ЛР6. Индексы. Транзакции

Индексы

https://dev.mysql.com/doc/refman/8.0/en/optimization-indexes.html

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, сбалансированное дерево.

Хотя индексы могут быть созданы для каждого столбца, используемого в запросах, ненужные индексы занимают дисковое пространство и увеличивают время выполнения запроса из-за необходимости оценки, какой индекс следует использовать. Индексы также увеличивают время выполнения запросов на добавление, изменение и удаление данных из-за необходимости обновления индекса после выполнения соответствующих запросов.

Для индексации доступны как обычные типы данных, так и пространственные. Для столбцов типов `CHAR` и `VARCHAR` можно индексировать префикс столбца.

Для создания индекса используется команда `CREATE INDEX`, синтаксис которой представлен ниже:

```
CREATE INDEX index_name ON table_name(column_name);
```

`index_name` – название индекса; `table_name` – название таблицы, в котором содержится индексируемое поле; `column_name` - поле таблицы, для которого создается индекс.

MySQL поддерживает ***уникальные*** индексы. Они применяются для тех полей таблицы, значения в которых должны быть уникальными по всей таблице.

Такие индексы улучшают эффективность выборки для уникальных значений. Для создания уникального индекса используется ключевое слово UNIQUE:

```
CREATE UNIQUE INDEX index_name ON table_name(column_name);
```

Индексы могут быть ***составными***. Так как MySQL часто может использовать только один индекс для выполнения запроса, появляется необходимость в создании составных индексов. Рассмотрим запрос:

```
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

Если по столбцам col1 и col2 существует составной индекс, то соответствующие строки могут выбираться напрямую. В случае, когда по столбцам col1 и col2 существуют раздельные индексы, оптимизатор пытается найти наиболее ограничивающий индекс путем определения, какой индекс найдет меньше строк, и использует данный индекс для выборки этих строк.

Если данная таблица имеет составной индекс, то любой крайний слева префикс этого индекса может использоваться оптимизатором для нахождения строк. Например, если имеется индекс по трем столбцам (col1,col2,col3), то существует потенциальная возможность индексированного поиска по (col1), (col1,col2) и (col1,col2,col3).

В MySQL нельзя использовать составной индекс, если столбцы не образуют крайний слева префикс этого индекса. Например, если индекс существует по (col1,col2,col3), то запрос:

```
SELECT * FROM tbl_name WHERE col2=val2;
```

не будет использовать индекс.

Последние версии MySQL поддерживают ***функциональные*** индексы, т.е. при создании индекса мы можем указать не просто столбец, а использовать некоторую функцию, принимающую данные столбца.

MySQL применяет индексы также для сравнений LIKE, если аргумент в выражении LIKE представляет собой постоянную строку, не начинающуюся с символа-шаблона. Например, следующие команды SELECT используют индексы:

```
SELECT * FROM tbl_name WHERE key_col LIKE "Patrick%";

SELECT * FROM tbl_name WHERE key_col LIKE "Pat%_ck%";
```

Следующая команда SELECT не будет использовать индексы:

```
SELECT * FROM tbl_name WHERE key_col LIKE "%Patrick%";
```

MySQL поддерживает ***полнотекстовые*** индексы. Эти индексы могут быть созданы в столбцах VARCHAR и TEXT во время создания таблицы командой CREATE TABLE или добавлены позже с помощью команд ALTER TABLE или CREATE FULLTEXT INDEX. Загрузка больших массивов данных в таблицу будет происходить намного быстрее, если таблица не содержит индекс FULLTEXT, который затем создается командой ALTER TABLE (или CREATE FULLTEXT INDEX). Загрузка данных в таблицу, уже имеющую индекс FULLTEXT, будет более медленной.

Полнотекстовый поиск выполняется с помощью конструкции MATCH(filelds)… AGAINST(words). Рассмотрим пример:

```
SELECT * FROM articles WHERE MATCH (title, body) AGAINST ('database');
```

В этом примере осуществляется поиск слова «database» в полях title и body таблицы articles. Полученная выборка будет автоматически отсортирована по релевантности (мера сходства между строкой поиска и текстом) – это происходит в случае указания конструкции MATCH-AGAINST внутри блока WHERE и не задано условие сортировки ORDER BY. Эта величина зависит от количества слов в полях tilte и body, того насколько близко данное слово встречается к началу текста, отношения количества встретившихся слов к количеству всех слов в поле и др. Например, релевантность будет не нулевая, если слово database встретится либо в title, либо body, но если оно встретится и там и там, значение релевантности будет выше, нежели если оно два раза встретится в body.

Полнотекстовый поиск позволяет выполнять поиск вхождения подстроки в строку, используя полнотекстовый индекс, в отличии от поиска с помощью оператора LIKE.

Для анализа эффективности выполнения запросов используется оператор EXPLAIN. Данный оператор используется перед оператором SELECT и возвращает таблицу. Колонка key показывает используемый индекс. Колонка possible_keys показывает все индексы, которые могут быть использованы для этого запроса. Колонка rows показывает число записей, которые пришлось прочитать базе данных для выполнения этого запроса.

Транзакции

https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-transactions.html

Транзакция — это операция, состоящая из одного или нескольких запросов к базе данных. Суть транзакций — обеспечить корректное выполнение всех запросов в рамках одной транзакции, а также обеспечить механизм изоляции транзакций друг от друга для решения проблемы совместного доступа к данным. Любая транзакция либо выполняется полностью, либо не выполняется вообще.

В транзакционной модели есть две базовых операции: `COMMIT` и `ROLLBACK`. `COMMIT` выполняет фиксацию всех изменений в транзакции. `ROLLBACK` отменяет (откатывает) изменения, произошедшие в транзакции.

При старте транзакции все последующие изменения сохраняются во временном хранилище. В случае выполнения `COMMIT`, все изменения, выполненные в рамках одной транзакции, сохранятся в физическую БД. В случае выполнения `ROLLBACK` произойдет откат и все изменения, выполненные в рамках этой транзакции, не сохранятся.

По умолчанию MySQL работает в режиме автоматического завершения транзакций, т. е. как только выполняется оператор обновления данных, который модифицирует таблицу, изменения тут же сохраняются на диске. Чтобы объединить операторы в транзакцию, следует отключить этот режим: `set AUTOCOMMIT=0`. Также отключить режим автоматического завершения транзакций для отдельной последовательности операторов можно оператором `START TRANSACTION`.

Для некоторых операторов нельзя выполнить откат с помощью `ROLLBACK`. Это операторы языка определения данных (Data Definition Language). Сюда входят запросы `CREATE, ALTER, DROP, TRUNCATE, COMMENT, RENAME`.

Следующие операторы неявно завершают транзакцию (как если бы перед их выполнением был выдан `COMMIT`): `ALTER FUNCTION, ALTER PROCEDURE, ALTER TABLE, ALTER VIEW, CREATE DATABASE, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE TABLE, CREATE TRIGGER, CREATE VIEW, DROP DATABASE, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP TABLE, DROP TRIGGER, DROP VIEW, RENAME TABLE, TRUNCATE TABLE, BEGIN, SET autocommit = 1, START TRANSACTION`.

Транзакция может быть разделена на точки сохранения. Оператор `SAVEPOINT identifier_name` устанавливает именованную точку сохранения транзакции с именем идентификатора. Если текущая транзакция имеет точку

сохранения с тем же именем, старая точка сохранения удаляется, а новая устанавливается. Оператор `ROLLBACK TO SAVEPOINT identifier_name` откатывает транзакцию до указанной точки сохранения без прерывания транзакции. Изменения, которые выполняются в текущей транзакции для строк после установки точки сохранения, отменяются при откате. Для удаления одной или нескольких точек сохранения используется команда `RELEASE SAVEPOINT identifier_name`.

Уровни изоляций транзакций с разной степенью обеспечивают целостность данных при их одновременной обработке множеством процессов (пользователей). В MySQL существует 4 уровня изоляции транзакций:

- `READ UNCOMMITTED` — самый низкий уровень изоляции. При этом уровне возможно чтение незафиксированных изменений параллельных транзакций.
- `READ COMMITTED` — на этом уровне возможно чтение данных только зафиксированных транзакций. Однако, возможна запись в уже прочитанные внутри транзакции данные.
- `REPEATABLE READ` — на этом уровне изоляции так же возможно чтение данных только зафиксированных транзакций. Так же на этом уровне отсутствует проблема «Неповторяемого чтения», то есть строки, которые участвуют в выборке в рамках транзакции, блокируются и не могут быть *изменены* другими параллельными транзакциями. Но таблицы целиком не блокируются и возможно фантомное чтение — одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям. Другая транзакция в интервалах между этими выборками *добавляет* или *удаляет* строки, используемые в критериях выборки первой транзакции, и успешно заканчивается. В результате получится, что одни и те же выборки в первой транзакции дают разные множества строк. Однако в MySQL проблема фантомного чтения решена на данном уровне изоляции: все чтения данных в пределе одной транзакции используют «снимок» данных, полученный при первом чтении внутри этой транзакции.
- `SERIALIZABLE` — максимальный уровень изоляции, гарантирует неизменяемость данных другими процессами до завершения транзакции. Но в то же время является самым медленным. В MySQL этот уровень изоляции схож с `REPEATABLE READ`, однако все простые операторы `SELECT` неявно преобразуются к виду `SELECT ... FOR SHARE`, если режим автоматического завершения транзакций выключен.

По умолчанию в MySQL установлен уровень изоляции `REPEATABLE READ`. Для смены уровня изоляции используется оператор `SET TRANSACTION`. Этот оператор устанавливает уровень изоляции следующей транзакции, глобально либо только для текущего сеанса.

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL

{READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
SERIALIZABLE }
```