第9章 触发器与主动数据库系统

9.1 主动数据库系统

一、被动数据库系统 vs. 主动数据库系统

1.被动数据库系统

传统数据库系统只能按用户或应用程序（用户事务）的要求对数据库进行操作，而不能根据发生的事件或数据库的状态主动地进行操作，这样的系统称为被动数据库系统（passive database systems）。

2.主动数据库系统

理想的数据库系统应能根据发生的事件（如：用户事务对数据库进行某种操作、时间事件、外来事件等）或数据库的状态，主动地执行某些操作（称具有主动数据库功能），且这种主动数据库功能是用户/DBA事先可定义的，这样的系统称为主动数据库系统（active database systems）。

二、主动数据库系统的实现

实现主动数据库系统的**基本方法**：在数据库系统中引入规则机制。

主动数据库系统有时也称为**规则系统（rules systems）**。

主动数据库系统的**主要规则**：
条件 - 动作规则 （condition-action rule, CA rule）
事件 - 条件 - 动作规则 （event-condition-action rule, ECA rule）

1.条件 - 动作规则 （condition-action rule, CA rule）

当数据库达到某种状态时（即某个“条件”满足时），就触发DBMS执行某些“动作”。
注: CA规则作为主动数据库规则有缺陷，因此，当前大多数DBMS并不支持。

2.事件 - 条件 - 动作规则 （event-condition-action rule, ECA rule）

当某个“事件”发生时，DBMS检测某个“条件”，若满足，则执行预定义“动作”。ECA规则被称为触发器（trigger）。
注: SQL标准从SQL:1999开始增设了触发器；□当前，大多数DBMS已支持触发器，但实现功能和语法不尽相同，与SQL标准语法也不尽一致。

9.2 ECA规则/触发器

触发器类型		当“条件”满足时，“动作”的执行频度	
		EACH ROW	EACH STATEMENT
当“条件”满足时，“动作”的执行时刻/方式	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

“事件”：SQL操纵语句（INSERT, DELETE, UPDATE）
“动作”执行频度
EACH ROW：对“事件”涉及的每个“行”，“动作”均执行一次
EACH STATEMENT：对整个“事件”，“动作”仅执行一次
“动作”执行时刻/方式
BEFORE：“动作”在“事件”前执行

AFTER：“动作”在“事件”后执行
INSTEAD OF：“动作”替代“事件”而执行（即:“事件”语句不执行）——ORACLE系统中支持；但SQL:1999标准并不支持。

1.ECA规则的表示

```
     SQL:1999中触发器的定义语法：
 <触发器定义> ::= CREATE TRIGGER <触发器名>
                  { BEFORE | AFTER } <事件> ON <表名>
                  [ REFERENCING
                    OLD { ROW | TABLE } AS <过渡行/表标识符>,
                    NEW { ROW | TABLE } AS <过渡行/表标识符> ]
                  FOR EACH { ROW | STATEMENT }
                  [ WHEN <条件> ]
                  <动作>;
<事件>::= INSERT | DELETE | UPDATE [OF <属性表>]
<条件>::= <SQL谓词表达式>
<动作>::= <SQL DML语句> | BEGIN <SQL DML语句>[; <SQL DML语句>]… END
```

<过渡行/表标识符>用于引用内存中“事件”操纵语句导致数据库表/行更新时的**过渡值（transition value）**。
与**OLD**对应的是更新前旧值，与NEW对应的是更新后新值；当引用行值时，它们被称为**过渡变量（transition variable）**，当引用整个表时，它们被称为**过渡表（transition table）**。

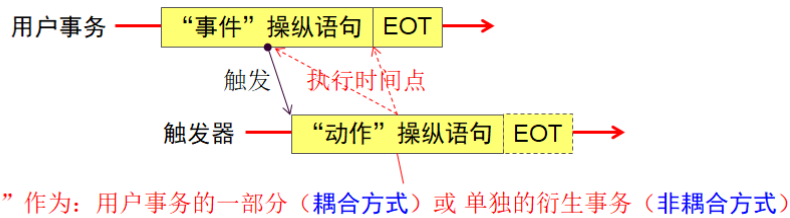
触发器（ECA规则）定义作为模式对象存放在**数据字典**中。

触发器（ECA规则）可被：**暂停（用DEACTIVE TRIGGER语句）**；**复活（用ACTIVE TRIGGER语句）**；**撤消（用DROP TRIGGER语句）**。

```
举例说明 ：
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     /* 定义了“动作”的执行时刻、方式、频度 */
       WHEN newtuple.sal < oldtuple.sal   /* 条件：当薪水值变低时 */
       UPDATE emp           /* 动作：更新emp ，以恢复薪水值 */
           SET sal = oldtuple.sal      /*  具体操作 */
           WHERE empno = newtuple.empno;    /*  具体操作的条件 */
```

2.ECA规则的执行

任何**“事件”操纵语句**的执行总是某个**用户事务**中的一部分或全部；一个用户事务中可能包含能触发多个**触发器**执行的（多个）“事件”。那么，这些触发器的**“动作”操纵语句**的执行与该用户事务的关系该如何处理呢?
——这决定了ECA规则的不同执行方式：
耦合方式（coupled mode）
非耦合/分离方式（decoupled/detached mode）



耦合方式（coupled mode）

“动作”操纵语句作为“事件”操纵语句所在的用户事务的一部分被执行。

根据执行时间的不同，可进一步分为：
立即执行（immediate execution）方式：一旦触发“事件”发生，“动作”操纵语句立即作为该用户事务的一部分而被执行。由于这种方式实现简单，大多数DBMS（如：ORACLE、IBM DB2）均采用此方式。
推迟执行（deferred execution）方式：“动作”操纵语句推迟到该用户事务的末尾（EOT）而被执行。这种方式虽理想（因为用户事务结束前，可能一些原先破坏完整性约束的现象已被消除，故有些ECA规则事实上无需执行了），但实现太复杂，很少有DBMS能实现此方式。

非耦合/分离方式（decoupled/detached mode）

“动作”操纵语句组合成一个与触发它的“事件”操纵语句所在的用户事务有因果依赖（causal dependency）关系的衍生事务而被执行。
只有在该用户事务被提交（commit）后才能提交这个衍生事务；
若该用户事务被撤消（rollback），也要撤消这个衍生事务。

连锁触发及其对策

极端情况下，用户事务中的“事件”触发了ECA规则中的“动作”（也是操纵语句），该“动作”可能会进一步触发其他ECA规则中的“动作”… ，此时，称发生了连锁触发（cascaded triggering）；相关的一组触发器被称为连锁触发器（cascading triggers） 。
对于连锁触发，一方面需正确控制ECA规则的嵌套执行，另一方面需有效防止因循环触发而导致的无休止执行（nontermination）。
通常做法是：为连锁触发次数规定一个上限，如16～64，当达到此上限时，DBMS强行撤消所有相关的ECA规则和用户事务。从这种意义上来说，ECA规则（触发器）的定义要十分小心！

3.ECA规则的实现

实现策略：将传统的数据库系统扩充改造成主动数据库系统（即引入ECA规则） 有以下几种不同的策略：
松耦合法（ loose coupling ）
在应用层和传统的DBMS之间加一个主动数据库功能模块层。该功能模块捕获用户事务中的触发“事件”，检测“条件”，若满足，则向DBMS提交“动作”。——**此法的优点是无需太多改造传统DBMS。缺点是：主动数据库功能和DBMS功能相分离，通信开销大、性能差，功能受限。**【早期方法】
紧耦合法（ close coupling ）
将主动数据库功能集成到DBMS中，因此，需彻底改造传统DBMS 。——**大型DBMS均已完成此种改造。**
嵌入法（ embedded ）
上述两种方法的折衷。由DBMS的查询处理子系统在适当时刻将ECA规则嵌入到用户事务的查询执行计划中，由DBMS执行。 ——**这种方法只能处理简单的规则。**

9.3 触发器的应用

一、触发器的内部应用

“内部应用”是为DBMS本身服务的，如：
完整性约束的维护：是触发器的主要应用！
导出数据（derived data）的实时更新。如：物化视图/实视图（materialized view）的刷新
数据库多副本一致性的维护 （略）

1.完整性约束的维护

例：定义行前触发器：实现针对选课表sc上INSERT操作的完整性约束的维护（即：要求插入实际存在的学号、课程号）：

```
    CREATE TRIGGER  sc_insertion
        BEFORE INSERT ON sc             /* sc为选课表 */
        REFERENCING
       NEW ROW AS new_row
FOR EACH ROW
WHEN ( NOT ( EXISTS (SELECT * FROM student
                     WHERE student.sno = new_row.sno)
             AND
             EXISTS (SELECT * FROM course
                     WHERE course.cno = new_row.cno)
            )
      )
ROLLBACK;
```

2.导出数据的实时更新

物化/实视图（materialized view）的刷新

例：女生成绩表fgrade（可看作是一个实视图）由下列SELECT语句导出：

```
INSERT INTO fgrade   /* 将子查询结果插入指定表中*/
SELECT sname, cno, grade
FROM student, sc
WHERE student.sno= sc.sno AND student.sex = '女';
```

上述实视图可通过触发器来维护：

例：定义语句后触发器：对sc表上DELETE操作的实视图刷新。

```
CREATE TRIGGER sc_deletion
    AFTER DELETE ON sc
    REFERENCING
           OLD TABLE AS old_table
    FOR EACH STATEMENT
    WHEN ( EXISTS  (  SELECT * FROM old_table, student
                      WHERE old_table.sno = student.sno
                      AND  student.sex='女' )         )
    BEGIN
         DELETE FROM fgrade;     /* 首先清空fgrade表 */
         INSERT INTO fgrade         /* 然后将子查询结果插入空表中 */
                SELECT sname, cno, grade     /* 子查询 = 实视图刷新 */
                FROM student, sc
                WHERE student.sno = sc.sno AND student.sex= '女'
    END;
```

二、触发器的外部应用

"外部应用"：是为用户应用服务的，如：

数据的自动归档

基于库存量的自动订购单产生，等。

这类应用实际上是将**特定应用领域**的业务规则（business rules）**抽象成ECA规则，以触发器（而非传统应用逻辑）的方式来实现业务功能，大大简化了应用的开发和维护。**

例：用行前触发器实现“客户信息自动归档”：

```
CREATE TRIGGER customer_archive
BEFORE DELETE ON customer
REFERENCING
       OLD ROW AS orow
FOR EACH ROW
BEGIN        /* 没有WHEN子句，表明无条件执行“动作”*/
       INSERT INTO customer_history
       VALUES (orow.id, orow.name)         /* 客户资料存档 */
END;
```

例：用触发器实现“基于库存量的自动订购单产生”。

假设数据库中定义了如下表：

```
inventory(item, amount)                  /* 货存清单：商品item的当前库存量amount */
min_level(item, min_amount)              /* 商品item应保持的最小库存量min_amount */
reorder_level(item, order_amount)        /* 商品item低于最小库存量时，需再订购的数量
order_amount */
purchase_orders(item, amount)            /* 商品item的订购单（订购数量amount）*/
```

定义行后触发器，实现基于库存量的自动订购单产生，如下：

```
CREATE TRIGGER item_reorder
AFTER UPDATE OF amount ON inventory
REFERENCING
       OLD ROW AS old_row,   NEW ROW AS new_row
FOR EACH ROW
WHEN new_row.amount <= (SELECT min_amount  FROM min_level
                        WHERE min_level.item = old_row.item )
              AND  NOT EXISTS (SELECT  *  FROM purchase_orders
                               WHERE purchase_orders.item = old_row.item )
BEGIN
    INSERT INTO purchase_orders
         SELECT item, order_amount  FROM reorder_level
         WHERE reorder_level.item = old_row.item
END;
```