Пример отчета по практике «SPbPU-student-thesis-template». Версия от 22 декабря 2020 г.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»**
НАЗВАНИЕ ИНСТИТУТА
НАЗВАНИЕ ВЫСШЕЙ ШКОЛЫ

Отчет о прохождении такой-то практики на тему: «Тема практики»

Батищева Михаила Николаевича, гр. N

Направление подготовки: XX.XX.XX Наименование направления подготовки.
Место прохождения практики: СПбПУ, ИКНТ, ВШИСиСТ.
Сроки практики: с дд.мм.гггг по дд.мм.гггг.
Руководитель практики от ФГАОУ ВО «СПбПУ»: Фамилия Имя Отчество, должность, степень.
Консультант практики от ФГАОУ ВО «СПбПУ»: Сабинин Олег Юрьевич, должность, степень.
Оценка: _____

Руководитель практики
от ФГАОУ ВО «СПбПУ» И.О. Фамилия

Консультант практики
от ФГАОУ ВО «СПбПУ» О.Ю. Сабинин

Обучающийся М.Н. Батищев

Дата: дд.мм.гггг

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Наша жизнь прочно связана с цифровыми технологиями. Сегодня трудно представить наше существование без компьютера или интернета. Большинство повседневных действий переводится в онлайн. С каждым годом все больше появляется новых информационных систем, а любая информационная система, будь то приложение или вебсайт, тем или иным способом взаимодействует с данными, которые нужно правильно хранить и обрабатывать. Для взаимодействия с данными большого объема были разработаны базы данных.

Ни для кого не является секретом, что одним из важных этапов разработки любого программного обеспечения является обработка исключительных ситуаций. Такие ситуации не возникают в идеальных условиях, но повсеместно встречаются в нашей жизни. Мы не можем предусмотреть абсолютно все возможные варианты развития событий, но постараться предугадать самые часто встречающиеся ошибки, нам никто не запрещает.

При разработке программного продукта любой программист задается вопросами разного рода. Что будет с машиной, если неожиданно перестанет поступать питание? А если пользователь выйдет из приложения, не сохранив отчет? Что нужно передать клиенту, если запрашиваемых данных не существует? Продолжать список можно бесконечно, но если мы предполагаем, что такая ситуация может возникнуть, то не лишним будет попробовать защититься от последствий. При некоторых исключительных ситуациях такие последствия могут быть фатальными, стоимость необработанной ошибки в крупных кампаниях может исчисляться миллионами долларов, а для некоторых сфер, может исчисляться и в человеческих жизнях. Поэтому для разработчиков нужно предоставить максимально удобный и простой способ для обработки исключительных ситуаций.

Одной из самых популярных систем управления базами данных уже достаточно долгое время является Oracle Database. Она славиться своей надежностью, производительностью, масштабируемостью и безопасностью.

Oracle предоставляет разработчикам баз данных очень гибкий и мощный механизм для работы с ошибками, но он не лишен недостатков, которые могут создать существенные проблемы для групп разработчиков, которые хотят построить систему управления ошибками, обладающую свойствами надежности, содержательности и последовательности.

Целью данной работы является изучение существующих способов взаимодействия с исключительными ситуациями на языке Oracle PL/SQL и выявление недостатков в данном механизме.

Для достижения поставленной цели необходимо:

A. Изучить предоставляемые Oracle механизмы работы с ошибками.
B. Выявить основные недостатки и неудобства при работе с данной системой.
C. Проанализировать, полученные на прошлом этапе результаты, и предложить способы исправления данных проблем.

ГЛАВА 1. МЕХАНИЗМЫ ОБРАБОТКИ ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ В ЯЗЫКЕ PL/SQL

В данной главе рассматривается базовые концепции работы с исключительными ситуациями в языке Oracle PL/SQL.

В параграфе 1.1 привидена основная информация об ошибках в PL/SQL. В параграфе 1.2 описаны методы работы с именованными исключениями. Возможности для инициирования исключительных ситуаций описаны в параграфе 1.3. Про обработку ошибок рассказывается в параграфе 1.4.

Основные недостатки при работе с ошибками и предложения по их исправлению описываются в параграфе 1.5.

1.1. Основные концепции обработки исключительных ситуаций в PL/SQL

Исключительными ситуациями в языке PL/SQL считаются любые ситуации, которые не должны возникать при нормальном выполнении программы[2].

Под данное определение попадает достаточно большое множество ситуаций, в качестве примера исключений можно привести следующие события: отсутствие данных, ошибки работы экземпляра, непредусмотренные программой действия пользователей. Любая из приведенных выше ситуаций может привести к нежелательному результату, если не будет вовремя найдена и обработана.

В языке PL/SQL ошибки перехватываются и обрабатываются в отдельном месте в коде, называемым блок обработки исключений. Такой подход позволяет работать с ошибками как с событиями, передавая управление нужному блоку сразу после возникновения исключительной ситуации, вне зависимости от того в какой конкретно месте возникла данная ошибка.

Продемонстрируем преимущества такого метода. На рис.1.1, показан вариант линейной обработки ошибок, в таком случае, после каждой команды SELECT мы должны проверить наличие данных.

```
BEGIN
   SELECT ...
      -- check for 'no data found' error
   SELECT ...
      -- check for 'no data found' error
   SELECT ...
      -- check for 'no data found' error
```

Рис.1.1. Пример кода с последовательной обработкой ошибок

Если нам заранее известно, что имеется необходимость во всех данных из каждой команды SELECT, а при отсутствии хотя бы одних из них, мы должны выполнить какие-либо действия, то имеется возможность сильно упростить код, как показано на рис.1.2.

```
BEGIN
   SELECT ...
   SELECT ...
   SELECT ...
   ...
EXCEPTION
   WHEN NO_DATA_FOUND THEN  -- catches all 'no data found' errors
```

Рис.1.2. Пример обработки ошибок в отдельном блоке

В таком случае обработка всех ошибок данного типа будет производиться в одном месте, отлаживать и дорабатывать такую программу будет гораздо проще, по сравнению с первым случаем. Также вынесение обработчика в отдельный блок, позволяет отделить логику работы основной программы, от раздела исключений, что позволяет уменьшить количество кода и повысить его читаемость.

В PL/SQL исключения бывают двух типов: системные и пользовательские. Системные исключения – это исключения, которые инициируются ядром PL/SQL в случае нарушением программы правил исполнения установленных Oracle. Пользовательские исключения, в свою очередь, определяются программистом и обычно связаны с конкретным приложением.

Каждая ошибка имеет свой номер, но обрабатывать исключения можно только по их имени. В Oracle имеется несколько предопределенных ошибок, это часто возникающие ошибки, для которых заданно имя в пакете STANDARD.

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Рис.1.3. Предопределенные ошибки из пакета STANDARD

На рис.1.3, приведены примеры предопределенных исключений.

1.2. Определение исключений

В подавляющем большинстве приложений определенных Oracle исключений недостаточно, поэтому необходимо уметь правильно определять и работать с пользовательскими ошибками, которые описывают исключительные ситуации для конкретного приложения.

1.2.1. Объявление пользовательских исключений

Пользовательские исключения позволяют программисту определить новые виды ошибок, для ситуаций, которые не покрываются системными исключениями.

Очень важной особенностью пользовательских исключений является то, что взаимодейтсвие с ними не отличается от работы с системными ошибками.

Определение именованных исключений похоже на объявление обычных переменных. Сначала указывается название ошибки, за которым следует ключевое слово EXCEPTION.

```
DECLARE
   past_due EXCEPTION;
   acct_num NUMBER;
BEGIN
   DECLARE  ---------- sub-block begins
      past_due EXCEPTION;  -- this declaration prevails
      acct_num NUMBER;
   BEGIN
      ...
      IF ... THEN
         RAISE past_due;  -- this is not handled
      END IF;
   END;  ------------- sub-block ends
EXCEPTION
   WHEN past_due THEN  -- does not handle RAISEd exception
      ...
END;
```

Рис.1.4. Пример объявление именованного исключения

На рис.1.4 показан пример объявления именованного исключения под названием past_due. Во вложенном блоке объявляется еще одно исключение с таким же именем, так как вложенный блок не имеет обработчика, то его (локальное) исключение past_due не будет обработано. Внешний блок ничего не знает об исключение past_due из вложенного блока, поэтому его обработчик не сможет отловить и обработать данное исключение[4].

1.2.2. Привязка исключений к кодам ошибок

При помощи директив компилятора можно связать именованное исключение с кодом ошибки. Делается это при помощи команды EXCEPTION_INIT, в которую передается имя исключения, объявленного ранее, и код ошибки. После такой привязки обращаться к ошибке в разделе WHEN можно по имени.

Обычно данная директива используется в двух случаях: для задания имени системному исключению, для которого не предусмотренно предопределенного имени, и для работы со специфичными для приложения ошибками.

```
DECLARE
   deadlock_detected EXCEPTION;
   PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
BEGIN
   ... -- Some operation that causes an ORA-00060 error
EXCEPTION
   WHEN deadlock_detected THEN
      -- handle the error
END;
```

Рис.1.5. Пример использования директивы компилятора EXCEPTION_INIT

В примере показанном на рис.1.5 происходит привязка ошибки с номером -60 к именованному исключению deadlock_detected.

Хорошим тоном считается вынесение часто используемых ошибок в один пакет и привязка их к номерам, как, например, это сделано в пакете STANDARD со стандартными исключениями. В таком случае, в остальном коде приложения, можно будет обращаться к ошибкам из данного пакета. Следуя такой практике можно повысить читаемость кода за счет стандартизации в определении ошибок.

1.3. Инициирование исключительных ситуаций

Существует несколько способов инициировать исключение. В первую очередь, исключения инициируются Oracle в случае возникновения ошибки. Программист может сам инициировать исключения либо при помощи команды RAISE, либо используя процедуру RAISE_APPLICATION_ERROR. Рассмотрим последние два способа подробнее.

Команда RAISE останавливает нормальное выполнение программы PL/SQL и передает управление обработчику ошибок. Данная инструкция имеет следующий синтаксис.

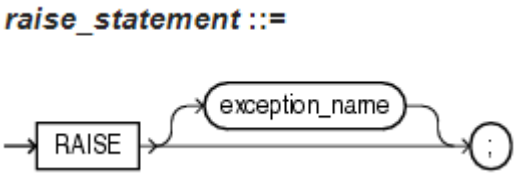

Рис.1.6. Синтаксис команды RAISE

Как видно на рис.1.6 сначала указывается ключевое слово RAISE за которым следует необязательное имя исключения.

Использование данной команды без указания имени исключения, можно только в разделе WHEN обработчика исключений. В таком случае обрабатываемое в данный момент исключение инициируется повторно, а процесс обработки исключения переходит в родительский блок. Это может быть полезно в случаях, когда нам нужно освободить какие-то ресурсы, либо сохранить какие-либо данные, но что делать с возникшей ошибкой в данный момент времени мы не знаем.

```
DECLARE
   out_of_stock   EXCEPTION;
   number_on_hand NUMBER := 0;
BEGIN
   IF number_on_hand < 1 THEN
      RAISE out_of_stock; -- raise an exception that we defined
   END IF;
EXCEPTION
   WHEN out_of_stock THEN
      -- handle the error
      DBMS_OUTPUT.PUT_LINE('Encountered out-of-stock error.');
END;
/
```

Рис.1.7. Пример использования команды RAISE

На рис.1.7 представлен пример инициирования ошибки с использованием инструкции RAISE. В блоке происходит объявление исключения out_of_stock и переменной number_of_hand проинициализированной значением 0, после чего в условном блоке происходит проверка значения переменной, если оно меньше 1, то происходит инициирование объявленного ранее исключения, после чего исполнение передается блоку обработки ошибок.

Другим способом инициировать программисту исключение является процедура RAISE_APPLICATION_ERROR из пакета DBMS_STANDARD.

```
PROCEDURE RAISE_APPLICATION_ERROR (
   num binary_integer,
   msg varchar2,
   keeperrorstack boolean default FALSE);
```

Рис.1.8. Заголовок процедуры RAISE_APPLICATION_ERROR

Как видно на рис.1.8 данная процедура может принимать три параметра, два обязательных и один необязательный. Первый параметр – это код ошибки, которую необходимо инициировать, в диапазоне от -20999 до -20000. Второй параметр – это сообщение длиной до 2048 байт, которое будет связано с ошибкой. Третий параметр keeperrorstack указывает, нужно ли добавить ошибку к уже имеющимся в стеке (данное поведение используется при указании значения TRUE), или необходим заменить существующую ошибку (значение FALSE, используемое по умолчанию).

```
IF l_customer_credit > l_credit_limit THEN
    raise_application_error(-20111,'Credit Limit Exceeded');
END IF;
```

Рис.1.9. Пример использования процедуры RAISE_APPLICATION_ERROR

Пример на рис.1.9 демонстрирует использование рассматриваемой процедуры. При нарушении бизнес-логики происходит инициирование ошибки с кодом -20111 с информацией об ошибке.

1.4. Обработка ошибок

При возникновении исключительной ситуации нормальное выполнение PL/SQL кода останавливается, а управление получает раздел обработки исключений, если обработчик данного блока не может справиться с такой ситуацией, то управление переходит в родительский блок.

Синтаксис блока обработки исключений имеет следующий вид, показанный на рисунке рис.1.10.

```
EXCEPTION
  WHEN exception1 THEN -- handler for exception1
    sequence_of_statements1
  WHEN exception2 THEN -- another handler for exception2
    sequence_of_statements2
  ...
  WHEN OTHERS THEN -- optional handler for all other errors
    sequence_of_statements3
END;
```

Рис.1.10. Синтаксис обработчика ошибок

Он может находиться только в конце PL/SQL блока либо вовсе может отсутствовать. Если имя возникшего исключения совпадает с именем, указанным в разделе WHEN, то выполняются команды, находящиеся после ключевого слова THEN. В случае, когда ни один раздел с указанным именем не подходит управление переходит в раздел WHEN OTHERS, если он указан.

Стоит отметить, что ошибка может быть обработана только одним из разделов. После выполнения команд управление передается в вызывающий блок.

1.5. Существующие проблемы обработки ошибок

Несмотря на то, что существующая система имеет довольно много возможностей для разработчиков, она не лишена недостатков. Далее будут рассмотрены некоторые проблемы, с которыми сталкиваются программисты при разработке программного обеспечения для базы данных.

Дублирование кода. Довольно часто встречаются ситуации, в которых возможно возникновение одних и тех же ошибок, для обработки которой придется воспроизводить уже написанный код. Дублирование кода противоречит принципу разработки программного обеспечения DRY (Don't repeat yourself, рус. не повторяйся), предложенного Энди Хайдом и Дэйвом Томасом в книге «Программист-прагматик»[3]. Увеличение кодовой базы приводит к увеличению затрат ресурсов при отладке и корректировки приложения. Одним из способов решения данной проблемы является вынесение повторяющегося кода в отдельные процедуры или функции, и заменой повторяющегося кода на вызов необходимых подпрограмм.

Очень явно данная проблема выражена при работе с пакетом UTL_FILE, а конкретнее, с процедурой GET_LINE. Данная процедура предназначена для извлечения строк из файлов, и в случае невозможности считать новую строку, будет выдано исключение NO_DATA_FOUND. Это приводит нас к тому, что каждый раз, когда нам необходимо считать данные из файла, мы должно написать код для обработки этой ошибки, который будет везде одинаковый. Данный код будет затруднять понимание основной логики, а допущенная ошибка при копировании кода, может привести к нежелательным последствиям[5].

```
PROCEDURE read_file_and_do_stuff (
   dir_in IN VARCHAR2, file_in IN VARCHAR2
)
IS
   l_file   UTL_FILE.file_type;
   l_line   VARCHAR2 (32767);
BEGIN
   l_file := UTL_FILE.fopen (dir_in, file_in, 'R', max_linesize => 32767);

   LOOP
      UTL_FILE.get_line (l_file, l_line);
      do_stuff;
   END LOOP;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      UTL_FILE.fclose (l_file);
      more_stuff_here;
END;
```

Рис.1.11. Пример использования UTL_FILE

На рис.1.11 показан пример процедуры для чтения и обработки файла, с использованием пакета UTL_FILE. Половину кода занимает обработчик ошибок, который должен присутствовать и обычно будет выполнять одну и ту же работу. Все это можно было бы сокрыть за одной функцией и работать с файлами при помощи вызова нужной подпрограммы, что займет всего одну строку кода, но позволит сократить количество ошибок и сэкономить время, затраченное на написание кода. В более сложных ситуациях, в которых использование такой функции будет затруднительно или не оптимально, все также можно будет воспользоваться пакетом UTL_FILE.

Другой немаловажной проблемой при работе с ошибками является тот факт, что EXCEPTION в PL/SQL это особая разновидность структуры данных. Мы имеем достаточно небольшой набор возможностей для работы с ней, после объявления переменной данного типа, мы можем только инициировать или обрабатывать ее. Нету возможности для расширения функциональности данной структуры, мы не можем добавить дополнительные поля, мы не можем передавать исключения как параметр для процедуры или функции.

Также Oracle не дает возможностей для организации и классификации исключительных ситуаций, относящихся к конкретному приложению. В нашем распоряжении находиться 1000 кодов в диапазоне от -20999 до -20000, необходимость контролировать эти значения является обязанностью разработчика[2].

Для решения последних двух проблем можно реализовать альтернативную систему. Мы можем создать свои исключения и работать непосредственно с ними.

Например, перечислить ошибки в отдельной таблице, тогда мы сможем передавать ошибки как параметры, дополнять всеми необходимыми для нас атрибутами, а необходимость контролировать это все перейдет от программиста к пакету, который будет этим заниматься. Под собственными ошибками будут скрыты исключения PL/SQL, в таком случае, мы, не потеряв преимуществ от работы со стандартных исключениями, будем иметь возможность при необходимости расширить их функциональность[1].

Дополнить такой пакет можно множеством различных способов, мы имеем полный контроль для аудита ошибок, и можем его корректировать под свои нужды, в зависимости от приложения, можно собирать необходимый контекст возникновения ошибки и дополнять его при необходимости. Расширение пакета подпрограммами для обработки часто возникающих исключительных ситуаций позволит избежать проблемы дублирования кода.

1.6. Выводы

В ходе данной главы была рассмотрена основная информация об ошибках в PL/SQL. Были изучены инструменты для объявления пользовательских ошибок, для привязки исключительных ситуаций к кодам ошибок, были рассмотрены способы инициирования ошибок и их обработки.

При изучение существующих механизмов обработки ошибок в PL/SQL, были замечены некоторые недостатки и неудобства при работе с ними. Для каждой выявленной проблемы были предложены способы ее устранения.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были изучены способы и основные подходы при работе с исключительными ситуациями в языке PL/SQL. Было выяснено, что существующий механизм обладает довольно мощным функционалом, но не лишен своих недостатков, которые могут создать трудности при написании приложений для базы данных, а также могут привести к повышению стоимости разработки конечного продукта. Каждая из замеченных проблем была описана, и были предложены способы её исправления.

В дальнейшей работе планируется реализация пакета с описанным функционалом, предназначенным для улучшения и доработки существующей системы, а также с исправлениями выявленных в данной работе проблем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Том К.* Oracle для профессионалов. — СПБ: ДиаСофтЮП, 2003. — 961 с.

2. *Фейерштейн С. П. Б.* Oracle PL/SQL. Для профессионалов. — 6-е изд. — СПБ: Питер, 2015. — 1024 с.

3. *Хант Э. Т. Д.* Прогаммист-прагматик. Путь от подмастерья к мастеру. — СПБ: Лори, 2007. — 289 с.

4. Handling PL/SQL Errors. — URL: https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/errors.htm#LNPLS007.

5. UTL_FILE Oracle documentation. — URL: https://docs.oracle.com/cd/B28359_01/appdev.111/b28419/u_file.htm#BABGGEDF.