数据库系统原理

陈岭

浙江大学计算机学院

6 SQL语言（4）

- 事务
- 完整性约束
- 触发器
- 数据安全性
- 审计跟踪

事务

- 事务（transaction）由查询和更新语句的序列组成。SQL标准规定当一条SQL语句被执行，就隐式地开始了一个事务。下列SQL语句之一会结束一个事务:
  - Commit work:提交当前事务，也就是将该事务所做的更新在数据库中持久保存。在事务被提交后，一个新的事务自动开始
  - Rollback work：回滚当前事务，即撤销该事务中所有SQL语句对数据库的更新。这样，数据库就恢复到执行该事务第一条语句之前的状态

事务

- 动机示例：

```
    update account set balance = balance - 100
    where account_number = 'A-101' ;
    update account set balance = balance + 100
    where account_number = 'A-201' ;

COMMIT WORK;
```

  - 如果其中一个更新成功，另一个更新失败，会数据库中导致数据不一致问题
  - 因此，这两个更新要么全部成功，要么全部失败

- 原子性（atomic）
  - 一个事务或者在完成所有步骤后提交其行为，或者在不能成功完成其所有动作的情况下回滚其多有动作

事务

- 事务的四个性质：
  - 原子性（atomic）
  - 一致性(consistency)
  - 隔离性(isolation)
  - 持久性(durability)
- 在很多SQL实现中，默认方式下每个SQL语句自成一个事务，且一执行完就提交
  - 如果一个事务要执行多条SQL语句，就必须关闭单独SQL语句的自动提交，如何关闭自动提交也依赖于特定的SQL实现
- 一个较好的选择是，作为SQL:1999标准的一部分，允许多条SQL语句包含在关键字begin atomic … end之间

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的一致性
- 完整性约束的例子有：
  - 教师姓名不能为null
  - 任意两位教师不能有相同的教师标识
  - course关系中的每个系名必须在department关系中有一个对应的系名
  - 一个系的预算必须大于0.00美元
- 域完整性、实体完整性（主键的约束）、参照完整性（外键的约束）和用户定义的完整性约束
- 完整性约束是数据库实例(Instance)必须遵循的
- 完整性约束由DBMS维护

单个关系上的约束

- 单个关系上的约束
  - not null
  - unique
  - check（<谓词>）
  - 例，

```
CREATE TABLE instructor2
        ( ID char(5)  primary key ,
          name varchar(20) not null,
          dept_name varchar(20),
          salary  numeric(8,2) not null,
          check (salary>= 0));
```

域约束

- 域约束是完整性约束的最基本形式，可用于检测插入到数据库中的数据的合法性
- 从现有数据类型可以创建新的域

```
create domain Dollars as numeric(12, 2) not null
create domain Pounds as numeric(12,2);

create table instructor
     ( ID char(5) primary key,
       name varchar(20),
       dept_name varchar(20),
       salary Dollars,
       comm Pounds
      );
```

域约束

- check子句也可以应用到域上
  - 例，check子句可以保证教师工资域中只允许出现大于给定值的值

```
create domain YearlySalary numeric(8,2)
constraint salary_value_test check(value >= 29000.00);
```

  - *YearlySalary* 域有一个约束来保证年薪大于或等于29 000.00美元
  - constraint *salary_value_test* 子句是可选的，它用来将该约束命名为*salary_value_test*。系统用这个名字来指出一个更新违反了哪个约束
  - 作为另一个例子，使用in子句可以限定一个域只包含指定的一组值

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

参照完整性

- 参照完整性约束定义：
  - 令关系r_1和r_2的属性集分别为R_1和R_2，主码分别为K_1和K_2
  - 如果要求对r_2中任意元组t_2，均存在r_1中元组t_1使得$t_1[K_1] = t_2[\alpha]$，我们称R_2的子集α为参照关系r_1中K_1的外码（foreign key）
  - 参照完整性约束也称为子集依赖，可写作：

$$\Pi_{\alpha}(r_2) \subseteq \Pi_{K1}(r_1)$$

参照完整性

- 回顾第二讲中外键的内容
- 假设存在关系r和s：r(A, B, C)，s(B, D)，则在关系r上的属性B称作参照s的**外码**，r也称为外码依赖的参照关系，s叫做外码被参照关系
  - 例，学生(学号, 姓名, 性别, 专业号, 年龄) – *参照关系*

    专业(专业号, 专业名称) – *被参照关系（目标关系）*

    其中属性*专业号* 称为关系*学生*的*外码*

    选修(学号，课程号，成绩)
    课程(课程号, 课程名，学分，先修课号)
  - *instructor*(*ID*, *name*, *dept_name*, *salary*) – 参照关系

    department(*dept_name*, *building*, *budget*) – 被参照关系

参照关系中外码的值必须在被参照关系中实际存在或为null

数据库的修改

- 数据库的修改会导致参照完整性的破坏。这里列出对各种类型的数据库修改应做的测试，以保持如下的参照完整性约束：

$$\Pi_{\alpha}(r_2) \subseteq \Pi_{K1}(r_1)$$

  - 插入。如果向r_2中插入元组t_2，则系统必须保证r_1中存在元组t_1使得$t_1[K] = t_2[\alpha]$。即

$$t_2[\alpha] \in \Pi_K(r_1)$$

  - 删除。如果从r_1中删除元组t_1，则系统必须计算r_2中参照t_1的元组集合。即

$$\sigma_{\alpha = t1[K]}(r_2)$$

    - 如果集合非空，要么删除命令报错并撤销，要么参照t_1的元组本身必须被删除（可能导致级联删除）

数据库的修改

- 更新。必须考虑两种更新：对参照关系r_2做更新，以及对被参照关系r_1做更新
  - 如果关系r_2中元组t_2被更新，并且更新修改外码α上的值，则进行类似插入情况的测试。令t_2'表示元组t_2的新值，则系统必须保证

$$t_2'[\alpha] \in \Pi_K(r_1)$$

  - 如果关系r_1中元组t_1被更新，并且该更新修改主码K上的值，则进行类似删除情况的测试。系统必须用旧的t_1值（更新前的值）计算

$$\sigma_{\alpha = t1[K]}(r_2)$$

  如果该集合非空，则更新失败，或者以类似删除的方式做级联更新

数据库的修改

- 更新。必须考虑两种更新：对参照关系r_2做更新，以及对被参照关系r_1做更新
  - 如果关系r_2中元组t_2被更新，并且更新修改外码α上的值，则进行类似插入情况的测试。令t_2'表示元组t_2的新值，则系统必须保证

$$t_2'[\alpha] \in \Pi_K(r_1)$$

  - 如果关系r_1中元组t_1被更新，并且该更新修改主码K上的值，则进行类似删除情况的测试。系统必须用旧的t_1值（更新前的值）计算

$$\sigma_{\alpha = t1[K]}(r_2)$$

  如果该集合非空，则更新失败，或者以类似删除的方式做级联更新

SQL中的参照完整性

- 主码、候选码和外码可在SQL的create table语句中指明
  - primary key子句包含一组构成主码的属性
  - unique子句包含一组构成候选码的属性
  - foreign key子句包含一组构成外码的属性以及被修改外码所参照的关系名

SQL中的参照完整性

- 默认地，外码参照被参照关系中的主码

  foreign key (*dept_name*) references *department*

- 可以使用如下的简写形式定义单个列为外码

  dept_name *varchar* (20) references *department*

- 被参照关系中的属性可以被明确指定，但是必须被声明为主码或候选码

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

SQL中的参照完整性

- 例，

```
create table classroom
        (building varchar (15),
         room_number varchar (7),
         capacity numeric (4,0),
         primary key (building, room_number))
create table department
        (dept_name varchar (20),
         building varchar (15),
         budget numeric (12,2) check (budget > 0),
         primary key (dept_name))
```

SQL中的参照完整性

```
create table course
        (course_id varchar (8),
         title varchar (50),
         dept_name varchar (20),
         credits numeric (2,0) check (credits > 0),
         primary key (course_id ),
         foreign key (dept_name) references department)
create table instructor
        (ID varchar (5),
         name varchar (20), not null
         dept_name varchar (20),
         salary numeric (8,2), check (salary > 29000),
         primary key (ID),
         foreign key (dept_name) references department)
```

SQL中的级联动作

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

- 由于有了与外码声明相关联的on delete cascade子句，如果删除*department*中的元组导致了此参照完整性约束被违反，则删除并不被系统拒绝，而是对course关系作“级联”删除，即删除了被删除系的元组
- “级联”更新也类似

SQL中的级联动作

- 如果存在涉及多个关系的外码依赖链，则在链一端所做的删除或更新可能传至整个链
- 但是，如果一个级联更新或删除导致的对约束的违反不能通过进一步的级联操作解决，则系统终止该事务
  - 即，该事务所做的所有改变及级联动作将被撤销

SQL中的级联动作

- 参照完整性只在事务结束时检查
  - 中间步骤可以破坏参照完整性，只要后续步骤解消这种破坏即可
  - 否则不可能建立某些数据库状态，例如插入两条互相有外键引用的元组
    - 例，关系*marriedperson* 的*spouse* 属性

      marriedperson (*name*, *address*, *spouse*)

SQL中的级联动作

- 除级联操作之外的其他选择:
  - on delete set null
  - on delete set default
- 外键属性上的空值使SQL的参照完整性语义变得复杂，最好用not null来防止
  - 若某外键属性为null，则该元组按定义是满足参照完整性约束的

断言

- 断言（assertion）是表达要求数据库永远满足的条件的谓词（复杂check条件）
- SQL 中的断言形式如下：

  create assertion <assertion-name> check <predicate>
- 创建了某断言之后，系统将检查它的合法性，并对每一个可能破坏该断言的数据库更新进行检测
  - 这种检测会产生大量的开销， 因此断言的使用应非常谨慎
- 由于SQL不提供“for all X, P(X) ”结构，我们可以通过迂回的方式表达：not exists X such that not P(X)

  ∵ $(\forall x)P(x) = \neg (\exists x) \neg P(x)$

断言

- 例1，对于*student*关系中的每个元组，它在属性*tot_cred*上的取值必须等于其所成功修完课程的学分总和

```
create assertion credits_earned_constraint check
        (not exists (select ID
          from student
          where tot_cred <> (select sum(credits)
                             from takes natural join course
                             where student.ID= takes.ID
                             and grade is not null and
                             grade <> ' F' );
```

断言

- 例2，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

```
create assertion ins_teaches_constraint check not exists
        (select ID, name, section_id, semester, year,time_slot_id,
                count(distinct building, room_number)
         from instructor natural join teaches natural join section
         group by (ID, name, section_id, semester, year, time_slot_id)
         having count(building, room_number) > 1)
```

触发器

- 触发器（trigger）是由数据库更新操作引起的被系统自动执行的语句
- 设计触发器必须:
  - 指明触发器被执行的条件
  - 指明触发器执行时所做的具体操作
- 引入触发器的SQL标准是SQL:1999，但多数数据库产品早已支持非标准语法的触发器

触发器

- 例，使用触发器来确保关系*section*中属性*time_slot_id*的参照完整性

```
create trigger timeslot_check1 after insert on section
        referencing new row as nrow
        for each row
        when (nrow.time_slot_id  not in
                  (select time_slot_id
                   from time_slot)) /* time_slot中不存在该
                                       time_slot_id */
        begin
             rollback
        end;
```

触发器

```
create trigger timeslot_check2 after delete on time_slot
        referencing old row as orow
        for each row
        when (orow.time_slot_id  not in
                    (select time_slot_id
                     from time_slot) /* 在time_slot 中刚刚被删除的time_
                                        slot_id */
        and orow.time_slot_id in
                    (select time_slot_id
                     from section)) /* 在section中仍含有该time_
                                       slot_id 的引用*/
        begin
           rollback
        end;
```

触发器

- 触发事件包括insert, delete和update
- 针对update的触发器可以指定具体修改的属性

  create trigger *takes_trigger* after update of *takes* on *grade*
- 更新前后的属性值可通过下列方法被引用
  - referencing old row as orow：对删除和修改有效
  - referencing new row as nrow：对插入和修改有效

语句级触发器

- 除了可以针对受影响的每一行执行一次单独的操作，也可以针对受到一个事务影响的所有行只执行一次操作
  - for each statement vs. for each row
  - 用referencing old table 或 referencing new table 来引用包含受影响的行的临时表
  - 对更新大量元组的SQL语句更高效

外部动作

- 有时要求数据库更新能触发外部动作
  - 例如当某种物品库存量小到一定程度就发订货单，或者打开报警灯
- 触发器不能直接实现外部动作，但是
  - 触发器可以在某个表中记录将采取的行动，而让另一个外部进程不断扫描该表并执行相应的外部动作
- 例，假设仓库库存有如下关系
  - *inventory*(*item*, *level*)：仓库中每种物品的库存量
  - *minlevel*(*item*, *level*)：每中物品的最小库存量
  - *reorder*(*item*, *amount*)：当物品小于库存量的时候要订购的数量
  - *orders*(*item*, *amount*) ：所下定单(由外部进程读取)

外部动作

```
create trigger reorder_trigger after update of level on inventory
      referencing old row as orow, new row as nrow
      for each row
      when nrow.level <=
             (select level
              from minlevel
              where minlevel.item = nrow.item) and orow.level >
                    (select level
                     from minlevel
                     where minlevel.item = orow.item)
      begin
         insert into orders
                  (select item, amount
                   from reorder
                   where reorder.item = orow.item)
      end
```

SQL Server触发器语法

```
create trigger timeslot_check1 on section
        after insert as
        if
        (inserted.time_slot_id  not in
                  (select time_slot_id
                   from time_slot)) /* time_slot中不存在该
                                       time_slot_id */
        begin
             rollback
        end;
```

inserted、deleted相当于前面的nrow和orow（成为过渡表）

Oracle触发器语法

```
create or replace trigger secure_student before insert or delete or
                                                    update on student
    begin
        if(to_char(sysdate, 'DY') in ('星期六', '星期日'))
        or(to_char(sysdate, 'HH24') not between 8 and 17 )
        then raise_application_error(-20506, '你只能在上班时间修改数据');
        end if;
    end;
```

- 注：运行该程序，实际是对其进行编译，若出错，可查看数据字典中user_errors的出错信息。user_triggers登记已建立了哪些触发器及定义内容
- 删除触发器：drop trigger <触发器名>

何时不用触发器

- 有时要求数据库更新能触发外部动作
  - 例如当某种物品库存量小到一定程度就发订货单，或者打开报警灯
- 早期触发器被用于如下任务
  - 维护综合数据（如，每门课的选课人数）
  - 复制数据库：记录特定关系(称为change或delta关系)的变化并由一单独进程将此变化反映到所有副本
- 上述任务现在有更好的做法：
  - 现在的数据库提供内建的物化视图来维护综合数据
  - 现代的数据库系统提供内置的数据库复制工具

讨论

- 比较一下三者的区别：
  - check
  - assertion
  - trigger

安全性

- 安全性：防止恶意更新或偷窃数据的企图
- 数据库系统级
  - 验证和授权机制使得特定用户存取特定数据
  - 本章中主要讨论授权机制
- 操作系统级
  - 操作系统超级用户可对数据库做任何事情！ 需要有一个好的操作系统级安全机制
- 网络级：使用加密防止
  - 偷听(未授权的读取信息)
  - 伪装(冒充授权用户)

安全性

- 物理级
  - 对计算机的物理访问使得入侵者可摧毁数据，需要传统的锁钥安全手段
  - 防止洪水，火灾等对计算机的损坏
- 人员级
  - 审查用户以确保授权用户不会将存取权给予入侵者
  - 训练用户选择口令与保密

授权

- 对数据的授权包括：
  - 读权限 - 允许读，但不允许更新数据
  - 插入权限 - 允许插入新数据，但不允许更新现有数据
  - 修改权限 - 允许修改，但不允许删除数据
  - 删除权限 - 允许删除数据

授权

- 对修改数据库模式的授权包括：
  - 索引权限 – 允许创建和删除索引
  - 资源权限 – 允许创建新关系
  - 修改权限 – 允许增加或删除关系的属性
  - 删除权限 – 允许删除关系

授权与视图

- 用户可被授予关于视图的权限，而不被授予关于该视图定义中涉及的关系的权限
- 视图隐藏数据的能力既能简化系统的使用又能增强安全性(只允许用户存取他们工作中需要的数据)
- 关系级安全性与视图级安全性的结合使用可精确地将用户存取限制在他所需要的数据上

视图例

- 假设，一个工作人员只需要知道一个给定系（比如Geology系）里所有员工的工资，但无权看到其他系中员工的相关信息
  - 方法：不允许对*instructor*关系的直接访问，但授予对视图*geo_instructor*的访问权限，该视图仅由属于Geology系的那些*instructor*元组构成
  - 视图*geo_instructor*用SQL定义如下：

```
create view geo_instructor as
(select *
 from instructor
 where dept_name = ’ Geology’ );
```

视图例

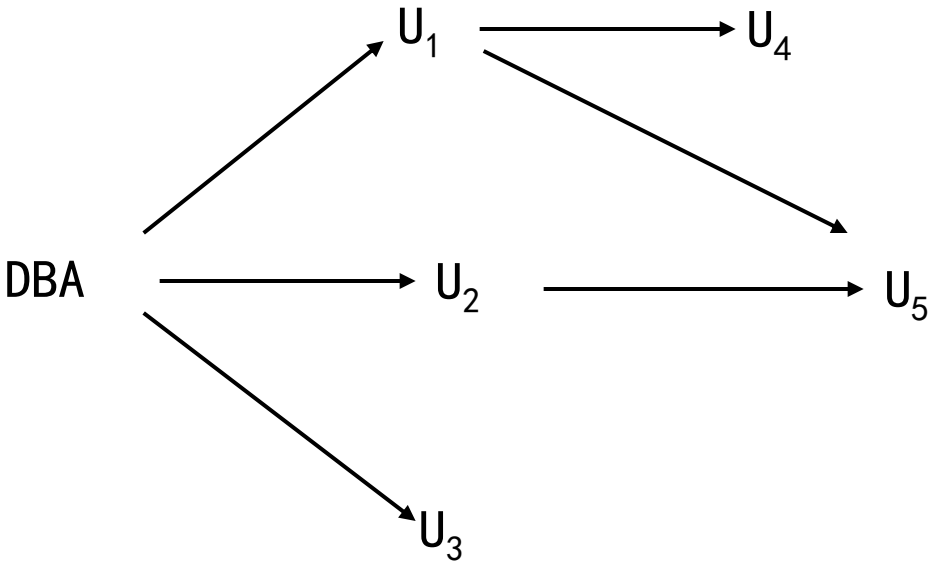
- 该工作人员有权看到如下查询的结果

```
select *
from geo_instructor;
```

- 当查询处理器将该查询转换为对数据库中实际关系的查询时，它产生了一个在*instructor*上的查询
- 必须在查询处理开始之前检查该工作人员的权限

权限的授予

- 权限从一个用户到另一个用户的传递可用授权图表示
- 图的节点是用户
- 图的根是数据库管理员
- 边$U_i \rightarrow U_j$表示用户U_i将某权限授予给了用户U_j



授权图

- 要求：授权图中的所有边都必须是某条从数据库管理员出发的路径的一部分
- 若DBA从U_1收回权限：
  - 必须从U_4收回权限，因为U_1不再有权限
  - 不能从U_5收回权限，因为U_5还有从DBA经U_2 的另一条授权路径
- 必须防止不经过根节点的循环授权：
  - DBA授权给U_7
  - U_7授权给U_8
  - U_8授权给U_7
  - DBA从U_7收回权限
- 必须收回从U_7到U_8以及从U_8到U_7的授权，因为不再有从DBA到U_7或U_8的路径

SQL中的安全性声明

- grant语句用于授权

  GRANT <*privilege list*>

  ON <*relation name or view name*> TO <*user list*>;

- < *user list* > 可以是：
  - 用户ID
  - public，代表所有合法用户
  - 角色
- 授予对视图的权限并不意味着授予对定义该视图的基础关系的权限
- 权限的授予者本身必须拥有相应的权限（或者是数据库管理员）

SQL中的权限

- select：允许读关系，或查询视图
  - 例如：授予用户U_1，U_2，U_3 对*instructor*关系的select权限：

    grant select on *instructor* to U_1 , U_2 , U_3
- insert ： 允许插入元组
- update ： 允许修改元组
- delete ： 允许删除元组
- references ： 创建关系时允许声明外键
- all privileges ： 所有权限

授权的权限

- with grant option：允许用户把被授予的权限再转授给其他用户
  - 例如：授予U_1对*instructor*的select权限并允许 U_1将此权限授予其他用户

    grant select on *instructor* to U_1 with grant option

角色

- 通过创建角色可以一次性对一类用户指定其共同的权限
- 像对用户一样，可以对角色授予或收回权限
- 角色可被赋予给用户，甚至给其他角色
- SQL:1999 支持角色

```
create role instructor;
grant select on takes to instructor;
grant dean to Amit;
create role dean;
grant instructor to dean;
grant dean to Satoshi;
```

SQL中的权限回收

- revoke语句用于回收权限

  REVOKE<*privilege list*> ON <*relation name or view name*>
  FROM <*user list*> [restrict | cascade]

- 例如:

  revoke select on *instructor* from U_1, U_2, U_3 cascade

- 从一用户收回权限可能导致其他用户也失去该权限，称为级联回收
- 指定restrict可以阻止级联回收

  revoke select on *instructor* from U_1, U_2, U_3 restrict

  - 如果要求级联回收，则带有restrict的revoke命令将会失败

SQL中的权限回收

- <privilege list> 可以是all ，以便收回某用户拥有的所有权限
- 如果同一权限被不同授予者两次授予同一用户，则该用户在回收一次后仍保持该权限
- 所有依赖于被收回权限的权限也被收回

SQL授权的局限性

- SQL不支持元组级的授权
  - 例如我们不能限制学生只能看他自己的分数
- 某些应用(如web应用)的所有最终用户可能被映射成单个数据库用户
- 以上情况下的授权任务只能依靠应用程序
  - 细粒度授权，如授权给个别元组，可以由应用程序来实现（优点）
  - 授权在应用程序代码中完成，并可能散布在整个应用中（缺点）
  - 检查是否有权限漏洞非常困难，因为需要读大量应用程序代码（缺点）

审计跟踪

- 审计跟踪（audit trail）是关于应用程序数据的所有更改（插入/删除/更新）的日志，以及一些信息，如哪个用户执行了更改和什么时候执行的更改
- 用于跟踪安全漏洞或错误更新
- 可以使用触发器实现审计跟踪，但是很多数据库提供了内置的机制创建审计跟踪

oracle中的审计

- 语句审计：

```
audit table by scott by access whenever successful;
```

  - 审计用户scott每次成功地执行有关table的语句(create table, drop table, alter table)
  - 格式：

```
AUDIT <st-opt> [ BY <users> ]
[ BY SESSION | ACCESS ]
[ WHENEVER SUCCESSFUL| WHENEVER NOT SUCCESSFUL ]
```

    - 当BY <users> 缺省，对所有用户审计
    - BY SESSION每次会话期间，相同类型的需审计的SQL语句仅记录一次
    - 常用的<st-opt>：table，view，role，index，…
    - 取消审计：NOAUDIT …(其余同audit语句)

oracle中的审计

- 对象（实体）审计：

```
audit delete, update on student;
```

  - 审计所有用户对student表的delete和update操作
  - 格式：

```
AUDIT <obj-opt> ON <obj>|DEFAULT
[ BY SESSION | BY ACCESS ]
[ WHENEVER SUCCESSFUL| WHENEVER NOT SUCCESSFUL ]
```

    - obj-opt: insert, delete, update, select, grant, …
    - 实体审计对所有的用户起作用
    - ON <obj> 指出审计对象表、视图名
    - ON DEFAULT 对其后创建的所有对象起作用
    - 取消审计：NOAUDIT …

oracle中的审计

- 怎样看审计结果：
  - 审计结果记录在数据字典表：sys.aud$中，也可从dba_audit_trail，dba_audit_statement，dba_audit_object中获得有关情况
  - 上述数据字典表需在DBA用户（system）下才可见

总结

- 事务是一个查询和更新的序列，他们共同执行某项任务。事务可以被提交或回滚。事务具有原子性、一致性、隔离性、持久性
- 完整性约束保证授权用户对数据库所做的改变不会导致数据一致性的破坏
  - 域完整性
  - 实体完整性（主键的约束）
  - 参照完整性（外键的约束）
  - 用户定义的完整性约束
- 断言是描述性表达式，它指定了我们要求总是为真的谓词
- 触发器定义了当某个事件发生而且满足相应条件时自动执行的动作

总结

- SQL的授权机制
- 角色有助于根据用户在组织机构中所扮演的角色，把一组权限分配给用户
- 审计跟踪用于跟踪安全漏洞或错误更新，oracle中的审计跟踪包括：
  - 语句审计
  - 对象（实体）审计

谢谢！