Тема 6. Индексы и производительность.

Цель занятия:

Понять принцип работы индексов в MySQL. Научиться создавать и использовать индексы. Овладеть навыками анализа планов выполнения запросов. Приобрести практические навыки оптимизации запросов.

Учебные вопросы:

1. Введение.
2. Понятие индекса.
3. Типы индексов.
4. Создание и использование индексов.
5. Оптимизация запросов.
6. Заключение.

1. Введение.

Производительность базы данных – это скорость, с которой база данных обрабатывает запросы.

Она напрямую влияет на пользовательский опыт, особенно в приложениях с высокой нагрузкой, таких как веб-сайты, мобильные приложения и системы управления предприятиями.

Почему производительность так важна?

- **Пользовательский опыт:** Медленная база данных приводит к длительным задержкам при выполнении запросов, что негативно сказывается на удовлетворенности пользователей.
- **Масштабируемость:** Высокая производительность позволяет системе обрабатывать растущее количество запросов без потери скорости.
- **Доступность:** Медленная база данных может стать узким местом, ограничивая доступность системы.
- **Конкурентоспособность:** В современном мире быстрая и надежная работа приложений является ключевым фактором успеха.

Факторы, влияющие на производительность баз данных

- **Объем данных:** Чем больше данных, тем дольше может занимать поиск и обработка.
- **Сложность запросов:** Сложные запросы с большим количеством соединений, агрегатных функций и подзапросов требуют больше ресурсов для выполнения.
- **Индексные структуры:** Наличие и качество индексов существенно влияют на скорость поиска данных.
- **Оборудование:** Производительность сервера, на котором работает база данных, включая процессор, оперативную память и дисковую подсистему.
- **Настройка базы данных:** Параметры конфигурации базы данных, такие как размер буферов, кэш и алгоритмы сортировки, также влияют на производительность.
- **Конкурентная нагрузка:** Одновременное выполнение большого количества запросов может привести к замедлению работы.
- **Программное обеспечение:** Версия базы данных, драйверы и приложения, взаимодействующие с базой данных.

Оптимизация базы данных направлена на устранение узких мест и повышение ее производительности. Она включает в себя:

- **Анализ запросов:** Идентификация медленно выполняющихся запросов и выявление причин их медленной работы.
- **Создание индексов:** Выбор оптимальных столбцов для индексации и создание индексов для ускорения часто выполняемых запросов.
- **Настройка параметров сервера:** Корректировка параметров конфигурации базы данных для оптимизации работы под конкретную нагрузку.
- **Оптимизация запросов:** Переписывание запросов для более эффективного использования индексов и уменьшения объема обрабатываемых данных.
- **Нормализация данных:** Правильная организация данных в базе данных для минимизации избыточности и улучшения целостности данных.
- **Фрагментация и дефрагментация:** Регулярная проверка и оптимизация структуры таблиц для предотвращения фрагментации данных.

2. Понятие индекса.

Индекс в базе данных – это специальная структура данных, которая ускоряет поиск информации.

Представьте обычный книжный указатель: в нем перечислены все темы, встречающиеся в книге, и указаны страницы, где эти темы обсуждаются.

Благодаря указателю вы можете быстро найти нужную информацию, не просматривая всю книгу от корки до корки.

Книга – это ваша таблица в базе данных, содержащая множество записей.

Указатель – это индекс, содержащий значения одного или нескольких столбцов таблицы и указатели на соответствующие записи.

Как устроен индекс и обеспечивает быстрый поиск

Индексы обычно строятся на основе деревьев, таких как B-дерево.

Это эффективные структуры данных, которые позволяют быстро находить элементы по ключу.

Структура индекса:

•**Ключ:** Это значение из одного или нескольких столбцов таблицы, по которому осуществляется поиск.

•**Указатель:** Ссылка на запись в таблице, содержащую это значение ключа.

Как происходит поиск:

1. **Сравнение ключа:** При выполнении запроса, система сравнивает искомый ключ с ключами в индексе.
2. **Переход по дереву:** Двигаясь по дереву индекса, система быстро находит нужное значение ключа.
3. **Получение данных:** По найденному указателю система обращается к таблице и извлекает необходимые данные.

Преимущества использования индексов:

- **Ускорение поиска данных**: Благодаря индексам база данных может быстро находить нужные записи, особенно при выполнении запросов, использующих условия WHERE по индексированным столбцам.
- **Улучшение производительности запросов**: Индексы оптимизируют процесс выполнения запросов, снижая количество данных, которые необходимо просканировать.
- **Создание уникальных значений**: Индексы могут использоваться для обеспечения уникальности данных в столбце.

Недостатки использования индексов:

- **Затраты на хранение и обслуживание**: Индексы занимают дополнительное место на диске и требуют ресурсов для создания и обновления.
- **Замедление операций** вставки, удаления и обновления: При изменении данных в индексированных столбцах, индекс также должен быть обновлен, что может замедлить эти операции.
- **Сложность в выборе индексов**: Неправильный выбор столбцов для индексации может привести к снижению производительности, а не к ее повышению.

Когда стоит использовать индексы?

- **Часто используемые условия в запросах**: Если какой-то столбец часто используется в условиях WHERE, JOIN или ORDER BY, то создание индекса по этому столбцу может значительно ускорить выполнение запросов.
- **Большие таблицы**: Для больших таблиц индексы особенно полезны, так как они позволяют быстро найти нужные данные среди огромного количества записей.
- **Уникальные значения**: Если столбец должен содержать только уникальные значения, то создание уникального индекса поможет предотвратить дублирование данных.

Важно помнить:

- Не злоупотребляйте индексами: Чрезмерное количество индексов может негативно сказаться на производительности, так как увеличивается нагрузка на запись и чтение данных.
- Регулярно анализируйте использование индексов: С течением времени структура данных может меняться, и индексы, которые были эффективны ранее, могут стать неэффективными.

3. Типы индексов.

Существует несколько типов индексов, каждый из которых имеет свои особенности и предназначен для решения определенных задач.

Основные типы индексов.

1. Кластерный индекс, определяет физическое расположение данных на диске. Он связан с таблицей один к одному и определяет порядок, в котором строки таблицы хранятся на диске.

Особенности:

- В таблице может быть только один кластерный индекс.
- Данные в таблице физически упорядочены по значениям ключа кластерного индекса.
- В MySQL кластерный индекс обычно создается неявно при определении первичного ключа.

2. Некластерный индекс - это дополнительная структура данных, которая содержит набор значений ключа и указателей на соответствующие строки в таблице.

Особенности:

- В таблице может быть несколько некластерных индексов.
- Не определяет физическое расположение данных.
- Когда использовать:
- Частые операции поиска по неключевым столбцам
- Частые операции сортировки

Другие типы индексов:

- Уникальный индекс: Гарантирует, что все значения в индексированном столбце уникальны.
- Составной индекс: Создается по нескольким столбцам и используется для ускорения запросов, которые используют комбинации этих столбцов в условиях WHERE.
- Полнотекстовый индекс: Используется для поиска по текстовым данным, поддерживает поиск по словам, фразам и морфологический анализ.
- Пространственный индекс: Используется для индексирования геопространственных данных (например, координат на карте).
- Функциональный индекс: Создается на основе результата выражения или функции, а не на простом столбце.

4. Создание и использование индексов.

Создать индекс можно с помощью команды CREATE INDEX. Синтаксис:

CREATE INDEX имя_индекса ON имя_таблицы (столбец1, столбец2, ...);

Пример создания индекса:

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

Этот запрос создаст индекс с именем **idx_column_name** на столбце **column_name** таблицы **table_name**.

Создание уникального индекса:

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

Создание уникального индекса в MySQL означает, что индексируемое поле (или комбинация полей) должно содержать только уникальные значения. Это означает, что MySQL не позволит вставить или обновить данные, которые нарушат уникальность значений в колонке или комбинации колонок, на которых был создан такой индекс.

Создание составного индекса:

```
CREATE INDEX idx_name ON users (last_name, first_name);
```

Составной индекс в MySQL — это индекс, созданный на нескольких столбцах таблицы. Он позволяет оптимизировать запросы, которые фильтруются или сортируются по комбинации этих столбцов.

В примере MySQL будет использовать этот индекс для ускорения запросов, которые фильтруются по **last_name** и **first_name.**

Индекс с указанием длины. В MySQL можно создавать индексы с указанием длины для текстовых или бинарных столбцов (например, VARCHAR, TEXT, BLOB). Указание длины индекса позволяет индексировать только первые N символов значения в колонке, что уменьшает размер индекса и может ускорить операции вставки и обновления данных.

Индекс, который будет индексировать только первые 10 символов столбца **email**:

```
CREATE INDEX idx_email ON users (email(10));
```

5. Оптимизация запросов.

Оптимизация запросов в MySQL — это процесс улучшения производительности запросов путем их переписывания, использования индексов, а также анализа планов выполнения.

План выполнения показывает, как MySQL будет обрабатывать запрос: какие таблицы будут затронуты, какие индексы используются, порядок выполнения операций и т.д.

Для анализа плана выполнения используется команда **EXPLAIN**. Она помогает увидеть, как MySQL оптимизирует и исполняет запрос, что позволяет выявить узкие места и улучшить производительность.

План выполнения — это последовательность операций, которые выполняет MySQL для получения результата запроса. План включает:

- Таблицы, к которым обращается запрос.
- Индексы, которые MySQL использует для поиска.
- Оценку количества строк, которые MySQL будет сканировать.
- Порядок соединений таблиц.

Получение плана выполнения.

Для получения плана выполнения нужно добавить ключевое слово EXPLAIN перед запросом:

```
EXPLAIN SELECT * FROM users WHERE email = 'john@example.com';
```

Результаты команды EXPLAIN обычно включают следующие столбцы:

Колонка	Описание
id	Идентификатор каждого запроса. Если запрос состоит из нескольких частей (например, подзапросы), будет несколько строк с разными id.
select_type	Тип запроса (например, SIMPLE, PRIMARY, SUBQUERY, DERIVED). Показывает, является ли это подзапросом или основным запросом.
table	Таблица, к которой обращается запрос на этой стадии.
type	Тип соединения (например, ALL, index, range, ref, eq_ref, const, NULL). Это один из важнейших показателей, который показывает эффективность запроса.
possible_keys	Индексы, которые могут быть использованы для выполнения запроса.
key	Индекс, который фактически используется для выполнения запроса.
key_len	Длина ключа, который MySQL использует. Это важно для понимания, какую часть индекса MySQL использует (всегда ли используется весь индекс).
ref	Показывает, какие столбцы или константы сравниваются с индексом.
rows	Примерное количество строк, которые MySQL должен проверить для выполнения запроса. Чем меньше, тем лучше.
Extra	Дополнительная информация, например, будет ли использована сортировка или объединение временных таблиц.

Важные значения поля type (типы соединений):

- **ALL** — Полный скан таблицы. Используется, когда нет индекса. Самый медленный тип.
- **index** — Полный скан индекса. Лучше, чем ALL, но всё равно затрагивает большое количество строк.
- **range** — Использование индекса для поиска диапазона значений (например, BETWEEN, >, <).
- **ref** — Индекс используется для поиска по точным значениям (например, сравнение с ключом).
- **eq_ref** — Используется, когда для каждой строки из одной таблицы есть точное соответствие в другой. Очень эффективный тип соединения.
- **const** — Таблица имеет только одну строку, или запрос возвращает одну строку. Очень быстрое выполнение.
- **NULL** — MySQL не выполняет доступа к таблице (например, используется COUNT() без условий).

Пример анализа плана выполнения. Допустим, у нас есть таблица users:

```sql
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(100),
  last_name VARCHAR(100),
  email VARCHAR(100)
);

CREATE INDEX idx_email ON users (email);
```

Выполним запрос с использованием индекса:

```
EXPLAIN SELECT * FROM users WHERE email = 'john@example.com';
```

Результат может быть следующим:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	users	ref	idx_email	idx_email	303	const	1	Using where

Интерпретация:

type = ref: Это значит, что MySQL использует индекс idx_email для поиска по точному значению (лучше, чем полный скан).

rows = 1: MySQL ожидает, что проверит только одну строку, что хорошо.

key = idx_email: Используется индекс на поле email.

Extra = Using where: Дополнительно используется условие WHERE, чтобы отфильтровать результаты.

Если бы не было индекса на поле email, запрос выполнялся бы медленнее:

```
EXPLAIN SELECT * FROM users WHERE first_name = 'John';
```

Результат:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	users	ALL	NULL	NULL	NULL	NULL	1000	Using where

Интерпретация:

type = ALL: Полный скан таблицы. Это медленный тип выполнения, так как MySQL должен просканировать все строки таблицы.

rows = 1000: Ожидается, что MySQL просканирует 1000 строк, что затратно по времени.

Общие рекомендации по оптимизации запросов в MySQL.

Существует ряд техник и подходов, которые помогают минимизировать задержки, снизить нагрузку на сервер и ускорить выполнение запросов.

1. Избегание функций в условиях WHERE

Использование функций в операторе WHERE может негативно влиять на производительность, так как MySQL не сможет использовать индексы эффективно. Вместо этого MySQL выполнит полное сканирование таблицы для каждого вызова функции.

Запрос с функцией. Этот запрос не использует индекс на поле created_at из-за функции YEAR().

```
SELECT * FROM users WHERE YEAR(created_at) = 2024;
```

2. Использование покрывающих индексов

Покрывающий индекс — это индекс, который содержит все данные, необходимые для выполнения запроса. Если индекс содержит все запрашиваемые столбцы, MySQL может извлечь данные из индекса, не обращаясь к строкам таблицы.

Пример:

```sql
SELECT email FROM users WHERE id = 1;
```

Для ускорения можно создать индекс, который будет покрывать как поле id (для поиска), так и поле email (для извлечения данных):

```sql
CREATE INDEX idx_users_id_email ON users (id, email);
```

Теперь MySQL сможет полностью обработать запрос, используя только индекс без обращения к таблице, что значительно ускорит выполнение.

Домашнее задание:

1. Повторить материал лекции.

Контрольные вопросы:

- Что такое индекс в базе данных и зачем он нужен?
- Какие факторы влияют на производительность базы данных?
- В чем разница между кластерным и некластерным индексами?
- Как создается индекс в MySQL? Приведите пример команды.
- Какие существуют типы индексов в MySQL и в чем их особенности?
- Какие преимущества и недостатки использования индексов?
- В каких случаях имеет смысл использовать уникальный индекс?
- Что такое покрывающий индекс и как он может ускорить выполнение запросов?
- Как анализировать план выполнения запросов в MySQL с помощью команды EXPLAIN?
- Какие основные методы оптимизации запросов в MySQL вы знаете?

Список литературы:

1. В. Ю. Кара-ушанов SQL — язык реляционных баз данных
2. А. Б. ГРАДУСОВ. Введение в технологию баз данных
3. А.Мотеев. Уроки MySQL

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com