数据库 完整性约束 规则

计算机科学 1994Vol. 21 №. 4

52-55

完整性约束规则的自动生成

TP311.13

姜跃平 董继润
(山东大学计算机系 济南 250100)

摘 要 This paper suggests a meaningful mechanism for integrity maintenance in rule-based active database systems, which supplies the end-users with a highly descriptive language to define integrity constraints independently from schema definition, automatically generates corresponding rules, storages and manages such rules and other user-defined rules in a unified mechanism. From the standpoint of the developement of a prototype rule system, this paper offers an introduction to our Constraint Definition Language and the ECA rules derived. It also, especially, presents the system architecture and the implementation technology.

关键词 Active database, ECA rule, Integrity constraint, Referential integrity

1. 前言

传统数据库一直被动地对用户(应用程序)的要求作出反应,八十年代末期提出的主动数据库突破了这一局限,能够在没有用户干预的情况下,自动地对系统内部或外部产生的事件做出响应[1],其主动性行为的基础是规则系统的有力支持。

后期推出的关系数据库产品(如 SYBASE)已经提供了规则机制,有许多基于规则的主动数据库已经或正在形成。但这些系统中提供的规则定义语言都是过程式的,要求用户对规则的条件和动作进行具体描述,规则定义成为一项很繁琐的工作,给用户充分利用规则造成困难。特别是对于大量的完整性约束保持规则来说,其触发条件和动作都有稳定一致的规律性,应该提供一种说明性的语言使用户能够简明安全地表达约束语义,而规则的细节能由系统自动生成。

说明性的完整性约束表达在 DB2 中得到一定的支持,但是必须作为数据模式定义的一部分,对约束说明的改变必然带来模式的改变,从而失去了逻辑上的独立性。用规则来存储和管理完整性约束,使之可以与模式定义相独立,而且在实现上能与其它规则保持一致。

从上述思想出发,在我们的原型规则系统 SD1 中建立了规则自动生成子系统,为用户提供了一种高级的完整性约束说明语言。系统根据用户的约束说明自动转化出相应的规则,通过执行规则保持数据的完整性。本文着重介绍了该系统的约束描述语言,所生成的规则及系统结构与实现。

2. 原型规则系统 SD1

SD1 是我们设计开发的一个原型关系数据库规则系统。它的数据查询语言是 SQL 的子集的扩展,包括表(table)的创建、删除,元组(tuple)操作,事务和视图。它的规则语言是文[3]中提出的规则语言的扩展。

SD1 支持的 ECA(Event-Condition-Action)规则是目前被广泛接受的主动规则的表达形式。其中,事件(event)是规则的触发者,它可能是数据库状态的变化,也可能来自系统的外部(如时间因素)。条件(condition)描述一种数据库状态,动作(action)是一个操作序列——当规则的触发事件发生时,规则系统将自动检查条件是否满足,满足的话就执行相关的动作。

2.1 规则语言

SD1 的规则语言基于 SQL,提供了(1)创建规则(2)删除规则(3)修改规则(4)激活(5)休眠。其主要特点是在规则体中,允许先利用扩展的 SQL 语言对条件进行预计算,条件判断语句可以引用预计算中定义的变量,从而丰富了条件的表达能力。此外,在事件集合中增加了表删除。SD1 的规则创建的语法如下:

```
CREATE RULE rulename ON tablename
WHEN Event AS
[condition calculation]
```

[IF condition THEN] action_list
END

其中，Event 是{insert，delete，update[column_list]，drop}的非空集合；condition calculation 是条件预计算；IF 语句的 condition 是 SQL 的布尔表达式；action_list 是 Event 作用在 tablename 表上且 condition 满足时要执行的扩展的 SQL 操作；rulename 是规则空间中唯一的规则名。另外，规则的条件预计算和动作部分可以引用 SD1 系统提供的四个数据转变表：inserted、deleted、new_updated、old_updated. Inserted (deleted)中存有触发规则的事务加入到数据库(从数据库中删除)的元组；new_updated(old-updated)存有被触发事务修改了的元组的现值(原值)。

2.2 系统概要结构

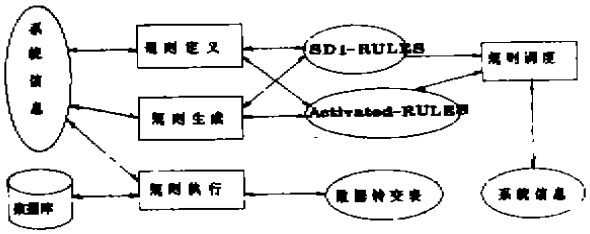

图 1

SD1 的概要结构如图 1。其规则与用户数据采用一致的存储形式——表，即图中的 SD1_RULES。图中的 ACTIVATED_RULES 为 SD1 的活动规则表，只有该表中的规则才可能被触发。系统为该表开辟专用缓冲区，并以规则的作用对象(tablename)为索引项建立索引，因而提高了规则处理的效率。系统各模块均调用数据查询语言以实现规则的存取。图中，"数据 A→模块 B"表示 B 引用 A，"模块 B→数据 A"表示 B 产生或修改 A。

3. 规则自动生成系统 RGS

RGS(Rules generation System)的目标是使用户可以简明地描述属性之间的依赖关系和对属性本身的约束。

3.1 属性值完整性与参照完整性

完整性约束是数据库中的数据必须满足的状态条件。属性值完整性限定属性取值范围、同一表中属性间的相互关系；参照完整性则用以表达不同表的元组之间的依赖关系，即一个表的外码对另一个表的主码的函数依赖。

假设 R1(X1)与 R2(X2)分别是表 r1 和 r2 的关系模式，参照完整性形如 R1[Y]⊆R2[K2]，其中 Y 和 K2 分别是 X1 和 X2 的相容子集，Y 是 R1 的外码，K2 是 R2 的主码，r1 的 r2 满足 R1[Y]⊆R2[K2] 当且仅当 $\prod_{Y}(r1)\subseteq\prod_{K2}(r2)$[1]。

3.2 完整性约束定义语言 ICDL

ICDL(integrity constraints definition language)是 RGS 与用户的接口，它使用户能直观、简洁地描述约束语义。

3.2.1 属性值完整性约束的定义语法

Creat_c_c ::= CREATE C_CONSTRAINT name ON tablename FOR Colname_list AS Expression
Expression::=Expression And Expression|NOT NULL|UNIQUE|Boolexp
Boolexp::=(Boolexp AND Boolexp)|Boolexp OR Boolexp)|NOT Boolexp|Item Comp Item
Item::=colname|constant|Item Oper Item
Colname_list::=colname1,colname2,…,colnamen|*
Comp::=<|>|=|<=|>=|!=
Oper::=+|-|*

其中 name 为约束名，用户不能定义两个同名的属性值完整性约束。从语法描述中可以看到，属性值完整性包括以下三类约束：

- NOT NULL——限定 Colname_list 中的任一属性不能为空值。
- UNIQUE——限定各元组在 Colname_list 上的投影不能有重复的取值。
- Bool 条件——不仅能同时表达多个属性的取值范围，而且能描述属性间的联系。

由于允许复合 Bool 表达式以及允许一条约束作用于多个属性，大大增强了 RGS 的约束表达能力。

3.2.2 参照完整性约束的定义语法

设有模式 R1 和 R2 上的参照完整性约束 R1[Y]⊆R2[K2]，r1 和 r2 是 R1 和 R2 的表。

(Ⅰ)在 r1 中插入元组 t(修改 t[Y])时，如果不存在 r2 中的元组 t'，使 t'[K2]=t[Y]，则本次事务不能提交。这种约束称 restricted insert(update)。

(Ⅱ)删除 r2 的元组 t'(修改 t'[K2])时，若 r1 中存在引用 t' 的元组 t，那么应该：(1)删除 t(修改 t[Y]为 t'[K2]的新值)。称 cascades delete(update)。(2)置 t[Y]为空值。称 nullifies delete(update)。(3)禁止删除 t'。称 restricted delete。

(Ⅲ)删除 r2 时，应该：(1)删除 r1。称 cascades drop。(2)置 r1[Y]为空值。称 nullifies drop。(3)禁止删除 r2。称 restricted delete。

根据上述语义，规定参照完整性约束的定义语法如下：

Creat_i_c ::= CREATE I_CONSTRAINT name FOR tablename1 (Colname_list1) REFER-

```
                RING  tablename2 ( Colname_list2 )
                [ DELETE  Act1 ] [ UPDATE  Act2 ]
                [DROP Act1]
Colnamelist::=colname1,colname2,…,colnamen | *
Act1::=cascades | restricted | nullifies
Act2::=cascades | nullifies
```

其中关于 tablename1 的 insert 和 update 约束只有 restricted 类型，因而不必显式说明。关于 tablename2 的约束默认为 cascades delete 和 restricted drop

3.3 规则自动生成

RGS 根据用户的约束定义说明，产生相应的规则定义语句。对于一个属性值完整性约束定义语句，需要为句中出现的 NOT NULL、UNIQUE 和 BOOL 条件约束各产生一条规则；对于一个参照完整性约束定义语句，需要产生 3(4)条规则；关于参照表 tablename1 的 insert_update 规则。关于被参照表 tablename2 的 delete 规则和 drop(、update)规则。限于篇幅，这里只列出其中的一部分。

对于属性集上的 NOT NULL 约束：

```
CREATE RULE c_c_name_nul ON tablename
WHEN INSERT,UPDATE(colname1,…,colnamen) AS
    declare @tmp1,@tmp2 int
    select @tmp1=count(*)from inserted
      where colname 1
    =NUL or…or colnamen=nul
    select @tmp2=count(*) from new_updated
      where colname1=NUL or…or colnamen=nul
IF @tmp1>0 or @tmp2>0 THEN
   error("NOT NULL for Colname_list violated!")
   rollback
END
```

对于被参照表上的 cascades delete 约束：

```
CREATE RULE i_c_name_del ON tablename2
WHEN DELETE AS
  declare @tmp int
  select @tmp=count(*)from deleted
    where Colname-list2 in
    (select Colname-list1 from tablename1)
IF @tmp>0 THEN
  delete tablename1
  where Colname_list1 in
  (select Colname-list2 from deleted)
END
```

对于被参照表上的 nullifies update 约束：

```
CREATE RULE i_c_name_upd ON tablename2
WHEN UPDATE(colname21,colname22,
    …,colname2n) AS
    declare @tmp int
    select @tmp=count(*) from old_updated
      where Colname_list2 in
      (select Colname_list1 from tablename1)
IF @tmp>0 THEN
   update tablename1
   set colname11=NUL,…,colnamln=NUL
     where Colname_list1 in
     (select Colname_list2 from old_updated)
END
```

对于被参照表上的 restricted drop 约束：

```
CREATE RULE i_c_name_drop ON tablename2
WHEN DROP
IF table_exist(tablename1) THEN
   error ("Cannot drop tablename2 because table-
   name1!")
   rollback
END {其中 table_exist(X)是判断表 X 是否存在的扩
     展的 SQL 调用}
```

为属性值完整性约束(参照完整性约束)生成的规则的名称分别由"c_c"("i_c")、约束名和触发类型连接而成，目的是满足 SD1对规则名的唯一性要求。

3.4 简例

设有关系模式 S_C(STUDENT,COURSE,mark)和 C(COURSE,credit,teacher)，大写属性为码。规定 S_C 的 mark 取值在0—100之间，S_C[COURSE] C[COURSE](见3.1节的参照完整性定义)。另设有 S_C 上的表 rsc，C 上的表 rc。规定对 C 的表删除约束是 nullifies drop。

在 RGS 中，用户只需作如下约束说明：

```
CREATE C_CONSTRAINT s_c1 ON rsc FOR mark
   AS(mark>=0 and mark<=100)
CREATE I_CONSTRAINT s_c_c_1
FOR rsc(course)REFERRING rc(course)
DROP nullifies
```

由 RGS 产生的部分规则为：

```
(对 S_C 的属性 mark 的约束)
CREATE RULE c_c_s_c1_bool ON rsc
WHEN insert,update(mark)AS
     declare @tmp1,@tmp2 int
     select @tmp1=count(*) from inserted
     where mark>=0 and mark<=100
     select @tmp2=count(*) from new_updated
     where mark>=0 and mark<=100
IF @tmp1>0 or @tmp2>0 THEN
   error("BOOL condition on mark violated!")
   rollback
END
(对 C 的 nullifies drop 约束)
CREATE RULE i_c_s_c_c1_drop ON rc WHEN DROP
   AS
IF table_exist(rsc)THEN update rsc
                        set course=NUL
(默认的对 C 的 cascades delete 约束)
CREATE RULE i_c_s_c_c1_del ON rc WHEN DELETE
   AS
   declare @tmp int
   select @tmp=count(*)from deleted
     where course in (select course from rsc)
IF @tmp>0 THEN
   delete rsc where course in (select course from deleted)
END
```

3.5 RGS 的实现

约束规则的生成是 RGS 的核心问题，其实现途径有两条。一是根据约束定义语言，直接产生规则的内部表示形式，也就是把生成的规则直接加入 SD1 的规则目录和活动规则目录；二是先由约束定义生成规则定义语言RDL，然后调用规则定义子系统的

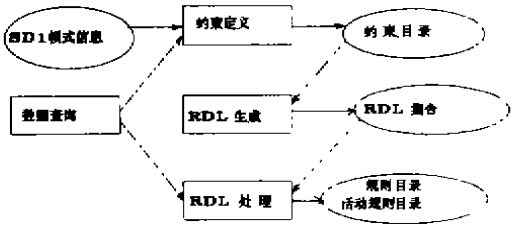

图2

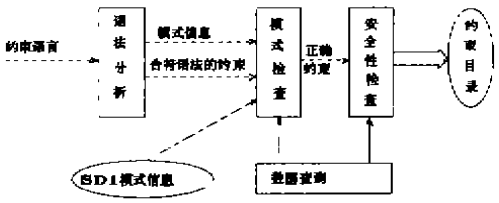

图3

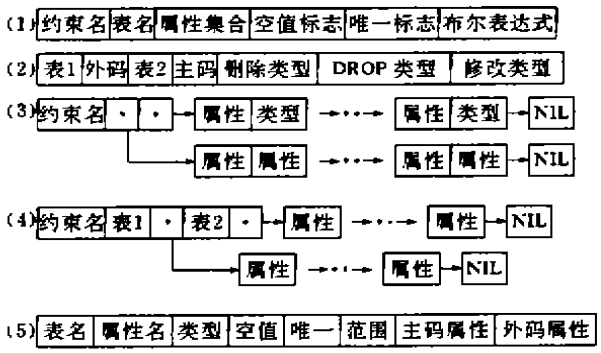

图4

RDL 处理模块产生新规则的内部形式，两者比较，途径一跳过了 RDL 生成这一中间环节，因此实现效率有所提高。但是由于它直接对规则目录进行操作，一旦 SD1的规则内部表示形式有所改变，RGS 也要随之改变，使 RGS 在整个规则系统中失去了逻辑独立性。途径二则不直接管理规则目录，只要规则语言不发生变化，RGS 生成的规则就将由规则定义系统的 RDL 处理模块转化为正确的内部形式。因为规则语言是整个规则系统的用户接口，比规则的内部表示具有更高的稳定性，所以途径二使 RGS 的逻辑独立性得到了提高。图2给出了用途径二实现的 SD1-RGS 的概要结构。

其中，约束定义模块的结构如图3。

图中，“数据 A→模块 B”表示 B 引用 A，“模块 B→数据 A”表示 B 产生或修改 A，“模块 B→模块 A”表示 A 调用 B。约束目录是约束的存储表，RGS 为属性值完整性约束和参照完整性约束各建立一个表，表的记录项内容分别见图4(1)和(2)。值得指出的是，SD1采用了变长记录存储和倒排索引技术。前者为空值属性和不定长属性的存储提供了有效支持，如〈约束表达式〉和〈属性集合〉；后者则提高了多属性布尔表达式的求值效率。图3语法分析模块产生的关于约束的模式信息的内部数据结构如图4(3)和(4)，它们分别对应于属性值完整性约束和参照完整性约束。模式检查模块利用约束模式信息和系统模式信息，对出现在约束语句中的表和属性作两方面的检查：(1)属性与表的隶属关系、属性类型和属性可比性；(2)主码与外码的合法性、一致性。图4(5)给出了 SD1的系统模式信息表的内容。像后期的许多关系数据库(如 DB2)一样，SD1允许用户在定义数据模式时给出简要的完整性说明，并把它们作为模式信息的一部分。如果用户在 RGS 中对系统模式信息里已有的约束语义又作了新的说明，那么 RGS 将改写模式信息，使新的约束语义得到 SD1的支持。

4. 结论与展望

在 SD1_RGS 中，我们为用户提供了说明性的完整性约束定义语言，允许用户动态定义完整性约束，由系统自动产生规则并用统一的机制表示、管理约束规则和其他用户规则。我们相信在基于规则的主动数据库中，这是一种富有意义的完整性保持机制。

此外在视图保持等其它方面，规则自动生成已经或正在得到深入的研究。我们认为下面几个问题是值得探讨的：

·使完整性约束具有更丰富的规范的动作语义。

·视图保持、权限保护和数据动态约束的规则自动生成。

·规则集合的自动维护——规则依赖追踪。

·完整性约束规则的版本及其维护。

参考文献

[1] V. M. Markwitz, Safe Referential Integrity In Relational Database, Proc. of 17th Conf. on VLDB, 1991

[2] O. Diaz, Generating Active Rules From High-level