ORACLE DATABASE 12c

Публикация Oracle

Июнь 2013 г.

Архитектура Oracle Multitenant

ORACLE

Краткий обзор

Oracle Multitenant — это новая опция для Oracle Database 12*c*, которая помогает клиентам снизить затраты на ИТ благодаря упрощению консолидации, провизионирования, обновлений и многих других процессов. Эта опция поддерживается новой архитектурой, в которой одна контейнерная база данных может содержать множество подключаемых баз данных. Она также полностью дополняет другие решения, включая Oracle Real Application Cluster и Oracle Active Data Guard. Существующую базу данных можно легко и без каких-либо изменений использовать как подключаемую. При этом на других уровнях приложения также не требуется никаких изменений. Нужно лишь применить другой вариант развертывания базы данных, чтобы получить преимущества Oracle Multitenant.
Ниже приведены самые яркие примеры преимуществ Oracle Multitenant.

- *Высокая плотность консолидации*. Многочисленные подключаемые базы данных в одной базе-контейнере совместно используют ее память и фоновые процессы. Благодаря этому на одной платформе можно использовать намного больше подключаемых баз данных, чем при развертывании отдельных баз, использующих старую архитектуру. Эти преимущества похожи на те, что предоставляет консолидация на основе схем. Но внедрение консолидации на основе схем связано с немалыми трудностями и ведет к постоянным проблемам эксплуатации. Новая архитектура устраняет эти недочеты.
- *Быстрое провизионирование и клонирование с помощью SQL.* Подключаемую базу данных можно отсоединить от базы-контейнера и подключить к другой. Вы также можете клонировать ее в той же самой или другой контейнерной базе данных. Эти операции, как и создание подключаемой базы данных, производятся с помощью новых команд SQL и занимают считанные секунды. Если базовая файловая система поддерживает тонкое провизионирование, многие терабайты могут быть склонированы практически мгновенно, используя ключевое слово *snapshot* в команде SQL.
- *Новые парадигмы для быстрой установки исправлений и обновлений.* Время и усилия, затрачиваемые на патчирование одной контейнерной базы данных, приводят к патчированию всех подключаемых баз в ее составе. Если требуется исправить только одну подключаемую базу данных, ее просто отключают и подключают к контейнерной с другой версией программного обеспечения Oracle Database.
- *Управление многими базами данных как одной*. За счет консолидации имеющихся подключаемых баз данных администраторы могут управлять многими базами как одной. Например, такие задачи, как резервное копирование и аварийное восстановление, можно выполнять на уровне контейнерной базы данных.
- *Динамическое управление распределением ресурсов между подключаемыми базами данных*. В ПО Oracle Database 12*c* функциональность Resource Manager была расширена, чтобы позволить моментально решать вопрос конкуренции между подключаемыми базами данных в контейнерной базе.

В этой публикации описывается новая архитектура Oracle Database, ее новые функции и соответствующие преимущества.

Отказ от ответственности

Ниже представлено общее направление развития продукта. Эта информация предназначена только для ознакомительных целей и не может быть включена в какой-либо договор. Документ не содержит обязательств по поставке каких-либо материалов, программного кода или функций и не должен использоваться для принятия решений о покупке. Разработка, релиз и время выхода на рынок всех упомянутых компонентов и функций продуктов Oracle относятся исключительно к компетенции корпорации Oracle.

Структура публикации

Для тех, кому не требуются технические подробности

Ознакомьтесь только со следующими разделами:

- *«Задачи, для решения которых предназначена архитектура Oracle Multitenant»* на *стр. 5*
- *«Общее описание архитектуры Oracle Multitenant»* на *стр. 9*
- *«Заключение»* на *стр. 48*

Рекомендуем также кратко ознакомиться со следующими разделами:

- *«Использование не-CDB базы данных в качестве подключаемой»* на *стр. 29*
- *«Варианты на уровне CDB в сравнении с вариантами на уровне PDB»* на *стр. 39*
- *«Управление распределением ресурсов между PDB и внутри CDB»* на *стр. 43*
- *«Одна PDB база данных в CDB и не-CDB база данных»* на *стр. 47*

Для желающих получить полную информацию

Ознакомьтесь со всеми разделами публикации.

Обзор разделов

В разделе *«Задачи, для решения которых предназначена архитектура Oracle Multitenant»* на *стр. 5* описываются три задачи: стремление объединить множество баз данных на одной платформе, чтобы получить максимальную окупаемость инвестиций, провизионирование баз данных и установка исправлений версии Oracle на большом числе баз.

В разделе *«Общее описание архитектуры Oracle Multitenant»* на *стр. 9* вводится терминология, характеризующая новую архитектуру, в частности термины *«мультиарендная контейнерная база данных»* (сокр. CDB) и *«подключаемая база данных»* (сокр. PDB). В нем также описываются в общих чертах высокоуровневые функции и их преимущества при решении задач клиентов.

В следующем разделе *«Статические аспекты многоклиентской архитектуры: горизонтально секционированный словарь данных и подключаемость»* на *стр. 12* мы объясняем недостатки процесса решения задач клиентов при использовании архитектуры для Oracle Database версий, предшествующих 12.1[1], и то, как новая архитектура, появившаяся в версии 12.1, позволяет устранить коренную причину этих проблем. Короче говоря, она позволяет реализовать виртуализацию внутри базы данных, в результате чего «супер база данных» (контейнерная база данных) содержит «подбазы данных» (подключаемые базы данных). Мы рассмотрим причины использования термина «подключаемая база данных» для описания клиентской системы.

В разделе *«Операции с подключаемыми базами данных как сущностями: отключение/подключение, клонирование, создание, удаление»* на *стр. 17* мы рассматриваем, как эти операции, реализованные с помощью команд SQL в контексте мультиарендной архитектуры, вводят новые парадигмы провизионирования и исправления версии Oracle.

[1] Мы будем использовать версию 12.1 в качестве сокращения для Oracle Database 12c, 11.2 — для Oracle Database 11g и так далее.

В разделе *«Присоединение не-CDB базы данных в качестве PDB»* на *стр. 29* раскрывается, как использовать базу данных версии, предшествующей 12.1, в качестве подключаемой.

Далее, в разделе *«Динамические аспекты мультиарендной архитектуры: экземпляр Oracle, пользователи и сессии»* на *стр. 31* мы поясняем причины разделения понятий локальных пользователей и общих пользователей, а также описываем процесс создания сессии, акцентируя внимания на том, что сессия не может покинуть PDB, к которой он» подключается. Мы также узнаем, как SGA, redo журнал и табличные пространства отката виртуализируются в большей степени *логически*, нежели *физически*, путем маркировки каждой структуры идентификатором системы, к которой она принадлежит.

Разобравшись с особенностями новой архитектуры, мы можем перейти к разделу *«Варианты на уровне CDB в сравнении с вариантами на уровне PDB»* на *стр. 39*, где поясняется, в какой степени свобода действий, присущая базам в версиях до 12.1, сохранилась в PDB и какие ограничения пришлось наложить в целях консолидации.

Далее, в разделе *«Управление распределением ресурсов между PDB и внутри CDB»* на *стр. 43* описывается управление распределением ресурсов между PDB в составе контейнерной базы.

В разделе *«Выбор между одиночной PDB в CDB и не-CDB базой данных»* на *стр. 47* сравнивается использование одной PDB внутри CDB, а также БД на основе архитектуры, предшествующей версии 12.1, с целью размещения одного приложения. Мы увидим, что использование одной PDB ничем не хуже и обладает некоторыми преимуществами.

Публикация завершается разделом *«Заключение»* на *стр. 48*. Также в публикации есть приложение:

- *«Приложение A. Использование мультиарендной архитектуры в библиотеке документации по базам данных Oracle»* на *стр. 49*.

Задачи, для решения которых предназначена архитектура Oracle Multitenant

В наше время довольно часто встречается ситуация, когда сотни и даже тысячи баз данных разбросаны по практически такому же числу систем на площадках клиентов от среднего до крупного размера. Связанные с этим издержки привели к попыткам объединить множество баз данных, то есть *консолидировать* их, чтобы сократить затраты. В течение многих лет администраторы баз данных тратили слишком много времени на их провизионирование, исправление версии Oracle на каждой из множества баз, планирование, настройку и управление режимами резервного копирования и аварийного восстановления, а также на прочие аспекты постоянного управления каждой базой в отдельности.

Стремление к максимальной плотности консолидации

Плюсы стандартизации вполне очевидны. Она позволяет объединять множество баз данных, соответствующих определенному стандарту, на одной платформе, реализующей этот стандарт.

Стандартизация сокращает эксплуатационные расходы

Для организации с большим количеством баз данных высокие затраты владения усугубляются различиями между этими базами. Чаще БД разнятся типом оборудования, версией операционной системы, конфигурацией и версией ПО Oracle Database, для которой самым мелким уровнем отличия является разовое исправление. Также имеют значение и другие свойства, например особенности режима резервного копирования и выполнения аварийного восстановления.

Многие клиенты поняли, что первый этап упрощения инфраструктуры заключается в размещении максимального числа баз данных на минимальном количестве аппаратных платформ. На первый взгляд, виртуализация операционной системы кажется неплохим способом, чтобы обеспечить автономное управление отдельными базами данных, каждая из которых размещена на отдельной платформе. Однако было обнаружено, что сокращение затрат на управление достигается именно путем максимальной унификации баз данных, а не за счет виртуализации. Таким образом, клиенты, объединившие множество баз данных на небольшом количестве серверов, пришли к выводу, что виртуализация операционной системы — это решение проблемы, которой у них нет. Поэтому они предпочитают использовать собственно операционную систему без виртуализации и развертывать минимально возможное число каталогов *Oracle Home*. Иными словами, они *приводят к единому стандарту* тип оборудования, версию операционной системы с определенным уровнем исправлений (patch-level) и конфигурацию Oracle Database.

Стандартизация позволяет еще больше сократить эксплуатационные и капитальные расходы

Плотность консолидации ограничена тем, что каждая дополнительная база данных заметно повышает уровень требований к ресурсам памяти и процессоров. Дело в том, что у каждой базы есть собственная SGA и набор фоновых процессов, число которых многократно возрастает при развертывании Oracle Real Application Clusters. В результате становится ясно, что важнейшим показателем является поддерживаемая платформой плотность *серверных приложений*, а не *баз данных*.

Под термином *серверное приложение* мы понимаем набор артефактов, реализующих в рамках базы данных механизм персистенции для конкретного приложения. Для каждого приложения обязательно существует механическая схема установки его серверной части. Она может состоять целиком из сценариев диагностики SQL*Plus, их сочетания с Data Pump import файлами и командами или специальных клиентских программ, выполняющих соответствующие команды SQL. Но механическая схема присутствует в любом случае. Мы используем термин артефакт для обозначения результата любого изменения базы данных, произведенного схемой установки серверного приложения. Самые очевидные примеры — это объекты, указанные в DBA_Objects, такие как таблицы, индексы, представления, пакеты PL/SQL и т. д. Но некоторые артефакты, например пользователи, роли и табличные пространства, не являются объектами в этом смысле. Это относится и к результатам

предоставления привилегий или ролей пользователям или ролям, а также к данным в таблицах приложений, используемым в качестве метаданных приложений.

Традиционно каждое серверное приложение размещалось в собственной выделенной базе данных. Однако в одну базу данных можно установить множество отдельных серверных приложений путем последовательного запуска схемы установки для каждого приложения. Такой подход обычно называют *консолидацией на основе схем*, и мы будем использовать этот термин в данной публикации[2]. Очевидно, что консолидация на основе схем поддерживает гораздо большую плотность, измеряемую числом серверных приложений на каждой платформе, чем размещение каждого серверного приложения в отдельной базе данных, которое далее будет называться *моделью выделенных баз данных*. Плотность консолидации зависит от свойств серверных приложений, таких как необходимый приложению объем памяти базы данных и пропускная способность, которую требуется поддерживать. Более того, значительного сокращения затрат на управление можно добиться путем существенного снижения числа баз данных.

Однако консолидация на основе схем не лишена серьезных недостатков.

- *Конфликты имен могут воспрепятствовать консолидации на основе схем.* Некоторые серверные приложения полностью разворачиваются в пределах одной схемы. В таких случаях имя этой схемы можно выбрать любым, лишь необходимо его указать в централизованном клиентском конфигурационном файле. Такие серверные приложения, как правило, можно без труда установить в одной базе данных. Впрочем, даже в этом случае могут возникнуть некоторые проблемы, как будет указано ниже. Безусловно, именно этот сценарий использования привел к возникновению термина «консолидация на основе схем».

  Другие серверные приложения разворачиваются в нескольких схемах, что подразумевает межсхемные ссылки и, соответственно, зависимость от конкретных имен схем, которые использовались при разработке приложения. Конечно, допускается, что одинаковые имена схем могут использоваться в разных серверных приложениях. Но имя схемы должно быть уникальным в пределах базы данных как единого целого, поэтому серверные приложения с конфликтующими именами схем не могут быть установлены в одну и ту же базу, если одно из них не изменить. Такое изменение, скорее всего, окажется слишком дорогостоящим.

  Кроме того, имена остальных элементов в пределах базы данных как единого целого также должны быть уникальными. К ним относятся роли, каталоги, редакции, публичные синонимы, публичные линки базы данных и табличные пространства. Даже в случае серверных приложений, каждое из которых разворачивается в пределах одной схемы, имена этих элементов могут конфликтовать между собой, что не позволит установить их в одну базу данных.

- *Консолидация на основе схем ослабляет систему безопасности.* Тот, кто устанавливает конкретное серверное приложение, должен войти в систему в качестве пользователя базы данных с высокими привилегиями, такими как *Create User*, *Alter User* и *Drop User*. Этот пользователь может вносить изменения в любое серверное приложение в той же базе данных, а также просматривать или редактировать в ней конфиденциальные данные. Более того, плохо спроектированное серверное приложение, для развертывания которого требуется несколько схем, может основываться на таких системных привилегиях, как *Select Any Table*, *Delete Any Table* и т. д. В таком случае уязвимость (например, инъекция SQL-кода) может позволить пользователю одного серверного приложения прочесть или изменить данные другого серверного приложения.

[2] В дальнейшем мы увидим, что консолидация на основе схем — это разновидность консолидации внутри базы данных. И проведем различие между ней и консолидацией, обеспечиваемой мультиарендной архитектурой, то есть консолидацией на основе подключаемых баз данных.

Для работы серверного приложения может потребоваться назначить определенные привилегии пользователю *public*, например *Execute* для *Sys.Utl_File*. Однако другое серверное приложение может запретить назначение этой привилегии пользователю *public*. Два таких серверных приложения не могут сосуществовать в одной базе данных.

- *Восстановление состояния серверного приложения на определенный момент времени — очень сложная процедура.* Как правило, восстановление состояния на определенный момент времени требуется, если ошибка в коде, вызванная применением патча приложения, привела к неисправимому повреждению данных. В таком случае требуется восстановить состояние отдельного серверного приложения (но не всей базы данных) на момент времени, предшествующий установке плохого патча. В лучшем случае можно обойтись восстановлением состояния табличного пространства на определенный момент времени. Но если данные повреждены в нескольких табличных пространствах, это может оказаться трудной задачей. Сложности также возникнут, если администратор по ошибке удалил одну или несколько взаимосвязанных таблиц.

- *Управление ресурсами между серверными приложениями — сложная задача.* Оно основано на сформировавшейся традиции, что один или несколько конкретных сервисов будут использоваться для открытия сессий с доступом к определенному серверному приложению. При этом одну службу можно использовать только для одного серверного приложения. Иными словами, различия между каждым серверным приложением известны только администратору, база данных не знает ничего об этих различиях.

- *Исправление версии Oracle для одного серверного приложения невозможно.* Невозможно исправить версию Oracle Database для одного серверного приложения, не затронув всех остальных. Единственная альтернатива — использовать Data Pump для перемещения серверного приложения, которому требуется исправление, в другую базу данных, где это исправление уже установлено. Однако некоторые типы артефактов баз данных, такие как схемы XML, нельзя переместить с помощью Data Pump.

- *Клонирование одного серверного приложения затруднено.* Единственный вариант — использовать технологию Data Pump.

Несмотря на значительные недостатки консолидации на основе схем, многие клиенты воспользовались этой практикой и добились серьезного роста окупаемости инвестиций. Причина в том, что снижение капитальных затрат за счет высокой плотности консолидации и эксплуатационных расходов благодаря сокращению числа баз данных, требующих управления, превышает затраты, вызванные описанными выше недостатками.

Провизионирование баз данных

В течение многих лет администраторам приходилось тратить много рабочего времени на создание новых и перемещение существующих баз данных между серверами, а также на создание наиболее актуальных клонов существующих баз данных для различных задач разработки, тестирования и диагностики проблем. Мы обозначим этот тип рутинных задач термином *провизионирование*.

Патчирование и апгрейд версии ПО Oracle Database

Хотя это происходит не так часто, как провизионирование базы данных, но установка разовых патчей (one-off patches), пакета патчей (bundled patches), обновлений набора патчей (PSU, patch set updates) (критически важных или обычных), наборов патчей (patch sets) на существующие базы данных, а также обновление баз данных с релиза .1 до .2 или релиза .2 до следующего релиза .1 отнимает много сил и времени. Назовем эти рутинные задачи как *патчирование версии Oracle*.

Мы будем часто использовать этот термин, поскольку, это слишком многословно всегда объяснять разницу между патчированием и апгрейдом или указывать, что мы имеем в виду именно версию ПО Oracle Database, включая исполняемые двоичные файлы и систему Oracle в базе данных, а не версию серверного приложения.

Общее описание архитектуры Oracle Multitenant

Система Oracle Database 12*c* поддерживает новую архитектуру, которая позволяет использовать множество «подбаз данных» в составе единой «супер базы данных». Впредь мы будем использовать официальную терминологию. «Супербаза данных» — это *мультиарендная контейнерная база данных* (сокр. *CDB*), а «подбаза данных» — это *подключаемая база данных* (сокр. *PDB*). Другими словами, эта новая архитектура позволит вам разместить множество баз данных PDB в единой базе CDB (в версии 12.1 — до 252 баз данных). Мы будем называть новую архитектуру *мультиарендной архитектурой*.

Теперь необходимо ввести термин для обозначения старого типа баз данных — единственного типа, поддерживаемого до версии Oracle Database 11*g* включительно. Назовем их *не-CDB базами данных*. Соответственно, старую архитектуру мы обозначим термином *не-CDB архитектура*.

Система Oracle Database 12*c* релиза 1 поддерживает как новую мультиарендную архитектуру, так и прежнюю не-CDB архитектуру. Иными словами, вы определенно можете обновить базу данных версии, предшествующей 12.1, которая, естественно, является не-CDB, до версии 12.1 и продолжать эксплуатировать ее в этом виде. Но при желании вы можете включить не-CDB БД версии 12.1 в состав CDB как PDB[3].

С точки зрения клиента, подключающегося через Oracle Net, PDB и есть база данных. PDB БД полностью совместима с не-CDB. Впредь мы будем называть эту особенность *гарантией PDB/не-CDB совместимости*[4]. Иначе говоря, схема установки серверного приложения, корректно работавшего в не-CDB БД, будет с тем же результатом работать *без изменений* и ошибок с PDB. Кроме того, процесс выполнения клиентского кода, подключающегося к PDB, в которой установлено серверное приложение, будет идентичен процессу выполнения клиентского кода, подключенного к не-CDB БД с этим серверным приложением. PDB может содержать только *одно* серверное приложение. Благодаря этому такая база предоставляет прямой декларативный механизм размещения серверного приложения, чтобы система Oracle явно видела связи между артефактами и серверными приложениями.
И наоборот, при использовании консолидации на основе схем в не-CDB БД у системы Oracle нет сведений о принадлежности различных артефактов. Мы скоро увидим, что каждый пользовательский процесс видит только одну PDB. Фактически в рамках простейшей модели пользователь, определенный в пределах PDB, может создавать только сессии, которые видят *только* эту PDB, и, соответственно, лишь артефакты, принадлежащие одному серверному приложению[5].

Признание гарантии PDB/не-CDB совместимости позволяет утвердительно ответить на многие вопросы. В ином случае этот принцип не получил бы признания. Тестирования, которые инженеры корпорации Oracle проводили в течение всего цикла разработки версии 12.1, показали, что при подключении через Oracle Net клиентский код не способен различить PDB и не-CDB БД. Вот несколько примеров.

- *Могут ли две PDB в одной и той же CDB содержать каждая пользователя с именем Scott?* Да, так как две не-CDB БД могут содержать пользователя с именем *Scott*. Две подключаемые базы данных в составе одной контейнерной могут также содержать следующие элементы с одинаковыми именами: роль, директорию, редакцию, публичный синоним, публичный линк и табличное пространство. Более того, в одной PDB вы можете назначить привилегию *Execute* для *Sys.Utl_File* пользователю *public*, а в другой ее не назначать.
- Привилегия, назначенная public в PDB, видна только в этой PDB. Так же и привилегия *Any* может быть использована только в той подключаемой БД, в которой она предоставлена.

---

[3] Подробные сведения можно найти в разделе *«Присоединение не-CDB базы данных в качестве PDB»* на стр. 29.

[4] В разделе *«Ошибка ORA-65040»* на *стр. 42* содержится оговорка относительно гарантии совместимости PDB и не-CDB БД.

[5] Это подробно описывается в разделе «Динамические аспекты мультиарендной архитектуры: экземпляр, пользователи и сессии Oracle» на стр. 31.

Иными словами, подключаемая БД, как и не-CDB, определяет глобальное пространство имен. И содержит результаты назначения привилегий, также как это происходит в не-CDB базе данных.

- *Можно ли создать линк между двумя подключаемыми базами данных?* Да, поскольку это возможно между двумя не-CDB базами. Линк базы данных создается путем выполнения команды SQL. Поэтому такая команда должна приводить к тому же результату в подключаемой БД, что и в не-CDB. Вы также можете создать линк из PDB к не-CDB базе данных и наоборот. При этом разница версий ПО Oracle Database между этими базами нивелируется таким же образом, как и между двумя не-CDB.

- *Можно ли настроить репликацию GoldenGate между двумя подключаемыми базами данных?* Да, а также можно настроить ее между PDB и не-CDB БД, устраняя различие версий ПО Oracle Database (эта возможность появилась в первой версии Oracle GoldenGate, поддерживающей версию 12.1[6]).

С точки зрения операционной системы CDB и есть база данных. Каждый экземпляр RAC открывает контейнерную БД как единое целое, а каждая SGA может содержать блоки данных и структуры библиотечного кэша, такие как дочерние курсоры, из каждой подключаемой БД в составе контейнерной[7] (это косвенный способ указать на то, что мультиарендная архитектура, безусловно, полностью совместима с Oracle Real Application Clusters).

Итак, мы видим, что мультиарендная архитектура поддерживает новую модель для консолидации внутри базы данных — консолидацию на основе PDB.

Заполнение SGA в модели консолидации на основе PDB можно напрямую сравнить с тем же процессом в модели консолидации на основе схем. В последней модели каждый экземпляр RAC также открывает не-CDB базу данных, а каждая глобальная системная область может содержать блоки данных и структуры библиотечного кэша из каждого серверного приложения, консолидированного в не-CDB базе данных. Отсюда следует, что уровень патчирования Oracle Database относится именно к CDB. PDB базы данных в ее составе наследуют его, как и схемы в не-CDB БД. Теперь вопрос, могут ли две подключаемые базы в одной контейнерной находиться на разных уровнях патчирования Oracle Database, аналогичен вопросу, могут ли схемы *Scott* и *Blake* находиться на разных уровнях патчирования Oracle Database в одной не-CDB базе.

Пользователь базы данных с соответствующими привилегиями позволяет тому, кто знает его пароль, открывать сессию с возможностью просмотра информации из представлений словаря данных и производительности во всех подключаемых базах в составе CDB[8]. Другими словами, один образ системы доступен через SQL и, следовательно, через такие инструменты, как SQL Developer и Enterprise Manager. Фактически эти инструменты были расширены, чтобы предоставлять все новые функции, появившиеся с выходом мультиарендной архитектуры.

---

[6] Потребовалась доработка продукта, поскольку в мультиарендной архитектуре каждая запись в redo журнале помечается идентификатором ее исходного контейнера. См. раздел *«Data Guard и RMAN (резервное копирование, redo и undo)»* на *стр. 39*.

[7] В разделе *«Логическая виртуализация SGA»* на *стр. 34* мы увидим, что каждая подключаемая база данных может иметь свой режим *Open_Mode* в каждом экземпляре RAC.

[8] Для полноценного понимания этого аспекта необходимо разобраться в понятиях, разъясняемых далее, а именно *root* (в разделе *«Статические аспекты мультиарендной архитектуры: горизонтально секционированный словарь данных и подключаемость»* на стр. 12), общий пользователь (в разделе *«Динамические аспекты мультиарендной архитектуры: экземпляр, пользователи и сессии Oracle»* на стр. 31) и *представления container_data* (в разделе *«Представления словаря данных и производительности»* на стр. 36).

Мультиарендная архитектура расширяет функциональность Resource Manager, позволяя плану на уровне CDB управлять распределением ресурсов между PDB базами данных.

Oracle Active Data Guard работает на уровне CDB базы данных, так же как и регулярное RMAN-архивирование. Восстановление состояния на определенный момент времени поддерживается на уровне подключаемых баз данных.

Наконец, вы можете отключить PDB базу данных от одной CDB базы данных и подключить к другой. Именно эта возможность и дала подключаемой базе данных такое название. Вы также можете создать новую PDB базу данных в качестве клона существующей. Если базовая файловая система поддерживает тонкое провизионирование, то клонировать многие терабайты данных можно практически мгновенно. Все операции с PDB как непрозрачными сущностями (создание совершенно новой базы данных, отключение и подключение, клонирование и удаление) представлены в виде команд SQL. Чтобы запросить клонирование, используя тонкое провизионирование, просто используйте ключевое слово *snapshot* в команде SQL. Отключение или подключение реализовано в виде естественного расширения технологии транспортируемых табличных пространств и возможно благодаря виртуализации внутри базы данных, характерной для мультиарендной архитектуры. Отключение или подключение поддерживается между разными версиями Oracle Database[9].

Новая мультиарендная архитектура доступна в версиях Standard Edition, Standard Edition One и Enterprise Edition. Однако без лицензирования опции Oracle Multitenant вы будете ограничены одной PDB[10]. Важность этой модели описывается в разделе *«Выбор между одиночной PDB в CDB и не-CDB базой данных»* на *стр. 47*.

[9] Все это подробно объясняется в разделе «Статические аспекты мультиарендной архитектуры: горизонтально секционированный словарь данных и подключаемость» на стр. 12.

[10] В руководстве по лицензированию Oracle Database 12*c* эти термины описываются в официальной форме.

Статические аспекты мультиарендной архитектуры: горизонтально секционированный словарь данных и подключаемость

Клиенты, выполнявшие консолидацию на основе схем, по сути, пытались реализовать виртуализацию внутри баз данных без встроенной поддержки со стороны Oracle Database. В этом разделе описаны сложности применения данного похода к исторически сложившейся архитектуре словаря данных. Затем приведена базовая концепция виртуализации, присущей мультиарендной архитектуре: это горизонтально секционированный словарь данных.

Таблицы: абсолютная логическая реальность

Остановленная база данных Oracle Database состоит только из входящих в нее файлов. За исключением загрузочных файлов (управляющий файл, spfile и другие), основная масса файлов обеспечивает реализацию табличных пространств. Табличные пространства, в свою очередь, содержат лишь таблицы и различные структуры, которые ускоряют доступ и обеспечивают возможность восстановления таблиц. При этом смыслом существования табличных пространств, разумеется, остаются сами таблицы. Другие артефакты баз данных всех видов (ограничения, представления, объекты PL/SQL и т. д.) в конечном итоге представляются в виде строк в таблицах. Таблицы содержат данные трех видов: метаданные, описывающие систему Oracle; метаданные, учитывающие созданные клиентом серверные части приложений; засчитываемые в квоту данные в таблицах серверных частей приложений.

Монолитный словарь данных не-CDB архитектуры

До версии 11.2 включительно метаданные, описывающие систему Oracle, а также созданные клиентом серверные части приложений, были представлены в одном наборе таблиц — словаре данных. Таблицы совершенно незаметно поддерживаются исполняемыми командами, которые генерируются экземпляром Oracle как побочный эффект выполнения команд DDL. Клиентам не разрешается напрямую вставлять, обновлять и удалять данные в этих таблицах, а их структура не задокументирована. Поэтому к данным, которые они представляют, необходимо производить запросы через представления словаря данных. Тем не менее клиентам следует знать названия этих таблиц (*Obj$*, Tab$, Col$ и т. д.) и учитывать их важность. Таблицы словаря данных хранятся в выделенных табличных пространствах. Самое важное из них — пространство *System*. На рис. 1 показан контент только что созданной не-CDB базы данных без каких-либо клиентских артефактов.

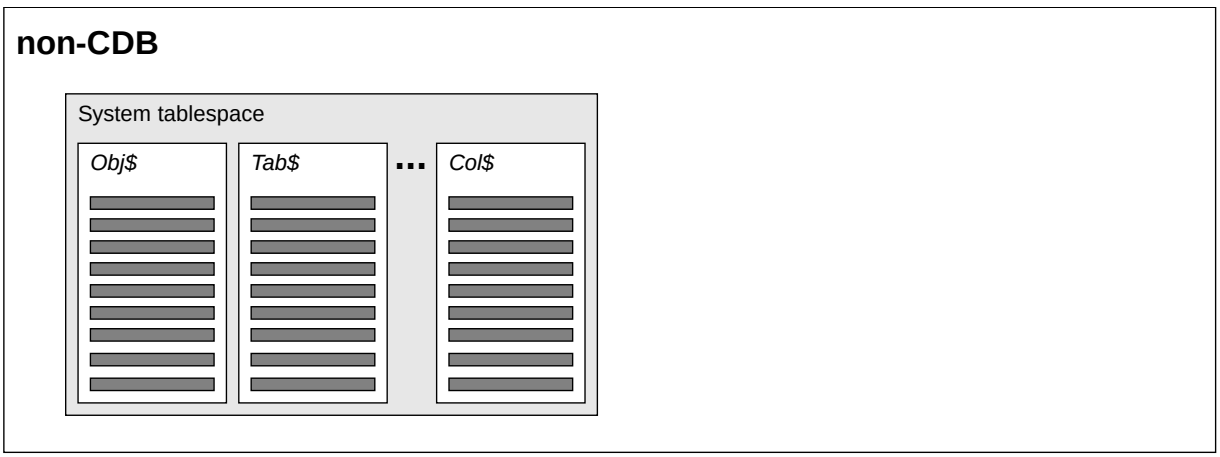

Рисунок 1. Вновь созданная не-CDB БД

Первым следствием установки серверной части приложения в не-CDB БД будет вставка в таблицы словаря данных строк, представляющих метаданные приложения. Поскольку конечная цель серверной части (бэкэнда) приложения — реализовать механизм сохраняемости данных приложения,

в состав метаданных, разумеется, войдет описание таблиц приложения. В соответствии с общепринятой лучшей практикой эти таблицы будут сохраняться в выделенных табличных пространствах данных приложения. Вскоре в таблицы приложения будут добавлены данные, потребляющие квоту, так что общий набор таблиц, определяющих не-CDB БД, примет показанный на *рис. 2* вид.

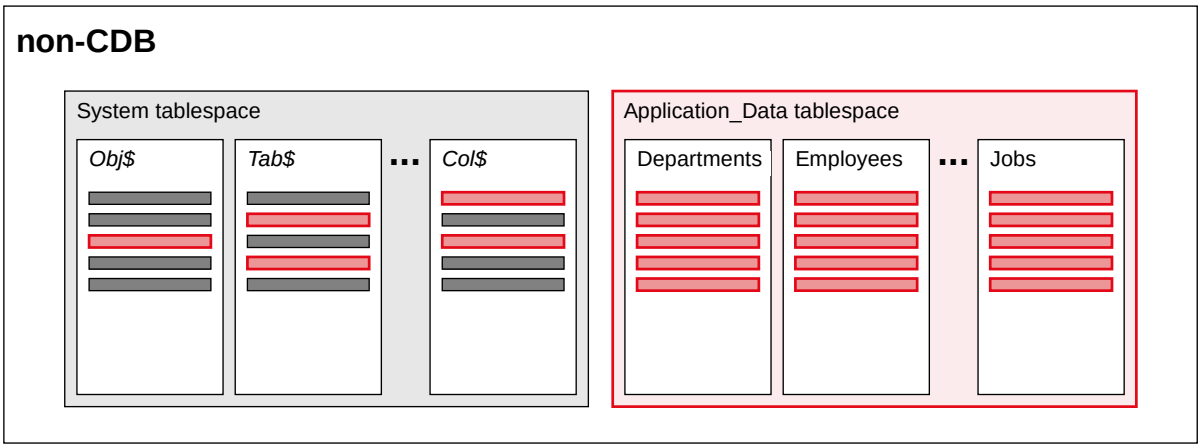

Рисунок 2. Не-CDB БД после установки артефактов серверной части приложения

На *рис. 2*, а также на *рис. 3* и *рис. 4* в следующем разделе красным цветом обозначаются созданные клиентом артефакты (как строки метаданных, так и строки данных, потребляющих квоту), а черный обозначает артефакты, представляющие систему Oracle (это только строки метаданных).

Разумеется, пространства имен для глобальных артефактов (пользователи, роли, редакции, табличные пространства и т. п.) определяются уникальными индексами по таблицам, таким как User$, Edition$, TS$ и т. п. Глобальные артефакты наподобие прав, предоставленных пользователям, и ролей, включая роль *public*, представлены в таблицах *SysAuth$* и *ObjAuth$*. Это объясняет коллизии имен, которые часто возникают при попытках установить серверные части двух и более приложений в одну и ту же не-CDB БД, а также проблемы безопасности, обусловленные такой установкой.

Рис. 2 позволяет также лучше понять, какой шаг вперед был сделан во втором поколении утилиты Data Pump — использование транспортируемых табличных пространств. В Data Pump первого поколения использовалась исключительно построчная (slow-by-slow, или «мало-помалу»)[11] схема конструирования SQL при экспорте и его воспроизведения при импорте (как команд DDL, так и команд DML). Во втором поколении исчезла необходимость использовать команды DML. Набор табличных пространств, в которых содержатся потребляющие квоту данные приложения (на рисунке они показаны справа и полностью выделены красным) стал рассматриваться в качестве автономных единиц, содержащих полную информацию о табличных данных и индексах по ним. При этом важно, что во время импорта транспортируемое табличное пространство подключается только посредством операций с метаданными, которые выполняются очень быстро.

Рис. 2 также показывает причину, по которой до версии 12.1 реализация Data Pump третьего поколения была невозможна: поскольку каждая таблица словаря данных содержала и клиентские метаданные, и системные метаданные Oracle, единственной возможностью переноса этой составляющей серверной части приложения оставались медленные (slow-by-slow) инструкции DDL.

В целом именно монолитный словарь данных архитектуры не-CDB БД вызывал сложности при консолидации на основе схем, обусловливал неудовлетворительную безопасность получаемого

[11] Выражение slow-by-slow, которое можно перевести на русский как «мало-помалу», принадлежит Тому Кайту (Tom Kyte) из Oracle. Оно часто используется на веб-сайте AskTom, а также в статьях Тома для Oracle Magazine.

режима работы и ограничивал мобильность серверных частей приложений. Другими словами, словарь данных архитектуры не-CDB БД не виртуализируется.

Горизонтально секционированный словарь данных мультиарендной архитектуры

Решение этих проблем очевидно: в мультиарендной архитектуре словарь данных виртуализируется.

Подход в общих чертах

Согласно учебникам, классический подход к виртуализации — выявить каждый феномен, с помощью которого реализуется система, и пометить его идентификатором, указывающим, кому он принадлежит. Простейший способ решить эту задачу — добавить по столбцу в каждую таблицу словаря данных, чтобы указать в нем идентификатор серверной части приложения (и использовать особое значение для строк, описывающих саму систему Oracle). Но этот подход не решает проблему мобильности серверных частей приложений. Поэтому схема логической маркировки тегами была реализована физически, посредством горизонтального секционирования словаря данных. (Реализацию этого функционала не следует путать с реализацией опции Oracle Partitioning, открытой для использования клиентами. Детали их реализации существенно отличаются. Словарь данных секционируется в обычном, абстрактном смысле этого термина.) На рис. 3 показана схема для CDB, содержащей серверную часть одного приложения.

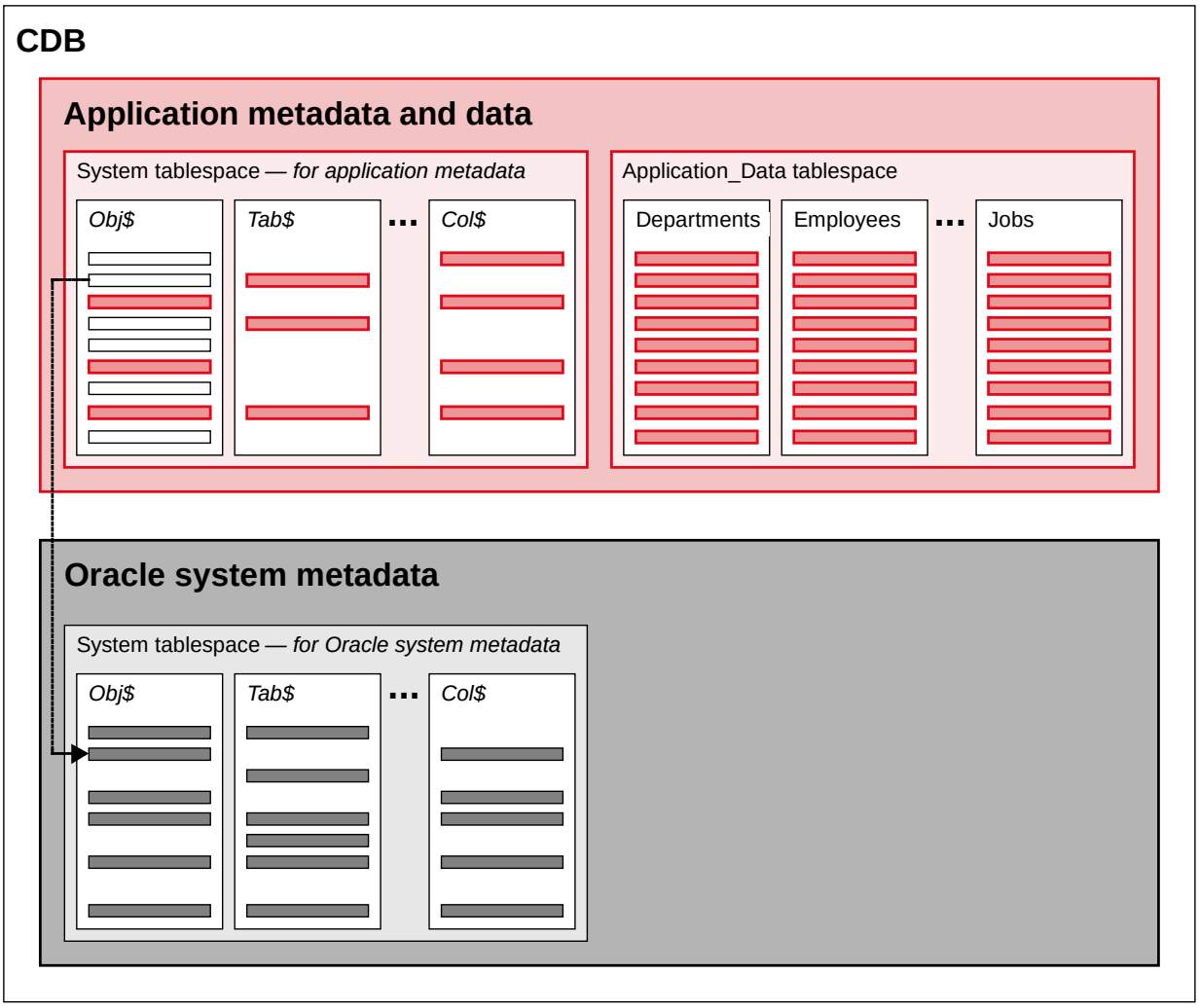

Рисунок 3. В мультиарендной архитектуре вводится горизонтально секционированный словарь данных

«Нижняя» часть (в расположении, выбранном нами в этом документе) содержит лишь метаданные системы Oracle. А в «верхней» части содержатся метаданные серверной части приложения — и ничего более. Другими словами, каждая таблица словаря данных теперь присутствует дважды: один раз для

системы Oracle и один раз для серверной части приложения. Поэтому каждый запрос к таблице словаря данных теперь является объединением (UNION) между двумя такими таблицами.

Обратите внимание, что каждый из наборов таблиц словаря данных расположен в собственном табличном пространстве (пространствах) и поэтому хранится в отдельных файлах данных. Разумеется, пути к файлам данных в файловой системе должны быть уникальными. Однако имена табличных пространств и содержащихся в них сегментов должны быть уникальными лишь в рамках таблицы словаря данных либо в «верхней», либо в «нижней» секциях. Поэтому естественно использовать одни и те же имена для таблиц словаря данных, а также одни и те же имена для содержащих их табличных пространств в обоих секциях. Это соображение подробнее рассматривается в следующем разделе.

Практическое определение PDB и root

Набор табличных пространств и их файлов данных, который реализует таблицы словаря данных с метаданными системы Oracle, обозначается как *root* (его имя — *CDB$Root*). А набор табличных пространств (и их файлов данных), которые реализуют таблицы словаря данных с метаданными для серверной части приложения — наряду с набором табличных пространств, содержащим потребляющие квоту данные (и их файлы данных) — называется подключаемой БД (PDB). Подключаемая база данных, таким образом, является полной базой данных, а *root* — это всего лишь метабаза. Поскольку теперь словарь данных виртуализирован, одна и та же CDB может содержать много PDB баз, каждая из которых представляет собой отдельный набор автономных файлов данных.

Эта схема показана на *рис. 4*.

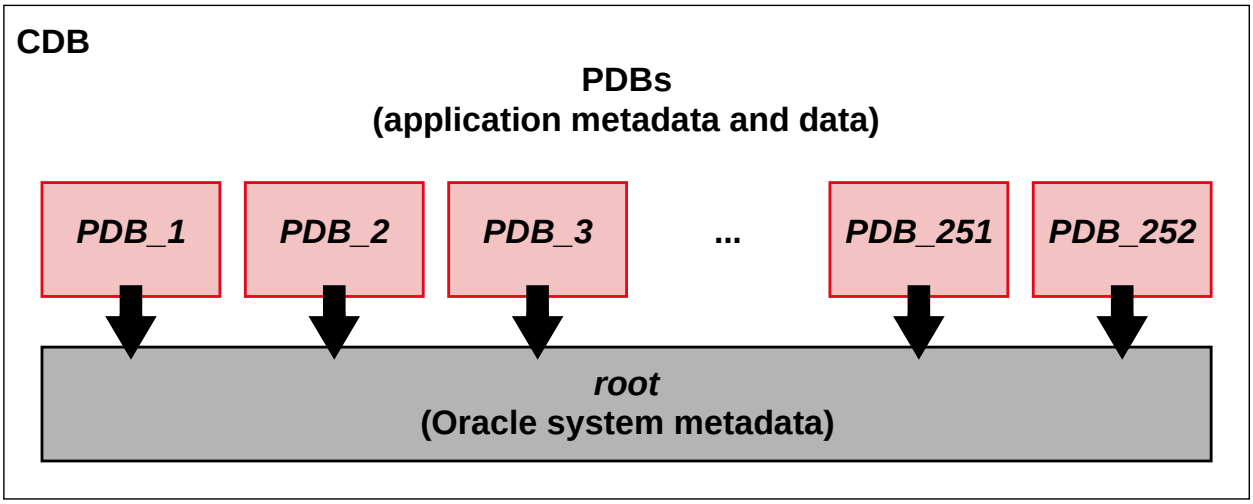

Рисунок 4. В CDB архитектуре к root подключается до 252 PDB баз данных

Важно отметить, что каждая из подключаемых баз определяет частное пространство имен для всех элементов (пользователи, роли, публичные синонимы т. д.), которые в не-CDB базе данных должны быть уникальными. В каждой из подключаемых баз данных, входящих в определенную CDB БД, может быть пользователь под именем *Scott*. Вследствие этого результаты предоставления привилегий и ролей, представленные в таблицах словаря данных *SysAuth$* и *ObjAuth$*, также ограничиваются пределами одной подключаемой БД.

Root принципиально отличается от PDB баз данных тем, что никогда не содержит потребляющие квоту данные. Но между этими компонентами CDB БД существует значительное сходство, в обоих имеется словарь данных, каждый может быть в фокусе пользовательских (foreground) процессов. Поэтому мы будем использовать термин контейнер как родительский (superclass) термин для *root* или PDB базы данных. У пользовательского процесса, и, таким образом, у сессии в каждый момент его существования имеется уникально определенный *текущий контейнер*.

Подключаемая БД называется так, потому что ее можно отключить от *root* одной CDB и подключить ее к *root* другой CDB БД. На *рис. 4* это обозначается большими черными стрелками. Разумеется, метаданные *root* описывают каждую подключенную PDB. При отключении метаданные PDB удаляются. При ее подключении создаются метаданные, описывающие эту подключенную БД. Это логичное расширение технологии транспортируемых табличных пространств, использующейся в версиях до 12.1. Таким образом, отключение и подключение PDB можно рассматривать как Data Pump третьего поколения. Потребляющие квоту данные и метаданные, совместно представляющие серверную часть приложения, могут перемещаться как непрозрачный автономный модуль. За счет этого устраняется необходимость в конструировании и воспроизведении последовательных и медленных команд DDL.

Некоторые детали новой реализации словаря данных

В этом разделе даны разъяснения, как реализуются запросы к словарю данных. На это объяснение нет ссылок в дальнейшем тексте, оно также не обязательно для понимания приведенного ниже материала.

Все запросы реализуются посредством переноса блоков в буферный кэш. Поэтому у блока должен существовать адрес. Частью такого адреса является файл, в котором блок хранится. Мультиарендная архитектура использует относительную схему нумерации файлов данных, основанную на идентификаторе текущего контейнера сессии. При обычной эксплуатации текущим контейнером служит подключаемая БД, поэтому у сессии есть доступ только к блокам из файлов данных этой базы. Такая схема относительной нумерации файлов данных реализована на низком уровне в модулях различных слоев, составляющих Oracle Database. Идентификатор сессии PDB передается в низкоуровневый модуль; при этом различные высокоуровневые API-интерфейсы, по которым подсистемы компиляции и выполнения SQL и PL/SQL взаимодействуют с нижележащим уровнем данных, остаются неизменными по сравнению с архитектурой не-CDB БД. Это означает, что для огромных объемов кода, реализующих Oracle Database — весь SQL (включая оптимизатор), весь PL/SQL и все остальные компоненты (например, Scheduler), основанные на них — не требуется изменений, чтобы обеспечить поддержку новой мультиарендной архитектуры в версии 12.1. Поэтому обеспечить гарантию совместимости PDB и не-CDB БД было сравнительно просто.

Когда запросу к словарю данных, исполняемому в контексте подключаемой БД, нужно получить доступ к строкам, описывающим систему Oracle, текущий контейнер переключается на *root*, а затем переключается обратно. Механизм такого переключения, обеспечивающий эффективность, обозначен пунктирной стрелкой на *рис. 3*, идущей из строки в таблице *Obj$* подключаемой БД на строку в таблице *Obj$* БД *root*. Поставляемый Oracle объект (например, таблица словаря данных) полностью представлен в *root* строками всех детальных таблиц, прямо или непрямо ссылающихся на строку этого объекта в таблице *Obj$* БД *root*. При этом он представлен в PDB как ссылка, одной строкой в ее таблице *Obj$*, которая указывает на соответствующую строку в *root*.

Операции над подключаемыми БД как сущностями: отключение/подключение, клонирование, создание, удаление

Далее мы рассмотрим, как эти операции, внедренные в контексте мультиарендной архитектуры, реализуют новые подходы к провизионированию и патчированию версии Oracle.

Отключение и подключение на другой машине

На *рис. 5* показано подключение PDB_1 к *root* в CDB_1, работающей на *машине 1*. Согласно схеме, в PDB_1 содержится серверная часть ровно одного приложения. Необходимо переместить эту серверную часть с *машины 1* на *машину 2*.

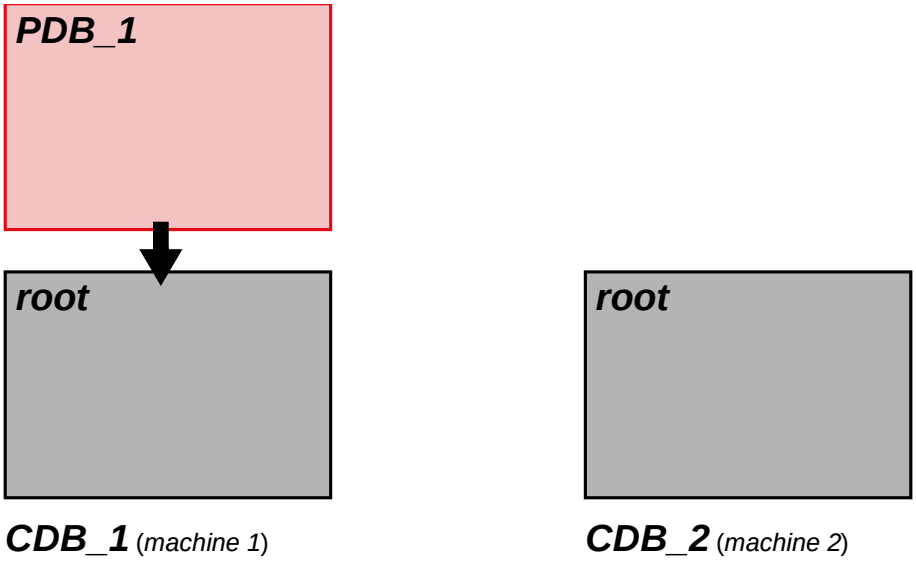

Рисунок 5. Отключение/подключение между двумя машинами: PDB_1 начинает работу в CDB_1

Когда PDB содержится в CDB, CDB содержит ее метаданные, такие как имя, которое PDB имеет в CDB, и пути к файлам данных PDB. (Часть метаданных хранится в *root*, а часть — в контрольном файле CDB. В контексте нашего материала это не имеет значения). При отключении метаданные записываются во внешний файл манифеста, который сопровождает файлы данных подключаемой БД (PDB). Отключение осуществляется одной командой SQL, как показано в *Code_1*.

```
-- Code_1
-- Подключаемую БД (PDB) необходимо закрыть перед отключением.
alter pluggable database PDB_1
unplug into '/u01/app/oracle/oradata/…/pdb_1.xml'
```

Используется следующая семантика: «укажите имя базы данных PDB», «укажите, что ее нужно отключить» и «укажите путь для внешнего файла манифеста».

Аналогичным образом подключение производится одной командой SQL, как показано в Code_2. (Команда SQL для подключения отключенной в данный момент PDB — это вариант команды create pluggable database, т. е. создание подключаемой БД. Другие варианты — это clone PDB, то есть клонирование PDB, и создание полностью новой PDB. В примерах, приведенных в *Code_2, Code_3 на стр. 25, Code_4 на стр. 25, Code_6 на стр. 25 и Code_7 на стр. 26*, предполагается, что функция Oracle-Managed Files включена).

```
-- Code_2
-- PDB необходимо открыть после ее подключения.
create pluggable database PDB_2
using '/u01/app/oracle/oradata/…/pdb_1.xml'
path_prefix = '/u01/app/oracle/oradata/pdb_2_Dir'
```

Выражение *path_prefix* необязательно. Его можно указывать только в команде SQL *create pluggable database*. Его нельзя изменить для существующей подключаемой базы данных (PDB). Оно влияет на результат выполнения команды SQL *create directory* в случае ее запуска из PDB, которая была создана с использованием этого выражения. Строка символов, указанная в выражении *path* команды *create directory*, добавляется к строке символов, указанной в выражении *path_prefix* команды *create pluggable database*, для получения фактического пути, который будет использоваться для создания новой директории. Это важная мера для обеспечения изоляции PDB баз данных друг от друга.

Результат выполнения *Code_2* показан на *рис. 6*.

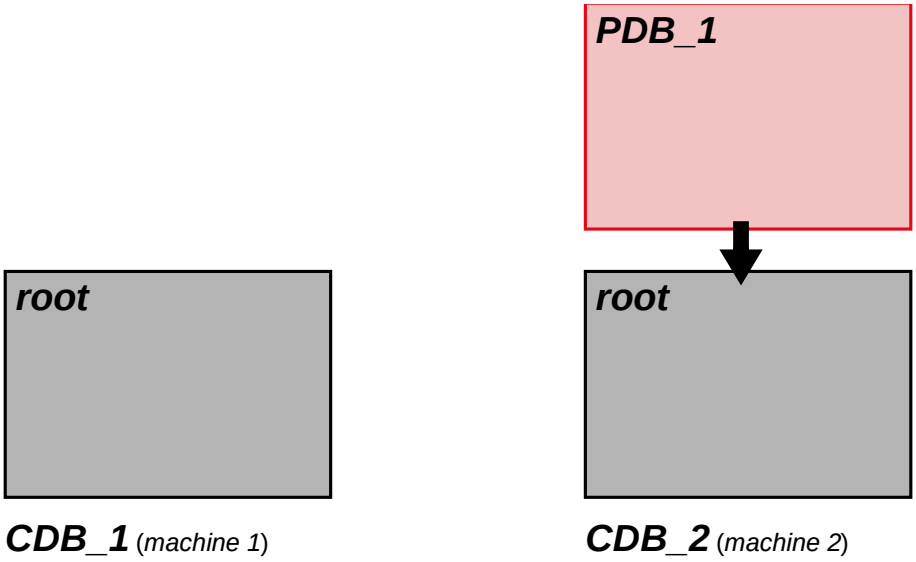

Рисунок 6. Отключение и подключение между двумя машинами: PDB_1 оказывается в CDB_2

Используется следующая семантика: «укажите, что вам нужна новая база данных PDB», «укажите, что она создается посредством подключения отключенной PDB с указанием пути к внешнему файлу манифеста», «задайте для подключенной PDB имя в новой CDB и обеспечьте создание всех объектов директорий PDB в заданном поддереве файловой системы, которую использует CDB».

Отключение и подключение между архитектурами с разным порядком следования байтов

К кроссплатформенному unplug/plug (отключению/подключению) применимы соображения, аналогичные варианту кроссплатформенного использования транспортируемых табличных пространств. Когда RMAN команда *convert* преобразует пользовательское табличное пространство в табличное пространство с другим порядком байтов (endianness), тогда фактически изменяются только заголовки блоков (так как их формат зависит от порядка байт процессора). Данные в столбцах пользовательских таблиц остаются нетронутыми. В большинстве из них (как мы надеемся) будут использоваться примитивные скалярные типы данных наподобие *varchar2* и *number* — а эти типы представляются в сетевом порядке байтов и нечувствительны к порядку байтов аппаратной архитектуры. С другой стороны (как мы опять же надеемся), при использовании столбцов типов данных *raw* или *blob* они будут применяться для хранения данных (таких как, например, PDF- или JPG-файлы), которые потребляются только кодом на стороне клиента, способным их интерпретировать. Однако пользователи могут кодировать данные чувствительным к порядку байтов образом. В этом случае они должны учитывать, где было произведено кодирование, и применять собственную

обработку для преобразования порядка, после того как транспортируемое табличное пространство, содержащее такие данные, будет подключено в системе с другим порядком байтов.

Но в отличие от транспортируемого табличного пространства ситуация с PDB несколько сложнее, так как отключенная PDB включает еще и таблицы словаря данных, а часть столбцов в них содержит информацию, закодированную чувствительным к порядку байтов образом. Поддерживаемого способа автоматически обрабатывать преобразование таких столбцов не существует. Попросту говоря, это означает, что отключенную базу данных PDB нельзя переносить между системами с различным порядком байтов.

Словарь данных остается чувствительным к порядку байтов. Поэтому, если поставлена задача обеспечить отключение и подключение на архитектурах с разным порядком байтов, единственным работающим подходом будет перенос контента с помощью Data Pump в новую, пустую базу данных PDB в целевой CDB. Если используются транспортируемые табличные пространства, их необходимо обработать с помощью RMAN *convert*. Когда объем данных невелик, использование «классического» варианта Data Pump без транспортируемых табличных пространств может работать быстрее.

Более того, помимо аспектов порядка байтов, в базе данных могла использоваться нативная компиляция PL/SQL. В таком случае в определенной таблице словаря БД будет содержаться машинный код, зависимый от чипсета платформы, на которой он был скомпилирован. Если отключенную базу PDB, содержащую такой машинный код, подключить к CDB, работающей на платформе с чипсетом, отличным от чипсета платформы, где работала CDB, от которой была изначально отключена эта PDB, то этот код перестанет работать. В таком случае нужно предпринять простые действия: все нативно скомпилированные блоки PL/SQL в поступившей PDB необходимо перекомпилировать до того, как PDB будет открыта для общего доступа.

Отключение и подключение для обновления версии Oracle

Мультиарендная архитектура поддерживает подключение PDB в CDB, версия программного обеспечения Oracle Database которой отличается от версии той CDB, из которой изначально была отключена эта подключаемая база данных.

Наименее «агрессивные» патчи меняют лишь двоичные файлы Oracle. Иными словами, такой тип патча по определению не может вносить исправления в файлы данных. Отсюда можно сделать вывод, что если исходная и целевая CDB базы данных отличаются лишь установкой изменений такого рода, то менять подсоединяемую PDB вообще не требуется. Необходимо лишь убедиться, используя информацию из PDB манифеста, что отличия в версии ПО Oracle Database сводятся к разнице такого рода.

Более «агрессивные» патчи меняют определения системы Oracle в словаре данных. (Обычно они также изменяют и двоичные файлы Oracle). Но чтобы заслужить имя патч, а не апгрейд, такие изменения словаря данных обычно не приводят к цепной инвалидации объектов. Например, если тело поставляемого Oracle пакета перекомпилируется, используя его новую реализацию для исправления бага, то спецификация пакета останется валидной и, следовательно, валидными останутся все зависящие от него объекты. С введением гранулярного (fine-grained) отслеживания зависимостей в Oracle Database 11g даже изменение спецификации поставляемого Oracle пакета, вызванное добавлением новой подпрограммы, которая не переопределяет существующую подпрограмму, не приводит к цепной инвалидации. Когда такой патч применяется к не-CDB базе

данных, ни один созданный пользователем объект не меняется. Из этого следует, что даже для исправлений такого рода не понадобится как-либо менять подсоединяемую PDB[12]. При этом даже для исправлений, приводящих к цепной инвалидации, объем необходимой компенсирующей работы в PDB будет невелик.

Чтобы воспользоваться новой парадигмой подключения/отключения для максимально эффективного патчирования версии Oracle, исходная и целевая CDB должны размещаться на одной и той же файловой системе, чтобы не потребовалось перемещать файлы данных PDB. Эта схема показана на *рис. 7*.

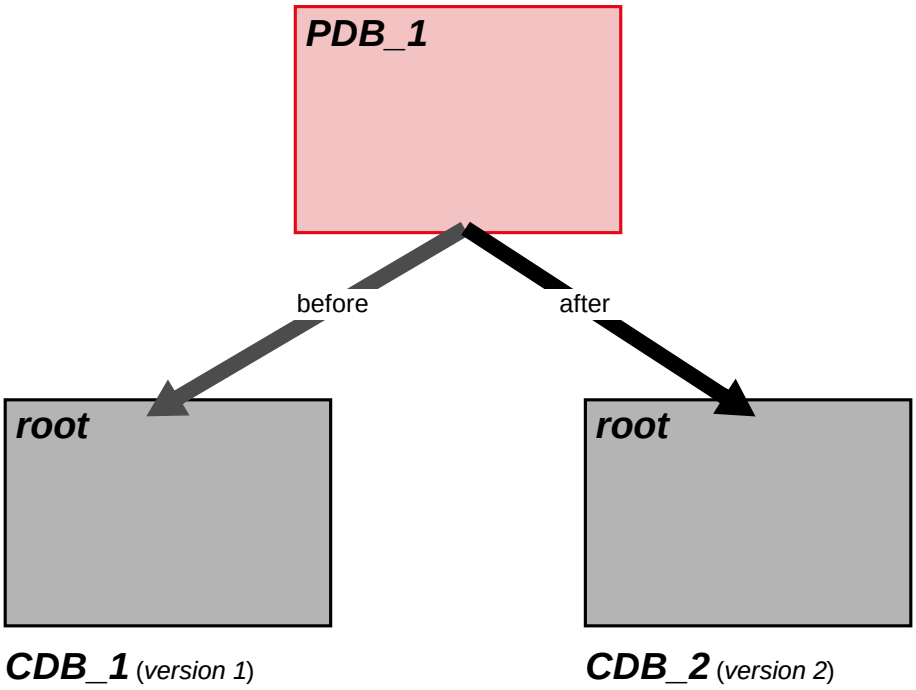

Рисунок 7. Отключение/подключение между двумя CDB базами данных с разными версиями ПО Oracle Database, при котором файлы данных остаются на месте в разделяемой файловой системе

Поскольку операции отключения и подключения затрагивают только метаданные в *root*, описывающие PDB, эти операции происходят очень быстро. Суммарная продолжительность закрытия PDB, ее отключения от исходной CDB, переподключения к целевой CDB и последующего открытия без переноса файлов данных занимает на типичной машине считанные секунды — при условии, что версии ПО Oracle Database совпадают. Можно заключить, что продолжительность будет той же, если разница в версиях относится только к изменению двоичных файлов Oracle, и лишь немного большей, когда разница обусловлена изменениями словаря данных, которые не приведут к инвалидации каких-либо пользовательских артефактов. Даже для патчей, которые вызывают инвалидацию части пользовательских артефактов, гранулярнрое отслеживание зависимостей

[12] Поскольку в данной реализации строка *Obj$* в PDB используется для указания на соответствующую строку *Obj$* в *root* (как описано в подразделе *«Некоторые детали новой реализации словаря данных»* на *стр. 16*), а строка в базе PDB кодирует сигнатуру парной ей строки в *root*, строка в PDB должна быть заново вычислена. Кроме того, таблица *Dependency$* в подключаемой базе данных полностью заполнена. Ее семантический контент для системных артефактов Oracle совпадает с содержимым ее парных строк в *root*, но в ней используются локальные значения номеров объектов для родительских и зависимых объектов в зависимостях. В ней записываются детали гранулярных зависимостей, такие как временные отметки компиляции родительского объекта зависимости, а также кодированное представление атрибутов родительского объекта, от которых зависит зависимый объект. Из этого можно сделать вывод, что PDB не может выражать свои зависимости от объектов в *root* с использованием номеров объектов, так как последние могут отличаться в разных CDB базах данных. Связь между строкой *Obj$* в подключаемой базе данных (PDB) и парной ей строкой в *root* реализуется через мапирование — а без него быстрое отключение и подключение не работало бы. Поэтому после подключения PDB, если исходная и целевая CDB базы данных отличаются наличием патча такого типа, потребуется также пересчитать детали гранулярных зависимостей и, возможно, некоторые строки в таблице *Dependency$* подключаемой базы данных. Эта операция выполняется очень быстро.

поможет минимизировать объем рекомпилирования, необходимого в присоединяемой PDB. Обратите внимание, что наши самостоятельно принятые правила совместимости гарантируют: изменения, сделанные в результате патчирования версии Oracle (или ее апгрейда), никогда не оставят пользовательский объект, который был валидным до внесения этих изменений, в необратимо инвалидном состоянии. Рекомпиляция всегда вернет эти каскадно инвалидированные объекты в валидное состояние.

Апгрейды версии ПО Oracle Database могут существенно изменять словарь данных, вызывая цепь изменений пользовательских артефактов. (В этом контексте термин апгрейд означает переход с *«точка один»* релиза на релиз *«точка два»*, например с 12.1 на 12.2, или переход с релиза *«точка два»* на следующий *«точка один»* релиз, например с 12.2 на 13.1). Если к не-CDB базе данных применяется апргрейд такого рода, то скрипты сначала обновляют систему Oracle, а затем пользовательские артефакты. Из этого следует, что, когда исходная и целевая CDB источника отличаются наличием такого апгрейда, в присоединяемой PDB необходимо произвести те же определенные в скриптах изменения, что следуют за изменениями системы Oracle в не-CDB базе данных. Время, необходимое, чтобы сделать требуемые изменения в присоедняемой PDB, следовательно, неизбежно меньше, чем в случае не-CDB базы данных, т. к. не требуется менять системные объекты Oracle.

Другими словами, если серверная часть приложения размещена в подключаемой БД, то время простоя приложения, вызванного патчированием версии Oracle при подходе «отключение/подключение», *всегда* меньше времени применения того же патча к не-CDB базе данных. Даже применение патчей, вызывающих ограниченную каскадную инвалидацию в PDB, могут занимать считанные секунды. Если обработка, связанная с патчированием, в присоединяемой PDB гарантированно не вызывает инвалидации пользовательских артефактов, то патчирование может быть выполнено неявно. В противном случае пользователю выдается уведомление о том, что нужно запустить поставляемый Oracle скрипт.

Также нужно отметить, что затраты на подготовку пропатченной целевой CDB эффективно окупаются. Они делаются лишь один раз, зато мы получаем выигрыш от автоматического патчирования версии Oracle для каждой из многих PDB баз данных, когда они отключаются/подключаются с одной версии ПО Oracle на другую.

Возможность отключения и подключения, когда целевая CDB относится к более ранней версии ПО Oracle Database по сравнению с CDB-источником, определяется в точности теми же правилами, что и возможность отката патча или понижения версии для не-CDB. (Существует негласное правило, согласно которому значение параметра *compatible* должно быть равен его значению в предыдущей версии базы данных. К тому же ни один из пользовательских артефактов не должен семантически зависеть от функциональности, обеспечиваемой лишь в более новой версии.) С нашей точки зрения, *возврат к предыдущему состоянию версии Oracle* обозначает как откат патча, так и понижение версии ПО Oracle Database. Выше показано, что факторы, которые необходимо учитывать при возврате к предыдущей версии Oracle, такие же, что и при патчировании версии Oracle.

Наконец, в этом разделе будет полезно рассмотреть установку исправлений версии Oracle для всей CDB, содержащей много PDB баз данных, как одну операцию. В наименее агрессивной части спектра изменений находятся изменения, которые меняют только root. Поэтому продолжительность патчирования n серверных частей приложений, когда каждая из них размещена в собственной PDB, а патчируется вся CDB, будет примерно равна времени установки исправлений для одной серверной части приложения, размещенной в не-CDB базе данных. Другими словами, мультиарендная архитектура позволяет установить исправления версии Oracle для n серверных частей приложений с затратами, аналогичными затратам на установку исправлений версии Oracle для одной серверной части приложения в не-CDB архитектуре. Мы рассчитываем, что этот подход станет предпочтительным в средах разработки и тестирования, где ото всех пользователей требуется одномоментно перейти на новую версию и в дальнейшем использовать только ее. Также мы ожидаем, что подход «подключение/отключение» будет предпочтительным в производственных средах, где сертификация каждой из консолидированных серверных частей приложений для новой версии Oracle Database будет происходить по собственному графику.

В разделе *«Удаленное клонирование с репликацией GoldenGate как альтернатива отключению/подключению»* на *стр. 24* обсуждается подход, эффективно реализующий *«горячее» отключение/подключение* (без прерывания работы).

Метод отключения и подключения для реагирования на изменения SLA

Новая парадигма отключения и подключения также очень эффективна для быстрого реагирования на изменения SLA. Опять-таки исходная и целевая базы CDB должны использовать общую файловую систему, чтобы файлы данных PDB оставались на своем месте.

Поэтому мы преполагаем, что клиенты будут создавать на одной платформе несколько CDB баз данных с разными версиями ПО Oracle Database, которые будут конфигурировать под различные уровни SLA.

Клонирование подключаемой базы данных

Отключенная база PDB — это полное, но скрытое представление всего того, что определяет серверное приложение, которое обычно используется и поэтому содержит данные, потребляющие квоту. Если требуется идентичная копия (*клон*) подобного серверного приложения, когда оно уже находится в эксплуатации, можно отключить PDB, скопировать все файлы при помощи методов операционной системы, снова подключить исходные файлы и клонированные файлы под другим именем и, возможно, к другой базе CDB[13]. Следующий мысленный эксперимент помогает понять, как работает SQL-команда *клонирования PDB*. Не отключая исходную базу PDB, пользовательский процесс Oracle копирует ее файлы и подключает их в качестве клона. Это показано на *рис. 8*.

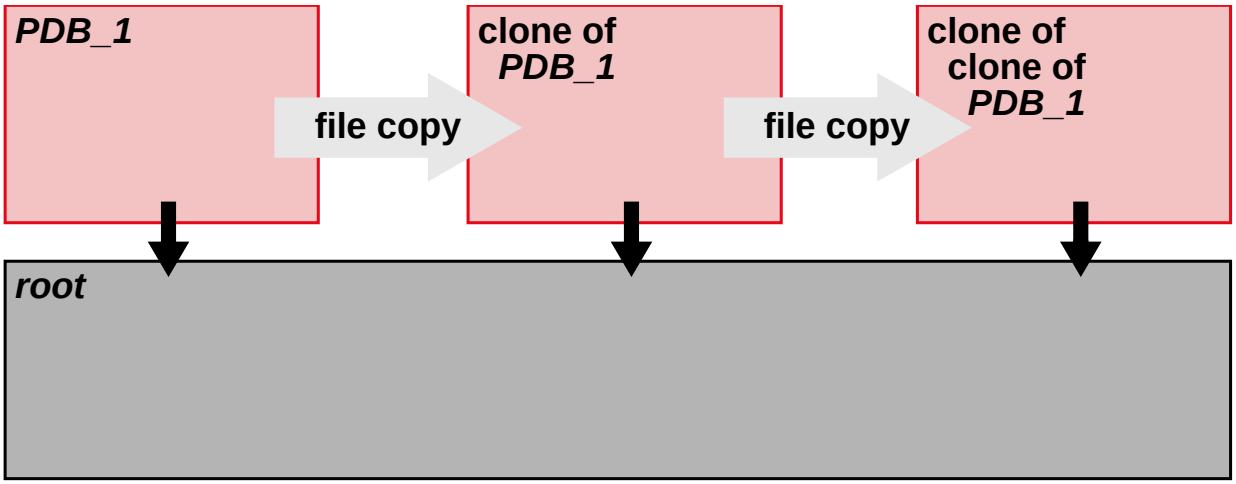

Рис. 8. Использование пользовательского (foreground) процесса для копирования файлов данных PDB, чтобы создать клон

Пользовательский процесс назначает новый, глобально уникальный идентификатор для клонированной базы PDB. Клон записывает глобально уникальный идентификатор базы PDB, из которой он был создан, и, конечно, родословная этой базы данных может предсталять собой цепочку баз данных произвольной длины..

Мультиарендная архитектура позволяет создать клонированную базу PDB в той же базе CDB, что и исходная PDB, или в другой CDB, если есть соединение Oracle Net между двумя базами CDB.

---

[13] Строго говоря, это не совсем верно. SQL-команда *create PDB* назначает подключаемой базе данных глобально уникальный идентификатор, который передается с ней во всех операциях отключения и подключения. Конечно, две PDB не должны иметь один и тот же глобально уникальный идентификатор. Поэтому нельзя вручную копировать файлы отключенной PDB. Если требуется клонирование, используйте SQL-команду *clone PDB*. (Она также позволяет создать клон из отключенной базы PDB, см. раздел *«Клонирование из отключенной базы PDB»* на *стр. 24*.) А если вам нужна копия для подстраховки, используйте резервное копирование RMAN или Data Guard.

Чтобы различать эти варианты, будем использовать термины *локальное клонирование* и *удаленное клонирование*.

Клонирование PDB с использованием полной копии

Поскольку СУБД Oracle создает копии файлов, для уменьшения общего времени выполнения задачи можно задействовать параллельные серверные процессы Oracle. Кроме того, копируемые файлы данных содержат блоки данных Oracle, и исполняемые файлы Oracle понимают их содержимое. Следовательно, степень параллелизма не ограничивается ни числом файлов данных, ни даже числом крупных единиц выделения пространства в этих файлах, которые может понять операционная система. Она ограничивается только числом блоков данных Oracle в наборе файлов данных. Это число столь велико, что фактически степень параллелизма ограничена лишь количеством параллельных серверных процессов Oracle, которые можно выделить для операции *clone PDB* с учетом других операций, одновременно выполняемых на компьютере.

В мультиарендной архитектуре операцию *clone PDB* можно выполнить, не замораживая исходную PDB, но такая возможность не поддерживается в версии 12.1.0.1. (При попытке выполнить эту операцию появляется сообщение *ORA-65081: database or pluggable database is not open in read only mode*.) Проблема аналогична той, что встречается при онлайн архивировании RMAN. Во время копирования набора файлов данных, начатого в момент *t*, некоторые блоки данных Oracle в этих файлах изменяются. Решение заключается в том, чтобы отследить, изменился ли копируемый блок с момента *t*, и, если да, перестроить его по состоянию на момент *t* с помощью информации undo. Конечно, подход RMAN можно приспособить для операции *clone PDB*. Мы будем называть *клонирование PDB* без замораживания исходной PDB *«горячим» клонированием*[14].

Клонирование PDB с использованием снапшот-копии

Уже на протяжении определенного времени, поставщики файловых систем, такие как корпорация Oracle (с Sun ZFS) или NetApp, предлагают метод для создания копии файла произвольного размера всего за несколько секунд. В основе этого метода лежит общая идея, применимая ко многим другим структурам (помимо файлов), — *копирование при записи*[15]. Кратко говоря, преимущество обозначенной идеи при копировании файла в том, что доступ к блокам осуществляется через отдельную запись, с которой связано имя файла и которая содержит указатели на блоки файла. Копируются не все блоки, а только перечень указателей, которому присваивается имя нового файла. Указатели на каждый блок в файле содержатся в перечнях указателей исходного файла и его копии. При этом используется небольшое пространство на диске, так что копирование выполняется почти мгновенно. Только при попытке записи в блок одного из файлов создается вторая копия блока, чтобы два файла различались этим блоком. Отсюда и название данного метода.

Когда таким образом создается копия большого файла и затем вносятся изменения лишь в небольшую часть его блоков, достигается огромная экономия времени и аппаратных ресурсов. По этой причине этот подход очень часто применяется при создании клона для целей разработки или тестирования. Обычно в таком сценарии через относительно короткое время файл удаляется, когда большинство его блоков не успевают измениться.

Поставщики файловых систем предпочитают использовать термин снапшот-копия.
А бизнес-преимущество, которое она дает, обычно называется тонким провизионированием.

---

[14] Мы специально уделяем внимание теме «горячего» клонирования в публикации. Важно понимать, что поддержка «горячего» клонирования и поддержка онлайн резервного копирования RMAN — это две разные задачи. Oracle Database 12.2 будет поддерживать «горячее» клонирование.

[15] Общие принципы описаны в следующей статье Википедии: *http://ru.wikipedia.org/wiki/Copy-on-write*.

Заказчики используют снапшот-копию, чтобы создавать тонкую архивную RMAN-копию для не-CDB базы данных и создавать из нее клоны не-CDB. (Для клонированной не-CDB БД должен быть создан новый, глобально уникальный идентификатор.) Однако процесс в целом включает разные шаги, которые выполняются в различных средах и требуют разных полномочий. Обычно нескольким специалистам с индивидуальными паролями приходится согласовывать и синхронизировать действия из-за традиционного разделения обязанностей между администраторами баз данных и систем хранения. Такое взаимодействие не всегда легко организовать.

В мультиарендной архитектуре пользовательский процесс отправляет запрос системе хранения, чтобы скопировать файлы данных исходной базы PDB по отдельному защищенному протоколу. Каждая отдельная база CDB записывает имя пользователя и пароль администратора системы хранения (для файлов данных CDB) в электронный кошелек.

Клонирование PDB с использованием снапшота поддерживается, только когда новая база PDB создается в той же CDB, где находится PDB — основа для клонирования. Пока существует PDB, созданная с помощью тонкого провизионирования, базу PDB, из которой она была клонирована, удалить нельзя. Базу PDB, созданную с помощью снапшот копирования, нельзя отключить. И исходную PDB, и ее клон можно открыть в режиме чтение-запись.

В версии СУБД 12.1.0.1 снапшот-клонирование PDB поддерживают следующие файловые системы: Sun ZFS, NetApp и Oracle ACFS. При попытке использовать снапшот-клонирование PDB в любой другой файловой системе выдается сообщение об ошибке (*ORA-17517: Database cloning using storage snapshot failed*).

Клонирование из отключенной базы PDB

Отключенную базу PDB можно использовать как очень удобное средство предоставления всей серверной части приложения, полностью заменяющее прежние методы, в которых применялись различные инструменты (например, SQL*Plus, Data Pump и т. п.). В этом сценарии использования было бы неправильно просто подключить отключенную базу PDB в несколько разных CDB баз данных, поскольку каждая PDB, созданная таким образом, имела бы один и тот же глобально уникальный идентификатор. Команда *plug in PDB* позволяет создать новую базу PDB как клон отключенной PDB, чтобы ей был назначен собственный, глобально уникальный идентификатор. В *Code_6* показана соответствующая команда SQL.

Как вариант, закачик может создать в одном месте набор эталонных отключенных баз PDB, которые будут представлять полезные стартовые точки, например, для стрессового тестирования или отладки логики приложения.

Удаленное клонирование с репликацией GoldenGate как альтернатива отключению/подключению

Независимо от поддержки «горячего» клонирования, вместо отключения и подключения можно использовать удаленное клонирование для создания новой базы PDB в CDB базе данных, при котором не возникнет простоя приложения, чей серверный компонент находится в перемещаемой базе PDB. Для этого можно использовать репликацию GoldenGate, записав системный номер изменений (SCN), с которым было начато «горячее» клонирование, чтобы синхронизировать состояние клонированной базы PDB и потом продолжать отслеживание ее изменений. Стадия постоянного отслеживания изменений аналогична стадии, которая используется во время скользящего обновления версии ПО Oracle Database с помощью временной логической резервной базы данных. Конечно, процедура переключения клиентской части приложения на новую базу данных будет аналогичной. Другими словами, удаленное клонирование PDB между исходной и целевой CDB базами данных с разными версиями ПО Oracle Database является естественным эквивалентом традиционного метода использования временной логической резервной базы данных для не-CDB базы.

Синтаксис SQL-команды *clone PDB*

В *Code_3* показана базовая SQL-команда *clone PDB*.

```
-- Code_3
create pluggable database PDB_2 from PDB_1
```

Семантика команды: «создать клон *PDB_1* и назвать его *PDB_2*».

В *Code_4* показана SQL-команда *clone PDB* для удаленного клонирования.

```
-- Code_4
create pluggable database PDB_2 from PDB_1@CDB_02_link
```

Семантика команды: «создать клон базы *PDB_1*, содержащейся в CDB, *root* которой указывается через *CDB_02_link*, и назвать его *PDB_2*». (В качестве альтернативы линк баз данных может указывать на клонируемую базу PDB.)

Линк базы данных содержит спецификацию адреса исходной CDB (имя и порт слушателя, а также имя сервиса для доступа к БД и ее файлам). Он также определяет полномочия, чтобы стартовать сессию, текущий контейнер которой является *root* исходной CDB. Когда расположение и полномочия установлены, для переноса файлов используется подходящий протокол нижнего уровня, а сам процесс переноса выполняется в параллельном режиме с помощью параллельных серверных процессов Oracle. При этом файлы не передаются через линк базы данных.

В *Code_5* показана SQL-команда *clone PDB* с использованием снапшот-копии.

```
-- Code_5
create pluggable database PDB_2 from PDB_1 snapshot copy
```

Семантика команды: «создать клон *PDB_1*, используя возможности тонкого провизионирования нижележащей файловой системы, и назвать его *PDB_2*».

В *Code_6* показана SQL-команда *clone PDB* клонирования из отключенной базы PDB.

```
-- Code_6
create pluggable database PDB_2 as clone
using '/u01/app/oracle/oradata/…/pdb_1.xml'
copy
```

Семантика команды: «создать клон отключенной базы *PDB_1* и назвать его *PDB_2*, скопировав файлы данных отключенной PDB в новый набор файлов, которые можно будет изменять независимо от источника».

Три команды *create PDB*, *plug in PDB* и *clone PDB* представляют собой варианты SQL-команды *create pluggable database*. Поэтому каждая команда допускает использование выражения *path_prefix*.

Создание подключаемой базы данных

Вместо построения таблиц словаря данных, определяющих пустую базу PDB с нуля, и заполнения таблиц *Obj$* и *Dependency$* CDB создается вместе с пустой PDB внутри. (Здесь слово *пустая* означает, что база данных не содержит артефактов, созданных пользователем.) Эта БД называется *seed PDB* и имеет имя *PDB$Seed*. Каждая контейнерная база обязательно содержит *seed PDB*, и она всегда открывается в режиме только для чтения. Это условие не имеет принципиального значения, а является, скорее, средством оптимизации. Операция *create PDB* реализована как специальный случай операции *clone PDB*. Размер *seed PDB* составляет лишь 1 Гбайт, и ее копирование на обычном компьютере занимает всего несколько секунд. Использование снапшот PDB клонирования недоступно для команды *create PDB*.

Синтаксис SQL-команды *create PDB*

В *Code_7* показана базовая SQL-команда *create PDB*.

```
-- Code_7
create pluggable database PDB_1
path_prefix = '/u01/app/oracle/oradata/pdb_2_Dir'
admin user PDB_1_Admin identified by p
roles = (DBA, Sysoper)
```

Семантика команды: «создать совершенно новую базу PDB с именем *PDB_1*; убедиться, что любые объекты типа директория, созданные в PDB, ограничены заданным поддеревом в файловой системе, которую использует CDB; создать локального пользователя [16] в новой базе PDB с именем *PDB_1_Admin* в качестве администратора и предоставить ему данный перечень ролей».

Удаление подключаемой базы данных

Перед удалением базу PDB необходимо закрыть[17].

Существует два сценария для использования команды *drop PDB*. Первый сценарий очевиден: база PDB больше не нужна. Это распространенная ситуация в непроизводственной среде, где PDB — начальная точка для конкретной задачи разработки либо объект исследования — многократно клонируется и удаляется. Будем называть такой тип удаления деструктивным. Он показан в *Code_8*.

Второй сценарий использования не столь очевиден. После выполнения команды *unplug PDB* сведения об отключенной PDB остаются в метаданных *root*, а ее файлы данных по-прежнему указаны в контрольном файле CDB. Это значит, что можно использовать резервное копирование RMAN, чтобы сохранить состояние PDB на момент ее отключения. Независимо от того, будет ли делаться резервная копия, после команды *unplug PDB* всегда должна следовать команда *drop PDB* — либо сразу (если резервное копирование не будет делаться), либо после успешного завершения резервного копирования. Конечно, смысл команды *plug in PDB* подразумевает, что файлы данных должны сохраняться. Будем называть такой тип удаления недеструктивным. Он показан в *Code_9*.

Синтаксис SQL-команды *drop PDB*

В *Code_8* показана деструктивная SQL-команда *drop PDB*.

```
-- Code_8
drop pluggable database PDB_1 including datafiles
```

Семантика команды: «удалить базу PDB с именем *PDB_1* и полностью удалить все ее файлы данных». Конечно, эту команду следует использовать с осторожностью.

---

[16] Определение термина локальный пользователь будет дано в разделе «Динамические аспекты мультиарендной архитектуры: экземпляр Oracle, пользователи и сессии» на стр. 31.

[17] Столбец *gv$PDBs.Open_Mode* принимает следующие три допустимых значения: MOUNTED, READ ONLY и READ WRITE. Пояснения см. в разделе «Логическая виртуализация SGA» на *стр. 34*.

В *Code_9* показана недеструктивная SQL-команда *drop PDB*.

```
-- Code_9
drop pluggable database PDB_1 keep datafiles
```

Семантика команды: «удалить базу PDB с именем *PDB_1*, но сохранить все ее файлы данных».

Почему важно, что *create PDB*, *clone PDB*, *drop PDB* и unplug/plug являются SQL-командами

Команды *create PDB* и *drop PDB* функционально эквивалентны использованию утилиты *dbca* для создания или удаления не-CDB БД. Однако *dbca* можно использовать только путем входа в систему на компьютере, где установлен соответствующий каталог *Oracle Home*, с правами пользователя операционной системы, который является владельцем установленного ПО, а значит, и файлов любых других баз данных, созданных этой инсталляцией ПО. Следовательно, только лица, пользующиеся особым доверием, могут выполнять эти задачи провизионирования. Операция *clone PDB* функционально эквивалентна использованию для не-CDB базы данных сложной последовательности шагов, которая обычно включает создание резервной копии RMAN и последующее восстановление из нее не-CDB базы данных. Для этого также требуется вход с правами пользователя операционной системы, который является владельцем каталога *Oracle Home*. Для не-CDB базы ближайшими функциональными эквивалентами команд *unplug* и *plug in PDB* являются полный экспорт и полный импорт базы данных с помощью Data Pump. Здесь также требуются полномочия владельца каталога *Oracle Home* в операционной системе для копирования и перемещения файлов дампа.

Напомним, что команды *unplug PDB* и *plug in PDB* могут быть использованы для патчирования версии ПО Oracle для PDB. Эта операция для не-CDB базы данных также требует полномочий владельца каталога *Oracle Home* в операционной системе.

В то же время SQL-команды, реализующие операции *create PDB*, *clone PDB*, *unplug PDB*, *plug in PDB* и *drop PDB,* могут быть выполнены с клиентского компьютера. Для них требуются только соответствующие полномочия пользователей Oracle Database. Эти SQL-команды являются прямыми атомарными выражениями семантических требований предполагаемой операции. Более того, кроме случаев, когда создается полная физическая копия файлов данных или выполняется их перенос, выполнение SQL-команд может занимать несколько секунд, в том числе при клонировании PDB с помощью снапшотов.

Достаточно просто PL/SQL может быть использован для автоматизации, например, для операций закрытия PDB, *unplug PDB* и последующего недеструктивного *drop PDB*, *plug in PDB* и последующего открытия PDB. Такие средства, как SQL Developer и Enterprise Manager[18], напрямую предоставляют функциональность SQL-команд по провизионированию PDB.

Результат — весьма эффективная новая парадигма для провизионирования баз данных.

Отключение/подключение и *clone PDB* для CDB баз данных, защищенных Data Guard

Резервная база данных будет автоматически реагировать на операции *create PDB* и *clone PDB* в основной базе данных. Но для *plug in PDB* в основной базе данных ее файлы данных нужно будет сделать доступными для резервной, так же, как в случае с файлами данных для переносимых табличных

[18] Enterprise Manager предоставляет возможности провизионирования для не-CDB баз данных версии до 12.1, но для этого требуются установка и использование агентов на стороне сервера и настройка авторизации в операционной системе. Реализация соответствующего функционала для PDB баз данных намного проще, и, как уже пояснялось, операции с PDB выполняются намного быстрее, чем с не-CDB базами данных.

пространств. Аналогичным образом после операции *drop PDB* с *keep datafiles*, которая выполняется после *unplug PDB*, файлы данных отключенной PDB базы необходимо вручную удалить из резервной среды.

Присоединение не-CDB базы данных в качестве PDB

Возможны два подхода: прямое присоединение не-CDB БД версии 12.1 в качестве PDB и перенос контента не-CDB БД в пустую базу PDB, например, с помощью Data Pump.

Прямое присоединение не-CDB базы данных версии 12.1 в качестве PDB

Oracle Database 12*c* поддерживает как новую мультиарендную архитектуру, так и прежнюю не-CDB архитектуру. Не-CDB базу данных версии до 12.1 нельзя напрямую присоединить в качестве PDB. Ее надо обновлять на месте обычным способом. Метод прямого присоединения очень прост как концептуально, так и практически. Не-CDB база данных просто останавливается, а затем используется как отключенная PDB. Напомним, что отключенная PDB имеет внешний манифест, созданный в результате операции *unplug PDB*. Поэтому, прежде чем остановить присоединяемую не-CDB БД, нужно перевести ее в режим «только для чтения» и затем выполнить процедуру *DBMS_PDB.Describe()*, указав путь для манифеста в качестве единственного аргумента, так же, как он указывается при использовании команды *unplug PDB*.

После остановки присоединяемой не-CDB БД логично сделать ее резервную копию с помощью RMAN.

С файлами данных не-CDB базы и созданным вручную манифестом теперь можно обращаться так, как если бы они были обычной отключенной PDB, которая до этого была подключена к CDB и открыта как PDB. Однако, как скоро будет объяснено, она должна быть открыта со статусом *Restricted,* установленным в `YES`. Табличные пространства, которые содержат данные, потребляющие квоту, немедленно становятся доступными, как если бы был выполнен импорт Data Pump с помощью переносимых табличных пространств. Однако словарь данных прежней не-CDB БД продолжает содержать полное описание системы Oracle. Теперь он не нужен, и его необходимо просто удалить. Когда статус *Restricted* все еще имеет значение `YES`, нужно выполнить скрипт *noncdb_to_pdb.sql* (содержится в каталоге *rdbms/admin* в *Oracle Home*). Теперь базу PDB можно закрыть и потом открыть как обычно.

Мы будем называть такой метод присоединения как *присоединение посредством обновления и подключения.*

Перенос контента не-CDB базы данных

Подходы, описанные в этом разделе, поддерживаются благодаря гарантии PDB/не-CDB совместимости. Полный экспорт и импорт базы данных с помощью Data Pump можно использовать для переноса большинства артефактов, созданных пользователем, из не-CDB базы в новую PDB. В версии СУБД 12.1 впервые решение Data Pump поддерживает полный экспорт и импорт баз данных. Это обеспечивает максимальную эффективность использования технологии перемещаемых табличных пространств с помощью одной команды. Такая способность называется *Full Transportable Export/Import*. Это усовершенствование было также перенесено в версию 11.2.0.3. Мы будем называть подобный метод *присоединение с помощью Data Pump*. Однако некоторые артефакты базы данных, например схемы XML, директории и привилегии, назначенные *public*, нельзя перенести с помощью Data Pump. Они могут быть смигрированы с помощью *специальных* методов.

Следует отметить, что, когда размер присоединяемой не-CDB БД относительно мал и весь контент можно обработать с помощью Data Pump, подход к присоединению с помощью Data Pump может оказаться быстрее, чем присоединение посредством обновления и подключения. Кроме того, процесс консолидации часто включает перенос серверных приложений со старого оборудования на современное. Это может означать перенос между средами с разным порядком следования байтов. Как объясняется в разделе *«Отключение и подключение между архитектурами с разным порядком следования байтов» на стр. 18*, подход к присоединению с помощью Data Pump — это единственно возможный вариант в этом сценарии.

В качестве альтернативы можно использовать репликацию GoldenGate для заполнения новой базы PDB и последующего отслеживания текущих изменений в исходной не-CDB БД. Затем можно, ненадолго частично прервав работу, код клиентской части приложения переключить с не-CDB БД на новую PDB, как это делается при патчировании версии ПО Oracle не-CDB базы данных с помощью временной логической резервной базы данных.

Динамические аспекты мультиарендной архитектуры: Oracle экземпляр, пользователи и сессии

В этом разделе рассматриваются последствия открытия всей CDB базы данных RAC-экземпляром.

Общность пользователей, ролей и прав

В CDB различаются *local users* и *common users (*локальные и общие пользователи). Наличие двух типов пользователей неизбежно следует из гарантии PDB/не-CDB совместимости.
Она требует выполнения двух условий:

- Каждая подключаемая БД в одной и той же CDB базе может содержать пользователя, созданного клиентом, с одним и тем же именем (например, *Scott*). Эти пользователи являются независимыми, и их имена совпадают по чистой случайности.

- Пользователи, поставляемые с СУБД Oracle, такие как *Sys* и *System*, должны присутствовать в каждом контейнере, так как они имеют особые, документированные назначение и характеристиками.

Локальные пользователи и локальные роли

Как пояснялось в разделе *«Горизонтальное секционирование словаря данных в мультиарендной архитектуре»* на *стр. 14*, каждая база PDB в определенной CDB БД может содержать пользователя с именем *Scott*. Такой пользователь является *локальным*, он определен только в таблице *User$* словаря данных базы PDB и, следовательно, распознается лишь в этой базе. (Разумеется, пользователь *Scott* в каждой базе PDB может иметь разные пароли.) В противном случае гарантия PDB/не-CDB совместимости не будет соблюдена. Соответственно, *local role* (локальная роль) определена только в конкретной базе PDB.

Ни локального пользователя, ни локальную роль нельзя создать в *root*.

Общие пользователи и общие роли

С другой стороны, сессия, использующая PDB в конкретной базе CDB, должна видеть тот же набор артефактов, поставляемых с СУБД Oracle, что и сессия, использующая любую другую базу PDB в этой CDB. Конечно, эти артефакты включают в себя пользователей типа *Sys* и *System* и роли, такие как *DBA* и *Select_Catalog_Role*. *Sys* и *System* — это *common users* (общие пользователи), а *DBA* и *Select_Catalog_Role - common roles* (общие роли). Имя и пароль общего пользователя определяются в *root*, но при этом поддерживается инвариантность, так чтобы идентификационные данные были известны в каждой PDB этой CDB, существующей и будущей, с одинаковыми именем и паролем. Соответственно, *общая роль* определяется в *root* и известна повсеместно.

По той же причине объекты типа *Sys.DBMS_Output* и *System.Sqlplus_Product_Profile* должны также присутствовать в каждом контейнере и иметь одно определение. Эти объекты называются *common jbjects (общими объектами)* и определяются в *root*[19]. Клиенты не могут создавать общие объекты.

[19] Общие объекты представлены в словаре PDB одной строкой в таблице *Obj$*. Эта строка соответствует по имени (*Owner*, *Object_Name* и *Namespace*) соответствующей строке в таблице *Obj$* в *CDB$Root*. Данный механизм обеспечивает работоспособность PDB сразу после отключения и подключения между базами CDB, хотя ее граф зависимостей в таблице *Dependency$* включает и общие объекты *CDB$Root*, и локальные объекты PDB, а факты в таблице *Dependency$* записываются с помощью значений *Object_ID*.

Общее предоставление привилегий и общие роли

Эффект предоставления привилегии или роли, когда текущим контейнером является база PDB, представлен в таблицах словаря данных *SysAuth$* и *ObjAuth$* в этой базе PDB. В результате действие назначенной привилегии ограничивается базой PDB, где она была предоставлена. Это справедливо даже в том случае, когда получателем привилегии является общий пользователь или общая роль. Следовательно, определенный общий пользователь или общая роль может получить разные права в каждой базе PDB. Однако для соблюдения гарантии PDB/не-CDB совместимости поставляемые с СУБД Oracle общие пользователи, такие как *Sys* и *System*, и общие роли типа *DBA* и *Select_Catalog_Role* должны иметь строго одинаковый набор прав в каждой базе PDB в этой CDB, настоящей и будущей. (Предполагается, что клиенты следуют рекомендуемым лучшим практикам и не назначают и не отзывают привилегии или роли у пользователей, поставляемых с СУБД Oracle. Такая лучшая практика не применяется к общим пользователям и общим ролям, созданным клиентом). Это условие должно формально поддерживаться, иначе привилегия или роль может быть отозвана в конкретной базе PDB и гарантия будет нарушена. Формальная поддержка реализована в виде особой разновидности SQL-команды *grant*. *Code_10* содержит пример.

```
-- Code_10
grant Create Session to A_Common_User continer = all
```

Семантика команды: «гарантировать, чтобы данная привилегия/роль оставалась назначенной получателю в каждом контейнере, настоящем и будущем». При общем предоставлении или отзыве привилегии либо общей роли текущим контейнером должен быть *root*. Логика команды требует, чтобы получатель привилегии должен быть известен повсеместно, являться общим пользователем или общей ролью. Предоставляемая привилегия/роль должна быть также известна повсеместно. Следовательно, это должна быть системная или объектная привилегия (повсеместно известная в силу того, что она является поставляемой Oracle) либо общая роль.

Общие пользователи и общие роли, созданные клиентом

Сценарий для общего пользователя, созданного клиентом – это способ авторизации для лица, который должен выполнять административные задачи для CDB в целом либо для двух или более баз PDB, причем в части имени, пароля и аудита пользователь должен быть единой сущностью.

Этого требует устоявшаяся лучшая практика для версии СУБД до 12.1. Она заключается в блокировке и завершении срока действия всех пользователей, поставляемых с СУБД Oracle, и применении вместо них пользователей, созданных клиентом, чтобы специалисты могли выполнять все необходимые административные задачи. В CDB такая лучшая практика основана на общих пользователях, создаваемых клиентом. В версиях СУБД до 12.1 такие пользователи, как правило, не являются владельцами объектов. Тем не менее эти пользователи могут стать владельцами объектов, представляющих собой, например, процедуры для безопасного выполнения определенных административных задач. Такие объекты не считаются частью реализации серверного приложения. Эта концепция предполагает, что в CDB общий пользователь, созданный клиентом, может быть владельцем локальных объектов в *root*. Общему пользователю, созданному клиентом, не запрещено быть владельцем локальных объектов в PDB. Но это не рекомендуется. Владельцами локальных объектов в базах PDB должны быть локальные пользователи.

То же относится и к общим ролям, созданным клиентом.

Имя общего пользователя или общей роли, созданных клиентом, должно начинаться с *C##* или *c##*. В *Code_11* показан синтаксис для создания общего пользователя.

```
-- Code_11
create user c##u1 container = all identified by p
```

Выражение *container* = *all* также используется для создания общей роли.

Следует отметить, что системные привилегии, данные локальному пользователю, не действуют в схеме общего пользователя.

Сервисы и сессии

Чтобы создать сессию в не-CDB БД при помощи Oracle Net, необходимо представить следующие пять элементов информации: адрес слушателя, порт слушателя, имя сервиса, имя пользователя и пароль. Если первые два элемента идентифицируют действующего слушателя и этот слушатель поддерживает указанный сервис, запускается пользовательский процесс в одном из экземпляров RAC, где открыта нужная не-CDB база данных. Какое-то время пользовательский процесс работает как *Sys*. Затем происходит запрос таблицы *User$* для проверки того, что указанное имя пользователя и пароль действительны и у этого пользователя есть привилегия *Create Session*. Если имя пользователя и пароль действительны, этот пользователь становится пользователем сессии и клиент может начать отправлять вызовы базы данных. В противном случае пользовательский процесс прекращает работу, а клиент остается неподключенным.

Точно так же с одним небольшим добавлением создается сессия для работы с выбранной PDB в CDB базе данных. Главное, что и здесь следует представить те же пять элементов информации. Когда запускается пользовательский процесс, работающий как *Sys*, текущим контейнером является *root*. Он может видеть свойства сервиса, для которго предпринимается попытка создать сессию. Появившееся в версии 12.1 у сервиса свойство указывает PDB, которая после завершения авторизации будет текущим контейнером сессии. (Если это свойство равно *null*, это означает, что текущим контейнером для новой сессии будет *root*). Сессия затем переключается на назначенный контейнер и только после этого запрашивает *локальную таблицу User$* для проверки того, что указанное имя пользователя и пароль действительны и у этого пользователя есть *Create Session* привилегия. Таким образом, успешная попытка или сбой будут такими же, как и в случае не-CDB БД.

Следует отметить, что для успешного подключения заданный пользователь просто должен быть известен в контейнере, указанном в свойстве сервиса. Он может быть локальным пользователем или общим пользователем — это не имеет значения.

Изменение текущего контейнера для уже созданной сессии

Команда SQL *alter session set container* меняет текущий контейнер на указанный. Пример содержится в *Code_12*.

```
-- Code_12
alter session set container = PDB_2
```

Пользователь, выполняющий эту команду, должен быть известен в целевом контейнере и иметь там *Set Container* системную привилегию. Конечно, этот пользователь должен быть известен и в контейнере, где выполняется команда SQL. Это означает, что только общий пользователь может выполнить указанную команду без ошибки. Результатом успешного выполнения команды является разовый переход с одного контейнера на другой; история переходов не имеет значения и не поддерживается.

Вы не можете запустить транзакцию в контейнере, если существует незавершенная транзакция в другом контейнере.

Команда SQL *alter session set container* допустима только как серверный вызов верхнего уровня (top-level server call).

Новая реализация *DBMS_Sql.Parse()* , сделанная в версии 12.1, использует команду *alter session set container* внутри PL/SQL кода строго в режиме «батута» (trampoline). По этой причине мы называем подход, где используется этот механизм, «батутом» *DBMS_Sql.Parse trampoline*. Формальный параметр *Container* указывает, где будет разбираться одна SQL-команда.

Когда выполнение команды завершается, текущий контейнер автоматически сбрасывается в значение, существовавшее на момент вызова процедуры *DBMS_Sql.Execute()*. Это происходит и в случае, когда команда SQL, выполняемая удаленно, вызывает ошибку. Пример содержится в *Code_13*.

```
-- Code_13
begin
...окружающий контекст PL/SQL...
-- Например
Stmt := 'create table t(PK...)';
declare
Cur integer := DBMS_Sql.Open_Cursor(Security_Level=>2);
Dummy integer;
begin
Dbms_Sql.Parse(
c=>Cur,
Statement=>Stmt,
Container=>Con_Name,
Language_Flag=>DBMS_Sql.Native);
Dummy := DBMS_Sql.Execute(Cur);
DBMS_Sql.Close_Cursor(Cur);
end;
...окружающий контекст PL/SQL...
end;
```

DBMS_Sql.Parse trampoline допустим только в случае, если начальным контейнером является *CDB$Root*[20]. Эти правила гарантируют, что сеанс, созданный локальным пользователем, не может выйти из PDB, указанной в имени сервиса, которое использовалось при подключении. Это означает, что для локальных пользователей модель изоляции PDB в одной и той же CDB такая же, как между не-CDB базами данных, которые находятся на одной и той же платформе и принадлежат одному и тому же пользователю операционной системы[21].

Для администрирования CDB может быть удобно использовать команду *alter session set container* в ситуативных скриптах SQL*Plus, чтобы избежать повторного использования команды CONNECT и связанных с этим проблем безопасности паролей. Подход с использованием *DBMS_Sql.Parse()* может быть полезен для задач администрирования CDB, реализуемых с помощью PL/SQL.

Логическая виртуализация SGA

Консолидация внутри базы данных дает преимущество большей плотности консолидации. Это объясняется тем, что все серверные части приложений каждого экземпляра RAC используют одну и ту же область SGA. Если используется консолидация на основе схем, то область SGA заполняется элементами данных, которые хотя и охватывают все серверные приложения и систему Oracle, не помечены как относящиеся к тому или иному серверному приложению или системе. Определить принадлежность в каждом случае может только человек. Если используется консолидация на основе PDB, то каждый элемент (прежде всего блоки данных и такие структуры библиотечного кэша, как дочерние курсоры) помечается с указанием их происхождения. Иными словами, область SGA логически виртуализована. Это важно. Словарь данных виртуализируется *физически*, чтобы обеспечить возможность подключения, а область SGA виртуализируется *логически* для поддержки физических характеристик консолидации внутри базы данных. Это показано на *рис. 9, страница 35*.

[20] В остальном действуют обычные правила PL/SQL: код может быть любого из трех типов (анонимный блок, блок с правами вызывающего или блок с правами создателя), а проверка привилегий основана на текущем пользователе, и это все.

[21] Чтобы этот принцип четко соблюдался, невозможно прямо ссылаться на объект в одной PDB из другой PDB (команды SQL или части кода, определяющей объект). Мы специально решили не изобретать схему именования из трех частей (имя PDB, имя схемы, имя объекта), которая позволяла бы кросс-PDB разрешение имен. Если вы хотите сослаться на объект в другой PDB, необходимо использовать линк базы данных.

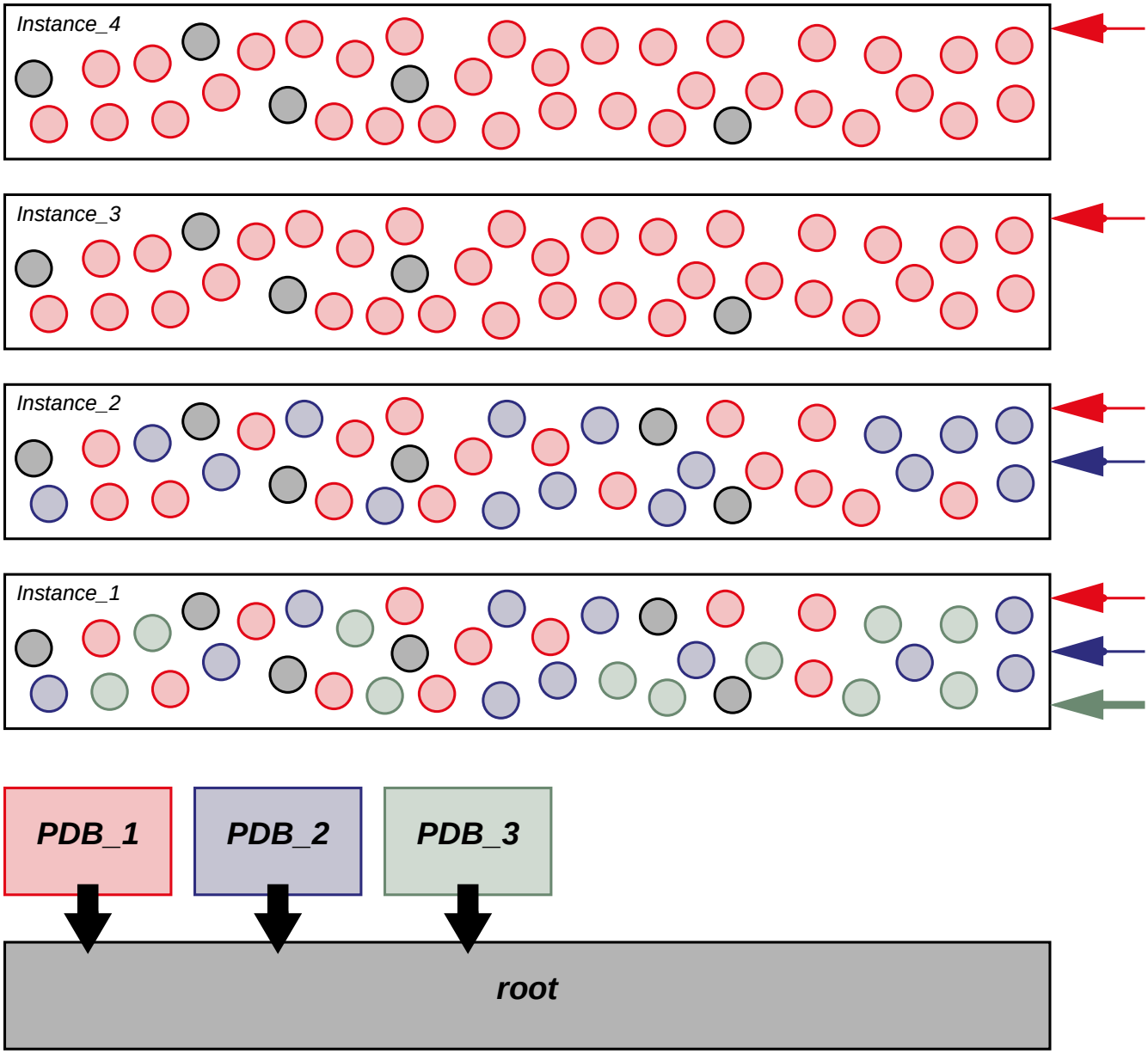

Рис. 9 Привязка PDB к RAC экземплярам. Стрелки справа окрашены в разные цвета, чтобы показать привязку каждой PDB к отдельным экземплярам RAC.

Некоторые клиенты следуют практике, использующей консолидацию на основе схем, когда сессии пользователей, работающие с конкретным серверным приложением, связываются с конкретным набором экземпляров RAC. Это обеспечивается следованием установленному самими клиентами порядку, когда для запуска сессий, назначенных конкретному серверному приложению, должен использоваться один или несколько определенных сервисов.

Такой порядок формально поддерживается в мультиарендной архитектуре. PDB может находиться в режиме `READ WRITE` (ЧТЕНИЕ ЗАПИСЬ), `READ ONLY` (ТОЛЬКО ЧТЕНИЕ) или `MOUNTED` (СМОНТИРОВАНА). Режим указывается в столбце *gv$PDBs.Open_Mode* системного представления и может быть явно задан и иметь разное значение для каждой PDB в каждом экземпляре RAC. Следует отметить, что режим `MOUNTED` — это максимальный режим неактивности, который может иметь PDB. Причина в том, что экземпляр RAC работает с CDB базой данных как единым целым и останавливать CDB ради одной PDB бессмысленно. С помощью команды SQL *alter pluggable database* можно задать параметр *Open_Mode* («режим открытия») для любой PDB, если текущий контейнер — *root*. Задать этот параметр для конкретной PDB можно также в случае, если текущий контейнер — эта PDB. Для совместимости могут использоваться команды `STARTUP` и `SHUTDOWN` SQL*Plus, если текущий

контейнер — PDB. Они транслируются в соответствующие команды SQL *alter pluggable database*.

Помимо *Open_Mode* можно задать для статуса *Restricted* значение `NO` или `YES`. Существует пять различных комбинаций *Open_Mode* и статуса *Restricted* для PDB. (Если параметр *Open_Mode* равен `MOUNTED`, статус *Restricted* может быть только *null*.) С помощью команды SQL *alter pluggable database* можно прямо перейти от подобной комбинации к любой другой с помощью ключевого слова *force*, как показано в *Code_14*.

```
-- Code_14
-- PDB_1 запускается в режиме READ WRITE
alter pluggable database PDB_1 open read only force
```

В этом примере все сессии, подключенные к *PDB_1* и участвующие в транзакциях, тип которых не является «только чтение», будут прекращены и произойдет откат их транзакций. Это гораздо удобнее по сравнению с правилами соответствующих изменений режима для не-CDB БД.

В мультиарендной архитектуре значительно сокращается количество уведомлений, которые один экземпляр RAC направляет другим экземплярам RAC в случае изменения SGA (во избежание несогласованности информации, кэшируемой в нескольких SGA). Нет смысла информировать экземпляры RAC, где конкретная PDB не открыта, об изменениях в элементах, кэшируемых в тех экземплярах RAC, где эта база открыта.

Таким образом, мультиарендная архитектура обеспечивает простую декларативную модель для привязки серверных приложений к экземплярам RAC. Эта модель реализует данную задачу более надежным образом и обеспечивает более высокую производительность, чем ручная схема с использованием консолидации на основе схем. Эта идея обсуждается более подробно в разделе «*Привязка PDB к RAC-экземплярам*» на *стр. 45*.

Представления словаря данных и представления производительности

Гарантия PDB/не-CDB совместимости означает, что представления словаря данных и представления производительности, когда по ним делается запрос и текущий контейнер — PDB, должны отображать информацию только о тех артефактах, которые определены в этой PDB, и артефакты, определенные в *root*. Например, если демонстрационные схемы установлены в *PDB_1* и *PDB_1* является текущим контейнером, то запрос в *Code_15* даст результаты, показанные ниже.

```
-- Code_15
select Owner, Object_Name from DBA_Objects
where Owner = 'SYS'
and Object_Type like 'VIEW'
and Object_Name like 'ALL%'
union
select Owner, Object_Name
from DBA_Objects
where Object_Type like 'VIEW'
and Owner in (select Username from DBA_Users where Common = 'NO')
```

Результаты запроса будут включать следующее:

```
SYS   ALL_ARGUMENTS
SYS   ALL_DIRECTORIES
SYS   ALL_ERRORS
SYS   ALL_INDEXES
SYS   ALL_OBJECTS
SYS   ALL_TABLES
SYS   ALL_VIEWS
HR    EMP_DETAILS_VIEW
OE    PRODUCTS
SH    PROFITS
```

Здесь не будет никаких объектов, определенных в *PDB_2* или любой другой аналогичной базе в этой CDB.

Точно так же, если *PDB_1* является текущим контейнером и делается запрос по *v$Sql*, будет отображаться информация только о тех SQL-предложениях, которые запускались, когда текущим контейнером был *PDB_1*.

Мультиарендная архитектура устанавливает новые правила для результатов, когда запрашиваются представления словаря данных и представления производительности и текущим контейнером является *root*. Теоретически результаты будут объединением по всем экземплярам запрашиваемого представления в открытых на данный момент контейнерах. Строки различимы благодаря тому, что в представлениях в версии 12.1 появился новый столбец *Con_ID* (идентификатор контейнера).

Таким образом достигается бизнес-цель — поддержка единого образа системы по всей CDB. Например, можно легко контролировать нарушения соглашения об именовании объектов базы данных, заданного клиентами. (Некоторые клиенты настаивают на том, что имена не должны включать такие специальные символы, как одинарные и двойные кавычки, точку с запятой или вопросительный знак). Еще одно возможное использование — поиск команд SQL, выполнение которых занимает больше всего времени в CDB.

Представления производительности просто получают столбец *Con_ID*. С представлениями словарей данных ситуация иная. Столбец *Con_ID* отображается только в новом семействе представлений словаря данных, где имена начинаются с *CDB_*. Они расширяют концепцию *DBA_ ** представлений словаря данных. *All_ ** и *User_** таким образом не расширяются.

Представления могут возвращать строки для нескольких контейнеров (это те представления, у которых появился *Con_ID)*. Все такие представления называются *container_data представлениями* («представление контейнерных данных»). Если текущий контейнер — PDB, то в каждом *container_data представлении* отображаются только те строки, у которых значение столбца *Con_ID* обозначает эту PDB или CDB как базу данных целиком. Если же текущим контейнером является *root*, то в *container_data представлении* могут быть строки из многих контейнеров. Помните о том, что локальный пользователь не может быть создан в *root*, и только общий пользователь может видеть результаты в *container_data представлении* с несколькими разными значениями *Con_ID*. Набор контейнеров, для которых конкретный общий пользователь может видеть результаты в *container_data представлении*, определяется атрибутом пользователя (который задается с помощью команды SQL *Alter User*).

Этот атрибут, с одной стороны, может задавать конкретное *container_data представление* для двух указанных PDB баз данных, как это показано в *Code_16*.

```
-- Code_16
-- Разрешить c##u1 видеть данные только для CDB_Objects_
-- в CDB$Root, PDB_1 и PDB_2
alter user c##u2
set container_data = (cdb$root, pdb_001, pdb_002)
for Sys.CDB_Objects
container = current
```

Следует отметить, что эта команда SQL допустима только в случае, если текущий контейнер — *root*, и что этот атрибут может быть задан только локально в *root*. Помимо этого, контейнер *CDB$Root* должен быть всегда включен в список.

С другой стороны, как показано в *Code_17*, этот атрибут может указывать все *container_data представления*, настоящие и будущие (которые могут появиться в новых релизах Oracle Database), для всех контейнеров, настоящих и будущих. Пользователи, поставляемые Oracle (такие как *Sys* и *System),* настроены таким образом.

```
-- Code_17
-- Позволить c##u2 видеть данные для каждого контейнера, настоящего или будущего,
-- в каждом представлении container_data
alter user c##u2
set container_data = all
container = current
```

Варианты на уровне CDB в сравнении с вариантами на уровне PDB

Консолидация неизбежно предполагает компромисс — отказ от некоторой доли автономии, которая была возможна в независимо управляемом серверном приложении, в обмен на предполагаемое сокращение капитальных и эксплуатационных расходов. В этом разделе показаны варианты, возможные только для CDB в целом, а также некоторые другие, которые можно выбрать по-разному для каждой PDB.

Варианты только для CDB в целом

Версия ПО Oracle Database и особенности платформы

Как и в случае консолидации на основе схем, основная идея консолидации — разместить в единой базе данных множество отдельных серверных приложений, использующих одну и ту же SGA (в каждом экземпляре RAC) и один и тот же набор фоновых процессов. Это означает, что каждая PDB в одной и той же CDB должна иметь общую для всех версию ПО Oracle Database. Вопрос, могут ли *разные PDB в одной и той же CDB* использовать разные версии ПО Oracle Database, имеет не больше смысла, чем вопрос о том, могут ли *разные схемы в одной и той же не-CDB базе данных* иметь разные версии ПО Oracle Database. Конечно, обе модели консолидации внутри базы данных подразумевают единственный выбор типа платформы, а также типа и версии операционной системы для всех консолидируемых серверных приложений.

Парадигма подключения и отключения для патчирования версии Oracle устраняет неудобство, которое иначе могло бы возникнуть из-за того, что CDB как целое использует определенную версию ПО Oracle Database. Кроме того, обеспечиваемая таким образом мобильность гарантирует ослабление существующей при консолидации на основе схем привязки серверного приложения к платформе, а также типу и версии ОС, на которых работает не-CDB БД, обслуживающая серверное приложение.

Файл *spfile*, контрольные файлы и файл паролей

Эти файлы являются общими для всей CDB. Это означает, что PDB не может *буквально* иметь свой собственный *spfile*. Как будет показано в разделе «*Инициализационные параметры, задаваемые на уровне PDB, и свойства базы данных*» на *стр. 42*, некоторые значения параметров могут быть установлены персистентно с помощью оператора *alter system* в пределах конкретной PDB. Эти значения параметров сохраняются должным образом, но не в файле *spfile*. Следовательно, вы не можете указать *pfile*, когда открываете PDB. Поэтому, если требуется дамп значений параметров, заданных явно с помощью оператора *alter system* в пределах PDB, для этого необходимо написать запрос SQL. Соответственно, чтобы эти значения использовались, необходимо запрограммировать последовательность операторов SQL *alter system*.

Data Guard, RMAN резервное копирование, redo и undo

Клиенты, использующие консолидацию на основе схем, отмечают значительное уменьшение эксплуатационных расходов, так как они могут настраивать и использовать Data Guard для одной не-CDB БД, в которой размещено множество различных серверных приложений. До консолидации им приходилось управлять Data Guard для каждого отдельного серверного приложения как отдельной задачей. Это канонический пример принципа управления многими как одним (manage-as-one). Точно так же эти клиенты снижают эксплуатационные расходы благодаря использованию одного режима регулярного копирования RMAN для множества серверных приложений. Однако, как указано в разделе «*Стремление к максимальной плотности консолидации*» на *странице 5*, это решение приемлемо не для всех случаев восстановления состояния серверных приложений на определенный момент времени в прошлом. Мы вернемся к этому вопросу в разделе «*Резервное RMAN копирование по требованию для PDB*» на *стр. 42*.

Как Data Guard, так и резервное копирование RMAN используют redo журналы базы данных: Data Guard постоянно применяет redo на резервном узле. Резервное копирование RMAN также использует redo при восстановлении — от момента создания резервной копии до требуемого времени восстановления. Следовательно, чтобы при консолидации на основе PDB получить такую же экономию от управления многими как одним, как в случае консолидации на основе схем, вся CDB имеет всего один поток redo. Это означает, что redo журнал базы данных, как и область SGA , виртуализован *логически*: каждая запись redo журнала аннотируется — указывается идентификатор контейнера, где произошло регистрируемое изменение. Повтор (redo) и откат (undo) работают вместе, и вся CDB имеет одно общее табличное пространство отката для каждого экземпляра RAC. Undo также виртуализован *логически*: здесь также в каждой undo записи указывается идентификатор исходного контейнера. При оценке влияния модели консолидации на производительность необходимо учитывать два момента.

- Во-первых, если используется консолидация на основе PDB, производительность остается на том же уровне, что и в случае консолидации на основе схем. Опыт клиентов показывает, что консолидация на основе схем не вредит производительности. Влияние оказывают другие факторы. В случае использования любой из двух моделей консолидации внутри базы данных набор серверных приложений обслуживается в сумме значительно меньшим количеством фоновых процессов, а серверные приложения более эффективно используют память, когда одна область SGA разделяется между ними, по сравнению с ситуацией, когда каждое приложение имеет собственную область SGA с собственным размерами и с собственными ограничениями.

- Во-вторых, новые внутри-redo и внутри-undo схемы индексирования могут ускорить доступ к записям для соответствующей PDB. Кроме того, отдельные логически виртуализованные «секции» redo и undo функционально независимы, поэтому параллельные процессы могут обращаться к ним, не ожидая друг друга.

Клиенты, которые хотят увеличить выгоды от инвестиций путем консолидации, обычно проводят эмпирические исследования, используя для этого репрезентативные модели собственных кандидатов на консолидацию.

Кодировка

Теоретически мультиаренданая архитектура допускает возможность существования разных кодировок в каждой PDB. Однако, разрешив это, мы потеряем преимущество управления многими как одним.

Кроме того, в последние годы возникла тенденция, когда огромное количество приложений имеют международную и многоязычную базу пользователей. В связи с этим постоянно растет доля обработки на клиентской стороне с использованием кодировки Unicode. Это означает снижение эффективности, так как данные постоянно преобразуются между клиентом и базой данных, если они не хранятся в базе в кодировке Unicode.

По этой причине было решено разрешить задавать кодировку только для CDB в целом. В качестве кодировки CDB мы рекомендуем использовать Unicode (*AL32UTF8*) .

Конечно, остается проблема старых приложений, когда серверное приложение использует конкретную кодировку и его нельзя переделать экономически эффективным способом. Именно здесь можно получить выгоду от того, что на одной и той же платформе может быть несколько CDB с собственными кодировками. (Это просто еще один пример более общего принципа. Самое очевидное преимущество размещения нескольких CDB на одной платформе заключается в том, что каждая CDB может иметь свою версию ПО Oracle Database.)

Инициализационные параметры, задаваемые на уровне CDB, и свойства базы данных

Семейство представлений *v$System_Parameter* имеет в версии 12.1 новый столбец *IsPDB_Modifiable*, возможные значения которого — FALSE и TRUE. (Мы рассмотрим представления с *IsPDB_Modifiable* TRUE в разделе *«Инициализационные параметры, задаваемые на уровне PDB, и свойства базы данных»* на *стр. 42*). Примерно 200 параметров имеют значение *IsPDB_Modifiable*, равное FALSE. Вот несколько примеров:

```
audit_file_dest
audit_trail
background_core_dump
background_dump_dest
core_dump_dest
cpu_count
db_block_size
db_name
db_recovery_file_dest
db_recovery_file_dest_size
db_unique_name
instance_name
local_listener
log_archive_dest
log_archive_duplex_dest
memory_max_target
memory_target
parallel_degree_policy
pga_aggregate_limit
pga_aggregate_target
Processes
result_cache_max_size
sga_max_size
sga_target
shared_pool_reserved_size
shared_pool_size
undo_management
undo_retention
undo_tablespace
user_dump_dest
```

Этот список подтверждает принцип управления многими как одним.

Кроме свойства NLS_CHARACTERSET, которое уже обсуждалось, все другие свойства, отображаемые в представлении свойств *(Database_Properties)* , можно задать с помощью команды SQL *alter database*, если текущий контейнер — PDB.

Отчеты AWR

Вы можете создавать отчеты AWR (Automatic Workload Repository) для CDB в целом, и мы предполагаем, что регулярные отчеты будут создаваться на этом уровне. Для поддержки *оперативного* исследования производительности в рамках одного серверного приложения вы также можете создавать отчеты AWR для конкретной PDB.

Варианты, которые можно выбирать отдельно для каждой PDB

Восстановление состояния PDB на определенный момент в прошлом

Для самостоятельности PDB особенно важна способность восстановления состояния на определенный момент времени в прошлом без ущерба для доступности других PDB в той же самой CDB. Именно это наиболее выгодно отличает консолидацию на основе PDB от консолидации на основе схем.
В консолидации на основе схем просто не существует удобного метода для достижения этой бизнес-цели — возможности восстановления состояния серверного приложения на момент времени в прошлом.
Команда *flashback pluggable database* в версии 12.1 не поддерживается.

Ситуативное резервное копирование RMAN для PDB

Как правило, проще всего поддержать принцип управления многими как одним, если использовать регулярное резервное копирование RMAN на уровне CDB, как было объяснено выше. Однако бывают случаи, когда лучше выполнить *ситуативное (Ad hoc)* резервное копирование для конкретной PDB. Это обсуждалось в разделе *«Удаление подключаемой базы данных»* на *стр. 26* в связи с совместным использованием команды *unplug PDB* и затем команды *drop PDB* (с сохранением файлов данных).

Команда alter system flush Shared_Pool

Как было сказано в разделе «*Динамические аспекты мультиарендной архитектуры: экземпляр Oracle, пользователи и сессии»* на *стр. 31*, блоки данных в буфере блоков и структуры библиотечного кэша аннотируются идентификаторами контейнеров. Это придает смысл выполнению команды *alter system flush Shared_Pool*, когда текущий контейнер — PDB.

Инициализационные параметры, задаваемые на уровне PDB, и свойства базы данных

Каждый инициализационный параметр, у которого в представлении *v$System_Parameter* атрибуты *IsSes_Modifiable* и *IsSys_Modifiable* не равны `FALSE`, можно задать с помощью команды *alter system*, когда текущий контейнер — PDB. Иными словами, у всех таких параметров атрибут *IsPDB_Modifiable* равен `TRUE`. Некоторые другие параметры с *IsSys_Modifiable*, не равным `FALSE` и *IsSes_Modifiable* , но равным `FALSE`, также могут быть заданы с помощью *alter system* в рамках PDB. Это относительно небольшой список в 12.1:

```
cell_offload_decryption
fixed_date
listener_networks
max_string_size
open_cursors
optimizer_secure_view_merging
resource_limit
resource_manager_plan
sessions
```

Что касается других допустимых значений для *IsSys_Modifiable* — `IMMEDIATE` и `DEFERRED`, выражение *scope* (*scope=memory*, *scope=spfile* или *scope=both*) дает ожидаемый эффект: значение будет сохранено, если требуется. Для гарантии PDB/не-CDB совместимости был сохранен прежний синтаксис команды *alter system*, но значение параметра не записывается в *spfile*. Вместо этого оно записывается в соответствующие таблицы словаря данных таким образом, что *оба* таких PDB специфичных значения будут действовать, когда она открыта, *и при этом* значения будут перемещаться вместе с PDB при ее отключении и подключении.

Ошибка ORA-65040

ПО Oracle Database поддерживает большой набор команд SQL. Некоторые из них используются только администратором, управляющим всей базой данных, в особых обстоятельствах. В CDB небольшой набор этих команд SQL, предназначенных исключительно для администратора, не может использоваться внутри PDB. При попытке использовать такой оператор выдается сообщение *ORA-65040 operation not allowed from within a pluggable database* («выполнение операции ORA-65040 внутри подключаемой базы данных запрещено»). Это запланировано разработчиками. Гарантия PDB/не-CDB совместимости должна рассматриваться в этом свете. Обратите внимание, что все команды SQL могут выдаваться автоматически кодом на клиентской стороне. Но клиентский код, который можно назвать *приложением* в отличие от *средства администрирования,* такого как Enterprise Manager, не выдает никаких команд, вызывающих ошибку *ORA-65040*. Даже в клиентском коде, который можно было бы назвать *автоматизацией установки*, такие команды не должны использоваться, если следовать лучшим практикам.

Управление распределением ресурсов между PDB и внутри CDB

Сессии, которые используют разные серверные приложения, размещенные на одной и той же платформе, конкурируют за следующие ресурсы:

• количество активных сессий

• CPU

• возможность использовать параллельные серверные процессы Oracle

• файловый ввод-вывод

• использование памяти SGA и способность разместить PGA

• сетевой ввод-вывод

Некоторые клиенты хотят полностью контролировать конкуренцию за каждый такой ресурс и поэтому размещают каждое серверное приложение в собственной не-CDB БД на собственной виртуальной машине, используя преимущество виртуализации ОС. Однако это полностью исключает экономию, которую обеспечивает принцип управления многими как одним. Многие клиенты обнаружили, что выгода от инвестиций будет максимальной, если поставить экономию за счет принципа управления многими как одним на первое место, и поэтому используют консолидацию на основе схем.

Можно использовать Resource Manager в контексте консолидации на основе схем. Но для этого требуется тщательно спланированная дисциплина, контролируемая вручную, когда доступ к каждому серверному приложению возможен только с помощью сервисов, специально созданных для этой цели. Проблема, конечно, в том, что при консолидации на основе схем только человек знает, где провести границы каждого серверного приложения, ПО Oracle Database этого не понимает. Если используется консолидация на основе PDB, то сама PDB предоставляет мощный декларативный механизм, с помощью которого человек может указать ПО Oracle Database, где находятся границы: каждое серверное приложение устанавливается в собственной PDB и ни в одной PDB не может быть больше одного серверного приложения.

В мультиарендной архитектуре механизм Resource Manager расширен и теперь позволяет создать план распределения ресурсов на уровне CDB для управления конкуренцией за ресурсы между PDB, содержащимися в одной CDB.

Вычислительные ресурсы, контролируемые планом на уровне CDB в 12.1

План Resource Manager на уровне CDB в версии 12.1 контролирует:

• количество активных сессий

• CPU

• возможность использовать параллельные серверные процессы Oracle

• файловый ввод-вывод (только в Exadata)

План Resource Manager не контролирует:

• использование памяти SGA и способность размещать PGA

• сетевой ввод-вывод

Привязка PDB к экземпляру RAC[22] может использоваться как грубый способ контроля над конкуренцией за SGA и PGA память.

Модель долей и лимитов

В Resource Manager реализована стандартная отраслевая модель, основанная на двух понятиях: доля (*share)* и лимит (*cap)*[23]. По этой схеме каждому конкуренту выделяется некоторое количество долей от *одной* до любого положительного целого числа (однако любое количество больше *десяти* может оказаться бесполезным). Обычно в определенный момент активны только некоторые из конкурентов. Каждый из активных конкурентов получает часть управляемого ресурса в размере n/t, где n — это количество долей, выделенных конкретному конкуренту, а t — общее количество долей, выделенных всем конкурентам, активным на данный момент. Дополнительно для любого конкурента может быть установлен лимит (cap) — дробное число в диапазоне от *нуля* до *единицы* включительно (которое часто выражается в виде процента). Конкурент с лимитом, равным c, никогда не получит больше этой дробной части управляемого ресурса независимо от того, что может показать расчет долей. Конечно, конкурент с таким лимитом может получать меньше в зависимости от текущего результата расчета долей.

Ценность понятия «доля» (share) очевидна, это то, без чего невозможно управление ресурсами. «Лимит» (cap) полезен, если количество конкурентов постоянно растет до какого-то запланированного числа. Это обеспечивает некоторый резерв, не создавая ложных ожиданий. (Пользователи довольны повышением производительности, когда приложение переносится без каких-либо лимитов с одной машины на другую, более мощную, и сначала является единственным на этой машине. Однако у них откровенно короткая память. По мере того как все больше приложений переносятся на машину для консолидации, клиенты начинают замечать, что производительность постоянно ухудшается. Поэтому лучше задать лимит производительности для приложений, которые переносятся на машину для консолидации одними из первых. Это должен быть уровень, который ожидается после того, как консолидация завершится.)

Для управления распределением ресурсов между PDB каждая создаваемая PDB получает по умолчанию одну долю, и никакие лимиты не задаются.

Управление сессиями, CPU, параллельными процессами Oracle и файловым вводом-выводом с помощью плана на уровне CDB в 12.1

Если план создается или изменяется, это немедленно отражается на всех активных сессиях[24].

---

[22] Это описано в разделе *«Привязка PDB к RAC экземплярам»* на *стр. 45*.

[23] Здесь можно ознакомиться с научной статьей на эту тему, опубликованной в 1995 г.:
http://people.cs.umass.edu/~mcorner/courses/691J/papers/PS/waldspurger_stride/waldspurger95stride.pdf
Обсуждения на форуме разработчиков от поставщика ПО для виртуализации операционных систем:
http://communities.vmware.com/docs/DOC-7272
На английском языке вместо термина *cap* (лимит или предел) иногда используется термин *resource utilization limit* (лимит использования ресурсов).

[24] План устанавливается с помощью команды *alter system set resource_manager_plan = My_Plan*. Как отмечено в разделе *«Инициализационные параметры, задаваемые на уровне PDB, и свойства базы данных»* на *стр. 42*, инициализационный параметр *resource_manager_plan* может быть задан на уровне контейнера. Поэтому, для установки плана на уровне CDB используется команда *alter system*, когда текущий контейнер — *root*.

Количество активных сессий ограничивается только лимитами, понятие доли здесь не имеет смысла. Фактически лимит сессий задается с помощью инициализационного параметра *sessions*[25].

Управление использованием CPU и файловым вводом-выводом осуществляется с помощью вышеописанных механизмов долей и лимитов. Диспетчеризация сессий производится каждые 100 мс.

В *Code_18* показано, как настраивается план на уровне CDB для выделения одной доли базе данных *PDB_1* и трех долей базе данных *PDB_2*.

```
-- Code_18
DBMS_Resource_Manager.Create_CDB_Plan(
'My_Plan', 'some comment');

DBMS_Resource_Manager.Create_CDB_Plan_Directive(
'My_Plan', 'PDB_1', Shares => 1);

DBMS_Resource_Manager.Create_CDB_Plan_Directive(
'My_Plan', 'PDB_2', Shares => 3);
```

В *Code_19* показано, как изменяется план на уровне CDB для выделения двух долей *PDB_1*, пяти долей *PDB_2* и для установки лимитов для этих баз данных 20 и 50 % соответственно.

```
-- Code_19
DBMS_Resource_Manager.Update_CDB_Plan_Directive(
'My_Plan', 'PDB_1', New_Shares => 2);

DBMS_Resource_Manager.Update_CDB_Plan_Directive(
'My_Plan', 'PDB_1', New_Utilization_Limit => 20);

DBMS_Resource_Manager.Update_CDB_Plan_Directive(
'My_Plan', 'PDB_2', New_Shares => 5);

DBMS_Resource_Manager.Update_CDB_Plan_Directive(
'My_Plan', 'PDB_2', New_Utilization_Limit => 50);
```

Возможность использования параллельных серверных процессов Oracle регулируется той же самой моделью долей. Параметр инициализации *parallel_degree_policy* должен быть установлен в значение `AUTO`. Тогда достаточное количество команд SQL выполняется параллельно, чтобы поддерживать машину под нагрузкой, а для того, чтобы избежать снижения степени параллелизма, используется очередь для SQL-команд. Здесь понятие «лимит» требует собственной отдельной настройки с помощью параметра *parallel_server_limit* в подпрограмме *Create_CDB_Plan_Directive* и параметра *new_parallel_server_limit* в подпрограмме *Update_CDB_Plan_Directive* из пакета *DBMS_Resource_Manager*.

Привязка PDB к RAC-экземплярам

Как объяснено в разделе «*Динамические аспекты мультиарендной архитектуры: экземпляр Oracle, пользователи и сессии*» на *стр. 31*, параметр *Open_Mode* каждой PDB может быть установлен на уровне PDB в значение `MOUNTED`, `READ ONLY` или `READ WRITE`. Следовательно, в каждом экземпляре RAC значения могут быть разными. Например, если CDB с восемью PDB настраивается как база данных RAC с восемью экземплярами Oracle, каждая PDB может быть открыта только в одном экземпляре RAC, и таким образом каждый экземпляр RAC открывает только одну PDB. Такая схема обеспечивает максимальную изоляцию ресурсов. Но одновременно она компрометирует совместное использование SGA и фоновых процессов, а ведь именно эта возможность обеспечивает высокую плотность консолидации, которая является отличительной чертой консолидации на основе PDB (и консолидации на основе схем).

[25] Как указано в разделе «*Инициализационные параметры, задаваемые на уровне CDB, и свойства базы данных*» на *стр. 42*, параметр инициализации *sessions* может быть задан с помощью команды SQL *alter system* , если текущий контейнер — PDB.

Возможно, имело бы больше смысла использовать привязку PDB к экземпляру RAC в случае, если CDB содержит пару сотен или больше PDB. В этом случае можно было бы связать верхнюю четверть списка PDB (с точки зрения требуемой вычислительной мощности) с 75 % экземпляров RAC, а остальные три четверти PDB — с остальными 25 % экземпляров RAC.

Выбор между одиночной PDB в CDB *и* не-CDB базой данных

Предположим, что серверное приложение требует настолько высокой пропускной способности, что для него необходимо выделить отдельную платформу. Такое приложение может быть установлено в не-CDB БД или PDB, которая (не считая *начальной PDB*) является *единственной* в своей CDB. Мы будет называть такую конфигурацию *одиночной PDB (lone-PDB)*. Другое название — *конфигурация с одним арендатором (single-tenant configuration)*.

Можно ли сказать, что не-CDB БД лучше конфигурации с одним арендатором?

Наш ответ — решительное *нет*. Гарантия PDB/не-CDB совместимости показывает, что никакой функциональной разницы нет. Тщательное тестирование, проведенное инженерами Oracle, подтвердило это, а также показало, что нет никакой разницы и с точки зрения производительности.

Более интересен обратный вопрос: является ли конфигурация с одним арендатором лучшим выбором, чем не-CDB БД?

Здесь ответ *да*. Даже если у вас никогда не будет больше одной PDB, новая мультиарендная архитектура дает следующие существенные преимущества:

- Отключение/подключение (unplug/plug) дает вам с точки зрения функциональности Data Pump 3-го поколения.
- Отключение/подключение в пустую, вновь созданную CDB, дает вам новую парадигму патчирования версии Oracle[26].

В Oracle Database 12.1 с опцией Oracle Multitenant можно иметь от *двух* до 252 PDB в каждой CDB. (*seed PDB* в число разрешенных 252 PDB не входит.)

Конфигурация с одним арендатором (означающая выбор мультиарендной архитектуры) доступна бесплатно в Standard Edition, Standard Edition One и Enterprise Edition[27].

[26] Горячее клонирование (если оно будет поддерживаться в будущем релизе) предполагает следующий подход. Горячее клонирование в «промежуточную» CDB с той же самой версией ПО Oracle Database и с регистрацией системного номера изменения (SCN), при котором это горячее клонирование было запущено. Затем — отключение и подключение к более новой версии CDB, синхронизация с исходной PDB и отслеживание изменений с помощью репликации GoldenGate. С точки зрения функциональности это будет эквивалентно использованию временной логической резервной БД, но окажется значительно более удобным.

[27] Эти условия указаны в Руководстве по лицензированию Oracle Database 12*c* (Oracle Database 12*c* Licensing Guide).

Заключение

Как мы видели, основное различие между старой не-CDB базой данных и новой мультиарендной архитектурой заключается в том, что последняя действительно обеспечивает виртуализацию внутри базы данных. Физически это реализовано в словаре данных путем горизонтального секционирования, которое отделяет систему Oracle в *root* от клиентской системы в PDB, и благодаря этому базовому разделению можно разместить множество PDB в одной и той же CDB. PDB — это декларативный механизм для определения границ консолидированного серверного приложения.

Подлинная виртуализация внутри базы данных устраняет основные недостатки консолидации на основе схем: коллизию глобальных имен, из-за которой необходимо вносить дорогостоящие и рискованные изменения в существующие серверные приложения, прежде чем они смогут сосуществовать в одной и той же не-CDB базе данных, а также то, что *any* привилегии и аналогичные административные привилегии, а также права, назначенные *public*, распространяются на все серверные приложения в не-CDB базе данных.

Именно физическое разделение *root* и PDB обеспечивает подключаемость (pluggability). Фактически, эта возможность реализует третье поколение Data Pump через механизм отключения и подключения. Кроме того, возможность отключения и подключения между разными CDB с разными версиями ПО Oracle Database дает новую парадигму патчирования версии Oracle. Таким образом, подключаемость устраняет две проблемы, возникающие при консолидации на основе схем: сложность провизионирования (т. е. переноса серверного приложения с места на место и его клонирования) и ситуацию, когда патчирование версии Oracle для одного серверного приложения затрагивает другие, не готовые к такому изменению в среде.

Мы также видели, что модель общего доступа для SGA и фоновых процессов по существу одинакова для консолидации на основе PDB и консолидации на основе схем. Таким образом, преимущество высокой плотности консолидации, которое обеспечивалось при старом подходе, полностью сохраняется. Кроме того, логическая виртуализация внутри SGA (которая через блок данных переносится в файлы данных), redo и undo дают новые возможности для управления распределением ресурсов между PDB внутри CDB и обеспечивают восстановление состояния PDB на момент времени в прошлом.

И, наконец, хочется сказать, что *CDB для PDB — это то же самое, что операционная система для не-CDB базы данных*. Гарантия PDB/не-CDB совместимости означает, что новым феноменом, характеризующим мультиарендную архитектуру, является *root*. Как примитивы операционной системы организуют задачи провизионирования и другие задачи обслуживания для не-CDB баз, так и команды SQL, выполняемые в *root*, реализуют аналогичные задачи для PDB. Это означает, что язык PL/SQL, знаменитый своей платформонезависимостью и переносимостью, доступен везде, где доступно ПО Oracle Database, и понятный всем администраторам баз данных, является языком для автоматизации всех операций с PDB.

Таким образом, Oracle Multitenant представляет собой новое поколение консолидации для серверных приложений. Эта опция обеспечивает преимущества принципа управления многими как одним, к которым так стремились те, кто использовал консолидацию на основе схем. Эта опция — отличный выбор для развертывания: не нужно менять ни серверное приложение, ни клиентский код.

Приложение A
Мультиарендная архитектура в библиотеке документации по базам данных Oracle

Для первого знакомства и обзора начните с книги Concepts. Ознакомьтесь с новым разделом *Multitenant Architecture*.

Основная информация содержится в Administrator's Guide. Обратите внимание на новый раздел *«Managing a Multitenant Environment»*. Конечно, из-за гарантии PDB/не-CDB совместимости в Database Development Guide не требуется добавлять что-либо о мультиарендной архитектуре, кроме ее упоминания. (Область действия *edition (редакции)* — это PDB, в которой она определена. Это означает, что в одной CDB можно одновременно выполнять много операций обновления на основе редакций (Edition-Based Redefinition, EBR). Это устраняет проблему клиентов, использовавших консолидацию на основе схем для внедрения нескольких серверных приложений в одной и той же базе данных, которая предшествовала версии 12.1 и обязательно была не-CDB, когда требовалось патчировать в онлайн-режиме более одного серверного приложения.)

Новое различие между локальными и общими пользователями, ограничение зоны действия привилегий текущим контейнером и то, как это приводит к новым понятиям для формального управления разделением обязанностей между администратором базы данных высшего уровня (администратор CDB) и администратором приложения (администратор PDB), описаны в Security Guide. В книге обсуждается еще одна важная концепция: *представления container_data*.

Новых конструкций SQL в мультиарендной архитектуре примечательно мало. В основном они используются в командах, которые оперируют PDB базами данных как сущностями. Формальные определения синтаксиса и семантики этих конструкций содержатся в SQL Language Reference.

Появилось несколько совершенно новых представлений словаря данных (например, *DBA_PDBs*) и представлений производительности (например, *v$PDBs*). Что еще важнее, в каждом представлении производительности есть новый столбец *Con_ID* — идентификатор контейнера, к которому относятся факты, отображаемые в представлении. Точно так же, но с помощью немного другой модели в набор разновидностей представлений словаря данных (*DBA_*, *All_* и *User_*) добавлена новая разновидность — *CDB_*. В каждом представлении *CDB_* есть новый столбец *Con_ID* . Если представление включает столбец *CDB_*, это отражает его статус как *представления container_data*. Описание всех этих представлений содержится в Database Reference.

Архитектура Oracle Multitenant
Июнь 2013 г.
Автор: Брин Луэллин (Bryn Llewellyn)

Oracle Corporation
Головной офис
500 Oracle Parkway
Redwood Shores, CA 94065
США

Для запросов со всего мира: Тел.: +1-650-506-7000
Факс: +1.650.506.7200

oracle.com/ru

Разрабатывая свои программы и продукцию, корпорация Oracle заботится об окружающей среде.

Hardware and Software, Engineered to Work Together