



C++ 教学课程

数据库基础--MySQL

内容



数据库设计基础

MySQL基础知识

SQL语言

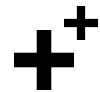
MySQL编程基础

触发器、存储过程、游标、事务

C语言访问MySQL数据库

数据库概述--关系数据库管理系统

- 几个概念
 - 数据库 (Database,简称DB)
 - 数据库管理系统(Database Management System,简称DBMS)
 - 关系型数据库管理系统 (Relational Database Management System,RDBMS)
- 主流的关系数据库管理系统
 - Oracle
 - DB2
 - Sybase
 - SQL Server
 - MySQL



数据库概述--关系数据库

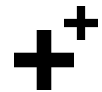
- 关系：描述两个元素间的关联或对应关系
- 使用关系模型把数据组织到二维数据表(Table)中
 - 列：字段
 - 行：记录
 - ...

课程名	人数上限	任课教师	课程描述	状态
java语言程序设计	60	张老师	暂无	未审核
MySQL数据库	150	李老师	暂无	未审核
C语言程序设计	60	王老师	暂无	未审核
英语	230	马老师	暂无	未审核
数学	230	田老师	暂无	未审核



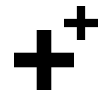
数据库概述--结构化查询语句

- SQL(Structured Query Language)：结构化查询语言
- SQL是在关系数据库上执行数据操作、检索及维护所使用的标准语言, 可以用来查询数据，操纵数据，定义数据，控制数据
- 所有数据库都使用相同或者相似的语言
- SQL可分为：
 - 数据定义语言（DDL）：Data Definition Language
 - 数据操纵语言（DML）：Data Manipulation Language
 - 事务控制语言（TCL）：Transaction Control Language
 - 数据查询语言（DQL）：Data Query Language
 - 数据控制语言（DCL）：Data Control Language



数据库设计基础--辅助工具

- 模型：现实世界中事物特征与事物行为得到抽象
 - 数学模型
 - 描述模型
 - 图形模型
 - E-R图、数据库模型图、.....
- 工具
 - PowerDesigner
 - Visio
- 技术
 - 关系数据库设计技术



关系数据库设计--为什么需要设计数据库

- 良好的数据库设计
 - 节省数据的存储空间
 - 能够保证数据的完整性
 - 方便进行数据库应用系统的开发
- 糟糕的数据库设计：
 - 数据冗余、存储空间浪费
 - 内存空间浪费
 - 数据更新和插入的异常



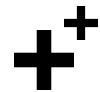
软件项目开发周期中的数据库设计

- 需求分析阶段
 - 需要分析客户的业务和数据处理需求；
- 概要设计阶段
 - 设计数据库的E-R模型图，确认需求信息的正确和完整；
- 详细设计阶段
 - 应用三大范式审核数据库结构；
- 代码编写阶段
 - 构建数据库，编码实现应用，
-



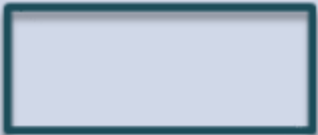


设计数据库的步骤

- 收集信息
 - 与该系统有关人员进行交流、座谈，充分了解用户需求，理解数据库需要完成的任务
- 标识实体（ Entity ）
 - 标识数据库要管理的关键对象或实体，实体一般是名词
- 标识每个实体的属性（ Attribute ）
- 标识实体之间的关系（ Relationship ）



绘制E-R图

• 常用符号

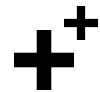
符号	含义
	实体，一般是名词
	属性，一般是名词
	关系，一般是动词

• 映射基数

– 一对一、一对多、多对一、多对多

关系模式

- 一个关系的属性名的集合称为关系模式
- 选课系统数据库的关系模式是：
 - 学生（学号、学生姓名、联系方式）
 - 课程（课程编号、课程名称、人数上限、描述、状态）
 - 教师（教师工号、教师名称、联系方式）
 - 班级（班号、班级名称、所属院系）



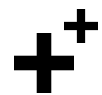
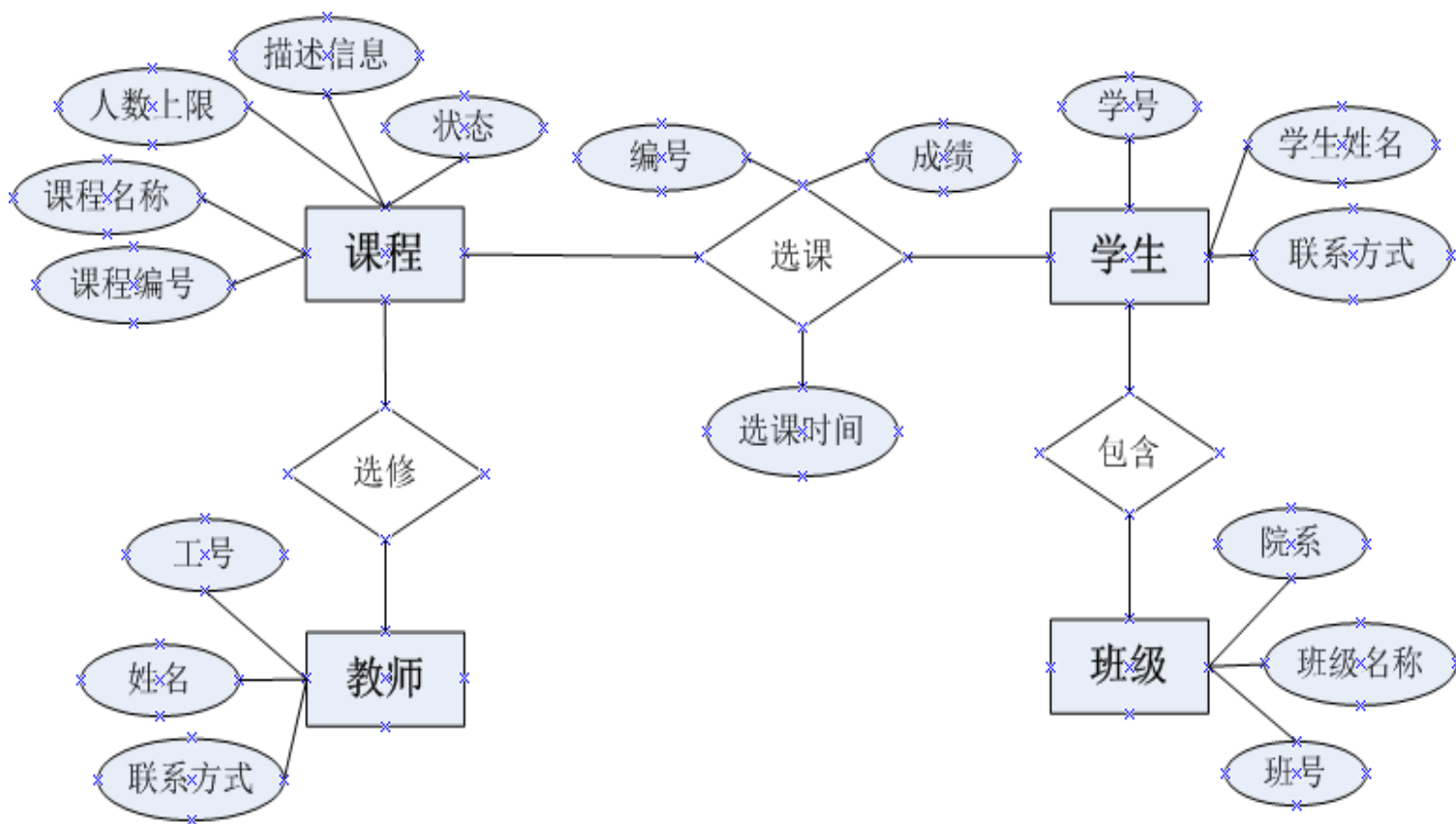
使用三大范式规范数据库表的设计

- 第一范式的目标是确保每列的原子性
 - 如果每列都是不可再分的最小数据单元（也称为最小的原子单元），则满足第一范式（1NF）
- 第二范式要求一张表只描述一件事
- 如果一个关系满足2NF，并且除了主键以外的其他列都不传递依赖于主键列，则满足第三范式（3NF）



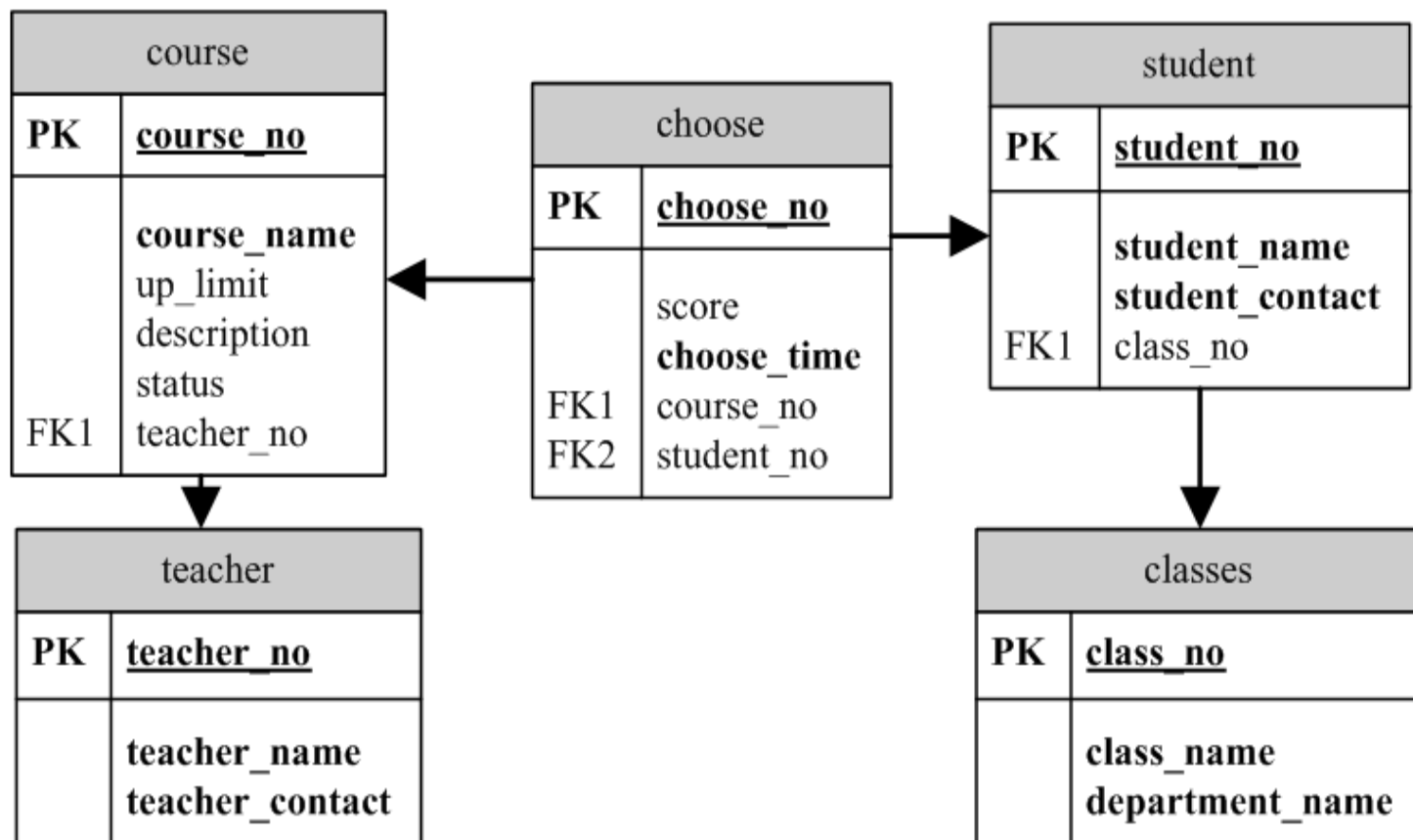
使用三大范式规范数据库表的设计(续1)

- 规范化的E-R图



使用三大范式规范数据库表的设计(续2)

- 规范化的数据库模型图



MySQL概述

- MySQL的特点
 - 性能高效、跨平台支持、简单易用、开源、支持多用户
- 启动和停止服务
 - 启动服务
service mysql start
service mysql restart
 - 停止服务
service mysql stop
- 连接MySQL服务器
 - 语法：
mysql [-h host] [-u username] [-p[pwd]][dbname]



MySQL数据库管理

- 创建数据库
 - create database database_name;
- 查看数据库
 - show databases;
- 查看数据库结构
 - show create database database_name;
- 选择数据库
 - use database_name;
- 删除数据库
 - drop database database_name;



MySQL表管理

- 创建数据库表

```
create table table_name(
    field_name type,
    field_name type,..
);
```

- 查看数据库表

- 查看当前数据库中所有的表：show tables
- 查看指定表的表结构：desc[ribe] table_name
- 查看指定表的详细信息：show create_table table_name

- 删除表

```
drop table table_name;
```



数据类型--整数类型

- MySQL只要支持5种整数类型：tinyint、smallint、mediumint、int和bigint
- 可以使用关键字“unsigned”对整数类型进行修饰
- 取值范围：

类型	字节数	范围（有符号）	范围（无符号）
tinyint	1字节	(-128,127)	(0,255)
smallint	2字节	(-32 768,32 767)	(0,65 535)
mediumint	3字节	(-8 388 608,8 388 607)	(0,16 777 215)
int	4字节	(-2 147 483 648,-2 147 483 647)	(0,4 294 967 295)
bigint	8字节	(-9 233 372 036 854 775 808, 9 233 372 036 854 775 807)	(0, 18 446 744 073 709 551 615)

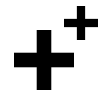


数据类型--小数类型

- 精确小数类型 decimal
 - decimal(length,precision)
 - length决定了该小数的最大位数
 - precision用于设置精度（小数点后数字的位数）
 - 例如：decimal(5,2)
- 浮点数类型
 - 单精度浮点型 float
 - 双精度浮点型 double
 - 例如，定义工资字段salary，可以用
salary float unsigned
 - 其中，unsigned用于约束工资不能为负数。

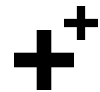
数据类型--字符串类型

- 定长字符串：char(n)
 - 最多容纳字符数：255
 - n的取值和字符集无关
- 变长字符串：varchar(n)
 - n的取值和字符集有关
- 例如在表student中的student_name列的定义如下：
student_name char(10);
表示student_name列中最多可存储长度为10的字符串。



数据类型--日期类型

- date
 - 3个字节
 - 取值范围：'1000-01-01' ~ '9999-12-31 '
 - 格式：YYYY-MM-DD
- datetime
 - 8个字节
 - 取值范围：'1000-01-01 00:00:00'~ '9999-12-31 23:59:59'
 - 格式：YYYY-MM-DD HH:ii:ss
- 例如在表choose中的choose_time列的定义如下：
choose_time datetime;
表示choose_time列中存放的是日期数据



创建表—设置约束

- 创建表的语法

```
create table 表名(
    字段名1 数据类型 [约束条件],
    字段名2 数据类型 [约束条件],
    ...
    [其他约束条件],
    [其他约束条件]
)其他选项
```

- 主键约束

- 单一字段作主键

字段名 数据类型 [其他约束条件] primary key

- 多个字段组合作主键

primary key(字段名1,字段名2)

创建表—设置约束

- 非空约束

- 直接在该字段的数据类型后加上关键字not null，语法：

字段名 数据类型 not null

- 默认值约束

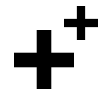
- 直接在该字段的数据类型及约束条件后加上“ default 默认值”即可，语法：

字段名 数据类型 [其他约束条件] default 默认值

- 唯一约束

- 直接在该字段的数据类型后加上关键字unique，语法：

字段名 数据类型 unique



创建表--设置约束

- 外键约束

- 主要用于定义表与表之间的关系

主表：提供数据的表

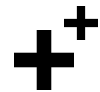
从表：使用主表中数据的表

外键字段的值，要么是NULL，要么来自于主表的主键

- 语法规则

constraint 约束名 foreign key (从表中的字段名或字段列表) references 主表(字段名或字段列表) [on delete 级联选项] [on update 级联选项]

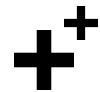
- 级联选项包括： cascade、 set null、 no action、 restrict(默认)



创建表--设置自增型字段

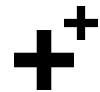
- 设置自增型字段
 - 如果要求数据库表的某个字段依次递增，且不重复，则可以将该字段设置为自增型字段。
 - 设置自增型字段的语法格式如下：

字段名 数据类型 auto_increment
 - 自增型字段的数据类型必须是整数。向自增型字段插入一个NULL值（推荐）或0时，该字段值会自动设置为比上一次插入值更大的值。如果新增加的记录时表中的第一条记录，则该值为1。
 - 自增型字段必须是主键



创建表—复制表结构

- 在create table语句的末尾添加like子句
 - 语法：
create table 表名 like 源表
- 在create table语句的末尾添加select语句
 - 语法：
create table 表名 select * from 源表



修改表结构--修改字段的相关信息

- 删除字段，语法：

```
alter table 表名 drop 字段名;
```

- 添加新字段，语法：

```
alter table 表名 add 新字段名 数据类型 [约束条件]
[first|after 旧字段名]
```

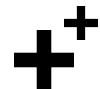
- 修改字段

- 修改字段名（及数据类型），语法：

```
alter table 表名 change 旧字段名 新字段名 数据类型
```

- 仅修改字段的数据类型，语法：

```
alter table 表名 modify 字段名 数据类型
```



修改表结构--修改约束条件

- 添加约束条件

- 语法:

```
alter table 表名 add constraint 约束名 约束类型 ( 字段名 )
```

- 删除约束条件

- 删除主键约束

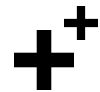
```
alter table 表名 drop primary key
```

- 删除外键约束

```
alter table 表名 drop foreign key 约束名
```

- 删除唯一约束

```
alter table 表名 drop index 唯一索引名
```



修改表结构--修改表名

- 修改表名的语法格式较为简单，格式如下：

```
rename table 旧表名 to 新表名
```

该命令等效于：

```
alter table 旧表名 rename 新表名
```



删除表

- 语法：
drop table 表名
- 对于存在外键约束关系的若干个表，需要注意删除表的顺序
 - 先删除子表，再删除主表
 - 先删除外键约束条件，再删除表



insert语句

- 语法

insert into 表名 [(字段列表)] values (值列表)

- 向教师表teacher中的所有字段中插入数据

teacher_no	teacher_name	teacher_contact
001	张老师	110000000000
002	李老师	120000000000
003	王老师	130000000000

- 向班级表classes的指定字段插入数据

class_no	class_name	Department_name
	2017自动化1班	机电工程
	2017自动化2班	机电工程
	2017自动化3班	机电工程

insert语句

- 自增型字段
 - 使用delete语句删除表中的记录，或者插入语句错误等，会造成自增型字段不连续
 - 例如，向班级表classes表中插入数据时，第一条插入语句中班级名称class_name违反了唯一约束，插入失败。第二条插入语句成功，且class_no为5（原有3条记录，class_no分别为1,2,3）

```
use choose;  
#插入失败  
insert into classes values(null,'2017自动化3班','机电工程');  
#插入成功，class_no为5  
insert into classes values(null,'2017计算机应用1班','信息工程');
```



insert语句--插入多条记录

- 使用insert语句可以一次性地向表中批量插入多条记录，语法格式如下：

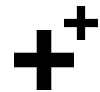
insert into 表名[(字段列表)]

values(值列表1), (值列表2), ... (值列表n);

- 在insert语句中使用select子句可以将源表的查询结果添加到目标表中，语法格式如下：

insert into 目标表名[(字段列表1)]

select (字段列表2) from 源表 where 条件表达式;



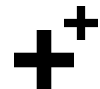
update语句

- 使用update语句可以对表的一行或多行进行修改，语法：

update 表名

set 字段名1=值1,字段名2=值2,...,字段名n=值n

[where 条件表达式];



delete语句

- 如果表中的某条（某几条）记录不再使用，可以使用 delete 语句删除，语法格式为：
delete from 表名 [where 条件表达式];
- 删除表记录时，需要注意表之间的外键约束关系。



truncate语句

- truncate table用于完全清空一个表，语法格式如下：
truncate [table] 表名;
- truncate语句和delete语句的区别：
 - 使用truncate语句清空父表，将会失败
 - 使用truncate语句清空表记录，会重置自增型字段的计数器
 - truncate语句不支持事务的回滚



select语句

- select语句的语法格式如下：

select 字段列表 from 数据源

[where 条件表达式]

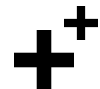
[group by 分组字段[having 条件表达式]]

[order by 排序字段 [asc | desc]]

- 可以使用关键字as为字段或表达式命名别名

语法：

字段（或表达式）[as] 别名



select语句

- 使用distinct过滤重复记录

distinct 字段名

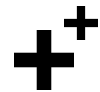
- 使用limit实现分页，语法格式如下：

select 字段列表 from 数据源 limit [start,]length;

其中，

start表示从第几行记录开始检索

length表示检索多少行记录。



表连接

- 语法：

- 使用join...on实现表连接

from 表名1 [连接类型] join 表名2 on 连接条件

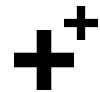
可以给表名指定别名

如果两张表中有同名字段，字段名前需要用表名作前缀

- 使用where子句实现表连接

- 分类

- 内连接：inner
- 外连接：outer



表连接--内连接

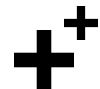
- 内连接将表中满足指定连接条件的记录连接成新的结果集，并过滤掉所有不满足连接条件的记录。
 - 内连接是最常用的连接类型，也是默认的连接类型
 - 语法：
from 表1 [inner] join 表2 on 表1和表2之间的连接条件

```
-- 插入班级
insert into classes values(4,'2017自动化4班','机电工程学院');
-- 插入学生
insert into student values('2017006','张三丰','200000000000',null);
-- classes表和student表内连接
select
student_no,student_name,student_contact,student.class_no,class_name,
department_name
from student join classes on student.class_no=classes.class_no;
```



表连接--外连接

- 外连接的结果集=内连接的结果集+匹配不上的记录
- 根据哪张表匹配不上的记录包含在结果集中，外连接分为两种
- 左外连接
 - 左外连接的结果集=内连接的结果集+左表匹配不上的记录
 - 语法：
from 表1 left join 表2 on 表1和表2之间的连接条件
- 右外连接
 - 右外连接的结果集=内连接的结果集+右表匹配不上的记录
 - 语法：
from 表1 right join 表2 on 表1和表2之间的连接条件



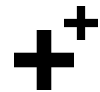
表连接--多表连接

- from子句可以指定多个数据源，实现多表连接，继而实现从更多的表中（以3个表为例）检索数据，语法格式如下：

```
from 表1
```

```
    [连接类型] join 表2 on 表1和表2之间的连接条件
```

```
    [连接类型] join 表3 on 表2和表3之间的连接条件
```



where子句

- 使用where子句可以设置结果集的过滤条件，语法格式如下：

where 条件表达式

- 单一的过滤条件可以使用下面的布尔表达式表示：

表达式1 比较运算符 表达式2

- 常用的比较运算符有=（等于）、>（大于）、>=（大于等于）、<（小于）、<=（小于等于）、!=（不等于）、<>（不等于）。

- 在where子句中使用连接条件实现内连接
 - 语法：from 表1,表2 where 连接条件



where子句

- between..and..运算符用于判断表达式的值是否在位于指定的取值范围内，语法格式如下：

表达式 between 起始值 and 终止值

- is null用于判断表达式的值是否为空值null，is null的语法格式如下：

表达式 is null

- in运算符用于判断一个表达式的值是否位于一个离散的数学集合中，in运算符的语法格式如下：

表达式 in(值1,值2[,...])



where子句--模糊查询

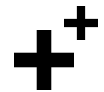
- like 运算符用于判断一个字符串是否与给定的模式相匹配，like运算符的语法格式如下：
 - 字符串表达式 like 模式
 - 模式是一种特殊的字符串，其中不仅包含普通字符，还包含通配符。MySQL中的通配符如下表所示：

通配符	功能
%	匹配零个或多个字符组成的任意字符串
_（下划线）	匹配任意一个字符

- 模糊查询中 “%” 或者 “_” 字符时，需要将 “%” 或者 “_” 字符转义
 - 使用转义字符 \
 - 使用escape关键字自定义一个转义字符

where子句--逻辑运算符

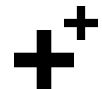
- 常用的逻辑运算符包括逻辑与（ and ）、逻辑或（ or ）以及逻辑非（ !、 not ）
- and 运算符
 - 语法：布尔表达式1 and 布尔表达式2
- or 运算符
 - 语法：布尔表达式1 or 布尔表达式2
- 逻辑非
 - 逻辑非为单目运算符，用于对布尔表达式取反，语法：
!布尔表达式



where子句--逻辑运算符

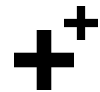
- 前面介绍的运算符，使用逻辑非取反，语法格式如下表所示。

运算符	逻辑取反
>	<=
<	>=
=	!=(<>)
between ... and	not between ... and
in	not in
like	not like
is null	is not null



order by 子句

- order by子句用于对结果集排序
 - 语法：
order by 字段1 {[asc]|desc} [...,字段n {[asc]|desc}]
asc : 升序 默认
desc : 降序
- 在order by子句中，可以指定多个字段作为排序的关键字
 - 其中第一个字段为排序主关键字，第二个字段为排序次关键字，以此类推
 - 排序时，MySQL总是将NULL当作“最小值”处理。

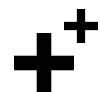


聚合函数

- 聚合函数用于对一组值进行计算并返回一个汇总值，常用的聚合函数有sum、avg、count、max和min等。

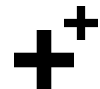
函数	功能
count	用于统计结果集中记录的行数
max	用于统计字段的最大值
min	用于统计字段的最小值
sum	用于对数值型字段的值累加求和
avg	用于对数值型字段的值求平均值

- 聚合函数在NULL值的处理为忽略
- 聚合函数的参数可以distinct修饰



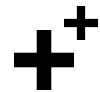
group by子句

- group by子句将查询结果按照某个字段（或多个字段）进行分组（字段值相同的记录作为一个分组），通常与聚合函数一起使用
- group by子句的语法格式如下：
 - group by 字段列表[having 条件表达式] [with rollup]
- having子句语法格式与where子句语法格式类似，having子句语法格式如下：
 - having 条件表达式



group by子句

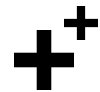
- group_concat()函数
 - group_concat()函数的功能是将集合中的字符串连接起来，此时group_concat()函数的功能与字符串连接函数concat()的功能相似。
 - group_concat()函数还可以按照分组字段，将另一个字段的值（NULL值除外）使用逗号连接起来。
- with rollup选项
 - group by 子句将结果集分为若干个组，使用聚合函数可以对每个组内的数据进行信息统计，有时对各个组进行汇总运算时，需要在分组后加上一条汇总记录，这个任务可以通过with rollup选项实现。



合并结果集

- 在MySQL数据库中，使用union可以将多个select语句的查询结果集组合成一个结果集，语法格式如下。

```
select 字段列表1 from table1 union[ all]  
select 字段列表2 from table2...
```



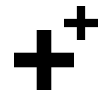
子查询--where子句中的子查询

- 子查询返回单值
 - 条件表达式可以使用>、<等比较运算符
- 子查询返回多行单列
 - 如果子查询返回的结果集是单行多列的，就不能使用比较运算符，而是使用可以处理多个值的运算符，比如in(not in)、any或者all等。
- 使用exists运算符
 - exists逻辑运算符用于检测子查询的结果集是否包含记录。如果结果集中至少包含一条记录，则exists的结果为true，否则为false。在exists前面加上not时，与上述结果恰恰相反。

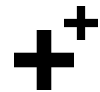


子查询--from子句中的子查询

- 每一个select语句可以看成是一个虚拟的内存表，可以在结果集的基础上进行进一步的查询。
 - select 字段列表 from (子查询语句) where 条件;

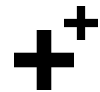


- 作用
 - 添加索引的目的是提高检索速度
 - 创建索引需要大量的时间和空间的
 - 索引会降低DML操作的速度
- 索引关键字的选取原则
 - 表的某个字段值的离散度越高，该字段越适合选作索引的关键字
 - 占用存储空间少的字段更适合选作索引的关键字
 - 存储空间固定的字段更适合选作索引的关键字
 - where子句中经常使用的字段、分组字段或排序字段、两个表的连接字段应该创建索引
 - 更新频繁的字段不适合创建索引，不会出现在where子句中的字段不应该创建索引



索引与约束

- 约束分为主键约束、唯一性约束、默认约束、检查约束、非空约束以及外键约束。其中，主键约束、唯一性约束以及外键约束与索引的联系较为紧密。
- 约束主要用于保证业务逻辑操作时数据的完整性；而索引则是将关键字数据以某种数据结构的方式存储到外存，用于提升数据的检索性能。
- 约束是逻辑层面的概念；而索引既有逻辑上的概念，更是一种物理存储方式，且事实存在，需要耗费一定的存储空间。



创建索引

- 创建索引的方法有两种：创建表的同时创建索引，在已有表上创建索引。
- 创建表的同时创建索引

```
create table 表名(  
    字段名1 数据类型[约束条件],  
    字段名2 数据类型[约束条件],  
    ...  
    [其他约束条件],  
    [其他约束条件],  
    ...  
    [unique] index [索引名](字段名[(长度)])  
)
```



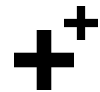
创建索引(续1)

- 在已有表上创建索引
 - 在已有表上创建有两种语法格式，这两种语法格式的共同特征是需要指定在哪个表上创建索引，语法格式分别如下：
 - 语法格式一

```
create [unique] index 索引名 on 表名 (字段名[(长度)])
```

- 语法格式二

```
alter table 表名 add [unique] index 索引名 (字段名[(长度)])
```



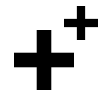
删除索引

- 如果某些索引降低了数据库的性能，或者根本就没有必要使用该索引，此时可以考虑将该索引删除，删除索引的语法格式如下。

```
drop index 索引名 on 表名
```

- 例如，删除书籍表book的复合索引complex_index，可以使用下面的SQL语句实现该功能

```
drop index complex_index on book;
```

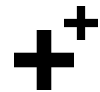


视图

- 创建视图
 - 视图中保存的仅仅是一条select语句，该select语句的数据源可以是基表，也可以是另一个视图。创建视图的语法格式如下。

```
create view 视图名[(视图字段列表)]  
as  
select语句
```

- 查看视图的定义
 - 使用查看表结构的方式查看视图的定义。
 - MySQL命令“show tables;”不仅显示当前数据库中所有的基表，也会将所有的视图罗列出来。
 - MySQL系统数据库information_schema的views表查看视图的详细信息。



视图

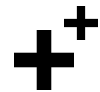
- 视图的作用

与直接从数据库表中提取数据相比，视图的作用可以归纳为以下几点：

- 使操作变得简单
- 避免数据冗余
- 增强数据安全性
- 提高数据的逻辑独立性

- 如果某个视图不再使用，可以使用drop view语句将其删除，语法格式如下：

- drop view 视图名



MySQL编程基础知识

变量

- 用户会话变量
 - 会话期间一直有效，但其它的客户机不能访问。
 - 一般情况下，用户会话变量的定义与赋值会同时进行。
 - 方法一：使用set命令定义用户会话变量，并为其赋值

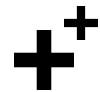
```
set @user_variable1=expre1 [,@user_variable2=expre2,...];
```

- 方法二：使用select语句定义用户会话变量，并为其赋值

```
select @user_variable1:=expre1[,@user_variable2:=expre2,...]
```

或

```
select expre1 into @user_variable1, expre2 into @user_variable2,...
```



变量

- 局部变量

- 语法：

- declare 变量名 数据类型;

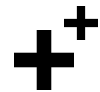
- 局部变量必须定义在存储程序中，且作用范围也只在存储程序中。

- 局部变量主要用于下面3种场合：

- 局部变量定义在存储程序的begin-end语句块中

- 局部变量作为存储过程或函数的参数使用

- 局部变量用于存储程序的SQL语句中

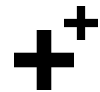


begin-end语句块

- begin-end语句块格式如下，其中开始标签名称与结束标签名称必须相同。

```
[开始标签] begin  
[局部] 变量的声明;  
错误触发条件的声明;  
游标的声明;  
错误处理程序的声明;  
业务逻辑代码;  
end [结束标签];
```

- 在MySQL中，单独使用begin-end语句块没有任何意义，只有将其封装到存储过程、函数、触发器以及事件等存储程序内部才有意义

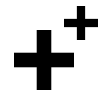


重置命令结束标记

- begin-end语句块中通常存在多条MySQL表达式，每条MySQL表达式使用“;”作为结束标记。由于begin-end语句块中的多条MySQL表达式密不可分，为了避免这些MySQL表达式被拆开，需要重置MySQL客户机的命令结束标记（delimiter）。例如，

```
delimiter $$  
select * from student where student_name like '张_ '$$  
delimiter ;  
select * from student where student_name like '张_';
```

其中，第一条命令将当前MySQL客户机的命令结束标记“临时地”设置为“\$\$”；紧接着在select语句中使用“\$\$”作为语句结束标记；第三条命令将命令结束标记恢复为“;”；最后的select重新使用“;”作为结束标记。

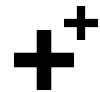


自定义函数

- 创建自定义函数的语法格式如下：

```
create function 函数名(参数1,参数2,...) returns 返回值的数据类型
begin
    函数体
    return 语句;
end;
```

- 自定义函数时数据库的对象，因此，创建自定义函数时，需要指定该自定义函数属于哪个数据库。
- 同一个数据库内，自定义函数不能和已有的函数名（包括系统函数名）重名。
- 函数必须指定返回值数据类型，且须与return语句中的返回值的数据类型相匹配。



函数的维护

- 查看函数的定义

- 查看当前数据库中所有的自定义函数的信息

```
show function status;  
show function status like 模式;
```

- 查看指定数据库中的所有自定义函数名

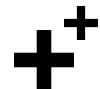
```
select name from mysql.proc where db = '数据库名' and type =  
'function';
```

- 查看指定函数名的详细信息

```
show create function 函数名\G
```

- 删除函数

```
drop function 函数名;
```



条件控制语句

- if 语句

- 语法格式：

```
if 条件表达式1 then 语句块1;
[elseif 条件表达式2 then 语句块2]...
[else 语句块n]
end if;
```

- case 语句

- case语句用于实现比if语句分值更为复杂的条件判断，语法格式如下：

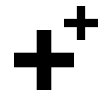
```
case 表达式
when value1 then 语句块1;
when value2 then 语句块2;
when value3 then 语句块3;
...
else 语句块n;
end case;
```



循环语句

- MySQL提供了3种循环语句，分别是while、repeat以及loop。除此外，MySQL还提供了iterate语句以及leave语句，用于循环的内部控制。
- while语句
 - while语句语法格式如下：

```
[循环标签:]while 条件表达式 do  
循环体;  
end while [循环标签];
```



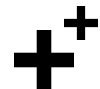
循环语句

- leave语句
 - leave语句用于跳出当前的循环语句，语法格式如下：

```
leave 循环标签;
```

- iterate语句
 - iterate语句用于跳出本次循环，继而进行下次循环。语法：

```
iterate 循环标签;
```



循环语句

- repeat语句

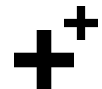
- 当表达式的值为false时反复执行循环，直到条件表达式的值为true，repeat语句的语法格式如下：

```
[循环标签:]repeat
循环体;
until 条件表达式
end repeat[循环标签];
```

- loop语句

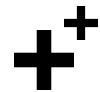
- 由于loop循环语句本身没有停止循环的语句，因此loop通常借助leave语句跳出loop循环，loop循环的语法格式如下：

```
[循环标签:] loop
循环体;
if 条件表达式 then
    leave [循环标签];
end if;
end loop;
```



系统函数--数学函数

- 求近似值函数
 - `round(x)` : 计算离x最近的整数。
 - `round(x,y)` : 计算离x最近的小数 (y可以取负值)
 - `truncate(x,y)` : 截取到小数点y位 (y可以取负值)
 - `ceil(x)` : 返回大于等于x的最小整数
 - `floor(x)` : 返回小于等于x的最大整数
- 随机函数
 - `rand()` : 返回随机数

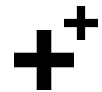


系统函数--字符串函数

- `char_length(x)` : 获取字符串x的长度
- `length(x)` : 获取字符串x占用的字节数
- `concat(x1,x2,...)` : 用于将x1,x2等若干个字符串连接成一个字符串。
- `ltrim(x)` : 用于去掉字符串x开头的所有空格字符。
- `rtrim(x)` : 用于去掉字符串x结尾的所有空格字符。
- `trim(x)` : 用于去掉字符串x开头以及结尾的所有空格字符。
- `trim([leading|both|trailing] x1 from x2)` : 用于从x2字符串的前缀或者（以及）后缀中去掉字符串x1。
- `left(x,n)` : 返回字符串x的前n个字符。
- `right(x,n)` : 返回字符串x的后n个字符。
- `upper(x)` : 返回将字符串x中的所有字母变成大写字母的字符串。
- `lower(x)` : 返回将字符串x中的所有字母变成小写字母的字符串。
- `substring(x,start,length)` : 从字符串x的第start个位置开始获取length长度的字符串。

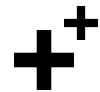
系统函数--日期和时间函数

- 获取MySQL服务器当前日期或时间函数
 - curdate() : 获取服务器当前日期
 - curtime() : 获取服务器当前时间
 - now()、sysdate() : 获取服务器当前日期和时间，并且允许传递一个 ≤ 6 的整数值作为参数，从而获取更为精确的时间信息。
- 获取MySQL服务器当前日期或时间函数
 - year(x)、month(x)、dayofmonth(x)、hour(x)、minute(x)、second(x)、microsecond(x)函数分别用于获取日期时间x的年、月、日、时、分、秒、微秒等信息。



系统函数--类型转换函数

- `convert()`函数，有两种用法格式：
 - `convert(x using charset)`：返回x的的charset字符集数据。
 - `convert(x,type)`：以type数据类型返回x数据。
- `cast()`函数
 - `cast(x as type)`：以type数据类型返回x数据。



系统函数--条件控制函数

- if()函数

if(condition,v1,v2)函数中，condition为条件表达式，当condition的值为true时，函数返回v1的值，否则返回v2的值

```
set @score1 = 40;
set @score2 = 70;
select if(@score1>=60,'及格','不及格'),if(@score2>=60,'及格','不及格');
```

- ifnull()函数

在ifnull(v1,v2)中，如果v1的值为NULL，则该函数返回v2的值；如果v1的值不为NULL，则该函数返回v1的值

```
set @score1 = 40;
select ifnull(@score1,'没有成绩'),ifnull(@score_null,'没有成绩');
```



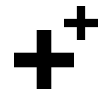
触发器

- 概念
 - 触发器定义了一系列操作，这一系列操作称为触发程序，当触发事件发生时，触发程序会自动运行。

- 语法

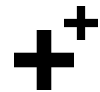
```
create trigger 触发器名 触发时间 触发事件 on 表名
for each row
begin
    触发程序
end;
```

- MySQL的触发时间有3种：insert、update及delete。
- 触发器的触发时间有两种：before与after。
- for each row表示行级触发器。



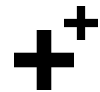
查看和删除触发器

- 查看触发器的定义
 - 使用show trigger命令查看触发器的定义。
show trigger\G
或
show trigger like 模式\G
 - 使用 show create trigger命令查看某一个触发器的定义。
show create trigger 触发器名\G
- 删除触发器
 - 语法格式如下：
drop trigger 触发器名



临时表

- 按照临时表的存储位置，分为：
 - 内存临时表
 - 外存临时表
- 按照临时表的创建时机，分为：
 - 自动创建的临时表
 - 手动创建的临时表



临时表的创建、查看和删除

- 手工创建临时表

- 语法和只需在create table上加上temporary关键字：

```
create temporary table temp(name char(100));
insert into temp values('test');
select * from temp;
```

- 临时表时数据库的对象，因此创建临时表时，需要指定该临时表隶属于哪个数据库。
- 手工创建的临时表时“会话变量”，仅在当前服务器连接中有效。

- 查看临时表的定义

- show create table 临时表名\G

- 删除临时表

- drop temporary table 临时表名;

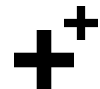
存储过程

- 创建存储过程的语法格式如下

```
create procedure 存储过程名 (参数1, 参数2,...)
begin
    存储过程语句块;
end;
```

- 存储过程有3种类型的参数：
 - in参数：默认 输入参数
 - out参数：输出参数
 - inout参数：既是输入参数，又是输出参数
- 调用存储过程需要使用call关键字，另外，还要向存储过程传递参数。

```
call 存储过程名 (实参1, 实参2,...);
```



查看存储过程

- 查看存储过程的定义

- 语法：

```
show procedure status\G
```

- 查看某个数据库中的所有存储过程名

- 语法：

```
select name from mysql.proc where db= '数据库名'  
and type=' procedure' ;
```

- 查看指定数据库的指定存储过程的消息信息

- 语法：

```
show create procedure get_choose_number_proc\G
```



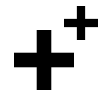
删除存储过程

- 如果某个存储过程不再需要，则可以使用drop procedure语句将其删除，语法格式如下：

```
drop procedure 存储过程名
```

- 例如，删除get_choose_number_proc存储过程可以使用下面的SQL语句：

```
drop procedure get_choose_number_proc;
```



错误处理

- 自定义错误处理程序时需要使用declare关键字，语法格式如下：

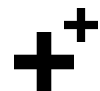
```
declare 错误处理类型 handler for 错误触发条件 自定义错误处理程序;
```

- 其中，

- 错误处理类型的取值有两种：continue,exit
- 错误触发条件有三种取值：MySQL错误代码、ANSI标准错误代码以及自定义错误触发条件。

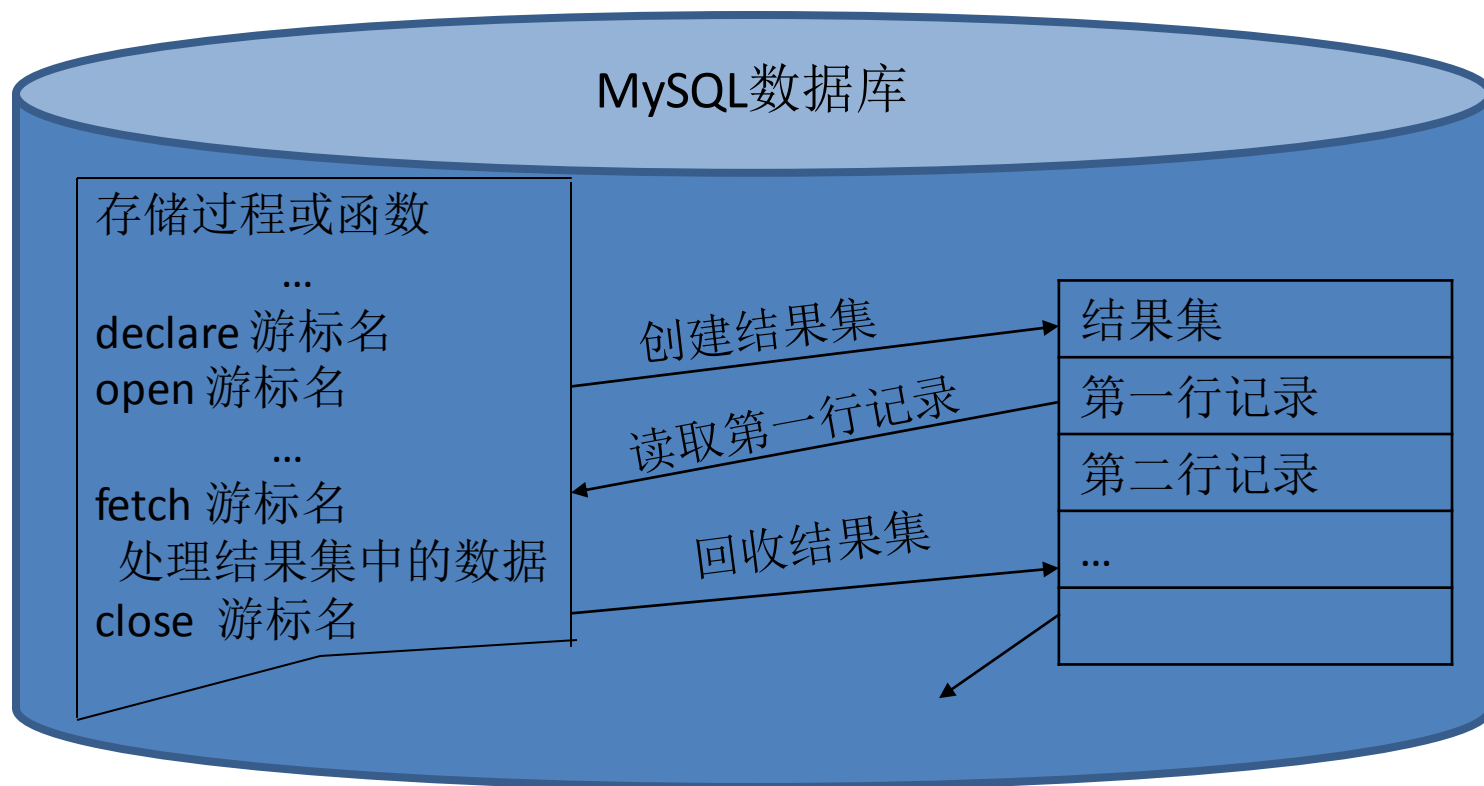
- 自定义错误触发条件允许数据库开发人员为MySQL错误代码或者ANSI标准错误代码命名，语法格式如下：

```
declare 错误触发条件 condition for 错误代码;
```



游标

- 游标本质上是一种能从select结果集中每次提取一条记录的机制，因此游标与select语句息息相关。
- 游标的使用可以概括为声明游标、打开游标、从游标中提取数据以及关闭游标。



游标的使用

- 声明游标

- 语法：

```
declare 游标名 cursor for select语句
```

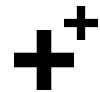
- 使用declare语句声明游标时，此时与该游标对应的select语句并没有执行，MySQL服务器内存中并不存在与select语句对应的结果集。

- 打开游标

- 语法格式：

```
open 游标名
```

- 使用open 语句打开游标后，与该游标对应的select语句被执行，MySQL服务器 内存中存放于select语句对应的结果集。



游标的使用

- 提取数据

- 语法：

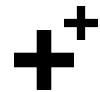
```
fetch 游标名 into 变量名1,变量名2,...
```

- 变量名的个数必须与生命游标时使用的select语句结果集中的字段个数保持一致。每执行一次fetch语句，从结果集中提取一行数据，同时游标向下移动一行。

- 关闭游标使用close语句，其语法格式如下：

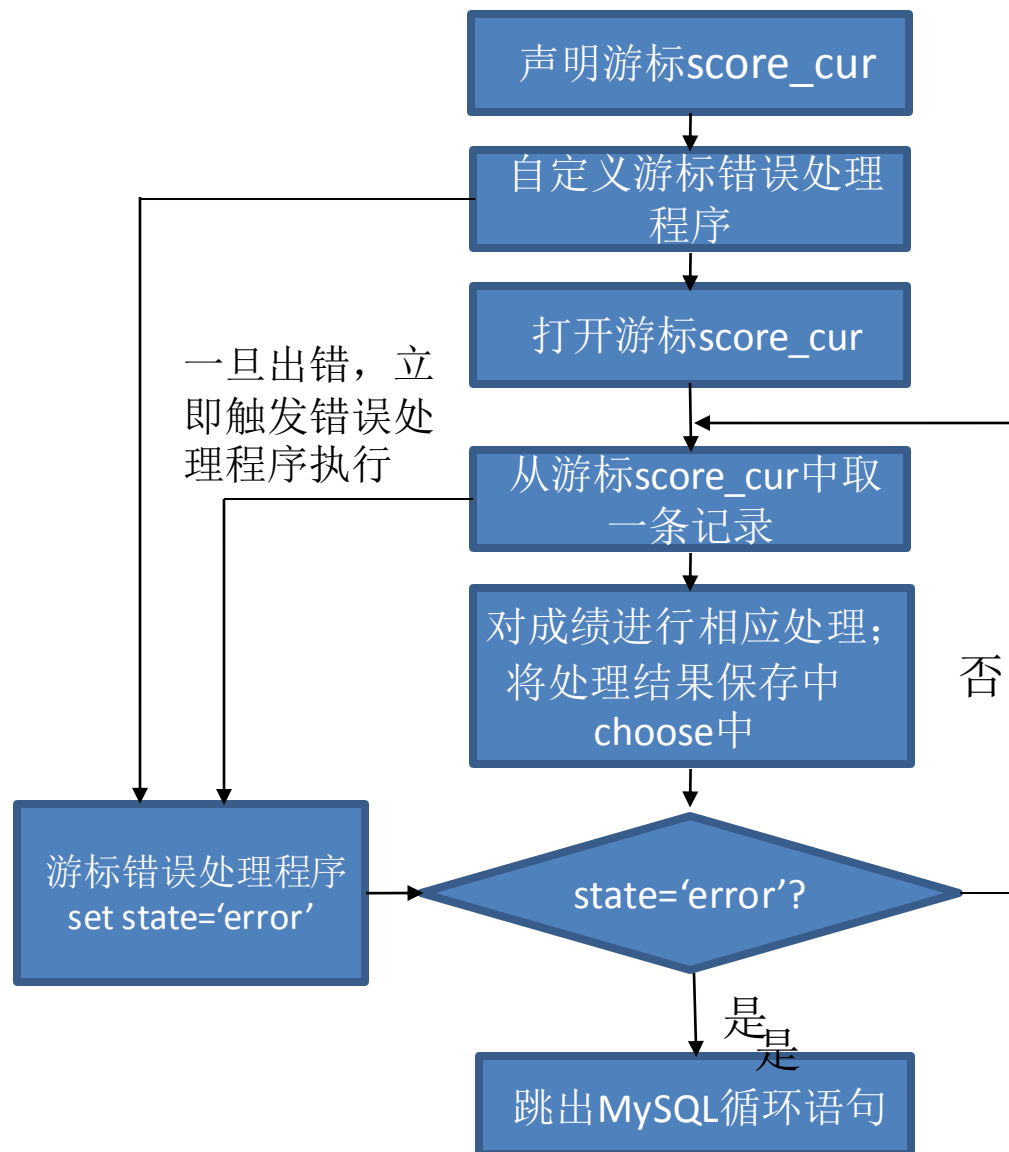
```
close 游标名
```

- 关闭游标的作用在于释放游标打开时产生的结果集，从而节省MySQL服务器的内存空间。游标如果没有被明确的关闭，那么它将在被打开的begin-end语句块的末尾关闭。



游标的使用(续2)

- 例如，某一门选修课程考试结束，教师录入学生的成绩后，出于某些原因（如试卷本身可能存在缺陷），教师需要将该课程所有的学生成绩加5分（但是总分不能超过100分），修改后的成绩如果介于55分~59分之间，将这些学生的成绩修改为60分。



预处理SQL语句—创建

- MySQL支持预处理SQL语句，预处理SQL语句的使用主要包含3个步骤：创建预处理SQL语句、预处理SQL语句以及释放预处理SQL语句。

- 创建预处理SQL语句

- 语法：

```
prepare 预处理SQL语句名 from SQL字符串
```

- 例如，下面的MySQL命令创建了名字为select_class_pre的预处理SQL语句

```
prepare select_class_pre from 'select * from classes';
```

- SQL字符串可以包含若干个“？” 问号占位符，每个“？” 问号占位符可以填充一个数据

```
prepare class_pre from 'select * from classes where class_no=?';
```

预处理SQL语句—执行

- 执行预处理SQL语句

- 语法：

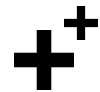
```
execute 预处理名 [using 填充数据[,填充数据...]]
```

- 例如，执行select_class_pre预处理SQL语句，可以使用下面的MySQL命令：

```
execute select_classes_pre;
```

- 预处理语句中包含‘?’占位符时，使用using关键字为占位符提供数据。例如，执行class_pre预处理SQL语句，可以使用下面的MySQL命令：

```
set @class_no = 1;  
execute class_pre using @class_no;
```

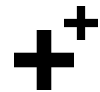


预处理SQL语句—释放

- 当预处理SQL语句不再使用时，可以使用deallocate语句将该预处理SQL语句释放。其语法格式如下：

```
deallocate prepare 预处理名
```

- 在MySQL客户机与MySQL服务器会话期间，保存在MySQL服务器内存的预处理SQL语句将一直存在，直到释放该预处理SQL语句。



事务机制

- 案例：银行转账功能
 - 创建账户表
账户不能透支
 - 测试数据
插入右表所示数据
 - 创建存储过程
实现转账功能
 - 测试存储过程
 - 再次测试

account_no	account_name	balance
1	甲	1000
2	乙	1000



关闭MySQL自定义提交

- 关闭自动提交的方法有两种：一种是显式地关闭自动提交；另一种是隐式地关闭自动提交。

- 显式地关闭自动提交

- 语法：

查看是否开启自动提交：show variables like 'autocommit';

关闭自动提交：set autocommit=0;

- 隐式地关闭自动提交

- 语法：

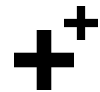
start transaction;

- 隐式地关闭自动提交。隐式地关闭自动提交不会修改系统会话变量@@autocommit的值。



回滚

- 关闭MySQL自动提交后，数据库开发人员可以根据需要回滚（也叫撤销）更新操作。
 - 语法：rollback;
- 案例：接上一案例，在关闭自动提交后，更新乙账户金额。
 - 查询全部账户信息
 - 打开另一个客户机，查询全部账户信息。对比二者的结果。
 - 在第一个客户机执行“ rollback” 命令，再次在两个客户机分别查询。



提交

- MySQL自动提交一旦关闭，数据库开发人员需要“提交”更新语句，才能将更新结果提交到数据库文件中，成为数据库永久的组成部分。
- 自动提交关闭后，MySQL的提交方式分为显式提交与隐式提交。
 - 显式提交
 - 隐式提交

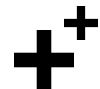
MySQL自动提交关闭后，使用下面的MySQL语句：

begin、set autocommit=1、start transaction、rename table、truncate等语句
ddl、dcl、锁语句等



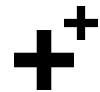
事务

- 开启事务
 - 语法：start transaction;
- 提交
 - commit;
- 回滚
 - rollback;
- 保存点
 - savepoint 保存点名;



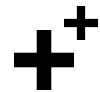
事务的ACID特性

- 事务的ACID特性原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）4个英文单词的首字母组成。
- 原子性
 - 原子性用于标识事务是否完全地完成。一个事务的任何更新都要在系统上完成，如果由于某种原因出错，事务不能完成它的全部任务，那么系统将返回到事务开始前的状态。
- 一致性
 - 事务的一致性保证了事务完成后，数据库能够处于一致性状态。如果事务执行过程中出现错误，那么数据库中的所有变化将自动地回滚，回滚到另一种一致性状态。



事务的ACID特性

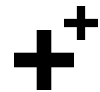
- 隔离性
 - 同一时刻执行多个事务时，一个事务的执行不能被其他事务干扰。事务的隔离性确保多个事务并发访问数据时，各个事务不能相互干扰，好像只有自己在访问数据。
- 持久性
 - 持久性意味着事务一旦成功执行，在系统中产生的所有变化将是永久的。



C语言访问MySQL数据库

连接数据库

- mysql_init函数
- 函数原型：
 - MYSQL *mysql_init(MYSQL *mysql)
- 功能
 - 分配或初始化与mysql_real_connect()相适应的MYSQL对象。
- 返回值
 - 初始化的MYSQL *句柄。
- 错误
 - 在内存不足的情况下，返回NULL。



连接数据库

- mysql_real_connect函数

- 函数原型：

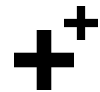
```
MYSQL *mysql_real_connect(
    MYSQL *mysql,           //已有MYSQL结构的地址
    const char *host,       // 主机名或IP地址
    const char *user,       // 登录名
    const char *passwd,     // 密码
    const char *db,         // 数据库名称
    unsigned int port,      // 端口号 默认0
    const char *unix_socket, // 套接字或命名管道 默认NULL
    unsigned long client_flag // 一般为0
)
```

- 返回值

– 连接成功，返回MYSQL *句柄。连接失败，返回NULL。

连接数据库

- mysql_close函数
- 函数原型：
 - void mysql_close(MYSQL *mysql)
- 功能
 - 关闭前面打开的连接。如果句柄是由mysql_init()分配的，该函数还将解除分配给mysql指向的连接句柄。
- 返回值
 - 无
- 错误
 - 无



DML操作

- mysql_query函数

- 函数原型

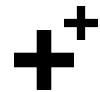
int mysql_query(MYSQL *mysql,const char *query)

- 描述

执行用“ NULL结束的字符串” 指向的SQL语句。正常情况下，字符串必须包含一条SQL语句。

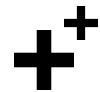
- 返回值

如果查询成功，返回0。如果出现错误，返回非0。



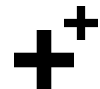
SELECT操作

- mysql_store_result函数
- 函数原型：
 - MYSQL_RES *mysql_store_result(MYSQL *mysql)
- 描述
 - 对于成功检索了数据的操作(select、show、describe等)，必须调用mysql_store_result()或mysql_use_result()。
 - 对于其他操作，不需要调用这两个函数。
 - mysql_store_result()将查询的全部结果读取到客户端。
- 返回值
 - 成功时，返回MYSQL_RES结果集。出现错误时，返回NULL



SELECT操作

- mysql_num_fields函数
- 函数原型
 - unsigned int mysql_num_fields(MYSQL_RES *res)
- 功能
 - 返回结果集中的列数
- 返回值
 - 表示结果集中列数的无符号数。



SELECT操作

- mysql_field_count函数
- 函数原型
 - unsigned int mysql_field_count(MYSQL *mysql)
- 功能
 - 返回结果集中的列数
 - 该函数的正常使用是在mysql_store_result()返回NULL（没有结果集指针）时。在这种情况下，可调用mysql_field_count()来判定mysql_store_result()是否应生成非空结果.
- 返回值
 - 表示结果集中列数的无符号数。



SELECT操作

- mysql_num_rows函数
- 函数原型：
 - my_ulonglong mysql_num_rows(MYSQL_RES* res)
- 功能
 - 返回结果集中的函数
- 返回值
 - 结果集中的行数



SELECT操作

- mysql_fetch_field函数
- 函数原型：
 - MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *res)
- 描述
 - 对于结果集，返回所有MYSQL_FIELD结构的数组。每个结构提供了结果集中一列的字段定义。
- 返回值
 - 关于结果集所有列的MYSQL_FIELD结构的数组。



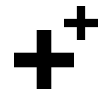
SELECT操作

- mysql_fetch_row函数
- 函数原型：
 - MYSQL_ROW mysql_fetch_row(MYSQL_RES *res)
- 描述
 - 检索结果集的下一行。
- 返回值
 - 下一行的MYSQL_ROW结构。如果没有更多要检索的行或出现了错误，返回NULL

```
MYSQL_ROW row;
```

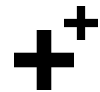
```
...
```

```
row = mysql_fetch_row(res);
```



错误处理

- mysql_errno函数
 - unsigned int mysql_errno(MYSQL *mysql)
- 对于由mysql指定的连接，mysql_errno()返回最近调用的API函数的错误代码。
- 返回值
 - 如果失败，返回上次mysql_xxx()调用的错误代码。没有错误返回0。



错误处理

- mysql_error函数
 - `const char *mysql_error(MYSQL*mysql)`
- 描述
 - 对于由mysql指定的连接，对于失败的最近调用的API函数，`mysql_error()`返回包含错误信息的、由NULL结束的字符串。如果该函数未失败，该函数的返回值可能是以前的错误，或指明无错误的空字符串。
- 返回值
 - 返回描述错误的、由NULL结尾的字符串。如果未出现错误，返回空字符串。

