一． 范式

第一范式：列不可分
第二范式：有主键，保证完全依赖。
第三范式：无传递依赖(非主键列 A 依赖于非主键列 B，非主键列 B 依赖于主键的情况)， eg:订单表【Order】（OrderID， OrderDate， CustomerID，CustomerName， CustomerAddr， CustomerCity） 主键是 （OrderID），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。

二． 数据库索引

1. 索引的底层原理及优化

使用 B tree，可以在 O(logn)的时间内完成查询
1. b-tree（平衡多路查找树）
2. b+ tree （InnoDB 存储引擎）
对于 B+树，不管查找成功与否，每次查找都是走了一条从根到叶子结点的路径。
3. 为什么说 B+-tree 比 B 树更适合实际应用中操作系统的文件索引和数据库索引？

- B+ tree 的磁盘读写代价更低：B+ tree 的内部结点并没有指向关键字具体信息的指针(红色部分)，因此其内部结点相对 B 树更小。如果把所有同一内部结点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多。一次性读入内存中的需要查找的关键字也就越多，相对来说 IO 读写次数也就降低了；
- B+ tree 的查询效率更加稳定：由于内部结点并不是最终指向文件内容的结点，而只是叶子结点中关键字的索引，所以，任何关键字的查找必须走一条从根结点到叶子结点的路。所有关键字查询的路径长度相同，导致每一个数据的查询效率相当；
- 数据库索引采用 B+树而不是 B 树的主要原因：B+树只要遍历叶子节点就可以实现整棵树的遍历，而且在数据库中基于范围的查询是非常频繁的，而 B 树只能中序遍历所有节点，效率太低。

4. 文件索引和数据库索引为什么使用 B+树?

文件与数据库都是需要较大的存储，也就是说，它们都不可能全部存储在内存中，故需要存储到磁盘上。而所谓索引，则为了数据的快速定位与查找，那么索引的结构组织要尽量减少查找过程中磁盘 I/O 的存取次数，因此 B+树相比 B 树更为合适。数据库系统巧妙利用了局部性原理与磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次 I/O 就可以完全载入，而红黑树这种结构，高度明显要深的多，并且由于逻辑上很近的节点(父子)物理上可能很远，无法利用局部性。最重要的是，B+树还有一个最大的好处 :方便扫库。B 树必须用中序遍历的方法按序扫库，而 B+树直接从叶子结点挨个扫一遍就完了，B+树支持 range-query（查询某个范围内的数据，先通过索引找到 min，再依次向后查询）非常方便，而 B 树不支持，这是数据库选用 B+树的最主要原因。

2. 索引的优点

- 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；
- 加速表和表之间的连接；
- 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；

3. 什么情况下设置了索引但无法使用？

- 以“%(表示任意 0 个或多个字符)”开头的 LIKE 语句，模糊匹配；
- OR 语句前后没有同时使用索引；
- 数据类型出现隐式转化（如 varchar 不加单引号的话可能会自动转换为 int 型）；
- 对于多列索引，必须满足最左匹配原则 (eg：多列索引 col1、col2 和 col3，则 索引生效的情形包括 col1 或 col1，col2 或 col1，col2，col3)。

4. 什么样的字段适合创建索引？

- 经常作查询选择的字段
- 经常作表连接的字段
- 经常出现在 order by, group by, distinct 后面的字段

5. 注意事项

- 非空字段：应该指定列为 NOT NULL，除非你想存储 NULL。在 mysql 中，含有空值的列很难进行查询优化，因为它们使得索引、索引的统计信息以及比较运算更加复杂。你应该用 0、一个特殊的值或者一个空串代替空值；
- 取值离散大的字段：(变量各个取值之间的差异程度）的列放到联合索引的前面，可以通过 count()函数查看字段的差异值，返回值越大说明字段的唯一值越多字段的离散程度高；
- 索引字段越小越好：数据库的数据存储以页为单位一页存储的数据越多一次 IO 操作获取的数据越大效率越高。

6. 缺点

- 时间方面：创建索引和维护索引要耗费时间，具体地，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；
- 空间方面：索引需要占物理空间。

7. 分类

- 普通索引和唯一性索引：索引列的值的唯一性
- 单个索引和复合索引：索引列所包含的列数

- 聚簇索引与非聚簇索引: 聚集索引。表数据按照索引的顺序来存储的，也就是说索引项的顺序与表中记录的物理顺序一致。非聚簇索引：表数据存储的顺序与索引顺序无关。

8. 主键、自增主键、主键索引与唯一索引概念区别

- 主键：指字段 唯一、不为空值的列；
- 主键索引：指的就是主键，主键是索引的一种，是唯一索引的特殊类型。创建主键的时候，数据库默认会为主键创建一个唯一索引；
- 自增主键：字段类型为数字、自增、并且是主键；
- 唯一索引：索引列的值必须唯一，但允许有空值；

9. 主键就是聚集索引吗？主键和索引有什么区别？

主键是一种特殊的唯一性索引，其可以是聚集索引，也可以是非聚集索引

三． 数据库事务

1. 事务的特征

- 原子性(Atomicity)：事务所包含的一系列数据库操作要么全部成功执行，要么全部回滚；
- 一致性(Consistency)：事务的执行结果必须使数据库从一个一致性状态到另一个一致性状态；
- 隔离性(Isolation)：并发执行的事务之间不能相互影响；
- 持久性(Durability)：事务一旦提交，对数据库中数据的改变是永久性的；

2. 问题

- 脏读：一个事务读取了另一个事务未提交的数据；
- 不可重复读：不可重复读的重点是修改，同样条件下两次读取结果不同，也就是说，被读取的数据可以被其它事务修改；
- 幻读：幻读的重点在于新增或者删除，同样条件下两次读出来的记录数不一样；

3. 隔离级别

- READ UNCOMMITTED：最低级别的隔离，通常又称为 dirty read，它允许一个事务读取另一个事务还没 commit 的数据，这样可能会提高性能，但是会导致脏读问题；
- READ COMMITTED：在一个事务中只允许对其它事务已经 commit 的记录可见，该隔离级别不能避免不可重复读问题；
- REPEATABLE READ：在一个事务开始后，其他事务对数据库的修改在本事务中不可见，直到本事务 commit 或 rollback。但是，其他事务的 insert/delete 操作对该事务是可见的，也就是说，该隔离级别并不能避免幻读问题。在一个事务中重复 select 的结果一样，除非本事务中 update 数据库。
- SERIALIZABLE：最高级别的隔离，只允许事务串行执行。

4. mysql 事务

MySQL 的事务支持不是绑定在 MySQL 服务器本身，而是与存储引擎相关：

- MyISAM：不支持事务，用于只读程序提高性能；
- InnoDB：支持 ACID 事务、行级锁、并发；
- Berkeley DB：支持事务;

四． 数据库优化

1. SQL 语句及索引的优化

通过 mysql 慢查询日志对效率较低的 sql 进行监控
explain 语句
优化：

- 优化 insert 语句：一次插入多值；
- 应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描；
- 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描；
- 优化嵌套查询：子查询可以被更有效率的连接(Join)替代；
- 很多时候用 exists 代替 in 是一个好的选择。

2. 数据表结构的优化

合适数据类型：

- 使用较小的数据类型解决问题；
- 使用简单的数据类型(mysql 处理 int 要比 varchar 容易)；
- 尽可能的使用 not null 定义字段；
- 尽量避免使用 text 类型，非用不可时最好考虑分表；

表的范式的优化：表的设计应该遵循三大范式
表的垂直拆分：

- 把不常用的字段单独放在同一个表中；
- 把大字段独立放入一个表中；
- 把经常使用的字段放在一起；

表的水平拆分：
表的水平拆分用于解决数据表中数据过大的问题，水平拆分每一个表的结构都是完全一致的。一般地，将数据平分到 N 张表中的常用方法包括以下两种：

- 对 ID 进行 hash 运算，如果要拆分成 5 个表，mod(id,5)取出 0~4 个值；
- 针对不同的 hash ID 将数据存入不同的表中；

3. 系统配置优化

- 操作系统配置的优化：增加 TCP 支持的队列数
- mysql 配置文件优化：InnoDB 缓存池设置(innodb_buffer_pool_size，推荐总内存的 75%)和缓存池的个数（innodb_buffer_pool_instances）

4. 硬件优化

- CPU：核心数多并且主频高的
- 内存：增大内存

- 磁盘配置和选择：磁盘性能

五． Nosql

优点：

- 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)；
- 支持丰富数据类型，支持 string，list，set，sorted set，hash；
- 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；
- 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。

六． 存储过程

- 存储过程只在创建时进行编译，所以使用存储过程可提高数据库执行效率；
- 减少网络传输，在客户端调用一个存储过程当然比执行一串 SQL 传输的数据量要小；
- 通过存储过程能够使没有权限的用户在控制之下间接地存取数据库，从而确保数据的安全

demo：

```
create procedure procedure2(

out p1 decimal(8,2),

out p2 decimal(8,2),

in p3 int

)

begin
select sum(uid) into p1 from user where order_name = p3;
select avg(uid) into p2 from user ;
end ;
```

七． 触发器

触发事件：INSERT、UPDATE、DELETE
触发时间：
before：表示在数据库动作之前触发器执行;
after：表示在数据库动作之后触发器执行。

```
alter trigger t_InsertStu
on Student
after insert
as
begin
    declare @Name varchar(50)
    declare @Age int
    select @Name=Name,@Age=Age from inserted
    insert into StudentLog values(0,'最新添加的学生是：'+@Name+'他的年龄是：'+CONVERT(nvarchar(10),@Age),GETDATE())
end
insert into Student values('许瑞',21,getdate())
```


八． drop、delete 与 truncate 的区别

- Delete 用来删除表的全部或者一部分数据行，执行 delete 之后，用户需要提交(commit)或者回滚(rollback)来执行删除或者撤销删除， delete 命令会触发这个表上所有的 delete 触发器；
- Truncate 删除表中的所有数据，这个操作不能回滚，也不会触发这个表上的触发器，TRUNCATE 比 delete 更快，占用的空间更小；
- Drop 命令从数据库中删除表，所有的数据行，索引和权限也会被删除，所有的 DML 触发器也不会被触发，这个命令也不能回滚。

九． 视图和游标

视图：视图是一种虚拟的表，通常是有一个表或者多个表的行或列的子集，具有和物理表相同的功能，可以对视图进行增，删，改，查等操作；
游标：游标是对查询出来的结果集作为一个单元来有效的处理；比如用于对存储过程中的数据集进行按行的处理；

十． 悲观锁与乐观锁

悲观锁的特点是先获取锁，再进行业务操作，即“悲观”的认为所有的操作均会导致并发安全问题，因此要先确保获取锁成功再进行业务操作。必须在事务中使用；
乐观锁的特点先进行业务操作，只在最后实际更新数据时进行检查数据是否被更新过，若未被更新过，则更新成功；否则，失败重试。
使用场景：一般情况下，读多写少更适合用乐观锁，读少写多更适合用悲观锁。乐观锁在不发生取锁失败的情况下开销比悲观锁小，但是一旦发生失败回滚开销则比较大，因此适合用在取锁失败概率比较小的场景，可以提升系统并发性能。

面试题

1.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。
2.什么是存储过程？用什么来调用？
存储过程是一个预编译的 SQL 语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。

如果某次操作需要执行多次 SQL，使用存储过程比单纯 SQL 语句执行要快。可以用一个命令对象来调用存储过程。

3.索引的作用？和它的优点缺点是什么？

索引就是一种特殊的查询表，数据库的搜索引擎可以利用它加速对数据的检索。它很类似与现实生活中书的目录，不需要查询整本书内容就可以找到想要的数据。

索引可以是唯一的，创建索引允许指定单个列或者是多个列。缺点是它减慢了数据录入的速度，同时也增加了数据库的尺寸大小。

4.什么是内存泄漏？

一般我们所说的内存泄漏指的是堆内存的泄漏。堆内存是程序从堆中为其分配的，大小任意的，使用完后要显示释放内存。

在 Java 中，内存泄漏就是存在一些被分配的对象，首先，这些对象是可达的，即在有向图中，存在通路可以与其相连；其次，这些对象是无用的，即程序以后不会再使用这些对象。如果对象满足这两个条件，这些对象就可以判定为 Java 中的内存泄漏，这些对象不会被 GC 所回收，然而它却占用内存。

在 C++中，内存泄漏的范围更大一些。有些对象被分配了内存空间，然后却不可达，由于 C++中没有 GC，这些内存将永远收不回来。在 Java 中，这些不可达的对象都由 GC 负责回收，因此程序员不需要考虑这部分的内存泄露。

5.维护数据库的完整性和一致性，你喜欢用触发器还是自写业务逻辑？为什么？

我是这样做的，尽可能使用约束，如 check,主键，外键，非空字段等来约束，这样做效率最高，也最方便。

其次是使用触发器，这种方法可以保证，无论什么业务系统访问数据库都可以保证数据的完整新和一致性。

最后考虑的是自写业务逻辑，但这样做麻烦，编程复杂，效率低下。

6.表空间的管理方式有哪几种？

数据字典管理方式

本地文件管理方式

字典管理 ：在数据字典中管理表空间的空间分配。本地管理 ：在每个数据文件中使用位图来管理空间的分配。表空间中所有区的分配信息都保存在该表空间对应的数据文件的头部。

7.说说索引的组成?

索引列、rowid

8.SQL 里面 IN 比较快还是 EXISTS 比较快？

EXISTS 比较快因为 EXISTS 返回一个 Boolean 型而 IN 返回一个值。

9.mysql 如何实现分页查询？

limit 基本实现方式

一般情况下，客户端通过传递 pageNo（页码）、pageSize（每页条数）两个参数去分页查询数据库中的数据，在数据量较小（元组百/千级）时使用 MySQL自带的 `limit` 来解决这个问题：

```
收到客户端{pageNo:1,pagesize:10}
select * from table limit (pageNo-1)*pageSize, pageSize;
收到客户端{pageNo:5,pageSize:30}
select * from table limit (pageNo-1)*pageSize,pageSize;
```

建立主键或者唯一索引

在数据量较小的时候简单的使用 `limit` 进行数据分页在性能上面不会有明显的缓慢，但是数据量达到了 **万级到百万级** sql语句的性能将会影响数据的返回。这时需要利用主键或者唯一索引进行数据分页；

```
假设主键或者唯一索引为 good_id
收到客户端{pageNo:5,pagesize:10}
select * from table where good_id > (pageNo-1)*pageSize limit pageSize;
–返回good_id为40到50之间的数据
```

基于数据再排序

当需要返回的信息为顺序或者倒序时，对上面的语句基于数据再排序。order by ASC/DESC 顺序或倒序 默认为顺序

```
select * from table where good_id > (pageNo-1)*pageSize order by good_id limit pageSize;
–返回good_id为40到50之间的数据,数据依据good_id顺序排列
```

10.MySQL 数据库有哪些类型的索引？

MySQL 主要提供 2 种方式的索引：B-Tree 索引，Hash 索引。

B 树索引具有范围查找和前缀查找的能力，对于有 N 节点的 B 树，检索一条记录的复杂度为 O(LogN)。相当于二分查找。

哈希索引只能做等于查找，但是无论多大的 Hash 表，查找复杂度都是 O(1)。

显然，如果值的差异性大，并且以等值查找（=、 <、>、in）为主，Hash 索引是更高效的选择，它有 O(1)的查找复杂度。

如果值的差异性相对较差，并且以范围查找为主，B 树是更好的选择，它支持范围查找。

11.mysql 数据库的两种引擎？

InnoDB：聚簇索引，支持事务和行级锁

MyISAM：非聚簇索引，不支持事务，只支持行级锁