基础篇
第三章
数据库的安全性与完整性

主讲：王全宝

- 为了保证数据库数据的安全可靠和正确有效，DBMS必须提供统一的数据保护功能
  - 数据库的安全性：
    - 是指数据库的任何部分都不允许受到恶意侵害或未经授权的存取和修改。
  - 数据库的完整性：
    - 一般指语义完整性和事务完整性两个方面。
    - 本章主要讨论数据库的语义完整性。

用户操作权限和完整性约束都存储在数据库管理系统的数据字典中

数据库的安全性和完整性都由数据库管理系统来确保

Outline

- 数据库安全性
- 数据库完整性

数据库安全性

- **Why?**
  - 数据库的一大特点是数据可以共享
  - 但数据共享必然带来数据库的安全性问题
  - 数据库系统中的数据共享不能是无条件的共享

例：军事秘密、国家机密、新产品实验数据、
市场需求分析、市场营销策略、销售计划、
客户档案、医疗档案、银行储蓄数据

- 1991年4月美国NCSC（国家计算机安全中心）颁布了《可信计算机系统评估标准关于可信数据库系统的解释》（Trusted Database Interpretation 简称TDI）
  - TDI中定义了数据库管理系统的设计与实现中需满足和用以进行安全性级别评估的标准。

- **TCSEC/TDI安全级别划分**

安全级别	定　　义
A1	验证设计（Verified Design）
B3	安全域（Security Domains）
B2	结构化保护（Structural Protection）
B1	标记安全保护（Labeled Security Protection）
C2	受控的存取保护（Controlled Access Protection）
C1	自主安全保护（Discretionary Security Protection）
D	最小保护（Minimal Protection）

按系统可靠或可信程度逐渐增高

安全级别具有向下兼容性

- 数据库安全性控制的常用方法：
  - 用户标识和鉴定
  - 存取控制
  - 视图
  - 审计
  - 加密存储

- 用户标识与鉴别(Identification & Authentication)
  - 系统提供的最外层安全保护措施
  - 用户身份鉴别的方法
    - 静态口令
    - 动态口令
    - 生物特征鉴别
    - 智能卡鉴别

账号	口令
张明	1234
李丽	2345
……	……

- 存取控制机制
  - 存取控制机制的组成
    - 定义存取权限
      - DBMS提供语言机制来定义用户权限，权限定义经过编译后存储在数据字典中
    - 检查存取权限
      - 用户向数据库发出存取请求（包括操作类型、操作对象、操作用户等信息）
      - DBMS查找数据字典，检查权限，用户的操作请求如果超出其权限，拒绝执行

    用户权限定义和合法权检查机制一起组成了DBMS的安全子系统

- 存取控制机制
  - 常用存取控制方法
    - 自主存取控制Discretionary Access Control (DAC)：C2级
      - 同一用户对于不同的数据对象有不同的存取权限
      - 不同的用户对同一对象也有不同的权限
      - 用户还可将其拥有的存取权限转授给其他用户
    - SQL对自主存取控制提供的支持
      - 用Grant语句向用户授予对数据的操作权限
      - 用Revoke语句收回授予的权限

定义模式

定义一个命名空间，其中可以进一步定义模式包含的数据库对象，例如基本表、视图、索引等.

CREATE SCHEMA “S-T” AUTHORIZATION Bob;

CREATE SCHEMA AUTHORIZATION Bob;

删除模式

DROP SCHEMA “S-T” CASCADE;

DROP SCHEMA “S-T” RESTRICT;

- 用户权限的两个要素
  - 数据库对象
  - 操作类型

对象类型	对象	操作类型
数据库模式	模式	create schema
	基本表	create table, alter table
	视图	create view
	索引	create index
数据	基本表和视图	select, insert, update, delete, references, all privileges
	属性列	select, insert, update, references, all privileges

- 授权
  - 语法格式

    GRANT <权限> [, 权限] …

    ON <对象类型> <对象名> [, <对象类型> <对象名>] …

    TO <用户> [, 用户] …

    [WITH GRANT OPTION];

    指定授予的权限

    数据库对象

    可以授权给一个或多个用户, PUBLIC表示全体用户

    获得权限的用户可以把权限授予其他用户，不允许循环授权(a to b, b to c, c to a)
  - 功能
    - 向用户授予操纵数据库对象的权限
  - 例如，

    把关系R上的SELECT权限赋予用户user

    GRANT select ON Table R TO user;

- 授权示例
  - 把查询Course表的权限授给用户Bob

    GRANT SELECT

    ON TABLE Course

    TO Bob;
  - 把Student表和SC表的所有权限授给用户Alice和Jim

    GRANT ALL PRIVILEGES

    ON TABLE Student, SC

    TO Alice, Jim;
  - 把查询Course表和修改课程号的权限授给用户Bob，允许Bob传播此权限

    GRANT SELECT, UPDATE(Cno) ON TABLE Course

    TO Bob WITH GRANT OPTION;

- 权限收回
  - 语法格式

    REVOKE <权限> [, 权限] …

    ON <对象类型> <对象名> [, <对象类型> <对象名>]

    FROM <用户> [, 用户] … [CASCADE | RESTRICT];
    - CASCADE: 级联回收，收回对象传播的权限也被回收
    - RESTRICT: 存在级联回收时拒绝执行权限回收
  - 功能
    - 撤销用户在数据库上的操作权限
  - 例如，

    把关系R上的SELECT权限从用户user收回

    REVOKE select ON R FROM user;

- 权限回收示例
  - 收回用户Bob修改Course表中课程号的权限

    REVOKE UPDATE(Cno) ON TABLE Course

    FROM Bob;
  - 收回用户Alice及其传播的在Student表中的所有权限

    REVOKE ALL PRIVILEGES ON TABLE Student

    FROM Alice CASCADE;
  - 收回所有用户在SC表中的查询权限

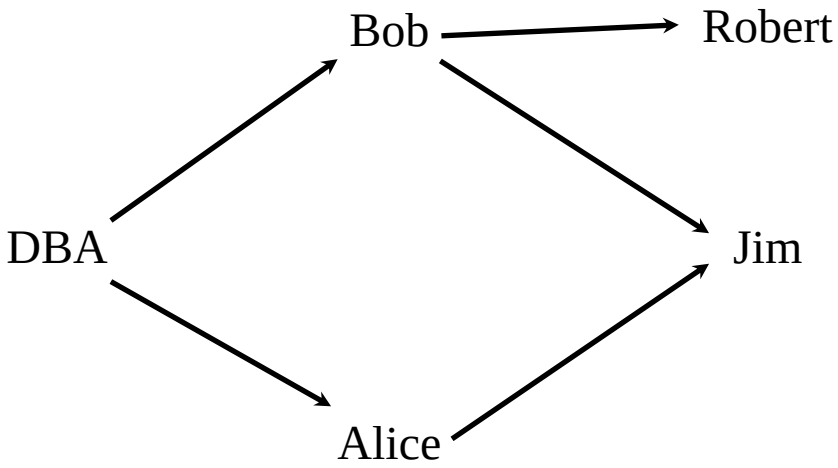
    REVOKE SELECT ON TABLE SC

    FROM PUBLIC;
  - 收回Bob传播在SC表中的查询权限(只收回传播权限)

    REVOKE GRANT OPTION FOR SELECT ON TABLE SC

    FROM Bob;

- **Student的SELECT权限传播**
  - **DBA使用CASCADE方式回收Bob在Student上的SELECT权限后，Robert和Jim是否还拥有该权限？**



Robert的权限被回收

Jim还拥有来自Alice的该权限

  - **DBA使用RESTRICT方式回收Bob在Student上的SELECT权限后，Bob是否还拥有该权限？**
    - **由于存在级联回收，权限回收操作被拒绝了**

- 模式的授权
  - 允许用户创建关系时声明外码
    - 示例
    - GRANT REFERENCES (被参照属性)
      ON 被参照关系
      TO 用户;

      GRANT REFERENCES (Cno) ON Course TO Alice;

CREATE USER并不是SQL标准中的内容，具体使用需要参考特定的DBMS.

- 创建数据库模式的权限
  - 对创建数据库模式一类的数据库对象的授权，由管理员在创建用户时实现
  - 创建用户CREATE USER

    CREATE USER <用户名> [WITH] [DBA | RESOURCE | CONNECT]　　（默认CONNECT）

    只有系统超级用户才能建立新的数据库用户

    新创建的用户有三种权限: CONNECT, RESOURCE, DBA

    CONNECT: 不能创建新用户，不能创建模式，不能创建基本表，只能登录数据库，后续权限由数据库管理员和其它用户授予

    RESOURCE: 能创建基本表和视图，成为这些表和视图的属主，不能创建模式和新的用户，可以将这些存取权限授予其他用户

    DBA: 超级用户，可以创建新用户、模式、基本表和视图等；拥有所有数据库对象的存取权限，可以传播存取权限

- 数据库角色
  - 被命名的一组与数据库操作相关的权限
    - 角色是权限的集合
  - 可以为一组具有相同权限的用户创建一个角色
  - 使用角色来管理数据库权限可以简化授权的过程

- 数据库角色
  - 角色的创建
    - CREATE ROLE <角色名>;
  - 给角色授权
    - GRANT <权限> [, 权限] …
      ON <对象类型> <对象名>
      TO <角色> [, 角色] …;
  - 将角色授予其他角色和用户
    - GRANT <角色> [, 角色] …
      TO <目标角色 | 用户> [,目标角色 | 用户] …
      WITH ADMIN OPTION;

一个用户/角色包含的权限:
- 直接授予这个用户/角色的全部权限
- 这个用户/角色所拥有的角色的全部权限

获得某种权限的角色和用户可以将该权限再授权给其他的角色和用户

- 数据库角色
  - 角色权限的收回
    - REVOKE <权限> [, 权限] …
      ON <对象类型> <对象名>
      FROM <角色> [, 角色] …;
  - 角色的收回
    - REVOKE <角色> [, 角色] …
      FROM <目标角色|用户> [, 目标角色用户] …;

REVOKE的执行者是回收角色的创建者，或者拥有这些角色上ADMIN OPTION的角色/用户

- 角色示例
  - 创建角色instructor

    CREATE ROLE instructor;
  - 把takes表的select、update权限授给角色instructor

    GRANT SELECT, UPDATE ON TABLE takes

    TO instructor;
  - 把角色instructor授给用户Bob、Alice和Jim

    GRANT instructor TO Bob, Alice, Jim;
  - 回收Bob的instructor角色

    REVOKE instructor FROM Bob;
  - 回收instructor角色在表takes上的update权限

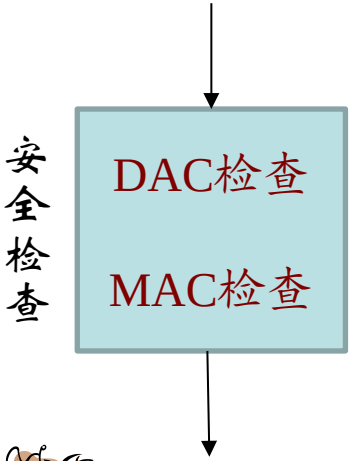
    REVOKE UPDATE ON TABLE takes FROM instructor;

- 以角色的身份授予权限
  - SQL允许权限由一个角色授予,而非用户授予
    - SQL有一个与会话关联的当前角色概念
    - 默认为空
    - 通过“SET ROLE 当前角色;”来设定
      - 指定角色必须已经授予用户，否则失败
    - 授予权限时授权人设置为当前会话关联的角色
      - 授权语句后加“GRANTED BY 当前角色”子句
      - GRANT *instructor* TO Alice GRANTED BY *dean*;

- 存取控制机制
  - 常用存取控制方法
    - 强制存取控制(MAC)：B1级
      - 用户不能直接感知或进行控制
      - 适用于对数据有严格而固定秘密分级的部门
      - 主体：系统中的活动实体，包括用户、代表用户的各个进程
      - 客体：系统中的被动实体，包括文件、基本表、索引、视图等

姓名	性别	年龄	工资
李勇(P)	男(C)	22(C)	5000(S)
张磊(P)	男(C)	25(C)	10000(TS)

- 存取控制机制
  - 强制存取控制Mandatory Access Control (MAC): B1级(续)
    - DBMS为主体和客体的每个实例(值)指派一个敏感度标记label
      - » 例如：绝密TS≥机密S≥可信C≥公开P
      - » 主体的标记称为许可证级别
      - » 客体的标记称为密级
    - 当一个用户以标记label进入系统时，他对任何客体的存取遵循以下规则：
      - » 仅当主体的许可证级别大于或等于客体的密级时，该主体才能读取相应的客体
      - » 仅当主体的许可证级别小于或等于客体的密级时，该主体才能写相应的客体
        - » 10000(TS)和10000(S)是不同的客体
        - » 用户Bob (TS)只能写入10000(TS)



- 审计
  - 审计功能启用一个专用的审计日志(Audit Log)，系统自动将用户对数据库的所有操作记录在上面
  - DBA可以利用审计日志中的追踪信息，重现导致数据库现有状况的一系列事件，以找出非法存取数据的人

C2以上安全级别的DBMS必须具有审计功能

对修改SC结构或修改SC中数据的操作进行审计

AUDIT ALTER, UPDATE ON SC;

取消对修改SC结构或修改SC中数据的操作的审计

NOAUDIT ALTER, UPDATE ON SC;

- 数据加密
  - 防止数据库中数据在存储和传输中失密的有效手段
  - 加密的基本思想
    - 根据一定的算法将原始数据（术语为明文，Plain text）变换为不可直接识别的格式（术语为密文，Cipher text）
    - 不知道解密算法的人无法获知数据的内容

- 数据库安全性
- 数据库完整性

数据库完整性

- 为保证数据库中数据的正确性和相容性，对关系模型提出的某种约束条件或规则。
- 完整性约束通常包括
  - 域完整性
  - 实体完整性
  - 关联完整性（亦称参照完整性）
  - 用户定义完整性

为维护数据库完整性，DBMS必须提供如下功能：
- 定义完整性约束条件的机制
- 完整性检查方法
- 违反约束的处理方法

域完整性，实体完整性和关联完整性，是关系模型必须满足的完整性约束条件！

- 域完整性
  - 保证关系表属性取值的合理性
  - 例如，
    - 属性值应是域中的值
    - 一个属性能否为NULL
    - 一个属性的列值是否唯一
    - 设置一个属性的默认值
  - 域完整性检查
    - 包括检查（CHECK）、默认值（DEFAULT）、不为空（NOT NULL）、列值唯一(UNIQUE)等

- CREATE TABLE时定义
  - 列值非空（NOT NULL）
  - 列值唯一（UNIQUE）
  - 检查列值是否满足一个布尔表达式（CHECK）
  - 设置属性默认值（DEFAULT）

在定义SC表时，说明Sno、Cno、Grade属性不允许取空值。

```
CREATE TABLE SC
  (Sno  CHAR(9)  NOT NULL,
   Cno  CHAR(4)  NOT NULL,
   Grade  SMALLINT NOT NULL,
   PRIMARY KEY (Sno,  Cno),
   /* 如果在表级定义实体完整性，隐含了Sno，Cno不允许取空值，
则在列级不允许取空值的定义就不必写了 */
  );
```

建立部门表DEPT，要求部门名称Dname列取值唯一，部门编号Deptno列为主码，Location默认值设置为‘哈尔滨’

```
CREATE TABLE DEPT
  (Deptno  NUMERIC(2),
   Dname  CHAR(9)  UNIQUE，/*要求Dname列值唯一*/
   Location  CHAR(10) DEFAULT ‘哈尔滨’,
   PRIMARY KEY (Deptno)
  );
```

用CHECK短语指定属性值应该满足的条件

[例7] Student表的Ssex只允许取“男”或“女”。

```
CREATE TABLE Student
   (Sno  CHAR(9) PRIMARY KEY,
    Sname CHAR(8) NOT NULL,
    Ssex  CHAR(2)  CHECK (Ssex IN ('男', '女') ),
      /*性别属性Ssex只允许取'男'或'女' */
    Sage  SMALLINT,
    Sdept  CHAR(20)
   );
```

Check子句:

- Check(P)子句指定一个谓词P，关系中的每个元组都必须满足谓词P
- 根据SQL标准定义，Check子句中的谓词可以包括子查询（使用时需参考具体DBMS）

数据库完整性

- 实体完整性
  - 指关系的主码不能重复也不能取空值

在列级定义主码

```
CREATE TABLE Student
  (Sno  CHAR(9)  PRIMARY KEY,
   Sname  CHAR(20) NOT NULL,
   Ssex  CHAR(2) ,
   Sage  SMALLINT,
   Sdept  CHAR(20));
```

在表级定义主码

```
CREATE TABLE Student
   (Sno  CHAR(9),
    Sname  CHAR(20) NOT NULL,
    Ssex  CHAR(2) ,
    Sage  SMALLINT,
    Sdept  CHAR(20),
    PRIMARY KEY (Sno)
);
```

将SC表中的Sno，Cno属性组定义为码

```
CREATE TABLE SC
    (Sno   CHAR(9)  NOT NULL，
     Cno  CHAR(4)  NOT NULL，
     Grade    SMALLINT，
     PRIMARY KEY (Sno，Cno)    /*只能在表级定义主码*/
    );
```

- 关联完整性定义(参照完整性)
  - 在CREATE TABLE中用FOREIGN KEY短语定义哪些列为外码
  - 用REFERENCES短语指明这些外码参照哪些表的主码

例如，关系SC中一个元组表示一个学生选修的某门课程的成绩，（Sno，Cno）是主码。Sno，Cno分别参照引用Student表的主码和Course表的主码

[例3] 定义SC中的参照完整性

```
CREATE TABLE SC
  (Sno   CHAR(9)  NOT NULL,
   Cno   CHAR(4)  NOT NULL,
   Grade  SMALLINT,
   PRIMARY KEY (Sno, Cno),   /*在表级定义实体完整性*/
   FOREIGN KEY (Sno) REFERENCES Student(Sno),
    /*在表级定义参照完整性*/
   FOREIGN KEY (Cno) REFERENCES Course(Cno)
   /*在表级定义参照完整性*/
  );
```

数据库完整性

- 对被参照关系和参照关系进行增、删、改操作时可能破坏关联完整性，必须进行检查以保证两个关系中数据的相容性
  - 在SC中插入一个元组，插入元组的Sno值在Student中找不到一个元组，其Sno取值与之相等
  - 修改SC中的一个元组，修改后的Sno值在Student中找不到一个元组，其Sno取值与之相等
  - 从Student中删除一个元组，造成SC中某些元组的Sno值在Student中找不到一个元组，其Sno取值与之相等
  - 修改Student中一个元组的Sno属性，造成SC中某些元组的Sno值在Student中找不到一个元组，其Sno取值与之相等

Student(Sno, Ssex, Sage, Dept)
SC(Sno, Cno, Cpno, Grade)

数据库完整性

- 一个操作破坏关联完整性时，系统可以采取的策略
  - 拒绝(No Action)执行
    - 不允许该操作执行，默认策略
  - 级联(Cascade)操作
    - 删除或修改被参照关系中元组时造成与参照关系不一致，删除或修改参照关系中所有不一致的元组
  - 设置为空值
    - 删除或修改被参照关系中元组时造成与参照关系不一致，将参照关系中所有不一致元组的相应属性置为空值

被参照关系(如Student)	参照关系(如SC)	破坏关联完整性时的处理策略
可能破坏关联完整性	插入元组	拒绝
可能破坏关联完整性	修改外码值	拒绝
删除元组	可能破坏关联完整性	拒绝/级联删除/设置为空值
修改主键值	可能破坏关联完整性	拒绝/级联修改/设置为空值

- 关联完整性违约处理说明示例

```
CREATE TABLE SC
( Sno CHAR(9),
  Cno CHAR(4),
  Grade SMALLINT,
  PRIMARY KEY(Sno, Cno),
  FOREIGN KEY(Sno) RERFENCES Student(Sno)
      ON DELETE CASCADE  /* 删除Student表中的元组时，级联删除SC
                            中的元组 */
      ON UPDATE CASCADE  /* 更新Student表中的Sno时，级联更新SC
                            中的相应元组 */
  FOREIGN KEY(Cno) REFERENCES Course(Cno)
      ON DELETE NO ACTION  /* 删除Course表中的元组造成与SC表不一致
                              时，拒绝删除 */
      ON UPDATE CASCADE  /* 更新Course表中的Cno时，级联更新SC
                            中的相应元组 */
);
```

数据库完整性

- 用户定义的完整性
  - 属性上的约束条件
    - NOT NULL
    - UNIQUE
    - CHECK子句
  - 元组上的约束条件
    - CHECK子句可以设置不同属性之间的取值的相互约束条件

```
CREATE TABLE SC
(Sno Char(9) NOT NULL,
 Cno Char(4) NOT NULL,
 Grade SMALLINT CHECK(Grade>=0 AND Grade<=100),
 Primary Key (Sno, Cno),
 … … );
```

```
CREATE TABLE Student
( Sno CHAR(9) PRIMARY KEY,
  Sname CHAR(8) NOT NULL,
  Ssex CHAR CHECK(Ssex IN (‘男’, ‘女’)),
  Sage SMALLINT,
  Sdept CHAR(20),
  CHECK(Ssex=‘男’ OR Sname NOT LIKE ‘Mr.%’));
```

向表中插入元组或修改属性值时，DBMS将检查属性上和元组上的约束条件是否满足，如果不满足则拒绝操作

- 完整性约束命名子句
  - 语法结构
    - CONSTRAINT <完整性约束条件名> <完整性约束条件>
    - <完整性约束条件>包括NOT NULL、UNIQUE、PRIMARY KEY、FOREIGN KEY、CHECK子句等

```
CREATE TABLE Student
( Sno Numeric(6)
      CONSTRAINT C1 CHECK(Sno BETWEEN 900000 AND 999999),
  Sname CHAR(8)
      CONSTRAINT C2 NOT NULL,
  Ssex CHAR
      CONSTRAINT C3 CHECK(Ssex IN ('男', '女')),
  Sage SMALLINT
      CONSTRAINT C4 CHECK (Sage<30),
  CONSTRAINT StudentKey PRIMARY KEY(Sno));
```

- 完整性约束命名子句
  - 语法结构
    - CONSTRAINT <完整性约束条件名> <完整性约束条件>
    - <完整性约束条件>包括NOT NULL、UNIQUE、PRIMARY KEY、FOREIGN KEY、CHECK子句等

```
ALTER TABLE Student DROP CONSTRAINT C1;

ALTER TABLE Student
        ADD CONSTRAINT C1 CHECK (Sno BETWEEN 900000 AND 999999);

ALTER TABLE Student DROP CONSTRAINT C3;

ALTER TABLE Student ADD CONSTRAINT C3 CHECK (Sage < 40);
```

数据库完整性

- 断言Assertion
  - 通过声明断言，可以定义更具一般性的约束，例如涉及多个表或聚集操作的比较复杂的完整性约束
  - 对断言所涉及关系的操作将触发RDBMS对断言的检查
  - 任何使断言不为真的操作都会被拒绝
  - 语法结构
    - CREATE ASSERTION <断言名> <Check子句>;
    - DROP ASSERTION <断言名>;

例. 限制每个学期每一门课程最多由100名学生选修.

```
CREATE ASSERTION SC_MAXSIZE
        CHECK(100>=ALL( SELECT COUNT(*) FROM SC
                        GROUP BY Cno, Semester, Year) );
```

SC(Sno, Cno, Grade, Semester, Year)

复杂的断言验证开销巨大，谨慎使用

数据库完整性

- 触发器Trigger
  - 触发器是用户定义在关系表上的一类由事件驱动的特殊过程
  - 定义之后的触发器存储在数据库服务器中
  - 任何用户对表的增、删、改操作均由服务器自动激活相应的触发器
  - 不同RDBMS实现的触发器语法各不行同，互不兼容

- 触发器Trigger
  - 定义触发器的一般语法

    CREATE TRIGGER <触发器名>
    {BEFORE | AFTER} <触发事件> ON <表名>
    REFERENCING NEW|OLD ROW|TABLE AS <变量>
    FOR EACH {ROW | STATEMENT}
    [WHEN <触发条件>] <触发动作体>;
  - 删除触发器的语法

    DROP TRIGGER <触发器名> ON <表名>;

- 只有表的拥有者（创建者）可以在表上定义触发器
- 触发器名中可以包含模式名，同一模式下触发器不能重名，而且触发器名和表名必须在同一模式下
- 触发器只能定义在基本表上，不能定义在视图上

- 触发器Trigger
  - 定义触发器的一般语法

    CREATE TRIGGER <触发器名>
    {BEFORE | AFTER} <触发事件> ON <表名>
    REFERENCING NEW|OLD ROW|TABLE AS <变量>
    FOR EACH {ROW | STATEMENT}
    [WHEN <触发条件>] <触发动作体>;
  - 删除触发器的语法

    DROP TRIGGER <触发器名> ON <表名>;

- 触发事件
  - INSERT、DELETE或UPDATE
  - 几个事件的组合: INSERT OR UPDATE
  - UPDATE OF <触发列, …>

- 触发器Trigger
  - 定义触发器的一般语法

    CREATE TRIGGER <触发器名>

    {BEFORE | AFTER} <触发事件> ON <表名>

    REFERENCING NEW|OLD ROW|TABLE AS <变量>

    FOR EACH {ROW | STATEMENT}

    [WHEN <触发条件>] <触发动作体>;
  - 删除触发器的语法

    DROP TRIGGER <触发器名> ON <表名>;

- 触发时机
  - AFTER / BEFORE
  - AFTER: 触发事件操作执行之后激活触发器
  - BEFORE: 触发事件操作执行之前激活触发器

数据库完整性

- 触发器Trigger
  - 定义触发器的一般语法

    CREATE TRIGGER <触发器名>

    {BEFORE | AFTER} <触发事件> ON <表名>

    REFERENCING NEW|OLD ROW|TABLE AS <变量>

    FOR EACH {ROW | STATEMENT}

    [WHEN <触发条件>] <触发动作体>;
  - 删除触发器的语法

    DROP TRIGGER <触发器名> ON <表名>;

- 触发器按照所触发动作的间隔粒度分为:
  - 行级触发器(FOR EACH ROW)
  - 语句级触发器(FOR EACH STATEMENT)
  - 在SC表上定义一个After Update触发器，SC有100个元组，执行UPDATE SC SET GRADE=0;
    - 行级触发器的触发动作体执行100次;语句级中执行1次

- 触发器Trigger
  - 定义触发器的一般语法

    CREATE TRIGGER <触发器名>

    {BEFORE | AFTER} <触发事件> ON <表名>

    REFERENCING NEW|OLD ROW|TABLE AS <变量>

    FOR EACH {ROW | STATEMENT}

    [WHEN <触发条件>] <触发动作体>;
  - 删除触发器的语法

    DROP TRIGGER <触发器名> ON <表名>;

- 触发条件:
  - 触发器被激活时，只有当触发条件为真时触发动作体才执行
  - 如果省略WHEN触发条件，触发动作体在触发器激活后立即执行

- 触发器Trigger
  - 定义触发器的一般语法

    CREATE TRIGGER <触发器名>

    {BEFORE | AFTER} <触发事件> ON <表名>

    REFERENCING NEW|OLD ROW|TABLE AS <变量>

    FOR EACH {ROW | STATEMENT}

    [WHEN <触发条件>] <触发动作体>;
  - 删除触发器的语法

    DROP TRIGGER <触发器名> ON <表名>;

- 触发动作体:
  - 可以是一个匿名的PL/SQL过程块,也可以是对已创建存储过程的调用
  - 行级触发器中可以使用NEW ROW和OLD ROW引用UPDATE/ INSERTION之后的新值以及UPDATE/DELETION之前的旧值
  - 语句级触发器使用NEW TABLE和OLD TABLE引用包含所有被影响的行的临时表

Course

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2

SC

学 号 Sno	课程号 Cno	成绩 Grade
200215121	1	92
200215121	2	85

例. 使用触发器表示SC表参照Course表的完整性

```
create trigger check1 after insert on SC
    referencing new row as nrow
    for each row
    when (nrow.cno not in (
            select cno
            from course) )
     begin
        rollback;
     end;
```

例(续). 使用触发器表示SC表参照Course表的完整性

```
create trigger check2 after delete on Course
    referencing old row as orow
    for each row
    when (orow.cno not in (select cno from Course)
            and  orow.cno in (select cno from SC))
     begin
        rollback;
     end;
```

- 仓库库存数据库
  - inventory(item,level):表示物品在仓库中的当前库存
  - minlevel(item,level):表示物品应该保持的最小库存量
  - reorder(item, amount):表示当物品小于最小库存量的时候要订购的数量
  - orders(item, amount):表示物品被订购的数量

```
create trigger recorder 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 = orow.item)
    and orow.level > (select level
                      from minlevel
                      where minlevel.item = orow.item)
    begin atomic
        insert into orders
            (select item, amount
             from reorder
             where reorder.item = orow.item);
    end;
```

用于重新订购物品的触发器

- 录入选课成绩后自动更新学分
  - student(Sno, Sname, Total_Credit) 学号、姓名、现有学分
  - course(Cno, Cname, Credit) 课程号、课程名、学分
  - sc(Sno, Cno, Grade) 学号、课程号、成绩
- 用户定义完整性：学生的总学分等于已修的及格课程的学分之和
- 更新SC中一条选课记录的成绩时，如果成绩之前为空或者低于60分，修改之后不低于60分，则更新student关系中的Total_Credit，在其中加上相应课程的学分。

```
create trigger creditor after update of grade on sc
        referencing old row as orow, new row as nrow
        for each row
        when (orow.grade is null or orow.grade < 60)
                and nrow.grade >= 60
        begin atomic
                update student
                set Total_Credit = Total_Credit + (select Credit
                                                   from course
                                                   where Cno = nrow.Cno)
                where Sno = nrow.Sno;
        end;
```

两个MySQL中触发器的例子

```
CREATE DEFINER=`root`@`localhost` TRIGGER `department_AFTER_DELETE` AFTER DELETE ON `department` FOR EACH ROW BEGIN
    delete from employee where dno = old.dno;
END
```

```
CREATE DEFINER=`root`@`localhost` TRIGGER `trigger1` BEFORE UPDATE ON `testtable` FOR EACH ROW begin
    if old.age > 100 then
        set new.age = old.age;
    end if;
end
```

数据库完整性

- 触发器Trigger
  - 触发器激活顺序

    执行该表上的BEFORE触发器

    激活触发器的SQL语句

    执行该表上的AFTER触发器

    多个BEFORE（AFTER）触发器时，先执行先创建的，或按名字排序

小结

- 数据库安全性
- TCSEC/TDI安全级别划分
- 数据库安全性控制常用方法
  - DAC、SQL授权机制、MAC
- 数据库完整性
- 域完整性、实体完整性、关联完整性、用户定义完整性
- SQL完整性定义机制
  - 属性级、元组级、完整性约束条件命名、断言、触发器

Now let's go to

Next Chapter