|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://www.intuit.ru/img/empty.gif |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Транзакции*****Транзакцией*** называется последовательность действий, которая или полностью фиксируется в базе данных, или полностью отменяется. Иногда под транзакцией также подразумевают не группу SQL-операторов, а интервал времени, выполняемые в течение которого SQL-операторы можно или все зафиксировать или все отменить.Так, операция перевода денег с одного счета на другой должна составлять единую *транзакцию*. Иначе может возникнуть ситуация, когда первый SQL-оператор переведет деньги на другой счет, а второй, выполняющий снятие их со счета, не доведет дело до конца из-за непредвиденного сбоя.***Фиксация транзакции*** может производиться принудительно по SQL-оператору или неявно после завершения каждого SQL-оператора. Во втором случае применяется режим *автокоммита*. Как правило, выполнение SQL-операторов в интерактивном режиме всегда использует *автокоммит* .Очень часто в интегрированных средах разработки классы, инкапсулирующие работу с базой данных, по умолчанию предполагают режим *автокоммита* .Следующая схема демонстрирует принцип использования *транзакций.*Принцип использования транзакцийПринцип использования транзакцийНовая *транзакция* начинается с начала каждого сеанса работы с базой данных. Далее все выполняемые SQL-операторы будут входить в одну *транзакцию* до тех пор, пока не будет выполнен оператор COMMIT WORK или ROLLBACK WORK.Оператор COMMIT WORK завершает текущую *транзакцию*, выполняя фиксацию сделанных изменений в базе данных. Иногда говорят, что оператор COMMIT WORK фиксирует *транзакцию*.Оператор ROLLBACK WORK выполняет откат *транзакции*, отменяя действие всех SQL-операторов, выполненных в текущей *транзакции*.Логически *транзакция* должна объединять только выполнение взаимосвязанных операций. Так, если делать *транзакции* "очень большими", состоящими из последовательности не связанных между собой операторов, то любой сбой, автоматически выполняющий откат *транзакции*, повлияет на отмену действий, которые могли бы быть успешно завершены при более "коротких"*транзакциях*.**Автоматическая фиксация изменений**Большинство коммерческих СУБД позволяет устанавливать режим автоматической фиксации изменений - ***автокоммит***.Для установки этого режима используется (но не всеми СУБД) оператор SET AUTOCOMMIT ON;, а для отмены режима - SET AUTOCOMMIT OFF;.**Проблемы параллельного доступа с использованием транзакций**При параллельном использовании транзакций могут возникать следующие проблемы:* *неповторяющееся чтение (non-repeatable read);*
* ***"грязное" чтение (dirty read)*** - чтение данных, которые были записаны откатанной транзакцией;
* *потерянное обновление (lost update);*
* *фантомная вставка (phantom insert).*

Рассмотрим ситуации, в которых возможно возникновение данных проблем.**Неповторяющееся чтение**Предположим, имеются две *транзакции*, открытые различными приложениями, в которых выполнены следующие SQL-операторы:

|  |  |
| --- | --- |
| **Транзакция 1** | **Транзакция 2** |
|

|  |
| --- |
| SELECT f2 FROM tbl1 WHERE f1=1; |
| UPDATE tbl1 SET f2=f2+1 WHERE f1=1; |
|   |

 |

|  |
| --- |
| SELECT f2 FROM tbl1 WHERE f1=1; |
|   |
| SELECT f2 FROM tbl1 WHERE f1=1; |

 |

В *транзакции* 2 выбирается значение поля f2, затем в *транзакции* 1 изменяется значение поля f2. При повторной попытке выбора значения из поля f2 в *транзакции* 1 будет получен другой результат ([рис. 7.1](http://www.intuit.ru/department/database/cdba/7/cdba_7.html#image.7.1)). Эта ситуация особенно неприемлема, когда данные считываются с целью их частичного изменения и обратной записи в базу данных.Неповторяющееся чтение**Рис. 7.1.**  Неповторяющееся чтение**"Грязное" чтение**Предположим, имеется две *транзакции*, открытые различными приложениями, в которых выполнены следующие SQL-операторы:

|  |  |
| --- | --- |
| **Транзакция 1** | **Транзакция 2** |
|

|  |
| --- |
| SELECT f2 FROM tbl1 WHERE f1=1; |
| UPDATE tbl1 SET f2=f2+1 WHERE f1=1; |
|   |
| ROLLBACK WORK; |

 |

|  |
| --- |
|   |
|   |
| SELECT f2 FROM tbl1 WHERE f1=1; |
|   |

 |

В *транзакции* 1 изменяется значение поля f1, а затем в *транзакции* 2 выбирается значение поля f2. После этого происходит откат *транзакции* 1. В результате значение, полученное второй *транзакцией*, будет отличаться от значения, хранимого в базе данных.**Потерянное обновление**Предположим, имеется две *транзакции*, открытые различными приложениями, в которых выполнены следующие SQL-операторы:

|  |  |
| --- | --- |
| **Транзакция 1** | **Транзакция 2** |
|

|  |
| --- |
| SELECT f2 FROM tbl1 WHERE f1=1; |
| UPDATE tbl1 SET f2=20 WHERE f1=1; |
|   |

 |

|  |
| --- |
| SELECT f2 FROM tbl1 WHERE f1=1; |
|   |
| UPDATE tbl1 SET f2=25 WHERE f1=1; |

 |

В *транзакции* 1 изменяется значение поля f1, а затем в *транзакции* 2 также изменяется значение этого поля. В результате изменение, выполненное первой *транзакцией*, будет потеряно.**Фантомная вставка**Предположим, имеется две *транзакции*, открытые различными приложениями, в которых выполнены следующие SQL-операторы:

|  |  |
| --- | --- |
| **Транзакция 1** | **Транзакция 2** |
|

|  |
| --- |
|   |
| INSERT INTO tbl1 (f1,f2) VALUES (15,20); |
|   |

 |

|  |
| --- |
| SELECT SUM(f2) FROM tbl1; |
|   |
| SELECT SUM(f2) FROM tbl1; |

 |

В *транзакции* 2 выполняется SQL-оператор, использующий все значения поля f2. Затем в *транзакции* 1 выполняется вставка новой строки, приводящая к тому, что повторное выполнение SQL-оператора в *транзакции*2 выдаст другой результат. Такая ситуация называется *фантомной вставкой* и является частным случаем *неповторяющегося чтения*. При этом, если выполняемый SQL-оператор выбирает не все значения поля f2, а только значение одной строки таблицы (используется предикат WHERE), то выполнение оператора INSERT не приведет к ситуации фантомной вставки.**Уровни изоляции**Стандарт SQL-92 определяет *уровни изоляции*, установка которых предотвращает определенные конфликтные ситуации.Введены следующие четыре *уровня изоляции*:* *SERIALIZABLE - последовательное выполнение* (используется по умолчанию). Этот уровень гарантирует предотвращение всех описанных выше конфликтных ситуаций, но, соответственно, при нем наблюдается самая низкая степень параллелизма;
* *REPEATABLE READ - повторяющееся чтение*. На этом уровне разрешено выполнение операторов INSERT, приводящих к конфликтной ситуации "фантомная вставка". Этот уровень целесообразно использовать, если на выполняющиеся SQL-операторы не влияет добавление новых строк;
* *READ COMMITTED - фиксированное чтение*. Этот уровень позволяет получать разные результаты для одинаковых запросов, но только после *фиксации транзакции*, повлекшей изменение данных;
* READ UNCOMMITTED - нефиксированное чтение. Здесь возможно получение разных результатов для одинаковых запросов без учета фиксации транзакции.

В следующей таблице приводится формальное описание уровней изоляции.

|  |  |
| --- | --- |
| **Уровень изоляции** | **Предотвращение конфликтной ситуации** |
| **неповторяющееся чтение (non-repeatable read)** | **"грязное" чтение (dirty read)** | **потерянное обновление (lost update)** | **фантомная вставка (phantom insert)** |
| SERIALIZABLE | + | + | + | + |
| REPEATABLE READ | + | + | + | - |
| READ COMMITTED | - | + | + | - |
| READ UNCOMMITTED | - | - | + | - |

**Блокировки*****Блокировками*** (locks) называются механизмы, применяемые для управления параллельными изменениями данных.Существует два типа *блокировок*:* ***оптимистические блокировки*** (optimistic locks) - предотвращают возникновение конфликтных ситуаций, выполняя при необходимости откат *транзакции* (такие *блокировки* в стандарте SQL-92 не поддерживаются);
* ***пессимистические блокировки*** (pessimistic locks) - предотвращают возникновение конфликтных ситуаций, блокируя одновременный доступ к данным для одновременных *транзакций*.

*Блокировки* используются для приостановки выполнения одних SQL-операторов, пока выполняются другие.Если при *пессимистической блокировке* выполнен SQL-оператор, который может вызвать конфликтную ситуацию для другого SQL-оператора, то выполнение второго SQL-оператора будет приостановлено. При *оптимистической блокировке* могут выполняться любые SQL-операторы, но в случае возникновения конфликтной ситуации все изменения будут потеряны.*Блокировки*, используемые уровнями изоляции, подразделяются на:* ***разделяемые блокировки (S-locks)***, которые могут одновременно устанавливаться несколькими пользователями;
* ***исключительные блокировки*** (X-locks), которые устанавливаются только одним пользователем, получающим эксклюзивный доступ к данным.

Существуют следующие логические и физические уровни*блокировок*:* *блокировка на уровне таблицы* (table-level locking);
* *блокировка на уровне строк* (row-level locking);
* *блокировка на уровне элемента таблицы* (item-level locking);
* *блокировка на уровне БД* (dbspace-level locking);
* *блокировка на уровне табличного пространства* (tablespace-level locking);
* *блокировка на уровне страницы или блока* (page-level locking).

**Определение параметров транзакции**Определение параметров транзакции выполняется оператором SET TRANSACTION, который имеет в стандарте SQL-92 следующее формальное описание:TRANSACTION { ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE } | { READ ONLY | READ WRITE } | { DIAGNOSTICS SIZE count\_message } } , ... ; Фраза ISOLATION LEVEL указывает устанавливаемый *уровень изоляции*.Фраза READ ONLY устанавливает режим, при котором разрешается только чтение. Этот режим устанавливается по умолчанию, если *уровень изоляции* определен как READ UNCOMMITTED.При режиме READ ONLY на данные не устанавливается никаких *блокировок*.Фраза READ WRITE устанавливает режим, который разрешает как чтение, так и запись данных. При этом режиме *уровень изоляции* не может быть установлен как READ UNCOMMITTED.Режим READ WRITE устанавливается по умолчанию для любого уровня, отличного от READ UNCOMMITTED.Фраза DIAGNOSTICS SIZE определяет размер области, используемой для записи диагностических сообщений, доступ к которым осуществляется оператором GET DIAGNOSTICS.Например, для определения *транзакции*, предотвращающей все описанные выше конфликтные ситуации, следует выполнить SQL-операторSET TRANSACTION ISOLATION LEVEL SERIALIZABLE; |

 |  |

**Понятие курсора**

Запрос к реляционной базе данных обычно возвращает несколько рядов (записей) данных, но приложение за один раз обрабатывает лишь одну запись. Даже если оно имеет дело одновременно с несколькими рядами (например, выводит данные в форме электронных таблиц), их количество по-прежнему ограничено. Кроме того, при модификации, удалении или добавлении данных рабочей единицей является ряд. В этой ситуации на первый план выступает концепция *курсора*, и в таком контексте курсор – указатель на ряд.

***Курсор*** в SQL – это область в памяти базы данных, которая предназначена для хранения последнего оператора SQL. Если текущий оператор – запрос к базе данных, в памяти сохраняется и строка данных запроса, называемая текущим значением, или текущей строкой *курсора*. Указанная область в памяти поименована и доступна для прикладных программ.

Обычно *курсоры* используются для выбора из базы данных некоторого подмножества хранимой в ней информации. В каждый момент времени прикладной программой может быть проверена одна строка *курсора*. *Курсоры* часто применяются в операторах SQL, встроенных в написанные на языках процедурного типа прикладные программы. Некоторые из них неявно создаются сервером базы данных, в то время как другие определяются программистами.

В соответствии со стандартом SQL при работе с *курсорами* можно выделить следующие основные *действия*:

* создание или *объявление курсора* ;
* ***открытие курсора,*** т.е. наполнение его данными, которые сохраняются в многоуровневой памяти ;
* *выборка из курсора* и *изменение* с его помощью строк данных;
* *закрытие курсора*, после чего он становится недоступным для пользовательских программ;
* *освобождение курсора*, т.е. удаление *курсора* как объекта, поскольку его *закрытие* необязательно освобождает ассоциированную с ним память.

В разных реализациях определение *курсора* может иметь некоторые отличия. Так, например, иногда разработчик должен явным образом освободить выделяемую для *курсора* память. После*освобождения курсора* ассоциированная с ним память также освобождается. При этом становится возможным повторное использование его имени. В других реализациях при *закрытии курсора* освобождение памяти происходит неявным образом. Сразу после восстановления она становится доступной для других операций: *открытие другого курсора* и т.д.

В некоторых случаях применение *курсора* неизбежно. Однако по возможности этого следует избегать и работать со стандартными командами обработки данных: SELECT, UPDATE, INSERT,DELETE. Помимо того, что *курсоры* не позволяют проводить операции *изменения* над всем объемом данных, скорость выполнения операций обработки данных посредством *курсора* заметно ниже, чем у стандартных средств SQL.