

第5章 数据库的完整性

本章目标

- 完成本章的学习，你应该能够
 - 理解和掌握数据库完整性设计及完整性语言的使用方法
 - 掌握实体完整性、参照完整性和用户自定义完整性的定义和维护方法
 - 掌握单属性和多属性的实体完整性和参照完整性的定义、修改、删除等各种基本功能
 - 掌握列级完整性约束和表级完整性约束的定义方法
 - 掌握创建表时定义完整性和创建表后定义实体完整性两种方法，并能够设计SQL语句验证完整性约束是否起作用
 - 理解和掌握数据库触发器的分类、设计和使用，包括创建、使用、删除、激活等功能，并能设计和执行相应的SQL语句验证触发器的有效性

大纲

- **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
- 断言
- 触发器
- 本章小结

数据库完整性概述

■ 数据库的完整性

– 数据的正确性

- 是指数据是符合现实世界语义，反映了当前实际状况的

– 数据的相容性

- 是指数据库同一对象在不同关系表中的数据是符合逻辑的

例如，

- 学生的学号必须唯一
- 性别只能是男或女
- 本科学生年龄的取值范围为14~50的整数
- 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

■ 数据库的完整性VS.安全性

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，数据库管理系统必须：

提供定义完整性约束条件的机制 + 提供完整性检查的方法 + 违约处理

- 完整性约束条件也称为完整性规则，是数据库中的数据必须满足的语义约束条件
- 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
- 这些完整性一般由SQL的数据定义语言语句来实现

- DBMS中检查数据是否满足完整性约束条件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

- DBMS若发现用户的操作违背了完整性约束条件，就采取一定的动作保证数据库的完整性。
- 拒绝执行该操作
- 级联执行其他操作

- 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
- 现在商用的关系数据库管理系统都支持完整性控制
 - 即完整性定义和检查控制由关系数据库管理系统实现，不必由应用程序来完成，减轻了应用程序员的负担
- 关系数据库管理系统使得完整性控制成为其核心支持的功能，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

大纲

- 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
- 断言
- 触发器
- 本章小结

实体完整性

■ 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

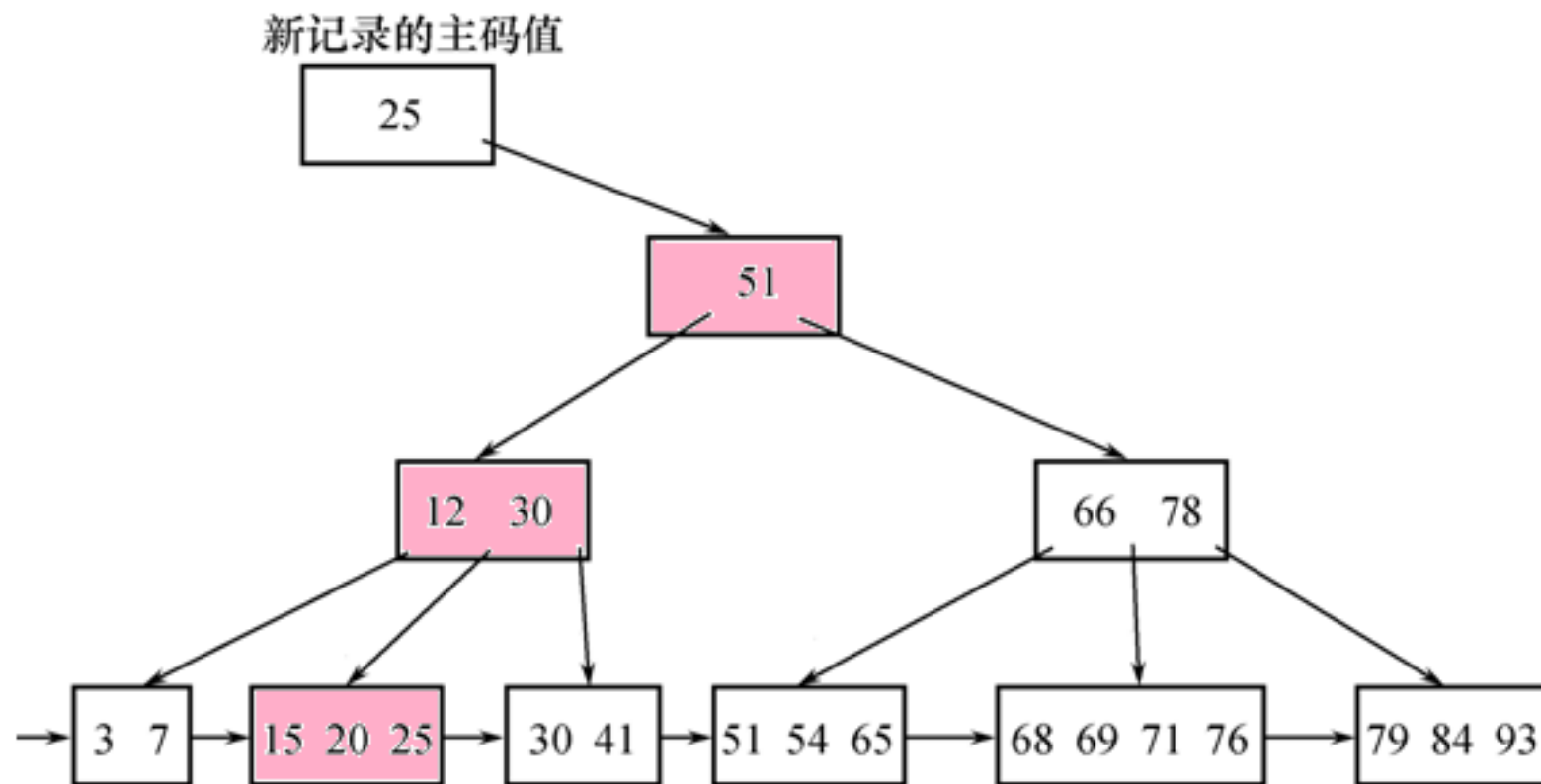
- 检查记录中主码值是否唯一的一种方法是进行**全表扫描**

- 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。



- **全表扫描缺点**

- 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

[例5.1] 将Student表中的Sno属性定义为码。

```
CREATE TABLE Student
    (Sno      CHAR(9)   PRIMARY KEY, --定义在列级
     Sname    CHAR(20)  NOT NULL,
     Ssex     CHAR(2) ,
     Sage     SMALLINT,
     Sdept    CHAR(20) ) ;
```

```
CREATE TABLE Student
    (Sno      CHAR(9) ,
     Sname    CHAR(20)  NOT NULL,
     Ssex     CHAR(2) ,
     Sage     SMALLINT,
     Sdept    CHAR(20) ,
     PRIMARY KEY(Sno) ) ; --定义在表级
```

[例5.2] 将SC表中的Sno, Cno属性组定义为码。

```
CREATE TABLE SC
    (Sno      CHAR(9) NOT NULL,
     Cno      CHAR(4) NOT NULL,
     Grade    SMALLINT,
     PRIMARY KEY (Sno, Cno) --只能定义在表级
    );
```

大纲

- 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
- 断言
- 触发器
- 本章小结

参照完整性

■ 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]:

```
CREATE TABLE SC
    (Sno      CHAR(9)   NOT NULL,
     Cno      CHAR(4)   NOT NULL,
     Grade    SMALLINT,
     PRIMARY KEY (Sno, Cno),
     FOREIGN KEY (Sno) REFERENCES Student (Sno),
     FOREIGN KEY (Cno) REFERENCES Course (Cno)
    );
```

■ 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/级连删除/设置为空值
更新主码值	可能破坏参照完整性	拒绝/级连更新/设置为空值

- 拒绝：NO ACTION(可延迟)/RESTRICT(不可延迟)
- 级连：CASCADE(级连删除，级连更新)
- 设置空值：SET NULL
- 设置默认值：SET DEFAULT

[例5.4] 显式说明参照完整性的违约处理示例。

```
CREATE TABLE SC
( Sno CHAR(9) NOT NULL,
  Cno CHAR(4) NOT NULL,
  Grade SMALLINT,
  PRIMARY KEY(Sno,Cno),
  FOREIGN KEY (Sno) REFERENCES Student(Sno)
    ON DELETE CASCADE, /*级联删除SC表中相应的元组*/
    ON UPDATE CASCADE, /*级联更新SC表中相应的元组*/
  FOREIGN KEY (Cno) REFERENCES Course(Cno)
    ON DELETE NO ACTION,
    /*当删除course 表中的元组造成了与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*/
);
```

openGauss参照完整性

- 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。

大纲

- 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
- 断言
- 触发器
- 本章小结

用户定义的完整性

- **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
- 约束条件分为：
 - 属性上的约束条件
 - 元组上的约束条件
- **属性上的约束条件**：
 - CREATE TABLE时定义属性上的约束条件
 - 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)

```
CREATE TABLE SC
(Sno CHAR(9) NOT NULL,
Cno CHAR(4) NOT NULL,
Grade SMALLINT,
PRIMARY KEY (Sno, Cno));
```

```
CREATE TABLE DEPT
(Deptno NUMERIC(2),
Dname CHAR(9) UNIQUE NOT NULL,
Location CHAR(10),
PRIMARY KEY (Deptno));
```

```
CREATE TABLE Student
(Sno CHAR(9) PRIMARY KEY,
Sname CHAR(8) NOT NULL,
Ssex CHAR(2) CHECK (Ssex IN ('男', '女')),
Sage SMALLINT,
Sdept CHAR(20) );
```

[例5.8] SC表的Grade的值应该在0和100之间。

```
CREATE TABLE SC
(Sno CHAR(9),
Cno CHAR(4),
Grade SMALLINT CHECK (Grade>=0 AND Grade<=100),
PRIMARY KEY (Sno, Cno),
FOREIGN KEY (Sno) REFERENCES Student(Sno),
FOREIGN KEY (Cno) REFERENCES Course(Cno) );
```

- 属性上的约束条件检查和违约处理

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

▪ 元组上约束条件的定义

- 在CREATE TABLE时可以用CHECK短语定义元组上的约束条件，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的取值的相互约束条件

▪ 元组上约束条件的检查和违约处理

- 插入元组或修改属性的值时，RDBMS检查元组上的约束条件是否被满足；如果不满足则操作被拒绝执行

[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(9),
Sname  CHAR(8) NOT NULL,
Ssex    CHAR(2),
Sage    SMALLINT,
Sdept   CHAR(20),
PRIMARY KEY (Sno),
CHECK (Ssex='女' OR Sname NOT LIKE 'Ms.%');
```

大纲

- 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**
- 断言
- 触发器
- 本章小结

完整性约束命名子句

■ 完整性约束命名子句语法:

CONSTRAINT <完整性约束条件名> <完整性约束条件>

- <完整性约束条件>包括: NOT NULL、UNIQUE、PRIMARY KEY短语、FOREIGN KEY短语、CHECK短语等5个约束。

完整性约束条件名的命名规范(具体取决于自己的规定):

- 约束类型+作用的表名+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90000~99999之间，姓名不能取空值，年龄小于30，性别只能是“男”或“女”。

```
CREATE TABLE Student
(Sno NUMERIC(6)
 CONSTRAINT ck_Student_Sno CHECK(Sno BETWEEN 90000 AND 99999),
 Sname CHAR(20)
 CONSTRAINT nn_Student_Sname NOT NULL,
 Sage NUMERIC(3)
 CONSTRAINT ck_Student_Sage CHECK(Sage<30 AND Sage>0),
 Ssex CHAR(2)
 CONSTRAINT ck_Student_ssex CHECK(Ssex IN ('男','女')),
 CONSTRAINT pk_Student PRIMARY KEY(Sno));
```

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

```
CREATE TABLE TEACHER
(Eno      NUMERIC(4) PRIMARY KEY,
 Ename    CHAR(10),
 Job      CHAR(8),
 Sal      NUMERIC(7,2),
 Deduct   NUMERIC(7,2),
 Deptno   NUMERIC(2),
 CONSTRAINT fk_TEACHER FOREIGN KEY(Deptno) REFERENCES DEPT(Deptno),
 CONSTRAINT ck_TEACHER CHECK(Sal+Deduct>=3000));
```

■ 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[例5.10] Student表中对性别的限制

```
ALTER TABLE Student DROP CONSTRAINT ck_Student_ssex;
```

[例5.13] 去掉修改表Student中的约束条件，要求学号改为在900000~999999之间，年龄由小于30改为小于40。

```
ALTER TABLE Student  
DROP CONSTRAINT ck_Student_Sno;
```

```
ALTER TABLE Student  
ADD CONSTRAINT ck_Student_Sno CHECK (Sno BETWEEN 900000 AND 999999),
```

```
ALTER TABLE Student  
DROP CONSTRAINT ck_Student_Sage;
```

```
ALTER TABLE Student  
ADD CONSTRAINT ck_Student_Sage CHECK(Sage<40);
```

openGauss典型约束示例

--指定约束名

```
openGauss=# CREATE TABLE tpcds.warehouse_t1
            (W_WAREHOUSE_SK INTEGER NOT NULL,
             W_WAREHOUSE_ID CHAR(16) NOT NULL,
             W_WAREHOUSE_NAME VARCHAR(20) ,
             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),
            CONSTRAINT W_CSTR_KEY1 PRIMARY KEY(W_WAREHOUSE_SK));
```

openGauss典型约束示例

--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约束查询

- 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>

```
postgres=# \d pg_constraint
```

```
Table "pg_catalog.pg_constraint"
```

Column	Type	Modifiers
conname	name	not null
connamespace	oid	not null
contype	"char"	not null
condeferable	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[]	
conpfeqop	oid[]	
conppeqop	oid[]	
conffeqop	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 pg_default
```

```
    "pg_constraint_conrelid_index" btree (conrelid) TABLESPACE pg_default
```

```
    "pg_constraint_contypid_index" btree (contypid) TABLESPACE pg_default
```

```
Replica Identity: NOTHING
```

```
postgres=#
```

pg_constraint表结构

大纲

- 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
- **断言**
- 触发器
- 本章小结

断言

- SQL中，可以使用 CREATE ASSERTION语句，通过声明性断言来指定更具一般性的约束
- 可以定义涉及多个表的或聚集操作的比较复杂的完整性约束
- 断言创建以后，任何对断言中所涉及的关系的操作都会触发RDBMS对断言的检查，任何使断言不为真值的操作都会被拒绝执行

■ 创建断言的语句格式:

CREATE ASSERTION <断言名> <CHECK子句>;

- 每个断言都被赋予一个名字, <CHECK子句>中的约束条件与WHERE子句的条件表达式类似

[例5.18] 限制数据库课程最多60名学生选修。

```
CREATE ASSERTION ASSE_SC_DB_NUM  
CHECK(60 >= (Select count(*) From Course, SC  
              Where Course.Cname='数据库' AND SC.Cno=Course.Cno));
```

[例5.19] 限制每一门课程最多60名学生选修。

```
CREATE ASSERTION ASSE_SC_CNUM1  
CHECK(60 >= ALL(Select count(*) From SC Group by Cno));
```

[例5.20] 限制每个学期每一门课程最多60名学生选修。

- 首先需要修改SC表的模式，增加一个“学期(TERM)”属性

```
ALTER TABLE SC ADD TERM DATE;
```

- 定义断言

```
CREATE ASSERTION ASSE_SC_CNUM2  
CHECK(60 >= ALL(Select count(*) From SC Group by Cno, Term));
```

- 删除断言的语句格式:

DROP ASSERTION <断言名>;

- 如果断言很复杂，则系统在检测和维护断言的开销较高，这是在使用断言时应该注意的

- 并非所有的数据库系统都支持断言

- 如，openGauss, Oracle, SQL Server都不支持

大纲

- 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
- 断言
- **触发器**
- 本章小结

触发器

- 触发器(Trieger)又叫做**事件-条件-动作规则**，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**
- **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少于最小值，就会**自动**生成一个新的订单

触发器的使用

- 定义(创建)触发器
- 激活触发器
- 删除触发器

1.定义(创建)触发器

■ 语法格式:

```
CREATE TRIGGER <触发器名> --CREATE [OR REPLACE] TRIGGER  
{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>
```

— 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: [table name]_ [trigger time]_ [trigger event]

— 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

特别注意！不同RDBMS产品的触发器语法各不相同

[例5.21] 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下面表中： SC_U(Sno, Cno, Oldgrade, Newgrade)

/*Oldgrade是修改前的分数，Newgrade是修改后的分数*/

```
CREATE TRIGGER SC_after_update --此处触发器名与课本有异
AFTER UPDATE OF Grade ON SC
REFERENCING
    OLD row AS OldTuple,
    NEW row AS NewTuple
FOR EACH ROW
WHEN (NewTuple.Grade >= 1.1*OldTuple.Grade)
    INSERT INTO SC_U(Sno,Cno,OldGrade,NewGrade)
    VALUES(OldTuple.Sno,OldTuple.Cno,OldTuple.Grade,NewTuple.Grade);
```

[例5.22] 将每次对表Student的插入操作所增加的学生个数记录到表StudentInsertLog中。

```
CREATE TRIGGER Student_after_insert
AFTER INSERT ON Student /*指明触发器激活的时间是在执行INSERT后*/
REFERENCING
    NEW TABLE AS DELTA
FOR EACH STATEMENT /*语句级触发器, 即执行完INSERT语句后下面的触发动作体才执行一次*/
    INSERT INTO StudentInsertLog(Numbers)
    SELECT COUNT(*) FROM DELTA;
```

[例5.23] 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 Teacher_before_Insert_or_Update
BEFORE INSERT OR UPDATE ON Teacher /*触发事件为插入或更新操作*/
FOR EACH ROW /*行级触发器*/
BEGIN /* 定义触发动作体，是PL/SQL过程块*/
    IF(new.job='教授')AND (new.sal<4000)
    THEN new.sal :=4000;
    ENDIF;
END;
```

2. 激活触发器

- 触发器的执行，是由触发事件激活的，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，遵循如下的执行顺序：
 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

3.删除触发器

- 语法格式:

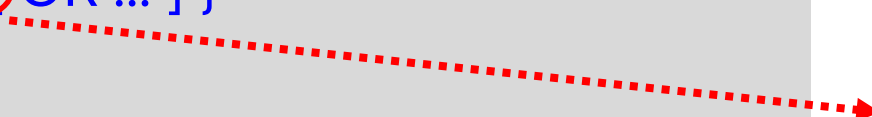
DROP TRIGGER <触发器名> ON <表名>;

- 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触发器

- openGauss支持创建、修改和删除触发器

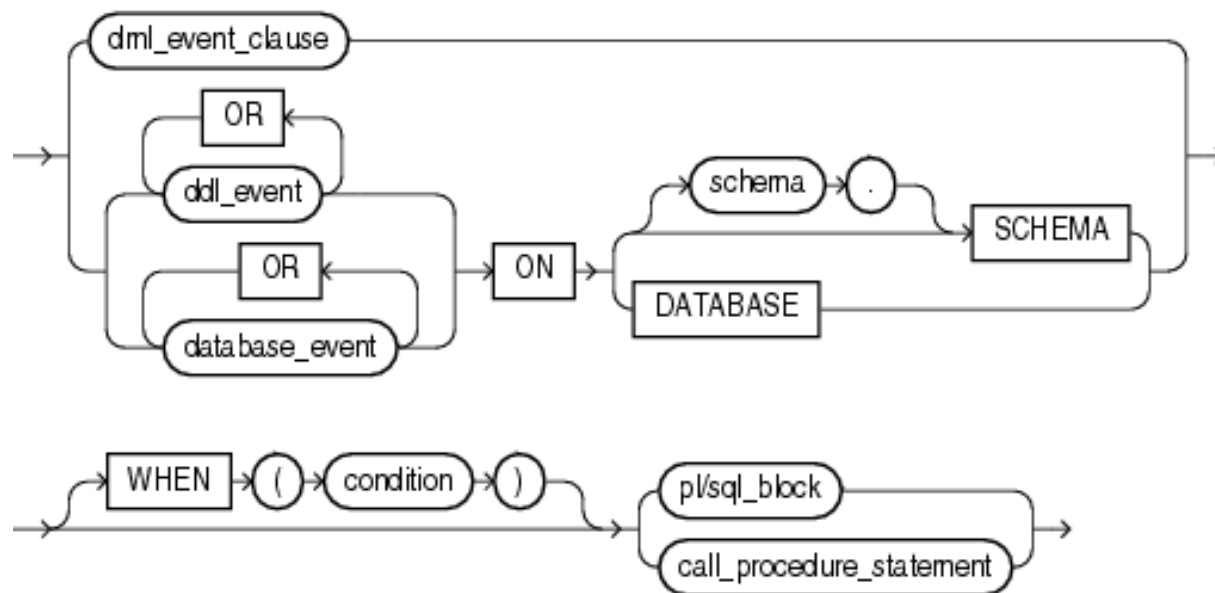
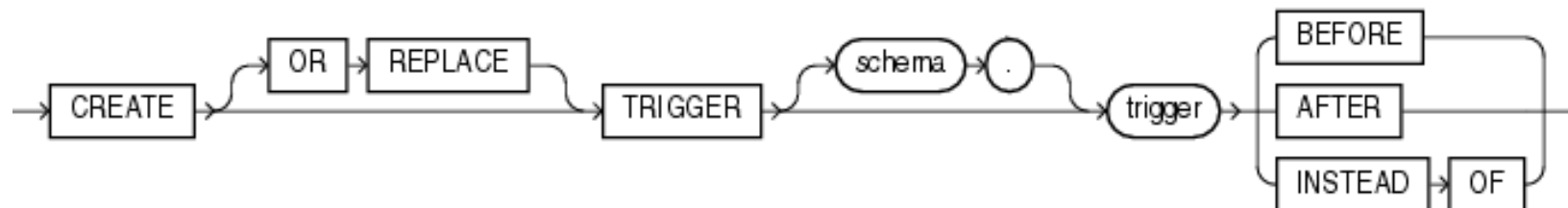
```
CREATE [ CONSTRAINT ] TRIGGER trigger_name { BEFORE |  
AFTER | INSTEAD OF } { event } OR ... ] }  
ON table_name  
[ FROM referenced_table_name ]  
{ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE |  
INITIALLY DEFERRED } }  
[ FOR [ EACH ] { ROW | STATEMENT } ]  
[ WHEN ( condition ) ] 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



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

Oracle触发器

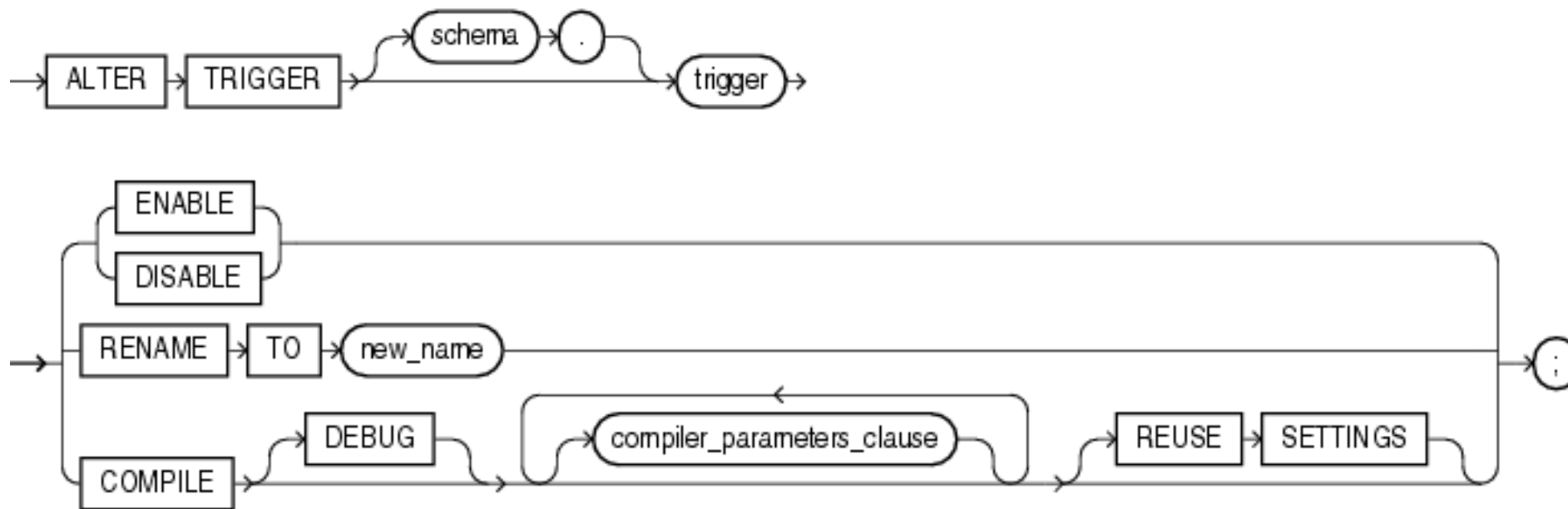
■ 创建Oracle触发器语法



```
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

■ 修改触发器语法



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

- 删除触发器语法



- 示例:

DROP TRIGGER dept_update;

本章小结

- 数据库的完整性是为了保证数据库中存储的数据是正确的。
- RDBMS完整性实现的机制：
 - 完整性约束定义机制
 - Primary key, Foreign key, Check, Not null, Unique
 - 完整性检查机制
 - 违背完整性约束条件时关系数据库管理系统应采取的动作
- 触发器用于实现未被SQL约束机制指定的某些更复杂完整性约束。
 - 定义、激活和删除触发器

课堂练习

- 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。

本章作业

- 教材第五章之习题1-7.