ЛР4. Представления. Индексы

Представления

https://dev.mysql.com/doc/refman/8.0/en/views.html

Представление (`VIEW`) - объект базы данных, представляющий собой виртуальную таблицу. В отличие от обычных таблиц реляционных баз данных, представление не является самостоятельной частью набора данных, хранящегося в базе. Содержимое представления динамически вычисляется на основании данных, находящихся в реальных таблицах. Преимущества использования представлений:

1. Дает возможность гибкой настройки прав доступа к данным за счет того, что права даются не на таблицу, а на представление. Это позволяет назначить права на отдельные строки таблицы или возможность получения не самих данных, а результата каких-то действий над ними.
2. Позволяет разделить логику хранения данных и программного обеспечения. Можно менять структуру данных, не затрагивая программный код. При изменении схемы БД достаточно создать представления, аналогичные таблицам, к которым раньше обращались приложения. Это очень удобно, когда нет возможности изменить программный код или к одной базе данных обращаются несколько приложений с различными требованиями к структуре данных.
3. Удобство в использовании за счет автоматического выполнения таких действий как доступ к определенной части строк и/или столбцов, получение данных из нескольких таблиц и их преобразование с помощью различных функций.
4. Поскольку SQL-запрос, выбирающий данные представления, зафиксирован на момент его создания, СУБД получает возможность применить к этому запросу оптимизацию или предварительную компиляцию, что может повысить производительность выполнения запросов.

Создание представления (упрощенный синтаксис):

```
CREATE [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Данное выражение создает новое представление или заменяет существующее, если используется выражение `OR REPLACE`. Выражение `select_statement` обеспечивает определение представления. SELECT может относиться как к таблицам, так и к другим представлениям, т.е. могут быть вложенными.

Представления должны иметь уникальные имена столбцов. По умолчанию имена столбцов, полученные выражением `SELECT`, используются для имени столбца представления. Явные имена для столбцов представления могут быть заданы с помощью выражения `column_list` как список разделяемых запятой идентификаторов.

Опция `ALGORITHM` определяет, как MySQL обрабатывает представления:

- `MERGE`: текст запроса к представлению объединяется с текстом запроса самого представления к таблицам БД, результат выполнения объединенного запроса возвращается пользователю.
- `TEMPTABLE`: содержимое представлению сохраняется во временную таблицу, которая затем используется для выполнения запроса.
- `UNDEFINED` (значение по умолчанию): MySQL самостоятельно выбирает какой алгоритм использовать при обращении к представлению.

Представление называется *обновляемым*, если к нему могут быть применимы операторы `UPDATE` и `DELETE` для изменения данных в таблицах, на которых основано представление. Для того чтобы представление было обновляемым, должны быть выполнены некоторые условия, в т.ч.:

1. Отсутствие функций агрегации в представлении.
2. Отсутствие следующих выражений в представлении: DISTINCT, GROUP BY, HAVING, UNION.
3. Отсутствие подзапросов в списке выражения SELECT
4. Колонки представления быть простыми ссылками на поля таблиц (а не, например, арифметическими выражениями) и т.д.

Обновляемое представление может допускать добавление данных (`INSERT`), если все поля таблицы-источника, не присутствующие в представлении, имеют значения по умолчанию. Для представлений, основанных на нескольких таблицах, операция добавления данных (`INSERT`) работает только в случае если происходит добавление в единственную реальную таблицу. Удаление данных (`DELETE`) для таких представлений не поддерживается.

При использовании в определении представления конструкции `WITH [CASCADED | LOCAL] CHECK OPTION` все добавляемые или изменяемые строки будут проверяться на соответствие определению представления.

- Изменение данных (`UPDATE`) разрешено только для строк, удовлетворяющих условию `WHERE` в определении представления. Кроме того, новые значения строки также должны удовлетворять значениями удовлетворяет условию `WHERE`.
- Добавление данных (`INSERT`) будет происходить, только если новая строка удовлетворяет условию `WHERE` в определении представления.

Ключевые слова `CASCADED` и `LOCAL` определяют глубину проверки для представлений, основанных на других представлениях:

- Для `LOCAL` происходит проверка условия `WHERE` только в собственном определении представления.
- Для `CASCADED` происходит проверка для всех представлений на которых основано данное представление. Значением по умолчанию является `CASCADED`.

В начале выполнения работы следует выполнить запрос

```
SET SQL_SAFE_UPDATES = 0
```

При значении параметра `SQL_SAFE_UPDATES = 1` выполнение команды `UPDATE` возможно только в том случае, если в запросе указан первичный ключ.

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.

ЛР4. Задание

Представления

1. Создать два не обновляемых представления, возвращающих пользователю результат из нескольких таблиц, с разными алгоритмами обработки представления.

2. Создать обновляемое представление, не позволяющее выполнить команду `INSERT`.

3. Создать обновляемое представление, позволяющее выполнить команду `INSERT`.

4. Создать вложенное обновляемое представление с проверкой ограничений (`WITH CHECK OPTION`).

Индексы

Для демонстрации эффективности использования индексов при выполнении лабораторной работы используется база данных с большим числом записей в таблицах - часть БД сервиса discogs.com. Discogs - веб-сайт с одной из крупнейших баз данных музыкальных исполнителей и их изданий от различных музыкальных компаний. В состав данной БД входят следующие таблицы:

- `artist` – содержит информацию об исполнителях (включая музыкальные коллективы).
- `group` – содержит информацию о принадлежности исполнителя к музыкальному коллективу
- `namevaration` – содержит информацию о различных вариациях имен (псевдонимов) исполнителей.
- `release` - содержит информацию о музыкальных релизах.
- `releaseartist` – описывает связь между исполнителем и релизом.
- `style` - содержит информацию о музыкальном стиле релизов.

База данных**:**

https://drive.google.com/file/d/18pra3Zu-47c0ZiPFGWLtQ8A5tIdhe7EX/view?usp=sharing

Импорт БД доступен через меню «Server – Data import».

Общий план лабораторной работы:

1. Составляются и выполняются запросы согласно заданиям.
2. Оценивается время выполнения запросов.
3. Анализируется план выполнения запросов.
4. Создаются необходимые индексы для повышения быстродействия запросов. Выполнение запроса должно исключать полное сканирование таблицы (отсутствие `Table Scan` в анализе запроса).
5. Оценивается время выполнения тех же запросов при наличии созданных индексов.

Запросы

1. Найти информацию по заданному исполнителю, используя его имя.
2. Найти всех участников указанного музыкального коллектива (по названию коллектива).
3. Найти все релизы заданного исполнителя и отсортировать их по дате выпуска. Вывести имя исполнителя, название релиза, дату выхода.
4. Найти все главные релизы, выпущенные в указанный год, с указанием стиля релиза. Релиз является главным, если поле `release.IS_MAIN_RELEASE` = 1.
5. Найти всех исполнителей, в описании (профиле) которых встречается указанное выражение, с использованием полнотекстового запроса.