**1. Реляционная модель. Свойства и основные особенности реляционной модели. Информационный принцип наполнения БД. Замкнутость реляционных систем, проявление замкнутости в синтаксисе языка SQL.**

**Реляционная модель** – данные представлены в виде набора таблиц, и имеются некоторые программные элементы, кот-е обеспечивают взаимодействие между данными, находящимися в различных таблицах.

Техническая статья "Реляционная модель данных для больших разделяемых банков данных" доктора **Е.Ф. Кодда, опубликованная в 1970 г.**, является родоначальницей современной теории реляционных БД. Доктор Кодд определил 13 правил реляционной модели (которые называют 12 правилами Кодда ).

**12 правил Кодда**

1. Реляционная СУБД должна быть способна полностью управлять базой данных через ее реляционные возможности.
2. Информационное правило - вся информация в реляционной БД (включая имена таблиц и столбцов) должна определяться строго как значения в таблицах.
3. Гарантированный доступ - любое значение в реляционной БД должно быть гарантированно доступно для использования через комбинацию имени таблицы, значения первичного ключа и имени столбца
4. Поддержка пустых значений (null value) - СУБД должна уметь работать с пустыми значениями (неизвестными или неиспользованными значениями), в отличие от значений по умолчанию и независимо для любых доменов.
5. Онлайновый реляционный каталог - описание БД и ее содержания должны быть представлены на логическом уровне как таблицы, к которым можно применять запросы, используя язык базы данных.
6. Исчерпывающий язык управления данными - по крайней мере, один из поддерживаемых языков должен иметь четко определенный синтаксис и быть всеобъемлющим. Он должен поддерживать описание структуры данных и манипулирование ими, правила целостности, авторизацию и транзакции.
7. Правило обновления представлений (views) - все представления, теоретически обновляемые, могут быть обновлены через систему.
8. Вставка, обновление и удаление - СУБД поддерживает не только запрос на отбор данных, но и вставку, обновление и удаление
9. Физическая независимость данных - на программы-приложения и специальные программы логически не влияют изменения физических методов доступа к данным и структур хранилищ данных.
10. Логическая независимость данных - на программы-приложения и специальные программы логически не влияют, в пределах разумного, изменения структур таблиц.
11. Независимость целостности - язык БД должен быть способен определять правила целостности. Они должны сохраняться в онлайновом справочнике, и не должно существовать способа их обойти.
12. Независимость распределения - на программы-приложения и специальные программы логически не влияет, первый раз используются данные или повторно.
13. Неподрывность - невозможность обойти правила целостности, определенные через язык базы данных, использованием языков низкого уровня

Для лучшего понимания РМД следует отметить *три важных обстоятельства:*

- **модель является логической**, то есть отношения являются логическими (абстрактными), а не физическими (хранимыми) структурами;

- для реляционных баз данных верен **информационный принцип:** всё информационное наполнение базы данных представлено одним и только одним способом, а именно — явным заданием значений атрибутов в кортежах отношений; в частности, нет никаких указателей (адресов), связывающих одно значение с другим;

- **наличие реляционной алгебры** позволяет реализовать декларативное программирование и декларативное описание ограничений целостности, в дополнение к навигационному (процедурному) программированию и процедурной проверке условий.

Кроме того, в состав реляционной модели данных включают теорию нормализации.

**Реляционная алгебра** представляет собой набор операторов, использующих отношения в качестве аргументов, и возвращающие отношения в качестве результата. Реляционная алгебра замкнута таким образом, что результаты одних реляционных выражений можно использовать в других выражениях. Для выполнения некоторых реляционных операторов требуется, чтобы отношения были совместимы по типу.

**Операторы:**

1. Объединением двух совместимых по типу отношений А и В называется отношение с тем же заголовком, что и у отношений А и В, и телом, состоящим из кортежей, принадлежащих или , А или В , или обоим отношениям. Синтаксис операции объединения: A UNION B.

2. **Пересечением** двух совместимых по типу отношений А и В называется отношение с тем же заголовком, что и у отношений А и В , и телом, состоящим из кортежей, принадлежащих одновременно обоим отношениям А и В. Синтаксис операции пересечения: A INERSECT B.

3. **Вычитанием** двух совместимых по типу отношений А и В называется отношение с тем же заголовком, что и у отношений А и В , и телом, состоящим из кортежей, принадлежащих отношению А и не принадлежащих отношению В. Синтаксис операции вычитания: A MINUS B.

4. **Декартовым произведением** двух отношений и называется отношение, заголовок которого является сцеплением заголовков отношений А и В: , а тело состоит из кортежей, являющихся сцеплением кортежей отношений А и В: , таких, что , .

Синтаксис операции декартового произведения: A TIMED B.

Замечание. Если в отношения A и B имеются атрибуты с одинаковыми наименованиями, то перед выполнением операции декартового произведения такие атрибуты необходимо переименовать.

5. **Выборкой на отношении** А с условием с называется отношение с тем же заголовком, что и у отношения А, и телом, состоящем из кортежей, значения атрибутов которых при подстановке в условие с дают значение ИСТИНА. с – представляет собой логическое выражение, в которое могут входить атрибуты отношения А и (или) скалярные выражения.

В простейшем случае условие с имеет вид , где – один из операторов =, <, > и т.д., а X и Y – атрибуты отношения A или скалярные значения. Такие выборки называются -выборки (тэта-выборки).

Синтаксис операции выборки: A WHERE c или A WHERE .

6. **Проекцией** отношения A по атрибутам X, Y,…,Z, где каждый из атрибутов принадлежит отношению A, называется отношение с заголовком (X, Y,…,Z) и телом, содержащим множество кортежей вида (x,y,…,z), таких, для которых в отношении A найдутся кортежи со значением атрибута X равным x, значением атрибута Y равным y, …, значением атрибута Z равным z.

Синтаксис операции проекции: A[X,Y,…,Z].

7**. Соединением отношений** А и В по условию с называется отношение (A TIMES B) WHERE c , где с представляет собой логическое выражение, в которое могут входить атрибуты отношений А и В и (или) скалярные выражения.

8**. Деление.** Пусть даны отношения А(X1,X2,…, Xn, Y1,Y2,…, Ym) и B(Y1,Y2,…, Ym), причем атрибуты Y1,Y2,…, Ym- общие для двух отношений. Делением отношений A на B называется отношение с заголовком (X1,X2,…, Xn) и телом, содержащим множество кортежей (x1,x2,…, xn), таких, что для всех кортежей (y1,y2,…, ym) принадлежит B, в отношении A найдется кортеж (x1,x2,…,xn,y1,y2,…,ym).

Синтаксис операции деления: A DEVIDBY B.

Примеры использования реляционной алгебры .

1. Получить имена поставщиков, которые поставляют деталь P2

((SP JOIN S ) WHERE P#=’P2’ ) [SNAME]

2.Получить номера поставщиков ,которые поставляют по крайней мере все те детали , которые поставляет поставщик S2

SP[S#,P#] DEVIDEBY (SP WHERE S#=’S2’)[P#]

4. Получить имена поставщиков которые не поставляют деталь P2

((S[S#] MINUS (SP WHERE P#=’P2’)[S#]

Наиболее распространенные реляцион. СУБД:

FoxPro, Access, Paradox, Clarion - персональные

MS SQL Server, Oracle, Sybase, Interbase, Ingres – корпоративные

**Целью теории реляционной БД** было создание минимального набора таблиц, в котором осуществлялось бы хранение минимально-избыточных данных.

**Осн. свойства реляцион. модели (фундаментальные свойства):**

1. Данные воспринимаются пользователями как таблица, и никак иначе.
2. Каждая таблица сост. из однотипных строк и имеет уникальное имя (в пределах БД).
3. Строки имеют фиксированное число полей и значений. Множественные поля и повторяющиеся группы недопустимы.
4. Таким образом, в каждой позиции таблицы на пересечении строки и столбца всегда имеется одно значение, которое может быть и пустым.
5. Строки таблицы обязательно отличаются друг от друга хотя бы единственным значением, что позволяет однозначно идентифицировать любую строку таблицы.
6. Столбцам таблицы однозначно присваиваются имена, и в каждом из них размещаются однородные значения данных.
7. Полное информационное содержание БД представляется в виде явных значений данных, и такой метод явл-ся единственным, т.е. не сущ. каких-либо связей или указателей, соединяющих одну таблицу с другой.
8. При выполнении операций с таблицей, ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информационному содержанию. Этому способствует наличие имен таблиц, имен столбцов, а так же возможность выделения строки или набора строк с указанными признаками.
9. Записи в таблице располагаются в том порядке, в котором они были туда занесены

2**. Отношения. Переменные отношения. Значения переменных отношения. Базовые переменные отношения и представления. Предикаты и высказывания.**

Определения n-арным отношением R , или R отношением степени n , называют подмножество декартовa произведения множеств D1,D2,D3...Dn (n>=1), не обязательно различных. Исходные множества D1,D2,D3…называют в модели доменами (в СУБД используется понятие тип данных). Отношение имеет простую графическую интерпретацию, оно может быть представлено в виде таблицы, столбцы (поля, атрибуты) которой соответствуют вхождениям доменов в отношение, а строки (записи) — наборам из значений, взятых из исходных доменов. Число строк (кортежей) называют кардинальным числом отношения (кардинальностью), или мощностью отношения. Такая таблица обладает рядом свойств:  В таблице нет двух одинаковых строк.  Таблица имеет столбцы, соответствующие атрибутам отношения.  Каждый атрибут в отношении имеет уникальное имя.  Порядок строк в таблице произвольный. Под атрибутом здесь понимается вхождение домена в отношение. Строки отношения называются кортежами. Далее следует формализованное определение введённых понятий. Заголовок Hr (или схема) отношения r — конечное множество упорядоченных пар вида , где A называется именем атрибута, а T обозначает имя некоторого базового типа или ранее определенного домена, то есть множества допустимых значений. По определению требуется, чтобы все имена атрибутов в заголовке отношения были различны. Кортеж tr, соответствующий заголовку Hr — множество упорядоченных триплетов вида , по одному такому триплету для каждого атрибута в Hr. Третий элемент – v – триплета должен являться допустимым значением типа данных или домена T. Замечание: так как имена атрибутов уникальны, то указание домена в кортеже излишне. Тело Br отношения — неупорядоченное множество различных кортежей tr. Значением Vr отношения r называется пара множеств Hr и Br. Полезно также понятие первичного ключа — это такой набор атрибутов, который однозначно определяет кортеж и минимален среди всех своих подмножеств (то есть нельзя убрать ни один из атрибутов). При добавлении новых записей первичный ключ обязан оставаться первичным ключом (например, неверным будет использование в качестве первичного ключа набора Имя + Отчество + Фамилия сотрудника, даже если на момент создания таблицы полных тёзок среди заносимых в неё людей не было). Осн. понятия: Объект – элемент предметной области, который можно четко идентифицировать. Свойства объекта отображаются с помощью переменных величин, которые являются элементарными единицами информации в рамках БД, и наз-ся атрибутами. Атрибут/поле/столбец – логически неделимый элемент, относящийся к свойствам некоторого объекта или процесса. Атрибуты разделяются на атрибуты-признаки и атрибуты-основания.  Атрибуты – признаки – являются качественной характеристикой объекта.  Атрибуты – основания – характеризуют количественную сторону объекта. Атрибуты имеют множество допустимых значений. Множество всех возможных значений атрибута наз-ся доменом. Совокупность атрибутов, характеризующих один объект, наз-ся записью/кортежем/строкой. Тип записи определяется свойствами объекта. Ключ – атрибут или совокупность атрибутов, однозначно определяющие объект. Потенциальный ключ – ключ, который может идентифицировать объект. Из множества потенциальных ключей выбирается один первичный ключ. Все остальные ключи – альтернативные. Суррогатный ключ – атрибут, который создан для того, чтобы однозначно определять объект. Вторичный ключ – атрибут, который относит объект к некоторой группе. Индексирование таблицы – процесс созд-я индексного файла, в кот-м описано, как отсортировать таблицу по выбранному полю или выражению. В современных СУБД в одном индексном файле может хранится неск-ко индексов. На таблицу может быть наложен фильтр. Фильтр – логич. усл-е, позволяющее отображать только те записи, кот-е удовл. этому условию. В БД сущ. понятие представление – вирт. таблица, кот-я может отображать данные из одной или нескольких таблиц.

**3. Язык SQL. Особенности языка SQL как инструмента работы с реляционными БД. Подмножества языка SQL. Набор операторов языка SQL для определения данных и ограничений целостности.**

Язык SQL(Structured Query Language - структурированный язык запросов) основывается на некоторой смеси алгебраических и логических конструкций, в настоящее время является промышленным стандартом, который в большей или меньшей степени поддерживает любая СУБД, претендующая на звание "реляционной". Язык SQL - наиболее распространённый язык управления базами данных типа клиент – сервер. В SQL определены два подмножества языка: SQL-DDL (Data Definition Language) - язык определения структур и ограничений целостности баз данных. Сюда относятся команды создания и удаления баз данных; создания, изменения и удаления таблиц; управления пользователями и т.д. CREATE SCHEMA - создать схему базы данных   DROP SHEMA - удалить схему базы данных   CREATE TABLE - создать таблицу   ALTER TABLE - изменить таблицу   DROP TABLE - удалить таблицу   CREATE DOMAIN - создать домен   ALTER DOMAIN - изменить домен   DROP DOMAIN - удалить домен   CREATE COLLATION - создать последовательность   DROP COLLATION - удалить последовательность   CREATE VIEW - создать представление   DROP VIEW - удалить представление Каждый оператор CREATE TABLE задает имя создаваемой базовой таблицы, имена и типы данных столбцов этой таблицы, а также первичный ключ таблицы и любые внешние ключи, присутствующие в ней. CREATE TABLE S ( Si CHAR(5), SNAME    CHAR(20), STATUS   NUMERIC(5), CITY      CHAR(15), PRIMARY KEY ( Si ) ) ; SQL-DML (Data Manipulation Language) - язык манипулирования данными: добавление, изменение, удаление и извлечение данных, управления транзакциями SELECT - отобрать строки из таблиц   INSERT - добавить строки в таблицу   UPDATE - изменить строки в таблице   DELETE - удалить строки в таблице   COMMIT - зафиксировать внесенные изменения   ROLLBACK - откатить внесенные изменения

**4. Язык SQL. Особенности языка SQL как инструмента работы с реляционными БД. Подмножества языка SQL. Набор операторов языка SQL для определения данных и ограничений целостности. Набор операторов языка SQL для извлечения, изменения данных.**

**5. Оператор SELECT как проекция результатов реляционных вычислений. Соединение отношений (JOIN) в операторе. Виды соединений (INNER и OUTER, LEFT и RIGHT).**

**SELECT** (*"селект"*) — оператор [DML](http://ru.wikipedia.org/wiki/DML) языка [SQL](http://ru.wikipedia.org/wiki/SQL), возвращающий набор данных (выборку) из [базы данных](http://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), удовлетворяющих заданному условию. В большинстве случаев, выборка осуществляется из одной или нескольких [таблиц](http://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)). В последнем случае говорят об операции слияния (англ. *join*).

При формировании запроса SELECT пользователь описывает ожидаемый набор данных: его вид (набор столбцов) и его содержимое (критерий попадания записи в набор, группировка значений, порядок вывода записей и т.п.).

SELECT [Предикат] Поля FROM Таблицы [IN БазаДанных] [WHERE ...] [GROUP BY ...] [HAVING ...] [ORDER BY ...]; ( Необязательные аргументы заключены в [])

Предикат - одно из четырёх слов ALL, DISTINCT, DISTINCTROW, TOP. Если предикат не указан, то устанавливается ALL. Предикат ALL позволяет отобрать все записи. При использовании предиката DISTINCT, записи, которые содержат повторяющиеся значения в выбранных в запросе полях, исключаются. Предикат DISTINCTROW исключает из выборки записи, если повторяется вся запись, а не одно из полей. Предикат TOP позволяет отобрать определённое количество записей.

Поля - имена одного или нескольких полей, выборка которых производится. Для выборки всех полей вместо имен полей можно поставить звёздочку [\*].

Таблицы - имена одной или нескольких таблиц, из которых производится выборка.

База данных - путь и имя внешней базы данных, в которой содержатся таблицы. Если таблицы находятся в текущей базе данных, то этот аргумент необязателен.

Минимальный синтаксис запроса на выборку выглядит так: SELECT поле FROM Таблица;

Запрос выполняется следующим образом: сначала извлекаются все записи из таблицы, а затем для каждой записи набора проверяется её соответствие заданному критерию. Если осуществляется слияние из нескольких таблиц, то

сначала составляется произведение таблиц, а уже затем из полученного набора отбираются требуемые записи.

* Список полей

SELECT first\_name, last\_name, phone\_no

FROM phone\_list

* Все поля

SELECT \*

FROM phone\_list

* Все поля в произвольном порядке

SELECT first\_name, last\_name, phone\_no,

location, phone\_ext, emp\_no

FROM phone\_list

* Вычисления

SELECT emp\_no, salary, salary \* 1.15

FROM employee

* Литералы

Для придания большей наглядности получаемому результату можно использовать литералы. Литералы - это строковые константы, которые применяются наряду с наименованиями столбцов и, таким образом, выступают в роли "псевдостолбцов". Строка символов, представляющая собой литерал, должна быть заключена в одинарные или двойные скобки.

SELECT first\_name, "получает", salary,

"долларов в год"

FROM employee

* Конкатенация

Имеется возможность соединять два или более столбца, имеющие строковый тип, друг с другом, а также соединять их с литералами. Для этого используется операция конкатенации (||).

SELECT "сотрудник " || first\_name || " " || last\_name

FROM employee

* Использование квалификатора AS

SELECT count(\*) AS number

FROM employee

* Работа с датами

Внешне дата может быть представлена строками различных форматов, например:

* 1. "October 27, 1995"
  2. "27-OCT-1994"
  3. "10-27-95"
  4. "10/27/95"
  5. "27.10.95"

Кроме абсолютных дат, в SQL-выражениях можно также пользоваться относительным заданием дат:

1. "yesterday" - вчера
2. "today" - сегодня
3. "now" - сейчас (включая время)
4. "tomorrow" – завтра

Дата может неявно конвертироваться в строку (из строки), если:

1. строка, представляющая дату, имеет один из вышеперечисленных форматов;
2. выражение не содержит неоднозначностей в толковании типов столбцов.

SELECT first\_name, last\_name, hire\_date

FROM employee

WHERE hire\_date > '1-1-94'

SELECT first\_name, last\_name, hire\_date

FROM employee

WHERE 'today' - hire\_date > 365 \* 7 + 1

* Агрегатные функции

К агрегирующим функциям относятся функции вычисления суммы (SUM), максимального (max) и минимального (MIN) значений столбцов, арифметического среднего (AVG), а также количества строк, удовлетворяющих заданному условию (COUNT).

SELECT count(\*), sum (budget), avg (budget), min (budget), max (budget)

FROM department

WHERE head\_dept = 100

**JOIN** — оператор языка [SQL](http://ru.wikipedia.org/wiki/SQL), позволяющий соединять записи из двух таблиц [реляционной базы данных](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Входит в раздел [FROM](http://ru.wikipedia.org/w/index.php?title=FROM_(SQL)&action=edit&redlink=1) оператора [SELECT](http://ru.wikipedia.org/wiki/Select_(SQL)) и отдельно от него не используется.



**INNER JOIN**

Оператор *внутреннего соединения* INNER JOIN соединяет две таблицы, причём порядок таблиц для оператора неважен. *Тело* результата логически формируется следующим образом. Каждая строка одной таблицы сопоставляется с каждой строкой второй таблицы, после чего для полученной «соединённой» строки проверяется условие соединения (вычисляется предикат соединения). Если условие истинно, в таблицу-результат добавляется соответствующая «соединённая» строка.

**SELECT** \*

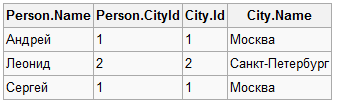
**FROM**

Person

**INNER** **JOIN**

City

**ON** Person.CityId = City.Id

****

### OUTER JOIN

Присоединение таблицы с необязательным присутствием записи в таблице.

#### LEFT OUTER JOIN

К левой таблице присоединяются все записи из правой, соответствующие условию (по правилам **inner join**), плюс все не вошедшие записи из левой таблицы, поля правой таблицы заполняются значениями *NULL*.

**SELECT** \*

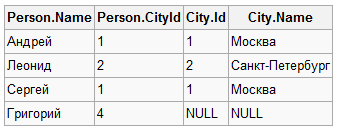
**FROM**

Person

**LEFT** **OUTER** **JOIN**

City

**ON** Person.CityId = City.Id

****

#### RIGHT OUTER JOIN

Аналогично **left outer join**, но применяется для правой таблицы.

К правой таблице присоединяются все записи из левой, соответствующие условию (по правилам **inner join**), плюс все не вошедшие записи из правой таблицы, поля левой таблицы заполняются значениями *NULL*.

**SELECT** \*

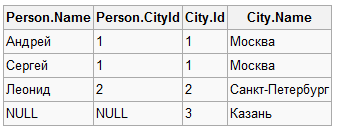
**FROM**

Person

**RIGHT** **OUTER** **JOIN**

City

**ON** Person.CityId = City.Id

****

**6. Оператор SELECT. Переименование атрибутов и отношений в операторе SELECT. Ключевое слово WHERE. Сортировка результатов запроса по значению атрибута.**

Операция переименования атрибутов воплощается следующим алгоритмом:

1) в списке имен атрибутов фразы Select перечисляются те атрибуты, которые необходимо переименовать;

2) к каждому указанному атрибуту добавляется специальное ключевое слово as;

3) после каждого вхождения слова as указывается то имя соответствующего атрибута, на которое необходимо поменять имя исходное.

WHERE — оператор в SQL, указывающий, что оператор языка управления данными (DML) должен действовать только на записи, удовлетворяющие определенным критериям. Критерии должны быть описаны в форме предикатов. Раздел WHERE — не обязательный раздел в SQL (DML) предложениях. Он используется в качестве условия в SQL-запросе для ограничения записей обрабатываемых в выражениях SQL (DML) или возвращаемых запросом.

WHERE — зарезервированное слово языка SQL. Раздел WHERE используется совместно с SQL DML операторами в следующей форме:

SQL-DML-выражение

FROM TABLE\_NAME

WHERE predicate

Все записи, для которых значением предиката раздела WHERE является истина — будут задействованы (или возвращены) в SQL DML выражении (или запросе). Записи, для которых значение предиката будет ложь или неопределенность (NULL) — исключатся из SQL DML выражения или выборки.

Типы предикатов, используемых в предложении WHERE:

сравнение с использованием реляционных операторов

= равно

<> не равно

!= не равно

> больше

< меньше

>= больше или равно

<= меньше или равно

BETWEEN

IN

LIKE

CONTAINING

IS NULL

EXIST

ANY

ALL

SELECT first\_name, last\_name, dept\_no

FROM employee

WHERE job\_code = "Admin"

SELECT first\_name, last\_name, dept\_no, job\_country

FROM employee

WHERE job\_country <> "USA"

SELECT first\_name, last\_name, salary

FROM employee

WHERE salary BETWEEN 20000 AND 30000

SELECT first\_name, last\_name, salary

FROM employee

WHERE last\_name BETWEEN "Nelson" AND "Osborne"

SELECT first\_name, last\_name, salary

FROM employee

WHERE last\_name BETWEEN "Nel" AND "Osb"

SELECT first\_name, last\_name, job\_code

FROM employee

WHERE job\_code IN ("VP", "Admin", "Finan")

SELECT first\_name, last\_name, job\_country

FROM employee

WHERE job\_country NOT IN

("USA", "Japan", "England")

% - замещает любое количество символов (в том числе и 0),

\_ - замещает только один символ.

SELECT first\_name, last\_name

FROM employee

WHERE last\_name LIKE "F%"

SELECT first\_name, last\_name

FROM employee

WHERE first\_name LIKE "%er"

SELECT first\_name, last\_name

FROM employee

WHERE first\_name LIKE "Jacq\_es"

SELECT first\_name, last\_name

FROM employee

WHERE last\_name CONTAINING "ne"

SELECT first\_name, last\_name

FROM employee

WHERE salary NOT CONTAINING 2000

SELECT department, mngr\_no

FROM department

WHERE mngr\_no IS NULL

SELECT \* FROM Salespeople WHERE city = ANY ( SELECT city FROM Customers )

ELECT DISTINCT model, price

FROM Laptop

WHERE price > ALL (SELECT price FROM PC )

SELECT \* FROM Request

WHERE IncomingDate > ALL (SELECT IncomingDate

FROM Request

WHERE FailureCD=2);.

**7. Оператор SELECT. Группировка результатов запроса. Вычисление итогов. Примеры итоговых функций. Отбор результатов по результатам вычисления итогов (выражение HAVING)**

В общем случае для создания вычисляемого (производного) поля в списке SELECT следует указать некоторое выражение языка SQL. В этих выражениях применяются арифметические операции сложения, вычитания, умножения и деления, а также встроенные функции языка SQL. Можно указать имя любого столбца (поля) таблицы или запроса, но использовать имя столбца только той таблицы или запроса, которые указаны в списке предложения FROM соответствующей инструкции. При построении сложных выражений могут понадобиться скобки.

Стандарты SQL позволяют явным образом задавать имена столбцов результирующей таблицы, для чего применяется фраза AS.

Рассчитать общую стоимость для каждой сделки. Этот запрос использует расчет результирующих столбцов на основе арифметических выражений.

SELECT Товар.Название, Товар.Цена,

Сделка.Количество,

Товар.Цена\*Сделка.Количество AS Стоимость

FROM Товар INNER JOIN Сделка

ON Товар.КодТовара=Сделка.КодТовара

Получить список фирм с указанием фамилии и инициалов клиентов.

SELECT Фирма, Фамилия+" "+

Left(Имя,1)+"."+Left(Отчество,1)+"."AS ФИО

FROM Клиент

С помощью итоговых (агрегатных) функций в рамках SQL-запроса можно получить ряд обобщающих статистических сведений о множестве отобранных значений выходного набора.

Пользователю доступны следующие основные итоговые функции:

• Count (Выражение) - определяет количество записей в выходном наборе SQL-запроса;

• Min/Max (Выражение) - определяют наименьшее и наибольшее из множества значений в некотором поле запроса;

• Avg (Выражение) - эта функция позволяет рассчитать среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество.

• Sum (Выражение) - вычисляет сумму множества значений, содержащихся в определенном поле отобранных запросом записей.

SELECT count(\*), sum (budget), avg (budget), min (budget), max (budget)

FROM department

WHERE head\_dept = 100

SELECT Count(\*) AS Количество\_сделок

FROM Сделка

При помощи HAVING отражаются все предварительно сгруппированные посредством GROUP BY блоки данных, удовлетворяющие заданным в HAVING условиям. Это дополнительная возможность "профильтровать" выходной набор.

Разбиение кортежей на группы выполняется с помощью ключевых слов GROUP BY и следующим за ними списком атрибутов, которые определяют группы. Если мы имеем GROUP BY A1, ⃛, Ak мы разделяем отношение на группы так, что два кортежа будут в одной группе, если у них соответствуют все атрибуты A1, ⃛, Ak.

Фраза GROUP BY (группировать по) инициирует перекомпоновку указанной во FROM таблицы по группам, каждая из которых имеет одинаковые значения в столбце, указанном в GROUP BY.

SELECT S.SNO, S.SNAME, COUNT(SE.PNO)

FROM SUPPLIER S, SELLS SE

WHERE S.SNO = SE.SNO

GROUP BY S.SNO, S.SNAME;

Если параметр GROUP BY в SELECT не задан, HAVING применяется к «группе» всех строк таблицы, полностью дублируя WHERE

SELECT DeptID, SUM(SaleAmount) FROM Sales

WHERE SaleDate = '01-Jan-2000'

GROUP BY DeptID

HAVING SUM(SaleAmount) > 1000

**8. Ограничения целостности БД. Классификация ограничений. Предикаты и высказывания. Предикаты переменной отношения и БД. Основное правило целостности данных.**

Термин целостность данных используется для описания точности и корректности хранящейся в базе информации.

Ограничения целостности классифицируются по четырем основным категориям: ограничения типа (домена), атрибута, переменной-отношения и базы данных.

Типы (виды) условий целостности данных:

1. обязательность данных – как только вы войдете в набор данных того или иного поля, пока не введете какие-либо данные, вас система из средства набора не выпустит (NOT NULL).

CREATE TABLE bon

(

id number,

ename varchar2 (50) not null,

sername varchar2 (50)

);

2. проверка на правильность (validity checking)– проверка диапазона значений (правильность введения даты, размера чисел)

CREATE TABLE bon4

(

id number,

ename varchar2 (50) not null,

sal number constraint con\_chk\_bon4 check (sal>= 0),

sername varchar2 (50),

constraint my\_con\_uq2 unique(id)

);

3. целостность (entity integrity) – соответствие внешнего ключа и primary key

CREATE TABLE NBON

(

id number primary key,

ename varchar2 (50) not null,

sal number check (sal>= 0),

sername varchar2 (50)

);

4. ссылочная целостность (referent integrity) – как правило, проверяют в двух местах: на клиенте, на сервере.

СREATE TABLE

(columns columns\_types,

…,

column column\_type [constraint constraint\_name]

REFERENCES parent\_table\_name(parent\_table\_column\_name)

[NULL | ON DELETE CASCADE | ON DELETE SET NULL],

….

);

или же на уровне таблицы

СREATE TABLE

(columns columns\_types,

…,

column\_X column\_X\_type,

constraint constraint\_name FOREIGN KEY (column\_X)

REFERENCES parent\_table\_name(parent\_table\_column\_name)

[NULL | ON DELETE CASCADE | ON DELETE SET NULL],

….

);

где

FOREIGN KEY определяет название ссылаемого поля

REFERENCES определяет название родительской таблицы и его поля

NULL - это значение по умолчанию, то есть при удаление родительской записи, удаление разрешается только если нет дочерних записей ссылающихся на родительскую запись

ON DELETE CASCADE - при удаление родительской записи, автоматом удаляются также все дочерние записи

ON DELETE SET NULL - при удаление родительской записи, автоматом все дочерние записи обновляются на значение NULL

5. непротиворечивость (business правила) – деловые правила, зависит от конкретных СУБД.

CREATE TABLE bon2

(

id number unique,

ename varchar2 (50) not null,

sername varchar2 (50)

);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<определение\_таблицы> ::=

CREATE TABLE имя\_таблицы

{(имя\_столбца тип\_данных [ NOT NULL ][ UNIQUE]

[DEFAULT <значение>]

[ CHECK (<условие\_выбора>)][,...n]}

[CONSTRAINT имя\_ограничения]

[PRIMARY KEY (имя\_столбца [,...n])

{[UNIQUE (имя\_столбца [,...n])}

[FOREIGN KEY (имя\_столбца\_внешнего\_ключа

[,...n])

REFERENCES имя\_род\_таблицы

[(имя\_столбца\_род\_таблицы [,...n])],

[MATCH {PARTIAL | FULL}

[ON UPDATE {CASCADE| SET NULL |SET DEFAULT

|NO ACTION}]

[ON DELETE {CASCADE| SET NULL |SET DEFAULT

|NO ACTION}]

{[CHECK(<условие\_выбора>)][,...n]})

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Классификация ограничений целостности

Ограничения целостности можно классифицировать несколькими способами:

1. По способам реализации.
2. По времени проверки.
3. По области действия.

**Классификация ограничений целостности по способам реализации**

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

1. Декларативная поддержка ограничений целостности.
2. Процедурная поддержка ограничений целостности.

Декларативная поддержка ограничений целостности заключается в определении ограничений средствами языка определения данных (DDL - Data Definition Language). Обычно средства декларативной поддержки целостности (если они имеются в СУБД) определяют ограничения на значения доменов и атрибутов, целостность сущностей (потенциальные ключи отношений) и ссылочную целостность (целостность внешних ключей). Декларативные ограничения целостности можно использовать при создании и модификации таблиц средствами языка DDL или в виде отдельных утверждений (ASSERTION).

Например, следующий оператор создает таблицу PERSON и определяет для нее некоторые ограничения целостности:

CREATE TABLE PERSON

(Pers\_Id INTEGER PRIMARY KEY,

Pers\_Name CHAR(30) NOT NULL,

Dept\_Id REFERENCES DEPART(Dept\_Id) ON UPDATE CASCADE ON DELETE CASCADE);

Процедурная поддержка ограничений целостности заключается в использовании триггеров и хранимых процедур.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных.

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

<Определение\_триггера>::=

{CREATE | ALTER} TRIGGER имя\_триггера

ON {имя\_таблицы | имя\_просмотра }

[WITH ENCRYPTION ]

{

{ { FOR | AFTER | INSTEAD OF }

{ [ DELETE] [,] [ INSERT] [,] [ UPDATE] }

[ WITH APPEND ]

[ NOT FOR REPLICATION ]

AS

sql\_оператор[...n]

} |

{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]

[UPDATE] }

[ WITH APPEND]

[ NOT FOR REPLICATION]

AS

{ IF UPDATE(имя\_столбца)

[ {AND | OR} UPDATE(имя\_столбца)] [...n]

|

IF (COLUMNS\_UPDATES(){оператор\_бит\_обработки}

бит\_маска\_изменения)

{оператор\_бит\_сравнения }бит\_маска [...n]}

sql\_оператор [...n]

}

}

Ограничение базы данных — это ограничение, устанавливающее взаимосвязь между различными переменными-отношениями. Приведем несколько примеров.

CONSTRAINT DBC1

IS EMPTY ( ( S JOIN SP )

WHERE STATUS < 20 AND QTY > QTY ( 500 ) ) ;

Смысл: "Поставщики со статусом, меньшим 20, не могут поставлять детали в количе¬стве свыше 500 штук".

Ранее мы установили, что любое отношение имеет связанный с ним предикат, а кортежи этого отношения выражают истинные высказывания, порождаемые данным предикатом.

Должно быть понятно, что переменная-отношение тоже имеет предикат, а именно — предикат, который будет общим для всех возможных отношений, представляющих допустимые значения для данной переменной-отношения. Например, рассмотрим переменную-отношение поставщиков S. Предикат для этой переменной-отношения можно сформулировать примерно так.

Поставщик с данным номером поставщика (Si) имеет данное имя (SNAME), данное значение статуса (STATUS) и размещается в данном городе (CITY), кроме того, в любой заданный момент никакие два поставщика не имеют одинаковых номеров поставщика.

Также должно быть понятно, что предикат данной переменной-отношения по свой сути является критерием приемлемости изменений для рассматриваемой переменной-отношения, т.е. он предписывает, будет ли допустима определенная операция INSERT, UPDATE или DELETE для данной переменной-отношения. Следовательно, в идеальном случае СУБД должен быть известен и понятен предикат каждой переменной-отношения в базе данных, что позволит системе корректно обрабатывать всевозможные попытки внесения изменений.

Но система "не знает и не понимает" (и "не может знать и понимать") предикат переменной-отношения поставщиков на все 100%. Однако ей известно достаточно хорошее приближение к этому предикату, а конкретнее — ей известны ограничения целостности, которые применимы к записям о поставщиках. Следовательно, мы определяем предикат переменной-отношения для переменной-отношения поставщиков (или в общем случае для любой переменной-отношения) как логическое умножение (логическая операция И) всех ограничений переменной-отношения, которые установлены для данной переменной-отношения.

Теперь можно сформулировать золотое правило (по крайней мере его первую версию).

Ни одна из операций изменения не имеет права переводить переменную-отношение в состояние, нарушающее ее собственный предикат.

**10. Первичный ключ. Внешний ключ. Ссылочная целостность. Базовые ссылочные действия, определенные в стандарте языка SQL.**

Некоторые переменные-отношения вполне могут иметь несколько потенциальных ключей. В таком случае в реляционной модели по традиции (по крайней мере, в случае базовой переменной-отношения) один из потенциальных ключей должен быть выбран в качестве **первичного ключа**, а все остальные потенциальные будут называться **альтернативными**.

С точки зрения теории все потенциальные ключи отношения эквивалентны, то есть обладают одинаковыми свойствами уникальности и минимальности. Однако в качестве первичного обычно выбирается тот из потенциальных ключей, который наиболее удобен для тех или иных практических целей.. Поэтому в качестве первичного ключа, как правило, выбирают тот, который имеет наименьший размер (физического хранения) и/или включает наименьшее количество атрибутов.

Если не придерживаться формальностей, то **внешний ключ** можно определить как множество атрибутов одной переменной-отношения R2, значения которых должны совпадать со значениями некоторого потенциального ключа некоторой другой переменной-отношения R1.

Можно дать следующее более точное определение:

Пусть R2— некоторая переменная-отношение. Тогда внешний ключ (скажем, FK) в переменной-отношении R2 представляет собой множество атрибутов этой переменной-отношения, такое, что:

а) существует переменная-отношение R1 (причем переменные-отношения R1 и R2 необязательно различны) с потенциальным ключом СК;

б) каждое значение внешнего ключа FK в текущем значении переменнойотношения R2 обязательно совпадает со значением ключа СК некоторого кортежа в текущем значении переменной-отношения R1.

Данное определение нуждается в дополнительных пояснениях.

1. По определению каждое значение данного внешнего ключа должно присутствовать в качестве значения соответствующего ему потенциального ключа (который обычно, но не всегда, является первичным ключом).

2. Некоторый внешний ключ будет составным или простым в зависимости от того, является ли простым или составным соответствующий потенциальный ключ.

3. Каждый входящий в некоторый внешний ключ атрибут должен иметь то же имя и тип, что и эквивалентный ему компонент соответствующего потенциаль¬ного ключа.

Значение внешнего ключа представляет ссылку на кортеж, содержащий соответствующее значение потенциального ключа (ссылочный кортеж). Поэтому проблема контроля того, чтобы база данных не включала никаких неверных значений внешних ключей, известна как проблема поддержания ссылочной целостности.

Ограничение, в соответствии с которым значения данного внешнего ключа должны отвечать значениям соответствующих потенциальных ключей, называют **ссылочным ограничением.**

Переменная-отношение, которая содержит внешний ключ, называется ссылающейся переменной отношения, а переменная-отношение, которая содержит соответствующий потенциальный ключ, — ссылочной переменной-отношением.

**Ссылочная целостность.**

FOREIGN KEY { <СПИСОК элементов> } REFERENCES <ямя переменной-отношения>

База данных не должна содержать значений внешних ключей, не имеющих соответствия.

Внешние ключи представляют собой столбцы или наборы столбцов, предназначенные для связывания каждой из строк дочерней таблицы, содержащей этот внешний ключ, со строкой родительской таблицы, содержащей соответствующее значение потенциального ключа. Стандарт SQL предусматривает механизм определения внешних ключей с помощью предложения FOREIGN KEY, а фраза REFERENCES определяет имя родительской таблицы, т.е. таблицы, где находится соответствующий потенциальный ключ. При использовании этого предложения система отклонит выполнение любых операторов INSERT или UPDATE, с помощью которых будет предпринята попытка создать в дочерней таблице значение внешнего ключа, не соответствующее одному из уже существующих значений потенциального ключа родительской таблицы. Когда действия системы выполняются при поступлении операторов UPDATE и DELETE, содержащих попытку обновить или удалить значение потенциального ключа в родительской таблице, которому соответствует одна или более строк дочерней таблицы, то они зависят от правил поддержки ссылочной целостности, указанных во фразах ON UPDATE и ON DELETE предложения FOREIGN KEY. Если пользователь предпринимает попытку удалить из родительской таблицы строку, на которую ссылается одна или более строк дочерней таблицы, язык SQL предоставляет следующие возможности:

CASCADE - выполняется удаление строки из родительской таблицы, сопровождающееся автоматическим удалением всех ссылающихся на нее строк дочерней таблицы;

SET NULL - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы записывается значение NULL;

SET DEFAULT - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы заносится значение, принимаемое по умолчанию;

NO ACTION - операция удаления строки из родительской таблицы отменяется. Именно это значение используется по умолчанию в тех случаях, когда в описании внешнего ключа фраза ON DELETE опущена.

Те же самые правила применяются в языке SQL и тогда, когда значение потенциального ключа родительской таблицы обновляется.

Определитель MATCH позволяет уточнить способ обработки значения NULL во внешнем ключе.

При определении таблицы предложение FOREIGN KEY может указываться произвольное количество раз.

В операторе CREATE TABLE используется необязательная фраза DEFAULT, которая предназначена для задания принимаемого по умолчанию значения, когда в операторе INSERT значение в данном столбце будет отсутствовать.

Фраза CONSTRAINT позволяет задать имя ограничению, что позволит впоследствии отменить то или иное ограничение с помощью оператора ALTER TABLE.

**11. Представления. Отличие представления от базовых переменных отношения**

Исходные (заданные) переменные-отношения называются **базовыми переменными-отношениями**, а присвоенные им значения называются базовыми отношениями. Отношение, которое получено или может быть получено из базового отношения в результате выполнения каких-либо реляционных выражений, называется производным отношением.

Реляционные системы, очевидно, должны предоставлять средства для создания, в первую очередь, базовых переменных-отношений. В языке SQL, например, эта функция обеспечивается оператором CREATE TABLE (здесь слово TABLE используется в узком смысле, как базовая переменная-отношение). Базовые переменные-отношения, конечно же должны быть именованными, как, например, показано ниже.

CREATE TABLE ЕМР

Однако реляционные системы обычно поддерживают еще один вид именованных переменных-отношений, называемых **представлениями**, В любой конкретный момент их значение является производным отношением (и поэтому упрощенно можно считать, что представление— это производная переменная-отношение). Значение данного представления в данное время является результатом вычисления определенного реляционного выражения в данный момент, а упомянутое реляционное выражение определяется в момент создания этого представления.

Поддержка представлений желательна по многим причинам. Укажем некоторые из них.

* Обеспечивается автоматическая защита скрытых данных

Под "скрытыми данными" здесь подразумеваются данные базовых таблиц, кото¬рые не видны в определенном представлении (например, в случае представления GOOD SUPPLIER это имена поставщиков).

* Представления позволяют разным пользователям различным образом видеть од¬ни и те же данные в одно и то же время

Другими словами, представления позволяют различным пользователям сосредо¬точить свое внимание и, возможно, логически реструктуризировать только ту часть базы данных, которая их интересует, игнорируя все остальные сохраняемые данные.

**Различие** между базовой переменной-отношением и представлением часто характеризуется следующим образом.

• Базовые переменные-отношения "реально существуют" в том смысле, что они представляют данные, которые действительно хранятся в базе данных.

• Представления, наоборот, "реально не существуют", а просто предоставляют различные способы просмотра "реальных" данных.

**9. Потенциальные ключи. Свойства. Виды потенциальных ключей. Значение потенциальных ключей.**

**Первичный ключ** (**Primary Key, PK)** – уникальный идентификатор для некоторого отношения.

**Потенциальный ключ К (Candidate Key, CK)** – это подмножество множества атрибутов отношения **R**, обладающих следующими свойствами:

o **cвойство уникальности**: нет двух различных кортежей в отношении **R** с одинаковым значением **K**;

o **свойство неизбыточности**: никакое из подмножеств **К** не обладает свойством уникальности.

Потенциальный ключ, состоящий из более чем одного атрибута, называется **составным (composite key)**. Потенциальный ключ, состоящий из одного атрибута, называется **простым**.

Пример: на рис. 9.1 представлено отношение **Students**, которое имеет следующие атрибуты: код студента, фамилия, имя, отчество, пол, дата рождения.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A1 | A2 | A3 | A4 | A5 | A6 |
| student\_ID | surname | name | lastname | sex | birthday |
|  | Иванов | Иван | Иванович | М | 01.01.1990 |
|  | Петров | Пётр | Петрович | М | 01.10.1990 |
|  | Сидоров | Сидор | Сидорович | М | 01.03.1991 |
|  | Иванов | Глеб | Иванович | М | 01.01.1990 |
|  | Иванов | Иван | Иванович | М | 01.03.1991 |

Рис.9.1. Отношение Students.

Проанализировав данные, представленные на рис. 9.1, мы видим, что одинаковые значения могут принимать атрибуты A2 («Иванов»), A3 («Иван»), A4 («Иванович»), A5 («М»), A6 («01.01.1990»). Только значения атрибута A1 (student\_ID) не повторяются, причем это не зависит от конкретного набора данных: код студента уникален. Таким образом, данное отношение имеет один потенциальный ключ - student\_ID.

|  |  |  |
| --- | --- | --- |
| B1 | B2 | B3 |
| student\_ID | subject | mark |
|  | История |  |
|  | Математический анализ |  |
|  | Философия |  |
|  | История |  |
|  | Философия |  |
|  | Информатика |  |
|  | Философия |  |
|  | Информатика |  |
|  | Математический анализ |  |
|  | Информатика |  |

Рис. 9.2. Отношение Marks, имеющее составной первичный ключ (атрибуты B1 и B2).

Пояснения к определению «потенциальный ключ»:

o на практике отношения, как правило, имеют только один потенциальный ключ;

o потенциальные ключи определены как множество атрибутов.

На рис. 9.2 приведен пример отношения, в котором первичный ключ является составным. Введем некоторые ограничения предметной области для упрощения учебной модели. Предположим, что студенты получают оценки по предметам, при этом, по каждому предмету студент получает оценку только 1 раз. Тогда отношение **Marks** будет состоять из трех атрибутов: код студента, предмет, оценка. Каждый из атрибутов, взятый отдельно от других, имеет повторяющиеся значения. Заметим, что пара атрибутов {student\_ID, subject} не могут иметь повторяющихся значений. В противном случае будет нарушено ограничение предметной области: по каждому предмету студент получает оценку только 1 раз.

**Назначение потенциальных ключей**: обеспечивают основной механизм адресации на уровне кортежей в реляционной системе.

Следствия:

o отношение, не имеющее потенциальных ключей, т.е. допускающее дублирование кортежей, ограничивает отображение нарушений или отклонений от нормального режима работы в определенных обстоятельствах;

o система, не использующая потенциальные ключи, ограничена в возможностях отображения состояния, которое не является реляционным.

**12. Функциональная зависимость. Тривиальные и нетривиальные ФЗ. Замыкание множества ФЗ. Неприводимые ФЗ.**

Проектирование связано с построением **логической структуры** **БД**. Иными словами, нужно решить вопрос, какие базовые отношения, с какими атрибутами следует задать.

Суть этой проблемы сводится, в конечном счете, к **нормализации отношений**.

Сначала рассмотрим основные понятия, необходимые для обсуждения вопросов нормализации отношений.

## 3.1.1. Понятие функциональной зависимости

Вспомним, что любое отношение рассматривается как переменная, принимающая определенные значения в определенные моменты времени.

Пусть R – переменная отношения, X, Y – произвольные подмножества множества всех атрибутов R. Y **функционально зависит** от X тогда и только тогда, когда для любого допустимого значения R каждое значение X связано только с одним значением Y.

Обозначается: **X→Y**

Говорится: «X функционально определяет Y» или «Y функционально зависит от X».

Левая часть выражения называется **детерминантом (детерминантой)**функциональной зависимости (ФЗ), правая – **зависимой частью**ФЗ.

Например, в отношении Студенты (№ЗачетнойКнижки, Фамилия, Имя, Отчество, Адрес, КодГруппы) существуют такие ФЗ

{№ ЗачетнойКнижки} → {Фамилия, Имя, Отчество}

{№ ЗачетнойКнижки} → {Адрес, КодГруппы}

{№ ЗачетнойКнижки, Фамилия, Имя, Отчество} → {Адрес, КодГруппы}

Это лишь некоторые ФЗ, из которых можно сделать вывод, что если детерминант содержит первичный ключ, то множество всех остальных атрибутов отношения функционально зависит от него.

Еще пример: в отношении Кафедры (КодКафедры, НазваниеКафедры, Кабинет, Телефон) существуют ФЗ

{КодКафедры} → {Кабинет, Телефон}

{НазваниеКафедры} → {Кабинет, Телефон}

Таким образом, аналогичный вывод можно сделать не только для первичных ключей, но и для альтернативных, то есть для всех потенциальных ключей.

Множество атрибутов отношения, которое содержит в качестве подмножества потенциальный ключ называется **суперключом** этого отношения.

Рассмотрим еще один пример: если в то же отношение Студенты добавить атрибут СтаростаГруппы, то появятся такие ФЗ:

{КодГруппы} → {СтаростаГруппы}

{СтаростаГруппы} → {КодГруппы}

(причем, ни атрибут КодГруппы, ни атрибут СтаростаГруппы не являются потенциальными ключами)

В этом случае имеется избыточность данных, которая может привести к вводу ошибочных сведений (пользователь случайно может ввести в качестве старосты некоторой группы не того студента, который на самом деле является старостой, но система не выдаст ошибку).

Фактически, если в отношении имеется ФЗ, в которой детерминант не является суперключом, то отношение избыточно.

Существуют такие ФЗ, которые учитываются только формально, т.к. они всегда существуют и подразумеваются самим определением ФЗ. Это тривиальные ФЗ.

**Тривиальная функциональная зависимость** – это такая ФЗ, зависимая часть которой является подмножеством детерминанта.

Например,

{№ЗачетнойКнижки, Фамилия, Имя, Отчество} → {Фамилия, Имя, Отчество}

{КодГруппы, Курс} → {Курс}

Такие тривиальные ФЗ не рассматриваются при нормализации, но все же они существуют и всегда формально учитываются.

## 3.1.2. Правила вывода функциональных зависимостей

Из некоторого множества ФЗ конкретного отношения можно получить производные ФЗ.

Например, из ФЗ отношения Студенты

{№ЗачетнойКнижки, Фамилия, Имя, Отчество} → {Адрес, Телефон}

можно получить такие ФЗ:

{№ ЗачетнойКнижки, Фамилия, Имя, Отчество} → {Адрес}

{№ ЗачетнойКнижки, Фамилия, Имя, Отчество} → {Телефон}

Пусть S – некое множество ФЗ. Тогда множество всех ФЗ, которые можно получить из S называется **замыканием** множества S и обозначается S+.

Чтобы получить замыкание некоторого множества ФЗ нужны правила вывода ФЗ. Такие правила вывода сформулировал Армстронг (Швеция), поэтому их называют **правилами Армстронга** (или **аксиомами Армстронга**).

Обозначим за А, В, С произвольные подмножества множества атрибутов заданного отношения R, а записью АВ будем обозначать объединение А и В.

**Правила вывода ФЗ Армстронга:**

1) Рефлексивность: если В является подмножеством А, то А → В

2) Дополнение: если А → В, то АС → ВС

3) Транзитивность: если А → В и В → С, то А → С

Каждое из этих правил может быть доказано на основе определения ФЗ, а первое правило – это определение тривиальной ФЗ.

Эти правила **полны**, т.к. их достаточно для вывода замыкания (т.е. всех ФЗ) начального множества ФЗ.

Они также **исчерпывающи**, т.к. никакие дополнительные ФЗ не могут быть выведены из начального множества ФЗ.

Но из этих правил для упрощения практического вывода ФЗ можно вывести несколько дополнительных правил (**следствий**):

4) Самоопределение: А → А

5) Декомпозиция: если А → ВС, то А → В и А → С

6) Объединение: если А → В и А → С, то А → ВС

7) Композиция: если А → В и С → D, то АС → ВD

8) Теорема всеобщего объединения: если А → В и С → D, то А(С-В) → ВD

(названо это правило так, потому что многие другие правила могут быть выведены как частные случаи этой теоремы)

Пример: рассмотрим отношение Группы (КодГруппы, Специальность, Курс,Староста).

В качестве начального множества ФЗ возьмем множество из следующих двух ФЗ:

(1) {КодГруппы} → {Специальность, Курс}

(2) {КодГруппы} → {Староста}

Выведем замыкание этого множества ФЗ.

·По правилу 1 можно записать все тривиальные зависимости:

(3) {Специальность, Курс} → {Специальность}

(4) {Специальность, Курс} → {Курс}

·По правилу 2:

(5) {КодГруппы, Староста} → {Специальность, Курс, Староста}

(6) {КодГруппы, Специальность} → {Староста, Специальность}

(7) {КодГруппы, Курс} → {Староста, Курс}

(8) {КодГруппы, Специальность, Курс} → {Староста, Специальность, Курс}

·По правилу 3 напрямую ничего не выведем.

·По правилу 4:

(9) {КодГруппы} → {КодГруппы}

(10) {Специальность} → {Специальность}

и т.д. (11), (12)

·По правилу 5:

(13) {КодГруппы} → {Специальность}

(14) {КодГруппы} → {Курс}

·По правилу 6:

(15) {КодГруппы} → {Специальность, Курс, Староста}

·По правилу 7 напрямую ничего не выведем

·По правилу 8 тоже.

Однако, знание этих правил и умение ими пользоваться не обеспечивают вывода замыкания множества ФЗ, т.к. не существует четкого алгоритма такого вывода.

## 3.1.3. Неприводимые функциональные зависимости

Функциональная зависимость называется **неприводимой** слева, если ни один атрибут не может быть опущен из ее детерминанта без нарушения зависимости (иными словами, детерминант неизбыточен).

Например: ФЗ {№ЗачетнойКнижки, Фамилия, Имя, Отчество} → {Адрес} приводима, т.к. из детерминанта можно исключить атрибуты Фамилия, Имя, Отчество без нарушения ФЗ.

А ФЗ {КодГруппы, Дисциплина} → {Преподаватель} отношения Занятия (КодГруппы, Дисциплина, Преподаватель) неприводима, т.к. из детерминанта нельзя исключить ни атрибут КодГруппы, ни атрибут Дисциплина без нарушения зависимости.

Множество ФЗ называется **неприводимым** тогда и только тогда, когда выполняются три свойства:

1) зависимая часть каждой ФЗ из данного множества содержит только один атрибут;

2) каждая ФЗ из данного множества является неприводимой слева;

3) ни одна ФЗ из данного множества не может быть опущена.

Например, рассмотрим множество ФЗ отношения Группы из примера в предыдущем п.3.1.2.

(1) {КодГруппы} → {Специальность, Курс}

(2) {КодГруппы} → {Староста}

(3) {Специальность, Курс} → {Специальность}

(4) {Специальность, Курс} → {Курс}

(5) {КодГруппы, Староста} → {Специальность, Курс, Староста}

(6) {КодГруппы, Специальность} → {Староста, Специальность}

(7) {КодГруппы, Курс} → {Староста, Курс}

(8) {КодГруппы, Специальность, Курс} → {Староста, Специальность, Курс}

(9) {КодГруппы} → {КодГруппы}

(10) {Специальность} → {Специальность}

(11) {Курс} → {Курс}

(12) {Староста} → {Староста}

(13) {КодГруппы} → {Специальность}

(14) {КодГруппы} → {Курс}

(15) {КодГруппы} → {Специальность, Курс, Староста}

По первому свойству исключаем из этого множества следующие ФЗ: (1), (5), (6), (7), (8), (15).

По второму свойству исключаем следующие ФЗ: (3), (4).

По третьему свойству исключаем: (9), (10), (11), (12).

Остались:

(2) {КодГруппы} → {Староста}

(13) {КодГруппы} → {Специальность}

(14) {КодГруппы} → {Курс}

Полученное множество ФЗ неприводимо. Можно сделать вывод, что полученное множество ФЗ выражает то, что отношение Группы содержит атрибуты КодГруппы, Староста, Специальность, Курс, и КодГруппы – первичный ключ.

## 3.1.4. Диаграммы (схемы) функциональных зависимостей

Множество неприводимых ФЗ некоторого отношения можно представить в виде диаграммы функциональных зависимостей.

Из таких диаграмм лучше видно, какие ФЗ включить в множество ФЗ, а какие исключить, чтобы оно было неприводимо. Исключить нужно те стрелки (обозначающие ФЗ), которые идут не от потенциального ключа.

|  |
| --- |
|  |
|  | http://siblec.ru/mod/html/content/8sem/072/072.files/image009.gif |

Таким образом, с помощью диаграмм можно привести множество ФЗ к неприводимому состоянию.

В качестве примера рассмотрим диаграмму первоначального множества ФЗ из п. 3.1.2.

|  |
| --- |
|  |
|  | http://siblec.ru/mod/html/content/8sem/072/072.files/image010.gif |

После того, как лишние стрелки будут убраны, получим диаграмму:

Из нее как раз и следуют перечисленные ранее три оставшиеся ФЗ:

{КодГруппы} → {Курс}

{КодГруппы} → {Специальность}

{КодГруппы} → {Староста}

Таким образом, существует как минимум два способа преобразования множества ФЗ к неприводимому состоянию - путем проверки всех свойств их определения и путем анализа диаграмм ФЗ.

# 3.2. Тривиальные и нетривиальные зависимости

Очевидным способом сокращения размера множества ФЗ было бы исключение тривиальных зависимостей, т.е. таких, которые не могут не выполняться. В качестве примера приведем тривиальную ФЗ для отношения ***Поставки\_из\_города***:

{ *Код\_поставщика, Код\_товара*} ® *Код\_Поставщика*

**Фактически ФЗ тривиальна тогда и только тогда, когда правая часть символической записи данной зависимости является подмножеством левой части.**

***Функциональная зависимость называется нетривиальной, если, по крайней мере, один из атрибутов Bi в функциональной зависимости A1A2 … An ® B1B2 … Bm не является элементом множества {A1A2 … An}.***

***Функциональная зависимость называется полностью нетривиальной. Если ни один из атрибутов Bi не является элементом множества {A1A2 … An}.***

## 3.3. Замыкание множества зависимостей

Предположим: {A1A2 … An} – множество атрибутов, а S – множество функциональных зависимостей.

***Замыканием множества {A1A2 … An} при условии выполнения функциональных зависимостей S называют множество B атрибутов, такое, что для каждого отношения, которому удовлетворяют все функциональные зависимости множества S справедлива и функциональная зависимость***

***A1A2 … An***à***B***

Замыкание множества атрибутов {A1A2 … An} обозначают как {A1A2 … An}+. Алгоритм вычисления замыкания множества атрибутов {A1A2 … An}, по отношению к некоторому множеству функциональных зависимостей имеет следующий вид:

1. Пусть Х представляет множество атрибутов, подлежащее расширению до множества достижения замыкания. Х инициализируется значением {A1A2 … An}.

2. Выполняется поиск некоторой функциональной зависимости B1B2 … Bm® С, такой, что все атрибуты B1B2 … Bmпринадлежат множеству Х, а С – нет. Если указанная функциональная зависимость существует. С добавляется в множество Х.

3. Шаг 2 повторяется до тех пор, пока существуют соответствующие функциональные зависимости, атрибуты которых подлежат включению в множество Х. Поскольку Х допускает только расширение, то количество атрибутов, которые могут быть добавлены в Х, рано или поздно будет исчерпано.

4. После завершения процедуры расширения множество Х содержит искомое значение замыкания {A1A2 … An}+.

Рассмотрим отношение ***R*** с атрибутами А, В, С, D, E, F. Предположим, что отношению R удовлетворяют следующие функциональные зависимости:

AB ® С,

BC ® AD,

D ® E,

CF ® B.

Что представляет собой замыкание множества {A,В}, т.е. {A,В}+.

Вначале предположим, что Х = {A,В}. Атрибуты левой части зависимости AB ® С присутствуют в Х. Получим

Х = {A,В,С}. Теперь Х содержит атрибуты левой части функциональной зависимости BC ® AD. Следовательно, атрибут А и атрибут D мы можем добавить в Х. А уже является членом множества Х, а атрибут D - нет. Включаем атрибут D в множество Х. Получим Х = {A,В,С,D}. Далее воспользуемся зависимостью D ® E и добавим Е в множество Х. Получим Х = {A,В,С,D,Е}. Рассмотрим последнюю зависимость CF ® B. Эта зависимость не даст дальнейшего расширения множества Х, так как атрибут F не является элементом множества Х. Таким образом,

{A,В}+= {A,В,С,D,Е}.

Описанный алгоритм позволяет проверить, является ли любая заданная функциональная зависимость A1A2 … An ® B из множества функциональных зависимостей S. Вначале вычислим {A1A2 … An}+,используя S. Если В является членом {A1A2 … An}+, функциональная зависимость A1A2 … An ® B следует из S. В противном случае -- не является.

Первая попытка решить проблему поиска функциональных зависимостей на основе заданных была предпринята в статье Армстронга. В ней автор предложил набор правил вывода ФЗ, которые были названы аксиомами Армстронга.

1. **1,2 и 3 НФ. НФБК. Нормализация БД.**

**https://habr.com/post/254773/**

**14. Многозначные зависимости и 4 НФ**

**15. Зависимости соединения и 5НФ**

**http://src-code.net/zavisimosti-soedineniya-i-pyataya-normalnaya-forma/**

**16. Транзакции. Роль транзакций в работе БД. Транзакция как логическая единица работы БД. Подтверждение и откат транзакций. Свойства транзакций.**

Для поддержания целостности данных (БД в актуальном состоянии) необходимо внутри программы организовать некую структуру, которая позволит поддерживать БД в непротиворечивом состоянии при различных ситуациях, включая сбои. Этот механизм для поддержания целостности БД называется **транзакцией**.

**Транзакция**– неделимая программная единица (логическая единица программы), обладающая свойством восстановления.

**Особенность транзакции:**

1.Если внутри транзакции происходит сбой или не выполняется хотя бы 1 команда (прерывается), то система осуществляет откат и возвращает БД в состояние, которое было до транзакции.

2.Если удачно завершились все команды тран-и, то система фиксирует новое состояние БД.

**Для транзакции используются команды**:

1.begin transaction – начало транзакции;

2.commit transaction - закрепление, удачное завершение транзакции;

3.rollback transaction - откат транзакции, восстановление текущего состояния БД.

**Различают 2 модели транзакции**:

1. Модель автоматического выполнения транзакции – транзакции автоматически начинается с выполнением пользователем или программой первой инструкции - begin. Здесь четко участвуют 3 команды: begin, commit, rollback. и возможны 2 исхода:

2. Модель управляемого выполнения транзакции – появляется еще один оператор, который позволяет внутри транзакции создать – **точку сохранения** и присвоить сохраненному состоянию имя точки. Здесь наблюдается промежуточные стадии по сохранению транзакции ( оператор SAVE TRANSACTION).

В случае неудачного завершения транзакции происходит откат к точке фиксации, кот является промежуточной.

Для сохранения БД и фиксации команд транзакции существует **журнал транзакций**. Он позволяет во внешней памяти сохранять промежуточные состояния БД в программе.

**Точки фиксации** позволяют запомнить текущее состояние БД перед транзакцией и в случае отката они являются основой для восстановления данных. Промежуточные точки фиксации в управляемых транзакция называют точками сохранения. Если транзакция завершается удачно, то организуется новая точка фиксации.

Транзакция в результате своей работы воздействует на состояние БД двумя способами:

· Явно – выполняя команды INSERT, DELETE, UPDATE;

· Неявно – возможно влияние на работу других транзакций, когда эти транзакции считывает данные, измененные текущей транзакцией.

При явном воздействии результаты работы транзакции могут быть аннулированы с помощью ROLLBACK.При неявном воздействии может возникнуть ситуация, при которой транзакция считывает данные, измененные другой транзакцией, после чего эти изменения аннулируются. В результате транзакция получает данные, не соответствующие целостному состоянию БД. Устранение такого вторичного влияния реализуется на основе механизма блокировок.

**Свойства транзакции:**

1. Атомарность - транзакция выполняется как атомарная операция - либо выполняется вся транзакция целиком, либо она целиком не выполняется.

2. Согласованность - транзакция переводит базу данных из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние. Внутри транзакции согласованность базы данных может нарушаться.

3. Изоляция – транзакции разных пользователей не должны мешать друг другу (например, как если бы они выполнялись строго по очереди).

4. Долговечность - если транзакция выполнена, то результаты ее работы должны сохраниться в базе данных, даже если в следующий момент произойдет сбой системы.

**Транзакция** – это логическая единица работы, которая лежит в основе проблемы параллелизма.

**Параллелизм**- возможность одновременной обработки в СУБД нескольких транзакций (параллельное выполнение транзакции). В связи с тем, что могут возникнуть проблемы в параллельных транзакциях, в системе существует механизм, позволяющий управлять транзакциями.

Главная задача управления параллелизмом заключается в следующем:− или все изменения, сделанные транзакцией, фиксируются и делаются постоянными (успешное завершение транзакции), или транзакция никак не влияет на состояние БД (ошибки при выполнении транзакции, откат транзакции).

**Основные проблемы, возникающие при параллелизме:**

**1. Пропавшие обновления.**Пусть выполняются параллельно две транзакции, каждая из которых выполняет извлечение некоторой записи, ее анализ и последующее обновление. В зависимости от того, какая из двух транзакций завершится позднее, будут утеряны изменения, сделанные ранее завершенной транзакцией.

**2. Чтение «грязных» данных**. Проблема незафиксированной зависимости. Здесь можно, в свою очередь, выделить две ситуации, каждая из которых определяется тем, что одна из двух параллельно выполняющихся транзакций завершается откатом.

а) Чтение грязных данных - Транзакция 1 прочитала данные, измененные транзакцией 2, но в последующем транзакция 2 отменила свои изменения.

б) Отсутствие потерянных изменений - Транзакция 1 изменяет данные после того, как эти данные были изменены транзакцией 2. Если транзакция 2 аннулирует свои изменения, состояние БД будет восстановлено таким, каким оно было перед началом выполнения транзакции 2. А значит, изменения, сделанные транзакцией 1, также будут аннулированы.

**3. Чтение несогласованных данных.** Проблема несовместимого анализа. Здесь также можно рассмотреть две ситуации:

а) Отсутствие неповторяющихся чтений - Транзакция 2 изменяет данные, прочитанные транзакцией 1. Если после этого транзакция 1 повторно читает те же данные, она получит уже другие результаты.

б) **Строки-призраки** Фантомы - Транзакция 1 считывает несколько записей, удовлетворяющих некоторому условию, после чего транзакция 2 добавляет новую запись, удовлетворяющую этому условию. Если после этого транзакция 1 повторно прочитает записи, удовлетворяющие тому же условию, она получит другой набор записей – в нем появились дополнительные записи-фантомы.

**Стандартом SQL-92 предусмотрено 4 уровня изоляции:**

· незафиксированное чтения (READ UNCOMITTED);

· зафиксированное чтения (READ COMITTED);

· повторяющееся чтения (REPETABLE READ);

· сериализация (SERIALIZABLE).

Если в транзакции установлен определенный уровень изоляции, то для нее гарантировано невозникновение определенных эффектов.

Уровень изоляции может устанавливаться для всего приложения или для отдельной транзакции.

Уровень изоляции, установленные стандартом, во-первых, определены недостаточно четко, во-вторых, , не являются обязательными для начального уровня соответствия стандарту. Поэтому реальные СУБД достаточно

**Сериализация транзакций** – последовательное выполнение. Это механизм выполнения транзакций по такому плану, когда результат совместного выполнения транзакций эквивалентен результату их последовательного выполнения.

В сериализации возникают следующие проблемы:

**1.**транзакция 2 пытается изменить объект, измененный не закончившейся транзакции 1.(WW конфликт)

**2.**транзакция 2 пытается изменить объект, прочитанный незаконченной транзакцией 1.(RW)

**3.**транзакция 2 пытается читать объект, измененный незаконченной транзакцией 1.(WR)

Для обеспечения сериализации транзакций. применяются методы **«захвата»** и **«освобождения»** объекта. Транзакция захватывает объект – это приводит к его **блокировке** для другой транзакции. И освобождает его только при своем завершении. При этом совместимытолько захваты несколькими транзакциями на чтение объекта.

**Выделяют 2 основных режима захватов (блокировки):**

**1.**совместный режим **S-режим** (блокировка с доступом) – отвечающий за разделяемый захват объекта, чтобы можно было выполнять чтение.

**2.**монопольный **X-режим** (блокировка без взаимного доступа) – необходим для выполнения операции записи, удаления, модификации.

К методу управления параллелизмом относится **блокировка**. Она нужна для объекта, чтобы предотвратить его непредсказуемые изменения. Если проанализировать приведенные выше проблемы, можно увидеть, что конфликты возникают, когда транзакции параллельно пытаются выполнить операции чтения или записи данных.

Для устранения таких конфликтов используется механизм блокировок.

Протокол двухфазной блокировки устанавливает следующие правила наложения и снятия блокировок:

1. Перед выполнением каких-либо операций с некоторым объектом транзакция должна установить на этот объект соответствующую блокировку.

2. После снятия блокировки транзакция не должна накладывать никаких других блокировок.

Таким образом, транзакция, подчиняющаяся протоколу двухфазной блокировки, имеет две фазы: фаза наложения блокировки – обычно при выполнении соответствующей команды SQL, фаза снятия блокировки – обычно при выполнении команд COMMIT или ROLLBACK.

**Для транзакции используются команды SQL**:

1.begin transaction – начало транзакции;

2.commit transaction - закрепление, удачное завершение транзакции;

3.rollback transaction - откат транзакции, восстановление текущего состояния БД.

**17. Восстановление системы. Описание процесса восстановления, транзакции как единицы восстановления. Средства поддержки транзакция в языке SQL**

|  |
| --- |
|  |

Транзакции представляют собой основную единицу восстановления в системах с базами данных. Именно диспетчер восстановления СУБД обеспечивает поддержку двух из четырех основных свойств ACID транзакций (неразрывность и устойчивость) даже при наличии сбоев в системе. Диспетчер восстановления должен обеспечить, что при восстановлении после сбоя для каждой отдельной транзакции в базе данных будут постоянно фиксироваться либо все внесенные ею изменения, либо ни одно из них. Ситуация осложняется тем фактом, что запись в базу данных не представляет собой неразрывного действия (выполняемого за один шаг), и поэтому существует вероятность, что, когда выполнение транзакции будет завершено путем фиксации, внесенные ею изменения не будут реально отражены в базе данных по той простой причине, что еще не достигли файлов базы данных.

Буфера СУБД занимают определенную часть оперативной памяти и используются для обмена данными со вторичной памятью системы. Только после того как соответствующий буфер выгружен во вторичную память, можно считать, что выполненные операции обновления приобрели постоянный характер. Выгрузка буферов в базу данных может инициироваться по специальной команде (например, по команде фиксации транзакции) или же автоматически, как только буфер будет заполнен. Выдачу явного указания о необходимости записи содержимого буферов во вторичную память называют принудительной записью.

Если отказ системы произойдет между записью данных в буфер и выгрузкой буфера во вторичную память, диспетчер восстановления должен уточнить состояние транзакции, выполнявшей запись в момент аварии. Если транзакция уже выдала команду фиксации, то для обеспечения устойчивости ее результатов диспетчер восстановления должен выполнить ее повторно (redo) (эту операцию часто называют накатом), чтобы восстановить все внесенные ею изменения.

С другой стороны, если на момент отказа системы транзакция еще не была зафиксирована, диспетчер восстановления должен отменить (undo) любые ее результаты (выполнить их откат), что будет гарантировать соблюдение ее неразрывности. Если требуется выполнить откат только одной транзакции, то это — частичный откат. Частичный откат может инициироваться планировщиком, когда транзакция откатывается и перезапускается по требованию протокола управления параллельным доступом, как описано в предыдущем разделе. Кроме того, транзакция может отменяться в одностороннем порядке, например, по требованию пользователя или в результате возникновения исключительной ситуации в прикладной программе. Если требуется выполнить откат всех активных транзакций, то это — глобальный откат.

**18. Три основные проблемы организации параллельной работы. Блокировка, виды блокировок. Предотвращение взаимоблокировок**

### Глава 10. Транзакции и параллелизм

В данной главе изучаются возможности параллельного выполнения транзакций несколькими пользователями, т.е. свойство (И) - изолированность транзакций.

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 их 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. Трудность состоит в том, что если не предпринимать никаких специальных мер, то данные измененные одним пользователем могут быть изменены транзакцией другого пользователя раньше, чем закончится транзакция первого пользователя. В результате, в конце транзакции первый пользователь увидит не результаты своей работы, а неизвестно что.

Одним из способов (не единственным) обеспечить независимую параллельную работу нескольких транзакций является метод блокировок.

#### Работа транзакций в смеси

Транзакция рассматривается как последовательность элементарных атомарных операций. Атомарность отдельной элементарной операции состоит в том, что СУБД гарантирует, что, с точки зрения пользователя, будут выполнены два условия:

1. Эта операция будет выполнена целиком или не выполнена вовсе (атомарность - все или ничего).
2. Во время выполнения этой операции не выполняются никакие другие операции других транзакций (строгая очередность элементарных операций).

Например, элементарными операциями транзакции будут считывание страницы данных с диска или запись страницы данных на диск (страница данных - это минимальная единица для дисковых операций СУБД). Условие 2 на самом деле является именно логическим условием, т.к. реально система может выполнять несколько различных элементарных операций в один и тот же момент. Например, данные могут храниться на нескольких физически различных дисках и операции чтения-записи на эти диски могут выполняться одновременно.

Элементарные операции различных транзакций могут выполняться в произвольной очередности (конечно, внутри каждой транзакции последовательность элементарных операций этой транзакции является строго определенной). Например, если есть несколько транзакций, состоящих из последовательности операций элементарных:

http://citforum.ru/database/dblearn/image350.gif,

http://citforum.ru/database/dblearn/image351.gif,

http://citforum.ru/database/dblearn/image352.gif

то реальная последовательность, в которой СУБД выполняет эти транзакции может быть, например, такой:

http://citforum.ru/database/dblearn/image353.gif.

*Определение 1*. Набор из нескольких транзакций, элементарные операции которых чередуются друг с другом, называется ***смесью транзакций***.

*Определение 2*. Последовательность, в которой выполняются элементарные операции заданного набора транзакций, называется ***графиком запуска*** набора транзакций.

Замечание. Очевидно, что для заданного набора транзакций может быть несколько (вообще говоря, достаточно много) различных графиков запуска.

Обеспечение изолированности пользователей, таким образом, сводится к выбору подходящего (в каком-то смысле правильного) графика запуска транзакций. Одновременно с этим график запуска должен быть оптимальным в некотором смысле, например, давать минимальное среднее время выполнения транзакций каждым пользователем.

Далее мы уточним понятие "правильного" графика и сделаем некоторые замечания об оптимальности.

#### Проблемы параллельной работы транзакций

Каким образом транзакции различных пользователей могут мешать друг другу? Различают три основные проблемы параллелизма:

* ***Проблема потери результатов обновления***.
* ***Проблема незафиксированной зависимости*** (***чтение "грязных" данных***, ***неаккуратное считывание***).
* ***Проблема несовместимого анализа***.

Рассмотрим подробно эти проблемы.

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 http://citforum.ru/database/dblearn/image354.gif будем обозначать http://citforum.ru/database/dblearn/image355.gif, где http://citforum.ru/database/dblearn/image356.gif - прочитанное значение. Операцию записи значения http://citforum.ru/database/dblearn/image357.gifв строку http://citforum.ru/database/dblearn/image354.gif будем обозначать http://citforum.ru/database/dblearn/image358.gif.

#### 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 http://citforum.ru/database/dblearn/image355.gif | http://citforum.ru/database/dblearn/image359.gif | --- |
| --- | http://citforum.ru/database/dblearn/image360.gif | Чтение http://citforum.ru/database/dblearn/image355.gif |
| Запись http://citforum.ru/database/dblearn/image358.gif | http://citforum.ru/database/dblearn/image361.gif | --- |
| --- | http://citforum.ru/database/dblearn/image362.gif | Запись http://citforum.ru/database/dblearn/image363.gif |
| Фиксация транзакции | http://citforum.ru/database/dblearn/image364.gif | --- |
| --- | http://citforum.ru/database/dblearn/image365.gif | Фиксация транзакции |
| **Потеря результата обновления** |  |  |

Результат. После окончания обеих транзакций, строка http://citforum.ru/database/dblearn/image354.gif содержит значение http://citforum.ru/database/dblearn/image366.gif, занесенное более поздней транзакцией B. Транзакция A ничего не знает о существовании транзакции B, и естественно ожидает, что в строке http://citforum.ru/database/dblearn/image354.gif содержится значение http://citforum.ru/database/dblearn/image357.gif. Таким образом, транзакция A потеряла результаты своей работы.

#### 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** |
| --- | http://citforum.ru/database/dblearn/image359.gif | Чтение http://citforum.ru/database/dblearn/image355.gif |
| --- | http://citforum.ru/database/dblearn/image360.gif | Запись http://citforum.ru/database/dblearn/image358.gif |
| Чтение http://citforum.ru/database/dblearn/image367.gif | http://citforum.ru/database/dblearn/image361.gif | --- |
| Работа с прочитанными данными http://citforum.ru/database/dblearn/image357.gif | http://citforum.ru/database/dblearn/image362.gif | --- |
| --- | http://citforum.ru/database/dblearn/image364.gif | Откат транзакции http://citforum.ru/database/dblearn/image368.gif |
| Фиксация транзакции | http://citforum.ru/database/dblearn/image365.gif | --- |
| **Работа с "грязными" данными** |  |  |

С чем же работала транзакция A?

Результат. Транзакция A в своей работе использовала данные, которых нет в базе данных. Более того, транзакция A использовала данные, которых нет, и *не было* в базе данных! Действительно, после отката транзакции B, должна восстановиться ситуация, как если бы транзакция B вообще *никогда не выполнялась*. Таким образом, результаты работы транзакции A некорректны, т.к. она работала с данными, отсутствовавшими в базе данных.

#### Проблема несовместимого анализа

Проблема несовместимого анализа включает несколько различных вариантов:

* ***Неповторяемое считывание***.
* ***Фиктивные элементы*** (***фантомы***).
* ***Собственно несовместимый анализ***.

#### 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 http://citforum.ru/database/dblearn/image355.gif | http://citforum.ru/database/dblearn/image359.gif | --- |
| --- | http://citforum.ru/database/dblearn/image360.gif | Чтение http://citforum.ru/database/dblearn/image355.gif |
| --- | http://citforum.ru/database/dblearn/image361.gif | Запись http://citforum.ru/database/dblearn/image358.gif |
| --- | http://citforum.ru/database/dblearn/image362.gif | Фиксация транзакции |
| Повторное чтение http://citforum.ru/database/dblearn/image367.gif | http://citforum.ru/database/dblearn/image364.gif | --- |
| Фиксация транзакции | http://citforum.ru/database/dblearn/image365.gif | --- |
| **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.

Результат. Транзакция A работает с данными, которые, с точки зрения транзакции A, самопроизвольно изменяются.

#### Фиктивные элементы (фантомы)

Эффект фиктивных элементов несколько отличается от предыдущих транзакций тем, что здесь за один шаг выполняется достаточно много операций - чтение одновременно нескольких строк, удовлетворяющих некоторому условию.

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 http://citforum.ru/database/dblearn/image369.gif.  (Отобрано n строк) | http://citforum.ru/database/dblearn/image359.gif | --- |
| --- | http://citforum.ru/database/dblearn/image360.gif | Вставка новой строки, удовлетворяющей условию http://citforum.ru/database/dblearn/image369.gif. |
| --- | http://citforum.ru/database/dblearn/image361.gif | Фиксация транзакции |
| Выборка строк, удовлетворяющих условию http://citforum.ru/database/dblearn/image369.gif.  (Отобрано n+1 строк) | http://citforum.ru/database/dblearn/image362.gif | --- |
| Фиксация транзакции | http://citforum.ru/database/dblearn/image364.gif | --- |
| **Появились строки, которых раньше не было** |  |  |

Транзакция A ничего не знает о существовании транзакции B, и, т.к. сама она не меняет ничего в базе данных, то ожидает, что после повторного отбора будут отобраны те же самые строки.

Результат. Транзакция A в двух одинаковых выборках строк получила разные результаты.

#### Собственно несовместимый анализ

Эффект собственно несовместимого анализа также отличается от предыдущих примеров тем, что 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 A** | **Время** | **Транзакция B** |
| Чтение счета http://citforum.ru/database/dblearn/image370.gif и суммирование.  http://citforum.ru/database/dblearn/image371.gif | http://citforum.ru/database/dblearn/image359.gif | --- |
| --- | http://citforum.ru/database/dblearn/image360.gif | Снятие денег со счета http://citforum.ru/database/dblearn/image372.gif.  http://citforum.ru/database/dblearn/image373.gif |
| --- | http://citforum.ru/database/dblearn/image361.gif | Помещение денег на счет http://citforum.ru/database/dblearn/image374.gif.  http://citforum.ru/database/dblearn/image375.gif |
| --- | http://citforum.ru/database/dblearn/image362.gif | Фиксация транзакции |
| Чтение счета http://citforum.ru/database/dblearn/image376.gif и суммирование.  http://citforum.ru/database/dblearn/image377.gif | http://citforum.ru/database/dblearn/image364.gif | --- |
| Чтение счета http://citforum.ru/database/dblearn/image378.gif и суммирование.  http://citforum.ru/database/dblearn/image379.gif | http://citforum.ru/database/dblearn/image365.gif | --- |
| Фиксация транзакции | http://citforum.ru/database/dblearn/image380.gif | --- |
| **Сумма $250 по всем счетам неправильная - должно быть $300** |  |  |

Результат. Хотя транзакция B все сделала правильно - деньги переведены без потери, но в результате транзакция A 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.

#### Конфликты между транзакциями

Итак, анализ проблем параллелизма показывает, что если не предпринимать специальных мер, то при работе в смеси нарушается свойство (И) транзакций - изолированность. Транзакции реально мешают друг другу получать правильные результаты.

Однако не всякие транзакции мешают друг другу. Очевидно, что транзакции не мешают друг другу, если они обращаются к *разным данным* или выполняются в *разное время*.

*Определение 3*. 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:

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

Конфликты типа R-R (Чтение - Чтение) отсутствуют, т.к. данные при чтении не изменяются.

Другие проблемы параллелизма (фантомы и собственно несовместимый анализ) являются более сложными, т.к. принципиальное отличие их в том, что они не могут возникать при работе с одним объектом. Для возникновения этих проблем требуется, чтобы транзакции работали с целыми *наборами*данных.

*Определение 4*. График запуска набора транзакций называется ***последовательным***, если транзакции выполняются строго по очереди, т.е. элементарные операции транзакций не чередуются друг с другом.

*Определение 5*. Если график запуска набора транзакций содержит чередующиеся элементарные операции транзакций, то такой график называется ***чередующимся***.

При выполнении последовательного графика гарантируется, что транзакции выполняются правильно, т.е. при последовательном графике транзакции не "чувствуют" присутствия друг друга.

*Определение 6*. Два графика называются ***эквивалентными***, если при их выполнении будет получен один и тот же результат, *независимо от начального состояния базы данных*.

*Определение 7*. График запуска транзакции называется ***верным*** (***сериализуемым***), если он эквивалентен какому-либо последовательному графику.

Замечание. При выполнении двух различных последовательных (а, следовательно, верных) графиков, содержащих один и тот же набор транзакций, могут быть получены *различные* результаты. Действительно, пусть транзакция A заключается в действии "Сложить X с 1", а транзакция B - "Удвоить X". Тогда последовательный график {A, B} даст результат 2(X+1), а последовательный график {B, A} даст результат 2X+1. Таким образом, может существовать несколько верных графиков запусков транзакций, приводящих к разным результатам при одном и том же начальном состоянии базы данных.

Задача обеспечения изолированной работы пользователей не сводится просто к нахождению правильных (сериальных) графиков запусков транзакций. Если бы этого было достаточно, то лучшим был бы простейший способ сериализации - ставить транзакции в общую очередь по мере их поступления и выполнять строго последовательно. Таким способом автоматически будет получен правильный (сериальный) график. Проблема в том, что этот график будет неоптимальным с точки зрения общей производительности системы. Получается ситуация, в которой борются противоположные силы - с одной стороны, стремление обеспечить сериальность за счет ухудшения общей эффективности работы, с другой стороны, стремление улучшить общую эффективность за счет ухудшения сериальности.

Один крайний случай (выполнение транзакций по очереди) мы рассмотрели. Рассмотрим другой крайний случай - попытаемся достичь оптимального графика - т.е. графика с максимальной эффективностью выполнения транзакций. Для этого сначала нужно уточнить понятие "оптимальность". С каждым возможным графиком запуска транзакций мы можем связать значение некоей стоимостной функции. В качестве стоимостной функции 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. Это время складывается из следующих компонентов:

1. Время ожидания начала транзакции - то время, которое проходит от момента, когда транзакция возникла до момента, когда началась реально выполняться ее первая элементарная операция.
2. Сумма времен выполнения элементарных операций транзакции.
3. Сумма времен всех элементарных операций других транзакций, вклинившихся между элементарными операциями транзакции.

Оптимальным будет график, дающий минимум стоимостной функции. Очевидно, оптимальность графика запуска зависит от выбора стоимостной функции, т.е. график, оптимальный с точки зрения одних критериев (например, с точки зрения приведенной функции стоимости) не будет оптимальным с точки зрения других критериев (например, с точки зрения достижения максимально быстрого начала выполнения каждой транзакции).

Рассмотрим следующую гипотетическую ситуацию. Предположим, что нам *заранее* на некоторый промежуток времени наперед известно, какие транзакции в какие моменты поступят, т.е. *заранее известна вся будущая смесь транзакций и моменты поступления каждой транзакции*:

Транзакция http://citforum.ru/database/dblearn/image381.gif поступит в момент http://citforum.ru/database/dblearn/image382.gif.

Транзакция http://citforum.ru/database/dblearn/image383.gif поступит в момент http://citforum.ru/database/dblearn/image384.gif.

…

Транзакция http://citforum.ru/database/dblearn/image385.gif поступит в момент http://citforum.ru/database/dblearn/image386.gif.

В этом случае, т.к. набор всех транзакций заранее известен, теоретически можно перебрать все возможные варианты графиков запусков (их конечное число, хотя и очень большое), и выбрать из них те графики, которые, во-первых, правильные, а во-вторых, оптимальны по выбранному критерию. В этом случае оптимальный график запуска транзакций достижим.

В реальной ситуации, однако, неизвестно не только какие транзакции будут поступать в какие моменты времени, но и неизвестна длительность периода времени, охватывающего набор транзакций. Реально, система может непрерывно работать несколько дней или месяцев и в этом случае набором транзакций будет набор всех транзакций за этот период. С другой стороны, прекращение работы сервера может произойти в любой момент либо по команде администратора системы, либо в результате сбоя. Необходимо, следовательно, чтобы система работала 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.

#### 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).

Если транзакция A блокирует объект при помощи X-блокировки, то всякий доступ к этому объекту со стороны других транзакций отвергается.

Если транзакция A блокирует объект при помощи S-блокировки, то

* запросы со стороны других транзакций на X-блокировку этого объекта будут отвергнуты,
* запросы со стороны других транзакций на S-блокировку этого объекта будут приняты.

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)** |

**Таблица 1 Матрица совместимости S- и X-блокировок**

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.

#### Решение проблем параллелизма при помощи блокировок

Рассмотрим, как будут себя вести транзакции, вступающие в конфликт при доступе к данным, если они подчиняются протоколу доступа к данным.

#### 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** |
| S-блокировка http://citforum.ru/database/dblearn/image387.gif- успешна | http://citforum.ru/database/dblearn/image359.gif | --- |
| Чтение http://citforum.ru/database/dblearn/image355.gif | http://citforum.ru/database/dblearn/image360.gif | --- |
| --- | http://citforum.ru/database/dblearn/image361.gif | S-блокировка http://citforum.ru/database/dblearn/image387.gif- успешна |
| --- | http://citforum.ru/database/dblearn/image362.gif | Чтение http://citforum.ru/database/dblearn/image355.gif |
| X-блокировка http://citforum.ru/database/dblearn/image387.gif- отвергается | http://citforum.ru/database/dblearn/image364.gif | --- |
| Ожидание… | http://citforum.ru/database/dblearn/image365.gif | X-блокировка http://citforum.ru/database/dblearn/image387.gif- отвергается |
| Ожидание… | http://citforum.ru/database/dblearn/image388.gif | Ожидание… |
| **Ожидание…** |  | **Ожидание…** |

Обе транзакции успешно накладывают S-блокировки и читают объект http://citforum.ru/database/dblearn/image354.gif. Транзакция A пытается наложить X-блокирокировку для обновления объекта http://citforum.ru/database/dblearn/image354.gif. Блокировка отвергается, т.к. объект http://citforum.ru/database/dblearn/image354.gif 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 http://citforum.ru/database/dblearn/image354.gif. Блокировка отвергается, т.к. объект http://citforum.ru/database/dblearn/image354.gif уже S-заблокирован транзакцией A. Транзакция B переходит в состояние ожидания до тех пор, пока транзакция A не освободит объект.

Результат. Обе транзакции ожидают друг друга и не могут продолжаться. Возникла ситуация *тупика*.

#### 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** |
| --- | http://citforum.ru/database/dblearn/image359.gif | S-блокировка http://citforum.ru/database/dblearn/image387.gif- успешна |
| --- | http://citforum.ru/database/dblearn/image360.gif | Чтение http://citforum.ru/database/dblearn/image355.gif |
| --- | http://citforum.ru/database/dblearn/image361.gif | X-блокировка http://citforum.ru/database/dblearn/image387.gif- успешна |
| --- | http://citforum.ru/database/dblearn/image362.gif | Запись http://citforum.ru/database/dblearn/image358.gif |
| S-блокировка http://citforum.ru/database/dblearn/image387.gif- отвергается | http://citforum.ru/database/dblearn/image364.gif | --- |
| Ожидание… | http://citforum.ru/database/dblearn/image365.gif | Откат транзакции http://citforum.ru/database/dblearn/image368.gif  (Блокировка снимается) |
| S-блокировка http://citforum.ru/database/dblearn/image387.gif- успешна | http://citforum.ru/database/dblearn/image388.gif | --- |
| Чтение http://citforum.ru/database/dblearn/image389.gif | http://citforum.ru/database/dblearn/image390.gif | --- |
| Работа с прочитанными данными http://citforum.ru/database/dblearn/image391.gif | http://citforum.ru/database/dblearn/image392.gif | --- |
| --- | http://citforum.ru/database/dblearn/image393.gif | --- |
| Фиксация транзакции | http://citforum.ru/database/dblearn/image394.gif | --- |
| **Все правильно** |  |  |

Результат. Транзакция A притормозилась до окончания (отката) транзакции B. После этого транзакция A продолжила работу в обычном режиме и работала с правильными данными. Конфликт разрешен за счет некоторого увеличения времени работы транзакции A (потрачено время на ожидание снятия блокировки транзакцией B).

#### Проблема несовместимого анализа

#### 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** |
| S-блокировка http://citforum.ru/database/dblearn/image387.gif- успешна | http://citforum.ru/database/dblearn/image359.gif | --- |
| Чтение http://citforum.ru/database/dblearn/image355.gif | http://citforum.ru/database/dblearn/image360.gif | --- |
| --- | http://citforum.ru/database/dblearn/image361.gif | X-блокировка http://citforum.ru/database/dblearn/image387.gif- отвергается |
| --- | http://citforum.ru/database/dblearn/image362.gif | Ожидание… |
| Повторное чтение http://citforum.ru/database/dblearn/image389.gif | http://citforum.ru/database/dblearn/image364.gif | Ожидание… |
| Фиксация транзакции (Блокировка снимается) | http://citforum.ru/database/dblearn/image365.gif | Ожидание… |
| --- | http://citforum.ru/database/dblearn/image388.gif | X-блокировка http://citforum.ru/database/dblearn/image387.gif- успешна |
| --- | http://citforum.ru/database/dblearn/image390.gif | Запись http://citforum.ru/database/dblearn/image358.gif |
| --- | http://citforum.ru/database/dblearn/image392.gif | Фиксация транзакции  (Блокировка снимается) |
| **Все правильно** |  |  |

Результат. Транзакция B притормозилась до окончания транзакции A. В результате транзакция A дважды читает одни и те же данные правильно. После окончания транзакции A, транзакция B продолжила работу в обычном режиме.

#### 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** |
| S-блокировка строк, удовлетворяющих условию http://citforum.ru/database/dblearn/image369.gif.  (Заблокировано n строк) | http://citforum.ru/database/dblearn/image359.gif | --- |
| Выборка строк, удовлетворяющих условию http://citforum.ru/database/dblearn/image369.gif.  (Отобрано n строк) | http://citforum.ru/database/dblearn/image360.gif | --- |
| --- | http://citforum.ru/database/dblearn/image361.gif | Вставка новой строки, удовлетворяющей условию http://citforum.ru/database/dblearn/image369.gif. |
| --- | http://citforum.ru/database/dblearn/image362.gif | Фиксация транзакции |
| S-блокировка строк, удовлетворяющих условию http://citforum.ru/database/dblearn/image369.gif.  (Заблокировано n+1 строка) | http://citforum.ru/database/dblearn/image364.gif | --- |
| Выборка строк, удовлетворяющих условию http://citforum.ru/database/dblearn/image369.gif.  (Отобрано n+1 строк) | http://citforum.ru/database/dblearn/image365.gif | --- |
| Фиксация транзакции | http://citforum.ru/database/dblearn/image388.gif | --- |
| **Появились строки, которых раньше не было** |  |  |

Результат. Блокировка на уровне строк не решила проблему появления фиктивных элементов.

#### Собственно несовместимый анализ

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 A** | **Время** | **Транзакция B** |
| S-блокировка счета http://citforum.ru/database/dblearn/image395.gif- успешна | http://citforum.ru/database/dblearn/image359.gif | --- |
| Чтение счета http://citforum.ru/database/dblearn/image370.gif и суммирование.  http://citforum.ru/database/dblearn/image371.gif | http://citforum.ru/database/dblearn/image360.gif | --- |
| --- | http://citforum.ru/database/dblearn/image361.gif | X-блокировка счета http://citforum.ru/database/dblearn/image396.gif- успешна |
| --- | http://citforum.ru/database/dblearn/image362.gif | Снятие денег со счета http://citforum.ru/database/dblearn/image372.gif.  http://citforum.ru/database/dblearn/image373.gif |
| --- | http://citforum.ru/database/dblearn/image364.gif | X-блокировка счета http://citforum.ru/database/dblearn/image397.gif- отвергается |
| --- | http://citforum.ru/database/dblearn/image365.gif | Ожидание… |
| S-блокировка счета http://citforum.ru/database/dblearn/image398.gif- успешна | http://citforum.ru/database/dblearn/image388.gif | Ожидание… |
| Чтение счета http://citforum.ru/database/dblearn/image376.gif и суммирование.  http://citforum.ru/database/dblearn/image377.gif | http://citforum.ru/database/dblearn/image390.gif | Ожидание… |
| S-блокировка счета http://citforum.ru/database/dblearn/image399.gif- отвергается | http://citforum.ru/database/dblearn/image392.gif | Ожидание… |
| Ожидание… | http://citforum.ru/database/dblearn/image393.gif | Ожидание… |
| **Ожидание…** |  | **Ожидание…** |

Результат. Обе транзакции ожидают друг друга и не могут продолжаться. Возникла ситуация *тупика*.

#### Разрешение тупиковых ситуаций

Итак, при использовании протокола доступа к данным с использованием блокировок часть проблем разрешилось (не все), но возникла новая проблема - тупики:

* Проблема потери результатов обновления - *возник тупик*.
* Проблема незафиксированной зависимости (чтение "грязных" данных, неаккуратное считывание) - *проблема разрешилась*.
* Неповторяемое считывание - *проблема разрешилась*.
* Появление фиктивных элементов - *проблема не разрешилась*.
* Проблема несовместимого анализа - *возник тупик*.

Общий вид ***тупика*** (***dead locks***) следующий:

|  |  |  |
| --- | --- | --- |
| **Транзакция A** | **Время** | **Транзакция B** |
| Блокировка объекта http://citforum.ru/database/dblearn/image395.gif- успешна | http://citforum.ru/database/dblearn/image359.gif | --- |
| --- | http://citforum.ru/database/dblearn/image360.gif | Блокировка объекта http://citforum.ru/database/dblearn/image398.gif-успешна |
| Блокировка объекта http://citforum.ru/database/dblearn/image398.gif- конфликтует с блокировкой, наложенной транзакцией B | http://citforum.ru/database/dblearn/image361.gif | --- |
| Ожидание… | http://citforum.ru/database/dblearn/image362.gif | Блокировка объекта http://citforum.ru/database/dblearn/image395.gif- конфликтует с блокировкой, наложенной транзакцией A |
| Ожидание… | http://citforum.ru/database/dblearn/image364.gif | Ожидание… |
| **Ожидание…** |  | **Ожидание…** |

Как видно, ситуация тупика может возникать при наличии не менее двух транзакций, каждая из которых выполняет не менее двух операций. На самом деле в тупике может участвовать много транзакций, ожидающих друг друга.

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.

Можно представить два принципиальных подхода к обнаружению тупиковой ситуации и выбору транзакции-жертвы:

1. СУБД не следит за возникновением тупиков. Транзакции сами принимают решение, быть ли им жертвой.
2. За возникновением тупиковой ситуации следит сама СУБД, она же принимает решение, какой транзакцией пожертвовать.

Первый подход характерен для так называемых настольных СУБД (FoxPro и т.п.). Этот метод является более простым и не требует дополнительных ресурсов системы. Для транзакций задается время ожидания (или число попыток), в течение которого транзакция пытается установить нужную блокировку. Если за указанное время (или после указанного числа попыток) блокировка не завершается успешно, то транзакция откатывается (или генерируется ошибочная ситуация). За простоту этого метода приходится платить тем, что транзакции-жертвы выбираются, вообще говоря, случайным образом. В результате из-за одной простой транзакции может откатиться очень дорогая транзакция, на выполнение которой уже потрачено много времени и ресурсов системы.

Второй способ характерен для промышленных СУБД (ORACLE, MS SQL Server и т.п.). В этом случае система сама следит за возникновением ситуации тупика, путем построения (или постоянного поддержания) графа ожидания транзакций. ***Граф ожидания транзакций*** - это ориентированный двудольный граф, 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.).

#### Преднамеренные блокировки

Как видно из анализа поведения транзакций, при использовании протокола доступа к данным не решается проблема фантомов. Это происходит оттого, что были рассмотрены только блокировки на уровне строк. Можно рассматривать блокировки и других объектов базы данных:

* Блокировка самой базы данных.
* Блокировка файлов базы данных.
* Блокировка таблиц базы данных.
* Блокировка страниц (Единиц обмена с диском, обычно 2-16 Кб. На одной странице содержится несколько строк одной или нескольких таблиц).
* Блокировка отдельных строк таблиц.
* Блокировка отдельных полей.

Кроме того, можно блокировать индексы, заголовки таблиц или другие объекты.

Чем крупнее объект блокировки, тем меньше возможностей для параллельной работы. Достоинством блокировок крупных объектов является уменьшение накладных расходов системы и решение проблем, не решаемых с использованием блокировок менее крупных объектов. Например, использование монопольной блокировки на уровне таблицы, очевидно, решает проблему фантомов.

Современные СУБД, как правило, поддерживают минимальный уровень блокировки на уровне строк или страниц. (В старых версиях настольной СУБД Paradox поддерживалась блокировка на уровне отдельных полей.).

При использовании блокировок объектов разной величины возникает проблема обнаружения уже наложенных блокировок. Если транзакция A пытается заблокировать таблицу, то необходимо иметь информацию, не наложены ли уже блокировки на уровне строк этой таблицы, несовместимые с блокировкой таблицы. Для решения этой проблемы используется ***протокол преднамеренных блокировок***, являющийся расширением протокола доступа к данным. Суть этого протокола в том, что перед тем, как наложить блокировку на объект (например, на строку таблицы), необходимо наложить специальную ***преднамеренную блокировку*** (***блокировку намерения***) на объекты, в состав которых входит блокируемый объект - на таблицу, содержащую строку, на файл, содержащий таблицу, на базу данных, содержащую файл. Тогда наличие преднамеренной блокировки таблицы будет свидетельствовать о наличии блокировки строк таблицы и для другой транзакции, пытающейся блокировать целую таблицу не нужно проверять наличие блокировок отдельных строк. Более точно, вводятся следующие новые типы блокировок:

* ***Преднамеренная блокировка с возможностью взаимного доступа*** (***IS-блокировка*** - Intent Shared lock). Накладывается на некоторый составной объект T и означает намерение блокировать некоторый входящий в T объект в режиме S-блокировки. Например, при намерении читать строки из таблицы T, эта таблица должна быть заблокирована в режиме IS (до этого в таком же режиме должен быть заблокирован файл).
* ***Преднамеренная блокировка без взаимного доступа*** (***IX-блокировка*** - Intent eXclusive lock). Накладывается на некоторый составной объект T и означает намерение блокировать некоторый входящий в T объект в режиме X-блокировки. Например, при намерении удалять или модифицировать строки из таблицы T эта таблица должна быть заблокирована в режиме IX (до этого в таком же режиме должен быть заблокирован файл).
* ***Преднамеренная блокировка как с возможностью взаимного доступа, так и без него*** (***SIX-блокировка*** - Shared Intent eXclusive lock). Накладывается на некоторый составной объект T и означает разделяемую блокировку всего этого объекта с намерением впоследствии блокировать какие-либо входящие в него объекты в режиме X-блокировок. Например, если выполняется длинная операция просмотра таблицы с возможностью удаления некоторых просматриваемых строк, то можно заблокировать эту таблицу в режиме SIX (до этого захватить файл в режиме IS).

IS, IX и SIX-блокировки должны накладываться на сложные объекты базы данных (таблицы, файлы). Кроме того, на сложные объекты могут накладываться и блокировки типов S и X. Для сложных объектов (например, для таблицы базы данных) таблица совместимости блокировок имеет следующий вид:

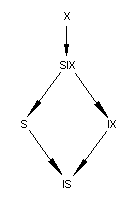
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Транзакция B пытается наложить на таблицу блокировку: | | | | |
| Транзакция A наложила на таблицу блокировку: | IS | S | IX | SIX | X |
| IS | **Да** | **Да** | **Да** | **Да** | **Нет** |
| S | **Да** | **Да** | **Нет** | **Нет** | **Нет** |
| IX | **Да** | **Нет** | **Да** | **Нет** | **Нет** |
| SIX | **Да** | **Нет** | **Нет** | **Нет** | **Нет** |
| X | **Нет** | **Нет** | **Нет** | **Нет** | **Нет** |

**Таблица 2 Расширенная таблица совместимости блокировок**

Более точная формулировка протокола преднамеренных блокировок для доступа к данным выглядит следующим образом:

1. При задании X-блокировки для сложного объекта *неявным* образом задается X-блокировка для всех дочерних объектов этого объекта.
2. При задании S- или SIX-блокировки для сложного объекта *неявным* образом задается S-блокировка для всех дочерних объектов этого объекта.
3. Прежде чем транзакция наложит S- или IS-блокировку на заданный объект, она должна задать IS-блокировку (или более сильную) по крайней мере для одного родительского объекта этого объекта.
4. Прежде чем транзакция наложит X-, IX- или SIX-блокировку на заданный объект, она должна задать IX-блокировку (или более сильную) для всех родительских объектов этого объекта.
5. Прежде чем для данной транзакции будет отменена блокировка для данного объекта, должны быть отменены все блокировки для дочерних объектов этого объекта.

Понятие относительной силы блокировок можно описать при помощи следующей диаграммы приоритета (сверху - более сильные блокировки, снизу - более слабые):



**Таблица 3 Диаграмма приоритета блокировок**

Замечание. Протокол преднамеренных блокировок не определяет однозначно, какие блокировки должны быть наложены на родительский объект при блокировании дочернего объекта. Например, при намерении задать S-блокировку строки таблицы, на таблицу, включающую эту строку, можно наложить любую из блокировок типа IS, S, IX, SIX, X. При намерении задать X-блокировку строки, на таблицу можно наложить любую из блокировок типа IX, SIX, X.

Посмотрим, как разрешается проблема фиктивных элементов (фантомов) с использованием протокола преднамеренных блокировок для доступа к данным.

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 B перед попыткой вставить новую строку должна наложить на таблицу IX-блокировку, или более сильную (SIX или X). Тогда транзакция A, для предотвращения возможного конфликта, должна наложить такую блокировку на таблицу, которая не позволила бы транзакции B наложить IX-блокировку. По таблице совместимости блокировок определяем, что транзакция A должна наложить на таблицу S, или SIX, или X-блокировку. (Блокировки IS недостаточно, т.к. эта блокировка *позволяет* транзакции B наложить IX-блокировку для последующей вставки строк).

|  |  |  |
| --- | --- | --- |
| **Транзакция A** | **Время** | **Транзакция B** |
| S-блокировка таблицы (с целью потом блокировать строки) - успешна | http://citforum.ru/database/dblearn/image359.gif | --- |
| S-блокировка строк, удовлетворяющих условию http://citforum.ru/database/dblearn/image369.gif.  (Заблокировано n строк) | http://citforum.ru/database/dblearn/image360.gif | --- |
| Выборка строк, удовлетворяющих условию http://citforum.ru/database/dblearn/image369.gif.  (Отобрано n строк) | http://citforum.ru/database/dblearn/image361.gif | --- |
| --- | http://citforum.ru/database/dblearn/image362.gif | IX-блокировка таблицы (с целью потом вставлять строки) - отвергается из-за конфликта с S-блокировкой, наложенной транзакцией A |
| --- | http://citforum.ru/database/dblearn/image364.gif | Ожидание… |
| --- | http://citforum.ru/database/dblearn/image365.gif | Ожидание… |
| S-блокировка строк, удовлетворяющих условию http://citforum.ru/database/dblearn/image369.gif.  (Заблокировано n строк) | http://citforum.ru/database/dblearn/image388.gif | Ожидание… |
| Выборка строк, удовлетворяющих условию http://citforum.ru/database/dblearn/image369.gif.  (Отобрано n строк) | http://citforum.ru/database/dblearn/image390.gif | Ожидание… |
| Фиксация транзакции - блокировки снимаются | http://citforum.ru/database/dblearn/image392.gif | Ожидание… |
| --- | http://citforum.ru/database/dblearn/image393.gif | IX-блокировка таблицы (с целью потом вставлять строки) - успешна |
| --- | http://citforum.ru/database/dblearn/image394.gif | Вставка новой строки, удовлетворяющей условию http://citforum.ru/database/dblearn/image369.gif. |
| --- | http://citforum.ru/database/dblearn/image401.gif | Фиксация транзакции |
| **Транзакция A дважды читает один и тот же набор строк  Все правильно** |  |  |

Результат. Проблема фиктивных элементов (фантомов) решается, если транзакция A использует преднамеренную S-блокировку или более сильную.

Замечание. Т.к. транзакция A собирается только *читать* строки таблицы, то *минимально необходимым*условием в соответствии с протоколом преднамеренных блокировок является преднамеренная IS-блокировка таблицы. Однако этот тип блокировки не предотвращает появление фантомов. Таким образом, транзакцию A можно запускать с *разными уровнями изолированности* - предотвращая или допуская появление фантомов. Причем, оба способа запуска *соответствуют* протоколу преднамеренных блокировок для доступа к данным.

#### Предикатные блокировки

Другим способом блокирования является блокировка не объектов базы данных, а условий, которым могут удовлетворять объекты. Такие блокировки называются ***предикатными блокировками***.

Поскольку любая операция над реляционной базой данных задается некоторым условием (т.е. в ней указывается не конкретный набор объектов базы данных, над которыми нужно выполнить операцию, а условие, которому должны удовлетворять объекты этого набора), то удобным способом было бы S или X-блокирование именно этого условия. Однако при попытке использовать этот метод в реальной СУБД возникает трудность определения совместимости различных условий. Действительно, в языке SQL допускаются условия с подзапросами и другими сложными предикатами. Проблема совместимости сравнительно легко решается для случая простых условий, имеющих вид:

{*Имя атрибута* {= | <> | > | >= | < | <=} *Значение*}

[{**OR** | **AND**} {*Имя атрибута* {= | <> | > | >= | < | <=} *Значение*}.,..]

Проблема фиктивных элементов (фантомов) легко решается с использованием предикатных блокировок, т.к. вторая транзакция не может вставить новые строки, удовлетворяющие уже заблокированному условию.

Заметим, что блокировка всей таблицы в каком-либо режиме фактически есть предикатная блокировка, т.к. каждая таблица имеет предикат, определяющий какие строки содержатся в таблице и блокировка таблицы есть блокировка предиката этой таблицы.

#### Метод временных меток

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций и не требующий построения графа ожидания транзакций основан на использовании ***временных меток***.

Основная идея метода состоит в следующем: если транзакция A началась раньше транзакции B, то система обеспечивает такой режим выполнения, как если бы A была целиком выполнена до начала B.

Для этого каждой транзакции T предписывается временная метка t, соответствующая времени начала T. При выполнении операции над объектом r базы данных транзакция T помечает его своей временной меткой и типом операции (чтение или изменение).

Перед выполнением операции над объектом r транзакция B выполняет следующие действия:

* Проверяет, не закончилась ли транзакция A, пометившая этот объект. Если A закончилась, B помечает объект r своей временной меткой и выполняет операцию.
* Если транзакция A не завершилась, то B проверяет конфликтность операций. Если операции неконфликтны, при объекте r остается или проставляется временная метка с меньшим значением (более ранняя), и транзакция B выполняет свою операцию.
* Если операции B и A конфликтуют, то если t(A) > t(B) (т.е. транзакция A является более "молодой", чем B), то транзакция A откатывается и, получив новую временную метку, начинается заново. Транзакция B продолжает работу.
* Если же t(A) < t(B) (A "старше" B), то транзакция B откатывается и, получив новую временную метку, начинается заново. Транзакция A продолжает работу.

В итоге система обеспечивает такую работу, при которой при возникновении конфликтов всегда откатывается более молодая транзакция (начавшаяся позже).

Очевидным недостатком метода временных меток является то, что может откатиться более дорогая транзакция, начавшаяся позже более дешевой.

К другим недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования блокировок. Это связано с тем, что конфликтность транзакций определяется более грубо.

#### 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 ***механизм 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 A читает страницу данных, то SCN транзакции A сравнивается с SCN читаемой страницы данных.
* Если SCN страницы данных меньше или равен SCN транзакции A, то транзакция A читает эту страницу.
* Если SCN страницы данных больше SCN транзакции A, то это означает, что некоторая транзакция B, начавшаяся позже транзакции A, успела изменить или сейчас изменяет данные страницы. В этом случае транзакция A просматривает журнал транзакция назад в поиске первой записи об изменении нужной страницы данных с SCN меньшим, чем SCN транзакции A. Найдя такую запись, транзакция A 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 A** | **Время** | **Транзакция B** |
| Проверка SCN счета http://citforum.ru/database/dblearn/image395.gif - SCN транзакции больше SCN счета.  Чтение счета http://citforum.ru/database/dblearn/image370.gif без наложения блокировки и суммирование.  http://citforum.ru/database/dblearn/image371.gif | http://citforum.ru/database/dblearn/image359.gif | --- |
| --- | http://citforum.ru/database/dblearn/image360.gif | X-блокировка счета http://citforum.ru/database/dblearn/image396.gif- успешна |
| --- | http://citforum.ru/database/dblearn/image361.gif | Снятие денег со счета http://citforum.ru/database/dblearn/image372.gif.  http://citforum.ru/database/dblearn/image373.gif |
| --- | http://citforum.ru/database/dblearn/image362.gif | X-блокировка счета http://citforum.ru/database/dblearn/image402.gif- успешна |
| --- | http://citforum.ru/database/dblearn/image364.gif | Помещение денег на счет http://citforum.ru/database/dblearn/image374.gif.  http://citforum.ru/database/dblearn/image375.gif |
| --- | http://citforum.ru/database/dblearn/image365.gif | Фиксация транзакции  (Снятие блокировок) |
| Проверка SCN счета http://citforum.ru/database/dblearn/image403.gif - SCN транзакции больше SCN счета.  Чтение счета http://citforum.ru/database/dblearn/image376.gif без наложения блокировка и суммирование.  http://citforum.ru/database/dblearn/image377.gif | http://citforum.ru/database/dblearn/image388.gif | --- |
| Проверка SCN счета http://citforum.ru/database/dblearn/image404.gif - SCN транзакции **МЕНЬШЕ** SCN счета.  Чтение старого варианта счета http://citforum.ru/database/dblearn/image405.gif и суммирование.  http://citforum.ru/database/dblearn/image406.gif | http://citforum.ru/database/dblearn/image390.gif | --- |
| Фиксация транзакции | http://citforum.ru/database/dblearn/image392.gif | --- |
| **Сумма на счетах посчитана правильно.** |  |  |

Результат. 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.

#### Теорема Есварана о сериализуемости

Концепция способности к упорядочению была впервые предложена Есвараном [50].

В этой работе был предложен ***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.

Транзакции, используемые в этом протоколе, не различаются по типам и считаются монопольными. Описанные выше протоколы доступа к данным с использованием S- и X-блокировок и протокол преднамеренных блокировок являются модификациями протокола двухфазной блокировки для случая, когда блокировки имеют различные типы.

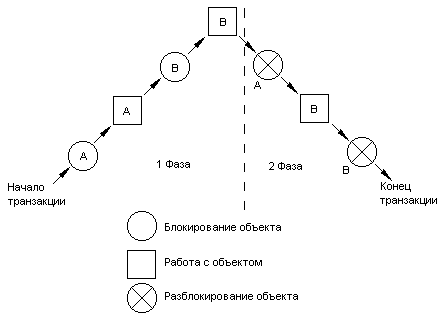
Есвараном сформулирована следующая теорема:

***Теорема Есварана***. Если все транзакции в смеси подчиняются протоколу двухфазной блокировки, то для всех чередующихся графиков запуска существует возможность упорядочения.

Протокол называется двухфазным, потому что он характеризуется двумя фазами:

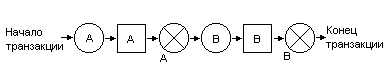
* ***1 фаза - нарастание блокировок***. Во время этой фазы накладываются блокировки, и производится работа с заблокированными объектами.
* ***2 фаза - снятие блокировок***. Во время этой фазы блокировки только снимаются. Работа с ранее заблокированными данными может продолжаться.

Работа транзакции может выглядеть приблизительно, как на рисунке:



**Рисунок 1 Работа транзакции по протоколу двухфазной блокировки**

На следующем рисунке показан пример транзакции, не подчиняющийся протоколу двухфазной блокировки:



**Рисунок 2 Транзакция, не подчиняющаяся протоколу двухфазной блокировки**

На практике, как правило, вторая фаза сводится к одной операции завершения транзакции (или отката транзакции) с одновременным снятием всех блокировок.

Если некоторая транзакция A не подчиняется протоколу двухфазной блокировки (и, следовательно, состоит не менее чем из двух операция блокирования и разблокирования), то всегда можно построить другую транзакцию B, которая при чередующемся выполнении вместе с A приводит к графику, не подлежащему упорядочению и неверному.

#### Реализация изолированности транзакций средствами SQL

#### Уровни изоляции

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 имеется множество способов нарушить способность к упорядочению. В стандарте SQL выделены три особых случая нарушения способности к упорядочению, фактически именно те, которые были описаны выше как проблемы параллелизма:

* ***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 или исключению этих особых случаев нарушения способности к упорядочению. Эти определения описываются следующей таблицей:

|  |  |  |  |
| --- | --- | --- | --- |
| **Уровень изоляции** | **Неаккуратное считывание** | **Неповторяемое считывание** | **Фантомы** |
| **READ UNCOMMITTED** | Да | Да | Да |
| **READ COMMITTED** | **Нет** | Да | Да |
| **REPEATABLE READ** | **Нет** | **Нет** | Да |
| **SERIALIZABLE** | **Нет** | **Нет** | **Нет** |

**Таблица 4 Уровни изоляции стандарта SQL**

#### Синтаксис операторов SQL, определяющих уровни изоляции

Уровень изоляции транзакции 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. Обычно, выполнение оператора SET TRANSACTION выделяется как отдельная транзакция:

… (предыдущая транзакция выполняется со своим уровнем изоляции)

COMMIT;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

COMMIT;

… (следующая транзакция выполняется с уровнем изоляции REPEATABLE READ)

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.

#### Выводы

Современные многопользовательские системы допускают одновременную работу большого числа пользователей. При этом если не предпринимать специальных мер, транзакции будут мешать друг другу. Этот эффект известен как проблемы параллелизма.

Имеются три основные проблемы параллелизма:

* ***Проблема потери результатов обновления***.
* ***Проблема незафиксированной зависимости*** (***чтение "грязных" данных***, ***неаккуратное считывание***).
* ***Проблема несовместимого анализа***.

График запуска набора транзакций называется ***последовательным***, если транзакции выполняются строго по очереди. Если график запуска набора транзакций содержит чередующиеся элементарные операции транзакций, то такой график называется ***чередующимся***. Два графика называются ***эквивалентными***, если при их выполнении будет получен один и тот же результат, *независимо от начального состояния базы данных*. График запуска транзакции называется ***верным*** (***сериализуемым***), если он эквивалентен какому-либо последовательному графику.

Решение проблем параллелизма состоит в нахождении такой стратегии запуска транзакций, чтобы обеспечить сериализуемость графика запуска и не слишком уменьшить степень параллельности.

Одним из методов обеспечения сериальности графика запуска является протокол доступа к данным при помощи блокировок. В простейшем случае различают ***S-блокировки*** (разделяемые) и ***X-блокировки***(монопольные). Протокол доступа к данным имеет вид:

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.

Если все транзакции в смеси подчиняются протоколу доступа к данным, то проблемы параллелизма решаются (почти все, кроме "фантомов"), но появляются тупики. Состояние ***тупика*** (***dead locks***) характеризуется тем, что две или более транзакции пытаются заблокировать одни и те же объекты, и бесконечно долго ожидают друг друга.

Для разрушения тупиков система периодически или постоянно поддерживает графа ожидания транзакций. Наличие циклов в графе ожидания свидетельствует о возникновении тупика. Для разрушения тупика одна из транзакций (наиболее дешевая с точки зрения системы) выбирается в качестве жертвы и откатывается.

Для решения проблемы "фантомов", а также для уменьшения накладных расходов, вызываемых большим количеством блокировок, применяются более сложные методы. Одним из таких методов являются ***преднамеренные блокировки***, блокирующие объекты разной величины - строки, страницы, таблицы, файлы и др.

Альтернативным является метод ***предикатных блокировок***

Имеются также методы сериализации, не использующие блокировки. Это ***метод временных меток*** и ***механизм выделения версий***. Механизм выделения версий удобен тем, что он позволяет одновременно, и читать, и изменять одни и те же данные разными транзакциями. Это увеличивает степень параллельности, и общую производительность системы.

Стандарт SQL *не предусматривает* понятие блокировок. Вместо этого вводится понятие ***уровней изоляции***. Стандарт предусматривает 4 уровня изоляции:

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

Транзакции могут запускаться с различными уровнями изоляции.

**19. Уровни изоляции транзакций. Влияние транзакций друг на друга на различных уровнях изоляции**

В  стандарте SQL описывается четыре уровня изоляции транзакций — Read uncommited (Чтение незафиксированных данных), Read committed (Чтение зафиксированных данных), Repeatable read (Повторяемое чтение) и Serializable (Сериализуемость). В данной статье будет рассмотрен жизненный цикл четырёх параллельно выполняющихся транзакций с уровнями изоляции **Read committed** и **Serializable**.

Для уровня изоляции Read committed допустимы следующие особые условия чтения данных:

Неповторяемое чтение — транзакция повторно читает те же данные, что и раньше, и обнаруживает, что они были изменены другой транзакцией (которая завершилась после первого чтения).

Фантомное чтение — транзакция повторно выполняет запрос, возвращающий набор строк для некоторого условия, и обнаруживает, что набор строк, удовлетворяющих условию, изменился из-за транзакции, завершившейся за это время.

Что же касается Serializable, то данный уровень изоляции самый строгий, и не имеет феноменов чтения данных.

## ACID или 4 свойства транзакций

Прежде чем приступим к рассмотрению уровней изоляции транзакции в паре слов вспомним об основных требованиях к транзакционной системе.

**Atomicity** (атомарность) — выражается в том, что транзакция должна быть выполнена в целом или не выполнена вовсе.

**Consistency** (согласованность) — гарантирует, что по мере выполнения транзакций, данные переходят из одного согласованного состояния в другое, то есть транзакция не может разрушить взаимной согласованности данных.

**Isolation** (изолированность) — локализация пользовательских процессов означает, что конкурирующие за доступ к БД транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит, как будто они выполняются параллельно.

**Durability** (долговечность) — устойчивость к ошибкам — если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах.

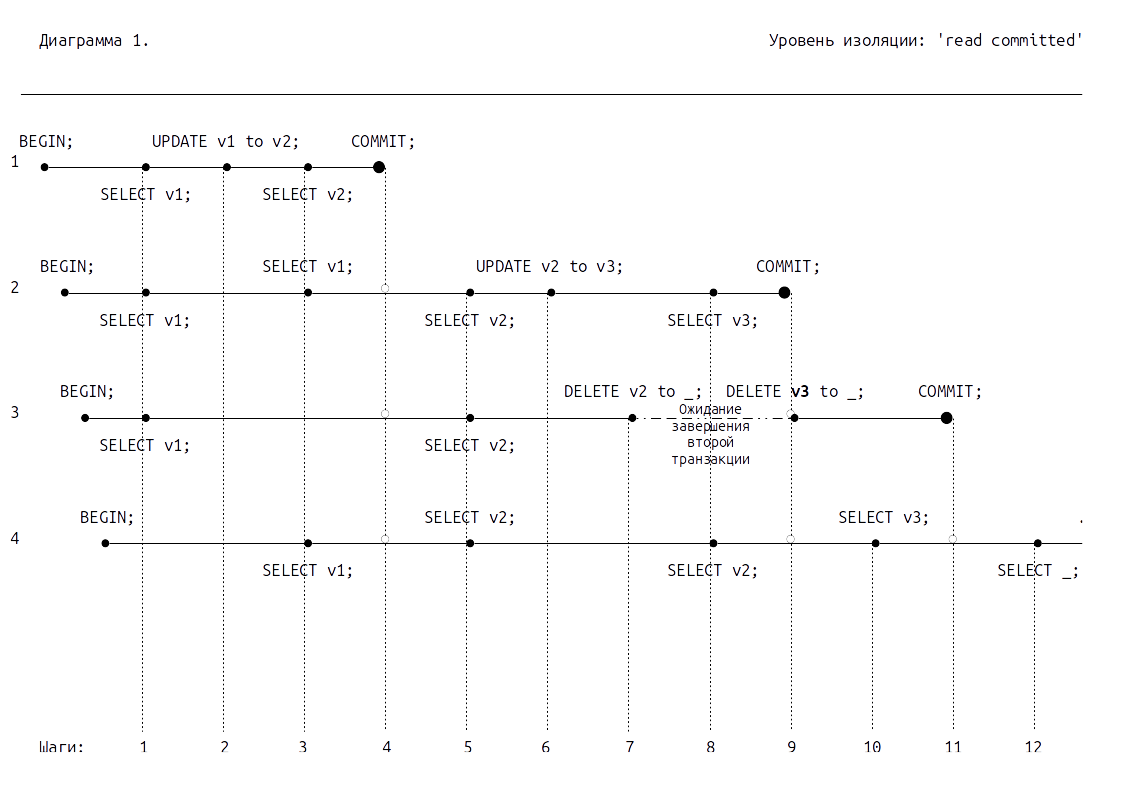
## Уровень изоляции Read Committed

По умолчанию в PostgreSQL уровень изоляции Read Committed. Такой уровень изоляции всегда позволяет видеть изменения внесённые успешно завершёнными транзакциями в оставшихся параллельно открытых транзакциях. В транзакции, работающей на этом уровне, запрос SELECT (без предложения FOR UPDATE/SHARE) видит только те данные, которые были зафиксированы до начала запроса; он никогда не увидит незафиксированных данных или изменений, внесённых в процессе выполнения запроса параллельными транзакциями. По сути запрос SELECT видит снимок базы данных в момент начала выполнения запроса. Однако SELECT видит результаты изменений, внесённых ранее в этой же транзакции, даже если они ещё не зафиксированы. Также заметьте, что два последовательных оператора SELECT могут видеть разные данные даже в рамках одной транзакции, если какие-то другие транзакции зафиксируют изменения после выполнения первого SELECT.

Суть уровня изоляции Read Committed показана на диаграмме 1.

Примечание: В таблице уже находится запись с первой версией данных (v1). Прошу воспринимать команды SELECT v1; — как команду возвращающую данные версии v1, а UPDATE v1 to v2; — как команду обновления данных с первой версии до второй.

Создадим к базе-данных 4 подключения и откроем в каждом из подключений по транзакции с уровнем изоляции Read Committed



Шаг 1. В начальный момент времени до каких-либо изменений данных всем транзакциям доступна изначальная версия данных (v1);

Шаг 2. В ходе работы Первой транзакции данные без каких либо блокировок успешно обновляются до «второй версии» (v2);

Шаг 3. Изменения сделанные в Первой транзакции будут видны только ей самой (SELECT возвращает v2), и не будут доступны остальным транзакциям (SELECT запрос во Второй и Четвёртой транзакциях возвращает v1);

Шаг 4. Закрытие Первой транзакции. Все изменения сделанные в ходе её работы успешно фиксируются;

Шаг 5. После закрытия Первой транзакции (предыдущий шаг), изменения сделанные в ходе её выполнения над данными (обновление с v1 до v2) были распространены на остальные транзакции, SELECT запрос в оставшихся 3 открытых транзакциях возвращает v2 («Неповторяемое чтение», отличите уровня изоляции Read Committed от Serializable);

Шаг 6. Запрос на обновление данных во Второй транзакции до «третьей версии» успешно выполняется, но запросы на обновление данных блокируют изменяемые строки на дальнейшее их изменение, до завершения Второй транзакции;

Шаг 7. Из-за блокировки наложенной на данные в предыдущем шаге, Третья транзакция переходит в режим ожидания с запросом на удаление данных. Ожидание Третьей транзакции будет происходить до закрытия Второй транзакции;

Шаг 8. Несмотря на то, что Третья транзакция ожидает закрытия Второй, как Вторая так и Четвёртая транзакции без каких либо проблем продолжают свою работу, возвращая данные согласно своим версиям. Вторая возвращает v3, Четвёртая возвращает v2;

Шаг 9. Закрытие Второй транзакции приводит к разблокированию данных для изменения. Уровень изоляции Read Committed позволяет продолжить работу Третьей транзакции без вызова ошибки. Получив доступ на изменение новой версии данных (v3) Третья транзакция УСПЕШНО тут же их «удаляет» (отличие Read Committed от Serializable);

Шаг 10. До закрытия Третьей транзакции, данные будут удалёнными только внутри Третьей транзакции. Четвёртой транзакции до закрытия Третьей данные доступны (SELECT запрос в Четвёртой транзакции возвращает v3);

Шаг 11. Закрытие Третьей транзакции. Все изменения сделанные в ходе её работы успешно фиксируются;

Шаг 12. Запрос на получение данных в Четвёртой транзакции ничего не возвращает («Фантомное чтение», SELECT запрос возвращает 0 записей).

Примечание. На диаграмме не показано действие запроса INSERT. В рамках данного уровня изоляции, строки добавленные, например в шаге 3, в Первой транзакции, были бы ВИДНЫ остальным транзакциям после завершения Первой транзакции.

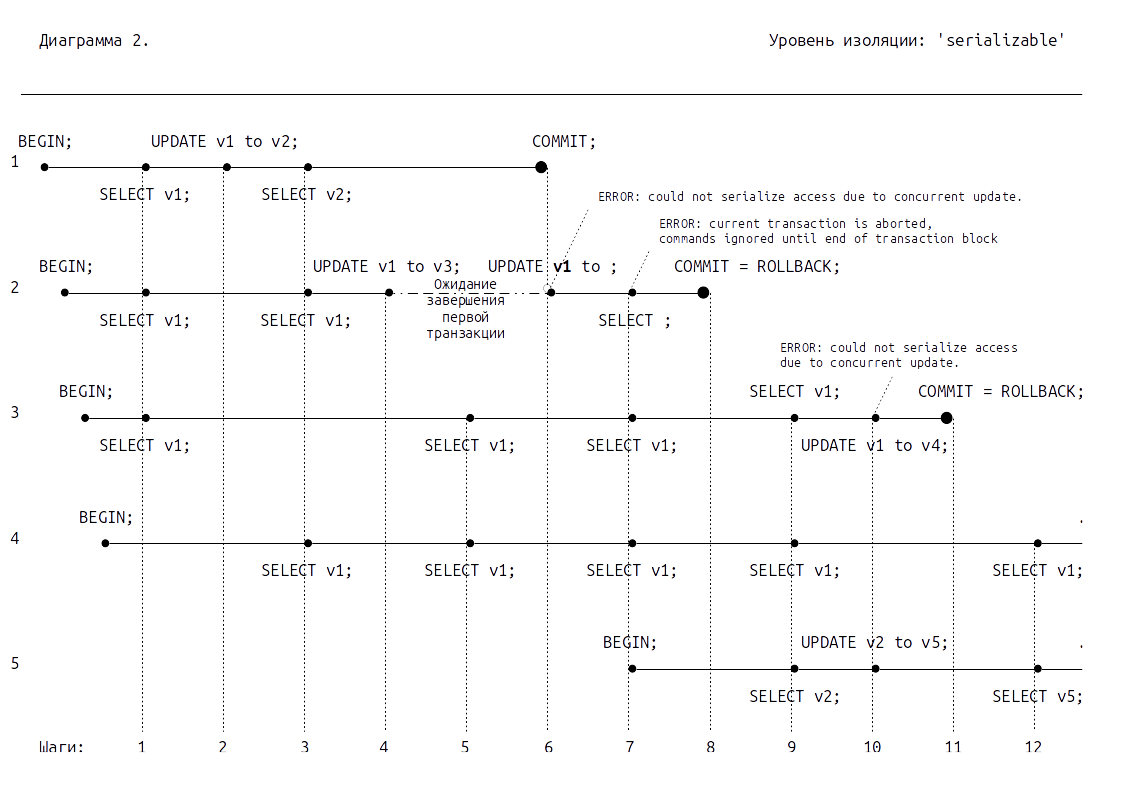
Частичная изоляция транзакций, обеспечиваемая в режиме Read Committed, приемлема для множества приложений. Этот режим быстр и прост в использовании, однако он подходит не для всех случаев. Приложениям, выполняющим сложные запросы и изменения, могут потребоваться более строго согласованное представление данных, например Serializable.

## Уровень изоляции Serializable

Изоляция уровня Serializable 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). Если две одновременные транзакции попытаются обновить одну и туже строку, то это будет не возможно. В таком случае PostgreSQL принудит транзакцию, вторую, да и все последующие, что пытались изменить строку к отмене (откату — ROLLBACK).

Суть уровня изоляции Serializable показана на диаграмме 2.

Создадим к базе-данных 4 подключения и откроем в каждом из подключений по транзакции с уровнем изоляции Serializable



Шаг 1. Всем транзакциям доступна изначальная версия данных (v1);

Шаг 2. В ходе работы Первой транзакции данные без каких либо блокировок успешно обновляются до «второй версии» (v2);

Шаг 3. Изменения сделанные в Первой транзакции будут видны только ей самой (SELECT возвращает v2), и не будут доступны остальным транзакциям (SELECT запрос во Второй и Четвёртой транзакциях возвращает v1);

Шаг 4. Запрос на обновление данных в первой транзакции (шаг 2), блокирует обновляемые строки, и переводит в режим ожидания Вторую транзакцию с запросом на удаление данных. Блокировка транзакций на обновляемые данных будет происходить до закрытия Первой транзакции;

Шаг 5. Несмотря на то, что Вторая транзакция ожидает закрытия Первой, как Третья так и Четвёртая транзакции без каких либо проблем продолжают свою работу, возвращая данные согласно своим версиям;

Шаг 6. Завершение Первой транзакции снимает блокировку с обновляемых данных, но в рамках уровня изоляции Serializable повторное обновление данных в параллельных транзакциях запрещено, и поэтому в ходе выполнения Второй транзакции возникает ошибка (отличие Serializable от Read Committed);

Шаг 7. Запрос SELECT во Второй транзакции становится не возможным, так как ошибка возникшая на предыдущем шаге отменяет («блокирует») транзакцию. Запрос SELECT в Третьей и Четвертой транзакциях возвращают первоначальную версию данных (v1). Несмотря на то, что Первая транзакция была завершена успешно, изменения не стали видны остальным открытым транзакциям (отличие Serializable от Read Committed). Открытие Пятой транзакции в левом верхнем окне;

Шаг 8. Закрытие Второй транзакции. Все изменения сделанные данной транзакцией будут отменены, из-за возникшей ошибки в ходе её работы;

Шаг 9. Запрос SELECT в Пятой транзакции возвращает новую версию данных (v2). Запрос SELECT в Третьей и Четвёртой транзакциях возвращают первоначальную версию данных (v1);

Шаг 10. Уровень изоляции Serializable всё также не даёт обновлять данные, запрос UPDATE в Третьей транзакции завершается не удачно, с вытекающими последствиями для хода всей транзакции (несмотря на то, что Первая транзакция уже удачно завершилась, и все внесённые ей изменения сохранены в базе данных). А вот запрос UPDATE в Пятой транзакции завершается успешно, так как она открыта после завершения Первой транзакции, и работает с новой версией данных;

Шаг 11. Закрытие Третьей транзакции. Все изменения сделанные данной транзакцией будут отменены, из-за возникшей ошибки в ходе её работы;

Шаг 12. Транзакция Четыре всё также показывает, что у транзакций с SELECT запросами никаких нет проблем, а Пятая транзакция получает уже обновлённые же собой данные (v5).

Примечание. На диаграмме не показано действие запроса INSERT. В рамках данного уровня изоляции, строки добавленные, например в шаге 3, в Первой транзакции, были бы НЕ ДОСТУПНЫ Второй, Третьей и Четвёртой транзакциям после завершения Первой транзакции. Также на диаграмме не показан результат ROLLBACK (Шаги 8 и 11). В случае если бы Вторая и Третья транзакции делали какие либо изменения над не заблокированными данными, то все эти изменения не были бы зафиксированы, так как транзакции завершаются неудачно (суть свойства — Atomicity).

Уровень изоляции Serializable гарантирует, что все затронутые в транзакции данные не будут изменены другими транзакциями. На этом уровне появление "фантомов" исключается, поэтому становятся возможными сложные конкурентные операции. На практике такой уровень изоляции требуется в учетных системах.

Для транзакций содержащих только SELECT запросы, использование уровня изоляции Serializable оправдывает себя тогда, когда вы не хотите видеть внесённые изменения параллельно завершёнными транзакциями в ходе работы текущей транзакции.

## Аномалия сериализации (Потерянное обновление)

Ещё один феномен чтения данных, описывается тем, что результат успешной фиксации группы транзакций оказывается несогласованным при всевозможных вариантах исполнения этих транзакций по очереди.

Сориентируйте, пожалуйста, меня в комментариях, если я заблуждаюсь насчёт того, что аномалия сериализации и потерянное обновление связанные между собой феномены.

Документация на сайте PostgreSQL PRO пишет, что Read Committed допускает [«Serialization Anomaly»](https://postgrespro.ru/docs/postgrespro/9.5/transaction-iso.html#MVCC-ISOLEVEL-TABLE). Отечественная Wikipedia, не настаивая на то, что таблица относится именно к PostgreSQL, пишет, что Read Commited предотвращает [аномалию сериализации](https://ru.wikipedia.org/wiki/%D0%A3%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C_%D0%B8%D0%B7%D0%BE%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D1%82%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D0%B9#.D0.9F.D0.BE.D0.B2.D0.B5.D0.B4.D0.B5.D0.BD.D0.B8.D0.B5_.D0.BF.D1.80.D0.B8_.D1.80.D0.B0.D0.B7.D0.BB.D0.B8.D1.87.D0.BD.D1.8B.D1.85_.D1.83.D1.80.D0.BE.D0.B2.D0.BD.D1.8F.D1.85_.D0.B8.D0.B7.D0.BE.D0.BB.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.BD.D0.BE.D1.81.D1.82.D0.B8). Английская Википедиа о таком феномене чтения данных [умалчивает](https://en.wikipedia.org/wiki/Isolation_(database_systems)#Isolation_Levels.2C_Read_Phenomena_and_Locks). Но немецкая Википедия приводит в своей версии таблицы феномен [«Lost Updates»](https://de.wikipedia.org/wiki/Isolation_(Datenbank)#Transaktionsisolation_bei_SQL), указывая на то, что Read Committed может быть не подвержен потере обновлений с дополнительной защитой через курсор (Cursor Stability). Украинская Википедия поддерживает русскоязычную версию статьи, испанская Википедия поддерживает английскую версию статьи. Англоязычная документация по PostgreSQL [не отличается](https://www.postgresql.org/docs/9.6/static/transaction-iso.html#MVCC-ISOLEVEL-TABLE) от документации с сайта PostgreSQL PRO.

Cursor Stability расширяет блокировочное поведение уровня READ COMMITED для SQL-курсоров, добавляя новую операцию чтения (Fetch) по курсору rc (означает read cursor, т.е. чтение по курсору) и требуя, чтобы блокировка устанавливалась на текущем элементе курсора. Блокировка удерживается до тех пор, пока курсор не будет перемещен (пока не измениться его текущий элемент) или закрыт, возможно, операцией фиксации. Естественно, транзакция, читающая по курсору, может изменить текущую строку (wc – запись по курсору), и в этом случае блокировка по записи этой строки будет сохраняться до тех пор, пока транзакция не зафиксируется, даже после передвижения курсора с последующей выборкой следующей строки.

Вот такой результат получился в PostgreSQL 9.6

## Заключение

Понимание уровней изоляции транзакций является важным аспектом при обработке данных в любой многопользовательской СУБД. Уровни изоляции обладают четко определенными характеристиками и поведением. Более высокие уровни изоляции уменьшают возможности параллельной обработки данных и повышают риск взаимной блокировки процессов. Поэтому корректное использование уровней в зависимости от задач приложений всегда является выбором разработчика в зависимости от требований к обеспечению логической целостности данных, к скорости и к возможности параллельной многопользовательской обработки.