



Урок 5

Транзакции и ОПТИМИЗАЦИЯ запросов

Понятие транзакции, для чего они нужны. Анализ выполнения запросов и повышение производительности.

[Основные понятия транзакции](#)

[Принципы ACID](#)

[Уровни изоляции транзакций](#)

[Транзакции в MySQL](#)

[Синтаксис команд BEGIN / COMMIT / ROLLBACK](#)

[Deadlock](#)

[Практическая работа по транзакциям](#)

[EXPLAIN](#)

[Практическая работа по EXPLAIN](#)

[Практическое задание](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Основные понятия транзакции

Транзакция – это целостная операция, объединяющая под собой несколько запросов в один. Вообще, любой набор действий, который выполняется единой процедурой, является транзакцией.

Отличным примером транзакций из реальной жизни являются банковские операции. Одно действие открытия счёта на самом деле состоит из ряда операций.

С точки зрения баз данных, транзакция – это группа последовательных действий с БД, представляющая собой логическую единицу работы с данными. В итоге транзакция заканчивается либо полным успешным выполнением, либо отменой изменений (и тогда она не должна вызывать последствий). По итогам работы СУБД создает журнал транзакций.

Принципы ACID

Atomicity – Атомарность.

Этот принцип отвечает за то, что ни одна транзакция не может быть выполнена частично. Как было сказано выше, выполняются либо все шаги, либо ни один из них. Если транзакция не завершена полностью, все действия отменяются, не влияя на систему.

Isolation – Изолированность.

В процессе выполнения параллельные транзакции не должны влиять друг на друга. Это довольно дорогое требование, а потому в MySQL есть не полная блокировка, а различные уровни изолированности.

Consistency – Согласованность/Консистентность.

Транзакция, которая завершилась неким результатом, т. е. зафиксировавшая результаты своей работы, сохраняет согласованность базы данных. Таким образом каждая успешная транзакция сохраняет базу в корректном состоянии.

Консистентность может иметь и более широкое понимание. К примеру, в той же банковской схеме существует требование к равенству суммы списания и суммы зачисления на определенном промежутке времени. Но это не техническое требование, а бизнес-правило. И такие правила гарантируются уже не технической реализацией, а логикой работы транзакций.

Отметим, что в ходе работы транзакции консистентность не требуется. Т. е. то же списание и зачисление будут проходить разными операциями, и между ними будет некий временной промежуток. Но при изоляции транзакций друг от друга другим транзакционным процессам такая несогласованность не будет доступна. При этом атомарность гарантирует, что транзакция будет завершена либо отменена.

Durability – Долговечность/Надежность.

В случае отказа СУБД по какой-либо причине, изменения, сделанные успешной транзакцией, должны оставаться целостными после восстановления работоспособности системы.

Уровни изоляции транзакций

Для повышения скорости работы в транзакциях реализуется разграничение доступа к данным с разделением по уровням доступа.

0 – Read Uncommitted, Dirty Read – чтение незаконченных изменений своей и других транзакций. При этом, разумеется, не гарантируется, что изменения данных не будут в любой момент отозваны, поэтому подобный тип чтения является источником ошибок.

1 – Read Committed – чтение только изменений, которые были зафиксированы по результатам завершения всех транзакций.

2 – Repeatable Read, Snapshot – чтение всех изменений, но только внутри своей транзакции. Заметьте, что при таком уровне доступа любые изменения, внесённые другими транзакциями после начала своей, недоступны.

3 – Serializable – сериализуемые транзакции. Являются результатом параллельного выполнения сериализуемой транзакции с другими транзакциями. Он должен быть идентичен результату их последовательного выполнения. Проблемы синхронизации не возникают.

Чем выше уровень изоляции, тем больше требуется ресурсов, чтобы его обеспечить. Соответственно, повышение изолированности может приводить к снижению скорости выполнения параллельных транзакций, что является «платой» за повышение надёжности.

Транзакции в MySQL

```
START TRANSACTION

    [transaction_characteristic [, transaction_characteristic] ...]

transaction_characteristic:

    WITH CONSISTENT SNAPSHOT

| READ WRITE

| READ ONLY

BEGIN [WORK]

COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

SET autocommit = {0 | 1}
```

Стоит сказать, что в СУБД MySQL транзакции поддерживаются только в движке InnoDB (мы поговорим о движках подробнее на 7 уроке). Для таблиц, которые не поддерживают транзакции (например, MyISAM), применяется другой принцип поддержания целостности данных – *атомарные операции*. В сравнении с транзакциями они зачастую дают сравнимую или лучшую целостность, имея более высокую производительность. MySQL поддерживает оба принципа, что позволяет выбирать между скоростью и возможностями транзакций. Если код приложения позволяет в случае критических ошибок вызов ROLLBACK, то транзакции явно предпочтительней атомарных операций. Также они гарантируют, что незавершенные обновления не будут сохраняться в БД. Сервер может произвести автоматический откат операций, а БД будет сохранена в консистентном состоянии.

Конечно же, зачастую транзакционные операции можно заменить на атомарные. Но даже при использовании транзакций в системе сохраняется возможность потери данных при непредвиденной остановке сервера. Для MySQL, вне зависимости от наличия транзакций, залогом целостности и безопасности является наличие резервных копий и журнала операций по изменению данных. Эти простые меры позволяют восстановить потерянную информацию.

Атомарность операции состоит в гарантии отсутствия влияния других запросов на выполнение отдельно взятой атомарной операции.

Для обновления данных при помощи одиночной операции можно использовать функции. Применяя следующие приемы, вы создадите весьма эффективную архитектуру:

- Поля модифицируются относительно их текущей величины.
- Обновляются только те поля, которые действительно изменились.

К примеру, при изменении некоторой информации происходит обновление только этой информации и связанных с ней строк. Затем производится проверка, модифицировались эти данные или зависящие

от них по сравнению с исходной строкой. Если обновление не удалось, возвращается сообщение, что «некоторые данные, которые вы изменяли, были модифицированы другим пользователем». Затем пользователю выводится предыдущая версия для выбора конечной версии данных.

```
UPDATE tablename SET pay_back=pay_back+'relative change';

UPDATE customer

SET

customer_date='current_date',

address='new address',

phone='new phone',

money_he_owes_us=money_he_owes_us+'new_money'

WHERE

customer_id=id AND address='old address' AND phone='old phone';
```

Обратите внимание: данный подход эффективно работает даже в случае, если другой пользователь заменит значения в pay_back или money_he_owes_us.

Синтаксис команд BEGIN / COMMIT / ROLLBACK

Из коробки СУБД MySQL функционирует с настройкой autocommit. Данная настройка говорит серверу, что обновления будут моментально фиксироваться на хранилище.

Для движков с поддержкой механизма транзакций (InnoDB) режим autocommit вы можете выключить при помощи:

```
SET AUTOCOMMIT = 0
```

Затем, чтобы сохранять изменения, вам потребуется применять команду COMMIT каждый раз при необходимости записать обновленные данные.

Также переключение на autocommit возможно и для отдельно взятой операции:

```
BEGIN;

SELECT @A:=SUM(salary) FROM table1 WHERE type=1;

UPDATE table2 SET summary=@A WHERE type=1;

COMMIT;
```

Выполняя команды `BEGIN` или `SET AUTOCOMMIT=0`, применяйте `binary log MySQL`, чтобы обеспечить оптимальное хранение резервных копий. Транзакции хранятся в таком журнале в виде единого пакета данных, завершающегося через `COMMIT`, для обеспечения консистентности.

Также текущую транзакцию автоматически завершают команды:

- `ALTER TABLE;`
- `BEGIN;`
- `CREATE INDEX;`
- `DROP DATABASE;`
- `DROP TABLE;`
- `RENAME TABLE;`
- `TRUNCATE.`

Deadlock

В высоконагруженных проектах, которые используют механизм транзакций, рано или поздно появится сообщение об ошибке:

«Deadlock found when trying to get lock; try restarting transaction».

Она говорит, что в процессе транзакции произошла взаимная блокировка двух транзакций. Прежде чем говорить о причинах и способах устранения этой проблемы, стоит разобраться с тем, какие типы блокировок существуют в MySQL.

Официальная документация говорит, что в MySQL имеется два типа блокировок – на чтение (`Shared – S`) и эксклюзивная (`Exclusive – X`). Блокировка типа `S` блокирует выбранные данные на изменение, но позволяет другим запросам читать заблокированные данные. Блокировка типа `X` более строгая. Она не даёт ни читать, ни писать, ни удалять, ни получать блокировку на чтение.

При более детальном изучении выясняется, что существует два дополнительных вида блокировок. Это `intention shared` и `intention exclusive` – блоки таблиц. Они запрещают создание иных блокировок, а также операции `LOCK TABLE`. Создание подобной блокировки со стороны транзакции – это предупреждение, что она хочет создать соответствующую `S` или `X` блокировку.

Итак, если созданная строковая блокировка не даёт выполнять команды, транзакция ожидает разблокировки. И когда две транзакции блокируют друг друга, снятия таких блокировок можно ожидать очень долго – это и есть `deadlock`.

Для получения `deadlock`-а нужны 2 транзакции, эксклюзивная блокировка и блокировка на чтение, а также строка, которую мы будем блокировать. Последовательность действий:

1. Транзакция 1 создаёт блокировку на чтение и продолжается.
2. Транзакция 2 хочет заблокировать эксклюзивно и ожидает момента, когда Транзакция 1 снимет блокировку на чтение.
3. Транзакция 1 хочет получить эксклюзивную блокировку и ожидает, когда Транзакция 2 получит эксклюзивную блокировку, завершит её и освободит ресурсы.

В момент 3 шага и возникает `deadlock`, или взаимная блокировка.

Разработчики СУБД MySQL рекомендуют чаще делать коммиты, контролировать коды ошибок и пытаться перезапустить неудавшуюся транзакцию. Либо можно сразу получать эксклюзивную блокировку. В таком случае на 3 шаге примера выше Транзакция 1 сможет получить блокировку и завершиться.

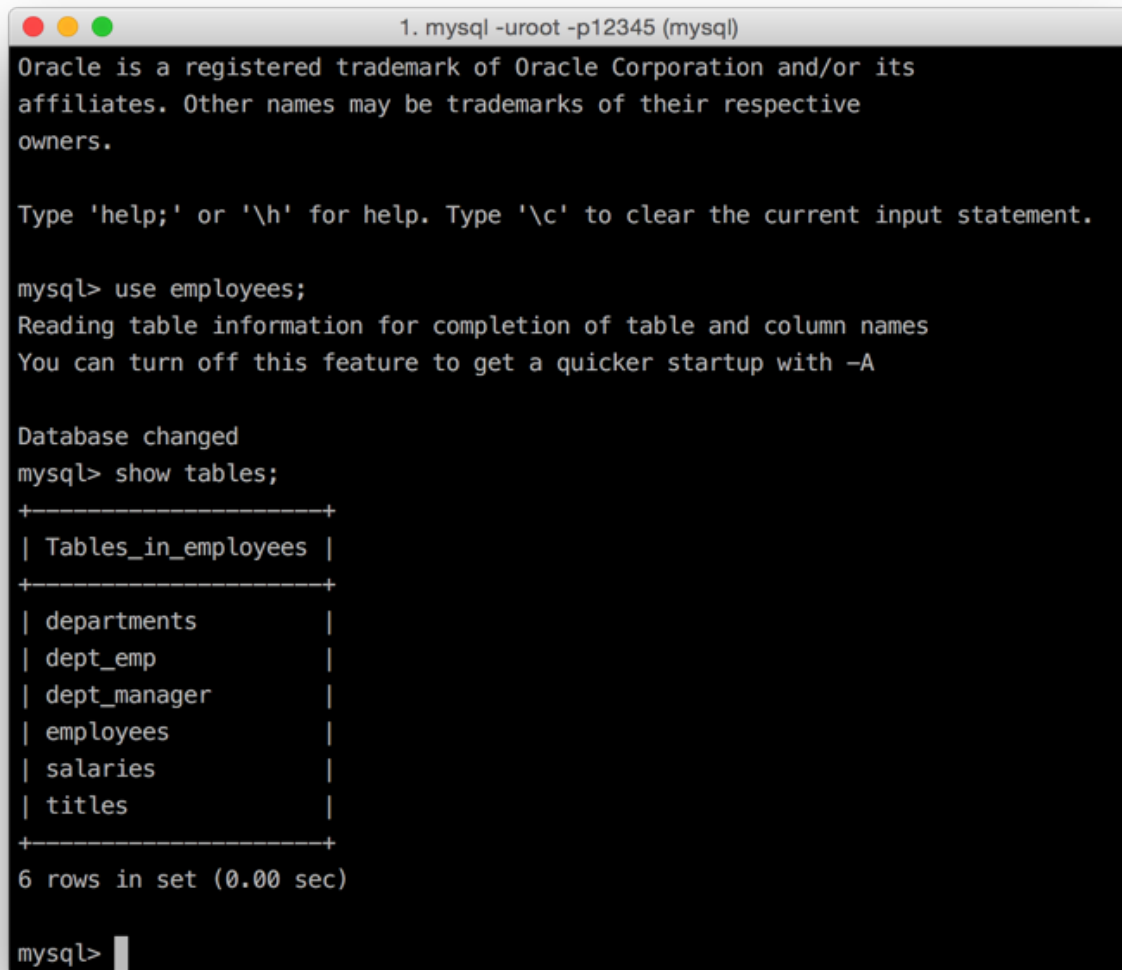
Практическая работа по транзакциям

Давайте рассмотрим несколько простых примеров использования транзакций.

Чтобы продемонстрировать работу транзакций, в консоли используем оператор `LOCK TABLES`,

поскольку работу этого оператора проще показать в виде наглядного примера.

Запустим терминал и авторизуемся в mysql.



```
1. mysql -uroot -p12345 (mysql)
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use employees;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_employees |
+-----+
| departments          |
| dept_emp              |
| dept_manager          |
| employees             |
| salaries              |
| titles                |
+-----+
6 rows in set (0.00 sec)

mysql> 
```

Используем схему employees для демонстрации.

Запустим второе окно и проведем точно такие же действия – авторизуемся и выберем ту же схему.

Пробуем закрыть таблицу из первого окна и выполнить запрос на выборку данных из таблицы.

```
1. mysql -uroot -p12345 (mysql)
mysql (mysql)  #1  mysql (mysql)  #2
mysql> LOCK TABLE employees READ;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM employees LIMIT 10 \G
***** 1. row *****
      emp_no: 10001
    birth_date: 1953-09-02
   first_name: Georgi
    last_name: Facello
        gender: M
     hire_date: 1986-06-26
***** 2. row *****
      emp_no: 10002
    birth_date: 1964-06-02
   first_name: Bezalel
    last_name: Simmel
        gender: F
     hire_date: 1985-11-21
***** 3. row *****
      emp_no: 10003
    birth_date: 1959-12-03
   first_name: Parto
    last_name: Bamford
        gender: M
     hire_date: 1986-08-28
```

Как мы видим, выборка прошла успешно. Давайте попробуем сделать выборку от второго пользователя.

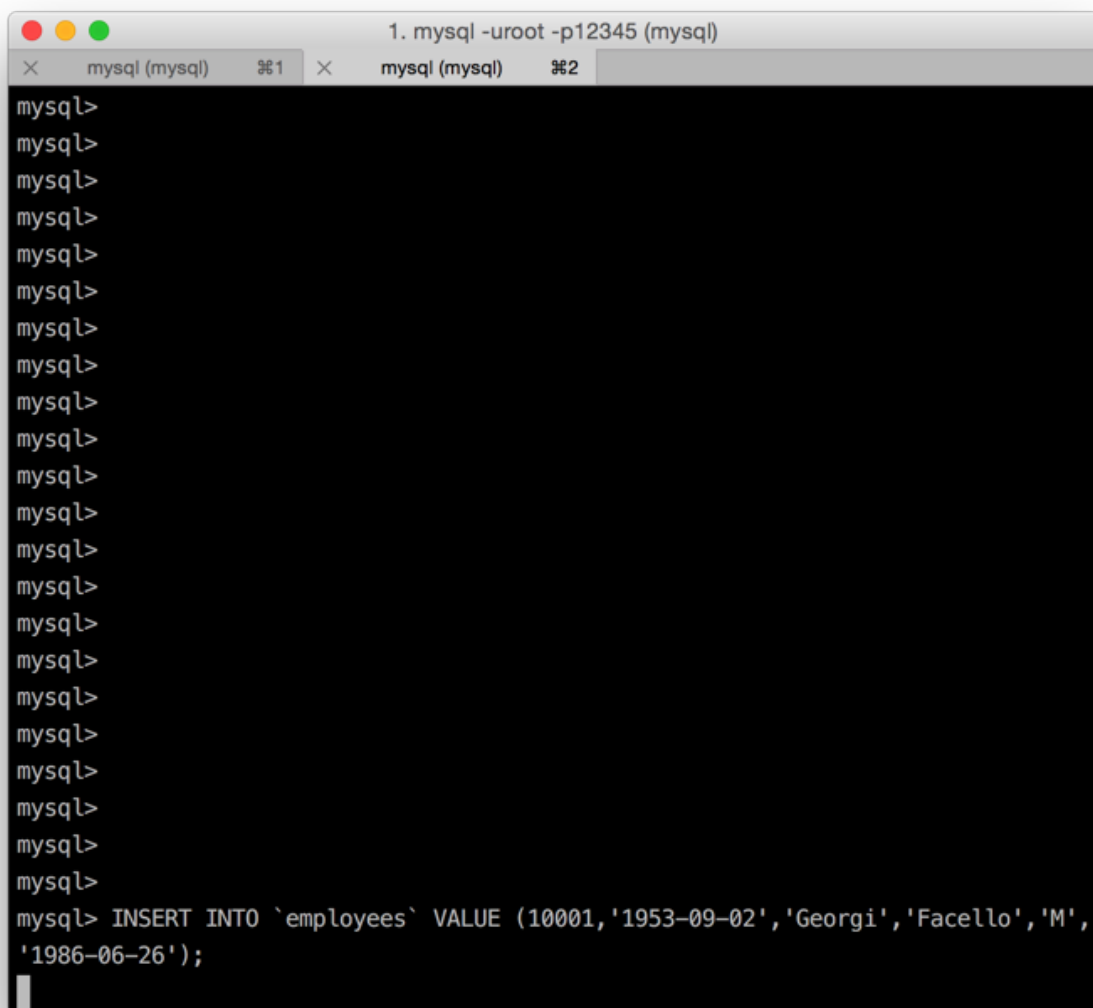

```
1. mysql -uroot -p12345 (mysql)
mysql (mysql) 1
mysql (mysql) 2

mysql> use employees;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT * FROM employees LIMIT 10 \G
***** 1. row *****
      emp_no: 10001
    birth_date: 1953-09-02
   first_name: Georgi
    last_name: Facello
         gender: M
    hire_date: 1986-06-26
***** 2. row *****
      emp_no: 10002
    birth_date: 1964-06-02
   first_name: Bezalel
    last_name: Simmel
         gender: F
    hire_date: 1985-11-21
***** 3. row *****
      emp_no: 10003
    birth_date: 1959-12-03
   first_name: Parto
    last_name: Bamford
```

У второго пользователя выборка тоже проходит успешно.

Теперь давайте попробуем добавить нового сотрудника от второго пользователя.



The image shows a screenshot of a MySQL terminal window. The window has a title bar with three colored buttons (red, yellow, green) and the text "1. mysql -uroot -p12345 (mysql)". Below the title bar, there are two tabs: "mysql (mysql) #1" and "mysql (mysql) #2". The main area of the window is black with white text. It shows a series of "mysql>" prompts. The last prompt is followed by the command: `INSERT INTO `employees` VALUE (10001,'1953-09-02','Georgi','Facello','M','1986-06-26');`. The cursor is at the end of the command, and the command has not been executed.

Добавление данных зависло и не продолжается. Это будет продолжаться до тех пор, пока мы не снимем блокировку данных с таблицы.

После снятия блокировки с таблицы запрос сразу выполнится.

```
1. mysql -uroot -p12345 (mysql)
mysql (mysql)  1  mysql (mysql)  2
birth_date: 1958-02-19
first_name: Saniya
last_name: Kalloufi
gender: M
hire_date: 1994-09-15
***** 9. row *****
emp_no: 10009
birth_date: 1952-04-19
first_name: Sumant
last_name: Peac
gender: F
hire_date: 1985-02-18
***** 10. row *****
emp_no: 10010
birth_date: 1963-06-01
first_name: Duangkaew
last_name: Piveteau
gender: F
hire_date: 1989-08-24
10 rows in set (0.00 sec)

mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)

mysql> 
```

С помощью подобного примера легко показать принцип работы транзакций.

EXPLAIN

Синтаксис оператора:

```
EXPLAIN [table_name]
```

Или:

```
EXPLAIN SELECT [options]
```

Когда SELECT имеет перед собой ключевое слово EXPLAIN, движок MySQL вернет вам план выполнения указанного запроса, а также уведомит о том, в каком порядке и как будут связываться таблицы.

Команда EXPLAIN позволяет проводить анализ, например, когда вам следует добавлять индексы на столбцы. Также есть возможность проанализировать оптимальность шагов выполнения запроса и связывания таблиц. Вы можете изменить порядок связывания через команду STRAIGHT_JOIN.

Для сложных многоуровневых связей EXPLAIN выдает данные по всем указанным в SQL-запросе таблицам. При выводе они указываются ровно в том же порядке следования, в котором их читает MySQL. Все связи создаются за 1 проход («single-sweep multi-join»). MySQL получает строку из первой таблицы в списке, ищет подходящую строку во 2 таблице, затем в 3, и т. д. При завершении обработки MySQL возвращает полученные столбцы и проходит список в обратную сторону до нахождения таблицы с наибольшим совпадением строк. Следующая строка будет считана из этой таблицы.

Вывод команды:

- table. Имя таблицы, с которой будут производиться действия.
- type. Тип связи в таблице.
- possible_keys. Здесь указываются индексы, возможные для отыскания подходящих строк в таблице. Столбец не зависит от порядка таблиц, т. е. в реальной ситуации не все указанные ключи подойдут для сгенерированного оптимизатором перечня таблиц. В столбце данные могут отсутствовать, т. е. оптимизатор не смог подобрать индекс. Тогда для ускорения работы запроса надо посмотреть на условие WHERE и попытаться отыскать в нем ссылки на столбец, который можно добавить в индекс.
- key. Здесь содержится имя индекса, который оптимизатор MySQL будет применять во время выполнения запроса.
- key_len. Длина ключа, примененного оптимизатором. По полученному значению можно увидеть, сколько частей выбранного составного ключа будет применено.
- ref. Показывает столбцы или константы, которые будут использованы с ключом key при выборке.
- rows. Здесь указывается количество строк, выбранных MySQL для анализа перед выполнением запроса.
- Extra. Здесь указывается дополнительная информация о выполнении запроса.
- Distinct. Поиск строк заканчивается после обнаружения первого совпадения строки. MySQL не будет продолжать поиск строк для текущей комбинации.
- Not exists. MySQL произвел анализ LEFT JOIN, нашел присоединяемую строку и не будет искать в этой таблице другие совпадающие строки. Например:

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;
```

Допустим, t2.id задан в NOT NULL. Тогда система проверит t1 и начнет сканировать t2 в поисках строк, соответствующих t1.id. Когда MySQL обнаруживает в t2 искомую запись, он понимает, что t2.id не принимает значение NULL, и останавливает поиск других строк в t2, имеющих идентичный id. Для всей таблицы t1 сервер производит 1 поиск в t2, вне зависимости от количества совпадений в t2.

- range checked for each record (index map: #). MySQL не удалось отыскать нужного и подходящего запросу индекса для использования. Тогда каждому набору строк в предшествующих таблицах он будет искать подходящий индекс и использовать его. Это медленная операция, но она быстрее, чем поиск без индекса.
- Using filesort. MySQL делает дополнительный поиск, чтобы выяснить метод извлечения строк для сортировки.
- Using index. Для получения данных используется только информация из индекса.
- Using temporary. Для выполнения выборки MySQL должен создать временную таблицу. Классическая ситуация для ORDER BY отличного от GROUP BY.
- Where used. WHERE применяется в процессе выявления строк, сопоставляемых со следующей таблицей, или данных, посылаемых в ответе.

В процессе оптимизации скорости работы ваших SQL-запросов смотрите на наличие Using filesort и Using temporary и старайтесь убирать их.

Нижеуказанные типы связывания таблиц мы рассмотрим в порядке убывания скорости работы:

- system. Указанная таблица состоит только из 1 строки.
- const. Таблица содержит до 1 соответствующей строки, считываемой в начале запроса. Таблицы const очень быстрые, т.к. читаются только единожды.
- eq_ref. Для всех наборов строк MySQL будет сопоставлять 1 строку из текущей таблицы. Это самый лучший тип связи строк в реальных многострочных таблицах.
- ref. Читаются абсолютно все строки с соответствующими им индексами для всех имеющихся комбинаций строк. Работает хорошо, когда ключ сопоставлен не всей таблице, а только нескольким её строкам.
- range. Обрабатываются строки, находящиеся в указанном диапазоне. Выбранный оптимизатором индекс выводится в значении key.
- index. Тот же ALL, но идет поиск только по дереву индексов. Он быстрее ALL, так как индекс почти всегда меньше реальных данных.
- ALL. Для всех комбинаций строк оптимизатор производит полный анализ таблицы. Самый медленный вариант.

Для определения эффективности типа связи необходимо получить произведение значений rows. Это довольно грубая оценка количества строк для анализа при выполнении выборки. То же число применяется для задания лимита на количества запросов в системной переменной max_join_size.

Практическая работа по EXPLAIN

Давайте попробуем выполнить несколько запросов и проанализируем вывод

Используем нашу служебную схему employees и выполним запрос.

```
1. mysql -uroot -p (mysql)
~/Documents/projects 9:58:09
$ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.16 Homebrew

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use employees;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

```
EXPLAIN SELECT d.*, e.first_name, e.last_name
FROM departments d
LEFT JOIN dept_manager m
ON d.dept_no = m.dept_no
LEFT JOIN employees e
ON m.emp_no = e.emp_no
```

```
mysql> EXPLAIN SELECT d.*, e.first_name, e.last_name
-> FROM departments d
-> LEFT JOIN dept_manager m
-> ON d.dept_no = m.dept_no
-> LEFT JOIN employees e
-> ON m.emp_no = e.emp_no;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	d	NULL	index	NULL	dept_name	122	NULL	9	100.00	Using index
1	SIMPLE	m	NULL	ref	dept_no	dept_no	12	employees.d.dept_no	2	100.00	Using index
1	SIMPLE	e	NULL	eq_ref	PRIMARY	PRIMARY	4	employees.m.emp_no	1	100.00	NULL

```
3 rows in set, 1 warning (0.00 sec)

mysql>
mysql>
mysql>
mysql>
```

Практическое задание

1. Реализовать практические задания на примере других таблиц и запросов.
2. Подумать, какие операции являются транзакционными, и написать несколько примеров с транзакционными запросами.
3. Проанализировать несколько запросов с помощью EXPLAIN.

Дополнительные материалы

1. «MySQL 5» – Игорь Симдянов, Максим Кузнецов.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_\(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0\)](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))
2. <https://ru.wikipedia.org/wiki/ACID>
3. http://www.mysql.ru/docs/man/ANSI_diff_Transactions.html
4. <http://www.mysql.ru/docs/man/COMMIT.html>
5. <http://www.mysql.ru/docs/man/EXPLAIN.html>

