第三章 SQL

一、SQL数据定义

1.基本类型

- 字符串类型
  - 定长字符串char(n)：不论字符串长度是多少都按n存
  - 变长字符串varchar(n)：按字符串的实际长度存
  - 定长字符串和变长字符串的最大长度都为n
- 数字类型
  - numeric(p , d)：p表示数据长度，d表示小数点后要保留几位

2.基本模式定义

（1）创建

A是属性，D是属性的域

- **create table**命令的**通用形式**是:

create table r
$(A_1 \quad D_1,$
$A_2 \quad D_2,$
$\cdots,$
$A_n \quad D_n,$
$<$完整性约束$_1>,$
$\cdots,$
$<$完整性约束$_k>);$

- **create table命令后面用分号结束**

- SQL支持的部分**完整性约束**
  - $\boldsymbol{primary\ key}(A_{j_1}, A_{j_2}, \ldots, A_{j_m})$:主码属性必须非空且唯一
  - $\boldsymbol{foreign\ key}(A_{k_1}, A_{k_2}, \ldots, A_{k_n})\ \boldsymbol{references}$ s: $foreign\ key$声明表示关系中任意元组在属性$(A_{k_1}, A_{k_2}, \ldots, A_{k_n})$上的取值必须对应于关系s中某元组在主码属性上的取值
  - $\boldsymbol{not\ null}$: 一个属性上的not null约束表明在该属性上不允许空值

(2)删除所有元组

- **delete from**：
  - **delete from** student：从student关系中**删除所有元组**
- **drop table**：
  - **drop table** student：不仅**删除student关系中的所有元组**，还**删除student关系的模式**。一旦student被去掉，除非用create table命令重建student，否则没有元组可以插入到student中

(3)对于表的修改

- **alter table** r **add** A D：
  - 使用该命令为已有关系**增加属性**;
  - r是现有关系的名字，A是待添加属性的名字，D是待添加属性的域
- **alter table** r **drop** A：
  - 使用该命令从关系中**去掉属性**
  - R是现有关系的名字，A是关系的一个属性的名字

二、SQL查询的基本结构

select $A_1, A_2, \ldots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

$$\Pi_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

A_i代表属性
r_i代表关系
P是一个谓词
当i=1时称为单关系查询，
否则称为多关系查询

SQL查询的结果是一个关系

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 70000;
```

如果 *instructor* 关系如图 2-1 所示，那么上述查询的结果关系如图 3-4

SQL 允许在 **where** 子句中使用逻辑连词 **and**、**or** 和 **not**。逻辑连词的运算对象可以是包含比较运算符 <、<=、>、>=、= 和 <> 的表达式。SQL 允许我们使用比较运算符来比较字符串、算术表达式以及特殊类型，如日期类型。

图

1.select子句

- 后面跟查询的内容，用投影来表示，广义的投影除了对基本的列进行投影之外，还能对列进行的四则运算的结果进行投影，也可以只投影出一个常量的值

```
select loan_number, branch_name,
       amount * 100, 100
from loan
```

- 一个最基本的SQL查询它至少要有**select**和**from**子句：

```
select name
from instructor
```

- 在SQL语句当中，**属性名**不区分大小写 *Name ≡ NAME ≡ name*

- 关键字：
  - **distinct**:去除重复数据
  - **all**:保留重复数据

```
select distinct dept_name
from instructor
```

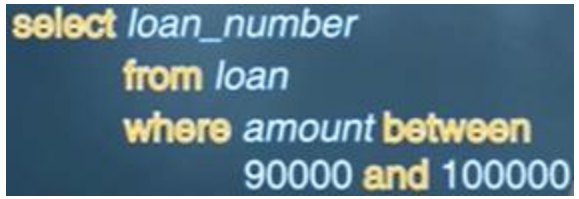
- **select ***：查询当前关系表中的所有属性

```
select *
from instructor
```

查询instructor关系表当中的所有属性

2.where子句

- 对多个条件的组合可以用到逻辑运算：**and**，**or**，**not**
- **between…and…**：



3.from子句

- 与笛卡尔乘积的关系：

select *
from *instructor, teaches*　　等价于　　*instructor X teaches*

- 当from子句中有n个表的时候，在where子句后至少要添加n-1个等值链接条件

三、SQL附加的基本运算

- 更名运算、自然连接

1.1更名运算 as(可省略)

- 当as在select子句中使用时:

在查询的结果集中将属性名loan_number更名为loan_id.

```
select customer_name, borrower.loan_
       number as loan_id, amount
from borrower, loan
where borrower.loan_number =
      loan.loan_number
```

可以省略,用空格代替.

1.2元组变量 as(可省略)

- 当as在from子句中使用时：

```
select customer_name, T.loan_number, S.amount
from borrower as T, loan as S
where T.loan_number = S.loan_number
```

  - 相当于在内存空间当中分别复制了一个内容与brrower表完全相同，而名字为T的表；内容于loan表完全相同，而名字为S的表。
  - 之后在T表和S表中进行查询工作

2.字符串操作

（1）模糊匹配：**like**

- 匹配符：
  - %：可以匹配任意长度的子串
  - _：可以匹配长度为1 的字符串
    - ```
      select name
      from instructor
      where name like '%dar%'
      ```
    - 如果要查找的字符串里面有百分号，可以用**escape**定义转义字符

      Match the string “100 %”

      like ‘100 \%' **escape** '\'
  - Underscore(_):可以匹配任意一个字符
      ```

2.字符串操作

（2）其他

字符串连接: ||

提取子串

计算字符串长度

大小写转换：**upper**(s)将字符串s转换为大写

lower(s)将字符串s转换为小写

去掉字符串后面的空格：**trim**(s)

3.排序 order by

- ```
  select distinct name
  from   instructor
  order by name
  ```
- 升序排列 **asc**(默认)，降序排列 **desc**

  ```
  order by name desc
  ```
- **order by** 子句必须放在**select**子句的最后一部分
- 可以多属性排序

  ```
  order by  dept_name, name
  ```
  ```

四、重复元组

- 重复元组
  用select语句查询的结果是一个多重集

Example: Suppose multiset relations r_1 (A, B) and r_2 (C) are as follows:

$r_1 = \{(1, a)\ (2, a)\}$ $\quad$ $r_2 = \{(2), (3), (3)\}$

Then $\Pi_B(r_1)$ would be {(a), (a)}, while $\Pi_B(r_1) \times r_2$ would be

{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)}

五、集合运算

- **union**(∪)、**intersect**(∩)、**except**(−)
- 参与运算的两个集合需要具有相同或相容的结构
  - 两个集合属性的个数是相同的
  - 数据类型是相容的或者是完全相同的

- Find courses that ran in Fall 2009 or in Spring 2010

```
(select course_id from section where sem = 'Fall' and year = 2009)
 union
(select course_id from section where sem = 'Spring' and year = 2010)
```

- Find courses that ran in Fall 2009 and in Spring 2010

```
(select course_id from section where sem = 'Fall' and year = 2009)
 intersect
(select course_id from section where sem = 'Spring' and year = 2010)
```

- Find courses that ran in Fall 2009 but not in Spring 2010

```
(select course_id from section where sem = 'Fall' and year = 2009)
 except
(select course_id from section where sem = 'Spring' and year = 2010)
```

六、聚集函数

- 聚集函数有：求平均值(**avg**)、求最小值(**min**)、求最大值(**max**)、求和(**sum**)、求值的个数[按行进行查询](**count**)
- 聚集函数不允许嵌套
- **group by**子句、**having**子句

```
select count (distinct ID)
from teaches
where semester = 'Spring' and year = 2010
```

统计2010春季教师的数量，去除相同ID的元组

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

统计course表的行数

- 在聚集函数中使用group by子句（分组聚集）

select后的属性必然出现在group by后，（聚集函数除外）

```
select dept_name, avg (salary) as avg_salary
from instructor
group by dept_name;
```

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

instructor关系的元组按照dept_name属性分组
(group by子句的作用)

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

查询“找出每个系的平均工资”的结果关系

- **having子句**
  - **select** *dept_name*, **avg** (*salary*)
    from *instructor*
    group by *dept_name*
    having avg (*salary*) > 42000;

  - **where子句与having子句的比较：**
    - **where子句(在group by子句之前判断)：只能对自然属性的列进行条件判断**
    - **having子句(在group by子句之后判断)：不仅能对自然属性的列进行条件判断，还能对计算出来的结果进行判断**

七、空值

- 在算数运算中：一切与null相关的算数运算的结果都为null
- 在比较运算中：结果为unknow
- 在集合运算中：
  - 如果where子句谓词对一个元组计算出false或unknown，那么该元组不能被加入到结果集中
  - SQL在谓词中使用特殊的关键词null测试空值

```
select name
from instructor
where salary is null
```

找出instructor关系中salary为空值的所有教师

  - 在聚集函数的操作过程中忽略空值
    - count(*)所有属性列不同时为空时计数加一

八、嵌套子查询

- **in**

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id in (select course_id
                      from section
                      where semester = 'Spring' and year= 2010);
```

查询既在2010春季学期开课又在2009年秋季学期开课的课程

- **not in**

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id  not in (select course_id
                      from section
                      where semester = 'Spring' and year= 2010);
```

查询既在2009年秋季学期开课但不在2010春季学期开课的课程

九、集合比较

- **select distinct** *T.name*
  from *instructor* **as** *T*, *instructor* **as** *S*
  where *T.salary* > *S.salary* **and** *S.dept name* = 'Biology';

等价

select *name*
from *instructor*
where *salary* > **some** (**select** *salary*
from *instructor*
where *dept name* = 'Biology');

- some关键字
  - some(a)：集合a中的某一个值
  - $= \boldsymbol{some}$等价于in

    $\neq \boldsymbol{some}$不等价于not in
- all关键字
  - all(a)：集合a中的所有值
  - $= \boldsymbol{all}$不等价于in

    $\neq \boldsymbol{all}$等价于not in

十、关系测试

- 空关系测试 **exists**
  - **exists** $r \Leftrightarrow r \neq \varnothing$
  - **not exists** $r \Leftrightarrow r = \varnothing$

exists的操作是判断子查询的返回的值是否是空集

- 是否有重复元组的测试 **unique**
  - 如果没有重复值，那么**unique**的判断为真
  - 如果有重复的值，那么**unique**的判断为假

十一、衍生关系

- **from子句中的子查询**：任何select-from-where表达式返回的结果都是关系，因而可以被插入到另一个select-from-where中任何关系可以出现的位置(from子句里插入select-from-where查询)

```
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
        from instructor
        group by dept_name)
where avg_salary > 42000;
```

- **with子句**：将from子句中的子查询提到前面，在逻辑上更清晰。
  - with子句提供定义临时关系的方法，这个定义只对包含with子句的查询有效

```
with max_budget (value) as
    (select max(budget)
      from department)
select budget
from department, max_budget
where department.budget = max_budget.value;
```

- **标量子查询**：SQL允许子查询出现在返回单个值的表达式能够出现的任何地方，只要该子查询只返回包含单个属性的单个元组，这样的子查询称为标量子查询。

```
select dept_name,
        (select count(*)
           from instructor
           where department.dept_name = instructor.dept_name)
        as num_instructors
from department;
```

上面例子中的子查询保证只返回单个值，因为它使用了不带**group by**的**count**(*)聚集函数

十二、修改数据库

- 删除(delete)、插入(insert)、更新(update)

1.删除(delete)

- 删除instructor关系中的所有元组。Instructor关系本身仍然存在，但它变成空的了

```
delete from instructor
```

- 删除instructor关系中dept_name属性为Finance的所有元组

```
delete from instructor
where dept_name= 'Finance';
```

- 在department表中查找building属性为Waston的元组并把这些元组的dept name做成一个集合a，然后删除instructor关系中dept name在集合a中的所有元组

```
delete from instructor
where dept name in (select dept name
                    from department
                    where building = 'Watson');
```

2.插入(insert) [一次只能插入一个元组]

- 给关系course插入一个元组

```
insert into course
        values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

等价

```
insert into course (course_id, title, dept_name, credits)
        values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- 部分属性可以是null

```
insert into student
        values ('3003', 'Green', 'Finance', null);
```

3.更新(update)

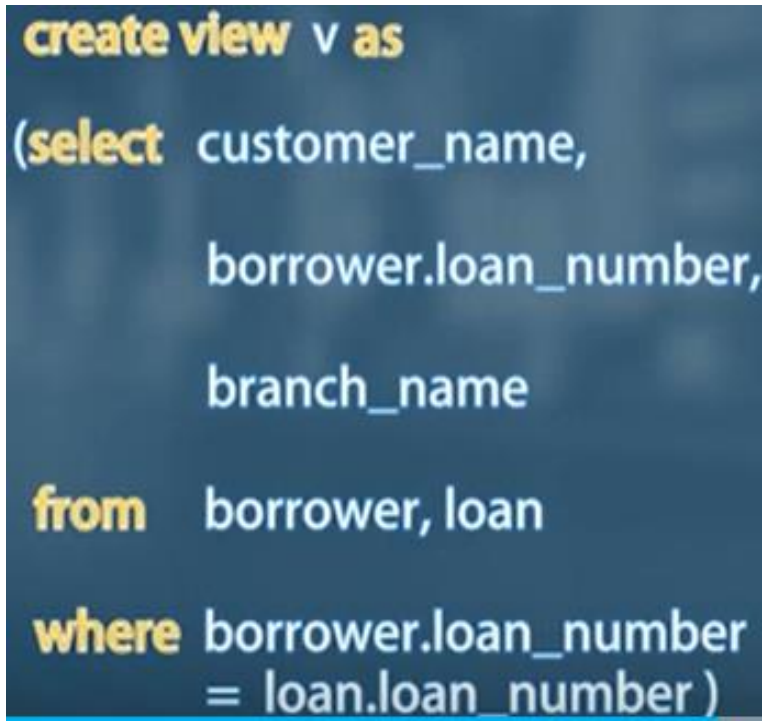
- 在不改变整个元组的情况下改变其部分属性的值
- Update子句后只能跟一个关系名
- 形式
  - 不用case语句时：需要注意update语句的顺序（不合理的顺序可能导致一个元组因为符合条件1进行相应操作之后，结果又符合条件2，这可能不是我们想要的效果）

```
update instructor
    set salary = salary * 1.03
    where salary > 100000;
```

  - 使用case语句时：不需要注意顺序

```
update instructor
    set salary = case
                    when salary <= 100000 then salary * 1.05
                    else salary * 1.03
                 end
```

十三、视图（虚表）

- 将常用的结果存储到数据库当中（实际上存储的是查询语句）
- 
```
create view v as
(select customer_name,
        borrower.loan_number,
        branch_name
 from   borrower, loan
 where  borrower.loan_number
        = loan.loan_number )
```
- 在一个视图的基础上，可以创建新的视图

- 视图的更新操作
  - 前提是该视图是可更新的
  - 可更新条件:
    - 创建的视图基于一个表，也就是这个视图是单表视图
    - 视图要包含主键
    - 视图中不包含聚集函数
- 视图数据修改
  - 插入
    - 在对原始数据构建的时候一次只能插入一行

```
insert into account (branch_name,
balance, account_number)
values ('Perryridge', 1200, 'A-9732')
```