**一，什么是主键**

**如果一个列可以唯一区分表中的每个行——这个列就是主键**

**如果多个列可以唯一区分表中的每个行——多个列同时作为主键**

**主键特点**

**①如果主键是一个列——每行都必须有不同于别人的主键值**

**如果主键是多个列——每行都必须有不同于别人的列值的组合（唯一区分表中的每个行）**

**②每个行都必须有一个主键值（不能为null）**

**二，数据库概念**

**DBMS（DataBase Management System数据库管理系统）**

**DBMS——基于共享文件系统的DBMS + 客户机-服务器的DBMS**

**MySQL是 基于客户机-服务器的数据库**

**服务器——负责所有数据访问和数据处理的软件**

**客户机——与用户打交道的软件**

**客户机把增删改查的请求发送给服务器，然后由服务器完成，与数据打交道的只有服务器**

**三，数据库，表的信息获取**

**show databases;——显示出所有数据库的名字**

**use 数据库名；——进入某一个数据库**

**show tables;——显示数据库中所有表的名字**

**describe 表名；/ show columns from 表名；——显示表的所有列的信息**

**show create database 数据库名；——显示创建这个数据库时候的SQL语句**

**show create table 表名；——显示创建这个表时的SQL语句**

**show grants;——显示 用户的安全权限**

**show errors/show warnings；——显示服务器错误或警告信息**

**四，关键字**

**①select——从表中提取我们要的东西，形成一张子表**

**select 列名 from 表——获取该列的所有记录**

**select 列名1，列名2......from 表——获取多个列的所有记录**

**select \*from 表——获取整张表的全部记录**

**!!!!!通配符 \* ，会返回表中的所有列，使用\* 会降低检索和应用程序的性能，最好别用\***

**②distinct关键字——必须放在列名前面，过滤掉相同的行，只保留一行**

**select distinct 列名 from 表；—— 在该列中，列值相同的行会被过滤掉，只会返回列值不同的行**

**同时要注意，distinct应用于它后面的所有列，而不仅只有它的前置列**

**select distinct 列1，列2......from 表；——多个列的列值都相同才会被过滤掉，有一个列不同就不会过滤**

**③限制limit**

**！！MySQL表中的行，从行0——行n**

**select something from 表 limit m；——从行0开始，最多返回m行**

**select something from 表 limit m，n；——从行m开始，最多返回n行**

**五，完全限定名**

**列名可以通过——表明.列名 来限定这个列是哪个表中的**

**表名可以通过——数据库明 来限定这个表是哪个数据库中的**

**当同时使用多张表时，我们不在单纯的使用列名来标识一个列，而是用它的完全限定名 表名.列名来标识一个列**

**六，排序order by**

**select something from 表名 order by 列；——得到的表记录会按照 列进行排序**

**select something from 表名 order by 列1，列2......——得到的表记录先按照 列1进行排序，列1相同的情况下，才会按照列2进行排序........(列1相同就不会管列2了）**

**排列顺序——没有设置排列顺序的话默认为升序——DESC降序，ACS升序**

**select something from 表名 order by 列 DESC；——得到的表记录按照 列进行降序排序**

**!!!!注意，DESC和ACS只会作用于他们前面的 列**

**select something from 表名 order by 列1 ACS，列2 DESC；——得到的记录 先按照列1进行升序排序，如果列1的值相同才会按照列2进行降序排序......（列1相同就不会管列2了）**

**七，过滤数据where字句**

**①where字句概述**

**where字句进行数据过滤——where字句会对表中的每一行记录进行判断，只有为true的行来会返回**

**where字句中的操作符：**

**= 等于**

**<>不等于**

**!=不等于**

**between ——在指定的两个值之间——between a and b;**

**特殊:NULL检查——where 列名 IS　NULL——返回列值为NULL的记录**

**②重要操作符**

**１）and与　　　or或**

**where字句可以包含任意个and和or——注意!and的优先级高于or**

**２）IN操作符——IN表示在某个集合中**

**num in (1,2,3,4,5,6,7)——如果num在in的集合中，返回true，否则返回false**

**!!!!IN的操作速度比or快，并且in的语法更直观，清楚**

**!!!!IN最大的优点——可以包含其他的select语句，能够更动态地建立子句**

**３）NOT操作符——条件取反**

**八，用通配符过滤数据**

**关键字LIKE——指示MySQL，后面跟地搜索模式是通配符匹配**

**①百分号%通配符——匹配任意个任意字符（0个也行）**

**例：**

**SELECT \*FROM products WHERE prod\_name LIKE 'jet%';——返回prod\_name以jet开头的记录**

**②下划线\_通配符——匹配单个字符**

**!!!注意，通配符的查询效率不高，慎重使用**

**十，创建计算字段**

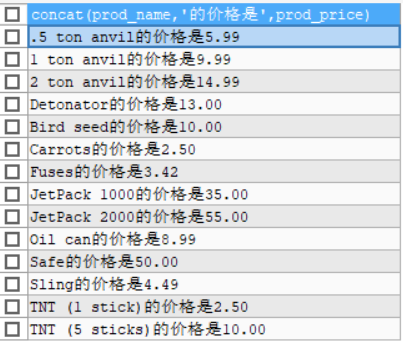
**有时候，数据库中存储的数据不是我们需要的格式，例如数据库中有列：省份，城市，街道，但是我们想要得到地址数据(省份+城市+街道)——那么我们就需要创建计算字段**

**①拼接字段——concat( )函数**

**!!!!多数的DBMS使用+或者||来实现拼接，但是MySQL使用concat( )进行拼接**

**Concat( )拼接后，形成的也是一个列**

**SELECT CONCAT(prod\_name,'的价格是',prod\_price) FROM products;**

****

**②使用别名——As**

**我们可以给列，表起别名**

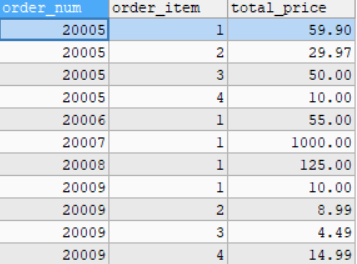
**格式——表名 As 别名 列名 As 别名**

**SELECT CONCAT(prod\_name,'的价格是',prod\_price)AS '商品描述' FROM products;**



**④执行算术运算——MySQL支持算数运算**

SELECT order\_num,order\_item,(quantity\*item\_price )AS 'total\_price' FROM orderitems;



**十一，普通函数**

**1，文本处理函数**

**①Trim( )函数**

**Rtrim( )——去掉列值右边的空格**

**Ltrim( )——去掉列值左边的空格**

**Trim( )——去掉列值两边的空格**

**②upper( )函数——把文本都转换为大写**

**Length()——返回串长度**

**Locate()**

**Left(列名，int m)——返回列值左边的m个字符**

**Right(列名，int m)——返回列值右边的m个字符**

**Soundex()——返回字符串的SOUNDEX值**

**SubString(列名，int begin，int end)——返回列值的字串，同Java的substring**

**Upper()——返回大写串**

**Lower()——返回小写串**

**2，日期和时间处理函数**

**日期——年，月，日**

**时间——小时，分钟，秒**

**！！注意MySQL提供的数据类型有**

**Data——日期（yyyy-mm-dd）**

**Time——时间（hh:mm:ss）**

**Year——年份（yyyy）**

**DateTime——日期时间（yyyy-mm-dd hh:mm:ss）**

**Timestamp——时间戳**

**Now( )——无参数，返回当前日期时间**

**curDate( )——无参数，返回当前日期**

**curTime( )——无参数，返回当前时间**

**Date( 列名)——返回列值所表示的日期时间的 日期部分（日期：年，月，日）**

**Year(列名)——返回列值所表示的日期时间的 日期部分中的 年份部分**

**Month(列名) ——返回列值所表示的日期时间的 日期部分中的 月份部分**

**Day(列名) ——返回列值所表示的日期时间的 日期部分中的 日部分**

**Time(列名)——返回列值所表示的日期时间的 时间部分（时间：小时，分钟，秒）**

**Hour(列名) ——返回列值所表示的日期时间的 时间部分中的 小时部分**

**Minute(列名) ——返回列值所表示的日期时间的 时间部分中的 分钟部分**

**Second(列名)——返回列值所表示的日期时间的 时间部分中的 秒部分**

**DayOfWeek(列名)——返回列值所表示的日期时间 对应星期几**

**AddTime(列名，int s)——列值所表示的时间部分增加**

**S——个位，十位——增加秒；百位千位——增加分；万位，十万位——增加小时**

**AddData(列名，int s)——列值所表示的日期时间增加s天**

**3，数值处理函数**

**Abs( )——返回绝对值**

**Cos( )——返回余弦值**

**Exp( )——返回指数值**

**Mod( )——取余**

**Pi()——返回圆周率**

**Rand( )——返回一个随机数**

**Sin( )——返回正弦**

**Sqrt( )——返回一个数的平方根**

**Tan( )——返回一个数的正切**

**十二，聚集函数——汇总数据**

**AVG( )——返回某列的平均值**

**——要只能用于一个列，要获得多个列的平均值，用多个AVG( )**

**——AVG( )会忽略 列值为NULL的行**

**COUNT( )——返回某列的行数**

**——count(\*)：对表中的行数进行技术，不管列中是否有NULL，都会计算**

**——count(单个列)：对列中具列值的行进行计数，忽略NULL值行  
MAX( )——返回某列的最大值——只作用于单个列，忽略NULL值  
MIN( )——返回某列的最小值——只作用于单个列，忽略NULL值  
SUM( )——返回某列值之和——只作用于单个列忽略NULL值**

**1，对一张表使用聚合函数——只会返回一行记录**

**2，聚合函数，默认是对某列上的所有记录都执行计算**

**但是我们可以通过指定 Distinct参数，让聚合函数只对某列上不同的记录进行计算**

**SELECT COUNT(vend\_id),COUNT(DISTINCT vend\_id)FROM products**

****

**十三，分组（分组，聚合函数，排序经常在一起）**

**1，数据分组**

**①group by 列名**

**根据指定列进行分组，列值相同的行会分为同一组，形成一张子表**

**如果该列有m个不同的列值——分为m组，形成m张子表**

**！！不管我们如何select，每张子表只会返回一行记录**

**②group by 与聚合函数**

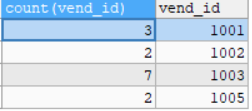
**前面我们说了，对一张表使用聚合函数——只会返回一行记录**

**Group by把一张表划为m张子表，这时如果在select后使用聚合函数**

**那么聚合函数会对这m张子表都执行操作——返回m行记录**

**SELECT COUNT(vend\_id) , vend\_id FROM products GROUP BY vend\_id;**

**首先，表根据vend\_id，划分为4组（4张子表），然后count( )会对每张子表中的行数进行计算，返回4行计数记录**

****

**③group by 列1，列2……——只有 列1的列值相同 and 列2的列值相同 and ……的行才会被划分为同一组**

**SELECT COUNT(vend\_id) FROM products GROUP BY vend\_id , prod\_price;**

**Vend\_id值相同 and prod\_price值相同的 记录会被划为同一组，形成一张子表(12张)**

**然后count(vend\_id)会统计每张子表的行数**

****

**④group by后不能跟 聚集函数**

**⑤group by 列1——如果列1中有多个NULL值，那么这多行NULL值会被划为同一组**

**2，过滤分组——Having**

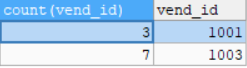
**group by会把初始表 划为多个分组(子表)**

**having会对这些子表进行过滤，过滤掉不满足条件的子表**

**例子1：**

**SELECT COUNT(vend\_id),vend\_id FROM products GROUP BY vend\_id HAVING COUNT(\*)>=3;**

**首先，表根据vend\_id划分为4张子表，然后count(\*)会对每张子表的行数进行计算，并且只会返回 行数>=3的子表（2张子表），然后显示出这两张子表的行数和vend\_id；**

****

**例子2：**

**SELECT COUNT(vend\_id),vend\_id FROM products GROUP BY vend\_id HAVING vend\_id=1003;**

**首先，根据vend\_id把表划分为4张子表，然后返回返回vend\_id=1003的子表(1张)，然后显示出这张子表的行数和vend\_id**

**!!!!where和having的区别（where中能用的操作符，having中也能用）**

**where在分组前进行过滤——对初始表的行记录进行过滤，过滤掉不符合条件的行记录**

**——过滤掉的记录不会参与分组**

**having在分组后进行过滤——对分好的组进行过滤，过滤掉不符合条件的组**

**例子：SELECT COUNT(vend\_id),vend\_id FROM products WHERE prod\_price>=10.00 GROUP BY vend\_id HAVING COUNT(vend\_id)>=3;**

****

**首先，where对初始表的所有记录进行过滤，只会保留prod\_price>=10.00的记录**

**然后，根据vend\_id对过滤后的记录进行分组，**

**然后，统计各个分组的行数，只保留行数>=3的分组**

**最后，现实保留下来的分组的行数和vend\_id**

**3，分组和排序**

**分组经常和排序一起使用，具体看下面**

**大合集1!!!!**

**Select XXX**

**from xx**

**where xxxx**

**group by xxx**

**having xxxx**

**order by xxx**

**limit xxx**

**执行顺序如下**

**①from 选定表**

**②where 根据where对选定的表中的记录进行过滤**

**③group by将过滤后的记录分组**

**④having过滤分组**

**⑤select**

**如果没有分组操作，就是从where过滤后的记录中挑选记录形成子表**

**如果有分组操作，就是从过滤后的分组中挑选所需要的记录形成子表**

**⑥order by 对子表进行排序**

**⑦limit 显示子表的 指定行**

**!!只要进行分组了，那么m张子表，每张子表最多只能显示 一条记录**

**!!聚合函数对于一张表——只能返回一行记录**

**!!分组，聚合函数，排序经常一起使用**

**!!聚合函数，普通函数一般都是出现在select 后，或者having后**

**值得注意的是聚合函数——因为聚合函数很特殊，一张表只会返回一行记录**

**如果聚合函数出现在having后——聚合函数对group by 划分后的分组执行操作**

**如果聚合函数出现在select后 ——聚合函数对having过滤后的分组进行操作**

**在学习子查询之前，我们先来看看一些关键字**

**①Any 任意——常跟>,>=,<,<=,=,!=配合使用**

**Any后跟一个集合**

**a>any(1,2,3)，a只要大于1，2，3中的任意一个就返回true，否则返回false**

**②All 所有——常跟>,>=,<,<=,=,!=配合使用**

**a>all(2,3,4)，a要大于2，3，4中的所有值才会返回true，否则返回false**

**③IN 在集合中**

**A IN(3,4,5)，a的值在(3,4,5)这个集合中就返回true，否则返回false**

**IN 相当于 =Any**

**NOT IN 相当于 <>all**

**十四，子查询**

**子查询——把一个查询嵌套在另一个查询中（多个select）**

**里面的select叫内部查询**

**外面的select叫外部查询**

**外部查询包含内部查询**

**按照子查询的位置分为以下3类**

**①where子查询——子查询的结果作为外查询的比较条件**

**②from子查询——子查询的结果作为临时表，供外查询再次查询(注意，临时表需要一个别名)**

**③select子查询——子查询结果，作为外查询结果的一部分**

**子查询按照返回结果 分为以下4类**

**①子查询结果为 一行一列**

**外部查询可以用>=，>，<=，<，=，!=对子查询的结果进行比较**

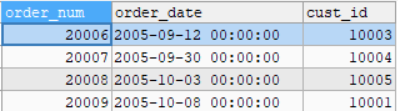
**例子：**

**SELECT \* FROM orders**

**WHERE order\_num >(SELECT order\_num FROM orderitems WHERE prod\_id='ANV01');**

**子查询结果**

**外查询会返回表orders中order\_num>2500的所有记录。如图**

****

**SELECT \* FROM t1 WHERE column1 = (SELECT MAX(column2) FROM t2)**

**②子查询结果为 N行一列**

**外部查询可以使用Any，All，IN对子查询的结果进行比较**

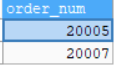
**不能直接使用>,>=,<,<=,=,!=**

**Any，ALL长配合>,>=,<,<=,=,!=一起使用**

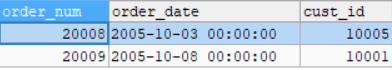
**例子：**

**SELECT \*FROM orders**

**WHERE order\_num > ALL (SELECT order\_num FROM orderitems WHERE prod\_id='TNT2')**

**子查询返回**

**外查询会会返回 表orders中 所有order\_num>20007的记录**

****

**③子查询结果为 一行多列**

**子查询结果为 (列1，列2，列3)**

**那么外查询通过 (列1，列2，列3)跟子查询的结果进行比较**

**SELECT \* FROM article WHERE (title,content,uid) = (SELECT title,content,uid FROM blog WHERE bid=2)**

**④子查询结构为 多行多列**

**SELECT \* FROM article WHERE (title,content,uid) IN (SELECT title,content,uid FROM blog)**

**十五，联结（join）**

**1，主键**

**如果一张表中，一个列可以区分每条记录——这个列就是主键**

**如果一张表中，多个列一起 可以区分每条记录——这多个列就是主键**

**2，外键**

**两张表通过 某个字段或者多个字段建立联系，这个字段是其中一张表的主键，这个字段在另一张表中就是外键**

**例子：我们有两张表，vendors存储供应商信息，products存储产品信息**

**vendors中有主键vend\_id用于区分每个供应商**

**Products中也有vend\_id，不过这个vend\_id在products中就是外键**

**用来建立products与vendors的联系**

**我们对vendors和products使用联结查询时，如果products中某行vend\_id出现了非法值(这个值在vendors中没有出现)，那么联结查询的外联结，和内联结会有不同的处理方法——但是，通常为了避免这种情况的发生，我们会对products中的vend\_id添加外键约束，对vendors中的vend\_id添加主键约束**

**联结分类**

**1）全相乘**

**select A.filed,[ A.field2,……,] B.field, [B.fild4……,]**

**from A，B**

**where <expression>**

**①如果没有where**

**A有m条记录，B有n条记录——那么会返回mn条记录（笛卡尔积表）**

**——A中的每一条记录都会与B中的每一条记录拼接**

**②有where**

**首先，A的m条记录与B的n条记录拼接为一张 含有mn条记录的临时表**

**然后根据where，从临时表中筛选出我们要的记录**

**!!!全相乘使用效率低下，因为全相乘首先会在内存中形成一张临时表（笛卡尔积表）**

**!!!临时表中没有索引，所有全相乘尽量不要使用**

**2）内联结**

**select A.filed, [A.filed2, .... ,] B.filed, [B.filed4...,]**

**from <left table> as A inner jion <right table> as B**

**on <expression>**

**同样是两张表vendors和products，情况跟上面相同**

**Select \* from vendors INNER JION products ON vendors.vend\_id=products.vend\_id;**

**内联结**

**①两张表之间用 INNER JION联结**

**②where 改为 ON**

**③内联结没有主表从表之分，只有两张表中满足条件的行才会拼接返回**

**左表中的一行记录有三种情况**

**①左表中的一行记录，与右表中的一行记录 满足条件——拼接为一条记录返回**

**②左表中的一行记录，与右表中的多行记录 满足条件——拼接为多条记录返回**

**③左表中的一行记录，与右表中的0行记录 满足条件——舍弃该行**

**3）外联结——左外联结**

**select A.filed, [A.filed2, .... ,] B.filed, [B.filed4...,]**

**from <left table> as A left join <right table> as B**

**on <expression>**

**左联结**

**——左表是主表，右表是从表**

**——左表的记录不动，右表根据条件一条条去匹配左表的记录**

**——左表中的一行记录有三种情况**

**①左表中的一行记录匹配右表中的一行记录——拼接为一条记录返回**

**②左表中的一行记录匹配右表中的多行记录——拼接为多条记录返回**

**③左表中的一行记录匹配右表中的0行记录——拼接为一条记录返回，**

**不过这条记录中右表的列值填充为NULL**

**!!!!左联结使用了索引，查询速度快，消耗内存小**

**4）外联结——右外联结**

**同左外联结——只不过主表变为右表罢了**

**!!!!!!!!!!!!包含了联结的大合集2（其实跟之前的大合集一样）**

**Select XXX**

**from XXX JOIN XXX ON XXXXX**

**where xxxx**

**group by xxx**

**having xxxx**

**order by xxx**

**limit xxx**

**执行顺序如下**

**①from后如果只有一张表，那么就选定该表（同前一个大合集）**

**from后如果有多张表，根据ON对表进行联结，形成一张新表，选定该新表**

**!!注意该新表 含有多张表的所有字段，后续对字段进行操作时，要使用全限定名，指明是哪张表的哪个字段**

**②where 根据where对选定的表中的记录进行过滤**

**③group by将过滤后的记录分组**

**④having过滤分组**

**⑤select**

**如果没有分组操作，就是从where过滤后的记录中挑选记录形成子表**

**如果有分组操作，就是从过滤后的分组中挑选所需要的记录形成子表**

**⑥order by 对子表进行排序**

**⑦limit 显示子表的 指定行**

**!!!!含有联结的查询操作——不过是把多张表拼成一张新表，然后对这张新表执行查询操**

**十七，组合查询（Union）**

**UNION——求多个查询的并集，不区分这些查询来自于哪一张表**

**Union组合多条查询select语句**

**——①多条select语句之间用 UNION联结**

**——②每条查询语句的列数必须相同**

**列名不一致时，以第一条SQL语句的列名对齐**

**列数据类型不需要完全相同，但必须要兼容**

**——③如果多条select语句返回的结果中有相同的记录，那么UNION会自动删除重复的记录，这是UNION的默认行为，如果你想要保留重复记录——使用UNION ALL**

**——④多条select语句的结果会通过UNION进行合并，如果我们想要对合并后的结果及逆行排序，只需要在最后一条select语句后接上order by**

**十八，全文本搜索**

**MyISAM支持**

**InnoBD不支持**

**要用再看**

**十九，插入数据——Insert**

**1，插入数据**

**①INSERT INTO 表名**

**VALUES（列值1，列值2……）**

**每个列必须提供一个值，如果某个列没有值要使用NULL**

**虽然语法简单，但是并不安全——尽量不要用**

**②INSERT INTO 表名**

**（列名1，列名2……）**

**VALUES (列值1，列值2……)**

**第一个值对应第一个指定的列名，第二个值对应第二个指定的列名……**

**VALUES以指定的次序匹配列名，不一定按各个列在表中的实际位置**

**即使表的结构改变，该语句仍能正常工作**

**2，插入多条数据**

**①INSERT INTO 表名**

**VALUES（列值1，列值2……），（列值1，列值2……），（列值1，列值2……）**

**②INSERT INTO 表名**

**（列名1，列名2……）**

**VALUES (列值1，列值2……) ，（列值1，列值2……），（列值1，列值2……），**

**2，插入检索出的数据**

**INSERT可以将一条select语句的结果插入表中**

**INSERT INTO 表名**

**(列名1，列名2……)**

**SELECT 列名1，列名2……from 表名**

**二十，对表的记录的操作**

**1）更新记录（UPDATE）**

**UPDATE 表名**

**SET 列名1=‘列值1‘，列名2=’列值2‘**

**WHERE 条件**

**①更新特性行——有where，更新符合where条件的行**

**②更新所有行——如果没有where，更新所有行**

**IGNORE关键字**

**如果用UPDATE更新多行，更新时一行或者多行出现错误，那么这次更新操作就会被取消(错误发生前完成的更新也会被复原)，为了使得即使发生错误，也能继续进行更新——使用IGNORE关键字——UPDATE IGNORE**

**2）删除记录（DELETE）**

**DELETE FROM 表名**

**WHERE 条件**

**①删除特定行——有where条件，删除满足条件的行**

**②删除所有行——没有where，删除所有行**

**二十一，对表结构的操作**

**1）创建表CREATE TABLE**

**CREATE TABLE 表名 [ IF NOT EXISTS ]**

**(**

**列名1 列类型1 [ 列附加说明1 ]，**

**列名2 列类型2 [ 列附加说明2 ]，**

**……………………………….**

**[ 附加说明 ]**

**) [ENGINE=XXX ]**

**[ ]——表示可选**

**列附加说明——主键，非空，准许空，自增长等**

**附加说明——主键，外键等**

**ENGINE——说明这场表的存储引擎**

**CREATE TABLE vendors**

**(**

**vend\_id int NOT NULL AUTO\_INCREASE,**

**vend\_name char(50) NOT NULL,**

**vend\_address char(50) NULL DEFAULT 1，**

**PRIMARY KEY(vend\_id)**

**) ENGINE=InnoDB;**

**列附加说明中——NULL ——表示这个列中 列值可以为NULL (列的默认设置)**

**NOT NULL——表示这个列中 列值不可以为NULL**

**AUTO\_INCREASE——表示这个列中的 列值是自增长的**

**DEFAULT 1 ——表示这个列的列值默认为1（如果不设置）**

**附加说明中 ——PRIMARY KEY(vend\_id) ——把vend\_id设置为主键**

**2）修改表结构ALTER TABLE**

**①添加一列**

**ALTER TABLE 表名**

**ADD**

**（列名１ 列类型１ 列附加说明１，**

**列名２　列类型２　列附加说明２，**

**．．．．．．．．．．．．．．．．．．．**

**）**

**②修改已存在的列**

**ALTER TABLE 表名**

**CHANGE 原列名 新列名 要修改成的类型(包含了列类型+列附加说明)**

**③删除一列**

**ALTER TABLE 表名**

**DROP COLUMN 列名**

**④添加外键**

**ALTER TABLE 表A**

**ADD CONSTRAINT 给外键起的名字(一般为 fk\_表A\_表B) FOREIGN KEY(表A中的列)**

**REFERENCES 表B(表B中的列)**

**3）删除表，重命名表**

**①删除表 DROP TABLE**

**DROP TABLE 表名**

**②重命名表 RENAME TABLE**

**RENAME TABLE 原表名 TO　新表名**

**4）主键PRIMARY KEY**

**主键的特性**

**①唯一性——每行都必须有唯一的主键值**

**如果主键使用单个列——每行的该列值必须唯一**

**如果主键使用多个列——每行的这些列的组合必须唯一**

**②非空——主键不可以含有NULL**

**③自增长——被设置为主键的列可以自增长**

**如何设置主键？**

**1），创建表时声明**

**①列附加说明中——PRIMARY KEY**

**CREATE TABLE mytable(**

**id INT PRIMARY KEY**

**);**

**这种方法只能指定 一个列作为主键，不能指定多个列作为主键**

**②附加说明中——PRIMARY KEY（列名1，列名2……）**

**CREATE TABLE mytable1(**

**id INT ,**

**age CHAR(40) ,**

**PRIMARY KEY(id,age)**

**);**

**这种方法可以指定 多个列作为主键**

**2），修改表时，更改某个列为主键**

**ALTER TABLE mytable1**

**CHANGE id id INT PRIMARY KEY;**

**5）自增长AUTO\_INCREASE**

**设为自增长的列每增加一行时会自动增量**

**每个表最多只可以有一个 AUTO\_INCREASE列**

**6）指定默认值DEFAULT**

**如果插入行时，对于某个列(设置为DEFAULT a)没有给出值，那么该列的值会设为a**

**二十二，视图**

**①视图是一张虚拟的表，保存有从一张表or多张表中查询出来的数据，视图中其实并没有存储数据，数据仍然存放在‘基表’**

**②使用视图查询时，仍然是从 基表 中取出数据，所以基表的数据发生变动，视图中的数据也会发生变动**

**同样的，如果我们对视图执行更新操作，那么其实上是对基本中的数据执行更新操作**

**视图的规则和限制**

**①视图可以嵌套，可以利用从其他视图检索来的数据来构造一个属兔**

**②ORDER BY 可以用在视图中，但如何从该视图中该视图中 检索数据SELECT也含有ORDER　BY，那么视图中的ORDER BY被覆盖**

**③视图不能有索引，也不能有触发器or默认值**

**④视图可以和表一起使用**

**视图的操作**

**1）创建视图**

**CREATE/REPLACE [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]**

**VIEW view\_name [(column\_list)]**

**AS select语句**

**CREATE——创建视图**

**REPLACE——替换视图**

**ALGORITHM——可选三个值，UNDEFINED，MERGE，TEMPTABLE，如果不写这个，默认为UNDEFINED**

**MEGRE：不会建立临时表，将 使用视图的语句A 与 定义视图的语句B 合并起来，直接去查询基表**

**TEMPTABLE：创建视图后，视图被存放在临时表中，然后使用视图执行语句**

**UNDEFINED：MySQL将自己选择算法**

**column\_list——视图的列名，默认情况下由select语句检索出的列名作为视图的列名，如果想要给视图定义明确的列名，**

**使用 [(column\_list)]字句**

**2）删除视图**

**DROP VIEW viewname；**

**3）查询视图——跟查询表一个样，但注意！查询语句中的ORDER BY 会覆盖 视图创建语句中的ORDER BY**

**4）更新视图**

**!!!!!!视图是一张虚拟的表，视图并不存储数据，数据仍然存放在 ‘基表’中**

**!!!!!!所以我们如果对视图执行更新操作——其实是对基表进行更新操作**

**通常，视图是可以更新的，但是如果视图中定义了以下操作，就不能对视图进行更新**

**①分组（使用GPOUP BY 和HAVING）**

**②联结**

**③子查询**

**④UNION**

**⑤聚合函数**

**⑥DISTINCT**

**⑦计算列**

**视图的作用**

**1）简化复杂SQL语句的使用，方便重用**

**利用视图简化复杂的联结查询语句，如下**

SELECT customers.cust\_id, customers.cust \_name , orders.order\_num , orders.order\_date

FROM customers INNER JOIN orders ON customers.cust\_id=orders.cust\_id

ORDER BY cust\_id;

**接下来，我们使用视图对其进行简化**

CREATE VIEW customers\_orders AS

SELECT customers.cust\_id,customers.cust\_name,orders.order\_num,orders.order\_date

FROM customers INNER JOIN orders ON customers.cust\_id=orders.cust\_id

ORDER BY cust\_id;

SELECT \*FROM customers\_orders;

**二十三，存储过程**

**存储过程 是可编程的函数，经过编译创建保存在数据库的数据字典中（存储过程先编译，后执行）**

**存储过程是多条SQL语句的集合，可以理解为是SQL语言的函数，里面可以声明变量，条件判断，执行多条SQL语句**

**1）创建存储过程**

**DELIMITER//**

**CREATE PROCEDURE 存储过程名（参数表） CREATE PROCEDURE 存储过程名(参数表)**

**BEGIN BEGIN**

**存储过程体 存储过程体**

**…… ……**

**END； END//**

**DELIMITER；**

**①左边的会报错，因为MySQL使用；作为SQL语句分隔符，MySQL实用程序也使用；作为语句分隔符，会报错**

**右边使用了DELIMITER//告诉 MySQL实用程序，使用//作为新的分隔符，这样就跟SQL语句分隔符区分开了**

**结束时，将实用程序的分隔符恢复**

**②参数表——参数表中可以定义变量**

**格式：OUT/IN/INOUT 变量名 变量类型**

**OUT：存储过程将值or记录 存储到OUT变量中，供调用者使用（只能是变量）**

**IN：调用者向存储过程传入值 （可以是字面量or变量）**

**INOUT：既表示 调用者向存储过程传入值，又表示 存储过程向调用者传出值（只能是变量）**

**③存储过程体（以BEGIN开始，以END结束）**

**存储过程体——SQL语句，IF-THEN—ELSE语句，WHILT-DO语句，声明变量的declare语句，使用参数表**

**参数表——参数表中的IN变量可以直接使用**

**——参数表中的OUT变量，配合关键字INTO使用，select语句将查询结果 INTO OUT变量**

**SQL语句——修改，删除语句：正常使用**

**——查询语句：select语句中，如果有INTO，那么该select语句的查询结果不会显示，而是存储到OUT变量中**

**Select语句中，如果没有INTO，那么该select语句中的查询结果会显示（多条时只显示第一条）**

**声明变量——DECLARE 变量名1 [,变量名2……] 变量类型 [DEFAULT 默认值]**

**变量赋值——SET 变量名=变量值；**

**IF-THEN-ELSE语句—— IF boolean表达式 THEN**

**Do\_something**

**ELSE**

**Do\_something**

**END IF;**

**CASE-WHEN-THEN\_ELSE语句—— CASE 变量or表达式**

**WHEN 值1 THEN**

**Do\_something1；**

**WHEN 值2 THEN**

**Do\_something2；**

**……………………………**

**ELSE**

**Do\_something；**

**END CASE；**

**WHILT-DO语句—— WHILE 条件**

**Do\_something**

**END WHILE；**

**REPEAT语句（do-while）—— REPEAT**

**Do\_something**

**UNTIL 条件**

**END REPEAT**

**标签——①配合LEAVE，ITERATOR使用 ②增加可读性 ——lable\_name：LOOP**

**——标签可以位于BEGIN，REPEAT，WHILE或者LOOP语句前 do\_something**

**END LOOP lable\_name;**

**LEAVE语句（break）——表示离开循环，一般配合标签使用**

1. **outer\_loop: LOOP**
2. **-> SET j=1;**
3. **-> inner\_loop: LOOP**
4. **-> SELECT CONCAT(i," times ",j," is ",i\*j);**
5. **-> SET j=j+1;**
6. **-> IF j>12 THEN**
7. **-> LEAVE inner\_loop;**
8. **-> END IF;**
9. **-> END LOOP inner\_loop;**
10. **-> SET i=i+1;**
11. **-> IF i>12 THEN**
12. **-> LEAVE outer\_loop;**
13. **-> END IF;**
14. **-> END LOOP outer\_loop;**

**在这里，LEAVE inner\_loop——离开label\_name为 inner\_loop的循环**

**LEAVE outer\_loop——离开lable\_name为outer\_loop的循环**

**ITERATOR（continue）语句——使用方法同LEAVE，不过ITERATOR相当于java的continue，LOOP相当于java的break**

**LOOP语句——LOOP：**

**Do\_something；**

**END LOOP；**

**——LOOP循环不需要初始条件，也没有结束条件，一般通过LEAVE来结束**

**2）存储过程体中可以使用的函数**

**①字符串类 注：mysql中处理字符串时，默认第一个字符下标为1，即参数position必须大于等于1**

CHARSET(str) //返回字串字符集  
CONCAT (string2 [,... ]) //连接字串  
INSTR (string ,substring ) //返回substring首次在string中出现的位置,不存在返回0  
LCASE (string2 ) //转换成小写  
LEFT (string2 ,length ) //从string2中的左边起取length个字符  
LENGTH (string ) //string长度  
LOAD\_FILE (file\_name ) //从文件读取内容  
LOCATE (substring , string [,start\_position ] ) 同INSTR,但可指定开始位置  
LPAD (string2 ,length ,pad ) //重复用pad加在string开头,直到字串长度为length  
LTRIM (string2 ) //去除前端空格  
REPEAT (string2 ,count ) //重复count次  
REPLACE (str ,search\_str ,replace\_str ) //在str中用replace\_str替换search\_str  
RPAD (string2 ,length ,pad) //在str后用pad补充,直到长度为length  
RTRIM (string2 ) //去除后端空格  
STRCMP (string1 ,string2 ) //逐字符比较两字串大小,  
SUBSTRING (str , position [,length ]) //从str的position开始,取length个字符,

TRIM([[BOTH|LEADING|TRAILING] [padding] FROM]string2) //去除指定位置的指定字符  
UCASE (string2 ) //转换成大写  
RIGHT(string2,length) //取string2最后length个字符  
SPACE(count) //生成count个空格

**②数学类**

ABS (number2 ) //绝对值  
BIN (decimal\_number ) //十进制转二进制  
CEILING (number2 ) //向上取整  
CONV(number2,from\_base,to\_base) //进制转换  
FLOOR (number2 ) //向下取整  
FORMAT (number,decimal\_places ) //保留小数位数  
HEX (DecimalNumber ) //转十六进制  
注：HEX()中可传入字符串，则返回其ASC-11码，如HEX('DEF')返回4142143  
也可以传入十进制整数，返回其十六进制编码，如HEX(25)返回19  
LEAST (number , number2 [,..]) //求最小值  
MOD (numerator ,denominator ) //求余  
POWER (number ,power ) //求指数  
RAND([seed]) //随机数  
ROUND (number [,decimals ]) //四舍五入,decimals为小数位数]

SIGN (number2 ) // 正数返回1，负数返回-1

**③日期时间类**

ADDTIME (date2 ,time\_interval ) //将time\_interval加到date2  
CONVERT\_TZ (datetime2 ,fromTZ ,toTZ ) //转换时区  
CURRENT\_DATE ( ) //当前日期  
CURRENT\_TIME ( ) //当前时间  
CURRENT\_TIMESTAMP ( ) //当前时间戳  
DATE (datetime ) //返回datetime的日期部分  
DATE\_ADD (date2 , INTERVAL d\_value d\_type ) //在date2中加上日期或时间  
DATE\_FORMAT (datetime ,FormatCodes ) //使用formatcodes格式显示datetime  
DATE\_SUB (date2 , INTERVAL d\_value d\_type ) //在date2上减去一个时间  
DATEDIFF (date1 ,date2 ) //两个日期差  
DAY (date ) //返回日期的天  
DAYNAME (date ) //英文星期  
DAYOFWEEK (date ) //星期(1-7) ,1为星期天  
DAYOFYEAR (date ) //一年中的第几天  
EXTRACT (interval\_name FROM date ) //从date中提取日期的指定部分  
MAKEDATE (year ,day ) //给出年及年中的第几天,生成日期串  
MAKETIME (hour ,minute ,second ) //生成时间串  
MONTHNAME (date ) //英文月份名  
NOW ( ) //当前时间  
SEC\_TO\_TIME (seconds ) //秒数转成时间  
STR\_TO\_DATE (string ,format ) //字串转成时间,以format格式显示  
TIMEDIFF (datetime1 ,datetime2 ) //两个时间差  
TIME\_TO\_SEC (time ) //时间转秒数]  
WEEK (date\_time [,start\_of\_week ]) //第几周  
YEAR (datetime ) //年份  
DAYOFMONTH(datetime) //月的第几天  
HOUR(datetime) //小时  
LAST\_DAY(date) //date的月的最后日期  
MICROSECOND(datetime) //微秒  
MONTH(datetime) //月  
MINUTE(datetime) //分返回符号,正负或0  
SQRT(number2) //开平方

**3)执行存储过程CALL**

**CALL 存储过程名(参数1，参数2，……)**

**IN类型的参数：调用者传递给存储过程，供存储过程使用，一般直接传递一个值就行（25000，‘hxq’）**

**OUT类型的参数：存储过程将值or记录存储到 OUT类型参数中，供调用者使用，变量名以@开头 (@avg，@total)**

**执行完CALL后，调用者可以直接使用 OUT类型参数（select @total）**

***DELIMITER//***

***CREATE PROCEDURE fun(OUT sum\_num DECIMAL(8,2),OUT avg\_num DECIMAL(8,2),OUT max\_num DECIMAL(8,2),IN id INT )***

***BEGIN***

***SELECT SUM(prod\_price) FROM products WHERE vend\_id=id INTO sum\_num;***

***SELECT AVG(prod\_price) FROM products WHERE vend\_id=id INTO avg\_num;***

***SELECT MAX(prod\_price) FROM products WHERE vend\_id=id INTO max\_num;***

***END//***

***DELIMITER;***

***CALL fun(@sum,@avg,@max,1001);***

***SELECT @sum;***

***SELECT @avg;***

***SELECT @max;***

**4)删除存储过程**

**DROP PROCEDURE [ IF EXISTS ] 存储过程名；**

**5）显示存储过程的创建语句**

**SHOW CREATE PROCEDURE 存储过程名；**

二十四，游标

1）创建游标

CREATE 游标名 CURSOR

FOR select语句；

2）打开游标

OPEN 游标名；

3）关闭游标

CLOSE 游标名；

4）使用游标

游标open后，可以使用FETCH