MySQL复习

MySQL概念

Mysql是关系型数据库，它是开源、关系型的。

与Oracle区别：

1. Mysql是开源、Oracle是闭源
2. Mysql默认是自动提交，Oracle需要手动提交
3. Oracle功能更强大，更安全
4. Mysql是表级锁（InnoDB除外），Oracle是行级锁，并发性更高

关系型数据库

关系模型就是指二维表格模型，因而一个关系型数据库就是由二维表及其之间的联系组成的一个数据库系统。一行代表一条记录，一条记录有多个字段。

三大范式

关系型数据库必须满足三大范式：

1. 属性不可再分，字段保持原子性

   1NF，是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。第一范式是对关系模式的最起码的要求。不满足第一范式的数据库模式不能称为关系数据库。

2. 不能出现部分依赖，增加单列关键字（表中的每条记录必须被唯一的区分，也就是必须设置主键）

   2NF，在满足1NF的基础上，要求表中的每条记录必须被唯一的区分。同时要求，实体属性应该完全依赖于主关键字。而不能是对主关键字形成部分函数依赖。

3. 不能出现传递依赖（减少冗余，建立表关系，通过主键关联）

   3NF，满足第二范式的基础上，要求不能出现传递依赖，也就是不能出现属性依赖于非主属性的.上面的教室就依赖于班级，而班级依赖于主键ID。就是传递依赖。应该将传递依赖的数据单独建立二维表，保存数据

非关系型数据库

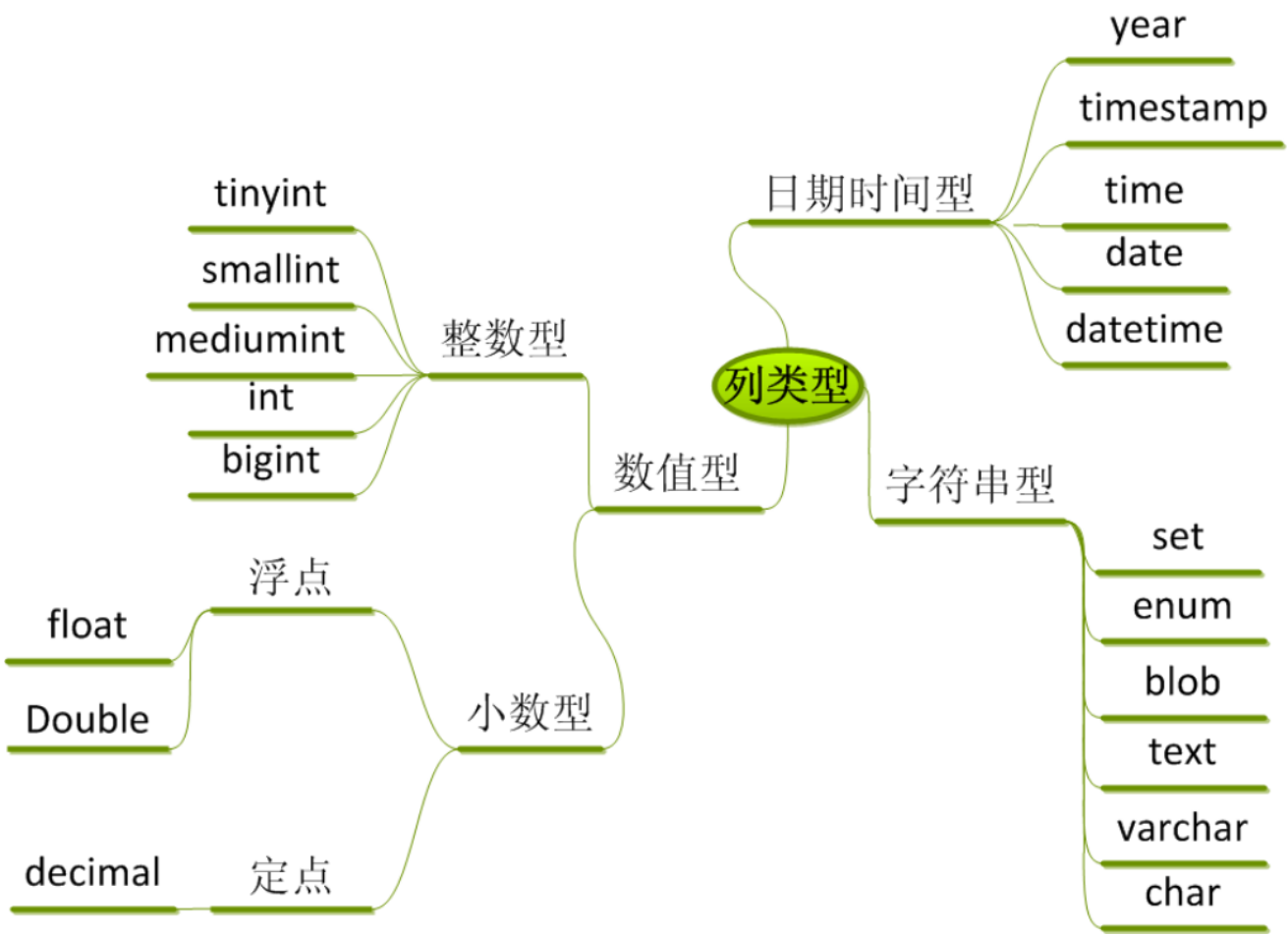
非关系型数据库不用预先建立表结构，数据是无序、不规则的，速度快，事务支持弱。常见的非关系型数据库有MongoDB、Redis、Sqlite等

数据库连接和使用

通过使用以下命令，比如：

```
mysql -h localhost -p 3306 -u root -p
use review;
```

字段数据类型



- 数值型
  1. 整数型

     TINYINT[(M)][UNSIGNED] [ZEROFILL]

     SMALLINT[(M)][UNSIGNED] [ZEROFILL]

     MEDIUMINT[(M)][UNSIGNED] [ZEROFILL]

     INT[(M)][UNSIGNED] [ZEROFILL]

     BIGINT(M) [ZEROFILL]

     也可以使用 INTEGERBIGINT[(M)][UNSIGNED] [ZEROFILL]T其中M，表示显示宽度，显示宽度不限制数值的范围。配合zerofill来使用，可以在小于显示宽度的位数前增加0.zerofill自动为unsignedUnsigned表示无符号，只表示正数。

  2. 小数型
     - float

       FLOAT[(M,D)][UNSIGNED] [ZEROFILL]
     - double

       DOUBLE[(M,D)][UNSIGNED] [ZEROFILL]

       其中M表示总的位数，D表示小数位数。

此M,D可以控制保存的范围。Float(10,2) -99999999.99 到 99999999.99如果省略M,D会根据计算机硬件进行处理。单精度，M大约为7左右。而双精度，M大约为15左右

    - decimal

      DECIMAL[(M[,D])][UNSIGNED] [ZEROFILL]

      其中M表示总的位数，D表示小数位数。此M,D可以控制保存的范围。M，D省略，默认为10,0；

- 字符串型
  1. char(M)

     Char定长字符串，保存时如果字符串长度不够，则后边补足空字符串；但是在读取到数据是，会截取后边所有的字符串。因此如果真实数据存在左边空格，则需要注意。

  2. varchar(M)

     Varchar，变长字符串。在保存字符串时，同时保存该字符串的长度，小于255采用一个字节保存，否则采用二个字节保存。不会像char一样截取空格

  3. tinytext、text、mediumtext、longtext
  4. enum

     ENUM('value1','value2',...)

     枚举值应该在值列表内，允许使用下标方式标识。1标识第一个元素，逐个递增

  5. set

     Set('value1','value2',...);

     表示可以选择可用值的0个或多个组合

  6. binary、varbinary、blob

     如果需要保存一个二进制文件内容的话，应该保存成这些类型，例如图片内容

- 日期时间型
  1. datetime

     '1000-01-01 00:00:00'到'9999-12-31 23:59:59'

  2. timestamp

     '1970-01-01 00:00:00'到'2038-01-19 03:14:07'

  3. date
  4. time
  5. year

SQL

SQL(Structured Query Language)：结构化查询语言。

1. DDL

   结构操作语言（数据定义语言，DDL，DataDefinitionLanguage ）：

   用于创建、修改、和删除数据库内的数据结构，如：1：创建和删除数据库(CREATE DATABASE || DROP DATABASE)；2：创建、修改、重命名、删除表(CREATE TABLE || ALTER TABLE|| RENAME TABLE||DROP TABLE)；3：创建和删除索引(CREATE INDEX || DROP INDEX)

2. DML

   数据操作语言（DML，DataManipulationLanguage ）（DQL+DML）：

   - DQL（数据查询语言）：

     从数据库中的一个或多个表中查询数据(SELECT)

   - DML（数据库管理语言）：

     修改数据库中的数据，包括插入(INSERT)、更新(UPDATE)和删除(DELETE)

3. DCL

   数据库管理语言（数据库控制语言，DCL，DataBaseControlLanguage）：

   用于对数据库的访问，如：1：给用户授予访问权限（GRANT）;2：取消用户访问权限（REVOKE）

DDL

1. 创建数据库

   语法：

   create database [if not exists] db_name [数据库选项];

   比如：

   ```
   create database if not exists review character set utf8 collate utf8_general_ci;
   ```

2. 删除数据库

   语法：

   drop database [if exists] db_name;

   比如：

   ```
   drop database if exists review;
   ```

3. 创建表

语法：

create [if not exists] table tb_name (列定义，[列定义...]) [表选项];

比如：

```
create table student(

    no varchar(3) PRIMARY key comment '学号',

    name varchar(4) not null comment '姓名',

    sex varchar(2) not null comment '性别',

    birthday datetime not null comment '生日',

    class_name varchar(5) not null comment '班级名称'

) ENGINE=INNODB DEFAULT charset=utf8 comment '学生信息表';
```

4. 修改表结构

语法：

alter table tb_name change 字段名 字段定义;

alter table tb_name modify 字段定义;

比如：

```
alter table student change no student_no varchar(3) comment '学号';
alter table student MODIFY student_no varchar(4) comment '学号';
```

5. 删除表字段

语法：

alter table tb_name drop 字段;

比如：

```
alter table student drop class_name;
```

6. 增加表字段

语法：

alter table tb_name add 字段定义;

比如:

```
alter table student add class_name varchar(5) not null comment '班级名称';
```

7. 重命名表

   语法:

   rename table tb_name to 新的表名;

```
rename table student to student_rename;
```

8. 创建索引

   语法:

   create [unique|fulltext] index 索引名字 on tb_name(字段名, [字段名...]);

   比如:

```
create index idx_student_name on student(name);
```

9. 删除索引

   语法:

   drop index 索引名字 on tb_name;

```
drop index idx_student_name on student;
```

DCL

1. grant(授权)

   语法:

   create user '用户名'@'主机地址' identified by '密码';

   grant 授权类型, [授权类型...] on 数据库.表名 to 用户名@'主机地址'

   比如:

```
create user 'qhcs_dev'@'localhost' identified by '密码';
-- 指定权限类型
grant select,update on review.student to 'qhcs_dev'@'localhost';
-- 所有权限类型
grant all on review.student to 'qhcs_dev'@'localhost';
-- 所有数据库
grant all on *.* to 'qhcs_dev'@'localhost';
-- 指定数据库下所有表
grant all on review.* to 'qhcs_dev'@'localhost';
-- 任意访问地址
grant all on review.* to 'qhcs_dev'@'%' identified by '123456';
```

2. revoke(取消授权)

   语法:

   revoke 授权类型, [授权类型...] on 数据库.表名 from 用户名@'主机地址'

   比如:

```
revoke all on review.* from 'qhcs_dev'@'%';
```

DML

1. 插入表数据

   语法:

   insert [ignore] [into] 表名[(需要插入字段, ...)] values (字段值, ...) ;

   insert [ignore] [into] 表名[(需要插入字段, ...)] select (相同的字段, ...)] from other_tb_name;

   insert [ignore] [into] 表名[(需要插入字段, ...)] on duplicate key update 字段=值, [字段=值...];

   比如:

```
-- 插入所有字段
insert into student values ('122','小茗同学','2','1992-03-26', '1801');
-- 指定需要插入的字段
insert into student(student_no, name, sex, birthday, class_name) values ('122','小茗同
学','2','1992-03-26', '1801');
-- 忽略错误
insert ignore into student values ('122222','小茗同学','2','1992-03-26', '1801');
-- 从另外的表查询出来的数据作为插入数据
insert into student select student_no, name, sex, birthday, class_name from student_copy;
-- 当插入的数据的主键已经存在更新某个字段的值
insert into student values ('122','小茗同学','2','1992-03-26', '1801') on duplicate key
update name = '哇哈哈';
```

2. 修改表数据

   语法：

   update [ignore] 表名 set 字段=值, [字段=值, ...] [where 条件] [order by 字段];

   比如：

```
-- 修改数据
update student set name = "康师傅", class_name = '7894' where student_no = 132;
```

   注意：除非确定修改全部数据，否则一定加上where条件！

3. 删除表数据

   语法：

   delete [ignore] from 表名 [where 条件] [order by 字段];

   比如：

```
-- 删除数据
delete from student where student_no = 123;
```

4. truncate、delete、drop区别

   TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同：二者均删除表中的全部行。但 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少。 DELETE 语句每次删除一行，并在事务日志中为所删除的每行记录一项。TRUNCATE TABLE 通过释放存储表数据所用的数据页来删除数据，并且只在事务日志中记录页的释放。

   TRUNCATE,DELETE,DROP放在一起比较：

   TRUNCATE TABLE：删除内容、释放空间但不删除定义。

   DELETE TABLE:删除内容不删除定义，不释放空间。

   DROP TABLE：删除内容和定义，释放空间。

```
-- truncate
truncate student_copy;
-- delete
delete from student_copy;
-- drop
drop table student_copy;
```

DQL

一般应用查询操作比修改操作多，也就是说读操作比写操作频繁，所以查询语句写的好不好、性能高不高直接影响整个应用的响应速度。

语法：

```
SELECT
```

```
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [MAX_STATEMENT_TIME = N]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

下面以这几张表演示：

```
DROP TABLE IF EXISTS `course`;
CREATE TABLE `course` (
  `course_no` varchar(5) PRIMARY key comment '课程编号',
  `name` varchar(10) NOT NULL comment '课程名称',
  `teacher_no` varchar(10) NOT NULL comment '教师编号'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 comment '课程表';

-- ----------------------------
-- Records of course
-- ----------------------------
INSERT INTO `course` VALUES ('3-105', '计算机导论', '825');
INSERT INTO `course` VALUES ('3-245', '操作系统', '804');
INSERT INTO `course` VALUES ('6-166', '数据电路', '856');
INSERT INTO `course` VALUES ('9-888', '高等数学', '100');

-- ----------------------------
-- Table structure for score
-- ----------------------------
DROP TABLE IF EXISTS `score`;
CREATE TABLE `score` (
  `student_no` varchar(3) NOT NULL comment '学号',
  `course_no` varchar(5) NOT NULL comment '课程编号',
  `degree` decimal(10,1) NOT NULL comment '分数'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 comment '成绩表';

-- ----------------------------
```

```
-- Records of score
-- ----------------------------
INSERT INTO `score` VALUES ('103', '3-245', '86.0');
INSERT INTO `score` VALUES ('105', '3-245', '75.0');
INSERT INTO `score` VALUES ('109', '3-245', '68.0');
INSERT INTO `score` VALUES ('103', '3-105', '92.0');
INSERT INTO `score` VALUES ('105', '3-105', '88.0');
INSERT INTO `score` VALUES ('109', '3-105', '76.0');
INSERT INTO `score` VALUES ('101', '3-105', '64.0');
INSERT INTO `score` VALUES ('107', '3-105', '91.0');
INSERT INTO `score` VALUES ('108', '3-105', '78.0');
INSERT INTO `score` VALUES ('101', '6-166', '85.0');
INSERT INTO `score` VALUES ('107', '6-106', '79.0');
INSERT INTO `score` VALUES ('108', '6-166', '81.0');

-- ----------------------------
-- Table structure for student
-- ----------------------------
DROP TABLE IF EXISTS `student`;
create table student(
    no varchar(3) PRIMARY key comment '学号',
    name varchar(4) not null comment '姓名',
    sex varchar(2) not null comment '性别',
    birthday datetime not null comment '生日',
    class_name varchar(5) not null comment '班级名称'
) ENGINE=INNODB DEFAULT charset=utf8 comment '学生信息表';

-- ----------------------------
-- Records of student
-- ----------------------------
INSERT INTO `student` VALUES ('108', '曾华', '男', '1977-09-01 00:00:00', '95033');
INSERT INTO `student` VALUES ('105', '匡明', '男', '1975-10-02 00:00:00', '95031');
INSERT INTO `student` VALUES ('107', '王丽', '女', '1976-01-23 00:00:00', '95033');
INSERT INTO `student` VALUES ('101', '李军', '男', '1976-02-20 00:00:00', '95033');
INSERT INTO `student` VALUES ('109', '王芳', '女', '1975-02-10 00:00:00', '95031');
INSERT INTO `student` VALUES ('103', '陆君', '男', '1974-06-03 00:00:00', '95031');

-- ----------------------------
-- Table structure for teacher
-- ----------------------------
DROP TABLE IF EXISTS `teacher`;
CREATE TABLE `teacher` (
  `teacher_no` varchar(3) PRIMARY key comment '教师编号',
  `name` varchar(4) NOT NULL comment '姓名',
  `sex` varchar(2) NOT NULL comment '性别',
  `birthday` datetime NOT NULL comment '生日',
  `prof` varchar(6) DEFAULT NULL comment '职位',
  `depart` varchar(10) NOT NULL comment '院系'
)  ENGINE=INNODB DEFAULT charset=utf8 comment '教师信息表';

-- ----------------------------
-- Records of teacher
-- ----------------------------
```

```
INSERT INTO `teacher` VALUES ('804', '李诚', '男', '1958-12-02 00:00:00', '副教授', '计算机系');
INSERT INTO `teacher` VALUES ('856', '张旭', '男', '1969-03-12 00:00:00', '讲师', '电子工程系');
-- INSERT INTO `teacher` VALUES ('856', '张旭', '男', '1969-03-12 00:00:00', '讲师', '电子工程
系');
INSERT INTO `teacher` VALUES ('825', '王萍', '女', '1972-05-05 00:00:00', '助教', '计算机系');
INSERT INTO `teacher` VALUES ('831', '刘冰', '女', '1977-08-14 00:00:00', '助教', '电子工程系');
```

1. 简单查询

```
select * from student;
select * from student order by student_no desc;
select * from student where name = '小茗同学';
```

2. 连接查询

笛卡尔积

两张表的字段个数相加，行数相乘，得到二维表，根据条件筛选。

笛卡儿积（列数相加，行数相乘）

teacher_no	name	sex	birthday	prof	depart	course_no	name	teacher_no
804	李诚	男	1958/12/2 0:00	副教授	计算机系	3-105	计算机导	825
804	李诚	男	1958/12/3 0:00	副教授	计算机系	3-245	操作系统	804
804	李诚	男	1958/12/4 0:00	副教授	计算机系	6-166	数据电路	856
804	李诚	男	1958/12/5 0:00	副教授	计算机系	9-888	高等数学	100
825	王萍	女	1972/5/5 0:00	助教	计算机系	3-105	计算机导	825
825	王萍	女	1972/5/5 0:00	助教	计算机系	3-245	操作系统	804
825	王萍	女	1972/5/5 0:00	助教	计算机系	6-166	数据电路	856
825	王萍	女	1972/5/5 0:00	助教	计算机系	9-888	高等数学	100
831	刘冰	女	1977/8/14 0:00	助教	电子工程系	3-105	计算机导	825
831	刘冰	女	1977/8/14 0:00	助教	电子工程系	3-245	操作系统	804
831	刘冰	女	1977/8/14 0:00	助教	电子工程系	6-166	数据电路	856
831	刘冰	女	1977/8/14 0:00	助教	电子工程系	9-888	高等数学	100
856	张旭	男	1969/3/12 0:00	讲师	电子工程系	3-105	计算机导	825
856	张旭	男	1969/3/12 0:00	讲师	电子工程系	3-245	操作系统	804
856	张旭	男	1969/3/12 0:00	讲师	电子工程系	6-166	数据电路	856
856	张旭	男	1969/3/12 0:00	讲师	电子工程系	9-888	高等数学	100

- 内连接

  获取两个表中字段匹配关系的记录，当条件在两个表中都成立的记录才会查询出来。

```
-- 内连接
select * from teacher t inner join course c on t.teacher_no = c.teacher_no;
-- 另外一种
select * from teacher t join course c on t.teacher_no = c.teacher_no;
-- 还有一种
select * from teacher t, course c where t.teacher_no = c.teacher_no;
```

teacher_no	name	sex	birthday	prof	depart	course_no	name1	teacher_no1
804	李诚	男	1958-12-02 00:00:00	副教授	计算机系	3-245	操作系统	804
825	王萍	女	1972-05-05 00:00:00	助教	计算机系	3-105	计算机导论	825
856	张旭	男	1969-03-12 00:00:00	讲师	电子工程系	6-166	数据电路	856

- 外连接

  1. 左外连接

     获取左表所有记录，即使右表没有对应匹配的记录。

```
-- 左外连接
select * from course c left join teacher t on c.teacher_no = t.teacher_no;
```

course_no	name	teacher_no	teacher_no1	name1	sex	birthday	prof	depart
3-245	操作系统	804	804	李诚	男	1958-12-02 00:00:00	副教授	计算机系
3-105	计算机导论	825	825	王萍	女	1972-05-05 00:00:00	助教	计算机系
6-166	数据电路	856	856	张旭	男	1969-03-12 00:00:00	讲师	电子工程系
9-888	高等数学	100	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)

  2. 右外连接

     与 LEFT JOIN 相反，用于获取右表所有记录，即使左表没有对应匹配的记录。

```
-- 右外连接
select * from course c right join teacher t on c.teacher_no = t.teacher_no;
```

course_no	name	teacher_no	teacher_no1	name1	sex	birthday	prof	depart
3-105	计算机导论	825	825	王萍	女	1972-05-05 00:00:00	助教	计算机系
3-245	操作系统	804	804	李诚	男	1958-12-02 00:00:00	副教授	计算机系
6-166	数据电路	856	856	张旭	男	1969-03-12 00:00:00	讲师	电子工程系
(Null)	(Null)	(Null)	831	刘冰	女	1977-08-14 00:00:00	助教	电子工程系

- 全连接

  1. UNION 操作符用于连接两个以上的 SELECT 语句的结果组合到一个结果集合中。多个 SELECT 语句会删除重复的数据。
  2. UNION ALL操作符用于连接两个以上的 SELECT 语句的结果组合到一个结果集合中。多个 SELECT 语句不会删除重复的数据。

```
-- 过滤重复数据
select course_no,name from course union select teacher_no,name from teacher;
-- 不过滤重复数据
select course_no,name from course union all select teacher_no,name from teacher;
```

3. 子查询

   语法：

   select * from tb_name where id = (select id from tb_name2 where id2 = 1);

比如：

```
-- 子查询，查询教师编号为825的教师所教的课程信息
select * from course where teacher_no = (select teacher_no from teacher where teacher_no =
825);
```

4. 分组查询

```
-- 分组查询，查询总分大于等于170分的学生信息和总分
select s.student_no, s.name, s.sex, sum(sc.degree) from student s left join score sc on
s.student_no = sc.student_no group by sc.student_no having sum(sc.degree) >= 170;
```

student_no	name	sex	sum(sc.degree)
103	陆君	男	178.0
107	王丽	女	170.0

5. 模糊查询

```
-- 模糊查询，查询姓王学生的信息
select * from student where name like '王%';
```

student_no	name	sex	birthday	class_name
107	王丽	女	1976-01-23 00:00:00	95033
109	王芳	女	1975-02-10 00:00:00	95031

6. 分页查询

```
-- 分页查询，查询课程信息，每页2条记录
select * from course limit 0,2;
-- limit m, n; m代表从哪条记录开始查询，n代表查询多少条记录
```

course_no	name	teacher_no
3-105	计算机导论	825
3-245	操作系统	804

7. 合并查询

1. UNION 操作符用于连接两个以上的 SELECT 语句的结果组合到一个结果集合中。多个 SELECT 语句会删除重复的数据。
2. UNION ALL操作符用于连接两个以上的 SELECT 语句的结果组合到一个结果集合中。多个 SELECT 语句不会删除重复的数据。

```
-- 过滤重复数据
select course_no,name from course union select teacher_no,name from teacher;
-- 不过滤重复数据
select course_no,name from course union all select teacher_no,name from teacher;
```

索引

索引是mysql提高查询速度的一种方式，它会为某个字段建立一个索引文件，当根据该字段查询数据时会先从索引文件查找数据的位置，然后把数据查询出来。它底层是通过BTree的算法建立索引。

索引类型

1. 普通索引
2. 唯一索引
3. 主键索引
4. 全文索引
5. 组合索引

```
-- 普通索引
create index idx_teacher_name on teacher(name);

-- 删除索引
drop index idx_teacher_name on teacher;

-- 唯一索引
create unique index idx_student_name on student(name);

-- 主键索引
alter table teacher add primary key(teacher_no);

-- 全文索引
create fulltext index idx_teacher_name on teacher(name);

-- 组合索引，把频繁查询的字段写在前面
create index idx_teacher_name_sex on teacher(name,sex);
```

索引应用场景

1. 较频繁的作为查询条件字段应该创建索引
2. 表数据比较多
3. 字段数据多样性

索引不适合的场景

1. 唯一性太差的字段不适合单独创建索引，即使频繁作为查询条件
2. 更新非常频繁的字段不适合创建索引
3. 不会出现在WHERE子句中字段不该创建索引

视图

视图是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。真实的表是持久化存在于磁盘上的。

视图操作

创建视图：create view 视图名 as select语句

修改视图：alter view 视图名 as select语句

删除视图：drop view 视图名

```
-- 创建视图
create view v_teacher_course as select t.teacher_no, t.name as teacher_name, c.name as
course_name from teacher t left join course c on t.teacher_no = c.teacher_no;
-- 修改视图
alter view v_teacher_course as select t.teacher_no, t.name, c.name as course_name from teacher t
left join course c on t.teacher_no = c.teacher_no;
-- 删除视图
drop view v_teacher_course;
```

视图作用

1. 隐藏真实表的结构和数据，只提供允许访问的数据，达到安全和权限控制的效果
2. 简化复杂查询，聚合不同表的数据，方便查询

触发器

触发器是指存放在数据库中，被隐含执行的存储过程,可以支持dml触发器，还支持基于系统事件(启动数据库,关闭数据库,登陆)和ddl操作建立触发器。

语法：

CREATE TRIGGER trigger_name trigger_time trigger_event　ON tbl_name FOR EACH ROW

BEGIN

sql语句...;

EDN;

说明:

trigger_time 可选 [after| before]

trigger_event 可选 [insert|update|delete]

```
-- 注意：OLD是指修改或删除前的那行数据，NEW是指新增后那行数据，可以通过OLD.col_name或者NEW.col_name获取某列数据

-- 删除后：
create TRIGGER trigger_update_student_count after DELETE on t_student for each row
BEGIN

delete from t_student_detail where student_no = OLD.student_no;
update t_class set student_count = student_count - 1 where id = OLD.class_id;

END

-- 新增后：

create TRIGGER trigger_insert_student after insert on t_student for each row
BEGIN

insert into t_student_detail(student_no,sex,phone_no,address) values
(NEW.student_no,'1','13645613378','广东深圳');
update t_class set student_count = student_count + 1 where id = NEW.class_id;

END
```

触发器使用场景

当需要根据某个表的操作insert/update/delete的前后自动执行某个操作，这个时候可以使用触发器。

存储过程

存储过程是预编译的sql语句块，一般用于一些完成固定的功能，比如转账、计算费用等。

存储过程的优点

1. 提高应用程序的运行性能
2. 模块化的设计思想[分页的过程，订单的过程，转账的过程..]
3. 减少网络传输量
4. 提高安全性

存储过程的缺点

1. 移植性不好
2. 不好维护
3. 不方便调试

存储过程使用

create procedure 过程名(in 变量 变量类型..., out 变量 变量类型,inout 变量 变量类型)

begin

执行语句;

end;

```
-- 1.
-- 创建存储过程
create PROCEDURE proc_get_emp_sal(in id_no int(11),out salary double(11,2))
BEGIN

# 根据员工id查询工资
select s.after_tax_salary into salary from t_employee e join t_salary s on e.id = s.emp_id where
e.id = id_no;

end

-- 在控制台调用
call proc_get_emp_sal(1,@salary);
select @salary;

-- 2.使用inout参数类型
create PROCEDURE proc_get_emp_sal2(in id_no int(11),inout salary double(11,2))
BEGIN

# 根据员工id查询工资
select s.after_tax_salary into salary from t_employee e join t_salary s on e.id = s.emp_id where
e.id = id_no and s.pre_tax_salary >= salary;

end

-- 在控制台调用
set @salary = 6000;
call proc_get_emp_sal(1,@salary);
select @salary;
```

```
-- 练习:
-- 1.创建一个存储过程
-- 2.将职位为“java”的员工加工资，每人加1000块

create PROCEDURE proc_add_salary(in position varchar(20), in salary double(11,2))
BEGIN

# 定义员工编号变量
DECLARE emp_no int(11);

#这里很重要设置一个标志符
DECLARE done INT DEFAULT 0;
```

```
# 根据条件查询员工信息放到游标
DECLARE cur cursor for select e.id from t_employee e where e.position = position;

# 如果游标指向最后一行则吧标志值改为1
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

# 打开游标
OPEN cur;

#遍历游标
REPEAT

FETCH cur into emp_no;

# 如果游标指向行不是最后一行
IF NOT done THEN

#修改员工工资
update t_salary s set s.after_tax_salary = s.after_tax_salary + salary where s.emp_id = emp_no;

# 结束if判断语句
END IF;

# 如果done的值为1代表游标指向的行时最后一行则结束循环
UNTIL done END REPEAT;

# 关闭游标
CLOSE cur;

END

# 执行存储过程
call proc_add_salary('java',1000);
```

存储过程参数类型

1. in：代表该参数是输入参数
2. out：代表该参数是输出参数
3. inout：代表该参数既是输入参数也是输出参数

注意：存储过程没有返回值，如果想返回数据则需要传入输出参数，然后将返回值绑定到输出参数，最好从输出参数取出返回值

方法

常用的函数

1. count(列名)：统计该列不为null的数据个数
2. sum(列名)：计算该列的所有数据的和
3. avg(列名)：计算该列的所有数据的平均数
4. max(列名)/min(列名)：求该列的最大/最小值

时间函数

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_SUB (date2 , INTERVAL d_value d_type)	在date2上减去一个时间
DATEDIFF (date1 ,date2)	两个日期差(结果是天)
TIMEDIFF(date1,date2)	两个时间差(多少小时多少分钟多少秒)
NOW ( )	当前时间
YEAR|Month|DATE (datetime)	年月日

字符串函数

CHARSET(str)	**返回字串字符集**
CONCAT (string2 [,...])	连接字串
INSTR (string ,substring)	返回substring在string中出现的位置,没有返回0
UCASE (string2)	转换成大写
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
REPLACE (str ,search_str ,replace_str)	在str中用replace_str替换search_str
STRCMP (string1 ,string2)	逐字符比较两字串大小,
SUBSTR(str , start_position [,end_position])	从str的start_position到end_position之间的字符
SUBSTRING (str , position [,length])	从str的position开始,取length个字符
LTRIM (string2) RTRIM (string2) trim	去除前端空格或后端空格

数学函数

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
LEAST (number , number2 [,..])	求最小值
MOD (numerator ,denominator)	求余
RAND([seed])	RAND([seed])

流程控制函数

IF(expr1,expr2,expr3)	**如果expr1为True ,则返回expr2 否则返回 expr3**
IFNULL(expr1,expr2)	如果expr1为空NULL,则返回expr2,否则返回expr1
SELECT CASE WHEN expr1 THEN expr2 [WHEN expr3 THEN expr4...] [ELSE expr5] END; [也可以多重分支.]	如果expr1 为TRUE,则返回expr2,否则返回expr3

其他函数

USER()	**查询用户**
DATABASE()	数据库名称
MD5(str)	为字符串算出一个 MD5 128比特检查和,通常用于对应用程序使用到的表的某个字段(比如用户密码)加密
PASSWORD(str)	从原文密码str 计算并返回密码字符串,通常用于对mysql数据库的用户密码加密

事务

事务用于保证数据的一致性,它由一组相关的dml语句组成,该组的dml语句要么全部成功，要么全部失败。如:网上转账就是典型的要用事务来处理，用以保证数据的一致性。

事务的四大特性

1. 原子性（Atomicity）

   原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

2. 一致性（Consistency）

   事务必须使数据库从一个一致性状态变换到另外一个一致性状态。

3. 隔离性（Isolation）

   事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。

4. 持久性（Durability）

   持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响

隔离级别

隔离级别	脏读	不可重复读	幻读
读未提交（Read uncommitted）	V	V	V
读已提交（Read committed）	x	V	V
可重复读（Repeatable read）	x	x	V
可串行化（Serializable ）	x	x	x

- 脏读（dirty read）：当一个事务读取另一个事务尚未提交的修改时，产生脏读。
- 不可重复读（nonrepeatable read）：同一查询在同一事务中多次进行，由于其他提交事务所做的修改或删除，每次返回不同的结果集，此时发生非重复读。
- 幻读（phantom read）：同一查询在同一事务中多次进行，由于其他提交事务所做的插入操作，每次返回不同的结果集，此时发生幻读。

事务使用步骤

1. start transaction //开始一个事务
2. savepoint 保存点名 //设置保存点
3. rollback to 保存点名 //取消部分事务 / rollback //取消全部事务
4. commit //提交事务

备份和恢复

备份

```
mysqldump -u root -p study > E:study.sql

mysqldump -u root -p study t_class t_student t_teacher > E:study2.sql

mysqldump -u root -p -B study shop_1 > E:study3.sql

mysqldump -u root -p -A > E:study4.sql
```

恢复

```
use recover1;

source E:study.sql;
```

常问的问题

1. 行列转换

   MySql行列转换

2. 去重

   MySQL 处理重复数据

附件

mysql复习-20180326.sql

study.sql