# 1. 主键 超键 候选键 外键

**主 键：**

[数据库](http://www.2cto.com/database/)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）。

**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。

**外 键：**

在一个表中存在的**另一个表的主键**称此表的外键。

# 2.数据库事务的四个特性及含义

数据库事务transanction正确执行的四个基本要素。ACID,原子性(Atomicity)、一致性(Correspondence)、隔离性(Isolation)、持久性(Durability)。  
**原子性**:整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。  
**一致性**:在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。  
**隔离性**:隔离状态执行事务，使它们好像是[系统](http://www.2cto.com/os/)在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行 相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间的混淆，必须串行化或序列化请 求，使得在同一时间仅有一个请求用于同一数据。  
**持久性**:在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

# 3.视图的作用，视图可以更改么？

视图是虚拟的表，与包含数据的表不一样，视图只包含使用时动态检索数据的查询；不包含任何列或数据。使用视图可以简化复杂的sql操作，隐藏具体的细节，保护数据；视图创建后，可以使用与表相同的方式利用它们。  
视图不能被索引，也不能有关联的触发器或默认值，如果视图本身内有order by 则对视图再次order by将被覆盖。  
创建视图：create view XXX as XXXXXXXXXXXXXX;  
对于某些视图比如未使用联结子查询分组聚集函数Distinct Union等，是可以对其更新的，对视图的更新将对基表进行更新；但是视图主要用于简化检索，保护数据，并不用于更新，而且大部分视图都不可以更新

# 4.数据库范式

**1 第一范式（1NF）**  
  
在任何一个关系数据库中，第一范式（1NF）是对关系模式的基本要求，不满足第一范式（1NF）的数据库就不是关系数据库。  
所谓第一范式（1NF）是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。如果出现重复的属性，就可能需要定义一个新的实体，新的实体由重复的属性构成，新实体与原实体之间为一对多关系。在第一范式（1NF）中表的每一行只包含一个实例的信息。简而言之，**第一范式就是无重复的列。**

**2 第二范式（2NF）**  
  
第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。这个惟一属性列被称为主关键字或主键、主码。  
第二范式（2NF）要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。简而言之，**第二范式就是非主属性非部分依赖于主关键字。**

**3 第三范式（3NF）**  
  
满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。例如，存在一个部门信息表，其中每个部门有部门编号（dept\_id）、部门名称、部门简介等信息。那么在员工信息表中列出部门编号后就不能再将部门名称、部门简介等与部门有关的信息再加入员工信息表中。如果不存在部门信息表，则根据第三范式（3NF）也应该构建它，否则就会有大量的数据冗余。简而言之，**第三范式就是属性不依赖于其它非主属性。（我的理解是消除冗余）**

**5.存储过程与触发器的区别**

触发器与存储过程非常相似，触发器也是SQL语句集，**两者唯一的区别是触发器不能用EXECUTE语句调用，而是在用户执行Transact-SQL语句时自动触发（激活）执行。触发器是在一个修改了指定表中的数据时执行的存储过程。**通**常通过创建触发器来强制实现不同表中的逻辑相关数据的引用完整性和一致性。**由于用户不能绕过触发器，所以可以用它来强制实施复杂的业务规则，以确保数据的完整性。触发器不同于存储过程，**触发器主要是通过事件执行触发而被执行的**，而**存储过程可以通过存储过程名称名字而直接调用**。当对某一表进行诸如UPDATE、INSERT、DELETE这些操作时，SQLSERVER就会自动执行触发器所定义的SQL语句，从而确保对数据的处理必须符合这些SQL语句所定义的规则。

**6、MySQL中varchar与char的区别以及varchar(50)中的50代表的涵义**  
**(1)、varchar与char的区别**

在单字节字符集下， char（ N） 在内部存储的时候总是定长， 而且没有变长字段长度列表中。 在多字节字符集下面， char(N)如果存储的字节数超过 N，那么 char（ N）将和 varchar（ N）没有区别。在多字节字符集下面，如果存

储的字节数少于 N，那么存储 N 个字节，后面补空格，补到 N 字节长度。 都存储变长的数据和变长字段长度列表。 varchar(N)无论是什么字节字符集，都是变长的，即都存储变长数据和变长字段长度列表。

**(2)、varchar(50)中50的涵义**

最多存放50个字符，varchar(50)和(200)存储hello所占空间一样，但后者在排序时会消耗更多内存，因为order by col采用fixed\_length计算col长度(memory引擎也一样)。在早期 MySQL 版本中， 50 代表字节数，现在代表字符数。

**(3)、int（20）中20的涵义**

是指显示字符的长度

不影响内部存储，只是影响带 zerofill 定义的 int 时，前面补多少个 0，易于报表展示

**(4)、mysql为什么这么设计**

对大多数应用没有意义，只是规定一些工具用来显示字符的个数；int(1)和int(20)存储和计算均一样；

## 7、MySQL中InnoDB引擎的行锁是通过加在什么上完成(或称实现)的？为什么是这样子的？

答：InnoDB是基于索引来完成行锁

例: select \* from tab\_with\_index where id = 1 for update;

for update 可以根据条件来完成行锁锁定,并且 id 是有索引键的列,

如果 id 不是索引键那么InnoDB将完成表锁,,并发将无从谈起

## 8、当MySQL数据库cpu飙升到500%的话他怎么处理？

当 cpu 飙升到 500%时，先用操作系统命令 top 命令观察是不是 mysqld 占用导致的，如果不是，找出占用高的进程，并进行相关处理。如果是 mysqld 造成的， show processlist，看看里面跑的 session 情况，是不是有消耗资源的 sql 在运行。找出消耗高的 sql，  
看看执行计划是否准确， index 是否缺失，或者实在是数据量太大造成。一般来说，肯定要 kill 掉这些线程(同时观察 cpu 使用率是否下降)，等进行相应的调整(比如说加索引、改 sql、改内存参数)之后，再重新跑这些 SQL。也有可能是每个 sql 消耗资源并不多，但是突然之间，

有大量的 session 连进来导致 cpu 飙升，这种情况就需要跟应用一起来分析为何连接数会激增，再做出相应的调整，比如说限制连接数等

## 9、500台db，在最快时间之内重启

可以使用批量 ssh 工具 pssh 来对需要重启的机器执行重启命令。 也可以使用 salt（前提是客户端有安装 salt）或者 ansible（ ansible 只需要 ssh 免登通了就行）等多线程工具同时操作多台服务器

## 10、表中有大字段X(例如：text类型)，且字段X不会经常更新，以读为为主，请问您是选择拆成子表，还是继续放一起?写出您这样选择的理由

答：拆带来的问题：连接消耗 + 存储拆分空间；不拆可能带来的问题：查询性能；

如果能容忍拆分带来的空间问题,拆的话最好和经常要查询的表的主键在物理结构上放置在一起(分区) 顺序IO,减少连接消耗,最后这是一个文本列再加上一个全文索引来尽量抵消连接消耗

如果能容忍不拆分带来的查询性能损失的话:上面的方案在某个极致条件下肯定会出现问题,那么不拆就是最好的选择