**一、基础部分**

**Q：SQL事务四大特性（ACID）**

A： **原子性（Atomicity）**：事务是数据库的逻辑工作单位，事务中操作要么都做，要么都不做。

**一致性（Consistency）**：事务应该把数据库从一个一致性状态转换到另外一个一致性状态。事务前后，数据库的状态都满足所有的完整性约束。拿银行转账来说，一致性要求事务的执行不应改变A、B 两个账户的金额总和。如果没有这种一致性要求，转账过程中就会发生钱无中生有，或者不翼而飞的现象。

**隔离性（Isolation）**：一个事务执行不能被其他事务干扰；

**持久性（Durability）**：一个事务一旦成功完成，它对数据库的改变必须是永久的。

**Q：分布式CAP原则**

**A： Consistency(一致性)**

数据一致更新，所有数据变动都是同步的；

**Availability(可用性)**

在一定时间内，用户的请求都会得到应答；

**Partition tolerance(分区容错性)**

系统出现分区后，各节点仍然可以访问数据；

**Q：强一致性、弱一致性、最终一致性**

A： 更新过的数据能被后续的访问都能看到，这是**强一致性**；

如果能容忍后续的部分或者全部访问不到，则是**弱一致性**；

如果经过一段时间后要求能访问到更新后的数据，则是**最终一致性**。

<http://blog.csdn.net/c289054531/article/details/15337575>

**Q：数据库有哪些锁**

A： 数据库的锁一般分为两类：乐观锁和悲观锁。乐观锁，认为没有其它操作会修改数据，在操作数据前，所以不加锁；在提交数据时，由用户自定义提交规则。悲观锁，认为数据随时都会修改，在操作数据前，使用关系数据库提供的锁机制对数据加锁。

**悲观锁分类：**

**共享锁**即读锁，事务可以并发读取数据，共享锁和共享锁兼容。

**排它锁**即写锁，获取排它锁的事务可以读和写数据，排它锁和所有其他锁不兼容，在对数据加排它锁时，需要等所有其它都解除。所以当事务希望修改数据时，需要将共享锁升级到排它锁，在并发环境下，两个写事务都在等对方释放共享锁，所以可能会产生死锁。

**更新锁**，是为了解决死锁问题，更新锁可以与共享锁兼容，但是不能和更新锁兼容，同时，只允许一个事务对数据加更新锁，加更新锁的事务在等所有共享锁释放后，可以升级为排它锁，对数据进行修改。

**意向锁**，当一个表中的某一行被加上排他锁后，该表就不能再被加表锁。对数据加排它锁时对表加意向锁，已方便其它事务判断能否操作该表。

**乐观锁的实现：**

**版本号**：给数据库中的表加一个version字段，每次更新数据对字段加1。读取数据时同时读取表的版本号，提交更新时与当前版本号比较，如果相同，直接提交；如果当前版本号大，再进行事务回滚或者重新操作一遍。

**时间戳**：同版本号一致，在表中增加一个字段，记录最后一次对更新表的时间。

**待更新字段**：思想和版本号一致，但是不用额外增加字段，直接通过比较某待更新字段的值在读取和提交更新期间是否变化判断期间是否有更新。

**所有字段**：和待更新字段一致，只有在操作期间，只有整个表所有字段都没有更新才执行更新。

<http://blog.csdn.net/aluomaidi/article/details/52460844>

**Q：数据库索引，索引的数据结构B/B+树，二者的区别？**

A： **为什么要使用索引：**

a、表中的数据以块的形式存储，逻辑上相邻的数据块在物理上可以相隔很远。遍历查询时，需要大量的磁盘IO。

b、在对未经排序的字段中查找时，需要线性查找。复杂度为O（n）；

**使用索引的优点：**

a、大大加快查询速度

b、索引记录只包含一个地址指针，索引实体比真实数据行要小很多，索引页较数据页要密集很多。所以一个索引页可以储存更多的索引记录，这样可以减少将页面载入内存的次数。

c、建立唯一索引可以保证行的唯一性；

d、可以给非主键的列添加索引，提高根据非主键列进行查询的速度。

**使用索引的缺点：**

a、增加了数据库的存储空间；

b、在增加、删除和修改数据时，需要维护索引，增加开销；

<https://www.cnblogs.com/myseries/p/5222634.html>

**常用的索引数据结构：**

我们常见的数据库系统，其索引使用的数据结构多是B-Tree或者B+Tree。例如，Mysql使用的是B+Tree，Oracle及Sysbase使用的是B-Tree。

**为什么选择使用B树和B+树，为什么不用哈希表？**

a、hash只能用=查询，不能用于大于、小于的范围查询；

b、hash表是无序的，无法利用hash表对键值进行排序；

c、hash无法用于组合查询；

d、hash需要处理hash冲突、扩容的问题，且占用的内存空间比B/B+树高。

**B/B+树的比较：**

B+树是B树的变种，相对于B树，B+树有以下不同：

a、将所有关键字（data）存储在叶子节点中，非叶子节点不存储真正的data；

b、为所有叶子节点增加了一个指向下一个链指针。

因为这些不同，在数据库索引的应用上，B+树性能要优于B树：

a、IO次数更少。由于B+树内节点无 data信息，所以一个页面可以存储更多的内结点信息，每次能索引的范围更大更精确，磁盘IO的次数更少，I/O效率更高。

b、每个关键字的查询次数一样长，查询更稳定。

c、B+树范围查询更快。

**聚集索引和非聚集索引？**

聚集索引和非聚集索引的根本区别是，表记录的排列顺序和与索引的排列顺序是否一致。

a、一个表只能有一个聚集索引，但可以有多个非聚集索引

b、聚集索引查询数据速度快，聚集索引的叶子节点中存储的就是数据记录，非聚焦索引叶子节点存的是记录的指针。

c、聚焦索引查找范围内的数据很快

d、增删数据时，聚集索引维护慢。

<https://www.cnblogs.com/shanshanlaichi/p/6568097.html>

**怎样合理使用索引：**

a、记录很少时，不要使用索引；

b、对于主键和外键，创建索引；

c、在需要经常搜索的列创建索引，对需要经常被排序的列创建聚集索引；

d、对取值很少的列（比如性别）不使用索引；

e、需要频繁增加、删除、修改的列，使用非聚集索引；

**Q：数据库的范式？**

A： **第一范式（1NF）**：

　　数据表中的每一列(字段)，必须是不可拆分的最小单元，也就是确保每一列的原子性。

　　例如： userInfo: '山东省烟台市 1318162008' 依照第一范式必须拆分成

                   userInfo: '山东省烟台市'　　 userTel: '1318162008'两个字段

**第二范式（2NF）：**

　　在满足1NF的基础上，增加两个内容：表中要有主键；非主键列不能部分依赖于主键。

例如：一个表四列：A、B、C、D，其中主键为A和B，D完全依赖与A-B主键，C只依赖于B，此时，根据第二范式，需要拆分为A、B、D一张表，B、C一张表。

**第三范式(3NF)：**

满足2NF后，要求：表中的每一列都要与主键直接相关，而不是间接相关（表中的每一列只能依赖于主键）。

例如：一张表三列A、B、C，其中A为主键，B依赖于A，C依赖于B。此时需要拆分为两张表：A、B一张表，B、C一张表。

**BC范式(BCNF)：**

多个关键字，不能耦合到一个表中。此时需要拆分表。

即：A依赖B，C也依赖B，但是A和C之间没有关系。此时将导致以下问题：当B和A的对应关系改动时，整张表都需要改动；而且，当没有C时，不能单独存储A、B的关系。

**第四范式（4NF）**

对多值属性拆分出来，单独建表。

**第二范式和第三范式的区别：**

在于有没有分出两张表。第二范式是说一张表中包含了多种不同实体的属性，那么必须要分成多张表，第三范式是要求已经分好了多张表的话，一张表中只能有另一张标的ID，而不能有其他任何信息，（其他任何信息，一律用主键在另一张表中查询）。

<http://blog.csdn.net/huzhaomu945/article/details/52821755>

**Q：数据库约束**

A： 主键约束（Primay Key Coustraint） 唯一性，非空性

唯一约束 （Unique Counstraint）唯一性，可以空，但只能有一个，主键不可再设唯一约束。

检查约束 (Check Counstraint) 对该列数据的范围、格式的限制（如：年龄、性别等）

默认约束 (Default Counstraint) 该数据的默认值

外键约束 (Foreign Key Counstraint) 需要建立两表间的关系并引用主表的列

<https://www.cnblogs.com/wcl2017/p/7043939.html>

**Q：数据库的优化**

A： 数据库的优化有以下方式：

a、合理创建索引，并且只通过索引访问数据。通过索引访问数据，可以大幅度减少数据库系统访问磁盘的次数。

b、优化查询语句，每次查询只返回需要的字段，减小数据处理和传输开销。尽量不要使用模糊查询。

c、减少交互次数，客户端查询后将结果存储起来，防止对数据库做重复查询操作。

d、将大量复杂运算在客户端进行，减轻服务器CPU压力；

e、优化服务器硬件、优化网络质量；

f、使用数据库连接池

<https://www.cnblogs.com/easypass/archive/2010/12/08/1900127.html>

**Q：主键和外键？**

A： 若某一个属性组（注意是组）能唯一标识一条记录，该属性组就是一个主键。主键不能重复，且只能有一个，也不允许为空。定义主键主要是为了维护关系数据库的完整性。

外键用于与另一张表的关联，是能确定另一张表记录的字段。外键是另一个表的主键，可以重复，可以有多个，也可以是空值。定义外键主要是为了保持数据的一致性。

**Q：数据库的join含义及其分类：内联结（INNER JOIN）、外联结（OUTER JOIN，包括左联结（LEFT OUTER JOIN）、右联结（RIGHT OUTER JOIN））的区别？**

A： 含义：数据库中join语句是指将多个表中的数据并联在一起，然后根据要求找出需要的数据。

内联结是保证两个表中所有的行都要满足联结条件。在外联结中，某些不满条件的列也会显示出来。左联结保证左表全部显示，右表中不匹配的部分显示为NULL；全联结全部显示，不匹配位置显示为NULL。

**Q：数据库隔离级别？**

A： 事务的并发操作中，可能会出现脏读、读取旧数据、幻读等问题。通过数据库的四种隔离级别，可以解决这些问题：

**Read Uncommited（未提交读）**

写事务阻止其他写事务，避免了更新遗失。但是没有阻止其他读事务。

存在的问题：脏读。即读取到不正确的数据，因为另一个事务可能还没提交最终数据，这个读事务就读取了中途的数据，这个数据可能是不正确的。解决办法就是下面的“可读取确认”。

**Read committed（提交读）**

写事务会阻止其他读写事务。读事务不会阻止其他任何事务。

存在的问题：不可重复读。即在一次事务之间，进行了两次读取，但是结果不一样，可能第一次id为1的人叫“李三”，第二次读id为1的人就叫了“李四”。因为读取操作不会阻止其他事务。解决办法就是下面的“可重复读”。

**Repeatable Read（可重复读）**

读事务会阻止其他写事务，但是不会阻止其他读事务。

存在的问题：幻读。可重复读阻止的写事务包括update和delete（只给存在的行加上了锁），但是不包括insert（新行不存在，所以没有办法加锁），所以一个事务第一次读取可能读取到了10条记录，但是第二次可能读取到11条，这就是幻读。解决办法就是下面的“串行化”。

**Serializable（可串行化）**

读加共享锁，写加排他锁，表加意向锁（整个表不能修改）。这样读取事务可以并发，但是读写，写写事务之间都是互斥的，基本上就是一个个执行事务，所以叫串行化。

<http://blog.csdn.net/gaopu12345/article/details/50868501>

**Q：SQL常用命令：**

A： Create：创建

CREATE TABLE Student(ID NUM PRIMARY KEY,NAME VARCHAR2(50) NOT NULL);//建表

CREATE VIEW view\_name AS Select \* FROM Table\_name;//建视图

Create UNIQUE INDEX index\_name ON TableName(col\_name);//建索引

Insert：增

INSERT INTO tablename {column1,column2,…}values(exp1,exp2,…);//插入

Delete：删

DELETE FROM Tablename WHERE condition //删除数据

Update：改

UPDATE tablename SET name=’zang 3’ condition; //更新数据

Select：查

Select E.NAME, S.NAME FROM EMPLOYEE E S WHERE E.SUPERName=S.Name

**Q：mysql数据类型**

A：数值（int、float、double）、时间（date、time、year）、字符串类型（char、enum、text）、set

**Q：有哪些数据库对象？**

A： 表、索引、视图、图表、缺省值、规则、触发器、存储过程、游标

**Q：存储过程与触发器？**

A： 在没有使用存储过程的数据库应用程序中，用户所编写的应用程序都是从本地计算机(client)向服务器(server)端发送SQL代码来请求对数据库中数据的增删改差操作，服务器对接收到的SQL代码进行编译后执行，并将结果返回给client，再由客户端的软件处理后输出。如果开发者对服务器安全性考虑不周全，就会为黑客提供盗取数据的机会。其中SQL注入是一种常见的方式。

存储过程是SQL语句和控制语句的预编译集合，存放在服务器端，用户调用时只需要传入存储过程的名称即可，减少了网络流量，同时可以防止SQL注入。

触发器是一种特殊的存储过程，它通过事件触发自动执行。例如，当表发生增删改操作时，自动执行对应触发器。

存储过程就像服务器提供给客户端的API，而触发器是服务器程序内部的Callback。

**Q：什么是视图，视图可以更改吗？**

A： a、视图是是从一个或多个表导出的虚拟的表，其内容由查询定义。它具有普通表的结构，但是不实现数据存储。

b、可以通过使用表的方式使用视图，简化复杂的sql操作，隐藏具体的细节，保护数据；

c、视图不能被索引，不能有关联的触发器或默认值。

**对视图的修改**：

单表视图对视图的修改会改变基本表的数据，多表视图不会改变基本表的数据。

**Q：什么是游标？**

A：关系型数据库中并未提供一种描述表中单一记录的表达形式，游标是一种能从包括多条数据记录的结果集中每次提取一条记录的机制，把表中的数据以平面文件的形式呈现给用户。

**Q：为什么使用数据库连接池？**

A： 创建数据库连接是一个很耗时的操作，频繁的打开和关闭连接数据库操作会影响服务器性能。数据库连接池的基本思想就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。

**使用数据库连接池的优点：**

a、资源重用。连接池里的连接可以重复使用，避免了频繁创建、释放连接引起的大量性能开销。

b、更快的系统响应速度 。连接池中的连接已经做了初始化工作，相对于重新创建可以提高效率。

c、实现对数据库的连接的数量、使用情况进行管理。

**Q：如何设计一个高并发的系统？**

A：① 数据库的优化，包括合理的事务隔离级别、SQL语句优化、索引的优化

② 使用缓存，尽量减少数据库 IO

③ 分布式数据库、分布式缓存

④ 服务器的负载均衡

**Q：模糊查询**

A： 一般模糊语句格式如下：

SELECT 字段 FROM 表 WHERE 某字段 LIKE 条件;

%：零个或者多个字符

eg：select \* from flow\_user where username like '%王%';

\_：任意单个字符

eg：select \* from flow\_user where username like '\_英\_';

[]：表示括号内所列字符中的一个（类似正则表达式）

eg：select \* from flow\_user where username LIKE'[王李张]飞';

[^]：表示不在括号所列之内的单个字符

eg：select \* from flow\_user where username LIKE'[^王李张]飞';

**Q：Mysql存储引擎**

A： MYISAM、InnoDB、MEMORY、MERGE

a、MYISAM：不支持事务、不支持外键，使用非聚集索引，但是访问速度快。

b、InnoDB：支持事务，支持

a、MYISAM不支持ACID事务，但是每次查询时原子的；

b、MYISAM支持表级锁，每次操作对整个表加锁，不支持写并发；InnoDB支持行锁和外键约束，支持写并发；

c、MYISAM采用非聚集索引，InnoDB支持聚集索引。

**二、Nosql数据库**

**Q：Nosql和sql数据库比较**

A： a、SQL是结构化存储，定义表的时候就限定了列数及每一列的数据类型；NoSQL是动态结构存储，可以存储hashtable等结构，并且可以扩展数据类型。

b、redis等nosql数据库，将数据放在内存中，读写速度较sql有大幅度提升

c、扩展性，向上扩展和水平扩展：SQL是中心化的，在服务量激增时，需要通过使用性能更好的服务器来实现扩展。Nosql数据库是分布式的，可以通过动态增加服务器数量实现扩展。

d、付费VS开源，SQL数据库大多数比较昂贵，而NoSQL数据库通常都是开源的。

**nosql和sql的选择：**

a、SQL是精确的，适用于数据类别可以精确描述的场景，例如公司员工信息管理；

b、Nosql是多变的，适用于具有不确定性需求的数据。例如社交软件、网络资源等。

<https://www.cnblogs.com/beilin/p/6007080.html>

<http://blog.csdn.net/alexdamiao/article/details/51457399>

**Q：Redis简介**

A： a、Redis 是一个基于内存的高性能key-value数据库。Redis通过Key-Value的单值不同类型来区分, 支持以下的类型: string、List、Set、sort set、hash；

b、Redis为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以redis具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘I/O速度为严重影响redis的性能。

c、redis支持主从的模式。采用读写分离方式，利用master来插入数据，slave提供检索服务。这样可以有效减少单个机器的并发访问数量

**使用Redis有哪些好处？**

a、速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)

b、 支持丰富数据类型，支持string，list，set，zset，hash

c、支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行

d、丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

**redis相比memcached有哪些优势？**

(1) memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类型

(2) redis的速度比memcached快很多

(3) 存储方式 Memecache把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。 Redis有部分存在硬盘上，这样能保证数据的持久性。

**Q：redis数据类型及其实现**

A： redis支持string，list，set，zset（有序集合），hash

**string**

String类型就是一个由字节组成的序列，他和其他编程语言或者其他键值对存储提供的字符串操作非常相似。

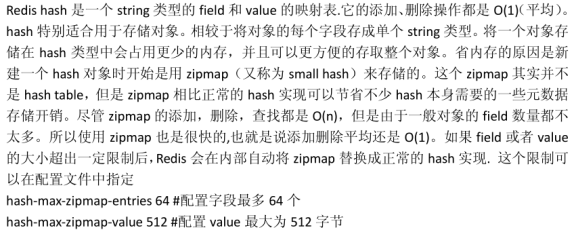
**list**

Redis中的List就是链表（redis 使用双端链表实现的 List）。

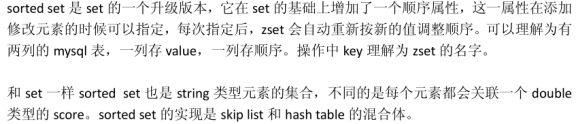
**set**

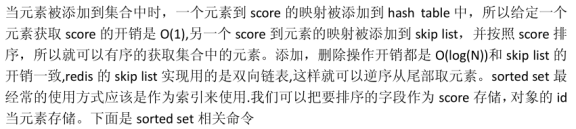
Redis的集合和列表都可以存储多个字符串，他们的不同支持在于，列表可以存储多个相同的字符串，而集合通过使用散列表来保证自己存储的每个字符串都是各不相同的。

**hash**



**zset**





**Q：Mysql主从复制原理**

A： 是用来建立一个和主数据库完全一样的数据库环境，称为从数据库。

**主从复制的目的：**

a、作为后备数据库，当主数据库挂了时，可以切换从数据库；

b、架构的扩展，多库存储，减轻主服务器的压力；

c、采用master和slave，读写分离，使数据库可以支撑更大开发。

**原理：**

从服务器定时拷贝主服务器的bin-log文件，并在本地按照日志文件进行同样操作。

**Q：如何保证数据库的一致性**

A： a、两阶段算法：第一阶段，单个服务器上有数据更新时，提交仲裁者，所有服务器投票，由协调者仲裁，仲裁结果不通过，事务取消；第二阶段，仲裁通过，仲裁者将事务提交给所有服务器，保证一致性更新。

b、分布式锁：服务器更新数据时，先竞争分布式锁，再进行操作，直到所有节点更新一致后，释放锁；

<http://blog.csdn.net/oanqoanq/article/details/51188436>

**Q：SQI执行计划**

A：

**Q：Mysql的数据量太大了怎么处理**

A： 进行分区或分表

分区：分区是将数据划分在不同位置存放，表仍然是一张表，读写时db自动去子区块中查询。

分表：将大表按照一定规则分解成多张子表。读写时根据预定规则得到子表名，然后去操作子表。

**//以上为已经整理过的面经**

**2、数据库隔离级别，每个级别会引发什么问题，mysql默认是哪个级别**

详见：数据库-》数据库的4种隔离级别

**3、MYSQL的两种存储引擎区别（事务、锁级别等等），各自的适用场景**

MYISAM 不支持事务，不支持外键，表锁，插入数据时，锁定整个表，查表总行数时，不需要全表扫描

INNODB 支持事务，支持外键，行锁，查表总行数时，全表扫描

**5、有哪些锁（乐观锁悲观锁），select时怎么加排它锁**

详见：数据库-》乐观锁悲观锁

索引有B+索引和hash索引，各自的区别

B+索引数据结构，和B树的区别

索引的分类（主键索引、唯一索引），最左前缀原则，哪些情况索引会失效

聚集索引和非聚集索引区别。

关系型数据库和非关系型数据库区别

了解nosql

数据库三范式，根据某个场景设计数据表（可以通过手绘ER图）

数据库的主从复制

使用explain优化sql和索引

long\_query怎么解决

内连接、外连接、交叉连接、笛卡儿积等

**深入**

MVCC机制

根据具体场景，说明版本控制机制

死锁怎么解决

varchar和char的使用场景。

mysql并发情况下怎么解决（通过事务、隔离级别、锁）

**Redis**

redis数据结构有哪些

redis队列应用场景

redis和Memcached（支持数据持久化）

分布式使用场景（储存session等）