默认端口号3306

1. 命令行到数据库服务器的全流程

1. 发送SQL指令到服务器
2. 服务器检查是否在查询缓存中命中该SQL
3. 服务器进行SQL解析、预处理、优化器生成对应的执行计划
4. 根据执行计划调用引擎的API进行查询
5. 结果返回

2. 数据库的特性

事务：作为单个逻辑工作单元执行的一系列操作，要么完全执行，要么完全不执行。事务处理可以确保除非事务性单元内的所有操作都成功完成，否则不会永久更新面向数据的资源。通过将一组相关操作组合为一个要么全部成功要么全部失败的单元，可以简化错误恢复并使应用程序更加可靠。一个逻辑工作单元要成为事务，必须满足所谓的ACID（原子性、一致性、隔离性和持久性）属性。事务是数据库运行中的逻辑工作单位，由DBMS中的事务管理子系统负责事务的处理。

ACID：原子性、一致性、隔离性、持久性

1. 原子性（Atomicity）

   原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

2. 一致性

   一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

   破坏一致性的操作：丢失修改、不可重复读、脏读、幻读

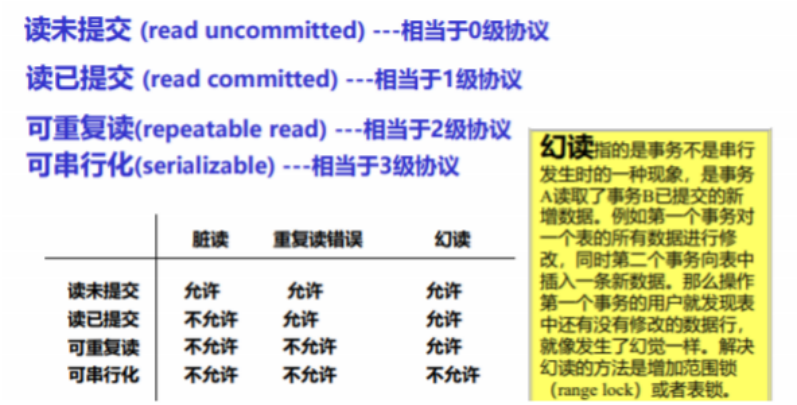
3. 隔离性

   隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

   即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

   多个事务并发访问时，事务之间是隔离的，一个事务不应该影响其它事务运行效果。这指的是在并发环境中，当不同的事务同时操纵相同的数据时，每个事务都有各自的完整数据空间。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。

   四种隔离级别



   1. **读未提交**

2. **读已提交**
3. **可重复读**
4. **可串行化**

4. 持久性

   持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

3. Mysql的四种引擎

1. MyISAM的存储引擎

   不支持事务、外键、访问快

   存在三个文件：索引文件，数据文件，表信息文件，支持b+树索引然后取数据文件中查找内容

   悲观锁，写入时加排他锁

2. InnoDB

   数据默认是B+树，索引文件和数据文件在一起

   默认以主键生成树

   如果需要创建索引，在IDB中创建一棵B+树，在叶子节点存放主键

3. InnoDB,最为通用/推荐的一种引擎，支持事务、行级锁、甚至间隙锁（避免幻读）、支持热备份，MVCC，在并发上占优势，系统资源占用多。

   MyISAM,不支持事务和行级锁，只支持表锁，某些场景性能很好：占用存储上优，查询速度上完胜（大概是InnoDB的3倍）系统资源占用少。

   InnoDB支持事务, MyISAM不支持；

   InnoDB支持行级锁、表锁；MyISAM只支持表锁；

   InnoDB支持MVCC，MyISAM不支持；

   InnoDB不支持全文索引，MyISAM支持；

   InnoDB支持外键，MyISAM不支持外键；

   InnoDB和MyISAM都支持B+树索引，InnoDB还支持自适应哈希索引

   MyISAM实现了前缀压缩技术，占用存储空间更小（但会影响查找），InnoDB是原始数据存储，占用存储更大。

4. 数据库的三大范式

1. 第一范式：关系模式中的属性不可分解
2. 第二范式：所有非主属性完全依赖于候选键
3. 第三范式：无传递依赖
4. Boyce_Codd方式：所有属性依赖于候选键

5. MVCC机制

多版本并发控制机制，InnoDB存储引擎实现隔离级别的具体实现方式，默认实现读已提交和可重复度

MVCC通过保存数据在某个时间点的快照来实现该机制。每行记录保存有两个隐藏的列，用于存储创建版本号和删除版本号，InnoDB的MVCC使用到的快照存储在Undo日志中，通过回滚指针连接

使用间隙锁防止幻读的出现

1. MVCC的更新操作：

   先插入一条新纪录，然后将原始的插入undo日志中，然后修改新记录的回滚指针

2. MVCC的读取操作：

   会生成一个read-view，包含所有未提交的活跃事务ID数组和最大的事务ID两部分组成

   读已提交：readview会更新

   可重复读：readview沿用原来的

   readview会生成所有表的readview，该事务所有的操作都会基于该readview

6. SQL的优化方式

1. 建立索引对查询优化
2. 尽量避免全表扫描（少使用 != <> ），同时考虑在where或者order by的表上建立索引
3. 避免or连接条件
4. in/not in避免使用

使用explain对sql的执行优化顺序进行分析

7. 乐观锁与悲观锁

1. 悲观锁

21. 相容性矩阵

8. 数据库的索引结构

索引的优点：

DB在执行一条Sql语句的时候，默认的方式是根据搜索条件进行全表扫描，遇到匹配条件的就加入搜索结果集合。如果我们对某一字段增加索引，查询时就会先去索引列表中一次定位到特定值的行数，大大减少遍历匹配的行数，所以能明显增加查询的速度。

优点：

通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

缺点：

创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

现在通常都使用B+树进行索引存储

1. B树：平衡的多叉查找树，m各分支称为m叉
2. B+树：保证数据稳定有序，插入修改都有稳定的对数时间复杂度，从下到上进行修改
3. 为什么要使用B树和B+树
    1. B树不用红黑树

       计算机局部性原理，局部的数据大概率会被访问，B树m叉可以满足同时多个数据访问，红黑树可能需要重新遍历
    2. B树将相关数据集中在一起，一次性读取多个数据，减少硬盘操作次数
    3. B、B+都可以大幅度减少树的深度
    4. B+树：非叶子节点指向区间，叶子节点指向关键字所在的位置
4. B、B+
    1. B树：每个节点都包含key、value，但是B+树只有叶子节点才有导航信息，叶子节点相连的节点由链表连接
    2. B+树：叶子节点是相连接的，可以通过遍历叶子节点进行整棵树的遍历，B树则需要进行递归遍历，相邻的元素可能在内存中不相邻

9. 最左匹配原则

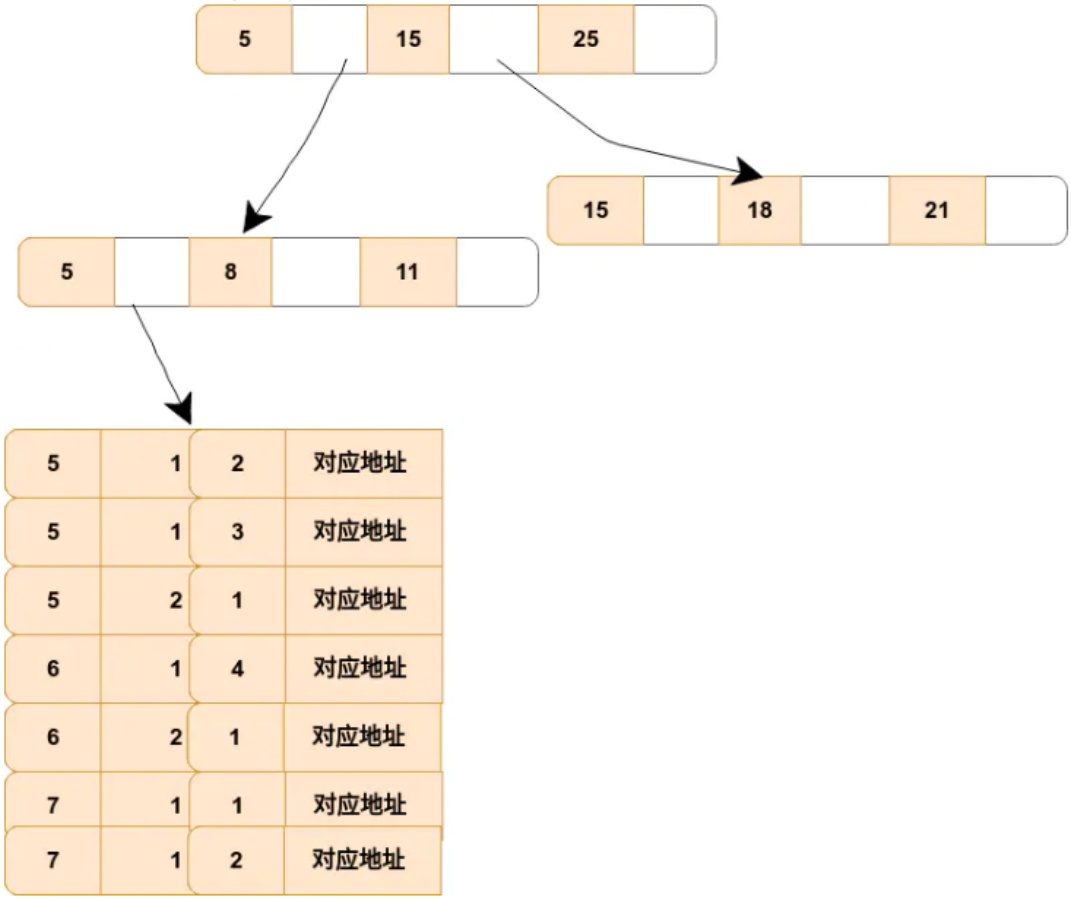
多个字段建立联合索引，name age，相当于创建了单列索引name和联合索引name，age

会优先查找最左侧那个属性的字段

即：如果你的 SQL 语句中**用到了**联合索引中的最左边的索引，那么这条 SQL 语句就可以利用这个联合索引去进行匹配。

即使顺序不对，只要用到了最左侧的，那么一定会使用联合索引

例子：假如创建一个（a,b,c)的联合索引，那么它的索引树是这样的：



10. Mysql查询的步骤

- 客户端发送查询到服务器；
- 服务器检查查询缓存query cache（大小写敏感的哈希查找，常数时间）。如果命中，返回缓存中的结果，否则下一步；
- 解析语句，生成执行计划；（SQL解析，预处理，优化器生成执行计划）；
- 根据执行计划，根据存储引擎的不同调用API，执行查询（一棵指令树）；
- 结果返回客户端。

11. B+树

1. 多叉查找树，每个节点可以有多个孩子，按照关键字大小排序
2. 每个内节点、叶结点的属性键值是有序排列的

12. Mysql的时间类型

1. time：“hh:mm:ss”
2. date：“yyyy-mm-dd”
3. datatime：“yyyy-mm-dd hh:mm:ss”
4. timestamp：“yyyymmddhhmmss”

13. 数据库的ACID实现

1. 原子性：undo log实现

   可以保证回滚/错误的时候能够恢复到原始的状态

2. 持久性：redo log实现

断电丢失数据

3. 隔离性：读写锁+MVCC实现

   读未提交、读已提交、可重复读、可串行化

4. 一致性：ACI保证--就是通过ACI的方式来保证一致性

14. innodb解决幻读

在RR隔离级别下（可重复读）：执行多次相同的select快照读，结果相同

Innodb使用MVCC和next-key locks解决幻读。MVCC解决普通读（快照读）解决幻读，next-key locks解决当前读的幻读

1. 幻读：已提交的事务B可能对操作完毕后的事务A的select产生影响
2. 当前读：加锁的select、update、delete，在RR隔离级别下，会使用next-key locks锁住本条记录以及索引区间（间隙锁）

   在RR的情况下，假设使用的是当前读，加锁了的读

   select * from table where id>3 锁住的就是id=3这条记录以及id>3这个区间范围，锁住索引记录之间的范围，避免范围间插入记录，以避免产生幻影行记录。

3. 普通读（快照读）：不加锁的读，不会存在next-key locks，使用MVCC解决

   给元组添加辅助字段记录创建版本号和删除版本号

   MVCC保证的隔离级别

   1. SELECT：读取创建版本<=当前事务版本号，删除版本为空或大于当前事务版本号的记录
   2. INSERT：将当前事务的版本号保存至行的创建版本号
   3. UPDATE：新插入一行，并以当前事务的版本号为新行的创建版本号，同时原纪录压入undolog，设置删除版本号
   4. DELETE：设置删除版本号