Оглавление

[1. В чем различие применения таблиц MyIsam и InnoDB в СУБД MySQL? ……………………………………………………………………..1](#_Toc60737185)

[2. Выбор типов полей в MySQL. Приведите практические примеры. ………………………………………………………………………3](#_Toc60737186)

[3. Приведите синтаксис SQL-инструкции, в соответствии с которыми задаются представления. Как задается использование алгоритмов UNDEFINED, MERGE и TEMPTABLE при создании представления? 8](#_Toc60737187)

[4. Чем отличается процедура от функции (назначение, синтаксис использования, oперации)? 10](#_Toc60737188)

[5. Чем отличается хранимая процедура от триггера (назначение, синтаксис спользования, операции)? 12](#_Toc60737189)

[6. Дайте определение триггеру (TRIGGER). Приведите синтаксис SQL-запроса для создания триггера. Сколько может быть создано триггеров для таблиц MyISAM и InnoDB? Где физически хранятся триггеры? Как они исполняются на физическом уровне? 13](#_Toc60737190)

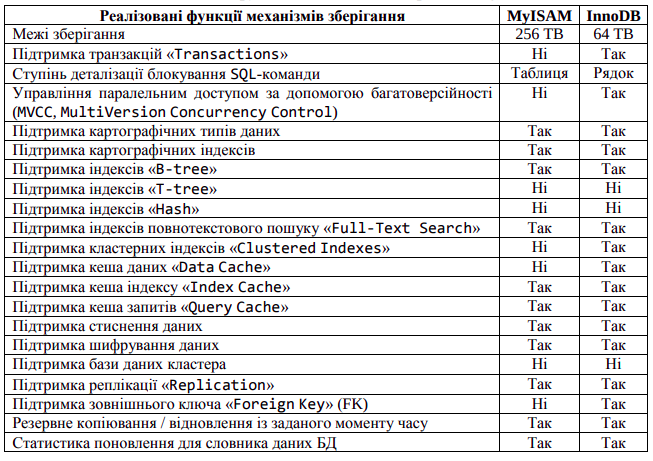
[7. Перечислите все возможные виды реализации ссылочной целостности связи по внешнему ключу (foreign key). В каких типах таблиц и для каких операций в СУБД MySQL реализована ссылочная целостность? Приведите примеры на все возможные действия триггера из любой предметной области. 14](#_Toc60737191)

[8. B-Tree-индексы. Типы запросов, в которых может использоваться B-Tree-индекс. Приведите пример использования B-Tree-индекса. 16](#_Toc60737192)

[9. Хеш-индексы. Построение собственных хеш-индексов. Приведите пример использования хеш-индекса 18](#_Toc60737193)

# В чем различие применения таблиц MyIsam и InnoDB в СУБД MySQL?

В чому відмінність застосування таблиць MyIsam та InnoDB



Перша істотна відмінність полягає в тому, що InnoDB реалізує блокування на рівні рядків, в той час як MyISAM може виконувати тільки блокування на рівні таблиць. Краще відновлення після збоїв в InnoDB. Проте, він не має FULLTEXT пошукових індексів до v5.6. InnoDB також реалізує транзакції, зовнішні ключі і обмеження відносин, в той час як MyISAM цього не робить.

Тим не менш, вони обидва мають свої унікальні переваги в їх користь і недоліки один проти одного. Кожен з них більш підходить в деяких сценаріях, ніж інший.

Ще одна важлива відмінність, полягає в тому, як здійснюється кешування для кожного механізму зберігання. Основний використовуваний механізм для MyISAM - це кеш ключів. Він кешує тільки індексні сторінки з файлів .MYI. Для 32-бітної ОС обмеження 4GB. Для 64-біт 8GB.

Основним використовуваним механізмом для InnoDB є буферний пул. Він кешує дані і індексні сторінки з таблиць InnoDB, до яких здійснюється доступ. Налаштування розміру log-файлу лежить за шляхом /etc/my.cnf.

InnoDB пропонує:

* ACID транзакції
* блокування на рівні рядків
* обмеження зовнішнього ключа
* автоматичне відновлення після збою
* стиснення таблиці (читання / запис)
* просторові типи даних (без просторових індексів)

У InnoDB всі дані поспіль, крім TEXT і BLOB, можуть займати не більше 8000 байт. У InnoDB COUNT (\*) (коли WHERE, GROUP BY або JOIN не використовується) виконується повільніше, ніж в MyISAM, тому що кількість рядків не зберігається всередині. InnoDB зберігає дані і індекси в одному файлі.

MyISAM пропонує:

* Швидке виконання COUNT (\*) з (коли WHERE, GROUP BY або JOIN не використовується)
* менше місця на диску
* дуже високий ступінь стиснення таблиці (тільки для читання)
* просторові типи даних і індекси (R-дерево) (оновлення: підтримується в InnoDB з MySQL 5.7)

Немає автоматичного відновлення після збою, але він пропонує функціональність таблиці відновлення. Немає обмежень по зовнішньому ключу. Таблиці MyISAM зазвичай більш компактні за розміром на диску в порівнянні з таблицями InnoDB. Таблиці MyISAM можуть бути значно зменшені в розмірах шляхом стиснення за допомогою myisampack, якщо це необхідно, але стають доступними тільки для читання. MyISAM зберігає індекси в одному файлі і дані в іншому. MyISAM використовує ключові буфери для кешування індексів і залишає управління кешуванням даних операційній системі.

В цілому, я б порекомендував InnoDB для більшості цілей і MyISAM тільки для спеціалізованих цілей. InnoDB - на сьогодні залишається двигуном за замовчуванням в нових версіях MySQL.

# Выбор типов полей в MySQL. Приведите практические примеры.

Вібир типів полів

При визначенні стовпців таблиці для них необхідно вказати тип даних. Кожен стовпець повинен мати тип даних. Тип даних визначає, які значення можуть зберігатися в стовпці, скільки вони будуть займати місця в пам'яті.

MySQL надає наступні типи даних, які можна розбити на ряд груп.

Символьні типи:

**CHAR:** надає рядок фіксованої довжини.

Довжина збереженої рядки вказується в дужках, наприклад, CHAR (10) - рядок з десяти символів. І якщо в таблицю в даний стовпець зберігається рядок з 6 символів (тобто менше встановленої довжини в 10 символів), то рядок доповнюється 4 пробілами і в підсумку все одно буде займати 10 символів

**VARCHAR:** надає рядок змінної довжини.

Довжина збереженої рядки також вказується в дужках, наприклад, VARCHAR (10). Однак на відміну від CHAR збережена рядок буде займати саме стільки місця, як необхідно. Наприклад, якщо визначення довжини в 10 символів, але в стовпець зберігається рядок в 6 символів, то збережена рядок так і буде займати 6 символів плюс додатковий байт, який зберігає довжину рядка.

Починаючи з MySQL 5.6 типи CHAR і VARCHAR за замовчуванням використовують кодування UTF-8, яка дозволяє використовувати до 3 байт для зберігання символу в залежності від мови (для багатьох європейських мов по 1 байту на символ).

Ряд додаткових типів даних представляють текст невизначеної довжини:

**TINYTEXT:** надає текст довжиною до 255 байт.

**TEXT**: надає текст довжиною до 65 КБ.

**MEDIUMTEXT**: надає текст довжиною до 16 МБ

**LARGETEXT**: представляє текст довжиною до 4 ГБ

Числові типи:

**TINYINT**: надає цілі числа від -127 до 128, займає 1 байт

**BOOL**: фактично не представляє окремий тип, а є лише псевдонімом для типу TINYINT (1) і може зберігати два значення 0 і 1. Однак даний тип може також в якості значення приймати вбудовані константи TRUE (представляє число 1) і FALSE (надає число 0 ).

**TINYINT UNSIGNED**: представляє цілі числа від 0 до 255, займає 1 байт

**SMALLINT**: надає цілі числа від -32тис. до 32тис., займає 2 байтa

**SMALLINT UNSIGNED**: надає цілі числа від 0 до 65 тис., займає 2 байти

**MEDIUMINT**: надає цілі числа від -84 тис до 84 тис., займає 3 байти

**MEDIUMINT UNSIGNED:** надає цілі числа від 0 до 16 млн, займає 3 байти

**INT:** представляє цілі числа від -2147483648 до 2147483647, займає 4 байта

**INT UNSIGNED:** надає цілі числа від 0 до 4294967295, займає 4 байта

**BIGINT:** надає цілі числа від -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, займає 8 байт

**BIGINT UNSIGNED:** представляє цілі числа від 0 до 18 446 744 073 709 551 615, займає 8 байт

**DECIMAL:** зберігає числа з фіксованою точністю. Даний тип може приймати два параметри точність і розмір: DECIMAL (точність, розмір).

Параметр точність представляє максимальну кількість цифр, які може зберігати число. Це значення повинно знаходитися в діапазоні від 1 до 65.

Параметр розмір представляє максимальну кількість цифр, які може містити число після коми. Це значення повинно знаходитися в діапазоні від 0 до значення параметра розміру. За замовчуванням воно дорівнює 0.

Наприклад, у визначенні у наступній колонці:

salary DECIMAL (5,2)

Число 5 - точність, а число 2 - розмір, тому даний стовпець може зберігати значення з діапазону від -999.99 до 999.99.

Розмір даних в байтах для DECIMAL залежить від значення, що зберігається.

Даний тип також має псевдоніми NUMERIC, DEC, FIXED.

**FLOAT:** зберігає дробові числа з плаваючою точкою одинарної точності від -3.4028 \* 1038 до 3.4028 \* 1038, займає 4 байта

Може приймати форму FLOAT (M, D), де M - загальна кількість цифр, а D - кількість цифр після коми

**DOUBLE:** зберігає дробові числа з плаваючою точкою подвійної точності від -1.7976 \* 10308 до 1.7976 \* 10308, займає 8 байт. Також може приймати форму DOUBLE (M, D), де M - загальна кількість цифр, а D - кількість цифр після коми.

Даний тип також має псевдоніми REAL і DOUBLE PRECISION, які можна використовувати замість DOUBLE.

Типи для роботи з датою і часом

**DATE:** зберігає дати з 1 січня 1000 року до 31 грудня 9999 року (c "1000-01-01" до "9999-12-31"). За замовчуванням для зберігання використовується формат yyyy-mm-dd. Займає 3 байта.

**TIME:** зберігає час від -838: 59: 59 до 838: 59: 59. За замовчуванням для зберігання часу застосовується формат "hh: mm: ss". Займає 3 байта.

**DATETIME:** об'єднує час і дату, діапазон дат і часу - з 1 січня 1000 року по 31 грудня 9999 року (з "1000-01-01 00:00:00" до "9999-12-31 23:59:59") . Для зберігання за замовчуванням використовується формат "yyyy-mm-dd hh: mm: ss". Займає 8 байт

**TIMESTAMP:** також зберігає дату і час, але в іншому діапазоні: від "1970-01-01 00:00:01" UTC до "2038-01-19 3:14:07" UTC. Займає 4 байти

**YEAR:** зберігає рік у вигляді 4 цифр. Діапазон доступних значень від 1901 до 2155. Займає 1 байт.

Тип Date може приймати дати в різних форматах, однак безпосередньо для зберігання в самій бд дати наводяться до формату "yyyy-mm-dd". Деякі з прийнятих форматів:

yyyy-mm-dd - 2018-05-25

yyyy-m-dd - 2018-5-25

yy-m-dd - 18-05-25

Приклад використання

declare d date;

set d = '2020-10-2';

Приклади значень для типів DATETIME і TIMESTAMP:

2018-05-25 19:21:34

**ENUM:** зберігає одне значення зі списку допустимих значень. Займає 1-2 байта

**SET:** може зберігати кілька значень (до 64 значень) з деякого списку допустимих значень. Займає 1-8 байт.

бінарні типи

**TINYBLOB:** зберігає бінарні дані у вигляді рядка довжиною до 255 байт.

**BLOB:** зберігає бінарні дані у вигляді рядка довжиною до 65 КБ.

**MEDIUMBLOB:** зберігає бінарні дані у вигляді рядка довжиною до 16 МБ

**LARGEBLOB:** зберігає бінарні дані у вигляді рядка довжиною до 4 ГБ

ВСЕГО 30 типов

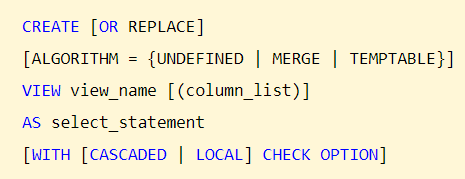
# Приведите синтаксис SQL-инструкции, в соответствии с которыми задаются представления. Как задается использование алгоритмов UNDEFINED, MERGE и TEMPTABLE при создании представления?

Приведіть синтаксис SQL – іструкцій згідно з яких створюються уявлення.

Уявлення (VIEW) - об'єкт бази даних, що є результатом виконання запиту до бази даних, визначеного за допомогою оператора SELECT, в момент звернення до уявлення.

Уявлення іноді називають «віртуальними таблицями». Така назва пов'язана з тим, що уявлення є для користувача як таблиця, але саме воно не містить даних, а витягує їх з таблиць в момент звернення до нього. Якщо дані змінені в базовій таблиці, то користувач отримає актуальні дані при зверненні до подання, котрі використовують цю таблицю; кешування результатів вибірки з таблиці при роботі уявлень не проводиться. При цьому, механізм кешування запитів працює на рівні запитів користувача безвідносно до того, чи звертається користувач до таблиць або уявленням.

Уявлення можуть грунтуватися як на таблицях, так і на інших уявленнях, тобто можуть бути вкладеними (до 32 рівнів вкладеності).



view\_name - ім'я створюваного уявлення.

select\_statement - оператор SELECT, що вибирає дані з таблиць і / або інших уявлень, які будуть міститися в поданні

Оператор CREATE VIEW містить 4 необов'язкові конструкції:

OR REPLACE - при використанні даної конструкції в разі існування уявлення з таким ім'ям старе буде видалено, а нове створено. В іншому випадку виникне помилка, що інформує про сществованіі уявлення з таким ім'ям і нове уявлення створено не буде. Слід зазначити одну особливість - імена таблиць і уявлень в рамках однієї бази даних повинні бути унікальні, тобто не можна створити уявлення з ім'ям вже існуючої таблиці. Однак конструкція OR REPLACE діє тільки на уявлення і заміщати таблицю не буде.

ALGORITM - визначає алгоритм, який використовується при зверненні до уявлення

column\_list - задає імена полів уявлення.

WITH CHECK OPTION - при використанні даної конструкції всі додані чи редаговані рядки будуть перевірятися на відповідність визначення уявлення. У разі невідповідності - дана зміна не буде виконана.

Існує два алгоритма, використовуваних MySQL при зверненні до уявлення: MERGE і TEMPTABLE.

У разі алгоритму MERGE, MySQL при зверненні до уявлення додає в використаний оператор відповідні частини з визначення уявлення і виконує отриманий оператор.

У разі алгоритму TEMPTABLE, MySQL заносить вміст уявлення в тимчасову таблицю, над якою потім виконується оператор звернений до уявлення. У разі використання цього алгоритму уявлення не може бути оновлюваних даних

При створенні уявлення є можливість явно вказати використовуваний алгоритм за допомогою необов'язкової конструкції ALGORITHM = UNDEFINED

UNDEFINED означає, що MySQL сам вибирає який алгоритм використовувати при зверненні до уявлення. Це значення за замовчуванням, якщо дана конструкція відсутня.

# Чем отличается процедура от функции (назначение, синтаксис использования, oперации)?

Сама загальна відмінність між процедурами та функціями полягає в тому, що вони викликаються по-різному і для різних цілей:

Процедура не повертає значення. Замість цього він викликається за допомогою оператора CALL для виконання такої операції, як зміна таблиці або обробка отриманих результатів.

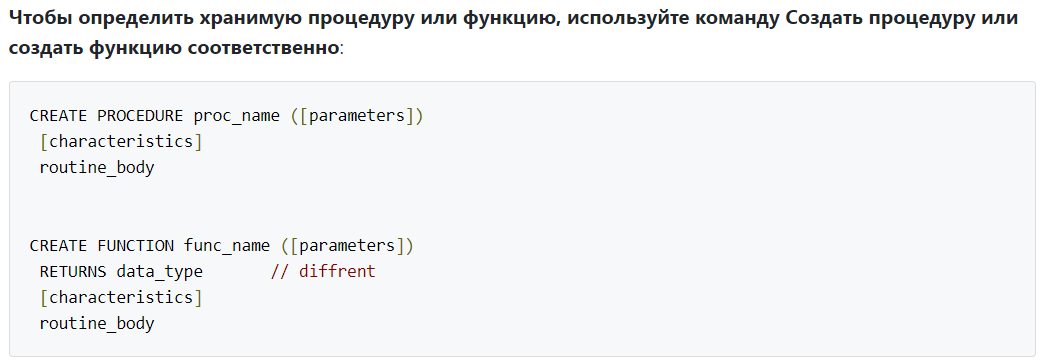
Функція викликається всередині виразу (select наприклад) і повертає одне значення безпосередньо об'єкту, який її викликає для використання в виразі. Неможна викликати функцію за допомогою оператора CALL, а також не можете викликати процедуру в вираженні.

Параметри процедури можуть бути визначені як тільки для введення, тільки для виведення або і те, і інше. А для FUNCTION параметри завжди розглядаються як параметри IN. Це означає, що процедура може передавати значення назад викликає об'єкту за допомогою вихідних параметрів.

Варіант з вихідним параметром, дає вам на виході конкретну змінну (або змінні), яку далі можна відразу використовувати поза процедурою:

Доступ до цих значень можна отримати в інструкціях, наступних за інструкцією CALL. Функції мають тільки вхідні параметри. В результаті, хоча і процедури, і функції можуть мати параметри, оголошення параметрів процедури відрізняється від оголошення параметрів функцій.

Функції повертають значення, тому у визначенні функції повинен бути оператор RETURNS для передачі даних об’єкту. Крім того, в тілі функції повинен бути хоча б один оператор RETURN, щоб повернути значення об'єкту. RETURNS і RETURN не фігурують в визначеннях процедур.



Процедура-це скомпільований план виконання, де в якості опції немає. Функція розбирається і компілюється під час виконання. Збережені процедури, що зберігаються у вигляді псевдокоду в базі даних тобто в компільованою формі.

Процедура має безпеку і зменшує мережу трафік. Функції зазвичай використовуються для обчислень, а процедури зазвичай використовуються для виконання бізнес-логіки.

Функції не можуть впливати на стан бази даних (Оператори, які роблять явну або неявну фіксацію або відкат, заборонені в функцій) У той час як Збережені процедури можуть впливати на стан бази даних за допомогою commit тощо.

Функції не можуть використовувати оператори FLUSH, в той час як збережені процедури можуть це робити.

Збережені функції не можуть бути рекурсивними. Рекурсивні процедури, відключені за замовчуванням, але можуть бути включені на сервері, встановивши системну змінну сервера max\_sp\_recursion\_depth в нульове значення.

В межах збереженої функції або тригера забороняється змінювати таблицю, яка вже використовується (для читання або запису) оператором, який викликав функцію або тригер.

# Чем отличается хранимая процедура от триггера (назначение, синтаксис спользования, операции)?

Збережені процедури представляють собою набір команд SQL, які можуть компілюватися і зберігатися на сервері. Таким чином, замість того, щоб зберігати часто використовуваний запит, клієнти можуть посилатися на відповідну процедуру, що зберігається. Це забезпечує кращу продуктивність, оскільки даний запит повинен аналізуватися тільки один раз і зменшується трафік між сервером і клієнтом. Концептуальний рівень можна також підвищити за рахунок створення на сервері бібліотеки функцій.

Тригер являє собою процедуру, що зберігається, яка активізується при настанні певної події. Наприклад, можна задати збережену процедуру, яка спрацьовує кожного разу при видаленні запису з транзакционной таблиці - таким чином забезпечується автоматичне видалення відповідного замовника з таблиці замовників, коли все його транзакції видаляються.

СМ отчёт по процедурам в таблицу.

<https://dl.nure.ua/pluginfile.php/242773/mod_resource/content/1/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0_%D0%9F%D0%97_%D0%9F%D0%92%D0%9D%D0%A1%D0%97%D0%94_v2020.pdf> 61

# Дайте определение триггеру (TRIGGER). Приведите синтаксис SQL-запроса для создания триггера. Сколько может быть создано триггеров для таблиц MyISAM и InnoDB? Где физически хранятся триггеры? Как они исполняются на физическом уровне?

Тригер являє собою процедуру, що зберігається, яка активізується при настанні певної події. Наприклад, можна задати збережену процедуру, яка спрацьовує кожного разу при видаленні запису з транзакционной таблиці - таким чином забезпечується автоматичне видалення відповідного замовника з таблиці замовників, коли все його транзакції видаляються.



CREATE TRIGGER ins\_sum BEFORE INSERT ON account

FOR EACH ROW SET @sum = @sum + NEW.amount;

DROP TRIGGER test.ins\_sum;

Може бути простіше визначити збережену процедуру окремо, а потім викликати її з тригера за допомогою простого CALLоператора.

Існують обмеження на те, що може з'являтися в операторах, які запускаються тригером при активації:

Тригер не може використовувати CALL оператор для виклику збережених функцій, які повертають дані клієнта або використовують динамічний SQL. (Збереженим процедурам дозволено повертати дані в тригер через параметри OUT або INOUT.)

Тригер не може використовувати оператори, які явно або неявно починають або завершують дію, наприклад START TRANSACTION, COMMIT або ROLLBACK. (ROLLBACK to SAVEPOINT дозволень, тому що це не завершує транзакцію.).

Тригери в даний час збережені в .TRG-файлах, один такий файл на таблицю. Іншими словами, тригер належить таблиці.

У майбутньому, планується змінити це так, щоб інформація тригера включалася в .FRM-файл, який визначає структуру таблиці. Також планується зробити тригери рівня бази даних, щоб привести їх у належність з SQL-стандартом.

# Перечислите все возможные виды реализации ссылочной целостности связи по внешнему ключу (foreign key). В каких типах таблиц и для каких операций в СУБД MySQL реализована ссылочная целостность? Приведите примеры на все возможные действия триггера из любой предметной области.

Посилальна цілісність - це обмеження бази даних, що гарантує, що посилання між даними є дійсно правомірними і неушкодженими. Посилальна цілісність є фундаментальним принципом теорії баз даних і виникає з тієї ідеї, що база даних повинна не тільки зберігати дані, але і активно сприяти забезпеченню їх якості.

Підтримка посилальної цілісності в базі даних забезпечує багато переваг.

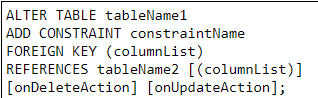
Покращена якість даних. Очевидною перевагою є підтримка якості даних, що зберігаються в базі даних. Помилки можуть як і раніше існувати, але, хоча б посилання будуть справжніми і неушкодженими.

Прискорення розробки. Посилальна цілісність оголошується в СУБД. Це набагато продуктивніше (на один або два порядки), ніж написання спеціального програмного коду.

Менша кількість помилок. Оголошення посилальної цілісності є більш лаконічними, ніж еквівалентний програмний код. По суті, такі оголошення призводять до повторного використання перевіреного і відтестувати коду загального призначення в сервері баз даних, а не до нової реалізації однієї і тієї ж логіки від випадку до випадку.

Узгодженість між додатками. Посилальна цілісність забезпечує якість даних для декількох прикладних програм, які можуть звертатися до бази даних.

Синтаксис SQL для визначення посилальної цілісності виглядає, по суті, подібно наступного. Квадратні дужки виділяють необов'язкові параметри. Стовпці зовнішнього ключа знаходяться в таблиці table1, а стовпці первинного ключа (або іншої комбінації стовпців з властивістю унікальності значень) - у таблиці table2.



У SQL зовнішній ключ може посилатися на будь-яку комбінацію стовпців з властивістю унікальності в таблиці, на яку веде посилання. У стандарті SQL забезпечуються наступні дії з підтримки посилальної цілісності при видаленні рядків.

Cascade. Видалення запису може привести до видалення записів з відповідним значенням зовнішнього ключа.

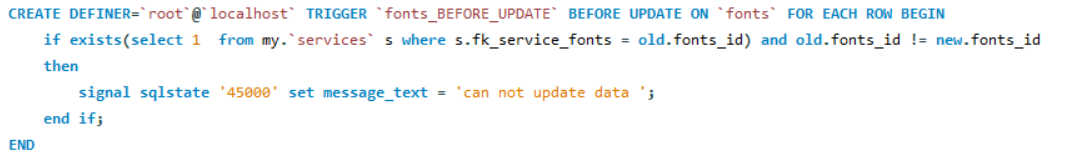
NO ACTION - при видаленні або оновленні записів з таблиці предка зв'язку з таблицями нащадками ігноруються, до закриття транзакції

Set null. Видалення запису може призводити до установки в відповідні зовнішні ключі невизначеного значення.

RESTRICT - при спробі видалення або оновлення записів з таблиці предка, при наявності пов'язаних записів в таблиці нащадка буде видана помилка

На рівні тригерів цілісність забезпечується діями перед вставкою, після вставки, перед оновленням та після, перед видаленням та після.

У якості преривання та виклику іменованої помилки можна використовувати SIGNAL SQLSTATE '01000' SET MESSAGE\_TEXT = 'A warning occurred', MYSQL\_ERRNO = 1000 – як шлях до виклику виключення.

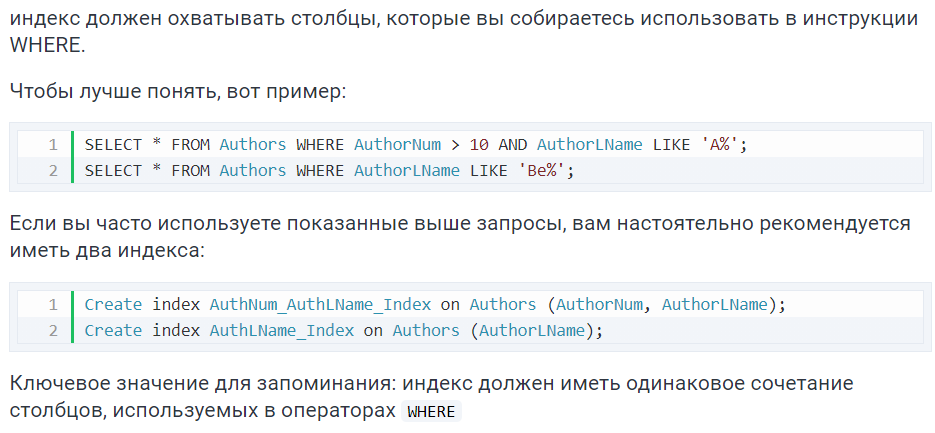


# B-Tree-индексы. Типы запросов, в которых может использоваться B-Tree-индекс. Приведите пример использования B-Tree-индекса.

<http://pushorigin.ru/sql/index-select> - Индексы

Індекс являє собою відмінну структуру даних в базі даних. Його основна мета - надати упорядковане представлення індексованих даних через логічне впорядкування, яке не залежить від фізичного упорядкування, використовуючи двозв’язаний список і деревоподібну структуру, відому як збалансоване дерево пошуку (B-tree). B-дерева вони сортують дані і дозволяють виконувати пошук, доступ, вставки і видалення в логарифмічному часі. Через двозв’язаний список ми можемо легко або швидко переходити назад або вперед у міру необхідності в індекс для різних запитів. Вставки стають простими, так як нам потрібно тільки переставити покажчики на різні частини даних. Бази даних використовують цей двозв’язаний список для з'єднання листових вузлів (зазвичай в дереві B + або B-дереві), кожен з яких зберігається на сторінці і встановлює логічне впорядкування між листовими вузлами. Операції типу UPDATE або INSERT стають повільніше, оскільки насправді це дві операції запису в файлової системі (одна для даних таблиці і одна для даних індексу).

Положення знаків підстановки має величезне значення. У реченнях LIKE необхідно використовувати символи до знаків підстановки під час обходу дерева; інші не звужують сканований діапазон індексів. Чим більше вибірковим є префікс речення LIKE, тим вужчим стає сканований індекс. Це прискорює пошук індексу. Необхідно уникати речень LIKE типу "% OTINGER%". Для повнотекстових запитів MySQL пропонує MATCH і AGAINST ключові слова. Починаючи з MySQL 5.6 можна мати повнотектові індекси



# Хеш-индексы. Построение собственных хеш-индексов. Приведите пример использования хеш-индекса

Хеш-індекси (керовані користувачем) використовуються тільки в движку MEMORY і можуть використовуватися для точного пошуку. Для кожного стовпця обчислюється хеш, який потім використовується для пошуку. Припустимо, що хеш для значення mylongstring дорівнює «1234». MySQL збереже хеш-значення разом з покажчиком на рядок. Якщо ви виконаєте такий запит:

SELECT \* FROM table WHERE my\_indexed\_column = 'mylongstring';

Умова WHERE буде хешуватись, і в результаті вийде '1234', індекс вкаже на правильну рядок. Основною перевагою індексу HASH є той факт, що хеші можуть бути набагато менше, ніж саме индексируемое значення. Однак основна проблема полягає в тому, що цей тип індексу не можна використовувати ні для чого іншого, крім простого пошуку. Ви не можете використовувати index on (column1, column2) для покриття запиту тільки з 'column1' в реченні WHERE. Хеши не унікальні - можуть статися колізії, які сповільнять пошук, оскільки MySQL повинен буде перевіряти більше одного рядка для кожного пошуку за індексом.

Ми згадували, що движок MEMORY використовує хеш-індекси, контрольовані користувачем - це тому, що InnoDB також може використовувати хеш-індекси. Для найбільш часто запитуваних даних InnoDB будує внутрішній хеш-індекс поверх індексів B-Tree. У деяких випадках це дозволяє прискорити пошук. Цей механізм називається «Адаптивний індекс хеширования» і є внутрішньою, не налаштовуємою користувачем функцією. Історично склалося так, що при деяких робочих навантаженнях це було предметом спору. Прямо зараз, в MySQL 5.6, ми можемо або повністю відключити його, або керувати кількістю «розділів» в цьому індексі. Однак користувач не може контролювати, які дані InnoDB буде індексувати таким чином.

Hash:

- являє собою результат функції по ключу

- не бачить відстані до найближчих елементів

- не може використовуватися для операцій діапазонів > і <

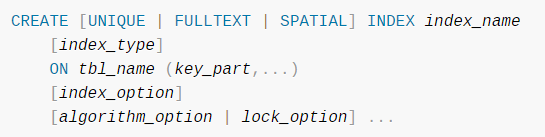
- порівнювання ключа цілком

- лінійний при індексуванні

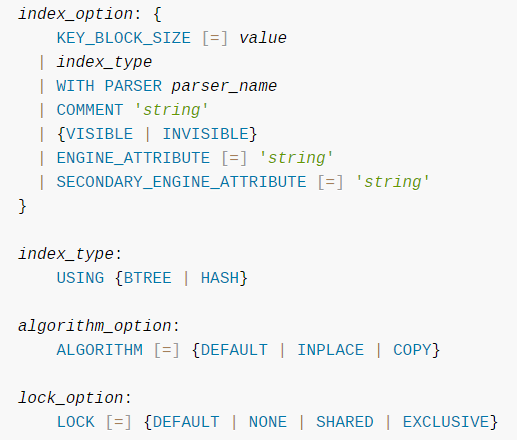
- О (1) при витягуванні значення

Вони не можуть використовуватися для впорядкування і не надають оптимізаторові інформації про те, скільки рядків існує між двома значеннями.

Хеш-індекси не дозволяють використовувати крайній лівий префікс - можна використовувати тільки весь індекс.



Создание индекса



Указания опций индекса





# Составные индексы. Устройства составных индексов. Селективность составного индекса. Приведите пример использования составного индекса. Приведите пример использования части составного индекса.



Создание индекса

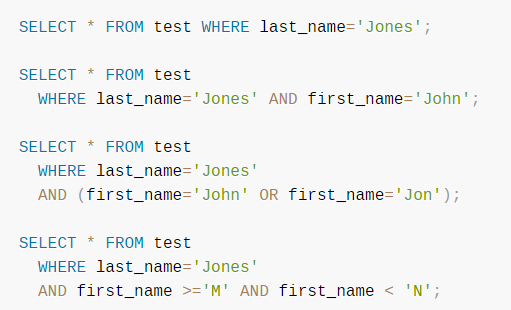


Пример использования

MySQL може створювати складені індекси (тобто індекси за кількома стовпцями). Індекс може складатися максимум з 16 стовпців. Для певних типів даних ви можете проіндексувати префікс стовпчика. Використовуючи синтаксис в специфікації індексу для строкового стовпчика, ви можете створити індекс, який використовує тільки перші символи стовпчика. Таке індексування тільки префікса значень стовпця може значно зменшити розмір індексного файлу. При індексуванні необхідно вказати довжину префікса для індексу.

CREATE TABLE test (blob\_col BLOB, INDEX(blob\_col(10)));

MySQL може використовувати індекси з декількома стовпцями для запитів, які перевіряють всі стовпці в індексі, або запитів, які перевіряють тільки перший стовпець, перші два стовпці, перші три стовпці і т. д. Якщо ви вкажете стовпці в правильному порядку у визначенні індексу, один складовою індекс може прискорити виконання декількох типів запитів до однієї і тієї ж таблиці.



Примеры

Если был создан индекс хранящий первым – last name, а потом first name

Індекс з декількома стовпцями можна розглядати як відсортований масив, рядки якого містять значення, створені шляхом об'єднання значень індексованих стовпців.

# Полнотекстовые индексы. Приведите пример использования полнотекстового индекса.

Багато СУБД підтримують методи повнотекстового пошуку (Fulltext search), які дозволяють дуже швидко знаходити потрібну інформацію в великих обсягах тексту.

На відміну від оператора LIKE, такий тип пошуку передбачає створення відповідного повнотекстового індексу, який представляє собою своєрідний словник згадок слів в полях. Під словом зазвичай розуміється сукупність з не менше 3-х не пробільних символів (але це може бути змінено). Залежно від даних словника може бути обчислена релевантність - порівняльна міра відповідності запиту знайденої інформації.

В MySQL можливості повнотекстового пошуку (тільки для MyISAM-таблиць) підтримуються починаючи з версії 3.23. У наступних версіях механізм зазнав істотні доопрацювання і розширення, перетворившись в могутній засіб для створення пошукових механізмів веб-додатків. Головна особливість - швидко шукати слова в дуже великих обсягах текстової інформації.

Отже, щоб працювати з повнотекстових пошуком, спочатку нам потрібно створити відповідний індекс. Він називається FULLTEXT, і може бути накладений на поля CHAR, VARCHAR і TEXT. Причому, як і у випадку зі звичайним індексом - якщо відбувається пошук по 2-м полях, то потрібен об'єднаний індекс 2-х полів, використовуйте пошук по одному полю - потрібен індекс тільки цього поля.

Завантаження великих масивів даних в таблицю буде відбуватися набагато швидше, якщо таблиця не містить індекс FULLTEXT, який потім створюється командою ALTER TABLE (або CREATE INDEX). Завантаження даних в таблицю, вже має індекс FULLTEXT, буде повільнішою.

Власне для самого повнотекстового пошуку в MySQL використовується конструкція MATCH (filelds) ... AGAINST (words). Вона може працювати в різних режимах, які досить сильно між собою відрізняються. Для всіх діє наступне правило: дана конструкція повертає умовну релевантність, але спосіб обчислення якої може бути різним у залежності від режиму. Ще варто додати що у всіх режимах пошук завжди чутливі до регістру.

MATCH-AGAINST IN NATURAL LANGUAGE MODE

- це основний вид пошуку, який використовується за умовчанням, тобто якщо режим не вказано:

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

У цьому прикладі ми шукаємо слово database в полях title і body таблиці articles на основі індексу

Ще слід зауважити що для IN NATURAL LANGUAGE MODE діє так зване «50% threshold». Це означає, що якщо слово зустрічається більш ніж в 50% всіх відвіданих полів, то воно не буде враховуватися, і пошук по цьому слову не дасть результатів.

MATCH-AGAINST IN BOOLEAN MODE

У бінарному режимі, на відміну від інших режимів, релевантність обчислюється дещо інакше - як умовна міра збігу заданого шаблону. Положення шуканого шаблона в тексті, кількість зустрілися варіантів ролі не грають.

Найважливіша особливість бінарного режиму - можливість вказівки логічних операторів.

У бінарному режимі відсутнє обмеження «50% threshold». Бінарний режим можна використовувати і без створення повнотекстового індексу, проте це буде працювати дуже повільно.

MATCH-AGAINST IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION

Або просто «WITH QUERY EXPANSION». Працює приблизно також, як NATURAL LANGUAGE MODE, з тією лише різницею, то в результат пошуку потрапляють не тільки збіги з шаблоном, а й можливі логічні збіги.

Спочатку MySQL виконує запит аналогічний NATURAL LANGUAGE MODE і формує результат. З цього результату проводиться спроба обчислити слова, які так само мають високу релевантність для отриманої вибірки. У разі, якщо ці слова присутні проводиться пошук і по ним теж, але значення їх на релевантність буде істотно нижче. Віддається змішана вибірка - спочатку ті результати, де слово присутній, а потім ті, які були отримані в результаті «повторного» пошуку.

WITH QUERY EXPANSION не рекомендується використовувати для великих обсягів інформації, так як в результат може потрапити дуже багато зайвого.

Приклади використання індексів в деяких задачах:

прибрати всі HTML-теги

прибрати всі недруковані символи, знаки пунктуації тощо

прибрати всі слова довжиною менше 3-х символів

перевести всі слова в нижній регістр

# 13 Принятие решения о денормализации таблиц. Способы осуществления денормализации

5 Проведення денормалізації фізичної структури даних з метою оптимізації SQL-запитів до баз даних з таблицями MyISAM і InnoDB. Під денормалізацією розуміють зміну фізичної структури даних нормалізованих таблиць і зв’язків для досягнення компромісів за допомогою навмисного введення надлишковості з метою збільшення продуктивності. 4.4.5.1 Обґрунтовується вибір і приймається рішення до використання можливих варіантів денормалізації: – низхідна денормалізація. Передбачає перенесення поля з батьківської таблиці в дочірню (залежну) таблицю; 25 – висхідна денормалізація. Передбачає перенесення поля з дочірньої (залежної) таблиці в батьківську таблицю або додавання нового (обчислюваного) поля в батьківську таблицю, яке залежить від полів дочірньої таблиці; – внутрішньотаблична денормалізація. Передбачає введення надлишковості до однієї з таблиць. Наприклад, якщо в запиті проводиться обчислення з використанням полів однієї таблиці, то поле обчислюваного значення додається в цю таблицю; – денормалізація методом об’єднань таблиць. Передбачає об’єднання двох або більше нормалізованих таблиць з метою усунення операцій зв’язування таблиць в SQL-запитах або зменшення числа запитів для вставки даних; – масштабування – денормалізація з реалізацією горизонтального або вертикального шардінгу зі зміною фізичної структури даних (проводиться в останню чергу). 4.4.5.2 Внесення змін до фізичної структури даних за варіантами денормалізації (крім масштабування) у бази даних з таблицями MyISAM і InnoDB. Навести оцінку часу виконання SQL-запитів для кожного обраного варіанту денормалізації (див. 4.4.5.1). Оцінку проводити для нормалізованої й денормалізованих баз даних.