Содержание

ГЛАВА 7

Параллелизм и многоверсионность

Как констатировалось в предыдущей главе, одной из ключевых проблем при разработке многопользовательских приложений, работающих с базами данных, является доведение до максимума уровня параллелизма доступа, но в то же самое время и обеспечение для каждого пользователя возможностей чтения и модификации данных в согласованной манере. В этой главе мы детально рассмотрим, каким образом Oracle достигает *многоверсионной согласованности чтения*, и что это означает для разработчика. Будет также представлен новый термин *согласованность записи*, который используется для описания работы Oracle не только в среде чтения с согласованностью прочитанных данных, но также в смешанной среде чтения/записи.

Понятие управления параллелизмом

Управление параллелизмом — это коллекция функций, предоставляемых базой данных для разрешения многим пользователям одновременного доступа к данным и их модификации. В предыдущей главе отмечалось, что *блокировка* — это один из основных механизмов, посредством которого Oracle регулирует параллельный доступ к разделяемым ресурсам базы данных и предотвращает взаимное влияние друг на друга параллельно выполняющихся транзакций базы данных. Подводя краткие итоги, можно сказать, что Oracle применяет разнообразные блокировки, включая перечисленные ниже.

- **Блокировки TX (транзакции).** Эти блокировки устанавливаются на протяжении транзакции, модифицирующей данные.
- **Блокировки TM (помещение в очередь DML) и блокировки DDL.** Эти блокировки гарантируют, что структура объекта не изменится во время модификации его содержимого (блокировка TM) или структуры самого объекта (блокировка DDL).
- **Защелки и семафоры.** Это внутренние блокировки, используемые Oracle при посредничестве доступа к разделяемым структурам данных. На защелки и семафоры в этой главе мы будем ссылаться просто как на защелки, хотя в зависимости от версии Oracle они могут быть реализованы в виде семафоров операционной системы.

В каждом случае с установкой блокировки связаны минимальные накладные расходы. Блокировки TX исключительно масштабируемы в отношении как производительности, так и кардинальности. Блокировки TM и DDL применяются в наименее ограничивающем режиме всякий раз, когда это возможно. Защелки и очереди являются очень легковесными и быстрыми (из них очереди немного сложнее, хотя обладают более развитыми возможностями). Проблемы возникают только из-за неудачно спроектированных приложений, которые удерживают блокировки *дольше, чем необходимо*, и вызывают блокирование в базе данных. Если вы аккуратно проектируете свой код, то механизмы блокировки Oracle позволят строить масштабируемые приложения с высокой степенью параллелизма.

На заметку! Выше используется формулировка "дольше, чем необходимо". Это не значит, что вы должны пытаться зафиксировать (завершить) транзакцию как можно скорее. Транзакции должны длиться столько, сколько нужно, но не дольше. То есть транзакция является единицей работы — либо все, либо ничего. Вы должны осуществлять фиксацию, когда единица работы завершена, но не раньше и не позже!

Однако поддержка параллелизма в Oracle не ограничивается эффективной блокировкой. В Oracle реализована архитектура *многоверсионности* (представленная в главе 1), которая обеспечивает управляемый, но с высокой степенью параллелизма доступ к данным. Многоверсионность характеризует способность Oracle параллельно материализовывать множество версий данных и является механизмом, с помощью которого Oracle обеспечивает согласованные по чтению представления данных (т.е. согласованные результаты на определенный момент времени). Довольно приятный побочный эффект многоверсионности состоит в том, что процесс чтения данных никогда не блокируется процессом записи данных. Другими словами, операции записи не блокируют операции чтения. В этом заключается одно из фундаментальных отличий Oracle от других баз данных. Запрос, который только читает информацию, в Oracle никогда не будет блокироваться, он никогда не попадет в состояние взаимоблокировки с другим сеансом и никогда не выдаст ответ, который в базе данных отсутствует.

На заметку! Во время обработки распределенной *двухфазной фиксации* (Two Phase Commit) существует короткий период, когда Oracle будет предотвращать доступ по чтению к информации. Поскольку эта обработка представляет собой редкое и необычное явление (проблема касается только запросов, которые запускаются между фазами подготовки и фиксации и пытаются прочитать данные до поступления команды фиксации), подробно здесь она не рассматривается.

Модель многоверсионности Oracle для обеспечения согласованности чтения по умолчанию применяется на *уровне операторов* (для каждого запроса) и также может применяться на *уровне транзакций*. Это значит, что каждый оператор SQL, отправленный базе данных, видит согласованное по чтению представление базы — по крайней мере. Если такое согласованное по чтению состояние базы данных необходимо получить на уровне транзакции (набора SQL-операторов), то это также можно сделать, как будет показано в разделе "Уровень изоляции `SERIALIZABLE`" далее в главе.

Основным назначением транзакции является перевод базы из одного согласованного состояния в другое. В стандарте ISO языка SQL описаны разнообразные *уровни изоляции транзакций*, которые определяют, насколько *чувствительна* одна транзакция к изменениям, произведенным другой транзакцией. Чем выше уровень чувствительности, тем выше степень изоляции, которую база данных должна обеспечивать между транзакциями, выполняемыми вашим приложением. В следующем разделе мы посмотрим, как через архитектуру многоверсионности с абсолютно минимальным блокированием Oracle может поддерживать каждый из определенных уровней изоляции.

Уровни изоляции транзакций

Стандарт ANSI/ISO языка SQL определяет четыре уровня изоляции транзакций с разными возможными исходами для одного и того же сценария транзакций. То есть одна и та же работа, выполненная в той же манере с теми же самыми входными данными, в зависимости от выбранного уровня изоляции может приводить к получению разных ответов. Эти уровни изоляции определены в терминах трех "феноменов", которые либо разрешены, либо запрещены на заданном уровне изоляции.

- **Грязное чтение (dirty read).** Смысл этого термина так же плох, как он звучит. Вам разрешено читать незафиксированные, или грязные, данные. Вы достигаете такого эффекта, просто открывая файл операционной системы, куда кто-то другой производит запись или чтение, какие бы данные там не оказались. При этом не обеспечивается целостность данных, нарушаются внешние ключи и игнорируются ограничения уникальности.
- **Невоспроизводимое чтение (nonrepeatable read).** Это означает, что если вы читаете строку в момент времени T1 и затем считываете ее повторно в момент времени T2, то за данный промежуток времени строка может измениться, исчезнуть, обновиться и т.д.
- **Фантомное чтение (phantom read).** Это значит, что если вы запустили запрос в момент времени T1, а затем выполните его повторно в момент времени T2, то в базе данных могут появиться дополнительные строки, что повлияет на полученные результаты. Отличие фантомного чтения от невоспроизводимого чтения состоит в том, что уже прочитанные данные не изменяются, но критерию запроса удовлетворяет больший объем данных, чем было до того.

На заметку! Стандарт ANSI/ISO языка SQL определяет характеристики уровня *транзакции*, а не характеристики на уровне индивидуальных операторов. В последующих разделах мы будем исследовать изоляцию уровня транзакций, а не только изоляцию уровня операторов.

Уровни изоляции SQL определены на основе того, допускают ли они каждый из описанных ранее феноменов. Интересно отметить, что стандарт SQL не навязывает специфичную схему блокирования и не регламентирует определенное поведение, а взамен описывает уровни изоляции в терминах этих феноменов, делая возможным существование разнообразных механизмов блокирования/параллелизма (табл. 7.1).

Таблица 7.1. Уровни изоляции ANSI

Уровень изоляции	Грязное чтение	Невоспроизводимое чтение	Фантомное чтение
READ UNCOMMITTED	Разрешено	Разрешено	Разрешено
READ COMMITTED	–	Разрешено	Разрешено
REPEATABLE READ	–	–	Разрешено
SERIALIZABLE	–	–	–

В Oracle явно поддерживаются уровни изоляции READ COMMITTED и SERIALIZABLE в том виде, как они определены стандартом. Однако это еще не все. Стандарт SQL пытался установить уровни изоляции, которые допускают разнообразные степени согласованности запросов, выполняемых на каждом уровне. REPEATABLE READ — это уровень изоляции, при котором стандарт SQL требует гарантии согласованного по чтению результата, получаемого из запроса. В определении этого стандарта READ COMMITTED не дает согласованные результаты, а READ UNCOMMITTED является уровнем, используемым для обеспечения неблокирующих чтений.

Тем не менее, в Oracle уровень READ COMMITTED обладает всеми атрибутами, необходимыми для достижения согласованных по чтению запросов. Во многих других базах данных запросы READ COMMITTED могут и будут возвращать ответы, которые никогда не существовали в базе ни в какой момент времени. Более того, Oracle также поддерживает *дух* уровня READ UNCOMMITTED. Цель предоставления грязного чтения заключается в обеспечении неблокирующего чтения, когда запрос не блокируется и не блокирует обновления некоторых данных. Однако для достижения этой цели Oracle не нуждается в грязных чтениях и не поддерживает их. Грязные чтения вынуждены реализовывать другие СУБД, чтобы предоставить возможность неблокирующих чтений.

Вдобавок к перечисленным выше четырем уровням изоляции SQL в Oracle предлагается еще один уровень — READ ONLY. Транзакция READ ONLY является эквивалентом транзакции REPEATABLE READ или SERIALIZABLE, которая не выполняет никаких модификаций в SQL.

Транзакция, применяющая уровень изоляции READ ONLY, видит только те изменения, которые были зафиксированы *на момент ее начала*, но вставки, обновления и удаления в этом режиме не разрешены (другие сеансы могут обновлять данные, но не транзакция READ ONLY). Используя этот режим, можно достичь уровней изоляции REPEATABLE READ и SERIALIZABLE.

Давайте теперь посмотрим, как именно многоверсионность и согласованность чтения вписывается в схему изоляции, и каким образом базы данных, не поддерживающие многоверсионность, добиваются тех же результатов. Эта информация будет полезна всем, кто работает с другими базами данных и уверен, что понимает, как должны работать уровни изоляции. Интересно также увидеть, каким образом стандарт, предназначенный для устранения различий между базами данных — ANSI/ISO SQL — в действительности позволяет делать это. Стандарт, хотя он и очень детальный, может быть реализован совершенно разными способами.

Уровень изоляции READ UNCOMMITTED

Уровень изоляции READ UNCOMMITTED допускает грязные чтения. В Oracle грязные чтения не применяются и даже не разрешены. Основная цель уровня изоляции READ UNCOMMITTED — предоставить основанное на стандартах определение, которое обслуживает неблокирующие чтения. Как уже было показано, Oracle обеспечивает неблокирующие чтения по умолчанию. Вы бы с трудом решились на блокировку запроса SELECT в базе данных (как отмечалось ранее, имеется специальный случай распределенной транзакции). Каждый одиночный запрос, будь то SELECT, INSERT, UPDATE, MERGE или DELETE, выполняется в согласованной по чтению манере. Причисление оператора UPDATE к запросам может показаться забавным, но это так и есть. Операторы UPDATE содержат два компонента: компонент чтения, определяемый конструкцией WHERE, и компонент записи, определяемый конструкцией SET. Операторы UPDATE выполняют чтение и запись в базу данных; такой возможностью обладают все операторы DML. Случай однострочного оператора INSERT, использующего конструкцию VALUES, является единственным исключением — операторы подобного рода не имеют компонента чтения, а только компонент записи.

В главе 1 метод обеспечения согласованного чтения Oracle был продемонстрирован на примере простого однотабличного запроса, который извлекал строки, удаленные *после* открытия курсора. Теперь мы обратимся к реальному примеру и посмотрим, что происходит в Oracle за счет применения многоверсионности, а также каким образом ведут себя в аналогичной ситуации другие базы данных.

Начнем с той же самой элементарной таблицы и запроса:

```
create table accounts
( account_number number primary key,
  account_balance number not null
);
select sum(account_balance) from accounts;
```

Предположим, что перед началом запроса мы имеем данные, приведенные в табл. 7.2.

Таблица 7.2. Таблица ACCOUNTS перед модификацией

Строка	Номер счета	Баланс счета
1	123	$500.00
2	456	$240.25
...	...	...
342 023	987	$100.00

Теперь наш оператор SELECT начинает свое выполнение и читает строку 1, строку 2 и т.д.

На заметку! В этом примере я вовсе не предполагаю, что строки размещаются на диске в каком-нибудь физическом порядке. На самом деле в таблице не существует первой, второй или последней строки. Есть просто набор строк. Здесь мы предполагаем, что строка 1 в действительности означает “первую прочитанную строку”, строка 2 — “вторую прочитанную строку” и т.д.

В какой-то момент посреди запроса другая транзакция переводит $400 со счета 123 на счет 987. Эта транзакция делает два обновления, но не фиксируется. Таблица теперь выглядит так, как показано в табл. 7.3.

Таблица 7.3. Таблица ACCOUNTS во время модификации

Строка	Номер счета	Баланс счета	Блокирована?
1	123	Значение ($500.00) изменилось на $100.00	X
2	456	$240.25	–
...	...	...	...
342 023	987	Значение ($100.00) изменилось на $500.00	X

Итак, блокированы две строки. Если кто-то попробует обновить их, эта попытка будет заблокирована. До сих пор наблюдаемое нами поведение является более или менее согласованным во всех базах данных. Отличия начнут проявляться в том, что произойдет, когда запрос доберется до блокированных данных.

Когда функционирующий запрос достигает блока данных, содержащего блокированную строку (строку 342 023) в нижней части таблицы, он обнаружит, что данные в этой строке изменились с момента его запуска на выполнение. Для предоставления согласованного (корректного) ответа Oracle создаст в этой точке копию блока, в котором находится данная строка, *в том виде, в каком он существовал на момент начала запроса*. То есть будет прочитано значение $100.00 — значение, которое было при запуске запроса. В сущности Oracle обходит модифицированные данные, в процессе чтения реконструируя их из сегмента отмены (который также называется *сегментом отката* и детально обсуждается в главе 9). Согласованный и корректный ответ возвращается обратно без ожидания, пока транзакция будет зафиксирована.

Теперь база данных, допускающая грязное чтение, просто возвратит значение, которое обнаружит в строке для счета 987 в момент ее чтения — в данном случае это $500.00. Запрос учтет переведенную сумму $400 дважды. Таким образом, он не только выдаст неправильный ответ, но также вернет общую сумму, которая никогда не существовала в базе данных ни в один зафиксированный момент времени. В многопользовательской базе грязное чтение может быть опасной характеристикой, и лично я совершенно не видел в нем хоть какой-нибудь пользы. Пусть вместо перевода транзакция просто добавила $400 на счет 987. Грязное чтение приняло бы во внимание $400.00 и выдало бы “правильный” ответ, не так ли? Хорошо, предположим, что произошел откат незафиксированной транзакции. Мы получаем сумму $400.00, которой в действительности никогда не было в базе данных.

Вывод из этого такой: грязное чтение — это не средство, а источник неприятностей. В Oracle оно просто не нужно. Вы получаете все преимущества грязного чтения (без блокирования) безо всяких некорректных результатов.

Уровень изоляции READ COMMITTED

Уровень изоляции `READ COMMITTED` устанавливает, что транзакция может читать только те данные, которые были зафиксированы в базе данных. Грязные чтения отсутствуют. Могут возникать невоспроизводимые чтения (т.е. повторные чтения той

же строки могут давать разные ответы в одной и той же транзакции) и фантомные чтения (т.е. вновь вставленные и зафиксированные строки становятся видимыми запросу, который не видел их ранее в транзакции). Вероятно, READ COMMITTED является наиболее часто используемым уровнем изоляции во всех приложениях баз данных, будучи стандартным режимом в базах данных Oracle. Другие уровни изоляции встречаются редко.

Однако обеспечение уровня изоляции READ COMMITTED не так просто, как может показаться. Если вы заглянете в табл. 7.1, то все выглядит простым. Очевидно, с учетом изложенных выше правил запрос, выполняемый в любой базе данных с применением уровня изоляции READ COMMITTED, будет вести себя одинаково? *Нет, не будет.* Если запросить несколько строк в одном операторе, то почти во всех других базах данных в зависимости от реализации изоляция READ COMMITTED будет настолько же плохой, как и грязное чтение.

В Oracle за счет использования многоверсионности и согласованных по чтению запросов ответ, получаемый из запроса таблицы ACCOUNTS, для уровня изоляции READ COMMITTED будет таким же, как в примере с READ UNCOMMITTED. СУБД Oracle воссоздает модифицированные данные в том виде, в каком они были на момент начала запроса, возвращая ответ, который содержался в базе данных, когда запрос только запускался.

Давайте теперь посмотрим, как предшествующий пример может работать в режиме READ COMMITTED в других базах данных. Ответ вас удивит. Обратимся к примеру в точке, описанной в предыдущей таблице.

- Мы находимся в середине таблицы. Мы прочитали и просуммировали первые *N* строк.
- Другая транзакция перевела $400.00 со счета 123 на счет 987.
- Транзакция пока не зафиксирована, поэтому строки, содержащие информацию о счетах 123 и 987, блокированы.

Мы знаем, что происходит в СУБД Oracle, когда она добирается до счета 987: она обходит модифицированные данные, выясняет, что они должны выглядеть как $100.00, и завершает обработку. В табл. 7.4 показано, какой ответ может выдать другая база данных, функционирующая в стандартном режиме READ COMMITTED.

Первое, что следует отметить — другая база данных, добравшись до счета 987, блокирует запрос. Сеанс должен ожидать, находясь на этой строке, пока не будет зафиксирована транзакция, удерживающая монопольную блокировку. Это одна из причин, почему многие имеют плохую привычку фиксировать каждый оператор вместо того, чтобы обрабатывать тщательно продуманные транзакции, которые состоят из всех операторов, необходимых для перевода базы данных из одного согласованного состояния в другое. *Обновления служат препятствием для чтений в большинстве других баз данных.* Но что действительно плохо в таком сценарии — то, что мы вынуждаем конечного пользователя ожидать *неправильного* ответа.

Мы получаем ответ, который не существовал в базе данных ни в какой момент времени, как было бы при грязном чтении, но на этот раз еще и заставляем пользователя ждать этого некорректного ответа. В следующем разделе мы посмотрим, что необходимо предпринять в других базах данных, чтобы получить согласованные по чтению, правильные результаты.

Таблица 7.4. Временная шкала в отличной от Oracle базе данных, использующей изоляцию READ COMMITTED

Момент времени	Запрос	Транзакция перевода между счетами
T1	Читает строку 1, счет 123, значение = $500. К этому моменту сумма = $500.00	–
T2	Читает строку 2, счет 456, значение = $240.25. К этому моменту сумма = $740.25	–
T3	–	Обновляет строку 1 (счет 123) и помещает на нее монопольную блокировку, предотвращая другие обновления и чтения. Строка 1 содержала $500.00, теперь она содержит $100.00
T4	Читает строку *N*. Сумма = ...	–
T5	–	Обновляет строку 342 023 (счет 987) и помещает на нее монопольную блокировку. Строка 342 023 содержала $100.00, теперь она содержит $500.00
T6	Пытается прочитать строку 342 023, счет 987. Обнаруживает, что она блокирована. Сеанс ожидает снятия блокировки с этой строки. *Вся обработка в этом запросе останавливается*	–
T7	–	Фиксирует транзакцию
T8	Читает строку 342 023, счет 987. Видит в ней $500.00 и предоставляет окончательный ответ, в котором дважды учтена сумма $400.00	–

Важный урок, который следует здесь извлечь, заключается в том, что разные базы данных, реализуя один и тот же очевидно безопасный уровень изоляции, в совершенно одинаковых обстоятельствах могут и будут возвращать значительно отличающиеся ответы. Важно понимать, что неблокирующие чтения в Oracle обеспечиваются не за счет корректности ответов. Иногда можно не только иметь мед, но и есть его ложкой.

Уровень изоляции REPEATABLE READ

Целью REPEATABLE READ является обеспечение такого уровня изоляции, который дает согласованные, корректные ответы и предотвращает потерянные обновления. Мы рассмотрим примеры того и другого, увидим, что нужно делать в Oracle для достижения этих целей, и исследуем происходящее в других системах.

Получение согласованного ответа

Если установлен уровень изоляции REPEATABLE READ, то результаты заданного запроса должны быть согласованными на определенный момент времени.

Большинство баз данных (не Oracle) добиваются воспроизводимых чтений через применение разделяемых блокировок чтения на уровне строки. Это, конечно же, снижает степень параллелизма. Для обеспечения согласованных по чтению ответов в Oracle была выбрана более параллельная, многоверсионная модель.

В Oracle с использованием многоверсионности мы получаем ответ, согласованный на момент времени, когда запрос начал свое выполнение. В других базах данных с применением разделяемых блокировок чтения мы получаем ответ на момент завершения запроса — т.е. тогда, когда вообще можно получить ответ (об этом речь пойдет далее в главе).

В системе, в которой для обеспечения воспроизводимых чтений задействована разделяемая блокировка чтения, мы можем наблюдать в таблице строки, заблокированные процессом, который их обрабатывает. Таким образом, в приведенном ранее примере при чтении таблицы ACCOUNTS запрос оставляет разделяемые блокировки чтения на каждой строке (табл. 7.5).

Таблица 7.5. Временная шкала 1 в отличной от Oracle базе данных, использующей изоляцию READ REPEATABLE

Момент времени	Запрос	Транзакция перевода между счетами
T1	Читает строку 1. К этому моменту сумма = $500.00. Устанавливает разделяемую блокировку чтения на строку 1	–
T2	Читает строку 2. К этому моменту сумма = $740.25. Устанавливает разделяемую блокировку чтения на строку 2	–
T3	–	Пытается обновить строку 1, но она заблокирована. Транзакция приостанавливается до тех пор, пока не сможет получить монопольную блокировку
T4	Читает строку *N*. Сумма = ...	–
T5	Читает строку 342 023, видит $100.00 и возвращает окончательный ответ	–
T6	Фиксирует транзакцию	–
T7	–	Обновляет строку 1 и помещает монопольную блокировку на эту строку. В строке теперь $100.00
T8	–	Обновляет строку 342 023 и помещает монопольную блокировку на эту строку. В строке теперь $500.00. Фиксирует транзакцию

В табл. 7.5 легко заметить, что мы теперь получаем корректный ответ, но за счет физического блокирования одной транзакции и выполнения двух транзакций последовательно. Так проявляется один из побочных эффектов разделяемых блокировок чтения для согласованных ответов: *процессы чтения данных будут блокировать*

процессы записи данных. Это еще и в дополнение к тому факту, что в таких системах процессы записи данных будут блокировать процессы чтения. Только представьте, если бы банкоматы работали подобным образом в реальной жизни.

Итак, вы видите, что разделяемые блокировки чтения могут подавлять параллелизм, но они также могут вызывать появление фиктивных ошибок. В табл. 7.6 мы начинаем с исходной таблицы ACCOUNTS, но на этот раз цель заключается в переводе $50.00 со счета 987 на 123.

Таблица 7.6. Временная шкала 2 в отличной от Oracle базе данных, использующей изоляцию READ REPEATABLE

Момент времени	Запрос	Транзакция перевода между счетами
T1	Читает строку 1. К этому моменту сумма = $500.00. Устанавливает разделяемую блокировку чтения на строку 1	–
T2	Читает строку 2. К этому моменту сумма = $740.25. Устанавливает разделяемую блокировку чтения на строку 2	–
T3	–	Обновляет строку 342 023 и помещает монопольную блокировку на строку 342 023, предотвращая другие обновления и установку разделяемых блокировок чтения. Строка теперь содержит $50.00
T4	Читает строку *N*. Сумма = ...	–
T5	–	Пытается обновить строку 1, но она заблокирована. Транзакция приостанавливается до тех пор, пока не сможет получить монопольную блокировку
T6	Пытается прочитать строку 342 023, но не может, т.к. установлена монопольная блокировка	–

Мы только что получили классическое условие взаимоблокировки. Наш запрос удерживает ресурсы, в которых нуждается обновление, и наоборот. Транзакция нашего запроса взаимно заблокирована транзакцией обновления. Одна из них должна быть выбрана в качестве жертвы и уничтожена. Мы потратили много времени и ресурсов лишь для того, чтобы в конце потерпеть неудачу и произвести откат. Это второй побочный эффект разделяемых блокировок чтения: *процессы чтения и процессы записи данных могут и часто будут попадать в ситуацию взаимоблокировки друг друга*.

В Oracle обеспечивается согласованность чтения на уровне операторов без блокирования операций записи операциями чтения или взаимоблокировок. В Oracle никогда не применяются разделяемые блокировки чтения — *вообще никогда*. Разработчики Oracle избрали трудную в реализации, но обеспечивающую намного более высокую степень параллелизма схему многоверсионности.

Потерянные обновления: еще одна проблема переносимости

Распространенное использование REPEATABLE READ в базах данных, эксплуатирующих разделяемые блокировки чтения, связано с предотвращением потерянных обновлений.

На заметку! Обнаружение потерянных обновлений и решения проблемы потерянных обновлений обсуждаются в главе 6.

При наличии уровня изоляции REPEATABLE READ в базе данных, которая *задействует разделяемые блокировки чтения* (и не поддерживает многоверсионность), ошибки потерянных обновлений происходить не могут. Причина отсутствия потерянных обновлений в этих базах данных связана с тем, что простая выборка данных *оставляет на них блокировку, и данные, однажды прочитанные нашей транзакцией, не могут быть модифицированы никакой другой транзакцией*. Теперь, если в приложении предполагалось, что REPEATABLE READ означает "потерянные обновления не возникнут", то вы столкнетесь с неприятным сюрпризом при переносе приложения в базу данных, в которой не применяются разделяемые блокировки чтения в качестве лежащего в основе механизма управления параллелизмом.

На заметку! В среде, не поддерживающей состояние, такой как веб-приложение, потерянные обновления, скорее всего, станут причиной проблем — даже при уровне изоляции REPEATABLE READ. Это объясняется тем, что единственный сеанс базы данных используется многими клиентами через пул подключений, а блокировки между вызовами не удерживаются. Уровень изоляции REPEATABLE READ предотвращает потерянные обновления только в среде с поддержкой состояния вроде той, которую можно наблюдать в клиент-серверном приложении.

Хотя это звучит неплохо, вы должны помнить, что оставление разделяемых блокировок чтения на всех данных после их чтения, разумеется, всерьез ограничит возможности параллельных операций чтения и модификации. Таким образом, хотя указанный уровень изоляции в этих базах данных предназначен для предотвращения потерянных обновлений, это достигается полным исключением возможности выполнения параллельных операций! Иметь мед и есть его ложкой можно не всегда.

Уровень изоляции SERIALIZABLE

Обычно SERIALIZABLE считается наиболее ограничивающим уровнем изоляции транзакций, но он предлагает максимальную степень изоляции. Транзакция SERIALIZABLE работает в среде, которая позволяет сделать вид, будто бы в базе данных нет других пользователей, изменяющих данные. Любая прочитанная строка гарантированно будет той же самой при ее повторном чтении, и любой выполняемый запрос гарантированно возвратит те же самые результаты на протяжении времени жизни транзакции. Например, если выполнить приведенные ниже запросы, то ответы, полученные из таблицы T, будут одинаковыми, хотя между ними прошло 24 часа (или может возникнуть ошибка ORA-01555: snapshot too old (ORA-01555: устаревший снимок), о которой речь пойдет в главе 8):

```
select * from T;
begin dbms_lock.sleep( 60*60*24 ); end;
select * from T;
```

Уровень изоляции SERIALIZABLE гарантирует, что эти два запроса будут всегда возвращать те же самые результаты. Побочные эффекты (изменения), сделанные другими транзакциями, для этого запроса остаются невидимыми независимо от того, сколько времени он выполнялся.

В Oracle транзакция SERIALIZABLE реализована так, что согласованность чтения, которую мы обычно получаем на уровне оператора, расширяется до уровня транзакции.

На заметку! Как отмечалось ранее, в Oracle также существует уровень изоляции, обозначенный как READ ONLY. Он обладает всеми характеристиками уровня изоляции SERIALIZABLE, но запрещает модификацию. Следует отметить, что пользователь SYS (или пользователь, подключенный с привилегией SYSDBA) *не может* иметь транзакцию READ ONLY или SERIALIZABLE. В этом отношении пользователь SYS является особенным.

Вместо обеспечения согласованности результатов на момент запуска оператора это делается для момента начала транзакции. Другими словами, Oracle применяет сегменты отката для воссоздания данных в том виде, в каком они существовали, когда начиналась транзакция, а не оператор.

Здесь скрыта довольно глубокая мысль: база данных уже знает ответ на любой вопрос, который вы можете задать, еще до того, как вы спросите.

Уровень изоляции SERIALIZABLE обходится не бесплатно, и ценой является возможность возникновения следующей ошибки:

```
ERROR at line 1:
ORA-08177: can't serialize access for this transaction
```

```
ОШИБКА в строке 1:
ORA-08177: не удается сериализовать доступ для этой транзакции
```

Вы будете получать это сообщение при каждой попытке обновления строки, которая изменилась с момента начала транзакции.

На заметку! СУБД Oracle пытается делать это исключительно на уровне строки, но вы можете получить ошибку ORA-08177, даже когда строка, которую вы хотите модифицировать, не была изменена. Ошибка ORA-08177 может возникнуть из-за того, что была модифицирована какая-то другая строка (или строки) из блока, в котором находится интересующая вас строка.

В Oracle принят оптимистический подход к сериализации — ставка делается на тот факт, что данные, которые ваша транзакция собирается обновить, не будут обновлены какой-то другой транзакцией. Обычно так и происходит, поэтому ставка выигрывает, особенно в системах типа OLTP с быстрыми транзакциями. Если никто другой не обновляет данные во время вашей транзакции, то этот уровень изоляции, который обычно будет снижать степень параллелизма в остальных системах, предоставит здесь ту же самую степень параллелизма, что и без транзакций SERIALIZABLE.

Недостаток заключается в том, что можно получить ошибку ORA-08177, если ставка оказалась проигрышной. Однако если подумать, то такой риск вполне оправдан. Если вы используете транзакции SERIALIZABLE, то не должны ожидать обновлений информации, с которой работают другие транзакции. В противном случае необходимо применять SELECT...FOR UPDATE, как было описано в главе 1, и это сериализирует доступ. Таким образом, использование уровня изоляции SERIALIZABLE достижимо и эффективно, если соблюдаются следующие условия.

- Высока вероятность того, что никто другой не модифицирует те же самые данные.
- Требуется согласованность чтения на уровне транзакции.
- Транзакции будут короткими (что поможет сделать реальным первое условие).

В Oracle считают этот метод достаточно масштабируемым для прогона всех их эталонных тестов TPC-C (эталонный тест OLTP, соответствующий индустриальным стандартам; подробные сведения доступны по адресу http://www.tpc.org). Во многих других реализациях вы обнаружите, что те же самые цели достигаются посредством разделяемых блокировок чтения с присущими им взаимоблокировками и блокированием. В Oracle мы не столкнемся с каким-либо блокированием, но получим ошибку ORA-08177, если другие сеансы будут модифицировать данные, которые мы также хотим изменять. Однако эта ошибка возникает не так часто, как взаимоблокировки и блокирование в других системах.

Но — всегда есть какое-то "но" — вы должны четко понимать разницу между уровнями изоляции и их реализациями. Помните, что при установке уровня изоляции в SERIALIZABLE вы не увидите никаких изменений, внесенных в базу данных после запуска вашей транзакции — вплоть до ее фиксации. Приложения, которые пытаются поддерживать собственные ограничения целостности данных, такие как планировщик ресурсов, описанный в главе 1, в связи с этим должны предпринимать дополнительные меры предосторожности. Если помните, в главе 1 была описана проблема, которая заключалась в невозможности принудительного применения ограничения целостности в многопользовательской системе из-за того, что мы не могли видеть изменения, сделанные незафиксированными сеансами. В случае использования уровня SERIALIZABLE мы по-прежнему не увидим незафиксированных изменений, но также не увидим и зафиксированных изменений, произведенных после начала нашей транзакции!

Наконец, имейте в виду, что уровень изоляции SERIALIZABLE *вовсе не* означает, что все транзакции, запущенные пользователями, будут вести себя так, как будто они выполняются друг за другом в последовательной манере. Это не подразумевает наличие какого-то последовательного упорядочения транзакций, что приведет к тому же самому результату. Феномены, ранее описанные в стандарте SQL, не позволяют такому случиться. Последнее утверждение отражает часто неправильно понимаемую концепцию, которую поможет прояснить небольшая демонстрация. В табл. 7.7 представлены два сеанса, выполняющие работу на протяжении некоторого времени. Изначально пустые таблицы базы данных A и B создаются следующим образом:

```
EODA@ORA12CR1> create table a ( x int );
Table created.
Таблица создана.
```

```
EODA@ORA12CR1> create table b ( x int );
Table created.
Таблица создана.
```

Мы имеем последовательность событий, описанную в табл. 7.7.

Таблица 7.7. Пример транзакции SERIALIZABLE

Момент времени	Действия в сеансе 1	Действия в сеансе 2
T1	`alter session set isolation_level=serializable;`	–
T2	–	`alter session set isolation_level=serializable;`
T3	`insert into a select count(*) from b;`	–
T4	–	`insert into b select count(*) from a;`
T5	`commit;`	–
T6	–	`commit;`

Теперь, когда все показанное сделано, таблицы A и B будут содержать строку со значением 0 каждая. Если бы существовало некоторое последовательное упорядочение транзакций, присутствие значения 0 в обеих таблицах было бы невозможным. Если бы сеанс 1 выполнялся *во всей своей полноте* перед сеансом 2, то таблица B имела бы строку со значением 1 в ней. Если бы сеанс 2 выполнялся *полностью* перед сеансом 1, то таблица A содержала бы строку со значением 1. Однако на самом деле мы получили строки со значением *0* в обеих таблицах. Каждая транзакция выполнялась так, как будто в конкретный момент времени она была единственной в базе данных. Независимо от того, сколько раз сеанс 1 выдавал запросы к таблице B, и независимо от состояния фиксации сеанса 2, счетчик получит значение, которое было зафиксировано в базе данных на момент времени T1. Аналогично, не играет роли, сколько запросов к таблице A производил сеанс 2 — счетчик имеет значение, которое было в момент T2.

Уровень изоляции READ ONLY

Транзакции READ ONLY очень похожи на транзакции SERIALIZABLE с единственным отличием: они не допускают модификаций, поэтому не восприимчивы к ошибке ORA-08177. Транзакции READ ONLY предназначены для поддержки потребностей, возникающих при формировании отчетов, когда содержимое отчета должно быть согласовано с состоянием на определенный момент времени. В других системах вам придется применять транзакцию REPEATABLE READ и страдать от последствий, вызванных разделяемой блокировкой чтения. В Oracle вы будете использовать транзакцию READ ONLY. В этом режиме вывод, генерируемый в отчете, который применяет 50 операторов SELECT для сбора данных, будет согласованным по состоянию на единственный момент времени — момент начала транзакции. Вы получите возможность делать это, не блокируя ни единой порции данных.

Эта цель достигается использованием той же самой многоверсионности, которая применялась для индивидуальных операторов. Данные реконструируются из сегментов отката по мере необходимости и предоставляются в том виде, в каком они пребывали на начало формирования отчета. Тем не менее, транзакции READ ONLY не лишены недостатков. В то время как в транзакциях SERIALIZABLE можно столкнуться с ошибкой ORA-08177, в транзакциях READ ONLY следует ожидать ошибки ORA-01555: snapshot too old (ORA-01555: устаревший снимок). Это будет происходить в системах, где другие пользователи активно модифицируют информацию, которую вы читаете. Изменения, вносимые в эту информацию, сохраняются в сегментах отката. Но сегменты отката используются в циклическом режиме — почти так же, как журнальные файлы повторения действий.

Чем дольше требуется времени на формирование отчета, тем выше вероятность, что какая-то операция отката, необходимая для воссоздания ваших данных, уже не сможет найти необходимых сведений в сегменте отката. Сегмент отката начинает перезаписываться другой транзакцией, и нужная вам часть информации утрачивается. В этот момент вы получаете ошибку ORA-01555 и вынуждены начинать заново.

Единственное решение этой неприятной проблемы предусматривает корректную установку размера табличного пространства отката для системы. Я постоянно сталкиваюсь с людьми, которые пытаются сэкономить несколько мегабайт дискового пространства, создавая табличное пространство отката с минимально возможным размером ("Зачем тратить пространство на то, в чем я действительно не нуждаюсь?" — примерно так они думают).

Проблема в том, что табличное пространство отката является ключевым компонентом при работе базы данных, и если его размер определен неправильно, вы обязательно столкнетесь с упомянутой выше ошибкой. В течение многих лет работы с версиями Oracle 6, 7, 8, 9, 10, 11 и 12 я никогда не сталкивался с ошибкой ORA-01555 за пределами тестовой системы или системы разработки. В случае возникновения такой ошибки вы должны знать, что некорректно задали размер табличного пространства отката, и это необходимо исправить. Мы вернемся к этой теме в главе 9.

Последствия многоверсионной согласованности чтения

До настоящего времени мы видели, каким образом многоверсионность обеспечивает возможность неблокирующего чтения, и я подчеркивал, что это хорошая вещь: согласованные (корректные) ответы при высокой степени параллелизма. Что же в этом может быть плохого? Если вы четко не понимаете весь механизм и его последствия, то вполне вероятно, что спроектируете некоторые из своих транзакций неправильно.

Вспомните пример с планированием ресурсов из главы 1, в котором мы пытались задействовать ряд приемов ручной блокировки (через SELECT FOR UPDATE для сериализации модификаций записей о ресурсах в таблице SCHEDULES). Но может ли это затронуть нас другими путями? Однозначно может. В последующих разделах приведены соответствующие детали.