# 目录

[目录](#4700-1610101838336)

[附加知识](#2980-1610344633730)

[笛卡尔乘积（笛卡尔积）](#7076-1610157610953)

[mysql加载顺序](#3027-1610103059258)

[sql的7种join](#9560-1610103305861)

[什么是索引](#5220-1610177152912)

[B-Tree索引](#8664-1610170963087)

[MySQL Optimizer](#6998-1610175066797)

[MySQL常见瓶颈](#5358-1610175203300)

[Explain](#8213-1610175241799)

[join 优化](#5035-1610181984798)

[如何避免索引失效](#3914-1610351736037)

[mysql优化实例](#5380-1610351694152)

# 附加知识

## 笛卡尔乘积（笛卡尔积）

例如，A={a,b}, B={0,1,2}，

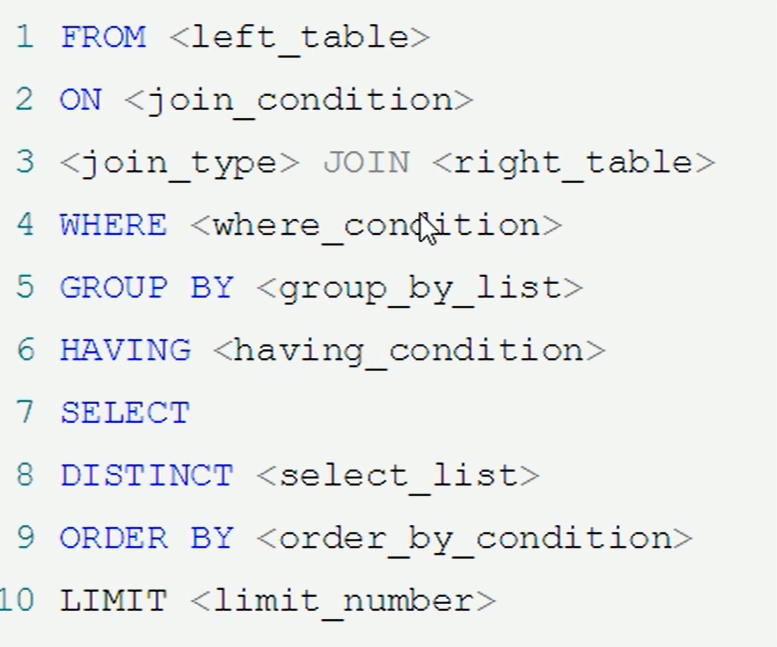
则：

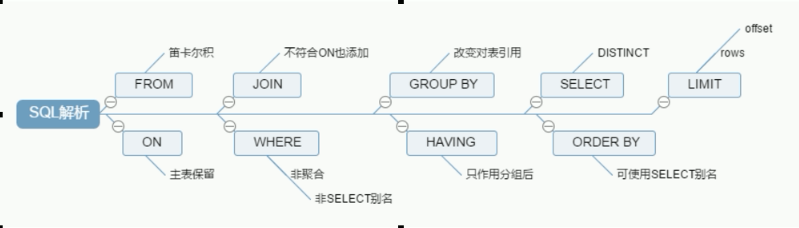
A×B={(a, 0), (a, 1), (a, 2), (b, 0), (b, 1), (b, 2)}

B×A={(0, a), (0, b), (1, a), (1, b), (2, a), (2, b)}

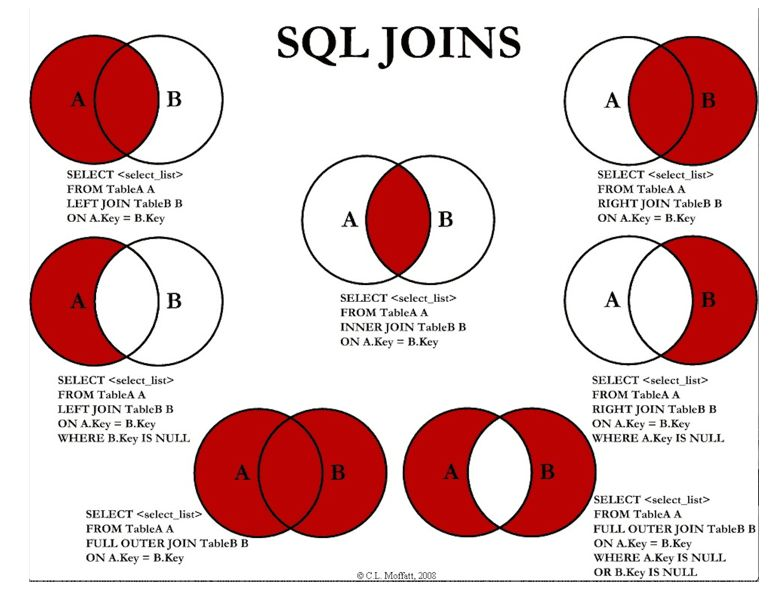
## mysql加载顺序

mysql执行顺序，是先从from开始读





## sql的7种join



# 什么是索引

Mysql官方定义：索引（index）是帮助mysql高效获取数据的数据结构。

**索引本质**: **索引是数据结构， 排好序的快速查找数据结构。**

**结论**：数据本身之外,数据库还维护着一个满足特定查找算法的数据结构，这些数据结构以某种方式指向数据，这样就可以在这些数据结构的基础上实现高级查找算法,这种数据结构就是索引。

我们平时所说的索引，如果没有特别指明，都是指B树(多路搜索树，并不一定是二叉树)结构组织的索引。其中聚集索引，次要索引，覆盖索引，复合索引，前缀索引，唯一索引默认都是使用B+树索引，统称索引。当然,除了B+树这种类型的索引之外，还有哈希索引(hash index)等。

**索引的优势/劣势**

**优势**：

* 类似大学图书馆建书目索引，提高数据检索效率，降低数据库的IO成本。
* 通过索引列对数据进行排序，降低数据排序成本，降低了CPU的消耗。

**劣势**：

* 虽然索引大大提高了查询速度，同时却会降低更新表的速度,如果对表INSERT,UPDATE和DELETE。因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件每次更新添加了索引列的字段，都会调整因为更新所带来的键值变化后的索引信息。
* 索引只是提高效率的一个因素，如果你的MySQL有大数据量的表，就需要花时间研究建立优秀的索引，或优化查询语句。

**索引分类**

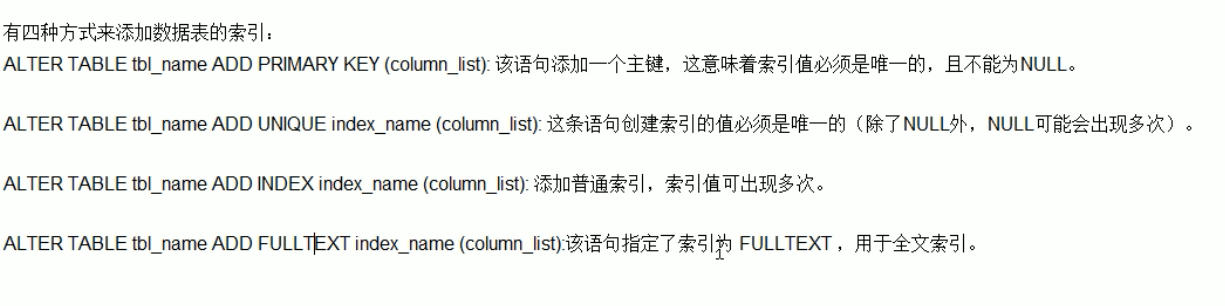
**单值索引：**一个索引只包含单个列，一张表可以有多个单值索引

建议一张表索引不要超过5个 优先考虑复合索引

**唯一索引：**索引列的值必须唯一，但允许有空值

**复合索引：**一个索引包含多个列

添加索引方式：



**索引结构**

## B-Tree索引

**B树：**平衡多路查找树（B-Tree）

特点：

1. 树内的每个节点都存储数据
2. 叶子节点之间无指针连接

**B+树**：

特点：

1. 数据只出现在叶子节点，非叶子节点只存储键值信息。
2. 所有叶子节点增加了一个链指针

**B-tree和 B+tree的区别**：

1. B树每个节点中不仅包含数据的key值，还有data值。而每一个页的存储空间是有限的，如果data数据较大时将会导致每个节点能存储的key的数量很小，要保存同样多的key，就需要增加树的高度。树的高度每增加一层，查询时的磁盘I/O次数就增加一次，进而影响查询效率。而在B+Tree中，所有数据记录节点都是按照键值大小顺序存放在同一层的叶子节点上，而非叶子节点上只存储key值信息，这样可以大大加大每个节点存储的key值数量，降低B+树的高度。
2. B+树的叶子节点上有指针进行相连，因此在做数据遍历的时候，只需要对叶子节点进行遍历即可，这个特性使得B+树非常适合做范围查询

参考资料：*https://blog.csdn.net/mu\_wind/article/details/110128016*

**hash索引**

**full-text全文索引**

**r-Tree索引**

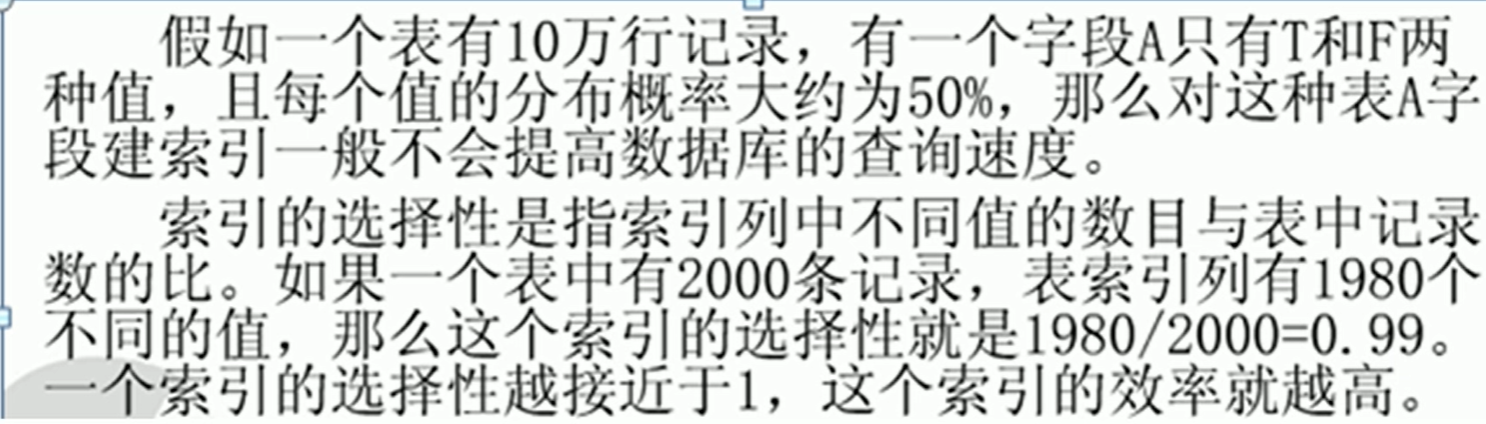
**索引使用场景**

**哪些情况下需要创建索引**：

主键自动建立唯一索引 频繁作为查询条件的字段应该创建索引 查询中与其他表关联的字段，外键关系建立索引 频繁更新的字段不适合创建索引，因为每次更新不单单是更新记录，还需要更新索引，会加重IO负担 where条件里面用不到的字段不创建索引 单间/组合索引的选择问题，who？（在高并发下倾向创建组合索引） 查询中排序的字段，排序字段若通过索引去访问将大大提高排序速度 查询中统计或者分组字段

**哪些情况下不要创建索引**：

表记录太少 经常增删改的表 数据重复且分布平均的表字段，因为应该只为经常查询和经常排序的字段建立索引，注意：如果某个字段包含许多重复内容，为它建立索引就没有太大的实际效果



**性能分析**

## MySQL Optimizer

MySQL Optimizer是一个专门负责优化SELECT 语句的优化器模块，它主要的功能就是通过计算分析系统中收集的各种统计信息，为客户端请求的Query 给出他认为最优的执行计划，也就是他认为最优的数据检索方式。

## MySQL常见瓶颈

1. CPU: CPU在饱和的时候一般发生在数据装入在内存或从磁盘上读取数据时候。
2. IO: 磁盘I/O瓶颈发生在装入数据远大于内存容量时。
3. 服务器硬件的性能瓶颈：top,free,iostat和vmstat来查看系统的性能状态。

## Explain

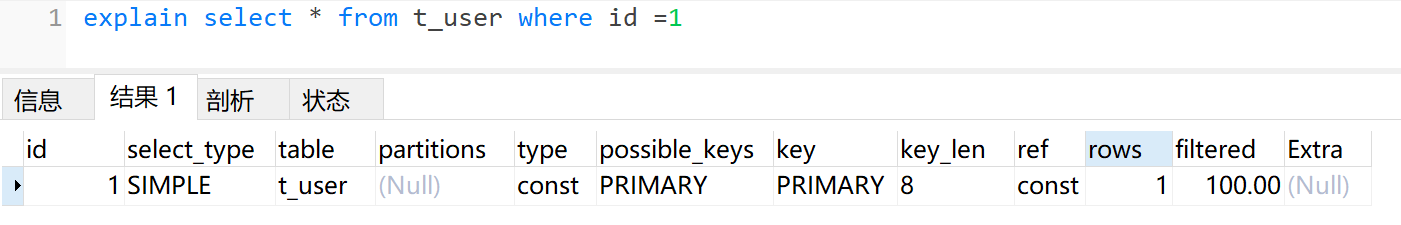
**概念**：使用EXPLAIN关键字可以模拟优化器执行SQL语句，从而知道MySQL是

如何处理你的SQL语句的。分析你的查询语句或是结构的性能瓶颈。

通过explain我们可以获得以下信息：

* 表的读取顺序
* 数据读取操作的操作类型
* 哪些索引可以使用
* 哪些索引被实际使用
* 表之间的引用
* 每张表有多少行被优化器查询

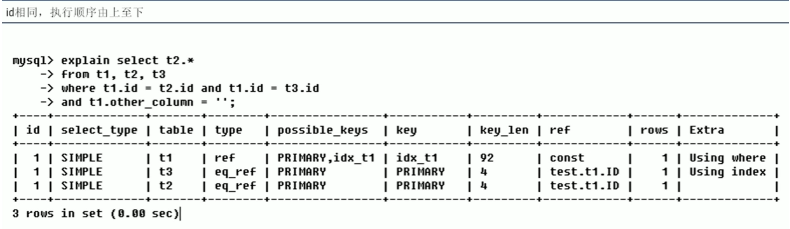
**explain使用**：



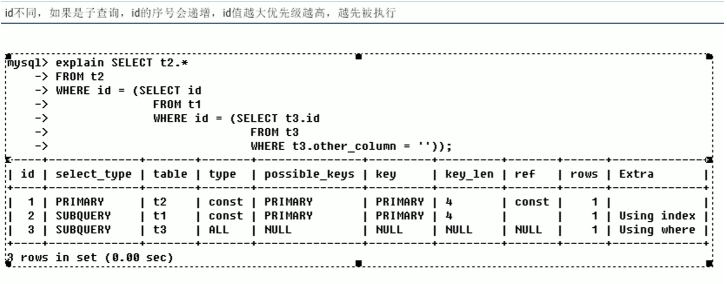
**执行计划各字段含义**

**id：**查询的序号，包含一组数字，表示查询中执行select子句或操作表的顺序。

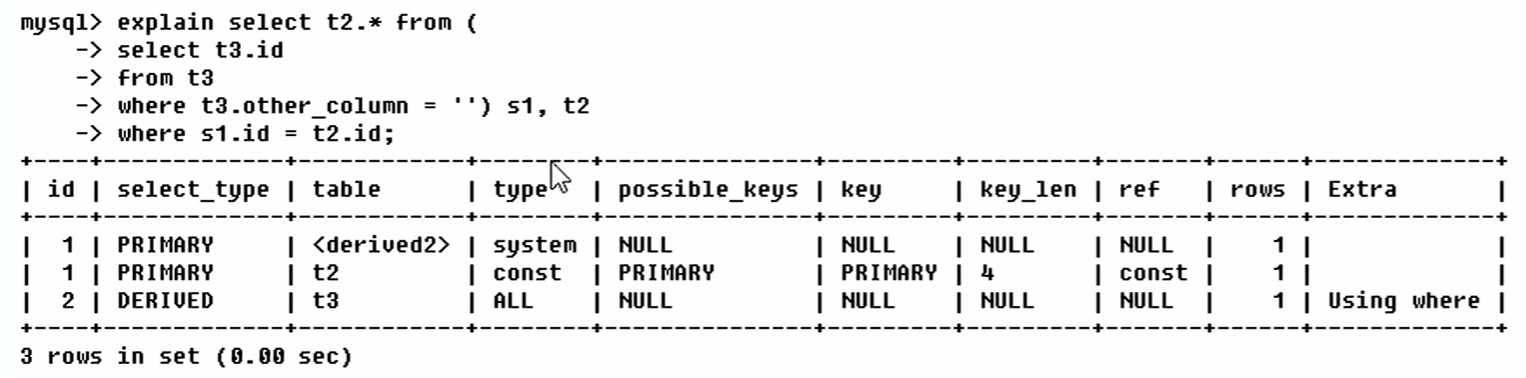
* id相同，执行顺序由上至下 。



* id不同，如果是子查询，id的序号会递增，id值越大优先级越高，越先被执行。



* id相同不同，同时存在  id相同的可以认为是一组，同一组中从上往下执行，所有组中id大的优先执行。



衍生：derived2，表示s1是由select\_type=2的表衍生来的

**select\_type：**查询类型(查询中每个select子句的类型)，主要用于区别普通查询，联合查询，子查询等的复杂查询。

simple ——简单的select查询，查询中不包含子查询或者UNION. primary ——查询中若包含任何复杂的子部分，最外层查询被标记. subquery ——在select或where列表中包含了子查询. derived ——在from列表中包含的子查询被标记为derived（衍生），MySQL会递归执行这些子查询，把结果放到临时表中. union ——如果第二个select出现在UNION之后，则被标记为UNION，如果union包含在from子句的子查询中，外层select被标记为derived. union result ——UNION 的结果.

**table:** 输出的行所引用的表，显示这一行的数据是关于哪张表的

**type：** MySQL在表中找到所需行的方式，又称“访问类型”, type包含的类型包括以下几种，从好到差依次是 **system > const > eq\_ref > ref > range > index > all**

system：表只有一行记录（等于系统表），这是const类型的特列，平时不会出现，这个也可以忽略不计 const：表示通过索引一次就找到了，const用于比较primary key 或者unique索引。因为只匹配一行数据，所以很快。如将主键置于where列表中，MySQL就能将该查询转换为一个常量。  eq\_ref：唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见于主键或唯一索引扫描 ref：非唯一性索引扫描，返回匹配某个单独值的所有行，本质上也是一种索引访问，它返回所有匹配某个单独值的行，然而，它可能会找到多个符合条件的行，所以他应该属于查找和扫描的混合体。  range：只检索给定范围的行，使用一个索引来选择行，key列显示使用了哪个索引，一般就是在你的where语句中出现between、< 、>、in等的查询，这种范围扫描索引比全表扫描要好，因为它只需要开始于索引的某一点，而结束于另一点，不用扫描全部索引。  index：Full Index Scan，Index与All区别为index类型只遍历索引树。这通常比ALL快，因为索引文件通常比数据文件小。（也就是说虽然all和Index都是读全表，但index是从索引中读取的，而all是从硬盘读取的）  all：Full Table Scan 将遍历全表以找到匹配的行（全表扫描）

一般来说，得保证查询至少达到range级别，最好能达到ref。

**possible\_keys ：** 指出能使用在这张表上的索引，一个或多个，查询涉及的字段上若存在索引，则该索引将被列出（但不一定被查询实际使用）。

**key ：** 指实际使用的索引，如果为NULL，则没有使用索引。（可能原因包括没有建立索引或索引失效）。

**key\_len ：**表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度，在不损失精确性的情况下，长度越短越好。

**rows ：** 根据表统计信息及索引选用情况，大致估算出找到所需的记录所需要读取的行数，也就是说，用的越少越好 。

**ref ：** 显示索引的哪一列被使用了，如果有可能的话，是一个常数，哪些列或常量被用于查询索引列上的值。

**rows ：** 根据表统计信息以及索引选用情况，大致估算出找到所需的记录所需要读取的行数

**Extra ：**包含不适合在其他列中显示但十分重要的额外信息

Using filesort：说明mysql会对数据适用一个外部的索引排序。而不是按照表内的索引顺序进行读取。MySQL中无法利用索引完成排序操作称为“文件排序” Using temporary: 使用了临时表保存中间结果，mysql在查询结果排序时使用临时表。常见于排序order by和分组查询group by。 Using index: 表示相应的select操作用使用覆盖索引，避免访问了表的数据行。如果同时出现using where，表名索引被用来执行索引键值的查找；如果没有同时出现using where，表名索引用来读取数据而非执行查询动作。 Using where : 表明使用where过滤 using join buffer: 使用了连接缓存 impossible where: where子句的值总是false，不能用来获取任何元组 select tables optimized away：在没有group by子句的情况下，基于索引优化Min、max操作或者对于MyISAM存储引擎优化count（\*），不必等到执行阶段再进行计算，查询执行计划生成的阶段即完成优化。 distinct：优化distinct操作，在找到第一匹配的元组后即停止找同样值的动作。

# join 优化

1. 左连接的特性决定左表全都查，加不加索引都要全表扫描，右表才是用于确定搜索行。所以left join需要将索引加到右表才有优化效果。
2. 右连接与左连接同理。
3. join语句 针对三表联查同理。
4. **join语句的优化**：

* 尽可能检索Join语句中的嵌套循环的循环总次数，“永远用小结果集驱动大结果集”。
* 优先优化嵌套循环的内层循环
* 保证join语句中被驱动表上join条件字段已经被索引

# 如何避免索引失效

**1. 全值匹配我最爱**。

2. 最佳左前缀法则（**带头大哥不能死，中间兄弟不能断**）。

3. **索引列上少计算**，不要在索引上做任何操作（计算、函数、自动/手动类型转换），不然会导致索引失效而转向全表扫描。

4. **范围之后全失效，**存储引擎不能继续使用索引中范围条件（bettween、<、>、in等）右边的列。

5. **覆盖索引不写星，**尽量使用覆盖索引（只查询索引的列（索引列和查询列一致）），减少select \*。

6. 索引字段上使用（！= 或者 < >）判断时，会导致索引失效而转向全表扫描。

7. 索引字段上使用 is null / is not null 判断时，会导致索引失效而转向全表扫描。

8. **Like百分写最右**，索引字段使用like以通配符开头（‘%字符串’）时，会导致索引失效而转向全表扫描。（扩展：如何在like %内容% 的时候使索引生效？ 答：覆盖索引，只查索引字段即可使索引生效）

9. varchar字段查询时不加单引号，会导致索引失效而转向全表扫描

10. 索引字段使用 or 时，会导致索引失效而转向全表扫描

**优化口诀：**

全值匹配我最爱，最左前缀要遵守；

带头大哥不能死，中间兄弟不能断；

索引列上少计算，范围之后全失效；

Like百分写最右，覆盖索引不写星；

不等空值还有or，索引失效要少用；

VAR引号不可丢，SQL高级也不难！

# mysql优化实例

1. mysql优化器，会自动调整and连接的字段顺序。达到最优效果。但是建议使用的时候按照索引顺序，避免mysql底层翻译和转换。

2. 分组之前必排序，group by 和order by原则一样。如果顺序错乱，会导致临时表的产生

# 