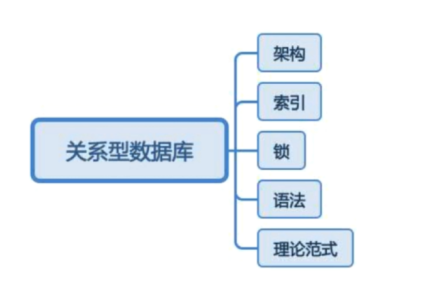
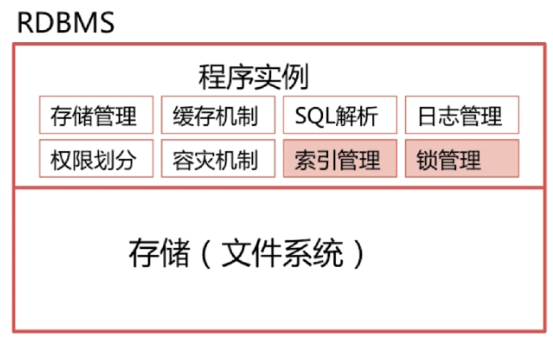
**数据库部分**

**关系型数据库的主要考点：**



## 一、如何设计关系型数据库：

**RDBMS（即关系数据库管理系统）**



**1.程序实例部分：**

1、存储管理：数据逻辑关系转换为物理存储关系

2、缓存机制：优化执行效率

3、SQL解析：将SQL语句进行解析

4、日志管理：记录操作

5、容灾机制：备份

6、权限划分：多用户管理

7、索引模块：提高数据查询效率

8、锁机制：使数据支持并发

**2.存储系统：**

类似文件系统，将数据写入其中

## 索引模块

### 索引的数据结构

Hash结构，二叉搜索树，红黑树，B树，B+树

###### Hash结构：

通过计算得到Hash值，然后得到数据的存放位置，例如JAVA中的HashMap采用就是这种数据索引结构。

优点：

检索时间快，平均检索时间为O(1)。

缺点：

①因为Hash值是通过算法计算出来的，存在Hash碰撞的几率，比如HashMap对于Hash值相同的数据，会在Hash值所在桶创建一个链表，用于存放相同Hash值的数据。

在数据量很大的情况下，内存无法加载全部的数据索引。

###### 二叉搜索树：

定义规则为“左边节点值比根节点小，右边节点值比根节点大，并且左右子节点都是排序树，没有键值相等的节点”。

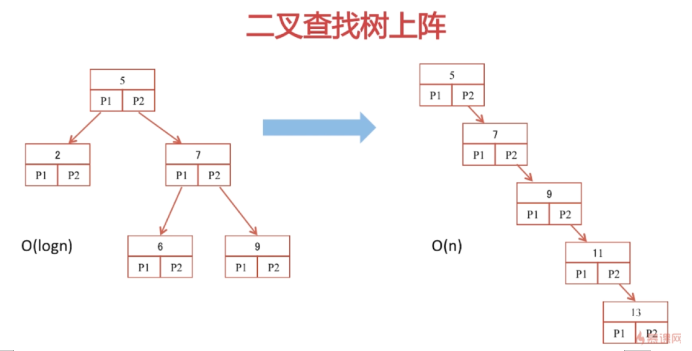
优点：

可以解决大量数据索引无法一次加载进内存中的问题，二叉搜索树可以批量加载数据进 内存。

缺点：

①检索时间与树的高度有关，树的高度越高，检索次数及时间相对就会越久。

②极端情况下，如果数据本身就是有序的，二叉搜索树会退化成链表，性能会急剧降低。



|  |
| --- |
| 备注：  平衡二叉树  平衡二叉搜索树，又被称为AVL树，且具有以下性质：它是一棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树。  拓展：<https://blog.csdn.net/qq_25940921/article/details/82183093> |

###### 红黑树：

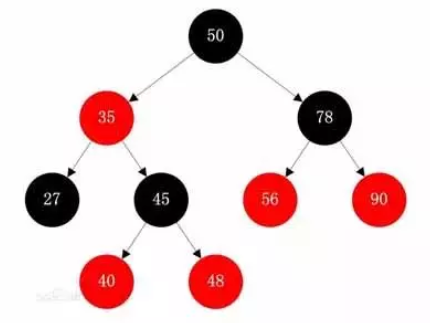
红黑树是一种自平衡二叉树，主要解决二叉搜索树在极端情况下退化成链表的情况，在数据插入的时候同时调整整个树，使其节点尽量均匀分布，保证平衡性，目的在于降低树的高度，提高查询效率。

优点：

解决二叉搜索树的极端情况的退化问题。

缺点：

检索时间依旧与树的高度有关，当数据量很大时，树的高度就会很高，检索的次数就会比较多，检索的时间会比较久，效率低。



备注：红黑树可以保证树的平衡

###### B树：

定义：

①根节点至少包括两个孩子。

②书中每个节点最多包含有m个孩子（m>=2）。

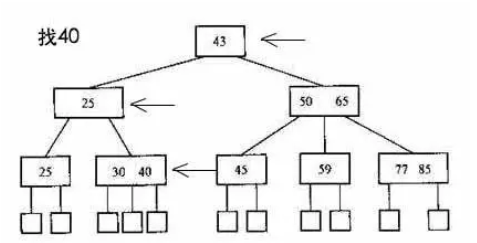
③除根节点和叶子节点外，其他每个节点至少有ceil（m/2）个孩子。

④所有叶子节点都位于同一层。

B树是一种多路搜索树，每个子节点可以拥有多于2个子节点，M路的B树最多可拥有M个子节点。设计成多路，其目的是为了降低树的高度，降低查询次数，提高查询效率。

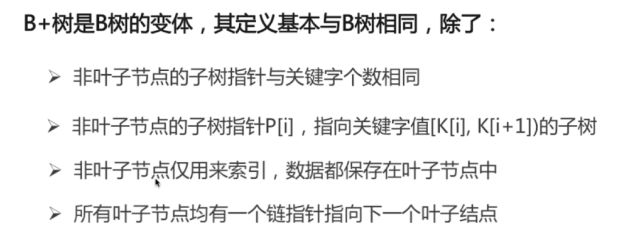
虽然多路可以降低树的高度，但是如果设计成无限多路，就会退化成有序数组，一般B树的使用场景是用于文件的索引，这些索引会存放于硬盘中，有时内存是无法一次性加载完，此时就无法进行查找。

如果全部在内存中，红黑树的查找效率要高于B树，但是涉及到磁盘操作，B树要优于红黑树，所以在JDK1.8版本的HashMap中，如果单个桶的链表长度多于8或全部桶的链表总长度多余64，会将链表转换成红黑树。



备注：查找的时候，每次载入一个节点进入内存就行

###### B+树：



B+树：B+树是对于B树进行优化的多路搜索树，主要设计是将数据全部存放于叶子节点，并将叶子节点用指针进行链表链接。

主要使用场景：常用于数据库的索引。

数据库的索引一般数据量不小，同时又存放于磁盘中，采用多路搜索树，可以降低树的高度，同时在大数据量下可以分批载入内存，提高查询效率。

不同于B树的使用场景，数据库的查询中，我们一般查询的数量不会是单条数据，例如列表常用查询中的分页查询--查询第1页的10条数据，此时如果采用B树，需要进行树的中序遍历，可能需要跨层访问。

而B+树的所有数据全部存放于叶子节点上，且叶子节点之间采用指针进行链表链接，一次查询多条时，确定首尾位置，便可以方便的确定多条数据位置。

|  |
| --- |
| **结论：**   1. **B+树的磁盘读写代价更低** 2. **B+树的查询效率更稳定** 3. **B+树更有利于对数据库的扫描** |

## Mysql数据库引擎

### 1.MyISAM存储引擎

不支持事务、也不支持外键，优势是访问速度快，对事务完整性没有要求或者以select，insert为主的应用基本上可以用这个引擎来创建表

支持3种不同的存储格式，分别是：静态表；动态表；压缩表

**静态表：**表中的字段都是非变长字段，这样每个记录都是固定长度的，优点存储非常迅速，容易缓存，出现故障容易恢复；缺点是占用的空间通常比动态表多（因为存储时会按照列的宽度定义补足空格）ps：在取数据的时候，默认会把字段后面的空格去掉，如果不注意会把数据本身带的空格也会忽略。

**动态表：**记录不是固定长度的，这样存储的优点是占用的空间相对较少；缺点：频繁的更新、删除数据容易产生碎片，需要定期执行OPTIMIZE TABLE或者myisamchk-r命令来改善性能

**压缩表**：因为每个记录是被单独压缩的，所以只有非常小的访问开支

### 2.InnoDB存储引擎（默认存储引擎）

该存储引擎提供了具有提交、回滚和崩溃恢复能力的事务安全。

但是对比MyISAM引擎，写的处理效率会差一些，并且会占用更多的磁盘空间以保留数据和索引。

InnoDB存储引擎的特点：支持自动增长列，支持外键约束

### 3.MEMORY存储引擎（了解）

Memory存储引擎使用存在于内存中的内容来创建表。每个memory表只实际对应一个磁盘文件，格式是.frm。memory类型的表访问非常的快，因为它的数据是放在内存中的，并且默认使用HASH索引，但是一旦服务关闭，表中的数据就会丢失掉。

MEMORY存储引擎的表可以选择使用BTREE索引或者HASH索引，两种不同类型的索引有其不同的使用范围

Hash索引优点：

Hash 索引结构的特殊性，其检索效率非常高，索引的检索可以一次定位，不像B-Tree 索引需要从根节点到枝节点，最后才能访问到页节点这样多次的IO访问，所以 Hash 索引的查询效率要远高于 B-Tree 索引。

Hash索引缺点： 那么不精确查找呢，也很明显，因为hash算法是基于等值计算的，所以对于“like”等范围查找hash索引无效，不支持；

Memory类型的存储引擎主要用于哪些内容变化不频繁的代码表，或者作为统计操作的中间结果表，便于高效地对中间结果进行分析并得到最终的统计结果，。对存储引擎为memory的表进行更新操作要谨慎，因为数据并没有实际写入到磁盘中，所以一定要对下次重新启动服务后如何获得这些修改后的数据有所考虑。

### 4.MERGE存储引擎（了解）

Merge存储引擎是一组MyISAM表的组合，这些MyISAM表必须结构完全相同，merge表本身并没有数据，对merge类型的表可以进行查询，更新，删除操作，这些操作实际上是对内部的MyISAM表进行的。

|  |
| --- |
| **补充：** |

## Mysql索引：

### 1.Mysql索引支持哪些索引

###### （1）从数据结构角度

1、B+树索引(O(log(n)))：关于B+树索引，可以参考 MySQL索引背后的数据结构及算法原理

2、hash索引：

a 仅仅能满足"=","IN"和"<=>"查询，不能使用范围查询

b 其检索效率非常高，索引的检索可以一次定位，不像B-Tree 索引需要从根节点到枝节点，最后才能访问到页节点这样多次的IO访问，所以 Hash 索引的查询效率要远高于 B-Tree 索引

c 只有Memory存储引擎显示支持hash索引

3、FULLTEXT索引（现在MyISAM和InnoDB引擎都支持了）

4、R-Tree索引（用于对GIS数据类型创建SPATIAL索引）

###### （2）从物理存储角度

1、聚集索引（clustered index）

2、非聚集索引（non-clustered index）

----------------------------------------------------

**聚集索引：**

**索引中键值的逻辑顺序决定了表中相应行的物理顺序（索引中的数据物理存放地址和索引的顺序是一致的）**，可以这么理解：只要是索引是连续的，那么数据在存储介质上的存储位置也是连续的。  
比方说：想要到字典上查找一个字，我们可以根据字典前面的拼音找到该字，注意拼音的排列时有顺序的。

打个比方：当我们想要找“啊”这个字，然后又想找“不”这个字，根据拼音来看“b”一定在”a“的后面。

聚集索引就像我们根据拼音的顺序查字典一样，可以大大的提高效率。在经常搜索一定范围的值时，通过索引找到第一条数据，根据物理地址连续存储的特点，然后检索相邻的数据，直到到达条件截至项。

**非聚集索引**

**索引的逻辑顺序与磁盘上的物理存储顺序不同**。非聚集索引的键值在逻辑上也是连续的，但是表中的数据在存储介质上的物理顺序是不一致的，即记录的逻辑顺序和实际存储的物理顺序没有任何联系。索引的记录节点有一个数据指针指向真正的数据存储位置。

###### （3）从逻辑角度

1、主键索引：主键索引是一种特殊的唯一索引，不允许有空值

2、普通索引或者单列索引

3、多列索引（复合索引）：复合索引指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。使用复合索引时遵循最左前缀集合

4、唯一索引或者非唯一索引

5、空间索引：空间索引是对空间数据类型的字段建立的索引，MYSQL中的空间数据类型有4种，分别是GEOMETRY、POINT、LINESTRING、POLYGON。MYSQL使用SPATIAL关键字进行扩展，使得能够用于创建正规索引类型的语法创建空间索引。创建空间索引的列，必须将其声明为NOT NULL，空间索引只能在存储引擎为MYISAM的表中创建

### 2.Mysql联合索引

###### （1）命名规则：表名\_字段名

1、需要加索引的字段，要在where条件中

2、数据量少的字段不需要加索引

3、如果where条件中是OR关系，加索引不起作用

4、符合最左原则

###### （2）联合索引概念

联合索引又叫复合索引。对于复合索引:Mysql从左到右的使用索引中的字段，一个查询可以只使用索引中的一部份，但只能是最左侧部分。例如索引是key index (a,b,c). 可以支持a | a,b| a,b,c 3种组合进行查找，但不支持 b,c进行查找 .当最左侧字段是常量引用时，索引就十分有效。

两个或更多个列上的索引被称作复合索引。

利用索引中的附加列，您可以缩小搜索的范围，但使用一个具有两列的索引 不同于使用两个单独的索引。复合索引的结构与电话簿类似，人名由姓和名构成，电话簿首先按姓氏对进行排序，然后按名字对有相同姓氏的人进行排序。如果您知 道姓，电话簿将非常有用；如果您知道姓和名，电话簿则更为有用，但如果您只知道名不姓，电话簿将没有用处。

所以说创建复合索引时，应该仔细考虑列的顺序。对索引中的所有列执行搜索或仅对前几列执行搜索时，复合索引非常有用；仅对后面的任意列执行搜索时，复合索引则没有用处。

**建立索引语句：**

ALTER TABLE用来创建普通索引、UNIQUE索引或PRIMARY KEY索引。

ALTER TABLE table\_name ADD INDEX index\_name (column\_list)

ALTER TABLE table\_name ADD UNIQUE (column\_list)

ALTER TABLE table\_name ADD PRIMARY KEY (column\_list)

建立联合索引语句

ALTER TABLE `table\_name` ADD INDEX index\_name ( `column1`, `column2`, `column3` )

删除和查询索引

DROP INDEX index\_name ON talbe\_name

ALTER TABLE table\_name DROP INDEX index\_name

ALTER TABLE table\_name DROP PRIMARY KEY

show index from tblname;

show keys from tblname;

###### （3）什么情况下使用索引

①表的主关键字

1.自动建立唯一索引

如zl\_yhjbqk（用户基本情况）中的hbs\_bh（户标识编号）

2.表的字段唯一约束

ORACLE利用索引来保证数据的完整性

如lc\_hj（流程环节）中的lc\_bh+hj\_sx（流程编号+环节顺序）

3.直接条件查询的字段

在SQL中用于条件约束的字段

如zl\_yhjbqk（用户基本情况）中的qc\_bh（区册编号）

select \* from zl\_yhjbqk where qc\_bh=’<????甼曀???>7001’

4.查询中与其它表关联的字段

字段常常建立了外键关系

如zl\_ydcf（用电成份）中的jldb\_bh（计量点表编号）

select \* from zl\_ydcf a,zl\_yhdb b where a.jldb\_bh=b.jldb\_bh and b.jldb\_bh=’540100214511’

5.查询中排序的字段

排序的字段如果通过索引去访问那将大大提高排序速度

select \* from zl\_yhjbqk order by qc\_bh（建立qc\_bh索引）

select \* from zl\_yhjbqk where qc\_bh=’7001’ order by cb\_sx（建立qc\_bh+cb\_sx索引，注：只是一个索引，其中包括qc\_bh和cb\_sx字段）

6.查询中统计或分组统计的字段

select max(hbs\_bh) from zl\_yhjbqk

select qc\_bh,count(\*) from zl\_yhjbqk group by qc\_bh

###### （4）什么情况下应不建或少建索引

①表记录太少

如果一个表只有5条记录，采用索引去访问记录的话，那首先需访问索引表，再通过索引表访问数据表，一般索引表与数据表不在同一个数据块，这种情况下ORACLE至少要往返读取数据块两次。而不用索引的情况下ORACLE会将所有的数据一次读出，处理速度显然会比用索引快。

如表zl\_sybm（使用部门）一般只有几条记录，除了主关键字外对任何一个字段建索引都不会产生性能优化，实际上如果对这个表进行了统计分析后ORACLE也不会用你建的索引，而是自动执行全表访问。如：

select \* from zl\_sybm where sydw\_bh=’5401’（对sydw\_bh建立索引不会产生性能优化）

②经常插入、删除、修改的表

对一些经常处理的业务表应在查询允许的情况下尽量减少索引，如zl\_yhbm，gc\_dfss，gc\_dfys，gc\_fpdy等业务表。

③数据重复且分布平均的表字段

假如一个表有10万行记录，有一个字段A只有T和F两种值，且每个值的分布概率大约为50%，那么对这种表A字段建索引一般不会提高数据库的查询速度。

④经常和主字段一块查询但主字段索引值比较多的表字段

如gc\_dfss（电费实收）表经常按收费序号、户标识编号、抄表日期、电费发生年月、操作 标志来具体查询某一笔收款的情况，如果将所有的字段都建在一个索引里那将会增加数据的修改、插入、删除时间，从实际上分析一笔收款如果按收费序号索引就已 经将记录减少到只有几条，如果再按后面的几个字段索引查询将对性能不产生太大的影响。

对千万级MySQL数据库建立索引的事项及提高性能的手段

一、注意事项：

首先，应当考虑表空间和磁盘空间是否足够。我们知道索引也是一种数据，在建立索引的时候势必也会占用大量表空间。因此在对一大表建立索引的时候首先应当考虑的是空间容量问题。

其次，在对建立索引的时候要对表进行加锁，因此应当注意操作在业务空闲的时候进行。

二、性能调整方面：

首当其冲的考虑因素便是磁盘I/O。物理上，应当尽量把索引与数据分散到不同的磁盘上（不考虑阵列的情况）。逻辑上，数据表空间与索引表空间分开。这是在建索引时应当遵守的基本准则。

其次，我们知道，在建立索引的时候要对表进行全表的扫描工作，因此，应当考虑调大初始化参数db\_file\_multiblock\_read\_count的值。一般设置为32或更大。

再次，建立索引除了要进行全表扫描外同时还要对数据进行大量的排序操作，因此，应当调整排序区的大小。

    9i之前，可以在session级别上加大sort\_area\_size的大小，比如设置为100m或者更大。

    9i以后，如果初始化参数workarea\_size\_policy的值为TRUE，则排序区从pga\_aggregate\_target里自动分配获得。

最后，建立索引的时候，可以加上nologging选项。以减少在建立索引过程中产生的大量redo，从而提高执行的速度。

###### （5）MySql在建立索引优化时需要注意的问题

设计好MySql的索引可以让你的数据库飞起来，大大的提高数据库效率。设计MySql索引的时候有一下几点注意：

1，创建索引

对于查询占主要的应用来说，索引显得尤为重要。很多时候性能问题很简单的就是因为我们忘了添加索引而造成的，或者说没有添加更为有效的索引导致。如果不加

索引的话，那么查找任何哪怕只是一条特定的数据都会进行一次全表扫描，如果一张表的数据量很大而符合条件的结果又很少，那么不加索引会引起致命的性能下降。

但是也不是什么情况都非得建索引不可，比如性别可能就只有两个值，建索引不仅没什么优势，还会影响到更新速度，这被称为过度索引。

2，复合索引

比如有一条语句是这样的：select \* from users where area=’beijing’ and age=22;

如果我们是在area和age上分别创建单个索引的话，由于mysql查询每次只能使用一个索引，所以虽然这样已经相对不做索引时全表扫描提高了很多效

率，但是如果在area、age两列上创建复合索引的话将带来更高的效率。如果我们创建了(area, age,salary)的复合索引，那么其实相当于创建了(area,age,salary)、(area,age)、(area)三个索引，这被称为最佳左前缀特性。

因此我们在创建复合索引时应该将最常用作限制条件的列放在最左边，依次递减。

3，索引不会包含有NULL值的列

只要列中包含有NULL值都将不会被包含在索引中，复合索引中只要有一列含有NULL值，那么这一列对于此复合索引就是无效的。所以我们在数据库设计时不要让字段的默认值为NULL。

4，使用短索引

对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个CHAR(255)的 列，如果在前10 个或20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和I/O操作。

5，排序的索引问题

mysql查询只使用一个索引，因此如果where子句中已经使用了索引的话，那么order by中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

6，like语句操作

一般情况下不鼓励使用like操作，如果非使用不可，如何使用也是一个问题。like “%aaa%” 不会使用索引而like “aaa%”可以使用索引。

7，不要在列上进行运算

select \* from users where

YEAR(adddate)

8，不使用NOT IN和操作

NOT IN和操作都不会使用索引将进行全表扫描。NOT IN可以NOT EXISTS代替，id3则可使用id>3 or id

###### （6）为什么要使用联合索引

减少开销。建一个联合索引(col1,col2,col3)，实际相当于建了(col1),(col1,col2),(col1,col2,col3)三个索引。每多一个索引，都会增加写操作的开销和磁盘空间的开销。对于大量数据的表，使用联合索引会大大的减少开销！

覆盖索引。对联合索引(col1,col2,col3)，如果有如下的sql: select col1,col2,col3 from test where col1=1 and col2=2。那么MySQL可以直接通过遍历索引取得数据，而无需回表，这减少了很多的随机io操作。减少io操作，特别的随机io其实是dba主要的优化策略。所以，在真正的实际应用中，覆盖索引是主要的提升性能的优化手段之一。

效率高。索引列越多，通过索引筛选出的数据越少。有1000W条数据的表，有如下sql:select from table where col1=1 and col2=2 and col3=3,假设假设每个条件可以筛选出10%的数据，如果只有单值索引，那么通过该索引能筛选出1000W10%=100w条数据，然后再回表从100w条数据中找到符合col2=2 and col3= 3的数据，然后再排序，再分页；如果是联合索引，通过索引筛选出1000w10% 10% \*10%=1w，效率提升可想而知！

备注

mysql sql 中如果使用了 not in ， not exists ， （<> 不等于 ！=） 这些不走

< 小于 > 大于 <= >= 这个根据实际查询数据来判断，如果全盘扫描速度比索引速度要快则不走索引 。

### 3.扩展

|  |
| --- |
| Mysql如何选择索引  Mysql选择索引是由查询优化器来决定的，其目的是尽可能使用索引，尽可能消除更多的数据行，以最快速度提交select查找数据行。  但查询优化器并不是每次都能选择最合适的索引（例如密集索引和稀疏索引的情况），特殊情况特殊分析。    **成因：**  mysql创建复合索引的规则是首先会对复合索引的最左边，也就是索引中的第一个字段进行排序，在第一个字段排序的基础上，在对索引上第二个字段进行排序，其实就像是实现类似order by 字段1，字段2这样的排序规则，那么第一个字段是绝对有序的，而第二个字段就是无序的了，因此一般情况下直接只用第二个字段判断是用不到索引的，这就是为什么mysql要强调联合索引最左匹配原则的原因。 |

## 数据库事务

### 数据库事务的四大特性

**ACID:**



## 数据库锁