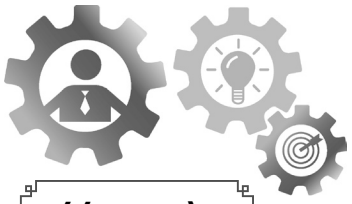

# 第 5 章

# 事 务 处 理

事务是数据库恢复和并发控制的基本单位，具有原子性、一致性、隔离性和持续性特性。KingbaseES 数据库提供了强大的事务模型来支持事务处理，允许多个用户同时处理数据库，并确保每个用户看到一致的数据版本，同时，所有更改都以正确的顺序执行。本章将主要介绍以下内容。

- 事务处理概述。
- 事务处理语句。
- 自治事务。

## 5.1 事务处理概述

KingbaseES 数据库中的事务是一个最小的执行单元。一个事务可以是一个 SQL 语句，也可以是多个 SQL 语句。一个事务中的语句要么全部执行，要么全不执行。如果所有操作完成，事务将提交，其修改将作用于所有其他数据库进程。如果一个操作失败，事务将回滚，该事务所有操作的影响都将取消。

## 5.2 事务处理语句

KingbaseES PL/SQL 提供了 COMMIT、ROLLBACK、SET TRANSACTION 以及 LOCK TABLE 等语句。

### 5.2.1 COMMIT 语句

COMMIT 语句结束当前事务，保存自上次 COMMIT 或 ROLLBACK 以来所有完成的更改，且对其他用户可见，并释放被锁的资源。

### 5.2.2 ROLLBACK 语句

ROLLBACK 语句结束当前事务，并撤销在该事务期间所作的任何更改，并释放被锁的资源。

如果在此过程中存在错误，例如从表中删除了错误的行，则回滚会恢复原始数据。如果

由于 SQL 语句失败或 PL/SQL 引发异常而无法完成事务，则回滚可采取纠正措施，并可能重新开始。

**示例 5.1**：插入商品类别信息。

**注**：如果 INSERT 语句试图存储重复的 catgid 主键，PL/SQL 将引发预定义的异常 DUP_VAL_ON_INDEX。为确保撤销对所有操作的更改，异常处理程序运行 ROLLBACK。

程序代码如下。

```
DECLARE new_catgid categories.catgid%TYPE;
BEGIN
  SELECT MAX(catgid) INTO new_catgid FROM categories;

  INSERT INTO categories(catgid,catgname,parentid,currlevel)
  VALUES (new_catgid+1,'手机',new_catgid,1);
  INSERT INTO categories(catgid,catgname,parentid,currlevel)
  VALUES (new_catgid+2,'电脑',new_catgid,2);
  INSERT INTO categories(catgid,catgname,parentid,currlevel)
  VALUES (new_catgid+1,'手机',new_catgid,1);

  COMMIT;
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    ROLLBACK ;
RAISE NOTICE 'Inserts were rolled back';
END;
```

### 5.2.3 SET TRANSACTION 语句

SET TRANSACTION 语句为当前事务设置特性，可用的事务特性是事务隔离级别和事务访问模式(只读或者读/写)。SET TRANSACTION 语句必须是事务中的第一条 SQL 语句，并且在事务中只能出现一次。

只读事务对于在其他用户更新同一个表时运行多个查询很有用。在只读事务期间，所有查询都引用数据库的同一个快照，提供多表、多查询、读一致的视图。其他用户可以像往常一样继续查询或更新数据。通过提交或回滚可以结束事务。

如果将事务设置为 READ ONLY，则后续查询只会看到事务开始之前提交的更改。READ ONLY 的使用不会影响其他用户或事务。在只读事务中只允许使用 SELECT、OPEN、FETCH、CLOSE、LOCK TABLE、COMMIT 和 ROLLBACK 语句。SELECT 语句不能是 FOR UPDATE。

**示例 5.2**：只读事务中的 SET TRANSACTION 语句。

**功能描述**：只读事务收集商品表中价格在 50 以上、100 以上和 200 以上的商品信息。

程序代码如下。

```
DECLARE
  mt_50 int;
```

```
  mt_100 int;
  mt_200 int;

BEGIN
  COMMIT; -- end previous transaction
  SET TRANSACTION READ ONLY ;

  SELECT count( * )
  INTO mt_50
  FROM goods
  WHERE price>'50.0'::money ;

  SELECT count( * )
  INTO mt_100
  FROM goods
  WHERE price>'100.0'::money ;

  SELECT count( * )
  INTO mt_200
  FROM goods
  WHERE price>'200.0'::money ;

  COMMIT; -- end previous TRANSACTION
END;
```

## 5.3　自治事务

自治事务是由主事务启动的另一个独立事务。自治事务执行 SQL 操作，并提交或回滚，而不提交或回滚主事务。二者之间的控制流程如图 5.1 所示。

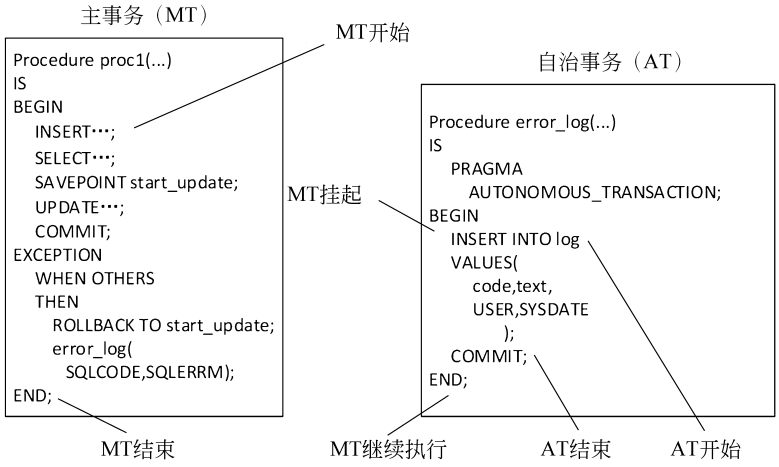


**图 5.1　主事务与自治事务间的调用关系**

自治事务的优势如下。

(1) 启动后,自治事务是完全独立的。它与主事务不共享锁、资源或提交依赖项。即使主事务回滚,也可以记录事件、增加重试计数器等。

(2) 自治事务有助于构建模块化、可重用的软件组件。可以将自治事务封装在存储的子程序中。应用程序不需要知道自治事务操作是否成功。

关于自治事务的上下文,主事务与嵌套例程共享其上下文,但不与自治事务共享。当一个自治事务调用另一个(或递归调用自身)时,这些例程不共享事务上下文。当自治事务调用非自治事务时,这些例程共享相同的事务上下文。

关于自治事务是否被调用,可以通过如下两个例子对比。

在 PL/SQL 中,例如匿名块中调用 p1()的场景,外部块中的 update 会被 p1 中的 COMMIT 提交掉,即使外部块最后有 ROLLBACK,也不会回滚外部块中的事务。

**示例 5.3**:外部块调用存储过程。

程序代码如下。

```
CREATE OR REPLACE PROCEDURE p1()
AS
BEGIN
  UPDATE customers
  SET email='lin@qq.com'
  WHERE custid=1;
  COMMIT ;
END;

BEGIN
  UPDATE customers
  SET email='jiang@foxmail.com'
  WHERE custid=2;
  p1();
    ROLLBACK;
END;
```

通过如下查询语句可以获取事务执行结果。

```
SELECT custid,email
FROM customers
WHERE custid=1 OR custid=2
```

程序运行结果如下。

```
custid | email |
------+------+
  1    |  lin@qq.com|
  2    |  jiang@foxmail.com|
```

根本原因在于,p1()内部的事务控制语句影响了外部调用者。而 KingbaseES 提供的

自治事务功能通过 PRAGMA AUTONOMOUS_TRANSACTION 语句将 p1()的事务控制语句完全独立出来，和外部块没有任何关系。

**示例 5.4**：外部块调用声明为自治事务的存储过程。

程序代码如下。

```
CREATE OR REPLACE PROCEDURE p1()
AS
PRAGMA AUTONOMOUS_TRANSACTION; -- 自治事务定义,表示当前块为自治事务
BEGIN
  UPDATE customers
  SET email='lin2@qq.com'
  WHERE custid=1;
  COMMIT ;
END;

BEGIN
  UPDATE customers
  SET email='jiang2@foxmail.com'
  WHERE custid=2;
  p1();
  ROLLBACK;
END;

  SELECT custid,email
  FROM customers
  WHERE custid=1 OR custid=2
```

程序运行结果如下。

```
custid | email |
------+------+
     1  |   lin2@qq.com|
     2  |   jiang@foxmail.com |
```

## 5.3.1　声明自治事务

将一个 PL/SQL 块定义为自治事务，只需要在声明部分包含该语句，语法格式如下。

```
PRAGMA AUTONOMOUS_TRANSACTION;
```

PL/SQL 块可以是如下的任意一种。

(1) 最顶层的(不是嵌套的)匿名块。

(2) 函数或过程，在一个包里或作为独立子程序定义。

在声明中添加自治事务标识是很容易的，但是使用自治事务有一些规则和限制，即不能将 PRAGMA 声明应用于整个包，而是需要在包主体中单独标识每个子程序。

**示例 5.5**：在包中声明自治函数。

程序代码如下。

```
-- package specification
CREATE OR REPLACE PACKAGE customer_actions AUTHID DEFINER AS
  FUNCTION change_email (cust_id NUMBER, new_email VARCHAR) RETURN VARCHAR;
END customer_actions;

CREATE OR REPLACE PACKAGE BODY customer_actions AS  --package body
  --code for function change_email
  FUNCTION change_email (cust_id NUMBER, new_email VARCHAR)
  RETURN VARCHAR IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    changed_email customers.email%TYPE;
  BEGIN
    UPDATE customers
    SET email = new_email
    WHERE custid= cust_id;
    COMMIT;
    SELECT email INTO changed_email FROM customers
    WHERE custid= cust_id;
    RETURN changed_email;
  END change_email;
END customer_actions;
```

**示例 5.6**：声明自治子程序。

程序代码如下。

```
CREATE OR REPLACE PROCEDURE change_mobile (cust_id NUMBER, new_moblie VARCHAR)
AS
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   UPDATE customers
   SET mobile = new_moblie
   WHERE custid= cust_id;

   COMMIT;
END change_mobile;
```

**示例 5.7**：声明自治 PL/SQL 块。

**注**：此示例将 PL/SQL 块标记为自治(嵌套的 PL/SQL 块不能是自治的)。

程序代码如下。

```
DECLARE
   PRAGMA AUTONOMOUS_TRANSACTION;
   cust_id INT      := 1;
   new_moblie VARCHAR := '12212212222';
BEGIN
```

```
    UPDATE customers
    SET mobile = new_moblie
    WHERE custid = cust_id;

    COMMIT;
END;
```

### 5.3.2　从 SQL 中调用自治函数

当需要从事务上下文中隔离一个模块中所作的更改时，应该把该模块定义为自治事务。记录操作日志是自治事务的一个常见的应用。

**示例 5.8**：调用自治函数。

**注**：包函数 log_msg 是自治的。因此，当查询调用该函数时，将 1 条消息插入数据库表的 dbg 中，而不会违反写数据库状态的规则(修改数据库表)。

程序代码如下。

```
DROP TABLE IF EXISTS dbg;
CREATE TABLE dbg (msg TEXT);

CREATE OR REPLACE PACKAGE pkg AUTHID DEFINER AS
  FUNCTION log_msg (msg TEXT) RETURN TEXT;
END pkg;

CREATE OR REPLACE PACKAGE BODY pkg AS
  FUNCTION log_msg (msg TEXT) RETURN TEXT IS
    PRAGMA AUTONOMOUS_TRANSACTION;
  BEGIN
    INSERT INTO dbg VALUES (msg);
    RETURN msg;
  END;
END pkg;

-- Invoke package function from query
DECLARE
  my_cust_id  INTEGER;
  my_cust_name  TEXT;
BEGIN
  my_cust_id := 1;

  SELECT pkg.log_msg(custname)
  INTO my_cust_name
  FROM customers
  WHERE custid = my_cust_id;

  ROLLBACK;
END;
```

# 第 6 章

# 动态 SQL 语句

动态 SQL 语句是指在执行时进行解析的 SQL 语句，通常 PL/SQL 程序中只执行静态的 SQL 语句。为了支持 PL/SQL 中动态 SQL 语句的执行，KingbaseES 提供了 Native dynamic SQL 和 DBMS_SQL 包两种技术。本章将主要介绍以下内容。

- 动态 SQL 语句概述。
- Native dynamic SQL。
- DBMS_SQL 包。
- SQL 注入。

## 6.1 动态 SQL 语句概述

**1. 动态 SQL 语句的必要性**

由于 PL/SQL 程序的执行是在编译阶段对变量进行绑定，识别程序中标识符的位置，检查用户权限、数据库对象等信息，因此在 PL/SQL 中可以直接执行静态 SQL 语句。

但是，编写 SQL 语句时经常需要根据程序运行、客户选择等需求来决定要操作的数据库对象，包括创建表、创建用户、为用户授权，这些操作都是在 SQL 语句运行时完成的。因此，类似 SQL 语句是无法通过编译器编译的，即静态 SQL 语句无法满足这类要求。为了在 PL/SQL 中支持 DDL 语句、DCL 语句以及更加灵活的 SQL 语句，在 PL/SQL 中引入动态 SQL 语句技术。

**示例 6.1**：创建 1 个存储过程，根据参数指定的列名和值查询店铺信息（使用静态 SQL 语句）。

程序代码如下。

```
CREATE OR REPLACE PROCEDURE dyn_sql_test(
  g_col VARCHAR2,
  g_value VARCHAR2)
AS
  g_name goods.goodname%TYPE;

BEGIN
  SELECT goodname INTO g_name FROM goods WHERE g_col=g_value;
  DBMS_OUTPUT.PUT_LINE(g_name);
END;
```

调用该存储过程时，将发生编译错误，程序代码如下。

```
BEGIN
  dyn_sql_test('good_no','150');
END;
SQL 错误 [42601]: 错误: 语法错误 在 "dyn_sql_test" 或附近的
  Position: 9 At Line: 2, Line Position: 2
```

由该示例可以看出，静态 SQL 语句不具有在运行时提供数据库标识符（如表名、视图名、列名等）的灵活性。为了满足动态查询需求，可以使用动态 SQL 语句。

**示例 6.2**：创建 1 个存储过程，根据参数指定的列名和值查询店铺信息（使用动态 SQL 语句）。

程序代码如下。

```
CREATE OR REPLACE PROCEDURE dyn_sql_test(
  g_col VARCHAR2,
  g_value VARCHAR2)
AS
  g_name goods.goodname%TYPE;
  g_str varchar2;
BEGIN
  g_str:='SELECT goodname FROM goods WHERE '||g_col||'='''||g_value||'''';
  EXECUTE IMMEDIATE g_str INTO g_name;
  RAISE NOTICE '%',g_name;
END;
```

此时，可以根据不同的输入参数进行动态查询，程序代码如下。

```
BEGIN
  dyn_sql_test('goodid','7532692');
  dyn_sql_test('model','SIS97915342');
END;
```

程序运行结果如下。

```
NOTICE:小迷糊素颜霜 20g
NOTICE:知识图谱:方法、实践与应用
```

动态 SQL 是一种用于在运行时生成和运行 SQL 语句的编程方法，将包含不确定数据库对象、创建数据库对象等的 SQL 语句封装成一个字符串，在编译阶段只进行字符串的语法检查，在运行时才进行 SQL 语句的分析与执行。通常，动态 SQL 被广泛应用于如下场景中。

（1）编写诸如临时查询类的通用且灵活的程序。

（2）编写必须运行数据库定义语言（DDL）语句的程序。

（3）在编译时不知道 SQL 语句的全文、编号或其输入和输出变量的数据类型。

**2. 动态 SQL 的编写方法**

PL/SQL 提供了以下两种编写动态 SQL 的方法。

(1) Native dynamic SQL,一种 PL/SQL 语言(即本机)的功能,用于构建和运行动态 SQL 语句。

(2) DBMS_SQL 包,用于构建、运行和描述动态 SQL 语句的 API。

Native dynamic SQL 代码比使用 DBMS_SQL 包的等效代码更易于读写,且运行速度更快(尤其是当它可以被编译器优化时)。

但是,要编写 Native dynamic SQL 代码,必须在编译时知道动态 SQL 语句输入和输出变量的数量和数据类型。如果在编译时不知道此信息,则必须使用 DBMS_SQL 包。

如果希望存储的子程序隐式返回查询结果(而不是通过 OUT REFCURSOR 参数),也必须使用 DBMS_SQL 包。

当需要 DBMS_SQL 包和 Native dynamic SQL 时,可以使用 DBMS_SQL.TO_REFCURSOR 函数和 DBMS_SQL.TO_CURSOR_NUMBER 函数,在它们之间切换。

**3. 动态 SQL 与静态 SQL 语句的比较**

静态 SQL 语句虽然在 PL/SQL 程序中有广泛使用,但在应用程序开发过程的某些情况下,只能使用动态 SQL 语句而不能使用静态 SQL 语句。

(1) 在程序编译阶段,不能提供完整的 SQL 语句。在一些复杂应用中,需要根据用户的选择或输入决定数据的查询,即用户的查询条件在编译阶段是不确定的,只有在执行时才能知道,此时只能使用动态 SQL 语句,而不能使用静态 SQL 语句。

(2) PL/SQL 中的静态 SQL 语句包括查询语句(SELECT),DML 语句(INSERT、UPDATE、DELETE、MERGE),事务控制语句(COMMIT、ROLLBACK、SAVEPOINT、SET TRANSACTION)以及表锁定语句(LOCK TABLE)。如果要在 PL/SQL 中执行其他语句,如 DDL 语句、DCL 语句、会话控制语句、系统控制语句等,只能使用动态 SQL 语句。

虽然动态 SQL 语句可以让开发人员在运行时动态切换表名、列名等数据库标识符,以及在 PL/SQL 中执行各种 DDL 操作、DCL 操作,但是在下列几种情况下,静态 SQL 语句更具优势。

(1) 静态 SQL 语句在编译时验证 SQL 语句引用的数据库对象是否有效,如果依赖的对象不存在或失效,那么 SQL 语句编译错误。动态 SQL 语句只有在运行时才能知道类似的错误。

(2) 静态 SQL 语句在编译时检查用户是否具有相应数据库对象的访问权限,如果用户没有权限,则编译失败。动态 SQL 语句对用户的权限检查在运行时进行。

(3) 使用静态 SQL 语句可以对要执行的 SQL 语句进行性能优化调整,提高程序的执行性能。但动态 SQL 语句无法进行优化处理。

## 6.2 Native dynamic SQL

Native dynamic SQL 与 DBMS_SQL 包相比更灵活,执行效率更高,Native dynamic SQL 为动态非查询语句、动态单行查询语句、动态多行查询语句提供了下列 3 种运行模式。