Программирование на PL/SQL: стандарты и рекомендации

Author:	Oleg V. Letaev
Creation Date:	22 августа 2000 г.
Last Updated:	26 сентября 2000 г.
Version:	8.1.6.1

Содержание

ЧАСТЬ

I

Требования к PL/SQL коду

ГЛАВА

1 Форматирование PL/SQL кода

В этой главе устанавливаются стандарты на то, как должен выглядеть PL/SQL код, чтобы повысить его «читабельность» и, как следствие, сопровождаемость. Эти стандарты должны быть зафиксированы в начале разработки любого проекта.

Основы форматирования

Форматирование программы должно отражать её структуру.

Идентичное форматирование SQL-выражений важно ещё и потому, что гарантирует повторное использование разобранных выражений в разделяемом пуле.

Отступы

Используйте отступы в 3 пробела.

Отступы в 3 пробела позволяют достаточно чётко обрисовать логическую структуру кода, сохраняя, в то же время, компактность по горизонтали.

Регистр

Используйте верхний регистр для зарезервированных слов, встроенных функций и предопределённых исключений (например SELECT, DECODE).

Используйте нижний регистр со прописной первой буквой каждого слова для имён объектов схемы, столбцов, параметров, переменных, кроме отдельных функций и процедур (например, EmployeeHistory, User_Tables). При этом аббревиатуры сохраняют верхний регистр (например, DBMS_Job, DBMS_SQL).

Это правило не означает, однако, что при создании объектов схемы следует использовать кавычки вокруг имени объекта, чтобы сохранить его регистр и в словаре данных. Допустимо, что в словаре данных имя объекта схемы будет зафиксировано строчными буквами. В противном случае, вынужденное использование кавычек и в дальнейшем во всех текстах сделает их громоздкими и неудобочитаемыми.

Используйте нижний регистр со прописной первой буквой каждого слова, кроме первого, для имён функций и процедур, как отдельных, так и пакетных, а также для имён методов (например, check_RDBMS_Version, fireEmployee).

Слова в верхнем регистре притягивают взгляд, позволяют легко сфокусироваться и служат как разметка кода, в то время как остальные слова в нижнем или смешаном регистре (имена таблиц, столбцов и т.п.) играют подчинённую роль по отношению к зарезервированным, раскрывают, детализируют их.

Декларации

Группируйте декларации логически, согласно логической связи декларируемых переменных. Группы логически связанных переменных разделяйте пробельной строкой.

Логическое группирование переменных помогает понять программу.

Например,

```
DECLARE
   CompanyName VARCHAR2(30);
   CompanyId INTEGER;

   EmployeeName VARCHAR2(60);
   HireDate DATE;
   TerminationDate DATE;

   MinValue NUMBER;
```

Прагмы, такие как AUTONOMOUS_TRANSACTION, помещайте перед всеми декларациями, чтобы любой, читающий программу, сразу мог идентифицировать её как автономную.

Не используйте выравнивание в декларациях.

Выравнивание, с одной стороны, облегчает «сканирование» используемых типов данных. С другой стороны, тип данных не так важен, как сам идентификатор, а идентификаторы уже выравнены слева. Кроме того, даже один очень длинный идентификатор вынуждает отодвигать тип данных остальных переменных очень далеко от их идентификаторов. Добавление такого очень длинного идентификатора в уже существующий блок кода заставит вернуться назад и добавить 1-2 табуляции к всем существующим декларациям. Вставка комментариев между декларациями ещё более изолирует идентификатор и соответствующий ему тип данных, если используется выравнивание.

Например,

```
DECLARE
   CompanyName       VARCHAR2(30);
   /* Первичный ключ таблицы Companies */
   CompanyId         INTEGER;
```

```
EmployeeName      VARCHAR2(60);
/* Дата найма не должна быть больше текущей */
HireDate          DATE;
TerminationDate   DATE;

MinValue          NUMBER;
```

Отдельные операторы

Располагайте каждый оператор (присвоение, вызов, декларацию) на отдельной строке.

Расположение операторов на отдельных строках облегчает чтение и, что немаловажно, работу строковых отладчиков.

Включайте один пробел перед каждым идентификатором или разделителем внутри оператора.

Например, вместо

```
Total_Sales<Maximum_Sales
```

следует писать

```
Total_Sales < Maximum_Sales,
```

а вместо

```
calcTotals(CompanyId,TotalType,LAST_DAY(EndOfYearDate));
```

следует писать

```
calcTotals (CompanyId, TotalType, LAST_DAY (EndOfYearDate));
```

Не следует вставлять пробел после открывающей и перед закрывающей скобками – выражение будет выглядеть разъединённым. Достаточно пробела после имени функции, процедуры, метода.

Многострочные операторы

Длинный оператор может быть разбит на несколько строк без использования дополнительных символов.

Используйте отступ в 3 пробела, чтобы визуально отделить строки-продолжения от первой строки и, в то же время, логически их объединить.

При вызовах процедур, функций, методов с большим количеством параметров на первой строке размещайте только имя процедуры, функции, метода. Фактические параметры располагайте, начиная со второй строки (с отступом в 3 пробела). Группу логически связанных фактических параметров располагайте на одной строке.

Например,

```
generateCompanyStatistics
   (CompanyId, LastYearDate, RollupType,
    Total, Average, Variance, Budgeted, NextYearPlan);
```

Такое форматирование позволяет визуально отделить имя процедуры от списка параметров и, в то же время, обеспечить компактное размещение параметров.

Старайтесь разбивать длинный оператор так, чтобы было очевидно, что следует продолжение.

Например, строку

```
FOR MonthIndex IN FirstMonth .. LastMonth
```

если она не будет поещаться, разумно разбить после IN, так как IN очевидно требует продолжения

```
FOR MonthIndex IN
   FirstMonth .. LastMonth
LOOP
```

Оператор присвоения разумно разбивать после знака операции, так как очевидно, что следует второй операнд:

```
Q1Sales :=
   Month1Sales +
   Month2Sales +
   Month3Sales;
```

Список разумно разбивать после запятой, так как очевидно, что будет следующий элемент списка.

Комментарии

При написании комментария разработчик пытается объяснить, что должен делать этот фрагмент кода. Если при этом возникают трудности, это может указывать на недостаточное понимание цели самим разработчиком.

Объяняйте в комментариях, ЧТО делает программа, а не КАК она это делает. Обясняйте, какое бизнес-правило Вы пытаетесь реализовать.

Используйте самообъясняющие имена переменных.

Используйте именованную нотацию при вызове процедур/функций. Аналогично, перечисляйте список столбцов в операторе INSERT.

Избегайте компьютерной лексики, например не «Сбросить количество элементов в 0», а «Очистить список». Ещё лучше создать процедуру emptyList, которая будет выполнять эту операцию, и имя которой – самообъясняющее и не нуждается в комментариях.

Избегайте комментариев, которые повторяют то, что очевидно из кода.

Пишите комментарии в формате, который легко вводить и поддерживать. Например, для многострочных комментариев не используйте конструкций типа

```
/*
=====================================================
| Каждому сотруднику, нанятому более 5 лет назад,  |
| выдать премию                                     |
|====================================================
*/
```

или

```
/*
=====================================================
| Каждому сотруднику, нанятому более 5 лет назад,
| выдать премию
=====================================================
*/
```

которые придётся расширять или сокращать при изменении длины комментария. Используйте конструкцию вида

```
/*
|| Каждому сотруднику, нанятому более 5 лет назад,
|| выдать премию
*/
```

Двойная вертикальная черта слева позволяет легко выделить комментарии при сканировании текста.

Однострочные комментарии можно располагать как в конце строки через ‘--‘, так и на отдельной строке через ‘--‘ или в ‘/* */’.

Располагайте комментарии с тем же отступом, что и код, к которому они относятся.

Форматирование SQL-выражений

PL/SQL допускает включение DML-выражений: SELECT, INSERT, UPDATE и DELETE, каждое из которых состоит из нескольких фраз.

Выравнивание справа

Выравнивайте фразы SQL-выражения справа.

Выравнивание фраз SQL-выражения справа позволяет создать визуальную границу между фразами (зарезервированными словами), определяющими структуру SQL-выражения, и остальной частью выражения.

Например,

```
SELECT
  FROM
 WHERE
   AND
    OR
 GROUP BY
HAVING
   AND
```

```
    OR
 ORDER BY

INSERT INTO
     VALUES

INSERT INTO
     SELECT
       FROM
      WHERE

UPDATE
   SET
 WHERE

DELETE
  FROM
 WHERE
```

Во фразах GROUP BY и ORDER BY достаточно выравнять первое слово, которое и является значащим.

Логические соединительные конструкции AND и OR выравниваются под своими фразами WHERE и HAVING.

Если какую-либо из строк приходится разбивать, строки-продолжения располагаются с отступом в 3 пробела по правилам для многострочных операторов.

Разбиение по строкам

Располагайте каждую логическую группу элементов списка SELECT или даже каждый элемент списка SELECT на отдельной строке.

Располагайте каждую таблицу во фразе FROM на отдельной строке.

Располагайте каждое выражение фразы WHERE на отдельной строке.

Располагайте каждое присваивание во фразе SET на отдельной строке.

```
SELECT LastName,
       Co.Name,
       MAX (SH.Salary) BestSalaryEver
  FROM Employee Emp,
       Company Co,
       SalaryHistory SH
 WHERE Emp.CompanyId = Co.CompanyId
   AND Emp.EmployeeId = SH.EmployeeId
   AND Emp.HireDate > ADD_MONTHS (SYSDATE, -60);

UPDATE Employee
   SET HireDate = SYSDATE,
       TerminationDate = NULL
 WHERE DepartmentId = 105;
```

Элиасы

Всегда префиксируйте имя столбца в сложных выражениях элиасом таблицы.

Это позволит избежать сбоев существующих модулей при добавлении столбца с тем же именем к другой таблице.

Используйте значащие 2-4 буквенные элиасы или не используйте их вообще (используйте имя таблицы), если имя таблицы достаточно коротко.

Форматирование управляющих структур

Форматирование управляющих структур имеет наибольшее влияние на читаемость кода.

Правильно используйте отступы, всегда сохраняйте операторы одного «логического уровня» на одном уровне отступа.

Условные выражения

Допустимы два варианта форматирования условных выражений, отличающихся расположением слова THEN:

```
IF <условие>
THEN
   <исполняемый код>
END IF;
```

```
IF <условие> THEN
   <исполняемый код>
END IF;
```

```
IF <условие>
```

```
IF <условие> THEN
```

```
THEN
   <исполняемый код>
ELSE
   <исполняемый код>
END IF;
```

```
   <исполняемый код>
ELSE
   <исполняемый код>
END IF;
```

```
IF <условие>
THEN
   <исполняемый код>

ELSIF <условие>
THEN
   <исполняемый код>

ELSE
   <исполняемый код>
END IF;
```

```
IF <условие> THEN
   <исполняемый код>

ELSIF <условие> THEN
   <исполняемый код>

ELSE
   <исполняемый код>
END IF;
```

Более предпочтительным является первый вариант, который визуально отделяет условие от исполняемого кода и дополнительно подчёркивает группирование строк исполняемого кода.

Обратите внимание на пробельные строки в последней конструкции.

Циклы

Допустимы два варианта форматирования циклов, отличающихся расположением слова LOOP:

```
LOOP
   <исполняемый код>
END LOOP;
```

```
WHILE <условие>
LOOP
   <исполняемый код>
END LOOP;
```

```
WHILE <условие> LOOP
   <исполняемый код>
END LOOP;
```

```
FOR <индекс> IN <список>
LOOP
   <исполняемый код>
END LOOP;
```

```
FOR <индекс> IN <список> LOOP
   <исполняемый код>
END LOOP;
```

Более предпочтительным является первый вариант, который визуально отделяет условие от исполняемого кода и дополнительно подчёркивает группирование строк исполняемого кода. Кроме того, при этом слова LOOP и END LOOP располагаются на одном уровне.

Обработчики исключений

Формат секции обработчиков исключений подобен условному выражению.

```
EXCEPTION
   WHEN <исключение1>
   THEN
      <исполняемый код>

   WHEN <исключение2>
   THEN
      <исполняемый код>

   WHEN OTHERS
   THEN
      <исполняемый код>
END;
```

Располагайте слова WHEN и THEN с отступом в 3 пробела от слова EXCEPTION.

Располагайте исполняемый код с отступом в 3 пробела от имени исключения.

Вставляйте пустую строку перед каждым обработчиком, за исключением первого.

Форматирование PL/SQL блоков

Размещение в файле

Каждый PL/SQL модуль должен находиться в отдельном файле. Это добавляет гибкости при создании и компиляции.

Спецификацию и тела пакета можно располагать как в одном файле, так и в разных.

При размещении спецификации и тела пакета в одном файле удобно открывать их для редактирования. При размещении спецификации и тела пакета в отдельных файлах удобно создавать сначала все спецификации приложения, а затем все тела. При этом все пакеты окажутся VALID. Кроме того, при изменении и перекомпиляции одного из тел без изменения спецификации зависимые объекты останутся VALID. Поэтому, предпочтительнее хранить спецификации и тела пакетов в отдельных файлах.

Храните тексты программных модулей в файлах со следующими расширениями:

Файл	Незакриптованный	Закриптованный
Процедура	.prc	.wpr
Фукнция	.fnc	.wfn
Триггер	.trg	.wtr
Спецификация пакета (если хранится отдельно от тела)	.psp	.wps
Тело пакета (если хранится отдельно от спецификации)	.pbd	.wpb
Пакет (если хранится целиком)	.pkg	.wpk
Спецификация типа (если хранится отдельно от тела)	.tsp	.wts
Тело типа (если хранится отдельно от спецификации)	.tbd	.wtb
Тип (если хранится целиком)	.typ	.wty

Имя файла должно совпадать с именем объекта внутри него.

Имя файла для триггеров должно совпадать с именем таблицы, для которой создаются эти триггеры, т.е. все триггеры для одной таблицы должны храниться в одном файле.

Рекомендуется криптовать все тела пакетов перед дистрибуцией, но оставлять открытыми спецификации, как API системы.

Секции PL/SQL блока

Любой PL/SQL блок включает до четырёх секций:

- Спецификация
- Декларации
- Исполняемый код
- Обработчики исключений

Форматирование кода должно выделять эти секции.

```
FUNCTION companyName (pi_CompanyId IN Company.Id%TYPE)
   RETURN VARCHAR2
IS
   CName Company.Name%TYPE;

BEGIN
   SELECT Name INTO CName FROM Company
    WHERE Id = pi_CompanyId;
   RETURN CName;

EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      RETURN NULL;
END companyName;
```

Располагайте слово IS (AS) на отдельной строке, чтобы визуально отделить спецификацию прогаммного модуля от секции деклараций.

Вставляйте пробельную строку перед секциями исполняемого кода и исключений. Если секции исключений нет, завершающий END можно не отделять от исполняемого кода.

Обязательно включайте после последнего слова END метку (имя модуля).

При форматировании пакетов размещайте все конструкции внутри пакета (как в спецификации, так и в теле) с отступом в 3 пробела от слова PACKAGE.

Размещение деклараций

Разделите все конструкции на программные модули и всё остальное.

Логически сгруппируйте остальные конструкции и располагайте их в следующем порядке внутри каждой логической группы:

- скалярные переменные
- сложные типы данных (записи, таблицы)
- курсоры
- исключения

Затем располагайте логически сгруппированные частные программные модули. И наконец, логически сгруппированные общие программные модули.

Логические группы разделяйте пробельными строками.

Комментирование

Используйте header'ы (стандартные блоки комментариев) для всех типов программных модулей.

Header должен содержать всю необходимую, но минимальную информацию о назначении и истории модуля. Шаблоны header'ов могут быть созданы в средствах разработки типа PL/SQL Developer, Platinum SQL*Station и т.п.

Располагайте header перед словом IS (AS) для отдельных процедур, функций, спецификаций и тел пакетов. Располагайте header перед словом PROCEDURE (FUNCTION) для пакетных процедур и функций.

Декларации

Пишите комментарий к каждой декларации и/или логической группе деклараций. Размещайте такой комментарий перед декларацией и/или группой деклараций на отдельной строке. Комментарий к группе деклараций выполняйте в виде banner'а вида

```
/*-------------------- <заголовок> --------------------*/
```

с пробельной строкой после него.

Разделяйте откомментированные декларации/группы деклараций пробельными строками.

Специальные banner'ы должны быть перед группой частных и перед группой общих модулей в теле пакета.

Процедура/функция

Рекомендуемый header для отдельных процедур и функций:

```
CREATE OR REPLACE FUNCTION <имя> (<параметры>)
   RETURN <тип_данных>
/*
|| Автор:
||
|| Назначение:
||
|| Ограничения:
||
|| Параметры:
||    <параметр - описание>
||
|| Зависимости:
||    <таблицы, другие модули, к которым обращается этот>
||
|| Исключения:
||    <исключения, явно генерируемые модулем>
||
|| История:
||    <версия, когда, кем, что изменено>
*/
IS
...
BEGIN
...
END <имя>;
```

Для пакетных процедур и функций header должен отличаться отсутствием упоминания автора и истории, так как и то, и другое ведётся в header'е пакета. Header'ы пакетных

процедур и функций размещаются в теле пакета вместе с реализацией процедуры/функции.

Пакет

Рекомендуемый header для спецификаций пакетов:

```
CREATE OR REPLACE PACKAGE <имя>
/*
|| Автор:
||
|| Назначение:
||
|| Ограничения:
||
|| История:
||    <версия, когда, кем, что изменено>
*/
AS
...
END <имя>;
```

В Ограничениях здесь должны фиксироваться только факты, существенные для всего пакета в целом. В Историю вносятся только изменения, затронувшие спецификацию.

Для тела пакета структура header'а остаётся неизменной, но в Историю вносятся только изменения, затронувшие тело.

Комплексный пример

Для иллюстрации вышеизложенного приводятся примеры процедуры, спецификации и тела пакета.

Процедура

```
CREATE OR REPLACE PROCEDURE checkRDBMSVersion
   (pi_Min IN VARCHAR2,
    pi_Max IN VARCHAR2)
/*
|| Автор: Oleg V. Letaev, Leaves, Inc.
||
|| Назначение: Проверяет номер версии СУБД
||
|| Ограничения:
||
|| Параметры:
||    pi_Min - наименьший номер поддерживаемой версии
||    pi_Max - наибольший номер поддерживаемой версии
||
|| Зависимости:
||    DBMS_Application_Info
||    V$Version
||    Exceptions
||
|| Исключения:
||    UnsupportedRDBMSVersion
||
|| История:
||    <версия, когда, кем, что изменено>
||    1.0    30.08.99   OVL   Production Release
||    1.0    12.08.99   OVL   Создана
*/
IS
   v_RDBMSVersion VARCHAR2(5);

BEGIN
   DBMS_Application_Info.Set_Module
      ('checkRDBMSVersion', 'Enter');

   SELECT SUBSTR(Banner,24,5)
     INTO v_RDBMSVersion
     FROM V$Version
    WHERE Banner LIKE 'Oracle7 Server Release%';

   IF v_RDBMSVersion NOT BETWEEN pi_Min AND pi_Max
   THEN
```

```
      RAISE Exceptions.UnsupportedRDBMSVersion;
   END IF;

EXCEPTION
   WHEN NO_DATA_FOUND
     OR TOO_MANY_ROWS
     OR Exceptions.UnsupportedRDBMSVersion
   THEN
      DBMS_Application_Info.Set_Action('Exception');
      RAISE_APPLICATION_ERROR
         (Exceptions.Unsupported_RDBMS_Version_Code,
          Exceptions.Unsupported_RDBMS_Version_Mesg);
END checkRDBMSVersion;
```

Спецификация пакета

```
CREATE OR REPLACE PACKAGE PLVLog
/*
|| Автор: Steven Feuerstein
||
|| Назначение: This is a part of the PL/Vision Professional.
||    Copyright (C) 1996-1999 RevealNet, Inc.
||    All rights reserved.
||
|| Ограничения:
||
|| История:
||    1.1     6/99   SEF   fastfile solution avoids PLVrep
||    1.1     3/98   SEF   Q298 Set create_by string
||    1.1    11/97   SEF   Add back get_line procedure, add
||                            columsn to todbtab.
||    1.0    11/97   SEF   Product release
||    1.0     5/97   SEF   Change default delimiter to |.
||    1.0     4/97   SEF   Remove reference to PLVrep in
||                            specifications.
||    1.0     3/97   SEF   Add backward compatibility to non-2.3
||                            PLVlog.
||    1.0     3/97   SEF   Support for incremental log access
||                            with file.
*/
IS
   /* Разделитель по умолчанию */
   c_Delim CONSTANT CHAR(1) := '|';

   /* Смещение по умолчанию */
   c_Offset CONSTANT INTEGER := 2;

   /*-------- Имена объектов-журналов по умолчанию --------*/

   c_File CONSTANT VARCHAR2(7) := 'plv.log';
   c_Pipe CONSTANT VARCHAR2(7) := 'plvlog';
   c_DBTab CONSTANT VARCHAR2(7) := 'plv_log';

   /*-------- Типы репозитория --------*/

   /* Не установлен */
   c_NotSet CONSTANT CHAR(2) := 'NS';

   /* Файл */
   c_FileType CONSTANT CHAR(1) := 'F';

   /* Pipe */
   c_PipeType CONSTANT CHAR(1) := 'P';

   /* Экран */
   c_ScreenType CONSTANT CHAR(2) := 'SO';

   /* Таблица БД */
   c_DBTabType CONSTANT CHAR(2) := 'DB';

   /* Index-by таблица */
   c_IBTabType CONSTANT CHAR(2) := 'PS';
```

```
/* Index-by таблица */
c_PSTabType CONSTANT CHAR(2) := 'PS';

/* Строка */
c_StringType CONSTANT CHAR(1) := 'S';

/* Alert */
c_AlertType CONSTANT CHAR(1) := 'A';

/*-------- Типы операций --------*/

c_Read CONSTANT CHAR(1) := 'R';
c_Write CONSTANT CHAR(1) := 'W';
c_Append CONSTANT CHAR(1) := 'A';
c_Both CONSTANT CHAR(1) := 'B';

/*-------- Опции поведения ROLLBACK TO SAVEPOINT --------*/

c_NoAction CONSTANT PLV.Identifier$typ := '*NO ROLLBACK*';
c_None CONSTANT PLV.Identifier$typ := '*FULL*';
c_Default CONSTANT PLV.Identifier$typ := '*DEFAULT*';
c_Last CONSTANT PLV.Identifier$typ := '*PLVRB-LAST*';

/*-------- Переключение журнализации --------*/

PROCEDURE turnOn;
PROCEDURE turnOff;
FUNCTION logging RETURN BOOLEAN;

/*-------- Управление репозиторием через PLVRep --------*/

/* Возвращает тип репозитория */
FUNCTION logType RETURN VARCHAR2;

/* Устанавливает тип репозитория - файл */
PROCEDURE toFile (
   pi_File IN VARCHAR2 := c_File,
   pi_FileMode IN VARCHAR2 := 'W',
   pi_Dir IN VARCHAR2 := NULL,
   pi_FlushAfter IN PLS_INTEGER := 100);

/* Устанавливает тип репозитория – таблица БД */
PROCEDURE toDBTab (
    pi_Tab IN VARCHAR2 := c_DBTab,
    pi_ContextCol IN VARCHAR2 := 'context',
    pi_CodeCol IN VARCHAR2 := 'code',
    pi_TextCol IN VARCHAR2 := 'text',
    pi_CreateTSCol IN VARCHAR2 :=  'create_ts',
    pi_CreateByCol IN VARCHAR2 := 'create_by',
    pi_Sch IN VARCHAR2 := NULL);

/* Устанавливает тип репозитория - pipe */
PROCEDURE toPipe (pi_Pipe IN VARCHAR2 := c_Pipe);

/* Устанавливает тип репозитория – index-by таблица */
PROCEDURE toPSTab;

/* Устанавливает тип репозитория - экран */
PROCEDURE toScreen;

/* Очищает журнал */
PROCEDURE clear;

/*-------- Управление журналом --------*/

PROCEDURE close;

PROCEDURE pipe2File
   (pi_Pipe IN VARCHAR2 := c_Pipe,
    pi_File IN VARCHAR2 := c_File);
PROCEDURE pipe2DBTab
   (pi_Pipe IN VARCHAR2 := c_Pipe,
```

```
        pi_Tab IN VARCHAR2 := c_DBTab);

   PROCEDURE pstab2File (pi_File IN VARCHAR2 := c_File);
   PROCEDURE pstab2DBTab (pi_Tab IN VARCHAR2 := c_DBTab);

   PROCEDURE display
      (pi_Header IN VARCHAR2 := 'PL/Vision Log');

   FUNCTION entry (pi_Row IN INTEGER) RETURN PLV_log%ROWTYPE;

   /*-------- Запись/чтение строки в/из журнала --------*/

   PROCEDURE putLine
      (pi_Context IN VARCHAR2,
       pi_Code IN INTEGER,
       pi_Sring IN VARCHAR2 := NULL,
       pi_CreateBy IN VARCHAR2 := USER,
       pi_RBTo IN VARCHAR2 := c_Default,
       pi_Override IN BOOLEAN := FALSE);

   PROCEDURE putLine
      (pi_String IN VARCHAR2,
       pi_RBTo IN VARCHAR2 := c_Default,
       pi_Override IN BOOLEAN := FALSE);

   PROCEDURE getLine
      (pi_Row IN INTEGER,
       po_Context OUT VARCHAR2,
       po_Code OUT INTEGER,
       po_String OUT VARCHAR2,
       po_CreateBy OUT VARCHAR2,
       po_CreateTS OUT DATE);

END PLVLog;
/
```

Тело пакета

```
CREATE OR REPLACE PACKAGE BODY PLVLog
/*
|| Автор: Steven Feuerstein
||
|| Назначение: This is a part of the PL/Vision Professional.
||    Copyright (C) 1996-1999 RevealNet, Inc.
||    All rights reserved.
||
|| Ограничения:
||
|| История:
||    1.2   07/99    SEF   Move trclog to PLVlog and out of
||                            PLVtrc.
||    1.2   07/02/99 TVG   M1: Moving from PLVrep to PLV_log_io
||    1.2    6/99    SEF   fastfile solution avoids PLVrep
||    1.1    4/98    SEF   Use INTEGER
||    1.1    3/98    SEF   Q298 Set create_by string
||    1.1   1/15/98  SEF   Q198 Add plvconfig calls
||    1.1   12/97   SEF   Add code for trial version.
||    1.1   11/97   SEF   Add back get_line procedure,
||                           add columns to todbtab.
||    1.1   9/9/97  SEF   Switch to PLVrep.closeif
||    1.1   8/12/97 SEF   Use PLVrep.tabsegs for
||                           replacement table and
|                            reenable set_dbtab.
||    1.1   5/23/97 SEF   Make sure rep is open in pstab2file,
||                            pstab2dbtab.
||    1.0   11/97   SEF   Product release
||    1.0    4/97   SEF   Replace c_logtable refs with c_DBTab.
||    1.0    4/97   SEF   Replace use of PLVRep.tabSegs with
||                           explicit seg calls.
||    1.0    4/97   SEF   Remove reference to PLVRep in
||                           specifications.
||    1.0    3/97   SEF   Add backward compatibility to
```

```
||                          non-2.3 PLVlog and
||                          avoid infinite loops with calls to
||                          PLVExc.recNStop.
||    1.0    3/97   SEF   Add variable file mode for
||                          incremental reads
*/
IS

   c_Pkg CONSTANT PLV.Identifier$typ := 'plvlog';

   g_Delim VARCHAR2(10) := c_Delim;

   g_FileMode VARCHAR2(2) := c_Write;

   /* По умолчанию журнализация включена */
   g_Logging BOOLEAN := TRUE;

   /* По умолчанию откат выключен */
   g_Rollback BOOLEAN := FALSE;

   g_Savepoint PLV.Identifier$typ := c_PLVLogSP;
   g_RBTo PLV.Identifier$typ := c_None;

   TYPE LogEntryRec$typ IS RECORD
      (Context PLV_Log.Context%TYPE,
       Code PLV_Log.Code%TYPE,
       Text PLV_Log.Text%TYPE,
       CreateTS PLV_Log.Create_TS%TYPE,
       CreateBy PLV_Log.Create_By%TYPE);

   TYPE DBTabRec$typ IS RECORD
      (TableName PLV.Identifier$typ := 'PLV_log',
       ContextCol PLV.Identifier$typ := 'context',
       CodeCol PLV.Identifier$typ := 'code',
       TextCol PLV.Identifier$typ := 'text',
       CreateTSCol PLV.Identifier$typ := 'create_ts',
       CreateByCol PLV.Identifier$typ := 'create_by',
       DML_Insert PLV.DBMaxVC2$typ :=
          'INSERT INTO PLV_Log ' ||
          '   (Context, Code, Text, CreateTS, CreateBy) ' ||
          'VALUES (:Context, :Code, :Text, ' ||
          '        :CreateTS, :Create_By)');
   g_LogStruc DBTabRec$typ;

   /*------------------ Частные модули ------------------*/

   /*
   || Назначение: Установить тип журнала - таблица БД,
   ||    если не задано иное
   ||
   || Ограничения:
   ||
   || Параметры:
   ||
   || Зависимости:
   ||
   || Исключения:
   */
   PROCEDURE initRep
   IS

   BEGIN
      IF g_LogType = c_NotSet
      THEN
         ToDBTab;
      END IF;
   END initRep;

   /*------------------ Общие модули ------------------*/

   /*-------- Переключение журнализации --------*/

   /*
```

```
|| Назначение: Включает журнализацию
||
|| Ограничения:
||
|| Параметры:
||
|| Зависимости:
||
|| Исключения:
*/
PROCEDURE turnOn
IS

BEGIN
   PLVConfig.enforceTrial;
   v_Logging := TRUE;
   PLVConfig.change (c_Pkg, 'turnOn', v_Logging);
END turnOn;

/*
|| Назначение: Выключает журнализацию
||
|| Ограничения:
||
|| Параметры:
||
|| Зависимости:
||
|| Исключения:
*/
PROCEDURE turnOff
IS

BEGIN
   PLVConfig.enforceTrial;
   v_Logging := FALSE;
   PLVConfig.change (c_Pkg, 'turnOn', v_Logging);
END turnOff;

/*
|| Назначение: Возвращает состояние журнализации
||
|| Ограничения:
||
|| Параметры:
||
|| Зависимости:
||
|| Исключения:
*/
FUNCTION logging RETURN BOOLEAN
IS

BEGIN
   RETURN v_Logging;
END logging;

/*-------- Управление репозиторием через PLVRep --------*/

/*
|| Назначение: Устанавливает тип репозитория - файл
||
|| Ограничения:
||
|| Параметры:
||    pi_File - имя файла
||    pi_FileMode - режим доступа
||    pi_Dir - путь к файлу
||    pi_FlushAfter -
||
|| Зависимости:
||
|| Исключения:
```

```
*/
PROCEDURE toFile
   (pi_File IN VARCHAR2 := c_File,
    pi_FileMode IN VARCHAR2 := 'W',
    pi_Dir IN VARCHAR2 := NULL,
    pi_FlushAfter IN PLS_INTEGER := 100)
IS
   v_Dir PLV.MaxVC2$typ := NVL (pi_Dir, PLVFile.Dir);

BEGIN
   PLV.assert
      ((v_Dir IS NOT NULL),
       'You must specify a directory when you call ' ||
          'PLVLog.toFile or through a prior call to ' ||
          'PLVfile.set_dir.');

   PLVLogIO.useFile
      (v_Dir,
       pi_File,
       pi_FileMode,
       pi_FlushAfter);

   v_FileMode := UPPER (NVL (pi_FileMode, c_Write));
END toFile;

/*
|| Назначение: Возвращает тип репозитория
||
|| Ограничения:
||
|| Параметры:
||
|| Зависимости:
||
|| Исключения:
*/
FUNCTION logType RETURN VARCHAR2
IS
   v_Settings PLVLogIO.SettingsRec$typ;

BEGIN
   v_Settings := PLVLogIO.Settings;
   RETURN (NVL (v_Settings.UseType, c_NotSet));
END logType;

/*-------- Управление журналом --------*/

/*
|| Назначение: Закрывает журнал
||
|| Ограничения:
||
|| Параметры:
||
|| Зависимости:
||
|| Исключения:
*/
PROCEDURE close
IS

BEGIN
   PLVLogIO.close;
END close;

/*
|| Назначение
||
|| Ограничения:
||
|| Параметры:
||    pi_Pipe - имя pipe
||    pi_File - имя файла
```

```
   ||
   || Зависимости:
   ||
   || Исключения:
   */
   PROCEDURE pipe2File
      (pi_Pipe IN VARCHAR2 := c_Pipe,
       pi_File IN VARCHAR2 := c_File)
   IS

   BEGIN
     PLVLogIO.pipe2File (pi_Pipe, pi_File);
   END pipe2File;

   /*
   || Назначение
   ||
   || Ограничения:
   ||
   || Параметры:
   ||    pi_Pipe - имя pipe
   ||    pi_Tab - имя таблицы
   ||
   || Зависимости:
   ||
   || Исключения:
   */
   PROCEDURE pipe2DBTab
      (pi_Pipe IN VARCHAR2 := c_Pipe,
       pi_Tab IN VARCHAR2 := c_DBTab)
   IS

   BEGIN
      PLVLogIO.pipe2DBTab (pi_Pipe, pi_Tab);
   END pipe2DBTab;

   /*
   || Назначение: Показывает журнал
   ||
   || Ограничения:
   ||
   || Параметры:
   ||
   || Зависимости:
   ||
   || Исключения:
   */
   PROCEDURE display (pi_Header IN VARCHAR2 := 'PL/Vision Log')
   IS

   BEGIN
      PLVLogIO.display (pi_Header);
   END display;

BEGIN
   initRep;
END PLVLog;
/
```

Программные средства форматирования

Средство разработки PL/SQL Developer

Средство разработки PL/SQL Developer компании Allround Automations v3.0.3 помимо прочих полезных функций включает возможность создания шаблонов различных программных конструкций, что помогает поддерживать единый стиль форматирования и комментирования программных модулей.

Так например, можно создавать шаблоны как для управляющих конструкций, так и для SQL-операторов, и для программных модулей с header'ами комментариев.

PL/SQL Developer поддерживает хранение шаблонов на разделяемом диске, тем самым обеспечивая единый набор шаблонов для всех разработчиков проекта.

Утилита PL/Formatter

Утилита PL/Formatter компании RevealNet v3.1.0\3.1.2.1 предназначена для форматирования и синтаксического контроля текста PL/SQL модулей.

PL/Formatter может быть настроен под различные стили форматирования. К сожалению, файл настроек не может быть сохранён на разделяемом диске, поэтому выходом, гарантирующим единый домен настроек форматирования для всех разработчиков проекта, является инсталляция самого продукта на разделяемый диск.

Утилита PL/Formatter подключается при помощи Plug-In интерфейса к PL/SQL Developer компании Allround Automations v3.0.3, что позволяет использовать эти средства в качестве единой среды.

ГЛАВА

2 Стандарты именования объектов

В этой главе приведены несколько стандартов именования объектов, каждый из которых предлагает согласованную и удобочитаемую структуру имён объектов. На начальной стадии проекта должен быть выбран один из предлагаемых стандартов в соответствии с предпочтениями и привычками разработчиков.

Замечания, общие для всех стандартов

Имена объектов схемы могут иметь префикс вида

```
<ApplicationCode>_,
```

где ApplicationCode – код приложения из 2-4 символов, которому принадлежат эти объекты. Такое префиксирование рекомендуется для баз данных, содержащих несколько приложений с глобальными синонимами.

По поводу регистров в именах объектов см. Регистр.

Стандарт 1 (стандарт с подчёркиваниями)

Общие принципы

В сложных именах объектов, состоящих из нескольких слов, слова разделяются символом подчёркивания – ‘_’.

Для удобства выборки из User_Objects всех объектов, связанных с данным, по условию вида

```
WHERE Name LIKE ‘OBJECT_NAME%’
```

все расширения имени объекта добавляются суффиксами (справа). Для объектов схемы суффикс отделяется от собственно имени символом ‘$’ и пишется строчными буквами, за исключением типов, у которых суффикс двойной, и первая часть отделяется от собственно имени символом ‘_’ и пишется по правилам регистра для имён. Для атрибутов типов, столбцов таблиц, переменных и параметров программных модулей суффикс отделяется от собственно имени символом ‘_’ и пишется по правилам регистра для имён.

Типы

Подразумеваются как объектные типы, так и типы внутри PL/SQL модулей.

Для типов-массивов, вложенных таблиц и index-by таблиц имя типа задаётся во множественном числе. В остальных случаях – в единственном числе.

Объект	Правило именования	Пример использования
Тип	`<Type_Name>$typ`	`Line_Item$typ`
Тип-запись	`<Type_Name>_Rec$typ`	`Line_Item_Rec$typ`
Тип-курсорная переменная	`<Type_Name>_CurV$typ`	`Line_Item_CurV$typ`
Тип-курсор	`<Type_Name>_Cur$typ`	`Line_Item_Cur$typ`
Тип-массив	`<Type_Name>_ListV$typ`	`Line_Items_ListV$typ`
Тип-вложенная таблица	`<Type_Name>_ListN$typ`	`Line_Items_ListN$typ`
Тип-index-by таблица	`<Type_Name>_ListT$typ`	`Line_Items_ListT$typ`

Атрибуты типов

Для атрибутов-массивов и вложенных таблиц имя атрибута задаётся во множественном числе. В остальных случаях – в единственном числе.

Объект	Правило именования	Пример использования
Скалярный атрибут	`<Attribute_Name>`	`Street_Address`
Атрибут-объект	`<Attribute_Name>`	`Street_Address`
Атрибут-массив	`<Attribute_Name>_ListV`	`Street_Addresses_ListV`
Атрибут-вложенная таблица	`<Attribute_Name>_ListN`	`Street_Addresses_ListN`
Атрибут-указатель	`<Attribute_Name>_Ref`	`Street_Address_Ref`

Таблицы

Обязательно определение краткого имени таблицы (alias), которое состоит ровно из трех символов.

Для получения краткого имени таблицы можно применять технику CDM, описанную ниже.

Если имя сущности состоит из нескольких слов, используются первые символы первого и второго слов и последний символ последнего слова. Если имя сущности состоит из одного слова из нескольких слогов, используются первые символы первого и второго слогов и последний символ. Если имя сущности состоит из одного слова, используются два первых и последний символ. Если полученное имя уже задействовано, используются два последних символа вместо одного.

Однако, эта алгоритмическая техника не всегда приводит к понятному результату. Поэтому предпочтительнее выбирать краткое имя, исходя из общепринятых сокращений и человеческой логики.

Имя таблицы для чисто реляционных схем – всегда во множественном числе, для объектно-реляционных схем – во множественном числе, только если необходимо подчеркнуть отношение один-ко-многим.

Объект	Правило именования	Пример использования
Реляционная таблица	`<Table_Name>`	`Line_Items`
Объектная таблица	`<Table_Name>$obj`	`Line_Item$obj`
Таблица хранения	`<Table_Name>$ntab`	`Line_Items$ntab`
Индексная таблица	`<Table_Name>$iot`	`Line_Items$iot`

Столбцы таблиц

Для столбцов-массивов и вложенных таблиц имя столбца задаётся во множественном числе. В остальных случаях – в единственном числе.

Для столбцов-внешних ключей используется либо полное имя в единственном числе, либо краткое имя таблицы, на которую ссылается внешний ключ. Допустимо вместо этого использование другого имени, если получается несколько столбцов с одинаковыми именами (несколько внешних ключей между двумя таблицами), или если другое имя точнее отражает смысл столбца. Если присоединённая таблица из другого приложения, её краткое имя может префиксироваться кодом приложения.

Объект	Правило именования	Пример использования
Столбец	`<Column_Name>`	`Street_Name`
Столбец-первичный ключ	`Id`	`Id`

Столбец-внешний ключ	`<Table_Name>_Id`
	или
	`<TableAlias>_Id`
		`Product_Id`
		или
		`Pdt_Id`

Столбец-массив	`<Column_Name>_ListV`	`Street_Addresses_ListV`
Столбец-вложенная таблица	`<Column_Name>_ListN`	`Street_Addresses_ListN`
Столбец-указатель	`<Column_Name>_Ref`	`Street_Address_Ref`

Ограничения целостности

Для внешних ключей, если присоединённая таблица из другого приложения, её краткое имя может префиксироваться кодом приложения.

Если внешних ключей между одними и теми же таблицами несколько, перед $fk через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице с внешним ключом.

Если уникальных ключей в одной таблице несколько, перед $uk через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице с уникальным ключом.

Если CHECK-ограничений целостности в одной таблице несколько, перед $ck через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице с ограничением целостности.

Объект	Правило именования	Пример использования

Первичный ключ	`<Table_Name>$pk`
	или
	`<TableAlias>$pk`
		`Location$pk`
		или
		`Loc$pk`

Внешний ключ	`<Table_Name>_<RefTableAlias>$fk`
	или
		`Projects_Loc$fk`
		или
		`Prs_Loc$fk`

	<TableAlias>_<RefTableAlias>$fk	
Уникальный ключ	<Table_Name>$uk или <TableAlias>$uk	Location$uk или Loc$uk
CHECK-ограничение целостности	<Table_Name>$ck или <TableAlias>$ck	Location$ck или Loc$ck
NOT NULL (если он явно именуется)	<Table_Name>_<Column_Name>$nn или <TableAlias>_<Column_Name>$nn	Location_Name$nn или Loc_Name$nn
WITH CHECK OPTION	<View_Name>$co	Line_Item$co

Представления

Имя представления и материализованного представления – всегда во множественном числе, имя объектного представления – во множественном числе, только если необходимо подчеркнуть отношение один-ко-многим.

Столбцы представлений имеют те же имена, что и соответствующие столбцы в базовой таблице. Для столбцов, не имеющих базовых, используются те же правила, что и для столбцов таблиц.

Объект	Правило именования	Пример использования
Представления	<View_Name>$vi	Line_Items$vi
Объектные представления	<View_Name>$ovi	Line_Item$ovi
Материализованные представления	<View_Name>$mvi	Line_Items$mvi

Последовательности

Если последовательностей в одной таблице несколько, перед $seq через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице, на который создана последовательность.

Объект	Правило именования	Пример использования
Для одного столбца	<Table_Name>$seq или <TableAlias>$seq	Location$seq или Loc$seq
Для нескольких столбцов	<Sequence_Name>$seq	Transaction_Id$seq

Индексы

Имена индексов по первичным, внешним и уникальным ключам совпадают с именами соответствующих ограничений целостности.

Объект	Правило именования	Пример использования
Отальные B*tree индексы	<Table_Name>_<Column_Names>$i или <TableAlias>_<Column_Names>$i	Employee_Emp_No$I или Emp_Emp_No$i
Битовые индексы	<Table_Name>_<Column_Names>$bmi или <TableAlias>_<Column_Names>$bmi	Employee_Hair_Color$bmi или Emp_Hair_Color$bmi
Кластерные индексы	<Table_Name>_<Column_Names>$ci или <TableAlias>_<Column_Names>$ci	Employee_Emp_No$ci или Emp_Emp_No$ci

Кластеры

Объект	Правило именования	Пример использования
Индексный кластер	<Cluster_Name>$iclu	Emp_Dept$iclu
Хэш-кластер	<Cluster_Name>$hclu	Emp_Dept$hclu

Роли

Объект	Правило именования	Пример использования
Роль	<Role_Name>$r	Advanced_User$r

PL/SQL модули

Имя программного модуля должно описывать его функцию. При этом имя процедуры

должны быть в форме глагол_Существительное, поскольку процедура выполняет действие. Имя функции должно быть именем существительным, поскольку процедура возвращает значение (допустим вариант is_Существительное).

Объект	Правило именования	Пример использования
Пакеты	<Package_Name>	Ins_Emp
Отдельные функции и процедуры	<procedure_Name>	ins_Emp
Процедуры и функции, включенные в пакет	<procedure_Name>	ins_Emp
Методы	<method_Name>	get_Name

Переменные и параметры

Для переменных, констант и формальных параметров в дополнение к суффиксной форме используется и префиксная.

Добавление префиксов к именам параметров и переменных позволяет отличить их от столбцов таблиц и друг от друга.

Префиксы

Объект	Правило именования	Пример использования
Формальные параметры	p<mode>_<Parameter_Name>	pi_Airport_Code mode может принимать одно из следующих значений: i – IN, o – OUT, io – INOUT.
Глобальные общие переменные пакетов	<Variable_Name>	Airport_Code
Глобальные частные переменные пакетов	g_<Variable_Name>	g_Airport_Code
Локальные переменные процедур	v_<Variable_Name>	v_Airport_Code
Глобальные и локальные константы	c_<Constant_Name>	c_String_Length

Суффиксы

Для массивов, вложенных таблиц и index-by таблиц имя задаётся во множественном числе. В остальных случаях – в единственном числе.

Объект	Правило именования	Пример использования
Записи	<Variable_Name>_Rec	List_Item_Rec
Курсорные переменные	<Variable_Name>_CurV	List_Item_CurV
Курсоры	<Variable_Name>_Cur	List_Item_Cur
Массивы	<Variable_Name>_ListV	List_Items_ListV
Вложенные таблицы	<Variable_Name>_ListN	List_Items_ListN
Index-by таблицы	<Variable_Name>_ListT	List_Items_ListT

Триггеры

Если триггеров с одинаковыми именами несколько, можно добавлять порядковый номер.

Объект	Правило именования	Пример использования
DML-триггеры	<Table_Name>${b\|a\|io}[i][u][d]{r\|s} или <TableAlias>${b|a|io}[i][u][d][{r|s}] b|a|io означает BEFORE | AFTER | INSTEAD OF, [i][u][d] означает INSERT, UPDATE, DELETE r|s означает Row level | Statement level	Projects$iod или Prs$bis
Триггеры уровня БД	Database${b|a}{st|sh|ln|lf|sr} b|a означает BEFORE | AFTER, st|sh|ln|lf|sr означает STARTUP, SHUTDOWN,	Database$bst

	LOGON, LOGOFF, SERVERERROR	
Триггеры уровня схемы	Schema${b|a}{ln|lf|sr|c|a|d} b|a означает BEFORE | AFTER, ln|lf|sr|c|a|d означает LOGON, LOGOFF, SERVERERROR, CREATE, ALTER, DROP	Schema$bln

Стандарт 2 (стандарт с слитного написания)

Общие принципы

В сложных именах объектов, состоящих из нескольких слов, слова пишутся слитно.

Для удобства выборки из User_Objects всех объектов, связанных с данным, по условию вида

```
WHERE Name LIKE 'OBJECTNAME%'
```

все расширения имени объекта добавляются суффиксами (справа). Для объектов схемы суффикс отделяется от собственно имени символом '$' и пишется строчными буквами, за исключением типов, у которых суффикс двойной, и первая часть не отделяется от собственно имени и пишется по правилам регистра для имён. Для атрибутов типов, столбцов таблиц, переменных и параметров программных модулей суффикс не отделяется от собственно имени символом и пишется по правилам регистра для имён.

Вместо символа '$' во всех случаях может использоваться символ подчёркивания '_'.

Типы

Подразумеваются как объектные типы, так и типы внутри PL/SQL модулей.

Для типов-массивов, вложенных таблиц и index-by таблиц имя типа задаётся во множественном числе. В остальных случаях – в единственном числе.

Объект	Правило именования	Пример использования
Тип	<TypeName>$typ	LineItem$typ
Тип-запись	<TypeName>Rec$typ	LineItemRec$typ
Тип-курсорная переменная	<TypeName>CurV$typ	LineItemCurV$typ
Тип-курсор	<TypeName>Cur$typ	LineItemCur$typ
Тип-массив	<TypeName>ListV$typ	LineItemsListV$typ
Тип-вложенная таблица	<TypeName>ListN$typ	LineItemsListN$typ
Тип-index-by таблица	<TypeName>ListT$typ	LineItemsListT$typ

Атрибуты типов

Для атрибутов-массивов и вложенных таблиц имя атрибута задаётся во множественном числе. В остальных случаях – в единственном числе.

Объект	Правило именования	Пример использования
Скалярный атрибут	<AttributeName>	StreetAddress
Атрибут-объект	<AttributeName>	StreetAddress
Атрибут-массив	<AttributeName>ListV	StreetAddressesListV
Атрибут-вложенная таблица	<AttributeName>ListN	StreetAddressesListN
Атрибут-указатель	<AttributeName>Ref	StreetAddressRef

Таблицы

Обязательно определение краткого имени таблицы (alias), которое состоит ровно из трех символов.

Для получения краткого имени таблицы можно применять технику CDM, описанную ниже.

Если имя сущности состоит из нескольких слов, используются первые символы первого и второго слов и последний символ последнего слова. Если имя сущности состоит из одного слова из нескольких слогов, используются первые символы первого и второго слогов и последний символ. Если имя сущности состоит из одного слова, используются два первых и последний символ. Если полученное имя уже задействовано, используются два последних символа вместо одного.

Однако, эта алгоритмическая техника не всегда приводит к понятному результату. Поэтому предпочтительнее выбирать краткое имя, исходя из общепринятых сокращений и человеческой логики.

Имя таблицы для чисто реляционных схем – всегда во множественном числе, для объектно-реляционных схем – во множественном числе, только если необходимо подчеркнуть отношение один-ко-многим.

Объект	Правило именования	Пример использования
Реляционная таблица	`<TableName>`	`LineItems`
Объектная таблица	`<TableName>$obj`	`LineItem$obj`
Таблица хранения	`<TableName>$ntab`	`LineItems$ntab`
Индексная таблица	`<TableName>$iot`	`LineItems$iot`

Столбцы таблиц

Для столбцов-массивов и вложенных таблиц имя столбца задаётся во множественном числе. В остальных случаях – в единственном числе.

Для столбцов-внешних ключей используется либо полное имя в единственном числе, либо краткое имя таблицы, на которую ссылается внешний ключ. Допустимо вместо этого использование другого имени, если получается несколько столбцов с одинаковыми именами (несколько внешних ключей между двумя таблицами), или если другое имя точнее отражает смысл столбца. Если присоединённая таблица из другого приложения, её краткое имя может префиксироваться кодом приложения.

Объект	Правило именования	Пример использования
Столбец	`<ColumnName>`	`StreetName`
Столбец-первичный ключ	`Id`	`Id`

Столбец-внешний ключ	`<TableName>Id`
	или
	`<TableAlias>Id`
		`ProductId`
		или
		`Pdt_Id`

Столбец-массив	`<ColumnName>ListV`	`StreetAddressesListV`
Столбец-вложенная таблица	`<ColumnName>ListN`	`StreetAddressesListN`
Столбец-указатель	`<ColumnName>Ref`	`StreetAddressRef`

Ограничения целостности

Для внешних ключей, если присоединённая таблица из другого приложения, её краткое имя может префиксироваться кодом приложения.

Если внешних ключей между одними и теми же таблицами несколько, перед $fk через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице с внешним ключом.

Если уникальных ключей в одной таблице несколько, перед $uk через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице с уникальным ключом.

Если CHECK-ограничений целостности в одной таблице несколько, перед $ck через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице с ограничением целостности.

Объект	Правило именования	Пример использования

Первичный ключ	`<TableName>$pk`
	или
	`<TableAlias>$pk`
		`Location$pk`
		или
		`Loc$pk`

Внешний ключ	`<TableName>_<RefTableAlias>$fk`
	или
	`<TableAlias>_<RefTableAlias>$fk`
		`Projects_Loc$fk`
		или
		`Prs_Loc$fk`

Уникальный ключ	`<TableName>$uk`
	или
	`<TableAlias>$uk`
		`Location$uk`
		или
		`Loc$uk`

CHECK-ограничение	`<TableName>$ck`	`Location$ck`

целостности	или <TableAlias>$ck	или Loc$ck
NOT NULL (если он явно именуется)	<TableName>_<ColumnName>$nn или <TableAlias>_<ColumnName>$nn	Location_Name$nn или Loc_Name$nn
WITH CHECK OPTION	<ViewName>$co	LineItem$co

Представления

Имя представления и материализованного представления – всегда во множественном числе, имя объектного представления – во множественном числе, только если необходимо подчеркнуть отношение один-ко-многим.

Столбцы представлений имеют те же имена, что и соответствующие столбцы в базовой таблице. Для столбцов, не имеющих базовых, используются те же правила, что и для столбцов таблиц.

Объект	Правило именования	Пример использования
Представления	<ViewName>$vi	LineItems$vi
Объектные представления	<ViewName>$ovi	LineItem$ovi
Материализованные представления	<ViewName>$mvi	LineItems$mvi

Последовательности

Если последовательностей в одной таблице несколько, перед $seq через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице, на который создана последовательность.

Объект	Правило именования	Пример использования
Для одного столбца	<TableName>$seq или <TableAlias>$seq	Location$seq или Loc$seq
Для нескольких столбцов	<SequenceName>$seq	TransactionId$seq

Индексы

Имена индексов по первичным, внешним и уникальным ключам совпадают с именами соответствующих ограничений целостности.

Объект	Правило именования	Пример использования
Отальные B*tree индексы	<TableName>_<ColumnNames>$i или <TableAlias>_<ColumnNames>$i	Employee_EmpNo$I или Emp_EmpNo$i
Битовые индексы	<TableName>_<ColumnNames>$bmi или <TableAlias>_<ColumnNames>$bmi	Employee_HairColor$bmi или Emp_HairColor$bmi
Кластерные индексы	<TableName>_<ColumnNames>$ci или <TableAlias>_<ColumnNames>$ci	Employee_EmpNo$ci или Emp_EmpNo$ci

Кластеры

Объект	Правило именования	Пример использования
Индексный кластер	<ClusterName>$iclu	EmpDept$iclu
Хэш-кластер	<ClusterName>$hclu	EmpDept$hclu

Роли

Объект	Правило именования	Пример использования
Роль	<RoleName>$r	AdvancedUser$r

PL/SQL модули

Имя программного модуля должно описывать его функцию. При этом имя процедуры должны быть в форме глаголСуществительное, поскольку процедура выполняет действие. Имя функции должно быть именем существительным, поскольку процедура возвращает значение (допустим вариант isСуществительное).

Объект	Правило именования	Пример использования
Пакеты	<PackageName>	InsEmp

Отдельные функции и процедуры	<procedureName>	insEmp
Процедуры и функции, включенные в пакет	<procedureName>	insEmp
Методы	<methodName>	getName

Переменные и параметры

Для переменных, констант и формальных параметров в дополнение к суффиксной форме используется и префиксная.

Добавление префиксов к именам параметров и переменных позволяет отличить их от столбцов таблиц и друг от друга.

Префиксы

Объект	Правило именования	Пример использования
Формальные параметры	p<mode>_<ParameterName>	pi_AirportCode mode может принимать одно из следующих значений: i – IN, o – OUT, io – INOUT.
Глобальные общие переменные пакетов	<VariableName>	AirportCode
Глобальные частные переменные пакетов	g_<VariableName>	g_AirportCode
Локальные переменные процедур	v_<VariableName>	v_AirportCode
Глобальные и локальные константы	c_<ConstantName>	c_StringLength

Суффиксы

Для массивов, вложенных таблиц и index-by таблиц имя задаётся во множественном числе. В остальных случаях – в единственном числе.

Объект	Правило именования	Пример использования
Записи	<VariableName>Rec	ListItemRec
Курсорные переменные	<VariableName>CurV	ListItemCurV
Курсоры	<VariableName>Cur	ListItemCur
Массивы	<VariableName>ListV	ListItemsListV
Вложенные таблицы	<VariableName>ListN	ListItemsListN
Index-by таблицы	<VariableName>ListT	ListItemsListT

Триггеры

Если триггеров с одинаковыми именами несколько, можно добавлять порядковый номер.

Объект	Правило именования	Пример использования
DML-триггеры	<TableName>${b\|a\|io}[i][u][d]{r\|s} или <TableAlias>${b|a|io}[i][u][d][{r|s}] b|a|io означает BEFORE | AFTER | INSTEAD OF, [i][u][d] означает INSERT, UPDATE, DELETE r|s означает Row level | Statement level	Projects$iod или Prs$bis
Триггеры уровня БД	Database${b|a}{st|sh|ln|lf|sr} b|a означает BEFORE | AFTER, st|sh|ln|lf|sr означает STARTUP, SHUTDOWN, LOGON, LOGOFF, SERVERERROR	Database$bst
Триггеры уровня схемы	Schema${b|a}{ln|lf|sr|c|a|d} b|a означает BEFORE | AFTER, ln|lf|sr|c|a|d означает LOGON, LOGOFF, SERVERERROR, CREATE, ALTER, DROP	Schema$bln

Стандарт 3 (комбинированный)

Общие принципы

В сложных именах объектов, состоящих из нескольких слов, слова разделяются символом подчёркивания – ‘_’, если описание объекта хранится в словаре данных, и не разделяются в остальных случаях.

Для удобства выборки из User_Objects всех объектов, связанных с данным, по условию вида

```
WHERE Name LIKE ‘OBJECT_NAME%’
```

все расширения имени объекта добавляются суффиксами (справа). Для объектов схемы суффикс отделяется от собственно имени символом ‘$’ и пишется строчными буквами, за исключением типов, у которых суффикс двойной, и первая часть отделяется от собственно имени символом ‘_’ и пишется по правилам регистра для имён. Для атрибутов типов, столбцов таблиц суффикс отделяется от собственно имени символом ‘_’ и пишется по правилам регистра для имён. Для переменных и параметров программных модулей суффикс не отделяется от собственно имени символом и пишется по правилам регистра для имён.

Типы

Подразумеваются как объектные типы, так и типы внутри PL/SQL модулей.

Для типов-массивов, вложенных таблиц и index-by таблиц имя типа задаётся во множественном числе. В остальных случаях – в единственном числе.

Объект	Правило именования	Пример использования
Тип	`<Type_Name>$typ`	`Line_Item$typ`
Тип-запись	`<Type_Name>_Rec$typ`	`Line_Item_Rec$typ`
Тип-курсорная переменная	`<Type_Name>_CurV$typ`	`Line_Item_CurV$typ`
Тип-курсор	`<Type_Name>_Cur$typ`	`Line_Item_Cur$typ`
Тип-массив	`<Type_Name>_ListV$typ`	`Line_Items_ListV$typ`
Тип-вложенная таблица	`<Type_Name>_ListN$typ`	`Line_Items_ListN$typ`
Тип-index-by таблица	`<Type_Name>_ListT$typ`	`Line_Items_ListT$typ`

Атрибуты типов

Для атрибутов-массивов и вложенных таблиц имя атрибута задаётся во множественном числе. В остальных случаях – в единственном числе.

Объект	Правило именования	Пример использования
Скалярный атрибут	`<Attribute_Name>`	`Street_Address`
Атрибут-объект	`<Attribute_Name>`	`Street_Address`
Атрибут-массив	`<Attribute_Name>_ListV`	`Street_Addresses_ListV`
Атрибут-вложенная таблица	`<Attribute_Name>_ListN`	`Street_Addresses_ListN`
Атрибут-указатель	`<Attribute_Name>_Ref`	`Street_Address_Ref`

Таблицы

Обязательно определение краткого имени таблицы (alias), которое состоит ровно из трех символов.

Для получения краткого имени таблицы можно применять технику CDM, описанную ниже.

Если имя сущности состоит из нескольких слов, используются первые символы первого и второго слов и последний символ последнего слова. Если имя сущности состоит из одного слова из нескольких слогов, используются первые символы первого и второго слогов и последний символ. Если имя сущности состоит из одного слова, используются два первых и последний символ. Если полученное имя уже задействовано, используются два последних символа вместо одного.

Однако, эта алгоритмическая техника не всегда приводит к понятному результату. Поэтому предпочтительнее выбирать краткое имя, исходя из общепринятых сокращений и человеческой логики.

Имя таблицы для чисто реляционных схем – всегда во множественном числе, для объектно-реляционных схем – во множественном числе, только если необходимо подчеркнуть отношение один-ко-многим.

Объект	Правило именования	Пример использования
Реляционная таблица	<Table_Name>	Line_Items
Объектная таблица	<Table_Name>$obj	Line_Item$obj
Таблица хранения	<Table_Name>$ntab	Line_Items$ntab
Индексная таблица	<Table_Name>$iot	Line_Items$iot

Столбцы таблиц

Для столбцов-массивов и вложенных таблиц имя столбца задаётся во множественном числе. В остальных случаях – в единственном числе.

Для столбцов-внешних ключей используется либо полное имя в единственном числе, либо краткое имя таблицы, на которую ссылается внешний ключ. Допустимо вместо этого использование другого имени, если получается несколько столбцов с одинаковыми именами (несколько внешних ключей между двумя таблицами), или если другое имя точнее отражает смысл столбца. Если присоединённая таблица из другого приложения, её краткое имя может префиксироваться кодом приложения.

Объект	Правило именования	Пример использования
Столбец	<Column_Name>	Street_Name
Столбец-первичный ключ	Id	Id
Столбец-внешний ключ	<Table_Name>_Id или <TableAlias>_Id	Product_Id или Pdt_Id
Столбец-массив	<Column_Name>_ListV	Street_Addresses_ListV
Столбец-вложенная таблица	<Column_Name>_ListN	Street_Addresses_ListN
Столбец-указатель	<Column_Name>_Ref	Street_Address_Ref

Ограничения целостности

Для внешних ключей, если присоединённая таблица из другого приложения, её краткое имя может префиксироваться кодом приложения.

Если внешних ключей между одними и теми же таблицами несколько, перед $fk через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице с внешним ключом.

Если уникальных ключей в одной таблице несколько, перед $uk через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице с уникальным ключом.

Если CHECK-ограничений целостности в одной таблице несколько, перед $ck через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице с ограничением целостности.

Объект	Правило именования	Пример использования
Первичный ключ	<Table_Name>$pk или <TableAlias>$pk	Location$pk или Loc$pk
Внешний ключ	<Table_Name>_<RefTableAlias>$fk или <TableAlias>_<RefTableAlias>$fk	Projects_Loc$fk или Prs_Loc$fk
Уникальный ключ	<Table_Name>$uk или <TableAlias>$uk	Location$uk или Loc$uk
CHECK-ограничение целостности	<Table_Name>$ck или <TableAlias>$ck	Location$ck или Loc$ck
NOT NULL	<Table_Name>_<Column_Name>$nn	Location_Name$nn

(если он явно именуется)	или <TableAlias>_<Column_Name>$nn	или Loc_Name$nn
WITH CHECK OPTION	<View_Name>$co	Line_Item$co

Представления

Имя представления и материализованного представления – всегда во множественном числе, имя объектного представления – во множественном числе, только если необходимо подчеркнуть отношение один-ко-многим.

Столбцы представлений имеют те же имена, что и соответствующие столбцы в базовой таблице. Для столбцов, не имеющих базовых, используются те же правила, что и для столбцов таблиц.

Объект	Правило именования	Пример использования
Представления	<View_Name>$vi	Line_Items$vi
Объектные представления	<View_Name>$ovi	Line_Item$ovi
Материализованные представления	<View_Name>$mvi	Line_Items$mvi

Последовательности

Если последовательностей в одной таблице несколько, перед $seq через подчёркивание добавляется дополнительная информация, например, имя столбца в таблице, на который создана последовательность.

Объект	Правило именования	Пример использования
Для одного столбца	<Table_Name>$seq или <TableAlias>$seq	Location$seq или Loc$seq
Для нескольких столбцов	<Sequence_Name>$seq	Transaction_Id$seq

Индексы

Имена индексов по первичным, внешним и уникальным ключам совпадают с именами соответствующих ограничений целостности.

Объект	Правило именования	Пример использования
Отальные B*tree индексы	<Table_Name>_<Column_Names>$i или <TableAlias>_<Column_Names>$i	Employee_Emp_No$I или Emp_Emp_No$i
Битовые индексы	<Table_Name>_<Column_Names>$bmi или <TableAlias>_<Column_Names>$bmi	Employee_Hair_Color$bmi или Emp_Hair_Color$bmi
Кластерные индексы	<Table_Name>_<Column_Names>$ci или <TableAlias>_<Column_Names>$ci	Employee_Emp_No$ci или Emp_Emp_No$ci

Кластеры

Объект	Правило именования	Пример использования
Индексный кластер	<Cluster_Name>$iclu	Emp_Dept$iclu
Хэш-кластер	<Cluster_Name>$hclu	Emp_Dept$hclu

Роли

Объект	Правило именования	Пример использования
Роль	<Role_Name>$r	Advanced_User$r

PL/SQL модули

Имя программного модуля должно описывать его функцию. При этом имя процедуры должны быть в форме глагол_Существительное, поскольку процедура выполняет действие. Имя функции должно быть именем существительным, поскольку процедура возвращает значение (допустим вариант is_Существительное).

Объект	Правило именования	Пример использования
Пакеты	<Package_Name>	Ins_Emp
Отдельные функции и процедуры	<procedure_Name>	ins_Emp
Процедуры и функции,	<procedure_Name>	ins_Emp

включенные в пакет		
Методы	<method_Name>	get_Name

Переменные и параметры

Для переменных, констант и формальных параметров в дополнение к суффиксной форме используется и префиксная.

Добавление префиксов к именам параметров и переменных позволяет отличить их от столбцов таблиц и друг от друга.

Префиксы

Объект	Правило именования	Пример использования
Формальные параметры	p<mode>_<ParameterName>	pi_AirportCode mode может принимать одно из следующих значений: i – IN, o – OUT, io – INOUT.
Глобальные общие переменные пакетов	<VariableName>	AirportCode
Глобальные частные переменные пакетов	g_<VariableName>	g_AirportCode
Локальные переменные процедур	v_<VariableName>	v_AirportCode
Глобальные и локальные константы	c_<ConstantName>	c_StringLength

Суффиксы

Для массивов, вложенных таблиц и index-by таблиц имя задаётся во множественном числе. В остальных случаях – в единственном числе.

Объект	Правило именования	Пример использования
Записи	<VariableName>Rec	ListItemRec
Курсорные переменные	<VariableName>CurV	ListItemCurV
Курсоры	<VariableName>Cur	ListItemCur
Массивы	<VariableName>ListV	ListItemsListV
Вложенные таблицы	<VariableName>ListN	ListItemsListN
Index-by таблицы	<VariableName>ListT	ListItemsListT

Триггеры

Если триггеров с одинаковыми именами несколько, можно добавлять порядковый номер.

Объект	Правило именования	Пример использования
DML-триггеры	<Table_Name>${b\|a\|io}[i][u][d]{r\|s} или <TableAlias>${b|a|io}[i][u][d][{r|s}] b|a|io означает BEFORE | AFTER | INSTEAD OF, [i][u][d] означает INSERT, UPDATE, DELETE r|s означает Row level | Statement level	Projects$iod или Prs$bis
Триггеры уровня БД	Database${b|a}{st|sh|ln|lf|sr} b|a означает BEFORE | AFTER, st|sh|ln|lf|sr означает STARTUP, SHUTDOWN, LOGON, LOGOFF, SERVERERROR	Database$bst
Триггеры уровня схемы	Schema${b|a}{ln|lf|sr|c|a|d} b|a означает BEFORE | AFTER, ln|lf|sr|c|a|d означает LOGON, LOGOFF, SERVERERROR, CREATE, ALTER, DROP	Schema$bln

ГЛАВА

3 Другие требования к PL/SQL коду

В этой главе собраны принципы и практики, ведущие к написанию надёжного, легко сопровождаемого и повторно используемого PL/SQL кода.

Переменные

%TYPE и %ROWTYPE

Всегда используйте %TYPE и %ROWTYPE для декларации переменных и параметров, которые на самом деле являются представлением столбцов БД или курсоров.

Это, во-первых, синхронизирует изменение деклараций с при изменением структур таблиц, а во-вторых, делает явной связь конкретных переменных и столбцов. Имя переменной, при этом, может совпадать с именем столбца – наличие префикса позволит их однозначно различить.

По тем же причинам используйте %TYPE и %ROWTYPE для декларации переменных и параметров, тип которых зависит от других переменных и параметров или совпадает с типом других переменных и параметров.

Литералы

Используйте именованные константы вместо литералов. Такой код легче читать и сопровождать.

Глобальные переменные

Не используйте глобальных общих переменных пакетов. Все глобальные переменные должны быть частными и доступ к ним должне осуществляться через пару процедур: set<VariableName> и get<VariableName> или процедуру set<VariableName> и функцию get<VariableName> (второй варант предпочтительнее, так как допускает использование непосредственно в правой части выражения). Тем самым обеспечивается гибкость в изменении реализации переменной.

PLS_INTEGER

Используйте, где это необходимо, тип данных PLS_INTEGER.

Этот тип представляет собой машинно-зависимое представление целого. Соответственно целочисленные операции с этим типом быстрее, чем с традиционными NUMBER и BINARY_INTEGER.

Условные выражения

ELSIF

Используйте ELSIF вместо ELSE с последующим IF.

Присвоение булевских выражений

Используйте возможность присвоить значение булевского выражения непосредственно булевской переменной.

Например, вместо

```
IF HireDate < SYSDATE
THEN
   DateInPast := TRUE;
ELSE
   DateInPast := FALSE;
END IF;
```

лучше

```
DateInPast := HireDate < SYSDATE;
```

Циклы

Индексные переменные

Никогда не декларируйте индексную переменную цикла FOR – PL/SQL делает это автоматически. Индексная переменная числового цикла, при этом, декларируется как BINARY_INTEGER, а индексная переменная цикла по курсору – как запись этого курсора. Если Вы продекларируете индексную переменную явно, это будет другая переменная.

Называйте индексные переменные числовых циклов не i, j, k, а именем, отражающим её смысл.

Выход из цикла

Не используйте EXIT в циклах FOR и WHILE. Например, вместо

```
WHILE MoreRecords
      LOOP
         nextRecord;
         EXIT WHEN :Caller.Name IS NULL;
      END LOOP;
```

лучше

```
WHILE MoreRecords
LOOP
   nextRecord;
   MoreRecords := :Caller.Name IS NOT NULL;
END LOOP;
```

Если без этого не обойтись, используйте другую конструкцию цикла.

С другой стороны, выход из цикла по исключению является корректным.

Не используйте RETURN внутри цикла.

Курсоры

Явный курсор vs. неявный

Наиболее предпочтительным типом курсора является явный курсор с параметрами.

Во-первых, явный курсор всегда немного быстрее неявного, который выполняет два fetch: первый, чтобы выбрать запись, второй, чтобы убедиться, что выбирается только одна.

Во-вторых, явный курсор более управляем: можно выбрать, где его открыть, где закрыть и не беспокоиться об обработке исключений NO_DATA_FOUND и TOO_MANY_ROWS.

В-третьих, определение курсора в пакете и с параметрами позволит использовать этот курсор в различных процедурах этого и других пакетов.

Курсор, определённый в спецификации пакета, остаётся открытым, даже если модуль, открывший его, завершился. Такие курсоры необходимо закрывать явно.

Цикл FOR по курсору

Используйте цикл FOR по курсору, как наглядную, краткую и не требующую большого внимания при программировании конструкцию (открывается и закрывается автоматически). Но только в тех случаях, когда нужно выполнить некоторое действие над каждой из выбираемых записей, чтобы не было выхода из цикла по EXIT или RETURN.

Если не используется цикл FOR по курсору, декларируйте запись курсора как %ROWTYPE.

Программные модули

Передача параметров

Избегайте передавать большие структуры данных в и особенно из процедур – это очень ресурсоёмкая операция. Передавайте только необходимый минимум данных.

IN-параметры передаются по ссылке, что быстрее всего. Это безопасно, так как появление IN-параметров в левой части выражения запрещено компилятором.

OUT-параметры передаются копированием. INOUT-параметры передаются двойным копированием (туда и обратно). Это для того, чтобы новое значение фактического параметра было видно только при нормальном завершении процедуры, не по исключению.

Опция NOCOPY заставляет передавать даже OUT- и INOUT-параметры по ссылке. Но тогды при выходе по исключению изменения фактических параметров будут видны.

В некоторых случаях, чтобы избежать передачи больших структур данных в качестве параметров, можно использовать глобальные переменные. Но необходимо помнить, что передача через параметры является более предпочтительной практикой, нежели через глобальные переменные (см. Побочные эффекты функций).

Исключите обращение к одной и той же переменной и как к глобальной, и как к параметру внутри одного и того же программного модуля. Фактически, Вы будете работать с одной и той же переменной под двумя разными именами, и, в зависимости от того, указана опция NOCOPY или нет, получите разные результаты. А с учётом того, что NOCOPY – всего лишь подсказка и может быть проигнорирована компилятором, результат работы программного модуля будет труднопредсказуем.

Проверка предположений

Каждый модуль имеет некоторые предположения относительно принимаемых параметров. Например, параметр находится в некотором диапазоне значений или принадлежит некоторому домену.

Проверяйте внутри модуля соответствие параметров этим предположениям. В противном случае, Вы рискуете получить необработанные исключения или просто некорректную работу модуля.

Проверку предположений можно проводить с помощью ***(см. ????)***.

Возврат значения из функции

Функция должна возвращать значение. Если алгоритм функции предусматривает ветвления, каждая ветвь должна содержать оператор RETURN. Но лучше всегда один оператор RETURN в конце функции.

Если функция содержит обработчики исключений, в которых исключения не регенерируются, в них должен содержаться оператор RETURN.

Нежелательно наличие OUT и INOUT параметров функции. Это неочевидно и относится к побочным эффектам. Рекомендуется либо собрать эти параметры в запись и возвращать запись, либо преобразовать функцию в процедуру.

Побочные эффекты функций

Функция должна выполнять только действия, необходимые для возврата требуемого значения. Например, функция, подсчитывающая и возвращающая сумму продаж, не должна выводить ту же сумму продаж на экран при помощи DBMS_Output.

Не обращайтесь к глобальным переменным и константам напрямую, так как зависимость функции от соответствующих глобальных переменных и констант становится неочевидной. Вместо этого передавайте значения глобальных переменных и констант через параметры.

Одним из побочных эффектов является также возврат значений в IN и INOUT параметрах (см. Возврат значения из функции).

Совместимость

При добавлении новых IN-параметров процедуры/функции им нужно присвоить значения по умолчанию. Это позволит сохранить совместимость с существующими вызовами процедуры/функции, особенно если используется именованная, а не позиционная нотация.

При добавлении новых OUT или INOUT-параметров может помочь создание перегружаемой процедуры/функции над изменяемой без новых параметров.

Прозрачность

Выделяйте сложные условия из условных операторов в булевские функции.

Выделяйте сложные секции кода в циклах и условных операторах в процедуры.

Таким образом, на верхнем уровне останется практически только алгоритм, описанный псевдокодом. Это облегчает чтение и понимание верхнего уровня алгоритма. Для детализации отдельных фрагментов можно затем опуститься в тексты упомянутых функций и процедур.

Например, вместо

```
IF TotalSal BETWEEN 10000 AND 50000 AND
      EmpStatus (EmpRec.EmpNo) = 'N' AND
      (MONTHS_BETWEEN (EmpRec.HireDate, SYSDATE) > 10)
THEN
   giveRaise (EmpRec.EmpNo);
END IF;
```

лучше

```
IF eligibleForRaise (EmpRec.EmpNo)
THEN
   giveRaise (EmpRec.EmpNo);
END IF;
```

Аналогично используйте функции в списке SELECT. Это упрощает вид оператора, облегчает его понимание и сопровождение, инкапсулирует сложную логику.

Локальные модули

Используйте локальные модули (определённые в секции деклараций процедуры/функции), если логика, заключённая в них, больше нигде не потребуется.

Выделение части кода процедуры/функции в такие локальные модули сделает процедуру/функцию более прозрачной, возможно устранит избыточность кода (повторяемость).

Перегрузка

Используйте перегрузку.

Перегружаются программные модули формальные параметры, которых отличаются типом.

Не перегружаются программные модули, если типы их формальных параметров принадлежат к одному семейству. Например, у одного – тип формального параметра CHAR, а у другого – VARCHAR2, или у одного – NUMBER, а у другого – INTEGER.

Перегружаются программные модули, которые отличаются количеством формальных параметров.

Не перегружаются программные модули, формальные параметры которых отличаются только режимом.

Перегружаются пакетные процедура и функция.

Не перегружаются отдельные процедуры и функции.

Не перегружаются функции, которые отличаются только типом возвращаемого значения.

Отдельные процедуры и функции

Избегайте отдельных процедур и функций. Объединяйте их в пакеты.

Поскольку в большинстве случаев достаточно изменить и перекомпилировать тело пакета, не затрагивая спецификацию, особенно при условии хранения спецификации и тела в отдельных файлах, зависимые объекты не получат статус INVALID и не потребуют перекомпиляции при очередном обращении.

Пакеты

Пакет констант

Создайте отдельный пакет для констант, которые используются более, чем в одном месте кода.

Пакет типов данных

Создайте пакет типов данных, которые используются более, чем в одном месте кода (первичный ключ, логическая переменная и т.п.), и в дальнейшем, при декларации переменных ссылайтесь на него.

Этот пакет может совпадать с пакетом констант (см. Пакет констант).

Инкапсуляция доступа

Инкапсулируйте доступ к таблицам и бизнес-правила в пакетах. Выдавайте привилегии только на пакеты и никогда на операции непосредственно с таблицей.

Поскольку нельзя выдать привилегии на отдельные процедуры пакета, над пакетом может потребоваться дополнительный уровень: несколько пакетов-фильтров, содержащих вызовы некоторого подмножества процедур из основного пакета. Тогда управление привилегиями будет выполняться при помощи пакетов-фильтров.

Изоляция

Изолируйте в отдельных пакетах код, который:

- содержит зависимости от операционной системы
- обращается непосредственно к структурам данных

Тем самым достигается изоляция остальных модулей от изменений ОС и структур данных.

Разбиение на уровни

Разбивайте код на уровни. Например, нужно написать пакет для управления такими структурами как стек на базе index-by таблиц. Поскольку стек – это разновидность списка, напрашивается разбиение задачи на три уровня:

1. пакет процедур управления index-by таблицами
2. пакет процедур управления списками
3. пакет процедур управления стеками, добавляющий дополнительные правила и ограничения.

Наличие отдельного пакета процедур управления index-by таблицами изолирует остальные пакеты от структур данных и открывает возможность к прозрачной замене структур данных в будущем.

Наличие отдельного пакета процедур управления списками открывает возможность работы с просто списками и, в частности, создания других структур на базе списков.

Хорошим кандидатом на создание дополнительного уровня кода над ним в силу своей сложности является пакет DBMS_SQL.

Надо помнить, однако, что наличие нескольких уровней может усложнть локализацию ошибки. Поэтому пакеты нужно сразу снабжать встроенными средствами трассировки и отладки ***(см. ????)***.

GRASP

GRASP или General Responsibility Assignment Software Patterns представляют собой набор правил, принципов выделения классов и распределения методов между классами, т.е. «назначения ответственности». В программировании на PL/SQL эти правила служат разбиению кода на пакеты, распределению процедур между пакетами, организации связи между пакетами.

Ниже приводятся основные правила GRASP и даётся их толкование.

Expert

Назначайте ответственность информационному эксперту – классу, который располагает всей информацией, необходимой для выполнения задачи.

Например, имеем следующие классы: Sale, SalesLineItem и Product Specification. ProductSpecification, помимо всего прочего, содержит цену товара, а каждый SalesLineItem – количество проданного товара.

Какой класс должен возвращать общую сумму продажи? Согласно принципу Expert – это Sale, потому что он знает все входящие в него объекты класса SalesLineItem и может подсчитать их сумму. С другой стороны, какой класс должен возвращать сумму каждого объекта SalesLineItem? Конечно, сам SalesLineItem, потому что он содержит количество проданного товара и знает «свой» Product Specification, следовательно может запросить у него стоимость единицы товара.

Таким образом, класс Sale содержит метод total, который вызывает метод subtotal класса SalesLineItem для каждого товара, который, в свою очередь, вызывает метод price класса ProductSpecification.

Creator

Назначайте классу B ответственность за создание экземпляра класса A в следующих случаях:

- объект B объединяет или содержит объекты A
- объект B использует объекты A
- объект B обладает данными, необходимыми для инициализации объекта A

Если эти правила выполняются для нескольких объектов B, предпочтительнее тот, который объединяет или содержит объекты A.

Например, объект SalesLineItem должен создаваться объектом Sale, т.е. объект Sale должен содержать метод makeLineItem.

Другой пример – объект Payment должен создаваться объектом Sale, так как объект Sale знает сумму продажи, которая необходима для инициализации объекта Payment.

Low Coupling

Назначайте ответственность так, чтобы связанность классов оставалась низкой.

Связанность есть мера того, насколько тесно класс связан с другими, насколько много знаний о других классах он имеет, насколько сильно зависит от них. Низкая связанность означает связь с минимальным количеством других классов.

Рассмотрим два варианта создания платежа:

- Метод makePayment класса POST вызывает метод create класса Payment, сохраняет созданный объект класса Payment, затем передаёт его методу addPayment класса Sale
- Метод makePayment класса POST вызывает метод makePayment класса Sale, который, в свою очередь, вызывает метод create класса Payment

Второй варант предпочтительнее, так как сохраняет низкую связанность: класс POST знает только о классе Sale, класс Sale знает только о классе Payment.

Связь классов может принимать следующие формы:

- Класс A имеет атрибут класса B
- Класс A имеет метод с параметром, переменной или возвращаемым значением класса B
- Класс A является подклассом класса B

High Cohesion

Назначайте ответственность так, чтобы сцепленность ответственностей класса

оставалась высокой.

Классы с высокой сцепленностью методов содержат не очень большое количество методов, но все методы очень тесно логически связаны, находятся в одной функциональной области, т.е. класс не выполняет «лишней» работы.

Например, один класс не должен отвечать за весь интерфейс с реляционной БД. Должны быть отдельные классы на управление сессией, выполнение запросов, вызов процедур и т.д.

Polymorphism

Если поведение различных подклассов класса в ответ на одну и ту же операцию варьируется, каждый подкласс должен нести ответственность за выполнение этой операции.

Например, класс Payment имеет 3 подкласса: CashPayment, CreditPayment, CheckPayment. Для каждого из подклассов операция авторизации платежа выполняется по-разному. Следовательно каждый подкласс должен иметь метод authorize.

Pure Fabrication

Назначайте сильно сцепленное множество ответственностей искусственному классу, представляющему собой объект, не существующий в реальном мире.

Это принцип применяется, если нет других способов сохранить низкую связанность и высокую сцепленность.

Indirection

При организации взаимодействия между различными компонентами, сервисами, уровнями кода и т.д. назначайте ответственность специальному промежуточному объекту.

Например, при организации интерфейса с устройством авторизации кредитных карт необходим промежуточный класс AuthorizationDevice, который будет инкапсулировать низкоуровневые вызовы. Тогда при смене ОС или API к устройству авторизации достаточно откорректировать или заменить только этот класс.

Controller

Частный случай принципа Indirection.

Назначайте ответственность за обработку системных сообщений (внешних – от внешних действующих лиц, от внешних устройств) специальному классу, который:

- представляет всю систему в целом (System, PMS), либо
- представляет всю организацию (Hotel, Store), либо
- представляет персону или роль в реальном мире, вовлечённую в эту задачу (Cashier, FrontDeskClerk), либо
- представляет искусственные обработчик для таких сообщений (ReservationHandler).

В частности, классы пользовательского интерфейса не должны непосредственно обращаться к классам уровня приложения. Для этого должен существовать промежуточный уровень классов-контроллеров, который и будет переадресовывать системные вызовы на уровень приложения.

Такая организация позволяет, например, хранить в классе-контроллере текущее состояние операции и запретит выполнение makePayment после endSale.

Don’t Talk to Strangers

Класс должен вызывать только методы следующих классов:

- свои собственные
- класса-параметра метода
- класса-своего атрибута
- объекта, созданного своим методом

Например, класс POST не должен вызывать методов класса Payment. Объект POST создавал объект Sale, который, в свою очередь, создавал объект Payment. Объект POST не имеет никаких непосредственных знаний об объекте Payment и должен обращаться к нему не иначе как при посредничестве объекта Sale.

Этот принцип может быть нарушен в некоторых случаях, таких как реализация принципов Pure Fabrication или Indirection.

Singleton

Класс может быть глобально видим, если он имеет только один экземпляр.

Например, класс с различными настроечными параметрами и/или константами уровня системы может быть глобально виден и доступен отовсюду в нарушение принципов Don’t Talk to Strangers и Low Coupling.

5