# Глава 3 Новые возможности языка ABAP

"Это прекрасная вещь - уничтожение слов. Конечно, большая расточительность заключается в глаголах и прилагательных, но есть сотни существительных, от которых тоже можно избавиться."

-Джордж Оруэлл, 7984

Новые возможности, новые вкусности... можно ли назвать это раем для программиста? В начале восьмидесятых, когда я был еще подростком, родители купили мне микрокомпьютер BBC на замену моему ZX81. Уже тогда моим основным интересом было программирование, а не игры, и больше всего в моей новой игрушке мне нравился широкий спектр команд, которые были доступны мне в версии BASIC от BBC (например, такие конструкции, как REPEAT UNTIL). Даже сейчас, много лет спустя, я чувствую то же самое, когда в язык ABAP добавляется что-то новое и интересное.

С течением времени в ABAP добавляется все больше команд и конструкций, но ничего не убирается, чтобы обеспечить обратную совместимость, и скорость изменений, похоже, резко возрастает. Достаточно много изменений было сделано в результате внедрения SAP NetWeaver7.02, но это ничто по сравнению с тем каскадом изменений, которые появились в версии 7.4 и продолжились в версиях 7.5 и последующих. Это неопровержимо доказывает, что ABAP не является мертвым языком, чего многие боялись в 2001 году, когда на самом верху SAP заговорили о том, чтобы замене ABAP на Java.

Цитата в начале главы из романа Джорджа Оруэлла "1984" говорит о разрушении слов. Персонаж, работающий над языком Newspeak, говорит главному герою, Уинстону Смиту, что правящая партия стремится уничтожить существующие слова, а не создавать новые. Идея, зародившаяся в ABAP 7.4, в некотором роде та же самая: многие изменения в 7.4 позволяют решать точно такие же задачи, как и раньше, но с половиной кода или меньше. Например, в Newspeak "это было замечательно, фантастично, лучше не бывает" превращается в "++good". В ABAP, *CONCATANATE this that INTO the\_other SEPARATED BY ' '* становится *the\_other = this && ' ' && that*.

137

Конечно, дело не только в уничтожении слов; релизы 7.4 и выше добавляют множество совершенно новых функций. Эта глава будет посвящена большому количеству новых функций, которые появились в ABAP в последних релизах. Эти возможности разделены на следующие категории, исходя из их общей области функциональности: создание данных, обработка строк, вызов функций, условная логика, внутренние таблицы, объектно-ориентированное (ОО) программирование и помощь в поиске.

Версии ABAP

На момент написания статьи последняя версия ABAP имеет два названия, что не является необычным для SAP. Одно название - 7.53, что соответствует именованию предыдущих версий. Но в документации SAP вы также увидите, что она упоминается как ABAP platform 1809, исходя из месяца выпуска - сентябрь 2018 года.

С этого момента новый релиз будет выходить каждые три месяца, так что на Рождество будет выпущен будет релиз 1812, и сообщество разработчиков ABAP может начать петь увертюру 1812 года. Очень жаль, что в году не пятнадцать месяцев, потому что 1815 было бы отличное название для релиза. Мы все могли бы спеть соответствующую песню ABBA, что было бы уместно, потому что ABBA звучит немного похоже на ABAP.

Версия 7.53/ABAP 1809 уничтожает последние остатки независимости платформы. Эта версия ABAP работает только с базой данных SAP HANA, которая, в свою очередь, работает только с операционной системой Linux. Основная часть книги предполагает наличие локальной системы ABAP 1809. В главе 14 речь пойдет о SAP Cloud Platform, среде ABAP, обычно сокращенно называемой ABAP in the cloud. Теоретически, она использует ту же самую строку кода, что и ABAP 1809 - по крайней мере, на момент написания книги - фактически она совершенно другая.

## 3.1 Объявление и создание переменных

Недавно компания SAP вскочила на крышу и прокричала через мегафон, что в ABAP 7.5 появился новый тип данных, INT8, который обрабатывает действительно большие числа (т.е, числа длиной в восемнадцать цифр, плюс или минус). Я предполагаю, что это то, что подразумевается большие данные; я всегда задавался этим вопросом. Возможно, вам нужны такие большие целые числа, чтобы записать количество изменений названий продуктов SAP в год.

Спускаясь с крыши, скажу, что одно из главных различий между ABAP и другими заключается в том, что в ABAP нас учили объявлять все наши переменные в начале

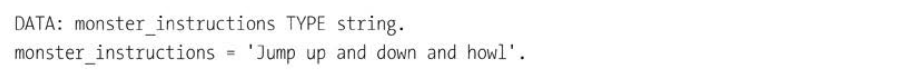
138

программы - например, в TOP INCLUDE. В других языках они объявляются непосредственно перед использованием, и такие переменные, как правило, имеют более локальную область применения - например, внутри цикла. Несмотря на официальные рекомендации по программированию ABAP, многие программисты ABAP перешли на объявление переменных непосредственно перед их первым использованием, чтобы сделать программу более читабельной для человека и, следовательно, более легкой для изменения.

Хорошей новостью является то, что со временем изменения в языке ABAP делают эту неофициальную практику все более приемлемой. В этом разделе мы обсудим несколько особенностей которые способствуют этому сдвигу.

### 3.1.1 Отказ от объявления типов данных

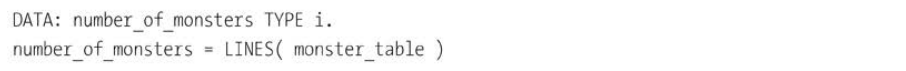
Одна из причин, по которой в ABAP 7.4 стало менее важным объявлять переменные в начале вашей процедуры или метода, заключается в том, что необходимость в большинстве объявлений данных отпала. Компилятор знает, какой тип данных ему нужен (он должен знать, чтобы иметь возможность выполнить проверку синтаксиса), так почему бы не позволить ему определить тип данных вашей переменной? В следующих примерах вы увидите, как предоставление права на определение. Например, вот фрагмент кода, объявляющий тип данных:



Без объявления типа данных это можно упростить до следующего:



Аналогично, следующий фрагмент кода также объявляет тип данных:



И снова, вы можете упростить это:



Как вы можете видеть, эта система способна значительно сократить количество строк в ваших программах. Вы увидите различные примеры таких встроенных деклараций на протяжении всей этой главы, потому что они находят применение во многих областях программирования ABAP.

139

### 3.1.2 Создание объектов с помощью команды NEW

В языке программирования Java команда NEW используется для создания экземпляров объектов (говорят, что все в Java является объектами). Например, MonsterFred = NEW(Monster) в Java создает экземпляр с именем Fred класса Monster. Точно так же как среда выполнения ABAP беззастенчиво крадет возможности из Eclipse, теперь язык ABAP крадет ключевые слова из Java - так что теперь у вас есть команда NEW.

Раньше вы использовали следующий код:



Теперь вы сможете использовать только следующую строку:



Как видите, это вдвое сокращает количество необходимых строк кода. Что еще более важно, дебаты о том, объявлять ли переменные в начале процедуры (согласно официальным руководствам ABAP) или непосредственно перед переменной (чтобы помочь людям, которые могут читать код) больше не актуальны. Вы имеете тип переменной прямо перед своим в момент создания переменной.

### 3.1.3 Заполнение структур и внутренних таблиц при их создании с помощью VALUE

Несомненно, вы знакомы со следующим распространенным утверждением:

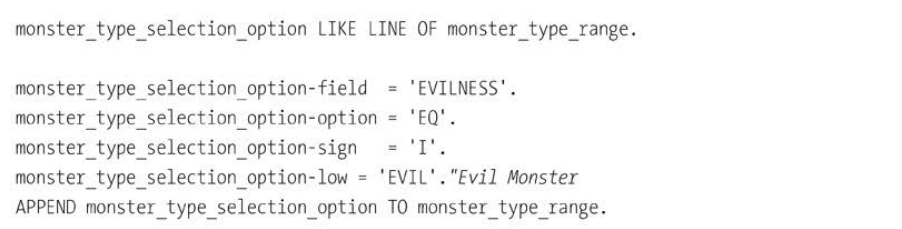


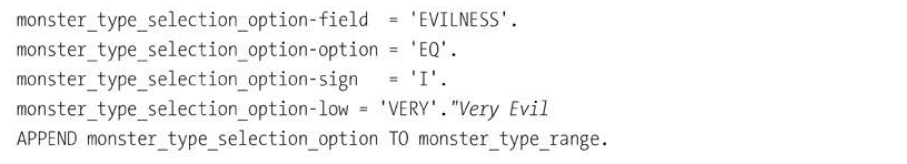
В этом операторе вы создаете переменную и присваиваете ей начальное значение, которое впоследствии может быть изменено. В прошлом возможность изменения начальных значений была доступна только для элементарных типов данных. Однако, начиная с версии 7.4, оператор VALUE достиг своего совершеннолетия, и теперь вы можете определять начальные значения для структур и внутренних таблиц.

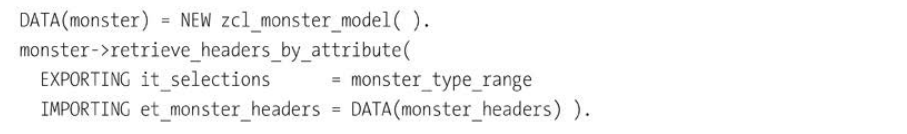
В листинге 3.1 приведен пример запроса к базе данных, существовавший до версии 7.4. В этом примере вы создаете таблицу выбора для использования в SQL-запросе, но вас интересуют только лаборатории, в которых будут созданы монстры. Однако, поскольку вы не можете использовать оператор VALUE, вы должны заполнить одну или несколько рабочих областей, а затем добавить их в таблицу выбора.



140

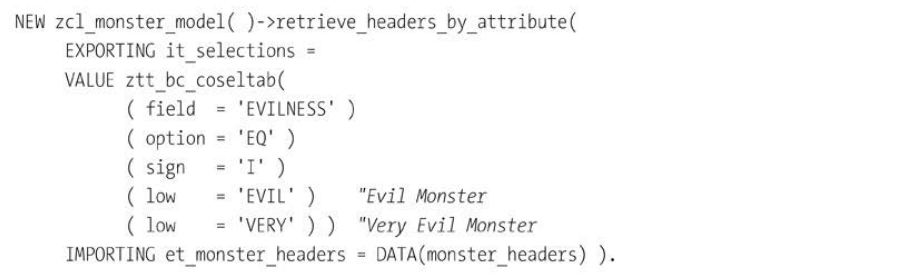






Listing 3.1 Database Query without VALUE

Однако, начиная с версии 7.4, вы можете использовать оператор VALUE и таким образом достичь того же эффекта с меньшим количеством строк кода (Листинг 3.2).

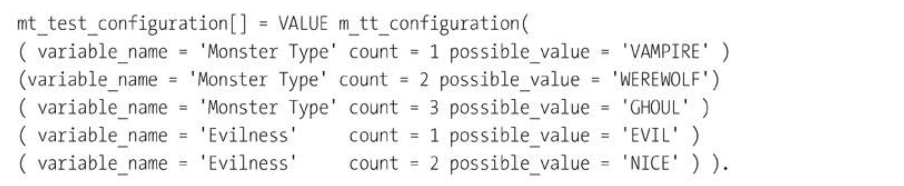


Listing 3.2 Database Query with VALUE

Вы избежали необходимости в промежуточной внутренней таблице (MONSTER\_TYPE\_RANGE) и избежали необходимости определять рабочую область для построения строк этой промежуточной таблицы, и в результате получили код, который четко показывает, что происходит.

141

В общем, вы жестко кодируете значения строк внутренних таблиц при построении тестовых данных для модульных тестов, как показано в листинге 3.3.



Listing 3.3 Using VALUE to Fill Multiple Lines of Internal Table

Листинг 3.3 демонстрирует эту концепцию для внутренней таблицы, но вы также можете использовать эту конструкцию для заполнения структур, и есть еще три важных момента, которые следует отметить о ключевом слове VALUE в этом контексте:

1. При заполнении существующей структуры, в отличие от создания новой, структура данных перед заполнением очищается.

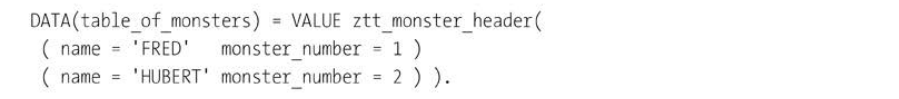
2. Если ранее вы заполняли структуру с помощью сотни строк кода, присваивая значение элементу структуры по одному, вам приходилось пропускать каждую строку вручную в отладчике. Если же вы заполняете их все сразу с помощью оператора VALUE, то отладчику придется пропустить только одну строка, которую отладчик должен пропустить.

3. При заполнении структуры с помощью VALUE требуется один набор скобок в начале; при заполнении таблицы требуется два набора скобок, поскольку каждая строка должна быть заключена в скобки.

### 3.1.4 Заполнение внутренних таблиц из других таблиц с помощью FOR

Как хорошо я помню, как начинал программировать, когда мне было четырнадцать лет, на старом добром ZX81. В языке BASIC, на котором я тогда программировал, были такие конструкции, как FOR x = 1 TO 10, что означало, что вы собираетесь выполнить цикл десять раз, при этом переменная X каждый раз увеличивается на единицу. Теперь команда FOR появилась в мире ABAP; давайте посмотрим, что она из себя представляет.

Вы читали об операторе VALUE в предыдущем разделе; вы можете использовать его для заполнения внутренней таблицы, как в листинге 3.4.



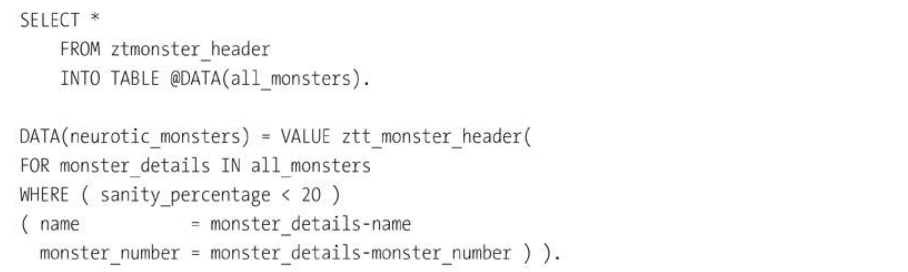
Listing 3.4 Fill Internal Table

142

Это отличный пример, но в реальной жизни вы заполняете внутренние таблицы либо из базы данных, либо из других внутренних таблиц; почти никогда вы не заполняете их жестко закодированными значениями. До версии 7.4 вы могли заполнить одну внутреннюю таблицу из другой таблицы только в том случае, если эти две таблицы имели идентичную структуру столбцов, и вам приходилось добавлять все строки одной таблицы в другую:



К счастью, благодаря команде FOR, появившейся в ABAP 7.4, вы теперь можете сделать это гораздо более элегантным способом: таблицы могут иметь разные столбцы, и вы можете ограничить что передается, основываясь на условной логике. Используя ключевые слова VALUE и FOR, это будет выглядеть как в листинге 3.5.



Listing 3.5 Filling Internal Tables from Other Tables

### 3.1.5 Создание короткоживущих переменных с помощью LET

ABAP 7.4 также позволяет объявлять переменные с коротким сроком жизни с помощью оператора LET. Эти переменные существуют только при создании данных с помощью выражений-конструкторов (Выражение-конструктор - это механизм для применения различной логики при создании переменной, например, внутренней таблицы. Это похоже на логику, которую можно найти в конструкторе ОО-класса, а VALUE является примером выражения конструктора).

Чтобы проиллюстрировать использование операторов LET, предположим, что у вас есть таблица смертоносного оружия, и вы хотите вооружить некоторых монстров. Вас не особенно волнует, какой монстр получит то или иное оружие, главное, чтобы каждый монстр получил по одному оружию. В листинге 3.6 показан код, который создает несколько очень короткоживущих локальных переменных.



143



Listing 3.6 Creating Short-Lived Variables

Примечание

В реальной жизни вы бы хранили описание в какой-нибудь внутренней таблице. Однако, здесь основное внимание уделяется утверждению LET.

В этом примере первая переменная получает описание следующего оружия в списке оружия с помощью вызова функционального метода. Затем вы читаете строку внутренней таблицы монстров - синтаксис здесь может вас озадачить, но он будет рассмотрен позже в разделе 3.5, и, наконец, вы заполняете строковую переменную системной датой, отформатированной в удобном для человека дружественным образом.

Теперь вы можете использовать те переменные, которые только что объявили, для заполнения строки результата.

Благодаря оператору LET, после завершения работы оператора эти переменные (weapon\_name, и т.д.) больше не существуют, в отличие от доступных из любой точки внутри как обычные переменные.

Field Symbols

Вы можете объявлять Field Symbols, как и переменные. Тип переменной или Field Symbols определяется динамически, глядя на передаваемое в него значение.

### 3.1.6 Перечисления

Как мы знаем, некоторые переменные могут иметь только определенный диапазон значений: дневная переменная может иметь значения только от 1 до 7, булева переменная может быть только TRUE или FALSE, а переменная

144

размер мозга монстра может быть только NORMAL, SMALL или MICRO. В каждом случае любое другое значение просто не имеет смысла.

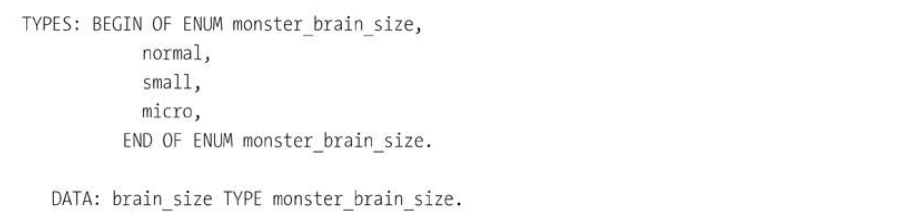
Как мы обычно справляемся с этой проблемой: элемент данных, представляющий концепцию, указывает на область с фиксированным набором значений. Затем технология пользовательского интерфейса, например, SAP GUI, выдает сообщение об ошибке, если пользователь пытается ввести любое другое значение.

Однако внутри программ мы оказываемся на дне. Ничто не может помешать вам поместить бессмысленное значение типа "P" в переменную, набранную как ABAP\_BOOL, или поместить девятку в целое число, предназначенное для представления дня, или поместить BIG в переменную размера мозга монстра.

Более того, переменная может быть настолько специфичной для вашего приложения, что для нее нет домена в DDIC, но вы все равно хотите ограничить возможные значения, которые переменная может иметь во время выполнения. В прошлом вы делали это, объявляя несколько констант с возможными значениями, но в других местах кода ничто не мешало присвоить переменной значения, отличные от констант.

В других языках, таких как Java, уже много лет существует концепция перечислений для решения подобных проблем, а в версии 7.51 эта концепция появилась и в языке ABAP.

Проще говоря, перечисление - это место в вашем коде, где вы перечисляете возможные значения семантического понятия (например, размер мозга монстра). В листинге 3.7 показано, как определить перечисление; по сути, это расширение существующей конструкции TYPE.



Listing 3.7 Basic Enumeration

Обычно вы объявляете TYPE переменной либо элементарным типом, таким как целое число, либо элемент данных (с базовым типом, полученным из связанного с ним домена). В листинге 3.7 вы объявляете тип переменной на основе перечисления. В этом случае TYPE переменной будет целым числом.

"Почему целое число?" спросите вы. Ну, под капотом произошло то, что были объявлены три константы:

• NORMAL= 0

• SMALL= 1

• MICRO= 2

145

Теперь, если вы попробуете что-то вроде этого:



Вы получите синтаксическую ошибку или даже ошибку времени выполнения (которую можно поймать с помощью исключения класса *CX\_SY\_CONVERSION\_NO\_ENUM\_VALUE*), если исходное значение определяется только во время выполнения. Вместо этого программисты вынуждены писать что-то вроде следующего:

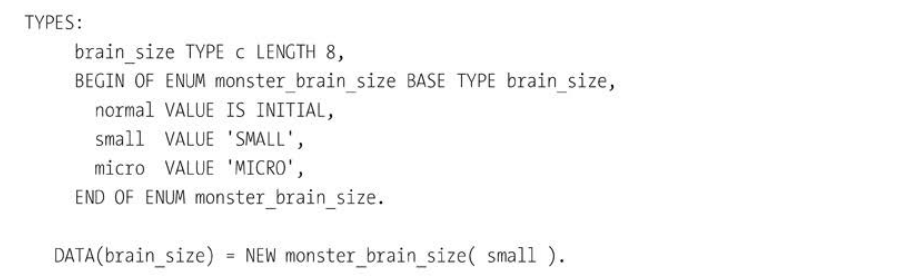


Это также заставляет код выглядеть более похожим на обычный английский язык, так что вы одновременно улучшаете стабильность и читабельность.

Удивительно то, что, если вы передадите константу в текстовое поле как строку, она волшебным образом превратится из целого числа 0 в текстовую строку NORMAL - то есть в имя константы.

Сейчас вы можете сказать, что этот пример не очень реалистичен. Вы правы: размер мозга монстра - это не целое число, а десятисимвольное поле. Конечно, это так;

Перечисление не обязательно должно быть целым числом. Вы можете определить его как другой тип данных, как в листинге 3.8, в котором мы хотели создать десятисимвольное поле, но столкнулись с синтаксической ошибкой, как только длина поля превысила восемь символов. Причина этого заключается в том, что пока перечисления не похожи на домены; предполагается, что вас не должно интересовать фактическое значение константы, которую представляет значение перечисления, только ее имя.

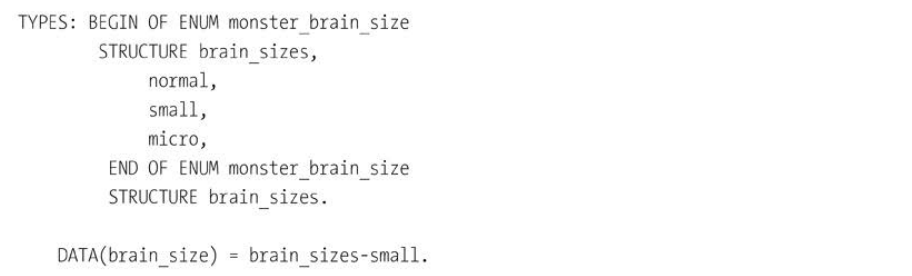


Listing 3.8 Non-Integer Enumeration

Вы можете спросить, почему NORMAL не имеет значения: это потому, что одно из значений должно быть начальным, поэтому обычно эта честь достается значению по умолчанию.

146

Теперь перейдем на следующий уровень (читабельности кода) и определим перечисление не как набор констант, а как структуру констант, как в листинге 3.9.



Listing 3.9 Enumeration with Structure

Как уже говорилось, на данный момент домены и перечисления - это две совершенно несвязанные вещи, несмотря на то, что они выполняют одну и ту же работу, но в будущем мы, вероятно, получим автоматическое связывание с реальными доменами DDIC.

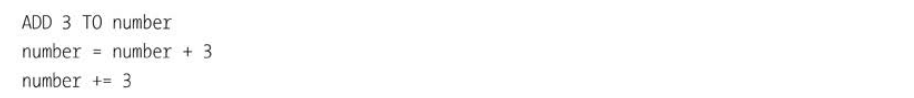
Нам также обещают еще более совершенный вид константы: константа, значение которой динамически присваивается во время выполнения программы, но после создания имеет неизменяемое значение. Если и когда это произойдет. Я буду танцевать по комнате от радости.

### 3.1.7 Новые математические операторы

Как упоминалось ранее, после версии 7.52 изменилось именование релизов ABAP, и они также выходят намного быстрее из-за ABAP в облаке. Релиз в сентябре 2018 года был назывался, например, 1809, а через два месяца был выпущен 1811.

Релиз 1811 "украл" концепцию из других языков, таких как Java, под названием "операторы присваивания". Она призвана сделать код более компактным.

Следующие три оператора выполняют одну и ту же задачу; последний - это новая конструкция:



Это прекрасный пример нового инструмента, который делает кодирование более коротким, но менее читаемым.

Каждый сможет догадаться, что делают первые две строчки кода выше, но для того, чтобы узнать, что делает +=, вам потребуются специальные знания.

147

Аналогичная схема используется и для других математических операторов; например, чтобы разделить число на три, вы используете *number /= 3*.

## 3.2 Обработка строк

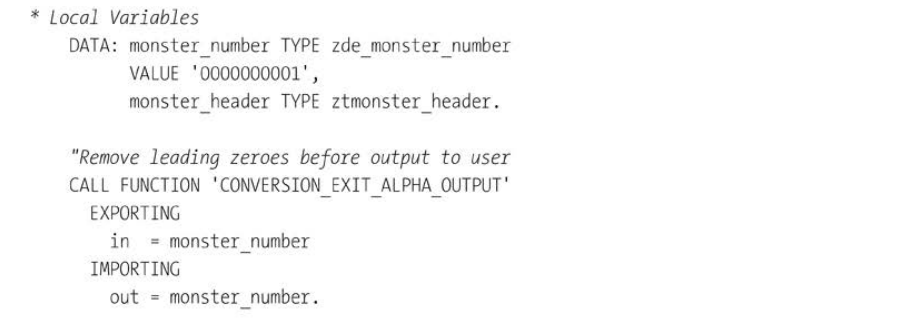
Английское комедийное трио The Goodies однажды спело:

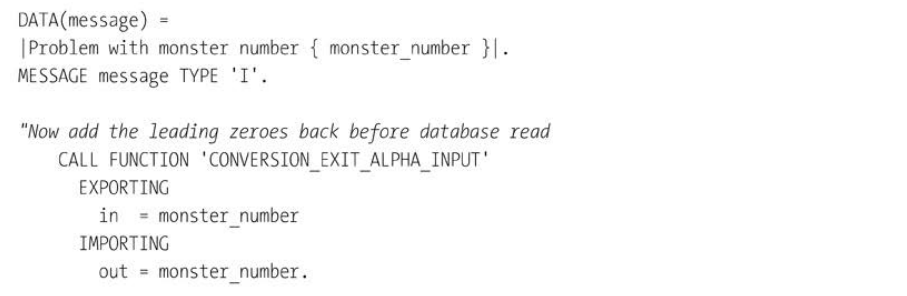
String, string, string, string, everybody loves string,

String for your pants, string for your vest!

Everybody knows- string is best!

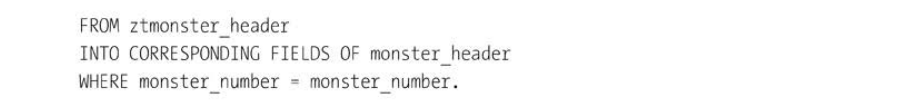
Если вы похожи на меня, вы можете провести половину своей жизни, вызывая функции *CONVERSION\_EXIT\_ALPHA\_INPUT* и *CONVERSION\_EXIT\_ALPHA\_OUTPUT* для добавления и удаления ведущих нулей из номеров документов, таких как номера доставки; например, вы можете удалить нули при показе сообщений пользователю, но затем добавить их обратно перед чтением базы данных (листинг 3.10).





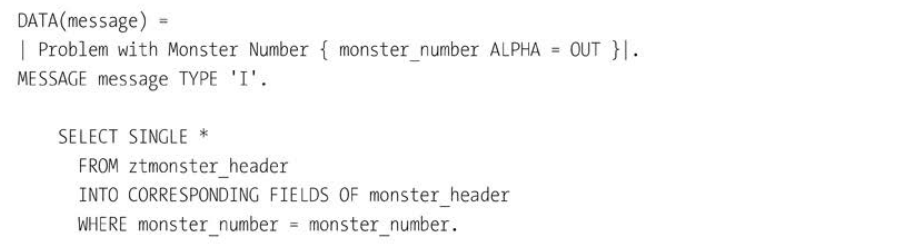


148



Listing 3.10 Removing and Adding Leading Zeroes by Function Call

В ABAP 7.4 это действие может быть выполнено более компактным образом с помощью опции форматирования ALPHA, которая делает то же самое, что и два функциональных блока использованные в листинге 3.10. Чтобы удалить ведущие нули в 7.4, вы можете написать код, подобный как в листинге 3.11.



Listing 3.11 Removing Leading Zeroes via Formatting Option

При использовании метода, показанного в листинге 3.11, вам больше не нужно добавлять ведущие нули обратно; они никогда не были удалены из переменной delivery (или monster number).

## 3.3 Вызов функций

Вам часто приходится прыгать через обручи, чтобы поиграть с переменными в ваших программах, прежде чем вы сможете вызвать другие методы или функции. К счастью для вас, это становится проще с течением времени. В этом разделе будут рассмотрены новые функциональные возможности ABAP, которые делают вызов функций намного проще, чем это было в прошлом.

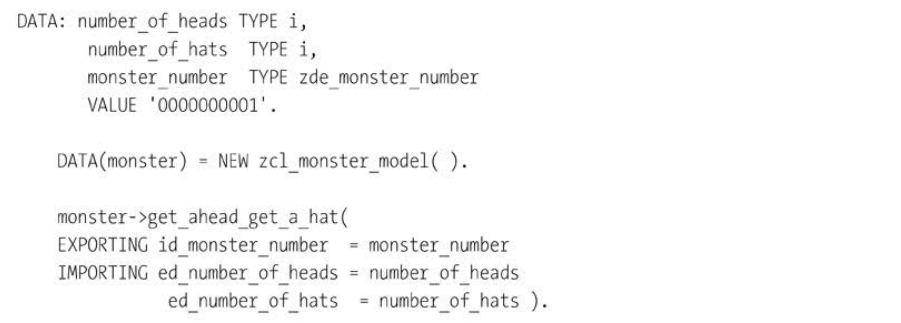
### 3.3.1 Избежание ошибок несоответствия типов при вызове функций

Когда вы объявляете переменную, которую нужно либо передать в метод, либо получить обратно, а тип переменной не соответствует ожидаемому типу параметра - это может вывести вас из себя. В методе вы получаете синтаксическую ошибку; при использовании ФМ вы получите короткий дамп во время выполнения. Чтобы обезопасить себя, вы должны

149

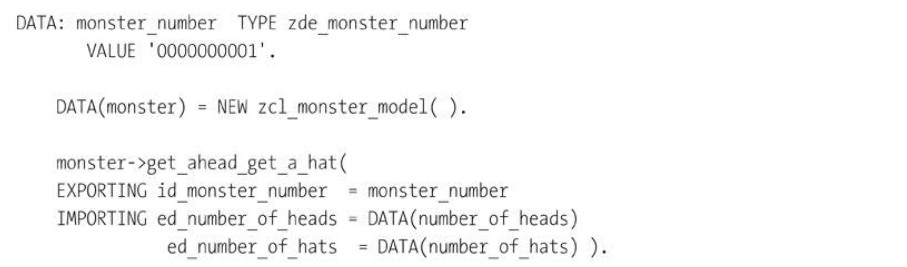
постоянно перескакивать из сигнатуры в сигнатуру, чтобы посмотреть, какие переменные были набраны. Этот процесс утомителен, поэтому Джед создает пользовательские шаблоны, в которых вы вводите имя ФМ, читаете сигнатуру, сигнатура считывается, а затем целый список переменных объявляется с тем же типом, что и все параметры функции. Ну, половина проблемы исчезла, потому что если единственное назначение переменной - получать значение от метода или функции, то переменная может быть объявлена в строке, а тип считывается из определения параметра.

Чтобы увидеть эту новую функциональность, посмотрите на листинг 3.12. Это пример того, что большинство людей всегда делали: объявляли некоторые локальные переменные для передачи в метод, скрестить пальцы и надеяться, что вы объявили переменную так же, как входной параметр. Если нет, вы получите ошибку.



Listing 3.12 Declaring Variables with (Hopefully) Same Types as Method Parameters

Однако в ABAP 7.4 вы можете добиться того же самого, объявив переменные, возвращаемые методом, не в начале процедуры, а в тот момент, когда их значения заполняются методом, как показано в листинге 3.13.



Listing 3.13 Using In line Declarations to Avoid Possible Type Mismatches

150

Последний подход имеет несколько преимуществ:

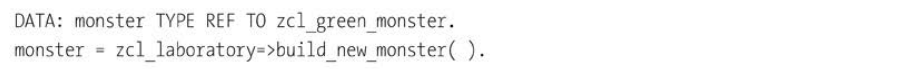
- Меньше строк кода.

- Невозможно получить ошибку несоответствия типов или свалку.

- Если вы измените определение сигнатуры, то переменная результата адаптируется соответствующим образом.

Таким образом, этот подход более компактен и, надеюсь, более удобен для чтения (и, следовательно, сопровождения), более безопасен, более устойчив к изменениям и в целом менее хрупок.

Этот подход проявляется при создании экземпляров объектов с помощью фабричных методов; фабрика будет возвращать различные подклассы базового объекта в зависимости от различной логики, но вызывающую программу это не должно волновать. Например, код до версии 7.4 выглядел бы следующим образом:



В версии 7.4 код будет выглядеть следующим образом:



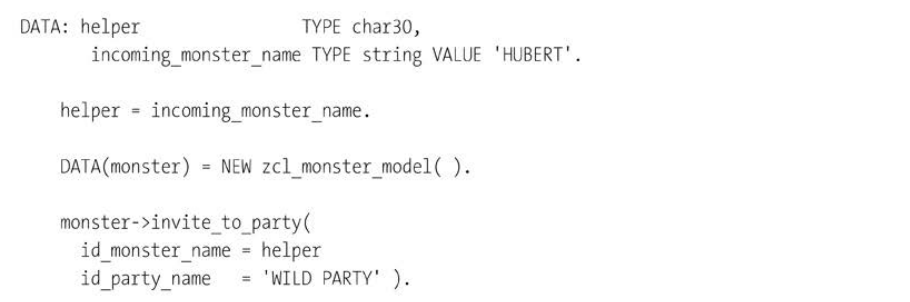
Позже в этой книге вы увидите, что в таких фреймворках, как Web Dynpro (глава 11) и Business Object Processing Framework (BOPF; глава 8), вы регулярно создаете множество объектов со сложными типами данных. Отсутствие необходимости объявлять действительно сложные типы переменных перед вызовом, чтобы заполнить такие переменные значениями, позволяет вам сосредоточиться на том, что действительно важно.

### 3.3.2 Использование операторов конструктора для преобразования строк

Часто одна программа состоит из вызова нескольких более мелких программ (таких как программы FORM, ФМ и методы). Проблема заключается в том, что иногда результат одной процедуры должен быть преобразован, прежде чем его можно будет передать в другую процедуру (например, период из стандартной функции SAP ERP Financials обычно имеет длину два символа, но если вы хотите передать этот период в функцию Controlling, то он должен быть длиной в три символа).

Другой, еще более распространенный пример: часто у вас есть переменная, которая представляет собой строка, и вы хотите передать эту переменную в функцию, которая принимает входные данные только определенного типа (например, CHAR20). Вы не можете передать переменную напрямую; вы должны переместить ее в вспомогательную переменную, как показано в листинге 3.14.

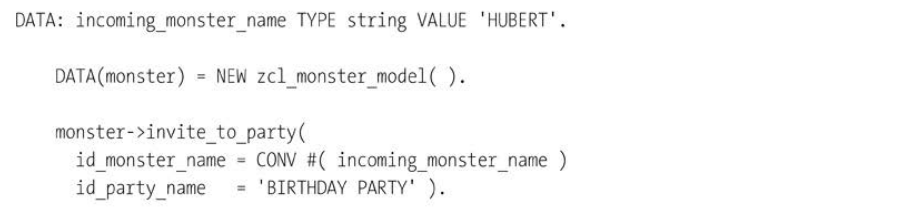
151



Listing 3.14 Moving Variable into Helper Variable

Однако в ABAP 7.4 это может быть упрощено за счет использования специального типа оператора конструктора, CONV, задачей которого является преобразование значений из одного типа в другой. В листинге 3.15 функция CONV считывает целевой тип данных из определения параметра IMPORTING и затем преобразует строку в тип, который ожидает параметр.

И снова это короче, безопаснее и адаптивнее.



Listing 3.15 Converting String with Constructor Variable

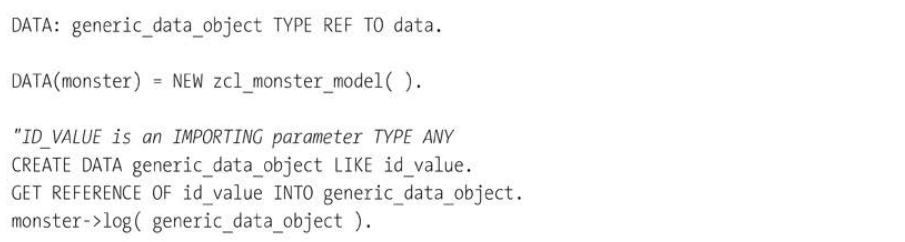
### 3.3.3 Функции, которые ожидают TYPE REF TO DATA

Когда вы импортируете параметры функций или методов, обычно требуется определенный тип параметра. Однако в некоторых ситуациях вы не знаете тип переменной до момента выполнения. В таких случаях единственный способ добиться желаемого - использовать динамическое программирование.

В случаях, когда точный тип данных не известен до выполнения, часто импортируемый параметр метода типизируется как TYPE REF TO DATA. Это происходит, например, когда вы вызываете методы, которые строят динамическую сигнатуру для передачи в динамически определенный метод. Вы также могли использовать параметры, объявленные как TYPE REF TO DATA, когда вам нужно было хранить произвольное количество значений различных данных в журнале вместе с их описаниями.

152

В ABAP версии до 7.4 вы бы сделали что-то вроде кода в листинге 3.16, чтобы удовлетворить требование метода LOG, который ожидает получить параметр типа TYPE REF TO DATA.



Listing 3.16 Filling TYPE REF TO DATA Parameter before 7.4

Однако теперь вы можете избавиться от большей части кода и добиться того же самого, используя оператор конструктора REF и символ хэша (#). Поскольку система времени выполнения знает тип данных VALUE, функция REF# может прочитать его и создать объект data который затем передается в метод протоколирования, ожидающий, что в него будет передан объект TYPE REF TO DATA. Этот процесс немного проще и выглядит следующим образом:



## 3.4 Условная логика

Сейчас уже никого не удивит, что я снова упомяну ZX81. Язык, который использовался в 1981 году, позволял обрабатывать утверждения типа IF (A + B) > ( C + D) THEN ....

Позже, в 1999 году, когда я начал программировать на ABAP, мне не хватало этой способности. К счастью, с появлением ABAP 7.02, к которому я впервые получил доступ в 2012 году, я снова смог делать подобные вещи. (Возможно, им потребовался тридцать один год, чтобы придумать, как это сделать, но ожидание того стоило).

В ABAP 7.40 конструкции IF/THEN и CASE, которые вы знаете и любите, становятся все проще. В следующих разделах мы объясним, как это сделать.

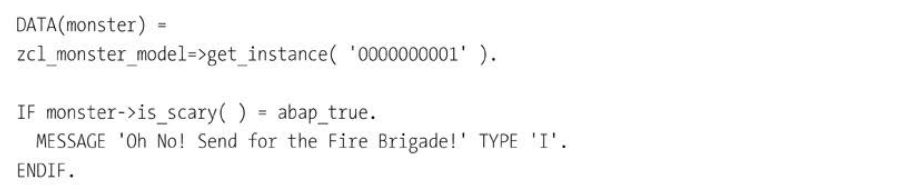
### 3.4.1 Опуская ABAP\_TRUE

Когда функциональные методы были введены в ABAP, часть идеи заключалась в том, что это что это сделает код более похожим на английский. С годами консенсусом

153

среди программистов ABAP как внутри, так и за пределами SAP было то, что если вы создаете метод, который возвращает значение, говорящее, истинно ли что-то или нет, то возвращаемые параметры должны быть типизированы как ABAP\_BOOL.

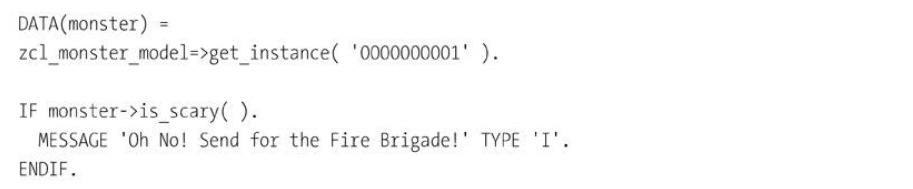
Например, метод ZCL\_MONSTER->IS\_SCARY должен возвращать ABAP\_TRUE, если монстр действительно страшен, но ABAP\_FALSE, если он не настолько чудовищен, как должен быть. Пока все хорошо. Однако, как показывает листинг 3.17, в Дании что-то прогнило.



Listing 3.17 ABAP\_TRUE

Зачем вам нужно = ABAP\_TRUE в конце? Это не делает предложение более читабельным, а только удлиняет его. Как скажет вам любой учитель английского языка, добавление в предложение слов, которые не меняют смысла предложения, только делает вас многословным. Ответ заключается в том, что вы должны были сделать это, потому что иначе проверка синтаксиса была бы неудачной.

Однако, начиная с версии 7.4 (SP 8), вы можете сделать то, что ожидали, и опустить ABAP\_TRUE, как показано в листинге 3.18.



Listing 3.18 Omitting ABAP\_TRUE

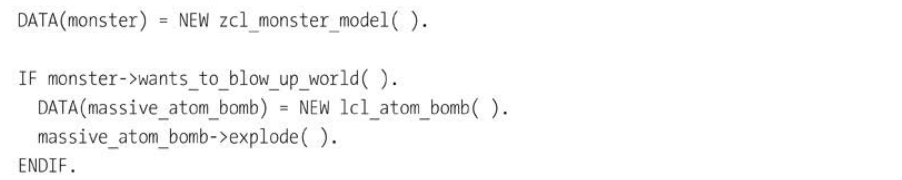
В листинге 3.18 с технической точки зрения происходит то, что если вы ничего не указываете после функционального метода, компилятор оценивает его как *IS\_PRODUCTION( ) IS NOT INITIAL*. Значение ABAP\_TRUE на самом деле является буквой X, поэтому результат не является начальным, и поэтому утверждение разрешается как истинное.

Мнения о том, хорошо ли иметь истинный булевский тип данных в языке программирования, разделились. SAP говорит, что нет, а создатели всех других языков программирования считают, что это хорошо.

154

Именно поэтому в ABAP существуют обходные пути, подобные этому.

Тем не менее, вы должны быть очень осторожны при использовании этого синтаксиса; он имеет смысл только тогда, когда функциональный метод передает обратно параметр, типизированный как ABAP\_BOOL. В качестве примера, рассмотрим код в листинге 3.19.



Listing 3.19 IF Statement without Proper Check

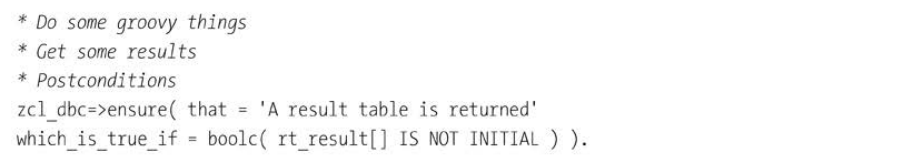
Если функциональный метод в листинге 3.19 возвращает строку, и эта строка – “NO NO NO! Не взрывайте мир, что бы вы ни делали, не взрывайте мир”, то результат NOT INITIAL и, следовательно, оценивается как true, а значит, это занавес для всех нас.

### 3.4.2 Использование XSDBOOL в качестве обходного пути для BOOLC

Другая распространенная ситуация, связанная с булевой логикой (или ее отсутствием) в ABAP - это случай, когда вы хотите отправить значение TRUE/FALSE в метод или получить такое значение обратно от функционального метода. В ABAP вы не можете просто сказать что-то вроде следующего:



Но в некоторых языках программирования вы можете сделать именно это (можете ли вы угадать, какой компьютер мог это сделать в 1981 году). Опять же, у нас есть обходной путь в виде встроенной функции BOOLC (листинг 3.20).

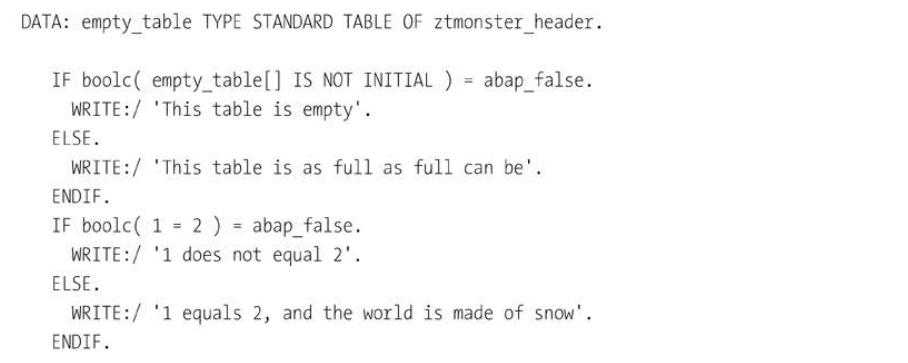


Listing 3.20 BOOLC

155

В листинге 3.20 вы передаете значение TRUE/FALSE, основанное на том, есть ли записи во внутренней таблице. Это работает нормально.

Но что, если вы хотите проверить отрицательное значение с помощью этого метода? Скажем, например, вы хотите передать в параметр IF\_TRUE значение TRUE/FALSE, которое будет истинным, если таблица пуста. Если вы используете предыдущий метод с использованием BOOLC, то все начинает идти ужасно неправильно. Это можно продемонстрировать, выполнив код в листинге 3.21.



Listing 3.21 Testing for Negative

При выполнении этого кода вывод будет следующим:

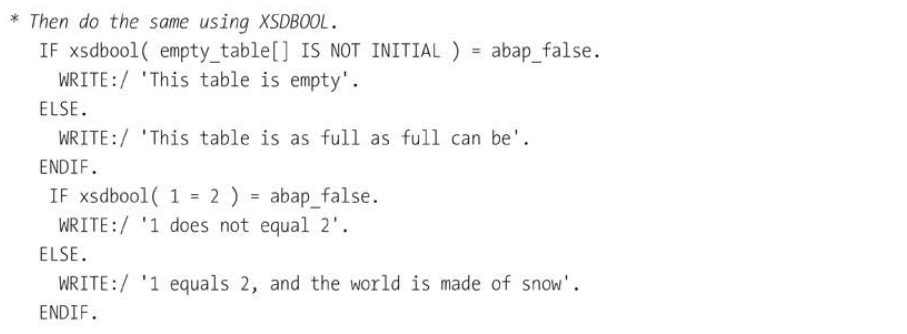
1. Эта таблица заполнена настолько, насколько это возможно.

2. 1 равно 2, и мир состоит из снега.

О, Боже! Причиной такого результата является фундаментальный недостаток конструкции встроенной функции BOOLC. Вместо того чтобы возвращать односимвольное поле, определенное так же, как ABAP\_BOOL, она возвращает строку. Если строка является X(TRUE), то все в порядке, но в ABAP сравнение строки из одного пустого символа с пустым символом внутри ABAP\_FALSE означает, что сравнение не удастся, даже если значения идентичны.

Поэтому, учитывая, что о настоящей булевой переменной по каким-либо причинам не может быть и речи, необходимо новое обходное решение. Исправить BOOLC так, чтобы она возвращала значение ABAP\_BOOL, было бы слишком просто, поэтому в ABAP 7.4 была добавлена новая встроенная функция под названием XSDBOOL, которая делает то же самое, что и BOOLC, но возвращает параметр типа ABAP\_BOOL. Пример ее использования можно увидеть в листинге 3.22. Функция не была придумана для этой цели, но она работает, а это главное.

156



Listing 3.22 Using XSDBOOL for Correct Logic Test Results

Когда вы запустите этот код, результат будет следующим:

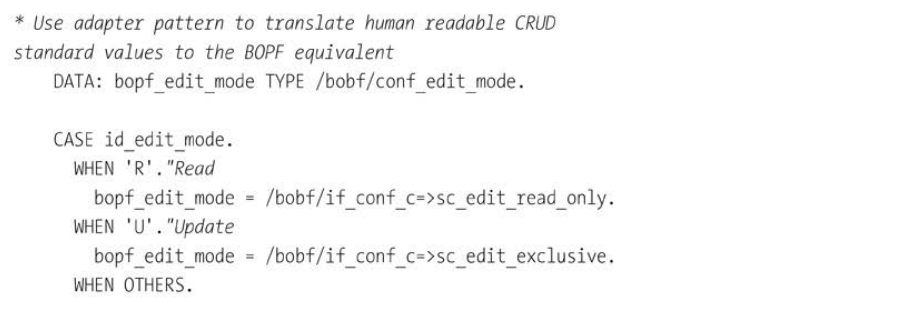
1. Эта таблица пуста.

2. 1 не равно 2.

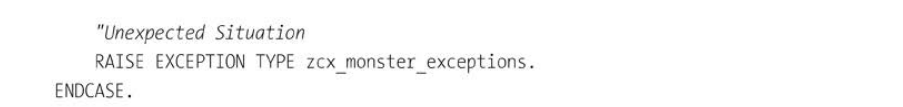
Еще лучше: если вы используете неправильный (BOOLC) в ABAP в Eclipse, вы получаете предупреждение, которое говорит вам использовать XSDBOOL вместо этого.

### 3.4.3 Выражение SWITCH как замена CASE

Сколько раз вы видели код, подобный приведенному в листинге 3.23? Здесь вы получаете одно значение и используете оператор CASE для перевода этого значения. Проблема заключается в том, что вам необходимо постоянно упоминать, какую переменную вы заполняете в каждой ветви вашего оператора CASE оператора.

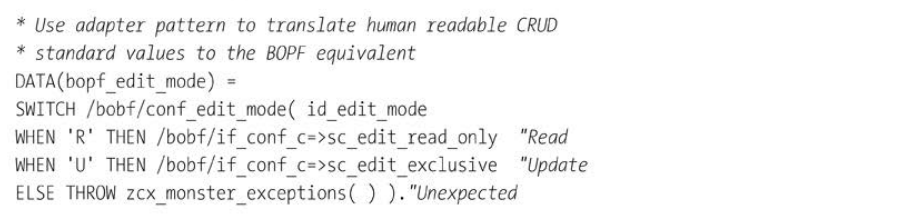


157



Listing 3.23 Filling in Variable Using CASE Statement

Как упоминалось в коде, это паттерн адаптера, очень распространенный в ОО-программировании. В 7.4 это можно немного упростить, используя новый оператор конструктора SWITCH, как показано в листинге 3.24.



Listing 3.24 Filling in Variable Using SWITCH Statement

Как видно из этого примера, определение данных для BOPF\_EDIT\_MODE (/bobf/conf\_edit\_mode в данном случае) переместилось в тело выражения, что значительно сокращает количество необходимых строк кода. Кроме того, поклонники Java будут прыгать от радости, увидев, что вместо ABAP-термина RAISE EXCEPTION TYPE у нас теперь есть эквивалентный Java-термин THROW. Однако его использование идентично; компилятор оценивает ключевые слова RAISE EXCEPTION TYPE и THROW как одно и то же. В качестве дополнительного бонуса, это действительно имеет больше грамматического смысла, потому что THROW и CATCH сочетаются лучше, чем RAISE EXCEPTION TYPE и CATCH. (Повезло, что классы исключений должны начинаться с CX, иначе какой-нибудь остроумный программист из SAP создал бы класс исключений под названием UP).

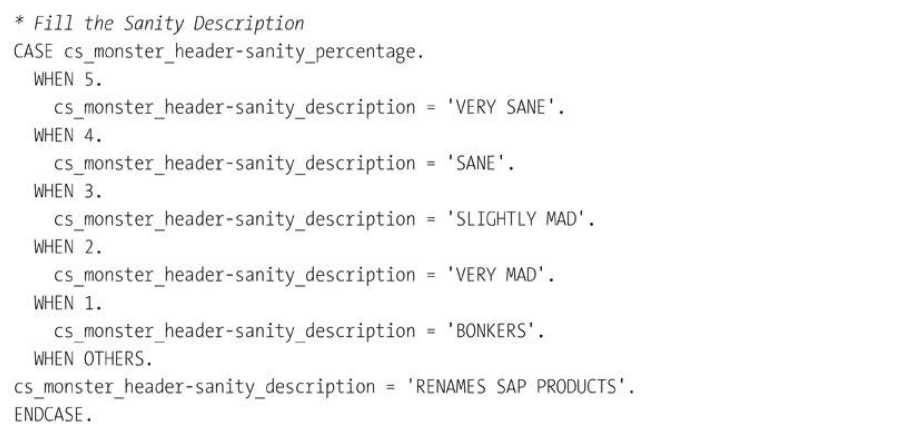
### 3.4.4 Выражение COND как замена IF/ELSE

На протяжении всей этой книги вы будете читать о том, что сопротивление изменениям - это плохо. Несмотря на это, давайте признаем это: Перемены могут раздражать, особенно когда они затрагивают ваш код. Одним из ярких примеров этого факта является кодирование оператора CASE, основанного на предположении, что одно значение выводится из значения другого - до того, как кто-то приходит и говорит вам, что правила изменились и последнее значение в CASE-выражении истинно, только если это вторник. В среду значение становится чем-то другим.

158

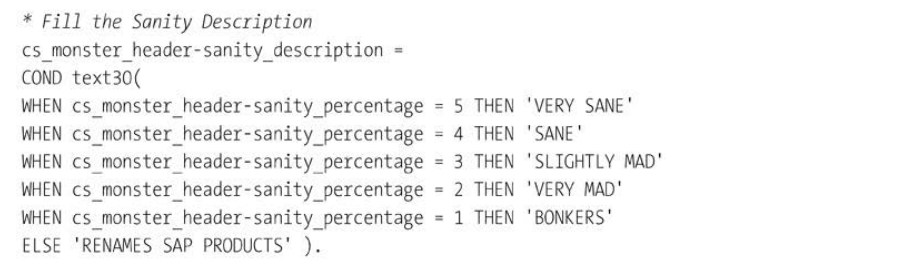
Операторы CASE могут оценивать только одну переменную за раз, поэтому в данном примере вам придется изменить всю конструкцию на IF/ELSE. Это не конец цивилизации, как мы ее знаем, но чем больше изменений вам придется сделать, тем больше риск.

Допустим, вы очень боитесь, что логика, которую вам дали, может измениться в какой-то момент в будущем, но, тем не менее, вы начинаете с CASE, как в коде в листинге 3.25, который является кодом до версии 7.4, оценивающим описание вменяемости монстра на основе числового значения.



Listing 3.25 CASE Statement to Evaluate Monster Sanity

В 7.4 вы можете добиться того же самого, но более компактным способом, используя оператор конструктора COND. Это также означает, что вам не нужно постоянно указывать целевую переменную снова и снова (см. листинг 3.26).



Listing 3.26 Using COND Constructor Operator

159

Это выглядит точно так же, как оператор CASE, и единственное преимущество изменения на данном этапе заключается в том, что оно немного компактнее. Однако, когда бизнес решит, что вам нужно учитывать день, чтобы сказать, является ли монстр сумасшедшим или нет, вы можете просто изменить часть конструкции COND. В ситуации до версии 7.4 вам приходилось отказываться от всей идеи оператора CASE и переписывать все в виде конструкции IF/ELSE. На сайте Единственное изменение, необходимое для логики COND, показано в листинге 3.27.



Listing 3.27 COND Constructor Operator with Updated Logic

## 3.5 Внутренние таблицы

Внутренние таблицы - это хлеб и масло программирования в ABAP. Многие другие языки, например Java, могут только мечтать о таких таблицах; им приходится иметь дело с различными массивами, стеками, хэш-браунами и т.п. В этом разделе вы узнаете о некоторых новых функциональных возможностях ABAP, связанных с внутренними таблицами.

### 3.5.1 Рабочие области (Work Areas) таблиц

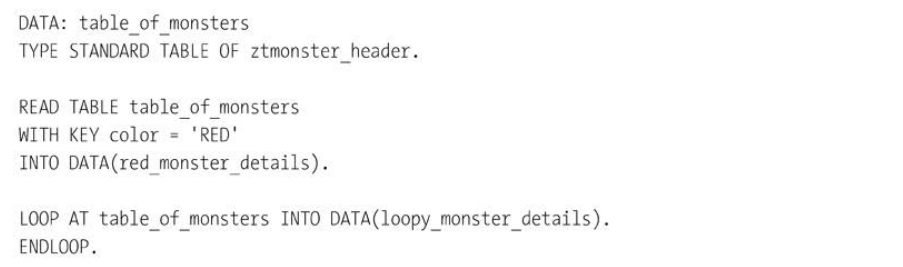
Некоторое время назад SAP постановила, что строки заголовков во внутренних таблицах - это работа дьявола. Это было связано с тем, что использование строк заголовков приводило к существованию в вашей программе двух объектов данных с одинаковыми именами, а это считалось запутанным. Например, нередко можно было встретить программу, в которой переменная ITAB относилась к внутренней таблице в целом, а рабочая область под названием ITAB относилась к текущей строке таблицы. Очевидно, что это неправильно с академической точки зрения, но в реальной жизни программисты ABAP настолько привыкли к идее,

160

что от этой привычки было трудно избавиться. Это было особенно верно, потому что приходилось явно объявлять переменную в качестве строки заголовка. Например:



Хорошей новостью является то, что теперь вы можете избежать дополнительного объявления переменной и по-прежнему иметь две переменные с разными именами, одну для таблицы и одну для рабочей области. В листинге 3.28 показан синтаксис для чтения в рабочей области и циклического просмотра через таблицу в релизе 7.4.



Listing 3.28 Reading into Work Area and Looping through Table

В разделе 3.1.1 вы узнали, что, начиная с версии 7.4, вам больше не нужно выполнять объявление DATA для элементарных типов данных. Точно так же обстоит дело и с рабочими областями, которые, конечно же, являются структурами. Если вы выполняете цикл в рабочей области MONSTER\_RECORD и структура рабочей области не соответствует типу таблицы TABLE\_OF\_MONSTERS, вы увидите синтаксическую ошибку.

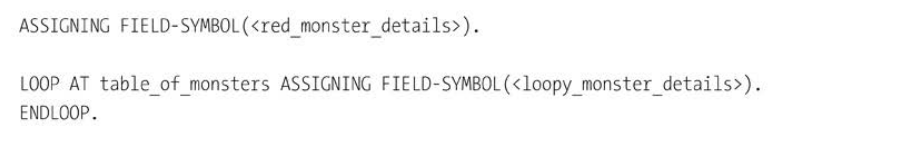
Таким образом, компилятор знает, какой тип ему нужен - так почему же вы должны сообщать ему этот тип в явном объявлении данных? Ответ заключается в том, что теперь это не так, и это изменение уберет все те дополнительные строки объявления данных, которые пришлось вставить, когда все отошли от строк заголовков (если вообще отошли) и вместо них стали явно объявлять рабочие области.

Точно так же, как вам больше не нужны объявления DATA для рабочих областей таблицы, вам также больше не нужны объявления FIELD-SYMBOL для (распространенных) ситуаций, когда вы хотите изменить данные в рабочей области во время цикла по внутренней таблице.

В версии 7.4, если вы хотите использовать field symbols для рабочей области, то синтаксис выглядит следующим образом показан в листинге 3.29.



161



Listing 3.29 Field Symbols for Work Area

Следует отметить один очень важный момент: если вы объявите рабочую область, а затем выполните цикл через внутреннюю область, то после завершения цикла рабочая область будет заполнена значением последней строки внутренней таблицы, которая была обработана. Однако если вы выполняете цикл через внутреннюю таблицу в рабочей области, созданной с помощью оператора DATA, то вы можете задаться вопросом, не прекращает ли рабочая область свое существование после выхода из цикла.

С академической точки зрения невозможность доступа к рабочей области за пределами цикла - это хорошо, потому что во всех учебниках говорится, что переменные не должны существовать за пределами своей области (в данном случае цикла). Действительно, в SAP такое поведение очень желательно. *К сожалению, факт заключается в том, что переменная существует после выхода из цикла.*

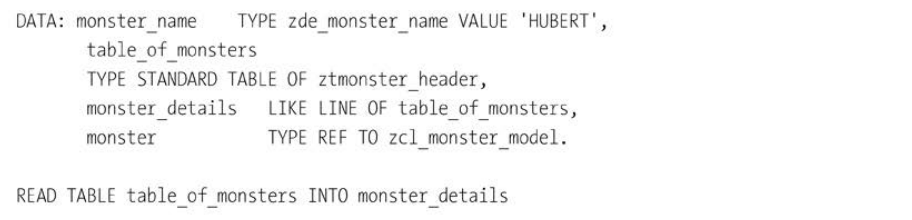
SAP обошла эту проблему, опубликовав руководство, в котором говорится, что если вы обращаетесь к такой переменной вне ее области видимости, то вы очень непослушны, и сотрудники SAP скажут "tut, tut", если узнают об этом. (Лично я не уверен, что эта страшная угроза будет иметь большой эффект, но я подумал, что должен сообщить вам об этом).

### 3.5.2 Чтение из таблицы

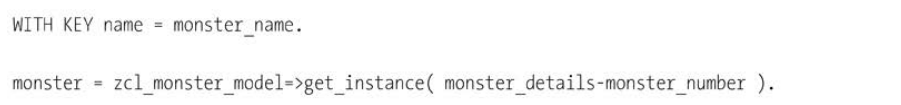
Если я скажу вам, что вам больше никогда не придется использовать оператор READ TABLE для получения строки из внутренней таблицы, то, возможно, в вашем мире наступит конец света.

На самом деле, именно это и происходит в ABAP 7.4. Вы можете вообще убрать ключевые слова и заменить английскую фразу READ TABLE на более компьютерную пару квадратных скобок ([ ]).

До версии ABAP 7.4 вызов строки из внутренней таблицы выглядел бы как в листинге 3.30.

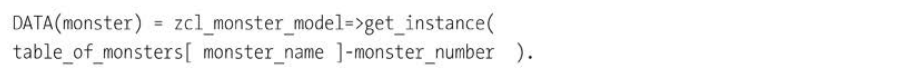


162



Listing 3.30 Reading Line from Internal Table before 7.4

Однако в ABAP 7.4 и выше это упрощено, как показано в листинге 3.31.



Listing 3.31 Reading Line from Internal Table after 7.40

Примечание

Некоторые скажут, что если делать код менее похожим на английский и более похожим на машинный код, то это мешает ясности. Например, если после прочтения строки внутренней таблицы вы захотите добавить ее в строку и передать строку в метод, то вы могли бы переборщить с используя знаки препинания в коде. Та часть, где вы читаете внутреннюю таблицу, окажется затерянной в толще кода, и кто-то, читая код, может с трудом понять, что происходит:



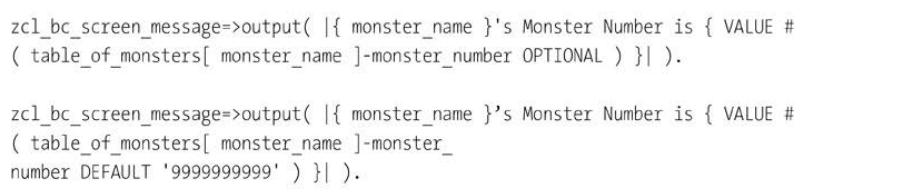
Многие люди говорят одно и то же о регулярных выражениях: они выполняют работу в очень небольшом объеме кода, но никто не может понять этот код.

В реальной жизни наиболее распространенное использование нового способа чтения внутренних таблиц заключается в следующем читать первую строку такой таблицы следующим образом:



Что произойдет, если такое выражение не сможет найти нужную вам строку внутренней таблицы? Ответ - ничего хорошего; возникает исключение, тогда как вы, вероятно, ожидали возврата пустого (начального) значения. В SAP, должно быть, поступило много жалоб по этому поводу, поэтому начиная с версии 7.4 (SP 8) SAP предложила обходной путь: если вы добавите слово *OPTIONAL* в конце запроса, то внезапно системе станет все равно, что не найдено ни одной записи, и будет возвращено начальное значение любого типа данных, который вы ищете, что и произошло бы при традиционной настройке *READ TABLE xyz INTO work\_area\_abc*, когда чтение не удалось, а рабочая область еще не была заполнена. Пример такого обходного пути показан в листинге 3.32.

163



Listing 3.32 Reading Internal Table with OPTIONAL/DEFAULT

Добавление *DEFAULT* заполняет жестко закодированное значение при неудачном чтении. Затем вы можете проверить, что чтение не удалось, сравнив результат с этим жестко закодированным значением так же, как если бы SY-SUBRC равнялся 4 при проверке успешности традиционного чтения внутренней таблицы.

### 3.5.3 CORRESPONDING для обычных внутренних таблиц

Вы, несомненно, часто используете ключевое слово ABAP *MOVE-CORRESPONDING*, перемещая значения переменных из одной структуры в другую.

Внимание: Хьюстон, у нас проблема

В некоторых отдельных документах SAP рекомендуется вообще не использовать MOVE-CORRESPONDING, скорее всего, чтобы избежать проблемы перемещения строк в числовые поля, о которой говорилось в начале этого раздела. Однако если вы все же используете его, то рассмотренная здесь функциональность должна быть вам полезна.

Новым в 7.4 является оператор-конструктор *CORRESPONDING* без слова move впереди перед ним. Этот оператор выводит перемещение данных между двумя внутренними таблицами на совершенно новый уровень. Допустим, у вас есть две внутренние таблицы, заполненные данными о монстрах, но они определены с разным набором столбцов. Вы хотите скопировать поля с одинаковыми именами из одной таблицы в другую, за двумя исключениями:

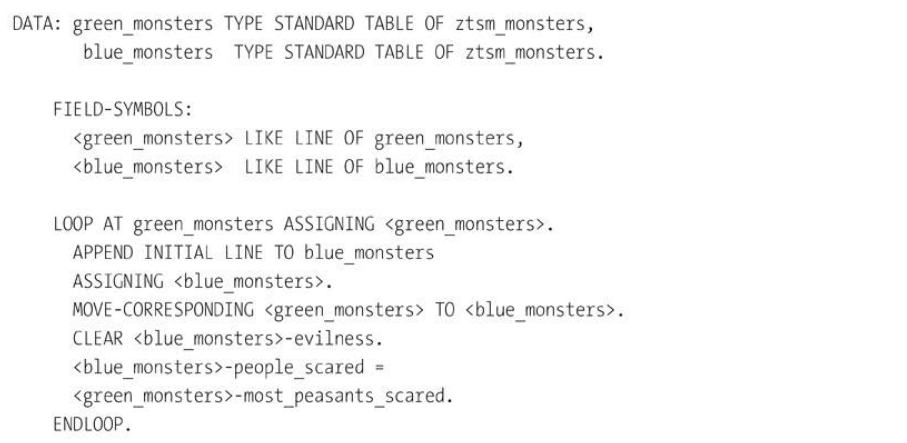
1. Вы не хотите копировать значение EVILNESS из одной таблицы в другую, хотя в обеих таблицах есть столбец EVILNESS.

2. Вы хотите скопировать столбец с именем MOST\_PEASANTS\_SCARED из одной таблицы в аналогичный столбец с именем PEOPLE SCARED из другой таблицы.

До версии 7.4 вы должны были объявить кучу вспомогательных переменных для хранения рабочих областей двух таблиц, а затем в цикле просматривали первую таблицу, перемещая все из одной

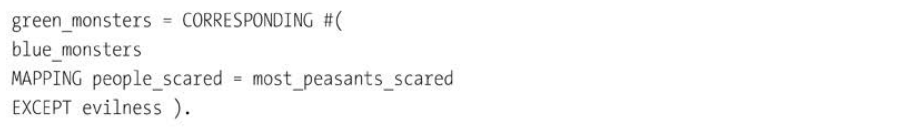
164

таблицы в другую. Затем вы выполните некоторую логику для обработки исключений (см. Листинг 3.33).



Listing 3.33 Moving One Table to Another before 7.4

В версии 7.4 вы можете использовать оператор конструктора *CORRESPONDING*, чтобы сделать то же самое, но на этот раз вам не нужно объявлять символы полей. Вы сообщаете оператору *CORRESPONDING* правила, и он прокручивает для вас таблицу, выполняя эти правила. выполняя эти правила. Более того, вам даже не нужно определять целевую таблицу; если вы поставите # после оператора CORRESPONDING, то он создаст целевую таблицу с теми же столбцами, что и в исходной таблице, но без столбца EVILNESS и со столбцом SCARED имеющее другое имя, но тот же тип, что и столбец источника, как показано в листинге 3.34.



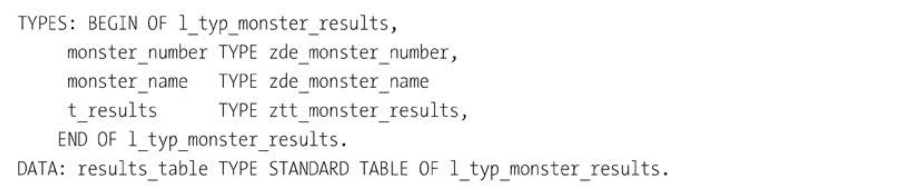
Listing 3.34 Moving One Table to Another in 7.4

### 3.5.4 MOVE-CORRESPONDING для внутренних таблиц с глубокими структурами

В ABAP глубокая структура - это не структура, которая много думает и обладает сложной индивидуальностью, а скорее структура, в которой помимо элементарных типов данных, таких как

165

строки и числа - имеет внутренние таблицы. В листинге 3.35 показан пример глубокой структуры. T\_RESULTS определен как тип таблицы, поэтому у вас есть внутренняя таблица, в которой каждая строка имеет столбец, который сам является внутренней таблицей.

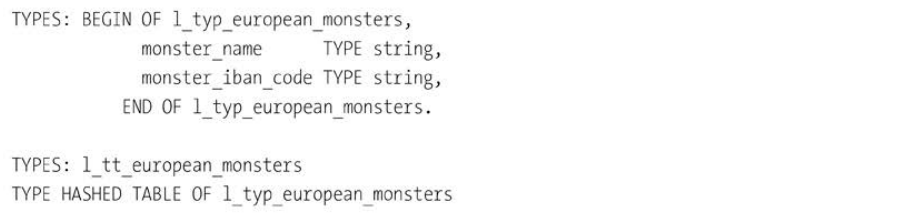


Listing 3.35 Deep Structure

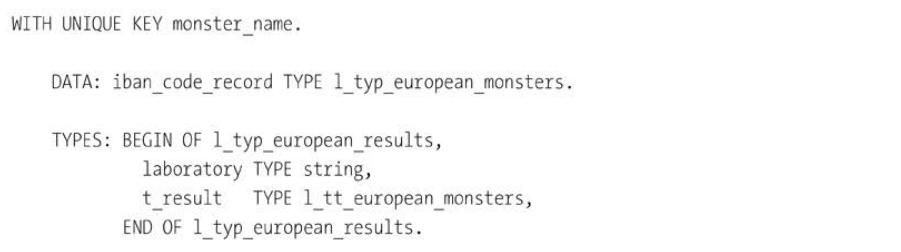
Недавно компания SAP явно задумалась о том, что происходит, когда вы выполняете маневр MOVE-CORRESPONDING между двумя немного разными структурами, которые обе являются глубокими, но определены немного по-разному. Как вы знаете, MOVE-CORRESPONDING сравнивает имена полей двух структур, и если поле под названием RESULT является строкой в одной структуре и числом в другой, а в первой структуре значение равно XYZ, то возникают проблемы (т.е. короткий дамп). Сделайте еще один шаг вперед: Допустим, у вас есть две внутренние таблицы в двух глубоких структурах с одинаковыми именами, но определенными по-разному. Что произойдет, когда вы выполните MOVE-CORRESPONDING?

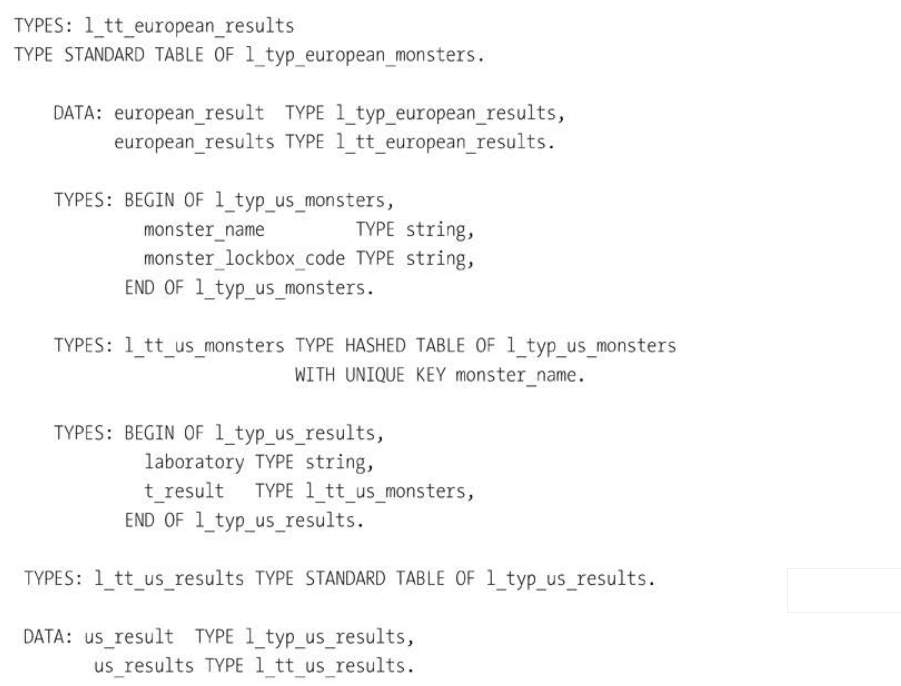
В следующих примерах рассматривается, что происходит в релизе 7.02 при перемещении из одной структуры в другую, а затем мы рассмотрим, как этот же процесс ведет себя в 7.4. Спойлер: В 7.4 вы можете выполнять MOVE-CORRESPONDING между внутренними таблицами в целом, что было невозможно в более ранних версиях.

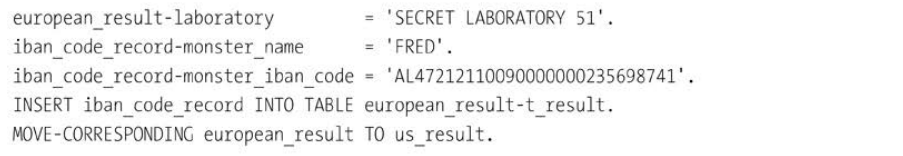
Скажем, в Европе важно отслеживать нечто, называемое IBAN-кодом для каждого монстра, который не имеет смысла нигде в мире, а в Соединенных Штатах важно отслеживать код IBAN для каждого банка. США важно отслеживать код LOCKBOX для каждого монстра, который также не имеет смысла что также бессмысленно в любой другой точке мира. Листинг 3.36 устанавливает структуры для хранения данных для каждого региона, а затем пытается использовать MOVE-CORRESPONDING для перемещения некоторых европейских данных о монстрах из Европы в эквивалент из США.



166







Listing 3.36 MOVE-CORRESPONDING Attempt

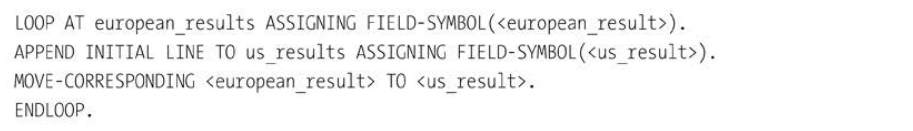
167

Как вы думаете, что происходит? В структуре монстра США в таблице результатов поле LOCKBOX заполнено кодом IBAN из европейского эквивалента, потому что поле T\_RESULT есть и в европейской, и в американской структуре.

Обычно это не то, что вам нужно. Для идентичных таблиц это всегда можно обойти, используя TABLE\_ONE[] = TABLE\_TWO[], что меняет первую таблицу на идентичную копию второй таблицы. Однако это не работает в случаях, когда, скажем, у вас есть таблица базы данных с десятью полями и внутренняя таблица с этими десятью полями плюс еще пять полей с текстовыми описаниями или вычисляемыми полями. Вместо этого вам нужно считывать таблицу базы данных в одну внутреннюю таблицу, содержащую только десять полей, и циклически просматривать эту первую внутреннюю таблицу, перемещая соответствующие элементы в какую-то вторую выходную таблицу, где вы заполняете дополнительные данные, такие как названия офисов продаж и тому подобное, а также вычисляемые поля.

Таким образом, вместо кода, показанного в листинге 3.37, вы бы использовали код, показанный в листинге

3.38.



Listing 3.37 Copying between Internal Tables with Different Structures before 7.4



Listing 3.38 Copying between Internal Tables with Different Structures in 7.4

Это сэкономит вам несколько строк кода и прекрасно подходит для случаев, когда структуры строк двух таблиц плоские. Однако это все равно не решает проблему для глубоких структур. Как вы уже видели, MOVE-CORRESPONDING сам по себе сбрасывает содержимое внутреннего компонента таблицы, например, T\_RESULT, в идентично названный компонент целевой структуры. Если T\_RESULT в целевой структуре имеет столбцы с разными именами, то могут возникнуть самые разные странные результаты. Для борьбы с этим SAP разработал два новых дополнения к MOVE-CORRESPONDING в версии 7.4 (плюс комбинацию этих двух функций):

- *MOVE-CORRESPONDING EXPANDING NESTED TABLES (С РАСШИРЕНИЕМ ВЛОЖЕННЫХ ТАБЛИЦ)*

В MOVE-CORRESPONDING EXPANDING NESTED TABLES все, что может находиться в целевой внутренней таблице, уже удаляется. Столбцы с простыми значениями, например, числа, копируются в их идентично названные поля. Однако, когда дело доходит до

168

сложных столбцов, таких как T\_RESULT, копируются только те поля, которые имеют одинаковое имя столбца во вложенных таблицах. Например, MONSTER NAME копируется, потому что в целевой таблице есть столбец с эквивалентным именем, а MONSTER\_IBAN\_CODE - нет, потому что нет столбца с таким же именем, только LOCKBOX.

- *MOVE-CORRESPONDING KEEPING TARGET LINES (СОХРАНЕНИЕ ЦЕЛЕВЫХ ЛИНИЙ)*

В MOVE-CORRESPONDING KEEPING TARGET LINES происходит довольно странная вещь: если в целевой внутренней таблице уже что-то есть, то оно остается там. Затем в конце целевой таблицы добавляются дополнительные строки - то же самое произошло бы, если бы вы просто выполнили MOVE-CORRESPONDING между двумя таблицами. Это похоже на APPEND LINES OF tablel TO table2, за исключением того, что две таблицы имеют разные структуры, и в них встречаются только идентично названные компоненты.

- *MOVE-CORRESPONDING EXPANDING NESTED TABLES KEEPING TARGET LINES* *(С СОХРАНЕНИЕМ ЦЕЛЕВЫХ СТРОК)*

Как можно представить, MOVE-CORRESPONDING EXPANDING NESTED TABLES KEEPING TARGET LINES выполняет обе предыдущие задачи сразу; он действует как APPEND LINES OF tablel TO table22, копируя только идентично названные компоненты в новые строки, но также немного умничает с любыми внутренними компонентами таблицы и копирует только идентичные компоненты внутри этих структур.

### 3.5.5 Динамический MOVE-CORRESPONDING

SAP может быть похож на дракона с двумя головами, каждая из которых говорит вам противоположное чем другая. В версии 7.4 было рекомендовано не использовать MOVE-CORRESPONDING вообще, но вам были предоставлены новые инструменты, с помощью которых это можно сделать. В 7.5 эта дихотомия становится еще более экстремальной; с одной стороны, вам советуют не использовать динамическое программирование (потому что оно не разрешено в некоторых сценариях SAP S/4HANA), и все же новый, динамический класс MOVE-CORRESPONDING предоставлен вам.

В примерах в разделе 3.5.3 вы могли бы жестко закодировать правила сопоставления полей с разными именами для копирования друг друга - например, двенадцать тысяч различных имен, которые SAP присваивает SY-MSGNO в различных структурах.

Теперь представьте, что такое отображение из действительно большой структуры в меньшую может меняться в зависимости от условий во время выполнения. Это может показаться довольно эзотерическим требованием, но нам не нужно слишком много думать, чтобы найти пример из реальной жизни; в конце концов, примеры с монстрами - это реальные проблемы, с которыми приходится сталкиваться людям, с измененными именами для защиты невинных.

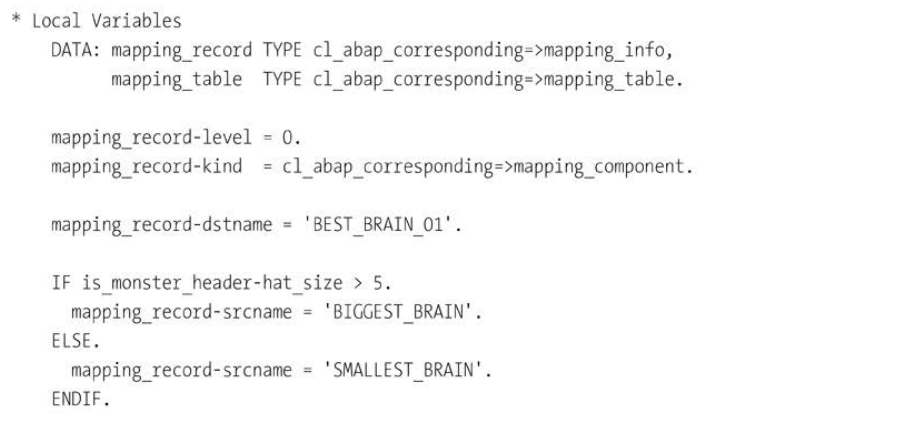
В следующем примере, как всегда, задача заключается в создании монстра. В таблице конфигурации есть около двадцати четырех различных видов преступных мозгов, которые можно использовать для заполнения

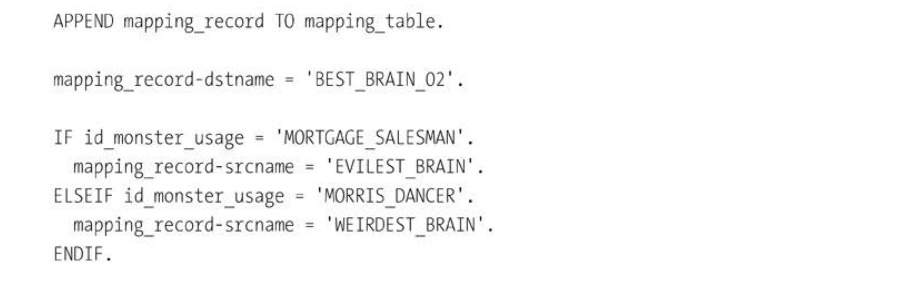
169

внутри головы монстра. Вы хотите сузить список до четырех лучших вариантов, определяемых во время выполнения, которые передаются в меньшую структуру для дальнейшей обработки.

Обе структуры начинаются с номера и имени монстра. Большая структура имеет список полей, описывающих серийные номера различных доступных мозгов, а меньшая структура имеет поля в формате BEST\_BRAIN\_XX.

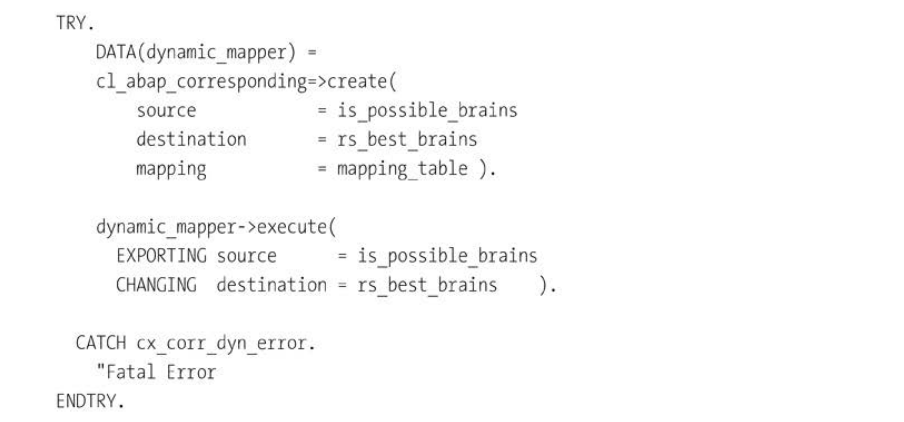
Вы хотите переместить поля с одинаковыми именами из большой структуры в маленькую структуру, а затем динамически определить, какие из возможных мозгов из конфигурационной структуры следует переместить в меньшую структуру. Вы добьетесь этого, заполнив таблицу отображения, предписывающую, какие значения копировать между разноименными полями, вызвав два метода из CL\_ABAP\_CORRESPONDING для создания объекта отображения, а затем выполнив само отображение, как показано в листинге 3.39.





170



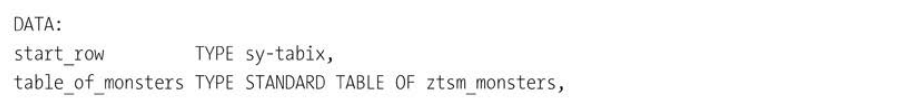


Listing 3.39 Dynamic MOVE-CORRESPONDING Usage

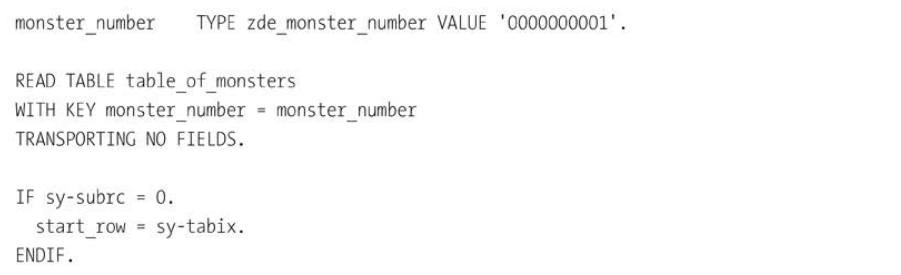
Вы также можете использовать параметр *EXCEPT* в качестве входа в параметр KIND в записи отображения, чтобы остановить перемещение некоторых одинаково названных полей; параметр *LEVEL* используется при отображении глубоких структур. В примере вы могли бы достичь того же того же самого, используя прямое присваивание с меньшим количеством строк кода, но этот подход становится выгодным только в том случае, если у вас действительно сложные структуры с большим количеством полей. Традиционно для решения такого рода задач использовались field-symbols. SAP считает, что использование этого нового класса облегчает чтение кода.

### 3.5.6 Новые функции для общих задач внутренних таблиц

При работе в коде ABAP бывают случаи, когда необходимо точно знать, в какой строке внутренней таблицы находятся интересующие вас данные. Чтобы найти эту информацию в ABAP до версии 7.4, вам пришлось бы написать код, в котором была бы объявлена вспомогательная переменная, чтобы хранить номер строки целевой записи, читать таблицу без какой-либо другой цели, кроме как найти этот номер строки, а затем передать системную переменную, содержащую результат чтения таблицы в вашу вспомогательную переменную, как показано в листинге 3.40.

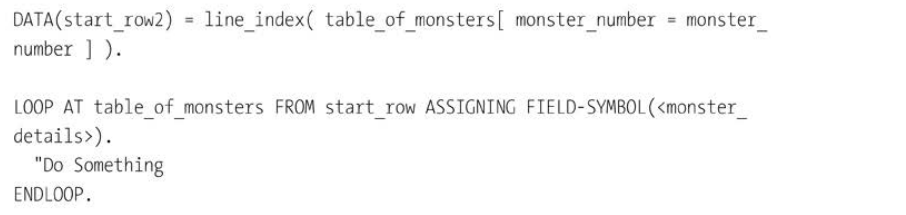


171



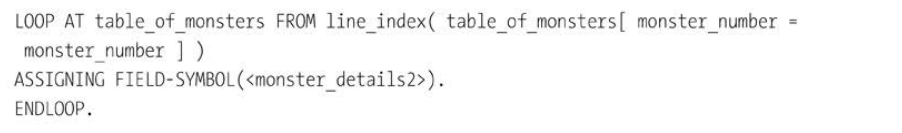
Listing 3.40 Reading Internal Table to Get Row Number

Логика из листинга 3.40 часто встречается при обработке вложенных циклов - вы выполняете такой код с помощью *LOOP AT itab STARTING AT ld\_tabix* - но также и во множестве других случаев использования. В любом случае, в релизе 7.4 это можно упростить, используя встроенную функцию *LINE\_INDEX*, которая выполняет точно такую же задачу, но без необходимости смотреть на значения SY-SUBRC и SY-TABIX (STARTING AT LINE INDEX и т.д.). Этот новый подход показан в листинге 3.41.



listing 3.41 LINE\_INDEX

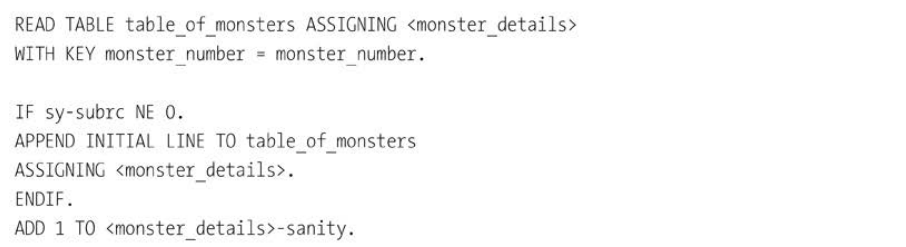
На протяжении всей этой главы одной из целей было избавиться от вспомогательных переменных. Поскольку *LINE\_ INDEX* является так называемой встроенной функцией, ее можно использовать в позиции операнда, что позволяет отказаться от вспомогательной переменной START\_ROW. При выполнении цикла для внутренней таблицы из строки, ранее хранившейся в START\_ROW, используйте код, показанный в листинге 3.42.



listing 3.42 Built-In Function LINE\_INDEX at Operand Position

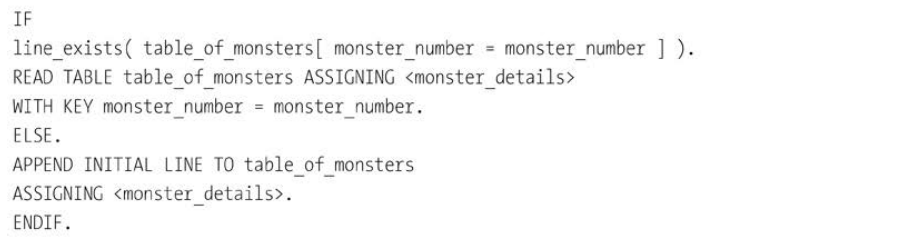
172

У этой новой встроенной функции также есть друг: *LINE\_EXISTS*. Допустим, вы хотите посмотреть, есть ли во внутренней таблице запись для рассматриваемого монстра. Если да, то вы хотите изменить существующую запись, если нет, то добавить новую. До версии 7.4 это делалось так: сначала нужно было прочитать внутреннюю таблицу, чтобы узнать, есть ли в ней запись о монстре. Если системных переменных не было, то SY-SUBRC не был равен нулю. Вы реагировали на это добавлением новой записи в таблицу для вашего монстра, как показано в листинге 3.43.



Listing 3.43 Checking If InternaI Table Line Exists before 7.4

Добавив новую встроенную функцию *LINE\_EXISTS*, код можно изменить, как показано в листинге 3.44.



Listing 3.44 Checking If Internal Table Line Exists in 7.4

"Почему это лучше?" слышу я ваш возглас. Во-первых, это делает более очевидным то, чего вы пытаетесь достичь. Кроме того, использование *LINE\_EXISTS* вместо SY-SUBRC более надежно; SY-SUBRC может быть сомнительным, потому что вы никогда не знаете, когда кому-то придет в голову светлая мысль вставить какой-то код между вашим оператором READ и оператором IF, который оценивает SY-SUBRC. Если вы используете *LINE\_EXISTS* для того, чтобы чтение таблицы и оценка были объединены в одном операторе, как в листинге 3.45, то это означает, что никто не сможет нарушить это утверждение при намеренном изменении.

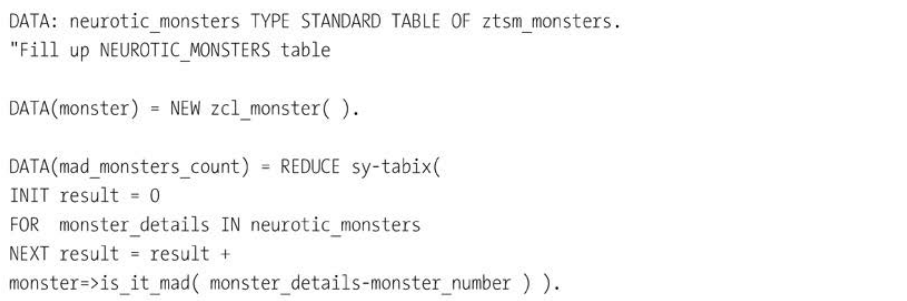
173



Listing 3.45 Code All in One Line, with No Reliance on SY-SUBRC

### 3.5.7 Запросы к внутренним таблицам с помощью REDUCE

В разделе 3.1.4 вы узнали новый способ создания внутренних таблиц с помощью цикла FOR. Однако после заполнения внутренних таблиц вы часто хотите сделать к ним запрос. Скажем, например, вы хотите узнать, сколько у вас действительно безумных монстров. Начиная с версии 7.4, вы можете сделать это с помощью оператора-конструктора *REDUCE*, который содержит логику для обработки внутренней таблицы и возврата одного результата. В примере в листинге 3.46, вы сначала скажете, какую переменную вы хотите создать и какого типа эта переменная будет; затем установите начальное значение для вашей переменной результата и, наконец, выполните цикл по таблицы, выполняя различные логические действия. Звучит очень сложно, но, как вы увидите, это не так.



Listing 3.46 How Many Really Mad Monsters?

Элемент SY-TABIX после оператора *REDUCE* определяет тип создаваемой переменной, MAD\_MONSTER\_COUNT. Он также определяет тип временной переменной после оператора *INIT*, которая будет результатом; и MAD\_MONSTER\_COUNT, и *RESULT* имеют тип SY-TABIX. Затем вы выполняете цикл по вашей внутренней таблице NEUROTIC\_MONSTERS в динамически созданной рабочей области MONSTER\_DETAILS, и каждый раз, когда ваш метод IS\_IT\_MAD возвращается со значением 1 для обрабатываемого монстра, вы увеличиваете счетчик переменной *RESULT*. После завершения цикла *FOR* значение *RESULT* копируется в MAD\_MONSTER\_COUNT.

174

### 3.5.8 Группировка внутренних таблиц

Скорее всего, вы знакомы с дополнением GROUP BY, которое можно добавить к оператору SELECT базы данных и которое сжимает похожие записи в одну строку. В прошлом, когда вам нужно было что-то подобное для внутренней таблицы, вы использовали оператор COLLECT или очень сомнительный оператор AT NEW, который не работал особенно хорошо во многих обстоятельствах. На самом деле, оператор AT NEW был настолько непредсказуем, что было бы лучше избегать его вообще и использовать другие средства для обработки связанных фрагментов из вашей внутренней таблицы.

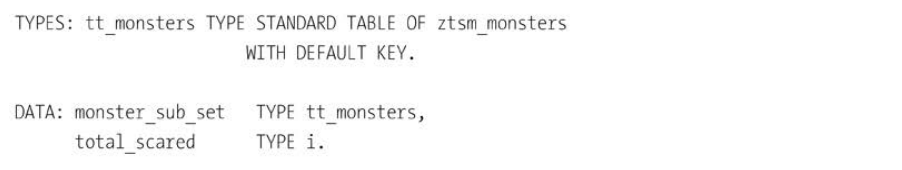
Если быть точным, то если бы вы сказали AT NEW MONSTER в цикле, вы бы ожидали, что код внутри этого блока будет выполняться, когда монстр изменится. Однако иногда он не выполнялся, в зависимости от того, как была отсортирована таблица, находился ли Юпитер в соединении с Нептуном или курица три раза подпрыгнула в полночь и снесла три яйца.

Чтобы ваша программа не зависела от таких астрологических или куриных факторов, в ABAP 7.4 появились хорошие новости: в циклы внутренних таблиц была добавлена опция GROUP BY.

Чтобы понять, что это значит, давайте рассмотрим пример. Допустим, у вас есть огромная таблица монстров. Есть монстры разных типов: некоторые безумные, некоторые нет. Вы хотите обрабатывать только одну комбинацию типа монстра/безумия за один раз. Каким-то образом вам нужно агрегировать такие записи, что-то сделать с результатом и перейти к следующей комбинации. Для этого вам нужен какой-то вложенный цикл обработки.

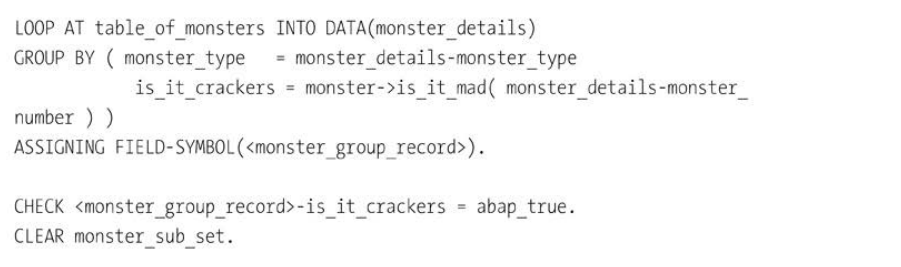
Обычный способ сделать это - построить меньшую таблицу всех возможных комбинаций, найденных в основной таблице, выполнить внешний цикл на этой маленькой таблице, и для каждой строки этой внешней таблицы выполнить внутренний цикл на основной таблице, ища записи, относящиеся к комбинации во внешнем цикле. Есть способы сделать это хорошо - например, убедиться, что внутренняя таблица является сортированной таблицей - но это все равно требует больших усилий.

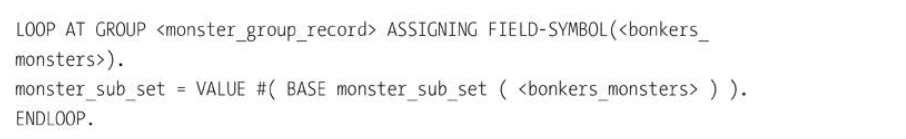
К счастью, GROUP BY спасает ситуацию. Листинг 3.47 демонстрирует, как применение конструкции GROUP BY при циклическом обращении к внутренней таблице позволяет просмотреть совокупность записей для каждой комбинации по одной за раз, без необходимости проходить через муки вложенных циклов. При каждом проходе по таблице вы получаете агрегированные данные на блюдечке.

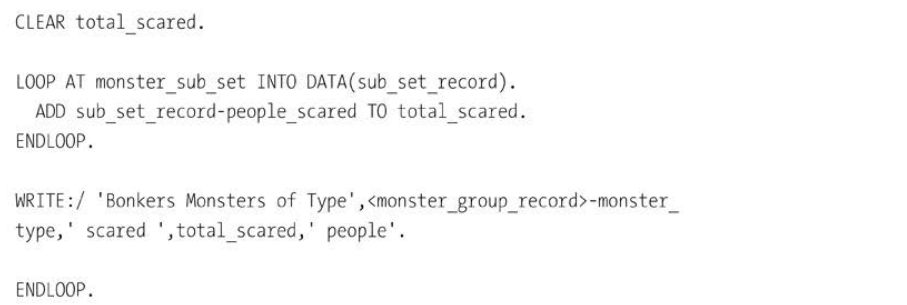


175









Listing 3.47 GROUP BY

176

Это сложно и сразу вводит целую кучу новых понятий, поэтому нам нужно пройтись по коду шаг за шагом. Отладка - лучший способ понять, что происходит.

Во-первых, чтение базы данных моделируется для заполнения таблицы монстров тремя вампирами и двумя зомби, каждый из которых напугал одного или нескольких человек. Бизнес-требованием будет агрегированный список всех напуганных людей, суммированный по типам монстров и исключающий всех монстров, которые не являются сумасшедшими.

Далее следует конструкция *LOOP AT/GROUP BY*. Здесь происходит то, что каждая строка таблицы просматривается по циклу, прежде чем перейти к строке, начинающейся с *CHECK*. При отладке можно увидеть, что метод IS\_IT\_MAD вызывается пять раз, по одному разу для каждой строки таблицы. Для упрощения в этом примере все монстры безумны.

Теперь есть некая невидимая туманная таблица с двумя строками и двумя столбцами, которая была построена процедурой, аналогичной COLLECT. В первой строке написано "VAMP/X", а во второй - "ZOMB/X". Это таблица возможных комбинаций, таблица, которая будет пропущена в рабочую область <MONSTER\_GROUP\_RECORD>.

Выполняется проверка, чтобы убедиться, что мы имеем дело с комбинацией, включающей безумных монстров. Как только это определено, наступает время создать таблицу, которая является подмножеством основной таблицы, которое включает только монстров, соответствующих текущей комбинации.

Эту задачу выполняет блок обработки, начинающийся с *LOOP AT GROUP*. Отдельные записи из основной таблицы, соответствующие текущей комбинации, передаются в рабочую область <BONKERS\_MONSTERS>. На данном этапе можно было бы просто добавить <BONKERS\_MONSTERS>-PEOPLE\_SCARED к переменной TOTAL\_SCARED, но чтобы доказать свою точку зрения, мы построим таблицу подмножеств.

Сначала мы рассмотрим строку, которая выглядит следующим образом:



Это можно перевести как "добавить к таблице MONSTER\_SUB\_SET рабочую область <BONKERS\_MONSTERS>".

Теперь у нас есть таблица, содержащая подмножество основной таблицы, и мы в цикле просматриваем эти записи, суммируя всех людей, напуганных текущим типом монстра, и выводим результат.

В результате мы получим две строки, в которых будет сказано, что (1) безумные вампиры напугали девять человек, а (2) безумные зомби напугали двух человек.

177

Примечание

Как уже говорилось, в этом примере нам не нужно было создавать таблицу подмножеств, но она приведена здесь, чтобы показать, что это возможно.

На данном этапе вы можете подумать, что это выглядит как геометрический цикл, но внутренний цикл обрабатывается только один раз для каждой группы похожих элементов, которые находит внешний цикл. Поэтому время выполнения увеличивается только линейно, что означает, что ваша программа не попадет в ситуацию, когда она будет работать нормально для небольшого количества записей, но при большом количестве данных будет работать с перебоями.

Совет

В SQL-версии GROUP BY можно использовать только имена столбцов. Поскольку версия GROUP BY для внутренних таблиц обрабатывается полностью в ABAP, вы можете делать всевозможные дополнительные вещи - сравнения, вызовы методов и тому подобное - вот почему в примере был вызов метода IS\_IT\_MAD.

Это требует довольно много экспериментов, прежде чем вы освоитесь с этим. Самая сложная часть, как и в случае со всеми этими новыми конструкциями, заключается в сочетании новых возможностей с проблемами реального мира (конечно, некоторые люди могут сказать, что суммирование открытых товаров по клиентам из внутренней таблицы, заполненной значениями BSID, более реалистично, чем суммирование людей, напуганных различными типами монстров, но что они знают?)

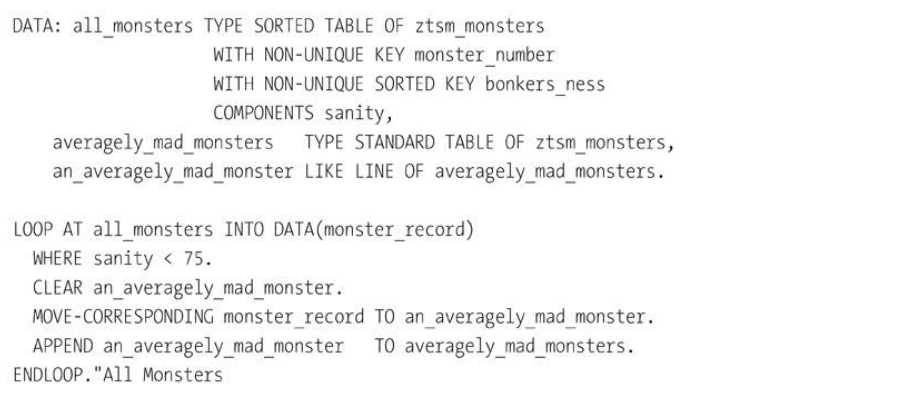
### 3.5.9 Извлечение одной таблицы из другой

Существует два новых способа извлечения одной внутренней таблицы из другой, которые были введены в версии 7.4, и оба используют оператор конструктора *FILTER*. В следующем подразделе мы поговорим об использовании оператора *FILTER* сначала с условной логикой, а затем как операции *FOR ALL ENTRIES* для внутренней таблицы.

**FILTER с условной логикой**

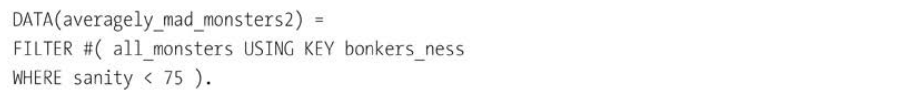
Чтобы понять процесс извлечения одной таблицы из другой в ASAP 7.4, вернемся к примеру с монстрами. Опять же, у вас есть большая таблица со всеми монстрами, и вы хотите извлечь меньшую внутреннюю таблицу, содержащую только среднебезумных монстров. Обычно вы просто просматриваете большую таблицу и добавляете строки в новую таблицу, как показано в листинге 3.48.

178



Listing 3.48 Extracting One Table from Another before 7.4

Начиная с версии ABAP 7.4 (SP 8), вы можете сделать то же самое, используя оператор конструктора FILTER, как показано в листинге 3.49.



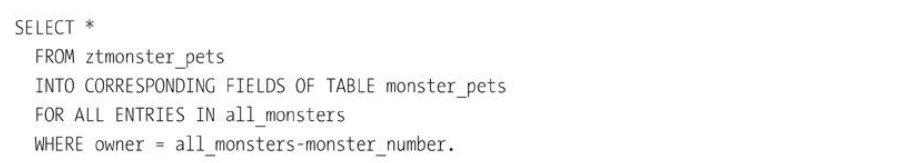
Listing 3.49 Extracting One Table from Another in 7.40

Поэтому в 7.4 вы можете делать то, чего не могли делать раньше - но есть оговорка. Проблема в том, что операция FILTER, представленная в 7.4, работает только в том случае, если большая таблица имеет либо хешированный, либо отсортированный ключ. В предыдущем примере ALL\_MONSTERS имеет отсортированный ключ по здравому смыслу, поэтому оговорка неприменима. (Если бы внутренняя таблица ALL\_MONSTERS не имела хэшированного или отсортированного индекса, то это не было бы такой большой проблемой; помните, что, начиная с версии 7.02 внутренние таблицы могут иметь несколько вторичных ключей, которые могут быть хешированными или отсортированными).

**Фильтр, используемый как FOR ALL ENTRIES во внутренней таблице**

При чтении из базы данных во внутреннюю таблицу, если вам не нужны все ваши данные в одной массивной таблице или вы не можете объединить все таблицы вместе с помощью внутренних объединений, то решением всегда было - начиная с 2000 года и по сей день - выполнить FOR ALL ENTRIES, как показано в листинге 3.50.

179



Listing 3.50 FOR ALL ENTRIES during Database Read

Если ранее в программе вы по какой-то причине уже считывали всю таблицу *monster\_pets* из базы данных во внутреннюю таблицу и хотели создать подмножество только для монстров, находящихся в данный момент в ALL\_MONSTERS, то теперь вы можете запустить эквивалент FOR ALL ENTRIES, но на внутренней таблице, а не в базе данных (Листинг 3.51).



Listing 3.51 FOR ALL ENTRIES on Internal Table

И снова, таблица фильтра (ALL\_MONSTERS в данном случае) должна иметь отсортированный или хешированный ключ, который соответствует тому, что мы ищем (номер монстра в данном случае).

### 3.5.10 Виртуальная сортировка внутренних таблиц

Что такое виртуальная сортировка? Эту функцию (которая появилась в ASAP 7.52) будет довольно сложно понять. Фактически, ее можно описать как решение, ищущее проблему. Тем не менее, даже если я не могу придумать ей практического применения, более чем возможно, что вы сейчас сидите на работе и решаете, казалось бы, неразрешимую проблему, которую она может решить. Это почти невозможно объяснить без примера, поэтому давайте рассмотрим один из них.

Допустим, у вас есть существующий отчет, который отображается в виде ALV-дерева. В начальной таблице содержится список споров с клиентами - случаев, когда клиент пожаловался барону, что монстр недостаточно напугал крестьян, и он хочет получить частичный возврат денег. На первых порах барон обычно отвечал тем, что посылал монстра убить и съесть жалующегося клиента, но это не слишком способствовало повторному бизнесу, поэтому в наши дни он часто выставляет кредитные ноты по обоснованным претензиям.

Таким образом, в отчете о спорах ALV каждая строка спора может быть расширена, чтобы показать кредитную ноту, выставленную против спора, или запись о том, что спор был отклонен. Таким образом,

180

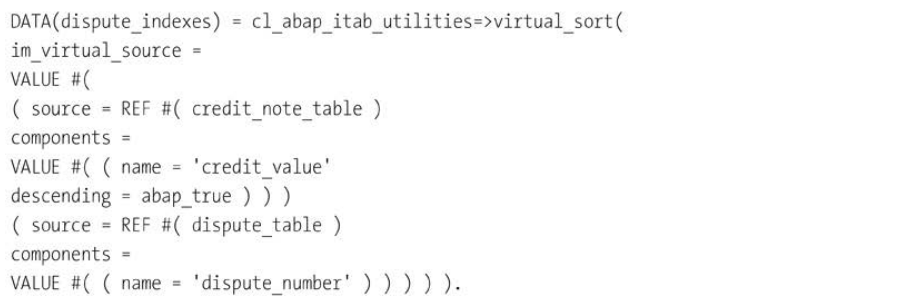
есть две внутренние таблицы, заголовочная для споров и элементная, связывающая споры с кредитными нотами. Обе таблицы одинакового размера, но данные разделены таким образом, чтобы все столбцы можно просмотреть на одном экране без прокрутки вправо. Все работало прекрасно в течение многих лет; все в саду было радужно.

И вот на прошлой неделе к Инге, главному разработчику, пришел главный бухгалтер. В данном случае главный бухгалтер сказал ей, что он расследует какое-то мошенничество, возможно, совершенное против барона, и ему нужен ABAP2XLSX-экспорт данных из отчета о споре. Все, что ему было нужно, - это данные о спорах, отсортированные таким образом, чтобы те, которые относятся к кредитным нотам высокой стоимости, находились в верхней части.

Изначально это казалось простой задачей (добавить опцию экспорта электронной таблицы в отчет так просто), но тот факт, что есть две таблицы, усложняет дело. Должна ли Инга сделать так, чтобы каждая таблица находилась на отдельном рабочем листе? Это могло бы сработать, но тогда пользователю придется вручную соединять две таблицы вместе, что скорее тем самым перечеркивая цель компьютерной автоматизации.

Поразмыслив, Инга поняла, что хочет сохранить таблицу заголовков и таблицу элементов в том виде, в котором они есть - отсортированными по НОМЕРУ ДИСПУТА - чтобы отчет выглядел так, как всегда. Для этого ей придется создать новую таблицу с данными из спора плюс значение кредитной ноты, отсортировать ее и использовать для экспорта в электронную таблицу.

На самом деле это не так уж сложно, но для смеха давайте сделаем это с помощью виртуальной сортировки. В листинге 3.52 вы увидите новый метод *CL\_ABAP\_ITAB\_UTILITIES*, в который передаются ссылки на две исходные таблицы (они должны быть одинакового размера, и вы можете передавать столько исходных таблиц, сколько хотите). Для каждой таблицы вы передаете критерии сортировки, по умолчанию, как обычно, по возрастанию; есть параметр, который нужно заполнить, если вы хотите, чтобы сортировка была по убыванию.

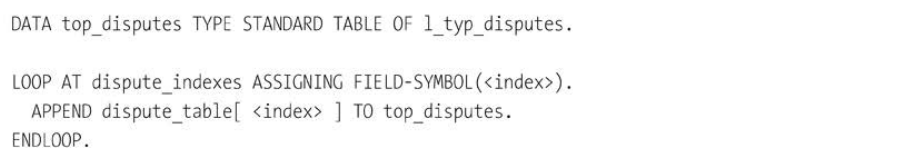


Listing 3.52 Virtual Sorting

181

Результат, DISPUTE\_INDEXES, представляет собой таблицу шириной в один столбец, содержащую номера индексов, которые были бы, если бы вы действительно создали новую таблицу с колонками из обеих таблиц и затем отсортировали ее по значению по убыванию и номеру спора по возрастанию.

Затем объявляется новая таблица для вывода в Excel, и она заполняется с использованием индексов, как показано в листинге 3.53.



Listing 3.53 Using Virtual Sort Result

Можно выполнить два предыдущих шага в одном большом утверждении, которое настолько сложно, что никто не сможет его понять, но мы не будем это демонстрировать.

На мой взгляд, это не более легкий способ решения задачи, а просто другой способ. Тем не менее, теперь вы знаете об этом, на случай, если это именно то, что вы ищете.

## 3.6 Объектно-ориентированное программирование

Компания SAP ввела концепцию ОО-программирования в ABAP в 2000 году. С тех пор это рекомендуемый метод программирования. В этом разделе будут описаны новые возможности ABAP, связанные с OO-программированием.

Прежде чем мы перейдем к этому, я должен упомянуть, что в ABAP 7.5 транзакция SE24 (Maintain Class) была изменена таким образом, что используется вся ширина экрана. Это такое простое исправление, но такое полезное, и давно, давно назревшее. Может быть, когда-нибудь та же полоса здравого смысла будет применена к SE93, который является одним из самых тесных экранов в мире - и по состоянию на ABAP 7.53 все еще не исправлен!

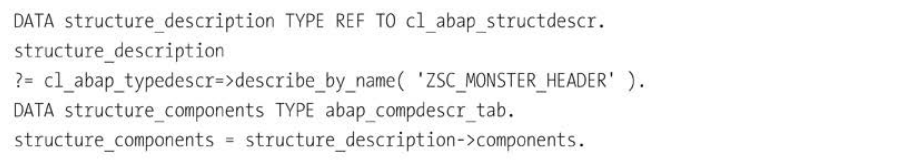
### 3.6.1 Кастинг/даункастинг (Upcasting/Downcasting) с помощью CAST

В ОО программировании, довнкастинг - это процесс, в котором вы превращаете общий объект, такой как монстра, в более конкретный объект, например, зеленого монстра. Апкастинг - это обратный процесс.

182

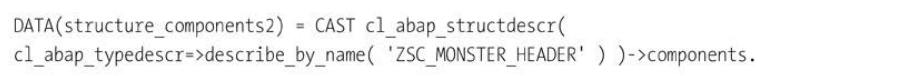
Эта функциональность доступна в ABAP уже давно, но в версии 7.4 она становится намного проще.

Для примера, не связанного с монстрами, рассмотрим ситуацию, в которой вам нужно получить все компоненты определенной структуры словаря. В листинге 3.54 показано, как это можно было сделать до появления ABAP 7.4. Сначала вызовите метод *CL\_ABAP\_TYPEDESCR* для получения метаданных об определенной структуре. Однако, чтобы получить список компонентов этой структуры во внутреннюю таблицу, вам нужен экземпляр *CL\_ABAP\_STRUCTDESCR*; это подкласс *CL\_ABAP\_TYPEDESCR*. Таким образом, вам необходимо выполнить преобразование вниз для преобразования экземпляр родительского класса в экземпляр подкласса.



Listing 3.54 Components of Specific Dictionary Structure without CAST

В 7.4 вы можете сделать это в одной строке с помощью оператора конструктора CAST (листинг 3.55).



Listing 3.55 Components of Specific Dictionary Structure with CAST

Код в листинге 3.54 и листинге 3.55 выполняет точно такую же функцию, но в последнем случае вам больше не нужна вспомогательная переменная STRUCTURE\_DESCRIPTION, а также не нужна строка, в которой вы определяете тип STRUCTURE\_COMPONENTS.

### 3.6.2 Поиск подкласса экземпляра объекта

Во многих других языках программирования можно определить по ссылке на объект, к какому именно подклассу относится данный экземпляр. До версии 7.5 команда разработчиков ABAP в SAP сопротивлялась этому, но после непрекращающегося спроса со стороны онлайн-сообщества, SAP предоставила новый оператор *IS\_INSTANCE\_OF*.

Просто чтобы еще больше раз развеселить SAP - и это очень жестоко, учитывая, что они добавили эту функцию только по многочисленным просьбам - некоторые пуристы скажут, что любой подкласс должен иметь возможность выдавать себя за своего родителя без того, чтобы какая-либо программа знала и, таким образом, вызывающей программе не нужно знать точный подкласс.

183

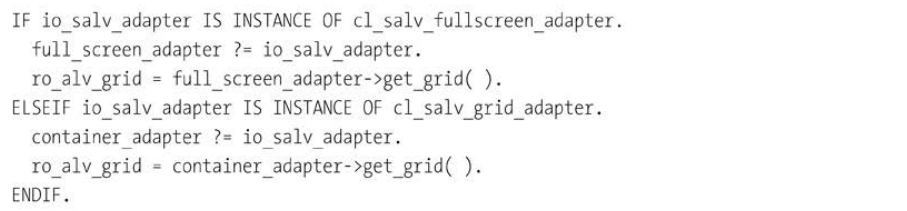
Однако мир ABAP не так чист, как всем хотелось бы, и иногда знание этой информации оказывается весьма полезным - как в следующем примере.

В главе 10, посвященной ALV, есть пример, в котором мы пытаемся получить ссылку на сетку ALV и пробуем один подкласс за другим, пока задание не увенчается успехом (что мы повторяем здесь, в листинге 3.56).



Listing 3.56 Trying to Find Subclass before 7.5

В версии 7.5 жизнь становится намного проще, как показано в листинге 3.57. Благодаря использованию конструкции *IS\_INSTANCE\_OF* цель кода становится гораздо более понятной для читателя.

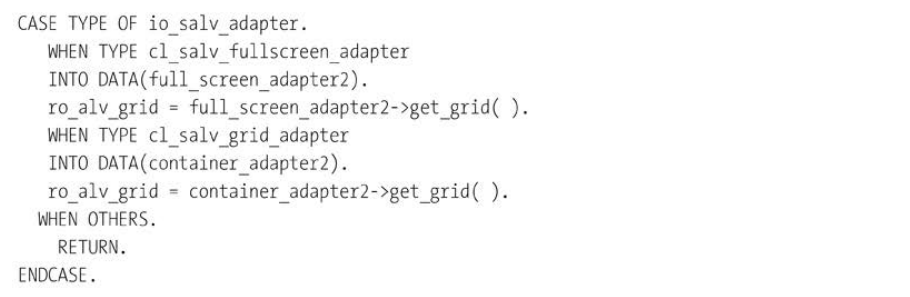


Listing 3.57 Trying to Find Subclass in 7.5

Той же задачи можно достичь несколько иным способом, используя конструкцию *TYPE OF* в сочетании с *CASE*. В коде в листинге 3.58 определяется точный подкласс,

184

и соответствующая ветвь оператора *CASE TYPE OF* гарантирует, что созданный экземпляр имеет правильный подкласс, используя другую новую конструкцию: *INTO DATA*.



Listing 3.58 Another Way to Find Subclass in 7.50 (CASE TYPE OF)

То, как вы должны написать код в таких примерах, как Листинг 3.58, не особенно понятно; фраза *INTO DATA* не очень похожа на английское предложение и поэтому может сбить с толку. Тем не менее, вы должны знать, что эта опция доступна.

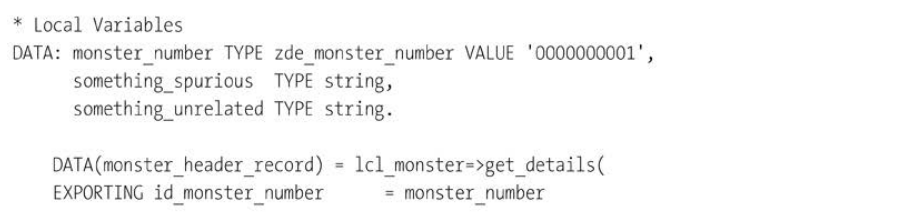
### 3.6.3 ИЗМЕНЕНИЕ и ЭКСПОРТИРОВАНИЕ (CHANGING and EXPORTING) параметров

В ABAP функциональный метод до сих пор определялся как метод с одним возвращающим параметром и от нуля до многих импортирующих параметров, например, следующим образом:

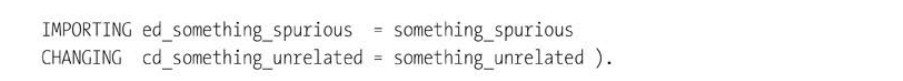


Многим программистам ABAP понравилось то, что переменную результата можно поместить в начало, а не в параметр EXPORTING. Однако они хотели иметь возможность иметь также параметры CHANGING и EXPORTING, то есть иметь свой пирог и съесть его тоже.

В версии 7.40 компания SAP взмахнула волшебной палочкой, и теперь вы можете получить и то, и другое. Пример показан в листинге 3.59.



185



Listing 3.59 CHANGING and EXPORTING Parameters

Несомненно, многие люди будут довольны этим, но пуристы, привыкшие к другим языкам, будут в ужасе. (Хотя я не так категоричен, я понимаю их точку зрения). Хороший ОО дизайн ведет вас к небольшим методам, которые делают одно дело, а для функциональных методов одно дело - выводить один результат. Если функциональный метод внезапно начинает возвращать вам всевозможные другие экспортируемые параметры и изменяет что-то еще, то метод явно делает больше, чем одну вещь, и это, вероятно, плохой дизайн. (Например, есть методы, предназначенные для возврата четырех значений полиномиального уравнения, и вы могли бы использовать новый дизайн, чтобы поместить первое значение в параметр RETURNING, а последние три значения в параметры EXPORTING, но это кажется немного глупым. Вместо этого можно просто вернуть структуру из четырех значений). Ни в коем случае не менее, поскольку это новая возможность версии 7.4, которая может иногда пригодиться вам, было важно упомянуть о ней здесь.

Интересный факт о функциональных методах

Если у вас есть функциональный метод, который возвращает структуру, но вам нужно только одно поле этой структуры, вы можете написать следующий код:



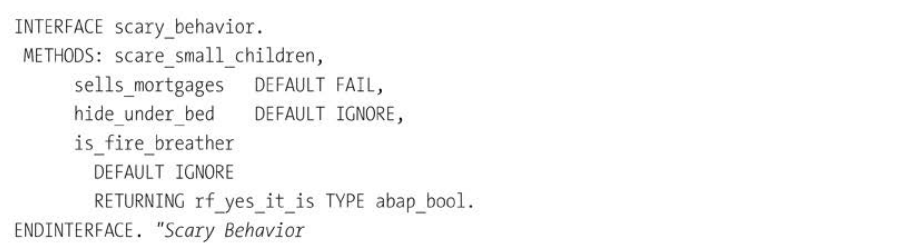
Результатом будет переменная вменяемости, набранная как ZDE\_MONSTER\_SANITY\_PERCENTAGE. Это происходит потому, что как во время компиляции, так и во время выполнения вызов функционального метода интерпретируется как переменная того типа, который он возвращает.

### 3.6.4 Изменения в интерфейсах

В этом разделе рассматривается, как компания SAP попыталась в версии 7.4 избавить вас от необходимости ежедневно использовать интерфейсы в ОО-программировании. Как вы знаете, интерфейс - это набор объявлений данных, имен и подписей методов. Если класс реализует какой-либо интерфейс, то он должен переопределить все методы интерфейса. Все это хорошо, но до версии 7.4 проблема заключалась в том, что некоторые стандартные интерфейсы имели очень большой список методов, только некоторые из которых были релевантными, поэтому вам приходилось перебирать нерелевантные методы и переопределять их, чтобы получить пустые реализации.

Начиная с версии 7.4, если вы создаете интерфейс и думаете, что некоторые из методов могут быть не нужны всем классам, реализующим интерфейс, то вы можете сказать об этом в определении интерфейса, как показано в листинге 3.60.

186



Listing 3.60 Defining Interface with Optional Methods

Это интерфейс, который должны реализовывать все классы монстров. Естественно, все монстры должны уметь пугать детей - поэтому не ставьте никаких дополнений после определения этого метода. Это означает, что каждый класс, реализующий этот интерфейс, будет вынужден переопределить метод в результате проверки синтаксиса. С другой стороны, большинство монстров не будут продавать закладные (только худшие из худших монстров), поэтому не заставляйте все классы реализовывать этот метод. Поскольку вы добавили *DEFAULT FAIL*, если программа, использующая экземпляр монстра, реализующего этот интерфейс, попытается заставить монстра продавать ипотечные кредиты, а метод не был реализован, возникнет ошибка времени выполнения (*CX\_SY\_OYN\_CALL\_ILLEGAL\_METHOD*).

Аналогично, не заставляйте все классы монстров прятаться под кроватями; очевидно, что у тех, чей рост составляет тысячу футов, есть проблемы в этой области. Добавив *DEFAULT IGNORE* в конец определения, мы можем убедиться, что эти классы не будут внедрены принудительно. Если программа скажет такому монстру спрятаться под кроватью, то ничего не произойдет, как если бы был вызван реализованный метод, внутри которого нет ни строчки кода.

Точно так же не все монстры дышат огнем. Для тех, кто дышит, можно реализовать метод IS\_FIRE\_BREATHER, который будет возвращать ABAP\_TRUE. Если метод не реализован в каком-либо классе монстров, то используется дополнение *DEFAULT IGNORE*, а параметр RETURN возвращает начальное значение, которое в данном случае равно ABAP\_FALSE.

## 3.7 Помощь при поиске

Помощь в поиске является одним из достоинств системы SAP. Вы можете определить один из них и прикрепить его к элементу данных, после чего во всей системе, где бы ни упоминался этот элемент данных, мгновенно появляется выпадающий список возможных значений по F4. В выпуске 7.4 ситуация даже стала немного лучше, и в этом разделе мы объясним, как это сделать.

187

### 3.7.1 Предиктивный поиск помогает

Если вы читали что-либо в ИТ-изданиях за последние несколько лет, то вам до смерти надоело, что авторы говорят о том, что бизнес-пользователи теперь ожидают на работе того же уровня удобства, который они получают в свободное время. На данный момент вы, вероятно, видели детей с iPad; они проводят пальцем по экрану, вызывая то-то и то-то, и уже более искусны в этом, чем многие взрослые. Если таков уровень удобства для детей, то чего они будут ожидать от корпоративного программного обеспечения, такого как SAP, когда вырастут и найдут работу? Чего ожидает человек, только что окончивший школу, от системы SAP, которую им показали, как использовать в первый день работы? Во-первых, они ожидают, что когда они начнут вводить что-то в поле поиска - например, название клиента или материала, - то после ввода одной-двух букв появится выпадающее окно с потенциальными кандидатами, из которых они смогут выбирать. Именно так происходит в Google и на многих других сайтах, и вы можете понять, почему недавние студенты будут шокированы, если этого не произойдет на экране SAP.

Хорошая новость заключается в том, что если ваша система SAP имеет версию 7.4 SP 3 или выше, а SAP GUI - 7.31 patch level 5 или выше, то вы обнаружите, что такая помощь в предиктивном поиске теперь возможна в SAP. Половина функциональности находится в бэкенде, а половина - в SAP GUI, вот почему вам необходимо, чтобы и то, и другое было на правильном уровне. (На самом деле, для автоматической работы необходимо иметь версию 7.4 SP 6; в противном случае вам придется вручную возиться с классом *CL\_DSH\_DYNPRO\_PROPERTIES*, как описано в SAP Note 1861491).

Предполагая, что у вас правильная версия, откройте Transaction SE11 и перейдите к опции Search Help примерно в середине экрана. Там вы увидите область сбора данных (Рисунок 3.1).

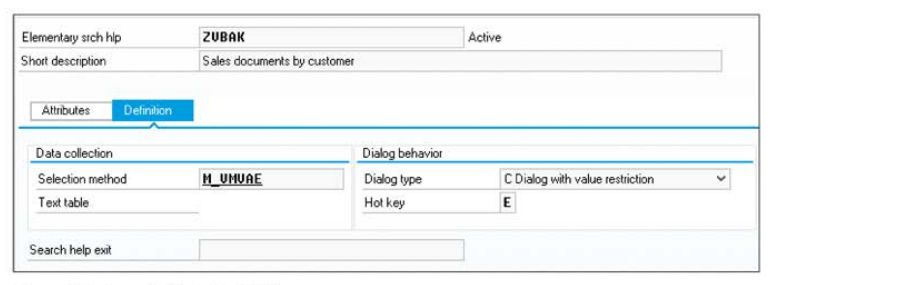


Figure 3.1 Search Help Definition

В версии 7.4 непосредственно над Search Help Exit по поиску появилось новое поле под названием Enhanced Options, в котором есть два флажка и поле ввода. Второй флажок

188

и поле ввода имеют отношение к нечеткому текстовому поиску в базе данных SAP HANA и выделены серым цветом, но в первом флажке указано Proposal Search for Input Fields.

Если вы установите этот флажок, то ваша поисковая подсказка волшебным образом начнет вести себя иначе, не совсем как Google, но всплывающий список кандидатов будет появляться по мере ввода пользователем текста, что является шагом вперед по сравнению с тем, что было раньше.

### 3.7.2 Поиск справки в SE80

Возможно, вы заметили и вас беспокоит тот факт, что, когда вы находитесь в SE80 и не можете вспомнить название своей программы, вы не можете просто нажать F4 и увидеть список результатов. Вы должны нажать выпадающую стрелку справа от поля ввода, что, предположительно, является причиной, по которой эта стрелка находится там в первую очередь. В конце концов, если бы F4 работала, то не нужна была бы отдельная выпадающая стрелка.

Поэтому вы можете представить мой восторг, когда я играл в один из последних выпусков SAP и находился в SE80, набрал половину названия своей программы и рефлекторно нажал F4. Я даже не ввел подстановочную звездочку, но что я увидел, кроме выпадающего списка результатов (Рисунок 3.2). Словами не передать, как я был счастлив. Наверное, это правда, что маленькие вещи радуют маленькие умы. Оказывается, эта функция была доступна в релизе 7.31 - но не все о ней знают, поэтому о ней стоит рассказать здесь.

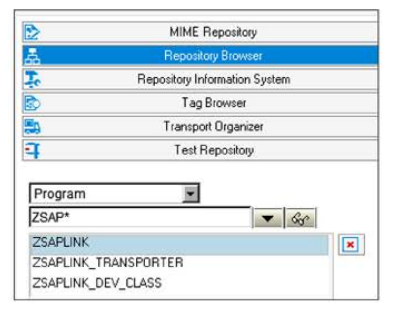


Рисунок 3.2 Справка по поиску F4 в SE80: Наконец-то работает

## 3.8 Резюме

В этой главе вы прочитали о несколько радикальных изменениях, которые были внесены в язык ABAP в последние годы, начиная с версии 7.02, но наиболее заметные в версии

189

релизы 7.4 и 7.5. В этой главе был приведен каталог огромного количества новых языковых конструкций, которые были введены в ABAP, и показаны примеры того, где они могут быть полезны в вашей повседневной работе, разбитые на несколько категорий.

Одно изменение в 7.5, которое не вписывается ни в одну из категорий, заключается в том, что теперь, если вы создаете программу, класс или что там у вас, и установить флажок Unicode, система позволяет вам создать объект, но затем вы получите фатальную ошибку при проверке синтаксиса до тех пор, пока вы не отключите ее снова. Я понимаю, зачем вам нужен этот флажок, чтобы изменить настройки для старых программ, но если бы я правил миром, я бы сделал так, чтобы система даже не позволяла вам создавать объект, если только флажок был установлен - и то же самое для арифметики с фиксированной запятой, на которую система 7.5 вообще не жалуется, если она выключена.

Теперь, когда вы узнали о среде разработки (Eclipse), о системе управлении версиями (abapGit), а также о возросших возможностях языка, на котором вы программируете, пришло время перейти к тому, как справиться с ситуациями, когда все валится в кучу - другими словами, исключения.

Recommended Reading

• Examples of New Constructs

<https://blogs.sap.com/2077/77/08/for-expression-in-abap-7.40-best-case-scenarios/>

(Atanu Mukherjee)

• List of Built-In Functions

190

<https://blogs.sap.com/2075/77/30/reminder-abap-built-in-functions/>

(Horst Keller)

190