

第5章 数据库的完整性



- 数据库完整性概述
- 实体完整性
- 参照完整性
- 用户定义的完整性
- 完整性约束命名子句
- 触发器
- 本章小结



■ 数据库的完整性

– 数据的正确性(corretness)

- 数据库数据是符合现实世界语义又反映了当前的实际状况

– 数据的相容性(compatibility)

- 数据库同一对象在不同关系表中的数据是相同的，一致的

– 例如：

- 学生的学号必须唯一
- 百分制的课程成绩取值范围为0-100
- 学生所选的课程必须是学校开设的课程
- 学生所在的院系必须是学校已成立的院系等



■ 数据库的安全性与完整性的联系与区别

区别 特性	概念不同	防范对象不同
完整性	<ul style="list-style-type: none">防止数据库中存在不符合语义的数据，也就是防止数据库中存在不正确的数据。	不合语义的、不正确的数据
安全性	<ul style="list-style-type: none">保护数据库防止恶意的破坏和非法的存取	非法用户和非法操作



■ 为维护数据库的完整性，RDBMS必须：

1.提供定义完整性约束的机制

- 完整性约束也称为完整性规则，是数据库中的数据必须满足的语义约束
- SQL标准使用了一系列概念来描述完整性，包括关系模型的实体完整性、参照完整性和用户定义完整性
- 这些完整性约束一般由SQL的数据定义语言语句来实现

2.提供检查完整性约束的方法

- RDBMS中检查数据是否满足完整性约束的机制称为完整性检查。
- 一般在INSERT、UPDATE、DELETE语句执行后开始检查，也可以在事务提交时检查

3.提供完整性的违约处理方法

- RDBMS若发现用户的操作违背了完整性约束，就采取一定的动作
 - 拒绝 (NO ACTION) 执行该操作
 - 级联 (CASCADE) 执行其他操作



- 早期的数据库管理系统不支持完整性检查，因为完整性检查费时费资源
- 现在商用的RDBMS都支持完整性控制
 - 即完整性定义和检查控制由RDBMS实现，不必由应用程序来完成，减轻了应用程序员的负担
- 完整性控制已成为RDBMS核心支持的功能，从而能够为所有用户和应用提供一致的数据库完整性
- 在openGauss中，表上定义的约束越多，通过应用程序维护数据的工作就越少，但更新数据所需要的时间就越多



■ 实体完整性定义

- 关系模型：CREATE TABLE中用PRIMARY KEY定义
- 单属性构成的码有两种说明方法：列级和表级
- 对多个属性构成的码只有一种说明方法：表级

■ 实体完整性检查和违约处理

- 插入或对主码列进行更新操作时，RDBMS按照实体完整性规则自动进行检查。
 - 检查主码值是否唯一，如果不唯一则拒绝插入或修改
 - 检查主码的各个属性是否为空，只要有一个为空就拒绝插入或修改



[例5.1] 创建学生表Student，将Sno属性定义为主码

```
CREATE TABLE Student
(Sno CHAR(8) PRIMARY KEY, --在列级定义主码
Sname CHAR(20) UNIQUE,
Ssex CHAR(6),
Sbirthdate Date,
Smajor VARCHAR(40) );
```

```
CREATE TABLE Student
(Sno CHAR(8),
Sname CHAR(20) UNIQUE,
Ssex CHAR(6),
Sbirthdate Date,
Smajor VARCHAR(40)
PRIMARY KEY (Sno)); --在表级定义主码
```

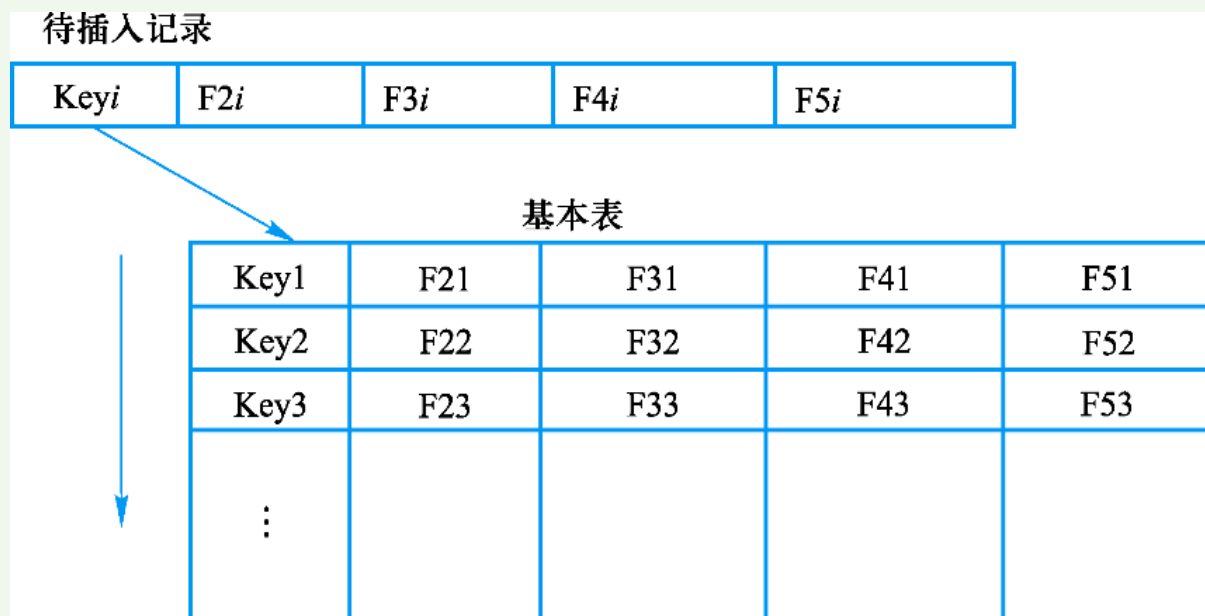
[例5.2] 创建SC表，将(Sno,Cno)属性组定义为主码

```
CREATE TABLE SC
(Sno CHAR(8),
Cno CHAR(5),
Grade SMALLINT,
Semester CHAR(5), /*开课学期*/
Teachingclass CHAR(8), /*学生选修某一门课所在的教学班*/
PRIMARY KEY (Sno,Cno) ); /*主码由两个属性构成，必须在表级定义主码*/
```

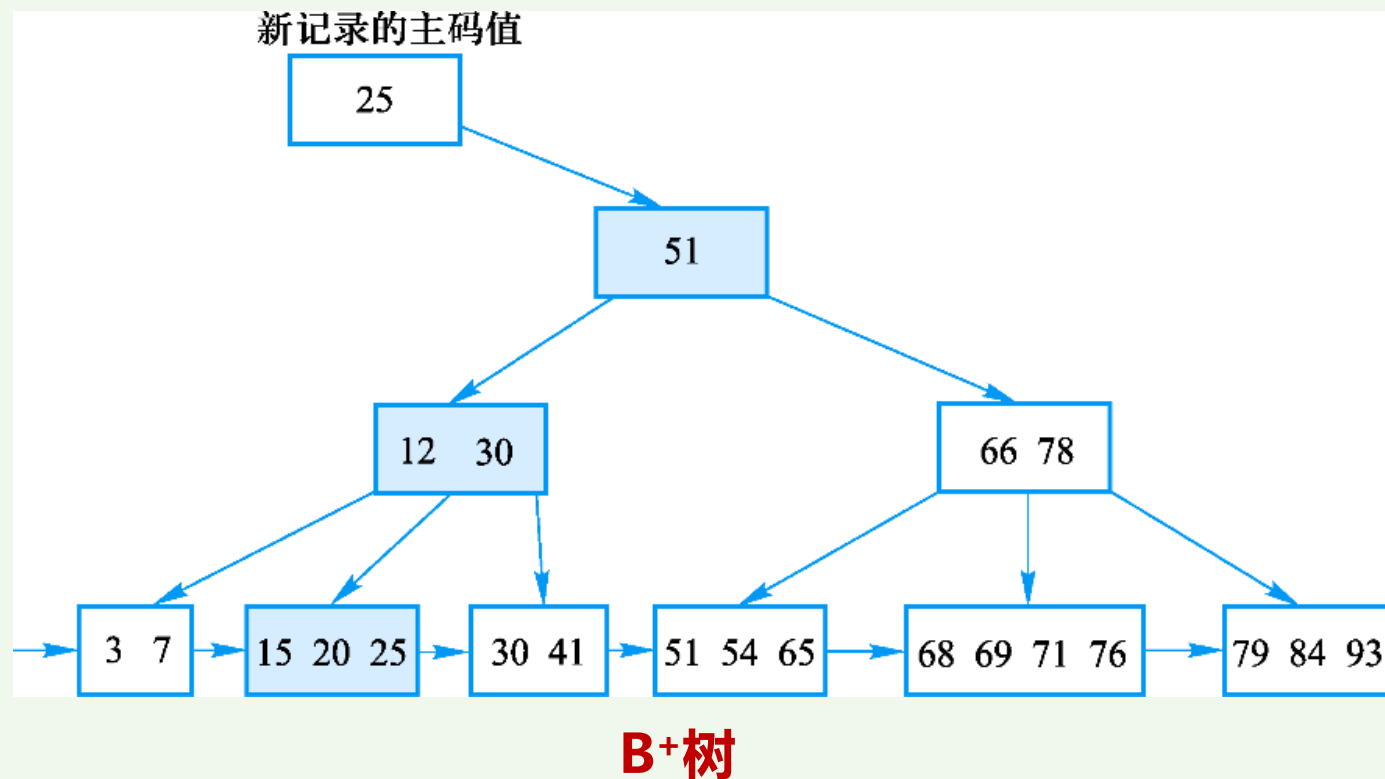


■ 全表扫描

- **DBMS**检查记录中主码值是否唯一的方法是进行**全表扫描**
 - 依次判断表中每一条记录的主码值与将插入记录上的主码值(或者修改的新主码值)是否相同
 - **缺点: 十分耗时**
 - 为避免对基本表进行全表扫描, **RDBMS**一般都在**主码上自动建立一个索引**



■ B+树索引



参考: https://blog.csdn.net/v_JULY_v/article/details/6530142



- 关系模型的参照完整性定义
 - 在CREATE TABLE中用FOREIGN KEY短语定义哪些列为外码
 - 用REFERENCES短语指明这些外码参照哪些表的主码

[例5.3] 定义SC中的参照完整性

```
CREATE TABLE SC
(Sno CHAR(8),
 Cno CHAR(5),
 Grade SMALLINT,
 Semester CHAR(5),
 Teachingclass CHAR(8),
 PRIMARY KEY (Sno, Cno), --主码定义在表级
 FOREIGN KEY (Sno) REFERENCES Student(Sno), --参照完整性定义在表级
 FOREIGN KEY (Cno) REFERENCES Course(Cno) --参照完整性定义在表级
);
```



■ 参照完整性检查和违约处理

- 一个参照完整性将两个表中的**相应元组**联系起来
- 对被参照表和参照表进行增删改操作时有可能破坏参照完整性，必须进行检查：
 - 例如，对表SC和Student有**四种可能破坏参照完整性**的情况

被参照表(例如Student)	参照表(例如SC)	违约处理
可能破坏参照完整性	插入元组	拒绝
可能破坏参照完整性	修改外码值	拒绝
删除元组	可能破坏参照完整性	拒绝/级联删除/设置为空值
更新主码值	可能破坏参照完整性	拒绝/级联更新/设置为空值

当发生违约时，系统可以采取的策略如下：

- 拒绝操作：NO ACTION(可延迟)/RESTRICT(不可延迟)，一般为默认策略
- 级联操作：CASCADE(级联删除，级联更新)
- 设置空值：SET NULL



- 当出现违约情况时，若已显式说明处理策略则系统按此方式执行

[例5.4] 显式说明参照完整性的违约处理示例

```
CREATE TABLE SC
(Sno CHAR(8),
 Cno CHAR(5),
 Grade SMALLINT,      --成绩
 Semester CHAR(5),    --选课学期
 Teachingclass CHAR(8), --学生选修某一门课所在的教学班
 PRIMARY KEY(Sno,Cno), --在表级定义实体完整性, Sno、Cno都不能取空值
 FOREIGN KEY (Sno) REFERENCES Student(Sno) --在表级定义参照完整性
 ON DELETE CASCADE      --当删除Student表中的元组时, 级联删除SC表中相应的元组
 ON UPDATE CASCADE,     --当更新Student表中的sno时, 级联更新SC表中相应的元组
 FOREIGN KEY (Cno) REFERENCES Course(Cno) --在表级定义参照完整性
 ON DELETE NO ACTION    --当删除Course表中的元组造成与SC表不一致时, 拒绝删除
 ON UPDATE CASCADE );  --当更新Course表中的Cno时, 级联更新SC表中相应的元组的Cno
```



- 参见《openGauss3.0.0开发者指南》(企业版)
 - 16.14.86节CREATE TABLE
 - 16.14.32节ALTER TABLE

另外，当被参考表中的数据发生改变时，某些操作也会在新表对应字段的数据上执行。ON DELETE子句声明当被参考表中的被参考行被删除时要执行的操作。ON UPDATE子句声明当被参考表中的被参考字段数据更新时要执行的操作。对于ON DELETE子句、ON UPDATE子句的可能动作：

- NO ACTION（缺省）：删除或更新时，创建一个表明违反外键约束的错误。若约束可推迟，且若仍存在任何引用行，那这个错误将会在检查约束的时候产生。
- RESTRICT：删除或更新时，创建一个表明违反外键约束的错误。与NO ACTION相同，只是动作不可推迟。
- CASCADE：删除新表中任何引用了被删除行的行，或更新新表中引用行的字段值为被参考字段的新值。
- SET NULL：设置引用字段为NULL。
- SET DEFAULT：设置引用字段为它们的缺省值。



- **用户定义的完整性是：**针对某一具体应用的数据必须满足的**语义要求**
- **RDBMS**提供了定义和检验用户定义完整性的机制，**不必由应用程序承担**
- 此类完整性约束分为**属性上的约束**与**元组上的约束**
- 本节主要内容：
 - **属性上的约束**
 - **元组上的约束**



■ 1. 属性上的约束定义

– **CREATE TABLE**时定义属性上的约束

- 列值非空(**NOT NULL**)
- 列值唯一(**UNIQUE**)
- 检查列值是否满足一个条件表达式(**CHECK**)

■ 2. 不允许取空值

[例5.5] 在定义**SC**表时, 说明**Sno**、**Cno**、**Grade**属性不允许取空值

```
CREATE TABLE SC
```

```
(Sno CHAR(8) NOT NULL, --Sno属性不能取空值
```

```
Cno CHAR(5) NOT NULL,      --Cno属性不能取空值
```

```
Grade SMALLINT NOT NULL,   --Grade属性不能取空值
```

```
Semester CHAR(5),
```

```
Teachingclass CHAR(8),
```

```
PRIMARY KEY (Sno, Cno) ); --Sno, Cno属性都是主属性, 不能取空值
```



■ 3.列值唯一

[例5.6] 建立学校学院表**School**，要求学院名称**SHname**列取值唯一，学院编号**SHno**列为主码

```
CREATE TABLE School
(SHno CHAR(8) PRIMARY KEY,      --SHno属性为主码
SHname VARCHAR(40) UNIQUE,    --SHname属性取唯一值
SHfounddate Date );          --学院创建日期
```

■ 4.用CHECK短语指定列值应该满足的条件

[例5.7] **Student**表的**Ssex**只允许取“男”或“女”

```
CREATE TABLE Student
(Sno CHAR(8) PRIMARY KEY,
Sname CHAR(20) NOT NULL,
Ssex CHAR(6) CHECK (Ssex IN ('男','女')), --性别属性Ssex只允许取'男'或'女'
Sbirthdate Date,
Smajor VARCHAR(40) );
```



■ 4.用CHECK短语指定列值应该满足的条件

[例5.8] SC表的Grade的值应该在0~100之间

```
CREATE TABLE SC
(Sno CHAR(8),
Cno CHAR(5),
Grade SMALLINT CHECK (Grade>=0 AND Grade <=100), --成绩取0-100之间
Semester CHAR(5),
Teachingclass CHAR(8),
PRIMARY KEY (Sno, Cno),
FOREIGN KEY (Sno) REFERENCES Student(Sno),
FOREIGN KEY (Cno) REFERENCES Course(Cno) );
```

■ 5. 属性上约束的检查和违约处理

- 插入元组或修改属性值时，RDBMS检查属性上的约束是否被满足
- 如果不满足则操作被拒绝执行



■ 1.元组上约束的定义

- 在**CREATE TABLE**语句中可以用**CHECK**短语定义元组上的约束，即元组级的限制
- 同属性值限制相比，元组级的限制可以设置不同属性之间取值的相互约束

[例5.9] 当学生的性别是男时，其名字不能以Ms.打头

```
CREATE TABLE Student
(Sno CHAR(8),
 Sname CHAR(20) NOT NULL,
 Ssex CHAR(6),
 Sbirthdate Date,
 Smajor VARCHAR(40),
 PRIMARY KEY (Sno),
 CHECK (Ssex='女' OR Sname NOT LIKE 'Ms.%')); --元组中Sname和 Ssex两个属性值之间的约束条件
```

■ 2. 元组上约束的检查和违约处理

- 插入元组或修改属性值时，**RDBMS**检查元组上的约束是否被满足
- 如果不满足则操作被拒绝执行



1.完整性约束命名子句格式

CONSTRAINT <完整性约束名><完整性约束>

- <完整性约束>包括NOT NULL、UNIQUE、PRIMARY KEY、FOREIGN KEY、CHECK短语等

完整性约束条件名的命名规范

- 具体取决于自己的规定，一般可用：约束类型+表名+字段名

- ck—check
- pk—primary key
- fk—foreign key
- uni—unique
- nn—not null

```
create table trouble(  
  city varchar2(13),  
  sample_date date,  
  noon number(4,1),  
  constraint pk_trouble primary key(city, sample_date));
```



[例5.10] 建立“学生”登记表**Student**，要求学号在10000000 ~ 29999999之间，姓名不能取空值，出生日期在1980年之后，性别只能是“男”或“女”。

```
CREATE TABLE Student
(Sno CHAR(8)
  CONSTRAINT C1 CHECK (Sno BETWEEN '10000000' AND '29999999'),
Sname CHAR(20)
  CONSTRAINT C2 NOT NULL,
Sbirthdate Date
  CONSTRAINT C3 CHECK (Sbirthdate > '1980-1-1'),
Ssex CHAR(6)
  CONSTRAINT C4 CHECK (Ssex IN ('男','女')),
Smajor VARCHAR(40),
  CONSTRAINT StudentKey PRIMARY KEY(Sno)
);
```



[例5.11] 建立教师表TEACHER, 要求每个教师的应发工资(每月)不低于3000元。应发工资是工资列Sal与扣除项Deduct之和。

```
CREATE TABLE TEACHER
```

```
(Eno CHAR(8) PRIMARY KEY, /*在列级定义主码*/
```

```
Ename VARCHAR(20),
```

```
Job CHAR(8),
```

```
Sal NUMERIC(7,2), /*每月工资*/
```

```
Deduct NUMERIC(7,2), /*每月扣除项*/
```

```
Schoolno CHAR(8), /*教师所在的学院编号*/
```

```
CONSTRAINT fk_teacher FOREIGN KEY (Schoolno) REFERENCES School(Schoolno),
```

```
CONSTRAINT C1 CHECK (Sal + Deduct >=3000) /*应发工资的约束条件C1*/
```

```
);
```



■ 2.修改表中的完整性限制

- 使用**ALTER TABLE**语句修改表中的完整性限制

[例5.12] 去掉[5.10]Student表中对出生日期的限制

```
ALTER TABLE Student DROP CONSTRAINT C3;
```

[例5.13] 修改表Student中的约束条件，要求学号改为在900000到999999之间，出生日期改为1985年之后

```
ALTER TABLE Student  
DROP CONSTRAINT C1;  
ALTER TABLE Student  
ADD CONSTRAINT C1 CHECK (Sno BETWEEN '900000' AND '999999');  
ALTER TABLE Student  
DROP CONSTRAINT C3;  
ALTER TABLE Student  
ADD CONSTRAINT C3 CHECK (Sbirthdate > '1985-1-1');
```



- --定义一个检查列约束

```
openGauss=# CREATE TABLE tpcds.warehouse_t2
            (W_WAREHOUSE_SK INTEGER PRIMARY KEY CHECK(W_WAREHOUSE_SK>0),
            W_WAREHOUSE_ID CHAR(16) NOT NULL,
            W_WAREHOUSE_NAME VARCHAR(20) CHECK(W_WAREHOUSE_NAME IS NOT NULL),
            W_WAREHOUSE_SQ_FT INTEGER ,
            W_CITY VARCHAR(60) ,
            W_COUNTY VARCHAR(30) ,
            W_STATE CHAR(2) ,
            W_ZIP CHAR(10) ,
            W_COUNTRY VARCHAR(20) ,
            W_GMT_OFFSET DECIMAL(5,2));
```

- --tpcds.warehouse_t2表增加一个检查约束

```
openGauss=# ALTER TABLE tpcds.warehouse_t2
            ADD CONSTRAINT W_CONSTR_KEY2 CHECK(W_STATE IS NOT NULL);
```



- openGauss的pg_constraint系统表存储表上的检查约束、主键和唯一约束
 - 系统表的查询需要有DBA权限
 - conname是约束名
 - contype是约束类型

```
postgres=# select conname,contype,consrc from pg_constraint;
          conname          | contype | consrc
-----+-----+-----
cardinal_number_domain_check | c       | (VALUE >= 0)
yes_or_no_check            | c       | ((VALUE)::text = ANY ((ARRAY['YES'::character varying, 'NO':
:character varying])::text[]))
(2 rows)
```

约束官网参考: <https://www.opengauss.org/zh/docs/3.0.0/docs/BriefTutorial/%E7%BA%A6%E6%9D%9F.html>



pg_constraint表结构

```
postgres=# \d pg_constraint
        Table "pg_catalog.pg_constraint"
    Column          |      Type      | Modifiers
-----+-----+-----
 conname           | name            | not null
 connamespace      | oid             | not null
 contype           | "char"         | not null
 condeferrable     | boolean         | not null
 condeferred       | boolean         | not null
 convalidated      | boolean         | not null
 conrelid          | oid             | not null
 contypid          | oid             | not null
 conindid          | oid             | not null
 confrelid         | oid             | not null
 confupdtype       | "char"         | not null
 confdeltype       | "char"         | not null
 confmatchtype     | "char"         | not null
 conislocal        | boolean         | not null
 coninhcount       | integer         | not null
 connoinherit      | boolean         | not null
 consoft           | boolean         | not null
 conopt            | boolean         | not null
 conkey            | smallint[]      |
 confkey           | smallint[]      |
 confeqop          | oid[]           |
 conpeqop          | oid[]           |
 confeqop          | oid[]           |
 conexclop         | oid[]           |
 conbin            | pg_node_tree    |
 consrc            | text            |
 conincluding      | smallint[]      |
Indexes:
    "pg_constraint_oid_index" UNIQUE, btree (oid) TABLESPACE pg_default
    "pg_constraint_conname_nsp_index" btree (conname, connamespace) TABLESPACE p
g_default
    "pg_constraint_conrelid_index" btree (conrelid) TABLESPACE pg_default
    "pg_constraint_contypid_index" btree (contypid) TABLESPACE pg_default
Replica Identity: NOTHING

postgres=#
```



- 触发器(Trigger)又叫做事件-条件-动作(Event-Condition-Action, ECA)规则, 是用户定义在关系表上的一类由事件驱动的特殊过程(Procedure)
 - 当特定的系统事件发生时, 对规则的条件进行检查。如果条件成立则执行规则中的动作, 否则不执行该动作。规则中的动作体可以很复杂, 通常是一段SQL存储过程。
 - 触发器可以实施更为复杂的检查和操作, 具有更精细和更强大的数据控制能力
 - 触发器保存在数据库服务器中
 - 任何用户对表的增、删、改操作均由服务器自动激活相应的触发器, 在RDBMS核心层进行集中的完整性控制
 - 触发器在SQL 99之后才写入SQL标准。
- 示例:
 - 假设一个仓库希望每种物品的库存保持一个最小量。在更新某种物品的库存时, 触发器会比较这种物品的当前库存和它的最小库存。如果库存数量等于或小于最小值, 就会自动生成一个新的订单



■ 本节主要内容:

- 定义触发器
- 执行触发器
- 删除触发器



- 定义触发器的语法格式：

```
CREATE TRIGGER <触发器名>  --一些产品使用CREATE [OR REPLACE] TRIGGER命令
{BEFORE | AFTER} <触发事件> ON <表名>
REFERENCING NEW|OLD ROW AS<变量>
FOR EACH {ROW | STATEMENT}
[WHEN <触发条件>]<触发动作体>
```

特别注意！
不同RDBMS产品的触发器语法各不相同

- 谁可以创建触发器？

- 表的拥有者(owner); 需要有CREATE TRIGGER系统权限

- 触发器名称：

- 触发器名可以包含模式名，也可以不包含模式名；同一模式下，触发器名必须是唯一的；触发器名和表名必须在同一模式下；一般命名约定：[trigger time]_[trigger event]_[table name]

- 表名：

- 触发器只能定义在基本表上，不能定义在视图上；当基本表的数据发生变化时，将激活定义在该表上相应触发事件的触发器



- **触发事件**: 触发事件可以是INSERT、DELETE或UPDATE, 也可以是这几个事件的组合; 还可以 UPDATE OF<触发列, ...>, 即进一步指明修改哪些列时激活触发器
- **AFTER/BEFORE** 是触发的时机: **AFTER**表示在触发事件的操作执行之后激活触发器; **BEFORE**表示在触发事件的操作执行之前激活触发器
- **触发器类型**: 行级触发器(FOR EACH ROW); 语句级触发器(FOR EACH STATEMENT)

UPDATE TEACHER SET Deptno=5;

假设表TEACHER有1000行, 对行级触发器, 执行1000次; 对语句级触发器, 执行1次

- **触发条件**: 触发器被激活时, 只有当触发条件为真时触发动作体才执行; 否则触发动作体不执行; 如果省略 WHEN触发条件, 则触发动作体在触发器激活后立即执行
- **触发动作体**: 触发动作体可以是一个匿名PL/SQL过程块, 也可以是对已创建存储过程的调用; 如果是语句级触发器, 则不能在触发动作体中使用NEW或OLD进行引用; 如果触发动作体执行失败, 激活触发器的事件就会终止执行, 触发器的目标表或触发器可能影响的其他对象不发生任何变化



[例5.18] 当对表SC的Grade属性进行修改时，若分数增加了10%，则将此次操作记录到另一个表SC_U(Sno CHAR(8)、Cno CHAR(5)、Oldgrade SMALLINT、Newgrade SMALLINT)中，其中Oldgrade是修改前的分数，Newgrade是修改后的分数

```
CREATE TRIGGER SC_T
AFTER UPDATE ON SC
REFERENCING
    OLD AS OldTuple, --触发事件为UPDATE+FOR EACH ROW, OLD和NEW代表修改前后的元组
    NEW AS NewTuple --若无FOR EACH ROW语句则OLD和NEW代表修改前后表的内容
FOR EACH ROW
WHEN (NewTuple.Grade >= 1.1 * OldTuple.Grade)
BEGIN
    INSERT INTO SC_U (Sno, Cno, OldGrade, NewGrade)
    VALUES(OldTuple.Sno, OldTuple.Cno, OldTuple.Grade, NewTuple.Grade); --应有分号
END; --教材少了一个分号
```



[例5.19] 将每次对表Student的插入操作所增加的学生个数记录到表Student InsertLog(numbers INT)中,运行触发器之前需要创建此表

```
CREATE TRIGGER after_insert_student
AFTER INSERT ON Student
REFERENCING
    NEWTABLE AS Delta
FOR EACH STATEMENT
BEGIN
    INSERT INTO StudentInsertLog(Numbers)
    SELECT COUNT(*) FROM Delta; --教材少了一个分号
END; --教材少了一个分号
```



[例5.20] 定义一个BEFORE行级触发器，为教师表Teacher定义完整性规则：教授的工资不得低于4000元，如果低于4000元，自动改为4000元

```
CREATE TRIGGER before_update_teacher
BEFORE UPDATE ON teacher
REFERENCING
    NEW AS NewTuple
FOR EACH ROW
BEGIN
    IF (NewTuple.job='教授') AND (NewTuple.sal< 4000)
    THEN NewTuple.sal :=4000; --赋值语句
    END IF;
END;
```

PostgreSQL 触发器参考: <https://blog.csdn.net/songyundong1993/article/details/131658533>



- 在编写触发器时离不开new和old关键字
 - old代表操作前的值，new代表操作后的值
 - old是只读的，而new则可以在触发器中使用set赋值
 - new的使用方法：NEW.column_name (列名)
 - 只有两种值：null和实际值

具体更新操作	OLD值	NEW值
INSERT	NULL	插入的新值
UPDATE	更新前的值	更新后的值
DELETE	删除前的值	NULL

- 参考：
 - <https://www.cnblogs.com/joyco773/p/5787088.html>



- 触发器的执行，是由触发事件激活，并由数据库服务器自动执行
- 一个数据表上可能定义了多个触发器，触发器的执行顺序依次为：
 1. 执行该表上的BEFORE触发器
 2. 激活触发器的SQL语句
 3. 执行该表上的AFTER触发器
- 对于同一个表上的多个BEFORE/AFTER触发器，遵循“谁先创建谁先执行”的原则，即按触发器创建的时间先后顺序执行。
- 有些RDBMS是按照触发器名称的字母排序顺序执行触发器



- 删除触发器的SQL语法格式:


DROP TRIGGER <触发器名> **ON** <表名>;

- 触发器必须是一个已经创建的触发器, 并且只能由具有相应权限的用户删除



■ openGauss支持创建、修改和删除触发器

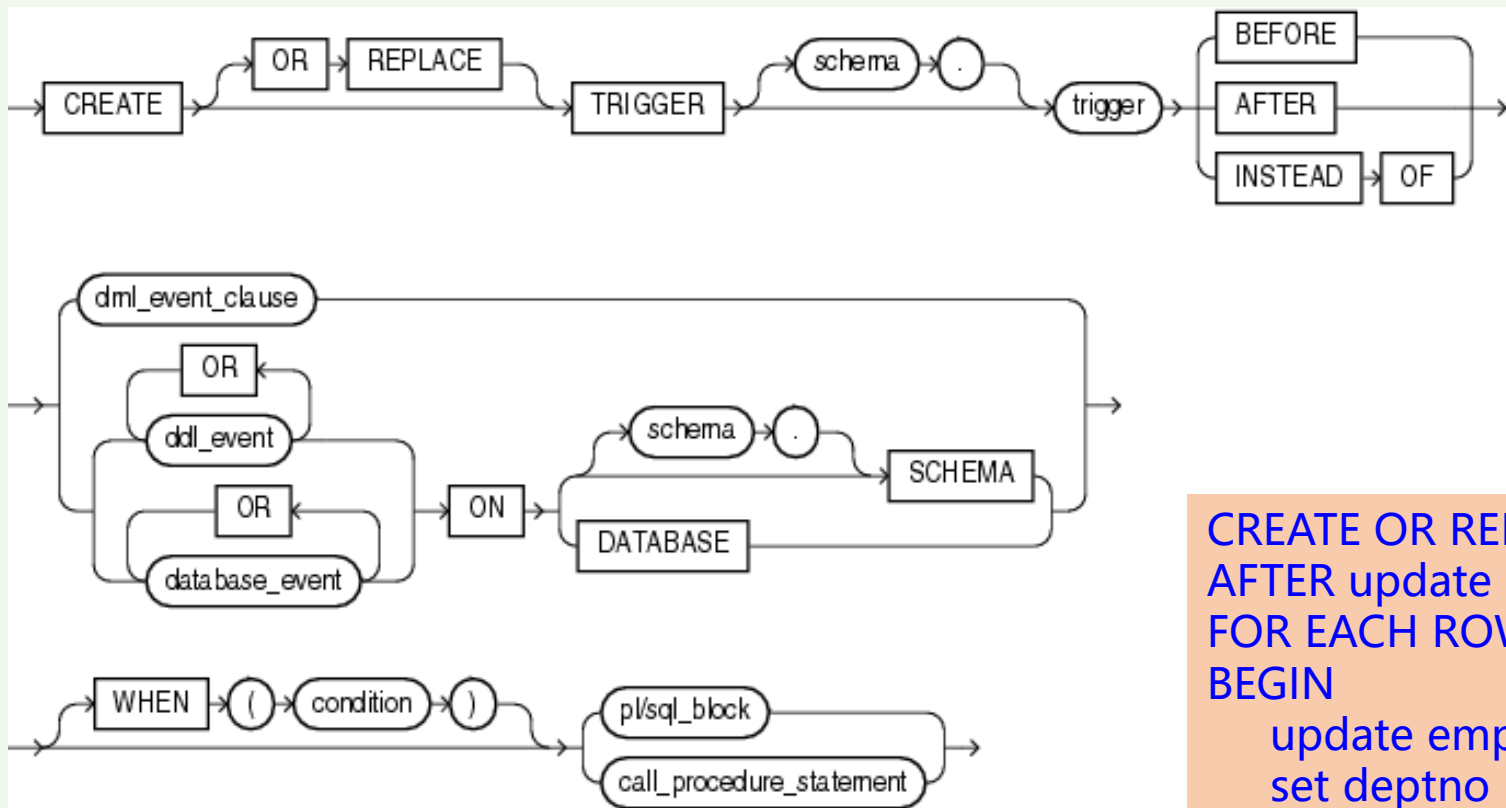
```
CREATE [ CONSTRAINT ] TRIGGER trigger_name { BEFORE | AFTER | INSTEAD OF }  
{ EVENT [ OR ... ] }  
ON table_name  
[ FROM referenced_table_name ]  
{ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }  
[ FOR [ EACH ] { ROW | STATEMENT } ]  
[ WHEN ( condition ) ]  
EXECUTE PROCEDURE function_name ( arguments );
```



- INSERT
- UPDATE [OF column_name [, ...]]
- DELETE
- TRUNCATE



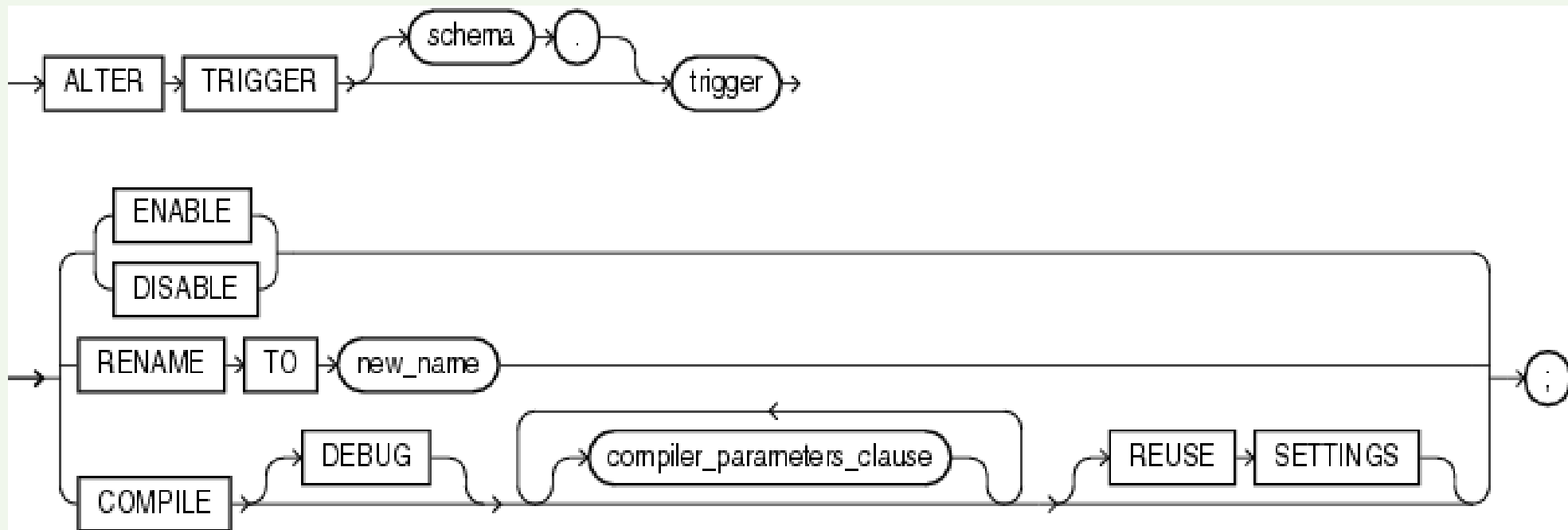
8 Oracle触发器-创建触发器



```
CREATE OR REPLACE TRIGGER dept_update
AFTER update on dept
FOR EACH ROW
BEGIN
    update emp
    set deptno =:new.deptno
    where deptno =:old.deptno;
END;
/
```

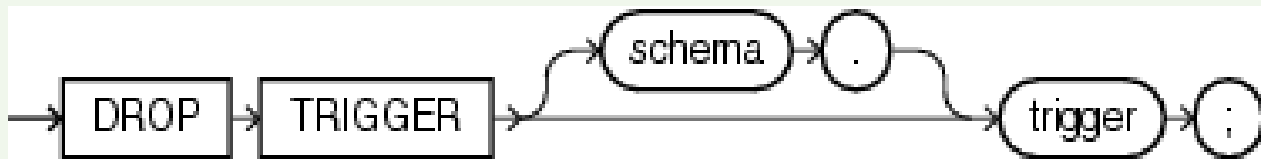
https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_7004.htm#i2153487





- **ALTER TRIGGER dept_update DISABLE;** --DISABLE: 禁用触发器
- **ALTER TRIGGER dept_update ENABLE;** --ENABLE: 启用触发器





示例:

DROP TRIGGER dept_update;

- 关系数据库管理系统完整性实现的机制
 - 完整性约束的定义机制
 - **Primary key, Foreign key, Check, Not null, Unique**
 - 完整性约束的检查方法
 - 违背完整性约束时应采取的动作
- 触发器
 - 可以实施更为复杂的完整性定义、检查和违约操作
 - 具有更精细和更强大的数据控制能力
 - 触发器规则中的动作体可以很复杂，通常是一段**过程化SQL**
 - 包括**定义、修改、执行和删除**触发器等命令



- 定义关系的主码意味着主码属性 ()
A.必须唯一 B.不能为空 C.唯一且部分主码属性不能为空 D.唯一且所有主码属性不能为空
- 关于语句`create table R(no int, sum int CHECK(sum>0))`和`CREATE TABLE R(no int, sum int, CHECK(sum>0))`, 以下说法不正确的是()
A.两条语句都是合法的
B.前者定义了属性上的约束条件, 后者定义了元组上的约束条件
C.两条语句的约束效果不一样
D.当`sum`属性改变时检查, 上述两种`CHECK`约束都要被检查
- 下列说法正确的是 ()
A.使用`ALTER TABLE ADD CONSTRAINT` 可以增加基于元组的约束
B.如果属性`A`上定义了`UNIQUE`约束, 则`A`不可以为空
C.如果属性`A`上定义了外码约束, 则`A`不可以为空
D.不能使用`ALTER TABLE ADD CONSTRAINT`增加主码约束



- 在**CREATE TABLE**时，用户定义的完整性可以通过__、__、__等子句实现
- 关系**R**的属性**A**参照引用关系**T**的属性**A**，**T**的某条元组对应的**A**属性值在**R**中出现，当要删除**T**的这条元组时，系统可以采用的策略包括 ____、____、____。
- 定义数据库完整性一般是由**SQL**的_____语句实现的

Not null, unique, check

拒绝执行，级联删除，设为空值

DDL，//注：create table，alter table， create trigger都可以实现



- 教材第五章习题1-7. 建议：第8题自行完成但不要求提交答案
- 要求：作业在布置后一周内完成并提交到课程网站

