# 数据库总结

- SQL(Structure Query Language):结构化查询语句
    - DQL(Data Query Language):数据查询语言
        - select
    - DML(Data Manipulation Language):数据操纵语言
        - update
        - insert
        - delete
        - merge
        - call
        - explain plan
        - lock table
    - DDL(Data Definition Language):数据定义语言
        - create
        - alter
        - drop
        - truncate
        - comment
        - rename
    - DCL(Data Control Language):数据控制语言
        - grant
        - revoke
    - TCL(Transaction Control Language):事务控制语言
        - savepoint
        - rollback
        - set transaction
- 事务
    - ACID特性
        - 原子性(Atomicity)
            - 事务开始后的所有操作，要么全部完成，要么全部不做。
        - 一致性(Consistency)
            - 事务开始前和结束后，数据库的完整性约束没有被破坏。
        - 隔离性(Isolation)
            - 同一时间，只允许一个事务请求同一数据，不同事务之间彼此之间没有任何干扰。
        - 持久性(Durability)
            - 事务完成后，对数据库的所有更新都被保存到数据库，不能回滚。
    - 并发问题
        - 1.脏读
            - 事务A读取事务B未提交更新的字段，但是之后事务B进行回滚，那么事务A读取的字段是无效的。
        - 2.不可重复读

        - 事务A多次读取一个字段，但是事务B也同时对字段进行了更新，导致多次读取的值不一致。
    - 3.幻读(因事务不独立执行而产生)
        - 事务A读取字段，事务B更新了事务A读取的字段，导致事务A本来执行的结果包含了B执行的结果。
- 隔离级别
    - 读未提交
        - 事务A可以读到事务B未提交的事务。可能出现脏读，不可重复读，幻读。
    - 读已提交
        - 事务A只能读到事务B已提交的事务。可能出现不可重复读，幻读。
    - 可重复读
        - 事务A读时，则事务B不能修改(同一事务内的查询都是在事务开始时刻一致的)。可能出现幻读。
    - 序列化
        - 事务串行化执行。

- 视图(子查询)

    - 是从一个或多个表导出的虚拟的表，其内容由查询定义。具有普通表的结构，但是不实现数据存储。
    - 优点
        - 简化了操作，把经常使用的数据定义为视图。
        - 安全性，用户只能查询和修改能看到的数据。
        - 逻辑上的独立性，屏蔽了真实表的结构带来的影响。
    - 缺点
        - 性能差，数据库必须把视图查询转化成对基本表的查询。
        - 修改限制。

- 索引

    - 索引类别
        - 1.唯一索引
            - 此索引的每一个索引值只对应唯一的数据记录，可以防重。
        - 2.主键索引
            - 在唯一索引的基础上，且不允许空值。
        - 3.聚合索引
            - 在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。提供了更快的数据访问速度。
            - 一个表只能有一个聚集索引。
    - 索引的实现方式
        - 1.B+树。
        - 2.hash索引。
        - 3.位图索引。

- 数据库范式

    - 第一范式
        - 每一列都是不可分割的数据项(原子性)。

    - 第二范式
        - 满足第一范式的基础上，且每个非键属性全然依赖主键。
    - 第三范式
        - 满足第二范式基础上，且所有的属性都与主键直接相关，而不是间接相关。
    - BCNF
        - 满足第三范式的基础上，主属性不依赖主属性。
    - 第四范式
        - 要求把同一表内的多对多关系删除。
    - 第五范式
        - 从最终结构重新建立原始结构。

- 数据模型

    - 对现实世界数据特征的抽象，用来定义数据如何组织，数据之间的关系。
    - 层次
        - 1.概念模型
        - 2.逻辑/实现模型
        - 3.物理模型
            - 数据实际的物理存储方式

- 数据库系统的三级模式结构

    - 1.内模式(存储模式)
        - 是对数据物理结构和储存方式的描写叙述，是数据在数据库内部的表示方式。
    - 2.概念模式(全局模式)
        - 是对数据库中全体数据的逻辑结构和特征的描写叙述。
    - 3.外模式(子模式/用户模式)
        - 是对数据库用户可以看见和使用的局部数据的逻辑结构和特征的描写叙述。
    - 两级映射
        - 1.概念模式/内模式映射
            - 数据的物理独立性
                - 当数据的物理结构发生变化时，仅仅须要改动内模式与概念模式之间的映射就可以。
        - 2.外模式/概念模式映射
            - 数据的逻辑独立性
                - 当数据的总体逻辑结构发生变化时，仅仅须要改动各个外模式与概念模式之间的映射就可以保证应用程序不受影响。

- 数据的约束条件

    - 1.域约束：对属性取值范围的约束。
    - 2.键约束：每一个关系必需要有主键，且每一个主键必须不同样。
    - 3.非空约束：属性值不能为NULL。
    - 4.实体完整性约束：主键值不能为空。
    - 5.参照完整性约束：外键能够取NULL值，但若外键为还有一关系主键，则不能为NULL。
    - 6.用户定义的完整性。

- 存储过程

    - 存储过程是一个预编译的SQL语句。
    - 优点
        - 1.存储过程是预编译过的，执行效率高。
        - 2.存储过程的代码直接存放于数据库中，通过存储过程名直接调用，减少网络通讯。
        - 3.安全性高，执行存储过程需要有一定权限的用户。
        - 4.存储过程可以重复使用，可减少数据库开发人员的工作量。
    - 缺点
        - 移植性差

- 触发器

    - 触发器是特殊类型的存储过程。在指定表中数据发生变化时自动生效。
    - 用于强制业务规则和数据完整性。
    - 触发器是针对每一行数据的。

- 临时表

    - 临时表是一种并不存储在数据库当中的基表，是建立在系统临时文件夹中的表。
    - 临时表只在当前连接可见，当关闭连接时，MySQL会自动删除表并释放所有空间。

- 锁

    - 乐观锁
        - 每次去拿数据不会上锁。但是在更新的时候会判断一下在此期间别人有没有更新这个数据。可以使用版本号等机制。
    - 悲观锁
        - 每次在拿数据的时候都会上锁。
        - 性质
            - 共享锁(Share Lock)
                - 读锁(S锁)，共享锁是非独占的，允许多个并发事务读取其锁定的资源。
                    - 多个事务可加锁同一个共享页。
                    - 任何事务都不能修改该页。
                    - 该页被读取完毕，S锁立即被释放。
            - 排他锁(Exclusive Lock)
                - 写锁(X锁)，排他锁是独占的。
                    - 仅允许一个事务封锁此页。
                    - 其他任何事务必须等到X锁被释放才能对该页进行访问。
                    - X锁一直到事务结束才能被释放。
            - 更新锁(Update Lock)
                - U锁，在修改操作的初始化阶段用来锁定可能要被修改的资源，避免使用共享锁造成的死锁现象。
                    - 用来预定要对此页施加X锁，它允许其他事务读，但不允许再施加U锁或X锁。
                    - 当被读取的页要被更新时，则升级为X锁。
                    - U锁一直到事务结束时才能被释放。
        - 作用范围
            - 表级锁
                - 表级锁一次将整个表锁定。

        - 开销小，加锁快，不要出现死锁，锁粒度大，发生锁冲突概率最大，并发度最低。
      - 行级锁
        - 行级锁一次锁定表中的一行或多行。
        - 开销大，加锁慢，会出现死锁，锁粒度最小，发生锁冲突的概率最低，并发度最高。
      - 页面锁
        - 开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。
  - 并发控制会造成两种锁
    - 活锁
      - T1对数据D加锁，同时T2和T3也请求对数据D加锁。当T1释放了锁，T3对数据加锁，同时给T4也请求对数据D加锁，导致T2一致等待下去。
      - 解决方法
        - 采用先来先服务策略。
    - 死锁
      - 因为竞争资源导致相互等待的状态。
      - 解决方法
        - 预防死锁
          - 一次加锁法
            - 一次性把所需要的数据全部封锁住。
            - 扩大了封锁的范围，降低系统的并发度。
          - 顺序加锁法
            - 事先对数据对象指定一个封锁顺序。
        - 判定死锁
          - 超时法
            - 如果某个事物的等待时间超过指定时限，则判定为出现死锁。
          - 等待图法
            - 如果事务等待图中出现了回路，则判断出现了死锁。

- 函数
  - 函数是数据库内定义的子程序，可以被SQL语句调用。
  - ABS(),CEIL(),PI(),TRUNCATE(),SIGN(),EXP()。

- 主从复制
  - 同步复制
    - master的变化，必须等待slave-1,slave-2,...,slave-n完成后才能返回。
  - 异步复制
    - master只需要完成自己的数据库操作即可。
  - 半同步复制
    - master只保证slaves中的一个操作成功，就返回，其他slave不管。

---

## 面试题总结

---

- drop,truncate和delete区别

    - drop是连表结构和数据一起删除，truncate是删除表中全部数据，delete是根据条件删除表中的数据(一行一行删除)。
    - 执行速度：drop>truncate>delete
    - 如果是外键约束的表，不能使用truncate删除表中全部数据，而是应该使用不带条件的delete。

- in,not in,exists和not exists区别

    - in与子查询一起使用时，只会对主查询使用索引。
    - not in不会使用任何索引。
    - exists与子查询一起使用时，只会对子查询使用索引。
    - not exists会对主子查询都使用索引。

- Mysql存储引擎

    - MyISAM
        - 不支持事务
        - 不支持外键
        - 支持全文索引
        - 仅支持表级锁
        - 可以没有主键
        - 索引b+树叶节点存储记录的地址，并不存储记录数据本身。索引是非聚集的。
    - InnoDB
        - 支持事务
        - 支持外键
        - 支持行级锁
        - 必须要有主键。没有指定就默认生成不可见的ROWID列作为主键。
        - 索引b+树叶节点存储记录数据本身。索引是聚集的。

- b树和b+树

    - b树
        - 每个节点中既要存索引信息，又要存其对应的数据。
    - b+树
        - 每个中间节点只存储索引信息，所有数据都存储在叶节点上。
        - 叶节点之间有指针连接，适合范围查找。
    - 区别
        - b树查找不稳定，中间节点可能会命中。b+树查找稳定，因为所有的数据存储在叶节点上。
        - b+树的中间节点数据更少，一次放入内存页的数量更多，查找IO也更少。
        - B+树的叶节点是链接的，所以对树中的所有对象进行全扫描只需要一次线性遍历所有叶节点。

- 为什么不都用Hash索引而使用B+树索引？

    - 1.Hash索引不能使用范围查询。
        - 经过相应的Hash算法处理之后的Hash值的大小关系，并不能保证和Hash运算前完全一样。
    - 2.Hash索引无法被用来避免数据的排序操作。
        - Hash值的大小关系并不一定和Hash运算前的键值完全一样。
    - 3.Hash索引不能利用部分索引键查询。
        - 组合索引时，计算hash值时是组合索引合并后一起计算，而不是独立计算hash值。

    - 4.Hash索引在任何时候都不能避免表扫描。
        - hash值冲突，无法从hash索引中直接完成查询，需要在表中查询数据。
    - 5.Hash索引遇到大量Hash值相等的情况后性能并不一定就会比B+树索引高。

- varchar和char区别

    - 1.char的长度是不可变的，而varchar的长度是可变的。
    - 2.char的存取数度还是要比varchar要快得多。var更加注重空间效率。
        - char长度固定，方便程序的存储与查找。
    - 3.存储方式
        - char对英文字符（ASCII）占用1个字节，对一个汉字占用两个字节。
        - varchar对每个英文字符占用2个字节，汉字也占用2个字节。
    - 4.两者的存储数据都非unicode的字符数据。

- 存储过程与函数的区别

    - 存储过程是由定义的一系列sql语句的集合，函数是数据库已定义的方法。
    - 存储过程可以返回参数，函数只能返回值或表对象。
    - 存储过程可以返回多个变量，函数只能返回一个变量。
    - 存储过程的参数可以有IN,OUT,INOUT三种类型，而函数只能有IN类型。
    - 存储过程声明时不需要返回类型，而函数声明时需要描述返回类型，且函数体中必须包含一个有效的RETURN语句。
    - 函数是可以嵌入在sql中使用的,可以在select中调用，而存储过程不行。

- 视图和临时表的区别

    - 视图时虚表，临时表是实表。
    - 视图只存在于单个查询当中，临时表存在于整个数据库会话过程中。
    - 视图更新时，会将更新永久地传递至底层基表中，临时表更新时，只是对临时表更新，这些更新是临时性的。

- like,%和_区别

    - like指示mysql后面的搜索模式使用通配符而不是直接进行匹配比较。
    - %表示匹配任何字符出现的任意次数。
    - _表示匹配任意出现一次的字符。

- count(*)、count(1)、count(column)的区别

    - count(*)表示对行的数目进行计算，包括NULL。
    - count(1)与count(*)得到的结果一样。
    - count(column)表示对特定的列的行数进行计算，不包括NULL。

- 内连接、外连接、交叉连接、笛卡尔积

    - 内连接只连接匹配的行
    - 外连接
        - 左外连接
            - 包含左边表所有的行和右边表匹配的行。
        - 右外连接
            - 包含右边表所有的行和左边表匹配的行。

        - 全外连接
            - 包含左边表和右边表所有的行。
    - 交叉连接
        - 生成笛卡尔积
            - 将一个数据源中的每个行与另一个数据源的每个行都进行一一匹配。

- 索引最左前缀原则

    - 有a,b,c三个索引顺序，那么产生的索引可能是a,ab,abc。
    - (a,b,c),数据库按照从左到右来建立索引b+树。索引优先比较a来确定下一步搜索方向，再比较b，最后比较c。

- 嵌套事务

    - 嵌套事务是子事务套在父事务中执行。进入子事务之前，父事务会设置savepoint。
    - 子事务回滚
        - 父事务会回滚到进入子事务前建立的save point。
    - 父事务回滚
        - 父事务回滚，子事务也会跟着回滚。
    - 嵌套事务的提交
        - 子事务先提交，父事务再提交。
            - 子事务是父事务的一部分，由父事务统一提交。

- 查询语句执行顺序

    - select-->from-->[where]-->[group by]-->[having]-->[order by]
        - from后面的表关联，是自右向左解析。
        - where条件的解析顺序是自下而上的。

- 使用explain优化sql和索引

- 慢查询

    - 特征
        - 1.数据库CPU负载高
            - 查询语句中有很多计算逻辑，导致数据库cpu负载。
        - 2.IO负载高导致服务器卡住
            - 全表查询却没有索引。
        - 3.查询语句正常，索引正常但是还是慢
            - 可能索引没有生效。
    - 解决方法
        - 打开慢日志查询，确定是否有SQL语句占用了过多资源。
            - 在不改变业务原意的前提下，对insert、group by、order by、join等语句进行优化。
        - 考虑调整MySQL的系统参数： innodb_buffer_pool_size、innodb_log_file_size、table_cache等。
        - 确定是否是因为高并发引起行锁的超时问题。
        - 如果数据量过大，需要考虑进一步的分库分表。

- 超键，候选键，主键，外键

    - 超键
        - 在关系中能唯一标识元组的属性集称为关系模式的超键。
        - 超键包含候选键和主键。
    - 候选键(候选码)
        - 是最小超键，没有冗余元素的超键。
    - 主键(主码)
        - 数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。
        - 一个数据列只能有一个主键，且主键的取值不能缺失，即不能为空值（Null）。
    - 外键
        - 在一个表中存在的另一个表的主键称。

- MySQL中InnoDB引擎的行锁是通过加在什么上完成？

    - InnoDB是基于索引来完成行锁。

- 数据库运行于哪种状态下可以防止数据的丢失？

    - 在archivelog mode(归档模式)只要其归档日志文件不丢失，就可以有效地防止数据丢失。

- 数据表损坏的修复方式

    - 使用 myisamchk 来修复。
    - 使用repair table修复。
    - 使用OPTIMIZE table命令来修复

- 表的水平拆分和垂直拆分

    - 水平拆分用于解决表中数据量过多问题。
    - 垂直拆分把含有多个列的表拆分成多个表，解决表宽度问题。

- 读写分离

    - 对数据库读和写的操作分开对应不同的数据库服务器。
    - 主数据库提供写操作，从数据库提供读操作。
    - 主数据库进行写操作时，数据要同步到从数据库，保证数据库完整性。

- slave是否可以主动的进行写操作?

    - 不可以。
        - 因为slave无法通知master，导致数据库状态的不一致性。