1. **基础**
2. **SQL概念**

SQL（Structured Query Language 结构查询语言）是一个功能强大的数据库语言。SQL是关系数据库管理系统的标准语言。

DML（数据操作语言）: 用于检索或修改数据

DDL（数据定义语言）:用于定义数据的结构，创建、修改、删除数据库对象

DCL（数据控制语言）: 用于定义数据库用户的权限

1. **常用函数**

**字符函数**

UPPER: 将输入的字符串变为大写返回

Select upper(‘hello’) from dual; //dual为数据库提供的一张虚拟表

LOWER: 将输入的字符串变为小写返回

Select lower(ename) from emp;

INITCAP: 开头首字母大写

Select initcap(ename) from emp;

LENGTH: 求出字符串的长度

Select ename, length(ename) from emp;

查询出姓名长度为5的雇员信息

Select ename,length(ename) from emp where length(ename)=5;

REPLACE: 字符串进行替换

使用”\_”替换姓名中所有字母”A”

Select replace(ename,’A’,’\_’) from emp;

SUBSTR: 字符串截取

Select ename,substr(ename,0,3) from emp ;

截取每个雇员姓名的后三个字母

Select ename,substr(ename,length(ename)-2) from emp;

Select ename,substr(ename,-3) from emp;

**数字函数**

ROUND: 四舍五入的操作

TRUNC: 截取指定位置的内容

MOD: 取模或取余

**聚合函数**

avg()：返回的是指定组中的平均值，空值被忽略。

count()：返回的是指定组中的项目个数。

max()：返回指定数据中的最大值。

min()：返回指定数据中的最小值。

sum()：返回指定数据的和，只能用于数字列，空值忽略。

group by()：对数据进行分组，对执行完 group by 之后的组进行聚合函数的运算，计算每一组的值。最后用having 去掉不符合条件的组，having 子句中的每一个元素必须出现在select 列表中（只针对于mysql）。

**通用函数**

NVL() 处理null

查询出每个雇员的全部年薪

Select ename,sal,comm,(sal+comm)\*12 from emp; //此时comm字段有null值

//此时要将null变为0

Select ename,sal,comm,(sal+nvl(comm,0))\*12,nvl(comm,0) from emp;

DECODE() 多数值判断

类似于if…else语，不同的是decode()函数判断的是数值，而不是逻辑条件。

显示全部雇员的职位，但是这些职位要求替换为中文显示

Select empno,ename,job,decode(job,’CLERK’,’办事员’,’SALESMAN’,’销售人员’,’MANAGER’,’经理’ ,’ANALYST’,’分析员’ ,’PRESIDENT’,’总裁’) from emp

1. **常用的数据字段**

每一张数据表实际是由若干字段组成，而每一个字段会有其对应的数据类型。

常用数据类型：

　（1）数值类型：

　　　　 1.tinyint：用于代替boolean，取0和1，1代表true，0代表false；

　　　　 2.int；相当于java中的int；

　　　　 3.bigint；相当于java中的lang；

　　　4.decimal：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;

　（2）日期和时间类型：

　　　　1.date：日期类型；格式：yyy-MM-dd；

　　　　2.time：时间类型；格式：hh:mm:ss；

　　　　3.datetime：日期时间类型；格式：yyy-MM-dd hh:mm:ss；

　（3）字符串类型：

　　　1.char(n)：固定长度字符串类型，n是字符串长度必须设置，默认255，性能好；

//char(10) ‘abc’；

　　　　2.varchar(n)：可变长度字符串类型，长度不到自动缩进，性能差；//varchar(10) ‘abc’；

　　　　3.text：字符串类型；

　　　　注释：所有字符串类型和日期类型都要用单引号括起来；

1. **集合操作**

在数学的操作中存在交、差、并、补的概念，而在数据的查询中也存在此概念。

UNION 连接两个，相同部分不显示。

UNION ALL连接两个，相同部分显示。

INTERSECT 返回两个查询中相同部分

MINUS 返回两个查询中不同部分

union和union all都是合并结果集

区别是：

1. union去除两个结果集的重复记录，union all不去除重复记录，是两个结果集的加和

2. union效率低，union all效率高

1. **视图和游标**

视图时一种虚拟表，具有与物理表相同的功能，可对视图增删改查。视图通常是一个或多个表的行或列的子集。对视图的修改不影响基本表。使获取数据更容易，相比多表查询。

游标: 对查询出来的结果集做一个单元来有效处理。游标可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。可对结果集当前行做修改。

一般不使用游标,但当需逐条处理数据时,游标显得很重要。

1. **数据库视图和表的区别，什么是视图？怎么使用视图？物化视图？**

视图是种虚表，不是真正的物理表，只是为业务查询方便 ，将几张物理表虚拟组成一个视图。

第一点：

使用视图，可以定制用户数据，聚焦特定的数据。

解释：

在实际过程中，公司有不同角色的工作人员，我们以销售公司为例的话，

采购人员，可以需要一些与其有关的数据而与他无关的数据，对他没

有任何意义，我们可以根据这一实际情况，专门为采购人员创建一个视

图，以后他在查询数据时，只需select \* from view\_caigou 就可以啦。

第二点：使用视图，可以简化数据操作。

解释：我们在使用查询时，在很多时候我们要使用聚合函数，同时还要

显示其它字段的信息，可能还会需要关联到其它表，这时写的语句可能

会很长，如果这个动作频繁发生的话，我们可以创建视图，这以后，我

们只需要select \* from view1就可以

第三点：使用视图，基表中的数据就有了一定的安全性

因为视图是虚拟的，物理上是不存在的，只是存储了数据的集合，我们可以

将基表中重要的字段信息，可以不通过视图给用户，视图是动态的数据的集

合，数据是随着基表的更新而更新。同时，用户对视图，不可以随意的更改

和删除，可以保证数据的安全性。

第四点：可以合并分离的数据，创建分区视图

随着社会的发展，公司的业务量的不断的扩大，一个大公司，下属都设有很

多的分公司，为了管理方便，我们需要统一表的结构，定期查看各公司业务

情况，而分别看各个公司的数据很不方便，没有很好的可比性，如果将这些

数据合并为一个表格里，就方便多啦，这时我们就可以使用union关键字，

将各分公司的数据合并为一个视图。

1. **如何防止sql注入?**

通过在 Web 表单中输入（恶意）SQL 语句得到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行 SQL 语句。

举例：当执行的 sql 为 select \* from user where username = “admin” or “a”=“a”时，sql 语句恒成立，参数 admin 毫无意义。

防止 sql 注入的方式：

1.预编译语句：如，select \* from user where username = ？，sql 语句语义不会发生改变，sql 语句中变量用？表示，即使传递参数时为“admin or ‘a’= ‘a’”，也会把这整体当做一个字符创去查询。

2.Mybatis 框架中的 mapper 方式中的 # 也能很大程度的防止 sql 注入（$无法防止 sql 注入）。

1. **MySQL 中 Float、Decimal 类型存储金额的区别？**

Float 是非标准数据类型，存储金额时存储的是近似值，存在精度问题；Decimal 是标准数据类型，存储金额时存储的是精确值（以字符串的形式保存数值）。

1. **MySQL的日志模块binlog和redo log有了解吗？**

在MySQL的使用中，更新操作也是很频繁的，如果每一次更新操作都根据条件找到对应的记录，然后将记录更新，再写回磁盘，那么IO成本以及查找记录的成本都很高。所以，出现了日志模块，即我们的update更新操作是先写日志，在合适的时间才会去写磁盘，日志更新完毕就将执行结果返回给了客户端。

MySQL中的日志模块主要有redo log（重做日志）和binlog（归档日志）。

**redo log：**

redo log是InnoDB引擎特有的日志模块，redo log是物理日志，记录了某个数据页上做了哪些修改。InnoDB的redo log是固定大小的，比如可以配置为一组4个文件，每个文件的大小是1GB，那么redo log总共就可以记录 4GB的操作。从头开始写，写到末尾就又回到开头循环写。

InnoDB的redo log保证了数据库发生异常重启之后，之前提交的记录不会丢失，这个能力称为crash-safe。

**binlog：**

binlog是Server层自带的日志模块，binlog是逻辑日志，记录本次修改的原始逻辑，说白了就是SQL语句。binlog是追加写的形式，可以写多个文件，不会覆盖之前的日志。通过mysqlbinlog可以解析查看binlog日志。binlog日志文件的格式：statement，row，mixed。

statement格式的binlog记录的是完整的SQL语句，优点是日志文件小，性能较好，缺点也很明显，那就是准确性差，遇到SQL语句中有now()等函数会导致不准确

row格式的binlog中记录的是数据行的实际数据的变更，优点就是数据记录准确，缺点就是日志文件较大。

mixed格式的binlog是前面两者的混合模式

1. **索引**
2. **索引定义**

数据库[索引](http://baike.baidu.com/view/262241.htm)好比是一本书前面的目录，能加快数据库的查询速度。[索引](http://baike.baidu.com/view/262241.htm)是对数据库表中一个或多个列（例如，employee 表的姓氏 (lname) 列）的值进行排序的结构。如果想按特定职员的姓来查找他或她，则与在表中搜索所有的行相比，索引有助于更快地获取信息。

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息，就像一本书的目录一样，可以加快查询速度。InnoDB 存储引擎的索引模型底层实现数据结构为B+树，所有数据都是存储在 B+ 树中的。

1. **建立索引的优缺点**

**优点：** 1.大大加快数据的检索速度;   
 2.创建唯一性索引，保证数据库表中每一行数据的唯一性;   
 3.加速表和表之间的连接;   
 4.在使用分组和排序子句进行数据检索时，可以显著减少查询中分组和排序的时间。

**缺点：**  
　　1.索引需要占用数据表以外的物理存储空间  
　　2.创建索引和维护索引要花费一定的时间  
　　3.当对表进行更新操作时，索引需要被重建，这样降低了数据的维护速度。

1. **建立索引的原则有哪些？**

WHERE / GROUP BY / ORDER BY / ON 的列

离散度大（不同的数据多）的列使用索引才有查询效率提升

索引字段越小越好，因为数据库按页存储的，如果每次查询IO读取的页越少查询效率越高

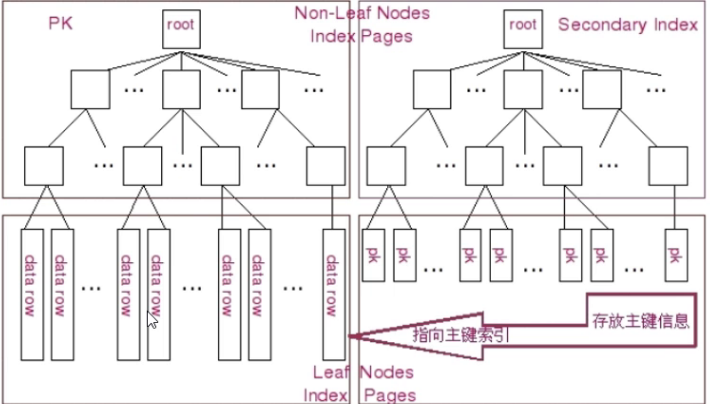
1. **索引类型**

根据数据库的功能，可以在数据库设计器中创建索引：**唯一索引、主键索引和聚集索引**。 尽管唯一索引有助于定位信息，但为获得最佳性能结果，建议改用主键或唯一约束。   
**唯一索引**： UNIQUE

例如：create unique index stusno on student（sno）；  
表明此索引的每一个索引值只对应唯一的数据记录，对于单列惟一性索引，这保证单列不包含重复的值。对于多列惟一性索引，保证多个值的组合不重复。  
**主键索引**： primary key  
数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。 在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。   
[**聚集索引**](http://www.cnblogs.com/aspnethot/articles/1504082.html)：cluster   
（1）聚集索引:将数据存储与索引放到了一块，找到索引也就找到了数据

（2）非聚集索引: 将数据存储于索引分开结构，索引结构的叶子节点指向了数据的对应行， B+树就是非聚集索引。

在 InnoDB 中它们还有一个非常重要的区别：聚集索引的叶子节点的的 data 域包含了完整的数据记录，而非聚集索引的叶子节点的 data 域记录着主键的值，因此在使用非聚集索引进行查找时，需要先查找到主键值，然后再到聚集索引中进行查找，这称之为回表查询。

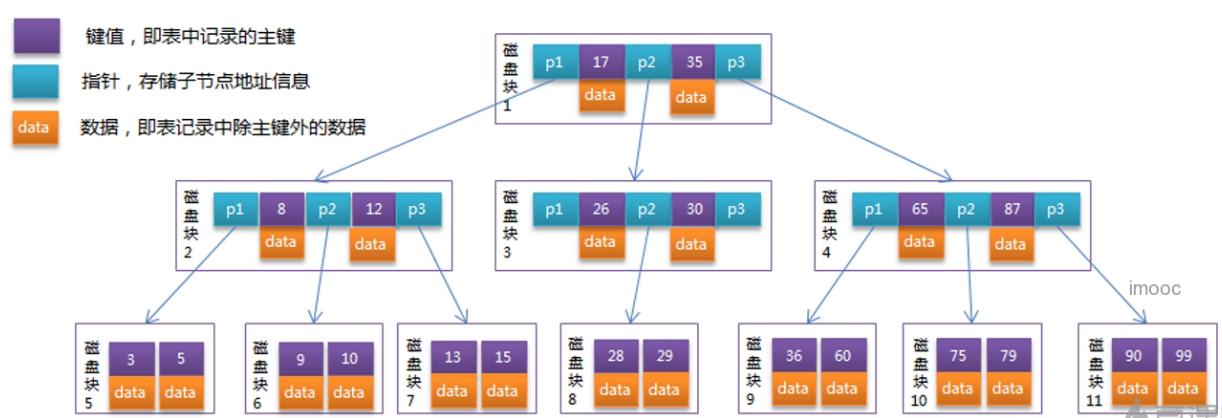


1. **为什么不适用 B 树？**

下面两张图分别是 B 树和 B + 树的示意图，可以看出有一个很大的区别：

B + 树所有Data 域在叶子节点，其余节点用来索引，而 B 树是每个索引节点都会有 Data 域；并且 B + 树所有叶子节点之间都有一个链指针。 这样遍历叶子节点就能获得全部数据，从而支持区分查询。在数据库中基于范围的查询是非常频繁的，而B树不支持这样的遍历操作。

**B树**



左边的key值少于中间的，右边的key值大于中间的

**定义：**

根节点至少包括两个孩子

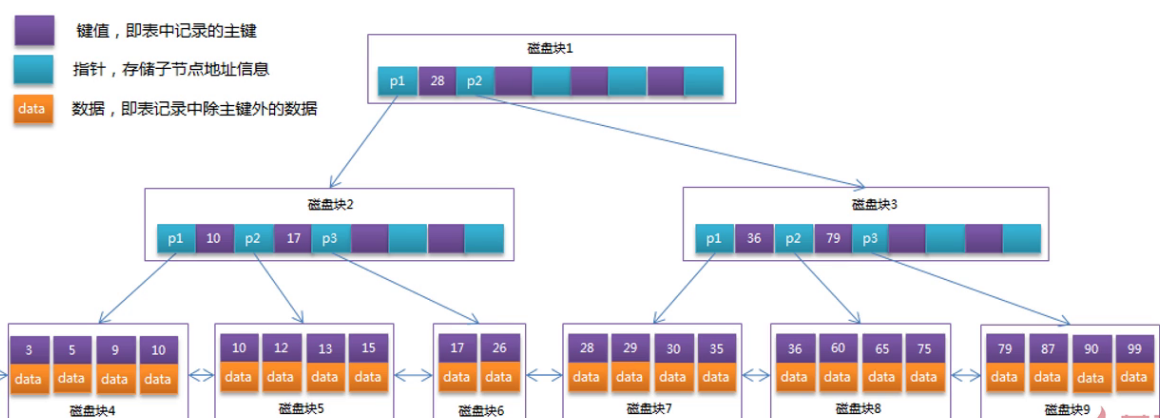
树中每个节点最多含有m个孩子（m>=2）

所有叶子结点在同一层

如果要找key=29,从根节点开始找，29在17和35之间，找到磁盘块1读入内存，磁盘块1中的p2指针，根据p2指针找到磁盘块3，将磁盘块3读入内存，磁盘块3的p2指针,根据P2指针再找到磁盘块8,将磁盘块8读入内存，根据关键字key=29找到数据data。经历3次IO,3次内存查找。

当数据越来越多，树的高度就会越来越高，在查找数据时查找的深度会越深。导致磁盘读取的次数增加。

[**B+树**](http://blog.csdn.net/kennyrose/article/details/7532032)



非叶子结点没有数据，只有键值，数据全在叶子结点上。增加每个结点存储key的数量。降低了树的高度。

页子结点都有双向指针

在Mysql中，innoDB数据引擎采用B+树并将磁盘块1读入内存，常驻内存。

所以在查找key=29时,只需要读取磁盘块3和磁盘块7即可。（最多1到3次就可以查找到）

1. **为什么不用哈希表？**

（1）哈希表只适用与查找等值查询， 不能支持区分条件（大于小于查询）、模糊查询等；

（2）hash 索引虽然在等值查询上较快，但是不稳定，性能不可预测，当某个键值存在大量重复的时候，发生 hash 碰撞，此时效率可能极差。而 B + 树的查询效率比较稳定。

1. **为什么不用红黑树？**

红黑树往往高度过大，从而造成磁盘 IO 读写过于频繁，效率低下。而且逻辑上很近的节点（父子）物理上可能很远，无法利用局部性原理。

红黑树最多只有两个子节点，所以高度会非常高，导致遍历查询的次数会多，又因为红黑树在数组中存储的方式，导致逻辑上很近的父节点与子节点可能在物理上很远，导致无法使用磁盘预读的局部性原理，需要很多次IO才能找到磁盘上的数据

但B+树一个节点中可以存储很多个索引的key，且将大小设置为一个页，一次磁盘IO就能读取很多个key，且叶子节点之间还加上了下个叶子节点的指针，遍历索引也会很快。

1. **索引调优**

大致思路：

根据慢日志定位慢查询sql->使用explain工具分析sql->修改sql或让sql尽量走索引

使用explain关键字查看sql的执行计划,可以看到该sql的索引命中情况、索引命中类型、命中率等相关的信息,通过这种方式可以对sql语句进行调优,对关键查询的条件创建索引以及注意一下sql语句的用法

尽量不使用like,尽量避免在某一列上进行运算,

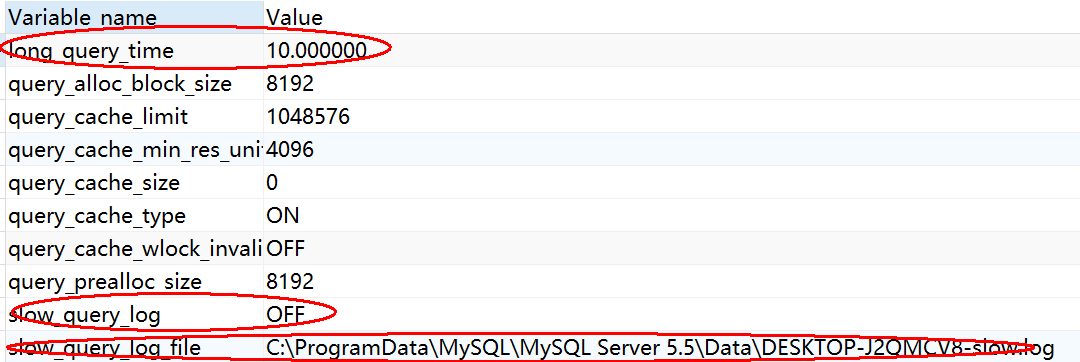
不使用 in not in 使用exist not exist等等代替

**开启慢日志**

show VARIABLES like '%query%'

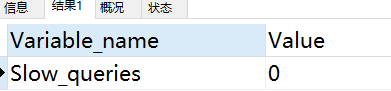
将查询时间long\_query\_time设置为1秒（查询超过1秒就认为是慢查询）

将show\_query\_log设置为true



查询慢sql的数量

show status like '%slow\_queries%'



开启慢查询

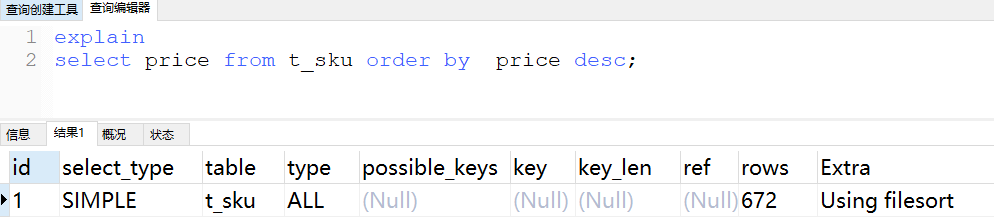
set global slow\_query\_log=on

设置慢查询时间

set global long\_query\_time=1

以上设置在重启数据库后会还原，可以直接修改数据库的配置文件my.ini

**Explain的使用**



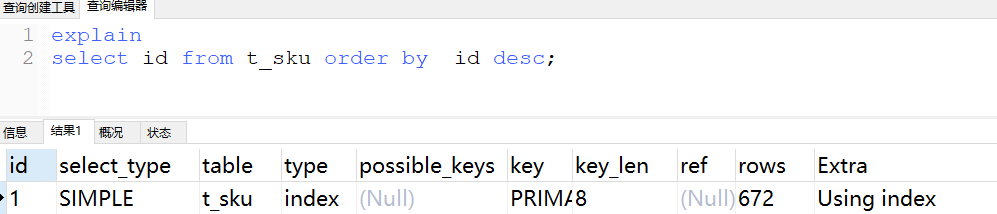
主要关注type列，key列，extra列

type列依次性能从好到差：system，const，eq\_ref，ref，fulltext，ref\_or\_null，unique\_subquery，index\_subquery，range，index\_merge，index，ALL，除了all之外，其他的type都可以使用到索引，除了index\_merge之外，其他的type只可以用到一个索引。一般来说，好的sql查询至少达到range级别，最好能达到ref。

key列正常情况要是非NULL，如果是NULL，表示没有用索引，性能低下。

extra列主要关注是不是用了临时表排序，索引等等。例如：using temporary，using filesort等都是需要优化的。

使用id查询，id有索引



1. **索引失效**

1.如果条件中有or，即使其中有条件带索引也不会使用。（注意：要想使用or，又想让索引生 效，只能将or条件中的每个列都加上索引）

2.like查询是以%开头

3.如果列类型是字符串，那一定要在条件中将数据使用引号引用起来，否则不使用索引；

4. 联合索引最左匹配原则

定义：在检索数据时从联合索引的最左边开始匹配。一直向右匹配直到遇到范围查询（>/</between/like）就停止后面的匹配。比如查询a = 3, b = 4 and c > 5 and d = 6，如果建立(a,b,c,d)索引，d是用不到索引的，如果建立(a,b,d,c)索引，则都可以用到，此外a,b,d的顺序可以任意调整

1. **事务**
2. MyISAM 和 InnoDB 是最常见的两种存储引擎

InnoDB支持事务

InnoDB支持外键

InnoDB有行级锁，MyISAM是表级锁

MyISAM相对简单所以在效率上要优于InnoDB。如果系统插入和查询操作多，不需要事务外键。选择MyISAM

如果需要频繁的更新、删除操作，或者需要事务、外键、行级锁的时候。选择InnoDB。

1. **事务**

[数据库事务](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E4%BA%8B%E5%8A%A1)(Database Transaction) ，是指作为单个逻辑工作单元执行的一系列[操作](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C/33052" \t "_blank)，要么完全地执行，要么完全地不执行。一个逻辑工作单元要成为事务，必须满足所谓的ACID（原子性、一致性、隔离性和持久性）属性。

1. 事务的四大特征是什么ACID

A 事务的原子性(Atomicity)：指一个事务要么全部执行,要么不执行.也就是说一个事务不可能只执行了一半就停止了.比如你从取款机取钱,这个事务可以分成两个步骤:1划卡,2出钱.不可能划了卡,而钱却没出来.这两步必须同时完成.要么就不完成.

C 事务的一致性(Consistency)：指事务的运行并不改变数据库中数据的一致性.例如,完整性约束了a+b=10,一个事务改变了a,那么b也应该随之改变.

I 独立性(Isolation）:事务的独立性也有称作隔离性,是指两个以上的事务不会出现交错执行的状态.因为这样可能会导致数据不一致.

D 持久性(Durability）:事务的持久性是指事务执行成功以后,该事务所对数据库所作的更改便是持久的保存在数据库之中，不会无缘无故的回滚.

1. **Mysql 中四种隔离级别分别是什么**



读未提交（READ UNCOMMITTED）：未提交读隔离级别也叫读脏，就是事务可以读取其它事务未提交的数据。

读已提交（READ COMMITTED）：在其它数据库系统比如 SQL Server 默认的隔离级别就是提交读，已提交读隔离级别就是在事务未提交之前所做的修改其它事务是不可见的。

可重复读（REPEATABLE READ）：保证同一个事务中的多次相同的查询的结果是一致的，比如一个事务一开始查询了一条记录然后过了几秒钟又执行了相同的查询，保证两次查询的结果是相同的，可重复读也是 mysql 的默认隔离级别。

可串行化（SERIALIZABLE）：可串行化就是保证读取的范围内没有新的数据插入，比如事务第一次查询得到某个范围的数据，第二次查询也同样得到了相同范围的数据，中间没有新的数据插入到该范围中。

1. **什么叫脏数据，什么叫脏读**

脏数据在临时更新（脏读）中产生。事务 A 更新了某个数据项 X，但是由于某种原因，事务 A 出现了问题，于是要把 A 回滚。但是在回滚之前，另一个事务 B 读取了数据项 X 的值(A 更新后)，A 回滚了事务，数据项恢复了原值。事务 B 读取的就是数据项 X 的就是一个“临时”的值，就是脏数据。

脏读就是指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是脏数据，依据脏数据所做的操作可能是不正确的。

**读未提交**：

它充许另外一个事务可以看到这个事务未提交的数据，

这种隔离级别会产生脏读，不可重复读和幻像读。

**读提交**：

保证一个事务修改的数据提交后才能被另外一个事务读取，

也是大多数数据库的默认值。可以避免脏读，但会产生不可重复读和幻像读。

**重复读**：

在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。

**串行化**：

顺序执行事务。除了防止脏读，不可重复读外，还避免了幻像读。

并发性也最低，但最安全。

不可重复读的重点是修改 :

同样的条件 , 你读取过的数据 , 再次读取出来发现值不一样了 。

幻读的重点在于新增或者删除 ：

同样的条件 , 第 1 次和第 2 次读出来的记录数不一样。

1. **事务并发访问引起的问题以及如何避免**

**丢失更新：**

两个不同事务同时获得相同数据，然后在各自事务中同时修改了该数据，那么先提交的事务更新会被后提交事务的更新给覆盖掉，这种情况先提交的事务所做的更新就被覆盖，导致数据更新丢失。

**脏读：**

事务A读取了事务B未提交的数据，由于事务B回滚，导致了事务A的数据不一致，结果事务A出现了脏读；

**不可重复读：**

一个事务在自己没有更新数据库数据的情况，同一个查询操作执行两次或多次得到的结果数值不同，因为别的事务更新了该数据，并且提交了事务。

**幻读：**

事务A读的时候读出了N条记录，事务B在事务A执行的过程中增加 了1条，事务A再读的时候就变成了N+1条，这种情况就叫做幻读。

注意：幻读是指一种结构上的改变，比如说条数发生了改变；不可重复读是指读出的数值发生改变。

1. **MySQL中的锁机制**

答：MySQL数据库的锁分为表级锁和行级锁。从数据库角度看，行级锁又可以分为独占锁和共享锁。

**独占锁（排他锁），也称X锁（Exclusive Lock）：**

独占锁锁定的资源只允许进行锁定操作的程序使用，其它任何对它的操作均不会被接受。执行数据更新命令，即INSERT、UPDATE 或DELETE 命令时，MySQL会自动使用独占锁。但当对象上有其它锁存在时，无法对其加独占锁。独占锁一直到事务结束才能被释放。

在select命令中使用独占锁的SQL语句为：select … for update;

**共享锁，也叫S锁（Shared Lock）：**

共享锁顾名思义，那就是其锁定的资源可以被其它用户读取，但其它用户不能修改。如果在select查询语句中要手动加入共享锁，那么对应的SQL语句为：select ... lock in share mode

这里需要注意：

一个事务在一行数据上加入了独占锁，那么其余事务不可以在该数据行上加入任何锁。也就是说加过排他锁的数据行在其他事务中是不能修改数据的，也不能通过for update和lock in share mode锁的方式查询数据，但可以直接通过select ...from...查询数据，因为普通查询没有任何锁机制。

**MySQL中的死锁**

争夺资源而造成的一种互相等待的现象,若无外力作用,它们都将无法推进下去. 此时称系统处于死锁状态或系统产生了死锁,这些永远在互相等待的进程称为死锁进程。 表级锁不会产生死锁.所以解决死锁主要还是针对于最常用的InnoDB。

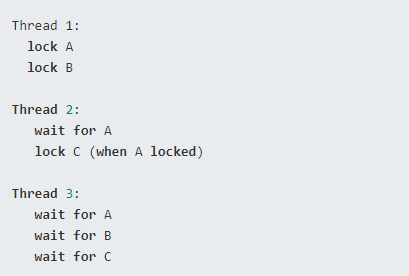
死锁的关键在于：两个(或以上)的Session加锁的顺序不一致。

那么对应的解决死锁问题的关键就是：让不同的session加锁有次序。



**业务逻辑上的死锁解决方案：**

指定锁的获取顺序



如果一个线程（比如线程3）需要一些锁，那么它必须按照确定的顺序获取锁。它只有获得了从顺序上排在前面的锁之后，才能获取后面的锁。

例如，线程2和线程3只有在获取了锁A之后才能尝试获取锁C(获取锁A是获取锁C的必要条件)。因为线程1已经拥有了锁A，所以线程2和3需要一直等到锁A被释放。然后在它们尝试对B或C加锁之前，必须成功地对A加了锁。

不要出现线程2：wait for A 线程3：wait for B，这样加锁顺序就不对，会死锁。

1. **乐观锁和悲观锁**

从程序员的角度看，数据库中的锁又可以分为悲观锁和乐观锁。

悲观锁：利用数据库的锁机制实现，在整个数据处理过程中都加入了锁，以保持排他性。

乐观锁：乐观锁可以利用CAS实现，在操作数据的时候进行一个比较，按照当前事务中的数据和数据库表中的该数据是否一致来决定是否要执行本次操作。

我们来举例说下乐观锁的CAS实现，有如下的数据库表和数据：

create table cas\_test(

phone varchar(32) primary key,

name varchar(32)

)engine=InnoDB;

// 插入数据

insert into cas\_test values("18810101035","zhangsan");

当我们将数据更新后，会比较该行数据与数据库表中的该行数据是否一致，以此来决定是否要更新数据。

1. **存储过程**
2. **应用场景**

存储过程是一个编译后的SQL脚本集合，可以单独调用，但是不能用在sql语句中。

业务场景：银行的核心业务依靠sql语句处理，而衍生业务会交给外包公司来做。对银行来说并不希望对外公开数据库设计和表结构，这样很不安全。银行会让内部开发人员编写存储过程，有的是用来开户，有的是用来转账的。封装好存储过程，银行只需要告诉外包公司这些存储过程的参数和返回值即可。



1. **存储过程优点**

存储过程是编译过的SQL脚本，省去了对SQL语法检查及优化的过程，执行速度快。

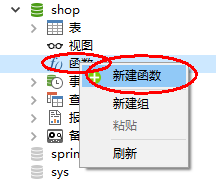
实现SQL编程，将大而复杂的SQL改成小而简单的SQL语句，可以降低锁表时间和范围。

（可以将子查询查询出的内容使用存储过程完成，查询结果定义成一个变量，这样就不会长时间的锁表）

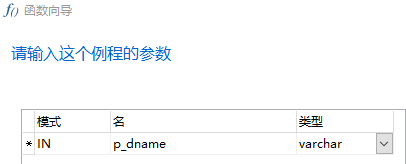
对外封装表结构，提升数据库的安全性。

1. **存储过程案例**

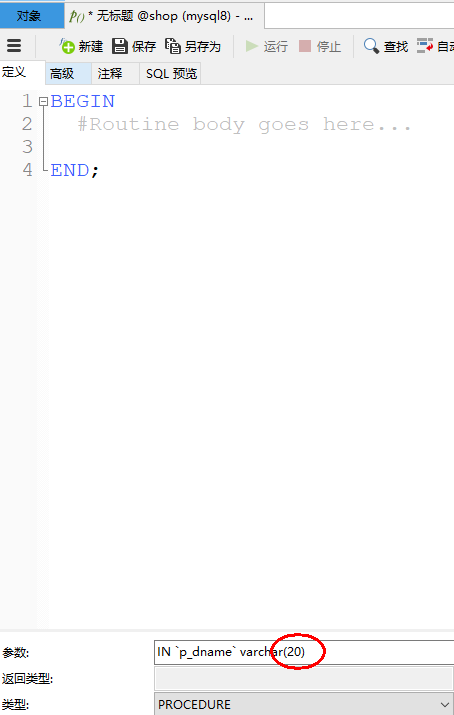
编写存储过程，根据部门名称，查询部门用户信息。



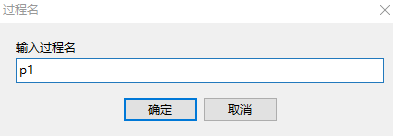




修改p\_dname的长度



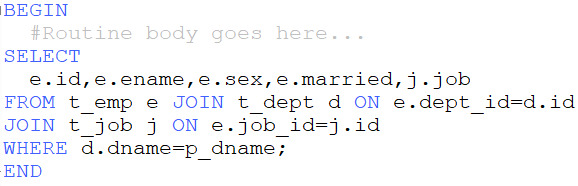
保存存储过程



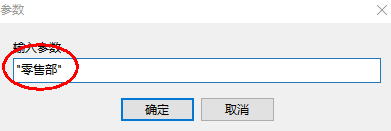
p1 (使用shop数据库中的表)

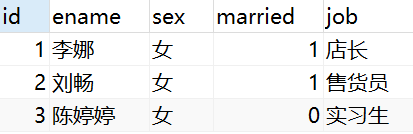
|  |
| --- |
| BEGIN  #Routine body goes here...  SELECT  e.id,e.ename,e.sex,e.married,j.job  FROM t\_emp e JOIN t\_dept d ON e.dept\_id=d.id  JOIN t\_job j ON e.job\_id=j.id  WHERE d.dname=p\_dname;  END |

测试，点运行



输入参数

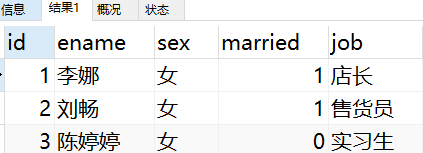




1. **调用存储过程**

新建查询面板

CALL p1("零售部");



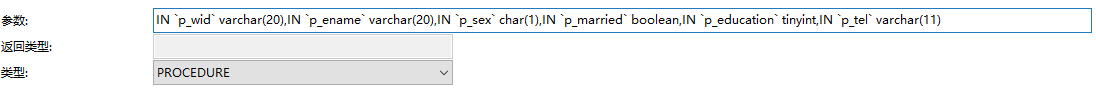
1. **案例2**

编写存储过程：插入实习员工数据时，如果是男性则分配到网商部实习，如果是女性，则分配到零售部实习

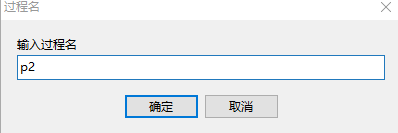




为参数设定长度



保存



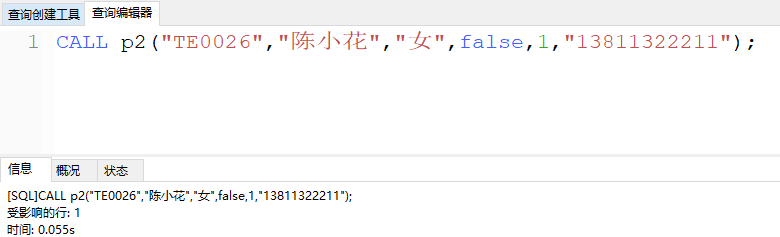
p1 (使用shop数据库中的表)

|  |
| --- |
| BEGIN  #Routine body goes here...  DECLARE dept\_id INT;  CASE  WHEN p\_sex="女" THEN  #分配到零售部  SET dept\_id=3;  ELSE  SET dept\_id=4;  END CASE;  INSERT INTO t\_emp(wid,ename,sex,married,education,tel,dept\_id,  hiredate,job\_id,`status`,is\_deleted)  VALUES(p\_wid,p\_ename,p\_sex,p\_married,p\_education,p\_tel,dept\_id,  CURDATE(),9,1,0);  END |

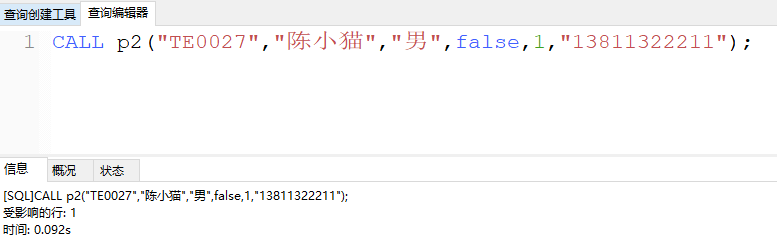
status不是单引号，是感叹号旁边的那个键。

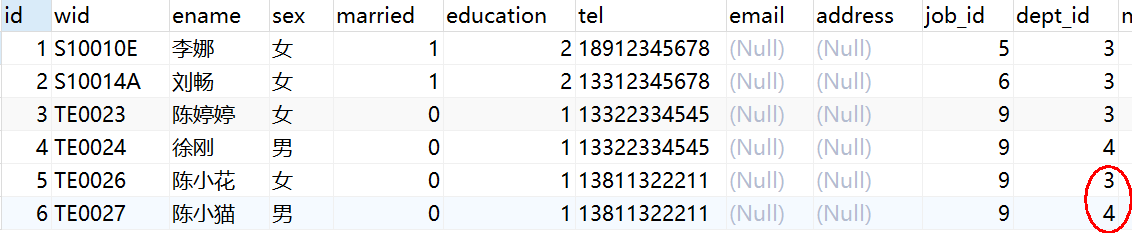
调用

CALL p2("TE0026","陈小花","女",false,1,"13811322211");



CALL p2("TE0027","陈小猫","男",false,1,"13811322211");





1. **函数**
2. **概念**

函数必须有返回值，并且是一个数据，不是多条记录。

1. **创建函数**

编写计算个税函数

个税计算公式=（收入-3500）\*税率-扣除数



新建函数







f1

|  |
| --- |
| BEGIN  #Routine body goes here...  DECLARE temp DECIMAL;  DECLARE tax DECIMAL;  SET temp=salary-3500;  CASE  WHEN temp<=0 THEN  SET tax=0;  WHEN temp>0 AND temp<=1500 THEN  SET tax=temp\*0.03-0;  WHEN temp>1500 AND temp<=4500 THEN  SET tax=temp\*0.10-105;  WHEN temp>4500 AND temp<=9000 THEN  SET tax=temp\*0.20-555;  WHEN temp>9000 AND temp<=35000 THEN  SET tax=temp\*0.25-1005;  WHEN temp>35000 AND temp<=55000 THEN  SET tax=temp\*0.30-2755;  WHEN temp>55000 AND temp<=80000 THEN  SET tax=temp\*0.35-5505;  ELSE  SET tax=temp\*0.45-13505;  END CASE;  RETURN tax;  END |

保存时报错

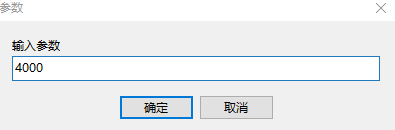
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you \*might\* want to use the less safe log\_bin\_trust\_function\_creators variable)

解决方法：

mysql>set global log\_bin\_trust\_function\_creators=1;

1. **调用函数**

点运行，传入参数





在sql中调用

SELECT f1(4000);



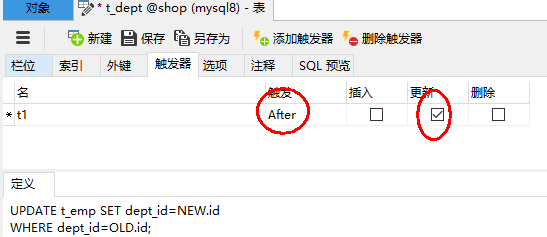
1. **触发器**
2. **概念**

触发器（trigger）是种特殊的存储过程，它的执行不是由程序调用，也不需要手动操作，它是由事件来触发，比如按钮的Click事件、网页的Load事件等。按钮的Click事件是通过鼠标单击按钮触发的，而触发器的事件，是由对表进行增删改操作所触发的。当对一个数据库或表进行增删改（ Insert，Delete，Update）的时就会激活触发器。

1. **创建触发器**

编写触发器，修改部门编号，同步更新该部门员工的dept\_id字段

找到部门表，在设计表中选触发器



t1

|  |
| --- |
| UPDATE t\_emp SET dept\_id=NEW.id  WHERE dept\_id=OLD.id; |

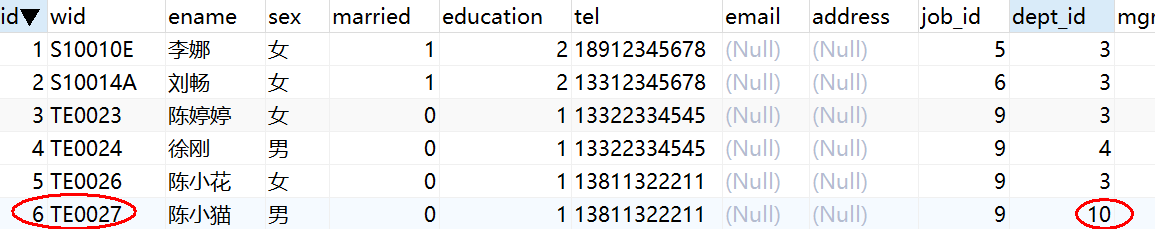
创建后保存

1. **测试**

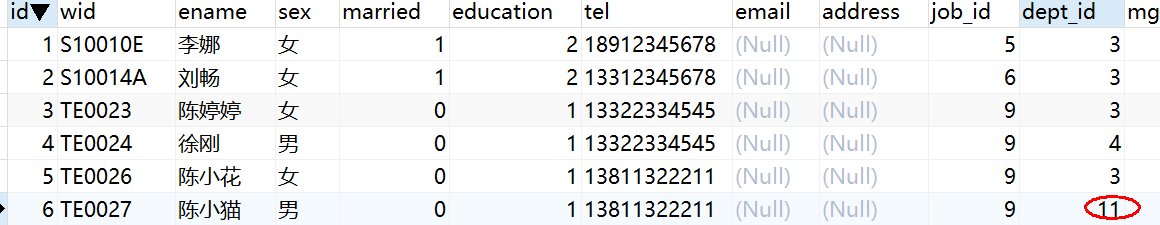
t\_dept



**t\_emp**

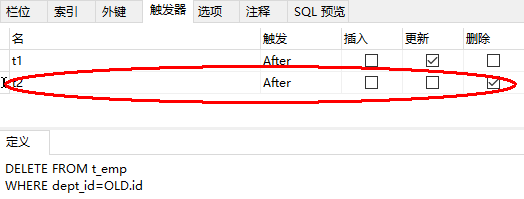


现在将测试部id由10改为11



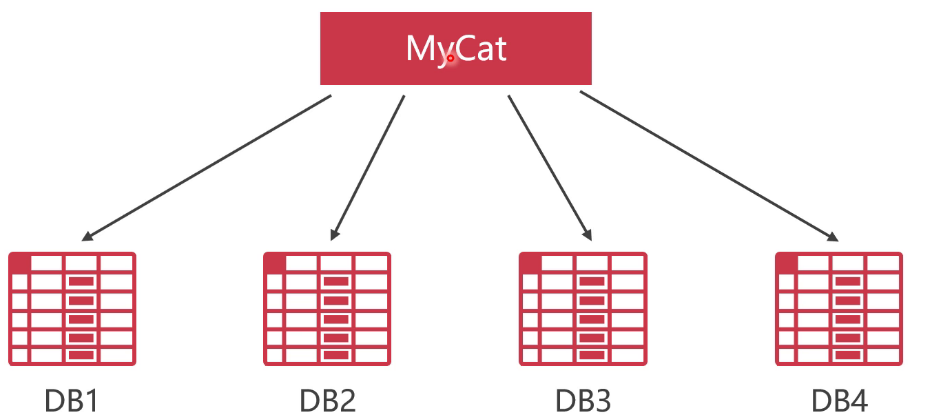
1. **案例2**

当删除部门后，删除这个部门的员工记录。



1. **为什么要放弃存储过程、函数、触发器**

在数据库集群场景里，因为存储过程，触发器、自定义函数都是在本地数据库执行，所以无法兼容集群场景。



在DB1到DB4中存储不同类型的商品，DB1存储食品，DB2存储电器，mycat相当于是sql路由，可以从sql语句中拿到商品的类型然后发送到不同的DB中存储。但是如何将sql做成存储过程则mycat无法解析出商品类型就无法发送到不同的商品库，导致数据库集群失效。

1. **Sql**
2. **Sql语法**

SELECT[DISTINCT] \* |分组字段1 [别名] [,分组字段2 [别名],…] | 统计函数

FROM表名　［表名］，［表名，［别名］，．．．］

[WHERE 条件]

[GROUP BY 分组字段1 [,分组字2,…]]

[HAVING 分组后的过滤条件]

[ORDER BY 排序字段 ASC｜DESC [,排序字段 ASC |DESC]];

SQL语句的执行顺序：

查询中用到的关键词主要包含六个，并且他们的顺序依次为

select--from--where--group by--having--order by

其中select和from是必须的，其他关键词是可选的，这六个关键词的执行顺序

与sql语句的书写顺序并不是一样的，而是按照下面的顺序来执行

from--where--group by--having

from:需要从哪个数据表检索数据

where:过滤表中数据的条件

group by:如何将上面过滤出的数据分组

having:对上面已经分组的数据进行过滤的条件

select:查看结果集中的哪个列，或列的计算结果

rder by :按照什么样的顺序来查看返回的数据

1. **数据库约束**

约束条件是我们建表的时候对数据库表做的一个限制条件。MySQL建表时候一般有如下的五个约束条件：

主键约束（Primay Key Coustraint） 唯一性，非空性

唯一约束 （Unique Counstraint）唯一性，可以空，但只能有一个

检查约束 (Check Counstraint) 对该列数据的范围、格式的限制

默认约束 (Default Counstraint) 该数据的默认值

外键约束 (Foreign Key Counstraint) 需要建立两表间的关系并引用主表的列

1. **左、右连接**

**1、内联接**

（典型的联接运算，使用像 =  或 <> 之类的比较运算符）。包括相等联接和自然联接。       
内联接使用比较运算符根据每个表共有的列的值匹配两个表中的行。例如，检索 students和courses表中学生标识号相同的所有行。    **2、外联接。**外联接可以是左向外联接、右向外联接或完整外部联接。       
在 FROM子句中指定外联接时，可以由下列几组关键字中的一组指定：       
1）LEFT  JOIN或LEFT OUTER JOIN       
左向外联接的结果集包括  LEFT OUTER子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值。         
2）RIGHT  JOIN 或 RIGHT  OUTER  JOIN       
右向外联接是左向外联接的反向联接。将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值。         
3）FULL  JOIN 或 FULL OUTER JOIN  
完整外部联接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。     
**3、交叉联接**交叉联接返回左表中的所有行，左表中的每一行与右表中的所有行组合。交叉联接也称作笛卡尔积。      
FROM 子句中的表或视图可通过内联接或完整外部联接按任意顺序指定；但是，用左或右向外联接指定表或视图时，表或视图的顺序很重要。有关使用左或右向外联接排列表的更多信息，请参见使用外联接。

1. **数据库设计的三范式**

**第一范式1NF:做到每列不可拆分**

中国北京可以拆分成两个字段中国和北京

**第二范式2NF:确保一个表只做一件事情**

id name sex address grade

1 tom male beijin 80

对于grade可以专门定义一张成绩表

**第三范式3NF:在满足2NF下，消除表中的传递依赖**

推理规则：A->B,B->C,则A->C 如果这种情况存在，就要消除这种关系。目的是为 了使空间最省。

id name number price totalprice

1 风扇 20 200 4000

id是A , number和price是B totalprice是C

id ->number,price->totalprice

可以将totalprice字段删除，节省空间。

**反三范式（用空间换时间）：**在进行数据库设计时，首先要确保表结构要达到第三 范式的要求。但在实际开发时，为满足业务的需要，会在符合三范式的表中加入冗 余字段，结果导致不符合第三范式的要求。

因为存在冗余字段需要维护它，如购销合同中的合同总金额就是一个冗余字段（由 于在页面上要显示总金额，不将计算的操作放在页面上进行，而是在数据库中计算 好直接获取到页面上。），就需要在添加货物或添加附件时，都要去更新购销合同中 的总金额。

1. **分页**

**Mysql**

0表示第一条记录的索引号，索引号从0开始, 2表示最多选取二个记录

select \* from users limit 0,2或 select \* from users limit 2;

查询出users第2条到第4条记录

select \* from users limit 1,3;

**Oracle**

方式一使用集合减运算: 显示emp表中3-8条记录

select rownum "伪列",emp.\* from emp where rownum<=8

minus

select rownum,emp.\* from emp where rownum<=2;

方式二：使用子查询，在from子句中使用，重点

select xx.\*

from (select rownum ids,emp.\* from emp where rownum<=8) xx

where ids>=3;

其中的xx是子查询结果表的别名

1. **drop、delete与truncate的区别**

Delete用来删除表的全部或者一部分数据行，执行delete之后，用户需要提交(commmit)或者回滚(rollback)来执行删除或者撤销删除， delete命令会触发这个表上所有的delete触发器；

Truncate删除表中的所有数据，这个操作不能回滚，也不会触发这个表上的触发器，TRUNCATE比delete更快，占用的空间更小；

Drop命令从数据库中删除表，所有的数据行，索引和权限也会被删除，所有的DML触发器也不会被触发，这个命令也不能回滚。

因此，在不再需要一张表的时候，用drop；在想删除部分数据行时候，用delete；在保留表而删除所有数据的时候用truncate。

1. **常用SQL语句**
2. 创建数据库   
   CREATE DATABASE database-name;
3. 删除数据库   
   DROP DATABASE database-name;
4. 创建新表   
   create table depart (dept\_id int(11) NOT NULL AUTO\_INCREMENT,   
   dept\_name varchar(255) DEFAULT NULL, PRIMARY KEY (dept\_id));   
   根据已有的表创建新表：   
   create table tab\_new like tab\_old (使用旧表B创建新表A)   
   备注：此种方式在将表B复制到A时候会将表B完整的字段结构和索引复制到表A中来   
   create table tab\_new as select col1,col2… from tab\_old definition only   
   备注：此种方式只会将表B的字段结构复制到表A中来，但不会复制表B中的索引到表A中来。这种方式比较灵活可以在复制原表表结构的同时指定要复制哪些字段，并且自身复制表也可以根据需要增加字段结构。   
   create table as select 会将原表中的数据完整复制一份，但表结构中的索引会丢失。   
   create table like 只会完整复制原表的建表语句，但不会复制数据。
5. 删除新表   
   drop table tabname；
6. 增加一个列   
   alter table tabname add column column\_name type
7. 添加主键： Alter table tabname add primary key(col)   
   删除主键： Alter table tabname drop primary key   
   一个数据表只可以有一个主键，所以不存在删除某一列的主键.
8. 创建索引：create [unique] index idxname on tabname(col….)   
   删除索引：drop index idxname   
   注：索引是不可更改的，想更改必须删除重新建。
9. 创建视图：create view viewname as select statement   
   删除视图：drop view viewname
10. 几个简单的基本的sql语句   
    选择：select \* from table1 where 范围   
    插入：insert into table1(field1,field2) values(value1,value2)   
    删除：delete from table1 where 范围   
    更新：update table1 set field1=value1 where 范围   
    查找：select \* from table1 where field1 like ’%value1%’   
    排序：select \* from table1 order by field1,field2 [desc]   
    desc：降序，asc：升序   
    总数：select count as totalcount from table1   
    求和：select sum(field1) as sumvalue from table1   
    平均：select avg(field1) as avgvalue from table1   
    最大：select max(field1) as maxvalue from table1   
    最小：select min(field1) as minvalue from table1
11. 分组:Group by:   
    一张表，一旦分组完成后，查询后只能得到组相关的信息。   
    组相关的信息：（统计信息） count,sum,max,min,avg 分组的标准)
12. 复制表select \* into b from a where 1<>1（仅用于SQlServer）
13. 子查询(表名1：a 表名2：b) select a,b,c from a where a IN (select d from b )
14. 显示文章、提交人和最后回复时间   
    select a.title,a.username,b.adddate from table a,(select max(adddate) adddate from table where table.title=a.title) b
15. 视图查询(表名1：a )   
    select \* from (SELECT a,b,c FROM a) T where t.a > 1;
16. between的用法,between限制查询数据范围时包括了边界值,not between不包括   
    select \* from table1 where time between time1 and time2   
    select a,b,c, from table1 where a not between 数值1 and 数值2
17. in 的使用方法   
    select \* from table1 where a [not] in (‘值1’,’值2’,’值4’,’值6’)
18. 两张关联表，删除主表中已经在副表中没有的信息   
    delete from table1 where not exists ( select \* from table2 where table1.field1=table2.field1 )
19. 日程安排提前五分钟提醒   
    SQL: select \* from 日程安排 where datediff(‘minute’,f开始时间,getdate())>5
20. 前10条记录   
    select top 10 \* form table1 where 范围
21. 选择在每一组b值相同的数据中对应的a最大的记录的所有信息(类似这样的用法可以用于论坛每月排行榜,每月热销产品分析,按科目成绩排名,等等.)   
    select a,b,c from tablename ta where a=(select max(a) from tablename tb where tb.b=ta.b)
22. 包括所有在 TableA 中但不在 TableB和TableC 中的行并消除所有重复行而派生出一个结果表   
    (select a from tableA ) except (select a from tableB) except (select a from tableC)
23. 随机取出10条数据  select top 10 \* from tablename order by newid()
24. **优化**
25. **数据库性能优化**

优化SQL语句和索引，在where/group by/order by中用到的字段建立索引，索引字段越小越好，复合索引建立的顺序

加缓存，Memcached, Redis

主从复制，读写分离

垂直拆分，其实就是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

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；

1. **SQL优化**

优化步骤：

根据慢日志定位慢查询SQL

用explain分析SQL（type和extra字段分析）

修改SQL或加索引（如下）

对经常查询的列建立索引，但索引建多了当数据改变时修改索引会增大开销

使用精确列名查询而不是\*，特别是当数据量大的时候

减少子查询，使用Join替代

不用NOT IN，因为会使用全表扫描而不是索引；不用IS NULL，NOT IS NULL，因为会使索引、索引统计和值更加复杂，并且需要额外一个字节的存储空间。

问：max(xxx)如何用索引优化？ 答：在xxx列上建立索引，因为索引是B+树顺序排列的，在下次查询的时候就会使用索引来查询到最大的值是哪个

问：有个表特别大，字段是姓名、年龄、班级，如果调用select \* from table where name = xxx and age = xxx该如何通过建立索引的方式优化查询速度？

答：由于mysql查询每次只能使用一个索引，如果在name、age两列上创建联合索引的话将带来更高的效率。如果我们创建了(name, age)的联合索引，那么其实相当于创建了(name, age)、(name)2个索引。因此我们在创建复合索引时应该将最常用作限制条件的列放在最左边，依次递减。其次还要考虑该列的数据离散程度，如果有很多不同的值的话建议放在左边，name的离散程度也大于age。

1. **优化方面**

数据库调优可以从两个方面考虑：一是应用层架构；二是数据库架构；

**1. 应用层架构优化**。核心是减少对数据库的调用次数，本质上是从业务应用层来审视流程是否合理，常用的方案有：

（1）引入缓存redis，虚拟一层中间层，减少对数据库的读写；

（2）将多次单个调用改为批量调用，比如说循环十次主键 select \* FROM t where id = 'xx’改为使用 IN 一性次读取 select \* FROM t where id IN (‘xx’,‘xx’,…)；

(3) 使用搜索引擎es。

**2. 数据库架构优化**。核心是优化各种配置，提升数据库的性能，可分为：

**（1）优化 SQL 及索引**，以达到减少数据访问、返回更少数据、减少交互次数等目标。常用的手段包括：创建并正确地使用索引（比如说减少回表查询）、优化执行计划、数据分页处理、只取需要的字段、慢查询优化、使用搜索引擎等；

例如：

永远用小结果集驱动大结果集（mysql数据库的联表查询利用join的嵌套查询，类似于双层for循环，最外层循环是循环某一张表里的行数，让行数少的表去做外层循环，取出表中的每一行再去内层表查询，所以外层表循环的次数越少越好）

避免复杂的join和子查询，不要超过3张表

合理利用并使用索引，索引一定不是越多越好。不要造成索引失效。

针对百万数量级，放弃在mysql中的join操作，推荐分别根据索引单表取数据，然后在程序中做join，merge数据。

不在数据库中做运算，运算移至业务层

平衡范式与冗余（效率优先，往往会牺牲范式）

拒绝3B(大sql、大事务、大批量)

避免使用null字段（null字段很难查询优化、null字段的复合索引无效）

少用text类型，尽量用varchar代替

避免使用触发器、函数

**（2）优化数据库表结构**。常用的的手段包括：使用占用空间最小的数据类型、使用简单的数据类型、尽可能地使用 NOT NULL 定义字段、尽量少使用 text 字段、分库分表等；

**（3）优化系统配置**。包括操作系统的配置优化和数据库的配置优化。

A、操作系统优化。数据库是基于操作系统（多为 Linux 系统）的，所以对于合理使用操作系统也会影响到数据库的性能。比如将数据库服务器应和业务服务器隔离、或者设置 net.ipv4.tcp\_max\_syn\_backlog = 65535 以增加 tcp 支持的队列数等等；

B、数据库配置文件优化，以 MySQL 配置为例，可以修改 innodb\_buffer\_pool\_size（设置数据和索引缓存的缓冲池）、max\_connections 等参数。

**（4）优化硬件配置**。比如说使用更好的服务器、更快的硬盘、更大的内存等等。

另一个需要注意的点是切忌上来就说用 explain 去解析执行计划，而应该把调优视为系统化的方案和工程。而且即使是使用 explain 查询执行计划，也需要清晰地描述其步骤：

（1）查看 slowlog，分析 slowlog，分析出查询慢的语句；

（2）按照一定优先级（比如说调用频率），进行一个一个的排查所有慢语句。

（3）分析 top sql，进行 explain 调试，查看语句执行时间。

（4）调整索引或语句本身。

1. 数据库设计

1.根据业务需求，将表与表之间的关系可以分类3大类，分别是一对一，一对多和多对多。在一对一设计时候我们即可以设计成一张表，也可以把一张表中不常用的大字段切分到单独的一张子表中通过一对一主键关联进行连接。如商品表可以将商品的基本信息和商品的描述分为两张表，通过一对一主键关联来建立关联关系。当表与表之间的关系是一对多时通常需要建立两张表并通过主外键进行关联而外键在多的一端。如果表与表之间的关系是多对多则需要通过中间表进行关联。

2.遵守数据库设计的三范式。

第一范式（1NF）：强调的是列的原子性，即列不能够再分成其他几列。

第二范式（2NF）: 首先是满足第一范式，另外包含两部分内容，一是表必须有一个主键；二是没有包含在主键中的列必须完全依赖于主键，而不是部分依赖。

第三范式（3NF）: 首先满足第二范式，非主键列直接依赖于主键，消除传递依赖。

但是，没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是： 在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，允许冗余。

3.对于频繁查询的字段加索引并且在编写sql语句时注意sql优化。

 外键必须加索引。

 避免在 where 子句中对有索引的字段进行运算,这会导致索引失效，从而进行全表扫描。

 在 where 及 order by 涉及的列上建立索引，要尽量避免全表扫描。

 在设计表时要避免表中字段出现null的情况，通常要为其设置默认值,避免在查找时放弃 使用索引而进行全表扫描。

 SELECT语句中避免使用'\*’，只查询需要返回的字段 ，这样可以减少oracle解析sql语句 的时间。

 用NOT EXISTS 替换 NOT IN 操作符，用 EXISTS 替换 IN

4.对于数据量大的表进行分区

分区支持的种类比较多，如Range（范围），Hash（哈希），List（预定义列表）

分区等。我们在项目中使用的是Range范围分区。考虑到日志表会随着系统的运行

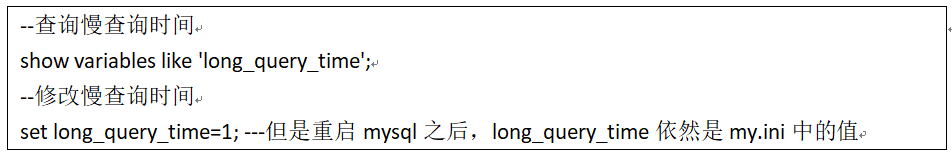
时间延长而不断的加大。所以我们就按照插入时间对日志表做了范围分区。结合

PARTITION BY RANGE以及VALUES LESS THAN以每年为分界线进行分区。

1. **什么是慢查询**

MySQL默认10秒内没有响应SQL结果,则为慢查询

可以去修改MySQL慢查询默认时间



在默认情况下，mysql不会记录慢查询，需要在启动mysql时候，指定记录慢查询才可以

bin\mysqld.exe --safe-mode --slow-query-log [mysql5.5 可以在my.ini指定]（安全模式启动，数据库将操作写入日志，以备恢复）

bin\mysqld.exe –log-slow-queries=d:/abc.log [低版本mysql5.0可以在my.ini指定]

先关闭mysql,再启动, 如果启用了慢查询日志，默认把这个文件放在my.ini 文件中记录的位置

#Path to the database root

datadir=" C:/ProgramData/MySQL/MySQL Server 5.5/Data/"

1. **维护数据库完整性和一致性**

尽可能使用约束：check、主外键、非空字段等，方便效率高；其次使用触发器。最后是自写业务逻辑

1. **连接池优点**

JDBC数据库连接池的必要性 没使用数据库连接池之前

1. 使用时连接数据库，使用后断开数据库。
2. 对于每一次数据库连接，使用完后都得断开。否则，如果程序出现异常而未能关闭，将会导致数据库系统中的内存泄漏，最终将导致重启数据库
3. 如果成千上万的用户同时使用数据库，频繁的进行数据库连接操作将占用很多的系统资源，严重的甚至会造成服务器的崩溃

新的资源分配手段对于多应用共享同一数据库的系统而言，可在应用层通过数据库连接池的配置实现某一应用最大可用数据库连接数的限制避免某一应用独占所有的数据库资源.

统一的连接管理，避免数据库连接泄露在较为完善的数据库连接池实现中，可根据预先的占用超时设定，强制回收被占用连接，从而避免了常规数据库连接操作中可能出现的资源泄露。

数据库连接池技术的优点

(1)资源重用：由于数据库连接得以重用，避免了频繁创建，释放连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增加了系统运行环境的平稳性。

(2)更快的系统反应速度:数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于连接池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接避免了数据库连接初始化和释放过程的时间开销，从而减少了系统的响应时间

1. **数据库数据量过大，数据库接近崩溃的时候怎么办？**

1、索引优化和SQL语句优化是必须的，避免模糊查询和非索引查询，删改操作根据聚集索引进行，删改操作太频繁的话还是需要考虑分表

2、看需求，如果需求不限制，那就分表

3、一般都是把历史数据定期转存其他表（一样的表名后加年月例如TABLE201205）归档。这样该表本年度的查询的压力也小点（90%查询量集中在本年度）,即使查询历史数据也不影响性能，强力推荐！

4结合你的业务去优化表结构。有时候可以考虑用空间去换时间。

1. **百万级别或以上的数据如何删除**

关于索引：由于索引需要额外的维护成本，因为索引文件是单独存在的文件,所以当我们对数据的增加,修改,删除,都会产生额外的对索引文件的操作,这些操作需要消耗额外的IO,会降低增/改/删的执行效率。所以，在我们删除数据库百万级别数据的时候，查询MySQL官方手册得知删除数据的速度和创建的索引数量是成正比的。

所以我们想要删除百万数据的时候可以先删除索引（此时大概耗时三分多钟）

然后删除其中无用数据（此过程需要不到两分钟）

删除完成后重新创建索引(此时数据较少了)创建索引也非常快，约十分钟左右。

与之前的直接删除绝对是要快速很多，更别说万一删除中断,一切删除会回滚。那更是坑了。

1. **以下 or 查询有什么问题吗？该如何优化？**

select \* from t where num=10 or num=20;

答：如果使用 or 查询会使 MySQL 放弃索引而全表扫描，可以改为：

select \* from t where num=10 union

select \* from t where num=20;

1. **练习**
2. **行列转换**

CREATE TABLE Scores(

ID INT,

Student varchar(10),

Subject varchar(10),

Score INT

);

INSERT INTO Scores VALUES(2, '张三', '语文', 93);

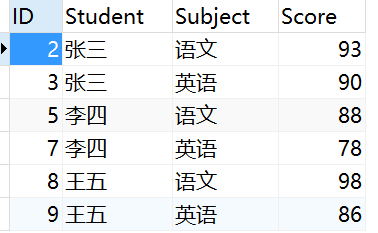
INSERT INTO Scores VALUES(3, '张三', '英语', 90);

INSERT INTO Scores VALUES(5, '李四', '语文', 88);

INSERT INTO Scores VALUES(7, '李四', '英语', 78);

INSERT INTO Scores VALUES(8, '王五', '语文', 98);

INSERT INTO Scores VALUES(9, '王五', '英语', 86);



SELECT student, max(yw) as '语文', max(yu) as '英语'

FROM(

SELECT student,

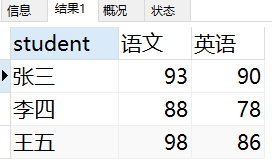
CASE subject WHEN '语文' THEN score ELSE 0 END as yw,

CASE subject WHEN '英语' THEN score ELSE 0 END as yu

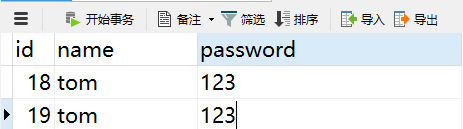
FROM Scores

) A

GROUP BY student



1. **删除重复数据**



DELETE FROM person WHERE id NOT IN (SELECT tab.id FROM

(SELECT MAX(id) AS id FROM person GROUP BY NAME ,PASSWORD

) AS tab )

这里面的第一个关键点是如何确定哪些数据是重复的呢？这里通过 group by name,password来确定。只要同时满足这两个条件，就可以归为相同一组数据

第二个关键点是通过max(id) 确定保留下的数据是id最大的。

第三点是 最内层的select语句的查询结果需要重启起名称为tab ，再从tab 中查出id .如果缺少这一步，语句会报错。

第四步是根据 id not in 删除除最大id外的所有数据

1. **省市县三级联动数据库是怎么设计的**

CREATE TABLE t\_area (

id int(11) NOT NULL auto\_increment,

area\_id varchar(50) default NULL,

area\_name varchar(60) default NULL,

area\_parentId varchar(6) default NULL,

primary key(id)

) ENGINE=MyISAM AUTO\_INCREMENT=3145 DEFAULT CHARSET=utf8;

INSERT INTO t\_area VALUES (1,'100','河北省','0');

INSERT INTO t\_area VALUES (2,'220999','石家庄市','100');

1. **用一条SQL语句查询出每门课都大于80分的学生姓名**

name kecheng fenshu

张三 语文 81

张三 数学 75

李四 语文 76

李四 数学 90

王五 语文 81

王五 数学 100

王五 英语 90

答案：

A：select distinct name from score where name not in (select distinct name from score where fenshu <=80)

B：select distinct name t1 from score where 80< all (select fenshu from score where name=t1);

C:select name from 表名 group by 姓名 having MIN(fenshu) >= 80

1. **练习1**

表结构

学生表student(id,name)

课程表course(id,name)

学生课程表student\_course(sid,cid,score)

**创建表的sql代码**

|  |
| --- |
| create table student(  id int unsigned primary key auto\_increment,  name char(10) not null  );  insert into student(name) values('张三'),('李四');  create table course(  id int unsigned primary key auto\_increment,  name char(20) not null  );  insert into course(name) values('语文'),('数学');  create table student\_course(  sid int unsigned,  cid int unsigned,  score int unsigned not null,  foreign key (sid) references student(id),  foreign key (cid) references course(id),  primary key(sid, cid)  );  insert into student\_course values(1,1,80),(1,2,90),(2,1,90),(2,2,70); |

**问题**

**1）查询student表中重名的学生，结果包含id和name，按name,id升序**

select id,name

from student

where name in (

select name from student group by name having(count(\*) > 1)

) order by name;

我们经常需要查询某一列重复的行，一般通过group by(有重复的列)然后取count>1的值。 关系型数据库有他的局限性， 有些看似简单的查询写出来的sql很复杂，而且效率也会很低。

**2）在student\_course表中查询平均分不及格的学生，列出学生id和平均分**

select sid,avg(score) as avg\_score

from student\_course

group by sid having(avg\_score<60);

group by和having是最常考的。 where子句中不能用聚集函数作为条件表达式，但是having短语可以，where和having的区别在于对用对象不同，where作用于记录，having作用于组。

**3）在student\_course表中查询每门课成绩都不低于80的学生id**

select distinct sid

from student\_course

where sid not in (

select sid from student\_course

where score < 80);

用到反向思想，其实就是数理逻辑中的∀x:P和¬∃x:¬P是等价的。

**4）查询每个学生的总成绩，结果列出学生姓名和总成绩 如果使用下面的sql会过滤掉没有成绩的人**

select name,sum(score) total

from student,student\_course

where student.id=student\_course.sid

group by sid;

更保险的做法应该是使用 左外连接

select name,sum(score)

from student left join student\_course

on student.id=student\_course.sid

group by sid;

**5）总成绩最高的学生，结果列出学生id和总成绩 下面的sql效率很低，因为要重复计算所有的总成绩。**

select sid,sum(score) as sum\_score

from student\_course group by sid having sum\_score>=all

(select sum(score) from student\_course group by sid);

因为order by中可以使用聚集函数，最简单的方法是：

select sid,sum(score) as sum\_score

from student\_course group by sid

order by sum\_score desc limit 1;

同理可以查总成绩的前三名。

**6）在student\_course表查询课程1成绩第2高的学生，如果第2高的不止一个则列出所有的学生**

这是个查询 第N大数 的问题。 我们先查出第2高的成绩：

select min(score) from student\_course where cid = 1 group by score order by score desc limit 2;

使用这种方式是错的，因为作用的先后顺序是group by->min->order by->limit，mysql提供了limit offset,size这种方式来取第N大的值，因此正确的做法是：

select score from student\_course where cid = 1 group by score order by score desc limit 1,1;

然后再取出该成绩对应的学生：

select \* from student\_course

where cid=1 and score = (

select score from student\_course where cid = 1 group by score order by score desc limit 1,1

);

类似的，可以查询 某个值第N高 的记录。

**7）在student\_course表查询各科成绩最高的学生，结果列出学生id、课程id和对应的成绩**

你可能会这样写：

select sid,cid,max(score) from student\_course group by cid;

然而上面是不对的，因为 使用了group by的查询字段只能是group by中的字段或者聚集函数或者是每个分组内均相同的字段。 虽然不会报错，但是sid是无效的，如果去掉sid的话只能查出每门课程的最高分，不包含学生id。 本题的正确解法是使用相关嵌套查询：

select \* from student\_course as x where score>=

(select max(score) from student\_course as y where cid=x.cid);

相关嵌套查询也就是在进行内层查询的时候需要用到外层查询，有一些注意事项：

子查询一定要有括号

as可以省略

使用相关查询；>=max等价于>=all，但是聚合函数比使用any或all效率高

**8）在student\_course表中查询每门课的前2名，结果按课程id升序，同一课程按成绩降序**

这个问题也就是取每组的前N条纪录

select \* from student\_course x where

2>(select count(distinct(score)) from student\_course y where y.cid=x.cid and y.score>x.score)

order by cid,score desc;

这也是一个相关嵌套查询，对于每一个分数，如果同一门课程下只有0个、1个分数比这个分数还高，那么这个分数肯定是前2名之一

**9）一个叫team的表，里面只有一个字段name,一共有4条纪录，分别是a,b,c,d,对应四个球队，两两进行比赛，用一条sql语句显示所有可能的比赛组合**

select a.name, b.name

from team a, team b

where a.name < b.name

其实就是一个表和自己连接查询。

**10）题目：数据库中有一张如下所示的表，表名为sales。**

年 季度 销售

1991 1 11

1991 2 12

1991 3 13

1991 4 14

1992 1 21

1992 2 22

1992 3 23

1992 4 24

要求：写一个SQL语句查询出如下所示的结果。

年 一季度 二季度 三季度 四季度

1991 11 12 13 14

1992 21 22 23 24

select 年,

sum(case when 季度=1 then 销售量 else 0 end) as 一季度,

sum(case when 季度=2 then 销售量 else 0 end) as 二季度,

sum(case when 季度=3 then 销售量 else 0 end) as 三季度,

sum(case when 季度=4 then 销售量 else 0 end) as 四季度

from sales group by 年;

同理，如果要查询每个人的每门课的成绩可以使用如下sql

create view temp as select student.name as sname,course.name as cname,score

from student\_course join (student,course)

on(student\_course.sid=student.id and student\_course.cid=course.id)

;

select sname,

sum(case when cname='语文' then score else 0 end) as 语文,

sum(case when cname='数学' then score else 0 end) as 数学

from temp

group by sname;

当然如果新增了一门课，第二条sql就需要跟着变。

1. **练习2**

本题用到下面三个关系表：

CARD 借书卡。 CNO 卡号，NAME 姓名，CLASS 班级

BOOKS 图书。 BNO 书号，BNAME 书名,AUTHOR 作者，PRICE 单价，QUANTITY 库存册数

BORROW 借书记录。 CNO 借书卡号，BNO 书号，RDATE 还书日期

备注：限定每人每种书只能借一本；库存册数随借书、还书而改变。

要求实现如下15个处理：

|  |
| --- |
| 1. 写出建立BORROW表的SQL语句，要求定义主码完整性约束和引用完整性约束  --实现代码：  CREATE TABLE BORROW(  CNO **int** FOREIGN KEY REFERENCES CARD(CNO),  BNO **int** FOREIGN KEY REFERENCES BOOKS(BNO),  RDATE datetime,  PRIMARY KEY(CNO,BNO))  2. 找出借书超过5本的读者,输出借书卡号及所借图书册数  --实现代码：  SELECT CNO,借图书册数=COUNT(\*)  FROM BORROW  GROUP BY CNO  HAVING COUNT(\*)>5  3. 查询借阅了"水浒"一书的读者，输出姓名及班级  --实现代码：  SELECT \* FROM CARD c  WHERE EXISTS(  SELECT \* FROM BORROW a,BOOKS b  WHERE a.BNO=b.BNO  AND b.BNAME=N'水浒'  AND a.CNO=c.CNO)  4. 查询过期未还图书，输出借阅者（卡号）、书号及还书日期  --实现代码：  SELECT \* FROM BORROW  WHERE RDATE<GETDATE()  5. 查询书名包括"网络"关键词的图书，输出书号、书名、作者  --实现代码：  SELECT BNO,BNAME,AUTHOR FROM BOOKS  WHERE BNAME LIKE N'%网络%'  6. 查询现有图书中价格最高的图书，输出书名及作者  --实现代码：  SELECT BNO,BNAME,AUTHOR FROM BOOKS  WHERE PRICE=(  SELECT MAX(PRICE) FROM BOOKS)  7. 查询当前借了"计算方法"但没有借"计算方法习题集"的读者，输出其借书卡号，并按卡号降序排序输出  --实现代码：  SELECT a.CNO  FROM BORROW a,BOOKS b  WHERE a.BNO=b.BNO AND b.BNAME=N'计算方法'  AND NOT EXISTS(  SELECT \* FROM BORROW aa,BOOKS bb  WHERE aa.BNO=bb.BNO  AND bb.BNAME=N'计算方法习题集'  AND aa.CNO=a.CNO)  ORDER BY a.CNO DESC  8. 将"C01"班同学所借图书的还期都延长一周  --实现代码：  UPDATE b SET RDATE=DATEADD(Day,7,b.RDATE)  FROM CARD a,BORROW b  WHERE a.CNO=b.CNO  AND a.CLASS=N'C01'  9. 从BOOKS表中删除当前无人借阅的图书记录  --实现代码：  DELETE A FROM BOOKS a  WHERE NOT EXISTS(  SELECT \* FROM BORROW  WHERE BNO=a.BNO)  10. 如果经常按书名查询图书信息，请建立合适的索引  --实现代码：  CREATE CLUSTERED INDEX IDX\_BOOKS\_BNAME ON BOOKS(BNAME)  11. 在BORROW表上建立一个触发器，完成如下功能：如果读者借阅的书名是"数据库技术及应用"，就将该读者的借阅记录保存在BORROW\_SAVE表中（注ORROW\_SAVE表结构同BORROW表）  --实现代码：  CREATE TRIGGER TR\_SAVE ON BORROW  FOR INSERT,UPDATE  AS  IF @@ROWCOUNT>0  INSERT BORROW\_SAVE SELECT i.\*  FROM INSERTED i,BOOKS b  WHERE i.BNO=b.BNO  AND b.BNAME=N'数据库技术及应用'  12. 建立一个视图，显示"力01"班学生的借书信息（只要求显示姓名和书名）  --实现代码：  CREATE VIEW V\_VIEW  AS  SELECT a.NAME,b.BNAME  FROM BORROW ab,CARD a,BOOKS b  WHERE ab.CNO=a.CNO  AND ab.BNO=b.BNO  AND a.CLASS=N'力01'  13. 查询当前同时借有"计算方法"和"组合数学"两本书的读者，输出其借书卡号，并按卡号升序排序输出  --实现代码：  SELECT a.CNO  FROM BORROW a,BOOKS b  WHERE a.BNO=b.BNO  AND b.BNAME IN(N'计算方法',N'组合数学')  GROUP BY a.CNO  HAVING COUNT(\*)=2  ORDER BY a.CNO DESC  14. 假定在建BOOKS表时没有定义主码，写出为BOOKS表追加定义主码的语句  --实现代码：  ALTER TABLE BOOKS ADD PRIMARY KEY(BNO)  15.1 将NAME最大列宽增加到10个字符（假定原为6个字符）  --实现代码：  ALTER TABLE CARD ALTER COLUMN NAME varchar(10)  15.2 为该表增加1列NAME（系名），可变长，最大20个字符  --实现代码：  ALTER TABLE CARD ADD 系名 varchar(20) |

1. **练习3**

**表名和字段**

–1.学生表

Student(s\_id,s\_name,s\_birth,s\_sex) –学生编号,学生姓名, 出生年月,学生性别

–2.课程表

Course(c\_id,c\_name,t\_id) – –课程编号, 课程名称, 教师编号

–3.教师表

Teacher(t\_id,t\_name) –教师编号,教师姓名

–4.成绩表

Score(s\_id,c\_id,s\_score) –学生编号,课程编号,分数

**测试数据**

-- 建表

-- 学生表

CREATE TABLE `Student`(

`s\_id` VARCHAR(20),

`s\_name` VARCHAR(20) NOT NULL DEFAULT '',

`s\_birth` VARCHAR(20) NOT NULL DEFAULT '',

`s\_sex` VARCHAR(10) NOT NULL DEFAULT '',

PRIMARY KEY(`s\_id`)

);

-- 课程表

CREATE TABLE `Course`(

`c\_id` VARCHAR(20),

`c\_name` VARCHAR(20) NOT NULL DEFAULT '',

`t\_id` VARCHAR(20) NOT NULL,

PRIMARY KEY(`c\_id`)

);

-- 教师表

CREATE TABLE `Teacher`(

`t\_id` VARCHAR(20),

`t\_name` VARCHAR(20) NOT NULL DEFAULT '',

PRIMARY KEY(`t\_id`)

);

-- 成绩表

CREATE TABLE `Score`(

`s\_id` VARCHAR(20),

`c\_id` VARCHAR(20),

`s\_score` INT(3),

PRIMARY KEY(`s\_id`,`c\_id`)

);

-- 插入学生表测试数据

insert into Student values('01' , '赵雷' , '1990-01-01' , '男');

insert into Student values('02' , '钱电' , '1990-12-21' , '男');

insert into Student values('03' , '孙风' , '1990-05-20' , '男');

insert into Student values('04' , '李云' , '1990-08-06' , '男');

insert into Student values('05' , '周梅' , '1991-12-01' , '女');

insert into Student values('06' , '吴兰' , '1992-03-01' , '女');

insert into Student values('07' , '郑竹' , '1989-07-01' , '女');

insert into Student values('08' , '王菊' , '1990-01-20' , '女');

-- 课程表测试数据

insert into Course values('01' , '语文' , '02');

insert into Course values('02' , '数学' , '01');

insert into Course values('03' , '英语' , '03');

-- 教师表测试数据

insert into Teacher values('01' , '张三');

insert into Teacher values('02' , '李四');

insert into Teacher values('03' , '王五');

-- 成绩表测试数据

insert into Score values('01' , '01' , 80);

insert into Score values('01' , '02' , 90);

insert into Score values('01' , '03' , 99);

insert into Score values('02' , '01' , 70);

insert into Score values('02' , '02' , 60);

insert into Score values('02' , '03' , 80);

insert into Score values('03' , '01' , 80);

insert into Score values('03' , '02' , 80);

insert into Score values('03' , '03' , 80);

insert into Score values('04' , '01' , 50);

insert into Score values('04' , '02' , 30);

insert into Score values('04' , '03' , 20);

insert into Score values('05' , '01' , 76);

insert into Score values('05' , '02' , 87);

insert into Score values('06' , '01' , 31);

insert into Score values('06' , '03' , 34);

insert into Score values('07' , '02' , 89);

insert into Score values('07' , '03' , 98);

**练习题及答案**

-- 1、查询"01"课程比"02"课程成绩高的学生的信息及课程分数

select a.\* ,b.s\_score as 01\_score,c.s\_score as 02\_score from

student a

join score b on a.s\_id=b.s\_id and b.c\_id='01'

left join score c on a.s\_id=c.s\_id and c.c\_id='02' or c.c\_id = NULL where b.s\_score>c.s\_score

-- 2、查询"01"课程比"02"课程成绩低的学生的信息及课程分数

select a.\* ,b.s\_score as 01\_score,c.s\_score as 02\_score from

student a left join score b on a.s\_id=b.s\_id and b.c\_id='01' or b.c\_id=NULL

join score c on a.s\_id=c.s\_id and c.c\_id='02' where b.s\_score<c.s\_score

-- 3、查询平均成绩大于等于60分的同学的学生编号和学生姓名和平均成绩

select b.s\_id,b.s\_name,ROUND(AVG(a.s\_score),2) as avg\_score from

student b

join score a on b.s\_id = a.s\_id

GROUP BY b.s\_id,b.s\_name HAVING ROUND(AVG(a.s\_score),2)>=60;

-- 4、查询平均成绩小于60分的同学的学生编号和学生姓名和平均成绩

-- (包括有成绩的和无成绩的)

select b.s\_id,b.s\_name,ROUND(AVG(a.s\_score),2) as avg\_score from

student b

left join score a on b.s\_id = a.s\_id

GROUP BY b.s\_id,b.s\_name HAVING ROUND(AVG(a.s\_score),2)<60

union

select a.s\_id,a.s\_name,0 as avg\_score from

student a

where a.s\_id not in (

select distinct s\_id from score);

-- 5、查询所有同学的学生编号、学生姓名、选课总数、所有课程的总成绩

select a.s\_id,a.s\_name,count(b.c\_id) as sum\_course,sum(b.s\_score) as sum\_score from

student a

left join score b on a.s\_id=b.s\_id

GROUP BY a.s\_id,a.s\_name;

-- 6、查询"李"姓老师的数量

select count(t\_id) from teacher where t\_name like '李%';

-- 7、查询学过"张三"老师授课的同学的信息

select a.\* from

student a

join score b on a.s\_id=b.s\_id where b.c\_id in(

select c\_id from course where t\_id =(

select t\_id from teacher where t\_name = '张三'));

-- 8、查询没学过"张三"老师授课的同学的信息

select \* from

student c

where c.s\_id not in(

select a.s\_id from student a join score b on a.s\_id=b.s\_id where b.c\_id in(

select c\_id from course where t\_id =(

select t\_id from teacher where t\_name = '张三')));

-- 9、查询学过编号为"01"并且也学过编号为"02"的课程的同学的信息

select a.\* from

student a,score b,score c

where a.s\_id = b.s\_id and a.s\_id = c.s\_id and b.c\_id='01' and c.c\_id='02';

-- 10、查询学过编号为"01"但是没有学过编号为"02"的课程的同学的信息

select a.\* from

student a

where a.s\_id in (select s\_id from score where c\_id='01' ) and a.s\_id not in(select s\_id from score where c\_id='02')

-- 11、查询没有学全所有课程的同学的信息

select s.\* from

student s where s.s\_id in(

select s\_id from score where s\_id not in(

select a.s\_id from score a

join score b on a.s\_id = b.s\_id and b.c\_id='02'

join score c on a.s\_id = c.s\_id and c.c\_id='03'

where a.c\_id='01'))

-- 12、查询至少有一门课与学号为"01"的同学所学相同的同学的信息

select \* from student where s\_id in(

select distinct a.s\_id from score a where a.c\_id in(select a.c\_id from score a where a.s\_id='01')

);

-- 13、查询和"01"号的同学学习的课程完全相同的其他同学的信息

select a.\* from student a where a.s\_id in(

select distinct s\_id from score where s\_id!='01' and c\_id in(select c\_id from score where s\_id='01')

group by s\_id

having count(1)=(select count(1) from score where s\_id='01'));

-- 14、查询没学过"张三"老师讲授的任一门课程的学生姓名

select a.s\_name from student a where a.s\_id not in (

select s\_id from score where c\_id =

(select c\_id from course where t\_id =(

select t\_id from teacher where t\_name = '张三'))

group by s\_id);

-- 15、查询两门及其以上不及格课程的同学的学号，姓名及其平均成绩

select a.s\_id,a.s\_name,ROUND(AVG(b.s\_score)) from

student a

left join score b on a.s\_id = b.s\_id

where a.s\_id in(

select s\_id from score where s\_score<60 GROUP BY s\_id having count(1)>=2)

GROUP BY a.s\_id,a.s\_name

-- 16、检索"01"课程分数小于60，按分数降序排列的学生信息

select a.\*,b.c\_id,b.s\_score from

student a,score b

where a.s\_id = b.s\_id and b.c\_id='01' and b.s\_score<60 ORDER BY b.s\_score DESC;

-- 17、按平均成绩从高到低显示所有学生的所有课程的成绩以及平均成绩

select a.s\_id,(select s\_score from score where s\_id=a.s\_id and c\_id='01') as 语文,

(select s\_score from score where s\_id=a.s\_id and c\_id='02') as 数学,

(select s\_score from score where s\_id=a.s\_id and c\_id='03') as 英语,

round(avg(s\_score),2) as 平均分 from score a GROUP BY a.s\_id ORDER BY 平均分 DESC;

-- 18.查询各科成绩最高分、最低分和平均分：以如下形式显示：课程ID，课程name，最高分，最低分，平均分，及格率，中等率，优良率，优秀率

--及格为>=60，中等为：70-80，优良为：80-90，优秀为：>=90

select a.c\_id,b.c\_name,MAX(s\_score),MIN(s\_score),ROUND(AVG(s\_score),2),

ROUND(100\*(SUM(case when a.s\_score>=60 then 1 else 0 end)/SUM(case when a.s\_score then 1 else 0 end)),2) as 及格率,

ROUND(100\*(SUM(case when a.s\_score>=70 and a.s\_score<=80 then 1 else 0 end)/SUM(case when a.s\_score then 1 else 0 end)),2) as 中等率,

ROUND(100\*(SUM(case when a.s\_score>=80 and a.s\_score<=90 then 1 else 0 end)/SUM(case when a.s\_score then 1 else 0 end)),2) as 优良率,

ROUND(100\*(SUM(case when a.s\_score>=90 then 1 else 0 end)/SUM(case when a.s\_score then 1 else 0 end)),2) as 优秀率

from score a left join course b on a.c\_id = b.c\_id GROUP BY a.c\_id,b.c\_name

-- 19、按各科成绩进行排序，并显示排名(实现不完全)

-- mysql没有rank函数

select a.s\_id,a.c\_id,

@i:=@i +1 as i保留排名,

@k:=(case when @score=a.s\_score then @k else @i end) as rank不保留排名,

@score:=a.s\_score as score

from (

select s\_id,c\_id,s\_score from score WHERE c\_id='01' GROUP BY s\_id,c\_id,s\_score ORDER BY s\_score DESC

)a,(select @k:=0,@i:=0,@score:=0)s

union

select a.s\_id,a.c\_id,

@i:=@i +1 as i,

@k:=(case when @score=a.s\_score then @k else @i end) as rank,

@score:=a.s\_score as score

from (

select s\_id,c\_id,s\_score from score WHERE c\_id='02' GROUP BY s\_id,c\_id,s\_score ORDER BY s\_score DESC

)a,(select @k:=0,@i:=0,@score:=0)s

union

select a.s\_id,a.c\_id,

@i:=@i +1 as i,

@k:=(case when @score=a.s\_score then @k else @i end) as rank,

@score:=a.s\_score as score

from (

select s\_id,c\_id,s\_score from score WHERE c\_id='03' GROUP BY s\_id,c\_id,s\_score ORDER BY s\_score DESC

)a,(select @k:=0,@i:=0,@score:=0)s

-- 20、查询学生的总成绩并进行排名

select a.s\_id,

@i:=@i+1 as i,

@k:=(case when @score=a.sum\_score then @k else @i end) as rank,

@score:=a.sum\_score as score

from (select s\_id,SUM(s\_score) as sum\_score from score GROUP BY s\_id ORDER BY sum\_score DESC)a,

(select @k:=0,@i:=0,@score:=0)s

-- 21、查询不同老师所教不同课程平均分从高到低显示

select a.t\_id,c.t\_name,a.c\_id,ROUND(avg(s\_score),2) as avg\_score from course a

left join score b on a.c\_id=b.c\_id

left join teacher c on a.t\_id=c.t\_id

GROUP BY a.c\_id,a.t\_id,c.t\_name ORDER BY avg\_score DESC;

-- 22、查询所有课程的成绩第2名到第3名的学生信息及该课程成绩

select d.\*,c.排名,c.s\_score,c.c\_id from (

select a.s\_id,a.s\_score,a.c\_id,@i:=@i+1 as 排名 from score a,(select @i:=0)s where a.c\_id='01'

)c

left join student d on c.s\_id=d.s\_id

where 排名 BETWEEN 2 AND 3

UNION

select d.\*,c.排名,c.s\_score,c.c\_id from (

select a.s\_id,a.s\_score,a.c\_id,@j:=@j+1 as 排名 from score a,(select @j:=0)s where a.c\_id='02'

)c

left join student d on c.s\_id=d.s\_id

where 排名 BETWEEN 2 AND 3

UNION

select d.\*,c.排名,c.s\_score,c.c\_id from (

select a.s\_id,a.s\_score,a.c\_id,@k:=@k+1 as 排名 from score a,(select @k:=0)s where a.c\_id='03'

)c

left join student d on c.s\_id=d.s\_id

where 排名 BETWEEN 2 AND 3;

-- 23、统计各科成绩各分数段人数：课程编号,课程名称,[100-85],[85-70],[70-60],[0-60]及所占百分比

select distinct f.c\_name,a.c\_id,b.`85-100`,b.百分比,c.`70-85`,c.百分比,d.`60-70`,d.百分比,e.`0-60`,e.百分比 from score a

left join (select c\_id,SUM(case when s\_score >85 and s\_score <=100 then 1 else 0 end) as `85-100`,

ROUND(100\*(SUM(case when s\_score >85 and s\_score <=100 then 1 else 0 end)/count(\*)),2) as 百分比

from score GROUP BY c\_id)b on a.c\_id=b.c\_id

left join (select c\_id,SUM(case when s\_score >70 and s\_score <=85 then 1 else 0 end) as `70-85`,

ROUND(100\*(SUM(case when s\_score >70 and s\_score <=85 then 1 else 0 end)/count(\*)),2) as 百分比

from score GROUP BY c\_id)c on a.c\_id=c.c\_id

left join (select c\_id,SUM(case when s\_score >60 and s\_score <=70 then 1 else 0 end) as `60-70`,

ROUND(100\*(SUM(case when s\_score >60 and s\_score <=70 then 1 else 0 end)/count(\*)),2) as 百分比

from score GROUP BY c\_id)d on a.c\_id=d.c\_id

left join (select c\_id,SUM(case when s\_score >=0 and s\_score <=60 then 1 else 0 end) as `0-60`,

ROUND(100\*(SUM(case when s\_score >=0 and s\_score <=60 then 1 else 0 end)/count(\*)),2) as 百分比

from score GROUP BY c\_id)e on a.c\_id=e.c\_id

left join course f on a.c\_id = f.c\_id

-- 24、查询学生平均成绩及其名次

select a.s\_id,

@i:=@i+1 as '不保留空缺排名',

@k:=(case when @avg\_score=a.avg\_s then @k else @i end) as '保留空缺排名',

@avg\_score:=avg\_s as '平均分'

from (select s\_id,ROUND(AVG(s\_score),2) as avg\_s from score GROUP BY s\_id)a,(select @avg\_score:=0,@i:=0,@k:=0)b;

-- 25、查询各科成绩前三名的记录

-- 1.选出b表比a表成绩大的所有组

-- 2.选出比当前id成绩大的 小于三个的

select a.s\_id,a.c\_id,a.s\_score from score a

left join score b on a.c\_id = b.c\_id and a.s\_score<b.s\_score

group by a.s\_id,a.c\_id,a.s\_score HAVING COUNT(b.s\_id)<3

ORDER BY a.c\_id,a.s\_score DESC

-- 26、查询每门课程被选修的学生数

select c\_id,count(s\_id) from score a GROUP BY c\_id

-- 27、查询出只有两门课程的全部学生的学号和姓名

select s\_id,s\_name from student where s\_id in(

select s\_id from score GROUP BY s\_id HAVING COUNT(c\_id)=2);

-- 28、查询男生、女生人数

select s\_sex,COUNT(s\_sex) as 人数 from student GROUP BY s\_sex

-- 29、查询名字中含有"风"字的学生信息

select \* from student where s\_name like '%风%';

-- 30、查询同名同性学生名单，并统计同名人数

select a.s\_name,a.s\_sex,count(\*) from student a JOIN

student b on a.s\_id !=b.s\_id and a.s\_name = b.s\_name and a.s\_sex = b.s\_sex

GROUP BY a.s\_name,a.s\_sex

-- 31、查询1990年出生的学生名单

select s\_name from student where s\_birth like '1990%'

-- 32、查询每门课程的平均成绩，结果按平均成绩降序排列，平均成绩相同时，按课程编号升序排列

select c\_id,ROUND(AVG(s\_score),2) as avg\_score from score GROUP BY c\_id ORDER BY avg\_score DESC,c\_id ASC

-- 33、查询平均成绩大于等于85的所有学生的学号、姓名和平均成绩

select a.s\_id,b.s\_name,ROUND(avg(a.s\_score),2) as avg\_score from score a

left join student b on a.s\_id=b.s\_id GROUP BY s\_id HAVING avg\_score>=85

-- 34、查询课程名称为"数学"，且分数低于60的学生姓名和分数

select a.s\_name,b.s\_score from score b LEFT JOIN student a on a.s\_id=b.s\_id where b.c\_id=(

select c\_id from course where c\_name ='数学') and b.s\_score<60

-- 35、查询所有学生的课程及分数情况；

select a.s\_id,a.s\_name,

SUM(case c.c\_name when '语文' then b.s\_score else 0 end) as '语文',

SUM(case c.c\_name when '数学' then b.s\_score else 0 end) as '数学',

SUM(case c.c\_name when '英语' then b.s\_score else 0 end) as '英语',

SUM(b.s\_score) as '总分'

from student a left join score b on a.s\_id = b.s\_id

left join course c on b.c\_id = c.c\_id

GROUP BY a.s\_id,a.s\_name

-- 36、查询任何一门课程成绩在70分以上的姓名、课程名称和分数；

select a.s\_name,b.c\_name,c.s\_score from course b left join score c on b.c\_id = c.c\_id

left join student a on a.s\_id=c.s\_id where c.s\_score>=70

-- 37、查询不及格的课程

select a.s\_id,a.c\_id,b.c\_name,a.s\_score from score a left join course b on a.c\_id = b.c\_id

where a.s\_score<60

--38、查询课程编号为01且课程成绩在80分以上的学生的学号和姓名；

select a.s\_id,b.s\_name from score a LEFT JOIN student b on a.s\_id = b.s\_id

where a.c\_id = '01' and a.s\_score>80

-- 39、求每门课程的学生人数

select count(\*) from score GROUP BY c\_id;

-- 40、查询选修"张三"老师所授课程的学生中，成绩最高的学生信息及其成绩

-- 查询老师id

select c\_id from course c,teacher d where c.t\_id=d.t\_id and d.t\_name='张三'

-- 查询最高分（可能有相同分数）

select MAX(s\_score) from score where c\_id='02'

-- 查询信息

select a.\*,b.s\_score,b.c\_id,c.c\_name from student a

LEFT JOIN score b on a.s\_id = b.s\_id

LEFT JOIN course c on b.c\_id=c.c\_id

where b.c\_id =(select c\_id from course c,teacher d where c.t\_id=d.t\_id and d.t\_name='张三')

and b.s\_score in (select MAX(s\_score) from score where c\_id='02')

-- 41、查询不同课程成绩相同的学生的学生编号、课程编号、学生成绩

select DISTINCT b.s\_id,b.c\_id,b.s\_score from score a,score b where a.c\_id != b.c\_id and a.s\_score = b.s\_score

-- 42、查询每门功成绩最好的前两名

-- 牛逼的写法

select a.s\_id,a.c\_id,a.s\_score from score a

where (select COUNT(1) from score b where b.c\_id=a.c\_id and b.s\_score>=a.s\_score)<=2 ORDER BY a.c\_id

-- 43、统计每门课程的学生选修人数（超过5人的课程才统计）。要求输出课程号和选修人数，查询结果按人数降序排列，若人数相同，按课程号升序排列

select c\_id,count(\*) as total from score GROUP BY c\_id HAVING total>5 ORDER BY total,c\_id ASC

-- 44、检索至少选修两门课程的学生学号

select s\_id,count(\*) as sel from score GROUP BY s\_id HAVING sel>=2

-- 45、查询选修了全部课程的学生信息

select \* from student where s\_id in(

select s\_id from score GROUP BY s\_id HAVING count(\*)=(select count(\*) from course))

--46、查询各学生的年龄

-- 按照出生日期来算，当前月日 < 出生年月的月日则，年龄减一

select s\_birth,(DATE\_FORMAT(NOW(),'%Y')-DATE\_FORMAT(s\_birth,'%Y') -

(case when DATE\_FORMAT(NOW(),'%m%d')>DATE\_FORMAT(s\_birth,'%m%d') then 0 else 1 end)) as age

from student;

-- 47、查询本周过生日的学生

select \* from student where WEEK(DATE\_FORMAT(NOW(),'%Y%m%d'))=WEEK(s\_birth)

select \* from student where YEARWEEK(s\_birth)=YEARWEEK(DATE\_FORMAT(NOW(),'%Y%m%d'))

select WEEK(DATE\_FORMAT(NOW(),'%Y%m%d'))

-- 48、查询下周过生日的学生

select \* from student where WEEK(DATE\_FORMAT(NOW(),'%Y%m%d'))+1 =WEEK(s\_birth)

-- 49、查询本月过生日的学生

select \* from student where MONTH(DATE\_FORMAT(NOW(),'%Y%m%d')) =MONTH(s\_birth)

-- 50、查询下月过生日的学生

select \* from student where MONTH(DATE\_FORMAT(NOW(),'%Y%m%d'))+1 =MONTH(s\_birth)