

mysql-review

知识点

1. 数据库概念
2. 数据管理的三个阶段及特点
3. 数据库系统结构图
4. 三个世界,两个抽象步骤
5. ER图
6. 三个数据模型
7. 完整性约束三个条件
8. 范式(Normal Form)
9. ACID原理
10. CAP原理

MySQL

1. 数据库
2. 表
3. 数据
4. 查询
 - 单表查询
 - 多表查询
 - 复杂查询
5. 视图
 - 不能更新视图的条件
6. 索引
 - 索引设计的原则
7. 权限: 创建用户、授权、收权
8. 触发器
9. 存储过程、存储函数
 - 存储过程
 - 存储函数

知识点

1. 数据库概念

- 数据(信息):

数据(信息)是数据库中存储的基本对象,是关于现实世界事物的存在方式或运动状态反应的描述

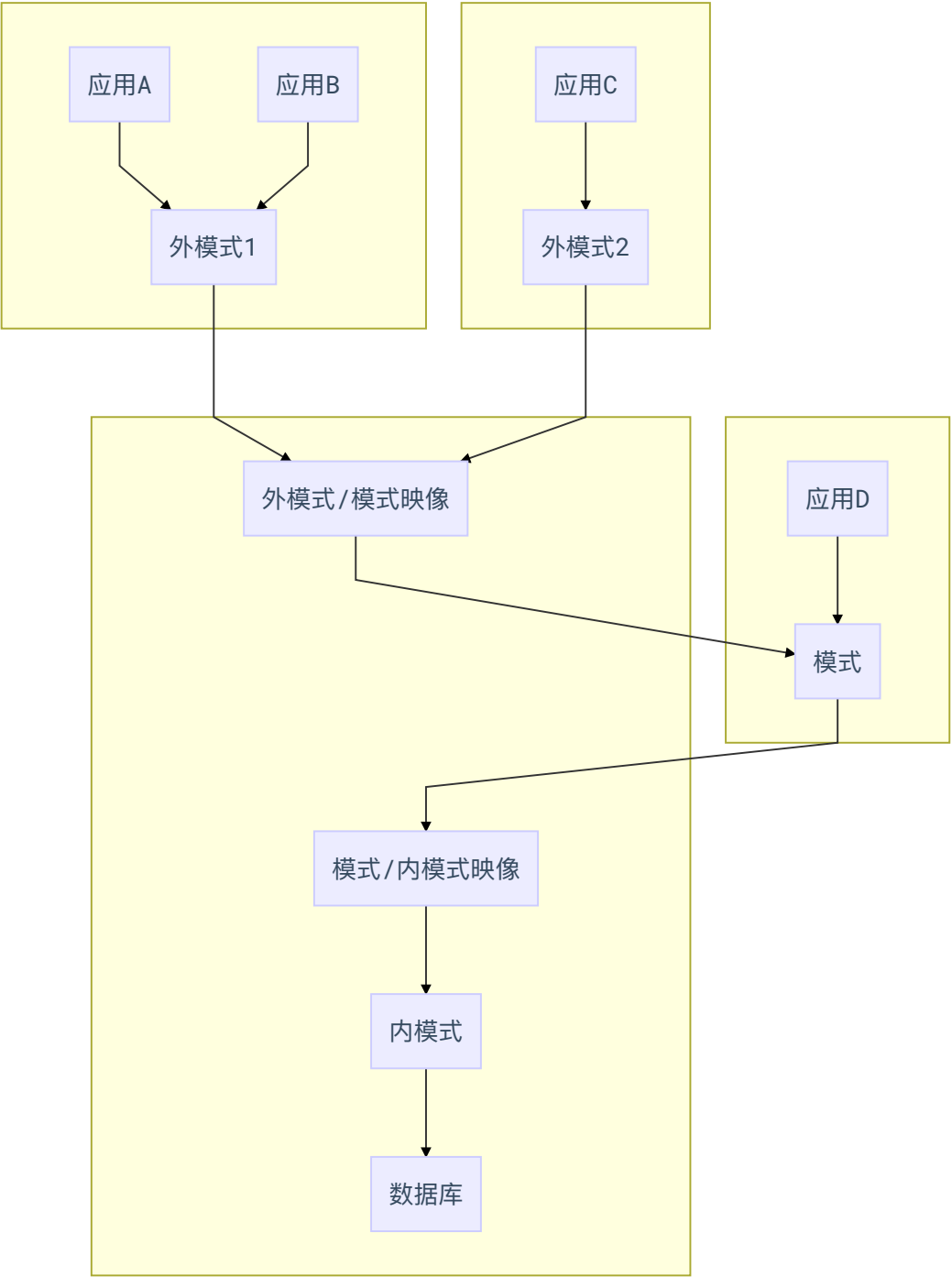
- 数据库：
是一个长期存储在计算机内的、有组织的、可共享的、统一管理的数据集合
- 数据库系统：
数据库系统是指在计算机系统中引入数据库后的系统
- ◦ 组成：
数据库、软件系统和用户

2. 数据管理的三个阶段及特点

| 人工管理数据阶段 | 文件管理数据阶段 | 数据库管理阶段 |
|----------------------|-------------------|----------------|
| (a) 数据一般不保存 | (a) 数据可长期保存 | (a) 数据结构化 |
| (b) 没有专门的软件对数据进行统一管理 | (b) 由文件系统对数据进行管理 | (b) 数据共享性高、冗余小 |
| (c) 数据无法共享 | (c) 文件可多样化组织 | (c) 数据独立性较高 |
| (d) 数据不具有独立性 | (d) 程序与数据间有一定的独立性 | (d) 有统一的数据控制功能 |

| 文件管理与人工管理阶段比较未解决的根本性问题 |
|------------------------|
| 1. 数据冗余度较大，数据一致性往往难以保证 |
| 2. 数据间的联系较弱 |
| 3. 数据独立性差 |
| 4. 缺乏对数据的统一控制 |

3. 数据库系统结构图



| 结构与优点 | |
|-------|-------------------|
| 三级模式 | 概念模式、外模式、内模式 |
| 二级映像 | 外模式/模式映像、模式/内模式映像 |
| 优点 | (a) 保证了数据的独立性 |
| | (b) 有利于实现数据共享 |
| | (c) 减轻了程序设计者的编程负担 |
| | (d) 提高了数据安全性 |

4. 三个世界,两个抽象步骤

- 客观世界
- 信息世界
- 计算机世界

- ①将现实世界中客观对象抽象为概念模型。
- ②将概念模型抽象为某个DBMS所控制的数据模型。

5. ER图

- **实体(entity):** 用矩形表示，矩形框内写明实体名

即数据模型中的数据对象

每个实体都有自己的实体成员(entity member)或者说实体对象(entity instance)

- **属性(attribute):** 用椭圆形或圆角矩形表示，并用无向边将其与相应的实体连接起来；多值属性由双线连接；主属性名称下加下划线

即数据对象所具有的属性

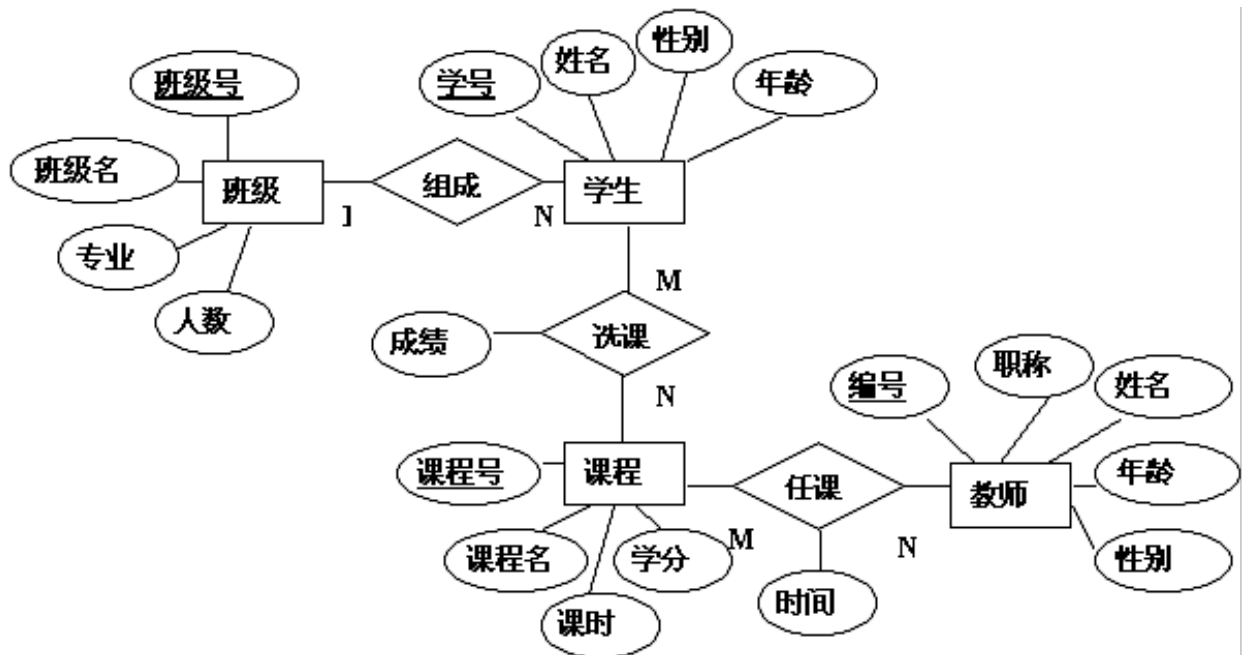
属性分为唯一属性（unique attribute）和非唯一属性，唯一属性指的是唯一可用来标识该实体实例或者成员的属性，用下划线表示，一般来讲实体都至少有一个唯一属性

- **关系(relationship):** 用菱形表示，菱形框内写明联系名，并用无向边分别与有关实体连接起

来，同时在无向边旁标上联系的类型

表现数据对象与数据对象之间的联系

- **1对1 (1:1)**：1对1关系是指对于实体集A与实体集B，A中的每一个实体至多与B中一个实体有关系；反之，在实体集B中的每个实体至多与实体集A中一个实体有关系。
- **1对多 (1:N)**：1对多关系是指实体集A与实体集B中至少有N(N>0)个实体有关系；并且实体集B中每一个实体至多与实体集A中一个实体有关系。
- **多对多 (M:N)**：多对多关系是指实体集A中的每一个实体与实体集B中至少有M(M>0)个实体有关系，并且实体集B中的每一个实体与实体集A中的至少N(N>0)个实体有关系



6. 三个数据模型

- 层次模型：
用树形结构表示实体及其之间的联系 (可以直接方便地表示一对一和一对多联系)
- 网状模型：
用网状结构表示实体及其之间联系的模型 (可以直接表示多对多联系，但其中的节点间关系更加复杂)
- 关系模型：
用关系描述实体间的联系的数据模型 (关系、元组、属性、域、分量、主码、关系模式)

7. 完整性约束三个条件

- 实体完整性规则

指关系（所谓的关系就是表）的主码不能取空值

- 参照完整性规则

指参照关系中每个元素的外码要么为空(NULL)，要么等于被参照关系中的某个元素的主码

- 用户自定义完整性规则

指对关系中每个属性的取值作一个限制（或称为约束）的具体定义

8. 范式(Normal Form)

- 非范式：

没有除去数据重复的表格

- 第一范式：

如果关系模式R中每个属性值都是一个不可分解的数据项，则称该关系模式满足第一范式，简称1NF，记为 $R \in 1NF$

将表格分割为单纯的二元表格，即一栏中只有一个项目，每一列都是不可分割的基本数据项

- 第二范式：

如果一个关系模式 $R \in 1NF$ ，且它的所有非主属性都完全函数依赖于R的码，则 $R \in 2NF$

按照通过可识别数据的键来确定其他列值的原则分割表格。这样，通过主键确定其他列的数值。 函数依赖：通过某一列的值确定其他列的数值的原则

- 第三范式：

如果一个关系模式 $R \in 2NF$ ，且所有的非主属性都不传递函数依赖于码，则 $R \in 3NF$

按照只能由主键确定其他列值的原则分割的表格。在关系数据库的函数依赖中,通过某一列的值间接确定其他列的值,称之为传递依赖。第三范式是去除传递依赖而分割表格得到的

9. ACID原理

- **A 原子(Atomicity)** 事务原子性

指每个事务都必须被看作是一个不可分割的单元

- **C 一致(Consistency)** 插入一张表数据，会影响其它(索引/其它表)等一致

一致性属性保证了数据库从不返回一个未处理完的事务

- **I 隔离性(Isolation)** 事务独立，封闭；隔离性强度

指每个事务在它自己的空间发生，和其他发生在系统中的事务隔离，而且事务的结果只有在它完全被执行时才能看到

- **D 持久性(Durability)** 数据永存

指即使系统崩溃，一个提交的事务仍然存在

10. CAP原理

- **Consistency (一致性)**，数据一致更新，所有数据变动都是同步的。
- **Availability (可用性)**，好的响应性能。
- **Partition tolerance (分区容错性)** 可靠性。

- CA 系统是要求高可用并且实时一致性。单点数据库是符合这种架构的，例如超市收银系统，图书管理系统。
- AP 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。例如博客系统。
- CP 系统是要求满足一致性，分区容忍性，通常性能不是特别高。例如火车售票系统。

MySQL

1. 数据库

```
1  /* 登录 (DOS环境) */
2  mysql -u root -p123456
3
4  /* 创建 */
5  create database school;
6
7  /* 查看 */
8  show databases;
9
10 /* 备份 (DOS环境) */
11 mysqldump -u root -p --databases school > d:\1108.sql
12
13 /* 删除 */
14 drop database school;
15
16 /* 还原 (DOS环境) */
17 mysql -u root -p<d:\1108.sql
```

2. 表

```
1  /* 创建student表 */
2  create table student(
3      sno char(10) primary key,
4      sname varchar(6) not null,
5      sgender enum('m','f') default 'f',
6      saddress varchar(10),
7      sqq char(10) unique
8  );
9
10 /* 创建course表 */
11 create table course(
12     cno char(6) primary key,
13     cname char(3) not null,
14     ccredit tinyint default 3
15 );
16
17 /* 创建sc表 */
18 create table sc(
19     sno int primary key auto_increment,
20     sno char(10),
21     cno char(6),
22     grade float(4,2),
23     constraint fk_1 foreign key (sno) references student(sno),
24     constraint fk_2 foreign key (cno) references
25     course(cno)
26 );
27
```



```

28  /* 查看student表结构 */
29  desc student;
30
31  /* 在student表里增加semail字段 */
32  alter table student add semail varchar(20);
33
34  /* 备份school数据库(DOS环境) */
35  mysqldump -u root -p --databases school > d:\1109.sql
36
37  /* 删除student表 */
38  alter table sc drop foreign key fk_1;
39  drop table student;
40
41  /* 还原数据库 */
42  mysql -u root -p < d:\1109.sql

```

3. 数据

```

1  /* 向student表插入数据 */
2  insert into student(sno,sname,sgender,saddress,sqq) values
3      ('2018001','tom','m','chn','12345'),
4      ('2018002','jim','m','usa',''),
5      ('2018003','mary','f','chn','54321')
6  );
7
8  /* 修改cname的数据长度为6 */
9  alter table course change cname cname char(6) not null;
10
11 /* 向course表插入数据 */
12 insert into course(cno,cname,ccredit) values
13     ('c001','math',3),
14     ('c002','db',4),
15     ('c003','os',2)
16 );
17
18 /* 向sc表插入数据 */
19 insert into sc(sno,cno,grade) values
20     ('2018001','c001',88),
21     ('2018001','c002',89),
22     ('2018001','c003',90),
23     ('2018002','c002',66),
24     ('2018003','c001',70)
25 );
26
27 /* 修改tom的数学成绩为70.6分 */
28 update sc set grade=70.6
29 where sno in (select sno from student where sname='tom')
30 and cno = (select cno from course where cname='math');

```

```

31
32  /* 增加jim的数学成绩为60.8分 */
33  insert into sc(sno,cno,grade) values('2018002','c001',60.8);
34
35  /* 删除mary的数学成绩 */
36  delete from sc where sno='2018003' and cno='c001';
37
38  /* 备份数据库 */
39  mysqldump -u root -p --databases school > /home/1112.sql
40
41  /* 删除tom的学生信息 */
42  delete from sc where sno='2018001';
43  delete from student where sno='2018001';

```

4.查询

单表查询

#

```

1  /* 查询每个雇员的所有数据 */
2  select * from employees;
3
4  /*查询每个雇员的姓名、电话和地址*/
5  select name,address,phonenumner from employees;
6
7  /*查询EmployeeID为000001的雇员的地址和电话*/
8  select address,phonenumner from employees where employeeid="000001";
9
10 /*查询Employees表中女雇员的地址和电话,
11 使用AS子句将结果中各列的标题分别指定为地址、电话*/
12 select address as "地址",phonenumner as "电话" from employees where sex = "0";
13
14 /*查询Employees表中员工的姓名和性别,
15 要求gender值为1时显示为'男',为0时显示为'女'*/
16 select name as "姓名",
17        case
18          when sex="1" then "男"
19          when sex="0" then "女"
20        end as gender
21 from employees;
22
23 /*计算每个员工的实际收入*/
24 select employeeid,income-outcome as "实际收入" from salary;
25
26 /*获得员工总数*/
27 select count(*) from employees;
28
29 /*找出所有王姓员工的部门号*/
30 select departmentid from employees where name like "王%";

```

```
31
32  /*找出所有收入在2000~3000元之间的员工号码*/
33  select employeeid from salary where income between 2000 and 3000;
```

多表查询

#

```
1  /* 查找在财务部工作的雇员的情况 */
2  select * from employees where departmentid = (select departmentid from
   departments where departmentname="财务部");
3
4  /* 查找研发部年龄不低于市场部所有雇员年龄的雇员的姓名 */
5  select name from employees where departmentid in (select departmentid from
   departments where departmentname="研发部") and birthday <= all(select birthday
   from employees where departmentid in(select departmentid from departments where
   departmentname="市场部"));
6
7  /* 查找比财务部所有的雇员收入都高的雇员的姓名 */
8  select name from employees where employeeid in (select employeeid from salary
   where income > all(select income from salary where employeeid in (select
   employeeid from employees where departmentid = (select departmentid from
   departments where departmentname="财务部"))));
```

复杂查询

#

```
1  ### 连接查询
2
3  /* 查询每个雇员的情况及其薪水的情况 */
4  select employees.*,salary.* from employees,salary where employees.salary where
   employees.employeeid=salary.employeeid;
5
6  /* 使用内连接的方法查询名字为“王林”的员工所在的部门 */
7  select departmentname from departments join employees on
   departments.departmentid=employees.departmentid where employees.name="王林";
8
9  /* 查找财务部收入在2000元以上的雇员姓名及其薪水详情 */
10 select name,income,outcome from employees,salary,departments where
   employees.employeeid=salary.employeeid and
   employees.departmentid=departments.departmentid and departmentname="财务部" and
   income>=2000;
11
12 ### GROUP BY、ORDER BY和LIMIT子句的使用
13
14 /* 查找Employees中男性和女性的人数 */
15 select sex,count(sex) from employees group by sex;
16
17 /* 查找员工数超过2人的部门名称和员工数量 */
```

```

18 select departmentname,count(*) as "人数" from employees,departments where
    employees.departmentid=departments.departmentid group by employees.departmentid
    having count(*)>2;
19
20 /* 将Employees表中的员工号码由大到小排列 */
21 select employeeid from employees order by employeeid desc;
22
23 /* 返回Employees表中的前5位员工的信息 */
24 select * from employees limit 5;

```

5. 视图

```

1  ### 创建视图
2
3  /* 创建YGGL数据库上的视图DS_VIEW, 视图包含Departments表的全部列 */
4  create view ds_view as select * from departments;
5
6  /* 创建YGGL数据库上的视图Employees_view, 视图包含员工号码、姓名和实际收入 */
7  create view employees_view(employeeid,name,realincome) as select
    employees.employeeid,name,income-outcome from employees,salary where
    employees.employeeid=salary.employeeid;
8
9  ### 查询视图
10
11 /* 从视图DS_VIEW中查询出部门号为3的部门名称 */
12 select departmentname from ds_view where departmentid = "3";
13
14 /* 从视图Employees_view查询出姓名为"王林"的员工的实际收入 */
15 select realincome from employees_view where name="王林";
16
17 ### 更新视图
18
19 /* 向视图DS_VIEW中插入一行数据: 6, 财务部, 财务管理 */
20 insert into ds_view values("6","财务部","财务管理");
21
22 /* 修改视图DS_VIEW, 将部门号为5的部门名称修改为"生产车间" */
23 update ds_view set departmentname="生产车间" where departmentid="5";
24
25 /* 修改视图Employees_view中号码为000001的雇员的姓名为"王浩" */
26 update employees_view set name="王浩" where employeeid="000001";
27
28 ### 删除视图
29 /* 删除视图DS_VIEW中部门号为"1"的数据 */
30 delete from ds_view where departmentid="1";
31
32 /* 删除视图DS_VIEW */

```

不能更新视图的条件

#

不能更新视图的条件

- 1 视图包含`sum()`、`count()`、`MAX()`和`MIN()`等函数
- 2 视图包含`UNION`、`UNION ALL`、`DISTINCT`、`GROUP BY`和`HAVING`等关键字
- 3 常量视图
- 4 视图中的`SELECT`中包含子查询
- 5 由不可更新的视图导出的视图
- 6 视图对应的表上存在没有默认值的列，而且该列没有包含在视图里

6. 索引

```
1  ### 使用Create Index语句创建索引
2
3  /* 对YGGL数据库的Employees表中的DepartmentID列建立索引 */
4  create index depart_ind on employees(departmentid);
5
6  /*在Employees表的Name列和Address列上建立复合索引 */
7  create index ad_ind on employees(name,address);
8
9  /* 对Departments表上的DepartmentName列建立唯一性索引 */
10 create unique index dep_ind on departments(departmentname);
11
12 ### 使用Alter Table语句向表中添加索引
13 /* 向Employees表中的出生日期列添加一个唯一性索引，姓名列和性别列上添加一个复合索引 */
14 alter table employees add unique index date_ind(birthday),add index
    na_ind(name,sex);
15
16 /* 假设Departments表中没有主键，使用Alter Table语句将DepartmentID列设为主键 */
17 alter table employees add primary key(departmentid);
18
19 ### 在创建表时创建索引
20
21 /* 创建与Departments表相同结构的表Departments1，将DepartmentName设为主
    键,DepartmentID建立一个索引 */
```

```

22  create table departments1
23  (
24      departmentid char(3),
25      departmentname char(20),
26      note text,
27      primary key(departmentname),
28      index did_ind(departmentid)
29  );
30
31  ### 删除索引
32
33  /* 使用Drop Index语句删除表Employees上的索引 */
34  drop index depart_ind on employees;
35
36  /* 使用Alter Table语句删除Departments上的主键和索引 */
37  alter table departments drop primary key,drop index dep_ind;

```

索引设计的原则

#

索引设计的原则

选择唯一性索引

重复值越少，可以添加索引

为经常需要排序、分组和联合操作的字段建立索引

为常作为查询条件的字段建立索引

限制索引的数目

尽量使用数据量少的索引

尽量使用前缀来索引

删除不在使用或者很少使用的索引

7. 权限：创建用户、授权、收权

```

1  /* 使用root用户创建exam1用户，初始密码设置为123456 */
2  create user "exam1"@"localhost" identified by "123456";
3
4  /* 授权exam1用户对所有数据库拥有Select、Create、Drop和Grant权限 */
5  grant select,create,drop on *.* to "exam1"@"localhost" with grant option;
6

```

```

7  /* 查看exam1的权限 */
8  show grants for "exam1"@"localhost";
9  select * from mysql.user where user="exam1" and host="localhost";
10
11 /* 创建exam2, 该用户没有初始密码 */
12 create user "exam2"@"localhost";
13
14 /* 用exam2登录, 将其密码设置为787878 */
15 set password=password("787878");
16
17 /* 用exam1登录, 为exam2授权为YGG数据库的Employees表查看Name字段的权限 */
18 grant select(name) on yggl.employees to "exam2"@"localhost";
19
20 /* 用root登录, 收回exam1和exam2的所有权限 */
21 revoke all privileges,grant option from
    "exam1"@"localhost","exam2"@"localhost";

```

8. 触发器

```

1  /* 创建触发器, 在Employees表中删除员工信息的同时将Salary表中该员工的信息删除, 以确保数据完
    整性 */
2
3  create trigger delete_em after delete
4      on employees for each row
5      delete from salary
6          where employeeid=old.employeeid;
7
8  /* 假设Department2表和Department表的结构和内容都相同, 在Departments上创建一个触发器, 如
    果添加一个新的部门, 该部门也会添加到Departments2表中 */
9
10 delimiter $$
11 create trigger departments_ins after insert
12     on departments for each row
13     begin
14         insert into departments2
15         values(new.departmentid,new.departmentname,new.note);
16     end$$
17 delimiter ;
18
19 /* 当修改表Employees时, 若将Employees表中员工的工作时间增加1年, 则将收入增加500, 增加2年则
    增加1000, 依次增加。若工作时间减少则无变化 */
20
21 delimiter $$
22 create trigger add_salary after update
23     on employees for each row
24     begin
25         declare years integer;
26         set years=new.workyear-old.workyear;

```

```

26         if years>0 then
27             update salary set income=income+500*years
28                 where employeeid=new.employeeid;
29         end if;
30     end$$
31 delimiter ;
32

```

9. 存储过程、存储函数

存储过程

#

```

1  /* 创建存储过程，使用Employees表中的员工人数来初始化一个局部变量，并调用这个存储过程 */
2
3  delimiter $$
4  create procedure test(out number1 int)
5  begin
6      declare number2 int;
7      set number2=(select count(*) from employees);
8      set number1=number2;
9  end$$
10 delimiter;
11 call test(@number);
12 select @number;
13
14 /* 创建存储过程，比较两个员工的实际收入，若前者比后者高就输出0，否则输出1 */
15
16 delimiter $$
17 create procedure compa(in id1 char(6),in ide2 char(6),out bj int)
18 begin
19     declare sr1,sr2 float(8);
20     select income-outcome into sr1 from salary where employeeid=id1;
21     select income-outcome into sr2 from salary where employeeid=id2;
22     if id1>id2 then set bj=0;
23     else set bj=1;
24     end if;
25 end $$
26 delimiter;
27 call compa('00001','108991',@bj);
28 select @bj;
29
30 /* 创建存储过程，使用游标确定一个员工的实际收入是否排在前三名。结果为TRUE表示是，结果为FALSE
    表示否 */
31
32 delimiter $$
33 create procedure top_three(in em_id char(6),out ok boolean)
34 begin
35     declare x_em_id char(6);

```



```

36     declare act_in,seq int;
37     declare found boolean;
38     declare salary_dis cursor for
39         select employeeid,income-outcome from salary order by 2 desc;
40     declare continue handle for not found set found=false;
41     set seq=0;
42     set found=true;
43     set ok=false;
44     open salary_dis;
45     fetch salary_dis into x_em_id,act_in;
46     while found and seq<3 and ok=false do
47         set seq=seq+1;
48         if x_em_id=em_id then set ok=true;
49         end if;
50         fetch salary_dis into x_em_id,act_in;
51     end while;
52     close salary_dis;
53 end $$
54 delimiter;

```

存储函数

#

```

1  /* 创建一个存储函数，返回员工的总人数 */
2
3  create function em_num()
4      returns int
5      return (select count(*) from employees);
6  select em_num();
7
8  /* 创建一个存储函数，删除在Salary表中有但在Employees表中不存在的员工号。若在Employees表中
   存在返回FALSE，若不存在则删除该员工号并返回TRUE */
9
10 delimiter $$
11 create function delete_em(em_id char(6))
12     returns boolean
13 begin
14     declare em_name char(10);
15     select name into em_name from employees where employeeid=em_id;
16     if em_name is null then
17         delete from salary where employeeid=em_id;
18         return true;
19     else
20         return false;
21     end if;
22 end $$
23 delimiter ;
24 select delete_em('000001');

```