MySQL基础篇

DDL-数据库操作（对数据库和表的操作语言）

➢查询

```
查询所有数据库
SHOW DATABASES;
查询当前数据库
SELECT DATABASE();
```

➢创建　　※方括号中为可选项，实际中方括号不需要输入

```
CREATE DATABASE [IF NOT EXISTS] 数据库名 [DEFAULT CHARSET 字符集] [COLLATE 排序规则];
・create database itcast;
・create database if not exits itcast;
・creat database itheima default charset utf8mb4;
```

➢删除

```
DROP DATABASE [IF EXISTS] 数据库名;
・drop database test;
・drop database if exists test;
```

➢使用

```
USE 数据库名;
・use itcast;
```

DDL-表操作-查询

➢查询当前数据库所有表

```
SHOW TABLES;
```

➢查询表结构

```
DESC 表名;
```

➢查询指定表的建表语句

```
SHOW CREATE TABLE 表名;
```

DDL-表操作-创建

```
➢CREATE TABLE 表名(
        字段1 字段1类型[COMMNET 字段1注释],
        字段2 字段2类型[COMMNET 字段2注释],
        字段3 字段3类型[COMMNET 字段3注释],
        . . . . . .
        字段n字段n类型[COMMNET 字段n注释]
)[COMMNET 表注释];
```

```
 create table tb_user(
    -> id int comment '编号',
    -> name varchar(50) comment '姓名',
    -> age int comment '年龄',
    -> gender varchar(1) comment '性别'
    -> )comment '用户表';

create table emp(
        id int comment '编号',
        empno varchar(10) comment '工号',
        name varchar(10) comment '姓名',
        gender char(1) comment '性别',
        age tinyint unsigned comment '年龄',
        idcard char(18) comment '身份证号',
        entrydate date comment '入职时间'
) comment '员工表';
```

DDL-表操作-修改

➢添加字段

```
ALTER TABLE 表名 ADD 字段名 类型(长度) [COMMENT 注释] [约束];
·alter table emp add nickname varchar(20);
```

➢修改数据类型

```
ALTER TABLE 表名 MODIFY 字段名 新数据类型(长度);
```

➢修改字段名和字段类型

```
ALTER TABLE 表名 CHANGE 旧字段名 新字段名 类型(长度)[COMMENT 注释] [约束];
·alter table emp change nickname username varchar(30);
```

➢删除字段

```
ALTER TABLE 表名 DROP 字段名;
·alter table emp drop username;
```

➢修改表名

```
ALTER TABLE 表名 RENAME TO 新表名;
·alter table emp rename to employee;
```

DDL-表操作-删除

➢删除表

```
DROP TABLE [IF EXISTS] 表名;
·drop table if exists tb_user;
```

➢删除指定表，并重新创建该表

```
TRUNCATE TABLE 表名;
```

DML数据操作语言，用来对数据库中表的数据记录进行增删改操作

➢添加数据（INSERT）
➢修改数据（UPDATE）
➢删除数据(DELETE)

DML-添加数据

➢给指定字段添加数据

```
INSERT INTO 表名（字段名1，字段名2，...） VALUES (值1，值2，...);
```

➢给全部字段添加数据

```
INSERT INTO 表名 VALUES （值1，值2，...）;
```

➢批量添加数据

```
INSERT INTO 表名（字段名1，字段名2，...） VALUES (值1，值2，...)，(值1，值2，...)，(值1，值2，...);
```

```
INSERT INTO 表名 VALUES （值1，值2，...），（值1，值2，...），（值1，值2，...）;
```

DML-修改数据

➢修改数据

```
UPDATE 表名 SET 字段名1 = 值1, 字段名2 = 值2, ...[WHERE 条件];
```

DML-删除数据

➢删除数据

```
DELETE FROM 表名 [WHERE 条件];
```

DQL数据查询语言，用来查询数据库中表的记录。

Data Query Language

➢DQL-语法

```
SELECT
        字段列表
    FROM
        表名列表
    WHERE
        条件列表
    GROUP BY
        分组列表
    HAVING
        分组后条件列表
    ORDER BY
        排序字段列表
LIMIT
        分页参数
```

DQL-基本查询

➢查询多个字段

```
SELECT 字段1, 字段2，字段3 ...FROM 表名;
SELECT * FROM 表名;
```

➢设置别名

```
SELECT 字段1 [AS 别名1]，字段2 [AS 别名2] ... FROM 表名;
```

➢去除重复记录

```
SELECT DISTINCT 字段列表 FROM 表名;
```

DQL-条件查询

➢语法

SELECT 字段列表 FROM 表名 WHERE 条件列表;

➢条件

比较运算符

>	大于
>=	大于等于
<	小于
<=	小于等于
=	等于
<> 或 !=	不等于
BETWEEN ... AND ...	在某个范围之内（含最小，最大值）
IN(...)	在in之后的列表中的值，多选一
LIKE 占位符	模糊匹配（_匹配单个字符，%匹配任意个字符）
IS NULL	是NULL

逻辑运算符

AND 或 &&	并且（多个条件同时成立）
OR 或 ||	或者（多个条件任意一个成立）
NOT 或 !	非，不是

DQL-聚合函数

➢介绍

将一列数据作为一个整体，进行纵向计算。

➢常见聚合函数

count	统计数量
max	最大值
min	最小值
avg	平均值
sum	求和

➢语法

SELECT 聚合函数（字段列表） FROM 表名;

```
select avg(age) from employee;

select count(id) from employee;
select count(*) from employee;

select max(age) from employee;

select min(age) from employee;

select sum(age) from employee where entrydate='2010-10-23';
```

DQL-分组查询

➢语法

SELECT 字段列表 FROM 表名 [WHERE 条件] GROUP BY 分组字段名 [HAVING 分组后过滤条件]；

➢where与having区别

执行时机不同：where是分组之前进行过滤，不满足where条件，不参与分组；而having是分组之后对结果进行过滤。

判断条件不同：where不能对聚合函数进行判断，而having可以。

➢注意

执行顺序：where > 聚合函数 > having

分组之后，查询的字段一般为聚合函数和分组字段，查询其他字段没有意义

```
select gender, count(*) from employee group by gender;

select gender, avg(age) from employee group by gender;

select address,count(*) from employee where age < 45 group by address;

select address,count(*) from employee where age < 45 group by address having count(*) >= 2;

select address,count(*) from employee where age < 45 group by address having address='福建';
```

DQL-排序查询

➢语法

SELECT 字段列表 FROM 表名 ORDER BY 字段1 排序方式1，字段2 排序方式2；

➢排序方式

ASC：升序（默认值）

DESC：降序

➢注意

如果是多字段排序，当第一个字段值相同时，才会根据第二个字段进行排序。

```
select * from employee order by age asc;

select * from employee order by age desc;

select * from employee order by entrydate desc;

select * from employee order by age asc, entrydate desc;
```

DQL-分页查询

➢语法

SELECT 字段列表 FROM 表名 LIMIT 起始索引，查询记录数；

➢注意

起始索引从0开始，起始索引=（查询页码-1）*每页显示记录数。

分页查询是数据库的方言，不同的数据库有不同的实现，MySQL中是LIMIT。

如果查询的是第一页数据，起始索引可以省略，直接简写为limit 10

```
select * from employee limit 0,3;

select * from employee limit 3,3;
```

#查询性别为男，且年龄在20-40岁（含）以内的前5个员工信息，对查询的结果按年龄升序排序，年龄相同按入职时间升序排序

```
select * from employee where gender = '男' and age  between 18 and 40 order by age asc ,entrydate desc limit 5;
```

DQL-执行顺序

➢编写顺序　　执行顺序

SELECT　　4
　　字段列表
FROM　　1
　　表名列表
WHERE　　2
　　条件列表
GROUP BY　　3
　　分组字段列表
HAVING
　　分组后条件列表
ORDER BY　　5
　　排序字段列表
LIMIT　　6
　　分页参数

DCL-数据控制语言

➢介绍

DCL-Data Control Language(数据控制语言)，用来管理数据库用户，控制数据库的访问权限。

DBA-Database Administrator

DCL-管理用户

➢查询用户

```
USE mysql;
SELECT * FROM user;
```

➢创建用户

```
CREATE USER '用户名'@'主机名' IDENTIFIED BY '密码';
#创建用户itcast，只能在当前主机localhost访问，密码123456
create user 'itcast'@'localhost' identified by '123456';
#创建用户黑马，可以在任意主机访问该数据库，密码123456
create user 'heima'@'%' identified by '123456';
```

➢修改用户密码

```
ALTER USER '用户名'@'主机名' IDENTIFIED WITH mysql_native_password BY '新密码';
#修改用户heima的访问密码为1234
alter user 'heima'@'%' identified with mysql_native_password BY '1234';
```

➢删除用户

```
DROP USER '用户名'@'主机名';
#删除itcast@localhost用户
drop user 'itcast'@'localhost';
```

DCL-权限控制

➢权限

ALL，ALL PRIVILEGES	所有权限
SELECT	查询数据
INSERT	插入数据
UPDATE	修改数据
DELETE	删除数据
ALTER	修改表
DROP	删除数据库/表/视图
CREATE	创建数据库/表

DCL-权限控制

➢查询权限

```
SHOW GRANTS FOR '用户名'@'主机名';
show grants for 'heima'@'%';
```

➢授予权限

```
GRANT 权限列表 ON 数据库名.表名 TO '用户名'@'主机名';
grant all on itcast.* to 'heima'@'%';
```

➢撤销权限

```
REVOKE 权限列表 ON 数据库名.表名 FROM '用户名'@'主机名';
revoke all on itcast.* from 'heima'@'%';
```

注意:

多个权限之间，使用逗号分隔

授权时，数据库名和表名可以使用*进行通配ui，代表所有

字符串函数

➢**CONCAT(S1,S2,...Sn)**

字符串拼接，将S1，S2，...Sn拼接成一个字符串

➢**LOWER(str)**

将字符串str全部转换为小写

➢**UPPER(str)**

将字符串str全部转为大写

➢**LPAD(str,n,pad)**

左填充，用字符串pad对str的左边进行填充，达到n个字符串长度

➢**RPAD(str,n,pad)**

右填充，用字符串pad对str的右边进行填充，达到n个字符串长度

➢**TRIM(str)**

去掉字符串头部和尾部的空格

➢**SUBSTRING(str,start,len)**

返回从字符串str从start位置起的len个长度的字符串

```
select concat('hello','world');

select upper('hello');

select lower('HELLO');

select lpad('01', 5, '-');
```

```
select rpad('01', 5, '-');

select trim('  hello world ');

select substring('hello world', 1, 5);
```

数值函数

➢CEIL(X)
　　向上取整

➢FLOOR(X)
　　向下取整

➢MOD(X,Y)
　　返回x/y的模 取余数

➢RAND()
　　返回0-1内的随机数

➢ROUND(X，Y)
　　求参数x的四舍五入的值，保留y为小数

```
select ceil(1.5);
->2
select ceil(1.1);
->2
select floor(1.9);
->1
select mod(7,4);
->3
select rand();

select round(2.345,2);
->2.35

#通过数据库的函数，生成一个六位数的随机验证码
select lpad(round(rand()*1000000,0),6,'0');
```

日期函数

➢CURDATE()
　　返回当前日期

➢CURTIME()
　　返回当前时间

➢NOW()
　　返回当前日期和时间

➢YEAR(date)
　　获取指定date的年份

➢MONTH(date)
　　获取指定date的月份
➢DAY(date)
　　获取指定date的日期

➢DATE_ADD(date,INTERVALexpr type)
　　返回一个日期/时间值加上一个时间间隔expr后的时间值

➢DATEDIFF(date1,date2)
返回起始时间date1和结束时间date2之间的天数

```
select curdate();

select curtime();

select now();

select year(now());

select month(now());

select day(now());

select date_add(now(),interval 70 day );
select date_add(now(),interval 70 month );
select date_add(now(),interval 70 year );

select datediff('2022-10-01','2021-11-13');

查询所有员工的入职天数，并根据入职天数倒序排序
select name, datediff(curdate(),entrydate) as 'entrydays' from employee order by entrydays desc;
```

流程函数

➢IF(value，t，f)
如果value为true，则返回t，否则返回f

➢IFNULL(value1，value2)
如果value1不为空，返回value1，否则返回value2

➢CASE WHEN [val1] THEN [res1] ...ELSE [default] END
如果val1为true，返回res1，...否则返回default默认值

➢CASE [expr] WHEN [val1] THEN [res1] ...ELSE [default] END
如果expr的值等于val1，返回res1，...否则返回default默认值

```
select if(true,'ok','error');
true

select if(false,'ok','error');
error

select ifnull('ok','default');
ok

select ifnull('','default');


select ifnull(null,'default');
default

select
    name,
    case address when '上海' then '一线城市' when '福建' then '二线城市' else '其他' end
from employee;

create table score(
    id int comment 'ID',
    name varchar(20) comment '姓名',
    math int comment '数学',
    english int comment '英语',
    chinese int comment '语文'
)comment '学员成绩表';

insert into score(id, name, math, english, chinese) VALUES (1,'tom',87,92,56),(2,'lina',95,70,60),(3,'jerry',80,70,60);

select
    id,
    name,
    case when math >= 85 then '优秀' when math >= 60 then '及格' else '不及格' end '数学',
    case when english >= 85 then '优秀' when english >= 60 then '及格' else '不及格' end '英语',
    case when chinese >= 85 then '优秀' when chinese >= 60 then '及格' else '不及格' end '语文'
```

```
from score;
```

约束

➢概念

约束是作用于表中字段上的规则，用于限制储存在表中的数据。

➢目的

保证数据库中数据的正确，有效性和完整性。

➢分类

```
NOT NULL                非空约束          限制该字段的数据不能为null
UNIQUE                  唯一约束          保证该字段的所有数据都是唯一，不能重复的
PRIMARY KEY     主键约束          主键是一行数据的唯一标识，要求非空且唯一
DEFAULT         默认约束          保存数据时，如果未指定该字段的值，则采用默认值
CHECK                   检查约束          保证字段值满足某一个条件
FOREIGN KEY     外键约束          用来让两张表的数据之间建立连接，保证数据的一致性和完整性
```

注意：约束是作用于表中字段上的，可以在创建表/修改表的时候添加约束

```
create table user(
    id int primary key auto_increment comment '主键',
    name varchar(10) not null unique comment '姓名',
    age int check ( age > 0 && age <= 120 ) comment '年龄',
    status char(1) default '1' comment '状态',
    gender char(1) comment '性别'
)comment '用户表';
```

外键约束

➢概念

外键用来让两张表的数据之间建立连接，从而保证数据的一致性和完整性。

注意：目前上述的两张表，在数据库层面，并未建立外键关联，所以无法保证数据的一致性和完整性的。

➢语法

添加外键

```
CREATE TABLE表名（
        字段名 数据类型，
        ...
        [CONSTRAINT][外键名称] FOREIGN KEY(外键字段名) REFERENCES 主表（主表列名）
）;
ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段名) REFERENCES 主表（主表列名）;
alter table emp add constraint fk_emp_dept_id foreign key (dept_id) references dept(id);
#删除外键
alter table emp drop foreign key fk_emp_dept_id;
```

➢删除/更新行为

```
NO ACTION               当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果
```

有则不允许删除/更新。（与RESTRICT一致）

```
RESTRICT                当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有则
```

不允许删除/更新。（与NO ACTION一致）

```
CASCADE         当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有，
```

则也删除/更新外键在子表中的记录。

```
SET NULL                当在父表中删除对应记录时，首先检查下该记录是否有对应外键，如果有则设
```

置子表中该外键值为null（这就要求该外键允许取null）。

```
SET DEFAULT     父表有变更时，子表将外键列设置成一个默认的值（Innodb不支持）
```

多表查询

➢概述

项目开发中，在进行数据库表结构设计时，会根据业务需求及业务模块之间的关系，
分析并设计表结构，由于业务之间相互关联，所以各表结构之间也存在着各种联系，
基本上分为三种：

➢一对多（多对一）

➢多对多

➢一对一

多表关系

➢一对多（多对一）

➢案例：部门与员工的关系

➢关系：一个部门对应多个员工，一个员工对应一个部门

➢实现：在多的一方建立外键，指向一的一方的主键

➢多对多

➢案例：学生与课程的关系

➢关系：一个学生可以选修多门课程，一门课程也可以供多个学生选择

➢实现：建立第三张中间表，中间表至少包含两个外键，分别关联两方主键

➢一对一

➢案例：用户 与 用户详情的关系

➢关系：一对一关系，多用与单表拆分，将一张表的基础字段放在一张表中，其他的详情字段放在另一张表中，以提升操作效率

➢实现:在任意一方加入外键，关联另外一方的主键，并且设置外键为唯一的（UNIQUE）

多表查询

连接查询

内连接：相当于查询A，B交集部分数据

➢隐式内连接

SELECT 字段列表 FROM 表1，表2 WHERE 条件 ... ;

```
select * from emp, dept where emp.dept_id = dept.id;
select emp.name, dept.name from emp, dept where emp.dept_id = dept.id;
select e.name, d.name from emp e, dept d where e.dept_id = d.id;
```

➢显示内连接

SELECT 字段列表 FROM 表1 [INNER] JOIN 表2 ON 连接条件 ... ;

```
select * from emp e inner join dept d on e.dept_id = d.id;
select * from emp e join dept d on e.dept_id = d.id;
```

内连接查询的是两张表交集的部分

外连接：

➢左外连接：查询左表所有数据，以及两张表交集部分数据

SELECT 字段列表 FROM 表1 LEFT [OUTER] JOIN 表2 ON 条件 ... ;

```
select e.*, d.name from emp e left outer join dept d on e.dept_id = d.id;
select e.*, d.name from emp e left join dept d on e.dept_id = d.id;
```

➢右外连接：查询右表所有数据，以及两张表交集部分数据

SELECT 字段列表 FROM 表1 RIGHT [OUTER] JOIN 表2 ON 条件 ... ;

```
select d.*, e.* from emp e right outer join dept d on d.id = e.dept_id;
select d.*, e.* from emp e right join dept d on d.id = e.dept_id;
```

自连接：当前表与自身的连接查询，自连接必须使用表别名

SELECT 字段列表 FROM 表A 别名A JOIN 表B 别名B ON 条件... ;

自连接查询，可以是内连接查询，也可以是外连接查询

```
select a.name,b.name from emp a, emp b where a.managerid = b.id;
select a.name '员工', b.name '领导' from emp a left outer join emp b on a.managerid = b.id;
```

联合查询：把多次查询的结果合并起来，形成一个新的查询结果集。

SELECT 字段列表 FROM 表A ... ;

UNION[ALL]

SELECT 字段列表 FROM 表B ... ;

对于联合查询的多张表的列数必须保持一致，字段类型也需要保持一致

```
select * from emp where salary < 5000
union all
select * from emp where age > 50
select * from emp where salary < 5000
union   去重
select * from emp where age > 50
```

子查询

➢概念：SQL语句中嵌套SELECT语句，称为嵌套查询，又称子查询

SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);

子查询外部的语句可以是INSERT/UPDATE/DELETE/SELECT的任何一个

➢根据子查询结果不同，分为：

➢标量子查询（查询结果为单个值） 常用操作符 = 、<> 、> 、 >= 、< 、<=

```
select * from emp where dept_id = (select id from dept where name = '总经办');
select * from emp where dept_id = (select id from dept where name = '研发部');
select * from emp where entrtydate > (select entrtydate from emp where name='杨晓');
```

➢列子查询（子查询结果为一列）

➢常用操作符

IN	在指定的集合范围之内，多选一
NOTIN	不在指定的集合范围之内
ANY	子查询返回列表中，有任意一个满足即可
SOME	与ANY等同，使用SOME的地方都可以使用ANY
ALL	子查询返回列表的所有值都必须满足

```
select id from dept where name = '研发部' or name = '总经办';
select * from emp where dept_id in (select id from dept where name = '研发部' or name = '总经办');
select salary from emp where dept_id = (select id from dept where name = '研发部');
select * from emp where salary > all ( select salary from emp where dept_id = (select id from dept where name = '研发部') );
select * from emp where salary > some ( select salary from emp where dept_id = (select id from dept where name = '研发部') );
```

➢行子查询（子查询结果为一行）

➢常用操作符 = 、<>、IN、NOT IN

```
select salary, managerid from emp where name = '张无忌';
select * from emp where (salary,managerid) = (select salary, managerid from emp where name = '张无忌');
```

➢表子查询（子查询结果为多行多列）

➢常用操作符 IN

```
select * from emp where (job,salary) in ( select job, salary from emp where name = '张无忌' or name = '杨晓');
```

```
select e.*,d.* from (select * from emp where entrtydate > '2000-5-20') e left join dept d on e.dept_id = d.id;
```

➢根据子查询位置，分为：

➢WHERE之后

➢FROM之后

➢SELECT之后

事务

事务是一组操作的集合，它是一个不可分割的工作单位，事务会把所有的操作作为一个整体一起向系统提交或撤销操作请求，即这些操作要么同时成功，要么同时失败。

➢操作演示

```
create table account(
    id int auto_increment primary key comment '主键ID',
    name varchar(10) comment '姓名',
    money int comment '余额'
)comment '账户表';

insert into account(id, name, money) VALUES (null, '张三',2000),(null, '李四',2000);

update account set money = 2000 where name = '张三' or name = '李四';

select * from account where name = '张三';

update account set money = money - 1000 where name = '张三';
程序抛出异常...
update account set money = money + 1000 where name = '李四';
```

➢事务操作

方式1

查看/设置事务提交方式

```
select @@autocommit;

set @@autocommit = 0;

事务。。。
```

提交事务

```
commit; 成功提交
```

回滚事务

```
rollback; 失败回滚
```

方式2

开启事务

```
start transaction ;
或者
begin ;

事务。。。
```

提交事务

```
commit ; 成功提交
```

回滚事务

```
rollback ; 失败回滚
```

➢事务的四大特性

- 原子性（Atomicity）：事务是不可分割的最小操作单元，要么全部成功，要么全部失败
- 一致性（Consistency）：事务完成时，必须使所有的数据都保持一致状态
- 隔离性（Isolation）：数据库系统提供的隔离机制，保证事务在不受外部并发操作影响的独立环境下运行。
- 持久性（Durability）：事务一旦提交或回滚，它对数据库中的数据的改变就是永久的。

➢并发事务问题

- 脏读　　一个事务读到另外一个事务还没有提交的数据。
- 不可重复读　一个事务先后读取同一条记录，但两次读取的数据不同，称之为不可重复读
- 幻读　　一个事务按照条件查询数据时，没有对应的数据行，但是在插入数据时，又发现这行数据已经存在，好像出现了“幻影”

➢事务隔离级别

隔离级别	脏读	不可重复读	幻读
Read uncommitted	✓	✓	✓
Read committed	×	✓	✓
Repeatable Read(默认)	×	×	✓
Serializable	×	×	×

查看事务隔离级别

```
select @@transaction_isolation;
```

设置事务隔离级别

```
set session transaction isolation level read uncommitted ;  // read committed,repeatable read,serializable
```
