Оглавление

Транзакции

Транзакции – это механизм, поддерживаемый SQL сервером, позволяющий выполнить группу операций модификации данных целиком, либо не выполнить вообще.

Транзакция является единственной единицей работы. Если транзакция выполнена успешно, все модификации данных, сделанные в течение транзакции, принимаются и становятся постоянной частью базы данных. Если в результате выполнения транзакции происходят ошибки и должна быть произведена отмена или выполнен откат, все модификации данных будут отменены.

Требования ACID

Это требования к транзакционной системе (например, к СУБД), обеспечивающие наиболее надёжную и предсказуемую её работу. Требования ACID состоят из четырех пунктов:

- **Atomicity — Атомарность**

Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback): если транзакцию не удаётся полностью завершить, результаты всех её до сих пор произведённых действий будут отменены и система вернётся во «внешне исходное» состояние — со стороны будет казаться, что транзакции и не было.

- **Consistency — Согласованность**

Транзакция, достигающая своего нормального завершения (EOT — end of transaction, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных. Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты. Это условие является необходимым для поддержки четвёртого свойства.

Согласованность является более широким понятием. Например, в банковской системе может существовать требование равенства суммы, списываемой с одного счёта, сумме, зачисляемой на другой. Это бизнес-правило и оно не может быть гарантировано только проверками целостности, его должны соблюсти программисты при написании кода транзакций. Если какая-либо транзакция произведёт списание, но не произведёт зачисления, то система останется в некорректном состоянии и свойство согласованности будет нарушено.

Наконец, ещё одно замечание касается того, что в ходе выполнения транзакции согласованность не требуется. В нашем примере, списание и зачисление будут, скорее всего, двумя разными подоперациями и между их выполнением внутри транзакции будет видно несогласованное состояние системы. Однако не нужно забывать, что при выполнении требования изоляции никаким другим транзакциям эта несогласованность не будет видна. А атомарность гарантирует, что транзакция либо будет полностью завершена, либо ни одна из операций транзакции не будет выполнена. Тем самым эта промежуточная несогласованность является скрытой.

- **Isolation — Изолированность**

Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат. Изолированность — требование дорогое, поэтому в реальных БД существуют режимы, не полностью изолирующие транзакцию (уровни изолированности Repeatable Read и ниже).

- **Durability — Устойчивость**

Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

Уровни изолированности транзакции

Уровни изоляции транзакции определяет степень, в которой одна транзакция должна быть изолирована от изменений данных, сделанных любой другой транзакцией в системе базы данных. Уровень изоляции транзакции определяется следующими явлениями, возникающими при параллельном выполнении транзакций:

Dirty read (Грязное чтение) - Грязное чтение - это ситуация, когда транзакция считывает данные, которые еще не были зафиксированы другой транзакцией. Например, транзакция T1 обновляет строку и оставляет ее незафиксированной, а транзакция T2 считывает обновленную строку. Если транзакция T1 откатывает изменения, транзакция T2 будет считывать данные, которые, как считается, никогда не существовали.

Non Repeatable read (Неповторяемое чтение) - Неповторяемое чтение происходит, когда транзакция читает одну и ту же строку дважды, и каждый раз получает другое значение. Например, предположим, что транзакция T1 считывает данные. Из-за параллелизма другая транзакция T2 обновляет те же данные и фиксирует. Теперь, если транзакция T1 перечитывает те же данные, она получит другое значение.

Phantom read (Фантомное чтение) Фантомное чтение происходит, когда выполняются два одинаковых запроса, но строки, полученные этими двумя запросами различны. Например, предположим, что транзакция T1 извлекает набор строк, которые удовлетворяют некоторым критериям поиска. Теперь транзакция T2 генерирует несколько новых строк, которые соответствуют критериям поиска для транзакции T1. Если транзакция T1 повторно выполняет инструкцию, которая читает строки, на этот раз она получает другой набор строк.

Основываясь на этих явлениях, стандарт SQL определяет четыре уровня изоляции:

Read Uncommitted - это самый низкий уровень изоляции. На этом уровне одна транзакция может считывать еще не зафиксированные изменения, сделанные другой транзакцией, что допускает грязное чтение. На этом уровне транзакции не изолированы друг от друга.

Read Committed - этот уровень изоляции гарантирует, что любое чтение данных фиксируется в момент их чтения. Таким образом, при таком уровне изолированности не допускается грязное чтение. Транзакция удерживает блокировку чтения или записи в текущей строке и, таким образом, не позволяет другим транзакциям читать, обновлять или удалять ее.

Repeatable Read - это наиболее строгий уровень изоляции. Транзакция удерживает блокировки на чтение всех строк, на которые она ссылается, и записывает блокировки на все строки, которые она вставляет, обновляет или удаляет. Поскольку другие транзакции не могут читать, обновлять или удалять эти строки, следовательно, они избегают неповторяемого чтения.

Serializable - это самый высокий уровень изоляции. Этот уровень изоляции обеспечивает беспрепятственный доступ к базе данных транзакциям с SELECT запросами. Но для транзакций с

запросами UPDATE и DELETE, уровень изоляции Serializable не допускает модификации одной и той же строки в рамках разных транзакций. При изоляции такого уровня все транзакции обрабатываются так, как будто они все запущены последовательно (одна за другой). Две одновременные транзакции не могут обновить одну и туже строку.

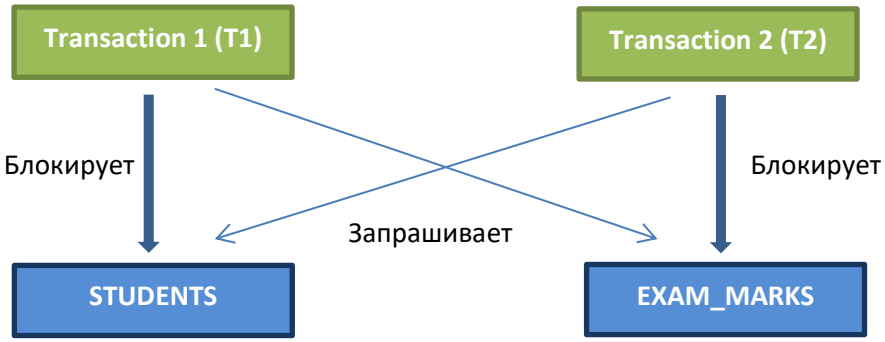
Deadlock (смертельные объятия) в СУБД

Deadlock - это нежелательная ситуация в работе СУБД, при которой две или более транзакции бесконечно ждут, пока одна из них не снимет блокировки и не освободит ресурс, ожидаемый для использования другими транзакциями. Считается, что Deadlock является одной из самых сложных и опасных ситуаций в работе СУБД, поскольку он останавливает всю систему.

Разберемся с концепцией Deadlock на примере:

Предположим, что транзакция T1 блокирует некоторые строки в таблице учеников и должна обновить некоторые строки в таблице оценок. Одновременно транзакция T2 удерживает блокировки в тех самых строках (которые нужно обновить T1) в таблице оценок, но должна обновлять строки в таблице ученика, содержащейся в транзакции T1.

Теперь возникает главная проблема. Транзакция T1 будет ожидать транзакции T2, чтобы снять блокировку, и аналогично транзакция T2 будет ожидать транзакции T1, чтобы снять блокировку. Как следствие, вся деятельность останавливается и остается в тупике навсегда, если СУБД не обнаружит тупик и не прекратит одну из транзакций.



Реализация транзакций в SQL серверах

SQL работает в следующих режимах транзакций.

Explicit transaction. (Явные транзакции):
Каждая транзакция явно начинается с инструкции BEGIN TRANSACTION (START TRANSACTION для MySQL) и явно заканчивается инструкцией COMMIT или ROLLBACK.

Implicit transaction. (Неявные транзакции):
Новая транзакция неявно начинается, когда предыдущая транзакция завершена.

Оператор BEGIN TRANSACTION

Начинает явную транзакцию. Инструкция BEGIN TRANSACTION предоставляет точку, где гарантируется логическая и физическая согласованность данных, на которые ссылается соединение. Если возникли ошибки, все изменений после BEGIN TRANSACTION можно откатить для возврата к известному состоянию согласованности данных. Каждая транзакция продолжается до тех пор, пока она не завершается без ошибок и выполняется инструкция COMMIT TRANSACTION, чтобы внести изменения

в базу данных. В случае возникновения ошибок все изменения удаляются с помощью инструкции ROLLBACK TRANSACTION.

Оператор COMMIT TRANSACTION

Окончательно записывает изменения в базу данных.

Оператор ROLLBACK TRANSACTION

Откатывает состояние базы данных до состояния, в котором она находилась непосредственно перед началом выполнения оператора BEGIN TRANSACTION.

Особенности механизма транзакций в MySQL

1. В MySQL существует несколько типов таблиц. Это ISAM, HEAP, MyISAM, InnoDB, BDB и т.д. Транзакционный механизм поддерживают только InnoDB и BDB. Поэтому все таблицы, с которыми вы хотите работать через транзакции, следует переконвертировать в соответствующий тип.
2. По умолчанию MySQL работает в режиме autocommit. Это означает, что результаты выполнения любого SQL-оператора, изменяющего данные, будут сразу сохраняться.

Режим autocommit можно отключить командой SET AUTOCOMMIT=0. При отключенном режиме autocommit каждую транзакцию надо явно завершать операторами COMMIT / ROLLBACK.

Модификация данных в SQL

Данные в таблицах модифицируются при помощи трех команд:

- insert
- update
- delete

Оператор INSERT INTO (вставка, добавление данных)

Оператор INSERT осуществляет вставку строки в таблицу. Базовая форма оператора имеет вид

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

В списке колонок могут содержаться как все колонки, так и некоторое их подмножество. Для всех колонок, для которых определено свойство NOT NULL, в списке значений должны присутствовать не NULL значения.

Пример:

```
INSERT INTO LECTURERS
           (ID
           ,SURNAME
           ,NAME
           ,CITY
           ,UNIV_ID)
     VALUES
           (26
           ,'Закусило'
           ,'Олег'
           ,'Киев'
           ,2);
```

Порядок следования полей и порядок следования значений должны совпадать. Если указываются значения для всех и порядок значений совпадает с порядком следования полей в таблице, то список полей можно опустить.

```
INSERT INTO LECTURERS
     VALUES
           (26
           ,'Закусило'
           ,'Олег'
           ,'Киев'
           ,2);
```

Для поля, которое допускает значение NULL, можно указать в списке значений константу NULL

```
INSERT INTO LECTURERS
     VALUES
           (27
           ,'Шевченко'
           ,NULL
           ,'Киев'
           ,2);
```

Одним оператором INSERT можно вставить сразу несколько записей:

```
    INSERT INTO UNIVERSITIES (ID, NAME, RATING, CITY)
    VALUES  (1,  N'КПИ',   1257, N'Киев'),
            (2,  N'КНУ',    608, N'Киев'),
            (3,  N'ЛПУ',    593, N'Львов'),
            (4,  N'КМА',    588, N'Киев'),
            (5,  N'ЛГУ',    556, N'Львов'),
            (6,  N'ХАИ',    534, N'Харьков'),
            (7,  N'ДПИ',    529, N'Днепропетровск'),
            (8,  N'ДНТУ',   501, N'Донецк'),
            (9,  N'ХНАДУ',  500, N'Харьков'),
            (10, N'ОНПУ',   496, N'Одесса'),
            (11, N'КНУСА',  483, N'Киев'),
            (12, N'ТНТУ',   441, N'Тернополь'),
            (13, N'ЗДИА',   427, N'Запорожье'),
            (14, N'БНАУ',   399, N'Белая Церковь'),
            (15, N'ХСХА',    45, N'Херсон');
```

Оператором INSERT можно перенести данные из одной таблицы в другую.

Создаем таблицу такой же структуры как и LECTURERS.

Для MS SQL

```
CREATE TABLE LECTURERS_NEW(
       ID int NOT NULL PRIMARY KEY,
       SURNAME nvarchar(50) NOT NULL,
       NAME nvarchar(30) NULL,
       CITY nvarchar(50) NULL,
       UNIV_ID int NULL
)

GO
```

Для MySQl

```
CREATE TABLE `lecturers_new` (
```

```
  `id` int(11) NOT NULL,
  `surname` varchar(50) DEFAULT NULL,
  `name` varchar(30) DEFAULT NULL,
  `city` varchar(50) DEFAULT NULL,
  `univ_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

Вставляем данные из таблицы LECTURERS в таблицу LECTURERS_NEW

```
INSERT INTO LECTURERS_NEW
SELECT ID
      ,SURNAME
      ,NAME
      ,CITY
      ,UNIV_ID
  FROM LECTURERS;
```

В операторе SELECT можно использовать выражение WHERE таким образом вставляя лишь некоторые записи из исходной таблицы.

Для MS SQL в операторе SELECT INTO можно указать имя несуществующей таблицы. При этом будет создана новая таблица со структурой ПОДОБНОЙ структуре исходной таблицы.

```
select * into LECTURERS_NEW_1 from LECTURERS_NEW;
```

Оператор DELETE FROM (удаление данных)

Оператор DELETE FROM удаляет (безвозвратно) существующие строки из таблицы. Общая форма оператора имеет вид:

```
DELETE FROM table_name [WHERE condition];
```

Если условие WHERE опущено, то ИЗ ТАБЛИЦЫ УДАЛЯЮТСЯ ВСЕ ЗАПИСИ. Если условие WHERE задано, то удаляются лишь те строки, для которых выполняется *condition*.

```
delete from LECTURERS_NEW_1 where SURNAME like 'Г%';

delete from LECTURERS_NEW_1;
```

Пример более сложного условия WHERE:

```
delete from exam_marks
where student_id = (select id from students where surname = 'Денисенко')
```

Оператор UPDATE (изменение, модификация данных)

Оператор UPDATE предназначен для модификации существующих данных в таблице и имеет общий вид:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
[WHERE condition];
```

Значения *value1, value2,…* могут быть константами или выражениями. Условие WHERE определяет какие строки будут модифицированы. Если условие WHERE опущено МОДИФИКАЦИИ БУДУТ ПРИМЕНЕНЫ КО ВСЕМ СТРОКАМ ТАБЛИЦЫ.

Примеры.

Установить для всех университетов рейтинг 200:

```
update universities set rating = 200;
```

Повысить рейтинг всех университетов в 2 раза (пример использования скалярного выражения):

```
update universities set rating = rating * 2
```

Установить рейтинги университетов города Киев как "неизвестные":

```
update universities set rating = null where city = 'Киев'
```

Обновить информацию о предмете «Математика»

```
update subjects set
       name = 'Высшая математика'
       , hours = 42
       , semester = 1
where name = 'Математика'
```

Обновление на основе информации из других таблиц

UPDATE FROM

Пример. Для совпадающих предметов поставить студенту Козьменко такие же оценки как у студентки Шуст.

```
-- показываем оценки для студентов 'Козьменко' и 'Шуст'
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Козьменко');
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Шуст');

-- используем транзакцию для того, что бы можно было отменить
-- изменения, вннесенные в базу данных, при модификации
begin transaction;

-- выполняем изменение данных
update t1 set
    mark = t2.mark
FROM
    EXAM_MARKS as t1 inner join EXAM_MARKS as t2 on
t1.subj_id = t2.subj_id
                    and t2.student_id = (select id from students where surname = 'Шуст')
WHERE
    t1.student_id = (select id from students where surname = 'Козьменко');

-- смотрим результат
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Козьменко');
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Шуст');

-- откатываем изменения
rollback;
```

Альтернативный вариант

```
-- показываем оценки для студентов 'Козьменко' и 'Шуст'
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Козьменко');
```

```
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Шуст');

-- используем транзакцию для того, что бы можно было отменить
-- изменения, внесенные в базу данных, при модификации
begin transaction;

-- выполняем изменение данных
update EXAM_MARKS set
       mark = ( select mark from EXAM_MARKS as exm
                     where exm.subj_id = EXAM_MARKS.subj_id and exm.student_id in (
                                   select id from students where surname = 'Шуст'
                     )
              )
where
    student_id = (select id from students where surname = 'Козьменко');

select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Козьменко');
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Шуст');

-- откатываем изменения
rollback;
```

ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
9	4	2	NULL	2012-06-18 00:00:00.000
10	4	4	NULL	2012-06-08 00:00:00.000
11	4	5	3.000	2012-06-07 00:00:00.000

Ups!!! Информация о некоторых оценках обнулилась. Это происходит для оценок по различным предметам у студента Козьменко и Шуст.

```
-- копируем данные об оценках во временную таблицу
select * into #ex_m from exam_marks;

-- показываем оценки для студентов 'Козьменко' и 'Шуст'
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Козьменко');
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Шуст');

-- используем транзакцию для того, что бы можно было отменить
-- изменения, внесенные в базу данных, при модификации
begin transaction;

-- выполняем изменение данных
update EXAM_MARKS set
       mark = isnull((select mark from EXAM_MARKS as exm
                     where exm.subj_id = EXAM_MARKS.subj_id and student_id in (
                                   select id from STUDENTS where surname = 'Шуст'
                     )
              ), mark)
where
    student_id = (select id from students where surname = 'Козьменко');

select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Козьменко');
select * from EXAM_MARKS where student_id =
       (select id from STUDENTS where surname = 'Шуст');

-- откатываем изменения
rollback;
```

Для MySQL

```
select * from exam_marks where student_id =
       (select id from students where surname = 'Козьменко');
select * from exam_marks where student_id =
       (select id from students where surname = 'Шуст');

start transaction;

update exam_marks t1 inner join exam_marks t2 on
              t1.subj_id = t2.subj_id
        and t2.student_id = (select id from students where surname = 'Шуст')
set
       t1.mark = t2.mark
where t1.student_id = (select id from students where surname = 'Козьменко');

select * from exam_marks where student_id =
       (select id from students where surname = 'Козьменко');
select * from exam_marks where student_id =
       (select id from students where surname = 'Шуст');

rollback;
```