第4章 中级SQL

主要内容

- 连接表达式
- 视图
- 事务
- 完整性约束
- SQL 的数据类型与模式
- SQL 中的索引定义
- 授权

4.1 连接表达式

- 上一章中使用笛卡尔积运算来组合来自多个关系的信息。本节介绍几种"连接"运算，它们允许程序员以一种更自然的方式编写一些查询，并表达某些只用笛卡尔积很难表达的查询。
- 连接运算通常用于 From 子句中。
- 三类连接：
  - 自然连接
  - 内连接
  - 外连接

4.1.1 SQL 中的自然连接

- 自然连接运算：
  - 作用于两个关系，并产生一个关系作为结果。它只考虑在两个关系的模式中都出现的那些属性上取值相同的元组对，并且两个模式中的公共属性只保留其中一个。

4.1.1 SQL 中的自然连接

- 例：查找教师的姓名以及他们所讲授的课程的课程号，之前我们是这样写的：

```
select name, course_id
from  student, takes
where student.ID = takes.ID;
```

- 现在用自然连接，可以这样写：

```
select name, course_id
from student natural join takes;
```

student 关系

ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

takes 关系

ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*null*

student 和 takes 的自然连接

ID	*name*	*dept_name*	*tot_cred*	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	*null*

- 在 from 子句中，可以用自然连接将多个关系结合在一起：

  select $A_1, A_2, \ldots A_n$

  from r_1 natural join r_2 natural join .. natural join r_n

  where P;

- 例：

  select *

  from student natural join takes natural join section;

自然连接中的危险

- 要小心无关的属性因为有相同的名字，而被错误地进行了等值比较
- 例：查找学生的姓名以及他们选修的课程名
  - 正确版本：

```
select name, title
from student natural join takes, course
where takes.course_id = course.course_id;
```

  - 错误版本：

```
select name, title
from student natural join takes natural join course;
```

  - 这个错误版本会忽略掉某个学生选修了别的系所开设的课程这种特殊情况，因为 student 和 course 都有 dept_name 属性

带 using 子句的自然连接

- 为了避免同名属性进行不正确的等值比较的危险，SQL 允许我们用 "using" 指定哪些列需要进行等值比较。
- 例：

```
select name, title
from (student natural join takes)
        join course using (course_id);
```

4.1.2 连接条件

- on 条件允许在参与连接的关系上设置通用的谓词，对应关系代数中的连接运算。
- 该谓词的写法与 where 子句谓词类似，只不过使用的是关键词 on 而不是 where。
- 例：

```
select *
from  student join takes on student.ID  = takes.ID;
```

- on 条件表明：如果一个来自 student 的元组和一个来自 takes 的元组在 ID 上的取值相同，那么它们是匹配的。
- 等价于：

```
select *
from  student , takes
where  student_ID  = takes_ID;
```

4.1.3 外连接

- 外连接是连接运算的一种扩展，可以避免丢失信息。
- 外连接先计算出连接运算的结果，然后加入那些没能在连接运算中匹配上的元组，用空值填充没能匹配上的属性列。
- 有三种形式的外连接：
  - 左外连接
  - 右外连接
  - 全外连接

外连接示例

- 为了方便讲解，我们现在假定 course 和 prereq 之间没有外码约束。
- course 关系为

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- prereq 关系为

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- 我们注意到：课程号为 CS-347 的课程信息在 course 中不存在，同时课程号为 CS-315 的课程在 prereq 中没有先修课信息。

左外连接

```
select *
from course natural left outer join prereq;
```

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

- 关系代数中：course $\mathbb{N}$ prereq

右外连接

```
select *
from course natural right outer join prereq;
```

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

- 关系代数中：course ⟖ prereq

全外连接

- course natural full outer join prereq

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

  - 关系代数中： course ⟗ prereq

连接类型和条件

- 为了把常规连接和外连接区分开来，SQL 中把常规连接称作内连接（inner join）
- 其中 inner 是可选的：

```
select *
from student join takes using(ID);
```

  - 等价于：

```
select *
from student inner join takes using(ID);
```

  - 类似地，natural join 等价于 natural inner join

Join types
inner join
left outer join
right outer join
full outer join

Join conditions
natural
on <predicate>
using $(A_1, A_2, \ldots, A_n)$

- 上图给出了各种连接类型和连接条件
- 任意的连接类型可以和任意的连接条件进行组合
- 例：

```
select * from
course left outer join prereq on course.course_id = prereq.course_id;
```

course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	*null*	*null*

4.2 视图

- 有时并不希望让所有用户看到整个逻辑模型（即，数据库中关系的完整集合）
- 考虑以下情况：一位职员需要知道教师的ID，姓名和所在系名，但是并没有权限看到教师的工资。此人应该看到的关系用 SQL 语句描述如下：

```
select ID, name, dept_name
from instructor;
```

- 视图提供了一种机制，对于特定用户隐藏特定数据。

4.2.1 视图定义

- 使用如下形式的 create view 语句可以定义一个视图：

  create view v as <查询表达式>;

- 其中<查询表达式>是任意合法的 SQL 表达式，视图名称用 v 表示。
- 一旦我们定义了一个视图，就可以用视图名来指代该视图所生成的虚关系。
- 视图定义时系统并不会执行查询表达式来创建一个新的关系，只是保存了这个查询表达式。

视图例子

- 定义一个不包含工资信息的教师视图：

```
create view faculty as
    select ID, name, dept_name
    from instructor;
```

- 通过该视图查询 Biology 系的所有教师信息

```
select name
from faculty
where dept_name = 'Biology';
```

- 定义一个视图描述每个系的所有教师的总年薪值

```
create view departments_total_salary(dept_name, total_salary) as
    select dept_name, sum(salary)
    from instructor
    group by dept_name;
```

- 该例中我们显式指定了视图的属性名。

用其它视图来定义一个视图

- 视图可以用在定义别的视图的查询表达式中：

```
create view physics_fall_2017 as
   select course.course_id, sec_id, building, room_number
   from course, section
   where course.course_id = section.course_id
            and course.dept_name = 'Physics'
            and section.semester = 'Fall'
            and section.year = '2017';
create view physics_fall_2017_watson as
    select course_id, room_number
    from physics_fall_2017
    where building= 'Watson';
```

展开视图

```
create view physics_fall_2017_watson  as
    select course_id, room_number
    from physics_fall_2017
    where building= 'Watson';
```

- 视图展开为：

```
create view physics_fall_2017_watson as
    select course_id, room_number
    from (select course.course_id, building, room_number
            from course, section
            where course.course_id = section.course_id
                and course.dept_name = 'Physics'
                and section.semester = 'Fall'
                and section.year = '2017') as physics_fall_2017
    where building= 'Watson';
```

4.2.3 物化视图

- 某些数据库系统允许物理存储视图表
  - 当定义视图时，就为它创建物理数据备份
  - 这种视图就称为物化视图
- 如果构成视图定义的任何关系被更新时，物化视图中的数据就会过时
  - 需要更新视图表的数据来维护物化视图

4.2.4 视图更新

- 在 faculty 视图中插入一条元组：

  insert into faculty

  values ('30765', 'Green', 'Music');

- 它最终要表示成对于 instructor 关系的插入
  - 必须有 salary 值
- 两种办法：
  - 拒绝插入
  - 向 instructor 关系中插入如下元组：

    ('30765', 'Green', 'Music', null)

有些更新无法表达成对于基本表的更新

```
create view instructor_info as
        select ID, name, building
        from instructor, department
        where instructor.dept_name = department.dept_name;
insert into instructor_info
        values ('69987', 'White', 'Taylor');
```

- 问题是：
  - 如果有多个系在 Taylor 楼，应该插入到哪个系?
  - 如果没有系在 Taylor 楼，怎么办?

可更新视图

- 大多数据库系统只允许在简单视图上进行更新：
  - from 子句中只有一个数据库关系
  - select 子句只包含关系的属性名，不包含任何表达式、聚集或 distinct 声明
  - 没有出现在 select 子句中的任何属性都可以取 null 值
  - 查询中不含有 group by 或 having 子句
- 满足以上条件的视图称为可更新视图

可更新视图仍然可能有问题

- 创建一个历史系教师的视图

```
create view history_instructors as
        select *
        from instructor
        where dept_name= 'History';
```

- 如果插入如下元组到 history_instructors 会怎样?

```
('25566', 'Brown', 'Biology', 100000)
```

- 加上 with check option 再试试看

```
create view history_instructors2 as
        select *
        from instructor
        where dept_name= 'History'
with check option;
```

4.3 事务

- 一个事务由查询和（或）更新语句的序列组成
- SQL 标准规定，当一条 SQL 语句被执行时，就隐式地开始了一个事务。
- 事务必须由以下两个语句中的一个作为结束：
  - Commit work：提交当前事务；事务中 SQL 语句执行的更新在数据库中成为永久性的。
  - Rollback work：回滚当前事务；撤销事务中 SQL 语句执行的所有更新。
- 原子性：
  - 要么事务的所有影响被反映到数据库中，要么任何影响都没有。
- 大多数据库默认每个 SQL 语句执行完就自动提交，如果需要由多个 SQL 语句构成一个事务，就必须关闭"自动提交"。

4.4 完整性约束

- 完整性约束保证授权用户对数据库所做的修改不会破坏数据一致性。例如：
  - 教师姓名不能为 null
  - 任意两位教师不额能有相同的 ID
  - 每个系的预算必须大于 0 美元

4.4.1 单个关系上的约束

- primary key
- not null
- unique
- check (<谓词>)

4.4.2 非空约束

- not null
  - 例如，声明 name 和 budget 不能为空值

    name varchar(20) not null

    budget numeric(12,2) not null

4.4.3 唯一性约束

- unique (A_1, A_2, ..., A_m)
  - 唯一性声明指出属性 A_1, A_2, ..., A_m 形成了一个超码。
  - 但是允许为 null，除非它们已被显式地声明为非空。

4.4.4 check 子句

- check (P) 子句指定一个谓词 P， 关系中的每一个元组都必须满足谓词 P
- 例：确保 semester 必须是 'Fall', 'Winter', 'Spring', 'Summer' 中的一个。

```
create table section
    (course_id varchar (8),
     sec_id varchar (8),
     semester varchar (6),
     year numeric (4,0),
     building varchar (15),
     room_number varchar (7),
     time_slot_id varchar (4),
     primary key (course_id, sec_id, semester, year),
     check (semester in ('Fall', 'Winter', 'Spring', 'Summer')));
```

4.4.5 引用完整性

- 我们常常希望保证一个关系（引用关系）中给定属性集合的取值也在另一个关系（被引用关系）的给定属性集的取值中出现，这称为引用完整性约束。
  - 例如：如果 "Biology" 出现在 instructor 关系的某个元组的 dept_name 属性列中，那在 department 关系中也必须有一个元组的 dept_name 属性列上为 "Biology"
- 外码是引用完整性约束的一种形式，其中被引用的属性构成被引用关系的主码。

外码

- 通过使用外码子句，可以将外码指定为 SQL 建表语句的一部分

  foreign key (dept_name) references department

- 缺省情况下，在 SQL 中外码引用的是被引用表的主码属性
- SQL 还支持引用子句中显式指定被引用关系的属性列表

  foreign key (dept_name) references department (**dept_name**)

引用完整性约束的级联操作

- 当一个引用完整性约束被违背的时候，常规的操作是拒绝执行导致破坏完整性的操作。
- 此外，还可以进行级联删除或修改操作

```
create table course (
 (...
  foreign key (dept_name) references department
      on delete cascade
      on update cascade,
 ...);
```

- 除了级联操作，还可以：
  - set null
  - set default

4.4.6 给约束赋名

- 我们可以用 constraint 子句为完整性约束赋予名称。
- 如果我们想要删除之前定义的某个约束，这样的名称是很有用的。
- 例如：

```
create table instructor
(ID  varchar(5),
 name  varchar(20) not null,
 dept_name  varchar(20),
 salary  numeric(8,2) constraint minsalary check (salary > 29000),
 primary key (ID),
 foreign key (dept_name) references department (dept_name)
          on delete set null);
```

复杂 check 条件

- check 子句中的谓词可以是包含子查询的任意谓词：

  check (time_slot_id in (select time_slot_id from time_slot))

- 这个 check 条件声明：section 关系的每个元组中的 time_slot_id 都是 time_slot 关系中某个时间段的标识
  - 这个条件不仅是在 section 关系中插入或修改元组的时候需要检测，在 time_slot 关系改变时也需要检测。

4.4.8 断言

- 断言就是一个谓词，它表达了我们希望数据库总能满足的某个条件。
- 以下约束可以使用断言来表示：
  - 对于 student 关系的每个元组，它在 tot_cred 属性上的取值必须等于该生已成功修完的课程的学分总和；
  - 每位教师不能在同一个学期的同一个时间段在两个不同的教室授课。
- SQL 断言的形式如下：

  create assertion <断言名> check (<谓词>);

- 检测和维护断言的开销较高

4.5 SQL 的日期和时间类型

- date: 日历日期，包括年(4位)，月，日
  - 例如：date '2018-04-25'
- time: 一天中的时间，用时、分、秒表示
  - 例如：time '09:30:00' 或 time '09:30:01.45'
- timestamp: 时间戳，date 和 time 的结合
  - 例如：timestamp '2018-04-25 09:30:01.45'

大对象类型

- 大型数据项(照片、视频、CAD 文件等) 可以用大对象类型表达
  - blob: 二进制大对象 --对象是一个大的二进制数据集
  - clob: 字符大对象 -- 对象是一个大的字符数据集
- 当一个查询要返回一个大对象的时候，一般是返回一个指向大对象的指针，而不是返回这个大对象本身

用户自定义类型

- SQL 支持用户自定义数据类型

  create type Dollars as numeric (12,2) final;

- 使用自定义数据类型：

```
create table department
        (dept_name varchar (20),
         building varchar (15),
         budget Dollars);
```

域

- 在 SQL-92 标准中引入了域的概念
- 创建用户自定义的域：

```
create domain person_name char(20) not null;
```

- 类型和域是相似的，但是域上可以指定约束，例如 not null
- 例如：

```
create domain degree_level varchar(10)
    constraint degree_level_test
    check (value in ('Bachelors', 'Masters', 'Doctorate'));
```

4.6 SQL 中的索引定义

- 许多查询仅涉及文件中的一小部分记录
- 数据库系统遍历所有的记录，以找到有某个值的记录，这种做法是低效的
- 关系属性上的索引是一种数据结构，它允许数据库系统高效地找到在关系中具有该属性指定值的那些元组，而不扫描关系的所有元组
- 我们使用 create index 命令来创建索引：

  create index <索引名> on <关系名> (属性列表);

创建索引的例子

```
create table instructor
    (ID              varchar(5),
     name            varchar(20) not null,
     dept_name       varchar(20),
     salary          numeric(8,2),
     primary key (ID),
     foreign key (dept_name) references department (dept_name)
    );
create index dept_index on instructor(dept_name);
```

- 查询:

```
select * from  instructor where dept_name = 'Music';
```

可以自动用索引来找到想要的元组，而不用查找 instructor 关系中的所有记录。

4.7 授权

- 我们可能会给一个用户在数据库的某些部分上授予几种形式的权限：
  - 授权读取数据
  - 授权插入新数据
  - 授权更新数据
  - 授权删除数据
- 每种这样类型的授权都称为一种权限
- 我们可以在数据库的某些特定部分（比如一个关系或视图）上授予用户所有这些类型的权限，或完全不授权，或授予这些权限的一个组合

权限的授予

- grant 语句用来授予权限

  grant <权限列表>

  on <关系名或视图名>

  to <用户/角色列表>;

- <用户/角色列表> 可以是:
  - 用户名
  - public：系统所有当前用户和将来的用户
  - 角色
- 例:

  grant select on department to Amit, Satoshi;

- 视图上的授权不意味着同时也授予底层基本表上的权限
- 权限的授予者必须自己已经拥有该项目上的权限，或者是DBA

SQL 中的权限

- select: 允许读关系表，或使用视图进行查询
- insert: 插入元组的权限
- update: 使用SQL update 语句更新的权限
- delete: 删除元组的权限

权限的收回

- revoke 语句用来收回授权

  revoke <权限列表>

  on <关系名或视图名>

  from <用户/角色列表>;

- 例:

  revoke select on department from Amit, Satoshi;

角色

- 创建角色：

  create role <角色名>;

- 然后角色就可以象用户那样被授予权限
- 也可以为用户授予某个角色：
  - grant <角色> to <用户>;

角色例子

- create role instructors;
- grant instructors to Amit;
- 为角色授予权限:

  grant select on takes to instructors;
- 将角色授予给用户，以及其它角色：

  create role teaching_assistant;

  grant teaching_assistant to instructors;
  - Instructors 继承了teaching_assistant的所有权限
- 角色链：

  create role dean;

  grant instructors to dean;

  grant dean to Satoshi;

视图的授权

```
create view  geo_instructor as
        (select *
        from instructor
        where dept_name = 'Geology');
grant select on geo_instructor to geo_staff;
```

- 假定工作人员 geo_staff 发出如下查询：

```
select *
from geo_instructor;
```

- 那么：
  - geo_staff 没有 instructor 上的权限会如何?
  - 视图的创建者没有 instructor 上的权限会如何?

其它授权特性

- 引用权限：创建外码

  grant reference(dept_name) on department to Mariano;

  - 为什么需要这个权限?

- 权限的转移

  grant select on department to Amit with grant option;

  revoke select on department from Amit, Satoshi cascade;

  revoke select on department from Amit, Satoshi restrict;