MySQL 經典38 問!

菜鳥教程 2022-01-25 08:53

以下文章來源於程序員大彬， 作者程序員大彬

程序員大彬

專注Java技術乾貨分享

來源| 程序員大彬
作者| 程序員大彬

今天給大家分享 **MySQL 常考的面試題**，看看你們能答對多少🧐

本期MySQL 面試題的**目錄**如下：

- 事務的四大特性？
- 事務隔離級別有哪些？
- 索引
  - 什麼是索引？
  - 索引的優缺點？
  - 索引的作用？
  - 什麼情況下需要建索引？
  - 什麼情況下不建索引？
  - 索引的數據結構
  - Hash 索引和B+ 樹索引的區別？
  - 為什麼B+ 樹比B 樹更適合實現數據庫索引？
  - 索引有什麼分類？
  - 什麼是最左匹配原則？
  - 什麼是聚集索引？
  - 什麼是覆蓋索引？
  - 索引的設計原則？
  - 索引什麼時候會失效？
  - 什麼是前綴索引？
- 常見的存儲引擎有哪些？
- MyISAM 和InnoDB 的區別？
- MVCC 實現原理？

- 快照讀和當前讀
- 共享鎖和排他鎖
- 大表怎麼優化？
- bin log / redo log / undo log
- bin log 和redo log 有什麼區別？
- 講一下MySQL 架構？
- 分庫分錶
- 什麼是分區表？
- 分區表類型
- 查詢語句執行流程？
- 更新語句執行過程？
- exist 和in 的區別？
- truncate、delete 與drop 區別？
- having 和where 的區別？
- 什麼是MySQL 主從同步？
- 為什麼要做主從同步？
- 樂觀鎖和悲觀鎖是什麼？
- 用過processlist 嗎？

事務的四大特性？

事務特性ACID：**原子性**（ `Atomicity` ）、**一致性**（ `Consistency` ）、**隔離性**（ `Isolation` ）、**持久性**（ `Durability` ）。

- **原子性**是指事務包含的所有操作要么全部成功，要么全部失敗回滾。
- **一致性**是指一個事務執行之前和執行之後都必須處於一致性狀態。比如a 與b 賬戶共有1000 塊，兩人之間轉賬之後無論成功還是失敗，他們的賬戶總和還是1000。
- **隔離性**。跟隔離級別相關，如 `read committed` ，一個事務只能讀到已經提交的修改。
- **持久性**是指一個事務一旦被提交了，那麼對數據庫中的數據的改變就是永久性的，即便是在數據庫系統遇到故障的情況下也不會丟失提交事務的操作。

事務隔離級別有哪些？

先了解下幾個概念：臟讀、不可重複讀、幻讀。

- **臟讀**是指在一個事務處理過程裡讀取了另一個未提交的事務中的數據。

- **不可重複讀**是指在對於數據庫中的某行記錄，一個事務範圍內多次查詢卻返回了不同的數據值，這是由於在查詢間隔，另一個事務修改了數據並提交了。
- **幻讀**是當某個事務在讀取某個範圍內的記錄時，另外一個事務又在該範圍內插入了新的記錄，當之前的事務再次讀取該範圍的記錄時，會產生幻行，就像產生幻覺一樣，這就是發生了幻讀。

不可重複讀和臟讀的區別是，臟讀是某一事務讀取了另一個事務未提交的髒數據，而不可重複讀則是讀取了前一事務提交的數據。

幻讀和不可重複讀都是讀取了另一條已經提交的事務，不同的是不可重複讀的重點是修改，幻讀的重點在於新增或者刪除。

事務隔離就是為了解決上面提到的髒讀、不可重複讀、幻讀這幾個問題。

MySQL 數據庫為我們提供的四種隔離級別：

- **Serializable** (串行化)：通過強制事務排序，使之不可能相互衝突，從而解決幻讀問題。
- **Repeatable read** (可重複讀)：MySQL 的默認事務隔離級別，它確保同一事務的多個實例在並發讀取數據時，會看到同樣的數據行，解決了不可重複讀的問題。
- **Read committed** (讀已提交)：一個事務只能看見已經提交事務所做的改變。可避免臟讀的發生。
- **Read uncommitted** (讀未提交)：所有事務都可以看到其他未提交事務的執行結果。

查看隔離級別：

```
select @@transaction_isolation;
```

設置隔離級別：

```
set session transaction isolation level read uncommitted;
```

索引

什麼是索引？

索引是存儲引擎用於提高數據庫表的訪問速度的一種**數據結構**。

索引的優缺點？

優點：

- **加快數據查找的速度**
- 為用來排序或者是分組的字段添加索引，可以**加快分組和排序的速度**
- **加快表與表之間連接的速度**

缺點：

- 建立索引需要**佔用物理空間**
- 會降低表的增刪改的效率，因為每次對錶記錄進行增刪改，需要進行**動態維護索引**，導致增刪改時間變長

索引的作用？

數據是存儲在磁盤上的，查詢數據時，如果沒有索引，會加載所有的數據到內存，依次進行檢索，讀取磁盤次數較多。有了索引，就不需要加載所有數據，因為B+ 樹的高度一般在2-4 層，最多只需要讀取2-4 次磁盤，查詢速度大大提升。

什麼情況下需要建索引？

1. 經常用於查詢的字段
2. 經常用於連接的字段建立索引，可以加快連接的速度
3. 經常需要排序的字段建立索引，因為索引已經排好序，可以加快排序查詢速度

什麼情況下不建索引？

1. `where` 條件中用不到的字段不適合建立索引
2. 表記錄較少
3. 需要經常增刪改
4. **參與列計算**的列不適合建索引
5. **區分度不高**的字段不適合建立索引，如性別等

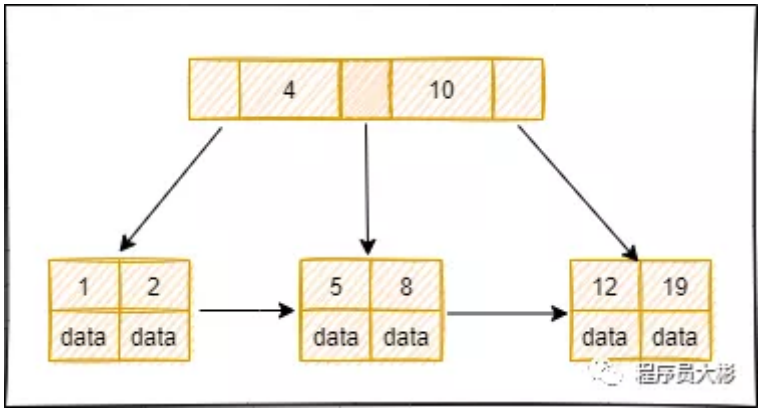
索引的數據結構

索引的數據結構主要有B+ 樹和哈希表，對應的索引分別為B+ 樹索引和哈希索引。InnoDB 引擎的索引類型有B+ 樹索引和哈希索引，默認的索引類型為B+ 樹索引。

B+ 樹索引

B+ 樹是基於B 樹和葉子節點順序訪問指針進行實現，它具有B 樹的平衡性，並且通過順序訪問指針來提高區間查詢的性能。

在B+ 樹中，節點中的 `key` 從左到右遞增排列，如果某個指針的左右相鄰 `key` 分別是key$_i$和key$_{i+1}$，則該指針指向所有 `key` 大於等於key$_i$且小於等於key$_{i+1}$的節點。



進行查找操作時，首先在根節點進行二分查找，找到 `key` 所在的指針，然後遞歸地在指針所指向的節點進行查找。直到查找到葉子節點，然後在葉子節點上進行二分查找，找出 `key` 所對應的數據項。

MySQL 數據庫使用最多的索引類型是 `BTREE` 索引，底層基於B+ 樹數據結構來實現。

```
mysql> show index from blog\G;
*************************** 1. row ***************************
        Table: blog
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: blog_id
    Collation: A
  Cardinality: 4
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Visible: YES
   Expression: NULL
```

哈希索引

哈希索引是基於哈希表實現的，對於每一行數據，存儲引擎會對索引列進行哈希計算得到哈希碼，並且哈希算法要盡量保證不同的列值計算出的哈希碼值是不同的，將哈希碼的值作為哈希表的key 值，將指向數據行的指針作為哈希表的value 值。這樣查找一個數據的時間複雜度就是O(1)，一般多用於精確查找。

Hash 索引和B+ 樹索引的區別？

- 哈希索引**不支持排序**，因為哈希表是無序的。
- 哈希索引**不支持範圍查找**。
- 哈希索引**不支持模糊查詢**及多列索引的最左前綴匹配。
- 因為哈希表中會**存在哈希衝突**，所以哈希索引的性能是不穩定的，而B+ 樹索引的性能是相對穩定的，每次查詢都是從根節點到葉子節點。

為什麼B+ 樹比B 樹更適合實現數據庫索引？

- 由於B+ 樹的數據都存儲在葉子結點中，葉子結點均為索引，方便掃庫，只需要掃一遍葉子結點即可，但是B 樹因為其分支結點同樣存儲著數據，我們要找到具體的數據，需要進行一次中序遍歷按序來掃，所以B+ 樹更加適合在區間查詢的情況，而在數據庫中基於範圍的查詢是非常頻繁的，所以通常B+ 樹用於數據庫索引。

- B+ 樹的節點只存儲索引key 值，具體信息的地址存在於葉子節點的地址中。這就使以頁為單位的索引中可以存放更多的節點，減少更多的I/O 支出。

- B+ 樹的查詢效率更加穩定，任何關鍵字的查找必須走一條從根結點到葉子結點的路。所有關鍵字查詢的路徑長度相同，導致每一個數據的查詢效率相當。

索引有什麼分類？

1、**主鍵索引**：名為primary 的唯一非空索引，不允許有空值。

2、**唯一索引**：索引列中的值必須是唯一的，但是允許為空值。唯一索引和主鍵索引的區別是：唯一約束的列可以為 `null` 且可以存在多個 `null` 值。唯一索引的用途：唯一標識數據庫表中的每條記錄，主要是用來防止數據重複插入。創建唯一索引的SQL 語句如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name UNIQUE KEY(column_1,column_2,...);
```

3、**組合索引**：在表中的多個字段組合上創建的索引，只有在查詢條件中使用了這些字段的左邊字段時，索引才會被使用，使用組合索引時需遵循最左前綴原則。

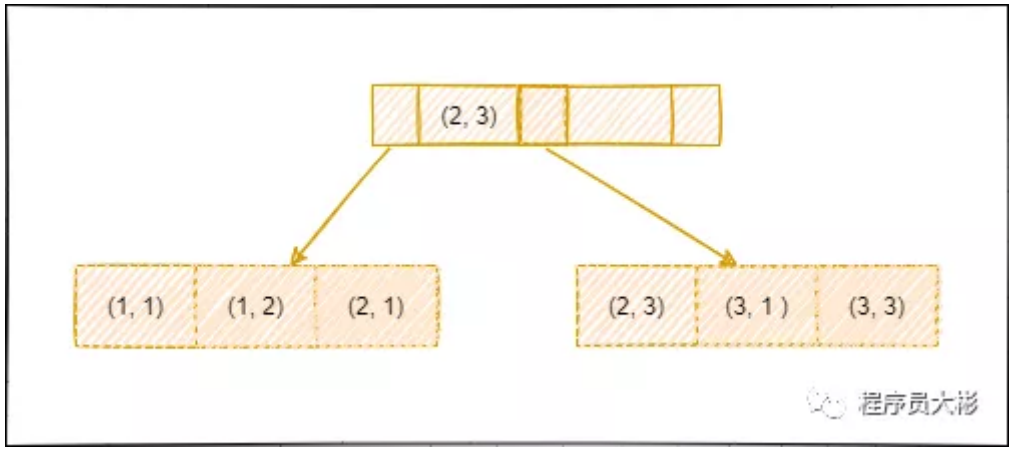
4、**全文索引**：只有在 `MyISAM` 引擎上才能使用，只能在 `CHAR` 、 `VARCHAR` 和 `TEXT` 類型字段上使用全文索引。

什麼是最左匹配原則？

如果SQL 語句中用到了組合索引中的最左邊的索引，那麼這條SQL 語句就可以利用這個組合索引去進行匹配。當遇到範圍查詢(`>` 、 `<` 、 `between` 、 `like`)就會停止匹配，後面的字段不會用到索引。

對 `(a,b,c)` 建立索引，查詢條件使用a/ab/abc 會走索引，使用bc 不會走索引。如果查詢條件為 `a = 1 and b > 2 and c = 3` ，那麼a、b 兩個字段能用到索引，而c 無法使用索引，因為b 字段是范圍查詢，導致後面的字段無法使用索引。

如下圖，對(a, b) 建立索引，a 在索引樹中是全局有序的，而b 是全局無序，局部有序（當a 相等時，會根據b 進行排序）。



當a 的值確定的時候，b 是有序的。例如 `a = 1` 時，b 值為1，2 是有序的狀態。當執行 `a = 1 and b = 2` 時a 和b 字段能用到索引。而對於查詢條件 `a < 4 and b = 2` 時，a 字段能用到索引，b 字段則用不到索引。因為a 的值此時是一個範圍，不是固定的，在這個範圍內b 的值不是有序的，因此b 字段無法使用索引。

什麼是聚集索引？

InnoDB 使用表的主鍵構造主鍵索引樹，同時葉子節點中存放的即為整張表的記錄數據。聚集索引葉子節點的存儲是邏輯上連續的，使用雙向鍊錶連接，葉子節點按照主鍵的順序排序，因此對於主鍵的排序查找和範圍查找速度比較快。

聚集索引的葉子節點就是整張表的行記錄。InnoDB 主鍵使用的是聚簇索引。聚集索引要比非聚集索引查詢效率高很多。

對於 InnoDB 來說，聚集索引一般是表中的主鍵索引，如果表中沒有顯示指定主鍵，則會選擇表中的第一個不允許為 NULL 的唯一索引。如果沒有主鍵也沒有合適的唯一索引，那麼 InnoDB 內部會生成一個隱藏的主鍵作為聚集索引，這個隱藏的主鍵長度為6 個字節，它的值會隨著數據的插入自增。

什麼是覆蓋索引？

select 的數據列只用從索引中就能夠取得，不需要**回表**進行二次查詢，也就是說查詢列要被所使用的索引覆蓋。對於 innodb 表的二級索引，如果索引能覆蓋到查詢的列，那麼就可以避免對主鍵索引的二次查詢。

不是所有類型的索引都可以成為覆蓋索引。覆蓋索引要存儲索引列的值，而哈希索引、全文索引不存儲索引列的值，所以MySQL 使用b+ 樹索引做覆蓋索引。

對於使用了覆蓋索引的查詢，在查詢前面使用 explain ，輸出的extra 列會顯示為 using index 。

比如 user_like 用戶點贊表，組合索引為 (user_id, blog_id) ， user_id 和 blog_id 都不為 null 。

```
explain select blog_id from user_like where user_id = 13;
```

explain 結果的 Extra 列為 Using index ，查詢的列被索引覆蓋，並且where 篩選條件符合最左前綴原則，通過**索引查找**就能直接找到符合條件的數據，不需要回表查詢數據。

```
explain select user_id from user_like where blog_id = 1;
```

explain 結果的 Extra 列為 Using where; Using index ， 查詢的列被索引覆蓋，where 篩選條件不符合最左前綴原則，無法通過索引查找找到符合條件的數據，但可以通過**索引掃描**找到符合條件的數據，也不需要回表查詢數據。

索引的設計原則？

- 索引列的**區分度越高**，索引的效果越好。比如使用性別這種區分度很低的列作為索引，效果就會很差。
- 盡量使用**短索引**，對於較長的字符串進行索引時應該指定一個較短的前綴長度，因為較小的索引涉及到的磁盤I/O 較少，查詢速度更快。
- 索引不是越多越好，每個索引都需要額外的物理空間，維護也需要花費時間。
- 利用**最左前綴原則**。

索引什麼時候會失效？

導致索引失效的情況：

- 對於組合索引，不是使用組合索引最左邊的字段，則不會使用索引
- 以% 開頭的like 查詢如 `%abc` ，無法使用索引；非% 開頭的like 查詢如 `abc%` ，相當於範圍查詢，會使用索引
- 查詢條件中列類型是字符串，沒有使用引號，可能會因為類型不同發生隱式轉換，使索引失效
- 判斷索引列是否不等於某個值時
- 對索引列進行運算
- 查詢條件使用 `or` 連接，也會導致索引失效

什麼是前綴索引？

有時需要在很長的字符列上創建索引，這會造成索引特別大且慢。使用前綴索引可以避免這個問題。

前綴索引是指對文本或者字符串的前幾個字符建立索引，這樣索引的長度更短，查詢速度更快。

創建前綴索引的關鍵在於選擇足夠長的前綴以**保證較高的索引選擇性**。索引選擇性越高查詢效率就越高，因為選擇性高的索引可以讓MySQL 在查找時過濾掉更多的數據行。

建立前綴索引的方式：

```
// email列创建前缀索引
ALTER TABLE table_name ADD KEY(column_name(prefix_length));
```

常見的存儲引擎有哪些？

MySQL 中常用的四種存儲引擎分別是：**MyISAM**、**InnoDB**、**MEMORY**、**ARCHIVE**。MySQL 5.5 版本後默認的存儲引擎為 `InnoDB` 。

InnoDB 存儲引擎

InnoDB 是MySQL **默認的事務型存儲引擎**，使用最廣泛，基於聚簇索引建立的。InnoDB 內部做了很多優化，如能夠自動在內存中創建自適應hash 索引，以加速讀操作。

優點：支持事務和崩潰修復能力；引入了行級鎖和外鍵約束。

缺點：佔用的數據空間相對較大。

適用場景：需要事務支持，並且有較高的並發讀寫頻率。

MyISAM 存儲引擎

數據以緊密格式存儲。對於只讀數據，或者表比較小、可以容忍修復操作，可以使用MyISAM 引擎。MyISAM 會將表存儲在兩個文件中，數據文件 `.MYD` 和索引文件 `.MYI` 。

優點：訪問速度快。

缺點：MyISAM 不支持事務和行級鎖，不支持崩潰後的安全恢復，也不支持外鍵。

適用場景：對事務完整性沒有要求；表的數據都是只讀的。

MEMORY 存儲引擎

MEMORY 引擎將數據全部放在內存中，訪問速度較快，但是一旦系統崩潰的話，數據都會丟失。

MEMORY 引擎默認使用哈希索引，將鍵的哈希值和指向數據行的指針保存在哈希索引中。

優點：訪問速度較快。

缺點：

1. 哈希索引數據不是按照索引值順序存儲，無法用於排序。

2. 不支持部分索引匹配查找，因為哈希索引是使用索引列的全部內容來計算哈希值的。
3. 只支持等值比較，不支持範圍查詢。
4. 當出現哈希衝突時，存儲引擎需要遍歷鍊錶中所有的行指針，逐行進行比較，直到找到符合條件的行。

ARCHIVE 存儲引擎

ARCHIVE 存儲引擎非常適合存儲大量獨立的、作為歷史記錄的數據。ARCHIVE 提供了壓縮功能，擁有高效的插入速度，但是這種引擎不支持索引，所以查詢性能較差。

MyISAM 和InnoDB 的區別？

1. **是否支持行級鎖**: `MyISAM` 只有表級鎖，而 `InnoDB` 支持行級鎖和表級鎖，默認為行級鎖。
2. **是否支持事務和崩潰後的安全恢復**： `MyISAM` 不提供事務支持，而 `InnoDB` 提供事務支持，具有事務、回滾和崩潰修復能力。
3. **是否支持外鍵：** `MyISAM` 不支持，而 `InnoDB` 支持。
4. **是否支持MVCC**： `MyISAM` 不支持， `InnoDB` 支持。應對高並發事務，MVCC 比單純的加鎖更高效。
5. `MyISAM` 不支持聚集索引， `InnoDB` 支持聚集索引。

MVCC 實現原理？

MVCC(`Multiversion concurrency control`) 就是同一份數據保留多版本的一種方式，進而實現並發控制。在查詢的時候，通過 `read view` 和版本鏈找到對應版本的數據。

作用：提升並發性能。對於高並發場景，MVCC 比行級鎖開銷更小。

MVCC 實現原理如下：

MVCC 的實現依賴於版本鏈，版本鍊是通過表的三個隱藏字段實現。

- `DB_TRX_ID` ：當前事務id，通過事務id 的大小判斷事務的時間順序。

- DB_ROLL_PRT ：回滾指針，指向當前行記錄的上一個版本，通過這個指針將數據的多個版本連接在一起構成 undo log 版本鏈。
- DB_ROLL_ID ：主鍵，如果數據表沒有主鍵，InnoDB 會自動生成主鍵。

每條表記錄大概是這樣的：

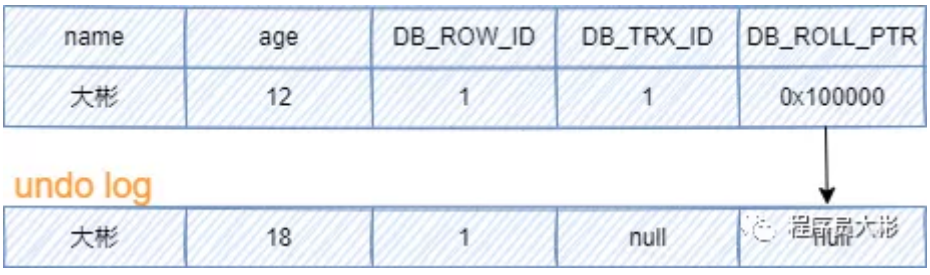
使用事務更新行記錄的時候，就會生成版本鏈，執行過程如下：

1. 用排他鎖鎖住該行；
2. 將該行原本的值拷貝到 undo log ，作為舊版本用於回滾；
3. 修改當前行的值，生成一個新版本，更新事務id，使回滾指針指向舊版本的記錄，這樣就形成一條版本鏈。

下面舉個例子方便大家理解。

1、初始數據如下，其中 DB_ROW_ID 和 DB_ROLL_PTR 為空。

2、事務A對該行數據做了修改，將 age 修改為12，效果如下：



3、之後事務B也對該行記錄做了修改，將 age 修改為8，效果如下：

4、此時undo log 有兩行記錄，並且通過回滾指針連在一起。

接下來了解下read view 的概念。

`read view` 可以理解成將數據在每個時刻的狀態拍成“照片”記錄下來。在獲取某時刻t 的數據時，到t 時間點拍的“照片”上取數據。

在 `read view` 內部維護一個活躍事務鍊錶，表示生成 `read view` 的時候還在活躍的事務。這個鍊錶包含在創建 `read view` 之前還未提交的事務，不包含創建 `read view` 之後提交的事務。

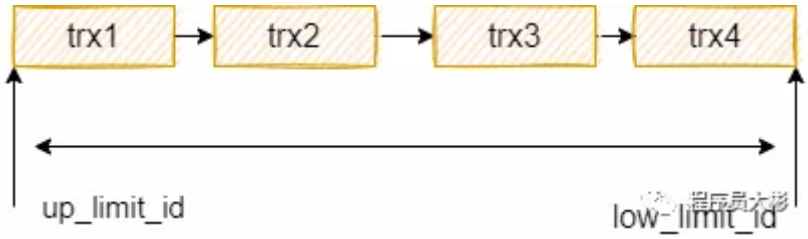
不同隔離級別創建read view 的時機不同。

- read committed：每次執行select 都會創建新的read_view，保證能讀取到其他事務已經提交的修改。

- repeatable read：在一個事務範圍內，第一次select 時更新這個read_view，以後不會再更新，後續所有的select 都是複用之前的read_view。這樣可以保證事務範圍內每次讀取的內容都一樣，即可重複讀。

read view 的記錄篩選方式

前提： `DATA_TRX_ID` 表示每個數據行的最新的事務ID； `up_limit_id` 表示當前快照中的最先開始的事務； `low_limit_id` 表示當前快照中的最慢開始的事務，即最後一個事務。



- 如果 `DATA_TRX_ID < up_limit_id` ：說明在創建 `read view` 時，修改該數據行的事務已提交，該版本的記錄可被當前事務讀取到。
- 如果 `DATA_TRX_ID >= low_limit_id` ：說明當前版本的記錄的事務是在創建 `read view` 之後生成的，該版本的數據行不可以被當前事務訪問。此時需要通過版本鏈找到上一個版本，然後重新判斷該版本的記錄對當前事務的可見性。
- 如果 `up_limit_id <= DATA_TRX_ID < low_limit_i` ：

1. 需要在活躍事務鍊錶中查找是否存在ID 為 DATA_TRX_ID 的值的事務。
2. 如果存在，因為在活躍事務鍊錶中的事務是未提交的，所以該記錄是不可見的。此時需要通過版本鏈找到上一個版本，然後重新判斷該版本的可見性。
3. 如果不存在，說明事務trx_id 已經提交了，這行記錄是可見的。

總結：InnoDB 的 MVCC 是通過 read view 和版本鏈實現的，版本鏈保存有歷史版本記錄，通過 read view 判斷當前版本的數據是否可見，如果不可見，再從版本鏈中找到上一個版本，繼續進行判斷，直到找到一個可見的版本。

快照讀和當前讀

表記錄有兩種讀取方式。

- 快照讀：讀取的是快照版本。普通的 SELECT 就是快照讀。通過mvcc 來進行並發控制的，不用加鎖。
- 當前讀：讀取的是最新版本。 UPDATE、DELETE、INSERT、SELECT … LOCK IN SHARE MODE、SELECT … FOR UPDATE 是當前讀。

快照讀情況下，InnoDB 通過 mvcc 機制避免了幻讀現象。而 mvcc 機制無法避免當前讀情況下出現的幻讀現象。因為當前讀每次讀取的都是最新數據，這時如果兩次查詢中間有其它事務插入數據，就會產生幻讀。

下面舉個例子說明下：

1、首先，user 表只有兩條記錄，具體如下：

2、事務a 和事務b 同時開啟事務 start transaction ；

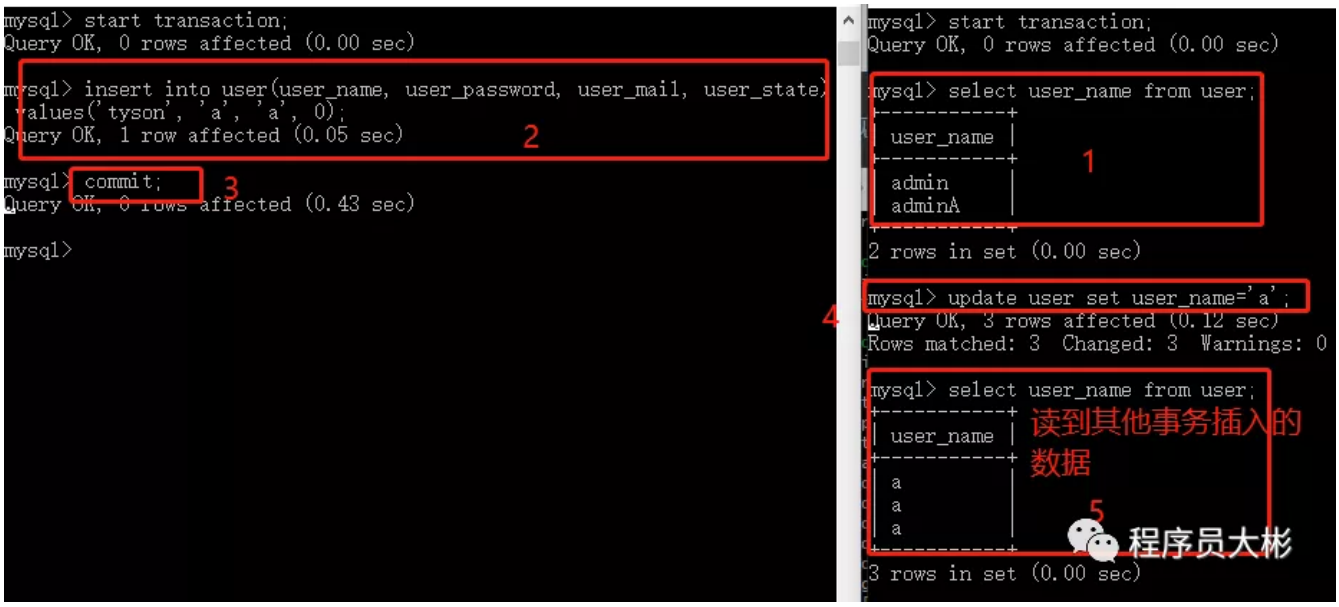
3、事務a 插入數據然後提交；

```
insert into user(user_name, user_password, user_mail, user_state) values('tyson', 'a', 'a', 0);
```

4、事務b 執行全表的update；

```
update user set user_name = 'a';
```

5、事務b 然後執行查詢，查到了事務a 中插入的數據。（下圖左邊是事務b，右邊是事務a。事務開始之前只有兩條記錄，事務a 插入一條數據之後，事務b 查詢出來是三條數據。）



以上就是當前讀出現的幻讀現象。

那麼MySQL 是如何避免幻讀？

- 在快照讀情況下，MySQL 通過 `mvcc` 來避免幻讀。
- 在當前讀情況下，MySQL 通過 `next-key` 來避免幻讀（加行鎖和間隙鎖來實現的）。

next-key 包括兩部分：行鎖和間隙鎖。行鎖是加在索引上的鎖，間隙鎖是加在索引之間的。

`Serializable` 隔離級別也可以避免幻讀，會鎖住整張表，並發性極低，一般不會使用。

共享鎖和排他鎖

SELECT 的讀取鎖定主要分為兩種方式：共享鎖和排他鎖。

```
select * from table where id<6 lock in share mode;--共享锁
select * from table where id<6 for update;--排他锁
```

這兩種方式主要的不同在於 `LOCK IN SHARE MODE` 多個事務同時更新同一個表單時很容易造成死鎖。

申請排他鎖的前提是，沒有線程對該結果集的任何行數據使用排他鎖或者共享鎖，否則申請會受到阻塞。在進行事務操作時，MySQL 會對查詢結果集的每行數據添加排他鎖，其他線程對這些數據的更改或刪除操作會被阻塞（只能讀操作），直到該語句的事務被 `commit` 語句或 `rollback` 語句結束為止。

`SELECT... FOR UPDATE` 使用注意事項：

1. `for update` 僅適用於innodb，且必須在事務範圍內才能生效。
2. 根據主鍵進行查詢，查詢條件為 `like` 或者不等於，主鍵字段產生**表鎖**。
3. 根據非索引字段進行查詢，會產生**表鎖**。

大表怎麼優化？

某個表有近千萬數據，查詢比較慢，如何優化？

當MySQL 單表記錄數過大時，數據庫的性能會明顯下降，一些常見的優化措施如下：

- 限定數據的範圍。比如：用戶在查詢歷史信息的時候，可以控制在一個月的時間範圍內；
- 讀寫分離：經典的數據庫拆分方案，主庫負責寫，從庫負責讀；
- 通過分庫分錶的方式進行優化，主要有垂直拆分和水平拆分。

bin log / redo log / undo log

MySQL 日誌主要包括查詢日誌、慢查詢日誌、事務日誌、錯誤日誌、二進制日誌等。其中比較重要的是 `bin log`（二進制日誌）和 `redo log`（重做日誌）和 `undo log`（回滾日誌）。

bin log

`bin log` 是MySQL 數據庫級別的文件，記錄對MySQL 數據庫執行修改的所有操作，不會記錄select和show 語句，主要用於恢復數據庫和同步數據庫。

redo log

`redo log` 是innodb 引擎級別，用來記錄innodb 存儲引擎的事務日誌，不管事務是否提交都會記錄下來，用於數據恢復。當數據庫發生故障，innoDB 存儲引擎會使用 `redo log` 恢復到發生故障前的時刻，以此來保證數據的完整性。將參數 `innodb_flush_log_at_tx_commit` 設置為1，那麼在執行commit 時會將 `redo log` 同步寫到磁盤。

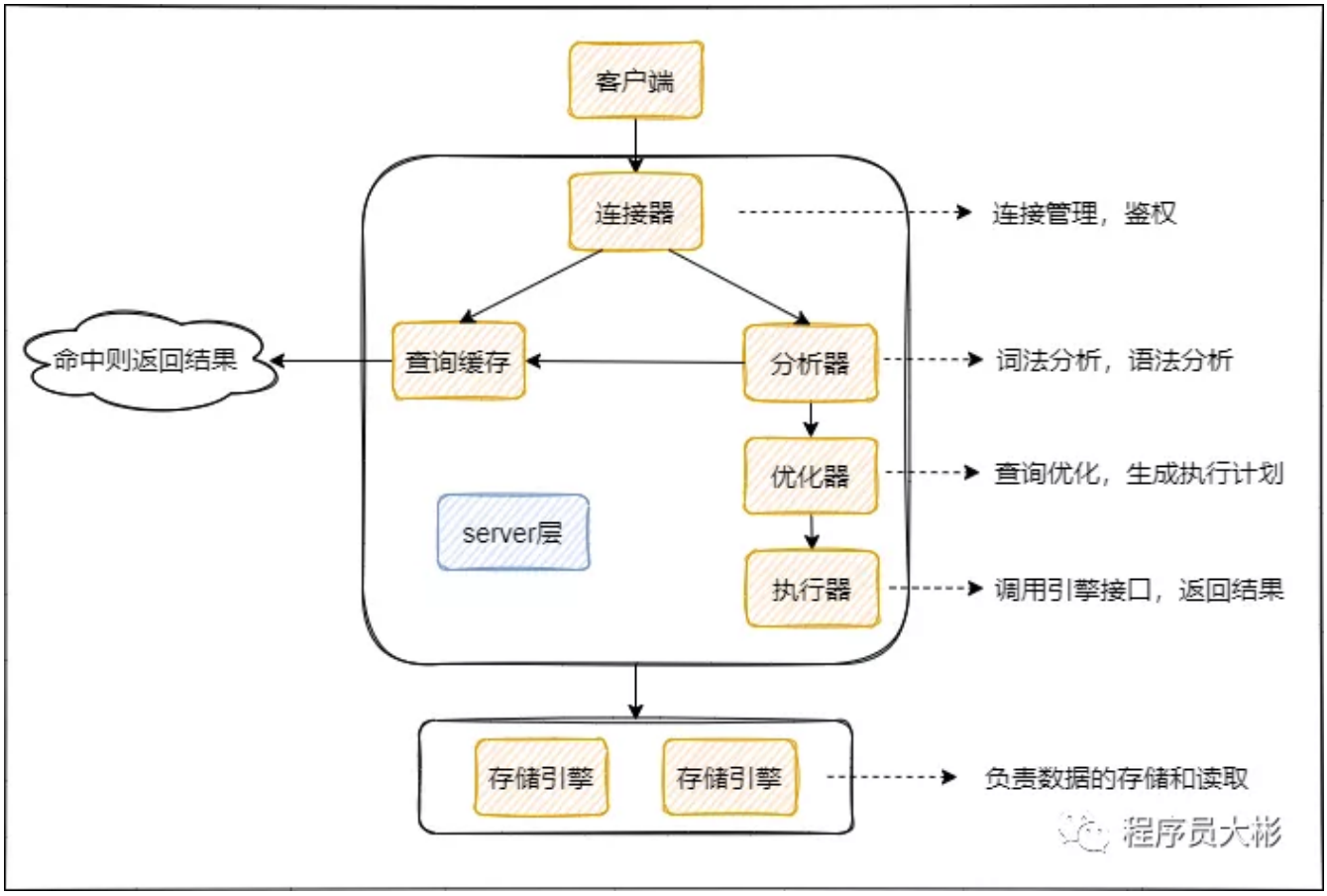
undo log

除了記錄 `redo log` 外，當進行數據修改時還會記錄 `undo log` ， `undo log` 用於數據的撤回操作，它保留了記錄修改前的內容。通過 `undo log` 可以實現事務回滾，並且可以根據 `undo log` 回溯到某個特定的版本的數據，**實現MVCC**。

bin log 和redo log 有什麼區別？

1. `bin log` 會記錄所有日誌記錄，包括InnoDB、MyISAM 等存儲引擎的日誌； `redo log` 只記錄innoDB 自身的事務日誌。
2. `bin log` 只在事務提交前寫入到磁盤，一個事務只寫一次；而在事務進行過程，會有 `redo log` 不斷寫入磁盤。
3. `bin log` 是邏輯日誌，記錄的是SQL 語句的原始邏輯； `redo log` 是物理日誌，記錄的是在某個數據頁上做了什麼修改。

講一下MySQL 架構？



MySQL 主要分為Server 層和存儲引擎層：

- **Server 層**：主要包括連接器、查詢緩存、分析器、優化器、執行器等，所有跨存儲引擎的功能都在這一層實現，比如存儲過程、觸發器、視圖，函數等，還有一個通用的日誌模塊binglog 日誌模塊。

- **存儲引擎**：主要負責數據的存儲和讀取。server 層通過api 與存儲引擎進行通信。

Server 層基本組件

- **連接器：**當客戶端連接MySQL 時，server 層會對其進行身份認證和權限校驗。
- **查詢緩存:**執行查詢語句的時候，會先查詢緩存，先校驗這個sql 是否執行過，如果有緩存這個sql，就會直接返回給客戶端，如果沒有命中，就會執行後續的操作。
- **分析器:**沒有命中緩存的話，SQL 語句就會經過分析器，主要分為兩步，詞法分析和語法分析，先看SQL 語句要做什麼，再檢查SQL 語句語法是否正確。
- **優化器：**優化器對查詢進行優化，包括重寫查詢、決定表的讀寫順序以及選擇合適的索引等，生成執行計劃。
- **執行器：**首先執行前會校驗該用戶有沒有權限，如果沒有權限，就會返回錯誤信息，如果有權限，就會根據執行計劃去調用引擎的接口，返回結果。

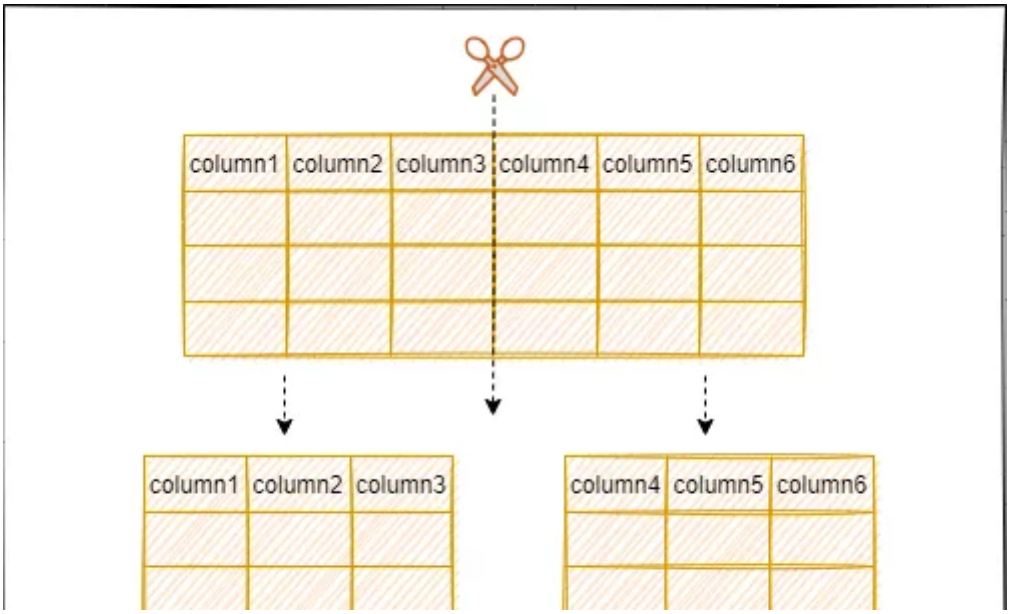
分庫分錶

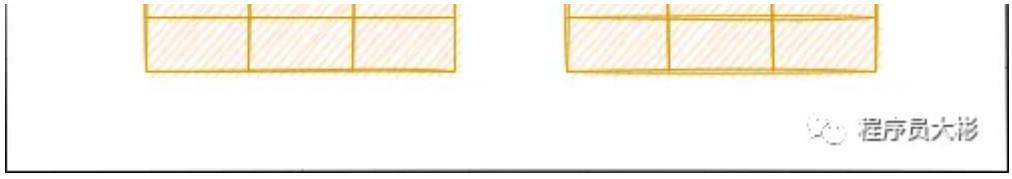
當單表的數據量達到1000W 或100G 以後，優化索引、添加從庫等可能對數據庫性能提升效果不明顯，此時就要考慮對其進行切分了。切分的目的就在於減少數據庫的負擔，縮短查詢的時間。

數據切分可以分為兩種方式：垂直劃分和水平劃分。

垂直劃分

垂直劃分數據庫是根據業務進行劃分，例如購物場景，可以將庫中涉及商品、訂單、用戶的表分別劃分出成一個庫，通過降低單庫的大小來提高性能。同樣的，分錶的情況就是將一個大表根據業務功能拆分成一個個子表，例如商品基本信息和商品描述，商品基本信息一般會展示在商品列表，商品描述會展示在商品詳情頁，可以將商品基本信息和商品描述拆分成兩張表。





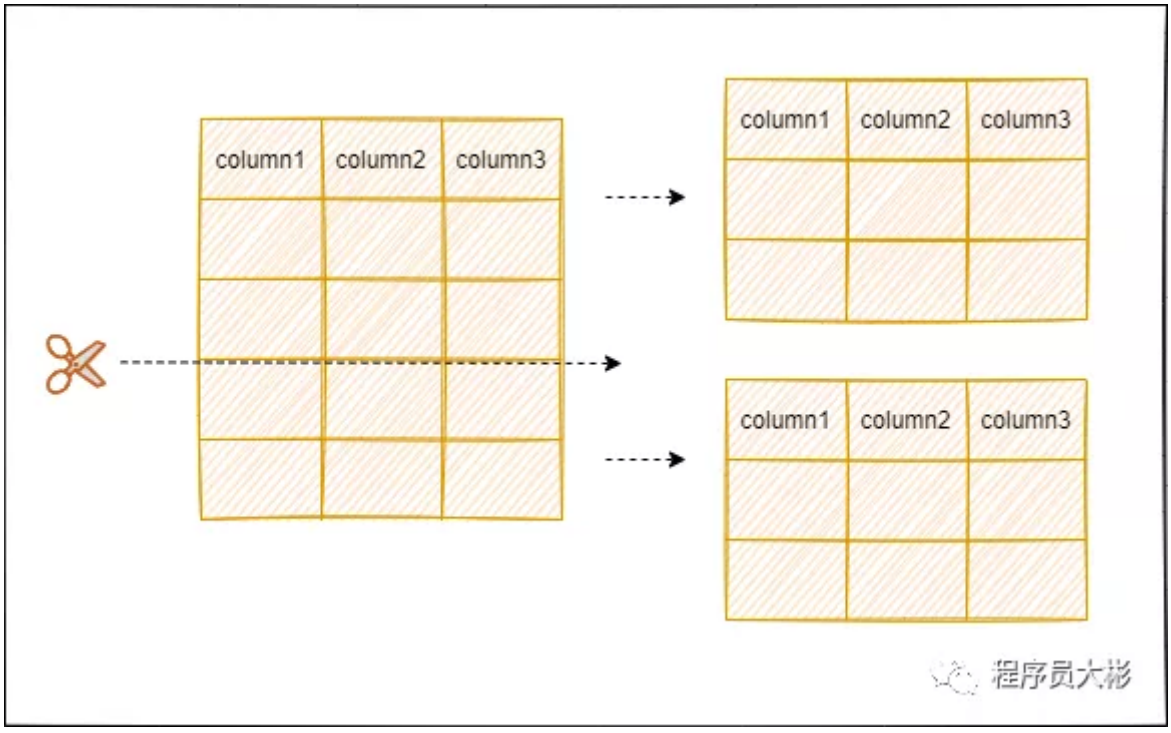
優點：行記錄變小，數據頁可以存放更多記錄，在查詢時減少I/O 次數。

缺點：

- 主鍵出現冗餘，需要管理冗餘列；
- 會引起表連接JOIN 操作，可以通過在業務服務器上進行join 來減少數據庫壓力；
- 依然存在單表數據量過大的問題。

水平劃分

水平劃分是根據一定規則，例如時間或id 序列值等進行數據的拆分。比如根據年份來拆分不同的數據庫。每個數據庫結構一致，但是數據得以拆分，從而提升性能。



優點：單庫（表）的數據量得以減少，提高性能；切分出的表結構相同，程序改動較少。

缺點：

- 分片事務一致性難以解決
- 跨節點 `join` 性能差，邏輯複雜
- 數據分片在擴容時需要遷移

什麼是分區表？

分區表是一個獨立的邏輯表，但是底層由多個物理子表組成。

當查詢條件的數據分佈在某一個分區的時候，查詢引擎只會去某一個分區查詢，而不是遍歷整個表。在管理層面，如果需要刪除某一個分區的數據，只需要刪除對應的分區即可。

分區表類型

按照範圍分區。

```
CREATE TABLE test_range_partition(
       id INT auto_increment,
       createdate DATETIME,
       primary key (id,createdate)
   )
   PARTITION BY RANGE (TO_DAYS(createdate) ) (
      PARTITION p201801 VALUES LESS THAN ( TO_DAYS('20180201') ),
      PARTITION p201802 VALUES LESS THAN ( TO_DAYS('20180301') ),
      PARTITION p201803 VALUES LESS THAN ( TO_DAYS('20180401') ),
      PARTITION p201804 VALUES LESS THAN ( TO_DAYS('20180501') ),
      PARTITION p201805 VALUES LESS THAN ( TO_DAYS('20180601') ),
      PARTITION p201806 VALUES LESS THAN ( TO_DAYS('20180701') ),
      PARTITION p201807 VALUES LESS THAN ( TO_DAYS('20180801') ),
      PARTITION p201808 VALUES LESS THAN ( TO_DAYS('20180901') ),
      PARTITION p201809 VALUES LESS THAN ( TO_DAYS('20181001') ),
      PARTITION p201810 VALUES LESS THAN ( TO_DAYS('20181101') ),
      PARTITION p201811 VALUES LESS THAN ( TO_DAYS('20181201') ),
      PARTITION p201812 VALUES LESS THAN ( TO_DAYS('20190101') )
   );
```

在 `/var/lib/mysql/data/` 可以找到對應的數據文件，每個分區表都有一個使用# 分隔命名的表文件：

```
   -rw-r----- 1 MySQL MySQL    65 Mar 14 21:47 db.opt
   -rw-r----- 1 MySQL MySQL  8598 Mar 14 21:50 test_range_partition.frm
   -rw-r----- 1 MySQL MySQL 98304 Mar 14 21:50 test_range_partition#P#p201801.ibd
   -rw-r----- 1 MySQL MySQL 98304 Mar 14 21:50 test_range_partition#P#p201802.ibd
```

```
   -rw-r----- 1 MySQL MySQL 98304 Mar 14 21:50 test_range_partition#P#p201803.ibd
...
```

list 分區

對於 List 分區，分區字段必須是已知的，如果插入的字段不在分區時枚舉值中，將無法插入。

```
create table test_list_partiotion
   (
       id int auto_increment,
       data_type tinyint,
       primary key(id,data_type)
   )partition by list(data_type)
   (
       partition p0 values in (0,1,2,3,4,5,6),
       partition p1 values in (7,8,9,10,11,12),
       partition p2 values in (13,14,15,16,17)
   );
```

hash 分區

可以將數據均勻地分佈到預先定義的分區中。

```
create table test_hash_partiotion
   (
       id int auto_increment,
       create_date datetime,
       primary key(id,create_date)
   )partition by hash(year(create_date)) partitions 10;
```

查詢語句執行流程？

查詢語句的執行流程如下：權限校驗、查詢緩存、分析器、優化器、權限校驗、執行器、引擎。

舉個例子，查詢語句如下：

```
select * from user where id > 1 and name = '大彬';
```

1. 首先檢查權限，沒有權限則返回錯誤；
2. MySQL8.0 以前會查詢緩存，緩存命中則直接返回，沒有則執行下一步；
3. 詞法分析和語法分析。提取表名、查詢條件，檢查語法是否有錯誤；
4. 兩種執行方案，先查 `id > 1` 還是 `name = '大彬'` ，優化器根據自己的優化算法選擇執行效率最好的方案；
5. 校驗權限，有權限就調用數據庫引擎接口，返回引擎的執行結果。

更新語句執行過程？

更新語句執行流程如下：分析器、權限校驗、執行器、引擎、 `redo log` （ `prepare` 狀態）、 `binlog` 、 `redo log` （ `commit` 狀態）。

舉個例子，更新語句如下：

```
update user set name = '大彬' where id = 1;
```

1. 先查詢到id 為1 的記錄，有緩存會使用緩存。
2. 拿到查詢結果，將name 更新為大彬，然後調用引擎接口，寫入更新數據，innodb 引擎將數據保存在內存中，同時記錄 `redo log` ，此時 `redo log` 進入 `prepare` 狀態。
3. 執行器收到通知後記錄 `binlog` ，然後調用引擎接口，提交 `redo log` 為 `commit` 狀態。
4. 更新完成。

為什麼記錄完 `redo log` ，不直接提交，而是先進入 `prepare` 狀態？

假設先寫 `redo log` 直接提交，然後寫 `binlog` ，寫完 `redo log` 後，機器掛了， `binlog` 日誌沒有被寫入，那麼機器重啟後，這台機器會通過 `redo log` 恢復數據，但是這個時候 `binlog` 並沒有記錄該數據，後續進行機器備份的時候，就會丟失這一條數據，同時主從同步也會丟失這一條數據。

exist 和in 的區別？

`exists` 用於對外表記錄做篩選。 `exists` 會遍歷外表，將外查詢表的每一行，代入內查詢進行判斷。當 `exists` 裡的條件語句能夠返回記錄行時，條件就為真，返回外表當前記錄。反之如果 `exists` 裡的條件語句不能返回記錄行，條件為假，則外表當前記錄被丟棄。

```
select a.* from A awhere exists(select 1 from B b where a.id=b.id)
```

in 是先把後邊的語句查出來放到臨時表中，然後遍歷臨時表，將臨時表的每一行，代入外查詢去查找。

```
select * from Awhere id in(select id from B)
```

子查詢的表比較大的時候，使用 exists 可以有效減少總的循環次數來提升速度；**當外查詢的表比較大的時候**，使用 in 可以有效減少對外查詢表循環遍歷來提升速度。

truncate、delete 與drop 區別？

相同點：

1. truncate 和不帶 where 子句的 delete 、以及 drop 都會刪除表內的數據。
2. drop 、 truncate 都是 DDL 語句（數據定義語言），執行後會自動提交。

不同點：

1. truncate 和delete 只刪除數據不刪除表的結構；drop 語句將刪除表的結構被依賴的約束、觸發器、索引；
2. 一般來說，執行速度: drop > truncate > delete。

having 和where 的區別？

- 二者作用的對像不同， where 子句作用於表和視圖， having 作用於組。
- where 在數據分組前進行過濾， having 在數據分組後進行過濾。

什麼是MySQL 主從同步？

主從同步使得數據可以從一個數據庫服務器複製到其他服務器上，在復制數據時，一個服務器充當主服務器（ master ），其餘的服務器充當從服務器（ slave ）。

因為複制是異步進行的，所以從服務器不需要一直連接著主服務器，從服務器甚至可以通過撥號斷斷續續地連接主服務器。通過配置文件，可以指定複製所有的數據庫，某個數據庫，甚至是某個數據庫上的某個表。

為什麼要做主從同步？

1. 讀寫分離，使數據庫能支撐更大的並發。
2. 在主服務器上生成實時數據，而在從服務器上分析這些數據，從而提高主服務器的性能。
3. 數據備份，保證數據的安全。

樂觀鎖和悲觀鎖是什麼？

數據庫中的並發控制是確保在多個事務同時存取數據庫中同一數據時不破壞事務的隔離性和統一性以及數據庫的統一性。樂觀鎖和悲觀鎖是並發控制主要採用的技術手段。

- 悲觀鎖：假定會發生並發衝突，在查詢完數據的時候就把事務鎖起來，直到提交事務。實現方式：使用數據庫中的鎖機制。
- 樂觀鎖：假設不會發生並發衝突，只在提交操作時檢查是否數據是否被修改過。給表增加 `version` 字段，在修改提交之前檢查 `version` 與原來取到的 `version` 值是否相等，若相等，表示數據沒有被修改，可以更新，否則，數據為臟數據，不能更新。實現方式：樂觀鎖一般使用版本號機製或 `CAS` 算法實現。

用過processlist 嗎？

`show processlist` 或 `show full processlist` 可以查看當前MySQL 是否有壓力，正在運行的 `SQL` ，有沒有慢 `SQL` 正在執行。返回參數如下：

1. **id**：線程ID，可以用 `kill id` 殺死某個線程
2. **db**：數據庫名稱
3. **user**：數據庫用戶
4. **host**：數據庫實例的IP
5. **command**：當前執行的命令，比如 `Sleep` ， `Query` ， `Connect` 等
6. **time**：消耗時間，單位秒
7. **state**：執行狀態，主要有以下狀態：
    - `Sleep` ，線程正在等待客戶端發送新的請求
    - `Locked` ，線程正在等待鎖
    - `Sending data` ，正在處理 `SELECT` 查詢的記錄，同時把結果發送給客戶端
    - `Kill` ，正在執行 `kill` 語句，殺死指定線程
    - `Connect` ，一個從節點連上了主節點
    - `Quit` ，線程正在退出
    - `Sorting for group` ，正在為 `GROUP BY` 做排序
    - `Sorting for order` ，正在為 `ORDER BY` 做排序

8. **info**：正在執行的 `SQL` 語句

- END -

喜歡此內容的人還喜歡

搞懂MySQL Explain 命令之前不要說自己會SQL優化

小林coding

MySQL數據查詢太多會OOM嗎?

業餘草

MySQL 中的反斜杠\\，真是太坑了！！

Java面試那些事兒