**什么是数据库？**

按照某种规则存放到计算机存储设备上可以被应用或用户访问的数据仓库.

**什么是数据库管理系统(DataBase Management System，简称DBMS)？**

指的是能够操作和管理数据库的软件，开发者可以通过使用数据库管理系统来访问数据库中的数据.

**主键**：指的是一个列或多列的组合，其值能唯一地标识表中的每一行，通过它可强制表的实体完整性.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)，默认唯一性约束，只有主键才能设置自动增长，自动增长一定是主键，主键不一定自动增长.

**超键**：在关系中能唯一标识元素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**：不含有多余属性的超键称为候选键，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.

**表**

数据在数据库中是以表的方式存储的，在数据库中可以创建多个表

**SQL**

SQL全称是Structure Query Language(结构化查询语言)，是关系型数据语言的标准，各大关系型数据库厂商都实现了这套标准，可以使用SQL通过数据库管理系统来操作数据库中的数据.因为SQL是标准，所以不论是哪一种数据库管理系统，都可以使用SQL来操作数据库中的数据.不过各个厂商在支持SQL标准的同时，在自己的数据库管理系统中做了一些扩展，这些扩展简称方言.

**SQL的分类**

**DDL**(Data Definition Language)：数据定义语言，用来操作数据库对象：库、表、列等；

**DML**(Data Manipulation Language)：数据操作语言，用来增删改数据库中的数据；

**DCL**(Data Control Language)：数据控制语言，用来设置访问权限和安全级别；

**DQL**(Data Query Language)：数据查询语言，用来查询数据库中的数据.

**CMD命令**

**启动和停止服务**

**cmd需使用管理员身份运行**

输入：net start mysql (如果失败的话，输入：net start mysql55)启动.

输入：net stop mysql (如果失败的话，输入：net start mysql55)停止.

**登录**

输入：mysql -u root -p

按下回车键，会提示输入密码，输入MySQL的登录密码即可.

如果要登录远程的mysql，需要输入下面的命令：mysql -h 远程mysql服务器的ip地址 -u root -p

例如：mysql -h 192.168.0.1 -u root –p

**修改密码**

mysql –u root –p 旧密码 password 新密码

**忘记密码**

停止mysql服务

输入：mysqld –skip-grant-tables 按下回车键(不要关闭该窗口)

新打开一个CMD，输入：mysql -u root -p 提示输入密码时，什么都不输入，直接按下回车键，进入mysql中.

输入：use mysql；

输入：update user set password=password(‘设置的新密码’) WHERE User=’root’

关闭两个命令提示符，在任务管理器中将mysqld.exe的进程杀掉，之后启动mysql服务登录即可.

**如果在命令提示符中登录mysql时提示”mysql不是内部或外部命令时”，需要将mysql的安装目录加入到环境变量path中.**

**DDL命令**

**创建数据库**：create database 数据库名；

**指定字符编码**：create database 数据库名 character set gbk；

**查看数据库**：show databases；

**查看数据的定义信息**：show create database 数据库名；

**修改数据库**(将数据库的编码修改为utf-8)：alter database study2 character set utf8；

**删除数据库**：drop database 数据库名；

**查看当前使用的数据库**：select database()；

**切换数据库**：use 数据库名；

**创建表**：

create table 表名(

字段1 字段类型，

字段2 字段类型，

...

字段n 字段类型

)；

**查看数据库中所有的表**：SHOW TABLES；

**查看表的字段信息**：DESC 表名；

**添加字段**：ALTER TABLE 表名ADD 字段名 数据类型；

**修改数据类型长度**：ALTER TABLE 表名MODIFY 字段名 varchar(30)；

**删除字段(一次只能删一列)**：ALTER TABLE 表名DROP 字段名；

**修改表名**：RENAME TABLE 表名 TO 新表名；

**查看表格的创建细节**： SHOW CREATE TABLE 表名；

**修改表的字符编码**：ALTER TABLE 表名 CHARACTER SET 字符编码；

**修改字段名**：ALTER TABLE 表名 CHANGE 字段名 新字段名 varchar(20)；

**删除表**：DROP TABLE 表名；

**删除表数据(表结构依然存在)**：DELETE TABLE 表名；

**字段数据类型**

**int**：整型

**double**：浮点型，例如double(5，2)表示最多5位，其中必须有2位小数，即最大值为999.99；

**char**：固定长度字符串类型； char(10) 如果不足10位则会自动补足10位

**varchar**：可变长度字符串类型；varchar(10) 如果不足10位不会补足，**性能不如char高**

**text**：字符串类型；**适用于大文本内容**.

**blob**：字节类型；

**date**：日期类型，格式为：yyyy-MM-dd；

**time**：时间类型，格式为：hh：mm：ss

**timestamp**：时间戳类型 yyyy-MM-dd hh：mm：ss 会自动赋值

**datetime**：日期时间类型 yyyy-MM-dd hh：mm：ss

**DML**

**INSERT**

语法： INSERT INTO 表名(字段名1，字段名2 …) VALUES (值1，值2…)；

注意：字段名与值的类型、个数、顺序要对应上.

值不要超出字段定义的长度，否则会报错.

如果插入空值，需要使用null

插入的日期和字符都使用单引号括起来.

**UPDATE**

语法：UPDATE 表名 SET 字段1=列值1，字段2=列值2 … WHERE 字段=值

**如果没有WHERE条件的话，会将表中所有的数据修改.**

**DELETE**

语法：DELETE FROM 表名 WHERE 字段=值

**drop，delete，truncate的区别**

三者都是删除的意思

速度 drop > truncate > delete

delete(删除部分)和truncate(删除所有)只删除表的数据不删除表的结构，drop删除表

想删除部分数据时，delete删除时要带上where语句，删除后的数据可以恢复

delete是DML语句，不会自动提交。drop/truncate都是DDL语句,执行后会自动提交。

保留表而将所有数据删除，如果和事务无关，用truncate

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.

**DQL**

**开发中不建议使用\*，因为会先将其编译成字段，然后再去查询，稍微影响一些性能，而且也不明确，在开发中建议将要查询的字段都写上.**

**在mysql中<>表示不等于，!=也表示不等于，两者的含义和使用方式是一样的.**

**使用between…and… 的范围包括条件的值，即相当于>=和<=.**

**and优先级高于or.**

**in：只要满足一项条件即可，完全可以采用or来表示，采用in会更简洁一些**

**MYSQL处理函数**

**lower**()：转换为小写

**upper**()：转换为大写

**substr**()：截取子串，该函数接收3个参数：substr(被截取字段名称，起始下标，截取长度)，起始下标从1开始.

**length**()：获取字段长度

**ifnull**()：空值处理，该函数接收两个参数： ifnull(字段名，替换值)，在数据库中，有Null参与数学运算的结果一定为Null；为了防止计算结果出现Null，建议先使用ifnull函数预先处理.

**case…when…then…else…end**：case 字段名 when 条件 then 处理 else 其它 end

**trim**()：去除首尾空格， MySQL默认去除字段后面的空格，原因：MySQL语法松散.

**round**()：四舍五入，该函数接收两个参数round(数字，保留的小数位数)

**rand**()：生成随机数，生成0~100之间的随机数：round(rand()\*100)

**date\_format**()：格式化日期，该函数接收两个参数，date\_format(日期类型数据，’日期格式’)

**str\_to\_date**()：字符串转日期，该函数接收两个参数，str\_to\_date(‘日期字符串’，’日期格式’).

其中日期格式如下：

%Y 代表四位的年份

%y 代表两位的年份

%m 代表月，格式(01 … 12)

%c 代表月，格式(1 … 12)

%d 代表日

%H 代表24小时制

%h 代表12小时制

%i 代表分种，格式(00 … 59)

%S或%s 代表秒 ， 格式(00..59)

**聚合函数**

**聚合函数在计算时会自动忽略空值，不用手动写sql将空值排除.**

**聚合函数不能直接写在where语句的后面.**

**sum**()：求和函数

**avg**()：取平均值函数

**max**()：取得最大值函数，日期也可以使用max()函数进行比较

**min**()：取得最小值函数

**count**()：取得数据总数

**distinct**：去除重复记录，将查询结果中某一字段的重复记录去除掉，distinct 字段名或 distinct字段名1， 字段名2…，distinct只能出现在所有字段最前面，后面如果有多个字段及为多字段联合去重.

**在有group by的DQL询句中，select语句后面只能跟聚合函数 + 参与分组的字段**

**order by语句只能放在group by语句后面**

**having过滤：**如果想对分组的数据进行过滤，需要使用having子句.

**能够在where后过滤的数据不要放到having中进行过滤，否则影响SQL询句的执行效率.**

**WITH ROLLUP 可以实现在分组统计数据基础上再进行相同的统计，使用有 WITH ROLLUP 子句的 GROUP BY 语句时，不能再使用 ORDER BY 语句对结果集进行排序**

**where和having区别**

where和having都是为了完成数据的过滤，它们后面都是添加条件；

where是在 group by之前完成过滤；

having是在group by之后完成过滤；

**select语句执行顺序**

from：将硬盘上的表文件加载到内存

where：将符合条件的数据筛选出来，生成一张新的临时表

group by：根据列中的数据种类，将当前临时表划分成若干个新的临时表

having：可以过滤掉group by生成的不符合条件的临时表

select：对当前临时表进行整列读取

order by：对select生成的临时表，进行重新排序，生成新的临时表

limit：对最终生成的临时表的数据行，进行截取

**连接查询**

在实际开发中，数据往往是同时存储在多张表中，这些表与表之间存在着关系，我们在检索数据的时候往往需要多张表联合起来检索，这种多表联合检索被称为链接查询.

**如果多表进行连接查询时没有任何条件，最终的结果是多表结果数量乘积的现象被称为笛卡尔积**

**内连接与外连接的区别**

**内连接**：指连接结果仅包含符合连接条件的行，参与连接的两个表都应该符合连接条件.

**外连接**：连接结果不仅包含符合连接条件的行同时也包含自身不符合条件的行.包括左外连接、右外连接和全外连接(mysql不支持).

**左外连接**：左边表数据行全部保留，右边表保留符合连接条件的行.

**右外连接**：右边表数据行全部保留，左边表保留符合连接条件的行.

**子查询**

select 语句嵌套 select 语句被称为子查询，select子句可出现在select(使用较少)、from、where关键字后面

**union**：将查询的结果集合并，合并结果集时查询字段的个数必须一致.

**limit**：获取一表前几条戒中间某几行数据，主要用来分页处理，limit关键字只在MySQL中起作用.

用法：limit(m，n)

m：记录开始的index，默认从 0 开始，表示第一条记录；

n ：指从第 m+1 条开始，取 n 条；

**存储引擎**

存储引擎是通过采用不同的技术将数据存储在文件或内存中，不同的技术有不同的存储机制，其功能和能力也不同，通过选择不同的技术，可以获得特殊的速度或功能，从而提高应用的性能.存储引擎是MySQL特有的，在不同的业务场景下选择不同的存储引擎，这样能够发挥MySQL的最佳性能

**常用的存储引擎**

**MyISAM：**节省数据库空间，当数据读远大于修改时，可以使用该存储引擎

**InnoDB：**支持事务，如果数据修改较多时，可以使用该存储引擎

**MEMORY：**存储在内存中，速度快，如果存储非永久性数据时，可以使用该存储引擎

**MyISAM与InnoDB的区别**

**InnoDB支持事务，MyISAM不支持**，对于InnoDB每一条SQL语言都默认封装成事务，自动提交，这样会影响速度，所以最好把多条SQL语言放在begin和commit之间，组成一个事务；

**InnoDB支持外键，而MyISAM不支持**.对一个包含外键的InnoDB表转为MYISAM会失败；

**InnoDB是聚集索引**，数据文件是和索引绑在一起的，必须要有主键，通过主键索引效率很高.但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据.因此，主键不应该过大，因为主键太大，其他索引也都会很大.而**MyISAM是非聚集索引**，数据文件是分离的，索引保存的是数据文件的指针.主键索引和辅助索引是独立的.

**InnoDB不保存表的具体行数**，执行select count(\*) from table时需要全表扫描.而MyISAM用一个变量保存了整个表的行数，执行上述语句时只需要读出该变量即可，速度很快；

**Innodb不支持全文索引，而MyISAM支持全文索引，查询效率上MyISAM要高；**

**备份的目的**

做灾难恢复：对损坏的数据进行恢复和还原

需求改变：因需求改变而需要把数据还原到改变以前

测试：测试新功能是否可用

**备份需要考虑的问题**

可以容忍丢失多长时间的数据；

恢复数据要在多长时间内完；

恢复的时候是否需要持续提供服务；

恢复的对象，是整个库，多个表，还是单个库，单个表.

**MyISAM不支持热备，InnoDB支持热备，但是需要专门的工具**

**数据的备份类型**

**完全备份**指的是备份整个数据集(即整个数据库)

**部分备份**指的是备份部分数据集(例如：只备份一个表)

**部分备份又分为增量备份和差异备份**

**增量备份**指的是备份自上一次备份以来(增量或完全)以来变化的数据；特点：节约空间、还原麻烦

**差异备份**指的是备份自上一次完全备份以来变化的数据 特点：浪费空间、还原比增量备份简单

**建议使用完全备份和差异备份的结合**

**备份数据的方式**

**热备份**指的是当数据库进行备份时， 数据库的读写操作均不是受影响

**温备份**指的是当数据库进行备份时， 数据库的读操作可以执行， 但是不能执行写操作

**冷备份**指的是当数据库进行备份时， 数据库不能进行读写操作， 即数据库要下线

**物理备份一般就是通过tar，cp等命令直接打包复制数据库的数据文件达到备份的效果**

**逻辑备份一般就是通过特定工具从数据库中导出数据并另存备份(逻辑备份会丢失数据精度)**

**备份工具**

**mysqldump**：逻辑备份工具， 适用于所有的存储引擎，支持温备、完全备份、部分备份、对于InnoDB存储引擎支持热备

**cp，tar等归档复制工具**： 物理备份工具， 适用于所有的存储引擎，冷备、完全备份、部分备份

**lvm2 snapshot**：几乎热备，借助文件系统管理工具进行备份

**mysqlhotcopy**：名不副实的的一个工具，几乎冷备，仅支持MyISAM存储引擎

**xtrabackup**：一款非常强大的InnoDB/XtraDB热备工具，支持完全备份、增量备份，由percona提供

**mysqldump基本语法：**

**mysqldump常用来做温备，所以我们首先需要对想备份的数据施加读锁**

**备份一个数据库**

mysqldump -u username -p dbname table1 table2 ... > backup.sql

**备份多个数据库**

mysqldump -u username -p --databases dbname2 dbname2 > backup.sql

**备份所有数据库**

mysqldump -u username -p --all-databases > backup.sql

**数据还原**

mysql -u root -p dbname < backup.sql

**什么是事务**

事务的英文是transaction，事务可以保证多个操作原子性，对于数据库来说，事务可以保证批量的DML要么全成功，要么全失败.通常一个事务对应一个完整的业务.

**事务控制语句**

**BEGIN或START TRANSACTION：**显式地开启一个事务；

**COMMIT**也可以使用**COMMIT WORK**，不过二者是等价的.COMMIT会提交事务，并使已对数据库进行的所有修改成为永久性的；

**ROLLBACK**也可以使用**ROLLBACK WORK**，不过二者是等价的.回滚会结束用户的事务，并撤销正在进行的所有未提交的修改；

**SAVEPOINT identifier：** SAVEPOINT允许在事务中创建一个保存点，一个事务中可以有多个SAVEPOINT；

**RELEASE SAVEPOINT identifier**：删除一个事务的保存点，当没有指定的保存点时，执行该语句会抛出一个异常；

**ROLLBACK TO identifier：**把事务回滚到标记点；

**SET TRANSACTION：**用来设置事务的隔离级别.InnoDB存储引擎提供事务的隔离级别有READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ和SERIALIZABLE.

**SET AUTOCOMMIT=0 禁止自动提交**

**SET AUTOCOMMIT=1 开启自动提交**

**事务具有四个特征ACID**

**原子性**(Atomicity)，事务开始后所有操作，要么全部做完，要么全部不做，不可能停滞在中间环节.事务执行过程中出错，会回滚到事务开始前的状态，所有的操作就像没有发生一样.也就是说事务是一个不可分割的整体，就像化学中学过的原子，是物质构成的基本单位.

**一致性**(Consistency)，事务的开始和结束，数据都必须保持一致状态.

**隔离性**(Isolation)，同一时间，只允许一个事务请求同一数据，不同的事务之间彼此没有任何干扰；

**持久性**(Durability)，在事务完成之后，该事务对数据库所作的更改将持久地保存在数据库中，并不会被回滚；

**事务的隔离级别**

**read uncommitted** 读未提交

事务A和事务B，事务A未提交的数据，事务B可以读取，这里读取到的数据叫做“脏数据”，该级别最低，一般只是理论上存在，数据库的默认隔离级别都高于该级别.

**read committed** 读已提交

事务A和事务B，事务A提交的数据，事务B才可读取到，换句话说：对方事务提交之后的数据，当前事务才可读取到，可以避免读取“脏数据”，但是改级别会有“不可重复读”的问题

**repeatable read** 重复读

MySQL默认的隔离级别是重复读，该级别可以达到“重复读”的效果，但是会有“幻读”的问题

**serializable** 串行化

事务A和事务B，事务A在操作数据库表中数据的时候，事务B只能排队等待，这样保证了同一个时间点上只有一个事务操作数据库，该级别可以解决“幻读”的问题.但是这种级别一般很少使用，因为吞吐量太低，用户体验不好.

**脏读**：事务A读取了事务B更新的数据，然后B回滚操作，那么A读取到的数据是脏数据

**不可重复读**：事务A多次读取同一数据，事务B在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果不一致.

**幻读**：事务A读取数据，此时事务B修改了这条数据，但是事务A读取的还是之前的旧数据的内容，就好像发生了幻觉一样，这就叫幻读.

**更新丢失**：当两个或多个事务选择同一行，然后基于最初选定的值更新该行时，由于每个事务都不知道其他事务的存在，就会发生丢失更新问题，最后的更新覆盖了其他事务所做的更新

**不可重复读的和幻读很容易混淆，不可重复读侧重于修改，幻读侧重于新增或删除.解决不可重复读的问题只需锁住满足条件的行，解决幻读需要锁表**

Read uncommitted (读未提交)：最低级别，任何情况都无法保证.

Read committed (读已提交)：可避免脏读的发生.

Repeatable read (可重复读)：可避免脏读、不可重复读的发生.

Serializable (串行化)：可避免脏读、不可重复读、幻读的发生.

**查看当前会话的隔离级别：select @@tx\_isolation；**

**设置事务隔离级别：**SET [ SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL [READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE ]

SESSION：本次会话

GLOBAL：全局

**索引的作用**

索引相当于一本字典目录，能够提高数据库的查询效率，表中每一个字段都可添加索引.主键会自动添加索引，在查询时，如果能通过主键查询的尽量使用主键查询，效率高.

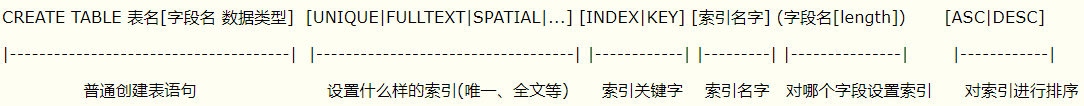
**什么情况下适合添加索引**

该字段数据量庞大；

该字段很少的DML操作(由于索引也需要维护，DML操作多的话，也影响检索效率)；

该字段经常出现在where条件中；

**创建索引：create index 索引名 on 表名(列名)；**

****

**删除索引：drop index 索引名 on 表名；**

**索引的分类**

**单列索引**

一个索引只包含单个列，但一个表中可以有多个单列索引.

**组合索引**

在表中的多个字段组合上创建的索引，只有在查询条件中使用了这些字段的左边字段时，索引才会被使用，使用组合索引时遵循最左前缀集合

**全文索引**

只有在MyISAM引擎上才能使用，只能在CHAR，VARCHAR，TEXT类型字段上使用全文索引，在一堆文字中，通过其中的某个关键字等，就能找到该字段所属的记录行.

**空间索引**

空间索引是对空间数据类型的字段建立的索引，MySQL中的空间数据类型有四种，GEOMETRY、POINT、LINESTRING、POLYGON.

**单列索引的分类**

**普通索引**： MySQL中基本索引类型，没有什么限制，允许在定义索引的列中插入重复值和空值，纯粹为了查询数据更快一点.

**唯一索引**：索引列中的值必须是唯一的，但是允许为空值，

**主键索引**：是一种特殊的唯一索引，不允许有空值.

**索引的优缺点**

**优点：**

大大加快数据的检索速度

创建唯一性索引，保证数据库表中每一行数据的唯一性

可以加速表和表之间的连接

**缺点：**

索引需要占物理空间.

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，降低了数据的维护速度.

**视图**

视图其实就是一个查询结果，视图的作用可以隐藏表的实现细节.

**创建视图**：create view 视图名称 as 查询语句；

**修改视图**：alter view 视图名称 as 查询语句

**删除视图**：drop view if exists 视图名称；

**三大范式**

**第一范式(1NF)**：数据表中的每一列(每个字段)必须是不可拆分的最小单元，也就是确保每一列的原子性

**第二范式(2NF)**：满足1NF后，要求表中的所有列，都必须依赖于主键，而不能有任何一列与主键没有关系，也就是说一个表只描述一件事情

**第三范式(3NF)**：必须先满足第二范式(2NF)，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，属性不依赖于其它非主属性

**三大范式的区分**

**1NF：字段不可分；**

**2NF：有主键，非主键字段依赖主键；**

**3NF：非主键字段不能相互依赖；**

第一范式和第二范式在于有没有分出两张表，第二范式是说一张表中包含了所种不同的实体属性，那么要必须分成多张表，第三范式是要求已经分成了多张表，那么一张表中只能有另一张表中的id(主键)，而不能有其他的任何信息(其他的信息一律用主键在另一表查询).

**数据库五大约束**

primary KEY：设置主键约束；

UNIQUE：设置唯一性约束，不能有重复值；

DEFAULT 默认值约束

NOT NULL：设置非空约束，该字段不能为空；

FOREIGN key ：设置外键约束.

**为什么要加锁**

**防止更新丢失，并不能单靠数据库事务控制器来解决，需要应用程序对要更新的数据加必要的锁来解决**

**共享锁(读锁、S锁)**：对某一资源加共享锁，自身可以读该资源，其他人也可以读该资源(也可以再继续加共享锁，即 共享锁可多个共存)，但无法修改.要想修改就必须等所有共享锁都释放完之后.语法为：SELECT \* FROM table\_name WHERE ... LOCK IN SHARE MODE

**排他锁(写锁、X锁)**：对某一资源加排他锁，自身可以进行增删改查，其他人无法进行任何操作，数据库的增删改操作默认都会加排他锁，而查询不会加任何锁。语法为：SELECT \* FROM table\_name WHERE ... FOR UPDATE

**更新锁(U锁)**：可以防止通常形式的死锁.如果两个事务获得了资源上的共享模式锁，然后试图同时更新数据，则两个事务需都要转换共享锁为排它锁，并且每个事务都等待另一个事务释放共享模式锁，因此发生死锁.若要避免这种潜 在的死锁问题，请使用更新锁.一次只有一个事务可以获得资源的更新锁.如果事务修改资源，则更新锁转换为排它锁.否则，锁转换为共享锁.

**悲观锁**(Pessimistic Concurrency Control，PCC)：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**(Optimistic Concurrency Control，OCC)：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.如果发现冲突了，则让返回用户错误的信息，让用户决定如何去做.一般的实现乐观锁的方式就是记录数据版本.

**锁的粒度**

**表级锁**：每次操作锁住整张表.开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低；

**行级锁**：每次操作锁住一行数据.开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高；

**页面锁**：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般.25个行锁可升级为一个页锁.

**数据库锁**：控制整个数据库操作

**死锁**：两个或者多个事务在同一资源上相互作用，并请求锁定对方占用的资源，从而导致恶性循环的现象。当多个事务试图以不同的顺序锁定资源时，就可能产生死锁。多个事务同时锁定同一个资源时，也会产生死锁。

**临时表在我们需要保存一些临时数据时是非常有用的.临时表只在当前连接可见，当关闭连接时，Mysql会自动删除表并释放所有空间.**

**SQL注入**

通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令

**防止SQL注入**

严格限制Web应用的数据库的操作权限，给此用户提供仅仅能够满足其工作的最低权限，从而最大限度的减少注入攻击对数据库的危害

对用户的输入进行校验，可以通过正则表达式，或限制长度

检查输入的数据是否具有所期望的数据格式，严格限制变量的类型

不要使用动态拼装sql，可以使用参数化的sql或者直接使用存储过程进行数据查询存取

对进入数据库的特殊字符(’”\尖括号&\*；等)进行转义处理，或编码转换.

不要把机密信息直接存放，加密或者hash掉密码和敏感的信息

应用的异常信息应该给出尽可能少的提示，最好使用自定义的错误信息对原始错误信息进行包装

sql注入的检测方法一般采取辅助软件或网站平台来检测

**mysql的字符集是latin1**

**SQL语句优化**

应尽量避免使用模糊查询，否则将导致引擎放弃使用索引而进行全表扫描

应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描

in 和 not in 也要慎用，否则会导致全表扫描

应尽量避免在 where 子句中对字段进行表达式或函数操作，这将导致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。

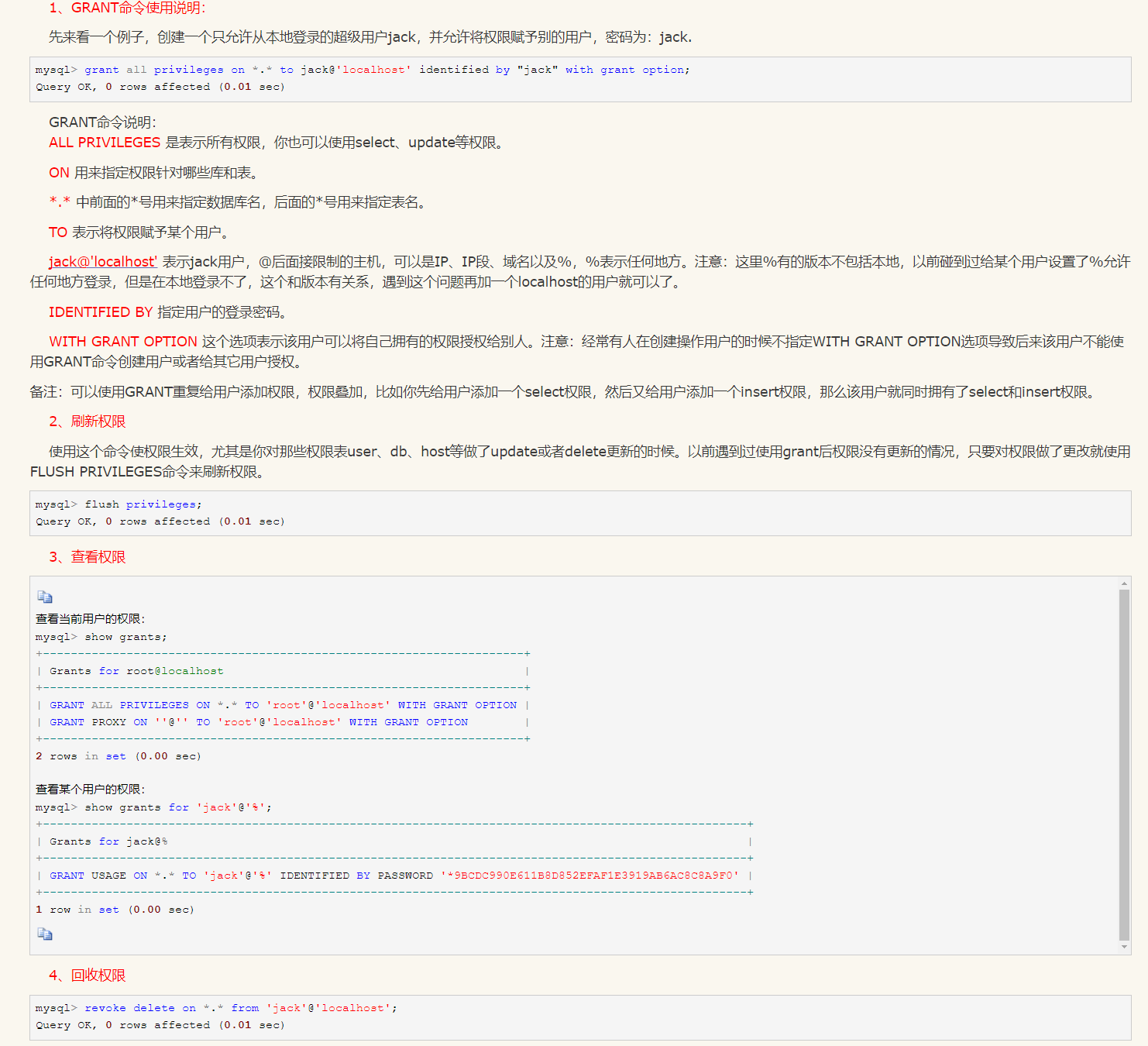
应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num is null

可以在num上设置默认值0，确保表中num列没有null值

很多时候用 exists 代替 in 是一个好的选择

用Where子句替换HAVING 子句 因为HAVING 只会在检索出所有记录之后才对结果集进行过滤

**权限**



**循环**

