sql语句

- DQL(数据查询语言) ---> 查询语句,凡是select语句都是DQL.
- DML(数据操作语言) ---> insert,delete,update,对表当中的数据进行增删改
- DDL(数据定义语言) ---> create,drop,alter,对表结构的增删改
- TCL(事务控制语言) ---> commit提交事物,rollback回滚事务
- DCL(数据控制语言) ---> grant授权,revoke撤销权限等

导入数据库

source D:\xxxx\ddd.sql

创建数据库

create database aaa;

desc用法

1. 查看sql脚本的执行情况，与explain命令效果一致
   desc select * from table_name;
2. 查看表结构
   desc table_name;
3. 对表某列数据进行排序
   select * from table_name order by col_name desc;

查询当前使用的数据库

select database();
select version();

查看创建表的语句

show create table aaa;

终止一条正在编写的语句

\c

使用数据库

use aaa;

查

给查询结果的列重命名

- 执行顺序

from
on
join
where
group by
having
select

distinct
union
order by

```
select ename,sal*12 as yearsal from aaa;
select ename,sal*12 as '年薪' from aaa;
select ename,sal*12 年薪, from aaa;
```

条件查询

```
select ename from aaa where sal=5000;

select sal from aaa where ename='bbb';

select ename,sal from aaa where sal>3000;

select ename,sal from aaa where sal != 3000;

select ename,sal from aaa where sal <> 3000;//!=和<>是一个意思

select ename,sal from aaa where sal >=1100 and sal <= 3000;

select ename,sal from aaa where sal between 1100 and 3000; between...and...是闭区间

/* 在数据库当中NULL不是一个值，代表什么也没有，为空。 空不是一个值u，不能用等号衡量。必须使用is null或者 is not null */

select ename, sal, comm from aaa where comm is null; /*不能使用comm=null */

select ename, sal, comm from aaa where comm is not null;

select ename,sal,comm from aaa where comm is null or comm =0;

select ename,sal,deptno from aaa where sal >1000 and (deptno =20 or deptno =30);
/* 小括号很关键  不知道谁优先级高的时候就加小括号来限定 */

/* in等同于or: */
/* not in:不在这几个值当中 */
select ename,job from aaa where job ='SALESMAN' or job ='MANAGER';
select ename,job from aaa where job in('SALESMAN','MANAGER');

/* %代表任意多个字符，_代表任意一个字符 */

select ename from where ename like '%O%';

select ename from where ename like '_A%';

select ename from where ename like '%\_%'; /* 寻找名字带有下划线的 */

select ename from where ename like '%T'; /* 最后一个字母带T的 */

select ename sal from aaa order by sal; /* 默认升序排列   asc升序，desc降序*/
```

```
select ename sal from aaa order by sal asc; /* 升序 */

select ename sal from aaa order by sal desc;/* 降序 */

select ename,sal from aaa order by sal desc, enmae asc; /* 按照工资的降序排列，当工资相同的时候再按照名字的升序排列 越考前的字段越能起到主导作用，只有当前面的字段无法完成排序的时候，才会启用后面的字段*/

select ename,sal from aaa order by 2/*按照第二列排序*/;
```

分组函数

> 分组函数自动忽略NULL
> 只要有NULL参与的运算结果一定是NULL

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

```
select sum(sal) from aaa;

select count(enmae) from aaa; /* 用户的个数 */

/* ifnull()空处理函数? ifnull(可能为NULL的数据,被当作什么处理) */

select ename,ifnull(comm,0) as comm from aaa;

/* 找出比平均工资高的 */
select
```

group by 和 having

group by: 按照某个字段或者某些字段进行分组。
having: having是对分组之后的数据进行再次过滤
分组函数一般都会和group by联合使用

```
select max(sal) from aaa group by job;

select ename,sal from aaa where sal > (select avg(sal) from emp);/* 找出高于平均工资的员工 */

select max(sal),job from aaa group by job; /* 找出每个工作岗位的最大薪资 */

/* 当一条语句中有group by的话, select后面只能跟分组函数和参与分组的字段 */

/* 找出不同部门不同工作岗位的最高薪资 */
select detno,job max(sal) from aaa group by deptna,job;

/* 找出每个部门的最高薪资,要求显示薪资大于2900的数据 */
```

```
select max(sal),deptno from aaa group by deptno having max(sal)>2900  /*这种效率低
*/
select max(sal),deptno from aaa where sal >2900 group by deptno;/* 这种效率高  建议
能使用where就尽量使用where*/
/* 找出每个部门的平均薪资，要求显示薪资大于2000的数据 */
select deptno, avg(sal) from aaa group by deptno having avg(sal)>2000;
```

查询结果集的去重

```
select distinct job from aaa; /* distinct关键字是去除重复 distinct只能出现在所有字段的
最前面 */
select distinct deptno,job from aaa; /* deptno和job都去重了 */

/* 统计岗位的数量 */
select count(distinct job) from aaa;
```

表的连接方式

笛卡尔积现象: 当两张表进行连接查询的时候，没有任何条件限制，最终的查询结果数条是两张表记录条数的乘机

避免笛卡尔积现象: 加条件过滤。避免了笛卡尔积现象.不会减少匹配次数只不过是显示的是有效记录

内连接

等值连接

/* 找出每一个员工的部门名称, 要求显示员工和部门名 */

```
select e.ename,d.dname from emp e, dept d where e.deptno=d.deptno;
```

sql99语法:

```
select e.ename,d.dname from emp e join dept d on e.deptno= d.deptno;
```

表的别名

```
select e.ename,d.dname from emp e, dept d;
/* 好处  第一  执行效率高.  第二   可读性好.*/
```

非等值连接

```
select e.ename,e.sal,s.grade from emp e join salgrade s on e.sal between s.losal
and s.hisal;
```

自连接

```
/* 找出每个员工的上级领导 要求显示员工和对应的领导名*/
select a.ename as '员工名',b.ename as '领导名' from emp a join emp b on a.mgr
=b.empno;
```

外连接

假设A和B表进行连接,使用外连接的话，AB两张表中有一张表是主表,一张表是副表,主要是查询主表中的数据,顺便查副表,当副表中的数据没有和主表中的数据匹配上,副表自动模拟出NULL与之匹配.

左外连接(左连接)
右外连接(右连接)

```
/*找出每个员工的上级领导(所有员工必须都查询出来) */
select a.ename '员工', b.ename '领导' from emp a left join emp b on a.mgr=b.empno;

/* 找出哪个部门没有员工 */
select d.* from emp e right join dept d on e.deptno =d.deptno where e.emptno is
null;

/* 找出每个员工的部门名称以及工资等级 */

A join B join C on .... /* A先和B连接然后A再和C连接 */

select e.ename,d.dname,s.grade from emp e join dept d on e.deptno=d.deptno join
salgrade s on e.sal between s.losal and hisal;
```

from后面嵌套子查询

```
/* 找出每个部门平均薪水的薪资等级 */
select deptno,avg(sal) as avgsal from emp group by deptno;

select t.*,s.grade from (select deptno,avg(sal) as avgsal from emp group by
deptno) t join salgrade s on t.avgsal between s.losal and s.hisal;

/* 每个部门平均的薪资等级 */
select e.deptno,avg(s.grade) from emp e join salgrade s on e.sal between s.losal
and s. hisal group by e.deptno;
```

select后面嵌套子查询

```
/* 找出每个员工所在的部门名称, 要求显示员工和部门名 */
select e.ename,(select d.dname from dept d where e.deptno=d.deptno) as dname
from emp e;
```

union可以将查询结果集相加)

```
/* 找出工作岗位是salesman和manager的员工? */
select ename,job from emp where job='salesman' or 'manager'
select ename,job from emp where job in('manager','salesman')

select ename,job from emp where job='salesman'
union
select ename,job from emp where job='manager';

/* 可以两张不相干的表拼接在一起 */
```

limit

limit取结果集中的部分数据,这是它的作用
limit是sql语句最后执行的一个环节

语法机制 ---> limit startIndex(表示起始位置),length(表示有几个)

```
/* 取出工资前5名的员工(降序前5个) */
select ename,sal from emp order by sal desc limit 0,5;

select ename,sal from emp order by sal desc limit 5;/* 跟上面的sql语句结果一样 */

/* 找出工资排名在第4到第9名的员工 */
select ename,sal from emp order by sal desc limit 3,6;

/*每页显示的数据 */

/* 第pageNo页: (pageNo -1)* pageSize,pageSize */
```

建表

```
create table t_student(

no bigint,
name varchar(255),
sex char(1),
classno varchar(255)
birth char(10)
)
```

插入语句

insert语句插入数据

```
insert into t_student(no,name,sex,classno,birth)
values(1,'zhangsan','1','gaosanyiban','1950-10-12');
```

表的复制

```
create table emp as select * from emp;
```

修改数据

```
update dept1 set loc ='SHANGHAI',dname= 'ReNsHIBU' deptno =10;

/* 更新所有记录 */
update dept1 set loc ='x',dname = 'y';
```

删库

```
drop databaes aaa;

delete from 表明 where 条件;

/* 删除10部门数据 */
delete from dept1 where deptno =10;

delete from dept1; /* 如果数据量特别大那么可能会出现删除时间过长的问题 */

/* 删除大表 */
truncate table emp1; /* 表被截断, 不可回滚 永久丢失 */
```

约束

非空约束(not null)

```
drop table if exists t_user;
create table t_user(
id int,
username varchar(255) not null,
password varchar(255)
primary key(id)
);
```

唯一约束(unique)

```
drop table if exists t_user;
create tble t_user(
id int,
username varchar(255) unique
password varchar(255)
);
```

```
/* 给两个列或者多个列添加unique */
drop table if exists t_user;
create table t_user(
id int,
usercode varchar(255),
username varchar(255),
unique()
/* unique(usercode,username) 是一样的 */
/* usercode varchar(255) unique, username varchar(255) unique,正行不能重复 */
);
```

主键约束(primary)

```
drop table if exists t_user;
create table t_user(
id int primary key,
username varchar(255),
email varchar(255)

/* primary key(id) 这样也行 */
);

/* 主键自增 */
drop table if exists t_user;
create table t_user(
id int primary key auto_increment,
username varchar(255),
email varchar(255)
```

外键约束(foreign key)

```
create table t_class(
cno int,
cname varchar(255),
primary key(cno)
);

create table t_student(
sno int,
sname varchar(255),
classno int,
foreign key(classno) references t_class(cno)
);
```

检查约束(check)

存储引擎

mysql默认使用的存储引擎是 InnoDB方式 默认采用的字符集是UTF-8

MyISAM存储引擎

1. mysql最常用的引擎
2. 不是默认的
3. 表内容包含 **表结构,表数据,表索引**

优点

可被压缩 节省空间 并且可以转换为只读表 提高检索效率

缺点

不支持事物

InnoDB

优点

支持事物。行级锁,外键等。这种存储引擎数据的安全得到保障

这种InnoDB存储引擎在mysql数据库崩溃之后提供自动恢复

支持外键及引用的完整性，包括级联删除和更新

缺点

表的结构存储在 ×××.frm文件中。数据存储在tablespace这样的表空间中，无法被压缩，无法转换成只读.

MEMORY存储引擎

不支持事务，数据容易丢失，因为所有数据和索引都是存储在内存当中的.

优点

查询速度最快

缺点

不支持事务，数据容易丢失。因为所有数据和索引都是存储在内存当中的

事物

- 原子性
    - 整个事物中的所有操作，必须作为一个单元全部完成(或者全部取消)
- 一致性
    - 在事物开始之前与结束之后，数据库都保持一致状态
- 隔离性
    - 一个事物不会影响其他事务的运行
- 持久性
    - 最终数据必须持久化到硬盘中，事务才算成功的结果

事务之间的隔离性

第一级别

读未提交(read uncommitted): 对方事物还没提交，我们当前事务可以读取到对方未提交的数据。 读未提交存在脏读现象(表示读到了脏的数据)

第二级别

读已提交(read committed)**: 对方事物提交之后的数据我方可以读取到。 读已提交存在的问题是: 不可重复读

解决了脏读现象

第三级别

可重复读(repeatable read)

这种隔离级别解决了不可重复读的问题

存在问题: 读取到的数据是幻象

第四级别

序列化读/串行化读

解决了所有问题
效率低 需要事务排队

oracle数据库默认的隔离级别是: 读已提交.
mysql数据库默认的隔离级别是: 可重复读

索引

使用条件:

1. 数据量庞大
2. 该字段很少的DML操作 (因为字段进行修改操作，索引也需要维护)
3. 该字段经常出现在where子句中.(经常根据哪个字段查询)

主键和unique约束的字段自动会添加索引

```
/* 给薪资sal字段添加索引 */
create index emp_sal_index on emp(sal);

/* 删除索引对象 */
drop index 索引名称 on 表名;

索引底层采用的数据结构是 B + Tree

**索引的实现原理**

通过B Tree缩小扫描范围，底层索引进行了排序，分区，索引会携带数据在表中的“物理地址”，最终通过索引检索到数据之后 ，获取到关联的物理地址，通过物理地址定位表中的数据 效率是最高的。

**索引失效**

模糊查询的时候。第一个通配符使用的是%
```

视图

作用

试图可以隐藏表的实现细节，保密级别较高的系统，数据库只对外提供相关的视图 java程序员只对视图对象进行crud

导出数据

mysqldump bjpowernode> D:\bjpowernode.sql -uroot -p333

导入数据

create database bjpowernode;
use bjpowernode;
source D:\bjpowernode.sql

数据库设计三范式

1. 任何一张表都应该有主键，并且每一个字段原子性不可再分
2. 建立在第一范式的基础之上，所有非主键字段完全依赖主键。不能产生部分依赖
3. 建立在第二范式基础之上，所有非主键字段直接依赖主键，不能产生传递依赖
   (注意：在实际开发中，以满足客户的需求为主，有的时候会拿冗(rong)余换取执行速度 表的连查会笛卡尔积 所以有时候会慢一些