搜集题目中发现：

* 事务是一个大头，包括事务锁；
* 索引是一个大头，包括索引建立、实现以及索引失效
* mysql数据库的底层存储引擎及存储的索引，前者索引实现和红黑树等的区别

建议各位自己查找这三个大类的资料进行系统学习，下面整理的资料都只是数据库的基础知识，并且不成体系。

**（1）什么是存储过程？有哪些优缺点？**

存储过程是一些预编译的SQL语句。

更加直白的理解：存储过程可以说是一个记录集，它是由一些T-SQL语句组成的代码块，这些T-SQL语句代码像一个方法一样实现一些功能（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。

* 存储过程是一个预编译的代码块，执行效率比较高
* 一个存储过程替代大量T\_SQL语句 ，可以降低网络通信量，提高通信速率
* 可以一定程度上确保数据安全

*存储过程科普：*<http://www.cnblogs.com/knowledgesea/archive/2013/01/02/2841588.html>

**（2）索引是什么？有什么作用以及优缺点？**

索引是对数据库表中一或多个列的值进行排序的结构，是帮助MySQL高效获取数据的数据结构。你也可以这样理解：索引就是加快检索表中数据的方法。数据库的索引类似于书籍的索引。在书籍中，索引允许用户不必翻阅完整个书就能迅速地找到所需要的信息。在数据库中，索引也允许数据库程序迅速地找到表中的数据，而不必扫描整个数据库。

MySQL数据库几个基本的索引类型：普通索引、唯一索引、主键索引、全文索引。

* 索引加快数据库的检索速度
* 索引降低了插入、删除、修改等维护任务的速度
* 唯一索引可以确保每一行数据的唯一性
* 通过使用索引，可以在查询的过程中使用优化隐藏器，提高系统的性能
* 索引需要占物理和数据空间

*索引科普*：<https://kb.cnblogs.com/page/45712/>

**（3）什么是事务？**

事务（Transaction）是并发控制的基本单位。所谓的事务，它是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。事务是数据库维护数据一致性的单位，在每个事务结束时，都能保持数据一致性。

*事务科普：*<https://blog.csdn.net/zdwzzu2006/article/details/5947062>

**（4）数据库的乐观锁和悲观锁是什么？**

数据库管理系统（DBMS）中的并发控制的任务是确保在多个事务同时存取数据库中同一数据时不破坏事务的隔离性和统一性以及数据库的统一性。乐观并发控制(乐观锁)和悲观并发控制（悲观锁）是并发控制主要采用的技术手段。

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性

**（5）使用索引查询一定能提高查询的性能吗？为什么**

通常,通过索引查询数据比全表扫描要快.但是我们也必须注意到它的代价。

索引需要空间来存储,也需要定期维护, 每当有记录在表中增减或索引列被修改时,索引本身也会被修改. 这意味着每条记录的INSERT,DELETE,UPDATE将为此多付出4,5 次的磁盘I/O. 因为索引需要额外的存储空间和处理,那些不必要的索引反而会使查询反应时间变慢.使用索引查询不一定能提高查询性能,索引范围查询(INDEX RANGE SCAN)适用于两种情况:

（1）基于一个范围的检索,一般查询返回结果集小于表中记录数的30%

1. 基于非唯一性索引的检索

**（6）简单说一说drop、delete与truncate的区别**

SQL中的drop、delete、truncate都表示删除，但是三者有一些差别。

* delete和truncate只删除表的数据不删除表的结构
* 速度,一般来说: drop> truncate >delete
* delete语句是dml，这个操作会放到rollback segement中，事务提交之后才生效；如果有相应的trigger，执行的时候将被触发.。truncate、drop是ddl，操作立即生效，原数据不放到rollback segment中，不能回滚。操作不触发trigger

**（7）drop、delete与truncate分别在什么场景之下使用？**

* 不再需要一张表的时候，用drop
* 想删除部分数据行时候，用delete，并且带上where子句
* 保留表而删除所有数据的时候用truncate

**（8）超键、候选键、主键、外键分别是什么？**

**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。

**（9）什么是视图？以及视图的使用场景有哪些？**

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增、改、查操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

* 只暴露部分字段给访问者，所以就建一个虚表，就是视图。
* 查询的数据来源于不同的表，而查询者希望以统一的方式查询，这样也可以建立一个视图，把多个表查询结果联合起来，查询者只需要直接从视图中获取数据，不必考虑数据来源于不同表所带来的差异

**（10）说一说三个范式**

**第一范式（1NF）**：数据库表中的字段都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整型、实数、字符型、逻辑型、日期型等。

**第二范式（2NF）**：数据库表中不存在非关键字段对任一候选关键字段的部分函数依赖（部分函数依赖指的是存在组合关键字中的某些字段决定非关键字段的情况），也即所有非关键字段都完全依赖于任意一组候选关键字。

**第三范式（3NF）**：在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递函数依赖则符合第三范式。所谓传递函数依赖，指的是如 果存在"A → B → C"的决定关系，则C传递函数依赖于A。因此，满足第三范式的数据库表应该不存在如下依赖关系： 关键字段 → 非关键字段 x → 非关键字段y。

*范式科普*<https://www.zhihu.com/question/24696366>

**（11）数据库事务的四个特性及含义**

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。

**（12）什么是视图？**

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

如下两种场景一般会使用到视图：

（1）不希望访问者获取整个表的信息，只暴露部分字段给访问者，所以就建一个虚表，就是视图。

（2）查询的数据来源于不同的表，而查询者希望以统一的方式查询，这样也可以建立一个视图，把多个表查询结果联合起来，查询者只需要直接从视图中获取数据，不必考虑数据来源于不同表所带来的差异。

注：这个视图是在数据库中创建的 而不是用代码创建的。

**（13）触发器的作用？**

触发器是一中特殊的存储过程，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。

**（14）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？**

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。

**（15）索引的作用？和它的优点缺点是什么？**

数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用B树及其变种B+树。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

为表设置索引要付出代价的：一是增加了数据库的存储空间，二是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。

**（16）对于建立索引的列，数据是均匀分布好还是不均匀分布好**

数据均匀分布比较好。最好每条记录的索引列的值各不相同，比如ID，这样查找效率最高。比如年龄，对于千万级的大表，分布还是太集中了，加上索引也没有太大用处。

**（17）MySQL查询记录时，每次只使用一个索引**

原因：

MySQL的查询SQL会经过查询优化器优化，生成执行计划，再进行查询；

优化过程中，如果要分析两个索引，那么整个查询时间将会比使用一个索引进行查询的时间长；

另一方面，不使用索引直接扫描全表的时间又比使用一个索引查询时间长，所以MySQL只使用一个索引；

如果有多个索引，MySQL会自己选择使用其中一个，当然，我们自己也可以强制让它使用某一个索引（select \* from table force index(ziduan1\_index) limit 2;(强制使用索引"ziduan1\_index")）。

**（18）索引失效的原因**

* 对单字段建立索引，where条件多字段
* 复合索引失效，where条件没有按照符合索引的顺序排列
* 对索引列运算，运算符包括（+、-、\*、/、！、<>、%、like'%\_'（%放在前面）、or、in、exist等），比如where a+1=c
* 类型错误，字段类型为varchar，where条件用number
* 对索引使用内部函数，这种情况下应该建立基于函数的索引
* is null 索引失效，is not null 索引有效
* where子句中使用参数，select id from t where num=@num

**（19）目前主要有两种方式实现ACID**

第一种是Write ahead logging，也就是日志式的方式。第二种是Shadow paging。

Write ahead logging（预写日志）：

1. 事务所引起的所有改动都要记录在日志中，在事务提交完成之前，所有的这些记录必须被写入硬盘；

2. 一个数据库的缓冲页直到被记入日志后才能发生修改。直到缓冲页对应的日志被写入硬盘后，该缓冲页才会存入硬盘；

3. 当缓冲页被修改和日志被更新修改时，必须加上互斥锁，以保证改动被记录到日志中的顺序与它发生的顺序是一致的。

以上规则的结果：

如果一条日志记录未被存入硬盘，则它可以被忽略，因为该日志中包含的改动一定属于未提交的事务。此外，这样的日志不能反映已持久化在数据库中的改动；

日志记录按顺序记录系统的改动。加锁协议（latch protocol）保证如果有对于同一页改动的两条日志记录，则两条记录的顺序反映对页发生改变的顺序。

**（20）事务隔离级别**

读未提交（Read uncommitted）：允许读取还未提交的改变了的数据。可能导致脏、幻、不可重复读。

读已提交（Read committed）：允许在并发事务已经提交后读取。可防止脏读，但是幻读和不可重复读仍可发生。

可重复读（Repeatable read）：对相同字段的多次读取是一致的，除非数据被事务本身改变。可防止脏、不可重复读，但是幻读仍可能发生。

串行化（Serializable）：完全服从数据库四种隔离级别，确保不发生脏、幻、不可重复读。速度最慢，它是通过完全锁定在事务中涉及的数据表来完成的，事务按顺序串行执行。

**（21）不考虑隔离性，会引发一下问题：**

脏读：一个事务读取了另一个事务改写但还未提交的数据，如果这些数据被回滚，则读到的数据是无效的。读到无效数据，导致查询结果前后不一致。

不可重复读：在同一个事务中，读到另一个事务已经提交更新的数据。读到更新的数据，导致多次读取同一个数据返回的结果有所不同。

幻读：一个事务读取了几行记录后，另一个事务插入一些记录，幻读就发生了。在后来的查询中，第一个事务就会发现有些原来没有的记录。读取到新的记录

**MySQL的默认事务隔离级别是Repeatable read**

**（22）SQL 约束有哪几种？**

NOT NULL: 用于控制字段的内容一定不能为空（NULL）。 UNIQUE: 控件字段内容不能重复，一个表允许有多个 Unique 约束。PRIMARY KEY: 也是用于控件字段内容不能重复，但它在一个表只允许出现一个。FOREIGN KEY: 用于预防破坏表之间连接的动作，也能防止非法数据插入外键列，因为它必须是它指向的那个表中的值之一。CHECK: 用于控制字段的值范围。DEFAULT: 用于设置新记录的默认值。

**（23）MySQL分表、分区、分库**

分表：将一张表分成多个小表。

分区：一张表的数据分成多个区块，这些区块可以在同一个磁盘上，也可以在不同的磁盘上。

分库：一个数据库按城市（或者其他标准）分成多个数据库。

**（24）创建索引时需要注意什么？**

**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。

**（25）索引的分类**

**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**：聚簇索引按照数据的物理存储进行划分的。对于一堆记录来说，使用聚集索引就是对这堆记录进行堆划分，即主要描述的是物理上的存储。正是因为这种划分方法，导致聚簇索引必须是唯一的。聚集索引可以帮助把很大的范围，迅速减小范围。但是查找该记录，就要从这个小范围中Scan了；而非聚集索引是把一个很大的范围，转换成一个小的地图，然后你需要在这个小地图中找你要寻找的信息的位置，最后通过这个位置，再去找你所需要的记录。

**（26）游标是什么？**

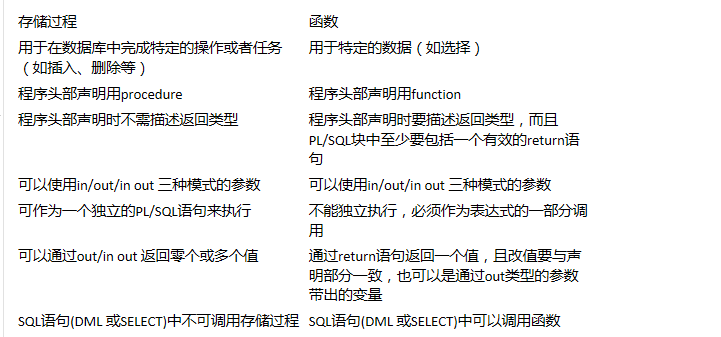
　　游标是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

在操作mysql的时候，我们知道MySQL检索操作返回一组称为结果集的行。这组返回的行都是与 SQL语句相匹配的行（零行或多行）。使用简单的 SELECT语句，例如，没有办法得到第一行、下一行或前 10行，也不存在每次一行地处理所有行的简单方法（相对于成批地处理它们）。有时，需要在检索出来的行中前进或后退一行或多行。这就是使用游标的原因。游标（cursor）是一个存储在MySQL服务器上的数据库查询，它不是一条 SELECT语句，而是被该语句检索出来的结果集。在存储了游标之后，应用程序可以根据需要滚动或浏览其中的数据。游标主要用于交互式应用，其中用户需要滚动屏幕上的数据，并对数据进行浏览或做出更改。

**（27）悲观锁与乐观锁的应用场景**

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。

**（28）存储过程与函数的区别**



**（29）视图的优缺点**

优点：

1)对数据库的访问，因为视图可以有选择性的选取数据库里的一部分。

2)用户通过简单的查询可以从复杂查询中得到结果。

3)维护数据的独立性，试图可从多个表检索数据。

4)对于相同的数据可产生不同的视图。

缺点：

性能：查询视图时，必须把视图的查询转化成对基本表的查询，如果这个视图是由一个复杂的多表查询所定义，那么，那么就无法更改数据。

**（30）非关系型数据库和关系型数据库区别，优势比较?**

**非关系型数据库的优势：**

性能：NOSQL是基于键值对的，可以想象成表中的主键和值的对应关系，而且不需要经过SQL层的解析，所以性能非常高。

可扩展性：同样也是因为基于键值对，数据之间没有耦合性，所以非常容易水平扩展。

**关系型数据库的优势：**

复杂查询：可以用SQL语句方便的在一个表以及多个表之间做非常复杂的数据查询。

事务支持：使得对于安全性能很高的数据访问要求得以实现。

其他：

1.对于这两类数据库，对方的优势就是自己的弱势，反之亦然。

2.NOSQL数据库慢慢开始具备SQL数据库的一些复杂查询功能，比如MongoDB。

3.对于事务的支持也可以用一些系统级的原子操作来实现例如乐观锁之类的方法来曲线救国，比如Redis set nx。

**（31）什么是 内连接、外连接、交叉连接、笛卡尔积等?**

内连接: 只连接匹配的行

左外连接: 包含左边表的全部行（不管右边的表中是否存在与它们匹配的行），以及右边表中全部匹配的行

右外连接: 包含右边表的全部行（不管左边的表中是否存在与它们匹配的行），以及左边表中全部匹配的行

例如1：

SELECT a.,b. FROM luntan LEFT JOIN usertable as b ON a.username=b.username

例如2：

SELECT a.,b. FROM city as a FULL OUTER JOIN user as b ON a.username=b.username

全外连接: 包含左、右两个表的全部行，不管另外一边的表中是否存在与它们匹配的行。

交叉连接: 生成笛卡尔积－它不使用任何匹配或者选取条件，而是直接将一个数据源中的每个行与另一个数据源的每个行都一一匹配

例如：

SELECT type,pub\_name FROM titles CROSS JOIN publishers ORDER BY type

注意：

很多公司都只是考察是否知道其概念，但是也有很多公司需要不仅仅知道概念，还需要动手写sql,一般都是简单的连接查询。

**（32）varchar和char的使用场景?**

1.char的长度是不可变的，而varchar的长度是可变的。

定义一个char[10]和varchar[10]。

如果存进去的是‘csdn’,那么char所占的长度依然为10，除了字符‘csdn’外，后面跟六个空格，varchar就立马把长度变为4了，取数据的时候，char类型的要用trim()去掉多余的空格，而varchar是不需要的。

2.char的存取数度还是要比varchar要快得多，因为其长度固定，方便程序的存储与查找。

char也为此付出的是空间的代价，因为其长度固定，所以难免会有多余的空格占位符占据空间，可谓是以空间换取时间效率。

varchar是以空间效率为首位。

3.char的存储方式是：对英文字符（ASCII）占用1个字节，对一个汉字占用两个字节。

varchar的存储方式是：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。

**（33）SQL语言分类**

SQL语言共分为四大类：

数据查询语言DQL

数据操纵语言DML

数据定义语言DDL

数据控制语言DCL。

1. 数据查询语言DQL

数据查询语言DQL基本结构是由SELECT子句，FROM子句，WHERE子句组成的查询块：

SELECT

FROM

WHERE

2 .数据操纵语言DML

数据操纵语言DML主要有三种形式：

1) 插入：INSERT

2) 更新：UPDATE

3) 删除：DELETE

3. 数据定义语言DDL

数据定义语言DDL用来创建数据库中的各种对象-----表、视图、索引、同义词、聚簇等如：

CREATE TABLE/VIEW/INDEX/SYN/CLUSTER

表 视图 索引 同义词 簇

DDL操作是隐性提交的！不能rollback

4. 数据控制语言DCL

数据控制语言DCL用来授予或回收访问数据库的某种特权，并控制数据库操纵事务发生的时间及效果，对数据库实行监视等。如：

1) GRANT：授权。

2) ROLLBACK [WORK] TO [SAVEPOINT]：回退到某一点。回滚---ROLLBACK；回滚命令使数据库状态回到上次最后提交的状态。其格式为：

SQL>ROLLBACK;

3) COMMIT [WORK]：提交。

在数据库的插入、删除和修改操作时，只有当事务在提交到数据

库时才算完成。在事务提交前，只有操作数据库的这个人才能有权看

到所做的事情，别人只有在最后提交完成后才可以看到。

提交数据有三种类型：显式提交、隐式提交及自动提交。下面分

别说明这三种类型。

(1) 显式提交

用COMMIT命令直接完成的提交为显式提交。其格式为：

SQL>COMMIT；

(2) 隐式提交

用SQL命令间接完成的提交为隐式提交。这些命令是：

ALTER，AUDIT，COMMENT，CONNECT，CREATE，DISCONNECT，DROP，

EXIT，GRANT，NOAUDIT，QUIT，REVOKE，RENAME。

(3) 自动提交

若把AUTOCOMMIT设置为ON，则在插入、修改、删除语句执行后，

系统将自动进行提交，这就是自动提交。其格式为：

SQL>SET AUTOCOMMIT ON；

**（34）为什么说B+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？**

1.B+的磁盘读写代价更低

B+的内部结点并没有指向关键字具体信息的指针。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。

2.B+tree的查询效率更加稳定

由于非终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。

**（35）什么是嵌套事务？**

嵌套是子事务套在父事务中执行，子事务是父事务的一部分，在进入子事务之前，父事务建立一个回滚点，叫save point，然后执行子事务，这个子事务的执行也算是父事务的一部分，然后子事务执行结束，父事务继续执行。重点就在于那个save point。看几个问题就明了了：

**如果子事务回滚，会发生什么？**

父事务会回滚到进入子事务前建立的save point，然后尝试其他的事务或者其他的业务逻辑，父事务之前的操作不会受到影响，更不会自动回滚。

**如果父事务回滚，会发生什么？**

父事务回滚，子事务也会跟着回滚！为什么呢，因为父事务结束之前，子事务是不会提交的，我们说子事务是父事务的一部分，正是这个道理。那么：

**事务的提交，是什么情况？**

是父事务先提交，然后子事务提交，还是子事务先提交，父事务再提交？答案是第二种情况，还是那句话，子事务是父事务的一部分，由父事务统一提交。