**\*Слайд номер 1\***

Доброго дня, сьогодні Андрій і я хочемо розказати про оптимізацію запитів в MySQL.

Спочатку я коротко розкажу про оптимізацію на прикладі пари запитів з WHERE, range, GROUP BY, потім ми перейдемо до основної частини нашої доповіді - індекси в MySQL. Після чого Андрій покаже на практиці усі запити, у тому числі з прикладів на слайдах.

**\*Слайд номер 2\***

Спершу, ми розглянемо оптимізації, які можна зробити для обробки запитів з WHERE. У прикладах використовуються оператори SELECT, але ті ж самі оптимізації застосовуються у операторах DELETE та UPDATE.

\*Відкидання непотрібних дужок для виразів\*

Перший приклад стосується більш не оптимізації, а читабельності.

MySQL робить автоматично оптимізацію роботи по пришвидшенню арифметичних операцій, тому вам не потрібно розставляти усі ці дужки, тому, залишайте запити у більш зрозумілій формі.

\*Згортання непотрібних констант\*

Другий приклад стосується згортання непотрібних констант, якщо є можливість уникнути додаткової умови для константи і значення виразу відоме, то це значення можна зразу записати в необхідну умову.

**\*Слайд номер 3\***

Тепер про оптимізацію запитів діапазону

Для інтервалу значення індексу можуть бути зручно представлені відповідними умовами у WHERE, позначеними як умови діапазону, замість інтервалів.

**\*Слайд номер 4\***

Loose Index Scan

Використовується при вибірці, під час виконання якої значення одного з індексів може залишатись незмінним для певної кількості значень, що дозволяє згрупувати такі значення і використовувати індекс, визначений в першому значенні з цієї групи.

**\*Слайд номер 5\***

Тепер я хочу розповісти про індекси в MySQL. Чому їх варто використовувати, які вони бувають та як вони працюють.

**\*Слайд номер 6\***

Спершу, як загалом дані читаються з диска.

На жорсткому диску немає такого поняття, як файл. Є поняття блок. Один файл зазвичай займає кілька блоків. Кожен блок знає, який блок йде після нього. Файл ділиться на фрагменти і кожен з них зберігається в порожній блок.

При читанні файлу, ми по черзі проходимся по всім блокам і збираємо файл з частин. Блоки одного файлу можуть бути фрагментовані. Тоді читання файлу сповільниться, тому що доведеться стрибати до різних ділянок диска.

**\*Слайд номер 7\***

\*Пошук даних в MySQL\*

Таблиці MySQL - це звичайні файли. Виконаємо запит такого виду:

SELECT \* FROM \_cities WHERE country\_id = 119

MySQL при цьому відкриває файл, де зберігаються дані з таблиці \_cities. А далі - починає перебирати весь файл, щоб знайти потрібні записи.

Крім цього, MySQL буде порівнювати дані в кожному рядку таблиці зі значенням в запиті. В нашому випадку, є таблиця, у якій 2 246 813 записів. Тому, MySQL прочитає їх усі, і порівняє колонку country\_id кожного з них зі значенням 119, і відбере тільки ті, які співпали.

Звідси випливають дві проблеми. Перша, багато блоків, які потрібно перевірити, а отже - низька швидкість. Друга, велика кількість операцій порівняння.

**\*Слайд номер 8\***

Для того щоб вирішити цю проблему, ми можемо, наприклад, відсортувати всі наші записи по зростанню, чи по спаданню, по ідентифікатору країни. Тоді, використовуючи, наприклад, алгоритм бінарного пошуку, ми могли б значно швидше знайти необхідне значення.

А індекс - це і є відсортований набір значень.

**\*Слайд номер 9\***

Як використовувати індекси, та які вони бувають.

Індекси бувають кластерні та некластерні.

У чому різниця? Некластерний індекс зберігає тільки посилання на записи таблиці. Коли відбувається робота з індексом, визначається тільки список записів (точніше список їх первинних ключів), що підходять під запит. Після цього відбувається ще один запит - для отримання даних кожного запису з цього списку.

Кластерні індекси зберігають дані записів цілком, а не посилання на них. При роботі з таким індексом не потрібно додаткової операції читання даних.

Некластерні в свою чергу поділяються на декілька видів.

Найпростіший з них, створюється для тих колонок, у яких присутні в умові WHERE та в умові яких є ORDER BY.

Також існують унікальні індекси для колонок, значення в яких повинні бути унікальними по всій таблиці. Такі індекси покращують ефективність вибірки для унікальних значень.

Ще є складені індекси, для запитів, в яких використовується кілька колонок, бо MySQL може використовувати тільки один індекс для запиту (крім випадків, коли MySQL здатний об'єднати результати вибірок за кількома індексами). Вони використовуються для пошуку по діапазону та сортування.

Для того щоб розуміти який саме індекс потрібно використовувати дуже корисним буде селектор EXPLAIN, бо з його допомогою можна дізнатися скільки рядків було задіяно у запиті.

**\*Переходимо до демонстрації запитів\***

Андрій буде покаже як оптимізувати запити на практиці, як створити простий, складений, унікальний індекси, коментуючи хід своїх дій.

**\*Слайд номер 10\***

Отже, тепер можна підбити підсумок. Оптимізація в MySQL відіграє важливу роль, в залежності від ситуації, потрібно використовувати відповідні інструменти належним чином.

Пишіть читабельні запити, MySQL робить необхідні оптимізації пришвидшення автоматично.

Індекси слід створювати в міру виявлення повільних запитів. Запити, які виконуються більш 1 секунди є першими кандидатами на оптимізацію.

Починайте створення індексів з найбільш частих запитів. Запит, що виконується секунду, але 1000 разів в день завдає більше шкоди, ніж запит на десять секунд, який виконується кілька разів в день.

Не створюйте індекси на таблицях, число записів в яких менше кількох тисяч. Для таких розмірів виграш від використання індексу буде майже непомітний.

Не створюйте індекси заздалегідь, наприклад, в середовищі розробки. Індекси повинні встановлюватися виключно під форму і тип навантаження працюючої системи.

Видаляйте індекси, що не використовуються.

Виділяйте час на аналіз і організацію індексів в MySQL (і інших базах даних). Зручно буде організувати тестову середу з копією реальних даних і перевіряти там різні структури індексів.

**\*Слайд номер 11\***

Дякуємо за увагу