第8章

数据库的安全与完整性约束

2012.04

目录 Contents

8.1.1 何谓数据库的安全？

- 数据库的安全(database security)
  - 防止非法使用数据库。即要求数据库的用户：
    - 通过规定的访问途径
    - 按照规定的访问规则（规范）
    - 访问数据库中的数据，并接受来自DBMS的各种检查。
  - 安全
    - 保护数据库，使其免受因不合法的存取所造成的数据泄密、更改甚至破坏。
  - 安全 & 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、防破坏、…。
    - 系统----包括：OS安全机制 / DBMS安全机制。

8.1.1 何谓数据库的安全？

- 访问数据库的规范有多种，但是：
  - 不同的规范适用于不同的应用。
  - 可以根据应用的不同需求来选择不同的数据库访问规范，即选择具有不同安全级别的数据库。
  - 那些能适应网络环境下安全要求级别的数据库称为安全数据库(secure database)，或称为可信数据库(trusted database)。

目录 Contents

8.1.2 DBMS的安全机制

- **视图机制**
- **用户标识与鉴别（User Identification & Verification）**
- **存取控制（Access Control）**
- **审计（Audit）**
- **数据加密（Data Encryption）**

8.1.2 DBMS的安全机制

- 视图机制
  - 一般用户在使用数据库时，其使用范围要受到一定的限制，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  salesman_view (empno, ename, deptno, mgr,
                             sal, com)
  AS  SELECT  empno, ename, deptno, mgr, sal, com
      FROM      emp
      WHERE   job＝ ‘salesman’;
```

8.1.2 DBMS的安全机制

- 用户标识与鉴别
  - 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事后可创建用户：User1, User2, ...
  - 鉴别方法
    - User-Name
    - Password
    - 其他

8.1.2 DBMS的安全机制

- 存取控制
  - DBMS根据用户的“存取权限”来控制用户对系统及数据对象的存取方式。—“授权/证实”。
  - 授权(Authorization): 对用户存取权限的定义。
    - 集中式授权：由DBA统一定义和管理。
    - 分散式授权：由DBA及授权的用户分散定义和管理。
    - 也称：任意存取控制技术(Discretionary Access Control)，常用，如: System R, Oracle, Sybase, etc.
  - 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

- 存取控制(cont.)
  - 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 Privilege)
      - 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，用于直接授予用户/角色。
    - 对象特权(Object Privilege)
      - 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 ），供对用户/角色授权时使用。

8.1.2 DBMS的安全机制

- 存取控制(cont.)
  - 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 ANT 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，例如：
    - **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

- 存取控制(cont.)
  - 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、sal列上更新数据。
  - Oracle所提供的Object privilege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

- 存取控制(cont.)
  - 角色(Role)
    - 一组特权集合的一个命名，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、修改功能**
      - **将某个权限授给一个角色**
      - **从一个角色那里回收某个权限**
      - **将某个角色授给一个用户**
      - **从一个用户那里回收某个角色**

8.1.2 DBMS的安全机制

- 角色(cont.)
  - 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

- 存取控制(cont.)
  - 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

- 存取控制(cont.)
  - 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  ;
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 ;
```

8.1.2 DBMS的安全机制

- 审计
  - 方法
    - 对选定的用户动作进行监控和记录。
    - 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(cont.)
- 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
  - 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”而非法存取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。

小结

- **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)
  - 语义施加在数据上的限制。
  - 利用完整性约束机制，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。
    - 正确性：数据的有效性、有意义
    - 一致性：在多用户(多程序)并发访问数据库的情况下，保证对数据的更新不会出现与实际不一致的情况。
  - 我们一方面要避免在数据库中出现错误的数据，即防止数据的完整性受到破坏。同时，如果因为某些不可抗拒的原因而导致数据库中的数据被破坏，也要能够及时发现并采取一定的措施将数据库中的数据恢复到正确的状态中。
  - 一个完整的DB设计，除了关于DB“结构”的设计外，还应包括“完整性约束”的设计。

8.2.1 完整性约束及其类型

- 完整性约束(cont.)
  - 对数据库中数据的正确性和移植性的维护，包括
    - 在执行更新操作时，检查是否违反完整性约束条件，并且在证明其无效后作出适当的反应。
    - 防止有存取权限的合法用户的误操作。
- 完整性约束的目的
  - 及时发现错误
  - 能够采取措施防止错误的进一步蔓延
  - 最终将数据库恢复到正确状态
- 完整性约束的实现措施
  - 完整性约束条件的定义及检查
  - 触发器
  - 并发控制技术

8.2.1 完整性约束及其类型

- 一条完整性约束规则一般有三个组成部分
  - 完整性约束条件的设置
  - 完整性约束条件的检查
    - 在 DBMS 内部设置专门的软件检查模块。
  - 完整性约束条件的处理
    - 在用户的操作会破坏数据的完整性（即违反完整性约束条件的要求）时，系统将：
      - 拒绝执行，并报警或报错；
      - 调用相应的函数（例程）进行处理，如：
        - 在外键定义子句中给出的处理方法
        - 在触发器中给出的处理过程

8.2.1 完整性约束及其类型

- 完整性约束条件的类型
  - 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。e.g. 关系数据模型中的1NF条件，etc.
    - 隐含约束(Implicit Constraints)：指隐含在数据模式中的约束，通常需用DDL来申明，约束的定义存于DD中。对关系模型来说，包括：
      - 域完整性约束(Domain Integrity Constraints)：
        - 属性值应在域中取值、属性值是否可为NULL
      - 实体完整性约束(Entity Integrity Constraints)：
        - 每个关系必须有一个键，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。

8.2.1 完整性约束及其类型

- 完整性约束的类型(cont.)
  - 静态约束(cont.)
    - 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，其需用特定的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，其同样需用特定的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，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)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 则对完整性约束的支持往往很不够，通常仅支持数据类型的说明及唯一索引机制。
  - 当DBMS没有提供足够的支持完整性约束的机制时，完整性约束的说明与检查的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

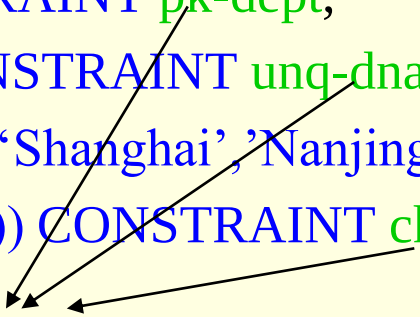
- RDBMS实现完整性约束的各种机制

<table>
<tr><th colspan="3">机 制</th><th colspan="2">实现的完整性约束</th><th>说 明</th></tr>
<tr><td rowspan="7">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；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；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

```
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);
```



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 (cont.)

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dba-constraints视图中可查询到。

```
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

基于元组的CHECK

8.2.2 完整性约束的SQL实现

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

被引用关系(the Referenced Relation)

Table: dept **PK**

	deptno	
	xxx	

悬空(dangling)

引用关系(the Referencing Relation)

Table: emp **FK**

	deptno	
	xxx	

FK及引用完整性约束

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

维护引用完整性约束的三种策略：

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

②**级联**(Cascade)：a.同上；b.执行③/④的同时连锁DELETE/UPDATE引用关系的相应元组

③**置空**(Set-Null)：a.同上；b.执行③/④的同时将引用关系的相应元组值置为NULL

8.2.2 完整性约束的SQL实现

- 全局约束：用断言说明的约束
  - 在基表的定义命令中，可以定义很复杂的、基于属性或元组的完整性约束条件，并且可以在约束条件中使用到其它的关系。
  - 例如：不允许‘英语系’(English)的学生选修‘CET-4’课程。则在创建‘选课’关系时可以定义一个完整性约束：

```
CHECK ( NOT
    ((SNO IN (SELECT SNO FROM Student WHERE Sd='English') )
    AND (CNO IN (SELECT CNO FROM Course WHERE Cn='CET-4'))
) );
```

  - ？问题：该完整性约束只对定义它的‘SC’关系起作用，而对‘Student’关系则不起作用。如果一个正在选修CET-4课程的学生转到英语系（将‘系别’属性的值修改为‘English’），这无疑会破坏‘SC’关系的数据完整性。

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的相应属性/元组起作用。为了使约束对整个关系模式(中的所有相关关系)均起作用，SQL2 / SQL:1999引入了一种全局约束(Global Constraints)机制——断言(Assertion)，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“条件”不满足，若是，则拒绝这种“更新”的执行。

8.2.2 完整性约束的SQL实现

- 单个关系中涉及到统计操作的约束条件
- 多个关系之间复杂的约束条件

- 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

- 举例说明
  - 用断言说明的约束(cont.)
  - [例]: (对单个关系的整体)每门课程的选修人数不少于20人。

```
CREATE ASSERTION ass_1 CHECK (
        20 >= ALL ( SELECT COUNT(*)
                    FROM SC
                    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)学生在选修‘数据结构’(DS)课程之前必需先选修过‘C++程序设计’(CPP)课。

```
CREATE ASSERTION ass_2 CHECK (
  NOT EXISTS (
    SELECT * FROM SC
    WHERE CNO IN ( SELECT CNO FROM Course
                   WHERE CNAME = 'DS' )
    AND
    SNO NOT IN ( SELECT SC1.SNO
                 FROM SC SC1, Course
                 WHERE SC1.CNO = Course.CNO
                   AND Course.CNAME='CPP' )
  ) );
```

8.2.2 完整性约束的SQL实现

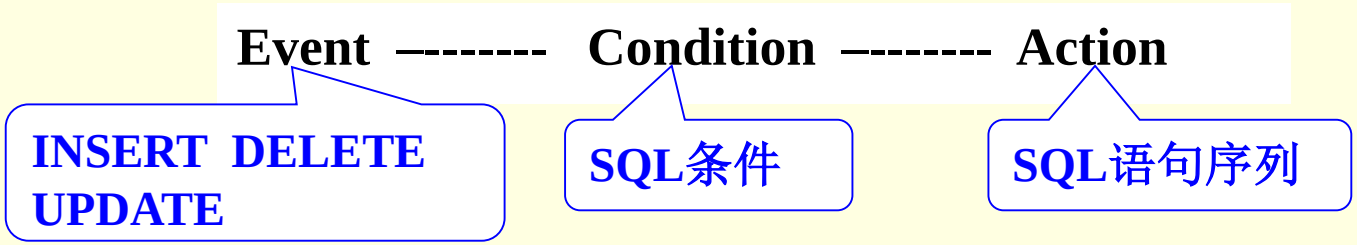
- 断言实际上定义了一种通用的约束(General Constraints)，从这种意义上说，前述的域完整性约束、引用完整性约束和CHECK约束是特殊类型的断言。
- 断言机制方便了程序员声明约束，但会导致DBMS复杂的“检测”动作，因此，断言在DBMS产品中是难于实现的(大多没有实现)。
- 在定义断言的条件时，由于SQL不提供 *“for all X, P(X)”*(其中，*P*是一个谓词)结构，因此，我们只能写：***“not exists X such that not P(X)”***。
- 另外，也可在某个关系的某个列上使用SUM()一类的SQL聚集函数，将其结果与某个常量比较。

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，一个事件的发生会导致另外一些事件的发生，这样的功能被称为触发器。
  - 触发器也称ECA rules, 是DBMS中的主动机制。当“事件”发生时，DBMS检测“条件”是否满足，若满足，则执行“动作”。
  - 触发器的功能：某个事件的发生会导致另外一些事件的执行，以消除前一个事件对数据完整性所起的影响。
  - 触发器最初是用于数据的完整性约束，但现在已经远远超出了此范围，也被应用于其它的目的，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

- 触发器的组成
  - 触发事件 (由用户定义)
    - 通常为某个完整性约束条件的否定或某种数据操纵事件
    - 如：用户登录，数据的增、删、改等
  - 结果事件 (由用户定义)
    - 当触发事件发生时，用以消除触发事件所引起的负面影响的程序。
    - 通常是一组由用户书写的SQL命令
  - 触发过程
    - 当 DBMS 检测到触发事件的发生时，自动调用并执行结果事件的过程。



8.2.2 完整性约束的SQL实现

- 用触发器说明的约束(cont.)
  - 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对触发器的实现语法差异很大，如：ORACLE无需指定FOR EACH STATEMENT。

8.2.2 完整性约束的SQL实现

- 举例说明—用触发器说明的约束(cont.)
  - [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 */
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 oldtuple.sal > newtuple.sal     /* 条件：当薪水值变低时 */
     UPDATE emp
     SET sal = oldtuple.sal               /* 动作：薪水值恢复 */
     WHERE empno = newtuple.empno;
```