# Глава 7 Программирование баз данных с SAP HANA

"Когда вы внедряете инновации, вы должны быть готовы к тому, что каждый скажет вам что вы спятили".

- Ларри Эллисон

"Когда SAP и, в частности, Хассо Платтнер сказали, что они собираются создать эту базу данных in-memory и конкурировать с Oracle, я сказал: "Назови мне имя этого фармацевта, они, должно быть, принимают наркотики".

-Ларри Эллисон

SAP S/4HANA - этот термин сегодня у всех на устах в стране SAP. Вы можете представить себе весь мир SAP как прыгуна на лыжах, мчащегося вниз по невероятно длинному склону: Это займет потребуется до 2025 года, чтобы достичь конца склона, но после этого мы все улетим в смелый новый мир. В этот момент вместо того, чтобы использовать любую базу данных, которую вы захотите, в SAP будет действовать будет действовать принцип горца (может быть только одна), и вы будете полагаться на старую добрую старой доброй SAP HANA.

Это пугает некоторых программистов ASAP - не в последнюю очередь потому, что они читали в Inter-сети о том, что такое SAP HANA и какие изменения она может принести в их работу (например, полное уничтожение). Для ясности - внедрение SAP HANA и конец ABAP - это не одно и то же. Как вы прочтете в этой книге, ASAP по-прежнему жива и здравствует и постоянно совершенствуется.

Это не значит, что вы можете просто сидеть сложа руки и игнорировать эти изменения в течение следующих шести лет.

SAP HANA уже сейчас с нами, многие клиенты уже используют ее (с каждым днем их становится все больше), и довольно много компаний уже используют преемника SAP Business Suite - вышеупомянутую SAP S/4HANA.

Когда SAP HANA появится, вам, по крайней мере, придется убедиться, что ваши пользовательские программы но, что более важно, вы должны использовать эту возможность для того, чтобы сделать так, чтобы они работали лучше, чем когда-либо прежде.

305

В этой главе вы рассмотрите оба вида изменений. Сначала вы познакомитесь с идеей вытеснения кода (раздел 7.1), в которой база данных является гораздо большим игроком в выполнении программ, чем когда-либо прежде. Эта концепция включает в себя три аспекта.

Поэтому каждому из них посвящен отдельный раздел: ABAP SQL (раздел 7.2), представления основных служб данных (CDS) (раздел 7.3), и ABAP Managed Database Procedures (ADMP; раздел 7.4).

Как только вы ознакомитесь с механикой различных способов продвижения кода, в разделе 7.5 будет рассказано о том. 7.5 объяснит, как определить, какие области вашего пользовательского кода могут выиграть от таких изменений, а какие - нет. от таких изменений и какую из трех техник использовать для каждого блока кода. определенного таким образом.

## 7.1 Три лика оттеснения кода

Одна из странных вещей в жизни заключается в том, что часто вам годами твердят: "Вот так надо делать (что бы то ни было). способ сделать (что бы это ни было), это единственный способ, и ничто другое не работает - так что делайте это вот так". Затем, когда вы уже смирились с этим, все меняется в одночасье. ночью, обычно из-за какого-то сейсмического сдвига в технологии, и те же самые эксперты начинают говорить вам, что нужно делать то же самое по-другому.

Когда речь идет о SAP HANA, изменения происходят следующим образом: В течение многих лет учебники по компьютерному программированию учебники по программированию прилагали все усилия, чтобы убедить всех в необходимости разделять код для доступа к базе данных и код для бизнес-логики. Вы все это делали, обычно отдельного класса для чтения базы данных. Теперь, с SAP HANA, из-за гигантских преимуществ в производительности, вас поощряют искать ситуации. в которых вы можете перенести (или передать) большие куски бизнес-логики на уровень базы данных. базовый уровень.

Это известно как code pushdown, потому что код для такой логики перемещается для выполнения в саму базу данных, а не на сервер приложений. Вся эта концепция концепция кажется некоторым традиционалистам чудовищной - и если code pushdown - это чудовище, то в этой книге он должен чувствовать себя как дома. Если продолжить метафору еще на один шаг, можно считать, что код pushdown - это трехголовое чудовище:

1. Первое лицо - это то, с которым вы уже хорошо знакомы, а именно, ABAP SQL SQL-запросы ABAP, которые вы пишете каждый день. С незапамятных времен такие запросы часто делали больше, чем просто слепое получение данных, а начиная с версии 7.4 их возможности значительно увеличиваются.

2. Второе лицо - это представление базы данных. Вы знаете, что такое представление базы данных в ABAP мире: вы определяете представление в SEll, в котором вы назначаете одну или более таблиц базы данных таблицы и объявляете все интересующие вас поля, условия соединения, где

306

подходящий, и, возможно, некоторые условия отбора (например, представление на VBAK, в котором вас интересуют только котировки). Представление CDS - это эквивалент в новой про-граммирования; лежащая в основе концепция практически полностью совпадает, за исключением того, что что возможности выбора гораздо шире.

3. Третье лицо - хранимая процедура, в которой вы пишете некоторый код, который будет выполняться непосредственно на уровне базы данных. вырезан непосредственно в слое базы данных. Это может быть просто SQL-оператор для выбора некоторых данных (хотя это было бы немного бессмысленно), но обычно это один или несколько запросов к базе данных. базовых запросов с некоторой бизнес-логикой.

В каждом случае идея, как всегда, состоит в том, чтобы уменьшить объем данных, передаваемых между базой данных и сервером приложений - то, к чему вы всегда стремитесь, потому что это одно из золотых правил создания быстродействующих программ.

В следующих трех разделах вы узнаете больше об этих трех аспектах вытеснения кода.

## 7.2 ABAP SQL

Если вы уже начинаете нервничать по поводу делегирования некоторых задач ABAP в базу данных, то вам следует понимать, что, скорее всего, вы уже делаете это с некоторыми задачами ABAP SQL. базе данных, вы должны понимать, что, скорее всего, вы уже делаете это в той или иной степени. степени. Например, если вы хотите узнать, сколько фиолетовых монстров с зелеными шляпами и это все, что вы хотите знать (только общее количество), то, скорее всего, вы скорее всего, не будете считывать все данные во внутреннюю таблицу, смотреть, насколько она велика, а затем выбрасывать внутреннюю таблицу. Вместо этого вы скажете SELECT COUNT(\*) и, таким образом, скажете базе данных пойти и подсчитать записи и передать обратно общее количество, чтобы минимизировать передачу данных между базой данных и сервером. Если вы думаете о хранимых процедурах и представления как просто более причудливые версии функций агрегирования, которые вы используете каждый день, то они уже не кажутся такими странными и пугающими.

**Странное новое имя**

Вы вполне можете подумать: "Что это за дела с ABAP SQL? SELECT state-всегда использовали OpenSQL!" Что ж, в типичной моде SAP, OpenSQL был переименован в ABAP SQL, начиная с релиза ABAP 1809.

На этот раз это не просто смена названия ради смены названия. Часть "открытый" относится к тот факт, что набор функций, поддерживаемых OpenSQL, был платформенно-независимым, то есть они работают на любой базе данных, поддерживаемой SAP.

В SAP S/4HANA вы можете иметь только базу данных SAP HANA, поэтому компания SAP ввела в действие ввела некоторые функции, которые работают только на базе данных SAP HANA. Таким образом, ABAP

307

SQL теперь "больше не является полностью платформонезависимым", по словам SAP. Это какговорят, что вы наполовину беременны - вы либо беременны, либо нет, и ABAP SQL больше не является не является платформонезависимым.

По сравнению с тем, что было возможно до версии 7.4, сейчас в операторах SELECT больше новых причудливых возможностей, чем звезд на небе. функций, доступных в операторах SELECT, больше, чем звезд на небе. перечислены в следующих разделах.

Если вы используете некоторые из этих новых причудливых возможностей при использовании оператора SELECT в ABAP, то, по сути, вы уже выполняете pushdown кода, то есть передаете на аутсорсинг передавая то, что вы обычно кодируете в своей ABAP-программе (например, оператор CASE) в базу данных. в базу данных, как правило, с целью сокращения объема данных, отправляемых обратно. Это означает, что возможности ABAP SQL уже настолько расширились, что, в большинстве случаев вам потребуется определить специальные представления и хранимые процедуры только тогда, когда расширенные возможности ABAP SQL просто не справляются с задачей.

Более того, это своего рода гонка; дополнительные возможности вводятся в CDS представления и хранимые процедуры с каждым пакетом поддержки. В то же время, однако, возможности ABAP SQL в ABAP также увеличиваются с каждым пакетом поддержки.

### 7.2.1 Новые команды в ABAP SQL

Скорее всего, вы знаете, что в ABAP существует два вида SQL-запросов: ABAP SQL и NativeSQL. ABAP SQL - это подмножество всех команд, которые могут использоваться во всех базах данных, которые поддерживает SAP (это похоже на область в середине диаграммы Венна). диаграммы). Другими словами, в ABAP SQL операторы, которые вы можете присоединить к SELECT в ABAP, такие как ORDER BY, INNER JOIN и SELECT DISTINCT, являются лишь подмножеством того, что доступных от любого конкретного поставщика базы данных - то есть, список возможных опций, которые может быть добавлен после оператора SELECT, упомянутого в документации данного поставщика. поставщика. ABAP SQL - это то, что вы используете большую часть времени (см. листинг 7.1).

SELECT monst er number monst er name

FROM zt monst ers

INTO CORRESPONDING FIELDS OF TABLE monst er table

WHERE monster number = monster number.

Listing 7.1 ABAP SQL

NativeSQL, с другой стороны, представляет собой запрос, записанный между командами EXEC и ENDEXEC. командами. NativeSQL позволяет писать любые команды запросов, поддерживаемые базой данных установленной в вашей системе. Например, на рисунке 7.1 показан запрос Microsoft SELECT

308

опции синтаксиса представлен в качестве примера того, что может быть сделано в этой компании реализации NativeSQL.

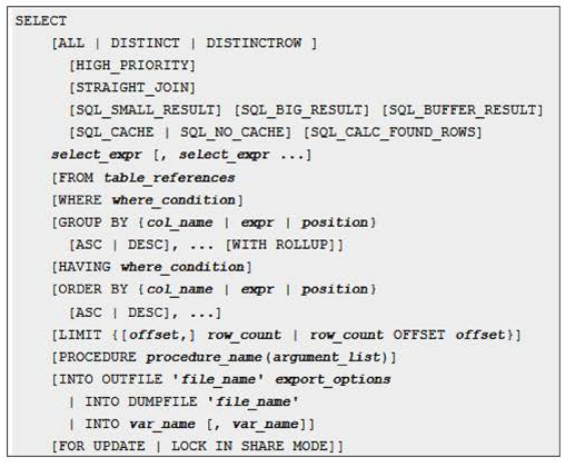


Figure 7.1 Microsoft SQL Syntax

Поскольку он предоставляет больше возможностей, NativeSQL является более мощным. Однако, если ваша организация когда-нибудь решит выполнить миграцию базы данных, то у вас будут проблемы; вам почти наверняка придется переписывать все запросы к NativeSQL, которые вы написали. Даже хотя миграция баз данных - это не прогулка по парку, она случается довольно часто, и люди вроде меня избегают таких потенциальных лишних "Норк": Я не написал ни одного запроса на NativeSQL за девятнадцать лет программирования на ABAP.

Цель SAP - усовершенствовать ABAP SQL до уровня, когда он будет соответствовать стандарту SQL-92. стандарту - то есть современному стандарту 1992 года, когда родилась Майли Кипр и были изобретены текстовые сообщения. рождения Майли Кипр и изобретения текстовых сообщений, за шесть лет до основания Google. Этот возможно, не вызовет у вас удивления по поводу молниеносной скорости работы SAP, но хорошая новость заключается в том, что в релизе 7.4 компания SAP, хотя и не совсем достигла стандарта SQL92, расширила диапазон возможностей SQL92. расширила диапазон опций, доступных для использования в ABAP-программах, когда вы выполняете ABAP SQL-запрос. Вы можете не использовать STRAIGHT JOINS и SOL\_BIG\_ RESULTS, как в синтаксисе Microsoft SQL на рисунке 7.1, но есть несколько новых полезных элементов. элементы, и в следующих разделах мы рассмотрим два наиболее важных из них. из них - CASE-операторы, встроенные в SQL-запросы, и выполнение вычислений внутри SQL-запросов - прежде чем перейти к тому, что ABAP 7.5 сделал для улучшения функциональность пункта WHERE в операторах SELECT.

309

*Вставка CASE-выражений в SQL-запросы*

Одной из новых возможностей ABAP 7.4 является возможность вставки CASE-запросов в запросы SQL. В листинге 7.2 показан пример, в котором имеется поле во внутренней таблице под названием SCARINESS\_STRING, и оно должно быть заполнено строкой, описывающей, насколько страшным является монстр, основываясь на значениях столбцов базы данных SANITY и STRENGTH.

CONSTANTS: lc\_scary1 TYPE char20 VALUE 'SLIGHTLY SCARY',

lc\_scary2 TYPE char20 VALUE 'REALLY SCARY',

lc\_scary3 TYPE char20 VALUE 'NORMAL ' .

DATA: table\_of\_monsters TYPE g\_tt\_monsters\_with\_extras,

monster\_number TYPE zde\_monster number VALUE '0000000001'.

SELECT name, monster\_number,

CASE

WHEN sanity <= 10 AND strength\_num >= 75 THEN @lc\_scary2

WHEN sanity <= 25 AND strength\_num >= 50 THEN @lc\_scary2

ELSE @lc\_scary3

END AS scariness\_string

FROM ztsm\_monsters

WHERE monster\_number = @monster\_number

INTO CORRESPONDING FIELDS OF TABLE @table\_of\_monsters .

Listing 7.2 CASE Statement within SQL Statement

Вы заметите, что перед переменными (или константами) при использовании новых причудливых функций, таких как CASE, чтобы компилятор знал, что что вы говорите не о поле в базе данных. Вы также должны ставить запятые между полями, которые вы извлекаете из базы данных, и поместить оператор INTO в конце, как в диаграмме Microsoft, которую вы видели на рисунке 7.1. Символ @, запятые и другие синтаксические элементы являются результатом новой "строгой" проверки синтаксиса, которая вступает в силу, когда компилятор замечает, что вы используете одну из новых функций. Другими словами, если вы попытаетесь использовать новую функцию, такую как оператор CASE, в запросе SQL, но не поставите символ @ рядом с именем переменной, то вы получите жесткую ошибку. Таким образом, SAP может заставить вас программировать немного по-другому при использовании новые функции, сохраняя при этом обратную совместимость.

Это все хорошо, но что здесь вообще делает оператор CASE? на первом месте? Разве оно потеряно? Конечно, вы используете операторы CASE после получения данных от

310

БД, не так ли? По сути, оператор CASE позволяет вам сделать следующее: передать работу по вычислению некоторой условной логики базе данных - вместо того, чтобы выполнения этих вычислений на сервере приложений. Как вы увидите в главе 5, передача части подобной работы в базу данных SAP HANA для обработки может привести к значительному повышению производительности (если все сделано правильно).

**Вычисления в SQL-запросах**

Еще одной функциональной новинкой в версии 7.4 является возможность выполнения арифметических вычислений внутри SQL-запросов. В предыдущих выпусках вам приходилось сначала получать данные из сначала получить данные из базы данных, а затем играть с ними. В качестве примера того, как это работало раньше, скажем, что вы хотите разделить силу на здравомыслие, чтобы получить коэффициент страшности: чем ниже здравомыслие, тем выше коэффициент. В листинге 7.3 показано, как это делалось до ABAP 7.4.

"Before

DATA: converted\_strength TYPE decfloat34,

converted\_sanity TYPE decfloat34,

table\_of\_monsters TYPE g\_tt\_monsters\_with\_extras,

monster\_number TYPE zde\_monster\_number VALUE '0000000001' .

FIELD-SYMBOLS: <monster\_details> LIKE LINE OF table\_of\_monsters .

SELECT name monster\_number strength\_num sanity

##too\_many\_itab\_fields "in the world

FROM ztsm\_monsters

INTO CORRESPONDING FIELDS OF TABLE table\_of\_monsters .

LOOP AT table\_of\_monsters ASSIGNING <monster\_details> .

converted\_strength = <monster\_details>-strength\_num.

converted\_sanity = <monster\_details>-sanity.

<monster\_details>-scary\_ratio = converted\_strength / converted\_sanity.

ENDLOOP.

Listing 7.3 Operations inside SQL Statements: Before

Однако, начиная с версии ASAP 7.4, появилась возможность выполнять эти операции в SQL-запросе, как показано в листинге 7.4.

311

SELECT name, monster\_number,

CAST( strength\_num AS FLTP ) / CAST( sanity AS FLTP ) AS scary\_ratio

FROM ztsm\_monsters

WHERE monster\_number = @monster\_number

INTO CORRESPONDING FIELDS OF TABLE @table\_of\_monsters .

Listing 7.4 Operations inside SQL Statements: After

Как вы можете видеть, пример также включает некоторые преобразования типов. Это избавляет вас от объявления двух вспомогательных переменных и, что более важно, от необходимости перебирать в цикле внутренней таблице, выполняя вычисления в каждой строке.

Более того, если вас не интересуют сила и разумность сами по себе и только заботитесь только о соотношении, вы можете иметь на два поля меньше во внутренней таблице (которые вы придется потратить дополнительные усилия, чтобы объявить их "техническими" в отчете ALV), и меньше информации будет отправлено обратно из базы данных на сервер приложений (соотношение в отличие от в отличие от силы и разумности). Чем реже отправляются такие данные, тем быстрее работает программа работает.

**Улучшения в WHERE Clause и друзьях**

Как вы только что видели, ABAP 7.4 позволяет вам делать всевозможные новые замечательные вещи внутри в списке SELECT. ABAP 7.5 и выше продвигает это на один этап дальше: теперь вы можете использовать все арифметические выражения (и вызовы функциональных методов), которые можно было использовать в 7.4 в списке SELECT в выражениях WHERE, ON и CASE. В листинге 7.5 показано несколько практических применений этой функциональности этой функциональности.

SELECT monster\_number, tax\_number

FROM ztmonster\_header

WHERE age + sanity > 100

INTO TABLE @DATA(old\_sane\_monster\_list) .

SELECT monster\_number, tax\_number

FROM ztmonster\_header

WHERE hat\_size = @( hat\_size\_of\_the\_day( ) )

INTO TABLE @DATA(fashionable\_monster\_list).

Listing 7.5 New Possibilities in WHERE Clauses

Кроме того, новые функции, которые можно использовать в предложении SELECT, теперь включают в себя переменные строковые функции, такие как LPAD для добавления ведущих нулей или новая функция CONCAT,

312

которые можно складывать только из двух элементов среди прочих. И снова, некоторые возможные применения этой функциональности для вашей повседневной обработки монстров показаны в Листинге 7.6.

SELECT

CONCAT( monster\_number, REPLACE( ' EVIL', 'VERY EVIL' , name ) ) AS monster\_description,

LENGTH( tax\_number ) AS tax\_type

FROM ztmonster\_header

WHERE hat\_size = @( hat\_size\_of\_the\_day( ) )

INTO TABLE @DATA(evilized\_monster\_tax\_types) .

Listing 7.6 String Functions in SELECT Statements

Более того, с каждым выпуском ABAP вы получаете все больше функций, которые можно использовать в операторе ABAP SQL. Например, в версии 7.51 вы получаете новые функции UPPER и LOWER, которые преобразуют строку в верхний или нижний регистр, соответственно, внутри базы данных перед отправкой обратно на сервер приложений. базы данных перед отправкой ее обратно на сервер приложений. В версии 7.51 также добавлена функция CONCAT\_WITH\_ SPACE, которая должна звучать знакомо, потому что выглядит почти как эквивалентная команда ABAP.

В ABAP 1809 вы получите одиннадцать новых функций времени и даты. Также, начиная с ABAP 1809, вы можете использовать IS INITIAL или IS NOT INITIAL в предложениях WHERE, как показано в листинге 7.7. Ранее для этого нужно было использовать литералы типа SPACE или '000000'.

SE LECT monster\_number

FROM ztmonsters

WHERE hat\_size IS INITIAL

INTO @data(monsters\_without\_hats) . "They don't dance

Listing 7.7 Using IS INITIAL in WHERE Clause

С каждым релизом появляется так много новых опций, что они продолжают сыпаться на вас как теннисные мячи из автомата для теннисных мячей. Идея заключается в том, что любая операция, которую вы раньше выполняли с данными после их возвращения из базы данных - возились с строками или числами - теперь вы должны быть в состоянии указать в самом запросе и позволить базе данных берет на себя всю нагрузку.

**Поиск без учета регистра**

Я не хочу показаться бесчувственным в этом вопросе, но ключевой вариант использования ключевого слова UPPER заключается в том, что вы можете осуществлять поиск текста без учета регистра, преобразуя ваш

313

критерии выбора в верхний регистр до вызова оператора SQL, а затем в самом SQL-запросе преобразовать текстовое поле, в котором выполняется поиск, в верхний регистр. Люди просят об этом уже много лет.

Например, допустим, в таблице основных данных о деревнях первая буква каждого слова в адресе деревни написана в верхнем регистре адреса деревни написана с заглавной буквы - например, "Shanty Town рядом с Cliff Edge". Деловая проблема заключается в том, что нам нужен список всех деревень, расположенных на краях утесов, чтобы монстры спихнули их на край обрыва, но до сих пор получение такого списка с помощью SQL было проблематично, потому что некоторые записи гласили "CLIFF EDGE", некоторые "Cliff Edge", а некоторые "Cliff EDGE". В листинге 7.8 показано, как ключевое слово UPPER может решить эту проблему, превратив и запрос, и запрашиваемые данные в текст в верхнем регистре.

METHOD derive\_villages\_by\_description .

\* Preconditions

CHECK id\_description IS NOT INITIAL .

DATA(upper\_case\_query) = '%' && to\_upper( id\_description ) && '%' .

SELECT \*

FROM zmn\_villages

WHERE upper( village\_address ) LIKE @upper\_case\_query

ORDER BY village\_number

INTO TABLE @rt\_villages .

ENDMETHOD.

Listing 7.8 Case-Insensitive Database Query

Я остаюсь при своем мнении.

### 7.2.2 Создание при чтении

В главе 10 этой книги вы узнаете о CL\_SALV\_TABLE и о том, как он достаточно умен, чтобы посмотреть на определение вашей внутренней таблицы, чтобы посмотреть на определение вашей внутренней таблицы и превратить это определение в ALV сетку без необходимости вручную определять каждый столбец заново. ABAP 7.4 сделала еще один шаг вперед, избавившись от необходимости выполнять объявление данных для внутренней таблицы.

Чтобы проиллюстрировать, что это значит, давайте рассмотрим пример. Допустим, вам нужен список всех монстров по имени Фред. Традиционно вы бы сделали это так, как показано в списке 7.9.

314

TYPES : BEGIN OF l\_typ\_monsters,

name TYPE ztsm\_monsters-name,

monster\_number TYPE ztsm\_monsters-monster\_number,

sanity TYPE ztsm\_monsters-sanity,

END OF l\_typ\_monsters .

DATA: table\_of\_monsters TYPE STANDARD TABLE OF l\_typ\_monsters .

SELECT name monster\_number sanity

FROM ztsm\_monsters

INTO CORRESPONDING FIELDS OF TABLE table\_of\_monsters

WHERE name = ' FRED'.

Listing 7.9 List of All Monsters Na med Fred

Обратите внимание, что код включает объявление TYPE. Чтобы сэкономить время, некоторые программисты просто определяют внутреннюю таблицу как тип словаря данных. Однако, это нерациональная трата памяти.

В старых версиях ABAP, если вы объявляли ТИП, а затем внезапно хотели получить дополнительное поле, то вам необходимо было бы внести изменения в двух местах: в определении и в операторе SELECT. Вы можете забыть внести одно из этих изменений - я знаю, и, хотя проверка синтаксиса предупредит вас, оно все равно может проскользнуть через сеть.

Однако в версии 7.4 можно пропустить не только определение TYPE, но и внутреннее объявление таблицы. Пример этого показан в листинге 7.10.

SELECT name, monster\_number, sanity

FROM ztsm\_monsters

WHERE name= 'FRED'

INTO TABLE @DATA(table\_of\_monsters2) .

Listing 7.10 Defining Internal Table Based on SQL Query

Как видите, нет необходимости объявлять ни TYPE, ни внутреннюю таблицу. Таблица создается на месте в момент чтения базы данных, а формат таблицы

315

берется из типов полей данных, которые вы извлекаете. Таким образом, два списка полей не могут быть рассинхронизированы, поскольку существует только один список.

Эта новая техника также работает для структур, если вы выполняете SELECT SINGLE для нескольких полей базы данных и для элементарных элементов данных, если вы выполняете SELECT SINGLE только для одного поля базы данных. Более того, если вы используете такую конструкцию, как KUNNR AS SHIP\_TO в операторе select, то имя столбца в вашей таблице будет ship\_to. (Подобные вещи полезны; например, вы можете выполнять объединение и KUNNR может иметь разные значения в разных таблицах: клиент в VBAK и ship\_to в LIKP).

### 7.2.3 Улучшение буферизации

Одна из самых приятных для меня задач - это использование Transaction ST05 для трассировки SQL для программ и обнаружить, что, внеся некоторые незначительные изменения, я могу воспользоваться преимуществами буферизации таблиц в таблице DDIC, чтобы значительно уменьшить доступ к базе данных, таким образом, делая программа работает намного быстрее и веселит моих пользователей. Хотя буферизация существует уже давно, с ней также были связаны некоторые проблемы; в основном, это целый ряд ситуаций, в которых буфер обходится стороной, и вы не получаете никакого улучшения производительности вообще. (Эту проблему можно решить с помощью Code Inspector.

Если вы запустите проверку Code Inspector на вашей программе после ее создания и после каждого последующего изменения - а это не так уж мало, вы будете предупреждены о любых подобных чтениях базы данных, которые вы могли закодировать в программе.

Чтобы продемонстрировать распространенную ситуацию, в которой буфер обходится, рассмотрим следующий пример, в которых половина ваших запросов получает данные из таблицы ZT\_MONSTERS, используя MONSTER\_NUMBER (который является первичным ключом), а другая половина получает данные, используя MONSTER\_ NAME (по которому вы составляете индекс).

Предположим, что это главная таблица данных, которая относительно небольшая, часто читаемая и редко изменяемая - поэтому, естественно, вы определили ее как полностью буферизованную в технических настройках.

Если вы выполняете SQL-запрос с использованием MONSTER\_NUMBER, то доступа к базе данных не происходит вообще (после первого чтения после запуска базы данных), потому что вы указали первичный ключ в запросе. Однако если вы выполните SELECT SINGLE с использованием MONSTER\_NAME, то Code Inspector предупредит вас, что буферизация таблицы не будет использоваться, потому что вы не указываете первичный ключ.

Раньше люди обходили эту проблему, считывая всю таблицу из базы данных во внутреннюю таблицу, а затем сортировали ее по MONSTER\_NAME. Затем, когда кто-то хотел найти монстра по имени, они выполняли БИНАРНЫЙ ПОИСК, чтобы получить нужную запись.

316

До версии 7.02 люди иногда использовали внутренние таблицы, отсортированные по разным ключам. Начиная с версии 7.02, вы могли иметь вторичные индексы на внутренних таблицах, так что вам нужна только одна таблица. Тем не менее, это все равно слишком много таблиц, с точки зрения потребления памяти.

Начиная с версии 7.4, это больше не является проблемой. Если вы выполните SELECT SINGLE на MONSTER\_NAME и затем запустить трассировку производительности ST05, вы не увидите никакого доступа к базе данных вообще даже если вы не указали первичный ключ. Это происходит потому, что, начиная с версии 7.4, если вы определили вторичный индекс на MONSTER\_NUMBER в таблице ZT\_MONSTERS, то волшебным образом учитывается и этот индекс. Вторичные индексы учитываются не только в таблицах с полной буферизацией, но и в таблицах с общей буферизацией, что является гигантским скачком вперед в плане производительности. Буферы, учитывающие вторичные индексы, можно описать как пассивное улучшение; если у вас есть операторы SELECT на индексированном поле, которые раньше обходили буфер, то начиная с версии 7.4 ваш код ускорится, не пошевелив и пальцем - и многие стандартные коды SAP также ускорятся.

Еще одним пассивным примером улучшения буферизации в ABAP 7.4 является то, что до сих пор вы не могли выполнить FOR ALL ENTRIES в буферизованной таблице, даже если вы использовали первичный ключ без обращения к базе данных. Приходилось выполнять цикл из операторов SELECT SINGLE, а затем предупреждать о выполнении SELECTS в цикле с помощью расширенной проверки синтаксиса. Теперь эта проблема осталась в прошлом. Поскольку все данные находятся в памяти, запрос рассматривает FOR ALL ENTRIES как цикл операторов SELECT SINGLE и таким образом, обращение к базе данных полностью исключено. В качестве примера, многие стандартные отчеты SAP выполняют FOR ALL ENTRIES на буферизованной таблице SETLEAF, которая относится к центру прибыли иерархии. В среде 7.4 все эти стандартные программы будут выполняться намного быстрее.

Иногда требуется просто проверить существование, чтобы убедиться, что определенная запись действительно существует в базе данных. Даже если вам совершенно безразлично содержимое этой строки, а важно только то, что она существует, по крайней мере, часть этой строки все равно должна быть перенесена из базы данных на сервер приложений, вызывая тем самым накладные расходы, какими бы незначительными они ни были.

В ABAP 7.4 SP 5 и далее вы можете указать SQL-запросу искать фиктивное значение @ABAP\_TRUE, как показано в листинге 7.11, что эквивалентно крику "Ты здесь? там?"

SELECT SINGLE @abap\_true

FROM ztsm\_monsters

WHERE monster number = @monster number

INTO @DATA(result) .

317

IF result = abap\_true .

\* Do Something

ENDIF .

Listing 7.11 Existence Check with No Database Rows Transported

В листинге 7.11 ни одна строка не переносится из базы данных на сервер приложений, что замечательно с точки зрения производительности.

### 7.2.4 Улучшения INNER JOIN

Если вы когда-нибудь кодировали (а если вы читаете эту книгу, я полагаю, что да!), то вы знаете, что довольно часто возникает необходимость прочитать все поля одной таблицы с данными базовой таблицы, а также одно или два поля из другой, смежной таблицы. Наиболее эффективным это сделать с помощью внутреннего объединения. Однако перечислять все поля из основной таблицы довольно утомительно, как показано в листинге 7.12.

TYPES: BEGIN OF l\_typ\_monsters,

name TYPE ztsm\_monsters-name,

monster\_number TYPE ztsm\_monsters-monster\_number,

sanity TYPE ztsm\_monsters-sanity,

strength\_num TYPE ztsm\_monsters-strength\_num,

hat\_size TYPE ztsm\_monsters-hat\_size,

no\_of\_heads TYPE ztsm\_monsters-no\_of\_heads,

bar\_code TYPE ztmo\_hats-bar\_code,

END OF l\_typ\_monsters .

DATA: table\_of\_monsters TYPE STANDARD TABLE OF l\_typ\_monsters .

SELECT ztsm\_monsters-name

ztsm\_monsters-sanity

ztsm\_monsters-hat\_size

ztsm\_monsters-monster\_number

ztsm\_monsters-strength\_num

ztsm\_monsters-no\_of\_heads

ztmo\_hats-bar\_code

FROM ztsm\_monsters

INNER JOIN ztmo\_hats

ON ztsm\_monsters-hat\_size = ztmo\_hats-hat\_size

INTO CORRESPONDING FIELDS OF TABLE table\_of\_monsters

WHERE name= ' FRED'.

Listing 7.12 All Fields from Main Table

318

Код в листинге 7.12 считывает все из таблицы монстров, но только одну незначительную из таблицы шляп. Начиная с версии ABAP 7.4, вы можете добиться того же самого другим способом, как показано в листинге 7.13.

SELECT ztsm\_monsters-\*,

ztmo\_hats-bar\_code

FROM ztsm\_monsters

INNER JOIN ztmo\_hats

ON ztsm\_monsters-hat\_size = ztmo\_hats-hat\_size

WHERE name = 'FRED'

INTO TABLE @DATA(t\_monsters2).

Listing 7.13 Reading Everything from One Table during Inner Join

Ключевым здесь является звездочка{\*}. Она действует так же, как дикая карта в SELECT \*, и поэтому ее можно использовать (возможно, даже злоупотреблять), если вам нужны не все поля таблицы, а только половина из них.

Внимание: Хьюстон, у нас проблема

Как всегда, использование звездочек в таких случаях - это простой выход, но он замедляет производительность базы данных и - при одновременном создании таблицы - потребление памяти. Это доказывает, что чем больше инструментов в вашем распоряжении, тем больший вред можно нанести, если использовать их неправильно.

В дополнение к функциональности звездочек в ABAP 7.4, синтаксис внутренних объединений также был расширен, так что теперь у вас есть гораздо больше гибкости в определении связи между таблицами, как показано в листинге 7.14.

INNER JOIN zt\_monster\_pets

ON zt\_monster\_pets~owner EQ zt\_monsters~monster\_number

AND zt\_monster\_pets~type LIKE 'GIGANTIC%'

AND zt\_monster\_pets~species IN ( 'SHARK', 'CROCODILE' , 'DINOSAUR' )

Listing 7.14 Joining Two Tables in Newly Possible Ways

Если вам нужна таблица всех домашних животных, принадлежащих различным монстрам, вместе с некоторыми подробностями о владельцах, но вас интересуют только гигантские динозавры и тому подобное, тогда запрос, показанный в листинге 7.14, - это то, что вам нужно.

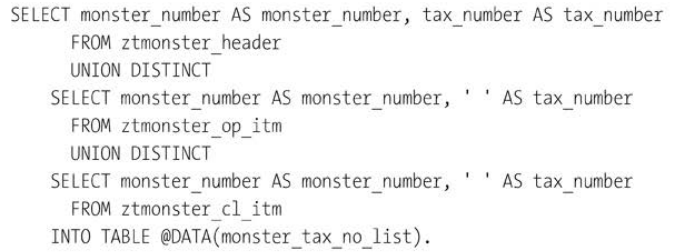
319

### 7.2.5 UNION

Одна из команд SQL, которую вы никогда не могли использовать в ABAP, но которая уже давно используется в различных базах данных, - это UNION. В ABAP 7.5 это упущение было исправлено.

Давайте рассмотрим пример одного бедного парня, написавшего пост в SCN в 2007 году. У него не было проблемы, связанной с монстрами (интересно, почему бы и нет?), но он пытался получить уникальный список налоговых номеров всех продавцов, у которых были либо (а) открытые товары, либо (б) закрытые товары. Более того, он хотел получить свой список, используя только один SQL-запрос, на основании того, что другие языки, кроме ABAP, поддерживают такую возможность.

Используя новый синтаксис ABAP 7.5 (листинг 7.1), давайте попробуем решить эту задачу с помощью наших монстров. В таблице базы данных монстр 1 имеет открытые и закрытые элементы, монстр 2 имеет только закрытые элементы, а у монстра 3 вообще нет элементов.



Listing 7.15 SELECT from Different Tables Using UNION

В результате получается таблица, в которой три строки содержат номер монстра и налоговый номер (взятые из таблицы основных данных), а в двух строках - номер монстра 1 и номер монстра 2, без налогового номера, но с указанием того, что у них есть открытые или закрытые позиции. Это достаточно информации для работы без необходимости снова обращаться к базе данных.

Здесь важно всегда считывать один и тот же набор столбцов из каждой таблицы, к которой вы присоединяетесь, даже если этот столбец фактически не существует в целевой таблице. Один другой пример, который приходит на ум - это таблицы условий ценообразования, в которых у вас есть десять различных таблиц в последовательности доступа, каждая из которых имеет немного отличающийся ключ; обычно вам приходится читать каждую таблицу по очереди, но с помощью UNION вы можете получить их все за один раз.

### 7.2.6 ПЕРЕКРЕСТНЫЕ СОЕДИНЕНИЯ (CROSS JOINS)

До сих пор вы привыкли к INNER JOINS и OUTER JOINS, которые являются спокойными членами общества SQL и никогда не вызывают проблем. Начиная с версии ABAP 7.51, новый ребенок в городе

320

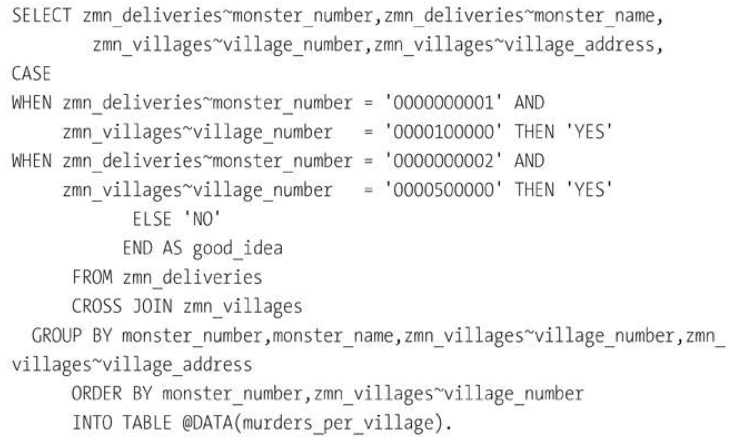
это CROSS JOIN, который очень зол и хочет выместить свою агрессию на всем мире.

Как вы знаете, внутреннее соединение возвращает только те записи, для которых выполняется некоторое условие в обеих таблицах базы данных, в то время как внешнее соединение возвращает все записи в одной из таблиц, плюс информацию в другой, если она находится в другой таблице, плюс информация из другой таблицы, если условие соединения выполняется.

При перекрестном соединении ни одна запись не остается без внимания. Результат эквивалентен условию ON в INNER JOIN всегда истинно. Другими словами, таблица, которая возвращается, содержит все строки из левой таблицы, умноженные на все строки из правой таблицы. Если в правой таблице нет строки, то значения из левой таблицы будут возвращены в любом случае.

Как можно себе представить, если вы не сформулируете условие WHERE должным образом, результат может быть огромным. На самом деле, это настолько опасно, что многие задаются вопросом, зачем SAP добавила эту функцию и SAP действительно советует быть крайне осторожными.

Одним из вариантов использования перекрестного соединения может быть отчет, в котором показаны все монстры, которые умноженное на количество возможных деревень. В каждом случае дается рекомендация о том, подходит ли данный монстр или нет. Вы можете увидеть такой запрос в Листинге 7.16.



Listing 7.16 Cross Join in ABAP

Допустим, есть два монстра и пять деревень - в этом случае у вас будет десять записей в таблице: все пять деревень для каждого монстра, плюс рекомендация о том подходит ли этот монстр для того, чтобы терроризировать эту деревню.

321

### 7.2.7 Завершение кода в операторах SELECT

Если бы вы набрали "SELECT", а затем просто начали вводить нужные вам поля - что мы всегда делали мы всегда так делали - завершение кода не будет иметь представления о том, какую таблицу вы хотите из какой таблицы вы хотите считать данные, поэтому оно не сможет предложить полезные предложения о том, какие поля вам могут понадобиться.

В версии 7.5 оператор SELECT нужно начинать с SELECT FROM ztmonster\_header FIELDS .... После этого, если вы наберете букву "M" (например) и попросите заполнить код, то вы увидите список всех полей в таблице, начинающихся с буквы М. На первый взгляд, это может показаться небольшим улучшением, но оно позволяет ускорить кодирование и сократить количество орфографических ошибок.

### 7.2.8 Заполнение таблицы базы данных обобщенными данными

Как это ни печально, но среди нас есть те, кто еще не работает с базой данных SAP HANA. Поэтому иногда необходимо запросить целую кучу таблиц базы данных, агрегировать результаты и сохранить эти данные в отдельной таблице базы данных, чтобы при появлении другого приложения оно могло быстро получить эти агрегированные данные без необходимости рыться в пятидесяти четырех различных таблицах базы данных, в которых находятся данные.

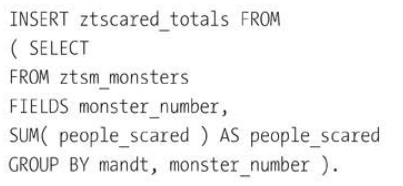
Оставив на секунду монстров в стороне, я однажды должен был написать программу, которая предоставляла водителям разбивку того, что вошло в их платежи ERS, сгенерированные из SAP через транзакцию MRRL. Разбивка начиналась с платежей на счета поставщиков и затем нужно было просмотреть около миллиона различных таблиц в стиле Шерлока Холмса, в конце концов, в таблицах условий ценообразования KOMP. Вы не захотите выполнять такие запросы, требующие больших объемов данных, больше, чем это абсолютно необходимо, а учитывая, что результат никогда не менялся после того, как платеж был произведен, использовалась отдельная таблица Z для хранения этих результатов.

В листинге 7.17 показано изменение, которое вносит SAP (если вы простите за каламбур). Традиционно, вы охотились за всеми этими таблицами базы данных, собирая эти данные обратно в набор внутренних таблиц на сервере приложений. Затем вы перебираете таблицы, собирая агрегированный результат в одну большую внутреннюю таблицу. Наконец, вы сохраняете эти данные в таблице Z агрегированного хранилища.

В настоящее время SAP хочет, чтобы вы сделали все это за один удар; изменения, внесенные в 7.4, сосредоточились на добавлении всей логики вычислений и агрегирования в оператор SQL, чтобы обработка не требовалась на сервере приложений. В версии 7.5 у вас теперь появилась возможность запрашивать данные, агрегировать их и сохранять обратно в отдельную в отдельную таблицу базы данных, не передавая ни одной записи на сервер приложений.

322

На простом примере, допустим, вы хотите хранить агрегированную таблицу того, сколько крестьян напугал каждый монстр. Код в листинге 7.17 будет обрабатывать всю операцию на уровне базы данных.

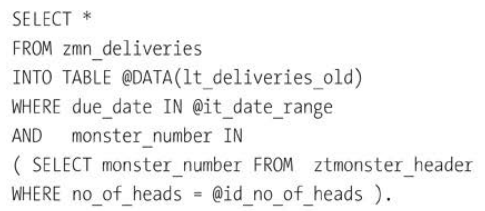


Listing 7.17 Populating Aggregated Database Table

### 7.2.9 Общие табличные выражения

Шикарный человек пропустит этот раздел, сказав: "Меня не интересуют эти обычные табличные выражения!" Все остальные, вероятно, задаются вопросом, что такое мире табличное выражение «общие». Ответ заключается в том, что они имеют отношение к (а) подзапросами и (b) читабельностью кода.

Уже давно программисты используют подзапросы в ABAP SQL, как в примере в листинге 7.18, в котором мы пытаемся получить все заказы для монстров с тремя головами.



Listing 7.18 Traditonal Way of Doing Subqueries in Open SQL

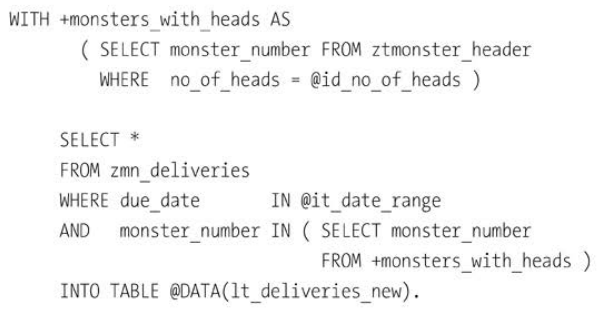
Это хорошо, когда у вас только один подзапрос, но вы можете попасть в ситуацию, в которой подзапросы ползают по вашему SQL-запросу, как муравьи на пикнике, и выражение становится очень трудно понять - и, следовательно, поддерживать в дальнейшем.

Я не сомневаюсь, что SAP пожелает, чтобы меня растерзали волки, когда я сделаю следующее сравнение, но общие табличные выражения подобны макросам, в которых вы определяете свой подзапрос, а затем используете этот "макрос" в операторе ABAP SQL немного позже. Еще один

323

способ описания этой концепции - разбить ваш запрос на более мелкие фрагменты, подобно тому как как вы разбиваете очень длинный ABAP-метод на несколько более мелких методов.

В примере в листинге 7.19 вы определяете подзапрос, используя новую конструкцию WITH (введенной в ABAP 7.51) и даете ему имя +MONSTERS\_WITH\_HEADS. Имя должно начинаться со знака плюс.



Listing 7.19 Subqueries from 7.51 Onward

На первый взгляд, это не кажется огромным преимуществом; если и есть, то вы увеличиваете количество строк кода. Дело в том, что вы можете дать этим подзапросам осмысленные имена, тем самым делая ваш код более похожим на простой английский язык. Это всегда одна из моих главных целей в жизни - облегчить жизнь моим коллегам и, конечно же, моему будущему "я", которому придется поддерживать этот код.

### 7.2.10 Более строгая проверка синтаксиса

Вам может быть знакома ситуация, когда в состоянии SELECT вы перечисляете десять полей, а в целевой рабочей области только девять или наоборот, а в целевой рабочей области только девять полей или наоборот. Раньше проверка синтаксиса была терпима, и вы получали предупреждение только в том случае, если выполнялась проверка Code Inspector check. Начиная с ABAP 1809, в таких случаях вы получаете жесткую синтаксическую ошибку. Это сделано для того, чтобы заставить вас не делать SELECT \* в маленькую рабочую область с шестью полями или не читать одно поле базы данных в структуру с двумя сотнями полей.

### 7.2.11 Юнит-тестирование SQL-выражений ABAP

Как уже упоминалось во введении, "четвертым мушкетером" новой модели программирования ABAP - это тестирование. Оно буквально лежит в основе всего. В главе 5 вы узнали

324

о том, как проводить модульное тестирование бизнес-логики, и в тот момент тестирование базы данных было за рамки: вы заменяли фактические вызовы базы данных вызовами тестового двойника, который отправлял обратно жестко закодированные результаты, не приближаясь к базе данных.

Пока все хорошо - но, конечно, это означает, что вы не тестируете свои SQL-запросы. Из-за концепции "вытеснения кода" ваши операторы SQL теперь делают гораздо больше, чем раньше - условная логика с помощью CASE-операторов и т.д. - и чем сложнее сложнее задача, которую выполняет любой программный модуль, тем больше шансов, что он будет тем больше шансов, что в нем будут ошибки. И мы не очень хотим, чтобы код попал в производство до того, как мы узнаем эту ужасную правду.

Поэтому компания SAP разработала концепцию OSQL (начиная с версии ABAP 7.51), которая расшифровывается как Open SQL Test Double Framework, неплохо сказано. Существует три этапа использования этой структуры, а именно:

1. Во время метода CLASS\_SETUP вашего тестового класса прокричите на весь мир, что вы собираетесь сделать модульные тесты для SQL-чтения одной или нескольких таблиц базы данных.

2. В методе GIVEN ваших тестовых методов объявите внутреннюю таблицу того же типа, что и таблица(и) базы данных того же типа, что и таблица(и) базы данных, которая будет удвоена, и заполните внутреннюю(ие) таблицу(ы) жестко закодированными данными, чтобы представить то, что будет находиться в реальной таблице (таблицах) базы данных.

3. Вставьте содержимое внутренней таблицы (таблиц) в структуру OSQL.

В этот момент происходит черная магия. Всякий раз, когда среда выполнения ABAP сталкивается с оператором SQL оператор, который читает из одной из назначенных таблиц базы данных, вместо того, чтобы читать из реальной базы данных, он считывает данные из внутренней таблицы. Таким образом, вы можете быть на 100% уверены, что ваш сложный SQL-оператор делает то, что должен (или что он безнадежно сломан) еще до того, как он попадет в QA. Как вы можете себе представить, этот трюк работает только во время модульных тестов.

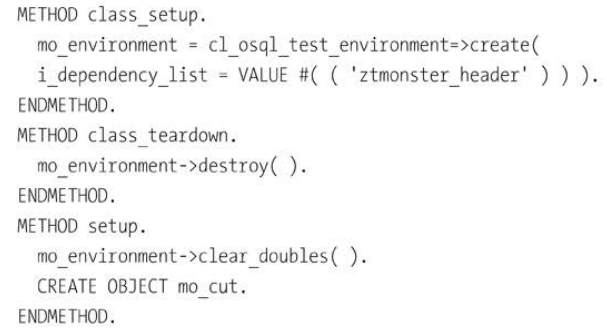
Таким образом, теоретически вам больше не нужно изменять производственный код, связанный с ABAP SQL, чтобы сделать код пригодным для тестирования. Тем не менее, вы должны иметь отдельный класс для чтения базы данных в любом случае, чтобы следовать принципам ОО, а именно, принципу единой ответственности.

Приведем пример: ранее в этой главе был рассмотрен сложный ABAP SQL-запрос с использованием конструкции CASE (листинг 7.2). Что ж. Я не доверяю этим новомодным причудливым конструкциям, поэтому я хочу провести модульное тестирование, чтобы доказать, что я действительно получаю правильный результат.

В листинге 7.20 показано, как подготовиться к тесту и убрать после него. В методе CLASS\_SETUP вы создаете так называемое окружение, в котором указывается, какая таблица (у вас может быть

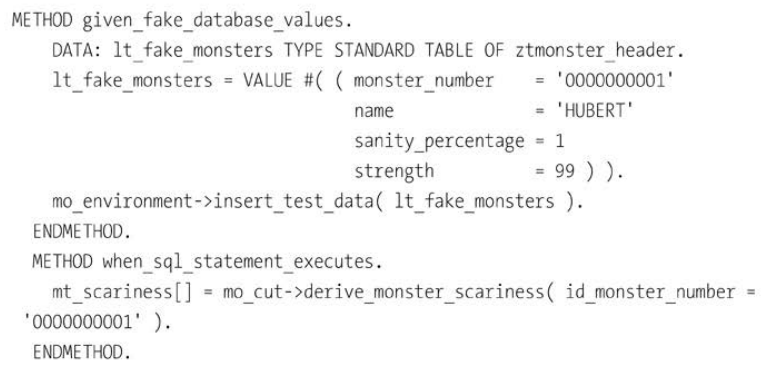
325

более одного; входной параметр - список) будет иметь тестовый дубль. Метод CLASS\_SETUP выполняется только один раз во время выполнения теста, а в конце всех тестов метод CLASS\_TEARDOWN очищает все, чтобы убедиться, что реальная база данных снова является главной снова. В методе setup (вызываемом перед каждым методом тестирования) метод CLEAR\_DOUBLES проверяет, что в среде все таблицы поддельной базы данных пусты.

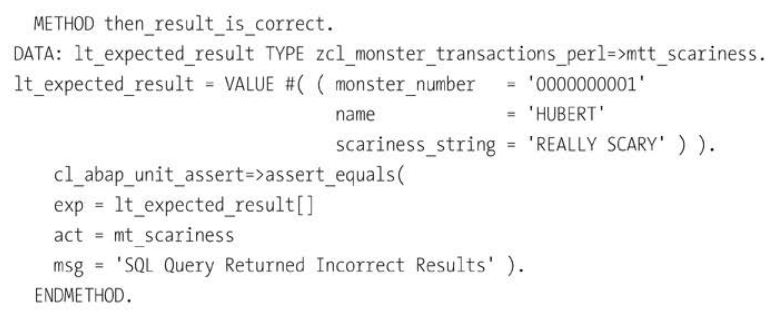


Listing 7.20 Setting Up and Tearing Down OSOL Tests

В листинге 7.21 приведен код собственно модульного теста. Разделите код на отдельные методы для GIVEN/WHEN/THEN. В методе GIVEN внутренняя таблица заполняется поддельными значениями, чтобы имитирующими реальную базу данных. Метод THEN вызывает производственный код, который фактически выполняет сложный SQL-запрос (ну, он думает, что выполняет, но из-за хитрости, которую мы разыгрываем, вместо этого считывается внутренняя таблица), а метод THEN проверяет что результат соответствует нашим ожиданиям.



326



Listing 7.21 Unit Test for SQL Statement

Таким образом, теперь вы можете проводить модульное тестирование не только бизнес-логики, но и операторов ABAP SQL. Это означает, что классы уровня персистентности могут иметь модульные тесты, так же как и классы бизнес-логики. Вау, ковбой (как сказал бы Роберт Мартин)!

CL\_OSQL\_REPLACE

Иногда SAP выпускает новые функции, которые я считаю глупыми, но я чувствую необходимость рассказать вам о них. Начиная с версии 7.52, вы можете использовать класс CL\_OSQL\_REPLACE, чтобы SQL-запрос читался не из внутренней таблицы, а из другой таблицы базы данных внутри вашей системы SAP, которая типизирована идентично настоящей таблице базы данных, но в ней размещены поддельные данные. Вы удаляете и вставляете записи в реальную таблицу базы данных во время модульного тестирования! Это безумие. Тем не менее, я должен был упомянуть об этом, и я упомянул.

## 7.3 CDS Views

Возможно, вы знакомы с такими телешоу о раскрытии преступлений, как CS! Майами. Так вот, на конференции SAP TechEd в октябре 2015 года можно было подумать, что конференция называется "CDS: Лас-Вегас". Представления CDS занимали центральное место практически в практически в каждой презентации, а SAPUI5 стоял позади него, размахивая руками, а SAP HANA стояла сзади и выглядела потерянной, пытаясь привлечь к себе внимание.

Два года спустя на мероприятии TechEd компания SAP объявила о новой модели программирования ABAP для SAP S/4HANA, и что одним из трех основных столпов этой модели будет представление CDS. Другими словами, SAP планирует, что все будущее ABAP программирования будет вращаться вокруг представлений CDS.

327

Поэтому, как программисту ABAP, было бы неплохо знать, что это такое!

Если вы найдете в Интернете слово CDS, то узнаете, что оно расшифровывается как "сельский танец и песня". В терминах SAP, однако, это означает основные службы данных, что не кажется не так захватывающе, как крик "Yeehaw!" во время езды на быке. Однако, как только вы увидите, что возможно внутри представления CDS, возможно, вы будете в таком же восторге. Это определенно превосходит эквивалент SE11.

Представления CDS можно назвать следующим поколением представлений баз данных. SAP утверждает, что если традиционное представление базы данных - это просто связь одной или нескольких таблиц, то представление CDS - это полноценная модель данных, которая, помимо того, что имеет дополнительные возможности, которых нет у SE11 определенных представлений, может использоваться даже приложениями, не входящими в домен SAP.

SAP HANA также может функционировать как отдельная платформа для разработки, и системы, не относящиеся к SAP. Системы должны иметь возможность использовать артефакты, созданные в SAP HANA, например, представления CDS. Вы можете создать представление CDS только в SAP HANA, но в этом примере вы создадите его в среде ABAP.

Представление CDS в сравнении с AMDP

Поначалу разница между представлением CDS и AMDP может показаться непонятной. В двух словах, при кодировании AMDP вы используете SQLScript, в то время как при создании представления CDS используется язык при создании представления CDS - это язык определения данных (DDL), который является расширением языка SQL. Это означает, что вы можете сделать гораздо больше в хранимой процедуре, чем в представлении CDS, потому что SQLScript имеет встроенные функции механизма вычислений, а также промежуточные шаги, локальные переменные и условную логику, такую как IF/ ELSE. Представление CDS – это просто большой оператор SELECT на стероидах; он может возвращать только один результат (строку или таблицу данных). Хранимая процедура гораздо больше похожа на метод и может возвращать столько параметров результата столько, сколько посчитает нужным.

Допустим, вы хотите создать представление в SE11, чтобы получить все данные о монстрах, владеющих определенных питомцев. Первое, что вы сделаете, это создадите представление базы данных в SE11, а затем выбрать, какие таблицы соединить - например, соединить элементы монстров с основной таблицей таблицу монстров (чего система достигает путем сравнения отношений внешних ключей и, таким образом, выполняя соединение на MONSTER\_NUMBER), а затем составить большой список полей из одной или нескольких связанных таблиц базы данных. Далее вы можете добавить некоторые жестко закодированные критерии отбора (например, вас интересуют только зеленые монстры или монстры с здравомыслием более 10%). Однако, если бы вы захотели выполнить внешнее соединение между таблицей монстров и таблицей monster pets и MONSTER\_NUMBER, чтобы получить подробную информацию о том, какие питомцы, если таковые имеются, были выбраны в таблице monster pets

328

монстров, то вам не повезет. Вы не можете использовать внешние соединения в базе данных SE11 представление. Вы должны создать представление, связывающее первые две таблицы, а затем выполнить внешнее соединение в вашей программе с помощью оператора SELECT между только что созданным представлением и таблицей монстров-животных. Создание представлений в SE11 работает хорошо, но, как и все разработки на основе форм в ABAP, это не так просто, она несколько неуклюжа и требует нажатия множества кнопок, выбора опций и перемещения, опций на всплывающих экранах и переходов от вкладки к вкладке.

К счастью, благодаря SAP HANA теперь есть новый объект ABAP-репозитория, который не только менее громоздкий, но и дает больше возможностей. Название ABAP объект репозитория называется DDL, и он выглядит как крошечная программа, состоящая из одного сложного оператора SELECT. Когда объект DDL создается в системе ABAP, в SAP HAP создается эквивалентное представление базы данных создается в SAP HANA (или в любой другой базе данных, которую использует ваша система), называемое представлением CDS. (Урезанную версию представления CDS можно увидеть в разделе SE11, но она не может быть изменена).

Напомним, что вместо основанного на форме подхода к созданию представлений с помощью SE11, вы задаете все с помощью кода.

Шаги по созданию и вызову представления CDS можно разбить на следующие:

1. Создание представления CDS в Eclipse

2. Кодирование представления CDS в Eclipse

3. Классификация представления CDS (необязательно, но рекомендуется)

4. Добавление проверок полномочий в представление CDS

5. Чтение представления CDS из программы ABAP

6. Модульное тестирование представления CDS

Далее каждый шаг рассматривается более подробно. Мы также включим дополнительный раздел посвященный некоторым дополнительным полезным соглашениям и инструментам для работы с представлениями CDS.

### 7.3.1 Создание представления CDS в Eclipse

Первым шагом в разработке нового представления CDS является создание нового источника DDL в вашей прекрасной среде разработки ABAP в Eclipse (вы не можете создать этот источник в SE80). Для этого нужно выбрать пакет, в котором вы хотите создать представление и следуя по пути меню New- Other- ABAP Repository Object- Core Data Services -DDL Source (см. рисунок 7.2).

После выбора опции DDL Source и нажатия кнопки Next откроется окно, показанное на рисунке 7.3.

329

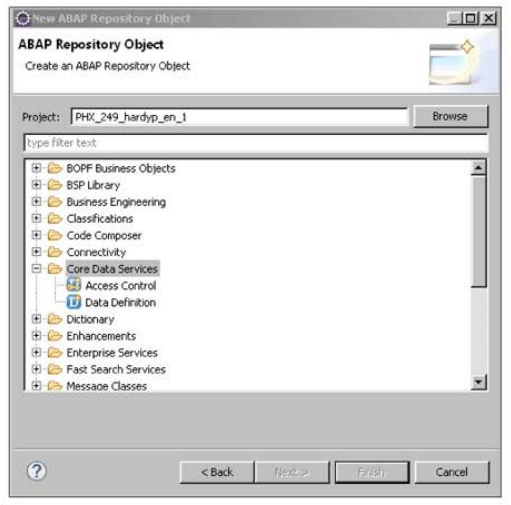


Figure 7.2 Creating DDL Source: Part 1

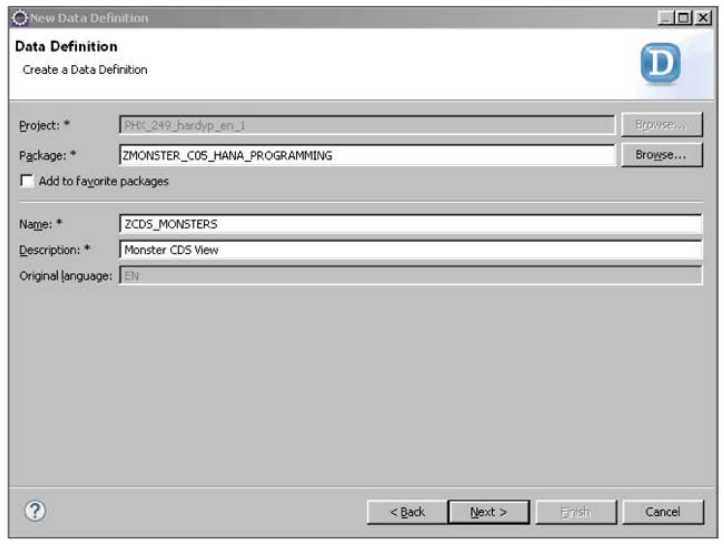


Figure 7.3 Creating DDL Source: Part 2

330

Введите имя, например, "ZCDS\_MONSTERS", и описание, затем нажмите кнопку Next. Если это не локальный объект, то вам будет предложено выбрать транспортный запрос. После того как вы его выберете, нажмите Готово. Для локального объекта просто нажмите Готово.

До версии 7.5 появлялся пустой экран, и вам приходилось кодировать каждую строку представления CDS самостоятельно. Многие люди считали это хорошей вещью, но, очевидно достаточно людей жаловались; теперь появляется экран, в котором вы выбираете шаблон (Рисунок 7.4).

После нажатия кнопки Finish вы вернетесь в обычный экран Eclipse с тем кодом, который был в нижнем поле (которое меняется в зависимости от выбранного шаблона), заполненным для вас. Текст на рисунке 7.4, который гласит DDL\_source\_description, был заменен на текст описанием, которое вы выбрали, а имя источника DDL\_source\_name\_editable было заменено именем выбранного вами представления CDS (например, ZCDS\_MONSTERS).

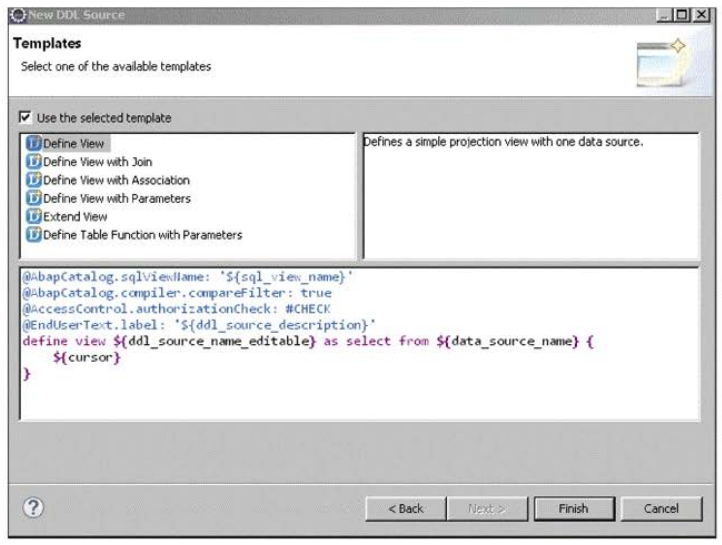


Figure 7.4 Creating DDL Source: Part 3

Что бы вы ни выбрали на последнем этапе - шаблон или не шаблон - следующий этап будет одинаковым; ваше представление CDS уже имеет имя, но иметь одно имя просто недостаточно. Вам нужно два имени. Новое имя предназначено для SQL представления, которое будет создано в словаре (то, которое вы сможете просматривать в SE11), а имя, которое вы уже использовали, - для представления SQL

331

- это имя для сущности представления CDS, которая просматривается и изменяется через Eclipse. Вы могли бы назвать и представление SQL, и представление CDS одинаково, но ... не стоит. Это разные вещи, поэтому имена должны отражать это. Назовите SQL-представление ZV\_MONSTERS и сущность ZCDS \_MONSTERS.

Первую можно увидеть в SE11; на вторую вы должны ссылаться в состоянии SELECT в ваших ABAP-программах. (Вы можете ссылаться на первый вариант в своих программах также, но это было бы некрасиво.)

### 7.3.2 Кодирование представления CDS в Eclipse

В конце предыдущего раздела вы выбрали шаблон, который привел вас в Eclipse с несколькими строками автогенерируемого кода. Для самых стойких вы также можете снять галочку с пункта. Использовать выбранный шаблон и продолжать кодировать все вручную, но мы будем использовать шаблоны. К концу этой главы мы рассмотрим все шаблоны, показанные на рисунке 7.4., каждый из которых демонстрирует важную функцию, которая может быть использована в представлении CDS.

В этом разделе будут рассмотрены первые пять, а последний шаблон покажет свою уродливую голову в разделе 7.4. Как вы видели на рисунке 7.4, есть несколько шаблонов на выбор, которые будут рассмотрены в порядке их перечисления, следующим образом:

- Определить вид/ Определить вид с присоединением

Это одно и то же, за исключением того, что первый читает только из одной таблицы, а второй читает из двух. В обоих случаях, как только вы начнете кодирование, вы можете изменить представление для чтения из любого количества таблиц.

- Определение представления с помощью ассоциации

С помощью этого шаблона вы познакомитесь с ассоциациями, типом объединения, которое предназначенный для того, чтобы быть семантически более простым в написании и понимании.

- Определение представления с параметрами

Представления SE11 могут иметь только жестко закодированные отборы; представления CDS могут иметь параметры, что делает выборки динамичными, как показано на примере этого шаблона.

- Расширить представление

Этот шаблон предназначен для случаев, когда представление CDS не принадлежит вам (например, стандартное представление SAP и вы хотите добавить некоторые поля, определяемые пользователем.

- Определить функции таблицы с параметрами

Этот способ предназначен для тех случаев, когда вы хотите выйти за рамки того, что возможно в стандартном представлении CDS и хотите вызвать функцию, использующую всю мощь базы данных SAP HANA (см. Раздел 7.4 для получения подробной информации).

332

Определить представление/определить представление с присоединением

Теперь пришло время написать DDL, который в конечном итоге создаст представление CDS, когда оно будет завершено. Как упоминалось ранее, он будет состоять из одного большого оператора SELECT. Если вы выбрали шаблон, вы увидите, что первые несколько строк уже написаны за вас с одной или несколькими аннотациями. Аннотация - это строка кода, которая начинается со знака @. После знака @ стоит термин, который описывает природу аннотации (например, термин AbapCatalog представляет настройки заголовка для представления).

Для первого примера выберите **Define View with Join**, чтобы начать с двух таблиц; на самом деле, в итоге нам понадобятся три таблицы, поэтому мы вручную добавим третью таблицу позже в этом разделе. Когда вы посмотрите на экран Eclipse, вы увидите, что код в листинге 7.22 уже написан для вас.

*@AbapCatalog.sqlViewName : 'sql\_view\_name'*

*@AbapCatalog.compiler.compareFilter: true*

*@AccessControl.authorizationCheck : #CHECK*

*@EndUserText.label : 'Monster CDS View'*

***define view*** *Zcds\_Monsters\_Join* ***as select from*** *data\_source\_name*

***left outer join*** *joined\_data\_source\_name*

***on*** *data\_source\_name.element\_name = joined\_data\_source\_name.joined\_element\_name {}*

Listing 7.22 Autogenerated CDS View Code

Вам всегда нужна хотя бы одна аннотация для определения имени SQL-вибратора, поэтому в первой строке вы измените местозаполнитель SOL\_VIEW\_NAME на фактическое имя представления базы данных, которое будет отображаться в SE11, следующим образом:

*@AbapCatalog.sqlViewName : 'ZV\_MONSTERS'*

Некоторые из этих автогенерируемых аннотаций являются обязательными, но есть также целый ряд других необязательных аннотаций, которые вы можете добавить. Например, если вы желаете (и только если базовая таблица или таблицы SAP, к которым вы будете обращаться, допускают буферизацию) вы можете задать некоторые параметры буферизации для представления во второй строке:

*@AbapCatalog.buffering.status: #ACTIVE*

*@AbapCatalog.buffering.type: #SINGLE*

В текущем примере в базовых таблицах не активирована буферизация, поэтому возникнет если представление CDS попытается объявить такую настройку, возникнет синтаксическая ошибка, но вам нужно знать, что это возможно.

333

Буферизация

Точно так же, как таблица DDIC может быть настроена на буферизацию одной записи, полную буферизацию и т.д. представление CDS может быть настроено таким же образом (вы всегда могли сделать это в SE11 для традиционных представлений). Можно подумать, что такие вещи больше не имеют смысла в базе данных SAP HANA, поскольку все равно все хранится в памяти, но буферизация так же важна, как и раньше. Эмпирическое правило гласит: если вы хотите получить некоторые хранимые данные, то вот ваши варианты, от самого быстрого к самому медленному:

- Считать внутреннюю таблицу

- Чтение из буфера таблицы

- Чтение из кэша базы данных

- Чтение из базы данных SAP HANA

- Чтение из скучной, квадратной базы данных, не хранящейся в памяти.

Поэтому, если у вас есть небольшая таблица базы данных, которая не часто меняется, буферизация ее в мире SAP HANA является хорошим решением, как это было всегда.

В OpenSQL нам никогда не нужно беспокоиться о добавлении клиента (MANDT) в SQL-запрос; система делает это за нас. В представлении CDS вы должны добавить аннотацию, чтобы унаследовать эту способность от базовой таблицы (при условии, что в ней есть поле MANDT!) следующим образом:

@ClientHandling.type :#INHERITED

С каждым новым выпуском ABAP появляется все больше аннотаций, которые приобретают все большее значение. Позже в книге - в главах 8 и 12- вы увидите, как аннотации позволяют представлению CDS передавать информацию о себе другим технологиям SAP, таким как BOPF и SAPUI5, и даже технологиям, не относящимся к SAP.

Идем дальше, после всех аннотаций. Точно так же, как процедура FORM начинается с FORM, а метод начинается с METHOD, представление CDS начинается со слов **define view**, как показано ниже:

define view zcds\_monsters as

После добавления аннотаций первые несколько строк представления будут выглядеть следующим образом: код, показанный в листинге 7.23.

*@AbapCatalog.sqlViewName: 'sql\_view\_name'*

*@AbapCatalog.compiler.compareFilter: true*

*@AccessControl.authorizationCheck: #CHECK*

*@ClientHandling.type : #INHERITED*

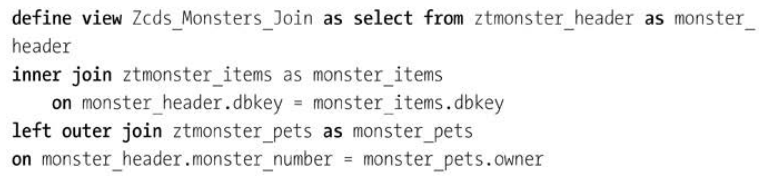
334

@EndUserText.label : 'Monster CDS View'

**define view** Zcds\_Monsters\_Join **as**

Listing 7.23 Header Settings of CDS View

С технической точки зрения, объект представления CDS определяется как SELECT из одного или нескольких источников данных (при этом источником данных является таблица или представление SAP или другое представление CDS представление). Естественно, следующей частью кодирования является оператор SELECT, который выглядит почти так же, как обычный оператор SELECT, который вы пишете в своих ABAP-программах (листинг 7.24). Если вы выбрали шаблон, то первые несколько слов были написаны за вас (не очень полезно - особенно потому, что код по умолчанию использует внешнее объединение, когда обычно требуется внутреннее объединение), а затем вы заменяете *data\_source\_name* и *joined\_data\_source\_name* на реальные имена таблиц базы данных SAP и добавляете любые другие соединения, которые вы хотите.



Listing 7.24 SELECT Statement

Существуют некоторые различия между оператором SELECT в традиционном ABAP SQL и эквивалентом представления CDS. Псевдонимы (как в TABLE\_NAME AS ALIAS) являются необязательными, но вы все же следует их использовать по причинам, которые сейчас станут понятны.

Псевдонимы необязательны в операторах ABAP SELECT и не часто используются за пределами SAP или кода, вырезанного и вставленного из примеров SAP. На самом деле, в интерактивной справке SAP примеры использования внутреннего объединения предлагают использовать псевдонимы T1 и T2, которые являются еще более бессмысленными, чем имена таблиц LIKP и LIPS.

При использовании представления CDS рекомендуется прямо противоположный подход; псевдонимы должны быть английские имена, а не немецкие аббревиатуры (например, *customer* вместо KNA1 и *material\_types* вместо T134). Я думаю, вам будет трудно привести аргументы в пользу того, чтобы не использовать псевдоним в последнем случае, хотя, конечно, некоторые люди любят тот факт, что что они знают, что означает T134, и никто другой не может об этом догадаться.

Далее, и это самая важная вещь в истории вселенной, вместо тильды (~) между именем таблицы и именем поля, у вас есть точка (т.е. вместо MONSTER~NUMBER у вас будет monster.number).

335

Кроме того, в традиционном ABAP после оператора SELECT стоит либо звездочка, обозначающая "все", либо список полей, которые вы хотите получить из таблиц базы данных. Вы увидите, что в представлении CDS вы сразу переходите к условиям объединения, прежде чем работать со списком полей, а бедное старое условие WHERE отходит на второй план.

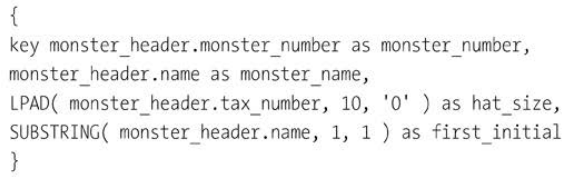
Один из аспектов, который остается таким же, как и в обычном операторе SELECT в ABAP, заключается в том, что при кодировании DDL, генерирующего представление CDS, вам не нужно помещать клиента (MANDT) в оператор SELECT; система выполнения выполняет это за вас, как вы и привыкли. В разделе 7.4 вы увидите, что для AMDP это не так.

Вы также заметите, что пример в листинге 7.24 содержит внешнее объединение, которое в базе данных SE11. На самом деле, в DDL, создающем представление CDS, вы можете выполнить не только левое внешнее объединение, но и правое внешнее объединение, объединение и объединение всех (не говоря уже о "union city blue"). Традиционные операторы ABAP SELECT здесь на много миль позади, хотя поддержка операции объединения была добавлена в ABAP 7.5.

Далее объявите список полей, как и в традиционном представлении SE11. В SE11 у вас есть два столбца: слева - удобные для человека имена, например, CUSTOMER и CUSTOMER\_NAME; справа - имена элементов данных, например, KUNNR и NAME1. В представлениях CDS все немного по-другому; вы собираетесь составить большой список удобных для человека имен для полей, которые будут доступны в представлении (вы все еще можете использовать немецкие аббревиатуры, если хотите), и, если хотите, можете использовать немецкие аббревиатуры, если хотите), а то, как они будут набраны, будет определено позже системой на основе того, какие значения из системы на основании того, какие значения из базы данных (или из расчета) будут помещены в них. Это похоже на оператор DATA( ), который вы рассматривали в гл. 2 при определении переменных в ABAP в том месте кода, в котором в них будет помещено значение, помещенное в них.

Вкратце, вы собираетесь определить имя каждого поля представления и то, как оно будет заполнено по одному за раз. Вместо того чтобы перечислять все поля после оператора SELECT, как это делается в запросе SQL, список полей и способы их заполнения обычно приводятся в теле кода, после критериев SELECT и JOIN, но перед предложением WHERE.

Этот список полей заключен в фигурные скобки ({}), как показано в листинге 7.25.



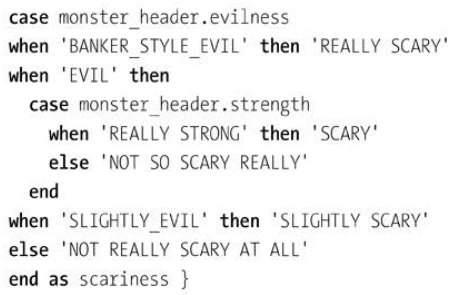
Listing 7.25 List of Fields for CDS View to Get from Database

336

В этом примере каждый раз объявляется псевдоним (например, AS monster\_number), но это необязательно; псевдоним действительно нужен, только если используется выражение или два поля из разных таблиц имеют одинаковые имена. Целевое поле не имеет явного объявления типа; оно типизируется в соответствии со значением, переданным в него. В листинге 7.25 вы заметите, что есть встроенные функции для случаев, когда вы хотите вернуть налоговый номер 3 как 0000000003 и вернуть первые инициалы монстра по имени Фред как "F". Существует таких функций очень много, и с каждым новым стеком поддержки их становится все больше. Для получения полного списка нажмите «F1» в ABAP в Eclipse.

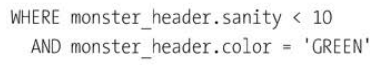
Еще в главе 3 вы увидели, что CASE-операторы пробрались в SQL-операторы ABAP, и как бы вы не кричали "Назад! Назад! Вернитесь туда, где вы когда-то вернитесь туда, где вы когда-то были!", это факт жизни в ABAP 7.4 и выше, что CASE операторы идут куда хотят и когда хотят (чванливые, высокомерные задиры, какими они и являются).

Здесь они могут быть использованы внутри представления CDS, как показано в листинге 7.26. Обратите внимание, что они также могут быть вложенными, и есть необязательный пункт ELSE.



Listing 7.26 CASE Statement within CDS View

Закрывающая фигурная скобка в конце листинга 7.26 означает, что список требуемых полей заканчивается, и вы можете добавить предложение WHERE сейчас, если хотите, как показано в листинге 7.27.



Listing 7.27 WHERE Clause

Часто в представлениях SE11 имеются жестко закодированные ограничения, например, такие, как показано в списке 7.27. Возможно, вы ограничите представление на VBAK только заказами на продажу, сказав AUART = C. То же самое в представлении CDS: жестко закодированные ограничения встречаются не часто и не всегда

337

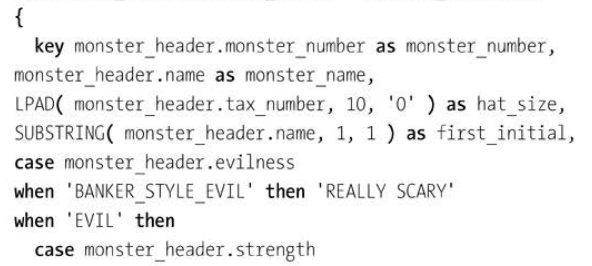
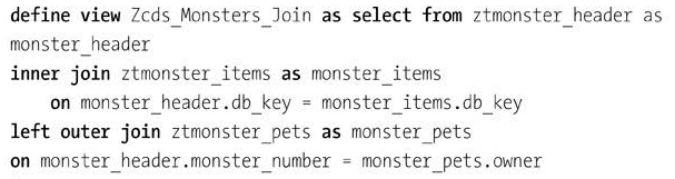
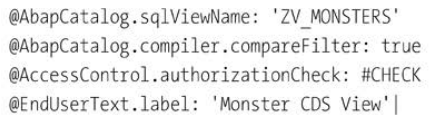
и, скорее всего, будут использоваться редко. Вы увидите, как обойти необходимость жестко закодированных условий WHERE в одном из следующих разделов, когда речь пойдет о представлениях с параметрами.

Следует отметить, что при кодировании DDL оператор WHERE может иметь значения, которыене имеют реального эквивалента в ABAP (например, IS NULL и IS NOT NULL). Почти каждый другой язык программирования, кроме ABAP, использует концепцию NULL для обозначения того, что значение данных не существует в базе данных. В ABAP вы привыкли говорить "равно пробелу" для текстового поля или "равно 0" для числа или количества. Это прекрасно работает, но это заставляет IT пуристов разрыдаться и биться головой об стол на протяжении многих лет.

В конце кода представления вы можете добавить предложение GROUP BY, если хотите. (Делать это не имеет большого смысла в данном примере, поскольку вас интересуют детали отдельных монстров, а не агрегированные данные по группам монстров).

В запросе ABAP SQL всегда есть предложение INTO, но в определении представления нет необходимости в предложении INTO в определении представления, потому что цель для извлеченных данных будет определена тем кодом, который запрашивает представление.

Прежде чем перейти к следующему разделу, стоит посмотреть на полный исходный код DDL исходного кода, который будет генерировать представление CDS (листинг 7.28).



338

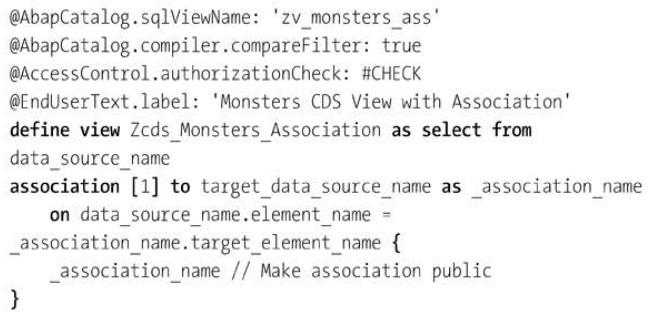


Listing 7.28 Complete DDL Source for Generating CDS View

Определение представления с помощью ассоциации

Используя те же шаги, что и ранее, создайте новый источник DDL в Eclipse, но на этот раз выберите шаблон Define View with Association и назовите представление ZCDS\_MONSTERS\_ASSOCIATION (что звучит как отличный клуб для вступления).

Вы не можете дать представлению SQL (представлению SE11) такое длинное имя, потому что оно ограничено шестнадцатью символами. В этом примере я должен использовать аббревиатуру, поэтому я изменяю имя SQL-представления из значения-заполнителя на ZV\_MONSTERS\_ASS, в результате чего получается автоматически сгенерированный код в листинге 7.29.



Listing 7.29 Autogenerated Code for CDS View with Association

На данном этапе, прежде чем заполнять недостающий код, вам необходимо понять всю концепцию ассоциаций. SAP хочет, чтобы вы использовали ассоциации вместо объединений, потому что они ближе к "концептуальному мышлению". Ассоциация не является соединением как таковым; это просто метаданные о возможном соединении, которые хранятся в словаре данных. Реальное соединение создается, когда ассоциация используется в выражении пути.

339

Идея о том, что две таблицы имеют связь по внешнему ключу, не является сложной для нас, программистов, но она не соответствует тому, как работает мозг. Если мы заказываем три разных монстров, мы не делаем сознательного различия между заголовком и данными о предмете, и очевидно, что должен быть хотя бы один предмет; мы не можем сделать заказ вообще без монстров. Это было бы бессмысленно. Точно так же, когда мы смотрим на автомобиль компании, мы не думаем о нем как о двадцати шести взаимосвязанных таблицах данных, все из которых начинаются с букв ANL.

Ранее вы узнали, что представления CDS поощряют использование псевдонимов для представления осмысленных и полное название для имен таблиц и полей для внешнего мира, в отличие от представления загадочных аббревиатур. Для примера из области основных данных поставщика. Вы часто объединяете таблицы LFA1 и LFM1, чтобы найти данные о закупках для поставщика. Используя ассоциацию, вы могли бы сослаться на Supplier.Purchasing-Data.BlockingFlag вместо LFM1~SPERM. Я уверен, что вы согласитесь, что первый вариант лучше по разного рода причинам.

Обратите внимание на соглашение об именовании ассоциаций: они используют так называемый верблюжий регистр, поэтому вместо подчеркивания между словами, все они задаются как одно большое слово с заглавными буквами, указывающими на начало нового слова, как в SalesOrder. На самом деле, SAP рекомендует использовать верблюжий регистр для всех имен внутри представления CDS.

В ассоциации объединение также обозначается точкой между именами таблиц, так что вы Orderltem.Customer.Address указывает на объединение между VBAP, KNA1 и ADRC. Все эта семантическая информация является внутренней для DDL внутри представления CDS. Идея заключается в том, что ассоциации являются вложенными; вы создаете одно представление ассоциации, которое связывает адрес с клиента, затем повторно используете это представление, связывая его с элементом заказа вместо прямой ссылки на заказчика.

Следующий пример соединяет таблицу Monster Header с таблицей Monster Pet s, на этот раз используя ассоциацию вместо обычного соединения. Как и раньше, замените место источника данных на имена таблиц, а имя элемента - на поля объединения. Однако в этот раз есть некоторые отличия; во-первых, добавьте кардинальность как значение между квадратными скобками сразу после слова association. Поскольку монстр может вообще не иметь питомцев или иметь их неограниченное количество, задайте значение кардинальности как [0 ... \*]. Если вы присоединяете таблицу элементов к таблице заголовков, вы бы задали кардинальность [1 ... \*], а если бы вы объединить два поля с одинаковым первичным ключом (например, AUFK и AFKO), вы бы задали значение кардинальность как [1 ... 1]. (Я не думаю, что установка кардинальности влияет на что-либо; это просто семантическая информация, помогающая описать модель данных случайному читателю.)

Еще одно отличие между ассоциацией и обычным объединением заключается в том, что, хотя это необязательно, SAP рекомендует начинать имя ассоциации с подчеркивания

340

чтобы отличить их от стандартных полей в представлении CDS, как в \_Pets. Окончательный код выглядит как листинг 7.30.



Listing 7.30 Coding CDS View Using Association

Определение представления с помощью параметров

Как упоминалось ранее, одним из основных ограничений традиционных представлений SE11 было то, что вы могли иметь параметры выбора, но они были жестко закодированы (например, цвет монстра задается как ЗЕЛЕНЫЙ). Поскольку жесткое кодирование во всех его многочисленных формах - это явно работа дьявола, лучше уйти от него и кодировать представление CDS с параметрами. Рискуя научить рыбу плавать, вы, несомненно, знаете об обычном процессе, в ходе которого, когда требуется динамический выбор (например, пользователь выбирает то, что он пользователь выбирает то, что ему нужно на экране выбора), вы добавляете в свою ABAP-программу пункты WHERE, чтобы ограничить то, что возвращает представление. Пример этого показан в листинге 7.31.

SELECT \*

FROM zv\_monster\_pars

WHERE color EQ p\_color

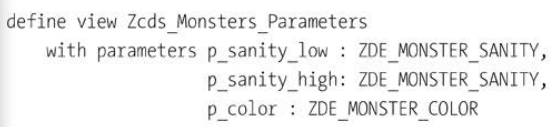
AND sanity IN s\_sanity.

Listing 7.31 Coding Parameters in Traditional ABAP

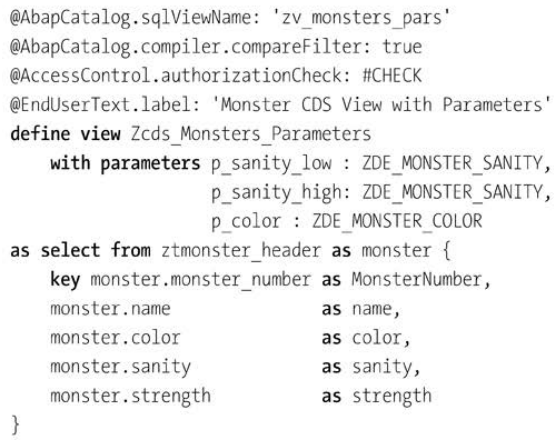
341

Вы можете сделать то же самое, обратившись к представлению CDS в операторе SELECT в вашей ABAP-программе, используя ABAP SQL; вы получите набор результатов, ограниченный как динамическими, так и жестко закодированными ограничениями. Когда вы начинаете читать о создании CDS представлений с параметрами, вы можете подумать, что это бред, но, пожалуйста, потерпите, все станет ясно.

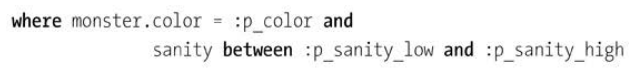
Начиная с SP 8 в версии 7.4 и непосредственно в версии 7.5, представления CDS могут быть определены с параметрами которые передаются через ABAP-программу во время выполнения. Например, создайте новое представление CDS представление, на этот раз используя шаблон Define View with Parameters. Как и раньше, здесь есть некоторый автогенерируемый код - нет необходимости показывать это; вы уже поняли идею - после чего вы заменяете заполнители и добавляете некоторые параметры. Они располагаются после строки define view line и состоят из слов with parameters, за которыми следует один или несколько параметров в формате имя *параметра: элемент данных*, например, p\_color: ZOE\_MONSTER\_COLOR, например:



Основная часть кода в DDL точно такая же, как и в предыдущих примерах и указывает, какие таблицы являются источниками данных и какие поля вы хотите вернуть. В данном случае, однако, предложение WHERE в конце состоит не из жестко закодированных отборов, а скорее параметров, которые вы определили в начале. Результат показан в листинге 7.32.



342



Listing 7.32 Defining CDS View with Parameters

Затем в своей программе вы пишете оператор, подобный тому, что приведен в листинге 7.33.



Listing 7.33 Calling CDS View with Parameters

Вы можете иметь смесь параметров и обычных пунктов WHERE, добавив жестко закодированное условие WHERE в операторе ABAP SQL (т.е. ситуация, противоположная обычной, в которой представление имеет жестко закодированные параметры, а ABAP SQL имеет жестко закодированные условия) обычной, в которой представление имеет жестко закодированные параметры, а ABAP SQL - динамические).

Возможно, вы начинаете недоумевать. Вы посмотрели на код ABAP для чтения представления CDS, и он практически такой же, как и для представления CDS без параметров. В примере, вы могли бы не беспокоиться о включении параметра цвета в представление и вместо этого выбрать цвет в операторе ABAP SQL обычным способом. Кроме того, если вы наблюдательны, вы могли заметить, что представления CDS не могут обрабатывать SELECT-OPTIONS должным образом.

Так есть ли метод в этом безумии? Как оказалось, есть. В определении представления CDS, параметры могут использоваться в любом месте списка полей, которые можно быть возвращены. Вы можете использовать сам параметр в качестве возвращаемого поля - что было бы немного странно, но возможно. Обычно возвращаемое поле - это результат функции, в которую вы передаете входной параметр или, в которую вы передаете входной параметр, или в CASE-операторе, основанном на значении параметра. Более того, CDS представление часто читают другие CDS представления, и параметры могут быть использованы для ограничения значений, возвращаемых из любого такого представления, которое читается.

Кроме того, с помощью предложений WHERE в ABAP SQL можно обращаться только к списку полей базы данных, определенных в представлении. Параметры представления CDS, с другой стороны, могут быть любыми по вашему усмотрению. Например, у вас может не быть поля под названием "Режим безумия" в списке, но вы можете легко создать параметр P\_MMODE. Затем пользователь

343

выбирает режим безумия на экране выбора, и выбранное значение передается в CDS представление, где всевозможная странная логика изменяет возвращаемые значения других полей в зависимости от режима безумия.

*Внимание: Хьюстон, у нас (может быть) проблема*

Предупреждение: Представления CDS с параметрами работают только в самых последних версиях. Начиная с версии ASAP 7.4 SP 8, только для некоторых баз данных (например, SAP HANA) существует поддержка, чтобы программа ABAP могла выполнять SELECT для представлений CDS с параметрами. В более высоких уровнях пакета поддержки версии 7.4 и в версии ABAP 7.5 все базы данных теперь могут воспользоваться преимуществами этой функции, хотя производительность значительно варьируется поскольку большинство оптимизаторов баз данных, не относящихся к SAP HANA, обычно начинают пускать пену у рта, когда встречают представление с параметрами.

**Расширенное представление**

Последний из шаблонов, которые вы рассмотрите в этом разделе, - это шаблон расширения для представления CDS. В мире представлений SE11, если SAP создал представление, то вы можете добавить к нему структуру APPEND с некоторыми собственными полями (например, ... Z-поля, которые вы добавили в таблицу VBAK, могут быть добавлены к стандартному представлению VBAKUK).

В этом примере вы хотите прочитать некоторые данные о некоторых клиентах барона, которые, как вы уже знаете, это злые землевладельцы в шапках. Стандартное представление SAP CDS\_KNA1\_NAME, которое делает некоторые замечательные вещи, когда дело доходит до форматирования имени клиента, будет нашим инструментом, но по какой-то причине стандартные основные данные клиента в SAP нет полей для размера шляпы или закрученности усов.

Чтобы продолжить работу, необходимо добавить некоторые поля в стандартные основные данные (в нашем примере – в таблице базы данных KNA1; в конце концов, данные должны где-то жить). Вы захотите, чтобы эти поля были в стандартном представлении CDS.

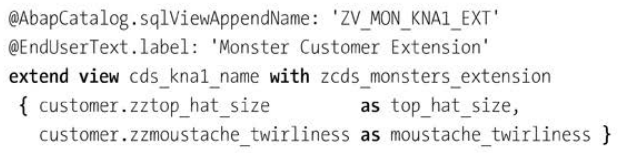
Catching Some Zs

В качестве примечания, правила SAP требуют, чтобы имена полей в расширениях таблиц SE11 начинались с буквы Z.

Создайте новое определение представления CDS, выберите шаблон Extend View и назовите представление CDS ZCDS\_MONSTERS\_EXTENSION и представление SE11 ZV\_MON\_KNA1\_EXT. На этот раз код будет читаться почти точно так же, как обычный английский; если бы вы писали на английском языке

344

предложение, вы скажете базе данных: "Я хочу расширить представление CDS\_KNA1\_NAME с помощью расширением под названием ZCDS\_MONSTERS\_EXTENSION, и вот список новых полей для клиентов полей, которые мне нужны". Вы увидите, что код в листинге 7.34 соответствует этому формату.



Listing 7.34 CDS View Extension

Вы можете удивиться, почему вместо KNA1 используется слово customer; это потому, что в оригинальном стандартном представлении SAP CDS, KNA1 был "затенен" псевдонимом customer, чтобы улучшить читаемость кода.

### 7.3.3 Классификация представления CDS

Вы, несомненно, знакомы с созданием прозрачной таблицы в SE11 и необходимостью заполнять технических параметров, таких как указание того, является ли таблица настраиваемой или прикладная таблица, количество ожидаемых записей и так далее, и вы, вероятно, втайне задавали себе вопрос, действительно ли классификация таблицы как APPL1 или APPL2 имеет хоть малейшую разницу.

То же самое можно проделать с представлением CDS. Как вы видели в предыдущих разделах, это не обязательно, но это хорошая идея - классифицировать ваши представления, чтобы помочь бедной старой базе данных база данных выполнять свою работу наилучшим образом. Например, в базе данных SAP HANA есть несколько кэшей и вы можете действительно помочь, если добавите аннотацию, подобную следующей:

*@ObjectModel.usageType.dataClass : 'CUSTOMIZING'.*

Это поможет базе данных выбрать правильный кэш и тем самым ускорить обработку. То, как SAP HANA оптимизирует запросы к базе данных, выходит за рамки этой книги, но если вы прочитаете книгу SAP PRESS "ABAP Development for SAP HANA" (см. раздел "Рекомендованная литература" в конце этой книги), вы увидите, что для выбора наилучшей базы данных используется много сложной логики для выбора наилучшей возможной стратегии доступа к базе данных, а также для того, чтобы сообщить базе данных как таблица может быть использована, будет действительно полезно.

Помните, что это компьютер: он поверит в любую чушь, которую вы ему скажете, поэтому убедитесь, что вы классифицируете представление CDS настолько точно, насколько это возможно. Возможные значения для класса данных (какие данные будет содержать представление CDS) следующие:

345

• TRANSACTIONAL (например, VBAP)

• MASTER (как в основных данных - например, KNA1)

• ORGANIZATIONAL (например, таблицы в организационной части IMG)

• CUSTOMIZING (другие таблицы IMG - например, TVAK)

• META (данные о данных; я не могу представить, что вы будете использовать эту таблицу)

• MIXED (если вы выполняете объединение основных и транзакционных данных, скажем)

Следующая аннотация, которую вы используете для классификации, выглядит следующим образом:

@ObjectModel.usageType.sizeCategory : 'XL'

Это почти то же самое, к чему вы привыкли в SE11. Возможные значения (количество ожидаемых строк таблицы базы данных в производстве) следующие:

- S = менее 1,000

- M = менее 100,000

- L = менее 10,000,000

- XL = менее 100 000 000

- XXL = больше 100 000 000

Я знаю, о чем вы думаете: в базе данных SAP HANA даже нет строк (rows)! Что ж, я рад, что вы не растерялись и заметили это. На самом деле может и не быть строк, но при выводе данных все равно создается впечатление, что они есть, поэтому подумайте о том, что бы вы ожидали, если бы сделали запрос "количество записей" из SE16.

Третий элемент аннотаций классификации выглядит следующим образом:

*@ObjectModel.usageType.serviceQuality : 'A'*

Это руководство по тому, сколько времени может занять запрос и какого рода приложения будут использовать (потреблять) представление CDS. Возможные значения следующие:

- A: бизнес-логика/большие объемы транзакций (1-2 мс)

- B: бизнес-логика/обычные транзакции (2- 5 мс)

- C: получение одного объекта для отображения в пользовательском интерфейсе (10-20 мс)

- D: аналитика (<500 мс)

- X: передача действительно сложной логики в базу данных SAP HANA (варьируется).

Очевидно, что иногда вам придется сделать предположение о том, как будет использоваться ваше представление, но пусть это будет лучшее предположение!

346

### 7.3.4 Добавление проверок полномочий в представление CDS

В примерах определений представлений CDS вы, возможно, заметили загадочную аннотацию в начале, которая гласит @AccessControl.authorizationCheck: #CHECK.

Кроме того, для таких определений представления CDS вы увидите в Eclipse желтый светофор говорящий No Access Control for Entity ZCDS\_MONSTERS Please Create DCL. До сих пор мы говорили об источниках DDL, которые создают представление CDS, но также можно создать язык управления данными (DCL) для представления CDS, что скорее похоже на AUTHORITY-CHECK внутри ABAP-программ. Если бы вы были математически подкованы, вы могли бы написать это как DDL + DCL = представление CDS.

Поскольку синтаксическая проверка Eclipse просит вас создать DCL для вашего представления Monster Parameter, то создать его необходимо. Для этого примера предположим, что только определенные люди могут просматривать детали зеленых монстров, потому что зеленые монстры являются секретными.

Перейдите в File- New- Other- ABAP Repository Object- Core Data Services- DCL Source. Вызовите новый объект ZDCL\_MONSTERS\_PARAMETERS, дайте ему длинное описание, и вы увидите всплывающее окно с предложением выбрать шаблон (рисунок 7.5). На момент написания статьи у вас есть только один вариант, но автоматически сгенерированный код указывает вам на правильном направлении.

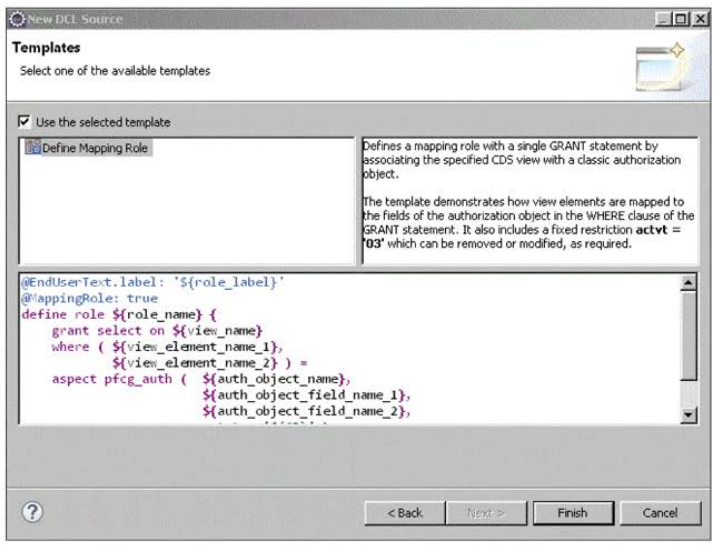
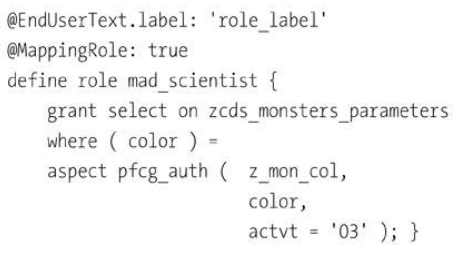


Figure 7.5 Creating DCL Source

347

Как выяснилось, в транзакции SU21 уже определен объект полномочий, называемый Z\_MON\_COL, который ограничивает полномочия на основе цвета. Вам просто нужно связать поле в представлении CDS с этим объектом полномочий, как показано в листинге 7.35.



Listing 7.35 Coding DCL Source

Это означает, что если вы запускаете отчет по представлению CDS и у вас нет полномочий в вашем мастере пользователя для определенных цветных монстров, то записи для монстров, которые вы не уполномочены просматривать, не появятся. Сотрудникам службы безопасности SAP не следует беспокоиться; существующие объекты полномочий и роли остаются неизменными. Разница заключается в том, что разработчикам ABAP больше не нужно явно кодировать проверку полномочий в отчетах читаемых из представлений CDS (что очень хорошо; это то, что они, как правило, забывают сделать в любом случае). Плохая новость заключается в том, что сбой не отображается в транзакции SU53, хотя я полагаю, что со временем это будет исправлено.

С технической точки зрения, все происходит так, что дополнительный параметр доступа-кондиции параметр доступа был добавлен к представлению CDS, поверх всех жестко закодированных и динамических параметров (т.е. если вы не хотите просматривать зеленых монстров, то MONSTER NE ‘GREEN’ - добавляется в фоновом режиме в качестве условия выбора). Вы можете увидеть это в STO5 – трассировке запроса к базе данных под загадочно выглядящим дополнением под названием RANGE\_RESTRICTION в условии WHERE.

Если вся эта концепция наполняет вас ужасом, измените DDL представления CDS так, чтобы аннотация гласила

*AccessControl.authorizationCheck: #NOT ALLOWED.*

### 7.3.5 Чтение представления CDS из программы ABAP

Когда вы закончите кодирование DDL в Eclipse, нажмите на значок Generate, и представление CDS будет сгенерировано внутри базы данных.

CDS представление генерируется внутри базы данных (вы можете только просматривать и изменять его из Eclipse только через объект DDL). Также создается представление SE11 в системе ABAP.

348

Если, как и я, вы любопытны, то вы быстро посмотрите на транспортный запрос, который был создан. Там вы увидите, что объекты хранилища будут не теми, которые вы получаете при создании представления SE11, а скорее объект R3TR/DDLS, так что все дополнительные возможности могут быть храниться. Как и в стандартном SE11 представлении, определение находится в ASAP, но при создании или изменении оно инициирует операцию CREATE в базе данных, подключенной к системе ABAP. Поэтому база данных всегда будет синхронизирована с системой ABAP, и когда вы переносите определение представления CDS через обычную транспортную систему, вам не нужно беспокоиться о том, чтобы вручную поддерживать базу данных в актуальном состоянии.

В своем коде вы должны ссылаться только на сущность представления CDS, а не на представление SE11, как показано в листинге 7.36. Завершение кода работает для имен представлений CDS так же, как и для таблиц и стандартных представлений. В листинге 7.36 приведен пример чтения всего представления во внутреннюю таблицу, а также пример того, как можно не беспокоиться о такой таблице и просто воспроизводить содержимое представления непосредственно в сетке ALV. В обоих случаях обратите внимание на имя сущности представления CDS, используемой в коде.

\* Чтение представления в внутреннюю таблицу

*SELECT \* FROM zcds\_monsters\_join "Use COS entity name*

*INTO TABLE @DATA(lt\_monsters) .*

\* Или просто отобразить содержимое напрямую

*cl\_salv\_gui\_table\_ida=>create\_for\_cds\_view( 'ZCOS\_MONSTERS\_JOIN' )->fullscreen( )->display( ) .*

Listing 7.36 Reading Data from CDS View Entity in ABAP

При выполнении SELECT на сущности представления CDS вы должны использовать синтаксис ABAP SQL (как описано в разделе 7.2), что означает, что переменные должны иметь знак @ перед собой, а предложение INTO должно быть в конце перед ними, а предложение INTO должно располагаться в конце. Я полагаю, вы знаете, как посмотреть содержимое реальной таблицы базы данных, например, VBAK . Вы также можете ввести сгенерированное имя представления CDS в SE11/SE16, чтобы увидеть содержимое, но SAP посчитает вас непослушным и отправит спать без ужина. Я подозреваю, что SAP даже не хочет, чтобы вы смотрели на представление SE11, но я уже поужинал, поэтому я собираюсь показать вам именно это. Во-первых, на рисунке 7.6 вы видите представление CDS (как было определено в Разделе 7.3.1) в Eclipse.

Как вы узнали в этом разделе, это представление определяет условия соединения для различных таблицы базы данных, а затем между скобками {} перечисляются поля представления, а также инструкции по заполнению этих полей. В SE11 вы можете увидеть представление соединений базы данных для представления ZV\_MONSTERS\_SQL (Рисунок 7.7).

349

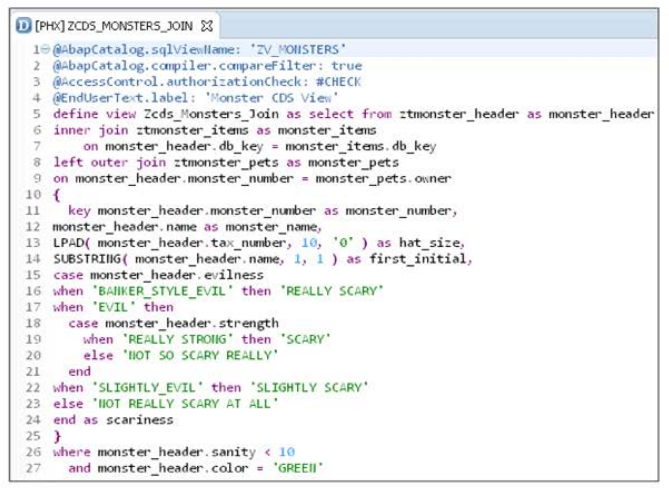


Figure 7.6 CDS View in Eclipse

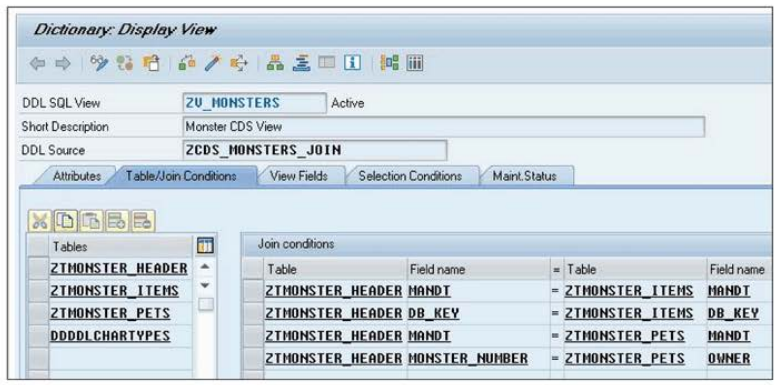


Figure 7.7 CDS View in SE11: Database Join Conditions

Обратите внимание на новый тип представления: вместо представления обслуживания (maintenance view) или любого другого обычного это представление DDL SQL. Также обратите внимание на дополнительное поле под именем представления с именем базового CDS представления (DDL Source). Далее посмотрите на поля, которые можно получить из представления, перейдя на вкладку View Fields (Поля представления) (Рисунок 7.8).

350

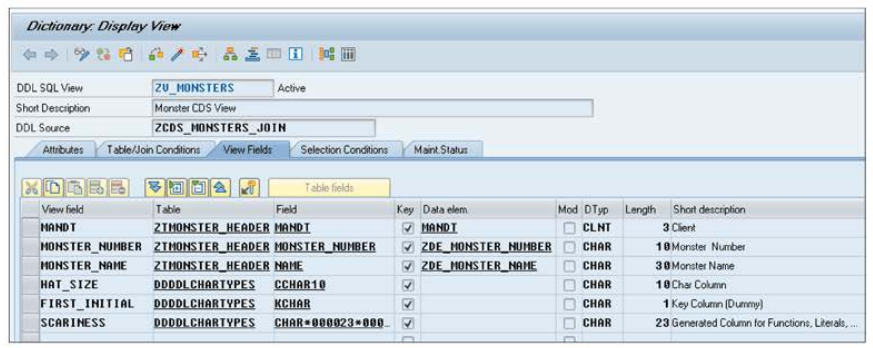


Figure 7.8 CDS View in SE11: Result Fields

Вы заметите, что система SAP была очень умна и разработала тип и длины полей HAT SIZE, FIRST\_INITIAL и так далее из представления CDS, полей, которые не ссылаются напрямую на поле таблицы базы данных и, следовательно, не привязаны к элементу данных.

Поле SCARINESS имеет странное значение в столбце Field, обозначающее некоторую форму действительно сложной логики - в данном примере CASE-конструкцию.

Вкладка Selection Conditions содержит именно то, что вы могли ожидать, а именно жестко закодированные отборы, которые говорят, что представление ищет зеленых монстров с разумностью ниже 10%.

### 7.3.6 Единичное тестирование представлений CDS

В разделе 7.2.11 вы узнали о фреймворке OSQL для выполнения модульных тестов ABAP SQL-запросов. Поэтому для вас, возможно, не будет шоком тот факт, что существует аналогичная структура для модульного тестирования представлений CDS, называемая (что неудивительно) двойным тестовым фреймворком CDS, также выпущенный вместе с ABAP 7.51.

Принцип работы точно такой же: вы сигнализируете среде выполнения, что когда производственный код читает из представления CDS, чтобы получить некоторые данные, не обращаться к базе данных, а вместо этого читать из внутренней таблицы, заполненной жестко закодированными данными.

Разница здесь в том, что представление CDS состоит из одной или нескольких зависимостей, к которым оно должно получить доступ, чтобы применить свою бизнес-логику. Зависимость в данном случае - это таблица базы данных или другое представление CDS. Последнее более вероятно, поскольку SAP поощряет вас выстраивать представления CDS в иерархию, при этом обычно только представление CDS верхнего уровня фактически считывается.

351

Итак, вы увидите пример, очень похожий на тот, который вы видели в разделе 7.2.11. только на этот раз не само представление CDS заменяется тестовым дублем, а скорее источник(и) данных внутри представления CDS. Таким образом, тестируется логика внутри представления CDS.

Листинг 7.37 почти на 100% идентичен тому, что вы видели в листинге 7.20. Единственное отличие заключается в том, что на этот раз это представление CDS, которое будет перехвачено системой в тот момент, когда при обращении к нему в производственном коде. Обратите внимание, что одновременно можно тестировать только одно представление CDS, что вполне логично; было бы странно, если бы вы делали JOIN на двух представлениях CDS.

*METHOD class\_setup.*

*mo\_environment = cl\_cds\_test\_environment=>create(*

*i\_for\_entity = 'ZCDS\_MONSTERS' ) .*

*ENDMETHOD.*

*METHOD class teardown.*

*mo\_environment->destroy( ) .*

*ENDMETHOD.*

*METHOD setup.*

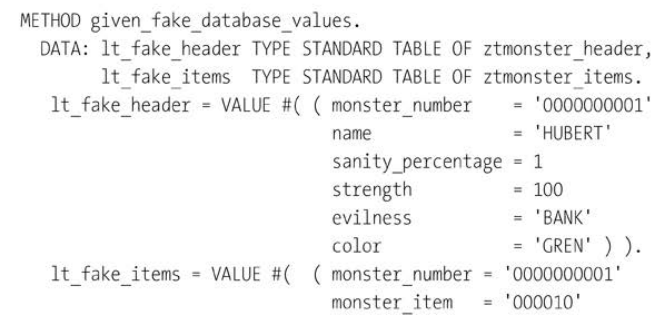
*mo\_environment->clear\_doubles( ) .*

*CREATE OBJECT mo\_cut .*

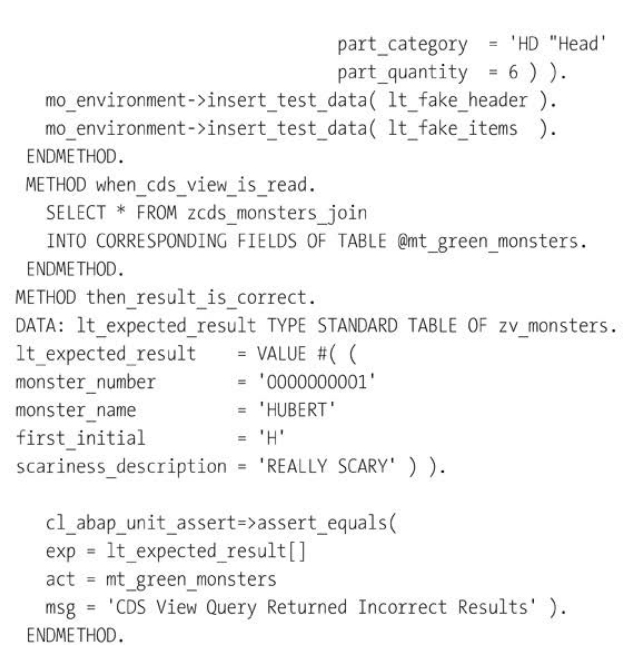
*ENDMETHOD.*

Listing 7.37 Setting Up and Tearing Down CDS Unit Test

Далее, в листинге 7.38, вы заполняете базовый источник(и) данных представления CDS поддельными данными. Когда производственный код запускается во время тестирования, считываются поддельные источники данных, а не из настоящей базы данных. Часть THEN метода тестирования проверяет, что результат является правильным, как обычно.



352



Listing 7.38 Unit Test for CDS View

Возможно, вы задаетесь вопросом, где именно вы пишете свои модульные тесты CDS. Ответ? В глобальном классе или программе у вас есть локальный класс тестов, встроенный в него, который содержит весь код модульных тестов. Определение представления CDS живет в объекте DDL, и у него не может быть встроенный локальный тестовый класс. Поэтому вы должны иметь отдельный класс, определенный как FOR TESTING, в котором вы кодируете модульные тесты представления CDS. Он может быть глобальным или может быть локальным классом живущий внутри глобального класса, который использует представление CDS. Я бы выбрал глобальный класс, потому что представление CDS может использоваться повсюду, и вы хотите, чтобы все тесты находились в одном центральном месте.

Важный момент появился в ABAP 1809, в котором вы можете добавить ABAP Doc комментарий - например, *!@testing ZCDS\_MONSTERS\_JOIN* - над определением тестового класса. Таким образом, тестовый класс знает, что он должен тестировать. Это кажется мне обратным. Я бы ожидал, что в определении представления CDS будет аннотация, которая говорит, что такое тестовый класс, но это, вероятно, было бы слишком логично. Если вы решите выполнять модульные тесты из определения представления CDS, тогда система

353

волшебным образом использует комментарий, чтобы определить, какой тестовый класс должен быть выполнен и затем запускает все его тесты.

Визуальный редактор (еще не выпущен)

Как вы уже убедились, писать множество аннотаций пользовательского интерфейса в представлении CDS не только очень скучно, но это занимает много времени, является просто шаблонным кодом (который мы всегда хотим избежать от необходимости писать вручную), и, что более важно, нарушает разделение проблем.

В предстоящем выпуске SAP (и я уже играл с прототипом) будет появится визуальный редактор WYSIWYG, подобный графическому редактору Dynpro, в котором вы перетаскиваете объекты по экрану и изменяете их свойства, а приложение-файл аннотаций, специфичный для конкретного приложения, генерируется автоматически.

На мой взгляд, это сделает представления CDS гораздо более пригодными для повторного использования, потому что вы потеряете тесную связь между уровнем базы данных и инструкциями пользовательского интерфейса.

## 7.4 Управляемые процедуры базы данных ABAP

До сих пор вы рассматривали продвижение кода с помощью ABAP SQL или представления CDS, оба из которых являются независимыми от базы данных. Теперь пришло время узнать о специфической для SAP HANA для выполнения действительно сложного кода внутри базы данных с помощью хранимой процедуры, но контролируемой (управляемой) с помощью ABAP - отсюда и название ABAP Managed Database Procedure (или AMDP, сокращенно).

Как вы знаете, когда вы создаете классы и методы класса, есть две вещи которые вам нужно закодировать: определение и реализация. Процесс создания AMDP точно такой же. Сначала вы рассмотрите, как определить класс в Eclipse, чтобы один или несколько его методов могут быть выполнены непосредственно в базе данных с помощью SAP HANA хранимой процедуры. Затем вы узнаете, как закодировать реализацию такого метода с помощью SQLScript, чтобы, хотя код написан в системе ABAP, он в действительности будет выполняться в базе данных SAP HANA во время выполнения. Наконец, вы узнаете как вызывать ADMP - либо из ABAP-программы, либо из CDS представления.

### 7.4.1 Определение AMDP в Eclipse

Если у вас есть класс, имеющий хотя бы один метод, в котором вы хотели бы делегировать обработку в базу данных, то единственное изменение, которое вам нужно внести в определение этого класса - включить интерфейс IF\_AMDP\_MARKER\_ HOB, как показано в листинге 7.39.

354

*CLASS zcl\_monsters DEFINITION.*

*PUBLIC SECTION.*

*INTERFACES IF\_AMDP\_MARKER\_HDB.*

*METHOD get\_complex\_monster\_data*

*IMPORTING VALUE( id\_monster\_number ) TYPE zde\_monster\_number*

*VALUE( id\_client ) TYPE sy-mandt*

*EXPORTING VALUE( et\_complex\_data ) TYPE g\_tt\_complex.*

Listing 7.39 Class Definition with Interface to Enable Use of AMDPs

Хотя представления CDS, рассмотренные в разделе 7.3, не зависят от базы данных, представления AMDP специфичны для SAP HANA. Вот почему \_HDB (для "базы данных HANA") появляется в конце в конце имени интерфейса в листинге 7.39. Другой момент, который может вызвать у вас недоумение в листинге 7.39 заключается в том, что вы должны сообщить базе данных SAP HANA о том, с каким клиентом вы говорите и, следовательно, метод должен получить клиента (MANDT) в качестве импортируюемого параметра. В мире ABAP вы привыкли к тому, что система автоматически считывает или записывает на клиента, на котором выполняется программа, но в мире баз данных клиент является жизненно важной частью информации. (Как упоминалось в разделе 7.3, при кодировании DDL для представления CDS, вам не нужно передавать клиента; этим занимается система времени выполнения).

Вы также заметили, что вам нужно передавать все по значению? Это обязательно, если вы хотите, чтобы метод использовал AMDP. Вы можете думать об этом как о вызове RFC ФМ. Параметры должны быть помечены как "по значению", потому что внешняя внешняя система не может удерживать ссылку на внутреннюю переменную SAP. База данных не является внешней системой, но она отделена от сервера приложений, на котором выполняется программа.

Кроме того, вы не можете иметь параметр RETURNING в классе AMDP до ABAP 1809 (мы называем эти функциональные методы в стране ASAP; остальной мир называет такие вещи скалярные функции), и даже тогда все параметры должны быть элементарными типами данных.

### 7.4.2 Реализация ADMP в Eclipse

Если вы просто сделали обычную реализацию метода класса, определенного, как показано в Листинг 7.39 с обычным кодом ABAP, то он будет запущен на сервере приложений как обычно. Однако это не то, что вам нужно, поэтому вы должны указать реализации метода, чтобы она вел себя по-другому.

Вы можете контролировать, выполняется ли метод на обычном сервере приложений ASAP или внутри базы данных, добавив несколько дополнительных строк в реализацию метода, как показано в листинге 7.40.

355

*CLASS zcl monsters IMPLEMENTATION.*

*METHOD get\_complex\_monster\_data*

*BY DATABASE PROCEDURE*

*FOR hdb*

*LANGUAGE SQLSCRIPT*

*USING zt\_monsters zt\_monster\_items.*

*" SQLScript code goes here*

*ENDMETHOD.*

*ENDCLASS.*

Listing 7.40 Implementing Method to Run inside Database

После реализации метода в листинге 7.40 вы увидите слова BY DATABASE PROCEDURE, что означает, что этот метод будет выполнять хранимую процедуру в базе данных во время выполнения. То, что управление находится в реализации, а не в определении, - это гениальный ход; это означает, что вы можете переопределить метод так, чтобы подклассы могут запускать метод на сервере или в базе данных, или во время модульных тестов тестовый двойник может возвращать жестко закодированные значения. В дополнение к очевидной пользе для модульного тестирования, если вы сомневаетесь, стоит ли переносить определенный метод в базу данных, то вы можете создать один класс, работающий на сервере, и подкласс, работающий в базе данных, который работает в базе данных, и выполнить несколько тестов, в которых вы меняете их местами, чтобы сравнить время выполнения.

Следующая строка в листинге 7.40 говорит FOR hdb, что означает, что вы хотите использовать базу данных SAP HANA базу данных, и LANGUAGE, чтобы сказать, какой язык вы хотите использовать. Как однажды сказал Генри Форд: "Вы можете иметь любой цвет, который хотите, пока он черный". Точно так же у вас имеете эти опции, чтобы указать, какую базу данных вы можете использовать (это должна быть SAP HANA) и на каком языке вы собираетесь кодировать AMDP (это должен быть SQLScript).

Наконец, в листинге 7.40 есть оператор USING, в котором вы должны дать явный список таблиц базы данных (или viev-s), которые вы собираетесь использовать в рамках процедуры. Также возможно, чтобы одна процедура базы данных вызывала другую, и если вы хотите, чтобы так было, то вы должны упомянуть процедуру базы данных, которую вы собираетесь вызвать в этом списке. Вся эта концепция покажется странной для программистов ABAP программистов, которые привыкли просто читать из любой таблицы базы данных, которая им нравится и вызывать глобальные функции и методы по своему усмотрению, но помните: вы находитесь в другом мире, мире базы данных, где законы физики другие, и вода течет вверх по склону. Это программирование, но не такое, каким вы его знаете.

356

Ограничения

Вы не можете создать код AMDP (или любой другой элемент SAP HANA, если на то пошло) в SE24 или SE80. Вы можете отобразить его, но для создания или изменения кода вам необходимо использовать ABAP в Eclipse. (Надеюсь, глава 1 убедила вас в том, что ABAP в Eclipse – лучший инструмент для использования в любом случае.) Еще одно ограничение заключается в том, что в методе, который должен быть реализован как AMDP, не может быть возвращаемых параметров, и все параметры должны передаваться как значения.

Аспект ABAP в Eclipse вряд ли когда-либо изменится, но я не был бы шокирован, если другие ограничения будут сняты в какой-то момент в будущем. Например, до выхода ABAP 7.4 SP 8, вы не могли иметь CHANGING параметры в методе, который должен был быть реализован как AMDP; теперь вы можете. Точно так же, до SP 8, любой вид исключение вызывало короткий дамп; после SP 8 исключение можно перехватить с помощью CATCH ex\_admp\_error. Кроме того, в ABAP 7.5 вы можете иметь возвращаемый параметр в виде таблицы с помощью функции таблицы CDS, которая рассматривается далее в этом разделе.

Теперь вы переходите к части кодирования на SQLScript - именно здесь некоторые профессионалы ABAP начнут бежать к двери, потому что вам придется начать кодировать на отличном от ABAP языке.

Может показаться, что после долгого времени кодирования только на ABAP, вы получаете больше новых языков, которые бросают в вас одновременно, чем вы можете потрясти палкой: DDL и DCL для представления CDS, теперь SQLScript для ADMP, а позже в главе 12 вы увидите, как JavaScript приветливо помашет вам рукой.

Немного предыстории: каждый поставщик баз данных (например, Oracle или Microsoft) написал свой собственный язык для написания для написания процедур базы данных, все из которых являются расширения ANSI-стандартного SQL. Поскольку каждая компания считает, что она знает все лучше всех, каждый такой язык процедур баз данных немного отличается. SAP HANA теперь присоединилась к этому клубу со своим языком под названием SQLScript, который, конечно, немного отличается от других.

Хотя язык (DDL) внутри представления CDS не так уж сильно отличается от того, к чему вы привыкли для ABAP SQL SELECT, SQLScript имеет гораздо больше возможностей.

Наиболее очевидным преимуществом SQLScript перед DDL в представлениях CDS является то, что вы можете иметь условную логику, такую как IF/THEN/ELSE. Кроме того, SQL-запрос ABAP или представление CDS возвращает только один результат, в то время как запрос, который вы собираетесь кодировать с помощью ADMP, может возвращать столько результатов, сколько вы хотите, поэтому определение метода ABAP может иметь множество параметров EXPORTING.

357

Наконец, и это самое главное, поскольку этот код работает внутри базы данных SAP HANA, он может в полной мере использовать все возможности, которые предоставляет такая база данных. В качестве примера, если вы вызовете в своих ABAP-программах пятьдесят четыре не связанных между собой SQL-запроса ABAP, они обратятся к базе данных и будут выполняться по одному. Если есть пятьдесят четыре таких запроса в AMDP, они будут выполняться все сразу, параллельно. Если бы вы прочитали стандартную документацию SAP о том, почему процедуры внутри базы данных SAP HANA выполняются намного быстрее, то вы были бы сбиты с толку такими фразами, как быстрые операции с колонками, оптимизация запросов и jungle is massive, которые сводятся к тому, что база данных (как и человек) может лучше реагировать на инструкции на своем родном языке, а не на инструкции через интерпретатор.

Примечание

В качестве примера того, как быстро развиваются события в мире SAP HANA, несколько лет назад позиция SAP заключалась в том, что для получения данных в SQLScript следует использовать то, что называется CE-функциями данных в SQLScript, когда это возможно. Теперь же инструкция гласит, что не следует трогать такие функции с десятифутовым шестом.

Такой быстрый темп изменений может доставить много хлопот при просмотре документации о SQLScript в Интернете. На конференции SAP TechEd 2015 мне показалось довольно забавным, что одна из ведущих знаменитостей SAP заявила, что написание кода на SQLScript - это совсем не весело и что скопировать существующий образец и изменить его гораздо проще. По моему опыту, код, скопированный из блога, скорее всего, не будет компилироваться, а SQLScript, мягко говоря, требователен.

Тем не менее, с ABAP когда-то было то же самое. Синтаксис SQLScript обязательно успокоится через некоторое время, и вряд ли он исчезнет в ближайшее время.

Приступая к делу (кодированию), SQL-запрос в SQLScript не так уж сильно радикально отличается от своего эквивалента в ABAP; у вас просто больше возможностей для выбора в отношении оператора SELECT, а синтаксис (знаки препинания) немного меняется. Подобный измененный синтаксис вы видели ранее, когда мы говорили о представлениях CDS. Изменение здесь заключается в том, что представления CDS управляют клиентом за вас; в AMDP вы должны указать клиента.

В листинге 7.41 показан код SQLScript для выполнения SELECT из базы данных в рамках реализации метода AMDP.

METHOD get\_people\_scared\_today BY DATABASE PROCEDURE

FOR HDB

LANGUAGE SQLScript

358

OPTIONS READ-ONLY

USING zt\_monsters zt\_scary\_event\_header zt\_scary\_event\_items.

-- declare a local variable

declare ld\_today date;

-- get current date

select current date into ld\_today from dummy;

-- select relevant scary events

scary\_events\_table =

select

event\_items.mandt as client ,

event\_header.event number as event\_number,

monster.monster number as monster\_number,

event\_header.created\_at as scaring\_date,

event\_items.people\_scared,

from zt\_scary\_event\_items as event\_items

join zt\_scary\_event\_header as event\_header

on

event\_items.mandt = event\_header.mandt

and

event\_items.event number = event\_header.event number

join zt\_monsters as monster

on

event\_header.client = monster.client

and

event\_header.monster\_number = monster.monster number

where

event\_header.mandt = :id mandt

and monster.number = :id monster number

and event\_header.scare\_date = :id\_today

and event\_header.scary\_status = 'REALLY\_SCARY';

Listing 7.41 Coding Database SELECT inside AMDP

Большая часть кода в листинге 7.41 - это обычная работа, за исключением следующего:

- Синтаксис объявления локальной переменной для сегодняшней даты не совсем такой, как вы привыкли, например, вам не нужно слово TYPE. Однако такие незначительные синтаксические изменения являются неотъемлемой частью изучения кода на новом языке.

- В предложении WHERE перед переменными нужно ставить двоеточие.

- Вы заметите, что локальная таблица для хранения страшных событий объявляется в процессе создания, как и конструкции DATA в ABAP 7.4.

359

- Вы сейчас находитесь в базе данных. Поэтому "внутренняя таблица" SCARY\_EVENTS\_TABLE на самом деле не является локальной внутренней таблицей, а скорее временная таблица базы данных, которая существует в течение время выполнения процедуры. Таким образом, позже в хранимой процедуре вы можете выполнить SELECT над этой таблицей, как и над любой другой таблицей базы данных.

И последнее замечание: даже если вы не кодируете на ABAP внутри AMDP, проверка синтаксиса будет знать, что язык внутри метода - SQLScript, и адаптирует его соответствующим образом (например, вы получите ошибку, если в вашем SQLScript есть синтаксические ошибки).

### 7.4.3 Вызов AMDP из ABAP-программы

Это самый простой момент! Когда вы вызываете определенный метод, ваша ABAP-программа не будет ни знать, ни заботиться о том, где выполняется фактический код (сервер приложений или база данных) ни знать, ни заботиться о том, где выполняется фактический код (сервер приложений или база данных).

Вы даже можете изменить существующую реализацию метода класса z, чтобы перейти от обычной реализации ABAP в хранимую процедуру (AMDP), и вам не придется менять какие-либо программы, вызывающие этот метод; это прекрасный пример принципа "открыто-закрыто" принцип, при котором вы изменяете поведение программы (в данном случае сильно), не изменяя фактическую программу ни в коей мере.

В листинге 7.42 нет никакого способа узнать, что вы вызываете AMDP - и это хорошо.

lo\_monsters->get\_complex\_monster\_data(

EXPORTING id monster number = monster number

IMPORTING et\_complex\_data = DATA(complex\_monster\_table) ) .

Listing 7.42 Calling AMDP from ABAP Program

Более того. начиная с ABAP 1809, вы можете иметь функциональные методы, которые являются АМДП и передавать их в качестве импортирующих параметров другому методу, например.

### 7.4.4 Вызов AMDP изнутри представления CDS

В ABAP 7.4 представление CDS еще не взяло на себя роль всеобъемлющей и конечной цели всех вещей во вселенной SAP, поэтому существовало четкое разделение между программой ABAP вызывающей представление CDS (независимое от базы данных) и AMDP (только для SAP HANA). Теперь CDS представление может само вызывать AMDP, хотя это, конечно, аннулирует его независимость от базы данных, если программист решит так поступить.

Для этого необходимо выполнить три следующих действия:

360

1. Задайте определение DDL, которое будет генерировать само представление CDS. Это происходит в первую очередь потому что позже, когда вы создадите определение метода в ABAP, он унаследует различные атрибуты от представления CDS, поэтому представление CDS должно существовать первым. Процедура для создания DDL, определяющего представление CDS, точно такая же, как описана в разделе 7.3.1, только в этот раз вы создадите представление CDS ZCDS\_MONSTERS\_TABLE \_FUNCTION и выберите шаблон Define Table Function with Parameters.

Как и раньше, для вас будет заполнена целая куча кода, в котором будут проставлены значения, которые вы для замены на фактические значения. Поскольку вы ввели имя и описание в всплывающем окне перед выбором шаблона, первые два заполнителя заполняются для вас: текстовое описание view и строка, которая гласит define table function Zcds\_Monsters\_Table\_Function.

Далее следует список параметров импорта. Мы начинаем со специальной аннотации, за которой следует параметр "клиент":

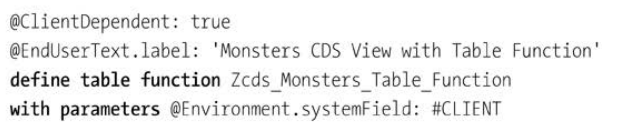
*@Environment.systemField : #CLIENT*

*clnt:abap.clnt*

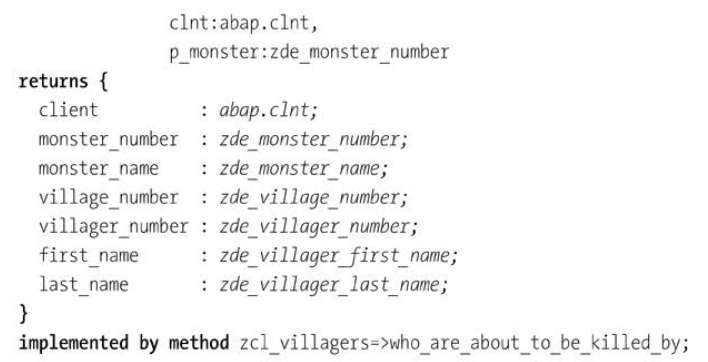
Как упоминалось ранее, SQLScript нуждается в передаче клиента, чего мы действительно не хотим делать это из ABAP-программы, поэтому предыдущий фрагмент гарантирует, что значение SY-MANDT будет передано автоматически без вызывающей программе ничего не нужно делать. Помимо клиента, единственным параметром, который вы хотите добавить сюда, это номер монстра, указанный в формате *parameter : data element*, точно так же, как когда вы кодировали представление CDS с параметрами в разделе 7.3.2.

Добавьте список возвращаемых параметров. Первый из них должен быть клиентом, поэтому он заполнен за вас. Затем добавьте данные, которые вас на самом деле интересуют, в том же формате, что и параметры: имя поля, затем элемент данных, разделенные двоеточием и завершается точкой с запятой в конце строки. Список полей образует структуру, а таблица, возвращаемая из этого представления CDS, состоит из одной или нескольких строк, типизированных в соответствии с определением этой структуры.

В последней строке DDL указывается класс и (статический) метод, которые будут использоваться для хранения SQLScript-кода AMDP. Этот класс и метод еще даже не были еще не созданы, но определение DDL, похоже, не возражает. Полученный код DDL выглядит как в листинге 7.43.



361



Listing 7.43 Coding CDS View Table Function

2. Активируйте источник DDL, а затем, так же как вы делали в разделе 7.4.1, создайте класс в ABAP-коде с интерфейсом IF\_AMDP\_MARKER\_HDB. Затем создайте реализацию, которая выглядит как в листинге 7.44.

CLASS zcl\_villagers DEFINITION

PUBLIC

CREATE PUBLIC .

PUBLIC SECTION.

INTERFACES if\_amdp\_marker\_hdb.

CLASS-METHODS who\_are\_about\_to\_be\_killed\_by

FOR TABLE FUNCTION zcds\_monsters\_table\_function .

ENDCLASS .

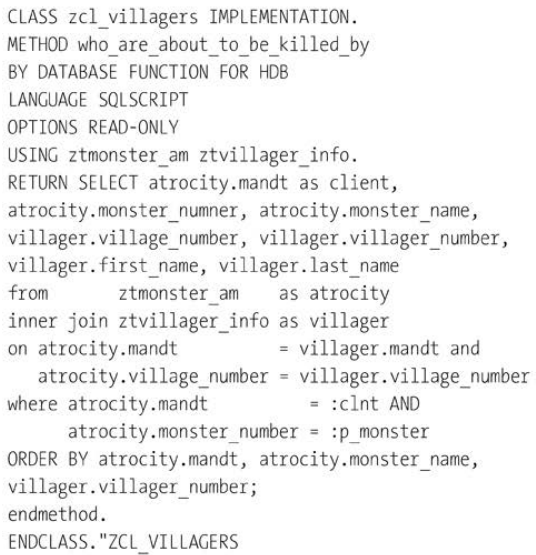
Listing 7.44 Defining CDS Table Function in ABAP Class

Это должен быть статический метод. Компилятор проверяет, чтобы убедиться, что таблица CDS действительно существует. Нет необходимости объявлять какую-либо сигнатуру в реализации; она неявно выводится из табличной функции. Это означает, что никто, глядя на определение в ABAP в Eclipse или SE24, не догадается о том, что это за сигнатура. Возможно, вы захотите добавить сигнатуру в качестве комментария.

3. Последняя часть - написание реализации метода на SQLScript. Код должен начинаться со всей той дребедени, которую вы видели ранее, например, с указания языка и базы данных. Это было бы небольшим шагом назад, если бы SAP разрешил ORACLE в качестве допустимой здесь, но SAP держит свои возможности открытыми. В прошлый раз вы использовали BY DATABASE PROCEDURE; в этот раз фраза будет BY DATABASE FUNCTION, и обязательно нужно указать OPTIONS-READ\_ONLY.

362

Затем следует слово USING и список всех таблиц ABAP, которые вы будете читать; Это предназначено для того, чтобы помочь, если что-то изменится, потому что система ABAP имеет немного слепое пятно, в котором живет код, написанный на других языках. После этого остается только кодирование вашего SQLScript. Готовый продукт выглядит как в листинге 7.45.



Listing 7.45 SQLScript Coding for Table Function

Этот пример глуп - не потому, что он касается монстров, которые являются смертельно серьезными, но потому, что он не делает ничего специфичного для SAP HANA. Это просто обычное соединение баз данных. В реальной жизни вы бы использовали SQLScript только тогда, когда вам нужны всевозможные причудливые функции, специфичные для базы данных SAP HANA, такие как логика управления внутри базы данных и т.д.

Теперь вы можете вызывать представление CDS с помощью оператора SELECT в вашей ABAP-программе на представление. Вы получите предупреждение о синтаксисе, сообщающее, что этот оператор будет работать только в базе данных SAP HANA. Это предупреждение может быть подавлено с помощью прагмы *##db\_feature\_mode [amdp\_table\_function)*, как показано в листинге 7.46.

PARAMETERS: p\_monst TYPE zde\_monster\_number.

SELECT \*

363

FROM zcds\_monsters\_table\_function( p\_monster = @p\_monst )

INTO TABLE @DATA(doomed\_villager\_table)

##db\_feature\_mode[amdp\_table\_function].

Listing 7.46 Reading Data from CDS View with Table Function

Подводя итог, можно сказать, что мы сделали здесь "усиление" представления CDS за счет того, что оно обернуто функцию AMDP, которая может выполнять гораздо более широкий набор операций внутри базы данных.

## 7.5 Нахождение и перемещение кода вниз

Теперь вы знаете, как продвигать код вниз в базу данных, но как вам решить, какой код заслуживает этой участи и какая техника для этого подходит? Не бойтесь! Этот раздел объясняет, как найти пользовательский код, который нужно опустить вниз, и как выяснить, какие из техник, о которых вы читали, следует использовать для реализации вытеснения кода. В конце этого раздела будет приведен пример, показывающий случай, в котором в котором код должен быть вытеснен вниз.

### 7.5.1 Поиск пользовательского кода, который необходимо вытеснить вниз

К счастью для вас, компания SAP понимает, что вам может понадобиться помощь в поиске кода, который должен быть и предоставила инструментальную поддержку именно для этой цели. Эти инструменты позволяют провести анализ среды SAP в вашей организации и выдать список какие разделы вашего пользовательского кода необходимо просмотреть с целью продвижения вниз. Для проведения такого анализа используются три операции, работающие в тандеме:

1. ABAP Test Cockpit (SATC)

2. SQL Monitor (SQLM)

3. Рабочий список настройки производительности SQL (SWLT)

ABAP Test Cockpit выполняет статическую проверку вашего пользовательского кода в системе разработки системе.

SQL Monitor - это транзакция для анализа вызовов базы данных в производстве; результаты выглядят так же, как и в ST05. Основные различия заключаются в следующем:

- ST05 предназначен для одноразового анализа одной программы; SQLM предназначен для анализа всех данных в системе и хранения этих данных в течение длительного времени (например, все, что происходило в системе в течение года).

- Эта информация может быть получена из системы разработки через RFC.

364

Последний член этой тройки - еще одна новая транзакция. Настройка производительности SQL. Рабочий список (рисунок 7.9), который получает производственные данные из SQLM через RFC. Затем, в сочетании со статическим анализом ABAP-кода через SATC, SWLT предоставляет вам ранжированный список работ, какие процедуры могут быть улучшены в целом, и, если вы рассматриваете возможность миграции базы данных, какой код может быть перенесен в базу данных SAP HANA.

В двух словах, SATC + SQLM 🡪 SWLT.

То, как именно использовать SWLT для поиска кода, который может быть улучшен путем переноса в базу данных, можно было бы написать целую книгу. К счастью, на эту тему написано так много, что вам не придется далеко ходить на охоту (см. в конце этой главы для начала).

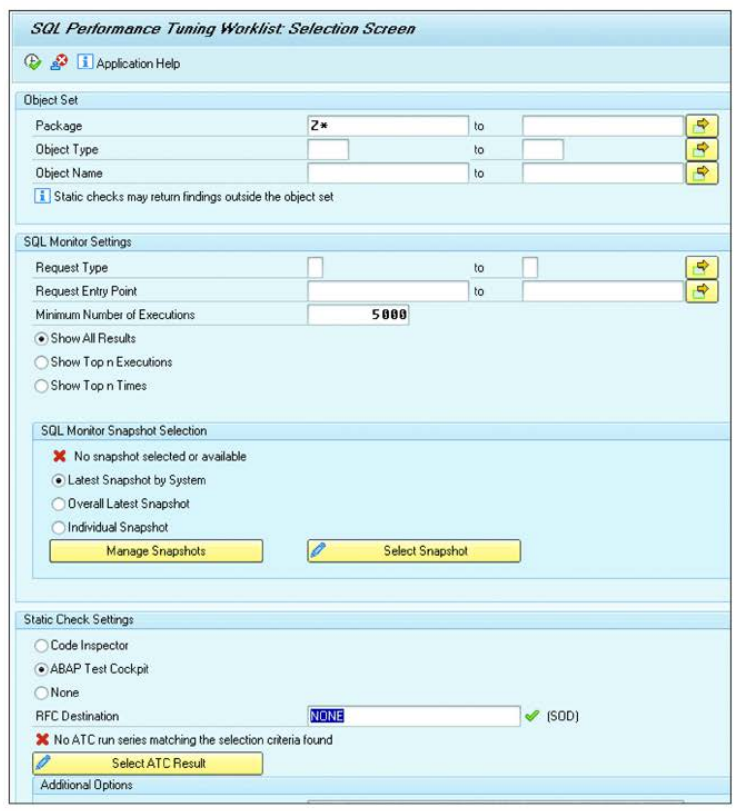


Figure 7.9 Transaction SWLT

365

Доступность SATC/SQLM/SWLT

ABAP Test Cockpit был доступен в SAP Netweaver 7.02 SP 12 или SP 5 SAP NetWeaver 7.31 и входит в стандартную комплектацию SAP NetWeaver 7.4.

SQL Monitor доступен в SAP Netweaver 7.02 SP 14 или SAP NetWeaver 7.31 SP 9 и входит в стандартную комплектацию SAP NetWeaver 7.4. Вам также необходимо, чтобы плагин ST\_Pl был на уровне 2008, стек поддержки 8.

SQL Performance Tuning Worklist поставляется с SAP NetWeaver 7.02 SP 14, SAP NetWeaver 7.31 SP 9 или SAP NetWeaver 7.4 SP 2.

Все эти инструменты являются стандартными для ABAP 7.5.

### 7.5.2 Какую технику использовать для перемещения кода вниз

После того, как вы составили список того, какой пользовательский код может выиграть от переноса в базу данных, вам нужно решить, как именно этого добиться для каждого такого блока кода. На самом деле существует три метода переноса кода в базу данных. Один из них это ABAP SQL (который вы всегда использовали для передачи логики в базу данных, хотя вы, возможно, и не думали, что делаете это); два других метода - это представления базы данных (CDS views) и хранимые процедуры (AMDP).

На этом этапе легко запутаться в том, какую технику использовать. В этом разделе вы рассмотрите различные критерии, чтобы принять решение, а в следующем разделе вы увидите рабочий пример принятия такого решения.

**Функциональность**

Функциональность - это, конечно, самый важный критерий: ваш код предназначен для решения проблему, и вы пытаетесь решить, какой инструмент может "протолкнуть" вашу логику в базу данных.

Можно подумать, что представление CDS обладает большей функциональностью, чем SQL-запрос ABAP, но на самом деле SAP стремится к функциональному паритету между ABAP SQL и CDS представлениями, хотя в любой момент времени набор функций между ними не будет идентичным. Вполне возможно, что в вашем релизе вы можете сделать что-то в ABAP SQL, чего вы не можете сделать в CDS или наоборот, и если это так, и дополнительная функциональность (которая у одного есть, а у другого нет) решает поставленную проблему, то выбор становится очевидным.

366

Обычно, однако. эти два метода настолько похожи, что трудно сделать выбор между ними, чтобы решить, как решить ту или иную проблему. Если ни одна из них не может сделать то, что вы хотите и у вас есть база данных SAP HANA, то пришло время посмотреть на AMDP, либо используемую напрямую, либо обернутую в представление CDS.

**Скорость**

Какую бы технику вы ни выбрали, код будет выполняться быстрее, чем аналогичный код на ABAP, потому что работа будет выполняться быстрее, поскольку работа выполняется в базе данных.

Люди часто думают, что представление CDS каким-то образом "быстрее", чем эквивалентный SQL-запрос в ABAP. Если задуматься, это не может быть так, потому что в конце представление CDS отправит в базу данных точно такой же SQL-запрос, как и тот, который генерируется из SQL-запроса ABAP.

ADMP может быть намного быстрее, чем любой из вышеперечисленных вариантов, потому что он может выполнять запросы параллельно и SQLScript оптимизирован для базы данных SAP HANA.

**Интеграция**

Представления CDS являются одним из трех столпов модели программирования RESTful ABAP и таким образом, они являются основой стратегии развития SAP. Как таковая, существует очень сильная интеграция с другими технологиями SAP, такими как BOPF и SAPUI5/SAP Gateway, посредством использования аннотаций.

Поэтому в некоторых ситуациях использование представлений CDS является обязательным для того, чтобы раскрытия базы данных для бизнес-логики и пользовательского интерфейса. Просто потому, что это радикально сокращает общее время разработки приложения.

Более того, если смотреть на средне- и долгосрочное будущее, когда появится ABAP в облаке, похоже, что единственным способом доступа к базе данных в этой среде будет CDS, открытое через сервис OData.

**Семантика**

Преимущество представлений CDS по сравнению с SQL-операторами ABAP заключается исключительно в семантики - то есть, в том, что они ближе к концептуальному мышлению. Это означает, что в представлении CDS вы используете длинные английские имена для полей, а не немецкие сокращения, и "ассоциации", а не соединения.

Идея заключается в том, что чем проще что-то понять, тем проще это поддерживать, а это немаловажное преимущество.

367

**Возможность повторного использования**

Одним из аргументов в пользу использования представлений CDS вместо SQL-операторов ABAP является то, что они каким-то образом "многоразовые". Это также был аргумент в пользу представлений и в SE11, но я не уверен в этом, по двум причинам.

Во-первых, в хорошо спроектированных ОО приложениях вы, как правило, имеете центральный класс, контролирующий доступ к любым данным, которые извлекает запрос. Иметь точно такой же SQL-запрос в сотнях мест никогда не имело особого смысла, как и наличие одного и того же представления CDS в сотне мест.

Во-вторых, и это чисто субъективное мнение. SAP очень гордится тем, сколько различных представлений CDS, которые она создала. На конференциях она хвасталась их количеством, показывая диаграмму, которая выглядела как метеоритный дождь в ночном небе, где каждая точка представляла собой представление CDS в стандартной системе SAP. У меня возникает следующий вопрос: Если они настолько многоразовые, то зачем SAP понадобилось создавать десятки тысяч таких точек? Я предполагаю, что все эти десятки тысяч созданы для индивидуальных случаев использования, что является тем, к чему мы привыкли – к SQL-запросам ABAP.

**Открытость**

Некоторое время идея - в соответствии с рекомендациями SAP - заключалась в том, что AMDP должно быть последним средством. Прежде чем идти по этому пути, вы должны сыграть в мысленную игру, в которой вы притворились, что вариант AMDP не существует. Причина этого в том, что, хотя AMDP имеет гораздо больше возможностей, SQL-запрос ABAP или представление CDS является открытым; то есть, он работает с любой базой данных. Это было одной из сильных сторон SAP на протяжении многих лет: независимо от того, какую базу данных вы решили использовать в SAP, ваш ABAP-код все равно будет работать, и вы могли даже менять базы данных без каких-либо негативных последствий. Такого рода преимущества слишком хороши, чтобы легкомысленно от них отказываться, поэтому SAP в прошлом призывала вас оставаться открытыми, когда это возможно, используя специфические возможности SAP HANA только в крайнем случае.

Как вы уже неоднократно слышали, с 2025 года единственной поддерживаемой ERP-системой будет SAP S/4HANA - и, как следует из названия, она работает только с базой данных SAP HANA, так что "открытость" уходит в прошлое.

Таким образом, пока вы не перейдете на SAP S/4HANA, вы все равно не сможете использовать AMDP, поэтому вам придется оставаться открытыми; после перехода вы больше не сможете обмениваться базами данных, поэтому оставаться открытым бессмысленно.

Логический вывод заключается в том, что как только вы окажетесь в положении, позволяющем воспользоваться преимуществами AMDP для решения конкретной проблемы, которая может быть решена с помощью AMDP, но которая не может быть решена с помощью SQL-оператора ABAP или представлением CDS, нет ни одной причины в мире, почему бы вам не сделать этого.

368

### 7.5.3 Пример

Лучший способ показать пример поиска кода, который можно сдвинуть вниз, - это взять пример из реальной жизни (изменено незначительно, чтобы защитить невинных), с которым я столкнулся во время рецензирования работы другого программиста на днях. Ничего не было плохого в коде, который он написал, но я подумал: "Это работа для SAP HANA". В этом примере мы расскажем вам об этом коде и объясним, почему он является хорошим кандидатом на внедрение.

Как любой хороший бизнесмен, барон Франкенштейн ожидает, что ему заплатят за монстров, которых он делает для своих клиентов, и он выставляет счета, в которых говорится, что оплата должна быть произведена через тридцать дней после окончания месяца, на который приходится дата выставления счета. Однако некоторые клиенты разоряются и не могут оплатить свои счета; с точки зрения SAP, открытая статья клиента должна быть классифицирована как безнадежный долг. Бухгалтер (например, Игорь) должен очистить открытую статью путем списания долга с баланса и отнесения его на счет прибылей и убытков в разделе "Прибыли и убытки".

Чтобы усугубить ситуацию для клиента (как будто банкротство - это не так уж плохо), барон нажимает кнопку на своей панели управления, и монстр, доставленный клиенту, впадает в неистовство, уничтожая все на своем пути. В результате у барона удивительно низкий коэффициент безнадежных долгов.

Тем не менее, клиенты, которые потеряли все, не очень хорошо выплачивают свои долги, поэтому в SAP-системе барона остается несколько очищенных открытых позиций, связанных с такими безнадежными долгами, связанных с такими безнадежными долгами. К счастью для барона, правительство Трансильвании признало эту проблему. Если барон отправляет электронную выписку о таких безнадежных долгах в электронном виде, то он сможет получить налоговые льготы на них.

Теперь представьте, что есть программа, которая просматривает очищенные открытые статьи, относящиеся к безнадежным долгов; она запускается каждый месяц и отправляет эти данные правительству в формате, который оно требует. Самое главное, естественно, - не заявлять об одном и том же безнадежном долге дважды, что было бы мошенничеством - гораздо хуже, чем если бы ваши монстры уничтожили имущество ваших клиентов.

Программа определяет, какие клиенты являются клиентами с плохими долгами (т.е. те, кто не оплатили свои счета), перебирает каждого такого клиента и считывает его очищенные открытые позиции. После того как эти данные получены из базы данных, программа перебирает каждую запись и применяет некоторую бизнес-логику, чтобы определить, должна ли эта запись быть отправлена правительству.

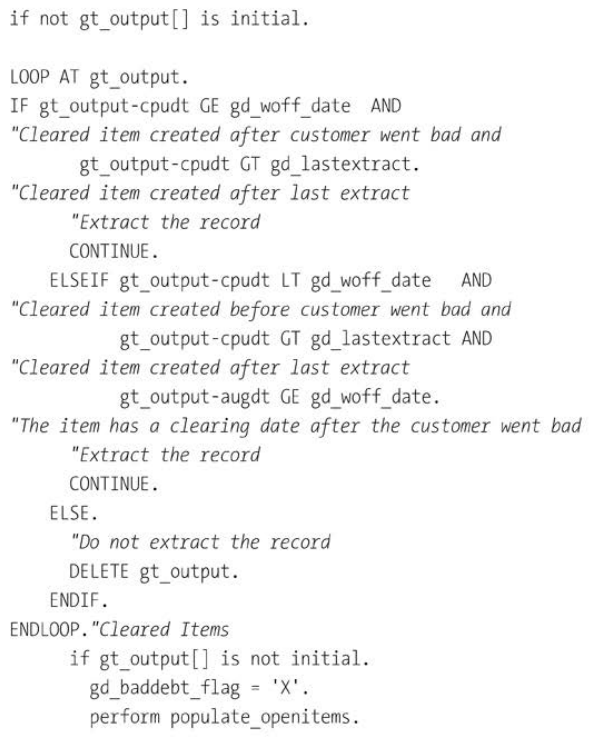
Существует переменная под названием GD\_WOFF\_DATE, которая хранит дату, на которую клиент пришел к ужасному концу. Поскольку это правительство, с которым имеет дело барон,

369

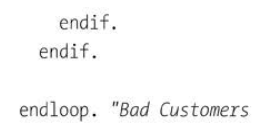
правила довольно сложны: Любые очищенные товары с датой создания (CPUDT), равной или более поздней, чем дата дефолта клиента (GD\_WOFF\_DATE), могут быть заявлены как налоговое списания. Для открытых позиций с датой создания раньше даты списания, только могут быть заявлены только те позиции, у которых дата очистки (AUGDT) больше или равна дате списания.

При первом запуске программы все такие позиции отправляются в правительство, а переменная GD\_LASTEXTRACT обновляется до даты, когда эти товары были отправлены правительству, чтобы они не были отправлены снова. Но что, если позже будут добавлены новые пункты? Их тоже нужно отправить. Поэтому программа считывает дату ввода очищенного товара (CPUDT), и если дата поступления позже, чем последняя выписка, то новая запись должна быть отправлена в правительство, а дата последней выписки обновлена соответствующим образом.

Код ABAP для выполнения этой логики на записях, извлеченных из базы данных, показан в листинге 7.47.



370

- непонятно, но так в книге!

Listing 7.47 Removing Bad Debt Records from Table on Application Server

В листинге 7.47 вы заметите, что каждая запись, которая может быть отправлена в правительство, извлекается из базы данных, а затем выполняется некоторая бизнес-логика, формируется для удаления тех, которые не нужны. Это нарушает одно из золотых правил SQL, а именно, минимизировать объем данных, отправляемых из базы данных на сервер приложений. Получить ненужные записи - все равно что заказать десять слонов, когда вам нужен только один, а затем, когда слоны прибывают, договариваться о том, чтобы девять из них были отправить обратно в Африку.

Узким местом (даже в базе данных SAP HANA) является передача таких данных, так что идея заключается в том, чтобы удалить ненужные записи до того, как они будут отправлены из базы данных в запрашивающую ABAP-программу. Почему же вы постоянно видите код, подобный листингу 7.47? Потому что до сих пор не было никакого выбора в этом вопросе. Единственный способ выполнить бизнес-требование, подобное приведенному в примере, - это получить каждую запись из базы данных, исследовать эти записи по одной за раз и игнорировать те, которые вам не нужны. Однако технологии шагнули вперед, и теперь у вас есть и другие варианты в вашем распоряжении.

Давайте рассмотрим, как вы можете решить эту проблему другими способами, рассмотрев ваши возможности в следующем порядке: ABAP SQL, представления CDS и методы AMDP.

**ABAP SQL**

Как вы видели ранее в этой главе, варианты кодирования состояния SELECT в ABAP SQL в традиционной ABAP-программе значительно увеличились и продолжают увеличиваться с каждым стеком поддержки. Например, вы можете встроить CASE-операторы в запрос SELECT, но не (на момент написания статьи) сложное состояние IF как в примере в листинге 7.47.

Правильный способ действовать - сначала притвориться, что расширенный синтаксис ABAP SQL – это единственная возможность, которая у вас есть. Затем вы должны подумать: можно ли ограничить число возвращаемых записей с помощью оператора CASE или одной из других новых возможностей? Если да, то действуйте; как отмечалось ранее, ABAP SQL всегда является предпочтительным выбором. Однако, в данном случае, похоже, бизнес-логика слишком сложна. Возможно, вы сможете использовать оператор CASE, чтобы отметить плохие записи, установив дату очистки на 12.31.9999,

371

но вам все равно придется отправлять эти записи обратно на сервер приложений для удаления - что скорее противоречит цели упражнения, которая заключается в удалении этих записей во время обработки базы данных. Пора приступить к изучению других вариантов.

**Представления CDS**

Как упоминалось ранее, представление CDS похоже на оператор SELECT на стероидах. DDL-артефакт, который находится в репозитории ABAP и создается через ABAP в Eclipse, представляет собой небольшой файл, состоящий из кода для одного сложного оператора SELECT.

Теперь посмотрите, что возможно в коде DDL, который будет генерировать представление CDS. Можете ли вы здесь использовать операторы IF? Как выясняется, хотя доступные вам варианты при использовании DDL для создания представления CDS значительно расширились, операторы IF пока не входят в них; вы по-прежнему ограничены оператором CASE.

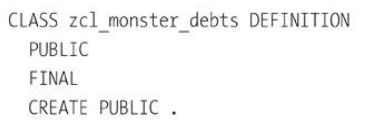
Просто для ясности: если CDS-view может обрабатывать нужную вам логику, то это то, что надо, но в данном примере мы должны обратиться к AMDP.

**Методы AMDP**

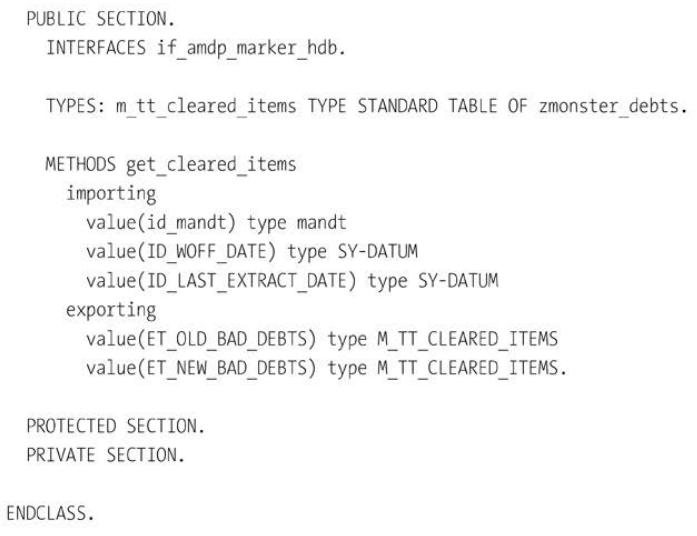
Как вы теперь знаете, AMDP - это метод, который выполняется в базе данных, а не на сервере приложений, как традиционный метод. К настоящему времени вы также поняли, что язык, используемый в таких методах AMDP, - SQLScript (отличен от ABAP), что заставляет некоторых людей дрожать в своих сапогах.

В этом примере вы знаете, чего хотите: удалить определенные записи до того, как набор результатов отправляется обратно на сервер приложений. Разве не было бы здорово просто иметь точно такой же ABAP-код, как и раньше, но на этот раз работающий в базе данных? Извините. Это не на самом деле так не работает.

Вы можете иметь операторы IF внутри AMDP для установки локальных переменных и тому подобное, но это не сильно поможет в данной ситуации. Самое простое решение – создать метод (определение которого показано в листинге 7.48) с двумя возвращаемыми параметрами: таблицей очищенных предметов, созданных до даты списания (которые соответствуют правилам для таких предметов) и таблица очищенных предметов, созданных после даты списания.

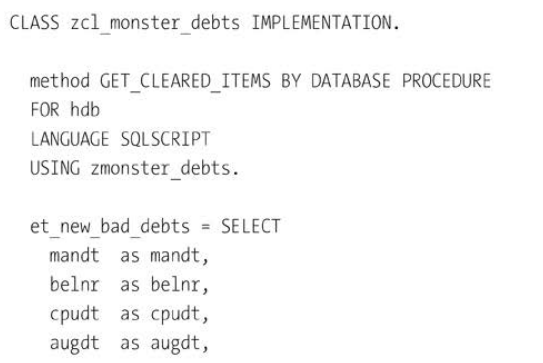


372

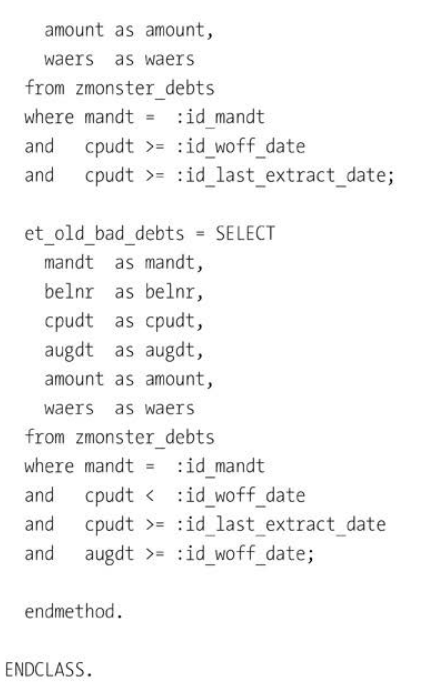


Listing 7.48 AMDP Method Definition to Get Monster Cleared Items

Внутри реализации AMDP (см. листинг 7.49), вам нужно только два чтения базы данных, чтобы заполнить две таблицы экспорта. Естественно, вы могли бы использовать эквивалентные операторы SELECT в методе, который выполняется на сервере приложений, но это подразумевает обращение к базе данных дважды, а это означает дважды обращаться к базе данных, а это иногда может занять больше времени, чем просто обращение один раз, чтобы получить кучу записей и затем отбросить некоторые. В этом случае вся работа выполняется в базе данных, и только нужные записи отправляются обратно на сервер приложений.



373



Listing 7.49 AMDP Method Implementation to Get Monster Cleared Items

Если вы запустите трассировку ST05 SQL для этого метода, когда данные были получены из базы данных, вы увидите, что не было операции FETCH, как это обычно бывает при выполнении оператора SELECT в традиционной базе данных. Вместо этого результат выглядит как на рисунке 7.10, где показан вызов AMDP внутри базы данных.

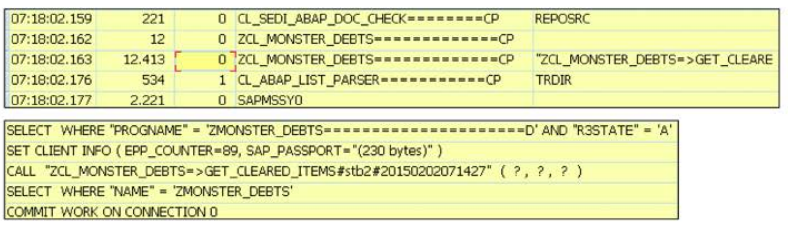


Figure 7.10 SQL Trace of AMDP

374

Эти два запроса к базе данных не связаны между собой и поэтому будут выполняться параллельно, а не последовательно, возвращая результат за гораздо меньшее время.

Это был упрощенный пример, но, надеюсь, вы поняли идею: всякий раз, когда вы обнаружите, что единственный способ получить то, что вы хотите, из базы данных с помощью традиционных методов, таких как ABAP SQL, не может работать без нарушения золотого правила (например, получить обратно больше данных, чем вам на самом деле нужно), то самое время подумать о переносе кода в базу данных, используя рекомендации этого раздела (т.е. используйте расширенный синтаксис ABAP SQL, если это возможно; если нет, то используйте представление CDS, если возможно; если нет, то используйте AMDP). В каждом случае идея заключается в том, чтобы сделать работу, которую вы обычно выполняли на сервере приложений (например, отфильтровать все данные, содержащиеся в приложении) в базе данных.

## 7.6 Резюме

Как вы можете себе представить, в наше время практически невозможно написать книгу по SAP (особенно очень техническую), не упомянув о всепобеждающей и танцующей базе данных в памяти SAP HANA. В этой главе было показано, как ASAP программирование меняется благодаря SAP HANA. Вы также можете ожидать быстрого развития в этой области; ABAP версии 7.4 был первым, который был специально включен для SAP HANA, и она стала общедоступной в мае 2013 года. С появлением версии 7.5 в январе 2016 года, все снова изменилось, и теперь представление CDS является звездой шоу программирования ABAP.

Это важно, потому что представления CDS не зависят от базы данных, поэтому вы можете реализовать новую "ориентированную на базу данных" модель программирования на существующей базе данных. Но не заблуждайтесь: SAP заинтересована в том, чтобы вы как можно быстрее перешли на базу данных SAP HANA.

На самом деле, различные люди на SCN проводили исследования производительности, и их результаты, похоже, показывают, что сдвиг кода не имеет преимуществ в производительности за пределами SAP HANA, хотя вы по-прежнему получаете интеграцию представлений CDS с другими технологиями, такими как BOPF и SAPUI5.

Возвращаясь к смехотворной скорости инноваций в этой области, исторически сложилось так, что пакеты поддержки для SAP HANA выпускались дважды в год - примерно в мае или июне (для SAP-PHIRE) и примерно в ноябре или декабре (для SAP Tech Ed). В последнее время эти пакеты поддержки выходят быстрее, чего и следовало ожидать от SAP, чтобы сейчас, когда конкуренты с запозданием осознали, что происходит, и теперь игра идет по принципу "поймай меня, если сможешь", причем SAP намного опережает всех. Возвращаясь к цитате в начале главы, история, скорее всего, скажет, что Хассо Платтнер не был под наркотиками, когда придумал SAP HANA!

375

Короче говоря: ясно, что хотя вы уже можете сделать огромное количество вещей как в представлениях CDS и AMDP, в обоих случаях будут постоянно добавляться новые функции, причем довольно быстро.

Рекомендуемое чтение

• ABAP Development for SAP HANA

www.sap-press.com/3973, Gahm, Schneider, Swanepoel, Westenberger, SAP PRESS, 2016

• ABAP Managed Database Procedures-Introduction

https://blogs.sap.com/2074/07/22/ abap-managed-data base-procedures-introduction/ (Jens Weiler)

• Six Golden Rules for New SAP HANA Developers

https://blogs.sap.com/2073/72/29/6-golden-rulesf-or-new-sap-hana-developers/ (John Appleby)

• CDS Views in ABAP Development

376

http://sapinsider.wispubs.com/Assets/Articles/2075/0ctober/SPI-enhaneed-ABAP-development-with-Core-Data-Services (Karl Kessler)