<https://blog.csdn.net/qq_37884273/article/details/82228928>

**事物的四大特性(ACID)介绍一下?**

关系性数据库需要遵循ACID规则，具体内容如

1. **原子性：** 事务是最小的执行单位，不允许分割。事务的原子性确保动作要么全部完成，要么完全不起作用；
2. **一致性：** 执行事务前后，数据保持一致，多个事务对同一个数据读取的结果是相同的；
3. **隔离性：** 并发访问数据库时，一个用户的事务不被其他事务所干扰，各并发事务之间数据库是独立的；
4. **持久性：** 一个事务被提交之后。它对数据库中数据的改变是持久的，即使数据库发生故障也不应该对其有任何影响。

**视图**

**为什么要使用视图？什么是视图？**

为了提高复杂SQL语句的复用性和表操作的安全性，MySQL数据库管理系统提供了视图特性。所谓视图，本质上是一种虚拟表，在物理上是不存在的，其内容与真实的表相似，包含一系列带有名称的列和行数据。但是，视图并不在数据库中以储存的数据值形式存在。行和列数据来自定义视图的查询所引用基本表，并且在具体引用视图时动态生成。

视图使开发者只关心感兴趣的某些特定数据和所负责的特定任务，只能看到视图中所定义的数据，而不是视图所引用表中的数据，从而提高了数据库中数据的安全性。

**视图有哪些特点？**

视图的特点如下:

* 视图的列可以来自不同的表，是表的抽象和在逻辑意义上建立的新关系。
* 视图是由基本表(实表)产生的表(虚表)。
* 视图的建立和删除不影响基本表。
* 对视图内容的更新(添加，删除和修改)直接影响基本表。
* 当视图来自多个基本表时，不允许添加和删除数据。

视图的操作包括创建视图，查看视图，删除视图和修改视图。

**视图的使用场景有哪些？**

视图根本用途：简化sql查询，提高开发效率。如果说还有另外一个用途那就是兼容老的表结构。

下面是视图的常见使用场景：

* 重用SQL语句；
* 简化复杂的SQL操作。在编写查询后，可以方便的重用它而不必知道它的基本查询细节；
* 使用表的组成部分而不是整个表；
* 保护数据。可以给用户授予表的特定部分的访问权限而不是整个表的访问权限；
* 更改数据格式和表示。视图可返回与底层表的表示和格式不同的数据。

**视图的优点**

1. 查询简单化。视图能简化用户的操作
2. 数据安全性。视图使用户能以多种角度看待同一数据，能够对机密数据提供安全保护
3. 逻辑数据独立性。视图对重构数据库提供了一定程度的逻辑独立性

**视图的缺点**

1. 性能。数据库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，即使是视图的一个简单查询，数据库也把它变成一个复杂的结合体，需要花费一定的时间。
2. 修改限制。当用户试图修改视图的某些行时，数据库必须把它转化为对基本表的某些行的修改。事实上，当从视图中插入或者删除时，情况也是这样。对于简单视图来说，这是很方便的，但是，对于比较复杂的视图，可能是不可修改的

这些视图有如下特征：1.有UNIQUE等集合操作符的视图。2.有GROUP BY子句的视图。3.有诸如AVG\SUM\MAX等聚合函数的视图。 4.使用DISTINCT关键字的视图。5.连接表的视图（其中有些例外）

**什么是游标？**

游标是系统为用户开设的一个数据缓冲区，存放SQL语句的执行结果，每个游标区都有一个名字。用户可以通过游标逐一获取记录并赋给主变量，交由主语言进一步处理。

**存储过程与函数**

**什么是存储过程？有哪些优缺点？**

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，就是说只需要创建一次，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。

**优点**

1）存储过程是预编译过的，执行效率高。

2）存储过程的代码直接存放于数据库中，通过存储过程名直接调用，减少网络通讯。

3）安全性高，执行存储过程需要有一定权限的用户。

4）存储过程可以重复使用，减少数据库开发人员的工作量。

**缺点**

1）调试麻烦，但是用 PL/SQL Developer 调试很方便！弥补这个缺点。

2）移植问题，数据库端代码当然是与数据库相关的。但是如果是做工程型项目，基本不存在移植问题。

3）重新编译问题，因为后端代码是运行前编译的，如果带有引用关系的对象发生改变时，受影响的存储过程、包将需要重新编译（不过也可以设置成运行时刻自动编译）。

4）如果在一个程序系统中大量的使用存储过程，到程序交付使用的时候随着用户需求的增加会导致数据结构的变化，接着就是系统的相关问题了，最后如果用户想维护该系统可以说是很难很难、而且代价是空前的，维护起来更麻烦。

**触发器**

**什么是触发器？触发器的使用场景有哪些？**

触发器是用户定义在关系表上的一类由事件驱动的特殊的存储过程。触发器是指一段代码，当触发某个事件时，自动执行这些代码。

使用场景

* 可以通过数据库中的相关表实现级联更改。
* 实时监控某张表中的某个字段的更改而需要做出相应的处理。
* 例如可以生成某些业务的编号。
* 注意不要滥用，否则会造成数据库及应用程序的维护困难。
* 大家需要牢记以上基础知识点，重点是理解数据类型CHAR和VARCHAR的差异，表存储引擎InnoDB和MyISAM的区别。

**MySQL中都有哪些触发器？**

在MySQL数据库中有如下六种触发器：

* Before Insert
* After Insert
* Before Update
* After Update
* Before Delete
* After Delete

**常用SQL语句**

**SQL语句主要分为哪几类**

数据定义语言DDL（Data Ddefinition Language）CREATE，DROP，ALTER

主要为以上操作 即对逻辑结构等有操作的，其中包括表结构，视图和索引。

数据查询语言DQL（Data Query Language）SELECT

这个较为好理解 即查询操作，以select关键字。各种简单查询，连接查询等 都属于DQL。

数据操纵语言DML（Data Manipulation Language）INSERT，UPDATE，DELETE

主要为以上操作 即对数据进行操作的，对应上面所说的查询操作 DQL与DML共同构建了多数初级程序员常用的增删改查操作。而查询是较为特殊的一种 被划分到DQL中。

数据控制功能DCL（Data Control Language）GRANT，REVOKE，COMMIT，ROLLBACK

主要为以上操作 即对数据库安全性完整性等有操作的，可以简单的理解为权限控制等。

**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。

**SQL 约束有哪几种？**

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。

**六种关联查询**

* 交叉连接（CROSS JOIN）
* 内连接（INNER JOIN）
* 外连接（LEFT JOIN/RIGHT JOIN）
* 联合查询（UNION与UNION ALL）
* 全连接（FULL JOIN）
* 交叉连接（CROSS JOIN）

**mysql中 in 和 exists 区别**

mysql中的in语句是把外表和内表作hash 连接，而exists语句是对外表作loop循环，每次loop循环再对内表进行查询。一直大家都认为exists比in语句的效率要高，这种说法其实是不准确的。这个是要区分环境的。

1. 如果查询的两个表大小相当，那么用in和exists差别不大。
2. 如果两个表中一个较小，一个是大表，则子查询表大的用exists，子查询表小的用in。
3. not in 和not exists：如果查询语句使用了not in，那么内外表都进行全表扫描，没有用到索引；而not extsts的子查询依然能用到表上的索引。所以无论那个表大，用not exists都比not in要快。

**varchar与char的区别**

**char的特点**

* char表示定长字符串，长度是固定的；
* 如果插入数据的长度小于char的固定长度时，则用空格填充；
* 因为长度固定，所以存取速度要比varchar快很多，甚至能快50%，但正因为其长度固定，所以会占据多余的空间，是空间换时间的做法；
* 对于char来说，最多能存放的字符个数为255，和编码无关

**varchar的特点**

* varchar表示可变长字符串，长度是可变的；
* 插入的数据是多长，就按照多长来存储；
* varchar在存取方面与char相反，它存取慢，因为长度不固定，但正因如此，不占据多余的空间，是时间换空间的做法；
* 对于varchar来说，最多能存放的字符个数为65532

总之，结合性能角度（char更快）和节省磁盘空间角度（varchar更小），具体情况还需具体来设计数据库才是妥当的做法。

**varchar(50)中50的涵义**

最多存放50个字符，varchar(50)和(200)存储hello所占空间一样，但后者在排序时会消耗更多内存，因为order by col采用fixed\_length计算col长度(memory引擎也一样)。在早期 MySQL 版本中， 50 代表字节数，现在代表字符数。

**int(20)中20的涵义**

是指显示字符的长度。20表示最大显示宽度为20，但仍占4字节存储，存储范围不变；

不影响内部存储，只是影响带 zerofill 定义的 int 时，前面补多少个 0，易于报表展示

**mysql为什么这么设计**

对大多数应用没有意义，只是规定一些工具用来显示字符的个数；int(1)和int(20)存储和计算均一样；

**mysql中int(10)和char(10)以及varchar(10)的区别**

* int(10)的10表示显示的数据的长度，不是存储数据的大小；chart(10)和varchar(10)的10表示存储数据的大小，即表示存储多少个字符。

int(10) 10位的数据长度 9999999999，占32个字节，int型4位  
char(10) 10位固定字符串，不足补空格 最多10个字符  
varchar(10) 10位可变字符串，不足补空格 最多10个字符

* char(10)表示存储定长的10个字符，不足10个就用空格补齐，占用更多的存储空间
* varchar(10)表示存储10个变长的字符，存储多少个就是多少个，空格也按一个字符存储，这一点是和char(10)的空格不同的，char(10)的空格表示占位不算一个字符

**FLOAT和DOUBLE的区别是什么？**

* FLOAT类型数据可以存储至多8位十进制数，并在内存中占4字节。
* DOUBLE类型数据可以存储至多18位十进制数，并在内存中占8字节。



**SQL优化**

**如何定位及优化SQL语句的性能问题？创建的索引有没有被使用到?或者说怎么才可以知道这条语句运行很慢的原因？**

对于低性能的SQL语句的定位，最重要也是最有效的方法就是使用执行计划，MySQL提供了explain命令来查看语句的执行计划。 我们知道，不管是哪种数据库，或者是哪种数据库引擎，在对一条SQL语句进行执行的过程中都会做很多相关的优化，**对于查询语句，最重要的优化方式就是使用索引**。 而**执行计划，就是显示数据库引擎对于SQL语句的执行的详细情况，其中包含了是否使用索引，使用什么索引，使用的索引的相关信息等**。

delete：删除表的内容，表的结构还存在，不释放空间，可以回滚恢复；

drop：删除表内容和结构，释放空间，没有备份表之前要慎用；

truncate：删除表的内容，表的结构存在，可以释放空间,没有备份表之前要慎用；

drop：drop test 删除表test，并释放空间，将test删除的一干二净

truncate：truncate test 删除表test里的内容，并释放空间，但不删除表的定义，表的结构还在

delete：（1）删除制定数据：删除表test中年龄等于30且国家为US的数据：delete from test where age=30 and country =‘US’；

（2）删除整个表：仅删除表test内的所有内容，保留表的定义，不释放空间：delete from test 或delete \*from test

drop > truncate > delete

5. 设有如下关系表R：（6分） R(NO,NAME,SEX,AGE,CLASS) 主关键字是NO 其中NO为学号(数值型)，NAME为姓名，SEX为性别，AGE为年龄，CLASS为班号。 写出实现下列功能的SQL语句。

1) 插入"95031"班学号为30，姓名为"郑和"的学生记录

INSERT INTO R VALUES (30,'郑和',NULL,NULL,95031);

**INSERT INTO R(NO,NAME,SEX,AGE,CLASS) VALUE(30,'郑和',NULL,NULL,95031);**

2) 删除学号为20的学生记录 (6.0分)

Delete from R where NO=20

3) 删除学号为20的学生姓名记录

UPDATE R SET SEX=NULL WHERE NO=20;

like 操作符

查找表中是否包含某行，like 后面接通配符，对相匹配的进行查找，这里是指找到所有，%表示匹配多个字符，\_表示匹配一个字符。

> select name from student where scor like 100; //找到考100分的

> select name from student where sex like '%a%'; //找到性别中有a的

> select name from student where sex like '\_a%'; //找到性别中有a的

**9、mysql函数**  
mysql 支持利用函数处理数据，函数在数据上执行，给数据的转换和处理提供了方便。支持的函数类型有：

1. 用于处理字符串的文本处理函数；
2. 用于在数值数据上进行算术操作的数值函数；
3. 用于处理日期的和时间的函数；
4. 返回数据库信息的系统函数；



**11、分组数据** （**group by**）  
将数据分成多个组，以便对数据进行聚集计算。**group by**创建分组，但是不保证分组内部数据的排序，使用 **order by**对内部数据排序。（group by 是按照列来分组，必须出现在where之后，order by 之前。查询字段中出现聚合函数和普通列，一起查询的时候，那么分组的条件就是普通列，**使用普通列进行分组查询要聚合的函数**）  
使用形式：

> select [聚合函数] 字段名 from 表名

> [where 分组前行过滤]

> [group by 字段名]

> [having 分组后行过滤]

**选择一列进行分组聚合，选择多列进行分组聚合。**

>select avg(score),subject from student group by subject; //对学科进行分组，计算平均分。（一列）

>select avg(score),max(score),min(score),subject，Sex from student

-> where score>=80

-> group by subject,Sex

-> having subject in ('math','eng')

>

**2、子查询**  
子查询实际上嵌套在其他查询中的查询，例如两张表，物品表 orderitems 中，保存订单号，客户ID，订单日期。customers表中保存客户信息。(两张表互不相连，需要查询两次)  
如果要找到物品 TNT2 的所有客户，需要采用下列步骤：

1. 查找包含物品 TNT2 的所有订单的编号；
2. 查找前一步骤列出的订单编号的所有客户的 ID；
3. 查找前一步骤返回的所有客户 ID 的客户信息；

所以 可以执行下列操作：

>select order\_num from orderitems where prod\_id='TNT2'; //查到为4，6

>select cust\_id from orders where order\_num in (4，6)；

**13、表联结**  
关系型数据库的强大在于关系表，关系表的设计原则时保证把信息分解为多个表，一类数据一个表，各个表通过某些常用的值互相关联，比如两个表之间通过**外键**来关联。（外键是某个表中的一列，它又是另外一个表的主键值。）  
表联结的优点：

1. 数据信息不重复，从而不浪费时间和空间，伸缩性好（scale well）
2. 如果某个数据信息变动，可以只更新该表中的某个记录，相关表数据不用变更

**1、创建联结**：select A\_name,B\_name from A,B where A.A\_id = B.B\_id order by A\_name, B\_name;  
（where子句指示MySQL匹配 A 表中的A\_id和 B表中的 B\_id 相匹配，要匹配的两列以A.A\_id 和 B.B\_id 完全限定表名（当引用的列可能存在二义性，必须使用完全限定表名），两个表的联结需要用到where语句，进行列过滤）

**2、笛卡尔积**：由没有联结条件的表关系返回的结果称为笛卡尔积（检索出的行的数目将是第一个表中的行数乘以第二个表中的行数）  
select A\_name,B\_name from A, B order by A\_name, B\_name;

**3、内部联结**：内部联结也叫等值联结（equijoin），它基于两个表之间的相等测试，返回两个表的交集部分  
例如：select A\_name,B\_name from A inner join B on A\_name = B\_name;  
(这条SQL语句中，select语句与之前的相同，相较于上面的from，这里使用了inter join指定（当使用inter join时，联结条件用on子句而不是where子句，在这里on等同于where）

**4、联结多个表**：SQL对一条select语句中可以联结的表的数目没有限制，创建规则也基本相同（首先列出所有表，然后定义表之间的关系）  
（PS：MySQL在运行时关联指定的每个表以处理联结，这样很耗费资源，因此使用中应注意，不要联结不必要的表。）  
select A\_name,B\_name C\_name from A,B where A\_name=B\_name and A\_name=B\_name;

**5、使用表别名**：别名除了用于列名和计算字段外，也可以给表起别名，可以缩短SQL语句，允许在单条select语句中多次使用相同的表。

**6、自联结**：自联结为在**同一个表**中做联结操作。（user\_id 是 TN1，查询其user\_name，最后找到user\_name下其他的user\_score的user\_id，user\_name ，最后找到的都是相同user\_name下其他的user\_score）

//使用子查询

>select user\_id,user\_name from usertable

> where user\_score=(select user\_score from usertable where user\_id='TN1'

//使用自联结

>select p1.user\_id, p1.user\_name

> from usertable as p1, usertable as p2

> where p1.user\_score = p2.user\_score

> and p2.user\_id = 'TN1';

**7、外联结**：外联结分为左联结和右联结（left join on / left outer join on）  
**左联结**：select A.Name from A left join B on A.id = B.id  
(左(外)连接，左表的记录将会全部表示出来，而右表(b\_table)只会显示符合搜索条件的记录。右表记录不足的地方均为NULL。)

**右联结**：select A.Name from A right join B on A.id = B.id  
(与左(外)连接相反，右(外)连接，左表只会显示符合搜索条件的记录，而右表的记录将会全部表示出来。左表记录不足的地方均为NULL。)

**8、创建组合查询**  
例如把两个查询结合在一起，使用**union**（union联结查询的列必须相同）

>select name from student where id=1

> union select select name from student where id=0;

**15、更新和删除数据**  
用update更新表中特定的行，或者更新所有的行，后面不跟where语句。  
例如：修改小王的成绩（更改一列一行） update student set score = 100 where name='xiaowang';  
例如：修改小王的成绩，以及id （更改两列一个行） update student set score = 100,id=5 where name='xiaowang';  
例如：更改一列下面所有行 update student set score=100

用delete删除数据  
delete from student where id=0; //删掉id=0这一行  
drop table student;//删掉这个表

**SQL的生命周期？**

1. 应用服务器与数据库服务器建立一个连接
2. 数据库进程拿到请求sql
3. 解析并生成执行计划，执行
4. 读取数据到内存并进行逻辑处理
5. 通过步骤一的连接，发送结果到客户端
6. 关掉连接，释放资源

**大表数据查询，怎么优化**

1. 优化shema、sql语句+索引；
2. 第二加缓存，memcached, redis；
3. 主从复制，读写分离；
4. 垂直拆分，根据你模块的耦合度，将一个大的系统分为多个小的系统，也就是分布式系统；
5. 水平切分，针对数据量大的表，这一步最麻烦，最能考验技术水平，要选择一个合理的sharding key, 为了有好的查询效率，表结构也要改动，做一定的冗余，应用也要改，sql中尽量带sharding key，将数据定位到限定的表上去查，而不是扫描全部的表；

**数据库优化**

**为什么要优化**

* 系统的吞吐量瓶颈往往出现在数据库的访问速度上
* 随着应用程序的运行，数据库的中的数据会越来越多，处理时间会相应变慢
* 数据是存放在磁盘上的，读写速度无法和内存相比

优化原则：减少系统瓶颈，减少资源占用，增加系统的反应速度。

**数据库结构优化**

一个好的数据库设计方案对于数据库的性能往往会起到事半功倍的效果。

需要考虑数据冗余、查询和更新的速度、字段的数据类型是否合理等多方面的内容。

**将字段很多的表分解成多个表**

对于字段较多的表，如果有些字段的使用频率很低，可以将这些字段分离出来形成新表。

因为当一个表的数据量很大时，会由于使用频率低的字段的存在而变慢。

**增加中间表**

对于需要经常联合查询的表，可以建立中间表以提高查询效率。

通过建立中间表，将需要通过联合查询的数据插入到中间表中，然后将原来的联合查询改为对中间表的查询。

**增加冗余字段**

设计数据表时应尽量遵循范式理论的规约，尽可能的减少冗余字段，让数据库设计看起来精致、优雅。但是，合理的加入冗余字段可以提高查询速度。

表的规范化程度越高，表和表之间的关系越多，需要连接查询的情况也就越多，性能也就越差。