Транзакции

Транзакция – это группа последовательных операций, образующих логическую единицу работы с данными.
Транзакция обладает четырьмя важными свойствами, известными как свойства АСИД:

- Атомарность – транзакция выполняется как атомарная операция: либо выполняется вся транзакция целиком, либо она отклоняется.
- Согласованность – транзакция не нарушает логику и отношения между элементами данных. Внутри транзакции согласованность может нарушаться.
- Изоляция – транзакции разных пользователей не должны мешать друг другу (например, как если бы они выполнялись строго по очереди).
- Долговечность – если транзакция выполнена, то результаты ее работы должны сохраниться в системе, даже если в следующий момент произойдет сбой системы.

Различают два исхода транзакции: подтверждение (фиксацию) и откат. Откат происходит автоматически при возникновении ошибки в операции транзакции или при нарушении свойств согласованности и изоляции транзакции, либо может быть запрошен программой, выполняющей транзакцию.
Рассмотрим несколько подходов к классификации транзакций.

1. По уровню организации данных:
    a. транзакции уровня информационной системы – операции выполняются над объектами информационной системы, моделирующими объекты предметной области;
    b. транзакции уровня источника данных – операции выполняются в терминах некоторой модели данных (сетевой, иерархической, реляционной);
    c. транзакции уровня ячейки памяти – концепция транзакционной оперативной памяти;
2. По количеству участников:
    a. локальные транзакции – все операции выполняются с одним источником данных;
    b. глобальные (распределенные) транзакции – операции транзакции выполняются над несколькими источниками данных;
3. По количеству уровней вложенности:
    a. плоские транзакции;
    b. вложенные транзакции – в рамках родительской транзакции может быть несколько дочерних транзакций, которые в свою очередь также могут делиться на вложенные транзакции; исход родительской транзакции определяет, фиксируются ли изменения, сделанные в дочерних транзакциях;
4. По продолжительности:
    a. атомарные транзакции;
    b. долговременные транзакции – не обладают вкупе свойствами атомарности, согласованности, изоляции и долговременности. В общем случае, долговременная транзакция может быть представлена как последовательность меньших транзакций, каждой из которых (за исключением последней)

сопоставлена компенсация[1]. Если одна из транзакций в последовательности должна быть отменена, то компенсации, ассоциированные с успешно завершившимися транзакциями, выполняются в порядке, обратном порядку завершения транзакций.

Транзакции в реляционных базах данных

В данном случае можно дать более узкое определение транзакции:
Транзакция – это последовательность операторов манипулирования данными, выполняющаяся как единое целое (все или ничего) и переводящая базу данных из одного целостного состояния в другое целостное состояние.
Транзакция обычно начинается автоматически с момента присоединения пользователя к СУБД и продолжается до тех пор, пока не произойдет одно из следующих событий:

- Подана команда зафиксировать транзакцию.
- Подана команда откатить транзакцию.
- Произошло отсоединение пользователя от СУБД.
- Произошел сбой системы.

При отсоединении пользователя от СУБД происходит автоматическая фиксация транзакций.
При сбое системы происходят более сложные процессы. Кратко суть их сводится к тому, что при последующем запуске системы происходит анализ выполнявшихся до момента сбоя транзакций. Те транзакции, для которых была подана команда фиксации, но результаты работы которых не были занесены в базу данных, выполняются снова (накатываются). Те транзакции, для которых команда фиксации подана не была, откатываются.
Для начала транзакции и ее завершения предусмотрены соответствующие операторы языка SQL. Их синтаксис и возможности могут различаться в разных СУБД. Существуют СУБД, которые обеспечивают оператор для явного начала новой транзакции, тогда как другие предполагают автоматическое начало транзакции при завершении предыдущей. При интерактивной и программной работе с СУБД часто по умолчанию используется режим AUTO COMMIT, когда каждый запрос выполняется в отдельной транзакции, автоматически завершающейся подтверждением (если в ходе выполнения запроса произошла ошибка, то транзакция откатывается). По окончании транзакции автоматически начинается новая.
В СУБД MySQL при использовании механизма InnoDB можно использовать следующие команды SQL:

- для явного начала новой транзакции (`START TRANSACTION`)
- для подтверждения транзакции (`COMMIT`)
- для отката транзакции (`ROLLBACK`)
- для установки уровня изоляции транзакции (`SET TRANSACTION ISOLATION LEVEL`)

По умолчанию СУБД MySQL работает в режиме AUTO COMMIT. Для его отключения используется команда `SET AUTOCOMMIT = 0`.
В СУБД MS SQL Server при явном начале транзакции можно указать ее имя с помощью оператора: `BEGIN TRANSACTION` *имя_транзакции*.

---

[1] Компенсация представляет собой акт выполнения поправок в случае возникновения ошибок или изменения планов. Хотя компенсация может быть и простой отменой произведенного действия (например, обратный перевод средств на счет, с которого они были ранее сняты), в общем она представляет собой более сложное действие. Например, в случае, если счет был отправлен покупателю, это действие невозможно отменить. В этом случае компенсация может заключаться или в отправке покупателю письма с просьбой не принимать счет во внимание, или в предоставлении оплаты за возврат счета.

Можно использовать этот оператор несколько раз, не заканчивая предыдущую транзакцию, так как MS SQL Server поддерживает вложенность транзакций и может откатывать их независимо от объемлющих транзакций.
Формат оператора отката для этого случая: `ROLLBACK TRANSACTION` *`имя_транзакции`*.
При подтверждении транзакции используется оператор: `COMMIT TRANSACTION` *`имя_транзакции`*.
При этом операторов `COMMIT` должно быть столько же, сколько операторов `BEGIN TRANSACTION`, а параметр `имя_транзакции` игнорируется, поскольку подтверждается самая внешняя транзакция.
По умолчанию в MS SQL Server работа ведется в режиме AUTO COMMIT. Для включения режима явного подтверждения используется оператор `SET IMPLICIT_TRANSACTIONS ON`.

Ограничения целостности

Свойство согласованности транзакций определяется наличием понятия согласованности базы данных.
Ограничение целостности - это некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния базы данных.
Любое ограничение целостности является семантическим понятием, т.е. появляется как следствие определенных свойств объектов предметной области и/или их взаимосвязей.
База данных находится в согласованном (целостном) состоянии, если выполнены (удовлетворены) все ограничения целостности, определенные для базы данных.
Таким образом, согласованность есть формальное свойство базы данных. База данных не понимает "смысла" хранимых данных. "Смыслом" данных для СУБД является весь набор ограничений целостности. Если все ограничения выполнены, то СУБД считает, что данные корректны.
Ограничения целостности можно классифицировать несколькими способами:

- По способам реализации.
- По времени проверки.
- По области действия.

Каждая система обладает своими средствами поддержки ограничений целостности. Различают два способа реализации:

- Декларативная поддержка ограничений целостности.
- Процедурная поддержка ограничений целостности.

Декларативная поддержка ограничений целостности заключается в определении ограничений средствами языка определения данных (DDL - Data Definition Language). Средства декларативной поддержки ограничений описаны в стандарте SQL и более подробно рассматриваются ниже.
Процедурная поддержка ограничений целостности заключается в использовании триггеров и хранимых процедур.
Если система не поддерживает ни декларативную поддержку ссылочной целостности, ни триггеры, то программный код, следящий за корректностью базы данных, приходится размещать в пользовательском приложении. Это сильно затрудняет разработку программ и не защищает от попыток пользователей напрямую внести некорректные данные в базу данных.
По времени проверки ограничения делятся на:

- Немедленно проверяемые ограничения.
- Ограничения с отложенной проверкой.

Немедленно проверяемые ограничения проверяются непосредственно в момент выполнения операции, могущей нарушить ограничение. Например, проверка уникальности потенциального ключа проверяется в момент вставки записи в таблицу. Если ограничение нарушается, то такая операция отвергается. Транзакция, внутри которой произошло нарушение немедленно проверяемого утверждения целостности, обычно откатывается.
Ограничения с отложенной проверкой проверяется в момент фиксации транзакции. Внутри транзакции ограничение может не выполняться. Если в момент фиксации транзакции обнаруживается нарушение ограничения с отложенной проверкой, то транзакция откатывается.
Классификация ограничений по области действия приведена в табл. 1.

Порядок проверки	Область действия	Пример	Немедленно проверяемое	С отложенной проверкой
–[1]	Домен	Значение домена "Возраст сотрудника" может быть не менее 18 и не более 65.	–	–
1	Атрибут	Ограничения атрибута в точности совпадают с ограничениями соответствующего домена.	+	–
2	Кортеж	Ограничение для отношения "Сотрудники": если атрибут "Должность" принимает значение "Директор", то атрибут "Зарплата" содержит значение не менее 1000$.	+	–
3	Отношение (таблица)	Первичный (или потенциальный) ключ отношения.	+	+
4	База данных	Внешний ключ отношения.	+	+

Примечания:
1. Ограничения домена сами по себе не проверяются. Если на каком-либо домене основан атрибут, то ограничение соответствующего домена становится ограничением этого атрибута.

Пример ограничений целостности

Рассмотрим пример ограничений целостности из предметной области «Электронный магазин». Допустим, что информация о заказе хранится в виде следующих таблиц:
Order

id	customer	Date	sum

LineItem

id	order	Product	price	quantity

При такой структуре хранения предполагается следующее:

1. поля `id` в каждой из таблиц принимают уникальные значения (первичные ключи)
2. поля `Order.customer`, `LineItem.order`, `LineItem.product` принимают значения, соответствующие значениям полей `Customer.id`, `Order.id`, `Product.id` соответственно (внешние ключи). Таблицы `Customer` и `Product` на рисунке не представлены.
3. в таблице `Order` не существует такой записи, с которой не связано не одной записи из таблицы `LineItem` (заказ должен содержать хотя бы 1 элемент).
4. сумма произведений значений `LineItem.price` на `LineItem.quantity` записей из таблицы `LineItem`, относящихся к одной и той же записи из таблицы `Order` должна быть равна значению `Order.sum` этой записи (сумма заказа складывается из суммы его частей).

Контроль выполнения первых двух правил возлагается на СУБД за счет определения первичных и внешних ключей в структуре БД.

Выполнение последующих двух правил, как правило, возлагается на приложение, при этом операции «добавление нового заказа» и «изменение заказа» (когда это изменение затрагивает элементы заказа) с точки зрения прикладной области выглядят как атомарные операции, а со стороны приложения их реализация требует выполнения нескольких операторов SQL. Так, для добавления заказа нужно выполнить один оператор `INSERT` для таблицы `Order`, а для таблицы `LineItem` – по одному для каждого элемента заказа.

При этом важно следующее:

1. нужно, чтобы либо все операторы выполнились успешно, либо ни один из них, даже после того как были выполнены несколько операторов, а потом произошел сбой
2. в процессе выполнения операций условия согласованности (3 и 4) нарушаются, однако по их завершении они соблюдаются.

В данном примере выполнение бизнес - операций с заказом должно выполняться в рамках транзакций, иначе соблюдение вышеуказанных требований будет невозможно.

Ограничение 4 в данном примере, вообще говоря, представляет собой правило вычисления атрибута `Order.sum`. В отношении вычисляемых атрибутов есть два диаметрально противоположных подхода, достоинства и недостатки которых приведены в табл. 2.

Достоинства	Недостатки
1. В отношении хранятся только базовые атрибуты.	
1. Структура отношения полностью неизбыточна. 2. Не требуется дополнительного программного кода для поддержания целостности кортежа. 3. Экономится дисковое пространство. 4. Уменьшается трафик сети.	1. Имеется риск в разных местах вычислять одни и те же данные по разным формулам. 2. Затруднена модификация приложений. 3. Если возникает нерегламентированный запрос, то человек, формирующий запрос, должен знать формулы.
2. В отношении хранятся все атрибуты, в том числе и вычислимые.	
1. Код, поддерживающий целостность кортежа (и содержащий формулы для вычислимых атрибутов), хранится в одном месте, например в триггере. 2. При изменении логики вычислений, изменения в формулы требуется внести только в одном месте (в триггере). 3. Запросы к базе данных содержат меньше формул и поэтому более просты.	1. При изменении логики расчета надобность в некоторых атрибутах может исчезнуть, зато может появиться потребность в новых атрибутах. Это потребует перестройки структуры отношения, что является весьма болезненной операцией для работающей системы. 2. Структура отношения становится более сложной и запутанной.

4. Легче формулировать нерегламентированные запросы.	3. Увеличивается объем базы данных. 4. Увеличивается трафик сети.
3. В отношении хранятся только базовые атрибуты, а для вычислений используются представления (динамические отношения, задаваемые оператором SQL).	
…	…

Реализация декларативных ограничений целостности средствами SQL

Стандарт SQL не предусматривает процедурных ограничений целостности, реализуемых при помощи триггеров и хранимых процедур.
Стандарт SQL позволяет задавать декларативные ограничения следующими способами:

- ограничение домена.
- ограничение, входящее в определение таблицы.
- ограничение, хранящееся в базе данных в виде независимого утверждения.

Допускаются как немедленно проверяемые, так и ограничения с отложенной проверкой. Режим проверки отложенных ограничений можно в любой момент изменить так, чтобы ограничение проверялось:

1. После исполнения каждого оператора, изменяющего содержимое таблицы, к которой относится данное ограничение.
2. При завершении каждой транзакции, включающей операторы, изменяющие содержимое таблиц, к которым относятся данное ограничение.
3. В любой промежуточный момент, если пользователь инициирует проверку.

Понятие ограничения используется во многих операторах определения данных (DDL). Синтаксис ограничений стандарта SQL:
Ограничение check::=
CHECK *Предикат*
Ограничения таблицы ::=
[**CONSTRAINT** *Имя ограничения*]
{
{**PRIMARY KEY** (*Имя столбца*.,..)}
| {**UNIQUE** (*Имя столбца*.,..)}
| {**FOREIGN KEY** (*Имя столбца*.,..) **REFERENCES** *Имя таблицы* [(*Имя столбца*.,..)]
[*Ссылочная спецификация*]}
| { *Ограничение check* }
}
[*Атрибуты ограничения*]
Ограничения столбца::=
[**CONSTRAINT** *Имя ограничения*]
{
{**NOT NULL**}
| {**PRIMARY KEY**}
| {**UNIQUE**}
| {**REFERENCES** *Имя таблицы* [(*Имя столбца*)] [*Ссылочная спецификация*]}
| { *Ограничение check* }
}
[*Атрибуты ограничения*]

Ссылочная спецификация::=
[**MATCH** {**FULL** | **PARTIAL**}]
[**ON UPDATE** {**CASCADE** | **SET NULL** | **SET DEFAULT** | **NO ACTION**}]
[**ON DELETE** {**CASCADE** | **SET NULL** | **SET DEFAULT** | **NO ACTION**}]
Атрибуты ограничения::=
{**DEFERRABLE** [**INITIALLY DEFERRED** | **INITIALLY IMMEDIATE**]}
| {**NOT DEFERRABLE**}
Ограничение типа CHECK содержит предикат, могущий принимать значения TRUE, FALSE и NULL. Ограничение типа CHECK может быть использовано как часть описания домена, таблицы, столбца таблицы или независимого утверждения. Ограничение считается нарушенным, если предикат ограничения принимает значение FALSE.
Пример ограничения типа CHECK, которое утверждает, что каждый продавец должен иметь либо ненулевую зарплату, либо ненулевые комиссионные (ограничение кортежа):
```
CHECK (Seller.Salary IS NOT NULL) OR (Seller.Commission IS NOT NULL)
```
Ограничения таблицы и ограничения столбца таблицы входят как часть описания соответственно таблицы или столбца таблицы. Ограничение таблицы может относиться к нескольким столбцам таблицы. Ограничение столбца относится только к одному столбцу таблицы. Любое ограничение столбца можно описать как ограничение таблицы, но не наоборот.
Ограничения таблицы или столбца могут иметь наименования, при помощи которого в дальнейшем можно отменять это ограничение или менять время его проверки.
Ограничение PRIMARY KEY для таблицы или столбца означает, что группа из одного или нескольких столбцов образуют первичный ключ таблицы. Для одной таблицы может быть определено единственное ограничение PRIMARY KEY.
Ограничение UNIQUE для таблицы или столбца означает, что группа из одного или нескольких столбцов образуют потенциальный ключ таблицы, в котором допускаются значения NULL. Две строки, содержащие NULL-значения считаются различными и допускаются. Для одной таблицы может быть определено несколько ограничений UNIQUE.
Ограничение FOREIGN KEY… REFERENCES… для таблицы и ограничение REFERENCES… для столбца определяют внешний ключ таблицы. Ограничение REFERENCES… для столбца определяет простой внешний ключ, т.е. ключ, состоящий из одной колонки. Ограничение FOREIGN KEY… REFERENCES… для таблицы может определять как простой, так и сложный внешний ключ, т.е. ключ, состоящий из нескольких колонок таблицы. Столбец или группа столбцов таблицы, на которую ссылается внешний ключ, должна иметь ограничения PRIMARY KEY или UNIQUE. Столбцы, на которые ссылается внешний ключ, должны иметь тот же тип данных, что и столбцы, входящие в состав внешнего ключа. Таблица может иметь ссылку на себя. Ограничение внешнего ключа нарушается, если значения, присутствующие во внешнем ключе, не совпадают со значениями соответствующего ключа родительской таблицы ни для одной строки из родительской таблицы. Операции, приводящие к нарушению ограничения внешнего ключа, отвергаются.
Ограничение NOT NULL столбца не допускает появления в столбце NULL-значений.

Транзакции и параллелизм

Современные СУБД являются многопользовательскими системами, т.е. допускают параллельную одновременную работу большого количества пользователей. При этом пользователи не должны мешать друг другу. Т.к. логической единицей работы для пользователя является транзакция, то работа СУБД должна быть организована так, чтобы у пользователя складывалось впечатление, что его транзакции выполняются независимо от транзакций других пользователей.

Простейший и очевидный способ обеспечить такую иллюзию у пользователя состоит в том, чтобы все поступающие транзакции выстраивать в единую очередь и выполнять строго по очереди. Такой способ не годится по очевидным причинам – теряется преимущество параллельной работы. Таким образом, транзакции необходимо выполнять одновременно, но так, чтобы результат был бы такой же, как если бы транзакции выполнялись по очереди.
Транзакция рассматривается как последовательность элементарных атомарных операций. Элементарные операции различных транзакций могут выполняться в произвольной очередности (конечно, внутри каждой транзакции последовательность элементарных операций этой транзакции является строго определенной).
Набор из нескольких транзакций, элементарные операции которых чередуются друг с другом, называется смесью транзакций. Последовательность, в которой выполняются элементарные операции заданного набора транзакций, называется графиком запуска набора транзакций. Очевидно, что для заданного набора транзакций может быть несколько различных графиков запуска.

Проблемы параллельной работы транзакций

Различают пять основных проблем параллелизма:

- Потеря результатов обновления.
- Незафиксированная зависимость (чтение "грязных" данных, неаккуратное считывание).
- Неповторяемое считывание.
- Фиктивные элементы (фантомы).
- Несовместимый анализ.

Рассмотрим две транзакции, A и B, запускающиеся в соответствии с некоторыми графиками. Пусть транзакции работают с некоторыми объектами базы данных, например со строками таблицы. Операцию чтение строки P будем обозначать $P = P_0$, где P_0 - прочитанное значение. Операцию записи значения P_1 в строку P будем обозначать $P_1 \rightarrow P$.

1. Проблема потери результатов обновления.
Две транзакции по очереди записывают некоторые данные в одну и ту же строку и фиксируют изменения.

Транзакция A	**Время**	**Транзакция B**
Чтение $P = P_0$	t_1	---
---	t_2	Чтение $P = P_0$
Запись $P_1 \rightarrow P$	t_3	---
---	t_4	Запись $P_2 \rightarrow P$
Фиксация транзакции	t_5	---
---	t_6	Фиксация транзакции
Потеря результата обновления		

После окончания обеих транзакций, строка P содержит значение P_2, занесенное более поздней транзакцией B. Транзакция A ничего не знает о существовании транзакции B, и естественно ожидает, что в строке P содержится значение P_1. Таким образом, транзакция A потеряла результаты своей работы.

2. Проблема незафиксированной зависимости (чтение "грязных" данных, неаккуратное считывание).
Транзакция B изменяет данные в строке. После этого транзакция A читает измененные данные и работает с ними. Транзакция B откатывается и восстанавливает старые данные.

Транзакция A	**Время**	**Транзакция B**
---	t_1	Чтение $P = P_0$
---	t_2	Запись $P_1 \rightarrow P$
Чтение $P = P_1$	t_3	---
Работа с прочитанными данными P_1	t_4	---
---	t_5	Откат транзакции $P_0 \rightarrow P$
Фиксация транзакции	t_6	---
Работа с "грязными" данными		

Транзакция A в своей работе использовала данные, которых нет и не было в базе данных. Действительно, после отката транзакции B должна восстановиться ситуация, как если бы транзакция B никогда не выполнялась. Таким образом, результаты работы транзакции A некорректны.

3. Неповторяемое считывание.
Транзакция A дважды читает одну и ту же строку. Между этими чтениями вклинивается транзакция B, которая изменяет значения в строке.

Транзакция A	**Время**	**Транзакция B**
Чтение $P = P_0$	t_1	---
---	t_2	Чтение $P = P_0$
---	t_3	Запись $P_1 \rightarrow P$
---	t_4	Фиксация транзакции
Повторное чтение $P = P_1$	t_5	---
Фиксация транзакции	t_6	---
Неповторяемое считывание		

Транзакция A ничего не знает о существовании транзакции B, и, т.к. сама она не меняет значение в строке, то ожидает, что после повторного чтения значение будет тем же самым. Таким образом, транзакция A работает с данными, которые, с точки зрения транзакции A, самопроизвольно изменяются.

4. Фиктивные элементы (фантомы)
Транзакция A дважды выполняет выборку строк с одним и тем же условием. Между выборками вклинивается транзакция B, которая добавляет новую строку, удовлетворяющую условию отбора.

Транзакция A	**Время**	**Транзакция B**
Выборка строк, удовлетворяющих условию α. (Отобрано n строк)	t_1	---

---	t_2	Вставка новой строки, удовлетворяющей условию α.
---	t_3	Фиксация транзакции
Выборка строк, удовлетворяющих условию α. (Отобрано n+1 строк)	t_4	---
Фиксация транзакции	t_5	---
Появились строки, которых раньше не было		

Транзакция А ничего не знает о существовании транзакции В, и, т.к. сама она не меняет ничего в базе данных, то ожидает, что после повторного отбора будут отобраны те же самые строки. Таким образом, транзакция А в двух одинаковых выборках строк получила разные результаты.

5. Несовместимый анализ.

В смеси присутствуют две транзакции - одна длинная, другая короткая. Длинная транзакция выполняет некоторый анализ по всей таблице, например, подсчитывает общую сумму денег на счетах клиентов банка для главного бухгалтера. Пусть на всех счетах находятся одинаковые суммы, например, по $100. Короткая транзакция в этот момент выполняет перевод $50 с одного счета на другой так, что общая сумма по всем счетам не меняется.

Транзакция А	**Время**	**Транзакция В**
Чтение счета P_1 = 100 и суммирование. $SUM = 100$	t_1	---
---	t_2	Снятие денег со счета P_3. P_3 : 100 → 50
---	t_3	Помещение денег на счет P_1. P_1 : 100 → 150
---	t_4	Фиксация транзакции
Чтение счета P_2 = 100 и суммирование. $SUM = 200$	t_5	---
Чтение счета P_3 = 50 и суммирование. $SUM = 250$	t_6	---
Фиксация транзакции	t_7	---
Сумма $250 по всем счетам неправильная - должно быть $300		

Хотя транзакция В все сделала правильно - деньги переведены без потери, но в результате транзакция А подсчитала неверную общую сумму. Т.к. транзакции по переводу денег идут обычно непрерывно, то в данной ситуации следует ожидать, что главный бухгалтер никогда не узнает, сколько же денег в банке.

Анализ проблем параллелизма показывает, что если не предпринимать специальных мер, то при работе в смеси нарушается свойство (И) транзакций - изолированность.

Транзакции называются конкурирующими, если они пересекаются по времени и обращаются к одним и тем же данным.
В результате конкуренции за данными между транзакциями возникают конфликты доступа к данным. Различают следующие виды конфликтов:

- Запись - Запись. Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается изменить этот объект. Результат - потеря обновления.
- Чтение - Запись. Первая транзакция прочитала объект и не закончилась. Вторая транзакция пытается изменить этот объект. Результат - несовместимый анализ (неповторяемое считывание).
- Запись - Чтение. Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается прочитать этот объект. Результат - чтение "грязных" данных.

Другие проблемы параллелизма (фантомы и несовместимый анализ) являются более сложными, т.к. для их возникновения требуется, чтобы транзакции работали с целыми наборами данных.
График запуска набора транзакций называется последовательным, если транзакции выполняются строго по очереди, т.е. элементарные операции транзакций не чередуются друг с другом. При выполнении последовательного графика гарантируется, что транзакции выполняются правильно, т.е. при последовательном графике транзакции не "чувствуют" присутствия друг друга.
График запуска набора транзакций называется верным (сериализуемым), если он эквивалентен какому-либо последовательному графику, то есть при их выполнении будет получен один и тот же результат, независимо от начального состояния базы данных.
Задача обеспечения изолированной работы пользователей состоит в нахождении сериализуемого графика запуска транзакций, оптимального с некоторой точки зрения. Простейший способ сериализации - ставить транзакции в общую очередь по мере их поступления и выполнять строго последовательно – является неоптимальным с точки зрения общей производительности системы. В качестве критерия оптимальности можно взять, например, суммарное время выполнения всех транзакций в наборе. Время выполнения одной транзакции считается от момента, когда транзакция возникла и до момента, когда транзакция выполнила свою последнюю элементарную операцию. Однако оптимальный график запуска транзакций достижим, только если заранее известна вся будущая смесь транзакций и моменты поступления каждой транзакции. В реальной ситуации эти данные неизвестны, однако СУБД должна работать так, чтобы к любому моменту времени набор выполненных и выполняющихся в этот момент транзакций был бы правильным и не слишком далек от оптимального.
Т.к. транзакции не мешают друг другу, если они обращаются к разным данным или выполняются в разное время, то имеется два способа разрешить конкуренцию между поступающими в произвольные моменты транзакциями:

1. "Притормаживать" некоторые из поступающих транзакций настолько, насколько это необходимо для обеспечения правильности смеси транзакций в каждый момент времени (т.е. обеспечить, чтобы конкурирующие транзакции выполнялись в разное время).
2. Предоставить конкурирующим транзакциям "разные" экземпляры данных (т.е. обеспечить, чтобы конкурирующие транзакции работали с разными версиями данными).

Первый метод - "притормаживание" транзакций - реализуется путем использованием блокировок различных видов или метода временных меток.

Второй метод - предоставление разных версий данных - реализуется путем использованием данных из журнала транзакций.

Концепция сериализуемости транзакций была впервые предложена Есвараном, который разработал протокол двухфазной блокировки:

1. Перед выполнение каких-либо операций с некоторым объектом, транзакция должна заблокировать этот объект.
2. После снятия блокировки, транзакция не должна накладывать никаких других блокировок.

Собственно концепция формулируется в виде теоремы Есварана:
Если все транзакции в смеси подчиняются протоколу двухфазной блокировки, то для всех чередующихся графиков запуска существует возможность упорядочения.

Блокировки

Основная идея блокировок заключается в том, что если для выполнения некоторой транзакции необходимо, чтобы некоторый объект не изменялся без ведома этой транзакции, то этот объект должен быть заблокирован, т.е. доступ к этому объекту со стороны других транзакций ограничивается на время выполнения транзакции, вызвавшей блокировку.
Различают два типа блокировок:

- Монопольные блокировки (X-блокировки, X-locks - eXclusive locks) - блокировки без взаимного доступа (блокировка записи).
- Разделяемые блокировки (S-блокировки, S-locks - Shared locks) - блокировки с взаимным доступом (блокировка чтения).

Правила взаимного доступа к заблокированным объектам можно представить в виде следующей матрицы совместимости блокировок. Если транзакция A наложила блокировку на некоторый объект, а транзакция B после этого пытается наложить блокировку на этот же объект, то успешность блокирования транзакцией B объекта описывается таблицей:

	Транзакция B пытается наложить блокировку:	
Транзакция A наложила блокировку:	S-блокировку	X-блокировку
S-блокировку	Да	НЕТ (Конфликт R-W)
X-блокировку	НЕТ (Конфликт W-R)	НЕТ (Конфликт W-W)

Три случая, когда транзакция B не может блокировать объект, соответствуют трем видам конфликтов между транзакциями.
Доступ к объектам базы данных на чтение и запись должен осуществляться в соответствии со следующим протоколом доступа к данным:

1. Прежде чем прочитать объект, транзакция должна наложить на этот объект S-блокировку.
2. Прежде чем обновить объект, транзакция должна наложить на этот объект X-блокировку. Если транзакция уже заблокировала объект S-блокировкой (для чтения), то перед обновлением объекта S-блокировка должна быть заменена X-блокировкой.
3. Если блокировка объекта транзакцией B отвергается оттого, что объект уже заблокирован транзакцией A, то транзакция B переходит в состояние ожидания.

Транзакция B будет находиться в состоянии ожидания до тех пор, пока транзакция A не снимет блокировку объекта.

4. X-блокировки, наложенные транзакцией A, сохраняются до конца транзакции A.

Рассмотрим, как будут себя вести транзакции, вступающие в конфликт при доступе к данным, если они подчиняются приведенному протоколу доступа к данным.
Пусть две транзакции по очереди записывают некоторые данные в одну и ту же строку и фиксируют изменения.

Транзакция A	**Время**	**Транзакция B**
S-блокировка *P* - успешна	t_1	---
Чтение $P = P_0$	t_2	---
---	t_3	S-блокировка *P* - успешна
---	t_4	Чтение $P = P_0$
X-блокировка *P* - отвергается	t_5	---
Ожидание…	t_6	X-блокировка *P* - отвергается
Ожидание…	t_7	Ожидание…

Обе транзакции успешно накладывают S-блокировки и читают объект *P*. Транзакция A пытается наложить X-блокирокировку для обновления объекта *P*. Блокировка отвергается, т.к. объект *P* уже S-заблокирован транзакцией B. Транзакция A переходит в состояние ожидания до тех пор, пока транзакция B не освободит объект. Транзакция B, в свою очередь, пытается наложить X-блокирокировку для обновления объекта *P*. Блокировка отвергается, т.к. объект *P* уже S-заблокирован транзакцией A. Транзакция B переходит в состояние ожидания до тех пор, пока транзакция A не освободит объект. Таким образом, обе транзакции ожидают друг друга и не могут продолжаться. Возникла ситуация тупика.
Т.к. нормального выхода из тупиковой ситуации нет, то такую ситуацию необходимо распознавать и устранять. Методом разрешения тупиковой ситуации является откат одной из транзакций (транзакции-жертвы) так, чтобы другие транзакции продолжили свою работу. После разрешения тупика, транзакцию, выбранную в качестве жертвы можно повторить заново. За возникновением тупиковой ситуации СУБД следит путем построения графа ожидания транзакций – ориентированного двудольного графа, в котором существует два типа вершин - вершины, соответствующие транзакциям, и вершины, соответствующие объектам захвата. Ситуация тупика возникает, если в графе ожидания транзакций имеется хотя бы один цикл. Одну из транзакций, попавших в цикл, необходимо откатить, причем, система сама может выбрать эту транзакцию в соответствии с некоторыми стоимостными соображениями (например, самую короткую, или с минимальным приоритетом и т.п.).

Механизм выделения версий данных

Использование блокировок гарантирует сериальность планов выполнения смеси транзакций за счет общего замедления работы - конфликтующие транзакции ожидают, когда транзакция, первой заблокировавшая некоторый объект, не освободит его. Без блокировок не обойтись, если все транзакции *изменяют* данные. Но если в смеси транзакций присутствуют как транзакции, изменяющие данные, так и *только читающие* данные, можно применить альтернативный механизм обеспечения сериальности, свободный от недостатков метода блокировок. Этот метод состоит в том, что транзакциям, читающим данные, предоставляется как бы "своя" версия данных, имевшаяся в момент начала читающей транзакции. При этом транзакция не накладывает

блокировок на читаемые данные, и, поэтому, не блокирует другие транзакции, изменяющие данные. Такой механизм называется механизмом выделения версий и заключается в использовании журнала транзакций для генерации разных версий данных.
Журнал транзакций предназначен для выполнения операции отката при неуспешном выполнении транзакции или для восстановления данных после сбоя системы. Журнал транзакций содержит старые копии данных, измененных транзакциями.
Кратко суть метода состоит в следующем:

- Для каждой транзакции (или запроса) запоминается текущий системный номер (SCN - System Current Number). Чем позже начата транзакция, тем больше ее SCN.
- При записи страниц данных на диск фиксируется SCN транзакции, производящей эту запись. Этот SCN становится текущим системным номером страницы данных.
- Транзакции, только читающие данные, не блокируют ничего в базе данных.
- Если транзакция А читает страницу данных, то SCN транзакции А сравнивается с SCN читаемой страницы данных.
- Если SCN страницы данных меньше или равен SCN транзакции А, то транзакция А читает эту страницу.
- Если SCN страницы данных больше SCN транзакции А, то это означает, что некоторая транзакция В, начавшаяся позже транзакции А, успела изменить или сейчас изменяет данные страницы. В этом случае транзакция А просматривает журнал транзакция назад в поиске первой записи об изменении нужной страницы данных с SCN меньшим, чем SCN транзакции А. Найдя такую запись, транзакция А использует старый вариант данных страницы.

Рассмотрим, как решается проблема несовместного анализа с использованием механизма выделения версий.
Длинная транзакция выполняет некоторый анализ по всей таблице, например, подсчитывает общую сумму денег на счетах клиентов банка для главного бухгалтера. Пусть на всех счетах находятся одинаковые суммы, например, по $100. Короткая транзакция в этот момент выполняет перевод $50 с одного счета на другой так, что общая сумма по всем счетам не меняется.

Транзакция А	**Время**	**Транзакция В**
Проверка SCN счета P_1- SCN транзакции больше SCN счета. Чтение счета $P_1 = 100$ без наложения блокировки и суммирование. $SUM = 100$	t_1	---
---	t_2	X-блокировка счета P_3- успешна
---	t_3	Снятие денег со счета P_3. $P_3 : 100 \rightarrow 50$
---	t_4	X-блокировка счета P_1- успешна
---	t_5	Помещение денег на счет P_1. $P_1 : 100 \rightarrow 150$

---	t_6	Фиксация транзакции (Снятие блокировок)
Проверка SCN счета P_2 - SCN транзакции больше SCN счета. Чтение счета $P_2 = 100$ без наложения блокировка и суммирование. $SUM = 200$	t_7	---
Проверка SCN счета P_3 - SCN транзакции **МЕНЬШЕ** SCN счета. Чтение старого варианта счета $P_3 = 100$ и суммирование. $SUM = 300$	t_8	---
Фиксация транзакции	t_9	---
Сумма на счетах посчитана правильно.		

Транзакция A, начавшаяся первой, не тормозит конкурирующую транзакцию B. При обнаружении конфликта (чтение транзакцией A измененного счета 3), транзакции A предоставляется своя версия данных, которая была на момент начала транзакции A.

Реализация изолированности транзакций средствами SQL

Стандарт SQL не предусматривает понятие блокировок для реализации сериализуемости смеси транзакций. Вместо этого вводится понятие уровней изоляции. Этот подход обеспечивает необходимые требования к изолированности транзакций, оставляя возможность производителям различных СУБД реализовывать эти требования своими способами (в частности, с использованием блокировок или выделением версий данных).
Стандарт SQL предусматривает 4 уровня изоляции:

- READ UNCOMMITTED - уровень незавершенного считывания.
- READ COMMITTED - уровень завершенного считывания.
- REPEATABLE READ - уровень повторяемого считывания.
- SERIALIZABLE - уровень способности к упорядочению.

Если все транзакции выполняются на уровне способности к упорядочению (принятом по умолчанию), то чередующееся выполнение любого множества параллельных транзакций может быть упорядочено. Если некоторые транзакции выполняются на более низких уровнях, то выделяется три случая нарушения способности к упорядочению:

- Неаккуратное считывание ("Грязное" чтение, незафиксированная зависимость).
- Неповторяемое считывание (Частный случай несовместного анализа).
- Фантомы (Фиктивные элементы - частный случай несовместного анализа).

Потеря результатов обновления стандартом SQL не допускается, т.е. на самом низком уровне изолированности транзакции должны работать так, чтобы не допустить потери результатов обновления.
Различные уровни изоляции определяются по возможности возникновения особых случаев нарушения способности к упорядочению, что описывается следующей таблицей:

Уровень изоляции	Неаккуратное	Неповторяемое	Фантомы

	считывание	считывание	
READ UNCOMMITTED	Да	Да	Да
READ COMMITTED	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Да
SERIALIZABLE	Нет	Нет	Нет

Уровень изоляции транзакции в SQL задается следующим оператором:

```
SET TRANSACTION {ISOLATION LEVEL
{READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SERIALIZABLE}
| {READ ONLY | READ WRITE}}.,..
```

Этот оператор определяет режим выполнения следующей транзакции, т.е. этот оператор не влияет на изменение режима той транзакции, в которой он подается.
Если задано предложение ISOLATION LEVEL, то за ним должно следовать один из параметров, определяющих уровень изоляции.
Кроме того, можно задать признаки READ ONLY или READ WRITE. Если указан признак READ ONLY, то предполагается, что транзакция будет только читать данные. При попытке записи для такой транзакции будет сгенерирована ошибка. Признак READ ONLY введен для того, чтобы дать производителям СУБД возможность уменьшать количество блокировок путем использования других методов сериализации (например, метод выделения версий).
Оператор SET TRANSACTION должен удовлетворять следующим условиям:

- Если предложение ISOLATION LEVEL отсутствует, то по умолчанию принимается уровень SERIALIZABLE.
- Если задан признак READ WRITE, то параметр ISOLATION LEVEL не может принимать значение READ UNCOMMITTED.
- Если параметр ISOLATION LEVEL определен как READ UNCOMMITTED, то транзакция становится по умолчанию READ ONLY. В противном случае по умолчанию транзакция считается как READ WRITE.

Уровень изоляции транзакции одновременно влияет на скорость выполнения с одной стороны и на целостность данных с другой стороны. Скорость выполнения может быть повышена за счет повышения риска нарушения целостности и наоборот.
Не все проблемы параллелизма актуальны для конкретного приложения. И поскольку для решения всех этих проблем необходимы значительные ресурсы со стороны СУБД, возникла необходимость управлять уровнем изоляции транзакции и, соответственно, ограничить спектр тех проблем, решение которых должна обеспечить СУБД при множественном доступе к данным. Именно с этой целью было введено понятие уровня изоляции транзакции.
Поскольку большинству транзакций не требуется делать выборки одних и тех же данных более одного раза, то наиболее разумно использовать уровень изоляции READ COMMITTED, который принят по умолчанию во многих СУБД.

Поддержка транзакций в JDBC

При использовании интерфейса JDBC для доступа к БД вся работа с транзакциями ведется через интерфейс `Connection`.

Транзакция начинается автоматически после окончания предыдущей, поэтому метод для явного начала транзакции отсутствует.
Для подтверждения или отката транзакции используются методы `commit()` и `rollback()`, соответственно.
Управление уровнем изоляции выполняется методом `setTransactionIsolation()`. В качестве параметра методу передается значение одной из 4-х констант, определенных в интерфейсе `Connection`: `TRANSACTION_READ_COMMITTED`, `TRANSACTION_READ_UNCOMMITTED`, `TRANSACTION_REPEATABLE_READ` или `TRANSACTION_SERIALIZABLE`.
Включение и выключение режима AUTO COMMIT выполняется методом `setAutoCommit()`. По умолчанию новые соединения создаются с включенным режимом AUTO COMMIT. Как правило, в URL подключения к БД или в свойствах источника данных можно отключить данный режим, название соответствующего параметра зависит от JDBC-драйвера.