数据库

- **过程存储**: 在大型数据库系统中，一组为了完成特定功能的 **SQL 语句集**，存储在数据库中，**经过第一次编译后再次调用不需要再次编译**，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象。
  优点
  1. 存储过程是一个编译过的代码块，执行效率比执行一堆T-SQL语句较高
  2. 可以重复使用，减少开发人员的工作量
  2. 存储过程位于服务器上，调用时只需要传递存储过程的名称和参数，降低网络传输的数据量，提高通信速率
  3. 使不够权限的用户能够在控制下间接的存取数据库，确保数据安全。防止SQL注入攻击

- **SQL注入**: 把sql语句插入到w eb表单或者请求域名、页面等字符串中，从而达到欺骗服务器执行恶意SQL语句。
  预防
  1. 不要完全信任用户的输入，对用户输入进行校验，通过正则化之类的方法
  2. 不要用sql拼接
  3. 对于重要的信息不要直接存取，加入hash加密之类的
  4. 不要用管理员权限直接操作数据库，对于每个应用有它自己的权限
  5. 用一些辅助软件或者网络平台来检测SQL注入

- **索引**:
  索引就是根据数据库表中的一列或者几列的值进行排序的结构，提高mysql检索效率的数据结构。类比：书籍或者图书馆的检索索引
  为什么需要索引：
  索引文件本身也很大，不可能全部存储在内存中，因此索引一般是以索引文件的形式存储在磁盘上。因此索引在查找的时候会有磁盘IO消耗，相对于内存存取，IO存取的消耗要高几个数量级
  评价索引的优劣：
  在查找过程中磁盘IO操作次数的渐进复杂度。尽量减少磁盘IO的存取次数。
  索引记录=键值（定义索引是指定的字段的值）+逻辑指针（指向数据页或者下一个索引页）
  B-TREE增加、删除节点实例分析
  https://w w w.2cto.com/database/201411/351106.html
  优点：
  1. 提高数据检索的效率，降低数据库的IO成本
  2. 利用索引来排序，降低CPU消耗
  缺点：
  1. 数据库表插入、删除更新的时候，速度降低，因为需要更新保存索引表
  2. 索引文件占用一定的磁盘空间
  3. 索引不一定能够提高查询的性能：索引需要空间存储，并且需要定期维护，当记录在增减时，索引本身也会被修改，每次插入、删除、更新会多付出4、5次的磁盘IO。如果使用不必要的索引反而会使查询时间变慢。
  B-Tree特性
  B-Tree是一种多路搜索树（并不是二叉的）：
  1.定义任意非叶子结点最多只有M个儿子；且M>2；
  2.根结点的儿子数为[2, M]；
  3.除根结点以外的非叶子结点的儿子数为[M/2, M]；
  4.每个结点存放至少M/2-1（取上整）和至多M-1个关键字；（至少2个关键字）
  5.非叶子结点的关键字个数=指向儿子的指针个数-1；
  6.非叶子结点的关键字：K[1], K[2], …, K[M-1]；且K[i] < K[i+1]；
  7.非叶子结点的指针：P[1], P[2], …, P[M]；其中P[1]指向关键字小于K[1]的子树，P[M]指向关键字大于K[M-1]的子树，其它P[i]指向关键字属于(K[i-1], K[i])的子树；
  8.所有叶子结点位于同一层；

B-TREE 类型	关键字分布	搜索结束	搜索性能	其他
B- Tree	整棵树	可能在非叶子节点结束	二分查找
B+ Tree	叶子节点	一定在叶子节点结束	二分查找	更适合文件索引系统；每个结点的指针上限为2d而不是2d+1。内结点不存储data，只存储key；叶子结点不存储指针。

为什么不用红黑树而用了B-TREE
在B-TREE中，磁盘按需读取，每次都会预读的长度一般为页的整数倍，并且数据库把一个节点的大小设置为一个页，每个节点只需要一次IO就可以完全载入。B-tree的m值设置为很大，那么让树的高度降低，有利于一次完全载入。在红黑树结构中，逻辑相邻的节点物理上不一定相邻，也就是说，读取同等的数据需要多次IO。也就是说读取同等的数据需要红黑树多次I/O。所以选择B-树效率更好。
红黑树结构，h明显要深得多。由于逻辑上很近的结点（父子结点）物理上可能离得很远，无法利用局部性原理。所以即使红黑树的I/O渐进复杂度也为O(h)

那为何最终选了B+树呢？
因为B+树内节点去掉了data域，因此可以拥有更大的出度，就是说一个结点可以存储更多的内结点，那么I/O效率更高。

MYSQL如何实现索引：
索引是存储引擎级别的概念，不同的存储引擎实现索引的方式不同。

存储引擎	索引结构	索引类型	叶节点的data域
MyISAM	B+ Tree	非聚集索引，指B+Tree的叶子节点上的data，并不是数据本身，而是数据存放的地址	存放数据记录的地址
InnoDB	B+ Tree	聚集索引，就是指主索引文件和数据文件为同一份文件	存储相应主键的值而不是地址

区别：
非聚簇索引比聚簇索引多了一次读取数据的IO操作，所以查找性能上会差。
第一个重大区别是：InnoDB的数据文件本身就是索引文件。
第二个与MyISAM索引的不同是：InnoDB的辅助索引data域存储相应记录主键的值而不是地址。聚集索引这种实现方式使得按主键的搜索十分高效，但是辅助索引搜索需要检索两遍索引：首先检索辅助索引获得主键，然后用主键到主索引中检索获得记录。

- **事务：** 事务是一个操作序列，事务里的操作要么执行，要么不执行。是维护数据库一致性的单位。
  四个属性：ACID，原子性，一致性，隔离性，持久性
  - 原子性：一个事务中的所有操作要么执行，要么不执行，如果中途某个操作执行失败了，就会回滚到事务开始的状态
  - 一致性：在事务的开始和结束后，数据库的完整性没有被破坏。
  - 隔离性：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。
  - 持久性：处理成功事务后，对数据库的修改是持久的。

锁：在所以的 DBMS 中，锁是实现事务的关键，锁可以保证事务的完整性和并发性。与现实生活中锁一样，它可以使某些数据的拥有者，在某段时间内不能使用某些数据或数据结构。当然锁还分级别的。

- **drop,delete, truncate的区别**
  truncate、delete用于删除表中的数据，但是不删除表，但delete可以配合where来使用，删除部分记录
  truncate、drop是一次性从表中删除所有的数据并且不会把单独的操作记录记入到日志中，删除是不可以恢复的，并且不会激活与表相关的删除触发器。truncate会将表和索引占用的空间恢复到初始大小。truncate只能对表进行操作。
  delete语句每次从表中删除一行，然后同时将该行的操作记录作为事务记录在日志中保存以便进行回滚操作。delete不会减少表和索引占用的空间。delete可以对表和视图。
  drop直接删除表
  速度：drop>truncate>delete(delete每次删除一行，所以比较慢)

DML(DATA MAINTAIN LANGUAGE):操作先放到rollback segment，事务提交后才生效。如果有相应的trigger，会激活。可以回滚。
DDL(DATA DEFINE LANGUAGE)：操作立即执行，原数据不会放到rollback segment中，不能回滚。

1. 在速度上，一般来说，drop> truncate > delete。
2. 在使用drop和truncate时一定要注意，虽然可以恢复，但为了减少麻烦，还是要慎重。
3. 如果想删除部分数据用delete，注意带上where子句，回滚段要足够大；

如果想删除表，当然用drop；
如果想保留表而将所有数据删除，如果和事务无关，用truncate即可；
如果和事务有关，或者想触发trigger，还是用delete；
如果是整理表内部的碎片，可以用truncate跟上reuse stroage，再重新导入/插入数据

- **超键、主键、候选键、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。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。超键包含候选键和主键。
  - 候选键：是 **最小超键**，即没有冗余元素的超键。
  - 主键：**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** 称此表的外键。
    举个例子：
    身份证、学号、姓名、年龄、系别
    <身份证,姓名>可以作为超键，但是不能作为候选键，因为去掉姓名仍然能够唯一标识该元祖，也就是不是最小超键，所以不是候选键。

- **视图**
  视图是一种虚拟的表，能够对视图进行增删改查的操作并且不影响基本表。使得我们获取数据更容易。

  - 只暴露部分字段给访问者
  - 查询的数据来源于不同的表，查询者想要用统一的方式查询，这时可以建立一个视图。查询者直接从视图中获取数据，不需要考虑数据来源于不同表所带来的差异。

- **三个范式**
  范式越高，数据的冗余度越小。其实没有冗余的数据库设计是可以做到的。但是，没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。降低范式就是增加字段，允许冗余。

  - 1NF：属性不可分
  - 2NF：符合1NF，表必须有一个主键；没有包含在主键中的列必须完全依赖于主键，而不能只依赖于主键的一部分。
    比如说主键为,那么非主键C一定是->C;如果A->C,则不满足二范式。
  - 3NF：符合2NF，并且，消除传递依赖。
    考虑一个订单表[Order]（OrderID，OrderDate，CustomerID，CustomerName，CustomerAddr，CustomerCity）主键是（OrderID）。
    其中 OrderDate，CustomerID，CustomerName，CustomerAddr，CustomerCity 等非主键列都完全依赖于主键（OrderID），所以符合 2NF。不过问题是 CustomerName，CustomerAddr，CustomerCity 直接依赖的是 CustomerID（非主键列），而不是直接依赖于主键，它是通过传递才依赖于主键，所以不符合 3NF。通过拆分[Order]为[Order]（OrderID，OrderDate，CustomerID）和[Customer]（CustomerID，CustomerName，CustomerAddr，CustomerCity）从而达到 3NF。
  - BCNF：符合3NF，并且，主属性不依赖于主属性。
    举个例子：
    码：(管理员、物品名),(仓库名、物品名)
    主属性：管理员、物品名、仓库名
    非主属性：数量
    这里属于3NF，但仍然存在插入异常（如果增加一个仓库，不能指派管理员，因为还没有物品），删除异常（物品被请空之后，仓库和管理员信息也被删除了）和更新异常（更新一个仓库的管理员名字，需要修改多条数据）的问题
    在 3NF 的基础上消除主属性对于码的部分与传递函数依赖
    表1：（仓库名）->(管理员)
    表2：（仓库名）->(物品名，数量)

- **异常**
  插入异常，删除异常，修改异常

  表中的一行就是一个元组。

- **数据库五大约束**

1. primary KEY:设置主键约束；
2. UNIQUE：设置唯一性约束，不能有重复值；
3. DEFAULT 默认值约束，height DOUBLE(3,2)DEFAULT 1.2 height不输入是默认为1,2
4. NOT NULL：设置非空约束，该字段不能为空；
5. FOREIGN key :设置外键约束。

- **触发器的作用**
  触发器是一种特殊的存储过程，主要是通过事件来触发而被执行的。它可以强化约束，来维护数据的完整性和一致性，可以跟踪数据库内的操作从而不允许未经许可的更新和变化。可以联级运算。如，某表上的触发器上包含对另一个表的数据操作，而该操作又会导致该表触发器被触发。

计算机网络

操作系统

数据结构