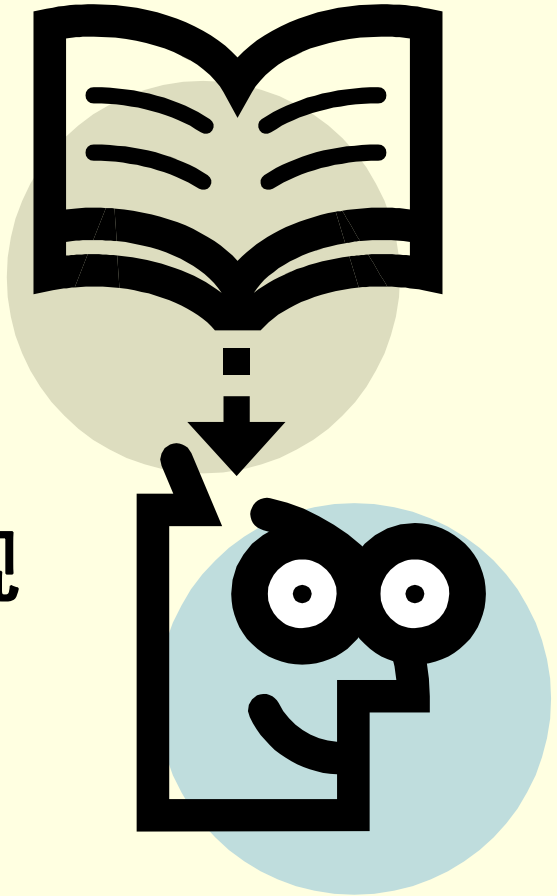
第8章 数据库的安全与完整性约束
Chapter 8 Database Security & Integrity Constraints

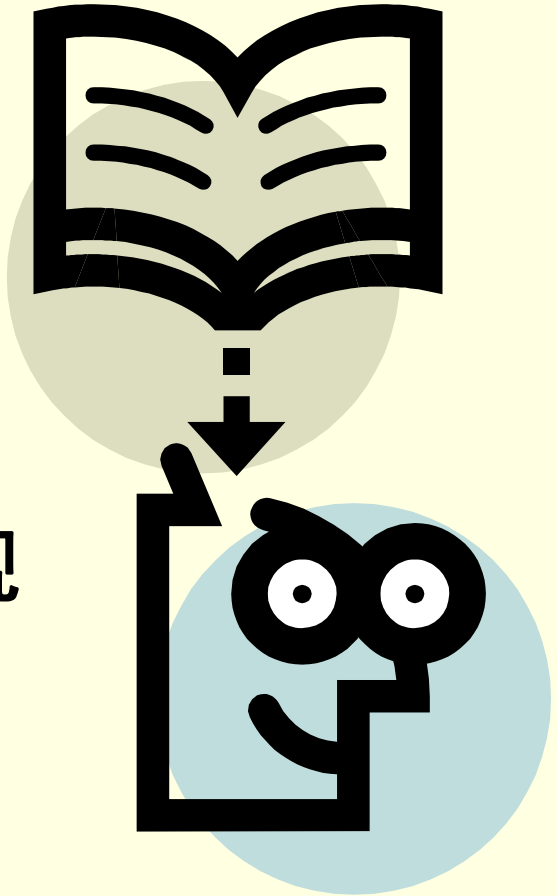
目录 Contents



8.1.1 何谓数据库的安全？

- 数据库的安全（database security）
  - 数据库的安全，是指保护数据库以防止不合法的访问所造成的数据泄密、更改或破坏。
  - 安全 vs. 保密（privacy）
    - 安全——更多的是技术上的问题。
    - 保密——更多的是法律、法规、道德上的问题 。
  - 两个层次的安全
    - 物理——如：防火、防盗、防灾害、防进入、防破坏…
    - 系统——包括：OS的安全机制（确保用户必须经由DBMS才能访问数据库中的数据） & DBMS的安全机制（本课程内容）

目录 Contents



8.1.2 DBMS的安全机制

- **DBMS的安全机制：**
  - 视图机制（view mechanism）
  - 用户标识与鉴别/核实（user identification & verification）
  - 访问控制（access control）
  - 审计（audit）
  - 数据加密（data encryption）

8.1.2 DBMS的安全机制

- 视图机制（view mechanism）
  - 一个最终用户在使用数据库时，其允许访问的数据范围要受到一定限制，即每个用户只能访问数据库中的一部分数据，此种限制可用SQL中的视图功能来实现。
  - 例：定义一个公司有关销售人员的视图。

```
CREATE  VIEW  sales_view(empno, ename, deptno, manager,
                                    salary, commission)
  AS  SELECT  empno, ename, deptno, mgr, sal, comm
      FROM  emp
      WHERE   job＝‘salesman’;
```

8.1.2 DBMS的安全机制

- 用户标识与鉴别（user identification & verification）
  - 用户标识
    - 每个数据库合法用户均有一个用户帐户，其中包括：user-name——作为数据库用户的身份标识（ID）。
    - 例如：Oracle中，缺省用户：SYSTEM, SYS
      DBA事后可创建的用户，例如，User1, User2, ...
  - 鉴别方法
    - user-name + password
    - 其他

8.1.2 DBMS的安全机制

- 访问控制（access control）
  - DBMS根据用户的“访问权限”来控制用户对系统及数据对象的访问方式。——“授权/证实”。
  - 授权（authorization）：对用户访问权限的定义。
    - 集中式授权：由DBA统一定义和管理。也称：强制访问控制（mandatory access control, MAC）。
    - 分散式授权：由DBA及授权的用户分散定义和管理。也称：自主访问控制（discretionary access control, DAC），大多数商用数据库系统（如Oracle）采用。
  - 证实（authentication）
    - 在查询处理时由DBMS统一实施权限检查。只有检查“通过”才能执行相应的操作。

8.1.2 DBMS的安全机制

- 自主访问控制（DAC）
  - 特权（privilege）： 执行一种特殊类型的SQL语句或存取（另一用户的）模式对象的权力。
    - 系统特权（system privileges）：
      - 执行某种特定动作的权力。
      - 系统总是预定义了许多（如：Oracle 9i中116种）命名的系统特权，用于直接授予特定用户/角色。
    - 对象特权（object privileges）：
      - 对某个具体（数据）对象执行某种特定操作的权力。
      - 系统总是预定义了许多对象特权名（如：SQL2标准建议6种；Oracle中8种 ），供对特定用户/角色授权时使用。

8.1.2 DBMS的安全机制

- 自主访问控制（DAC）
  - 系统特权举例：
    - **ALTER ANY TABLE** ——允许被授权者修改任何模式中的任何表/视图的定义
    - **CREATE TABLE** ——允许被授权者在自己模式中创建表
    - **GRANT ANY PRIVILEGE** ——允许被授权者将任何系统特权授给其他用户
  - Oracle的系统特权实际上就是某类语句的执行权限，如：
    - **CREATE SESSION**：连接数据库的权限
    - **TABLE**：CREATE，ALTER，DROP，SELECT，INSERT，UPDATE，DELETE，LOCK 等
    - **PROCEDURES**：CREATE，ALTER，DROP，EXECUTE 等

8.1.2 DBMS的安全机制

- 自主访问控制（DAC）
  - 对象特权举例：
    - **SELECT ON emp** ——允许被授权者在emp表上查询数据
    - **DELETE ON emp** ——允许被授权者在emp表上删除数据
    - **UPDATE (ename, sal) ON emp** ——允许被授权者在emp表的ename列和sal列上更新数据
  - Oracle提供的Object Privileges包括：
    - 表、视图或属性上的**SELECT、INSERT、UPDATE、DELETE**等操作权限
    - 表以及属性上的**REFERENCES**权限
    - 存储过程上的**EXECUTE**权限
    - 表上的**ALTER、INDEX**权限

8.1.2 DBMS的安全机制

- 自主访问控制（DAC）
  - 角色（role）
    - 一组特权的一个命名，可授予其他角色/用户
    - 可通过对角色的授权和回收操作来达到对具有该角色的用户的授权功能
    - 系统总是预定义了一些例行的角色；DBA或授权用户也可以创建其他角色
    - 系统提供的与角色有关的操作
      - **创建、删除、修改角色**
      - **将某个权限授给一个角色**
      - **从一个角色回收某个权限**
      - **将某个角色授给一个用户**
      - **从一个用户回收某个角色**

8.1.2 DBMS的安全机制

- 角色（续）
  - Oracle预定义的一些例行角色:
    - DBA — 拥有全部系统特权，并具有再授权的特权
    - RESOURCE— 略
    - CONNECT— 略
    - IMP-FULL-DATABASE— 略
    - EXP-FULL-DATABASE — 略
  - 好处
    - 简化特权管理
    - 灵活特权管理

8.1.2 DBMS的安全机制

- 自主访问控制（DAC）
  - 授权的SQL DDL / DCL语句
    - 创建角色: **CREATE ROLE** 角色名…;
    - 删除角色: **DROP ROLE** 角色名;
    - 授予系统特权:

    GRANT 系统特权名/角色名… **TO** 用户名/角色名…;
    - 收回系统特权:

    REVOKE 系统特权名/角色名… **FROM** 用户名/角色名… ;
    - 授予对象特权:

    GRANT 对象特权名… **ON** 对象标识 **TO** 用户名/角色名…;
    - 收回对象特权:

    REVOKE 对象特权名… **ON** 对象标识 **FROM** 用户名/角色名… ;

8.1.2 DBMS的安全机制

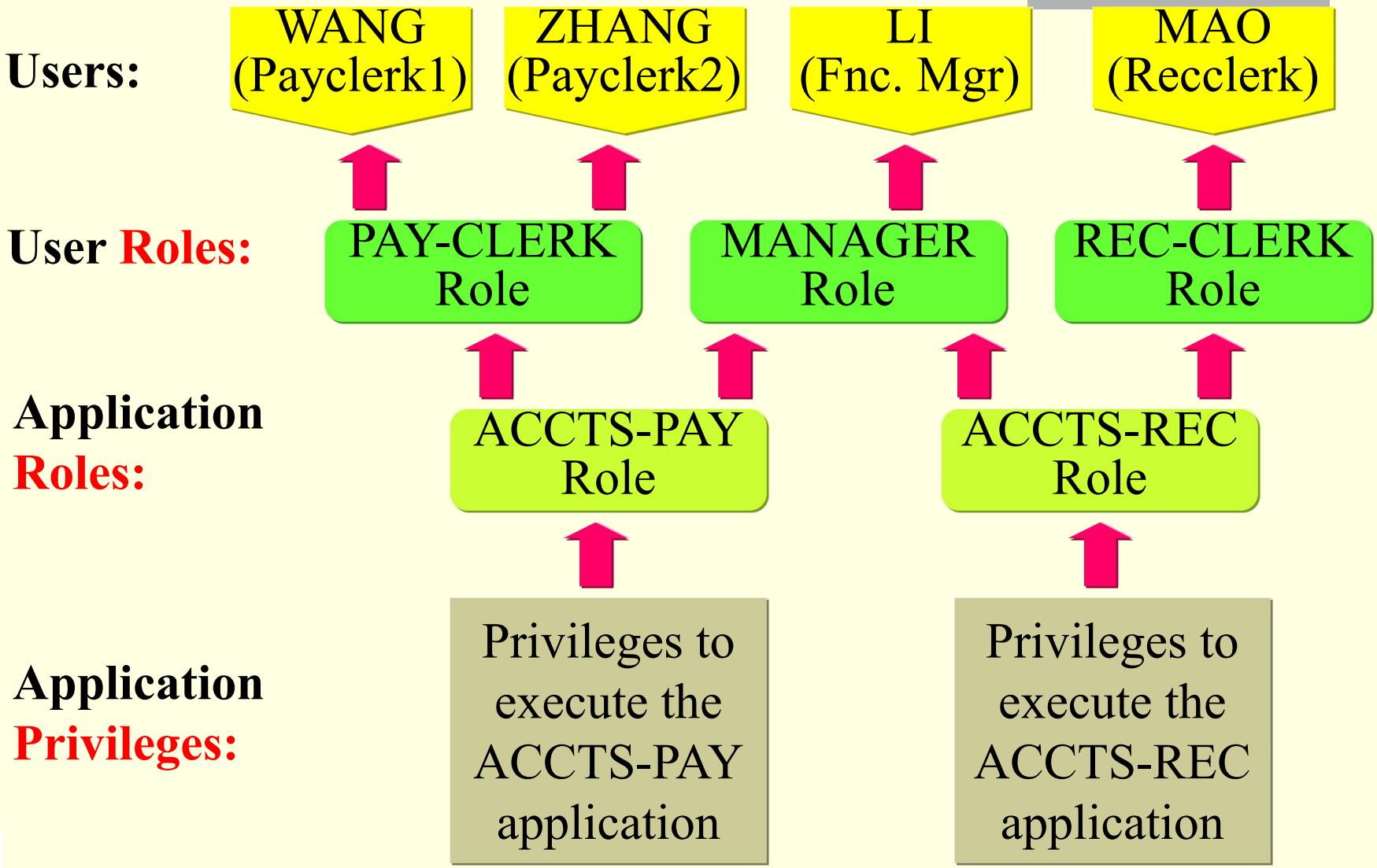
- 自主访问控制（DAC）
  - SQL DDL / DCL例：

```
CREATE ROLE teller IDENTIFIED BY cashflow ;
DROP ROLE teller ;
```

```
GRANT create view, select any table TO richard, thomas ;
GRANT create table, select any table TO teller ;
GRANT teller TO travel_agent WITH ADMIN OPTION ;
GRANT create session TO PUBLIC ;
GRANT DBA TO wang ;
REVOKE create view FROM thomas;
REVOKE DBA FROM wang ;
```

```
GRANT select, update(ename, sal) ON emp TO blake;
REVOKE update ON emp FROM blake ;
```

补充：How to Build a Security Schema for a Database Application System?



8.1.2 DBMS的安全机制

- 审计（audit）
  - 方法
    - 对选定的用户动作进行记录和监控。
    - 作用：“威慑”＋“证据”。
    - DBMS将审计记录写入审计痕迹表（audit trail）中：

      操作的终端号、用户名；操作所涉及的数据对象；

      操作的类型；操作发生的日期、时间。
  - 审计事件的设置
    - 用户审计：由用户设置审计事件与审计对象
    - 系统审计：由DBA设置审计事件与审计对象

8.1.2 DBMS的安全机制

- 审计（续）
- Oracle 8i的三个层次上的审计功能
  - 语句审计（statement auditing）
    - 对某种类型的SQL语句进行审计，不指定对象。
  - 特权审计（privilege auditing）
    - 对执行相应动作的系统特权的使用进行审计。
  - 模式对象审计（schema object auditing）
    - 对某个数据库对象（如表、索引等）上特定类型操作的审计。

- 上述的审计功能既可以是作用在所有用户上的，也可以是针对特定几个用户的操作进行审计。
- 对上述的每一类被审计事件，既可以对其成功执行进行审计，也可以只审计执行失败的情况，或者两者都进行审计。

8.1.2 DBMS的安全机制

- 数据加密（data encryption）
  - 方法
    - 数据加密存储/传输。DBMS必须支持数据加密/解密。

  - 原因
    - 对付“绕开DBMS”而非法访问DB，如：
      - 窃取DB存储介质；在通信线路上窃听；etc.
  - 商业DBMS很少用
    - DB性能方面考虑。
    - Oracle 只支持password的加密/解密。

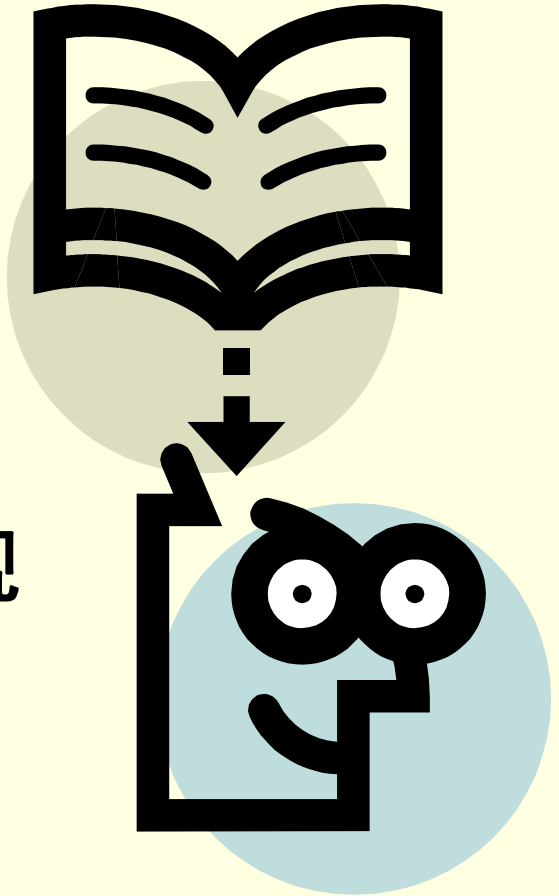
补充：Oracle中访问控制的附加机制

- 设置用户表空间的空间限额
  - 缺省情况下没有空间使用限额
  - DBA可以指定用户表空间的使用限额
- 在用户环境文件中限制用户资源
  - 防止用户无限制地消耗系统资源，可设置：
    - CPU时间
    - 内存量
    - 数据块I/O数目
    - 会话空闲时间
    - 每个用户的并行会话数，etc.

WWW总结

- **What?**
  - 数据库安全?　- 特权?　- 角色?
  - 系统特权?　- 对象特权?　- SQL DCL?
- **Why?**
  - 要安全控制?　- 用户password要保密、常换?
  - 常用分散授权?　- 有了特权，还要引入角色?
  - 要审计?　- 商业DBMS一般不支持加密?
- **How?**
  - DBMS如何进行安全控制?
  - DBA/一般用户如何使用SQL DCL进行授权/收权?

目录 Contents



8.2.1 完整性约束及其类型

- 完整性约束（integrity constraints）
  - 完整性约束：语义施加在数据上的限制，旨在维护（数据库更新过程中的）数据一致性
  - **Integrity constraints** ensure that changes made to the database *by authorized users* do **not** result in a loss of **data consistency**. 完整性约束确保授权用户对数据库所做的更改不会导致丧失数据一致性（i.e., 数据不一致）。
  - 利用完整性约束定义与检查机制，可防止无效或错误数据进入DB，从而保证DB始终处于正确、一致的状态（i.e., 符合实际语义的状态）。
  - 一个完整的DB设计，除了DB“结构”设计外，还应包括“完整性约束”的设计。完整性约束是DB模式的一部分。

8.2.1 完整性约束及其类型

- 与“完整性约束” 相关的有三个组成部分
  - 完整性约束的定义
  - 完整性约束的检查
    - 在 DBMS 中的检查模块
  - 违反完整性约束的处理
    - 用户的操作会破坏数据的完整性（即违反完整性约束的要求）时，系统将：
      - 拒绝执行，并报警或报错；
      - 按照预先定义的动作进行处理，例如：
        - 在外键定义中规定的处理
        - 在触发器（triggers）定义中规定的处理

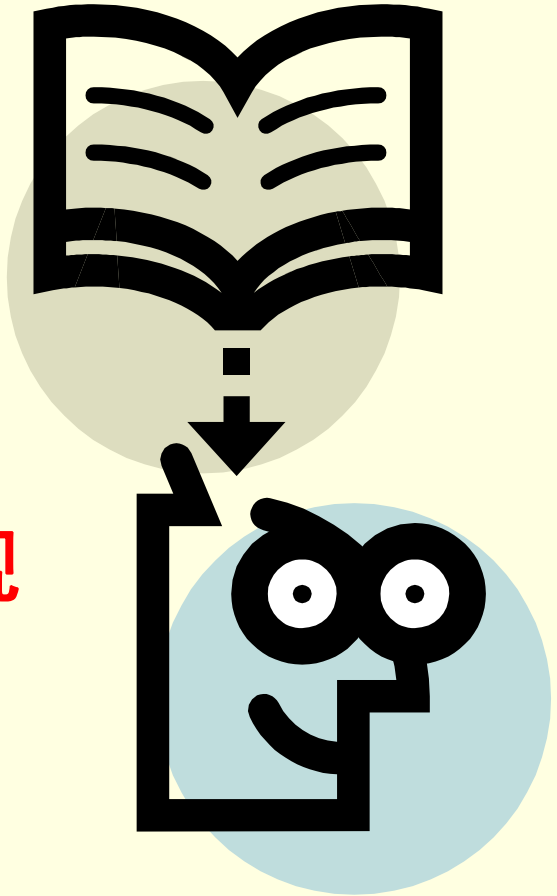
8.2.1 完整性约束及其类型

- 完整性约束的类型
  - 静态约束（static constraints）：对DB状态的约束。
    - 固有约束（inherent constraints）：指数据模型固有的约束。例如： 关系模型中的1NF条件
    - 隐含约束（implicit constraints）：指隐含在数据模式中的约束，用DDL来申明，约束定义存于DD中。对关系DB，包括：
      - 域完整性约束（domain integrity constraints）：
        - 属性值应在域中取值；属性值是否可取null值
      - 实体完整性约束（entity integrity constraints）：
        - 每个关系必须有一个键（key），每个元组的键值应唯一，且不能为null
        - 在SQL实现时，可为每个关系选定一个主键（PK）
      - 引用完整性约束（referential integrity constraints）：
        - 一个关系中的FK值必须引用（另一个关系或本关系中）实际存在的PK值，否则只能取null值

8.2.1 完整性约束及其类型

- 完整性约束的类型（续）
  - 静态约束 （续）
    - 显式约束（explicit constraints）：指更为广泛的语义约束，通常与特定的应用领域有关，需用特定的DDL语句来申明，约束的定义存于DD中
  - 动态约束（dynamic constraints）：对DB状态演变的约束，通常与特定的应用领域有关，同样需用特定的DDL语句来申明，约束的定义存于DD中
  - 显式约束与动态约束源于特定应用领域的业务规则（business rules），在关系数据库中称一般完整性约束（general constraints）

目录 Contents



8.2.2 完整性约束的SQL实现

- 完整性约束的SQL实现
  - 根据E.F. Codd的观点，一个关系数据库管理系统（RDBMS）对以上完整性约束的支持程度体现了其对“关系模型”的支持程度。
  - SQL标准（e.g. SQL2及SQL3等）明确提出了实现完整性约束的要求与机制。
  - 大型商用RDBMS （e.g. Oracle, Sybase）通常提供了诸多机制来实现完整性约束；而小型RDBMS （e.g. PC上的Xbase）则对完整性约束的支持往往很不够。
  - 当DBMS没有提供足够的机制来支持完整性约束时，完整性约束的说明与检查任务就落在应用程序的身上，此时，称用过程说明约束。
    但其缺点是显然的：①加重了程序（员）的负担；②约束分散，难于统一管理与维护。

8.2.2 完整性约束的SQL实现

- RDBMS实现完整性约束的各种机制

<table>
<tr><th colspan="3">机 制（语句构造子）</th><th colspan="2">实现的完整性约束类型</th><th>说 明</th></tr>
<tr><td rowspan="8">在基表定义中</td><td colspan="2">数据类型（+规则）</td><td colspan="2" rowspan="2">域完整性约束</td><td rowspan="8">SQL2 / SQL:1999标准所要求；除ASSERTION外，大型商用RDBMS大多已实现。</td></tr>
<tr><td colspan="2">NOT NULL</td></tr>
<tr><td colspan="2">UNIQUE</td><td colspan="2" rowspan="2">实体完整性约束</td></tr>
<tr><td colspan="2">PRIMARY KEY</td></tr>
<tr><td colspan="2">FOREIGN KEY</td><td colspan="2">引用完整性约束</td></tr>
<tr><td rowspan="2">CHECK</td><td>基于属性</td><td>属性层</td><td rowspan="3">显式完整性约束</td></tr>
<tr><td>基于元组</td><td>元组层</td></tr>
<tr><td>用断言</td><td colspan="2">ASSERTION</td><td>关系层</td></tr>
<tr><td>用触发器</td><td colspan="2">TRIGGER</td><td colspan="2">动态完整性约束</td><td>SQL:1999标准所要求；大多已实现。</td></tr>
</table>

8.2.2 完整性约束的SQL实现

- 举例说明
  - 在基表定义中说明的约束

约束名

```
CREATE TABLE dept (
    deptno INT PRIMARY KEY CONSTRAINT pk-dept,
    dname VARCHAR(10) UNIQUE CONSTRAINT unq-dname,
    loc VARCHAR(10) CHECK (loc IN ('Shanghai','Nanjing',
                'Wuhan', 'Xian', 'Beijing')) CONSTRAINT ck-loc
);
```

注："CONSTRAINT 约束名"可以省略。
当定义中未指定约束名时，系统会自动赋予它一个约束名。
如Oracle中：SYS-Cn 。在视图dba-constraints中可查询到。

8.2.2 完整性约束的SQL实现

- 举例说明 （续）

```
CREATE TABLE emp (
    empno INT,
    ename VARCHAR(10) NOT NULL CONSTRAINT nn-ename
        CHECK (ename=UPPER(ename)) CONSTRAINT upper-ename,
    job VARCHAR(9) CHECK (job IN (SELECT job-id FROM job-list)),
    mgr INT REFERENCES emp(empno) CONSTRAINT fk-mgr,
    hiredate DATE DEFAULT SYSDATE,
    sal DEC(7,2) CHECK (sal>1000.0) CONSTRAINT ck-sal,
    comm DEC(7,2) DEFAULT 0.0,
    deptno INT NOT NULL CONSTRAINT nn-deptno,
    PRIMARY KEY (empno) CONSTRAINT pk-emp,
    FOREIGN KEY (deptno) REFERENCES dept(deptno)
        ON DELETE CASCADE CONSTRAINT fk-deptno,
    CHECK (sal+comm <= 20000.0) CONSTRAINT ck-earnings
);
```

列约束

表约束

基于属性的CHECK

级联删除（待讲）

基于元组的CHECK

8.2.2 完整性约束的SQL实现

FK及引用完整性约束

可能违反引用完整性约束的数据库操作：
③ DELETE一个（已被引用的）PK值
④ UPDATE一个（已被引用的）PK值

被引关系（referenced relation）

dept

PK

deptno		
xxx		
yyy		
zzz		

施引关系（referencing relation）

emp

FK

	dno	
	yyy	
	NULL	
	ddd	

悬空（dangling）

无参照的FK值

违反引用完整性约束的数据库操作：
① INSERT一个非空的、无参照的FK值
② UPDATE为一个非空的、无参照的FK值

维护引用完整性约束的三种可选策略（默认为“禁止”）：

禁止(Restrict)： a. DBMS拒绝执行操作①/②；b. DBMS也拒绝执行操作③/④。

级联(Cascade)： a.同上；b.执行③/④的同时连锁DELETE/UPDATE施引关系的相应元组。

置空(Set-null)： a.同上；b.执行③/④的同时将施引关系的相应元组的FK值置为NULL 。

8.2.2 完整性约束的SQL实现

- **全局约束（global constraints）：**
  用断言（ASSERTION）说明的约束。
  - 在创建基表时，可定义基于属性或元组的CHECK约束，且可在约束条件中查询其它关系。但CHECK约束有局限性。
  - 例：若不允许‘计算机系’（CS）的学生选修‘R程序设计’课程（R），则在创建‘选课’关系时可定义如下完整性约束：

```
CHECK (
   NOT ( (sno IN (SELECT sno FROM student WHERE sd='CS') )
          AND (cno IN (SELECT cno FROM course WHERE cn='R')) )
);
```

  - 存在问题：该**CHECK**约束只对定义它的‘选课’关系起作用，而对student关系不起作用。故这种CHECK约束会导致以下问题：
    如果一个正在选修‘R程序设计’课程的学生从别的系转到了计算机系【操作：将student关系中相应学生的“系”值修改为‘CS’】，这无疑破坏了‘选课’关系的完整性，但**CHECK**约束不能阻止！

8.2.2 完整性约束的SQL实现

- 用断言说明的约束
  - CHECK约束仅对定义它的关系中的相应属性/元组起作用。为了使约束对整个数据库（中的所有相关关系）均起作用，SQL2 / SQL:1999标准引入了一种称为断言（assertion）的全局约束机制，用于声明数据库状态必须满足的条件。
  - 当数据库更新事件发生时，DBMS检测这种“更新”是否会导致断言中“条件”的不能满足，若是，则拒绝这种“更新”的执行。
  - 定义断言：

    CREATE ASSERTION <name> CHECK(<condition>);
  - 撤消断言：

    DROP ASSERTION <assertion-name-list>;

8.2.2 完整性约束的SQL实现

- 用断言说明的约束（续）
  - 例：【对单个关系的整体进行约束】
    每门课程的选修人数不应少于10人。

```
CREATE ASSERTION ass_1 CHECK (
    10 <= ALL ( SELECT COUNT(*)
                FROM sc          // sc为学生选课关系
                GROUP BY cno )
);
```

8.2.2 完整性约束的SQL实现

- 例：【对多个关系间的联系进行约束】
  学生在选修‘数据结构’（DS）课程之前必须先学过‘C++程序设计’（CCP）课程。

```
CREATE ASSERTION ass_2 CHECK
( NOT EXISTS
    ( SELECT * FROM sc
      WHERE cno IN ( SELECT cno FROM course
                     WHERE cn = 'DS' ) AND
            sno NOT IN ( SELECT sc1.sno
                         FROM sc sc1, course
                         WHERE sc1.cno = course.cno
                               AND course.cn='CCP' )
    )
);
```

‘DS’ 的课程号

学过‘CCP’的学号

此连接查询可改写为嵌套查询

8.2.2 完整性约束的SQL实现

- 例：【对多个关系间的联系进行约束】
  学生在选修‘数据结构’（DS）课程之前必须先学过‘C++程序设计’（CCP）课程。

```
CREATE ASSERTION ass_2 CHECK
( NOT EXISTS
    ( SELECT * FROM sc
      WHERE cno IN ( SELECT cno FROM course
                     WHERE cn = 'DS' ) AND
            sno NOT IN ( SELECT sc1.sno
                         FROM sc sc1
                         WHERE sc1.cno IN
                           ( SELECT cno FROM course
                             WHERE cn = 'CCP' ) )
    )
);
```

‘DS’ 的课程号

学过‘CCP’的学生的学号

子查询：‘CCP’ 的课程号

8.2.2 完整性约束的SQL实现

- **前面的例子说明：**

  在定义断言的条件时，由于SQL不提供 *“for all X, P(X)”*（其中，*P*是一个谓词）的逻辑结构，因此，我们只能写成：***“not exists X such that not P(X)”***。

  另外，也可在某个关系的某个列上使用**COUNT()、SUM()**等SQL聚集函数，将其结果与某个常量进行比较。

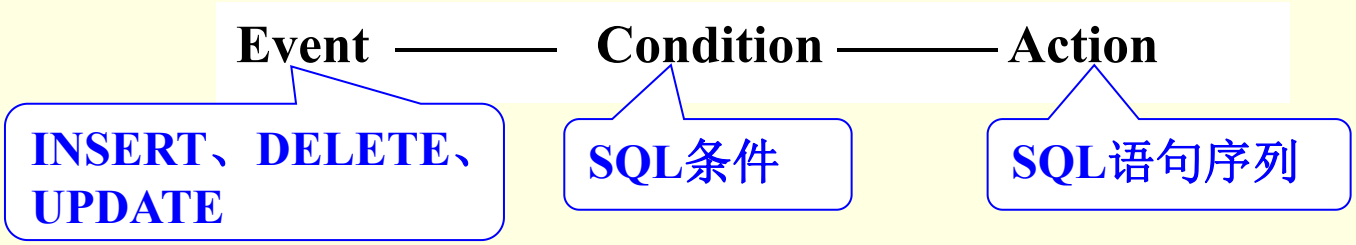
- **总结：**

  断言实际上定义了一种通用约束（general constraints），从这种意义上说，前述的域完整性约束、引用完整性约束和CHECK约束均是特殊类型的断言。

  断言机制方便了程序员声明约束，但会导致DBMS进行复杂的“检测”操作，因此，断言在实际DBMS产品中是难于实现的（大多没有去实现）。

8.2.2 完整性约束的SQL实现

- 用触发器说明的约束
  - 触发器（trigger）：在数据库系统中，一个事件的发生导致另一些事件的发生，这样的功能被称为触发器。
  - 触发器也称为ECA rules（“事件-条件-动作”规则），是DBMS中的主动（active）机制。当“事件”发生时，DBMS检测“条件”是否满足，若满足，则执行“动作”。



  - 触发器的主要功能：触发器最初用于维护完整性约束，但现在已远远超出了此范围，也被用于其它目的，如：
    - 数据的安全性保护
    - 用户的应用逻辑处理
    - 数据库系统的主动功能

8.2.2 完整性约束的SQL实现

- 用触发器说明的约束（续）
  - 触发器的类型

选项	FOR EACH ROW	无 / FOR EACH STATEMENT
BEFORE	行前触发器	语句前触发器
AFTER	行后触发器	语句后触发器
INSTEAD OF	行替代触发器	语句替代触发器

  - 注： SQL:1999标准没有替代（INSTEAD OF）触发器；各RDBMS产品对触发器的实现语法差异很大，如：ORACLE无需指定FOR EACH STATEMENT即表示语句触发器。

8.2.2 完整性约束的SQL实现

- 用触发器说明的约束（续）
  - 例：【行后触发器——SQL:1999语法】

```
CREATE TRIGGER sal_never_lower
AFTER UPDATE OF sal ON emp   /* 事件：更新emp表上sal列值*/
   REFERENCING
        OLD ROW AS oldtuple,  /* 建立过渡变量（transition variable），
                                 表示旧元组 */
        NEW ROW AS newtuple  /* 建立过渡变量，表示新元组 */
FOR EACH ROW
     WHEN newtuple.sal < oldtuple.sal    /* 条件：当薪水值变低时 */
     UPDATE emp          /* 动作：薪水值恢复更新前旧值 */
        SET sal = oldtuple.sal
        WHERE empno = newtuple.empno;
```

下一章详介触发器

The End

- 第八章作业：
  - 教材Page 182：习题8的第1、2题。
  - **提醒：请在截止时间（11月12日23:59）之前提交答案！**