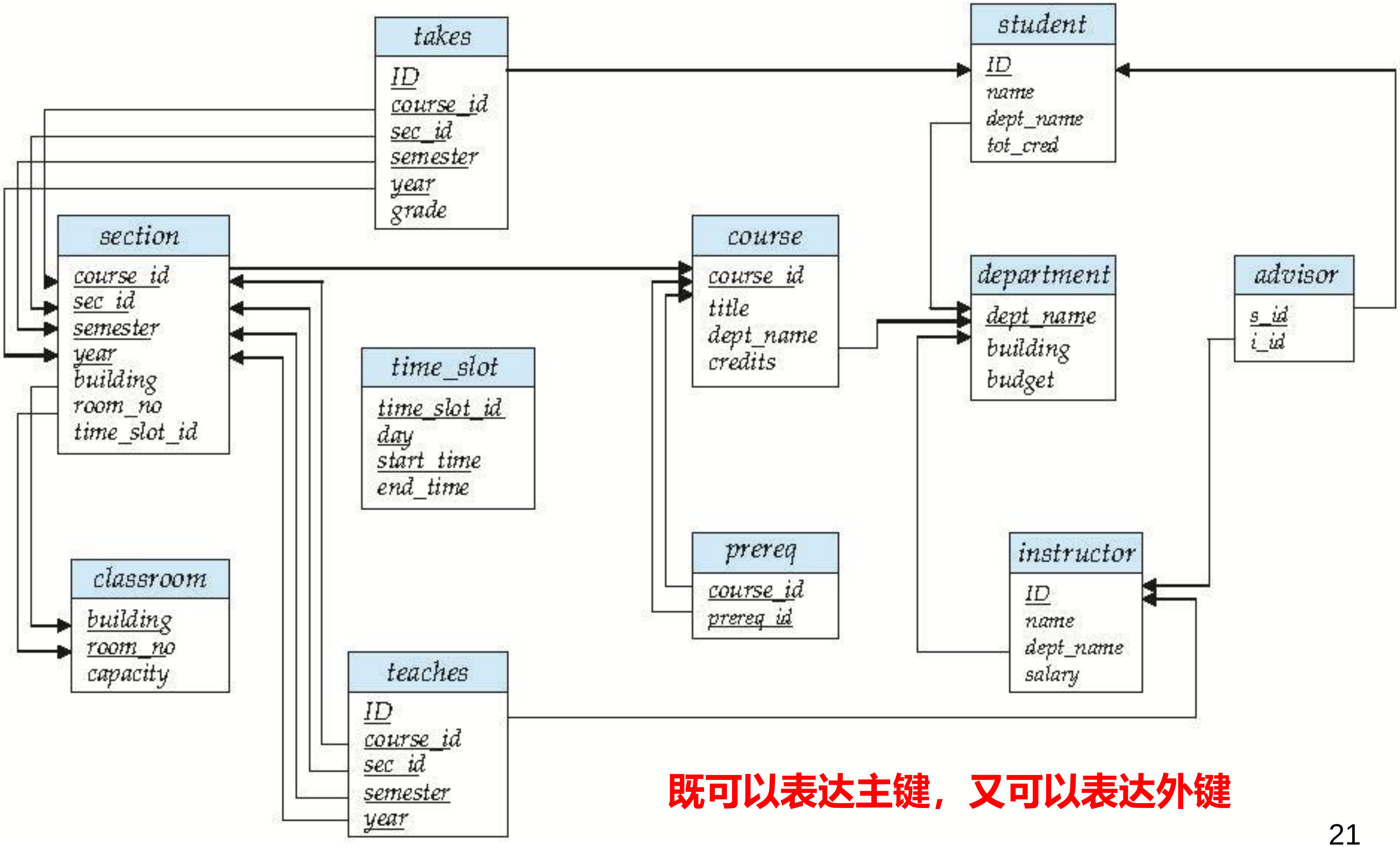
  customer : stores information about customers

Schema of the University Database

1. *classroom(<u>building</u>, <u>room_number</u>, capacity)*
2. *department(<u>dept_name</u>, building, budget)*
3. *course(<u>course_id</u>, title, dept_name, credits)*
4. ***instructor(<u>i_ID</u>, name, dept_name, salary)***
5. *section(<u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, building, room number, time_slot_id)*
6. *teaches(<u>i_ID</u>, <u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>)*
7. *student(<u>s_ID</u>, name, dept_name, tot_cred)*
8. *takes(<u>s_ID</u>, <u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, grade)*
9. *advisor(<u>s_ID</u>, <u>i_ID</u>)*
10. *Time_slot(<u>time_slot_id</u>, <u>day</u>, <u>start_time</u>, end_time)*
11. *prereq(<u>course_id</u>, <u>prereq_id</u>)*

这样表达关系模式的不足：无法表达外键关联

大学数据库模式图



既可以表达主键，又可以表达外键

Basic Concepts

- **Domain (域)，** **Relation (关系),** **Attribute(属性),** **Tuple(元组)**
- **Superkey(码)，** **Candidate Key(候选码),** **Primary Key (主码)**
- **Relation Schema (关系模式),** **Relation (instance) (关系实例)，** **Relational Database (关系数据库)**

Main Contents

1. 关系数据模型*
2. **关系代数基础***
3. 元组关系演算
4. 域关系演算
5. 小结

What is Relational Algebra?

- 集合 + 运算 = 代数系统
- Relations + Operators = Relational algebra
  - **The operators take one or more relations as inputs and give a new relation as a result**
- **Relation algebra can be used as a *query language* for relations**

Query Language

- **A language in which a user specifies the query from the database**

- Procedural Language
  - A sequence of operations
  - **Relational algebra**

- Non-Procedural Language
  - Description of the desired information
  - **Relational calculus**

6 Basic Operators

① Union	(并)
② set difference	(差)
③ Cartesian product	(笛卡尔积)
④ Select	(选择)
⑤ Project	(投影)
⑥ Rename	(换名)

① Union Operation

- 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*, *s* must have the *same* **arity** (same number of attributes)
  - 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*)

Union – Example

- 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*

② Set Difference Operation

- Notation $r - s$
- Defined as:

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

- 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

Set Difference – 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

③ Cartesian-Product Operation

- Notation *r* **x** *s*
- Defined as:

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

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

Cartesian-Product - Example

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

④ Select Operation

- Notation: $\sigma_p(r)$
- p is called the **selection predicate(选择谓词)**
- Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

  – Where p is a formula in **propositional calculus(命题逻辑)** 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: $=, \neq, >, \geq, <, \leq$

Select – Example

- Relation *r*

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

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

A	B	C	D
α	α	1	7
β	β	23	10

⑤ Project Operation

- Notation:

$$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

where $A_1, A_2, \ldots, A_k$ are attribute names and r is a relation

  - 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

Project – 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

⑥ Rename Operation

Notation:

$\rho_{X}(E)$

returns the result of expression *E* under the name *X*

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

$\rho_{X(A_1, A_2, \ldots, A_n)}(E)$

returns the result of expression *E* under the name *X*, and with the attributes renamed to $A_1, A_2, \ldots, A_n$

Rename Example

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions
  - Let E is a relational algebra expressions

    $\boldsymbol{\rho_X(E)}$ return the result relation of *E as X* relation

- When the Cartesian product of a relation with itself is desired
  - Let r_1 is relation on the relation schema R_1(son,father)

    $\Pi_{r_1.father, r_2.son} (\sigma_{r_1.son=r_2.father}(r_1 X \rho_{r_2}(r_1)))$

Relational Algebra Expression

- A relation in the database is a Relational Algebra Expression
- A constant relation is a Relational Algebra Expression
  – {(A-101, Downtown, 500) (A-215, Mianus, 700)},

Relational Algebra Expression(c1)

- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - *(并)* $E_1 \cup E_2$
  - *(差)* $E_1 - E_2$
  - *(笛卡尔积)* $E_1 \times E_2$
  - *(选择)* $\sigma_p(E_1)$, P is a predicate on attributes in E_1
  - *(投影)* $\Pi_S(E_1)$, S is a list consisting of some of the attributes in E_1
  - *(换名)* $\rho_x(E_1)$, x is the new name for the result of E_1

Example 1: $\sigma_{A=C}(r \times s)$

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	19	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	10	a
β	2	β	20	b

Example 2：Bank Database

- *Branch 营业所*
  (branch-name, branch-city, assets)
- *Customer 客户*
  (id, customer-name, customer-street, customer-city)
- *Account 存款明细*
  (account-number, branch-name, balance)
- *Loan 贷款明细*
  (loan-number, branch-name, amount)
- *Depositor 存款*
  (id, account-number)
- *Borrower 贷款*
  (id, loan-number)

Queries 1/2

1、Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer\text{-}name}\,(\sigma_{Borrower.id\,=\,Customer.id}\,(Borrower\ X\ Customer)) \cup$$
$$\Pi_{customer\text{-}name}\,(\sigma_{Depositor.id\,=\,Customer.id}\,(Depositor\ \ X\ Customer))$$

2、Find the names of all customers who have a loan and an account at bank

$$\Pi_{customer\text{-}name}\,(\sigma_{Borrower.id\,=\,Customer.id}\,(Borrower\ X\ Customer)) \cap$$
$$\Pi_{customer\text{-}name}\,(\sigma_{Depositor.id\,=\,Customer.id}\,(Depositor\ \ X\ Customer))$$

Queries 3/4

3、Find the names of all customers who have a loan at the Perryridge branch

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“Perryridge”}\ \wedge\ Borrower.loan\text{-}number\ =\ Loan.loan\text{-}number\ \wedge\ Loan.id\ =Customer.id}\ (Borrower\ X\ Loan\ X\ Customer)))$$

4、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

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“Perryridge”}\ \wedge\ Borrower.loan\text{-}number\ =\ Loan.loan\text{-}number\ \wedge\ Loan.id\ =Customer.id}\ (Borrower\ X\ Loan\ X\ Customer)))\ -$$

$$\Pi_{customer\text{-}name}\ (\sigma_{Depositor.id\ =\ Customer.id}\ (Depositor\ \ X\ Customer))$$

Query 5

- Find the largest loan amount in the bank
  - Step 1: find loans that are less than some other loans (i.e. not maximum)

$$\Pi_{Loan.amount}(\sigma_{Loan.amount < L.amount}(Loan \times \rho_L(Loan)))$$

  - Step 2: Find the largest salary

$$\Pi_{amount}(Loan) - \Pi_{Loan.amount}(\sigma_{Loan.amount < L.amount}(Loan \times \rho_L(Loan)))$$

4个附加运算

① Set intersection (交)

② Natural join/Theta Join (自然/条件连接)

③ Division (除)

④ Assignment (赋值)

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

① 交集运算
Set-Intersection Operation

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

Set-Intersection - Example

- Relation *r*, *s*:

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

- $r \cap s$

A	B
α	2

➢ 求同时在关系 *r* 与关系 *s* 中出现的元组

② 自然连接运算 Natural-Join Operation

- Notation: r $\bowtie$ s
- Let *r* and *s* be relations on schemas *R* and *S* respectively
  - The Relation schema is $R \cup S$
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, a tuple *t* is added 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 – Example

- Relations r, s:

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

$r \bowtie s$

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

求 *r* 和 *s* 笛卡尔积中 *B* 属性值相等的集合，并消去同名属性 *B*

Natural-Join Operation (c1)

- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$

- Result schema = (A, B, C, D, E)
- $r \bowtie s$ is defined as:

$$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B\ =\ s.B\ \wedge\ r.D\ =\ s.D}\ (r\ \times\ s))$$

假如 r(R) 和 s(S) 没有公共属性, 那么

$r \bowtie s$ = ?

自然连接的泛化：条件连接 Condition-Join Operation

- Notation: $r \bowtie_C s$， 也被称为Theta Join
- $_C$ is a condition on attributes in $R \cup S$; result schema is the same as that of Cartesian Product. If $R \cap S \neq \emptyset$, some of these attributes must be renamed. And then condition $_C$ can refer to these attributes.
- 等价表示： $r \bowtie_C s = \sigma_C(r \times s)$

条件连接举例

- Relations *r*, *s*:

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

r

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

s

$r \bowtie_{B=F} s$

A	*B*	*C*	*D*	*F*	*s.D*	*E*
α	1	α	a	1	a	α
α	1	α	a	1	a	γ
α	1	γ	a	1	a	α
α	1	γ	a	1	a	γ
δ	2	β	b	2	b	δ

③ 除法运算-Division Operation

- Notation:

$$r \div s$$

- Suited to queries that include the phrase “for all”.
- 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

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

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

除法举例

Relations *r*, *s*:

A	*B*
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
β	2

r

B
1
2

s

关系 r 中 A 属性的值组成的集合，该值对应的 B 属性值的集合是关系 s 中 B 属性值的集合的超集

$Q = r \div s$:

A
α
β

QX s:

A	*B*
α	1
α	2
β	1
β	2

另一个除法的例子

Relations *r*, *s*:

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

r

D	E
a	1
b	1

s

$\prod_{R-S}(r)$

A	B	C
α	a	α
α	a	γ
β	a	γ
γ	a	γ
γ	a	β

$r \div s$:

A	B	C
α	a	γ
γ	a	γ

关系除法的应用例题

Table PilotSkills (被除数) is about the pilots and the planes they can fly, ***table Hangar*** (除数) is about planes in the hangar,

we want the names of the pilots who can fly every plane (quotient) in the hangar.

PilotSkills

pilot	plane
'Celko'	'C919'
'Higgins'	'H-6N Bomber'
'Higgins'	'J-35 Fighter'
'Higgins'	'C919'
'Jones'	'H-6N Bomber'
'Jones'	'J-35 Fighter'
'Smith'	'H-6K Bomber'
'Smith'	'H-6N Bomber'
'Smith'	'J-35 Fighter'
'Wilson'	'H-6K Bomber'
'Wilson'	'H-6N Bomber'
'Wilson'	'J-35 Fighter'
'Wilson'	'J-20 Fighter'

Hangar

plane
'H-6K Bomber'
'H-6N Bomber'
'J-35 Fighter'

用关系基本运算来表示除法

- 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 $u \in s$, $tu \notin r$.

除法的逆运算：笛卡尔积X（乘法）

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	19	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

④ 赋值运算-Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries

- Assignment must always be made to a temporary **relation variable**

Assignment Operation-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

3个扩展关系运算

① Generalized Projection (泛化投影)

② Aggregate Functions (聚集函数)

③ Outer Join (外连接)

These operations do add the power to the relational algebra

① 泛化投影 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***

Generalized Projection Example

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

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

- Naming the *GP* attributes

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

② 聚合运算 Aggregate Operations

$$G_1, G_2, \ldots, G_n \; \mathcal{g} \; F_1(A_1), F_2(A_2), \ldots, F_n(A_m) \; (E)$$

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

? How many attributes in the result schema

? How many tuples in the result

Aggregate Functions

- **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

MULTISET(BAG)

e.g.

(1, 2, 2, 3, …)

Aggregate Operation – Example

Given *account* Relation

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

- grouped by *branch-name*

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

branch-name	*balance*
Perryridge	1300
Brighton	1500
Redwood	700

- grouped by *none*

$$\mathcal{G}_{\,sum(balance)}\,(account)$$

balance
3500

Rename in Aggregation

- Result of aggregation does not have a name
  - Can use rename operation to give it a name

$\rho_{\mathbf{X}\ (\mathbf{A_1, A_2, \ldots, A_n})}(E)$

  - For convenience, using **as clause**

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

③ 外连接运算 - Outer Join

Relation *loan* & *borrower*

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	**Perryridge**	**1700**

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	**L-155**

Dangling Tuple

Natural Join

loan ⋈ *Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

左外连接 - Left Outer Join

Relation *loan* & *borrower*

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

Left 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*

右外连接 - Right Outer Join

Relation *loan* & *borrower*

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

- **Right Outer Join**

  loan $\rtimes$ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

全外连接 - Full Outer Join

Relation *loan* & *borrower*

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

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

外连接的涵义

- An extension of the join operation **that avoids loss of information**
- Computes the join and **then adds tuples from one relation that does not match tuples in the other relation to the result**
- Uses ***null*** values
  - *null* signifies that the value is unknown or does not exist

Null Values

- *null* signifies an unknown value or that a value does not exist

total
Null
Null
9
12

$$\Pi_{(A1+A2+A3)\ as\ totol}(R)$$

$$\mathcal{g}_{\ \boldsymbol{avg(A1),avg(A2),avg(A3)}}\ (R)$$

avg(A1)	avg(A2)	avg(A3)
3.50	3.00	2.50

A1	A2	A3
Null	Null	1
Null	2	2
3	3	3
4	4	4

- **The result of any arithmetic expression involving null is null**
- **Aggregate functions simply ignore null values**

Null Values (c1)

- **For duplicate elimination and grouping**

  null is treated like any other value, and two nulls are assumed to be the same

$$\boxed{\Pi_{A1}(R)}$$

- Comparisons with null values return the special value *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to   *A >= 5*

Null Values (c2)

- 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*

➢ **Result of select predicate is treated as *false* if it evaluates to *unknown***

Main Contents

1. 关系数据模型*
2. 关系代数基础*
3. **元组关系演算**
4. 域关系演算
5. 小结

元组关系演算的一般形式

{*t* | *P* (*t*) }　　非过程语言（ **Nonprocedural** ）

– **It is the set of all tuples *t* such that predicate *P* is true for *t***

– ***t* is a *tuple variable*, *t*[*A*] denotes the value of tuple *t* on attribute *A***

– $t \in r$ **denotes that tuple *t* is in relation *r***

– ***P* is a *formula* similar to that of the predicate calculus**

谓词公式

1. Set of attributes/属性 and constants/常量
2. Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
3. Set of connectives: and ($\wedge$), or (v), not ($\neg$)
4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:
   - $\exists\ t \in r\ (Q(t)) \equiv$ "there exists" a tuple t in relation r such that predicate $Q(t)$ is true
   - $\forall t \in r\ (Q(t)) \equiv Q$ is true "for all" tuples t in relation r

已知某银行数据库模式

- *Branch 营业所*
  (branch-name, branch-city, assets)
- *Customer 客户*
  (customer-name, customer-street, customer-city)
- *Account 存款明细*
  (account-number, branch-name, balance)
- *Loan 贷款明细*
  (loan-number, branch-name, amount)
- *Depositor 存款*
  (customer-name, account-number)
- *Borrower 贷款*
  (customer-name, loan-number)

课堂练习

- **Find the loan-number, branch-name, and amount for loans of over $1200**

$$\{t \mid t \in Loan \wedge t\,[amount] > 1200\}$$

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

$$\{t \mid \exists s \in \text{Loan}\ (t[loan\text{-}number] = s[loan\text{-}number] \\ \wedge s\,[amount] > 1200)\}$$

Notice that a relation on schema [*loan-number*] is implicitly defined by the query

课堂练习(c1)

- **Find the names of all customers having a loan, an account, or both at the bank**

$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name]) \lor \exists u \in depositor(t[customer\text{-}name] = u[customer\text{-}name])$

- **Find the names of all customers who have a loan and an account at the bank**

$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name]) \land \exists u \in depositor(t[customer\text{-}name] = u[customer\text{-}name])$

课堂练习(c2)

- **Find the names of all customers having a loan at the Perryridge branch**

$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name] \\ \wedge \exists u \in loan(u[branch\text{-}name] = \text{“Perryridge”} \\ \wedge\ u[loan\text{-}number] = s[loan\text{-}number]))\}$$

- **Find the names of all customers who have a loan at the Perryridge branch, but no account at any branch of the bank**

$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[\text{customer-name}] \\ \wedge \exists u \in loan(u[branch\text{-}name] = \text{“Perryridge”} \\ \wedge\ u[loan\text{-}number] = s[\text{loan-}number])) \\ \wedge \textbf{not } \exists v \in depositor\ (v[customer\text{-}name] = \\ t[\text{customer-name}])\ \}$$

课堂练习(c3)

- **Find the names of all customers having a loan from the Perryridge branch and the cities they live in**

$$\{t \mid \exists s \in \mathit{loan}(s[\mathit{branch\text{-}name}] = \text{"Perryridge"}$$
$$\wedge \exists u \in \mathit{borrower}\,(u[\mathit{loan\text{-}number}] = s[\mathit{loan\text{-}number}]$$
$$\wedge\; t[\mathit{customer\text{-}name}] = u[\mathit{customer\text{-}name}]$$
$$\wedge \exists v \in \mathit{customer}\,(u[\mathit{customer\text{-}name}] = v[\mathit{customer\text{-}name}]$$
$$\wedge\; t[\mathit{customer\text{-}city}] = v[\mathit{customer\text{-}city}])))\}$$

演算结果的关系模式?

loan ***(<u>loan-number</u>, branch-name, amount)***

borrower (<u>customer-name</u>, <u>loan-number</u>)

Customer(<u>customer-name</u>, customer-street, customer-city)

课堂练习(c4)

- **Find the names of all customers who have an account at all branches located in Brooklyn**

$\{t \mid \exists\ \mathbf{c} \in \mathbf{customer}\ (\boldsymbol{t}[\mathbf{customer\text{-}name}] = \mathbf{c}[\mathbf{customer\text{-}name}]) \wedge$

$\forall\ s \in branch(s[branch\text{-}city] = \text{“Brooklyn”} \Rightarrow$

$\exists\ u \in account\ (\ s[branch\text{-}name] = u[\text{branch-name}]$

$\wedge \exists\ d \in depositor\ (\ t[customer\text{-}name] = d[\text{customer-name}]$

$\wedge\ d[account\text{-}number] = u[\text{account-number}]\ )))\}$

表达式的安全性

- It is possible to write tuple calculus expressions that generate infinite relations

  UNSAFE

  $\{t \mid \neg\, t \in r\}$

- **To guard against the problem, we restrict the set of allowable expressions to safe expressions**

- **$\{t \mid P(t)\}$ is *safe* if every component of *t* appears in one of the relations, tuples, or constants that appear in *P***

Main Contents

1. 关系数据模型*
2. 关系代数基础*
3. 元组关系演算
4. **域关系演算**
5. 小结

域关系演算

Another non-procedural query language

- Each query is an expression of the form

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - P represents a formula similar to that of the predicate calculus

已知某银行数据库模式

- *Branch 营业所*
  (<u>branch-name</u>, branch-city, assets)
- *Customer 客户*
  (<u>customer-name</u>, customer-street, customer-city)
- *Account 存款明细*
  (<u>account-number</u>, branch-name, balance)
- *loan 贷款明细*
  (<u>loan-number</u>, branch-name, amount)
- *depositor 存款*
  (<u>customer-name</u>, <u>account-number</u>)
- *borrower 贷款*
  (<u>customer-name</u>, <u>loan-number</u>)

课堂练习

- **Find the *branch-name, loan-number,* and *amount* for loans of over $1200**

$$\{< l, b, a > \mid < l, b, a > \in loan \wedge a > 1200\}$$

- **Find the names of all customers who have a loan of over $1200**

$$\{< c > \mid \exists\, l, b, a\, (< l, b, a > \in loan \wedge < c, l > \in borrower \wedge a > 1200)\}$$

- **Find the names of all customers who have a loan from the Perryridge branch and the loan amount**

$$\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge \exists b (< l, b, a > \in loan \wedge b = \text{“Perryridge”}))\}$$

$$\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge < l, \text{“Perryridge”}, a > \in loan)\}$$

课堂练习(c1)

- **Find the names of all customers having a loan, an account, or both at the Perryridge branch**

$$\{< c > \mid \exists\, l\,(\{< c, l > \in borrower \\ \wedge \exists\, b,a(< l, b, a > \in loan \wedge b = \text{“Perryridge”})) \\ \vee \exists\, a(< c, a > \in depositor \\ \wedge \exists\, b,n(< a, b, n > \in account \wedge b = \text{“Perryridge”}))\}$$

- **Find the names of all customers who have an account at all branches located in Brooklyn**

$$\{< c > \mid \exists\, s,n\,(< c, s, n > \in \text{customer}) \wedge \\ \forall\, x,y,z(< x, y, z > \in branch \wedge y = \text{“Brooklyn”}) \Rightarrow \\ \exists\, a,b(< a, x, b > \in account \wedge < c,a > \in depositor)\}$$

表达式的安全性

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

is safe if all of the following hold:

1. **All values that appear in tuples of the expression are values from *dom*(*P*) (that is, the values appear either in *P* or in a tuple of a relation mentioned in *P*)**
2. **For every "there exists" subformula of the form $\exists\, x\, (P_1(x))$, the subformula is true if and only if $P_1(x)$ is true for a value x from $dom(P_1)$**
3. **For every "for all" subformula of the form $\forall_x\, (P_1(x))$, the subformula is true if and only if $P_1(x)$ is true for all values x from $dom\, (P_1)$**

综合练习

- Let R = (A, B) and S = (A, C), and let r(R) and s(S) be relations. Write relational-algebra expressions equivalent to the following domain-relational-calculus expressions:

1. $\{ < a > \mid \exists\, b\, (< a, b > \in r \land b = 17)\}$
2. $\{ < a, b, c > \mid < a, b > \in r \land < a, c > \in s\}$
3. $\{ < a > \mid \exists\, b\, (< a, b > \in r) \lor \forall\, c (\exists\, d\, (< d, c > \in s) \Rightarrow < a, c > \in s)\}$
4. $\{ < a > \mid \exists\, c\, (< a, c > \in s \land \exists\, b1, b2\, (< a, b1 > \in r \land < c, b2 > \in r \land b1 > b2))$

Main Contents

1. 关系数据模型*
2. 关系代数基础*
3. 元组关系演算
4. 域关系演算
5. **小结**

关系代数运算

6个基本运算		符号
① Union	(并)	∪
② set difference	(差)	−
③ Cartesian product	(笛卡尔积)	X
④ Select	(选择)	δ
⑤ Project	(投影)	π
⑥ Rename	(换名)	ρ

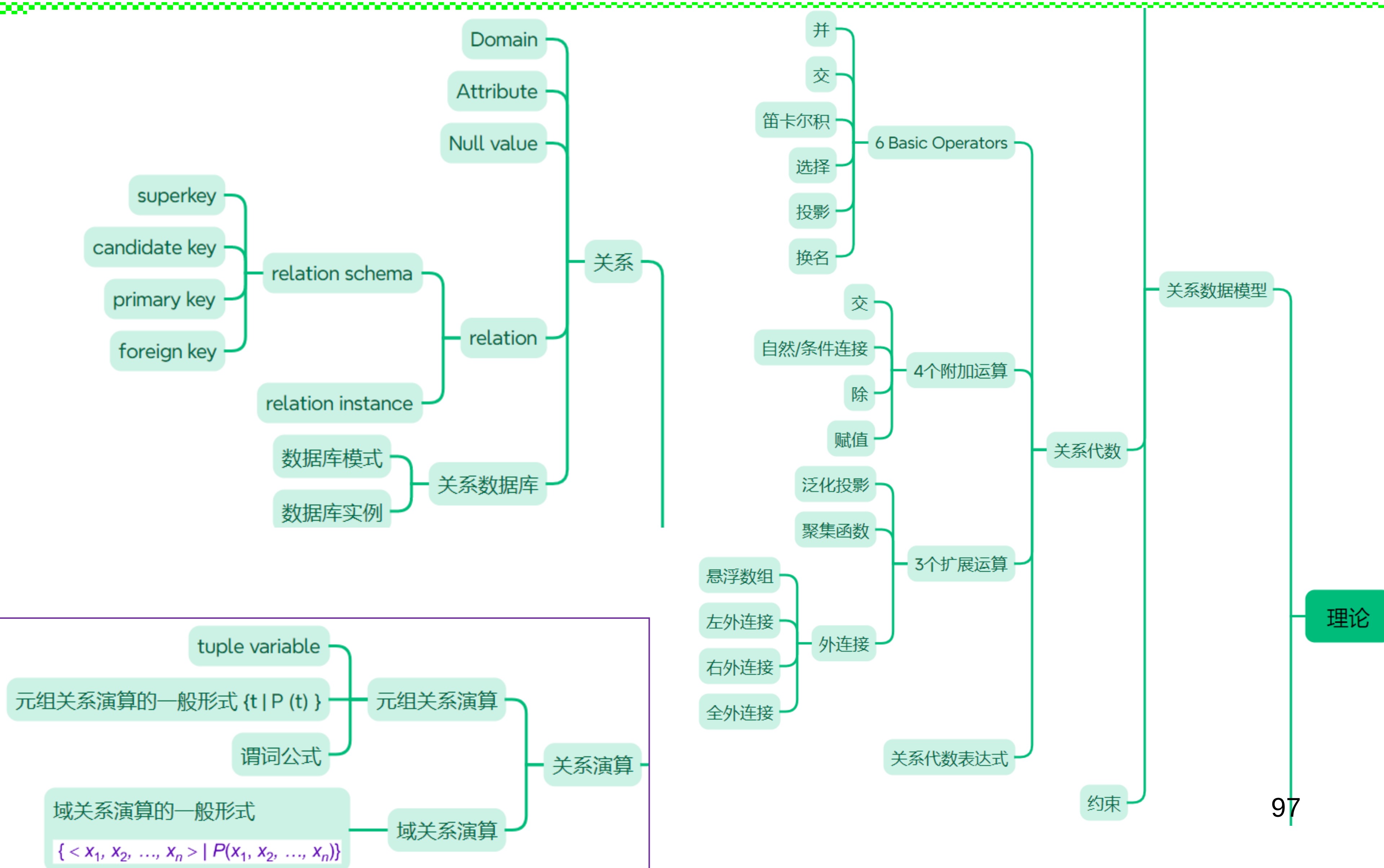
4个附加运算		
① Set intersection	(交)	∩
② Natural join/Theta Join	(自然/条件连接)	⋈
③ Division	(除)	÷
④ Assignment	(赋值)	←

3个扩展运算		
① Generalized Projection	(泛化投影)	π
② Aggregate Functions	(聚集函数)	∂
③ Outer Join	(外连接)	⟗

本节课的内容

内容	核心知识点	对应章节
DBMS理论	**关系数据模型（以及关系演算)**、SQL语言	CHP. 1、2、3
DBMS设计	存贮、索引、查询、优化、事务、并发、恢复	CHP. 12、13、14、15、16、17、18、19
DBMS实现	大作业	小班辅导
DBMS应用	实体-联系图、关系范式、DDL、JDBC	CHP. 4、5、6、7

小结



理论
关系数据模型
关系
Domain
Attribute
Null value
relation
relation schema
superkey
candidate key
primary key
foreign key
relation instance
关系数据库
数据库模式
数据库实例
关系代数
6 Basic Operators
并
交
笛卡尔积
选择
投影
换名
4个附加运算
交
自然/条件连接
除
赋值
3个扩展运算
泛化投影
聚集函数
外连接
悬浮数组
左外连接
右外连接
全外连接
关系代数表达式
关系演算
元组关系演算
tuple variable
元组关系演算的一般形式 {t | P (t) }
谓词公式
域关系演算
域关系演算的一般形式
$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$
约束

音乐数据模型

Abstract

A music data model, its query language and its application are proposed in this paper. Firstly, a music data model and its algebraic operations are given, which can be used to describe and manipulate musical data efficiently. Secondly, a structured query language on the model is proposed, which can be used to define and manage musical data. Finally, a digital music library, one of the applications of this model, is presented, which can be used to retrieve musical information, especially against musical instruments.

1. Motivation

Database systems have been a phenomenal success in terms of facilitating organization and processing of volumes of musical data. Current music data (base) systems, however, are either unscalable or unable to provide content-based retrieval functions on various musical instruments. Usually they are based on one of the following techniques: storing a piece of music as a file in a directory, or as a naive binary large object (BLOB) in a database.

Besides the performance inefficiency, a common problem with most existing music data systems is that they usually ignore the theoretical basis of data models for musical

implemented as algebraic operations of the database management system. From the library, users can retrieve musical information based on content, especially against different kinds of musical instruments.

Our main contributions are as follows. 1) Propose a music data model in which musical data is structurally organized. 2) Describe a set of algebraic operations on the data model. These operations are used to cope with state-of-the-art applications on musical data, e.g. content-based music information retrieval. 3) Present an accompanying query language constructed on the model. The language supports most applications. 4) Construct a digital music library based on the data model and query language.

The rest of this paper is organized as follows. A review of related work is provided in Section 2. The music data model and algebraic operations are respectively described in Section 3 and Section 4. Based on the model and algebraic operations, Section 5 presents a query language and Section 6 reports a digital music library. Section 7 concludes the paper and highlights some future research directions.

2. Related Work

There has been some work on the modelling of musical data. Rubenstein gave a database design for musical information [9]. He considered extensions to the entity-relationship data model to implement the notion of hierar-

A music data model and its application

Publisher: IEEE

Cite This

PDF

Chaokun Wang ; Jianzhong Li ; Shengfei Shi **All Authors**

图数据模型

TABLE 1: Fundamental Operators of Our Graph Algebra and Some Additional Operators. The column *"Unary/Binary"* means the number of the processed graph sets when the operator is applied, and the column *"New Schema"* indicates whether the schema of the obtained graph set is different from any of the processed graph sets.

Operator	Sign	Description	Unary /Binary	New Schema
Fundamental Operators				
Projection	ρ	choose vertices and edges with necessary labels, and choose their attributes with the given new schema	Unary	✓
Selection	σ	obtain the subgraphs of the graphs in a graph set, and the subgraphs should satisfy the given constraints	Unary	×
Reduce	γ	obtain a new graph by putting all the graphs in a given graph set together	Unary	×
Cartesian Product	$\times$	suppose $\mathcal{G}_1$ contains n graphs $G_{11}, \cdots, G_{1n}$, and $\mathcal{G}_2$ contains m graphs $G_{21}, \cdots, G_{2m}$, $\mathcal{G}_1 \times \mathcal{G}_2$ contains mn graphs $G_{11} \cup_g G_{21}, \cdots, G_{11} \cup_g G_{2m}, \cdots, G_{1n} \cup_g G_{2m}$, where $\cup_g$ means to unite two graphs	Binary	✓
Align	ζ	align every two vertices (edges) that represent the same entity (relationship)	Unary	✓
Difference	$-$	set difference between two graphs sets	Binary	×
Map	μ	deal with the graphs in a graph set respectively	Unary	×
Additional Operators				
Join	$\bowtie$	given two graph sets $\mathcal{G}_1 = (\widehat{G}_1, \mathcal{S})$ and $\mathcal{G}_2 = (\widehat{G}_2, \mathcal{S})$, $\mathcal{G}_1 \bowtie \mathcal{G}_2 = \zeta_{h_1}(\cdots(\zeta_{h_m}(\mathcal{G}_1 \times \mathcal{G}_2))\cdots)$, after uniting a graph in $\mathcal{G}_1$ and a graph in $\mathcal{G}_2$, a vertex labeled $1.x$ and a vertex labeled $2.x$ are aligned, where $x \in \mathcal{S}.\mathcal{T}$, iff they have the same values in *all* attributes. Besides, two edges labeled $1.x$ and $2.x$ respectively are aligned iff they have the same attribute values and the same source and target vertices.	Binary	×
Intersection	$\cap$	set intersection, $\mathcal{G}_1 \cap \mathcal{G}_2 = \mathcal{G}_1 - (\mathcal{G}_1 - \mathcal{G}_2)$	Binary	×
Maximal	$\mathcal{M}$	choose the maximal graphs in a graph set $\mathcal{M}(\mathcal{G}) = \mu(\mathcal{G} \| \mathcal{G}' : \gamma((\mathcal{G}' \bowtie \mathcal{G}) \cap \mathcal{G}) \cap \mathcal{G}')$	Unary	×
Union	$\cup$	given two graph sets $\mathcal{G}_1 = (\widehat{G}_1, \mathcal{S})$ and $\mathcal{G}_2 = (\widehat{G}_2, \mathcal{S})$ with the same schemas, put the graphs in them into a new graph set $\mathcal{G}_1 \cup \mathcal{G}_2 = (\widehat{G} = \{G \mid G \in \widehat{G}_1 \vee G \in \widehat{G}_2\}, \mathcal{S})$	Binary	×

Journals & Magazines > IEEE Transactions on Big Data > Early Access

Graph Data Model and Graph Query Language Based on the Monadic Second-Order Logic

Publisher: IEEE

Cite This | PDF

Yunkai Lou ; Chaokun Wang ; Songyao Wang All Authors

谢谢！