***Примеры хранимого кода для предметной области Автовокзал***

Основные пользователи системы – кассиры, которые продают билеты, и диспетчер, который закрывает продажу билетов на рейс в момент отправки автобуса.

Для реализации этого функционала напишем соответствующие процедуры. Они будут храниться на сервере и выполняться сервером при вызове из клиентского приложения (т.е., допустим, Кассир нажимает на кнопочку «Продажа», а в обработчике этой кнопки содержится вызов хранимой процедуры).

**1. Продажа билетов (процедура sale)**

Алгоритм процедуры очень простой – сначала проверяем, есть ли свободные места на заданный рейс в нужном количестве. Если есть - изменяем столбец tickets (количество проданных билетов) в таблице trips и вычисляем стоимость билетов (сумму, которую должен заплатить клиент). Всё это делаем при помощи SQL запросов.

*Входными* параметрами являются **Код рейса** (cod\_r), **Код пункта, до которого покупается билет** (cod\_p), и необходимое клиенту **количество билетов** (n\_ticket),

*выходными* — **признак**, успешно ли выполнена продажа (err – если билеты проданы успешно, то err=0, в противном случае возвращаем код ошибки), а также **стоимость** всех купленных билетов (s).

Текст этой процедуры (один из возможных):

create or replace procedure sale (cod\_r trips.cod\_trip%type,

cod\_p points.cod\_point%type,

n\_ticket number,

err out number, s out number

)

as

free\_pl number;

begin

select free\_places into free\_pl from points\_trips

where cod\_point=cod\_p and cod\_trip=cod\_r;

if n\_ticket>free\_pl then

err:=1;

else

err:=0;

update trips set tickets=tickets+n\_ticket

where cod\_trip=cod\_r;

select km\_prices.price\*points.distance\*n\_ticket into s

from km\_prices, points\_trips, points

where km\_prices.class=points\_trips.class and

points\_trips.cod\_point=points.cod\_point and

points\_trips.cod\_point=cod\_p and points\_trips.cod\_trip=cod\_r;

end if;

exception

when TOO\_MANY\_ROWS then

err:=2;

when NO\_DATA\_FOUND then

err:=3;

when OTHERS then

err:=4;

end;

/

Теоретически при правильном функционировании системы исключительные ситуации произойти не должны. Но все-таки раздел exception принято включать в текст хранимой процедуры.

Исключение TOO\_MANY\_ROWS могло бы возникнуть при исполнении запроса на выборку, если бы он возвратил более одной строки, исключение NO\_DATA\_FOUND  — если бы select вообще не возвратил строк, OTHERS — любые другие сбойные ситуации.

Создайте эту процедуру – не забудьте последний /, он является символом окончания процедуры. Если увидите сообщение об ошибке, для её локализации можно ввести команду:

select line, text from user\_errors;

Для проверки работоспособности успешно созданной процедуры рекомендуем написать небольшой отладочный блок PL/SQL, который позволит получить значения выходных параметров.

Для того, чтобы в SQL\*Plus вывод работал корректно, сначала введите команду

set serveroutput on

А теперь сам отладочный блок, содержащий вызов процедуры sale и вывод значений выходных параметров err и s.

declare err number;

s number;

begin

sale(1,1,3,err,s);

dbms\_output.put\_line('err=' || err);

dbms\_output.put\_line('s=' || s);

end;

/

Входные параметры обозначают, что мы продаем 3 билета на рейс с кодом 1 до пункта тоже с кодом 1. Если рейс 1 проходит через пункт 1 и на него имеется не менее трех свободных мест, то параметр err должен получиться равным 0, а параметр s должен иметь значение стоимости 3 билетов на тот автобус, который назначен на рейс с кодом 1.

Протестируйте процедуру на разных значениях входных параметров.

**2. Процедура закрытия рейса в момент отправки автобуса (send\_trip)**

Алгоритм такой – проверяем, продан ли на заданный рейс хотя бы один билет (допустим, диспетчер нечаянно второй раз нажимает на кнопку **Закрыть рейс**). Если количество проданных билетов >0, то сначала отправляем сведения об отправляемом рейсе в Хранилище данных, а затем обнуляем столбец tickets (количество проданных билетов) в таблице trips, т.е. начинаем новый процесс продажи билетов с нуля, но уже на тот рейс, который отправляется через неделю.

Поясним наполнение таблицы sent\_trips – как раз в неё переносятся данные об отправленном рейсе.

Столбец id\_trip – суррогатный ключ, уникальный для каждого отправленного рейса (не путать с кодом рейса!), можно заполнить знакомым способом – последовательность и триггер. Последовательность создадим – от этого никуда не деться:

create sequence sent\_trips\_seq

А без триггера можно обойтись, если заполнение id\_trip выполнять непосредственно в процедуре отправки рейса.

Один из возможных вариантов процедуры **send\_trip** (можно предложить множество других, примерно, равноценных):

create or replace procedure send\_trip(cod\_t trips.cod\_trip%type, err out number)

as

tickets\_number number;

begin

select tickets into tickets\_number

from trips

where trips.cod\_trip = cod\_t;

if tickets\_number=0 then

err:=1;

else

insert into sent\_trips (id\_trip, cod\_trip, cod\_route, cod\_bus, cod\_model, hour, minute, week\_day, sent\_day, fill\_bus, tickets)

select sent\_trips\_seq.nextval, cod\_trip, cod\_route, cod\_bus, cod\_model, hour, minute, week\_day, sysdate, tickets/places\*100, tickets

from trips join buses using(cod\_bus) join models using(cod\_model)

where cod\_trip=cod\_t;

update trips set tickets = 0

where trips.cod\_trip = cod\_t;

err:=0;

end if;

exception

when NO\_DATA\_FOUND then

err:=2;

when OTHERS then

err:=3;

end;

Для проверки процедуры закроем (отправим) рейс с кодом 1

declare err number;

begin

send\_trip(1,err);

dbms\_output.put\_line('err=' || err);

end;

Можно проверить, что в таблице sent\_trips появилась строка об отправленном рейсе с сегодняшней датой:

select \* from sent\_trips

а количество проданных билетов в таблице trips обнулилось:

select \* from trips where cod\_trip=1

**3. Примеры триггеров**

Если продолжать тему хранилища данных автовокзала, то можно привести два примера коротких, но очень полезных триггеров, которые позволят всегда поддерживать в актуальном состоянии таблицы-справочники хранилища.

Например, возьмём таблицу a\_trips. Каждый раз при добавлении нового рейса в таблицу trips оперативной базы данных информация о нём должна быть добавлена и в таблицу a\_trips. При удалении какого-либо рейса из таблицы trips в таблице a\_trips должна быть установлена дата окончания существования рейса, но сам рейс из этой архивной таблицы не удаляется.

Соответствующие триггеры:

create or replace trigger insert\_trip  
after insert on trips  
for each row  
begin  
 insert into a\_trips(cod\_trip, date\_begin) values(:new.cod\_trip, sysdate);  
end;

create or replace trigger delete\_trip  
before delete on trips  
for each row  
begin  
 update a\_trips  
      set date\_end = sysdate  
      where a\_trips.cod\_trip = :old.cod\_trip;

end;