浙江大学
ZHEJIANG UNIVERSITY

第十一章 PostgreSQL服务器编程

陶煜波

计算机科学与技术学院

第十章 数据库安全性与完整性回顾

- 存取控制
  - 仅让用户看到或修改他们有权看到或修改的数据
  - Grant privs On R to users [With Grant Option]
  - Revoke privs On R From users [Cascade | Restrict]
  - Grant diagram
- 数据完整性 – 静态
  - Primary Key, Foreign Key, Unique, NOT NULL, Check
  - 如何定义和修改，何时做完整性检查
- 触发器 – 动态
  - When *event* occurs, check *condition*; if true, do *action*

第十章 数据库安全性与完整性回顾

- 数据完整性
  - Data-entry errors (inserts)
  - Correctness criteria (updates)
  - Enforce consistency
  - Tell system about data (store, query processing)
- 触发器
  - Move logic from applications to DBMS
  - To enforce constraints
    - Expressiveness
    - Constraint “repair” logic

第十章 数据库安全性与完整性回顾

- 触发器SQL Standard写法

  Create Trigger name

  Before | After | Instead Of events

  [referencing-variables]

  [For Each Row]

  When (condition)

  action
- 用Instead of触发器实现视图用户自定义修改

第十章 数据库安全性与完整性回顾

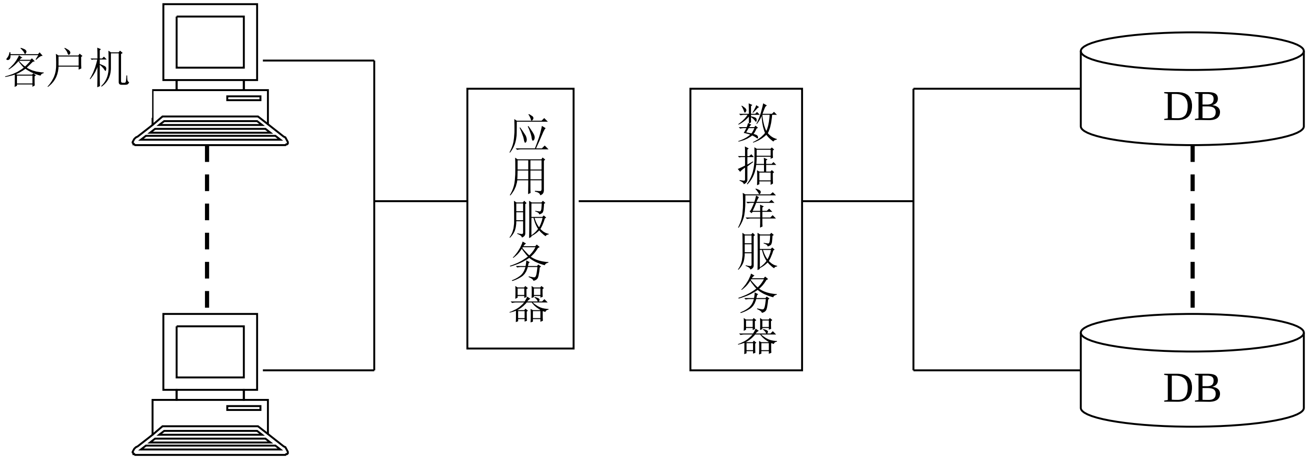
- 视图作用
  - Hide some data from some users
  - Make some query easier / more natural
  - Modularity of database access
- Materialized Views
  - 除View的作用外，与索引类似，提高查询效率
- SQL Standard中updatable views的定义
  - Select (no Distinct) on single table T
  - Attributes not in view can be ‘NULL’ or have default value
  - Subqueries must not refer to T
  - No Group by or aggregation

第十一章 PostgreSQL服务器编程

- 11.1 PostgreSQL扩展
  - 11.1.1 PostgreSQL服务器
  - 11.1.2 PL/pgSQL
  - 11.1.3 自定义类型和操作符
  - 11.1.4 定制排序方法和索引
- 11.2 函数
- 11.3 触发器

11.1.1 PostgreSQL服务器

- 三层结构的数据库管理系统的逻辑功能划分



- 例
  - Create table accounts(owner text, balance numeric);
  - Insert into accounts values (‘Bob’, 100);
  - Insert into accounts values (‘Mary’, 200);
  - Update accounts set balance – 14 where owner = ‘Bob’;
  - Update accounts set balance + 14 where owner = ‘Mary’;

11.1.1 PostgreSQL服务器

- 问题1：确保Bob账户有足够的余额?
  - Begin;
  - Select balance from accounts where owner = ‘Bob’ for update;
  - --在应用程序中核对账户余额实际上大于14美元
  - Update accounts set balance – 14 where owner = ‘Bob’;
  - Update accounts set balance + 14 where owner = ‘Mary’;
  - Commit;
- 问题2：Mary一定有账户吗?
- 问题3：新需要要求单笔转账金额小于5美元
- 使用PL/pgSQL进行完整性检查
  - 与SQL语句集成一起使用, if/then/else语句和循环功能

11.1.1 PostgreSQL服务器

```
CREATE OR REPLACE FUNCTION transfer (
            i_payer text,
            i_recipient text,
            i_amount numeric(15,2))
RETURNS text
AS
$$
DECLARE
   payer_bal numeric;
BEGIN
   SELECT balance INTO payer_bal
      FROM accounts
   WHERE owner = i_payer FOR UPDATE;
   IF NOT FOUND THEN
      RETURN ‘Payer account not found’;
   END IF;
   IF payer_bal < i_amount THEN
      RETURN ‘Not enough founds’;
   END IF;

   UPDATE accounts
      SET balance = balance + i_amount
   WHERE owner = i_recipient;
   IF NOT FOUND THEN
      RETURN ‘Recipient account not found’;
   END IF;

   UPDATE accounts
      SET balance = balance - i_amount
   WHERE owner = i_payer;

   RETURN ‘OK’;
END;
$$ LANGUAGE plpgsql;
```

思考：transfer(‘Bob’, ‘Mary’, 15)
transfer(‘Bob1’, ‘Mary’, 15)
transfer(‘Bob’, ‘Mary1’, 15)

11.1.1 PostgreSQL服务器

- 如何调用transfer函数？
  - SELECT * FROM transfer(‘Bob’, ‘Mary’, 14);
    - SELECT transfer(‘Bob’, ‘Mary’, 14);
    - msg = transfer(‘Bob’, ‘Mary’ 14);
- 为什么在服务器中进行程序设计
  - **性能** (直接在数据库内部访问数据，无需数据传输)
  - **易于维护** (直接更新数据库服务器，无需客户端更新)
  - **保证安全的简单方法** (仅授权用户访问函数，无法看到表)
- PostgreSQL服务器端程序设计
  - 使用函数让你的数据变得更为安全
  - 使用触发器审核你的数据访问
  - 使用定制化的数据类型来丰富你的数据
  - 使用定制化的操作符来分析你的数据

11.1.2 PL/pgSQL

- PL/pgSQL
  - 受PL/SQL(Oracle的存储过程语言)影响
  - 一个功能强大的SQL脚本语言，能够实现所有的功能
  - PostgreSQL并没有声称要拥有存储过程，但PL/pgSQL逐渐拥有一套丰富的控制结构，并借助触发器、运算符和索引获得了各种能力，实际上拥有了一套完整的存储过程开发系统
- PL/pgSQL优点
  - 易于上手
  - 在大多数PostgreSQL部署中为默认项
  - 为数据密集型任务进行性能优化

11.1.2 PL/pgSQL

- 函数
  - 扩展PostgreSQL最基本的构建模块
  - PL/pgSQL目标从最初作为简单的标量函数，变成了带有完整控制结构的、可以对所有PostgreSQL系统提供访问的内部构建
  - 以参数的形式输入，以输出参数或返回值的形式输出
  - 除了PL/pgSQL，PostgreSQL也支持其他语言，如Tcl、Perl、Python等

http://www.postgresql.org/docs/current/static/plpgsql.html

11.1.2 PL/pgSQL

- PL/pgSQL is a block-structured language
  - [<<*label*>>]
    [DECLARE
        declarations] -- 变量没有初始化时，值为NULL
    BEGIN
        statements   思考：为什么函数的END后还是加;？
    END [*label*];
  - 变量申明和语句都以分号;结尾
  - 除了最后的END，其他嵌套block的END后需要分号;结尾
  - 单行注释--，多行注释/* */
  - 所有变量和关键词都不区分大小写，除非用”A”
  - Block内部可以申明变量，类似于C++，同名变量将覆盖block外的变量，可以通过*label.variable*访问
    - ***name*** [CONSTANT] ***type*** [COLLATE ***collation_name***] [NOT NULL] [{DEFAULT | := | =} ***expression***];

11.1.3 自定义类型和操作符

- 例：定义类型fruit_qty来表示水果的数量，比较苹果和橘子的价值，假设一个橘子等于1.5个苹果的价值

```
CREATE TYPE FRUIT_QTY as (name text, qty int);
CREATE FUNCTION fruity_qty_larger_than (left_fruit FRUIT_QTY,
                                        right_fruit FRUIT_QTY)
    RETURENS BOOL
AS $$
BEGIN
  IF (left_fruit.name = ‘APPLE’ AND right_fruit.name = ‘ORANGE’) THEN
    RETURN left_fruit.qty > (1.5 * right_fruit.qty);
  END IF;
  IF (left_fruit.name = ‘ORANGE’ AND right_fruit.name = ‘APPLE’) THEN
    RETURN (1.5 * left_fruit.qty) > right_fruit.qty;
  END IF;
  RETURN left_fruit.qty > right_fruit.qty;
END;
$$ LANGUAGE plpgsql;
```

11.1.3 自定义类型和操作符

类型转换：::数据类型
‘Point(10 20)’::geometry

- 例：定义类型fruit_qty来表示水果的数量，比较苹果和橘子的价值，假设一个橘子等于1.5个苹果的价值
  - SELECT ‘(“APPLE”, 3)’::FRUIT_QTY
  - SELECT fruit_qty_larger_than(‘(“APPLE”, 3)’ ::FRUIT_QTY, ‘(“ORANGE”, 2)’::FRUIT_QTY);

```
CREATE OPERATOR > (
   leftarg = FRUIT_QTY,
   rightarg = FRUIT_QTY,
   procedure = fruit_qty_larger_than,
   commutator = >);
```

  - SELECT ‘(“ORANGE”, 2)’::FRUIT_QTY > ‘(“APPLE”, 3)’::FRUIT_QTY

思考：PostGIS的Geometry类型、空间函数、几何操作符的实现？

11.1.3 自定义类型和操作符

- 举例：创建扩展的几何类型数据

```
Create Type Geometry As Object (
Private Dimension SmallInt Default -1,
Private CoordinateDimension SmallInt Default 2,
Private Is3D SmallInt Default 3,
Private IsMeasured SmallInt Default 0)
Not Instantiable
Not Final
```

OGC SFA Part 2

11.1.3 自定义类型和操作符

- 举例：定义Dimension函数

Method Dimension()

Return SmallInt

Language SQL

Deterministic

Contains SQL

Returns Null On Null Input

……

- && - Returns TRUE if A's 2D bounding box intersects B's 2D bounding box

OGC SFA Part 2

11.1.4 定制排序方法和索引

- 例：仅仅通过元音的逆向顺序对单词进行排序

```
CREATE OR REPLACE FUNCTION reversed_vowels(word test)
      RETURNS text AS $$
   vowels = [c for c in word.lower() if c in ‘aeiou’]
   vowels.reverse();
   return ‘’.join(vowels);
$$ LANAGE plpythonu IMMUTABLE;
```

  - Select word, reversed_vowels(word) from words order by reversed_vowels(word)
  - 定义索引

```
CREATE INDEX reversed_vowels_index ON words (reversed_vowels(word));
```

  - 在where子句或order by中使用reversed_vowels(word)函数时，系统就会自动使用这个索引

第十一章 PostgreSQL服务器编程

- 11.1 PostgreSQL扩展
- 11.2 函数
  - 11.2.1 函数结构
  - 11.2.2 条件表达式
  - 11.2.3 循环表达式
  - 11.2.4 返回集合
  - 11.2.5 错误处理与异常
  - 11.2.6 几何函数应用举例
- 11.3 触发器

11.2.1 函数结构

- 基本元素
  - 函数名、参数、返回类型、主体和语言
- 例：笛卡尔距离计算

```
create or replace function ST_P2PDistance(x1 float, y1 float, x2 float, y2 float)
    returns float
as $$
begin
    return sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
end;
$$ language plpgsql;
```

  - 函数名：ST_P2PDistance
  - 参数(变量名 类型名)：x1 float, y1 float, x2 float, y2 float
  - 返回类型：float
  - 语言：language plpgsql
  - 函数使用：select ST_P2PDistance(103.5, 200.4, 105.6, 200.7);

11.2.1 函数结构

- SELECT *select_expressions* INTO *target* FROM … (例11.1.1)
- 在BEGIN之前可以申明变量
  - 函数中除参数外，所有变量都需在block中提前申明
  - 仅在函数执行过程的block有效

```
CREATE FUNCTION mid(str varchar, start integer)
    RETURNS varchar
AS $$
DECLARE temp varchar;
BEGIN
   temp := substring(str, start);
   return temp;
END
$$ LANGUAGE plpgsql;
```

- 函数重载： mid(str varchar, start integer, end integer)
  - 类似于C++，返回类型不同不属于函数重载

11.2.2 条件表达式

- IF/THEN/ELSE语句 (例11.1.1)

  -
    ```
    IF boolean-expression THEN
        statements
    [ ELSIF boolean-expression THEN
        statements
    [ ELSIF boolean-expression THEN
        statements ...]]
    [ ELSE
        statements ]
    END IF;
    ```

  -
    ```
    IF number = 0 THEN
        result := 'zero';
    ELSIF number > 0 THEN
        result := 'positive';
    ELSIF number < 0 THEN
        result := 'negative';
    ELSE
        result := 'NULL';
    END IF;
    ```

注意：赋值(= | :=)与相等(=)判断的差别

- IF语句
  - IF(trim(firstname) = ‘’, NULL, firstname)

11.2.2 条件表达式

- CASE语句

  -
    ```
    CASE search-expression
      WHEN expression [, expression [ ... ]] THEN
        statements
     [ WHEN expression [, expression [ ... ]] THEN
        statements ... ]
     [ ELSE
        statements ]
     END CASE;
    ```

  -
    ```
    CASE x
       WHEN 1, 2 THEN
          msg := 'one or two';
       ELSE
          msg := 'other value than one or two';
    END CASE;
    ```

11.2.3 循环表达式

- LOOP语句
  - LOOP, WHILE, FOR, CONTINUE, EXIT
  - LOOP

```
LOOP
    count := count + 1;
    CONTINUE WHEN count < 0;
    …
    EXIT WHEN count > 100;
END LOOP;
```

  - count := 0;

```
WHILE count < 100 LOOP
    counter := counter + 1;
    ….
END LOOP;
```

11.2.3 循环表达式

- LOOP语句
  - LOOP, WHILE, FOR, CONTINUE, EXIT
  - FOR语句的循环变量自动创建，无需在Declare中创建

  -
    ```
    FOR i IN 1…100 LOOP
        ….
    END LOOP;
    ```

  -
    ```
    FOR i IN REVERSE 100…1 LOOP
        ….
    END LOOP;
    ```

  -
    ```
    FOR i IN REVERSE 100…1 BY 2 LOOP
        ….
    END LOOP;
    ```

11.2.3 循环表达式

- LOOP语句
  - LOOP, WHILE, FOR, CONTINUE, EXIT
  - FOR语句的循环变量自动创建，无需在Declare中创建
  - Looping through query results
    - DECLARE movie RECORD; -- 记录类型

      FOR movie IN *SELECT * FROM movies* LOOP

      movie.id, movie.name… -- 自动创建游标

      END LOOP;
    - FOR result IN execute sql LOOP

pgr_dijkstra中的sql语句

  - Looping through arrays
    - values int[]

      FOREACH x IN ARRAY values LOOP

      …

      END LOOP;

11.2.3 循环表达式

- EXECUTE语句
  - 以字符串的形式动态构建PL/pgSQL命令，然后作为数据库的语句来调用
  - EXECUTE ‘TRUNCATE TABLE’ || table_name
- PERFROM语句
  - 执行语句，忽略执行结果
  - PERFROM cs_log(‘Done refreshing materialized views’);
    - 记录日志，忽略结果‘Done refreshing materialized views’
  - SELECT cs_log(‘Done refreshing materialized views’);
    - 记录日志，返回结果‘Done refreshing materialized views’

11.2.3 循环表达式

- 通过计数器循环 (Fibonacci序列计算)

F(n) = F(n-1) + F(n-2)

```
CREATE OR REPLACE FUNCTION fib(n integer)
   RETURNS decimal(1000, 0)
AS $$
   DECLARE counter integer := 0;
   DECLARE a decimal(1000, 0) := 0;
   DECLARE b decimal(1000, 0) := 1;
BEGIN
   IF (n < 1) THEN RETURN 0; END IF;
   LOOP
      EXIT WHEN counter = n;
      counter := counter + 1;
      SELECT b, a+b INTO a, b;
   END LOOP;
   RETURN a;
END;
$$ LANGUAGE plpgsql;
```

11.2.4 返回集合

- 返回Fibonacci序列整数集合

F(n) = F(n-1) + F(n-2)

```
CREATE OR REPLACE FUNCTION fib_seq(num integer)
   RETURNS SETOF integer AS $$
   DECLARE a int := 0;
                 b int := 1;
BEGIN
   IF (num < 1) THEN RETURN; END IF;
   RETURN NEXT a;
   LOOP
      EXIT WHEN num <= 1;
      RETURN NEXT b;
      num := num - 1;
      SELECT b, a+b INTO a, b;
   END LOOP;
END;
$$ LANGUAGE plpgsql;
```

11.2.4 返回集合

- 使用返回集合的函数
  - SELECT fib_seq(3);
  - SELECT * FROM fib_seq(3);
  - SELECT * FROM fib_seq(3) WHERE 1 = ANY(SELECT fib_seq(3));
- 返回查询结果的函数

```
CREATE OR REPLACE FUNCTION installed_languages()
  RETURNS SETOF pg_language AS $$
BEGIN
  RETURN QUERY SELECT * FROM pg_language;
END;
$$ LANGUAGE plpgsql;
```

  - SELECT * FROM installed_language();

RETURN SETOF 变量总结

RETURNS…	RECORD结构	INSIDE函数
SETOF <type>	来自于类型定义	声明ROW或者RECORD类型的行变量分配到行变量、RETURN NEXT变量
SETOF <table/view>	同表或者视图结构一样	
SETOF RECORD	动态的，在调用场景使用AS(名称类型, …)	
SETOF RECORD	使用OUT与INOUT函数参数。分配到OUT变量 RETURN NEXT;	
TABLE(…)	在TABLE关键字后面，在括号内声明，转换为在函数中使用OUT变量。从声明的TABLE(…)部分分配OUT变量 RETURN NEXT;	

11.2.5 错误处理与异常

- RAISE语句
  - RAISE [***level***] ‘***format***’ [, ***expression*** [, …]] [USING ***option*** = ***expression*** [, …]];
  - Level: DEBUG, LOG, INFO, NOTICE, WARNING, and EXCEPTION
  - Default: EXCEPTION
    - Raise an error, and abort the current transaction
  - NOTICE
    - RAISE NOTICE ‘a = %, b = %, c = %’, a, b, c
    - pgAdmin 4消息窗口查看输出的消息

http://www.postgresql.org/docs/current/static/plpgsql-errors-and-messages.html

11.2.5 错误处理与异常

- 错误处理

```
SELECT * INTO myrec FROM emp WHERE empname = myname;
IF NOT FOUND THEN
    RAISE EXCEPTION 'employee % not found', myname;
END IF;
```

- 异常

```
BEGIN
    SELECT * INTO STRICT myrec FROM emp WHERE empname = myname;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE EXCEPTION 'employee % not found', myname;
        WHEN TOO_MANY_ROWS THEN
            RAISE EXCEPTION 'employee % not unique', myname;
END;
```

11.2.5 错误处理与异常

- 异常错误获取
  - GET STACKED DIAGNOSTICS ***variable*** { = | := } ***item*** [, …]

Name	Type	Description
RETURNED_SQLSTATE	text	the SQLSTATE error code of the exception
COLUMN_NAME	text	the name of the column related to exception
CONSTRAINT_NAME	text	the name of the constraint related to exception
PG_DATATYPE_NAME	text	the name of the data type related to exception
MESSAGE_TEXT	text	the text of the exception's primary message
TABLE_NAME	text	the name of the table related to exception
SCHEMA_NAME	text	the name of the schema related to exception
PG_EXCEPTION_DETAIL	text	the text of the exception's detail message, if any
PG_EXCEPTION_HINT	text	the text of the exception's hint message, if any
PG_EXCEPTION_CONTEXT	text	line(s) of text describing the call stack

11.2.5 错误处理与异常

- 异常错误获取
  - GET STACKED DIAGNOSTICS ***variable*** { = | := } ***item*** [, …]
  - FOUND变量
    - A SELECT INTO statement sets FOUND true if a row is assigned, false if no row is returned
    - A PERFORM statement sets FOUND true if it produces (and discards) one or more rows, false if no row is produced
    - UPDATE, INSERT, and DELETE statements set FOUND true if at least one row is affected, false if no row is affected
    - A FETCH statement sets FOUND true if it returns a row, false if no row is returned
    - A MOVE statement sets FOUND true if it successfully repositions the cursor, false otherwise
    - A FOR or FOREACH statement sets FOUND true if it iterates one or more times, else false. FOUND is set this way when the loop exits; inside the execution of the loop, FOUND is not modified by the loop statement, although it might be changed by the execution of other statements within the loop body
    - RETURN QUERY and RETURN QUERY EXECUTE statements set FOUND true if the query returns at least one row, false if no row is returned

11.2.6 几何函数应用举例

- 例1 获得折线的每个顶点(非线段类型返回空集)

```
create or replace function ST_PointsFromLine(geom geometry)
   returns geometry
as $$
declare g geometry[];
begin
   if ST_GeometryType(geom) != ‘ST_LineString’ then
      return 'MULTIPOINT EMPTY'::geometry;
   end if;
   for i in 1..ST_NumPoints(geom) loop
       g = array_append(g, ST_PointN(geom, i));
   end loop;
   return ST_Collect(g);
end;
$$ language plpgsql;
```

数组Array相关函数https://www.postgresql.org/docs/current/static/functions-array.html

11.2.6 几何函数应用举例

- 数组Array相关函数
  - https://www.postgresql.org/docs/current/static/functions-array.html
- 举例

```
declare v1 geometry[2];
v1 = ARRAY[ST_StartPoint(geom), ST_EndPoint(geom)] from road where id = 123;

id_cinema integer[] = ARRAY(select id from poi where name like '%影%' order by id);
for i in 1..array_length(id_cinema,1) loop
...
end loop;

declare dist float[][];
dist = array_fill(0.0, ARRAY[10, 10]) ;
```

11.2.6 几何函数应用举例

- 例2 统计多边形的内环数

```
create or replace function ST_NInteriorRings(geom geometry)
    returns integer
as $$
declare num integer = 0;
begin
    if ST_GeometryType(geom) = ‘ST_Polygon’ then
        num = ST_NumInteriorRings(geom);
    elsif ST_Geometype(geom) = ‘ST_MultiPolygon’ then
        for i in 1..ST_NumGeometries(geom) loop
            num = num + ST_NumInteriorRings(ST_GeometryN(geom, i));
        end loop;
    end if;
    return num;
end;
$$ language plpgsql;
```

11.2.6 几何函数应用举例

- 例3 获得几何对象的包围盒(ST_Envelope)

```
create or replace function ST_AABBEnvelope(g geometry)
   returns geometry
as $$
declare minX float; minY float; maxX float; maxY float;
begin
   minX  = min(ST_X(geom))  from (select geom from ST_DumpPoints(g)) foo;
   maxX = max(ST_X(geom)) from (select geom from ST_DumpPoints(g)) foo;
   minY  = min(ST_Y(geom))  from (select geom from ST_DumpPoints(g)) foo;
   maxY = max(ST_Y(geom)) from (select geom from ST_DumpPoints(g)) foo;
   return ST_MakeEnvelope(minX, minY, maxX, maxY);
end;
$$ language plpgsql;
```

11.2.6 几何函数应用举例

- 例4 交叠空间关系判断 (ST_Overlaps)
  - Dim(I(a)) = Dim(I(b)) = Dim(I(a)∩I(b))，a∩b≠a，a∩b≠b，则a和b交叠

```
create or replace function ST_GeomOverlaps(g1 geometry, g2 geometry)
   returns boolean
as $$
declare g geometry
begin
   g = ST_Intersection(g1, g2);
   return not (ST_Dimension(g1) != ST_Dimension(g2) or
               ST_Dimension(g1) != ST_Dimension(g) or
               ST_Equals(g, g1) or
               ST_Equals(g, g2));
end;
$$ language plpgsql;
```

第十一章 PostgreSQL服务器编程

- 11.1 PostgreSQL扩展
- 11.2 函数
- 11.3 触发器
  - 11.3.1 创建触发器
  - 11.3.2 审核触发器 (应用一)
  - 11.3.3 数据保护触发器 (应用二)
  - 11.3.4 触发器效率与调试

11.3 触发器

- 触发器
  - 一种向表修改事件添加自动化函数调用的工具
  - 作为数据模型的一部分，而不是应用程序代码，确保它们不会被忘记或被省略
- PostgreSQL触发器
  - 使用CREATE FUNCTION，定义触发器函数
  - 使用CREATE TRIGGER，将触发器函数与表关联

11.3.1 创建触发器

- 创建触发器函数
  - CREATE FUNCTION mytriggerfunc() RETURNS trigger AS $$ ….
  - 通过特殊的TriggerData结构，调用环境的信息传递，在PL/pgSQL通过一组局部变量来访问
- 创建触发器
  - CREATE TRIGGER name

    { BEFORE | AFTER | INSTEAD OF} { event [OR …] }

    ON table_name

    [FOR {EACH} {ROW | STATEMENT}]

    EXECUTE PROCEDURE function_name (arguments)
  - Event是insert, update, delete或truncate
  - 参数是TG_ARGV一个文本数组(text[]),数目是TG_NARGS

11.3.1 创建触发器

变量	变量类型	变量含义
OLD, NEW	RECORD	触发器调用before与after OLD分配给delete/update NEW分配给insert/update 两者在语句/事务级触发器中均未分配
TG_NAME	name	触发器的名称 (来自触发器定义)
TG_WHEN	text	BEFORE, AFTER, 或INSTEAD OF
TG_LEVEL	text	ROW或STATEMENT
TG_OP	text	INSERT, UPDATE, DELETE, 或 TRUNCATE
TG_RELID	oid	触发器创建依赖表的OID
TG_TABLE_NAME	name	表的名称
TG_TABLE_SCHEMA	name	表模式的名称
TG_NARGS, TG_ARGS[]	int, text[]	触发器定义中的参数个数与参数数组

11.3.1 创建触发器

```
CREATE OR REPLACE FUNCTION notify_trigger()
    RETURNS TRIGGER AS $$
BEGIN
    RAISE NOTICE ‘Hi, I got % invoked for % % % on %’,
            TG_NAME, TG_LEVEL, TG_WHEN, TG_OP, TG_TABLE_NAME;
END
$$ LANGUAGE plpgsql;

CREATE TABLE notify_test(i int);

CREATE TRIGGER notify_insert_trigger
    AFTER INSERT ON notify_test
    FOR EACH ROW
    EXECUTE PROCEDURE notify_trigger();
```

11.3.1 创建触发器

- INSERT INTO notify_test VALUES (1), (2);
- 触发器需要返回一个ROW或RECORD类型的值

```
CREATE OR REPLACE FUNCTION notify_trigger()
   RETURNS TRIGGER AS $$
BEGIN
   RAISE NOTICE ‘Hi, I got % invoked for % % % on %’,
          TG_NAME, TG_LEVEL, TG_WHEN, TG_OP, TG_TABLE_NAME;
   RETURN NEW;
END
$$ LANGUAGE plpgsql;
```

- 如何创建Update和Delete触发器？

11.3.1 创建触发器

- 三合一触发器

```
CREATE TRIGGER notify_insert_trigger
    AFTER INSERT OR UPDATE OR DELETE ON notify_test
    FOR EACH ROW
    EXECUTE PROCEDURE notify_trigger();
```

- TRUNCATE notify_test;
- TRUNCATE命令不能用于单行，需要单独设置

```
CREATE TRIGGER notify_insert_trigger
    AFTER TRUNCATE ON notify_test
    FOR EACH STATEMENT
    EXECUTE PROCEDURE notify_trigger();
```

11.3.2 审核触发器

- 触发器最常见的用途之一
  - 采用前后一致且透明的方式向表中记录数据的变化
- 需要记录的内容

```
CREATE TABLE audit_log (
    username text,                 -- who did the change
    event_time_utc timestamp,  -- when the event was recorded
    table_name text,              -- contains schema-qualified table name
    operation text,                -- INSERT, UPDATE, DELETE or TRUNCATE
    before_value json,           -- the OLD tuple value
    after_value json,             -- the NEW tuple value
);
```

  - username使用SESSION_USER变量
  - table_name使用schema.table存储
  - operation使用TG_OP

11.3.2 审核触发器

```
CREATE OR REPLACE FUNCTION audit_trigger()
   RETURNS TRIGGER AS $$
DECLARE old_row   json := NULL;
              new_row json := NULL;
BEGIN
   IF TG_OP IN (‘UPDATE’, ‘DELETE’) THEN
      old_row = row_to_json(OLD);
   END IF;
   IF TG_OP IN (‘INSERT’, ‘UPDATE’) THEN
      new_row = row_to_json(NEW);
   END IF;
   INSERT INTO audit_log VALUES(session_user, current_timestamp AT
TIME ZONE ‘UTC’, TG_TABLE_SCHEMA || ‘.’ || TG_TABLE_NAME,
TG_OP, old_row, new_row);
   RETURN NEW;
END; $$ LANGUAGE plpgsql;
```

11.3.2 审核触发器

- NEW与OLD对于DELETE触发器与INSERT触发器并不为NULL
- 创建触发器

```
CREATE TRIGGER audit_log_trigger
   AFTER INSERT OR UPDATE OR DELETE
   ON audit_log
   FOR EACH ROW
   EXECUTE PROCEDURE audit_trigger();
```

11.3.3 数据保护触发器

- 需求：数据只能在一些表中被添加和修改，但不能被删除
  - 从所有用户处撤销对这些表的DELETE操作 (记得同时要从PUBLIC处撤销DELETE)
  - 借助触发器实现

11.3.3 数据保护触发器

```
CREATE OR REPLACE FUNCTION cancel_op()
    RETURNS TRIGGER AS $$
BEGIN
    IF TG_WHEN = ‘AFTER’ THEN
        RAISE EXCEPTION ‘You are not allowed to % rows in %.%’,
                                     TG_OP, TG_TABLE_SCHEMA, TG_TABLE_NAME
    END IF;
    RAISE NOTICE ‘% on rows in %.% won’t happen’,
                                    TG_OP, TG_TABLE_SCHEMA, TG_TABLE_NAME
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

11.3.3 数据保护触发器

- BEFORE与AFTER触发器
  - BEFORE触发器，输出消息，返回NULL
  - AFTER触发器，引发错误，当前事务回滚

```
CREATE TRIGGER disallow_delete
    AFTER DELETE ON audit_log
    FOR EACH STATEMENT
    EXECUTE PROCEDURE cancel_op();
```

- TRUNCATE

```
CREATE TRIGGER disallow_truncate
    AFTER TRUNCATE ON audit_log
    FOR EACH STATEMENT
    EXECUTE PROCEDURE cancel_op();
```

思考：如何为TRUNCATE创建BEFORE触发器？

11.3.3 数据保护触发器

- 另一种常用的审核方式
  - 同一行的特定字段中，如同记录数据一样记录操作信息
- 需求：在INSERT和UPDATE事务发生时，在字段last_changed_at和last_changed_by中记录操作时间与当前用户
  - 在行级别的BEFORE触发器中，可以通过变更NEW记录，修改实际需要被写入数据库的内容

```
CREATE TABLE modify_test (
    id serial PRIMARY KEY,
    data text,
    created_by text default SESSION_USER,
    created_at timestamp default CURRENT_TIMESTAMP,
    last_changed_by text default SESSION_USER,
    last_changed_at timestamp default CURRENT_TIMESTAMP);
```

11.3.3 数据保护触发器

```
CREATE OR REPLACE FUNCTION changestamp()
   RETURNS TRIGGER AS $$
BEGIN
      NEW.last_changed_by = SESSION_USER;
      NEW.last_changed_at  = CURRENT_TIMESTAMP;
      RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER changestamp
   BEFORE UPDATE ON modify_test FOR EACH ROW
   EXECUTE PROCEDURE changestamp();

INSERT INTO modify_test(data) VALUES(‘something’);
UPDATE modify_test SET data = ‘something else’ WHERE id = 1;
```

11.3.3 数据保护触发器

- 需求：如何保证created_by与created_at字段？

```
CREATE OR REPLACE FUNCTION usagestamp()
  RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = ‘INSERT’ THEN
      NEW.created_by = SESSION_USER;
      NEW.created_at  = CURRENT_TIMESTAMP;
    ELSE
      NEW.created_by = OLD.created_by;
      NEW.created_at  = OLD.created_at;
    END IF;
    NEW.last_changed_by = SESSION_USER;
    NEW.last_changed_at  = CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

11.3.3 数据保护触发器

- 需求：如何保证created_by与created_at字段?

```
CREATE TRIGGER usagestamp
   BEFORE INSERT OR UPDATE ON modify_test FOR EACH ROW
   EXECUTE PROCEDURE usagestamp();

DROP TRIGGER changestamp on modify_test;
UPDATE modify_test SET created_by = ‘notpostgres’, created_at = ‘2001-01-01’;
SELECT * FROM modify_test;
```

11.3.4 触发器效率与调试

- 大批量的数据加载或对表中大部分内容进行更新
  - 触发器会影响效率，仅在真正需要的时候调用

```
CREATE TRIGGER name
    { BEFORE | AFTER | INSTEAD OF} { event [OR …] }
    [OF column_name [OR column_name …]] ON table_name
    [FOR {EACH} {ROW | STATEMENT}]
    [WHEN (condition) ]
    EXECUTE PROCEDURE function_name (arguments)
```

SQL标准
Create Trigger name
Before | After | Instead Of events
[referencing-variables]
[For Each Row]
When (condition)
action

11.3.4 触发器效率与调试

- 需求：周五下午禁止更新

```
CREATE OR REPLACE FUNCTION cancel_with_message()
   RETURNS TRIGGER AS $$
BEGIN
      RAISE EXCEPTION ‘%’, TG_ARGV[0];
      RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

思考：cancel_with_message()输出什么？

```
CREATE TRIGGER no_updates_on_friday_afternoon
   BEFORE INSERT OR UPDATE OR DELETE OR TRUNCATE ON new_t
   FOR EACH STATEMENT
   WHEN (CURRENT_TIMESTAMP > ‘12:00’ AND extract (DOW from
CURRENT_TIMESTAMP) = 5)
   EXECUTE PROCEDURE cancel_with_message(‘Sorry, we have a “No
task change on Friday afternoon” policy!’);
```

11.3.4 触发器效率与调试

- 需求：仅当列有变化时，才执行触发器

```
WHEN (
    NEW.column1 IS DISTINCT FROM OLD.column1
    OR
    NEW.column2 IS DISTINCT FROM OLD.column2)
```

- 需求：修改主键时报错

```
CREATE TRIGGER disallow_pk_change
    AFTER UPDATE OF id ON table_with_pk_id
    FOR EACH ROW
    EXECUTE PROCEDURE cancel_op();
```

- 多版本并发控制规则(Multiversion Concurrency Control)
  - 触发器引起的数据修改，是否会引发新的触发器？

11.3.4 触发器效率与调试

- 创建函数使用CREATE OR REPLACE FUNCTION
  - 自动更新函数，避免手动删除drop function xxx;
- 使用RAISE NOTICE进行“手动”调试
  - NOTICE (信息输出，不终止执行，pgAdmin 4消息查看)
  - EXCEPTION (异常，终止执行，pgAdmin 4查看错误)
  - LOG (日志文件)
- 程序bug检测：ASSERT *condition* [, *message*];
  - 常规错误报告：RAISE语句
- 可视化调试
  - pgFoundry (PostgreSQL 8.2之后版本)
  - 单步执行，设置断点，……

PostgreSQL触发器总结

- 适用场景
  - 审计、日志、执行复杂约束、复制等
- 应用程序逻辑仅可能避免使用触发器
- INSERT, UPDATE, DELETE, TRUNCATE
  - FOR EACH ROW, OLD和NEW
  - FOR EACH STATEMENT, OLD和NEW未分配
  - TRUNCATE只能用于FOR EACH STATEMENT
  - TRUNCATE不会触发DELETE，抛出异常终止事务
  - 表执行BEFORE或AFTER操作
    - 对于BEFORE，INSERT, UPDATE, DELETE返回NULL处理
    - 对于AFTER， INSERT, UPDATE, DELETE抛出异常终止事务
  - 视图执行INSTEAD OF操作

第十一章 PostgreSQL服务器编程

- 11.1 PostgreSQL扩展
  - 三层结构，PL/pgSQL
  - 自定义类型和操作符，定制排序方法和索引
- 11.2 函数
  - 函数结构，条件表达式，循环表达式
  - 返回集合，错误处理与异常
  - 几何函数应用举例
- 11.3 触发器
  - 创建触发器
  - 应用：审核触发器，数据保护触发器
  - 触发器效率与调试