目录

[9.1关于应用程序 4](#_Toc721451373)

[9.1.1申请目的 4](#_Toc497697191)

[9.1.2应用程序的结构 5](#_Toc806306258)

[9.1.2.1应用程序的模式对象 5](#_Toc500409120)

[9.1.2.2应用程序的模式 6](#_Toc2054860649)

[9.1.3应用程序中的命名约定 8](#_Toc90299482)

[9.2为应用程序创建模式 10](#_Toc764652930)

[9.3授予模式权限 12](#_Toc711771423)

[9.3.1授予app\_data架构权限 12](#_Toc1242390996)

[9.3.2授予app\_code Schema权限 13](#_Toc1813259678)

[9.3.3授予app\_admin架构权限 13](#_Toc1151488434)

[9.3.4向app\_user和app\_admin\_user授予权限 14](#_Toc468241699)

[9.4创建模式对象并加载数据 14](#_Toc956630983)

[9.4.1创建表 15](#_Toc1799634994)

[2.创建部门＃，其中存储有关部门的信息 16](#_Toc1087543598)

[3.创建员工＃，其中存储有关员工的信息 17](#_Toc1420270947)

[4.创建job\_history＃，它存储公司中每个员工的工作历史 19](#_Toc2022601951)

[9.4.2创建编辑视图 20](#_Toc421529612)

[9.4.3创建触发器 21](#_Toc1651875769)

[9.4.3.1创建触发器以实施第一个业务规则 22](#_Toc913319555)

[9.4.3.2创建触发器以实施第二个业务规则 24](#_Toc2029907440)

[9.4.4创建序列 27](#_Toc205855893)

[9.4.5加载数据 28](#_Toc1078302984)

[2.使用表HR.DEPARTMENTS中的数据加载部门： 29](#_Toc351482199)

[3.使用HR.EMPLOYEES和表中的数据加载员工 29](#_Toc1466526200)

[4.使用表HR.JOB\_HISTORY中的数据加载job\_history： 30](#_Toc1219350357)

[9.4.6添加外键约束 31](#_Toc1315801719)

[9.4.7向用户授予模式对象的权限 32](#_Toc916052189)

[9.5创建employees\_pkg包 33](#_Toc1083105906)

[9.5.1为employees\_pkg创建包规范 35](#_Toc530456169)

[9.5.2为employees\_pkg创建包体 36](#_Toc44358869)

[9.5.3教程：显示employees\_pkg子程序的工作方式 41](#_Toc441794625)

[2.为子程序参数的值声明一个绑定变量 42](#_Toc1028153361)

[3.向部门90中的员工展示： 42](#_Toc850665128)

[4.显示员工101的工作历史记录： 42](#_Toc942203745)

[5.显示有关员工101的一般信息： 43](#_Toc935530362)

[6.显示有关工作管理副总裁的信息： 43](#_Toc940964610)

[7.尝试为员工101提供超出工作范围的新薪水： 44](#_Toc1706856676)

[8.为员工101提供工作范围内的新工资，并显示一般情况 44](#_Toc1647301785)

[9.将员工101的工作改为当前薪水较低的工作： 45](#_Toc35871958)

[9.5.4向app\_user和app\_admin\_user授予执行权限 46](#_Toc1372632706)

[9.5.5教程：将get\_job\_history调用为app\_user或 47](#_Toc651306571)

[9.6创建admin\_pkg包 48](#_Toc504113658)

[9.6.1创建admin\_pkg的包规范 49](#_Toc181780042)

[9.6.2为admin\_pkg创建包体 51](#_Toc303457917)

[9.6.3教程：显示admin\_pkg子程序的工作原理 54](#_Toc1591657256)

[9.6.4向app\_admin\_user授予执行权限 57](#_Toc1602050989)

[9.6.5教程：将add\_department调用为app\_admin\_user 57](#_Toc178576221)

9开发简单的Oracle数据库应用

           按照开发这个简单应用程序的说明，您将学习开发Oracle数据库应用程序的一般过程。

           •关于应用程序

               该应用程序具有以下目的，结构和命名约定。

           •为应用程序创建架构

               使用本节中的过程，为应用程序创建模式。

           •授予模式权限

               要为模式授予权限，请使用SQL语句GRANT。

           •创建架构对象并加载数据

               本节介绍如何创建表，编辑视图，触发器和应用程序的序列，如何将数据加载到表中以及如何授予这些架构对象的权限给需要它们的用户。

           •创建employees\_pkg包

               本节介绍如何创建employees\_pkg包，如何创建它子程序有效，如何将包的执行权限授予用户 谁需要它，以及这些用户如何调用其子程序之一。

           •创建admin\_pkg包

               本节介绍如何创建admin\_pkg包，其子程序如何工作，如何将包的执行权限授予需要它的用户，以及该用户如何调用其子程序之一。

9.1关于应用程序

           该应用程序具有以下目的，结构和命名约定。

           •申请的目的

               该应用程序适用于公司中的两种用户。

           •应用程序的结构

               该应用程序使用以下架构对象和架构。

           •应用程序中的命名约定

               该应用程序使用这些命名约定。

9.1.1申请目的

           该应用程序适用于公司中的两种用户。

           •典型用户（员工经理）

           •应用程序管理员

           典型用户可以执行以下操作：

            •让员工进入特定部门

            •获取给定员工的工作历史记录

            •显示给定员工的一般信息（姓名，部门，工作，经理，工资等等）

            •更改给定员工的薪水

            •更改给定员工的工作

            应用程序管理员可以执行以下操作：

            •更改现有作业的ID，标题或工资范围

            •添加新工作

            •更改现有部门的ID，名称或经理

            •添加新部门

9.1.2应用程序的结构

            该应用程序使用以下架构对象和架构：

            •应用程序的模式对象

            •应用程序的模式

9.1.2.1应用程序的模式对象

            应用程序由以下模式对象组成：

            •四个表，用于存储以下数据：

               - 工作

               - 部门

               - 雇员

               - 员工的工作经历

            •四个编辑视图，它们覆盖了表，使您能够在使用的时候使用基于版本的重新定义（EBR）来升级完成的应用程序

            •两个触发器，用于执行业务规则

            •两个序列，为新部门和新雇员生成唯一主键

            •两个包：

                - employees\_pkg，典型用户的应用程序接口（API）

                - admin\_pkg，应用程序管理员的API

               典型用户和应用程序管理员仅访问应用程序 通过其API。因此，他们只能通过调用包来更改数据子程序。

9.1.2.2应用程序的模式

           为了安全，应用程序使用这五个模式（或用户），每个模式仅具有它需要的特权：

            •app\_data，拥有除包之外的所有模式对象并加载它包含示例模式HR中表的数据的表创建包的开发人员永远不会在此模式中工作。因此，他们不能无意中更改或删除应用程序模式对象。

            •app\_code，仅拥有包employees\_pkg

                 employees\_pkg的开发人员在此架构中工作。

            •app\_admin，仅拥有admin\_pkg包

                 admin\_pkg的开发人员在此架构中工作。

            •app\_user，典型的应用程序用户，他什么都不拥有，只能执行employees\_pkg中间层应用程序服务器连接到连接池中的数据库作为app\_user。 例如，如果此模式受到SQL注入错误的影响，则攻击者只能查看和更改employees\_pkg子程序允许其查看和更改的内容。 攻击者无法删除表，升级权限，创建或更改架构对象或其他任何内容。

            •app\_admin\_user，一个应用程序管理员，他什么也不拥有，只能执行admin\_pkg和employees\_pkg

                 此架构的连接池非常小，只有特权用户才可以访问它。如果此架构遭到破坏，攻击者只能查看和更改admin\_pkg和employees\_pkg子程序允许它查看和更改。假设应用程序只有一个，而不是app\_user和app\_admin\_user，没有任何东西的架构，可以执行employees\_pkg和admin\_pkg。此架构的连接池必须足够大 典型用户和应用程序管理员。如果有一个SQL注入错误 employees\_pkg，利用该bug的典型用户可以访问admin\_pkg。假设应用程序没有应用app\_data，app\_code和app\_admin只有一个拥有所有架构对象的架构，包括包。该然后，包将拥有表的所有权限，这两者都是不必要和不受欢迎。例如，假设您有一个审计跟踪表AUDIT\_TRAIL。你想要的employees\_pkg的开发人员能够写入AUDIT\_TRAIL，但不能读取或 更改。您希望admin\_pkg的开发人员能够读取AUDIT\_TRAIL写信给它，但不要改变它。如果是AUDIT\_TRAIL，employees\_pkg和admin\_pkg 属于同一模式，然后两个包的开发人员都拥有AUDIT\_TRAIL的特权。但是，如果AUDIT\_TRAIL属于app\_data， employees\_pkg属于app\_code，admin\_pkg属于app\_admin，然后您可以以app\_data身份连接到数据库并执行以下操作：

           授权将AUDIT\_TRAIL插入app\_code;

           GRANT INSERT，选择AUDIT\_TRAIL到app\_admin;

9.1.3应用程序中的命名约定

           该应用程序使用这些命名约定。

            Item Name

Table table#

Editioning view for table# table

Trigger on editioning view table table\_{a|b}event[\_fer] where:

• a identifies an AFTER trigger.

• b identifies a BEFORE trigger.

• fer identifies a FOR EACH ROW trigger.

• event identifies the event that fires the

trigger. For example: i for INSERT, iu

for INSERT or UPDATE, d for DELETE.

PRIMARY KEY constraint in table# table\_pk

NOT NULL constraint on table#.column table\_column\_not\_null1

UNIQUE constraint on table#.column table\_column\_unique1

CHECK constraint on table#.column table\_column\_check1

REF constraint on table1#.column to table1\_to\_table2\_fk1

table2#.column

REF constraint on table1#.column1 to table1\_col1\_to\_table2\_col2\_fk1 2

table2#.column2

Sequence for table# table\_sequence

Parameter name p\_name

Local variable name l\_name

1表，table1和table2缩写为emp表示员工，部门表示部门，job\_hist表示  JOB\_HISTORY。

           2 col1和col2是列名column1和column2的缩写。约束名称不能包含超过30个字符。

9.2为应用程序创建模式

          使用本节中的过程，为应用程序创建模式。

          模式名称是：

          • 应用程序数据

          •app\_code

          •app\_admin

          •app\_user

          •app\_admin\_user

                  注意：

                  对于以下过程，您需要用户的名称和密码要具有CREATE USER和DROP USER系统权限。

          创建模式（或用户）schema\_name：

          1.使用SQL \* Plus，以具有CREATE USER的用户身份连接到Oracle数据库和DROP USER系统权限。出现SQL>提示符。

          2.如果模式存在，请使用此SQL删除模式及其对象声明：

              DROP USER schema\_name CASCADE;

              如果存在架构，则系统响应：

              用户掉线了

              如果架构不存在，系统会响应：

              DROP USER schema\_name CASCADE \*

              第1行的错误：

              ORA-01918：用户'schema\_name'不存在

          3.如果schema\_name是app\_data，app\_code或app\_admin，则创建架构使用此SQL语句：

             CREATE USER schema\_name IDENTIFIED BY password

DEFAULT TABLESPACE USERS

QUOTA UNLIMITED ON USERS

ENABLE EDITIONS;

Otherwise, create the schema with this SQL statement:

CREATE USER schema\_name IDENTIFIED BY password

ENABLE EDITIONS;

警告：

                        选择安全密码。有关安全密码的准则，请参阅

                        Oracle数据库安全指南。

                系统响应：

                用户创建。

            4.（可选）在SQL Developer中，使用。创建架构的连接

                “从SQL Developer连接到Oracle数据库”中的说明。

                   也可以看看：

                   •“关于应用程序”

                   •“从SQL \* Plus连接到Oracle数据库”

                   •Oracle Database SQL语言参考，以获取有关DROP的信息USER声明

                   •Oracle数据库SQL语言参考，以获取有关的信息CREATE USER语句

9.3授予模式权限

            要为模式授予权限，请使用SQL语句GRANT您可以在SQL \* Plus或SQL Developer的工作表中输入GRANT语句。。为了安全起见，只为每个模式授予它所需的权限。

            •授予app\_data架构权限

            •授予app\_code架构权限

            •向app\_admin架构授予权限

            •向app\_user和app\_admin\_user模式授予权限

                   也可以看看：

                   •“关于应用程序”

                   •Oracle数据库SQL语言参考，以获取有关的信息

                        GRANT声明

9.3.1授予app\_data架构权限

            仅向app\_data架构授予执行以下操作的权限：

            •连接到Oracle数据库：

                 授予创建会话app\_data;

            •为应用程序创建表，视图，触发器和序列：

                 GRANT CREATE TABLE，创建视图，创建触发器，为app\_data创建序列;

            •将示例模式HR中的四个表中的数据加载到其自己的表中：

                 在HR.DEPARTMENTS上选择给app\_data;

                 在HR.EMPLOYEES上选择给app\_data;

                 GROH选择HR.JOB\_HISTORY到app\_data;

                 在HR.JOBS上选择给app\_data;

9.3.2授予app\_code Schema权限

            仅向app\_code模式授予执行以下操作的权限：

            •连接到Oracle数据库：

                 授予创建会话app\_code;

            •创建包employees\_pkg：

                 授予app\_code创建程序;

            •创建同义词（为方便起见）：

                 GRANT创建SYNONYM到app\_code;

9.3.3授予app\_admin架构权限

            仅向app\_admin架构授予执行以下操作的权限：

            •连接到Oracle数据库：

                 授予创建会话到app\_admin;

            •创建包admin\_pkg：

                 给予app\_admin GRANT CREATE PROCEDURE;

            •创建同义词（为方便起见）：

                 GRANT创建SYNONYM到app\_admin;

9.3.4向app\_user和app\_admin\_user模式授予权限仅向app\_user和app\_admin\_user模式授予执行以下操作的权限：

            •连接到Oracle数据库：

                 授予app\_user创建会话;

                 授予创建会话到app\_admin\_user;

            •创建同义词（为方便起见）：

                 GRANT创建SYNONYM到app\_user;

                 授予创建SYNONYM到app\_admin\_user;

9.4创建模式对象并加载数据

            本节介绍如何创建表，编辑视图，触发器和序列对于应用程序，如何将数据加载到表中，以及如何授予权限这些架构对象指向需要它们的用户。要创建架构对象并加载数据：

            1.以用户app\_data身份连接到Oracle数据库。

                 有关说明，请参阅“从SQL \* Plus连接到Oracle数据库”或“从SQL Developer连接到Oracle数据库”。

            2.使用除外键约束之外的所有必要约束创建表您必须在加载数据后添加。

            3.创建编辑视图。

            4.创建触发器。

            5.创建序列。

            6.将数据加载到表中。

            7.添加外键约束。

            •创建表格

            •创建编辑视图

            •创建触发器

            •创建序列

            •加载数据

            •添加外键约束

            •向用户授予模式对象的权限

9.4.1创建表

            本节介绍如何为应用程序创建表，以及所有必需的表除了一个之外的约束，您必须在加载数据后添加这些约束。

                    注意：

                    您必须以用户app\_data身份连接到Oracle数据库。 在以下过程中，您可以在SQL \* Plus或SQL Developer的工作表中输入语句。 或者，您可以使用SQL Developer工具Create Table创建表。

            创建表：

            1.创建作业＃，它存储有关公司中作业的信息（每个工作一行）：

  CREATE TABLE jobs#

( job\_id VARCHAR2(10)

CONSTRAINT jobs\_pk PRIMARY KEY,

job\_title VARCHAR2(35)

CONSTRAINT jobs\_job\_title\_not\_null NOT NULL,

min\_salary NUMBER(6)

CONSTRAINT jobs\_min\_salary\_not\_null NOT NULL,

max\_salary NUMBER(6)

CONSTRAINT jobs\_max\_salary\_not\_null NOT NULL

)

   /

2.创建部门＃，其中存储公司的有关部门的信息（每个部门一行）：

    CREATE TABLE departments#

( department\_id NUMBER(4)

CONSTRAINT departments\_pk PRIMARY KEY,

department\_name VARCHAR2(30)

CONSTRAINT department\_name\_not\_null NOT NULL

CONSTRAINT department\_name\_unique UNIQUE,

manager\_id NUMBER(6)

)

   /

3.创建员工＃，其中存储公司的有关员工的信息（每位员工一行）：

  CREATE TABLE employees#

( employee\_id NUMBER(6)

CONSTRAINT employees\_pk PRIMARY KEY,

first\_name VARCHAR2(20)

CONSTRAINT emp\_first\_name\_not\_null NOT NULL,

last\_name VARCHAR2(25)

CONSTRAINT emp\_last\_name\_not\_null NOT NULL,

email\_addr VARCHAR2(25)

CONSTRAINT emp\_email\_addr\_not\_null NOT NULL,

hire\_date DATE

DEFAULT TRUNC(SYSDATE)

CONSTRAINT emp\_hire\_date\_not\_null NOT NULL

CONSTRAINT emp\_hire\_date\_check

CHECK(TRUNC(hire\_date) = hire\_date),

country\_code VARCHAR2(5)

CONSTRAINT emp\_country\_code\_not\_null NOT NULL,

phone\_number VARCHAR2(20)

CONSTRAINT emp\_phone\_number\_not\_null NOT NULL,

job\_id CONSTRAINT emp\_job\_id\_not\_null NOT NULL

CONSTRAINT emp\_jobs\_fk REFERENCES jobs#,

job\_start\_date DATE

CONSTRAINT emp\_job\_start\_date\_not\_null NOT NULL,

CONSTRAINT emp\_job\_start\_date\_check

CHECK(TRUNC(JOB\_START\_DATE) = job\_start\_date),

salary NUMBER(6)

CONSTRAINT emp\_salary\_not\_null NOT NULL,

manager\_id CONSTRAINT emp\_mgr\_to\_empno\_fk REFERENCES employees#,

department\_id CONSTRAINT emp\_to\_dept\_fk REFERENCES departments#

)

   /

   REF约束的原因是：

   •员工必须拥有现有工作。也就是说，列中的值employees＃.job\_id也必须是列作业＃.job\_id中的值。

   •员工必须有一名也是员工的经理。也就是说，列中的值employees＃.manager\_id也必须是列员工＃.employee\_id中的值。

   •员工必须在现有部门工作。 也就是说，列中的值employees＃.department\_id也必须是列部门＃.department\_id中的值。

  此外，员工的经理必须是员工所在部门的经理。 也就是说，列中的值employees＃.manager\_id也必须是列部门＃.manager\_id中的值。 但是，在创建departments＃时，您无法指定必要的约束，因为employees＃尚不存在。 因此，您必须稍后向departments＃添加外键约束（请参阅“添加外键约束”）。

4.创建job\_history＃，它存储公司中每个员工的工作历史（员工每个工作一行）：

  CREATE TABLE job\_history#

( employee\_id CONSTRAINT job\_hist\_to\_employees\_fk REFERENCES employees#,

job\_id CONSTRAINT job\_hist\_to\_jobs\_fk REFERENCES jobs#,

start\_date DATE

CONSTRAINT job\_hist\_start\_date\_not\_null NOT NULL,

end\_date DATE

CONSTRAINT job\_hist\_end\_date\_not\_null NOT NULL,

department\_id

CONSTRAINT job\_hist\_to\_departments\_fk REFERENCES departments#

CONSTRAINT job\_hist\_dept\_id\_not\_null NOT NULL,

CONSTRAINT job\_history\_pk PRIMARY KEY(employee\_id,start\_date),

CONSTRAINT job\_history\_date\_check CHECK( start\_date < end\_date )

)

   /

   REF约束员工，工作和部门必须存在的原因是：

   •job\_history＃.employee\_id列中的值也必须是employees#.employee\_id列中的值。

   •job\_history＃.job\_id列中的值也必须是jobs＃.job\_id列中的值 。

   •job\_history＃.department\_id列中的值也必须是department＃.department\_id列中的值。

9.4.2创建编辑视图

                    注意：

                    您必须以用户app\_data身份连接到Oracle数据库。 要创建编辑视图，请使用以下语句（按任意顺序）。您可以 在SQL \* Plus或SQL Developer的工作表中输入语句。或者，您可以使用SQL Developer工具Create创建编辑视图

   CREATE OR REPLACE EDITIONING VIEW jobs AS SELECT \* FROM jobs#

/

CREATE OR REPLACE EDITIONING VIEW departments AS SELECT \* FROM departments#

/

CREATE OR REPLACE EDITIONING VIEW employees AS SELECT \* FROM employees#

/

CREATE OR REPLACE EDITIONING VIEW job\_history AS SELECT \* FROM job\_history#

/

                    注意：

                    应用程序必须始终通过编辑视图引用基表。 否则，编辑视图不会覆盖表格，并且您无法在使用时使用EBR升级已完成的应用程序。

                    也可以看看：

                    •“创建视图”

                    •有关版本视图的一般信息，请参见“Oracle数据库开发指南

                    •有关准备应用程序以使用编辑视图的信息，请参见“Oracle数据库开发指南

9.4.3创建触发器

                    注意：

                    您必须以用户app\_data身份连接到Oracle数据库。应用程序中的触发器会强制执行以下业务规则：

              •具有工作j的员工必须具有最小和最大之间的工资 工作j的工资。

              •如果有工作j的员工有工资，那么您无法更改最低工资j为大于s的值或j的最大工资为小于s的值。（这样做会使现有数据无效。）

              •创建触发器以实施第一个业务规则

                   第一个业务规则是：有工作j的员工必须有工资工作j的最低和最高工资。

              •创建触发器以实施第二个业务规则

                   第二个业务规则是：如果有工作j的员工有薪水，那么你不能将j的最低工资更改为大于s或最大值的值 j的工资值小于s。 （这样做会使现有数据无效。）

                       也可以看看：

                       使用触发器，获取有关触发器的信息

9.4.3.1创建触发器以实施第一个业务规则

              第一个业务规则是：第一个业务规则是：具有工作j的员工必须具有工作j的最低工资和最高工资之间的工资。 将新行插入employees表或更新employees表的salary或job\_id列时，可能违反此规则。 要强制执行该规则，请在版本控制视图员工上创建以下触发器。 您可以在SQL \* Plus或SQL Developer的工作表中输入CREATE TRIGGER语句。 或者，您可以使用SQL Developer工具Create Trigger创建触发器。

              CREATE OR REPLACE TRIGGER employees\_aiufer

AFTER INSERT OR UPDATE OF salary, job\_id ON employees FOR EACH ROW

DECLARE

l\_cnt NUMBER;

BEGIN

LOCK TABLE jobs IN SHARE MODE; -- Ensure that jobs does not change

-- during the following query.

SELECT COUNT(\*) INTO l\_cnt

FROM jobs

WHERE job\_id = :NEW.job\_id

AND :NEW.salary BETWEEN min\_salary AND max\_salary;

IF (l\_cnt<>1) THEN

RAISE\_APPLICATION\_ERROR( -20002,

CASE

WHEN :new.job\_id = :old.job\_id

THEN 'Salary modification invalid'

ELSE 'Job reassignment puts salary out of range'

END );

END IF;

END;

/

             共享模式下的LOCK TABLE作业可防止其他用户在触发器查询时更改表作业。 在查询期间阻止对作业的更改是必要的，因为非阻塞读取会阻止触发器“看到”其他用户在触发器更改员工时对作业所做的更改（并防止这些用户“看到”触发器对员工所做的更改）。 在查询期间防止更改作业的另一种方法是在SELECT语句中包含FOR UPDATE子句。 但是，SELECT FOR UPDATE比SHARE MODE中的LOCK TABLE作业更多地限制并发。 SHARE MODE中的LOCK TABLE作业可以防止其他用户更改作业，但不会阻止其他用户更改共享模式中的作业。 对工作的改变可能比对员工的改变少得多。 因此，在共享模式下锁定作业比在独占模式下锁定单行作业提供更多的并发性。

9.4.3.2创建触发器以实施第二个业务规则

              第二个业务规则是：如果具有工作j的员工具有工资s，则您不能将j的最低工资更改为大于s的值或将j的最大工资更改为小于s的值。 （这样做会使现有数据无效。）更新jobs表的min\_salary或max\_salary列时，可能会违反此规则。 要强制执行规则，请在编辑视图作业上创建以下触发器。 您可以在SQL \* Plus或SQL Developer的工作表中输入CREATE TRIGGER语句。 或者，您可以使用SQL Developer工具创建触发器

             Create Trigger.

CREATE OR REPLACE TRIGGER jobs\_aufer

AFTER UPDATE OF min\_salary, max\_salary ON jobs FOR EACH ROW

WHEN (NEW.min\_salary > OLD.min\_salary OR NEW.max\_salary < OLD.max\_salary)

DECLARE

l\_cnt NUMBER;

BEGIN

LOCK TABLE employees IN SHARE MODE;

SELECT COUNT(\*) INTO l\_cnt

FROM employees

WHERE job\_id = :NEW.job\_id

AND salary NOT BETWEEN :NEW.min\_salary and :NEW.max\_salary;

IF (l\_cnt>0) THEN

RAISE\_APPLICATION\_ERROR( -20001,

'Salary update would violate ' || l\_cnt || ' existing employee records' );

END IF;

END;

/

           LOCK TABLE员工在共享模式下阻止其他用户在触发器查询时更改表员工。在查询期间阻止对员工进行更改是必要的，因为非阻塞读取会阻止触发器“看到”其他用户在触发器更改作业时对员工所做的更改（并防止这些用户“看到”触发器对作业所做的更改） 。对于此触发器，SELECT FOR UPDATE不能替代LOCK TABLE IN SHARE MODE。当您尝试更改此作业的工资范围时，此触发器会阻止其他用户将工资更改为超出新范围。因此，触发器必须锁定employees表中具有此job\_id的所有行，并锁定有人可以更新的所有行以具有此job\_id。 LOCK TABLE员工在SHARE MODE中的一种替代方法是使用DBMS\_LOCK包来创建具有job\_id名称的命名锁，然后在employees和jobs表上使用触发器来使用此命名锁来防止并发更新。但是，使用DBMS\_LOCK和多个触发器会对运行时性能产生负面影响。 LOCK TABLE员工在SHARE MODE中的另一种替代方法是在employees表上创建一个触发器，对于每个更改的员工行，该触发器锁定作业中的相应作业行。但是，这种方法会导致对employees表的更新工作过多，这种情况经常发生。

            LOCK TABLE员工在SHARE MODE中比上述替代方案更简单，并且对应用程序表的更改很少，并且可能在应用程序维护时发生，此时锁定表不会给用户带来不便。

9.4.4创建序列

                    注意：

                    您必须以用户app\_data身份连接到Oracle数据库。 要创建为新部门和新员工生成唯一主键的序列，请使用以下语句（按任意顺序）。 您可以在SQL \* Plus或SQL Developer的工作表中输入语句。 或者，您可以使用SQL Developer工具Create Sequence创建序列。

             CREATE SEQUENCE employees\_sequence START WITH 210;

             CREATE SEQUENCE departments\_sequence START WITH 275;

            为避免与将从示例模式HR中的表加载的数据冲突，employees\_sequence和departments\_sequence的起始编号必须分别超过employees.employee\_id和departments.department\_id的最大值。 在“加载数据”之后，此查询显示以下最大值：

            SELECT MAX(e.employee\_id), MAX(d.department\_id)

FROM employees e, departments d;

Result:

MAX(E.EMPLOYEE\_ID) MAX(D.DEPARTMENT\_ID)

------------------ --------------------

206 270

9.4.5加载数据

                    注意：

                    您必须以用户app\_data身份连接到Oracle数据库。使用示例模式HR中的表中的数据加载应用程序的表。

                    注意：

                    在以下过程中，您可以在SQL \* Plus或SQL Developer的工作表中输入语句。

            将数据加载到表中：

            1.使用表HR.JOBS中的数据加载作业：

                 NSERT INTO jobs (job\_id, job\_title, min\_salary, max\_salary)

SELECT job\_id, job\_title, min\_salary, max\_salary FROM HR.JOBS

   /

   结果：

   创建了19行。

2.使用表HR.DEPARTMENTS中的数据加载部门：

  INSERT INTO departments (department\_id, department\_name, manager\_id)

SELECT department\_id, department\_name, manager\_id

FROM HR.DEPARTMENTS

/

   结果：

   创建了27行。

3.使用表HR.EMPLOYEES和HR.JOB\_HISTORY中的数据加载员工，使用搜索的CASE表达式和SQL函数从HR.phone\_number和SQL函数获取employees.country\_code和employees.phone\_number以及从HR获取employees.job\_start\_date的标量子查询。JOB\_HISTORY：

  INSERT INTO employees (employee\_id, first\_name, last\_name, email\_addr,

hire\_date, country\_code, phone\_number, job\_id, job\_start\_date, salary,

manager\_id, department\_id)

SELECT employee\_id, first\_name, last\_name, email, hire\_date,

CASE WHEN phone\_number LIKE '011.%'

THEN '+' || SUBSTR( phone\_number, INSTR( phone\_number, '.' )+1,

INSTR( phone\_number, '.', 1, 2 ) - INSTR( phone\_number, '.' ) - 1 )

ELSE '+1'

END country\_code,

CASE WHEN phone\_number LIKE '011.%'

THEN SUBSTR( phone\_number, INSTR(phone\_number, '.', 1, 2 )+1 )

ELSE phone\_number

END phone\_number,

job\_id,

NVL( (SELECT MAX(end\_date+1)

FROM HR.JOB\_HISTORY jh

WHERE jh.employee\_id = employees.employee\_id), hire\_date),

salary, manager\_id, department\_id

FROM HR.EMPLOYEES

/

   结果：

   创建了107行。

           注意：

           前面的INSERT语句触发在“创建触发器以强制执行第一个业务规则”中创建的触发器。

4.使用表HR.JOB\_HISTORY中的数据加载job\_history：

   INSERT INTO job\_history (employee\_id, job\_id, start\_date, end\_date,

department\_id)

SELECT employee\_id, job\_id, start\_date, end\_date, department\_id FROM HR.JOB\_HISTORY

                      /

                      结果：

                      创建了10行。

                 5.提交更改：

                      提交;

9.4.6添加外键约束

                         注意：

                         您必须以用户app\_data身份连接到Oracle数据库。 既然表部门和员工包含数据，请使用以下ALTER TABLE语句添加外键约束。 您可以在SQL \* Plus或SQL Developer的工作表中输入语句。 或者，您可以使用SQL Developer工具添加外键添加约束。

                 ALTER TABLE departments#

ADD CONSTRAINT dept\_to\_emp\_fk

FOREIGN KEY(manager\_id) REFERENCES employees#;

                如果在部门＃和employees＃包含数据之前添加此外键约束，则在尝试使用数据加载其中任何一个时会出现此错误：

ORA-02291: integrity constraint (APP\_DATA.JOB\_HIST\_TO\_DEPT\_FK) violated - parent key not found

9.4.7向用户授予模式对象的权限

                   注意：

                   您必须以用户app\_data的身份连接到Oracle数据库。 要向用户授予权限，请使用SQL语句GRANT。 您可以在SQL \* Plus或SQL Developer的工作表中输入GRANT语句。

            仅向app\_code授予创建employees\_pkg所需的权限：

            GRANT SELECT, INSERT, UPDATE, DELETE ON employees TO app\_code;

GRANT SELECT ON departments TO app\_code;

GRANT SELECT ON jobs TO app\_code;

GRANT SELECT, INSERT on job\_history TO app\_code;

GRANT SELECT ON employees\_sequence TO app\_code;

Grant to app\_admin only the privileges that it needs to create admin\_pkg:

GRANT SELECT, INSERT, UPDATE, DELETE ON jobs TO app\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON departments TO app\_admin;

GRANT SELECT ON employees\_sequence TO app\_admin;

GRANT SELECT ON departments\_sequence TO app\_admin;

9.5创建employees\_pkg包

            本节介绍如何创建employees\_pkg包，子程序如何工作，如何将包的执行权限授予需要它的用户，以及这些用户如何调用其子程序。

创建employees\_pkg包：

            1.以用户app\_code身份连接到Oracle数据库。 有关说明，请参阅“从SQL \* Plus连接到Oracle数据库”或“从SQL Developer连接到Oracle数据库”。

            2.创建这些同义词：

          CREATE OR REPLACE SYNONYM employees FOR app\_data.employees;

CREATE OR REPLACE SYNONYM departments FOR app\_data.departments;

CREATE OR REPLACE SYNONYM jobs FOR app\_data.jobs;

CREATE OR REPLACE SYNONYM job\_history FOR app\_data.job\_history;

                 您可以在SQL \* Plus或SQL Developer的工作表中输入CREATE SYNONYM语句。 或者，您可以使用SQL Developer工具创建同义词来创建同义词。

            3.创建包规范。

            4.创建包体。

            •为employees\_pkg创建包规范

            •为employees\_pkg创建包体

            •教程：显示employees\_pkg子程序的工作原理使用SQL \* Plus，本教程将介绍employees\_pkg包的子程序如何工作。 本教程还显示了触发器employees\_aiufer和CHECK约束job\_history\_date\_check的工作原理。

            •向app\_user和app\_admin\_user授予执行权限

            •教程：将get\_job\_history调用为app\_user或app\_admin\_user使用SQL \* Plus，本教程将介绍如何以用户app\_user（通常是管理员）或app\_admin\_user（应用程序管理员）调用子程序app\_code.employees\_pkg.get\_job\_history。

9.5.1为employees\_pkg创建包规范

                    注意：

                    您必须以用户app\_code身份连接到Oracle数据库。 要为employees\_pkg创建包规范（管理器的API），请使用以下CREATE PACKAGE语句。 您可以在SQL \* Plus或SQL Developer的工作表中输入语句。 或者，您可以使用SQL Developer工具Create Package创建包。

CREATE OR REPLACE PACKAGE employees\_pkg

AS

PROCEDURE get\_employees\_in\_dept

( p\_deptno IN employees.department\_id%TYPE,

p\_result\_set IN OUT SYS\_REFCURSOR );

PROCEDURE get\_job\_history

( p\_employee\_id IN employees.department\_id%TYPE,

p\_result\_set IN OUT SYS\_REFCURSOR );

PROCEDURE show\_employee

( p\_employee\_id IN employees.employee\_id%TYPE,

p\_result\_set IN OUT SYS\_REFCURSOR );

PROCEDURE update\_salary

( p\_employee\_id IN employees.employee\_id%TYPE,

p\_new\_salary IN employees.salary%TYPE );

PROCEDURE change\_job

( p\_employee\_id IN employees.employee\_id%TYPE,

p\_new\_job IN employees.job\_id%TYPE,

p\_new\_salary IN employees.salary%TYPE := NULL,

p\_new\_dept IN employees.department\_id%TYPE := NULL );

END employees\_pkg;

/

9.5.2为employees\_pkg创建包体

                     注意：

               您必须以用户app\_code身份连接到Oracle数据库。 要为employees\_pkg（管理器的API）创建包体，请使用以下CREATE PACKAGE BODY语句。 您可以在SQL \* Plus或SQL Developer的工作表中输入语句。 或者，您可以使用SQL Developer工具Create Body创建包。

            CREATE OR REPLACE PACKAGE BODY employees\_pkg

AS

PROCEDURE get\_employees\_in\_dept

( p\_deptno IN employees.department\_id%TYPE,

p\_result\_set IN OUT SYS\_REFCURSOR )

IS

l\_cursor SYS\_REFCURSOR;

BEGIN

OPEN p\_result\_set FOR

SELECT e.employee\_id,

e.first\_name || ' ' || e.last\_name name,

TO\_CHAR( e.hire\_date, 'Dy Mon ddth, yyyy' ) hire\_date,

j.job\_title,

m.first\_name || ' ' || m.last\_name manager,

d.department\_name

FROM employees e INNER JOIN jobs j ON (e.job\_id = j.job\_id)

LEFT OUTER JOIN employees m ON (e.manager\_id = m.employee\_id)

INNER JOIN departments d ON (e.department\_id = d.department\_id)

WHERE e.department\_id = p\_deptno ;

END get\_employees\_in\_dept;

PROCEDURE get\_job\_history

( p\_employee\_id IN employees.department\_id%TYPE,

p\_result\_set IN OUT SYS\_REFCURSOR )

IS

BEGIN

OPEN p\_result\_set FOR

SELECT e.First\_name || ' ' || e.last\_name name, j.job\_title,

e.job\_start\_date start\_date,

TO\_DATE(NULL) end\_date

FROM employees e INNER JOIN jobs j ON (e.job\_id = j.job\_id)

WHERE e.employee\_id = p\_employee\_id

UNION ALL

SELECT e.First\_name || ' ' || e.last\_name name,

j.job\_title,

jh.start\_date,

jh.end\_date

FROM employees e INNER JOIN job\_history jh

ON (e.employee\_id = jh.employee\_id)

INNER JOIN jobs j ON (jh.job\_id = j.job\_id)

WHERE e.employee\_id = p\_employee\_id

ORDER BY start\_date DESC;

END get\_job\_history;

PROCEDURE show\_employee

( p\_employee\_id IN employees.employee\_id%TYPE,

p\_result\_set IN OUT sys\_refcursor )

IS

BEGIN

OPEN p\_result\_set FOR

SELECT \*

FROM (SELECT TO\_CHAR(e.employee\_id) employee\_id,

e.first\_name || ' ' || e.last\_name name,

e.email\_addr,

TO\_CHAR(e.hire\_date,'dd-mon-yyyy') hire\_date,

e.country\_code,

e.phone\_number,

j.job\_title,

TO\_CHAR(e.job\_start\_date,'dd-mon-yyyy') job\_start\_date,

to\_char(e.salary) salary,

m.first\_name || ' ' || m.last\_name manager,

d.department\_name

FROM employees e INNER JOIN jobs j on (e.job\_id = j.job\_id)

RIGHT OUTER JOIN employees m ON (m.employee\_id = e.manager\_id)

INNER JOIN departments d ON (e.department\_id = d.department\_id)

WHERE e.employee\_id = p\_employee\_id)

UNPIVOT (VALUE FOR ATTRIBUTE IN (employee\_id, name, email\_addr, hire\_date,

country\_code, phone\_number, job\_title, job\_start\_date, salary, manager,

department\_name) );

END show\_employee;

PROCEDURE update\_salary

( p\_employee\_id IN employees.employee\_id%type,

p\_new\_salary IN employees.salary%type )

IS

BEGIN

UPDATE employees

SET salary = p\_new\_salary

WHERE employee\_id = p\_employee\_id;

END update\_salary;

PROCEDURE change\_job

( p\_employee\_id IN employees.employee\_id%TYPE,

p\_new\_job IN employees.job\_id%TYPE,

p\_new\_salary IN employees.salary%TYPE := NULL,

p\_new\_dept IN employees.department\_id%TYPE := NULL )

IS

BEGIN

INSERT INTO job\_history (employee\_id, start\_date, end\_date, job\_id,

department\_id)

SELECT employee\_id, job\_start\_date, TRUNC(SYSDATE), job\_id, department\_id

FROM employees

WHERE employee\_id = p\_employee\_id;

UPDATE employees

SET job\_id = p\_new\_job,

department\_id = NVL( p\_new\_dept, department\_id ),

salary = NVL( p\_new\_salary, salary ),

job\_start\_date = TRUNC(SYSDATE)

WHERE employee\_id = p\_employee\_id;

END change\_job;

END employees\_pkg;

/

9.5.3教程：显示employees\_pkg子程序的工作原理使用SQL \* Plus，本教程将介绍employees\_pkg包的子程序如何工作。 本教程还显示了触发器employees\_aiufer和CHECK约束job\_history\_date\_check的工作原理。

                     注意：

                     您必须以SQL \* Plus中的用户app\_code连接到Oracle数据库。 要使用SQL \* Plus来显示employees\_pkg子程序的工作方式：

            1.使用格式化命令提高输出的可读性。例如：

                 SET LINESIZE 80

SET RECSEP WRAPPED

SET RECSEPCHAR "="

COLUMN NAME FORMAT A15 WORD\_WRAPPED

COLUMN HIRE\_DATE FORMAT A20 WORD\_WRAPPED

COLUMN DEPARTMENT\_NAME FORMAT A10 WORD\_WRAPPED

COLUMN JOB\_TITLE FORMAT A29 WORD\_WRAPPED

COLUMN MANAGER FORMAT A11 WORD\_WRAPPED

2.为子程序参数p\_result\_set的值声明一个绑定变量：

VARIABLE c REFCURSOR

3.展示90部门中的员工

   EXEC employees\_pkg.get\_employees\_in\_dept( 90, :c );

PRINT c

Result:

EMPLOYEE\_ID NAME HIRE\_DATE JOB\_TITLE

----------- --------------- -------------------- --------------------------

MANAGER DEPARTMENT

----------- ----------

100 Steven King Tue Jun 17th, 2003 President

Executive

===========================================================================

102 Lex De Haan Sat Jan 13th, 2001 Administration Vice President

Steven King Executive

===========================================================================

101 Neena Kochhar Wed Sep 21st, 2005 Administration Vice President

Steven King Executive

===========================================================================

4.显示员工101的工作历史记录：

   EXEC employees\_pkg.get\_job\_history( 101, :c );

PRINT c

Result:

NAME JOB\_TITLE START\_DAT END\_DATE

--------------- ----------------------------- --------- ---------

Neena Kochhar Administration Vice President 16-MAR-05

Neena Kochhar Accounting Manager 28-OCT-01 15-MAR-05

Neena Kochhar Public Accountant 21-SEP-97 27-OCT-01

5.显示有关员工101的信息：

   EXEC employees\_pkg.show\_employee( 101, :c );

PRINT c

Result:

ATTRIBUTE VALUE

--------------- ----------------------------------------------

EMPLOYEE\_ID 101

NAME Neena Kochhar

EMAIL\_ADDR NKOCHHAR

HIRE\_DATE 21-sep-2005

COUNTRY\_CODE +1

PHONE\_NUMBER 515.123.4568

JOB\_TITLE Administration Vice President

JOB\_START\_DATE 16-mar-05

SALARY 17000

MANAGER Steven King

DEPARTMENT\_NAME Executive

11 rows selected.

6.显示有关工作管理副总裁的全部信息：

   SELECT \* FROM jobs WHERE job\_title = 'Administration Vice President';

Result:

JOB\_ID JOB\_TITLE MIN\_SALARY MAX\_SALARY

---------- ----------------------------- ---------- ----------

AD\_VP Administration Vice President 15000 30000

7.尝试为员工101提供超出工作范围的新薪水：

   EXEC employees\_pkg.update\_salary( 101, 30001 );

Result:

SQL> EXEC employees\_pkg.update\_salary( 101, 30001 );

BEGIN employees\_pkg.update\_salary( 101, 30001 ); END;

\*

ERROR at line 1:

ORA-20002: Salary modification invalid

ORA-06512: at "APP\_DATA.EMPLOYEES\_AIUFER", line 13

ORA-04088: error during execution of trigger 'APP\_DATA.EMPLOYEES\_AIUFER'

ORA-06512: at "APP\_CODE.EMPLOYEES\_PKG", line 77

ORA-06512: at line 1

8.为员工101提供工作范围内的新工资，并显示有关她的全部信息：

   EXEC employees\_pkg.update\_salary( 101, 18000 );

EXEC employees\_pkg.show\_employee( 101, :c );

PRINT c

Result:

ATTRIBUTE VALUE

--------------- ----------------------------------------------

EMPLOYEE\_ID 101

NAME Neena Kochhar

EMAIL\_ADDR NKOCHHAR

HIRE\_DATE 21-sep-2005

COUNTRY\_CODE +1

PHONE\_NUMBER 515.123.4568

JOB\_TITLE Administration Vice President

JOB\_START\_DATE 16-mar-05

SALARY 18000

MANAGER Steven King

DEPARTMENT\_NAME Executive

11 rows selected.

9.将员工101的工作改为当前薪水较低的工作：

   EXEC employees\_pkg.change\_job( 101, 'AD\_VP', 17500, 90 );

Result:

SQL> exec employees\_pkg.change\_job( 101, 'AD\_VP', 17500, 90 );

BEGIN employees\_pkg.change\_job( 101, 'AD\_VP', 17500, 80 ); END;

\*

ERROR at line 1:

ORA-02290: check constraint (APP\_DATA.JOB\_HISTORY\_DATE\_CHECK) violated

ORA-06512: at "APP\_CODE.EMPLOYEES\_PKG", line 101

ORA-06512: at line 1

 10.显示有关员工的信息。 （注意工资没有改变上一步的值;它是18000，而不是17500.）

exec employees\_pkg.show\_employee( 101, :c );

print c

Result:

ATTRIBUTE VALUE

--------------- ----------------------------------------------

EMPLOYEE\_ID 101

NAME Neena Kochhar

EMAIL\_ADDR NKOCHHAR

HIRE\_DATE 21-sep-2005

COUNTRY\_CODE +1

PHONE\_NUMBER 515.123.4568

JOB\_TITLE Administration Vice President

JOB\_START\_DATE 10-mar-2015

SALARY 18000

MANAGER Steven King

DEPARTMENT\_NAME Executive

11 rows selected.

9.5.4向app\_user和app\_admin\_user授予执行权限

                    注意：

                    您必须作为用户app\_code连接到Oracle数据库。若要将.\_pkg包上的执行特权授予app\_user（通常是管理器）和app\_admin\_user（应用程序管理员），请使用以下GRANT语句（按任意顺序）。您可以在SQL\*Plus或SQL Developer的工作表中输入语句。

GRANT EXECUTE ON employees\_pkg TO app\_user;

GRANT EXECUTE ON employees\_pkg TO app\_admin\_user;

9.5.5教程：使用SQL\*Plus调用get\_job\_history作为app\_user或app\_admin\_user，本教程演示如何调用子程序app\_code..\_pkg.get\_job\_history作为用户app\_user（通常是管理器）或app\_admin\_user（应用程序管理员）。要调用.\_pkg.get\_job\_history作为app\_user或app\_admin\_user：

            1.以用户app\_user或app\_admin\_user连接到Oracle数据库

                在SQL \* Plus。

                有关说明，请参阅“从SQL \* Plus连接到Oracle数据库”。

            2.创建此同义词：

                创建SYNONYM employees\_pkg FOR app\_code.employees\_pkg;

            3.显示员工101的工作历史记录：

                EXEC employees\_pkg.get\_job\_history( 101, :c );

PRINT c

Result:

NAME JOB\_TITLE START\_DAT END\_DATE

--------------- ----------------------------- --------- ---------

Neena Kochhar Administration Vice President 16-MAR-05 15-MAY-12

Neena Kochhar Accounting Manager 28-OCT-01 15-MAR-05

Neena Kochhar Public Accountant 21-SEP-97 27-OCT-01

9.6创建admin\_pkg包

            本节展示如何创建admin\_pkg包、它的子程序如何工作、如何向需要它的用户授予包上的执行特权，以及该用户如何调用它的一个子程序。

            要创建admin\_pkg包：

            1.以用户app\_admin身份连接到Oracle数据库。

                有关说明，请参见“从SQL\*Plus连接到Oracle数据库”或“从SQL Developer连接到Oracle数据库”。

            2.创建这些同义词：

                CREATE SYNONYM departments FOR app\_data.departments;

CREATE SYNONYM jobs FOR app\_data.jobs;

CREATE SYNONYM departments\_sequence FOR app\_data.departments\_sequence;

                 您可以在SQL\*Plus或SQL Developer的工作表中输入CREATE SYNONYM语句。或者，可以使用SQL Developer工具Create Synonym创建表。

            3.创建包规范。

            4.创建包体。

            •为admin\_pkg创建包规范

            •为admin\_pkg创建包体

            •教程：如何显示管理员\_ PKG的SQL \* Plus的工作子程序，该子程序的教程展示如何管理\_ PKG的包装工作。本教程还显示如何触发\_ aufer厂工作。

            •向app\_admin\_user授予执行权限

            •教程：invoking add as \_百货\_管理员用户应用程序使用SQL \* Plus \_ shows how to this教程，invoke the function程序\_ admin.admin \_ pkg.add as the user app \_百货\_ admin（管理员用户应用\_安）然后看信息about the new百货。

9.6.1创建admin\_pkg的包规范

              注意：

              您必须作为用户app\_admin连接到Oracle数据库。要创建admin\_pkg的包规范，即应用程序管理员的API，请使用以下CREATE PACKAGE语句。您可以在SQL\*Plus或SQL Developer的工作表中输入语句。或者，可以使用SQL Developer工具Create Package创建包。

            CREATE OR REPLACE PACKAGE admin\_pkg

AS

PROCEDURE update\_job

( p\_job\_id IN jobs.job\_id%TYPE,

p\_job\_title IN jobs.job\_title%TYPE := NULL,

p\_min\_salary IN jobs.min\_salary%TYPE := NULL,

p\_max\_salary IN jobs.max\_salary%TYPE := NULL );

PROCEDURE add\_job

( p\_job\_id IN jobs.job\_id%TYPE,

p\_job\_title IN jobs.job\_title%TYPE,

p\_min\_salary IN jobs.min\_salary%TYPE,

p\_max\_salary IN jobs.max\_salary%TYPE );

PROCEDURE update\_department

( p\_department\_id IN departments.department\_id%TYPE,

p\_department\_name IN departments.department\_name%TYPE := NULL,

p\_manager\_id IN departments.manager\_id%TYPE := NULL,

p\_update\_manager\_id IN BOOLEAN := FALSE );

FUNCTION add\_department

( p\_department\_name IN departments.department\_name%TYPE,

p\_manager\_id IN departments.manager\_id%TYPE )

RETURN departments.department\_id%TYPE;

END admin\_pkg;

9.6.2为admin\_pkg创建包体

                     注意：

                     您必须作为用户app\_admin连接到Oracle数据库。要创建admin\_pkg的包体，即应用程序管理员的API，请使用以下CREATE PACKAGE BODY语句。您可以在SQL\*Plus或SQL Developer的工作表中输入语句。或者，可以使用SQL Developer工具Create Body创建包。

            CREATE OR REPLACE PACKAGE BODY admin\_pkg

AS

PROCEDURE update\_job

( p\_job\_id IN jobs.job\_id%TYPE,

p\_job\_title IN jobs.job\_title%TYPE := NULL,

p\_min\_salary IN jobs.min\_salary%TYPE := NULL,

p\_max\_salary IN jobs.max\_salary%TYPE := NULL )

IS

BEGIN

UPDATE jobs

SET job\_title = NVL( p\_job\_title, job\_title ),

min\_salary = NVL( p\_min\_salary, min\_salary ),

max\_salary = NVL( p\_max\_salary, max\_salary )

WHERE job\_id = p\_job\_id;

END update\_job;

PROCEDURE add\_job

( p\_job\_id IN jobs.job\_id%TYPE,

p\_job\_title IN jobs.job\_title%TYPE,

p\_min\_salary IN jobs.min\_salary%TYPE,

p\_max\_salary IN jobs.max\_salary%TYPE )

IS

BEGIN

INSERT INTO jobs ( job\_id, job\_title, min\_salary, max\_salary )

VALUES ( p\_job\_id, p\_job\_title, p\_min\_salary, p\_max\_salary );

END add\_job;

PROCEDURE update\_department

( p\_department\_id IN departments.department\_id%TYPE,

p\_department\_name IN departments.department\_name%TYPE := NULL,

p\_manager\_id IN departments.manager\_id%TYPE := NULL,

p\_update\_manager\_id IN BOOLEAN := FALSE )

IS

BEGIN

IF ( p\_update\_manager\_id ) THEN

UPDATE departments

SET department\_name = NVL( p\_department\_name, department\_name ),

manager\_id = p\_manager\_id

WHERE department\_id = p\_department\_id;

ELSE

UPDATE departments

SET department\_name = NVL( p\_department\_name, department\_name )

WHERE department\_id = p\_department\_id;

END IF;

END update\_department;

FUNCTION add\_department

( p\_department\_name IN departments.department\_name%TYPE,

p\_manager\_id IN departments.manager\_id%TYPE )

RETURN departments.department\_id%TYPE

IS

l\_department\_id departments.department\_id%TYPE;

BEGIN

INSERT INTO departments ( department\_id, department\_name, manager\_id )

VALUES ( departments\_sequence.NEXTVAL, p\_department\_name, p\_manager\_id )

RETURNING department\_id INTO l\_department\_id;

RETURN l\_department\_id;

END add\_department;

END admin\_pkg;

/

9.6.3教程：本教程使用SQL\*Plus演示admin\_pkg子程序如何工作，它显示了admin\_pkg包的子程序如何工作。本教程还展示了触发器.\_aufer如何工作。

                   注意：

                   您必须作为用户app\_admin从SQL\*Plus连接到Oracle数据库。

            要显示admin\_pkg子程序的工作方式：

            1.显示ID为AD\_VP的作业信息：

                SELECT \* FROM jobs WHERE job\_id = 'AD\_VP';

Result:

JOB\_ID JOB\_TITLE MIN\_SALARY MAX\_SALARY

---------- ----------------------------------- ---------- ----------

AD\_VP Administration Vice President 15000 30000

            2.增加此工作的最高工资并再次显示相关信息：

                EXEC admin\_pkg.update\_job( 'AD\_VP', p\_max\_salary => 31000 );

SELECT \* FROM jobs WHERE job\_id = 'AD\_VP';

Result:

JOB\_ID JOB\_TITLE MIN\_SALARY MAX\_SALARY

---------- ----------------------------------- ---------- ----------

AD\_VP Administration Vice President 15000 31000

            3.显示ID为IT\_PROG的作业信息：

                SELECT \* FROM jobs WHERE job\_id = 'IT\_PROG';

Result:

JOB\_ID JOB\_TITLE MIN\_SALARY MAX\_SALARY

---------- ----------------------------------- ---------- ----------

IT\_PROG Programmer 4000 10000

            4.尝试增加这份工作的最高工资：

                EXEC admin\_pkg.update\_job( 'IT\_PROG', p\_max\_salary => 4001 );

Result (from SQL\*Plus):

SQL> EXEC admin\_pkg.update\_job( 'IT\_PROG', p\_max\_salary => 4001 );

BEGIN admin\_pkg.update\_job( 'IT\_PROG', p\_max\_salary => 4001 ); END;

\*

ERROR at line 1:

ORA-20001: Salary update would violate 5 existing employee records

ORA-06512: at "APP\_DATA.JOBS\_AUFER", line 12

ORA-04088: error during execution of trigger 'APP\_DATA.JOBS\_AUFER'

ORA-06512: at "APP\_ADMIN.ADMIN\_PKG", line 10

ORA-06512: at line 1

            5.添加新作业并显示相关信息：

           EXEC admin\_pkg.add\_job( 'AD\_CLERK', 'Administrative Clerk', 3000, 7000 );

SELECT \* FROM jobs WHERE job\_id = 'AD\_CLERK';

Result:

JOB\_ID JOB\_TITLE MIN\_SALARY MAX\_SALARY

---------- ----------------------------------- ---------- ----------

AD\_CLERK Administrative Clerk 3000 7000

            6.显示有关部门100的信息：

                SELECT \* FROM departments WHERE department\_id = 100;

Result:

DEPARTMENT\_ID DEPARTMENT\_NAME MANAGER\_ID

------------- ------------------------------ ----------

100 Finance 108

            7.更改部门100的名称和经理，并显示有关信息：

                EXEC admin\_pkg.update\_department( 100, 'Financial Services' );

EXEC admin\_pkg.update\_department( 100, p\_manager\_id => 111,p\_update\_manager\_id => true );

SELECT \* FROM departments WHERE department\_id = 100;

Result:

DEPARTMENT\_ID DEPARTMENT\_NAME MANAGER\_ID

------------- ------------------------------ ----------

100 Financial Services 111

9.6.4向app\_admin\_user授予执行权限

                   注意：

                   您必须作为用户app\_admin连接到Oracle数据库。若要将包admin\_pkg上的执行特权授予app\_admin\_user（应用程序管理员），请使用以下GRANT语句。您可以在SQL\*Plus或SQL Developer的工作表中输入语句。

            GRANT EXECUTE ON admin\_pkg TO app\_admin\_user;

9.6.5教程：将add\_department调用为app\_admin\_user

   使用SQL\*Plus，本教程展示了如何调用函数app\_admin.admin\_pkg.add\_department作为用户app\_admin\_user（应用程序管理员），然后查看有关新部门的信息。要调用admin\_pkg.add\_department作为app\_admin\_user：

            1.从SQL\*Plus作为用户app\_admin\_user连接到Oracle数据库。有关说明，请参见“从SQL\*Plus连接到Oracle数据库”。

            2.创建此同义词：

                CREATE SYNONYM admin\_pkg FOR app\_admin.admin\_pkg;

            3.为函数的返回值声明一个绑定变量：

                VARIABLE n NUMBER

            4.添加一个没有经理的新部门：

                EXEC :n := admin\_pkg.add\_department( 'New department', NULL );

            5.显示新部门经理的ID：

                PRINT :n

Result:

N

----------

275

            要查看有关新部门的信息：

            1.以用户app\_admin身份连接到Oracle数据库。

            2.显示有关新部门的信息：

               SELECT \* FROM departments WHERE department\_name LIKE 'New department%';

Result:

DEPARTMENT\_ID DEPARTMENT\_NAME MANAGER\_ID

------------- ------------------------------ ----------

275 New department