#### 14.2.2 关系数据库基本概念和MySQL基本命令

数据库（DataBase）仅仅是存放用户数据的地方，当用户访问、操作数据库中的数据时，需要数据库管理系统的帮助。数据管理系统的全称是DataBase Management System，简称DBMS。通常情况下我们会把数据库和数据库管理系统笼统的称为数据库，通常所说的数据库既包括存储用户数据的部分，也包括管理数据库的管理系统。

DBMS是所有数据的存储单元，它负责管理数据的存储、安全、一致性、并发、恢复和访问操作。DBMS有一个数据字典（即：系统表），用于存储它拥有的每个事物的相关信息，例如：名字、结构、位置和类型，这种关于数据的数据也被称为元数据。

数据库按发展顺序，分为：网状数据库、层次数据库、关系数据库、面向对象数据库。其中关系数据库是理论最成熟、应用最广泛的数据库。在大量数据的查找、排序操作上非常成熟且快速，并对数据库的并发、隔离有非常完善的解决方案。

对于关系数据库而言，最基本的数据存储单元就是数据表，我们可以简单的把数据库理解为大量数据表的集合。

数据表示存储数据的逻辑单元，数据表可以理解为表格，其中每一行称为一条记录，每一列称为一个字段。为数据库建表时，通常需要指定该表包含多少列，每列的数据类型信息，不需要指定数据表包含多少行，因为数据库表的行是动态改变的。每行用于保存一条用户数据，除此之外还需要为每个数据表指定一个特殊列，该特殊列的值可以唯一的确定一条记录，则该特殊列被称为主键列。

MySQL数据库的一个实例可以同时包含多个数据库，MySQL使用以下命令来查看当前实例下包含多少个数据库：

命令：show database;

删除指定的数据库

命令：drop database 数据库名;

创建数据命令

命令：create database 数据库名;

MySQL数据库通常支持如下两种存储机制。

* MyISAM：这是MySQL早期默认的存储机制，对事物支持不够好。
* InnoDB：InnoDB提供事务安全的存储机制。InnoDB通过建立行级锁来保证事务完整性并以Oracle风格的共享锁来处理Select语句。系统默认启动InnoDB存储机制。对于MySQL数据5.0以上的版本，通常无须指定数据表的存储机制，因为系统默认使用InnoDB存储机制。

#### 14.2.3 SQL语句基础

SQL全称是Structured Query Language，即结构化查询语言。SQL是操作和检索关系数据库的标准语言，标准的SQL语言可以用于操作任何关系数据库。

使用SQL语句，可以完成如下任务

* 在数据库中检索信息
* 对数据库的信息进行更新
* 改变数据库的结构
* 更改系统的安全设置
* 增加用户对数据库或表的许可权限。

标准的SQL语句通常可分为如下几种类型

* 查询语句：主要由select关键字完成，查询语句是SQL语句中最复杂、功能最丰富的语句。
* DML（Data Manipulation Language，数据操作语言）语句：主要由insert（插入内容）、update（修改内容）和delete（删除内容）三个关键字完成。
* DDL（Data Definition Language，数据定义语言）语句：主要由create（创建表）、alter（修改表）、drop（删除表）和truncate（删除该表里的全部数据，但是保留表的结构，truncate的速度要比delete速度块的多。）四个关键字完成。
* DCL（Data Control Language，数据控制语言，DCL 语句主要是DBA用来管理系统中的对象权限时所使用）语句：主要由grant（对用户授权）和revoke（取回权限）两个关键字来完成。
* TCL事物控制语句：主要由commit、rollback、和savepoint三个关键字完成。

SQL语句的关键字不区分大小写，即：select和SELECT的作用完全一样。在上面5种SQL语句中，DCL语句用于为数据库用户授权，或者回收指定用户的权限，通常无须程序人员操作，所以本章不介绍任何关于DCL的知识。

在SQL命令中也可能需要使用标识符，标识符可以用于定义表名、列名，也可用于定义变量等。这些标识符的命名规则如下：

* 标识符通常必须以字母开头
* 标识符包括字母、数字和三个特殊字符（# \_ $）
* 不要使用当前数据库系统的关键字、保留字，通常建议使用多个单词连缀而成，单词之间以\_分隔。

#### 14.2.4 DDL语句

DDL语句是操作数据库对象的语句，包括创建（create）、删除（drop）和修改（alter）数据库对象。

前面已经介绍过，最基本的数据库对象是数据表，数据表示存储数据的逻辑单元。但数据库里绝不仅仅是包含数据表，也包含如下几种常见的数据库对象：

常见的数据库对象：

|  |  |  |
| --- | --- | --- |
| 对象名称 | 对应关键字 | 描述 |
| 表 | Table | 表是存储数据的逻辑单元，以行和列的形式存在：列就是字段，行就是记录 |
| 约束 | Constraint | 执行数据校验的规则，用于保证数据完整性的规则 |
| 数据字典 |  | 就是系统表存放数据库相关信息的表，系统表里的数据通常由数据库系统维护，程序员不应该手动修改系统表及内部数据，程序员只可查看系统表的数据 |
| 视图 | View | 一个或者多个数据表里数据的逻辑显示，视图并不存储数据 |
| 索引 | Index | 用于提高查询性能，相当于书的目录 |
| 函数 | Function | 用于完成一次特定的计算，具有返回值 |
| 存储过程 | Procedure | 用于完成一次完整的业务处理，没有返回值，但可通过传出参数将多个值传给调用环境 |
| 触发器 | Trigger | 相当于一个事件监听器，当数据库发生特定事件后，触发器被触发，完成相应的处理。 |

因为存在上面几种数据数据库对象，所以在create语句后面可以跟不同的关键字，用于表示要创建哪种对象，例如创建表使用create table，创建索引使用create index，创建视图使用create view，drop和alter后面也需要跟不同的关键字来表示删除、修改哪种数据库对象。

1. **创建表的语法**

create table [模式名.]表名

(

columnName1 dataType [default expr]

)

圆括号里可以包含多个列定义，每个列定义之间以英文逗号隔开，最后一个列定义不需要使用英文逗号，而是直接以圆括号结束。

建立数据表只是建立表结构，就是指定该数据表有多少列，每列的数据类型。所以建表语句中的重点就是圆括号里的列定义，列定义由列名、列类型和可选的默认值组成，中间使用空格隔开。列定义使用英文逗号“,”隔开。

举例：

create table test

(

test\_id int,

test\_price decimal,

test\_name varchar(255) default ‘xxx’,

test\_desc text, #大文本类型

test\_img blod, #图片

test\_date datetime

)

建表时需要指定每列的数据类型，不同数据库所支持的列类型不同。使用不同数据库时需要查阅不同的数据库的相关文档，MySQL支持如下几种类型。

MySQL支持的列类型

|  |  |
| --- | --- |
| 列类型 | 说明 |
| tinyint/smallint/mediumint  int/bigint | 1字节/2字节/3字节/4字节/8字节整数，又可分为有符号和无符号两种。这些整数类型的区别仅仅是表示的范围不同。 |
| float/double | 单精度、双精度浮点类型 |
| decimal/dec | 精确小数类型，相对于float和double不会产生精度丢失的问题 |
| date | 日期类型，不能保存时间，当把Java中的Date对象保存到date列时，时间部分丢失 |
| time | 时间类型，不能保存日期，当把Java中的Date对象保存到date列时，日期部分丢失 |
| datetime | 日期、时间类型 |
| timestamp | 时间戳类型 |
| year | 年类型，仅仅保存时间的年份 |
| char | 定长字符串类型 |
| varchar | 可变长度字符串类型 |
| binary | 定长二进制字符串类型，它以二进制形式保存字符串 |
| varbinary | 可变长度的二进制字符串类型，它以二进制形式保存字符串 |
| tinyblob/blob/  mediumtblob/longblob | 1字节/2字节/3字节/4字节的二进制大对象，可以用于存储图片，音乐等二进制数据，分别可存储255B/64KB/16MB/4GB大小 |
| tinytext/text  mediumtext/longtext | 1字节/2字节/3字节/4字节的文本对象，可以用于存储超长的字符串，分别可存储255B/64KB/16MB/4GB大小的文本 |
|  | 递增，而且必须是主键 |
|  | 自动补0，比如 int(4) 如输入21，，则显示0021 |

上面是比较常见的建表语句，这种建表语句只是创建一个空表，该表中没有任何数据，如果使用子查询建表语句，则可以在建表的同时插入数据，子查询表语句的语法格式如下：

create table userinfo colum1,colum2

as

select colum1,colum2 from user\_info\_meesage

上面语法中新表的字段列必须与子查询中的字段列数量匹配。创建新表时的字段列可以省略，省略了该字段列，那么新表的字段列与选择结果完全相同，如下所示：

举例：

create table user\_info

as

select \* from user\_info\_mesg;

1. **修改表结构的语法**

修改表结构使用alter table，修改表结构包括增加列定义、修改列定义、删除列、重命名列等操作。增加列定义的语法如下：

alter table

add

(

# 可以有多个列定义

column\_name1 datatype [default expr],

);

上面的语法格式中圆括号部分与建表语法的圆括号部分完全相同，只是此时圆括号里的列定义是追加到已有表的列定义后面。还有一点需要指出，如果只是新增一列，则可以省略圆括号，仅在add后紧跟一个列定义即可。为数据表增加字段的SQL语句如下：

举例1：

alter table user\_info

add user\_remark varchar(255); #如果对表的列的修改只增加一列，可以不使用小括号

如果为表增加两个及以上字段需要使用圆括号，如下所示：

alter table user\_info

add

(

user\_remark varchar(255) default ‘xxxxx’,

user\_address varchar(255)

)

增加字段时需要注意：如果数据表中已有记录，那么在对该表进行增加列时，新增加的列不能指定非空约束（即不能为空内容），因为那些已有的记录在新增列上内容肯定为空，那么将导致新增的约束与已有数据冲突，导致修改失败。

=====================================================================================

修改列名字：

alter table table\_name rename column column\_old to column\_new;

修改列定义的语法如下：

alter table 表名

modify column\_name detatype [default exper] [first | after col\_name]

可选信息first或者after col\_name表示将目标修改到指定位置。

注意：修改列定义每次只能对一个列进行修改

举例1：

alter table user\_info

modify user\_id varhcar(255); #将user\_info表中的字段user\_id修改为varchar类型，长度为255

使用SQL修改数据表中列定义的语法和为数据表只增加一个列定义的语法几乎完全一样，关键是增加列使用add关键字，而修改列定义使用modify关键字，增加新列时，列名在原表中不存在，修改列时，列名在原表中必须存在。

如果数据表里已有数据记录，则修改列定义非常容易失败因为有可能修改的列定义规则与原有的数据记录不符合。如果修改数据列的默认值，则只会对以后的插入操作有作用，对以前已经存在的数据不会有任何影响。

======================================================================================

从数据表中删除列的语法：

alter table 表名

drop column\_name

举例1：

alter table user\_info

drop user\_remark;

从数据表中删除列定义通常总是可以成功，删除列定义时将从每行中删除该列的数据，并释放该列在数据块中占用的空间。所以删除大表中的字段时需要比较长的时间，因为还需要回收空间。

上面介绍的这些增加列，删除列，修改列的语法是标准的SQL语法，对所有的数据库都通用。除此之外，MySQL还提供了两种特殊的语法：重命名表和完全改变列定义。

=======================================================================================

重命名表的语法

alter table 表名

rename to 新表名

举例1：

alter table user\_info

rename to user\_info1;

同样还可以使用change语法修改列定义，change还可以修改列名

语法如下：

alter table 表名

change old\_name new\_name datatype [default expr] [first | after col\_name]

举例1：

alter table user\_info

change user\_id id varchar(255);

========================================================================================

3、删除表的语法

删除表的语法格式如下：

drop table 表名;

举例1：

drop table user\_info; #删除数据库中以有的表user\_info

删除表的后果：

* 表的结构被删除，表对象不再存在
* 表中的所有数据被删除
* 该表相关的索引、约束也被删除

1. truncate表

对于大部分数据库而言，truncate被当成DDL处理，truncate被称为截断某个表——它的作用是删除该表里的全部数据，但保留表结构。相对于DML的delete命令而言，truncate的速度要快的多，而且truncate不像delete可以删除指定的记录，truncate只能一次性删除整个表的全部记录。truncate的语法如下：

truncate 表名

#### 14.2.5 数据库约束

前面创建的数据表仅仅指定了一些列定义，这仅仅是数据表的基本功能。除此之外，所有的关系数据库都支持对数据表适用约束。通过约束可以更好地保证数据表里数据的完整性。约束是在表上强制执行的数据校验规则，约束主要用于保证数据里数据的完整性。除此之外，当表中数据存在互相依赖性时，可以保护相关的数据部被删除。

举例1：数据库表A与B，其中A表的数据记录与B表中的数据记录一一对应，当建立某种约束后，如果要单独删除A表中的某条记录时，会无法删除。因为如果删除后，与之对应的B表中的数据就失去了关联性。所以无法删除，但可以同时删除A表与B表中相对应的数据。

数据库支持的5种完整性约束：

1. NOT NULL：非空约束，指定某列不能为空。
2. UNIQUE：唯一约束，指定某列或者几列组合不能重复。
3. PRIMARY KEY：主键，指定该列的值可以唯一地标识该条记录
4. FOREIGN KEY：外键，指定该行记录从属于主表中的一条记录，主要用于保证参照完整性。
5. CHECK：检查，指定一个布尔表达式，用于指定对应列的值必须满足该表达式。

大部分数据库都支持上面5中约束，但MySQL不支持CHECK约束，虽然MySQL的SQL语句也可以使用CHECK约束，但这个CHECK约束不会有任何作用。

虽然约束的作用只是用于保证数据表里数据的完整性，但约束也是数据库对象，并被存储在系统表中，也拥有自己的名字。根据约束对数据列的限制，约束分为如下两类：

1. 单列约束：每个约束只约束一列。
2. 多列约束：每个约束可以约束多个数据列。

为数据表指定约束有如下两个时机：

1. 创建表的同时为相应的数据列指定约束。
2. 创建表后创建，以修改表的方式来增加约束。

大部分约束都可以采用列级约束语法或者表级约束语法。下面依次介绍5中约束的建立和删除。

* NOT NULL：非空约束，指定某列不能为空。

非空约束采用确保指定列不允许为空，非空约束是比较特殊的约束，它只能作为列级约束使用，只能使用列级语法定义。下面介绍一下在SQL中的null的值，SQL中的null不区分大小写。SQL中的null值有如下特征：

1、所有的数据类型都可以是null，包括int、float、boolean等数据类型，与Java相似的是空不等于null，0也不等于null。

举例1：

create table user\_info

(

user\_id varchar(30) not null,

user\_name varchar(50) null

)

举例2：

**alter table user\_info #增加非空约束**

**modify user\_id varchar(30) not null**

#取消非空约束

alter table user\_info

modify user\_id varchar(30) null

* UNIQUE： 唯一约束，用于保证指定的列或者指定列组合不允许出现重复值。虽然唯一约束的列不可以出现重复值，但是可以出现多个null值。因为在数据库中null不等于null

同一个表内可以建多个唯一约束，唯一约束也可以由多列组合而成。当为某列创建唯一约束时，MySQL会为该列相应地创建唯一索引。如果不给唯一的约束起名，该唯一的约束默认与列名相同。

唯一约束既可以使用列级约束语法建立，也可以使用表级约束语法建立。如果需要为多个列建组合约束，或需要为唯一约束指定约束名，则只能使用表级约束语法建立。

当建立唯一约束时，MySQL在唯一约束所在列或列组合上建立对应的唯一索引。

使用列级约束语法建立唯一约束非常简单，只要简单地在列定义后面增加unique关键字即可。

举例1：

create table unique\_test(

#建立非空约束

test\_id int not null,

test\_name varchar(30) unique

)

如果需要为多列组合建立唯一约束，或者想自行指定约束名，则需要使用表级约束语法。表级约束语法如下：

[constraint 约束名] 约束定义

上面的表级约束语法既可以防止create table语句中与列定义并列，也可以放在alter table语句中使用add关键字来添加约束。SQL语句如下：

#创建表时创建唯一约束，使用表级约束语法建立约束

creat table user\_info

(

user\_id varchar(30) not null,

user\_name varchar(60) ,

user\_pass varchar(60),

# 使用表级约束语法建立唯一约束

unique (user\_name),

# 使用表级约束语法建立唯一约束，同时指定约束名为test\_uk

constraint test\_uk unique (user\_pass)

)

**constraint 关键字用于定义约束名**，上面的建表语句为user\_name，user\_pass分别建立了唯一约束，这就意味着这两列都不能出现重复值。除此之外，我们还可以为这两列组合建立唯一约束，SQL语句如下

# 建立表时创建唯一约束，使用表级约束语法建立约束

create table unique\_test3(

#建立了非空约束，这意味值test\_id不可以为null

test\_id int not null,

test\_name varchar(30),

test\_pass varchar(30),

constraint test\_uk unique(test\_name,test\_pass)

)

除了在建立表时增加唯一约束外，还可以在修改表结构时使用add关键字来增加唯一约束，SQL语句如下：

举例2：

**alter table user\_info**

**add unique(user\_name,user\_pass);**

还可以在修改表时使用modify关键字，为单列采用列级约束语法来增加唯一约束，例如：

举例3：

**alter table user\_info**

**modify user\_name varchar(30) unique**

对于大部分数据库而言，删除约束都是在alter table语句后面使用“drop constraint 约束名”语法来完成的，但MySQL并不使用这种方式，而是使用“drop index 约束名”的方式来删除约束。例如如下SQL语句：

**alter table user\_info**

**drop index test\_uk（或者列名）**

* PRIMARY\_KEY：主键约束

主键约束相当于非空约束和唯一约束的组合，即主键既不能为null也不能出现重复值，如果对多列组合建立主键约束，则多列里包含的每一列都不能为空，并且这些列组合的值不能出现重复。主键列的值可以用于唯一地标识表中的一条记录。

每一个表中最多允许有一个主键，但这个主键约束可以由多个列组合而成，主键是表中能唯一确定一行记录的字段或字段组合。

建立主键约束时既可以使用列级约束语法，也可以使用表级约束语法。如果需要对多个字段建立组合主键约束，则只能使用表级约束语法。使用表级约束语法来建立约束时，可以为该约束指定约束名。但不管用户是否为该主键约束指定约束名，MySQL总是将所有的主键约束命名为PRIMARY。

MySQL允许在建立主键约束时为该约束命名，但这个名字没有任何作用，这是为了保持与标准的SQL的兼容性。大部分数据库都允许自行定义主键约束的名字，而且一旦指定了主键约束名，则该约束名就是用户指定的名字。

当创建主键约束时，MySQL在主键约束所在列或列组合上建立对应的唯一索引。

创建主键约束的语法和创建唯一约束的语法非常相似，一样允许使用列级约束为单独的数据列创建主键，如果需要为多列组合建立主键约束或者需要为约束命名，则应该使用表级约束语法来建立主键约束。与建立唯一约束不同的是，建立主键约束使用 primary key

建表时创建主键约束，使用列级约束语法：

举例1：

create table user\_info

(

user\_id varchar(30) primary key

)

建表时创建主键约束，使用表级约束语法：

create table user\_info(

test\_id int not null,

test\_name varchar(50) not null,

test\_pass varchar(50) null,

# 建立多列组合的主键约束，并指定了约束名，但在MySQL中约束名依然为PRIMARY

**contraint test\_pk primary key(test\_id,test\_name)**

)

如果需要**删除**指定表的主键约束，则在alter table语句后使用drop primary key子句即可。SQL语法如下

**alter table user\_info**

**drop primary key**

如果需要为指定表增加主键约束，除了在建表时增加外，还可以在修改表结构时进行指定，可以通过add关键字或者modify关键字来进行追加，但是**使用modify只能使用列级约束语法指定，使用add只能使用表级约束语法指定**。例如：

举例2：

**alter table user\_info （表级约束）**

**add primary key (user\_id);**

**alter table user\_info （列级约束）**

**modify user\_id varchar(30) primary key;**

很多数据库对主键列都支持一种自增长的特性——如果某个数据列的类型是整型，而且该列作为主键列，则可指定该列具有自增长功能。指定自增长功能通常用于设置逻辑主键列——该列的值没有任何物理意义，仅仅用于标识每行记录。MySQL使用auto\_increment来设置自增长，SQL语句如下：

create table test\_primay(

test\_id int **auto\_crement** primary key

)

一旦指定了某列具有自增长特性，那么插入记录时可以不为该列指定值，该列的值由数据库系统自动生成。

* FOREIGN\_KEY：外键约束

外键约束主要用于保证一个或两个数据表之间的参照完整性，外键是构建于一个表的两个字段或者两个表的两个字段之间的参照关系。外键确保了相关的两个字段的参照关系：子表（从表）外键列的值必须在主表被参照列的值范围之内，或者为空。

当主表的记录被从表记录参照时，主表记录不允许被删除，必须先把从表参照记录的所有记录全部删除后，才可以删除主表的该记录。还有一种方式，删除主表记录时级联删除从表中所有参照该记录的从表记录。

从表外键参照的只能是主表**主键列或者唯一键列**，这样才可以保证从表记录可以准确定位到被参照的主表记录，**同一个表内可以拥有多个外键**。

**建立外键约束时，MySQL也会为该列建立索引**。

外键约束通常用于定义两个实体之间的一对多，一对一的关联关系。对于一对多的关联关系，通常在多的一端增加外键列，例如老师——学生（假设一个老师对应多个学生，但每个学生只有一个老师，这是典型的一对多的关联关系）。为了建立他们之间的关联关系，可以在学生表中增加一个外键列，该列中保存此条学生记录对应老师的主键。对于一对一的关联关系，则可以选择任意一方来增加外键列，增加外键列的表被称为从表，只要为外键列增加唯一约束就可以表示一对一的关联关系了。**对于多对多的关联关系，则需要额外增加一个连接表来记录他们的关联关系。**

建立外键约束同样可以采用列级约束语法和表级约束语法。如果仅对单独的数据列建立外键约束，则使用列级约束即可，如果需要对多列组合创建外键约束，或者需要为外键约束建指定约束名，则必须使用表级约束语法。

采用**列级**约束语法建立外键约束直接使用references关键字，**references指定该列参照哪个主表**，以及参照主表的哪一列，SQL语句如下：

举例1：

# 为了保证从表参照主表存在，通常应该先建立主表

create table teacher\_info(

tea\_id varchar(30) primary key,

tea\_name varchar(50) not null

)

create table student\_info

(

stu\_id varchar(30) primary key,

stu\_name varchar(50) not null,

# 指定java\_teacher参照teacher\_info表的tea\_id列

**teacher\_id varchar(30) references teacher\_info(tea\_id)**

)

需要指出的是，虽然MySQL支持使用列级约束语法来建立外键约束，但这种列级约束语法建立的外键约束不会生效，MySQL提供这种列级约束语法仅仅是为了和标准的SQL保持良好的兼容性。因为，**如果要使MySQL中的外键约束生效，则应该使用表级约束。**

create table student\_info(

stu\_id varchar(30) primary key,

stu\_name varchar(50) not null,

teacher\_id varchar(30) not null,

**foreign key (teacher\_id) references teacher\_info(tea\_id)**

)

如果使用**表级**约束语法，则需要使用**foreign key**来指定表的外键列，并使用**references**来指定参照哪个主表，以及参照到主表的哪个数据列。使用表级约束语法可以为外键约束指定约束名称，如果创建外键约束时没有指定约束名，则MySQL会为该外键约束命名为table\_name\_ibfk\_n（teacher\_id\_ibfk\_1），其中table\_name是从表的名称，而n是从1开始的整数。

如果需要显示的指定外键约束名称，则可以使用**constraint来指定约束名字**，如下SQL所示：

create table student\_info(

stu\_id varchar(30) primary key,

stu\_name varchar(50) not null,

teacher\_id varchar(30) not null,

**constraint stud\_info\_teacher\_fk foreign key (teacher\_id) references teacher\_info(tea\_id)**

)

如果需要建立多列组合的外键约束，则必须使用表级约束语法。如下SQL所示：

create table student\_info(

stu\_id varchar(30) primary key,

stu\_name varchar(50) not null,

teacher\_id varchar(30) not null,

teacher\_name varchar(30) not null,

**foreign key (teacher\_id,teacher\_name) references teacher\_info(tea\_id,tea\_name)**

)

**删除外键约束名**的语法很简单，在alter table后增加“drop foreign key 约束名”子句即可，如下SQL所示：

**alter table student\_info**

**drop foreign key stud\_info\_teacher**

增加外键约束除了在创建表时指定外，还可以在**更改表结构时使用add foreign key命令，**如下所示：

**alter table student\_info**

**add foreign key(teacher\_id,teacher\_name) references teacher\_info(tea\_id,tea\_name)**

需要指出的是，外键约束不仅可以参照其他表，而且还可以参照自身，这种参照自身的情况通常被称为**自关联**。例如，使用一个表保存某个公司的所有员工记录，员工之间有部门经理和普通员工的区分，部门经理和普通员工之间就存在了一对多的关系，但他们都是保存在同一个数据表里的记录，这就是典型的自关联，如下SQL所示：

#使用表级约束语法建立外键约束，且直接参照自身

create table foreign\_test(

foreign\_id int auto\_increment primary key,

foreign\_name varchar(50) not null,

refer\_id int,

**foreign key(refer\_id) references foreign\_test(foreign\_id) #自关联**

)

如果想定义**当删除主表记录时，从表记录也会随之删除**，则需要在外键约束后添加o**n delete cascade(级联删除) 或添加on delete set null，**第一种删除主表记录时，把参照主表记录的从表记录全部级联删除；第二种是指当删除主表记录时，把参照主表记录的外键设为null。如下SQL所示：

# 为了保证从表参照主表存在，通常应该先建立主表

create table teacher(

tea\_id int auto\_increment primary key,

tea\_name varchar(50) not null,

tea\_type varchar(30)

)

create table student(

stud\_id int auto\_increment primary key,

stud\_name varchar(50) not null,

java\_teacher int,

foreign key(java\_teacher) references teacher(tea\_id) on delete cascade #或者on delete set null

)

* CHECK：检查约束

当前版本的MySQL支持建表时指定CHECK约束，但这个CHECK约束不会有任何作用。建立CHECK约束的语法很简单，只要在建表的列定义后面增加check（逻辑表达式）即可。如下SQL所示：

create table check\_test(

check\_id int auto\_increment primary key,

check\_name varchar(30) not null,

check\_num decimal,

check (check\_num>10)

)

#### 14.2.6 索引

索引是存放在模式中的一个数据库对象，虽然索引总是从属于数据表，但它和数据表一样从属于数据库对象。**创建索引的唯一作用就是加速对表的查询**，索引通过使用快速路径访问方法来快速定位数据，从而减少了磁盘的I/O。

索引作为数据库对象，在数据字典中独立存放，但不能独立存在，必须从属于某个表。

创建索引有两种方式：

1. 自动：当在表上定义主键约束、唯一约束和外键约束时，系统会为该数据列自动创建对应的索引。
2. 手动：用户可以通过create index。。。 语句来创建索引。

删除索引的方式也有两种：

1. 自动：数据表被删除时，该表上的索引自动删除
2. 手动：用户可以通过delete index…语句来删除指定数据表上的指定索引。

索引的作用类似于书的目录，所以没有一个表是没有索引。一个表中可以有多个索引列，每个索引都可以用于加速该列的查询速度。因此对于一个表中涉及到查询的列都需要为它建立索引，**可以在一列或者多列上创建索引**。创建索引的语法格式如下：

create index index\_name

on table\_name (column[,column]…);

举例1：

**create index emp\_user\_name**

**on user\_info (user\_name,user\_age);**

MySQL中删除索引需要指定表，采用如下语法格式：

drop index 索引名 on 表名

举例：

**drop emp\_user\_name on user\_info**

#### 14.2.7 视图

视图看上去像一个数据表，但它不是数据表，因为它并不能存储数据。**视图只是一个或多个数据表中数据的逻辑显示**，使用视图有以下几点好处：

1. 可以限制对数据的访问
2. 可以使复杂的查询变得简单
3. 提供了数据的独立性
4. 提供了对相同数据的不同显示。

因为视图只是数据表中数据的逻辑显示，即一个查询结果，所以**创建视图就是建立视图名和查询语句的关联**。创建视图的语法格式如下：

create or replace view 视图名 as subquery查询结果

从上面的语法可以看出，创建、修改视图都可以使用上面的语法。上面语法的含义是，如果该视图不存在，则创建视图；如果指定视图名已经存在，则使用新视图替换原有视图。后面的subquery是一个查询语句，这个查询语句可以非常的复杂。

举例1：

**create or replace view view\_student\_teacher**

**as**

**select s.stu\_name,s.stu\_age,t.teacher\_name from student\_info s, teacher\_info t**

**where s.teacer\_id=t.teacher\_id**

通过建立视图的语法规则可以看出，所谓的视图的本质，其实就是一条被命名好的SQL查询语句。

建立视图后，使用视图与使用数据表就没有什么区别了，但通常只是查询视图数据，不能修改视图里的数据，因为视图本身没有存储任何数据。

为了强制不允许改变视图的数据，MySQL允许在创建视图时使用with check option子句，使用该子句创建的视图不允许修改，如下所示：

**create or replace view view\_test**

**as**

**select tea\_id,tea\_name from teacher**

**with check option**

大部分数据库都使用with check option来强制不允许修改视图的数据，但是Oracle采用with read only来强制不允许修改视图的数据

删除视图使用如下语句：

drop view 视图名

举例：

**drop view view\_test**

#### 14.2.8 DML语句

与DDL（数据库定义语句）操作数据库对象不同，**DML（数据库操作语句）主要操作数据表里的数据**，使用DML可以完成以下三件事。

1. 插入新数据
2. 修改已有数据
3. 删除不需要的数据

DML语句由insert into、update和delete from3个命令组成。

1. insert into 语句

**insert into用于向指定数据表中插入记录**。对于标准的SQL语句而言，每次只能插入一条记录。insert into语句的语法格式如下：

insert into table\_name (column1,column2….) values(value1,value2,….)

举例1：

**insert into user\_info values(‘column1\_value’,’column2\_value’,’column3\_value’);**

执行插入操作时，表名后面可以用括号列出所有需要插入值的列名，而values后用括号列出对应的插入的值。

如果省略了表名后面的括号及括号里的列名列表，默认将为所有的列插入值，则需要为每一列都指定一个值。如果既不想在表名后面列出列名，又不想为所有列指定值，则可以为那些无法确定值的列分配null。如下所示：

teacher表的列为 id int 主键 tea\_name varcher 教师名称

**insert into teacher(tea\_name) values(‘abc’)**

如果不想在表后用括号列出所有的列，则需要为所有的列指定值，如果某些值不确定可以**null**作为值进行分配。

**inert into teacher values(null,’zhangsan’)**

在一些特殊的情况下，我们可以使用带子查询的插入语句，带子查询的插入语句可以一次插入多条记录，如下所示：

**insert into tacher(tea\_name) select tea\_name from another\_table**

如上所示，带子查询的插入语句甚至不要求查询数据的源表和插入数据的目的表是同一个表。它只要求选择出来的数据列和插入目的表的数据列个数相等、数据类型匹配即可。

1. update 语句

update语句用于修改数据表的记录，每次可以修改多条记录，通过使用where子句限定修改哪些记录。where子句是个条件表达式，该条件表达式类似于java编程语言的if语句，只有符合条件的记录才会被更改，没有where子句则意味着where表达式使用为true，即该表的所有记录都会被修改。update的语法格式如下：

update table\_name

set colum1=value1,colum2=value2,….

where …

使用update语句不仅可以一次修改多条记录，也可以一次修改多列。修改多列都是通过在set关键字后面使用column1=value1,column2=value2.。。。来实现的，修改多列的值之间使用英文逗号“,”隔开。

举例1：

update user\_info

set user\_name=’张三’ where user\_id=’S001’

**update user\_info**

**set user\_name=’张三’, user\_age=’23’ where user\_id=’0001’**

1. delete from 语句

**delete from语句用于删除指定表中的记录。使用delete from语句删除时不需要指定列名，因为删除总是整行的删除**。

使用delete from语句可以一次删除多行，删除哪些行采用where子句限定，只删除满足where条件的记录。没有where子句限定表示删除表中所有的记录。

delete from语句的语法格式如下：

delete from table\_name where 条件

举例1：

**delete from user\_info where user\_id=’S001’**

**如果不加条件过滤，则删除整张表的数据**

**delete from user\_info**

#### 14.2.9 单表查询

select语句的功能就是查询数据。select语句也是SQL语句中功能最丰富的语句，select语句不仅可以执行单表查询，而且可以执行多表连接查询，还可以进行子查询，select语句用于从一个或多个数据表中选出特定行、特定列的交集。单表查询的select语句语法格式如下：

select column1,column2… from 数据源 [where conditon]

语法格式中的数据源既可以是数据表也可以是视图等。从上面的语法格式可以看出，select后面的列表用于确定选择哪些列，where条件用于确定选择哪些行，只有满足where条件的记录才会被选择出来；如果没有选择where选择条件，则默认选出所有行。**如果想选出所有列则可以用（\*）表示所有列**。如下SQL所示：

select \* from user\_info

如果增加where条件，则只能选择出符合where条件的记录。如下SQL所示：

select \* from user\_info where user\_name=’张三’

当使用select语句进行查询时，还可以在select语句中使用算术运算符（+ - \* /），从而形成算术表达式。使用算术表达式的规则如下：

1. 对数值型数据列、变量、常量可以使用算术运算符（+ - \* /）创建表达式。
2. 对日期型数据列、变量、常量可以使用部分算术运算符（+ -）创建表达式，两个日期之间可以进行减法运算，日期和数值之间可以进行加法、减法运算。
3. 运算符不仅可以在列和常量、变量之间运算，也可以在两列之间运算。

下面的select语句使用了算术运算符；

举例2：

select tea\_id+5 from teacher #将tea\_id当成变量来使用在原有的基础上+5

select \* from teacher where tea\_id\*3>12 # 查询出teacher表中id的值\*3之后会大于12的所有记录

需要指出的是，select后的不仅可以是数据列，也可以是表达式，还可以是变量、常量等。如下SQL所示：

select 3\*5 ,20 from teacher

SQL语句中算术运算符的优先级与java语言中的运算符优先级完全相同，乘法和除法的优先级高于加法和减法，同级运算的顺序是从左到右，表达式中使用括号可提高优先级。

对于MySQL而言，如果在算术表达式中使用null，将会导致整个算术表达式返回值为null；如果在字符串中连接运算符中出现null，将会导致连接后的结果也为null。如下SQL所示：

select concat(tea\_name,null) from teacher #concat函数的作用是拼接字符串

对于某些数据库而言，如果让字符串与null进行拼接，它会把null当成空字符串处理。

如果不希望直接使用列名作为列标题，则可以为数据列或表达式起一个别名，为数据列或表达式起别名时，**别名**紧跟数据列，中间使用空格隔开或者使用**as**关键字隔开。如下所示：

**select tea\_name as 姓名 from teacher**

如果需要为多列指定别名，则列于列之间使用逗号隔开，列名与别名之间使用空格隔开。

如果在查询记录中出现重复记录，则可以使用**distinct关键字去除重复数据**，distinct紧跟select语句，如下SQL所示：

**select distinct tea\_name from student**

where子句的作用可以控制只选择指定的行，是因为where子句是一个条件表达式，所以可以使用>、>=、<、<=、=和<>等基本的比较运算符。不仅可以比较数值之间的大小，也可以比较字符串、日期之间的大小。除此之外，SQL还支持其他特殊比较运算符。如下表所示：

|  |  |
| --- | --- |
| exper1 **between** exper2 and exper3 | 要求exper1>=exper2并且exper1<=exper3 |
| exper1 **in**(exper2,exper3,exper4) | 要求exper1等于后面括号中的任意一个表达式的值 |
| **like** | 字符串匹配，like后的字符支持**通配符** |
| is null | 要求指定的值等于null |

举例1：

select \* from student where stud\_age between 22 and 29

举例2：

select \* from student where teacher\_id in (select tea\_id from teacher where tea\_age between 22 and 29)

举例3：

select \* from user\_info where user\_name like ‘%丽%’

举例4：

select \* from user\_info where user\_name is null

SQL语句中使用下划线**\_（占位符的作用）和百分号%（模糊匹配的作用）**作为通配符，例如查询学生名称为张姓的学生，SQL如下

**select \* from student where stud\_name like ‘张%’**

而上面示例3则表示，凡是用户名中带有“丽”这个字的用户都会被选出来，比如张丽英、王丽丽等等

**select \* from student where stud\_name like ‘\_\_’**

表示查询学生表中，学生名字为两个的所有学生，在某些特殊的情况下，查询条件里需要使用下划线或百分号，不希望SQL把下划线和百分号当成通配符使用，这就需要使用转义字符，MySQL使用**反斜线(\)作为转义字符**，如下SQL所示：

select \* from student where stud\_name like ‘\\_%’

需要指出的是is null用于判断某些值是否为空，判断是否为空不要用=null来判断，因为SQL中null=null返回的是null。

如果where子句后面有多个条件需要组合，SQL提供了**and和or逻辑运算符来组合2个条件，并提供了not来对逻辑表达式求否**。如下SQL所示：

select \* from student where stud\_age>22 and stud\_name like ‘张%’

select \* from student where stud\_age=22 or stud\_age=25

如果需要查询出学生年龄不为22和25的所有学生记录，SQL如下：

select \* from student whre stud\_age not in(22,25)

在数据库中比较运算符、逻辑运算符的优先级

所有的比较运算符 1

not 2

and 3

or 4

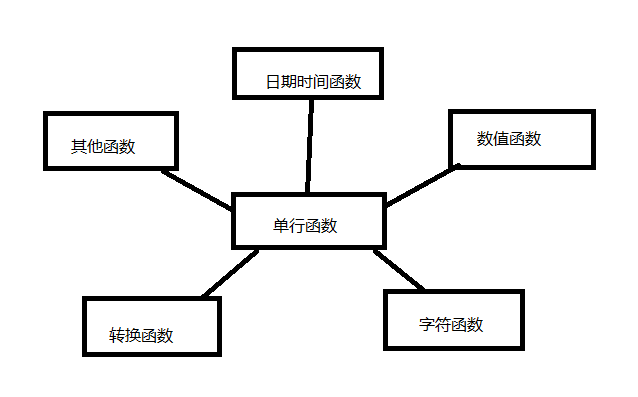
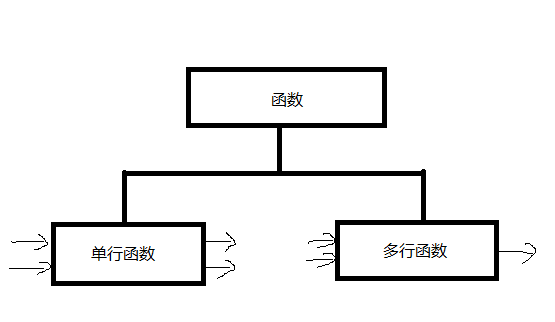
如果需要按多列排序、则每列的**asc(升序)、desc(降序)**必须单独设定。如果指定了多个排序列，则第一个排序列是首要排序列，只有当第一列中存在多个相同值时，第二个排序列才会起作用。如下SQL所示：

**select \* from student order by teacher\_id asc , student\_name desc;**

#### 14.2.10 数据库函数

每个数据库都会在标准的SQL基础上扩展一些函数，例如上面看到的concat，这些函数用于进行数据处理或复杂计算，它们通过对一组数据进行计算，得到最终需要的输出结果。函数一般都会有一个或者多个输入，这些输入被称为函数的参数，函数内部会对这些参数进行判断和计算，最终只有一个值作为返回值。函数可以出现在SQL语句的各个位置，比较常见的位置是select之后和where子句中。

根据函数对多行数据的处理方式，函数被分为单行函数和多行函数，单行函数对每行输入值单独计算，每行得到一个计算结果返回给用户；多行函数对多行输入值整体计算，最后只会得到一个结果。单行函数和多行函数如下图所示：



SQL中的函数和java语言中的方法有点相似，但SQL中的函数是独立的程序单元，也就是说，调用函数时无须使用任何类、对象作为调用者，而是直接使用函数即可。执行函数的语法格式如下：

function\_name（参数1，参数2，参数3.。。。。。。）

多行函数也称为聚集函数、分组函数，主要用于完成一些统计功能，在大部分数据库中基本相同。但不同数据库中的单行函数差别非常大，MySQL中的单行函数具有如下特征：

1. 单行函数的参数可以是变量、常量或者数据列。单行函数可以接收**多个参数**，但只能**返回一个值**。
2. 单行函数会**对每行单独起作用**，每行返回一个结果。
3. 使用单行函数可以改变参数的数据类型。单行函数支持**嵌套使用**，即内层函数的返回值是外层函数的参数。

MySQL数据库的数据类型大致分为数值型、字符型和日期时间型，所以MySQL分别提供了对应的函数。转换函数主要负责完成类型转换其他函数大致分为如下几类：

1. **位函数**
2. **流程控制函数**
3. **加密解密函数**
4. **信息函数**

每个数据库都包含了大量的单行函数，这些函数的用法也存在一些差异，但有一点是相同的，即每个数据库都会为一些常用的计算功能提供相应的函数，这些函数的函数名可能不同。用法可能有差异，但所有数据库提供的函数库所能完成的功能大致相似，可以参考各数据库系统的文档来进行函数的学习。下面介绍MySQL单行函数的用法。

# 选出teacher表中tea\_name列的长度

select char\_length(tea\_name) from teacher

**# 为指定日期添加一定的时间**

**# 使用date\_add函数需要两个参数，日期、数值类型带单位**

**select date\_add(‘1998-01-02’,interval 2 month) ---- day year**

**# 使用adddate函数更简单**

**select adddate(‘1998-01-02’,3); #只能加天数**

**# 获取当前日期**

**select curdate();**

**# 获取当前时间**

**select curtime()**

**# 对字符串使用MD5进行加密**

**select MD5(‘test’)**

MySQL提供了如下几个**处理null**的函数。

1. ifnull(exper1,exper2)：如果exper1为null，则返回exper2，否则返回exper1
2. nullif(exper1,exper2)：如果exper1和exper2相等，则返回null，否则返回exper1
3. if(exper1,exper2,exper3)：类似于？：三目运算符，如果exper1为true，不等于0且不等于null，则返回exper2，否则返回exper3
4. isnull(exper1)：判断exper1是否为null，如果为null返回true，否则返回false。

# 如果stud\_name列为null，则返回没有名字

select ifnull(stud\_name,’没有名字’) from student

使用复杂的查询结果作为参数，参数的SQL语句执行需要加入小括号提高优先级，让SQL提前执行。SQL如下：

select ifnull((select is\_show from student where stud\_id='00003'),

(select is\_show from student where stud\_id='000ST')) as result;

# 如果stud\_name列的值为张三，则返回null

select nullif(stud\_name,’张三’) from student

# 如果stud\_name列为null则返回没有名字，否则返回有名字

select if(isnull(stud\_name),’没有名字’,’有名字’)

MySQL还提供了一个**case函数**，该函数是一个**流程控制函数**。case函数有两个用法，case函数的第一个用法的语法格式如下：

case value

when compare\_value1 then result1

when compare\_value2 then result2

else result

end

case函数用value和后面的compare\_value1、compare\_value2、…依次进行比较，如果value和指定的compare\_value1相等，则返回对应的result，否则返回else后的result。例如下面SQL：

**# 如果java\_teacher为1返回Java老师，为2返回Ruby老师，否则返回其他老师**

**select student\_name,case tea\_name**

**when 1 then ‘Java老师’**

**when 2 then ‘Ruby老师’**

**else ‘其他老师’**

**end**

**from student**

case函数的第二种用法如下所示：

case

when condition1 then result1

when condition2 then result2

……..

else result

end

在第二种用法中，condition1、conditon2都是返回一个boolean值的条件表达式，因此这种用法更加灵活，例如如下SQL：

**select stud\_name,case**

**when stud\_age<=3 then ‘初级班’**

**when stud\_age<=6 then ‘中级班’**

**else ‘高级班’**

**end**

**from student**

虽然此处介绍了一些MySQL常用函数的简单用法，但通常不推荐在Java程序中使用特定数据库的函数，因为将导致程序代码与特定的数据库耦合；如果把程序移植到其他数据库系统上时，可以能需要打开源码，重新修改SQL语句。

#### 14.2.11 分组和组函数

组函数也就是前面提到的多行函数，组函数将一组记录作为整体计算，每组记录返回一个结果，而不是每条记录返回一个结果。常用的组函数有如下5个：

1. **avg([distinct|all] exper)**：计算多行的exper的平均值，其中，exper可以是变量、常量或数据列，但其数据类型必须是数值型。还可以在变量、列前使用distinct或者all关键字，如果是distinct，则表明不计算重复值，all用和不用效果完全一样，表明需要计算重复值。
2. **count(\*|[distinct|all]exper)：**计算多行exper的总条数，其中exper可以是变量、常量或数据列，其数据类型可以是任意类型：用星号表示统计该表内的记录行数；distinct表示不计算重复值。
3. **max(exper)**：计算多行exper的最大值，其中exper可以是变量、常量或数据列，其数据类型可以是任意类型。
4. **min(exper)：**计算多行exper的最小值，其中exper可以是变量、常量或数据列，其数据类型可以是任意类型。
5. **sum([distinct|all]exper)**：计算多行的总和，其中，exper可以是变量、常量或数据列，但其数据类型必须是数值型；distinct表示不计算重复值。

**# 计算student表中的记录条数**

**select count(\*) from student**

**# 计算student表中teacher\_id列有多少个值**

**select count(distinct teacher\_id) from student**

**# 统计所有student的主键的和**

**select sum(stud\_id) from student**

**# 选出student表中stud\_id的最大值**

**select max(stud\_id) from student**

**# 选出student表中stud\_id的最小值**

**select min(stud\_id) from student**

**# 计算student表中所有的分数的总和是多少**

**select sum(stud\_sorce) from student**

**# 计算student表中所有学生的平均分值，如果有的记录该列为null，则用0代替**

**select avg(ifnull(stud\_sorce,0)) from student**

在默认情况下，组函数会把所有记录当成一组，为了对记录进行显示分组，可以在select语句后使用group by子句，**group by**子句后通常跟一个或多个列名，表明查询结果根据一列或多列进行分组，即当一列或多列组合的值相同时，系统会把这些记录当成一组。如下SQL所示：

# count(\*) 将会对每组得到一个结果

**select count(\*) from student group by stud\_name**

如果对多列进行分组，则要求多列的值完全相同才会被当成一组。如下SQL所示：

**select count(\*) from student group by stud\_name,stud\_age**

对于很多数据库而言，分组计算时有严格的规则——如果查询列表中使用了组函数，或者select语句中使用了group by分组子句，则要求出现在select列表中的字段，要么使用组函数包起来，要么必须出现在group by子句中。这条规则很容易理解，因为一旦使用了组函数或者使用了group by子句，都将导致多条记录只有一条被输出，系统无法确定输出哪一条。

对于MySQL来说，并没有上面的规则要求，如果某个数据列既没有出现在group by之后，也没有使用组函数包起来，则MySQL会输出该列的第一条记录的值。

如果需要对分组进行过滤，则应该使用**having子句**，having子句后面也是一个条件表达式，只有满足该条件表达式的分组才会被选出来。having子句和where子句非常容易混淆，它们都有过滤功能，但是它们有区别：

1. 不能在where子句中过滤组，where子句仅用于过滤行，**过滤组必须使用having子句**
2. 不能在where子句中使用组函数，having子句中可以使用组函数。

如下SQL语句所示：

select \* from stud1 group by stud\_cname having count(\*)>2

但是可以把组函数作为查询后的条件在where中使用

select \* from student where id=(select max(id) from student)

#### 14.2.12 多表连接查询

在某些情况下我们所需要的数据并不是来源于一个表，而是来源于多个数据表，这就需要使用多表连接查询。例如，对于学生表和教师表，如果希望查询出所有学生以及他的老师名字，这就需要从两个表中取数据。

多表连接查询有两种规范，较早的SQL 92规范支持如下几种多表连接查询。

* 等值连接
* 非等值连接
* 外连接
* 广义笛卡儿积

SQL 99规范提供了可读性更好的多表连接语法，并提供了更多类型的连接查询。SQL 99支持如下几种多表连接查询：

* 交叉连接
* 自然连接
* 使用using子句的连接
* 使用on子句的连接
* 全外连接或者左、右、全外连接

1. SQL92的连接查询

SQL92的多表连接语法比较简洁，这种语法把多个数据表都放在from之后，多个表之间以逗号隔开；连接条件放在where之后，与查询条件之间用and逻辑运算符连接。如果连接条件要求两列值相等，则成为等值连接，否则成为非等值连接，如果没有任何连接条件，则称为广义卡迪尔积。SQL92中多表连接查询的语法格式如下：

select column1,column2…. from table\_name [where join\_condition]

多表连接查询中可能出现两个或多个数据列具有相同的列名，则需要在这些同名列之间使用表名前缀或表名别名前缀作为限制避免系统混淆。

实际上，所有的列都可以增加表名前缀或表别名前缀，只是进行单表查询时，绝不可能出现同名列，所以系统不能能混淆，因此通常会省略表明前缀。

如下SQL查询所有学生资料以及学生对应的老师的资料

**select s.\*,t.tea\_name from student s,teacher t where s.teacher\_id=t.tea\_id（多表查询）**

如果是查询广义卡迪儿积，则where子句后没有任何连接条件，广义卡迪儿积的结果会是nXm条记录。

select s.\*,t.\* from student s,test t

1. SQL99的连接查询

SQL99的连接查询与SQL92的连接查询原理基本相似，不同的是SQL99连接查询的可读性更强，查询用的多个数据表显示使用XXX join连接，而不是直接一次排列在from之后，from后只需要放一个数据表；连接条件不再放在where之后，而是提供了专门的连接条件子句。

* 交叉连接（cross join）：交叉连接的效果就是SQL92的广义卡笛儿积，所以交叉连接无须任何连接条件，如下SQL所示：

举例1：

select s.\*,teacher\_name from student\_table s cross join teacher\_table t;

* 自然连接（natural join）：自然连接表面上看起来也无须指定连接条件，但自然连接是有连接条件的，自然连接会以两个表中的同名列作为连接条件，如果两个表中没有同名列，则自然连接与交叉连接的效果完全一样。

举例1：

select s.\*,teacher\_name from student\_table s natural join teacher\_table t;

* using子句连接

using子句可以指定一列或多列，用于显式指定两个表中的同名列作为连接条件。假设两个表中有超过一列的同名列，如果使用using子句，就可以显示指定使**用哪些同名列作为连接条件。**

举例1：

select s.\*,teacher\_name from student s join teacher using(teacher\_id)

* on子句连接

最常用的连接方式，SQL99语法的连接条件放在on子句中指定，而且每个on子句只指定一个连接条件。如果需要进行N个表连接，则需要N-1个join…on对。

举例1：

**select s.\*,teacher\_name from sutdent s join teacher on s.teache\_id=teacher\_id;**

select role.\*,stud1.stud\_address,student.stud\_cname

from role

join stud1 on role.role\_id=stud1.id

join student on role.role\_id=student.stud\_id

* 左、右、全外连接

这三种外连接分别使用left join、right join和full join，这三种外连接的连接条件一样通过on子句来指定。既可以是等值连接，也可以是非等值连接。

下面使用右外连接，连接条件是非等值连接

select s.\*,teacher\_name from student s right join teacher on s.teacher\_id<t.teacher\_id

下面使用左外连接，连接条件是非等值连接

select s.\*,teacher\_name from student s left join teacher on s.teacher\_id<teacher\_id;

全外连接，此版本的MySQL不支持了

select s.\*,teacher\_name from student\_table s full join teacher t on s.teacher\_id=teacher\_id;

#### 14.2.13 子查询

子查询就是指在查询语句中嵌套另一个查询语句，子查询可以支持多层嵌套。对于一个普通的查询语句而言，子查询可以出现在两个位置。

1. 出现在from语句后当成数据表，这种用法被称为行内视图，因为该子查询的实质就是一个临时的视图。
2. 出现在where条件后作为过滤条件的值。

使用子查询时需要注意如下几点：

1. 子查询要用小括号括起来
2. 把子查询当成数据表时，可以为该子查询起别名，尤其是作为前缀来限定数据列时，必须给子查询起别名。
3. 把子查询当成过滤条件时，将子查询放在比较运算符的右边，这样可以增加查询的可读性。
4. 把子查询当成过滤条件时，单行子查询使用单行运算符，多行子查询使用多行运算符。

对于把子查询当成数据表来使用，只是把之前的表名变成子查询，其他部分与普通查询没有任何区别。下面SQL语句示范了把子查询当成数据表的用法。

**select \* from (select \* from student) t where t.teacher\_id=2**

把子查询当成数据表的用法更准确地说是当成视图，可以把上面的SQL语句理解成在执行查询时创建了一个临时的视图，该视图名称为t，所以这种临时创建的视图也被称为**行内视图**。理解了这种子查询的实质后，我们就可以明白这种子查询可以完全替代查询语句中的数据表，包括在多表连接查询中使用这种子查询。

还有另外一种情况，把子查询当成where条件中的值，如果子查询返回单行、单列值，则被当成一个标量值使用，也就可以使用单行记录比较运算符。如下SQL所示：

举例1：

select \* from student where teacher\_id=(select teacher\_id from teacher where teacher\_name like ‘%XX%’)

如果子查询返回多个值，则需要使用**in any 和all**关键字，in可以单独使用。

select \* from student where teache\_id in (select teacher\_id from teacher)

#### 14.2.14 集合运算

select查询语句执行后得到一个包含多条数据的查询结果集，类似于数学的集合，可以进行交（intersect）、并（union）和差（minus）运算，select查询得到的结果集也可以进行这三种运算。

为了对两个结果集进行**集合运算**，这两个结果集必须满足以下条件：

* 两个结果集所包含的**数据列的数量必须相等**
* 两个结果集里所包含的数据列的**数据类型也必须一一对应**。

1. union运算：**并集运算**

union运算的语法格式如下：

select 语句 union select 语句，给出的列名以第一个查询为准

举例：

select id,stud\_cname,stud\_pass from stud1

union

select stud\_id,stud\_cname,stud\_pass from student

1. minus运算：**差集运算**

minus运算的语法格式如下：

select 语句 minus select 语句，给出的列名以第一查询为准

举例：

select id,stud\_cname,stud\_pass from stud1

minus

select stud\_id,stud\_cname,stud\_pass from student

可惜MySQL不支持minus减法运算，如果想要实现这种效果，只能通过子查询的方式进行实现：

select id,stud\_cname,stud\_pass from stud1

where (id,stud\_cname,stud\_pass) not in (

select stud\_id,stud\_cname,stud\_pass from student)

1. intersect运算：**交集运算**

intersect的语法格式如下：

select 语句 intersect select 语句，给出的列名以第一查询为准

可惜MySQL也不支持intersect减法运算，如果想要实现这种效果，只能通过子查询的方式进行实现：

select id,stud\_cname,stud\_pass from stud1

join student on stud1.id=student.tea\_id and stud1.stud\_cname=student.tea\_cname and stud1.stud\_pass=student.stud\_pass

#### 14.2.15 存储过程

使用的大多数SQL语句都是针对一个或多个表的单条语句。并非所有操作都这么简单，经常会有一个完整的操作需要多条语句才能完成。例如，考虑以下的情形。

1. 为了处理订单，需要核对以保证库存中有相应的物品。
2. 如果库存有物品，这些物品需要预定以便不将它们再卖给别的人， 并且要减少可用的物品数量以反映正确的库存量。
3. 库存中没有的物品需要订购，这需要与供应商进行某种交互。
4. 关于哪些物品入库（并且可以立即发货）和哪些物品退订，需要通知相应的客户。

这显然不是一个完整的例子，它甚至超出了本书中所用样例表的范围，但足以帮助表达我们的意思了。执行这个处理需要针对许多表的多条MySQL语句。此外，需要执行的具体语句及其次序也不是固定的，它们可能会根据哪些物品在库存中商品哪些不在而变化。

那么，怎样编写此代码？可以单独编写每条语句，并根据结果有条件地执行另外的语句。在每次需要这个处理时（以及每个需要它的应用中）都必须做这些工作。

可以创建存储过程，存储过程简单来说，就是为以后的使用而保存的一条或多条MySQL语句的集合。可将其视为批处理文件，虽然它们的作用不仅限于批处理。

**创建存储过程**的语法格式如下：

create procedure 存储过程名(pram[参数列表])

begin

sql语句

end;

举例：

**drop procedure if exists query\_sub\_action;**

**delimiter //**

**create procedure query\_sub\_action(in username varchar(50),in pp\_code varchar(30), in cout int)**

**begin**

**set @total = (select p\_store from product where p\_code=pp\_code);**

**update product set p\_store=(@total-cout) where p\_code=pp\_code;**

**insert into userlist values(null,pp\_code,username,cout);**

**end //**

解释说明：如语法格式所写，常规的存储过程的语法是没有开始的delimiter // 和结尾的//的这个双斜线的，因为如果按照正常的方式去创建存储过程是出错的，因为使用常规的创建存储过程方法，发现各种报错，无法执行，在SQL SERVER下很容易成功的写法，在MySQL下居然不行，因为在MySQL中需要设置DELIMITER，简单解释下这个命令的用途，在MySQL中每行命令都是用“；”结尾，回车后自动执行，在存储过程中“；”往往不代表指令结束，马上运行，而DELIMITER原本就是“；”的意思，因此用这个命令转换一下“；”为“//”，这样只有收到“//”才认为指令结束可以执行。

**执行存储过程**的语法格式如下：

**call query\_sub\_action('zhangsan','CD003',10);**

**调用带有返回值的存储过程：**

**call test\_1(@aa);**

**select @aa;**

**删除存储过程**的语法格式如下：

**drop procedure 存储过程名;**

* 在创建存储过程中使用if then 进行分支处理 最后需要使用end if来结束分支SQL语句。

存储过程中**声明变量使用关键字declare**，**判断使用if then end if**语法格式如下： if then else end if

drop procedure if exists test\_1;

delimiter //

create procedure test\_1(in a int,in b int,out c int)

begin

if isnull(a) then

set a=1;

end if;

if isnull(b) then

set b=2;

else

set b = b + 2;

end if;

set c=(a+b);

end //

调用有返回值的存储过程，SQL如下：

call test\_1(5,7,@c);

select @c as num;

举例2：对学生表中指定的记录操作，如果学生年龄age为偶数则更新age，让此值除2，如果为奇数则让age字段的值乘2。

drop procedure if exists test\_5;

delimiter //

create procedure test\_5(in id int)

begin

**declare age int;---声明变量**

set age=(select stud\_age from student where stud\_id=id);

if (age%2)=0 then

update student set stud\_age=(age/2) where stud\_id=id;

else

update student set stud\_age=(age\*2) where stud\_id=id;

end if;

end;

如果在创建存储过程中，MySQL可以不使用declare来声明一个变量，直接可以使用set关键来声明一个变量直接使用，此时变量前面可以使用@符号，如下SQL所示：

drop procedure if exists test\_1;

delimiter //

create procedure test\_1(in a int,in b int)

begin

if isnull(a) then

set a=1;

end if;

if isnull(b) then

set b=2;

end if;

**set @c=(a+b); -- 直接使用变量**

select @c as num;

end //

在存储过程中使用**case when**语句，类似于java中switch case语句，如下SQL所示：

drop procedure if exists test\_2;

delimiter //

create procedure test\_2(in val int)

begin

case val

when 0 then

insert into test values(1,'aaa');

when 1 then

insert into test values(2,'aaa');

else

insert into test values(3,'aaa');

end case;

end ;

在存储过程中使用**while do end while进行循环操作：**如下SQL所示：

drop procedure if exists test\_3;

delimiter //

create procedure test\_3(in val int)

begin

**while val>0 do**

**insert into test values(val,'aaa');**

**set val = val - 1;**

**end while;**

end;

在存储过程中使用**repeat until end repeat进行循环操作**，类似于java中do while循环，不管条件是否符合，首先进行一次循环体的操作，SQL语句如下：

drop procedure if exists test\_4;

delimiter //

create procedure test\_4(in val int)

begin

**repeat**

**insert into test values(val,'aaa');**

**set val = val + 1;**

**until val>5 end repeat;**

end;

#### 14.2.16 游标

从mysql V5.5开始，进行了一次大的改变，就是将InnoDB作为默认的存储引擎。InnoDB支持事务，而且拥有相关的RDBMS特性：ACID事务支持，数据完整性（支持外键），灾难恢复能力等等。

现在简单总结一下游标的知识。

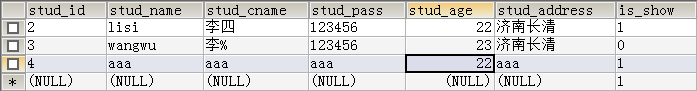
游标实际上是一种能从包括多条数据记录的结果集中每次提取一条记录的机制。游标充当指针的作用。尽管游标能遍历结果集中的所有行，但是它一次只能指向一行。

概括来讲，SQL的游标是一种临时的数据库对象，既可以用来存放在数据库表中的数据行副本，也可以指向存储在数据库中的数据行的指针。游标提供了在逐行的基础上操作表中数据的方法。

游标的一个常见用途就是保存查询结果，以便以后使用。游标的结果集是由select语句产生，如果处理过程需要重复使用一个记录集，那么创建一次游标而重复使用若干次，比重复查询数据库要快的多。

大部分数据库设计语言都能使用游标来检索SQL数据库中的数据，在程序中嵌入游标和在程序中嵌入SQL语句相同。

创建游标的SQL如下：



如上图所示，创建存储过程，获取记录，如果年龄为偶数则在年龄的基础上加1，并将is\_show展示更改为隐藏，SQL如下：

drop procedure if exists test\_4;

delimiter //

create procedure test\_4()

begin

declare done int default 0;# 声明变量表示循环是否结束

declare s\_age int;

declare s\_show char(1);

**declare cur1 cursor for select stud\_age,is\_show from student; #声明游标**

**declare continue handler for not found set done=1;**

#设置在从游标中取数据时如果没有了设置变量done为1

**open cur1;#打开游标**

**fetch cur1 into s\_age,s\_show;（先手动fetch出一行来，以防止第一次fetch就为空，也防止了最后一行被取出两次的现象）**

while done<>1 do

update student set stud\_age=(s\_age+1),is\_show=0;

**fetch cur1 into s\_age,s\_show;#游标继续向下走**

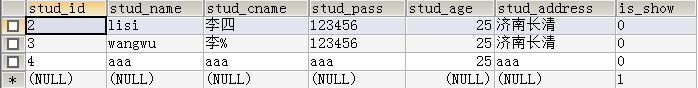
end while;#结束while循环

close cur1;#关闭游标

end;

**（若最后一行被取出两次的解决方案）**

**解决方案：声明处理的hanlder不再是continue，而是exit即可达到目的。即：declare exit handler for NOT FOUND set ergodic:=0;  
 //exit与continue的区别是：exit触发后，后面的语句不再执行，而continue还需要继 续执行。**



#### 14.2.17 触发器

**触发器（trigger）是个特殊的存储过程**，它的执行不是由程序调用，也不是手工启动，而是由**事件来触发**，当对一个表进行增、删、改操作时就会触发，或者更改表结构时也会触发。触发器经常用于加强数据的完整性约束以及业务规则等。综上所述：**触发器是一种与表操作有关的数据库对象，当触发器所在的表上发生指定事件时，将会调用该对象，即表的操作事件触发表上触发器的执行。**

触发器分为三类：**DML触发器，DDL触发器，登录触发器**。我们重点来看一下DML触发器和DDL触发器。DML触发器是指在数据库中发生DML操作时将会触发的事件。DDL触发器是指在数据库中发DDL操作时将会触发的事件。

触发器的作用：

1. 安全性：可以基于数据库的值使用户具有操作数据库的某种权利。可以基于时间限制用户操作，例如不允许节假日或下班时间对数据库操作。可以基于数据库中的数据限制用户的操作，例如不允许股票的价格一次上浮10%。
2. 审计：可以跟踪用户对数据库的操作。例如：审计用户操作数据库的语句；例如：把用户对某些关键表的操作写入到审计表中。
3. 实现复杂的数据完整性规则。例如：实现非标准的数据完整性检查和约束。触发器可产生比规则更为复杂的限制。与规则不同，触发器可以引用列或数据库对象。例如，触发器可回退任何企图吃进超过自己保证金的期货。
4. 实现复杂的非标准的数据库相关完整性规则。触发器可以对数据库相关的表进行连环更新，例如在某个表上的删除触发器可以导致相应的删除与之关联的其他表的数据或更改某列的值。
5. 同步的实时复制表中的数据。
6. 自动计算数据值，如果某个表中的某列字段值达到了一定的要求，则进行特定的处理。

创建触发器的SQL语法如下：

**create trigger trigger\_name trigger\_time trigger\_event on table\_name for each row trigger\_stmt**

触发器是**与表操作相关的数据库对象**，所以触发的命名与表相关，在创建触发器时需要指定表名table\_name，表为**永久性的表**，所以触发器不能与临时表和视图建立关联。

trigger：创建触发器的关键字，定义触发器名称。

trigger\_name：触发器的名称。

trigger\_time：触发器的执行时间，**它的值为before或者after**，是指在激活它的语句之前还是之后执行。一般情况下是在激活它的语句运行完成后再执行触发器。

trigger\_event：触发事件，它的值可以使insert、update、delete。在这个三个动作上我们需要了解数据库里两个临时的虚拟表：deleted、inserted，在mysql中使用old和new关键字来表示。

|  |  |  |
| --- | --- | --- |
| 动作 | deleted表（old） | inserted表（new） |
| insert | 不存储记录 | 存储新插入的记录 |
| update | 存放之前的记录 | 存放更新后的记录 |
| delete | 存放被删除的记录 | 不存储记录 |

举例：一个update的更新操作可以看成是，往inserted表中插入一条新的记录，复制旧的记录到deleted表中，然后删除指定表中的记录，然后写入inserted表中新的记录。

**for each row：对每行都添加触发操作，只要被更改了就触发。**

举例：对于学生的两个表，如果改变了student表中学生的姓名，那么stud1表也跟随改动。

**drop trigger if exists trigger\_1;**

**delimiter //**

**create trigger trigger\_1 after update on student for each row**

**begin**

**update stud1 set stud\_cname=new.stud\_cname where stud\_cname=old.stud\_cname;**

**end;**

**测试如下：**

**update student set stud\_cname='李思思' where stud\_id=2;**

#### 14.2.18 事务处理

MySQL 事务主要用于处理操作量大，复杂度高的数据。比如说，在人员管理系统中，你删除一个人员，你即需要删除人员的基本资料，也要删除和该人员相关的信息，如信箱，文章等等，这样，这些数据库操作语句就构成一个事务！

事务的特性：

事务有以下四个标准属性的缩写**ACID**，通常被称为：**Atomicity（原子性）、Consistency（稳定性）、Isolation（隔离性）、Durability（可靠性）。**

原子性: 确保工作单元内的所有操作都成功完成，否则事务将被中止在故障点，和以前的操作将回滚到以前的状态。

一致性: 确保数据库正确地改变状态后，成功提交的事务。

隔离性: 使事务操作彼此独立的和透明的。事务独立运行。一个事务处理后的结果，影响了其他事务，那么其他事务会撤回。事务的100%隔离，需要牺牲速度。

持久性: 确保提交的事务的结果或效果的系统出现故障的情况下仍然存在。

在MySQL中使用**commit和rollback**关键字来管理事务，commit表示**提交**，当整个事务在执行时没有出现任何差错，则提交事务。一旦出现某个不可预料的问题，导致了数据的不完整性，则使用**rollback关键字进行事务回滚。**之前做的所有的操作将都会被撤销。

保留点：savepoint sa\_name;(设置保留点)

rollback to sa\_name;(回滚到保留点处)

autocommit标志决定是否自动提交更改，不管有没有commit语句。将autocommit设置为0指示MySQL不自动提交更改，设置autocommit为1就表示自动提交更改。

set autocommit = 0;

````````````````````````````````````````

CREATE PROCEDURE tfer\_funds (from\_account int, to\_account int, tfer\_amount numeric(10,2))

BEGIN

start transaction;

update account\_balance set balance=balance-tfer\_amount where account\_id=from\_account;

update account\_balance set balance=balance+tfer\_amount where account\_id=to\_account;

commit;

END;

**一个事务是一个连续的一组[数据库](http://www.111cn.net/database/database.html" \t "https://www.cnblogs.com/ymy124/p/_blank)操作**

#### 14.2.19 数据库的备份与还原

备份MySQL数据库的命令

mysqldump -hhostname -uusername -ppassword databasename > backupfile.sql

格式：mysqldump -h localhost(填数据所在的主机名) -u root（用户名） -p db\_name(要备份的数据库名)>path(要备份的地址)

例如：打开[dos](https://www.2cto.com/os/dos/" \t "https://www.2cto.com/database/201403/_blank)窗输入 mysqldump -u root -p db\_name>E:/11.sql;

备份MySQL数据库为带删除表的格式

备份MySQL数据库为带删除表的格式，能够让该备份覆盖已有数据库而不需要手动删除原有数据库。

mysqldump -–add-drop-table -uusername -ppassword databasename > backupfile.sql

备份MySQL数据库某个(些)表

mysqldump -hhostname -uusername -ppassword databasename specific\_table1 specific\_table2 > backupfile.sql

同时备份多个MySQL数据库

mysqldump -hhostname -uusername -ppassword –databases databasename1 databasename2 databasename3 > multibackupfile.sql

仅仅备份数据库结构

mysqldump –no-data –databases databasename1 databasename2 databasename3 > structurebackupfile.sql

还原MySQL数据库的命令

mysql -hhostname -uusername -ppassword databasename < backupfile.sql