Chapter 5 The Security and Integrity Constraints

1. Introduction

- 数据库的破坏通常由以下因素引起：1和2应由数据库管理系统的恢复机制（Chapter 4）解决；3属于数据库安全；4属于完整性约束。
  - 系统故障
  - 并行访问引起的冲突
  - 人为破坏（有意或无意）
  - 数据输入不正确，更新事务未遵守一致性维护规则

2. 数据库安全 DB Security

2.1. 简介

- 保密性Secrecy：用户不应该看到他们不应该看到的东西
- 完整性Integrity：用户不应该能够修改他们不应该修改的东西
- 可用性Availablility：用户应该能够看到和修改他们被允许的东西

2.2. 目标与方法

- 保护数据库不被非法访问
  - 视图和查询重写 View and query rewriting
  - 访问控制 Access control
    - 一般用户、具有资源权限的用户、DBA（数据库管理员）
  - 用户的识别和身份验证 Identification and authentication of users

2.3. 访问控制 Access controls

- 安全策略规定谁被授权做什么。
- 安全机制用于强制执行所选的安全策略。
- 在数据库管理系统级别主要有两种机制：
  - 自主访问控制 Discretionary access control

- 强制访问控制 Mandatory access control

2.4. 自由访问控制

- 基于对象（表和视图）的访问权限或特权的概念，以及赋予用户特权（并撤销特权）的机制
- 表的创建者或视图的创建者自动获得所有权限
- DMBS跟踪谁随后获得和失去权限，并确保只有具有必要权限的用户发出的请求才会被允许

2.4.1. GRANT命令

GRANT privileges ON object TO users [WITH GRANT OPTION]

- 其中可以指定以下权限：
  - **SELECT**：可以读取所有列（包括后期通过ALTER TABLE命令添加的列）
  - **INSERT(col-name)**：可以在该列中插入非空或非默认值的元组
  - **DELETE**：可以删除元组。
  - **REFERENCES(col-name)**：可以定义引用其他表的键，这些键引用此列。
- 如果用户具有某个权限的**GRANT OPTION**权限，可以给其他用户授予权限
  - 授予权限时可以决定该用户时候具有**GRANT OPTION**
- 只有所有者可以执行**CREATE**、**ALTER**和**DROP**。

2.4.2. 权限的授予和撤销 GRANT and REVOKE of Privileges

- GRANT INSERT, SELECT ON Sailors TO Horatio
  - Horatio可以查询Sailors表或向其插入元组
- GRANT DELETE ON Sailors TO Yuppy WITH GRANT OPTION
  - Yuppy可以删除元组、授予他人这个权限
- GRANT UPDATE (rating) ON Sailors TO Dustin
  - Dustin只能更新水手元组的评级字段
- GRANT SELECT ON ActiveSailors TO Guppy, Yuppy
  - 这并不允许“uppies”直接查询Sailors表，因为ActiveSailors可能只是Sailors表的一个视图或子集，而不是直接查询整个Sailors表，这可能是出于安全性或数据隔离的考虑
- 撤销权限REVOKE：当从X撤销权限时，它也会撤销所有仅从X获得该权限的用户

2.4.3. 视图上的授权/撤销操作 GRANT/REVOKE on Views

- 如果一个视图创建者失去了底层表上的SELECT权限，那么该视图将被删除
- 如果一个视图创建者失去了底层表上某个具有GRANT OPTION的权限
    - 那么他也将失去视图上的这个权限
    - 那些在视图中被授予该权限的用户也会失去该权限

2.4.4. 视图和安全 Views and Security

- 视图可以用来展示必要的信息，同时隐藏底层关系中的详细信息
- 如果视图创建者对所有底层表都有某一权限，则对该视图具有相应的权限

2.4.5. 基于角色的授权 Role-Based Authorization

- 可以将角色授予用户和其他角色
- 反映了现实组织是如何运作的
- 说明了标准通常如何追赶流行系统中所体现的“事实”标准

2.4.6. 字段级别的安全 Security to the Level of a Field

- 可以创建一个视图，只返回一个元组的一个字段，然后相应地授予对该视图的访问权限
- 这允许任意级别的控制精度，但是：
    - 虽然可以通过良好的用户界面来隐藏
    - 性能不可接受的，如果我们频繁地定义字段级别的访问权限（创建和查找视图太多。）

2.4.7. 面向互联网的安全关键问题 Internet-Oriented Security

- 用户身份验证和信任
    - 当必须在安全的位置访问数据库时，基于密码的方案通常就足够了
- 对于从外部网络进行访问，要实现信任是非常困难的
    - 加密技术（Encryption）是用来解决这些问题的技术

2.4.8. 加密 Encryption

- 为了安全传输或存储而“Mask”数据
    - Encrypt(data, encryption key) = encrypted data
    - Decrypt(encrypted data, decryption key) = original data
    - 没有解密密钥，加密的数据是无意义的乱码
- 对称加密Symmetric Encryption：

    - 加密密钥 = 解密密钥；所有授权用户都知道解密密钥（这是一种弱点）
    - DES（自1977年以来使用）具有56位密钥；AES具有128位（可选192位或256位）密钥
- 公钥加密：每个用户都有两个密钥
    - 用户的公钥：所有人都知道
    - 解密密钥：只有该用户知道
    - 在RSA方案中使用

2.5. 强制访问控制

- 基于不能由单个用户更改的全系统策略
    - 每个数据库对象都被分配一个安全类别（security class）
    - 每个主体（用户或用户程序）都被分配一个安全类别相对应的权限（clearance）
    - 基于 security class 和 clearance 的规则规定谁可以读取/写入哪个对象

2.5.1. 为什么使用强制访问控制

自由控制存在一些缺陷，例如特洛伊木马问题：

Dick创建了Horsie并给予Justin（对此一无所知）INSERT 权限。Dick修改了Justin使用的应用程序程序代码，以将一些秘密数据写入表Horsie。现在，Justin可以看到秘密信息。代码的修改超出了数据库管理系统（DBMS）的控制范围，但它可以尝试阻止使用数据库作为秘密信息的通道。

2.6. 审计追踪 Audit trail

在emp表上成功执行以下操作时进行审计：SELECT, INSERT, DELETE, UPDATE

2.7. 统计数据库的安全性 Security of Statistical Database

在很多情况下，统计数据是公开的，而详细的个人数据是保密的。但一些详细的个人数据可以从公共统计数据中推断出来

2.7.1. 单独追踪器 Individual tracker

- 假设谓词 $p = p_1\ and\ p_2$ ， $SET(p)$ 是满足 p 的元组的集合，那么
    - $SET(p) = SET(p_1\ and\ p_2) = SET(p_1) - SET(p_1\ and\ not\ p_2)$

2.7.2. 通用追踪器 General tracker

- T 是一个满足以下条件的谓词： $2b \leq |SET(T)| \leq (n-2b)$，$b < n/4$
- 假设一个元组 R 可以被唯一地限定为谓词 p ，即 $SET(p) = \{R\}$ ，那么
  - $SET(p) = SET(p\ or\ T) \cup SET(p\ or\ not\ T) - SET(T) - SET(not\ T)$
- $\cup$ 表示不消除重复元组的并集（union all）

3. 完整性约束（Integrity Constraints）

- 完整性约束（IC）描述了关系中的每一个合法实例必须满足的条件
  - 违反完整性约束的插入/删除/更新操作是被禁止的
  - 可以用来确保应用语义（例如 sid 是键），或者防止不一致（例如 $sname$ 必须是字符串，年龄必须小于200）

3.1. 完整性约束的类型

- 静态约束（Static constraint）：对数据库状态进行的约束
  - 内在约束（数据模型），如第一范式
  - 隐含约束：在数据模式中隐含的约束，通常由DDL指示
    - 如：域约束、主键约束、外键约束
  - 显式约束或一般约束 Explicit constraints or general constraints
- 动态约束（Dynamic constraint）
  - 在数据库从一个状态转移到另一个状态时进行的约束。可以与触发器结合使用

3.1.1. e.g 外键的修改检查

- If α is foreign key in r_2 which references to K1 in r_1, the following tests must be made in order to preserve the following referential integrity constraint:

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

- ***Insert.*** If a tuple t_2 is inserted into r_2, the system must ensure that there is a tuple t_1 in r_1 such that $t_1[K_1] = t_2[\alpha]$. That is

  $$t_2[\alpha] \in \prod_{K1}(r_1)$$

- ***Delete.*** If a tuple, t_1 is deleted from r_1, the system must compute the set of tuples in r_2 that reference t_1:

  $$\sigma_{\alpha = t1[K1]}(r_2)$$

  If this set is not empty, either the delete command is rejected as an error, or the tuples that reference t_1 must themselves be deleted (cascading deletions are possible).

- ***Update.*** There are two cases:
  - If a tuple t_2 is updated in relation r_2 and the update modifies values for foreign key α, then a test similar to the insert case is made. Let t_2' denote the new value of tuple t_2. The system must ensure that

    $$t_2'[\alpha] \in \prod_{K1}(r_1)$$

  - If a tuple t_1 is updated in r_1, and the update modifies values for the primary key (K_1), then a test similar to the delete case is made. The system must compute

    $$\sigma_{\alpha = t1[K1]}(r_2)$$

    using the old value of t_1 (the value before the update is applied). If this set is not empty, the update may be rejected as an error, or the update may be cascaded to the tuples in the set, or the tuples in the set may be deleted.

3.2. 完整性约束的定义

- 用过程说明约束：让应用程序负责检查完整性约束
- 用断言说明约束：使用断言规范语言进行定义，由DBMS自动检查

    - ASSERT balanceCons ON account: balance>=0；
- 在基本表定义中使用CHECK子句指示，并由DBMS自动检查
- 用触发器表示约束

3.3. 通用约束

- 当涉及到的通用完整性约束数量超过keys数量时非常有用
- 可以使用查询来表达约束
- 约束可以命名

```
CREATE TABLE Reserves
        (sname CHAR(10),
         bid INTEGER,
         day DATE,
         PRIMARY KEY (bid, day),
         CONSTRAINT noInterlakeRes
         CHECK ('Interlake'<>
                (SELECT B.bname
                FROM Boats B
                WHERE B.bid=bid)))

/* -- Constraints Over Multiple Relations
Awkward and wrong!
If Sailors is empty, the number of Boats tuples can be anything!
应该补充ifnull*/
CREATE TABLE Sailors
        (sid INTEGER,
        sname CHAR(10),
        rating INTEGER,
        age REAL,
        PRIMARY KEY (sid),
        CHECK
          ((SELECT COUNT (S.sid) FROM Sailors S)
          +(SELECT COUNT (B.bid) FROM Boats B) < 100))

-- Assertion 与表无关联
CREATE ASSERTION smallClub
CHECK
  ((SELECT COUNT (S.sid) FROM Sailors S)
  +(SELECT COUNT (B.bid) FROM Boats B) < 100)
```

3.3.1. 比较

Type	Where Declared	When activated	Guaranteed to hold?
Attribute CHECK	with attribute	on insertion or update	not if contains subquery
Tuple CHECK	relation schema	insertion or update to relation	not if contains subquery
Assertion	database schema	on change to any relation mentioned	Yes

4. 触发器 Triggers

4.1. 介绍

- 触发器：当DBMS中的指定更改发生时自动启动的程序过程。由三个部分组成：
  - 事件 Event（触发器被激活）
  - 条件 Condition（测试触发器是否应该运行）
  - 动作 Action（如果触发器运行会发生什么）
- 触发器与激活语句（数据库修改）的同步
  - 之前before、之后after、代替instead of、延迟deffered（在事务结束时）
- 更新事件可以指定特定的列或一组列
- 使用WHEN子句指定condition
- 触发器的激活次数number of activations
  - 每个修改的元组一次（FOR EACH ROW）
  - 每个激活的语句一次（FOR EACH STATEMENT）
- 引用语句选项 Options for the REFERENCING clause：
  - NEW TABLE：新插入的元组集（INSERT操作）
  - OLD TABLE：已删除或旧版本的元组集（DELETE/UPDATE操作）
  - OLD [ROW]：更新操作中的旧版本元组
  - NEW [ROW]：更新操作中的新版本元组
- 触发器的动作可以由多个SQL语句组成，用BEGIN . . . END包围

```sql
-- Row Level Trigger
CREATE TRIGGER NoLowerPrices
AFTER UPDATE OF price ON Product
REFERENCING
    OLD AS OldTuple
    NEW AS NewTuple
FOR EACH ROW
WHEN (OldTuple.price > NewTuple.price)
    UPDATE Product
    SET price = OldTuple.price
    WHERE name = NewTuple.name


-- Statement Level Triggers
CREATE TRIGGER youngSailorUpdate
AFTER INSERT ON SAILORS
REFERENCING NEW TABLE NewSailors
FOR EACH STATEMENT
  INSERT
  INTO YoungSailors(sid, name, age, rating)
  SELECT sid, name, age, rating
  FROM NewSailors N
  WHERE N.age <= 18

CREATE TRIGGER notTooManyReservations
AFTER INSERT ON Reserves                          /* Event */
REFERENCING NEW ROW NewReservation
FOR EACH ROW
WHEN (10 <= (SELECT COUNT(*) FROM Reserves
WHERE sid =NewReservation.sid))                   /* Condition */
DELETE FROM Reserves R
  WHERE R.sid= NewReservation.sid                 /* Action */
        AND day=
            (SELECT MIN(day) FROM Reserves R2 WHERE R2.sid=R.sid);

```

4.2. 触发器与一般约束 Triggers vs. General Constraints

- 触发器可能更难以理解
  - 一个SQL语句可以激活多个触发器（任意顺序）
  - 触发器可以激活其他触发器（链式激活）
- 触发器是过程性的
  - 断言对任何数据库修改做出反应，而触发器仅对特定事件做出反应

    - 触发器执行不能由DBMS优化
- 触发器的应用范围比约束更广泛
    - 监视完整性约束
    - 构建日志
    - 收集数据库统计信息等

4.3. 何时不用触发器（为什么不用了）

- 触发器早期用于以下任务：
    - 维护汇总数据（例如各部门总工资）
    - 通过记录特殊关系（称为更改或增量关系）的变化并由单独的过程将更改应用于副本来复制数据库
- 现在有更好的方法来做这些：
    - 数据库提供了内置的物化视图设施来维护汇总数据
    - 数据库为复制提供了内置支持
- 在许多情况下可以使用封装设施代替触发器
    - 定义更新字段的方法
    - 执行作为更新方法的一部分的行动，而不是通过触发器。
- 触发器意外执行的风险
    - 从备份副本加载数据时
    - 在远程站点复制更新时
    - 在执行此类操作之前可以禁用触发器
- 触发器的其他风险
    - 导致触发器触发的关键事务失败的错误
    - 级联执行

4.4. 触发器一些使用规则

- 立即执行（Immediate execution）
- 延迟执行（Deferred execution）触发器的执行被延迟到稍后的时间点
- 解耦模式（Decoupled or detached mode）：触发器的执行在一个独立的模式下进行，与其他操作解耦
- 级联触发器（Cascading trigger）：当一个触发器被执行时，它可能触发其他的触发器执行

- 控制规则的嵌套执行（Control nested execution of rules）：
- 防止非终止（Prevent nontermination）：规则引擎通常会提供机制来防止规则的无限循环执行
  - 触发图（Triggering graph）
  - 指定级联次数上限（Specify the upper limit of cascading times）

4.5. ECA的实现

- 松耦合（Loosely coupling）：ECA规则与数据库管理系统（DBMS）之间相互独立
- 紧耦合（Tightly coupling）：将ECA规则直接嵌入到数据库管理系统中
- 嵌套方法（Nested method）：在嵌套方法中，ECA规则被嵌套到事务中，并由DBMS作为事务的一部分来执行
  - 接枝方法（Grafting method）：接枝方法将规则作为数据库的扩展功能进行实现
  - 查询修改方法（Query modification method）：通过修改数据库查询语句来实现规则的执行

5. 总结

- SQL允许指定丰富的完整性约束（ICs）：基于属性、元组的CHECK约束和断言（与表无关）
- CHECK约束仅在修改其基于的表时才会被激活。断言则会在可能违反约束的任何修改时被激活
- 对于特定的IC，选择最合适的方法由DBA决定
- 触发器响应数据库中的更改，也可以用来表示约束