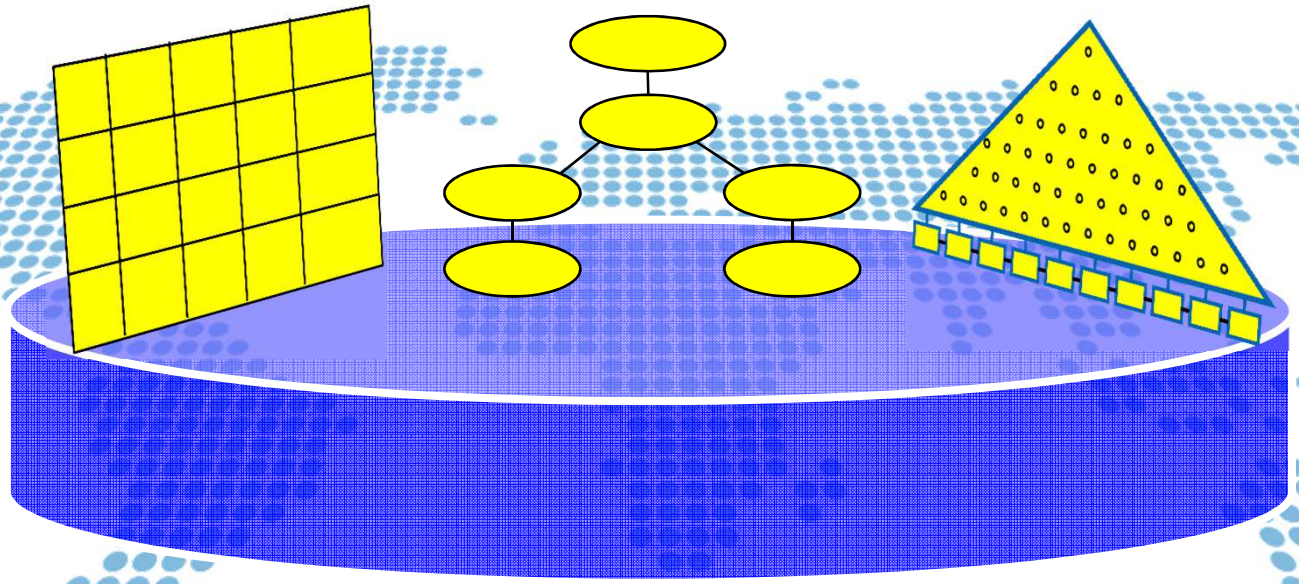
数据库系统

关系模型与SQL（2）

陈世敏

（中科院计算所）



上节课

- SQL 初步
  - 表的增删改
  - 记录的增删改
  - 简单的查询
    - 集合操作：并、交、差
    - 选择行或列：选择、投影
    - 两个关系元组之间的操作：笛卡尔积、连接
    - 其它：重命名、除
- 关系代数

Outline

- 基础SQL复习举例
- 关系演算
- 丰富的SQL Select功能
- 完整性约束

假设下面的关系模式

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- 书上的例子，水手、船、预订三个表

例9：找出所有红色船的名字？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- SQL? 关系代数？

例9：解答

select *bname*
from *Boats*
where *color = 'red';*

$\pi_{bname}(\sigma_{color='red'}(Boats))$

例10：名为Interlake的船都在哪些天被预订？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- SQL? 关系代数？

例10：解答

select *day*
from *Boats, Reserves*
where *Boats.bid=Reserves.bid* and *bname = 'Interlake';*

$\pi_{day}(\sigma_{bname='Interlake'}(Boats) \bowtie_{Boats.bid=Reserves.bid} Reserves)$

例11：找出预订了红色船或绿色船的水手名字？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- SQL? 关系代数？

例11：解答（1）

select *sname*
from *Sailors, Reserves, Boats*
where *Sailors.sid=Reserves.sid*
and *Reserves.bid=Boats.bid*
and *(color='red'* or *color='green');*

$\pi_{sname}(Sailors \bowtie_{Sailors.sid=Reserves.sid} Reserves \bowtie_{Reserves.bid=Boats.bid} \sigma_{color='red' \text{ or } color='green'}(Boats))$

例11：解答（2）

```
(select  sname
from Sailors, Reserves, Boats
where Sailors.sid=Reserves.sid
    and Reserves.bid=Boats.bid
    and color='red')

union

(select  sname
from Sailors, Reserves, Boats
where Sailors.sid=Reserves.sid
    and Reserves.bid=Boats.bid
    and color='green');
```

例12：找出预订了红色船和绿色船的水手名字？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- SQL? 关系代数？

例12：？？

```
select  sname
from Sailors, Reserves, Boats
where Sailors.sid=Reserves.sid
    and Reserves.bid=Boats.bid
    and (color='red' and color='green');
```

不对！同一条船不可能既是红色又是绿色。

这个select语句没有返回的结果！

例12：解答

(select *sname*
from *Sailors, Reserves, Boats*
where *Sailors.sid=Reserves.sid*
and *Reserves.bid=Boats.bid*
and *color='red'*)

intersect

(select *sname*
from *Sailors, Reserves, Boats*
where *Sailors.sid=Reserves.sid*
and *Reserves.bid=Boats.bid*
and *color='green'*);

Outline

- 基础SQL复习举例
- 关系演算
  - 元组关系演算
  - 举例
  - 域关系演算
  - 关系演算与传统关系代数的表达能力
- 丰富的SQL Select功能
- 完整性约束

关系演算（Relational Calculus）

- 关系代数（Relational Algebra）
- 关系演算（Relational Calculus）
  - 把关系运算的结果表达为一个集合
  - 给出集合中元素需要满足的条件
  - 基本形式如下

{集合元素 | 集合元素需要满足的条件}

那么如何表达集合元素和条件？

- 元组关系演算（Tuple Relational Calculus, TRC）
  - 变量是整个记录(元组)

- 域关系演算（Domain Relational Calculus, DRC）
  - 变量是单个列

元组关系演算 TRC （Tuple Relational Calculus）

- 例1：找出评价高于8的所有水手？

$$\{S \mid S \in Sailors \wedge S.rating > 8\}$$

- 注意：
  - 变量是S，代表一个记录
  - 用逻辑表达式来表述条件
    - S是Sailors中的元组
    - 并且
    - S的rating列大于8

元组关系演算 TRC

- 更准确地说，TRC表现为{T|p(T)}
  - T是一个记录/元组
  - p(T)是条件
  - 即满足p(T)的所有记录的集合

- p(T)可能的形式
  - $R \in Rel$
  - 各种比较操作，例如$R.a\ op\ S.b$，$R.a\ op$ 常量
  - 逻辑运算：$p \wedge q, p \vee q, \neg p, p \Rightarrow q$
  - $\exists R(p(R))$
  - $\forall R(p(R))$

例2：找出所有水手的名字和评价？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );
- TRC？

例2：解答

$$\{P|\exists S \in Sailors(P.sname = S.sname \wedge P.rating = S.rating)\}$$

- 注意：
  - 这里实现了投影
  - 定义了一个新的元组变量P
  - 给需要投影的列赋值

例4：预订了103号船水手的名字？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );
- TRC？

例4：解答

$$\{P|\exists S \in Sailors\ \exists R \in Reserves(S.sid = R.sid \wedge R.bid = 103 \wedge P.sname = S.sname)\}$$

- 注意：
  - 连接：用了两个存在表达式，join key
  - 投影

例5：找出预订了红色船的水手的名字？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- TRC？

例5：解答

$$\{P|\exists S \in Sailors\ \exists R \in Reserves \exists B \in Boats(R.sid = S.sid \wedge R.bid = B.bid \wedge B.color =' red' \wedge P.sname = S.sname)\}$$

- 连接3个表

例12：找出预订了红色船和绿色船的水手名字？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- TRC？

例12：解答

$$\{P|\exists S \in Sailors(P.sname = S.sname$$
$$\wedge \exists R1 \in Reserves \exists B1 \in Boats($$
$$R1.sid = S.sid \wedge R1.bid = B1.bid \wedge B1.color =' red')$$
$$\wedge \exists R2 \in Reserves \exists B2 \in Boats($$
$$R2.sid = S.sid \wedge R2.bid = B2.bid \wedge B2.color =' green')$$
$$)\}$$

- 使用了两组Reserves和Boats，分别找到了红色和绿色的船

域关系演算 DRC（Domain Relational Calculus）

- 与TRC相似，主要不同在于变量的表达方式
- 例2：找出所有水手的名字和评价？
  - TRC:

$$\{P|\exists S \in Sailors(P.sname = S.sname \wedge P.rating = S.rating)\}$$

  - DRC:

$$\{< SName, SRating > | \exists SID, SAge(< SID, SName, SRating, SAge > \in Sailors)\}$$

  - 变量是列，元组用尖括号表示，其它基本相似

更多的例子

- 例5：找出预订了红色船的水手的名字？
- TRC
  - $\{P|\exists S \in Sailors\ \exists R \in Reserves \exists B \in Boats($
    $R.sid = S.sid \wedge R.bid = B.bid \wedge B.color =' red'$
    $\wedge P.sname = S.sname)\}$
- DRC
  - $\{< SName >$
    $|\exists SID, SRating, SAge, BID, Bname, BColor, RDay($
    $< SID, SName, SRating, SAge > \in Sailors$
    $\wedge < BID, Bname, BColor > \in Boats$
    $\wedge < SID, BID, RDay > \in Reserves$
    $\wedge BColor =' red')\}$

关系代数与关系演算的表达能力

- 传统关系代数 vs. 关系演算
  - 当查询是安全时，表达能力等价

- 安全的查询
  - 结果是有限的，必须在输入的实例和查询中出现
  - 前面的所有例子都是安全的
  - 反例
    - $\{S|\neg(S \in Sailors)\}$是不安全的
    - 返回所有不在当前水手表中的元组？无限结果

更准确的定义：TRC的安全查询

- 设
  - Q：查询
  - I：数据库的实例
  - Dom(Q,I)：I和Q中出现的所有值的集合

- TRC公式是安全的，如果满足下列条件
  1) 任取Q和I，结果集合仅包含Dom(Q,I)中出现的值
  2) Q中的$\exists R(p(R))$和$\forall R(p(R))$可以根据Dom(Q,I)来判断真假，也就是说：
     - $\exists R(p(R))$：如果$p(r)$为真，那么r仅包含Dom(Q,I)中出现的值
     - $\forall R(p(R))$：如果$p(r)$为假，那么r仅包含Dom(Q,I)中出现的值

Outline

- 基础SQL复习举例
- 关系演算
- 丰富的SQL Select功能
  - 扩展关系代数
  - 单个Select语句
  - 嵌套Select语句
- 完整性约束

扩展关系代数

- 传统关系代数
  - 关系是集合（Set）

- 扩展关系代数
  - 关系是多集（Multi-set）或包(Bag)，允许重复的元素
  - 实际上，在RDBMS中，表允许有重复的元素（当表中没有Primary Key或Unique时），计算的结果允许有重复的元素
  - 所以包更符合RDBMS的实现

包

a	b
10	**11**
20	21
30	31
10	**11**

回顾一下上节讲的关系运算

- 集合操作：并、交、差
- 选择行或列：选择、投影
- 两个关系元组之间的操作：笛卡尔积、连接
- 其它：重命名、除（我们不进一步讨论除）

包的选择σ和投影π

- 选择
  - 提取行
- 投影
  - 提取列

- 涵义没有变，允许多个重复的值

包的连接⋈

- 涵义与集合的连接相同，允许重复

R

a	b
10	11
20	21
30	31
10	11

S

b	c
11	100
11	200

$R \bowtie_{R.b=S.b} S$ =?

a	b	c
10	11	100
10	11	200
10	11	100
10	11	200

包的笛卡尔积X

- 涵义与集合的连接相同，允许重复

R

a	b
10	11
20	21
30	31
10	11

S

b	c
11	100
11	200

R X S = ?

a	R.b	S.b	c
10	11	11	100
10	11	11	200
20	21	11	100
20	21	11	200
30	31	11	100
30	31	11	200
10	11	11	100
10	11	11	200

重命名ρ

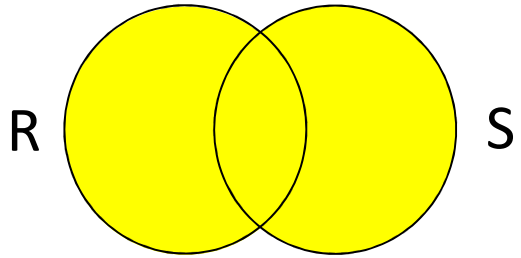
- $\rho_{S(A1,A2,\ldots,An)}(R)$
  - 把关系R重命名为关系S，各列的名字分别是A1, …, An

- 与之前相同

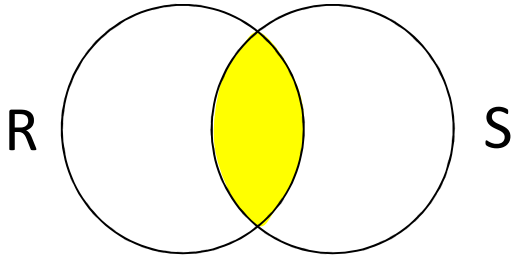
包的并交差

- R={1,1,2}，S={1,2,2,3}

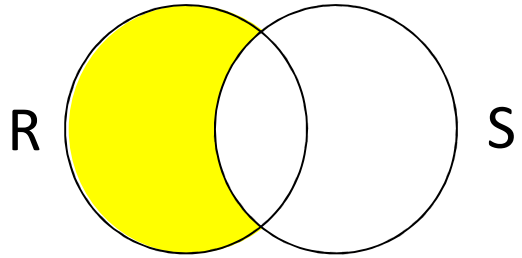
- 并R∪S
  - {1,1,1,2,2,2,3}



- 交R∩S
  - {1,2}



- 差R−S
  - {1}



包的并交差

- 包R和包S进行运算
- 假设对于一个元组t，R中有n_R个t，S中有n_S个t
- 并R∪S
  - 包含$n_R + n_S$个t
- 交R∩S
  - 包含$\min(n_R , n_S)$个t
- 差R−S
  - 包含$\max(0, n_R - n_S)$个t

包的并交差是否满足集合的运算规律？

- 不一定满足，要重新考虑
- 交换律仍然正确
  - R∪S=S∪R
  - R∩S=S∩R
- 但是有些不成立，例如
  - (R∪S)-T = (R-T) ∪ (S-T)
  - 假设R={1}, S={1}, T={1}，左边={1}，右边={}
  - 注意：对于允许重复的情况，一次差和两次差是不同的

包的并交差与SQL的并交差？

- 注意
  - SQL的并交差默认是集合运算
  - 默认去重
  - 所以不是包的并交差

回顾一下上节讲的关系运算

- 集合操作：并、交、差
- 选择行或列：选择、投影
- 两个关系元组之间的操作：笛卡尔积、连接
- 其它：重命名、除（我们不进一步讨论除）

新的运算：去重δ

- 相当于SQL的Distinct
- 如果A是一个包，那么δ(A)就是一个集合
- 例如
  - A={1,1,2,2,2,3}
  - δ(A)={1,2,3}

单个 Select

select 列名,…,列名
from 表,…, 表
where 条件

投影
选择，连接
选择，连接

group by 列名,…,列名
having 条件
order by 列名,…,列名

Aggregation（聚集）

Student

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

- 求统计信息
  - 总学生人数？
  - 平均GPA？
  - 最高GPA？
  - 最低GPA？
  - GPA总和？（这个操作对于这个例子不很实际）

Aggregation: COUNT

Student

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

- 求统计信息
  - 总学生人数？

```
select count(*)
from Student;
```

COUNT(π_{ID}(Student))

注意：COUNT(列)，对一个列统计

Aggregation: AVG

Student

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

- 求统计信息
  - 平均GPA?

```
select avg(GPA)
from Student;
```

$AVG(\pi_{GPA}(Student))$

注意：AVG(列)，对一个列统计

Aggregation: MAX

Student

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

- 求统计信息
  - 最高GPA?

```
select max(GPA)
from Student;
```

$MAX(\pi_{GPA}(Student))$

注意：MAX(列)，对一个列统计

Aggregation: MIN

Student

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

- 求统计信息
  - 最低GPA?

```
select min(GPA)
from Student;
```

$MIN(\pi_{GPA}(Student))$

注意：MIN(列)，对一个列统计

Aggregation: SUM

Student

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

- 求统计信息
  - GPA的总和？

```
select sum(GPA)
from Student;
```

$SUM(\pi_{GPA}(Student))$

注意：SUM(列)，对一个列统计

Aggregation（聚集）

Student

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

- 求统计信息
  - SUM([distinct] 列)
  - COUNT([distinct] 列)
  - AVG([distinct] 列)
  - MAX(列)
  - MIN(列)

聚集运算(列)，对一个列统计

聚集运算(π_{GPA}(Student))

当有distinct时

聚集运算(δ (π_{GPA}(Student)))

Group by：分组，然后统计

Student

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

统计各系的学生人数

```
select Major, count(*) as Number
from Student
group by Major;
```

输出结果

Major	Number
法律	1
经管	1
计算机	3

分组运算γ

Student

ID	Name	Birthday	Gender	Major	Year	GPA

统计各系的学生人数

select *Major*, count(*) as Number
from *Student*
group by *Major*;

$\gamma_{\text{Major, COUNT(ID)}\rightarrow\text{Number}}(\text{Student})$

单独的列：表示分组所使用的列

- 例如：Major

聚集(列)→别名：表示在每个分组上进行的统计

- 例如：COUNT(ID)→ Number

Group by + where

$\gamma_{Major,\ COUNT(ID)\rightarrow Number}(\sigma_{Year>=2013\ and\ Year<=2014}(Student))$

Student

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

统计各系2013-2014年入学的学生人数

```
select Major, count(*) as Number
from Student
where Year >= 2013 and Year <= 2014
group by Major;
```

输出结果

Major	Number
经管	1
计算机	2

Group by 多个列的分组

Student

$\gamma_{Major,Year,COUNT(ID)\rightarrow Number}(Student)$

ID	Name	Birthday	Gender	Major	Year	GPA
131234	张飞	1995/1/1	男	计算机	2013	85
145678	貂蝉	1996/3/3	女	经管	2014	90
129012	孙权	1994/5/5	男	法律	2012	80
121101	关羽	1994/6/6	男	计算机	2012	90
142233	赵云	1996/7/7	男	计算机	2013	95

统计各系各年的学生人数

```
select Major, Year, count(*) as Number
from Student
group by Major, Year;
```

输出结果

Major	Year	Number
法律	2012	1
经管	2014	1
计算机	2012	1
计算机	2013	2

Group by的输出列

- 包含两种情况（对应于γ的下标的两种情况）
  - 用于分组的列
  - 聚集

- 注意：除去这两种情况，其它的列是没有意义的

例如: 对于组而言，Name是没有意义的，非法的

```
select Major, Year, Name, count(*) as Number
from Student
group by Major, Year;
```

Group by和Distinct

- Distinct是Group by的一个特例
- 具体而言，假设R(A1,A2,…,Ak)
- 那么

$$\delta(R) = \gamma_{A1,A2,\ldots,Ak}(R)$$

对R去重

把R的所有列的组合进行分组
组名只保留一个，自动去重了！

Group by和Distinct

- Distinct是Group by的一个特例
- 例如

```
select distinct GPA
from Student;
```

$\delta(\pi_{GPA}(Student))$

也可以写为

```
select GPA
from Student
group by GPA;
```

$\gamma_{GPA}(Student)$

Group by 与选择

- 选择在分组之前
  - 采用where子句，如前面的例子
- 选择在分组之后
  - 采用having子句，下面介绍

Having：在group by 基础上选择

统计各系2013-2014年入学的学生人数，只输出多于1人的系

```
select Major, count(*) as Number
from Student
where Year >= 2013 and Year <= 2014
group by Major
having Number > 1;
```

输出结果

Major	Number
计算机	2

注意：having中只能出现分组之后在组上有意义的列

Order by: 排序

$\tau_{Number}(\gamma_{Major,\ COUNT(ID)\rightarrow Number}(\sigma_{Year>=2013\ and\ Year<=2014}(Student)))$

统计各系2013-2014年入学的学生人数，并按照人数排序

```
select Major, count(*) as Number
from Student
where Year >= 2013 and Year <= 2014
group by Major
order by Number;
```

输出结果

Major	Number
经管	1
计算机	2

desc (descending 减少)表示从大到小排序
asc (ascending 增加) 表示从小到大排序

SQL Select

select 列名,...,列名	投影π，聚集，去重δ
from 表,..., 表	选择σ，连接⋈
where 条件	选择σ，连接⋈
group by 列名,...,列名	分组统计γ
having 条件	分组后选择
order by 列名,...,列名	结果排序τ

问题？

Outline

- 基础SQL复习举例
- 关系演算
- 丰富的SQL Select功能
  - 扩展关系代数
  - 单个Select语句
  - 嵌套Select语句
- 完整性约束

嵌套查询

- 嵌套：一个查询Select中包含另一个Select
  - 内层的Select被称作子查询
  - 外层的Select被称作主查询

- 例如：找出预订了103号船水手的名字？

```
select S.sname
from Sailors S
where S.sid  in (select R.sid
                 from Reserves R
                 where R.bid=103);
```

From 中可以省略as

嵌套查询

- 例如：找出没有预订103号船水手的名字？

```
select  S.sname
from Sailors S
where S.sid  not in (select R.sid
                     from Reserves R
                     where R.bid=103);
```

嵌套查询

- 从概念上讲，实际上相当于嵌套循环
  - 主查询是外层循环，子查询是内层循环
  - 对于主查询的每个记录，执行整个子查询
- 例如：找出预订了103号船水手的名字？

```
select  S.sname
from Sailors S
where S.sid  in (select R.sid
                 from Reserves R
                 where R.bid=103);
```

- 对于Sailors中的每条记录，执行子查询，然后判断sid是否在子查询的结果中

嵌套查询

- 例如：找出预订了103号船水手的名字？

```
select  S.sname
from Sailors S
where S.sid  in (select R.sid
                 from Reserves R
                 where R.bid=103);
```

- 当然在这里似乎不需要对子查询执行多次，因为它不依赖于外层的主循环
  - 所以可以提取这种静态的子查询结果，反复使用
  - RDBMS会通过SQL的等价变形来完成这种优化（今后的课会提到）

相关嵌套查询

- 更复杂的情况是当子循环与主循环相关时
- 例如：找出预订了103号船水手的名字？

```
select  S.sname
from Sailors S
where exists (select *
                  from Reserves R
                  where R.bid=103 and R.sid = S.sid);
```

子查询引用了主查询

- Exists检查集合是否为空
- 那么，这里对于Sailors的每条记录，检查子查询的结果是否为空

相关嵌套查询

- 例如：找出没有预订了103号船水手的名字？

```
select  S.sname
from Sailors S
where not exists (select *
                  from Reserves R
                  where R.bid=103 and R.sid = S.sid);
```

集合比较操作

- in：属于
- not in：不属于
- exists：存在
- not exists：不存在
- unique：无重复元素
- not unique：有重复元素

op ANY

例13：找出评分高于某个名字包含Tom的水手的水手名字

```
select  S.sname
from Sailors S
where S.rating > any (select S2.rating
                      from Sailors S2
                      where S2.sname like '%Tom%');
```

如果名字包含Tom的水手不存在，那么>any返回false，主查询结果为空

op ALL

例14：找出评分高于所有名字包含Tom的水手的水手名字

```
select  S.sname
from Sailors S
where S.rating > all (select S2.rating
                      from Sailors S2
                      where S2.sname like '%Tom%');
```

例15：找出评分最高的水手名字

```
select  S.sname
from Sailors S
where S.rating >= all (select S2.rating
                       from Sailors S2);
```

直接比较不加ANY/ALL

- 当子查询采用了聚集而返回唯一结果时
- 就可以不加ANY、ALL，而直接比较
- 例15：找出评分最高的水手名字

```
select  S.sname
from Sailors S
where S.rating =  (select max(S2.rating)
                   from Sailors S2);
```

例16：找出预订了所有船只的水手的名字

```
select  sname
from Sailors
where not exists (
    (select Boats.bid from Boats)
    except
    (select Reserves.bid from Reserves
     where Reserves.sid=Sailors.sid)
);
```

差的结果为空

所有船只

求差

Sid预订的所有船只

下面我们练习一下

例17：计算评价高于8的水手的平均年龄？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- SQL?

例17：解答

select *avg(age)*
from *Sailors*
where *rating* > 8;

例18：计算最年长水手的年龄？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- SQL?

例18：解答

select *max(age)*
from *Sailors*;

例19：计算最年长水手的名字和年龄？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

- SQL?

例19：？？

select *sname, max(age)*
from *Sailors*;

如果select使用了聚集，那么只能出现聚集或者分组的列

上述查询中sname是非法的

例19：解答

select *S.sname, S.age*
from *Sailors S*
where *S.age* = (select max(*S2.age*) from *Sailors S2*);

子查询使用了聚集，返回了唯一的结果

例20：找出比评分为10的最年长的水手年龄还要大的水手的名字？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );
- SQL?

例20：解答

```
select  S.sname
from Sailors S
where S.age > (select max(S2.age)
                from Sailors S2
                where S2.rating=10);
```

例20：解答(2)

```
select  S.sname
from Sailors S
where S.age > all (select S2.age
                   from Sailors S2
                   where S2.rating=10);
```

注意：这里如果all误写为any就错了

例21：找出包含至少两个水手的评价等级？

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );
- SQL?

例21：解答

```
select  rating
from Sailors
group by rating
having count(*) >= 2;
```

例22：对于至少包含两个水手的评价等级，找出至少18岁的水手的平均年龄

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );
- SQL?

例22：解答

```
select  S.rating, avg(s.age) as average
from Sailors S
where S.age >= 18
group by S.rating
having S.rating in (select  S2.rating
                    from Sailors S2
                    group by S2.rating
                    having count(*) >= 2);
```

Outline

- 基础SQL复习举例
- 关系演算
- 丰富的SQL Select功能
- 完整性约束

完整性约束（Integrity Constraint）

- 已经学习的完整性约束
  - 域约束（Domain Constraint）：规定列的类型
  - 主键约束（Primary Key）
  - 候选键约束（Unique）
  - 外键约束（Foreign Key … References …）
    - 又称作引用完整性（Referential Integrity）
  - Not Null

RDBMS怎样保证约束正确?

- 进一步介绍
  - 表约束（Check）
  - 断言（Assertion）
  - 触发器（Trigger）
- 合法的实例（Legal Instance）：满足完整性约束的实例

保证完整性约束

- 在记录的增删改操作时
  - insert, delete, update
- 数据库系统会自动地检查完整性约束是否成立
- 如果下述约束不成立，那么就拒绝执行
  - 域约束（Domain Constraint）
  - 主键约束（Primary Key）
  - 候选键约束（Unique）
  - Not Null

☞外键约束？

记录增删改：外键约束

TakeCourse

Couse ID	Student ID			
7001	131234			
7012	145678			
7005	129012			

Student

ID	Name				
131234	张飞				
145678	貂蝉				
129012	孙权				

- 什么操作可能破坏外键约束呢？
  - Student: update, delete
  - TakeCourse: insert, update
- 什么操作肯定不会破坏外键约束呢？
  - Student: insert
  - TakeCourse: delete

记录增删改：外键约束

TakeCourse

Couse ID	Student ID			
7001	131234			
7012	145678			
7005	129012			

Student

ID	Name				
131234	张飞				
145678	貂蝉				
129012	孙权				

- TakeCourse表的修改
  - delete: 肯定不会破坏参照完整性
  - insert, update：
    - 如果StudentID取值在Student表中不存在
    - 那么RDBMS将拒绝操作

记录增删改：外键约束

TakeCourse

Couse ID	Student ID			
7001	131234			
7012	145678			
7005	129012			

Student

ID	Name				
131234	张飞				
145678	貂蝉				
129012	孙权				

- Student表的修改
  - insert: 肯定不会破坏参照完整性
  - delete, update：删除或修改ID，如果TakeCourse中存在引用这个ID的记录，可以有3种解决方法
    - No Action：拒绝操作
    - Set Null: 把外键设为空（当然外键不能是主键的一部分）
    - Cascade: 删除或修改对应的TakeCourse中的记录

No Action（当不特殊说明时，采用的方法）

- 没有特殊说明，就是No Action
- 当检测出问题，就拒绝执行

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year,
        Semester enum('Spring', 'Summer', 'Fall'),
        Grade float,
        primary key (CourseID, StudentID),
        foreign key (CourseID) references Course(ID),
        foreign key (StudentID) references Student(ID)
        );
```

Set Null和Cascade

- Set null: 设为空
- Cascade: 执行相应的修改

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year,
        Semester enum('Spring', 'Summer', 'Fall'),
        Grade float,
        foreign key (StudentID) references Student(ID)
                on delete set null
                on update cascade
        );
```

当Student(ID)被删除，则设为空；当被修改，则相应地修改StudentID

Set Null和Cascade

- Set null: 设为空
- Cascade: 执行相应的修改

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year,
        Semester enum('Spring', 'Summer', 'Fall'),
        Grade float,
        foreign key (StudentID) references Student(ID)
                on delete cascade
                on update set null
        );
```

当Student(ID)被删除，则删除相应记录；当被修改，则设为空

记录增删改：外键约束，总结一下

- 对Primary Key所在的表（例如：Student）
  - Insert: ✓
  - Delete, update: no action / set null / cascade
- 对Foreign Key所在的表（例如：TakeCourse）
  - Insert, update: 如果不符合，就拒绝
  - Delete: : ✓

完整性约束（Integrity Constraint）

- 已经学习的完整性约束
  - 域约束（Domain Constraint）：规定列的类型
  - 主键约束（Primary Key）
  - 候选键约束（Unique）
  - 外键约束（Foreign Key … References …）
    - 又称作引用完整性（Referential Integrity）
  - Not Null
- 进一步介绍
  - 表约束（Check）
  - 断言（Assertion）
  - 触发器（Trigger）

RDBMS怎样保证约束正确?

表约束（Check）

- SQL支持使用Check设置表约束(Table Constraint)条件
- 用法

Check (*条件*)

条件可以是where子句中允许的任何条件

基于单个属性的Check(条件)

- 例如：

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year Check( Year>=1900 ),
        Semester enum('Spring', 'Summer', 'Fall'),
        Grade float Check( Grade>=0 and Grade<=100 )
        );
```

如果Check只涉及一列，可以直接写在列的声明之后

检查基于单个属性的Check(条件)

- 例如：

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year Check( Year>=1900 ),
        Semester enum('Spring', 'Summer', 'Fall'),
        Grade float Check( Grade>=0 and Grade<=100 )
        );
```

当insert时，进行检查

当update时，如果规定了check的列被修改了，那么就对新值检查

用Check(条件)表达enum

- 例如：

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year,
        Semester varchar(20)
                Check(Semester in ('Spring', 'Summer', 'Fall')),
        Grade float
        );
```

当某些数据库系统不支持enum时，可以这样表达

对于整个表的Check(条件)

- 例如：

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year,
        Semester enum('Spring', 'Summer', 'Fall'),
        Grade float,
        Check( not(Semester='Summer' and Grade<60) )
        );
```

在这个例子中，条件涉及多个属性，夏季学期的课程成绩不低于60。

Check(条件)

- 例如：

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year,
        Semester enum('Spring', 'Summer', 'Fall'),
        Grade float,
      Check( not exists (
              select CourseID, Year, Semester, count(*) as Number
              from TakeCourse
              where Grade >= 90
              group by CourseID, Year, Semester
              having Number > 10))
        );
```

在这个例子中，条件包含子查询，保证每门成绩至少90分的人数不超过10人。

检查表约束

- 对于这种一般的Check
  - 当insert或update时，运行Check中的条件
- 注意
  - delete时，不检查Check中的条件
  - 修改其它的表，也不检查这个表的Check条件
- 代价可能很大

约束命名

- 可以在约束前面使用Constraint 名字来命名

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year,
        Semester enum('Spring', 'Summer', 'Fall'),
        Grade float,
        constraint TC1 foreign key (StudentID) references Student(ID)
            on delete cascade
            on update set null
        );
```

延迟执行（Deferred）

- 可以推迟约束的执行

  Set constraint *TC1* deferred;

  这时，RDBMS将在一个事务的结束时才检查这个约束，一个事务通常包含多个语句
  - 例如，先更新Student表，再更新TakeCourse，最后检查完整性

- 可以恢复约束的立即执行

  Set constraint *TC1* immediate;

  这个是默认的，RDBMS对于每个语句都检查约束

完整性约束（Integrity Constraint）

- 已经学习的完整性约束
  - 域约束（Domain Constraint）：规定列的类型
  - 主键约束（Primary Key）
  - 候选键约束（Unique）
  - 外键约束（Foreign Key … References …）
    - 又称作引用完整性（Referential Integrity）
  - Not Null
- 进一步介绍
  - 表约束（Check）
  - 断言（Assertion）
  - 触发器（Trigger）

RDBMS怎样保证约束正确?

断言(Assertion)

- 断言是Schema的一部分
  - 用create, drop

- 创建断言

  create assertion *断言名* check (*条件*);

  条件可以是where允许的任何条件

- 删除断言

  drop assertion *断言名*;

回到前面的例子

- 例如：

```
create table TakeCourse (
        CourseID integer,
        StudentID integer,
        Year year,
        Semester enum('Spring', 'Summer', 'Fall'),
        Grade float,
      Check( not exists (
              select CourseID, Year, Semester, avg(Grade) as ag
              from TakeCourse
              group by CourseID, Year, Semester
              having ag < 60))
        );
```

我们改了一下，要求每门课的平均分至少为60

这个Check有什么问题？

Check在delete时不执行！

- Check只在insert和update时执行
- 所以，当delete时，一定思考一下check条件是否可能出问题！
  - 前面的例子不会，因为delete总是把人数变少
  - 这个例子有可能，因为delete可能是avg变小！
- 这种情况应该使用assertion

使用Assertion

- 例如：

```
create assertion TCassert
        Check( not exists (
                select CourseID, Year, Semester, avg(Grade) as ag
                from TakeCourse
                group by CourseID, Year, Semester
                having ag < 60));
```

数据库会保证对于TableCourse的任何修改，都检查这个assertion

代价比表约束更大！

多个表的约束：使用Assertion

- 例如：

```
create assertion CSassert
        Check( not exists (
                select CourseID, Year, Semester, avg(Grade) as ag
                from TakeCourse, Student
                 where TakeCourse.StudentID=Student.ID
                       and Student.Major='计算机'
                group by CourseID, Year, Semester
                having ag < 80));
```

每门课中计算机系学生的平均成绩不低于80分。

多个表的约束：使用Assertion

- 例如：

```
create assertion CSassert
        Check( not exists (
                select CourseID, Year, Semester, avg(Grade) as ag
                from TakeCourse, Student
                 where TakeCourse.StudentID=Student.ID
                       and Student.Major='计算机'
                group by CourseID, Year, Semester
                having ag < 80));
```

注意：虽然语法上允许把这个Check约束放到TakeCourse表的create table中成为表约束，但是，如此一来，Student表的任何修改，都不会检查这个约束，就有问题了。

断言 vs. 表约束

	关于单个表，在delete时不检查	关于单个表，在delete时需要检查	关于多个表
表约束	✓		
断言	✓	✓	✓

触发器（Trigger）

- 触发器与完整性约束相关
- 一个触发器由三部分组成
  - 事件（Event）
  - 条件（Condition）
  - 动作（Action）
- 当出现了给定的事件，检查给定的条件。如果条件满足，那么执行动作。
- 主动数据库（Active Database）：主要指采用触发器的数据库

触发器的例子

```
create trigger ImproveTrigger
after update of GPA on Student
referencing
      old row as OldTuple
      new row as NewTuple
for each row
when ( NewTuple.GPA > OldTuple.GPA*1.1 )
      insert into StudentImprovement
      values (NewTuple.ID, OldTuple.GPA, NewTuple.GPA);
```

事件：在修改了GPA之后

条件：如果GPA提高超过10%

动作：记录到StudentImprovement表中

触发器的设计

```
create trigger ImproveTrigger
after update of GPA on Student
referencing
      old row as OldTuple
      new row as NewTuple
for each row
when ( NewTuple.GPA > OldTuple.GPA*1.1 )
      insert into StudentImprovement
      values (NewTuple.ID, OldTuple.GPA, NewTuple.GPA);
```

事件：

可以是**before** 或 **after**，后面的具体修改可以是

- **update of** *列名* **on** *表名*
- **insert on** *表名*
- **delete on** *表名*

触发器的设计

```
create trigger ImproveTrigger
after update of GPA on Student
referencing
      old row as OldTuple
      new row as NewTuple
for each row
when ( NewTuple.GPA > OldTuple.GPA*1.1 )
      insert into StudentImprovement
      values (NewTuple.ID, OldTuple.GPA, NewTuple.GPA);
```

引用旧值和新值：

当然与具体的修改操作有关，

insert没有旧值，delete没有新值

触发器的设计

```
create trigger ImproveTrigger
after update of GPA on Student
referencing
      old row as OldTuple
      new row as NewTuple
for each row
when ( NewTuple.GPA > OldTuple.GPA*1.1 )
      insert into StudentImprovement
      values (NewTuple.ID, OldTuple.GPA, NewTuple.GPA);
```

一个修改语句可能修改多个记录

For each row： 对于每个记录都分别执行一次条件+动作

For each statement（默认）： 对于整个修改语句，只进行一次条件+动作

触发器的设计

```
create trigger ImproveTrigger
after update of GPA on Student
referencing
      old row as OldTuple
      new row as NewTuple
for each row
when ( NewTuple.GPA > OldTuple.GPA*1.1 )
      insert into StudentImprovement
      values (NewTuple.ID, OldTuple.GPA, NewTuple.GPA);
```

条件可以缺省，如果缺省，那么总是真
如果存在when子句，那么就判断条件是否为真

触发器的设计

```
create trigger ImproveTrigger
after update of GPA on Student
referencing
      old row as OldTuple
      new row as NewTuple
for each row
when ( NewTuple.GPA > OldTuple.GPA*1.1 )
    insert into StudentImprovement
    values (NewTuple.ID, OldTuple.GPA, NewTuple.GPA);
```

动作：

可以是一组语句，用Begin和End括起来

再举一个例子

```
create trigger GlobalImproveTrigger
after update of GPA on Student
referencing
      old table as OldStuff
      new table as NewStuff
when ( 80 < (select avg(GPA) from NewStuff))
declare
    oldgpa float;
    newgpa float;
begin
    select avg(GPA) into oldgpa from OldStuff ;
    select avg(GPA) into newgpa from NewStuff ;
    insert into GlobalStudentImprovement
      values (now( ), oldgpa, newgpa);
end
```

引用整个表

声明变量

多个语句

触发器的使用

- 如果可以用简单约束或断言，尽量不用触发器
- 同一个事件最好不要定义多个触发器
  - 当这个事件发生时，没有明确规定触发器的执行顺序
  - 可能会产生问题

小结

- 基础SQL复习举例
- 关系演算
  - 元组关系演算
  - 域关系演算
  - 关系演算与传统关系代数的表达能力
- 丰富的SQL Select功能
  - 扩展关系代数
  - 单个Select语句： aggregation, group by, having, order by
  - 嵌套Select语句: in, exists, unique, op ANY/ALL
- 完整性约束
  - domain, unique, primary key, not null
  - foreign key, 执行
  - check
  - Assertion, trigger