数据库技术-关系模型

赵亚伟

zhaoyw@ucas.ac.cn

中国科学院大学 大数据分析技术实验室

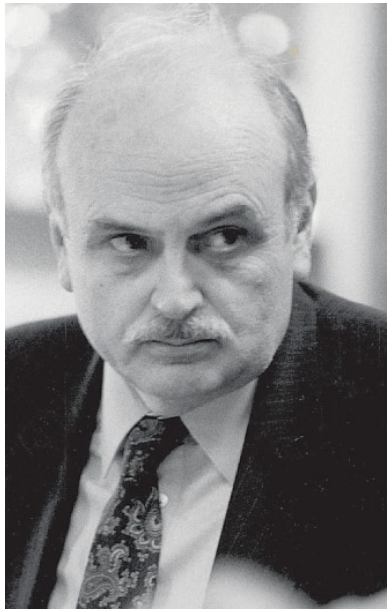
2017.11.19

目录

- 关系数据库理论的发展
- 关系模型
  - 关系数据结构
  - 关系代数
  - 扩展关系代数运算
  - 关系的完整性
  - 空值
  - 数据库修改
- 视图

关系数据理论的发展

- 1970年，IBM公司San Jose研究室的研究员E.F.Codd系统提出关系数据库理论。Codd撰写了一篇名为“A Relational Model of Data for Large Shared Data Banks”文章，奠定了关系数据库的基础，E.F.Codd被誉为“数据库之父”。
- ACM在1983年把该文列为从1958年以来四分之一世纪中具有里程碑式意义的25篇论文之一。
- 关系模型简单明了，有坚实的数学基础，即关系代数。
- 从关系代数的基础推演出了一套关系数据库理论，即“范式”，可以检查数据库是否有冗余和不一致性等。定义了一系列通用的数据基本操作，原来在层次网状数据库中的复杂操作变得简明扼要



业界跟进-IBM

- 两个开创性的原型系统:
  - **1974, IBM “System R”=> SQL**
  - **UC Berkely, “Ingres” => QUEL**
  - 两系统双双获得**ACM**的**1988**年“软件系统奖”
- **1977**年，**IBM**完成了**System R**原型（**DB2**基础）。**1982**年，**IBM**推出第一个关系数据库产品（**SQL/DS for VSE and VM**，基于**SR**）。**IBM**产品化步伐缓慢的三个原因:
  - **IBM**重视信誉，重视质量，尽量减少故障
  - **IBM**是个大公司，官僚体系庞大
  - **IBM**内部已经有层次数据库产品（**IMS**），相关人员不积极，甚至反对

业界跟进-Oracle

- **Oracle 前 身 叫 SDL ， 由 Larry Ellison和另两个编程人员在1977创办，1979开发自己的拳头产品，市场上大量销售（三个人，两个用户，一个产品）**

- **Oracle开发关系数据库产品做事方法三个要点：**
  - 第一，不做研究，只做产品开发
  - 第二，以尽快推出产品为第一目标
  - 第三，产品要能够在销量较大的平台上运行，并具有良好的移植性

主流数据库系统

- 当前，主流数据库产品基本上都源于基于关系数据库技术的两个典型实验系统
  - **System R**
  - **University INGRES（PostgreSQL）**
- 典型商用系统
  - **ORACLE**
  - **SYBASE**
  - **INFORMIX**
  - **DB2**
  - **SQL Server**
  - **……**

目录

为什么称之为“关系”模型？

数学上的关系

- 数学上，关系表达了集合中元素之间的联系
- 关系的定义
  - “令**X**和**Y**是任意两个集合，直积（笛卡尔积）**X**×**Y**的子集**R**称为**X**到**Y**的关系。”
- 笛卡尔积的定义
  - “令**A**和**B**是任意两个集合，若序偶的第一个成员是**A**的元素，第二个成员是**B**的元素，所有这样序偶的集合成为集合**A**和**B**的笛卡尔积或直积，写作**A**×**B**。”

例子

- 考虑两个集合， $D1$和 $D2$
  - $D1 = \{2, 4\}$，$D2 = \{1, 3, 5\}$
- 卡氏积
  - $D1 \times D2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$
- $D1$到$D2$的关系是$D1 \times D2$的任意子集
  - 如$R = \{(x, y) \mid x \in D_1, y \in D_2, \text{and } y = 1\}$，表示$y = 1$情况下的$D1 \times D2$，是$D1 \times D2$的一个子集
- 上述是二元关系的定义，可以推广到n关系， n个集合$D_1, D_2, \ldots, D_n$的卡氏积$D_1 \times D_2 \times \ldots \times D_n$的子集构成一个n元关系。

关系数据模型的“关系”

- 关系数据库是表的集合，在关系模型中只有表这一种存储结构
- 表的一行代表的是一系列值之间的联系，一个表就是这种联系的集合

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

关系

- 如果把属性的取值（域）作为集合，那么表一定是所有属性域的直积的子集。
- 很显然，直积的子集就是关系。如上述**account**表就是属性域$D_{account_number}$, D_{branch_name}, $D_{balance}$的笛卡尔积的一个子集。即
  $account \subset D_{account\_number} \times D_{branch\_name} \times D_{balance}$
- 因此表的概念和数学上的关系的概念在本质上一致，这是关系数据库名称的由来。
- 关系数据库是表的集合，每个表有唯一的名字，表中的一行代表的是一系列值之间的联系。

几个基本概念的不同称谓

正式术语	别名1	别名2
关系(relation)	表(table)	文件(file)
元组(tuple)	行(row)	记录(record)
属性(attribute)	列(column)	字段(field)

注意别名1与别名2中的名词经常交叉应用，我们在后面教学中也可能交叉使用各种概念。

关系数据模型

关系数据模型

- 数据模型三要素(回顾):数据结构, 数据操作, 约束条件

- 数据结构（关系及关系数据库）---核心是 关系:现实世界的实体集以及实体间联系集均用关系来表示
- 关系操作（关系代数）:
  - 查询操作:选择, 投影, 连接, 除, 并, 交, 差
  - 更新操作:增加, 删除, 修改, 更名
- 完整性约束（码）:
  - 实体完整性
  - 参照完整性
  - 用户定义的完整性

本次课程探讨的核心内容：关系模型三个要素，**数据结构,数据操作,约束条件**

基本结构—关系描述

- 域(Domain)：是一组具有相同数据类型的值的集合，如果域上的元素被看做是不可再分的单元，则该域是原子的（1NF）。我们约定课程例子中的域都是原子的。
- 笛卡尔积(Cartesian Product)：给定一组域D1, D2, …,Dn, 则D1, D2, …, Dn的笛卡尔积为:

  $D1 \times D2 \times \ldots \times Dn = \{(d1, d2, \ldots, dn) | di \in Di, i=1,2,\ldots,n\}$
- 其中每一个元素(d1, d2, …, dn)称为一个n元组(n-tuple)或简称元组 (tuple)

- 元素中的每一个值**di**称为一个分量
- **若Di(i=1, 2, … n)为有限集，其基数(Cardinal number)为mi(i=1, 2, …, n)则D1× D2 × … × Dn的基数M为:**

$$M = \prod_{i=1}^{n} m_i$$

- 基数表达了卡式积所拥有的元组数量

基本结构—关系描述

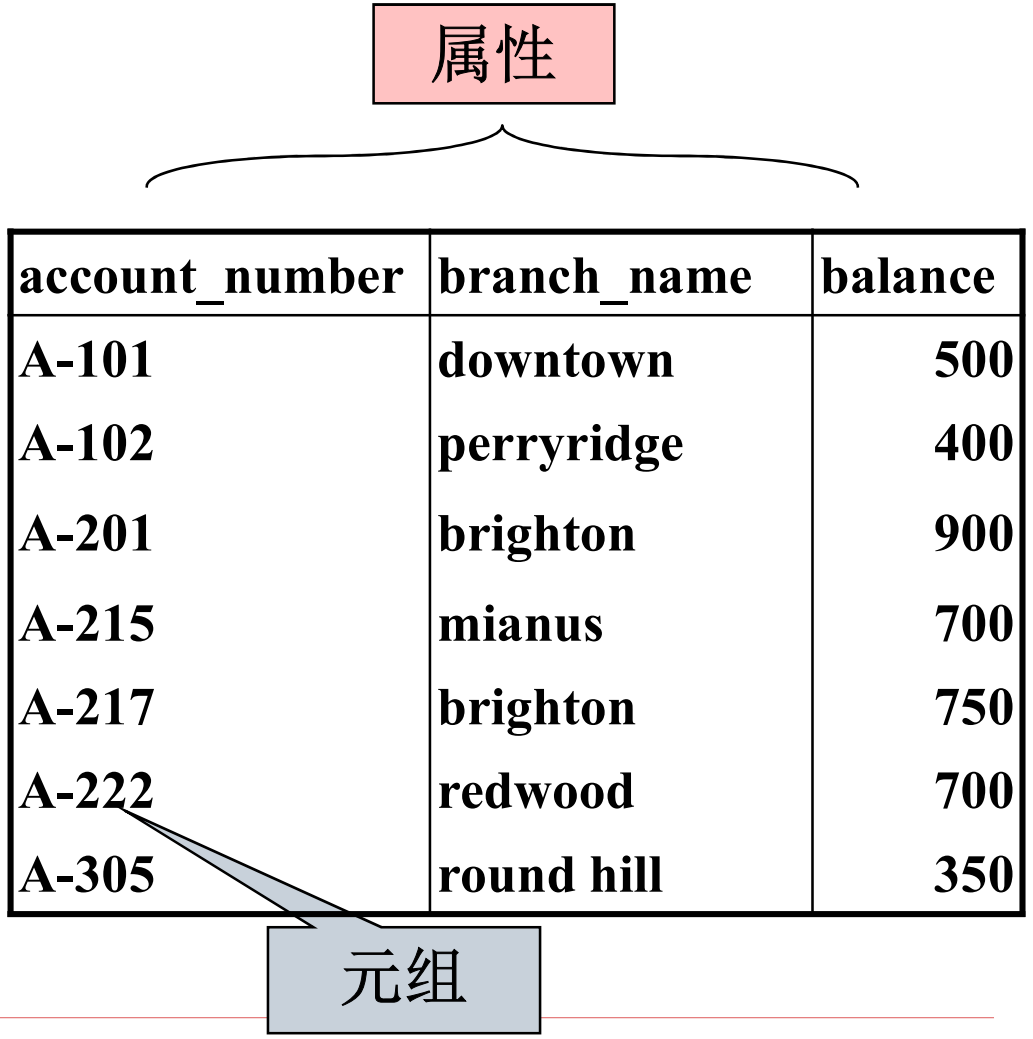
- 笛卡尔积可表示为一个二维表
- 关系(Relation) : D1× D2 × … × Dn的子集叫作在域D1, D2, … , Dn上的关系，表示为
  - R (D1, D2, … , Dn)，这里R表示关系的名字，n是关系的目或度(Degree)，属性数量。当n=1时，称该关系为单元关系，当n=2时，称该关系为二元关系，……
  - 关系中的每个元素是关系中的元组，通常用t表示。
  - 关系也是一个二维表，表的每行对应一个元组，表的每列对应一个域。由于域可以相同，为加以区分，必须对每列起一个名字，称为属性(Attribute)。n目关系必有n个属性，n元关系

- **Order of tuples is irrelevant (tuples may be stored in an arbitrary order)** 由于关系是元组的集合，所以元组在关系中的顺序是无关紧要的。
- **Example: *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

例子

- 三元组 **(account_number , branch_name, balance)**
- 基数**mi=7×6×6＝252**（假设域取值如右表，可能不完整）
- **account_number**域为所有帐号或字符串
- 目：**3**
- **3**元关系



关系分类

- 关系可以有三种类型：基本关系(通常又称为基本表或基表)、查询表和视图表。
  - 基本表是实际存在的表，它是实际存储数据的逻辑表示。
  - 查询表是查询结果对应的表。
  - 视图表是由基本表或其他视图表导出的表，是虚表，不对应实际存储的数据。

基本表性质

- 基本关系具有以下六条性质：

①列是同质的，即每一列中的分量是同一类型的数据，来自同一个域。

②不同的列可出自同一个域，称其中的每一列为一个属性，要给予不同的属性名。

③列的顺序无所谓，即列的次序可以任意交换。由于列顺序是无关紧要的，因此，在许多实际关系数据库产品中(如**Oracle)**，增加新属性时，永远是插至最后一列。

④任意两个元组不能完全相同。

⑤行的顺序无所谓，即行的次序可以任意交换。

⑥分量必须取原子值，即每一个分量都必须是不可分的数据项。

注意

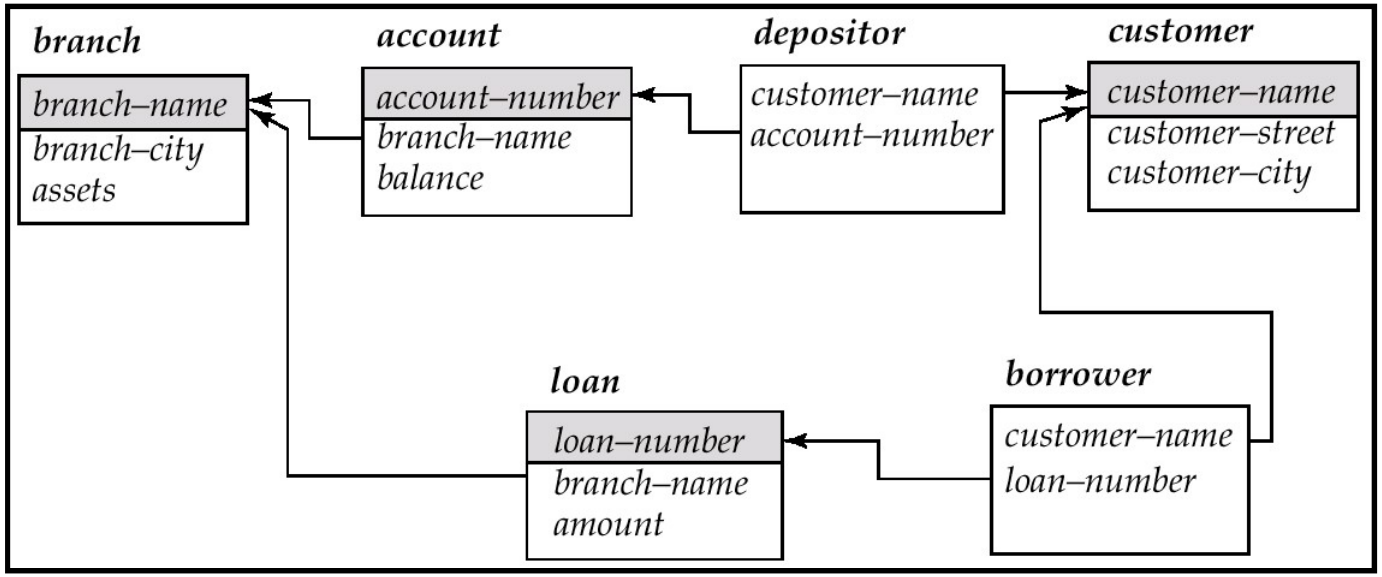
- 在许多实际关系数据库产品中，基本表并不完全具有这六条性质。例如，有的数据库产品(如**FoxPro**)仍然区分了属性顺序和元组的顺序；
- 在许多关系数据库产品中，例如**Oracle**，**FoxPro**等，它们都允许关系表中存在两个完全相同的元组，除非用户特别定义了相应的约束条件。

数据库模式

- 关系数据库也有型和值之分。
- 关系数据库的型也称为关系数据库模式，是对关系数据库的描述，是数据库的逻辑设计结果。
- 关系数据库的值是给定时刻数据库中数据的一个快照。这些关系模式在某一时刻对应的关系的集合，通常就称为关系数据库。
- 模式化的电影，容易写，容易演，容易猜，容易用数据库管理：
  - 模式化电影（片名，类型， 起，承，转，合，收尾）
- 模式化数据 好管理

数据库的模式图

- 一个含有主码和外码依赖的数据库模式可以用模式图（**schema diagram**）。模式图是逻辑设计的结果。
- 模式图与**E-R**图的区别是：**E-R**图不会显式地表示外码属性，而模式图是显示表示的。
- 很多数据库系统提供模式图的设计工具



关系模式

- 在数据库中关系也要区分型和值。
- 关系模式是型，关系是变量，实例是值（快照）。
- 关系的描述称为关系模式(**Relation Schema**)。它可以形式化地表示为：

  R(U, D, dom, F)

  其中**R**为关系名，**U**为组成该关系的属性名集合，**D**为属性组**U**中属性所来自的域，**dom**为属性向域映像的集合，**F**为属性间数据的依赖关系集合
- 关系模式通常简记为

  R(U) 或 **R(A1, A2, … , An)**

  A1, A2, … , An为属性名

关于模式、关系、实例的探讨

- 根据前面的讨论，与程序设计语言对比
  - 模式 对应 类型
  - 关系 对应 变量
  - 实例 对应 值（快照）
- 由于
  - 关系 对应 表
  - 关系 对应 变量
  - 表在某一时刻的取值就是该变量（表）的实例
- 可以得出：表是变量，而表是通过某个**DBMS**创建的

存在的问题与自圆其说

- 通过**DBMS**定义一个表时，首先定义了该表的结构（即模式），然后进行存储，存储后为一个表，这个表在某个时刻的取值就是实例
- 在实际中，我们看到的只有实例，即某个时刻的表的取值，而变量、模式（类型）在那里？
- 因此，关系模型在具体实现过程中存在问题：
  - 模式、变量、值的概念不清晰
- 一个自圆其说的说法是，模式、变量与实例采用同一个名字，三合一。

例子：模式与关系

- 在设计中，通常给关系模式一个显式的名字。
- 例如，用**Account_schema**表示关系**account**的关系模式：
  - **Account_schema＝(account_number , branch_name, balance)**
- 用**account(Account_schema)**表示**account**是**Account_schema**上的关系：
  - **account (Account_schema)**

实例是具有某模式的关系在某一时刻的取值状态或内容，关系模式是静态的，而关系是动态的，实例则是关系在某一时刻的快照

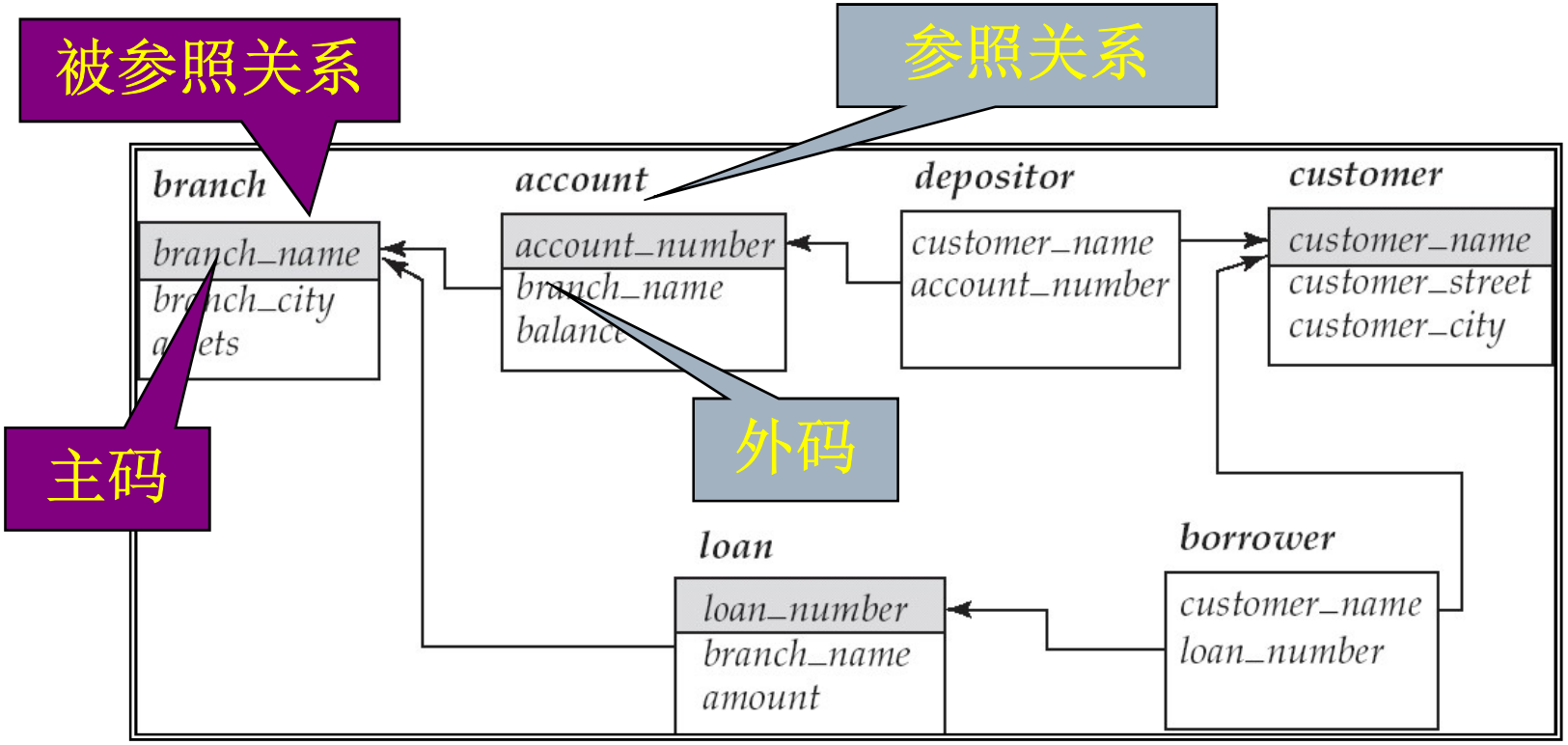
码 — 元组及联系描述

- 码用于区别元组,可以唯一地标识元组的属性
- 若关系中的某一属性组的值能唯一地标识一个元组，则称该属性组为超码
- 任意真子集都不能成为超码的超码称为候选码，候选码是最小的超码
- 若一个关系有多个候选码，则选定其中一个为主码（**Primary Key**），主码的诸属性称为主属性，不包含任何候选码的属性称为非码属性。简单情况下，候选码只包含一个属性，最极端情况下，关系中的所有属性组都是候选码，称为全码
- 注意：主码可能是一个属性组，不总是一个属性

外码

- 参照关系/被参照关系：一个关系模式**r1**包含另一个关系模式**r2**的主码，这个属性称为**r1**参照**r2**的外码（**foreign key**）
- 关系**r1**被称为**r2**的外码依赖的参照关系（**referencing relation**），**r2**被称为**r1**的外码的被参照关系（**referenced relaiton**）。
- 通过外码可以实现关系的参照完整性。

例子：主码、外码


被参照关系
参照关系
branch
branch_name
branch_city
assets
account
account_number
branch_name
balance
depositor
customer_name
account_number
customer
customer_name
customer_street
customer_city
主码
外码
loan
loan_number
branch_name
amount
borrower
customer_name
loan_number

目录

关系操作：集合操作

- 关系操作的特点是集合操作方式，即操作的对象和结果都是集合
- 这种操作方式被称为一次一集合（**set-at-a-time**）的方式。
- 非关系数据模型的数据操作方式则为一次一记录（**record-at-a-time**）的方式。
- 关系代数、元组关系演算和域关系演算都是抽象的查询语言。
- **SQL**是另外一种介于关系代数和关系演算之间的查询语言

关系数据操作-关系代数

- 关系代数是一种抽象的查询语言，是关系数据操纵语言的一种传统表达方式。
- 运算（表达式）= 对象+运算符+规则
- 关系代数的运算按运算符的不同可分为：
  - 传统的集合运算
  - 专门的关系运算

关系代数运算符

运算符		含义	运算符		含义
集　合 运算符	∪ − ∩	并 差 交	比　较 运算符	＞ ≥ ＜ ≤ ＝ ≠	大于 大于等于 小于 小于等于 等于 不等于
关　系 运算符	× σ Π ⋈ ÷	广义笛卡尔积 选择 投影 连接 除	逻　辑 运算符	¬ ∧ ∨	非 与 或

关系代数-集合运算

- 传统的集合运算是二目运算，包括并、差、交、广义笛卡尔积四种运算。
- 条件：设关系**R**和关系**S**具有相同的目**n**(即两个关系都有**n**个属性)，且相应的属性取自同一个域，则可以进行并、差、交运算。

集合运算（1）-并，差，交

- 并（**union**）：关系**R**与关系**S**的并记为

  $R \cup S=\{t \mid t \in R \vee t \in S\}$

  其结果仍为**n**目关系，由属于**R**或属于**S**的元组组成。
- 差（**Difference**）：关系**R**与关系**S**的差记为

  $R-S=\{t \mid t \in R \wedge t \notin S\}$

  其结果仍为**n**目关系，由属于**R**但不属于**S**的元组组成。
- 交（**Intersection**）：关系**R**与关系**S**的交集为

  $R \cap S=\{t \mid t \in R \wedge t \in S\}$

  其结果仍为**n**目关系，由既属于**R**又属于**S**的元组组成。

  关系的交可以用差来表示，即$R \cap S=R-(R-S)$

例子：运算用的两个关系

customer_name	loan_number
adams	L-16
curry	L-93
hayes	L-15
johnson	L-14
jones	L-17
smith	L-11
smith	L-23
williams	L-17

关系：**borrower**

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

关系：**depositor**

例子：并

- 所有出现在这两个集合之一的或同时出现在这两个集合中的客户姓名
- 两个关系都包含**customer_name**属性
- 表达式：$\Pi_{customer\_name}(borrower) \cup \Pi_{customer\_name}(depositor)$
- 注意消除重复的元组

customer_name
adams
curry
hayes
johnson
jones
smith
williams
lindsay
turner

例子：差

- 找出在一个关系中而不在另一个关系中的元组。
- 找出所有有账户而无贷款的客户
- 两个关系都包含**customer_name**属性
- 表达式：
  $\Pi_{customer\_name}(depositor) - \Pi_{customer\_name}(borrower)$

customer_name
lindsay
turner

例子：交

- 找出在两个关系中都存在的元组。
- 找出所有有账户并有贷款的客户
- 两个关系都包含**customer_name**属性
- 表达式：
  $\Pi_{customer\_name}(depositor) \cap \Pi_{customer\_name}(borrower)$

customer_name
adams
curry
hayes
johnson
jones
smith
williams

集合运算（2）-广义笛卡尔积

- 两个分别为n目和m目的关系R和S的广义笛卡尔积是一个(n+m)列的元组的集合。
- 元组的前n列是关系R的一个元组，后m列是关系S的一个元组。
- 若R有k1个元组，S有k2个元组，则关系R和关系S的广义笛卡尔积有kl×k2个元组。记作：

  $R \times S = \{t_r t_s \mid t_r \in R \wedge t_s \in S\}$

集合运算（2）-广义笛卡尔积

R

A	B	C
a1	b1	c1
a1	b2	c2
a2	b2	c1

S

A	B	C
a1	b2	c2
a1	b3	c2
a2	b2	c1

R×S

A	B	C	A	B	C
a1	b1	c1	a1	b2	c2
a1	b1	c1	a1	b3	c2
a1	b1	c1	a2	b2	c1
a1	b2	c2	a1	b2	c2
a1	b2	c2	a1	b3	c2
a1	b2	c2	a2	b2	c1
a2	b2	c1	a1	b2	c2
a2	b2	c1	a1	b3	c2
a2	b2	c1	a2	b2	c1

关于广义笛卡尔积的讨论

- 广义笛卡尔积与笛卡尔积
  - 前面讨论卡式积运算实际上是域上的卡式积。
  - 笛卡尔积扩展到关系上，即关系上的卡式积。
  - 广义卡式积是指所有参与关系的所有元组的交叉乘积，所有属性的累加。
  - 根据定义，同名的两个属性可以同时出现在结果中。
- 同名的两个属性可以同时出现在结果中会引起歧义

属性的命名机制

- 采用属性上附加该属性所来自的关系的命名机制以区分该属性的来源。
- 实际系统中关于属性的命名机制
  - 两个关系的卡式积，不重名属性不带前缀（该属性所属的关系）
  - 重名的属性则需要通过前缀加以区别该属性来源哪个关系，使语义更明确

例子

customer_name	loan_number
adams	L-16
curry	L-93
hayes	L-15
johnson	L-14
jones	L-17
smith	L-11
smith	L-23
williams	L-17

关系：borrower

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

关系：loan

例子

- 属性附加关系的命名机制：如**r=borrower×loan**的关系模式为
  - **(borrower.customer_name, borrower.loan_number, loan.loan_number, loan.branch_name, loan.amount)**
- 简化处理：对那些只在其中一个模式中出现的属性不加关系名前缀，而在两个关系中都出现的属性加关系名前缀，不会导致任何歧义：如**r=borrower×loan**的关系模式为
  - **(customer_name, borrower.loan_number, loan.loan_number, branch_name, amount)**

关系代数-关系运算

- 专门的关系运算包括

  选择

  投影

  连接

  除

  …

关系运算-选择

- 选择又称为限制，它是在关系**R**中选择满足给定条件的诸元组（注意是元组的选择），记作

$$\sigma_{F(R)} = \{t|t \in R \wedge F(t)=\text{‘真’}\}$$

其中，**F**表示选择条件，是一个逻辑表达式，取逻辑值“真”或“假”。

如，下面的选择表达式表示从关系**loan**中选择**amount>1200**元的所有元组：

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

- 选择操作不会改变原表

例子：选择运算

- 选择运算是从行的角度进行操作,一般用于查询
- 选择**loan**关系中支行名称为“**perryridge**”的元组：
  - $\sigma_{branch\_name=\text{“perryridge”}}(loan)$
- 查询结果如下

loan_number	branch_name	amount
L-15	perryridge	1500
L-16	perryridge	1300

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

关系**loan**

例子：使用连接词

- 可以使用连词**and** ($\wedge$), **or** ($\vee$)和**not** ($\neg$)将多个谓词合并为一个较大的谓词，如：找出所有贷款大于**1200**美元并由**perryridge**支行产生的贷款，可以写作：
  - $\sigma_{\text{branch\_name}=\text{“perryridge”} \wedge \text{amount}>1200}(\text{loan})$

loan_number	branch_name	amount
L-15	perryridge	1500
L-16	perryridge	1300

关系运算-投影

- 关系**R**上的投影是从**R**中选择出若干属性列组成新的关系（注意投影是属性选择）。记作

  $$\Pi_A(R)=\{t[A]|t\in R\}$$

  其中，**A**为**R**中的属性列

- 投影操作是从列的角度进行的运算。
- 投影之后不仅取消了原关系中的某些列，而且还可能取消某些元组，因为取消了某些属性列后，就可能出现重复行，由于关系是一个集合，所以重复的行均被去除。
- 与选择运算相同，投影不改变原表

例子

- 例子：列出关系**loan**中所有贷款号码及金额。使用如下投影表达式：
  - $\Pi_{\text{loan-number,amount}}(\text{loan})$
- 查询结果：

loan_number	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

关系运算的组合

- 由于关系运算是集合操作，运算结果也是一个集合，因此，可以把一个运算结果（关系）也作为一个关系嵌套在另一个关系运算中。
- 可以把多个关系代数元算组合称一个关系代数表达式。
- 例如：找出居住在**harrison**的所有客户。
  - $\Pi_{customer\_name}(\sigma_{cuatomer\_city=“harrison”}(customer))$

视为一个关系

关系运算-连接

- 连接也称为θ连接。它是从两个关系的笛卡尔积中选取属性间满足一定条件的元组。记作：

$$R \underset{A\,\theta\,B}{\bowtie} S=\{\overset{\frown}{tr\,ts} \mid tr \in R \wedge ts \in S \wedge tr[A]\theta ts[B]\}$$

- A、B为属性组
- 最为常用的两种连接：
  - 等值连接(θ取=号)
  - 自然连接(仅用⋈符号，等值连接的一种特殊情况，要求两个关系中的进行比较的分量必须是相同的属性组(A=B)，结果中去掉重复的属性列)

例子：等值连接和自然连接

R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

S

B	E
b1	3
b2	7
b3	10
b3	2
b5	2

等值连接

A	R.B	C	S.B	E
a1	b1	5	b1	3
a1	b2	6	b2	7
a2	b3	8	b3	10
a2	b3	8	b3	2

自然连接

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2

关系运算-除

- **Find all customers who have an account at all branches located in Brooklyn city.** 找出所有位于**Brooklyn**的所有支行都有帐户的客户。
- 已知：关系**branch**、**depositor**、**account**

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

关系branch

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

关系depositor

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

关系account

- 找出所有位于**Brooklyn**的所有支行名：

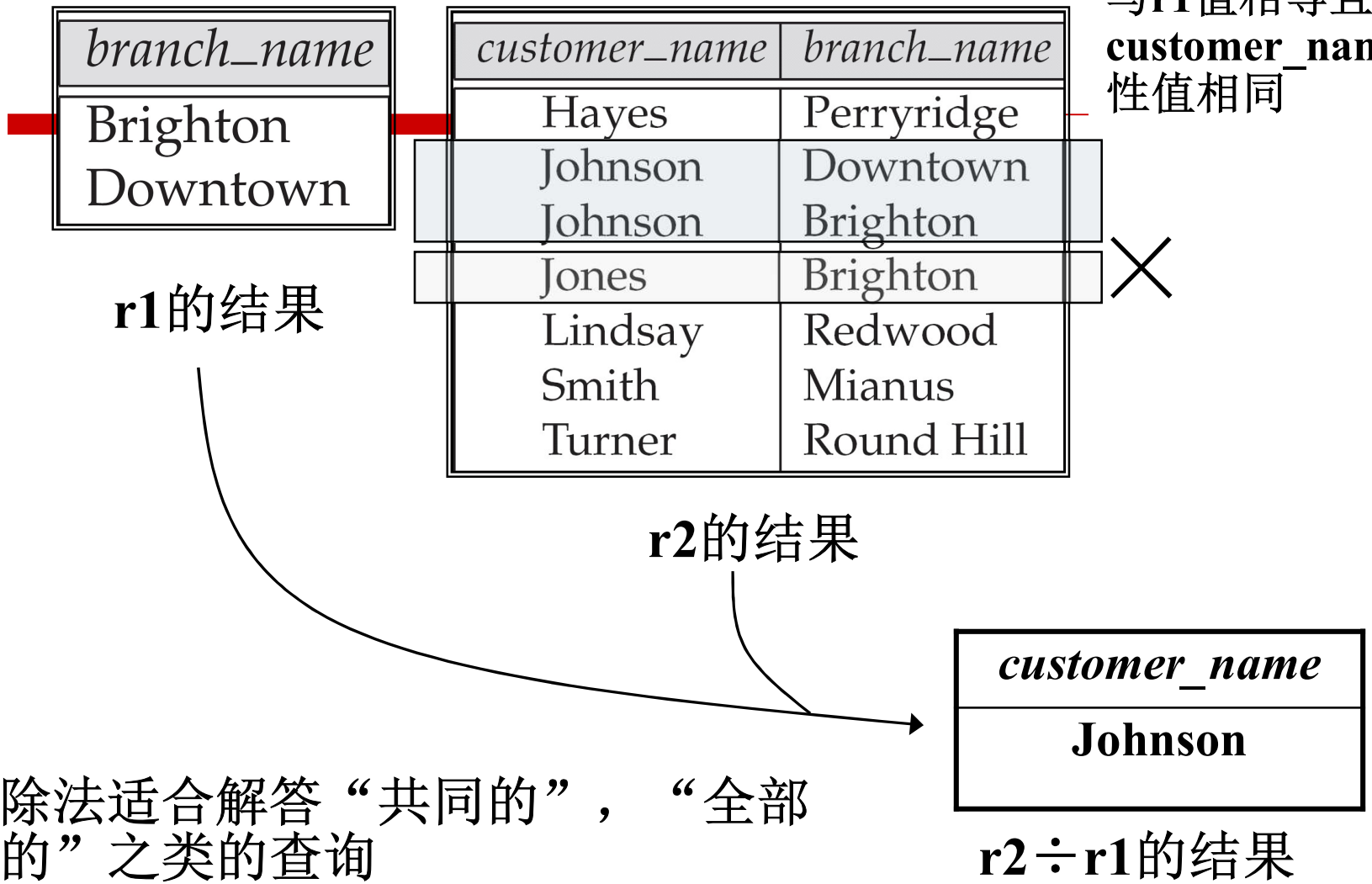
$$r1=\prod_{branch\_name}(\sigma_{branch\_city=\text{“Brooklyn”}}(branch))$$

- 找出客户在支行有帐户的所有(*customer_name, branch_name*) 客户名，支行名

$$r2=\prod_{customer\_name,\ branch\_name}(depositor \bowtie account)$$

- 所有位于**Brooklyn**的所有支行都有帐户的客户：

$$r2 \div r1=\prod_{customer\_name,\ branch\_name}(depositor \bowtie account) \div \prod_{branch\_name}(\sigma_{branch\_city=\text{“Brooklyn”}}(branch))$$



除法适合解答“共同的”，“全部的”之类的查询

关系运算-除

- 元组t属于r ÷s，当且仅当以下两个条件同时成立：

  1. t在$\Pi_{R-S}(r)$中（上例R2-R1中, customer_name）
  2. 对s中的每一个元组ts，在r中都有元组tr同时满足以下两个条件：

     tr[S]=ts[S]; （Down，Brighton）

     tr[R-S]=t （只有Johnson同时满足二条件）

- 用基本运算定义除运算（课下验证）：

Let *r(R)* and *s(S)* be relations, and let $S \subseteq R$

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

例子：关系除运算-运用象集

在**R**中，**A**可以取四个值**{a1, a2, a3, a4}**。其中

a1的象集

{(b1,c2),(b3,c3),(b2,c1)}

a2的象集

{(b3,c7),(b2,c3)}

a3的象集

{(b4,c6)}

a4的象集

{(b6,c6)}

S的集合形式为：

{(b1,c2),(b2,c1),(b3,c3) }

显然，只有**a1**的象集包含了**S**，所以**R÷S=a1**

R

A	B	C
a1	**b1**	**c2**
a2	b3	c7
a3	b4	c6
a1	**b3**	**c3**
a4	b6	c6
a2	b2	c3
a1	**b2**	**c1**

S

B	C
b1	c2
b2	c1
b3	c3

R÷S

A
a1

Another Division Example

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

***r* ÷ *s*:**

A	B	C
α	a	γ
γ	a	γ

更名运算

- 更名运算用ρ表示，提供两种形式的更名操作：
  - 表达式更名：$\rho_x(E)$，将表达式E更名为x。
  - 属性更名：$\rho_{x(A1, A2, \ldots, An)}(E)$，如果$E$是$n$元的，将返回在名字$x$下表达式E的结果，同时将属性依次更名为$A1, A2, \ldots, An$。
- 由于关系自身被认为是最小的关系表达式，所以可以对关系进行更名，**SQL：as**

例子：表达式更名

- 从关系**account**中产生非最大余额的关系：
  - $\Pi_{account.balance}(\sigma_{account.balance<d.balance}(account \times \rho_{d}(account)))$
- 将关系***account***更名为**d**，则表达式 ***account.balance<d.balance***就不会有歧义
- 对应的**SQL**语句：

```
select distinct account.balance
from account, account as d
where account.balance < d.balance
```

例子：属性更名

- 将关系**account**的属性**(account_number, branch_name, balance)**更名（有称之为别名）为 **(n, bn, b)**的**SQL**语句为：

```
select account_number as n, branch_name as bn, balance as b
from account
```

	n	bn	b
1	A-101	downtown	500.0
2	A-102	perryridge	400.0
3	A-201	brighton	900.0
4	A-215	mianus	700.0
5	A-217	brighton	750.0
6	A-222	redwood	700.0
7	A-305	round hill	350.0

赋值运算

- 有时可以通过临时关系变量，将关系代数表达式一部分一部分分开，这样比较方便
- 赋值运算：← （SQL：as, with）
- 如$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$可以通过赋值运算实现：

  $Temp1 \leftarrow \Pi_{R-S}(r)$

  $Temp2 \leftarrow \Pi_{R-S}((Temp1 \times s) - r)$

  $r \div s = Temp1 - Temp2$

```
select branch_name, avg_balance
from (select branch_name,avg(balance)
      from account
      group by branch_name)
      as
branch_avg(branch_name,avg_balance)
Where avg_balance>1200
```

目录

- 关系数据库理论的发展
- 关系模型
  - 关系数据结构
  - 关系代数
  - 扩展关系代数运算
  - 关系的完整性
  - 空值
  - 数据库修改
- 视图

扩展关系代数运算

- 对基本运算进行了多方面的扩展，一个简单扩展是将算术运算作为投影的一部分；另一个重要扩展是允许做聚集运算；第三个重要扩展是外连接运算，使关系代数表达式可以对缺失信息的空值进行处理
  - 广义投影
  - 聚集运算
  - 外连接
  - 空值

广义投影

- 允许在投影列表中使用算术函数来对投影进行扩展，表达式为：
  - $\Pi_{F1, F2, \ldots, Fn}(E)$
  - **E**是任意关系代数表达式
  - $F_1, F_2, \ldots, F_n$中的每一个都是涉及常量以及**E**的模式中属性的算术表达式
  - 特别地，算术表达式可以仅仅是个属性或常量

例子：广义投影

- 关系**credit_info**描述的是信贷额度和目前为止的花费（**credit_balance**），如果希望找出每个用户最多还能花费多少，用广义投影表达式：
  - $\Pi_{customer\text{-}name,\ limit - credit\text{-}balance\ as\ credit\_available}$ ***(credit-info)***

customer_name	*limit*	*credit_balance*
Curry	2000	1750
Hayes	1500	1500
Jones	6000	700
Smith	2000	400

	customer_name	credit_available
1	curry	250.0
2	hayes	0.0
3	jones	5300.0
4	smith	1600.0

聚集函数

- 聚集函数(**aggregate function**)输入值为一个集合，返回值为一个单一值，常见的聚集函数：
  - **sum**
  - **count**
  - **avg**
  - **min**
  - **max**
  - **…**

聚集运算

- 聚集运算用如下形式描述：

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

  其中，

  （1）E是任意关系代数表达式， G1, G2, …, Gn是用于分组的一系列属性

  （2）每个Fi 是一个聚集函数

  （3）Ai是一个属性名

- 运算的含义：
  - 同一组中所有元组在G1, G2, …, Gn上的值相等
  - 不同组中的元组在G1, G2, …, Gn上的值不同
- **SQL: group by (G), sum, count, max, ……(F)**

Aggregate Operation – Example

- **Relation *r*:**

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

没有指定分组属性

$g_{sum(c)}(r)$

sum(c)
27

例子

- 查询每个支行balance的和，Relation account如下：

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

指定了分组属性

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

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

外连接

- 外连接(**outer-jion**)运算是连接运算的扩展，可以处理缺失信息。
- 外连接包括：
  - 左外连接：$\overset{\sqsupset}{\bowtie}$ 左边的全保留
  - 右外连接：$\overset{}{\bowtie}\!\!\!\!\!\!\sqsubset$ 右边的全保留
  - 全外连接：$\overset{\sqsupset}{\bowtie}\!\!\!\!\!\!\sqsubset$ 两边的全保留
- 这三种连接都要计算连接，然后在连接结果上附加额外的元组

例子：连接用关系

- 连接用的两个关系：

关系**loan**

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

关系**borrower**

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

例子：左外连接

- 左外连接：在自然连接的结果上把运算符左侧关系未连接上的元组附加在结果，即运算符左侧关系的元组将全部被保留。左边的全保留
- 结果中的运算符右侧的关系属性将被赋予空值“**null**”

loan ⟕ ***Borrower***

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

例子：右外连接

- 右外连接同左外连接，只是左右位置换一下，右边的全保留

$loan \overline{\bowtie} borrower$

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

例子：全外连接

- 全外连接则是左外连接和右外连接的结合，即用空值填充左侧关系中与右侧关系的任一元组都不匹配的元组，又填充右侧关系中与左侧关系的任一元组都不匹配的元组，把结果都加到自然连接的结果上。两边的全保留

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

基本运算、附加运算与扩展运算

- 基本运算：选择、投影、并、差、笛卡尔积、更名
- 附加运算：交、自然连接、除、赋值
- 扩展运算：广义投影、聚集、外连接、空值

- 基本运算足以表达任何关系代数的查询，由于所有查询均用基本运算表达可能会显得冗长，因此，又定义了附加运算和扩展运算。附加运算和扩展运算都可以由基本运算组合实现。

附加运算与扩展运算的作用：使运算变得简单

基本运算的讨论

- 六个操作符是基本的:
  - 选择是唯一允许属性上的值相互比较的操作符。因此选择是基本的。
  - 投影是唯一能减少属性数目的操作符，它不能用别的操作符来表达。所以投影是基本的。
  - 并是除积以外唯一能增加元组数目的操作符，积通常还增加属性的数目。假设进行并操作的两个关系是**A**和**B**。注意**A**和**B**必须是同一类型，并且它们的并和它们有相同的属性。为了实现**A**和**B**的积，如果首先改掉**B**的所有属性的名称，然后利用投影减少结果中的属性，使只剩下**A**的属性，结果是又简单地回到了**A**

基本运算的讨论

- 积是除了并外唯一能使属性数目增加的操作符，它不能用别的操作符来表达。所以积是基本的。
- 差不能通过并（因为并永远不能减少元组的数目）或积（同样的原因）或投影（因为投影通常减少属性）。也不能利用选择来表达，因为第二个关系中的值对差很重要，选择中却不然。因此差是基本的。
- 更名是一个其他基本操作都不能替代的唯一能修改关系及属性名字的操作，因此，更名是基本的。

目录

现实世界的三个问题

- 如何保证一个数据（实体）是可识别的？
  - 实体完整性（主码）
- 如何由一个数据找到另一个数据？
  - 参照完整性（主码、外码）
- 如何保证一个数据的取值合理？
  - 用户定义的完整性（约定：域）

- 关系模型的完整性规则是对关系的某种约束条件。实体完整性和参照完整性是关系模型必须满足的完整性约束条件，被称作是关系的两个不变性，应该由关系系统自动支持。

完整性约束–实体完整性

- 一个基本关系通常对应现实世界的一个实体集。
- 实体完整性规则：若属性**A**是关系**R**的主属性（作为主码的属性），则属性**A**不能取空
- 实体完整性规则是针对基本关系而言的

完整性约束–参照完整性

- 现实世界中的实体之间往往存在某种联系，在关系模型中实体及实体间的联系都是用关系来描述的。这样就自然存在着关系与关系间的引用。
- 例子，学生实体集和专业实体集，主码下划线

  学生(学号, 姓名, 性别, 专业号, 年龄)

  专业(专业号, 专业名)

  学生关系中的“专业号”值必须是专业关系中存在的专业，或者为空(没有分派专业)，即学生关系中的某个属性的取值需要参照专业关系的属性

完整性约束-用户定义完整性

- 不同的关系数据库系统根据其应用环境的不同,需要一些特殊的约束条件—用户定义完整性
- 用户定义完整性反映了某具体应用所涉及的数据必须满足的语义要求。
- 如，某属性必须取唯一值，某些属性之间应满足一定的函数关系，某属性的取值范围等
- 系统（**DBMS**）应提供定义和检验这类完整性的机制，而不是由应用程序提供这一功能

目录

空值

- 空值是指信息空缺（**Missing Information)**
- 空值参加运算时会带来复杂性
- 在现实世界中，信息是经常空缺的
- 解决方法是基于空值**(null)**和三值逻辑（**three-valued logic，3VL**，在二值逻辑的基础上再增加一个值）
- 比如零件**P7**的重量是“空”
  - （**a**）我们知道这个零件存在，
  - （**b**）也知道这个零件有重量，
  - （**c**）但我们不知道重量是多少。
- “**null**”是标记，**null**不是值

Null Values 空值

- 方便处理暂时的缺值或未知值
- 含NULL的算术表达式 当作NULL
- 聚集函数跳过 **null values，For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same** 同属性的两个NULL，看作相等
- 空值 理解为暂时不知道（unknown） 比较科学
  - 例如 王大明今年12岁，说他的未婚妻现在为NULL，不是不存在（已经出生了，现在还不知道，将来会知道）

Null Values

- Comparisons with null values return the special truth value: *unknown*。注意：教材上将unknown作为null的一个取值
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic(三值逻辑) using the truth value *unknown*:（与unknown有关的运算）
  - 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*（unknown作为一个真值了）

目录

数据库修改

- 删除：与查询类似，只是不将找出的元组显示给用户，而是将它们删除掉，只能删除元组，不能仅删除某些属性上的值。表达式为：

  $r \leftarrow r - E$， r 为关系，E为表达式

- 插入：插入一个元组或插入一个查询结果（元组集合），表达式为：

  $r \leftarrow r \cup E$

- 更新：只改变元组中的某个值，而不改变元组中所有值，通过广义投影实现。表达式为：

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

例子：删除

- 删除*Perryridge*支行的所有账户

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

- 删除贷款额在0到50之间的所有贷款

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

- 删除位于*Needham*的支行的所有账户

  $r1 \leftarrow \sigma_{branch\text{-}city = \text{“}Needham\text{”}}(account \bowtie branch)$

  $r2 \leftarrow \prod_{branch\text{-}name,\ account\text{-}number,\ balance}(r1)$

  $account \leftarrow account - r2$

例子：插入

- 插入**Smith**在**Perryridge**支行的账户**A-973**上**1200**美元
- 涉及两个关系： ***account***（账户）和***depositor***（存款人）

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

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

例子：更新

- 要付给所有账户**5**%的利息：

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

- 余额**10000**元以上的账户得到**6**%的利息，其他用户得到**5**%的利息

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

  AN, BN, BAL分别表示***account_number, branch_name*** 和 ***balance***

目录

- 关系数据库理论的发展
- 关系模型
  - 关系数据结构
  - 关系代数
  - 扩展关系代数运算
  - 关系的完整性
  - 空值
  - 数据库修改
- 视图

视图

- 实际系统中，处于各种因素（如安全、个性化等）并不希望用户看到整个逻辑模型，可以隐藏一些数据。如通过投影隐藏一些属性。
- 视图（**view**）是用户可以看见的虚关系，不是逻辑模型的一部分，任何给定的实际关系集之上都可以支持大量视图。

视图的定义

- 视图可以通过下面的表达式进行定义：

  create view *v* **as** <query expression>

- 视图可以通过以下两种方式进行定义
  - 通过表达式定义
  - 通过视图定义

例子：视图定义

- 通过表达式（关系）定义：考虑一个想看到关系**loan**中除了**loan_amount**以外所有贷款数据的员工，可以为这个员工定义一个视图***branch-loan*** **：**

  create view *branch-loan* as $\Pi_{\textit{branch-name, loan-number}}$ ***(loan)***

- 通过视图定义：

  create view *new-branch-loan* as $\Pi_{\textit{branch-name}}$ **(*branch-loan*)**

物化视图

- 由于视图是虚关系，因此，一般情况下视图在物理上不进行保存。
- 某些数据库系统允许存储视图关系，但又能够确保定义视图的实际关系改变时，视图是最新的。
- 这样的视图称为物化视图**(materialized view)**，物化视图要付出存储代价。
- 物化视图用于使用频繁的视图或要求相应速度快的视图。代价与效率需要进行权衡

通过视图进行更新与空值

- 通过视图进行数据的更新、插入和删除会带来一些重大问题，原因是用视图表达的对数据库的修改必须要转换为对数据库逻辑模型中实际关系的修改。
- 但视图是个虚表，其数据可能来源于多个关系，并且一些基本表中的数据已经被隐藏，因此，通过视图修改数据会产生一些空值**null**。

例子：视图更新

- 例如，上述建立的视图***branch-loan***，使用该视图的员工进行如下操作：

  branch-loan ← ***branch-loan*** ∪{(**“Perryridge”, L-37**)}

- 由于***branch-loan***是在实际关系上构造的，因此，此插入必须被表示为对关系**loan**的插入。
- 然而，要对**loan**进行插入操作则必须有一个**amount**值，但那位员工不知道有这个属性，不可能插入这个值。

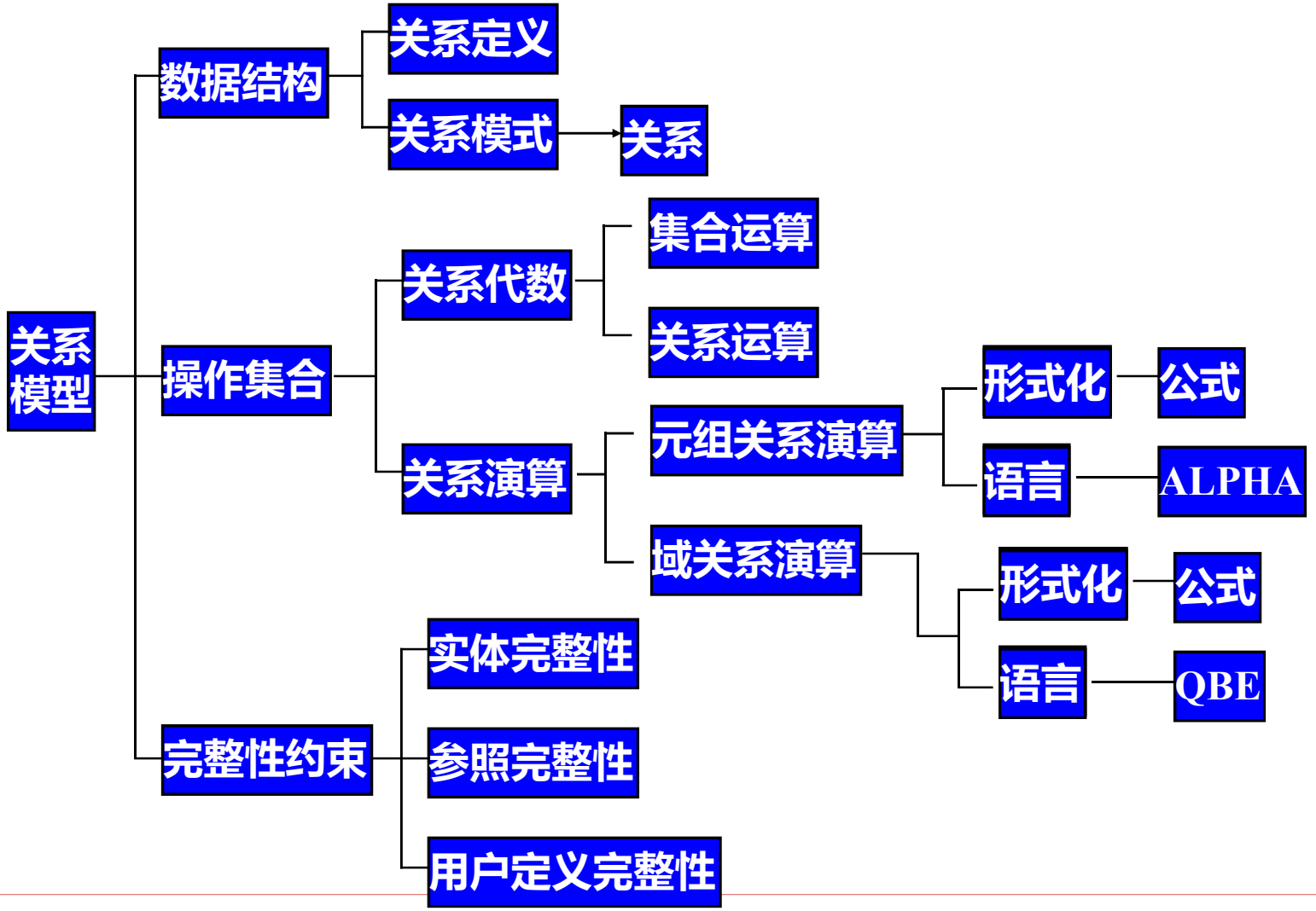
例子：视图更新处理方式

- 有两种合理的处理方式：
  - 拒绝访问
  - 用空值处理非视图属性值，往关系**loan**中插入 **(“Perryridge”, L-37, null)**
- 同时还有多个关系构造的视图，在通过视图更新中很可能会有更新了错误的关系的情况发生
- 鉴于上述情况，对视图关系的修改通常是不允许的
- 通过视图修改数据一直以来都是一个研究的题目。不同的系统在实现上采取的策略也不尽相同。

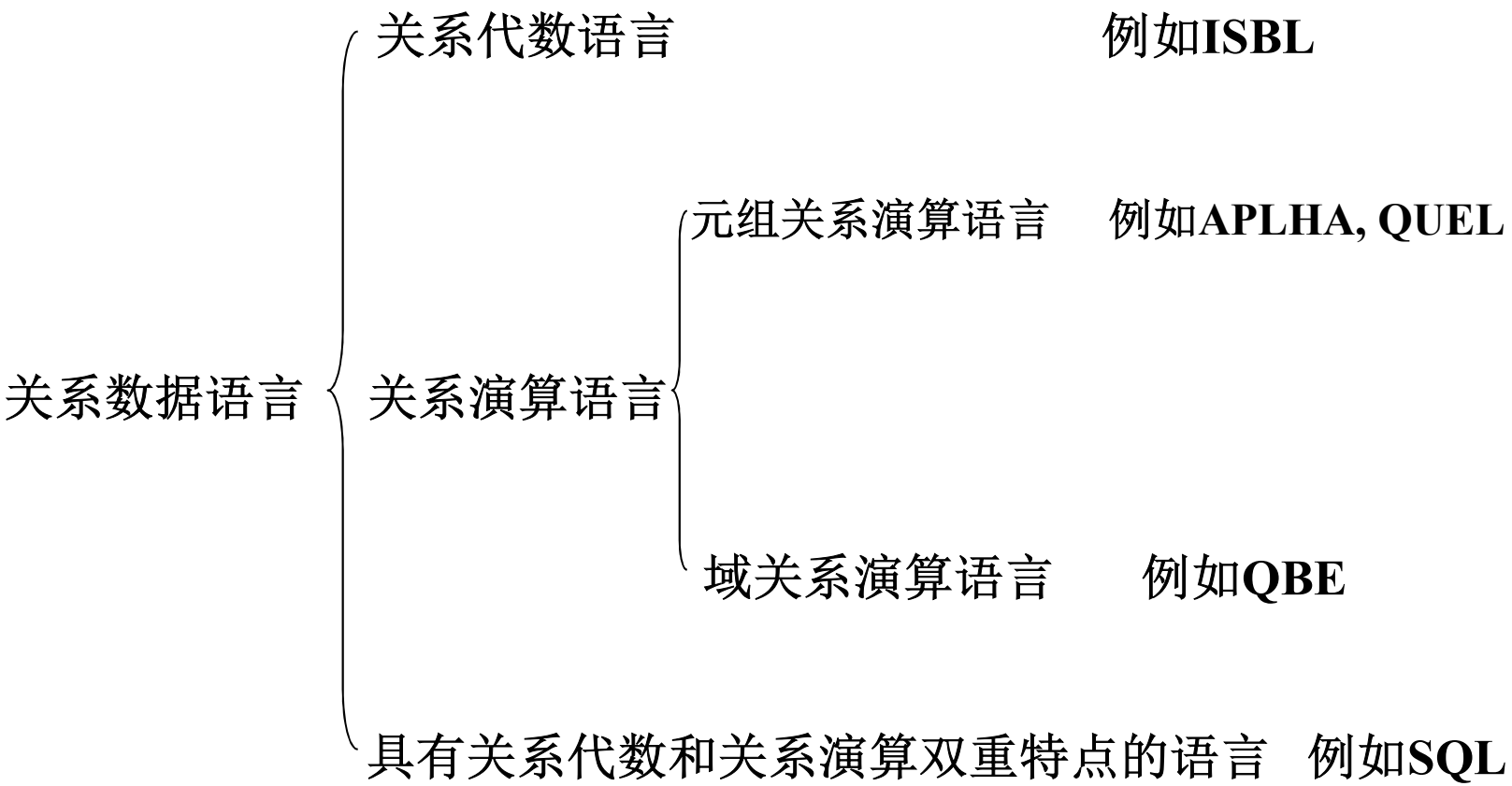
总结

- 关系模型建立在表的集合的基础之上，所有操作都基于表。
- 关系代数定义了一套在表上的运算，且以表作为输出结果。
- 关系代数运算可以分为基本运算、附加运算和扩展运算。
- 完整性约束保证数据库整体的正确性。
- 视图是一种建立在基本关系上的虚关系，是简化查询的有效机制。
- 关系理论的确立标志着数据库系统的基础研究已经接近顶峰。关系数据库中依然存在一些问题需要改进。

附录：关系模型的知识体系



关系数据库语言的分类


关系数据语言
关系代数语言
例如ISBL
关系演算语言
元组关系演算语言
例如APLHA, QUEL
域关系演算语言
例如QBE
具有关系代数和关系演算双重特点的语言
例如SQL