

数据库

day01

Oracle

11g

Mysql

5.6 5.7 8.0

sql语句分类

1.SQL语句分类 授权 用户 建表 增删改查 事务 锁

2.函数 分组 排序 去重

3.约束 子查询（嵌套查询） 连表查询

4.伪列 分页 各种关键字 plsql编程 存储过程 函数 触发器

Mysql 讲一下区别

sqlplus = 让我们和数据库进行沟通的工具

regedit = 注册表

用户：

system

scott

退出：

exit

day02

Oracle = 数据库

sqlplus = 让我们和数据库进行沟通的工具

regedit = 注册表

services.msc = 服务

用户：

system

scott

退出：

exit

SQL结构化查询语言

Structured Query Language = 结构化查询语言

DDL数据定义语言

Data Definition Language = 数据定义语言

create 创建 alter 修改 drop 删除 truncate 截断

DML数据操纵语言

Data Manipulation Language = 数据操纵语言

insert 增加 delete 删除 update 修改

DQL数据查询语言

Data Query Language = 数据查询语言

select 查询

DCL数据控制语言

Data Control Language = 数据控制语言

grant 授权 **revoke** 取消授权

TCL事务控制语言

Transaction Control Language = 事务控制语言

commit 提交 **rollback** 回滚 **savepoint** 保存点

给用户解锁：

```
alter user scott account unlock;
```

给用户加锁：

```
alter user scott account lock;
```

给用户授权：

```
grant dba to scott;
```

给用户取消授权：

```
revoke dba from scott;
```

创建新用户：

```
create user ET2301 identified by etoak;
```

给用户修改密码：

```
alter user scott identified by etoak;
```

删除用户：

```
drop user ET2301;
```

查看当前用户:

show user;

切换用户:

conn 用户名/密码;

conn scott/etoak;

练习:

在scott用户下新建用户ET, 在ET用户下新建用户ET2301, 连接ET2301, 删除ET2301

Oracle的数据类型:

字符型:

varchar2(20) 可变类型 0-4000字节

char(2) 固定类型 0-2000字节

数值型:

number(5)

number(5,1)

日期型:

date

timestamp 包含毫秒数

创建表:

```
create table student(  
id number(5),  
name varchar2(20),  
birthday date,  
sal number(5,1)  
);
```

练习:

创建teacher表, 字段有老师工号, 老师姓名, 老师入职时间, 老师年薪 (五位数以上, 两位小数)

建表以后, 修改表名:

```
alter table stu rename to student;
```

建表以后, 新增字段:

```
alter table student add email varchar2(30);
```

建表以后, 修改列名:

```
alter table student rename column email to youxiang;
```

建表以后, 修改数据类型:

```
alter table student modify youxiang varchar2(50);
```

建表以后，删除字段：

alter table student drop column youxiang;

建表以后，删除表：

drop table student2; 12

建表以后，截断表：

truncate table student; 23

练习：

给teacher表重命名tea，新增字段aihao，将aihao修改为hobby，删除hobby，截断表，删除表

1.表结构

2.表数据

3.表空间

查看表结构：

desc student;

查看当前用户下有哪些表：

select table_name from user_tables;

增删改查：

新增数据：insert

insert into student values(1,'zs',sysdate,5000);

insert into student (id,name,sal) values(2,'ls',6000);

删除数据：delete

delete from student;

delete from student where id = 1;

修改数据：update

update student set sal = sal + 500;

update student set sal = sal + 500 where id = 2;

update student set sal = sal + 500,name = 'lss' where id = 2; 用，隔开表示并且条件

查询数据：

select * from student;

select name,sal from student;

select name,sal from student where name = 'zs';

练习：

1) 新增数据：张三，李四，王五

2) 给张三和李四工资涨百分之三十

3) 用查询的方式展示出题二的效果

别名:

```
select name,sal * 1.3 as salary from student where name = 'zs' or name = 'lss';
select name,sal * 1.3 salary
from student
where name = 'zs' or name = 'lss';

select s.name,s.sal * 1.3 salary
from student s
where s.name = 'zs' or s.name = 'lss';
```

like: 模糊查询

%: 代表任意位的任意字符

_: 代表一位上的任意字符

```
select name from student where name like 'ET%';
select name from student where name like 'ET';
select name from student where name like 'ET%';
select name from student where name like '%s%';
```

not like:

```
select name from student where name not like '%s%';
```

练习:

姓张的和姓李的工资涨百分之三十

escape: 逃离符

通过指定一个字符位进行逃离, 来保证like之后的特殊字符看作是普通字符

```
select name from student where name like 'ET,' escape ',';
select name from student where name like 'ET..%' escape '.';
```

条件:

and: 并且 两个条件必须都成立

or: 或者 有一个条件成立即可

between and: 闭合区间

```
select name,sal from student where sal between 5000 and 5500;
select name,sal from student where sal >= 5000 and sal <= 5500;
```

比较:

```
< >= <= !=
<> 不等于
```

计算:

+ - * /

is null: 是空

is not null: 非空

```
select * from student where birthday is not null;
```

函数：

*：经过函数修饰的**列起别名**

dual：万能表，测试表

聚组函数：聚簇函数：组函数：多行函数 和分组group by一起使用

max() 最大值 min() 最小值 sum() 求和 avg() 求平均值 count() 求记录数
select max(sal) maxsal,min(sal) minsal,sum(sal) sumsal,avg(sal) avgsal
from student;

select count(*) num from student; 经过函数修饰的**列起别名** 并计算该列的数量
select count(1) num from student;
select count(birthday) num from student;

去重 distinct

select count(distinct sal) num from student;

单行函数：

ceil(): 向上取整

floor(): 向下取整

select ceil(12.3) num from dual; dual **//万能表，测试表**

abs(): 求绝对值

select abs(-666) num from dual;

power(a,b): 求a的b次方

select power(2,3) num from dual;

sqrt(): 求正平方根

select sqrt(16) num from dual;

sign(): 求符号位 正数返回1，负数返回-1，零返回0

select sign(-666) num from dual;

round(): **求四舍五入**

select round(3.141592654) num from dual;

select round(3.141592654,3) num from dual;

trunc(): 求直接截断

select trunc(3.141592654) num from dual;

select trunc(3.141592654,3) num from dual;

字符函数：

日期函数：

转换函数：

通用函数：

=====

TCL：事务控制语句

*：事务只会影响DML操作

事务：在一些列操作中有多个步骤，只有所有的步骤成功执行那么整个操作才算完成，如果有其中一个环节失败，那么整个操作都算失败。

Oracle中在sqlplus中开启事务：

set autocommit off; -- 关闭自动提交

set autocommit on; -- 开启自动提交

提交操作：将受影响的数据持久化到数据库中

事务特性：

原子性：事务不可再分

一致性：数据类型保持一致

持久性：事务能够将数据持久化到数据库中

隔离性：多个事务之间可能会产生一些隔离性问题

隔离性问题：

脏读：事务B读取到事务A中未提交的数据，然后事务A对刚才的操作进行了回滚操作，那么事务B读取到的数据就无效了，也就是**脏数据**

-- 读到了其他事务未提交的数据

不可重复读：事务A对表内数据修改提交后，事务B再次读取数据，发现和之前读取到的数据不一样

-- 一次读取到的数据其他事务对数据进行了修改，再次读取数据不一致

幻读：事务A对表新增数据并提交以后，事务B再次读取这张表，就会发现数据多了，就像发生了幻觉一样

-- 相同的查询条件，在别的事务添加或者删除数据后，再次查询不一致

隔离级别：

数据库事务的隔离级别一共有4个，由低到高依次为Read uncommitted

,Read committed,Repeatable read,Serializable,这四个隔离级别可以逐个解决脏读，不可查重复读，幻读这几个问题。

		脏读	不可重复读	幻读
未提交读	Read uncommitted	会	会	会
提交读	Read committed	不会	会	会
可重复读	Repeatable read	不会	不会	会
序列化	Serializable	不会	不会	不会

1.ISOLATION_READ_UNCOMMITTED:【很少应用，效果很差，效率也没高哪去】

这个是**事务最低的隔离级别**，它允许另外一个事务可以看到这个事务未提交的数据。

2.ISOLATION_READ_COMMITTED:【Oracle默认的隔离级别 大多数数据库默认】

保证一个事务修改的数据只有在提交以后别的事务才能读取。

3.ISOLATION_REPEATABLE_READ:【Mysql默认的隔离级别】

4.ISOLATION_SERIALIZABLE:【根治所有问题 但是牺牲效率】

花费高代价但是最可靠的事务隔离级别，事务被处理为顺序执行

Oracle中的提交读级别：（默认）

set transaction isolation level read committed;

*：更改数据库的隔离级别必须在开启事务的第一句话来更改

commit：提交 将数据持久化到数据库中

rollback：回滚 将事务中的操作回滚到第一步操作之前，就当什么都没发生

savepoint：还原点 可以回退到指定的位置

显示提交：在事务中**手动commit**

隐式提交：在事务中，如果正在执行DML操作，突然做了一个DDL操作，数据库会自动在DDL操作之前隐式的做一个commit操作

当发生以下操作时，**事务将结束**

1.利用commit/rollback进行事务提交和回滚

2.执行DDL语句时，事务将自动提交

3.如果使用sqlplus时，正常退出事务会自动提交，**非正常退出事务回滚TCL**：事务控制语句

*：事务只会影响DML操作

事务：在一些**列操作**中有多个步骤，只有所有的步骤成功执行那么整个操作才算完成，如果有其中一个环节失败，那么整个操作都算失败。

=====

锁

-- 锁的介绍

锁可以防止事务之间的破坏性交互，约束了最大程度的并发性，数据的完整性

-- 锁的分类

1.**排他锁（X锁）**防止资源共享，也就是当一个事务正在操作数据时，其他事务不可以操作这个事务的数据。

2.**共享锁（S锁）**被锁住的数据**只能被读取**，但是**不能修改**。

-- 锁的类型

DML锁：也就**数据锁**，用于**保护数据**，事务在最开始时添加，通过commit或者rollback释放。

DDL锁：可以**保护数据对象的结构**。Oracle自动的施加的释放。

内部锁：保护数据库的**内部结构**，完全自动调用。

行级锁：也叫**事务锁**，防止数据被同时多个事务进行修改，直到commit或者rollback。

表级锁：防止在修改数据的时候，表结构发生变化。

select name,salary from student for update;

会对student表进行加锁，此时只允许当前session对已经存在的数据进行更新，其他session仍可以进行insert操作。

-- 锁等待和死锁

锁等待也叫**锁冲突**，锁等待会严重影响数据库的性能和日常工作。

死锁，也就锁等待的一种，但是死锁会让事务一直处于锁等待的状态。

-- 查看是否有死锁

```
select sid,serial#,username from v$session where sid in (select blocking_session from v$session);
```

-- 查看死锁的语句

```
select sql_text from v$sql where hash_value in (select sql_hash_value from v$session where sid in(select session_id from v$locked_object));
```

解决死锁：

<https://localhost:1158/em>

性能-->其他监视链接-->实例锁

day03

review

Oracle

SQL

DDL

create alter drop truncate

DML

insert delete update

DQL

select

DCL

grant revoke

TCL

commit rollback savepoint

Oracle的数据类型：

字符型：varchar2(20) char(6)

数值型：number(5,1)

日期型：date timestamp

建表：

```
create table student(  
id number(5),  
name varchar2(20),  
birthday date,  
sal number(5,1)  
);
```

drop table student;

增删改查:

新增数据: insert

insert into student values(1,'zs',sysdate,5000);

insert into student(id,name,sal) values(2,'ls',6000);

删除数据: delete

delete from student;

delete from student where id = 1;

修改数据: update

update student set sal = sal + 500;

update student set sal = sal + 500,name = 'lss' where id = 2;

查询数据: select

select * from student;

select name,sal from student where id = 2;

select name,sal from student where name = 'lss' and sal = 5500;

select name,sal * 1.3 as salary from student;

select name,sal from student where name like '张%';

select name,sal from student where name like '张,%' escape ',';

事务:

四个特性: 原子性, 一致性, 隔离性, 持久性 CIAD ACID

隔离性问题: 脏读, 幻读, 不可重复读

隔离级别: 未提交读, 提交读, 可重复读, 序列化

is null

is not null

函数:

聚组函数: max() min() avg() sum() count()

单行函数: ceil() floor() abs() sign() power() sqrt() round() trunc()

=====

Day 03

函数: (聚组、单行)

聚组函数: max() min() avg() sum() count()

单行函数: ceil() floor() abs() sign() power() sqrt() round() trunc()

字符函数:

upper(): 转换成大写

lower(): 转换成小写

initcap(): 首字母大写

length(): 求长度

select name,upper(name) unname,lower(name) lname,initcap(name) iname,
length(name) lename from student;

substr(a1,a2,a3): 截取字符串

a1: 原字符串

a2: 从哪开始截取

a3: 截取长度

```
select substr('woshizhizhuxia',3) str from dual;  
select substr('woshizhizhuxia',3,9) str from dual;
```

replace(a1,a2,a3): 替换字符串

a1: 原字符串

a2: 要替换的字符

a3: 替换成的字符

```
select replace('woshizhizhuxia','h') str from dual;  
select replace('woshizhizhuxia','h','0000') str from dual;
```

instr(a1,a2,a3,a4): 索引字符串

a1: 原字符串

a2: 想要找到的字符

a3: 从哪开始找

a4: 第几次出现

```
select instr('woshizhizhuxia','h') str from dual;  
select instr('woshizhizhuxia','h',5) str from dual;  
select instr('woshizhizhuxia','h',5,2) str from dual;
```

concat(a1,a2): 拼接字符串

```
select concat(name,sal) str from student;  
select concat(concat(name,birthday),sal) str from student;
```

练习:

新增字段phone, 新增phone数据, 15556785678, 18888888888, 13856987896

查询效果展示如: 1555678, 1888888, 1387896

--14查询员工名字中不包含字母"S"员工 -- 用三种做法

--14查询员工名字中包含字母"S"员工 -- 用三种做法

--15查询员工姓名的第2个字母为"M"的员工信息 -- 用三种做法

转换函数:

to_number():

将一个字符类型的数值转换成数值类型

```
select name,phone from student where to_number(phone) = 15556785678;
```

to_char():

1.将数值类型转换成字符类型

```
select name,sal from student where to_char(sal) = '5150';
```

2.将日期类型转换成字符类型

```
select to_char(sysdate,'yyyy-mm-dd hh24:mi:ss') str from dual;  
select to_char(systimestamp,'yyyy-mm-dd hh24:mi:ss:ff3') str from dual;
```

3.格式化字符串, 常用在货币单位

```
select to_char('1000000','999,999,999.99') str from dual;
```

to_date():

将一个字符类型的日期转换成日期类型

```
select to_date('20230306111313','yyyy-mm-dd hh24:mi:ss') time from dual;
```

练习:

有一个字符串'20230306111616', 取出年, 取出月, 取出日

展示效果:

year	month	day
2023	03	06

日期函数:

两个日期可以相减, 单位是天

sqlplus中修改日期展示格式:

一次性修改方法:

```
alter session set nls_date_format = 'yyyy-mm-dd hh24:mi:ss';
```

永久修改方法: 详见环境变量

yyyy	年	year
mm	月	month 带'月'的月份
ddd	日	年中的日
dd	日	月中的日
d	日	周中的日
hh24	24小时制	
hh12	12小时制	
mi	分	
ss	秒	
xff	毫秒	
ff3	毫秒保留三位	

add_months(): 在某个日期上添加多少个月

```
select add_months(sysdate,3) time from dual;
```

months_between(): 两个日期之间存在多少月

```
select months_between(sysdate,to_date('20230606','yyyy-mm-dd')) time from dual;
```

next_day(): 下一个周几是哪天

```
select next_day(sysdate,'星期一') time from dual;
```

last_day(): 给定日期所在月份的最后一天

```
select last_day(sysdate) time from dual;
```

练习:

--10查询各月倒数第3天入职的员工信息

通用函数:

nvl(原字符串,是空展示什么): 空值处理

```
select name,nvl(birthday,sysdate) birth from student;
```

nvl2(原字符串,不是空展示什么,是空展示什么): 空值处理二代

```
select name,nvl2(birthday,birthday,sysdate) birth from student;
```

练习:

--8查询所有员工 工资和奖金的和

decode(c1,c2,c3,c4,c5...Cx,Cx+1):

c1是原字符串, 从第二个参数开始, 每两个参数看作是一组, 拿每组的第一个参数和c1进行比较, 如果相同则返回该组的第二个参数

相当于:

第一次比较: $c2 == c1 ? c3 :$

第二次比较: $c4 == c1 ? c5 :$

...

如果参数个数是奇数个, 并且最终判断没有相同的值, 则返回空

如果参数个数是偶数个, 并且最终判断没有相同的值, 则返回最后一个参数的值

*: 如果部门编号是10, 则工资涨五百, 如果部门编号是20, 则工资减五百, 其他部门加二百

```
select ename,deptno,sal,decode(deptno,10,sal + 500,20,sal - 500,sal + 200) salary from emp;
```

*: 如果部门编号是10, 则工资涨五百, 如果部门编号是20, 则工资减五百

```
select ename,deptno,sal,decode(deptno,10,sal + 500,20,sal - 500) salary from emp;
```

练习:

如果工种为CLERK, 则工资减三百, 如果工种为SALESMAN, 则工资加三百, 其他工种减二百

条件取值语句:

(case -- 拿来做比较的值

when -- 如果..

then -- 则..

else -- 否则..

end) -- 结束

*: 如果部门编号是10, 则工资涨五百, 如果部门编号是20, 则工资减五百, 其他部门加二百

```
select ename,deptno,sal,(case deptno when 10 then sal + 500 when 20 then sal - 500 else sal + 200 end) salary from emp;
```

*: 如果部门编号是10, 则工资涨五百, 如果部门编号是20, 则工资减五百

```
select ename,deptno,sal,(case deptno when 10 then sal + 500 when 20 then sal - 500 end) salary from emp;
```

练习:

如果工种为CLERK, 则工资减三百, 如果工种为SALESMAN, 则工资加三百, 其他工种减二百

分组: group by

在一张表中, 将某个或者多个列上相同的值划分为一个组, 那么这张表就被分为多个组

*: 如果以字段A分组, 那么只能查询字段A, 其他字段需要以组函数的形式出现

```
select deptno from emp group by deptno;
```

```
select deptno,job from emp group by deptno,job;
```

```
select deptno,count(ename) num from emp group by deptno;
```

练习:

面试题第五题

条件: having

```
select deptno,count(ename) num from emp group by deptno
having count(ename) >= 5;
```

练习:

--29查询最低工资大于2500的各种工作

分组: 聚合统计

去重: distinct

*: 支持单列, 多列的去重

```
select distinct deptno from emp;
select distinct deptno,ename from emp;
```

排序: order by

升序: asc

降序: desc

```
select deptno from emp order by deptno;
select deptno from emp order by deptno asc;
select ename,deptno,sal from emp order by deptno asc,sal desc;
```

查询关键字的优先级:

```
select    列名 -- 优先级高于order by
from      表名 -- 优先级最高
```

连表查询

```
where    条件 -- 优先级次高
group by 分组 -- 优先级次于where
having   条件 -- 优先级一定在group by之后
order by 排序 -- 优先级最低
```

```
select deptno, count(ename) num
from emp
where job = 'CLERK'
group by deptno
having count(ename) >= 2
order by num desc;
```

约束: constraint

主键约束: primary key

主键: 在一张表中能够**唯一**定位一条数据的列称为**主键列**

*: 非空且唯一

```

1  *: 建表时添加主键
2  create table test(
3  id number(5) primary key,
4  name varchar2(20));
5
6  *: 建表后添加主键
7  create table test2(id number(5),name varchar2(20));
8  alter table test2 add constraint zj primary key(id);

```

外键约束: foreign key *: references

外键: 在**子表**中有一个列引用了**父表**中的主键列, 那么这个列在子表中就被称为**外键**

*: 一张表可以有多个外键

1	表名	主键	外键
---	----	----	----

```

1. emp    empno    deptno    子表
    dept    deptno        父表
    emp.deptno = dept.deptno
2. dept    deptno    salno    子表
    salgrade    salno        父表
    dept.salno = salgrade.salno
3. emp    empno    deptno salno 子表
    dept    deptno        父表
    salgrade    salno        父表
    emp.deptno = dept.deptno
    emp.salno = salgrade.salno

```

非空约束: not null

唯一约束: unique

检查约束: check

查看当前用户有哪些约束:

```
select constraint_name,constraint_type,table_name from user_constraints;
```

**:

先建父表, 在建子表, 先删子表, 再删父表

```

create table school(
id number(5) primary key,
name varchar2(20) unique not null,
addr varchar2(20));

insert into school values(1,'北京大学','北京');
insert into school values(2,'清华大学','北京');
insert into school values(3,'厦门大学','厦门');
insert into school values(4,'山东师范','济南');

create table teacher(
id number(5) primary key,
name varchar2(20) not null,
hobby varchar2(20));

```

```
insert into teacher values(1,'琛哥','剑道');
insert into teacher values(2,'周哥','打篮球');
insert into teacher values(3,'乐哥','敲代码');
insert into teacher values(4,'老大','键盘');
```

```
create table class(
id number(5) primary key,
name varchar2(20) not null unique,
tid number(5) references teacher(id));
```

```
insert into class values(1,'ET2210',3);
insert into class values(2,'ET2211',2);
insert into class values(3,'ET2212',1);
```

```
create table student(
id number(5) primary key,
name varchar2(20) not null,
birthday date,
sal number(5,1) check(sal between 5000 and 10000),
email varchar2(30) unique,
sid number(5) references school(id),
cid number(5) references class(id));
```

```
insert into student values(1,'葫芦娃',sysdate,5000,'hlw@126.com',1,2);
insert into student values(2,'金刚',to_date('19981212101010','yyyy-mm-dd hh24:mi:ss'),6000,'jg@163.com',2,2);
insert into student values(3,'蜘蛛侠',to_date('19961212101010','yyyy-mm-dd hh24:mi:ss'),7000,'zzx@yahoo.com',1,1);
insert into student values(4,'白龙',to_date('19951212101010','yyyy-mm-dd hh24:mi:ss'),8000,'bl@et.oak.com',3,3);
```

表名	主键	外键
school	id	
teacher	id	
class	id	tid(teacher.id)
student	id	cid(class.id) sid(school.id)

```
student.cid = class.id
student.sid = school.id
class.tid = teacher.id
```

嵌套查询 = 子查询 = 某些条件是通过查询得出来的

select 嵌套查询 from 嵌套查询 where 嵌套查询 group by 嵌套查询;

*: 谁和葫芦娃一个学校的?

1) 葫芦娃是哪个学校的?

```
select sid from student where name = '葫芦娃';
```

2) 谁还是这个学校的?

```
select name from student where sid = 1;
```

```
select name from student where sid = (select sid from student where name = '葫芦娃') and name <> '葫芦娃';
```


select name from student where (sid,name) = (select sid,name from student where name = '葫芦娃');

练习:

--24查询工资比SMITH员工工资高的所有员工信息

*: 学生都上那些学校?

select student.name,(select school.name from school where student.sid = school.id) schname
from student;

练习:

--2查询所有工种为CLERK的员工的工号、员工名和部门名。

emp.deptno = dept.deptno

=====

select school.name school,teacher.name teacher,class.name class,student.name student from
student join school on student.sid = school.id join class on class.id = student.cid join teacher on
class.tid = teacher.id;

day04

review

函数

聚组函数: max() min() avg() sum() count()

单行函数: ceil() floor() abs() sign() power() sqrt() round() trunc()

日期函数: add_months() months_between() next_day() last_day()

字符函数: upper() lower() initcap() length()

substr(3/2) replace(3/2) instr(4/3/2) concat(2)

lpad() rpad() trim() ltrim() rtrim()

转换函数: **to_number() to_char() to_date()**

通用函数: **nvl(2) nvl2(3)**

lpad(a1,a2,a3): 左侧补全

rpadd(a1,a2,a3): 右侧补全

select lpad(phone,11,'138') phone from test;

trim(): 默认去除两侧空格

trim(a1 from a2): 把a2的两侧去除a1

select trim('a' from 'aaabaabaaa') str from dual;

ltrim(): 左侧去除

rtrim(): 右侧去除

select ltrim(' abc') str from dual;

select ltrim(' abc',' ') str from dual;

select ltrim(' abc','a') str from dual;

,decode() 别名,

,(case when then else end) 别名,

分组: group by

条件: having

去重: distinct

排序: order by

升序: asc

降序: desc

查询关键字的优先级:

select

from

连表查询

where

group by

having

order by

约束:

主键约束: **primary key**

外键约束: **foreign key** *: **references**

非空约束: not null

唯一约束: unique

检查约束: check

嵌套查询: 子查询

```
select empno,ename,(select dname from dept where emp.deptno = dept.deptno) dname
from emp
where job = 'CLERK';
```

自查询:

```
select * from A,B;
select empno,ename,dname
from emp,dept
where emp.deptno = dept.deptno and job = 'CLERK';
```

=====

复制表:

1.复制表结构和表数据

```
create table emp2 as select * from emp;
```

2.复制表结构

```
create table emp3 as select * from emp where 1 = 2;
```

3.复制指定列的表结构和表数据

```
create table emp4 as select empno,ename from emp;
```

4.复制表在新表中给列重命名

```
create table emp5(id,name) as select empno,ename from emp;
```

5.一次性添加多条数据

```
insert into emp3 select * from emp;
```

Day 04

连表查询：

内连接：inner join on = join on

- *：两张表有关联关系的数据
- *：from 表 join 表 on 关联关系

```
1 select student.name stuname,school.name schname
2 from student
3 join school on student.sid = school.id;
4
5 select student.name stuname,school.name schname,class.name cname
6 from student
7 join school on student.sid = school.id
8 join class on class.ID = student.cid
9 where student.name = '葫芦娃';
```

练习：

面试题的第二题

练习：

- 1.展示学校名称，老师姓名，班级名称，学生姓名（一条语句）
- 2.每个班级的学生平均工资是多少，要求展示班级名称
- 3.每个班级有多少人，分别都是哪些同学，要求展示班级名称，按照人数降序排列

外连接：不仅包含有关联关系的数据，还包含没有关联关系的数据

左外连接：left join on

- *：以左表为主，左表的所有数据都展示，对应不上的右表的数据以空展示
- *：左表关联上右表的数据 + 关联不上的左表的数据
- *：from 左表 left join 右表 on 关联关系

```
1 select school.name schname,student.name stuname
2 from school
3 left join student on student.sid = school.id;
```

练习：

--27查询所有部门及其员工信息，包括那些没有员工的部门

右外连接：right join on

- *：以右表为主，右表的所有数据都展示，对应不上的左表的数据以空展示
- *：左表关联上右表的数据 + 关联不上的右表的数据
- *：from 左表 right join 右表 on 关联关系

```

1 select school.name schname,student.name stuname
2 from student
3 right join school on student.sid = school.id;
4
5 select school.name schname,student.name stuname,class.name cname
6 from student
7 left join school on student.sid = school.id
8 right join class on student.cid = class.id;

```

全外连接: full join on

- *: 两张表的数据都展示
- *: 左表关联上右表的数据 + 关联不上的左表的数据 + 关联不上的右表的数据

```

1 select student.name stuname,school.name schname
2 from student
3 full join school on student.sid = school.id;

```

交叉连接: cross join

- *: 两张表的数据一一对应, 交叉形成最后结果
- ```

select book.bname,student.name
from student
cross join book;

```

### 自查询:

```

select school.name schname,student.name stuname
from school,student
where student.sid = school.id;

```

```

select school.name schname,student.name stuname
from school,student
where student.sid = school.id(+);

```

```

select school.name schname,student.name stuname
from school,student
where student.sid(+) = school.id;

```

### 伪列:

#### rownum = 行号

- \*: Oracle用来**分页**的依据, 一个**有序的整数列值**, 每多一条自动加一
- \*: 不能使用表名.rownum的写法, 如果rownum用在where之后, rownum>=1 rownum<=任意值

1) 第一页数据, 编号1-10

```
select content from lyric where rownum <= 10;
```

2) 第二页数据, 编号11-20

```

select content from
(select rownum rn,content from lyric where rownum <= 20) lyrics
where lyrics.rn >= 11;

```

3) 第三页数据, 编号21-30

```

select content from
(select rownum rn,content from lyric where rownum <= 30) lyrics

```

where lyrics.rn >= 21;

#### 4) 带有排序效果的分页

select content from

(select rownum rn,content from (select \* from lyric order by content desc) where rownum <= 20)

lyrics

where lyrics.rn >= 11;

\*\*\*:

curpage: 当前页    2    3

pagesize: 每页条数 10    25

当前页的开始数: (curpage - 1) \* pagesize + 1    11    51

当前页的截止数: curpage \* pagesize                    20    75

**rowid**: 数据库会给每条数据添加物理地址唯一标识

\*: 适用于删除完全重复的数据

delete from lyric where rowid not in

(select max(rowid) from lyric group by content);

练习:

删除表中5-10条数据

**in: 表示是查询结果中的任意一个值**

字段 in (值1,值2...)

相当于:

字段 = 值1 or 字段 = 值2 or ...

select id,name from school where id **in** (select sid from student);

练习:

--35列出工资等于30号部门中某个员工工资的所有员工的姓名和工资。

**not in: 表示不能是查询结果中的任意一个值**

字段 not in (值1,值2...)

相当于:

字段 <> 值1 and 字段 <> 值2 and ...

select id,name from school where id not in (select sid from student);

**some/any: 用法和in相同, in用在无符号的情况下, some/any用在有符号的情况下**

select id,name from school where id = some (select sid from student);

select id,name from school where id > any (select sid from student);

**all: 比所有值都大, 或者比所有值都小**

select id,name from school where id > all (select sid from student);

\*\*\*:

any >min

<any max

<all <min

练习:

--36查询工资高于30号部门中工作的所有员工的工资的员工的姓名和工资

\*: 在某些情况下, in的效率低, 所以用exists代替

**exists: 存在**

**not exists: 不存在**

select id,name from school where exists (select sid from student  
where student.sid = school.id);

**联合关键字:**

**union: 结果唯一, 并且按照默认顺序排序**

**union all: 结果不唯一**

select id,name from school where name like '山东%'  
union

select id,name from school where name like '%大学';

select id,name from school where name like '山东%'  
union all

select id,name from school where name like '%大学';

intersect: 求交集

select id,name from school where name like '山东%'  
intersect

select id,name from school where name like '%大学';

minus: 从第一个结果集中减去第二个结果集中重复的数据

select id,name from school where name like '山东%'  
minus

select id,name from school where name like '%大学';

=====

## day05

---

### review

连表查询

内连接 join on

外连接

左外连接 left join on

右外连接 right join on

全外连接 full join on

交叉连接 cross join

自查询

伪列:

rownum

select content from

(select content,rownum rn from lyric where rownum <= 10) lyrics  
where lyrics.rn >= 5;

rowid

in

not in

some/any

all

exists

not exists

union

union all

intersect

minus

=====

## Day 05

### 视图: view

将查询结果保存在数据库中, 下次查询**不需要编译**, 直接从数据库中获取

\*: 经过函数修饰的列一定要起别名

创建视图:

#### create view 视图名 as 查询语句;

```
create view v_name as
select student.name stuname,school.name schname
from student
left join school on student.sid = school.id;
```

```
select * from v_name;
select stuname,schname from v_name;
```

删除视图:

#### drop view v\_name;

减少编译时间, 提高查询效率, 避免没有权限的用户查看隐私字段  
简单视图数据可以动态来源于原表, 复杂视图需要手动编译(alter view 视图名 compile;)

练习:

面试题第四题

### 序列: sequence

\*: Oracle通过调用序列**实现主键自增**, 一个有序的整数列值

\*: 在新的会话中需要调用序列的下一个值才能查看当前值

\*: 调用序列的下一个值会作为下一次调用的初始值

创建序列:

```
create sequence seq_test
start with 1 -- 从1开始
increment by 1 -- 一次增长1
minvalue 1 -- 最小值
maxvalue 100 -- 最大值
```

nocycle            -- 不循环  
nocache;            -- 不缓存

### 1.消除延迟段创建特性 (序列)

```
alter system set deferred_segment_creation = false;
```

### 2.创建表

```
create table test2(id number(5),name varchar2(20));
```

### 3.创建序列

```
create sequence seq_test2;
```

### 4.新增数据

```
insert into test2 values(seq_test2.nextval,'橘子');
insert into test2 values(seq_test2.nextval,'西瓜');
```

### 5.查询数据

```
select * from test2;
```

查看当前用户下有哪些序列：

```
select * from user_sequences;
```

查看序列的当前值：

```
select seq_test.currval from dual;
```

查看序列的下一个值：

```
select seq_test.nextval from dual;
```

删除序列：

```
drop sequence seq_test;
```

### 索引：index

\*：数据库会在具有**唯一约束的列上**自动添加**唯一索引**

### 创建索引：

```
create index 索引名 on 表名(列名);
create index ind_name on student(name);
```

索引类型：

### 普通索引：normal

```
create index 索引名 on 表名(列名);
create index ind_name on student(name);
```



## 唯一索引: unique

create **unique index** 索引名 on 表名(列名);  
create **unique** index ind\_sid on student(sid);

## 位图索引(分类索引): bitmap

常用在数据量比较大, 基数比较小

create **bitmap index** 索引名 on 表名(列名);  
create **bitmap** index ind\_cid on student(cid);

## 函数索引:

create index 索引名 on 表名(函数(列名));  
create index ind\_lengthname on school(**length(name)**);

## 删除索引:

drop index ind\_sid;

**数据量大, 查询多的列适合添加索引, 增改数据数据库会耗费资源去维护索引**

## 10.什么情况会使索引失效?

10.1使用**like关键字**模糊查询时, % 放在前面索引不起作用, 只有“%”不在第一个位置, 索引才会生效 (like '%文'-索引不起作用)

10.2使用联合索引时, 只有查询条件中使用了这些字段中的第一个字段, 索引才会生效

10.3使用**OR关键字**的查询, 查询语句的查询条件中只有OR关键字, 且OR前后的两个条件中的列都是索引时, 索引才会生效, 否则索引不生效。

10.4尽量避免在where子句中使用**!=或<>**操作符, 否则引擎将放弃使用索引而进行**全表扫描**。

10.5对查询进行优化, 应尽量避免全表扫描, 首先应考虑在where以及order by涉及的列上建立索引。

10.6应尽量**避免**在 where 子句中**对字段进行表达式操作**, 这将导致引擎放弃使用索引而进行全表扫描。

如:

```
select id from t where num/2=100
```

应改为:

```
select id from t where num=100*2
```

10.7尽量避免在where子句中对字段进行**函数操作**, 将导致引擎放弃使用索引而进行全表扫描。

10.8不要在 where 子句中的**“=”左边进行函数、算术运算或其他表达式运算**, 否则系统将可能无法正确使用索引。

10.9并不是所有的索引对查询都有效, sql是根据表中的数据来进行查询优化的, **当索引列有大量数据重复时**, sql查询不会去利用索引。

10.10索引并不是越多越好, 索引固然可以提高相应的 select 的效率, 但同时也降低了 insert 及 update 的效率,

因为 insert 或 update 时有可能会重建索引, 所以怎样建索引需要慎重考虑, 视具体情况而定。 —

**个表的索引数最好不要超过6个,**

若太多则应考虑一些不常使用到的列上建的索引是否有必要。

## 表空间: tablespace

在一段内存中多数存储的是**表**, 所以称为表空间

### 创建表空间：

```
create tablespace table_user -- 表空间的名字
datafile 'D:/table_user.dbf' -- 表空间的位置
size 5M -- 初始大小
autoextend on next 5M -- 下一次自动拓展多少
maxsize 100M; -- 最大值
```

### 创建用户并给分配默认的表空间：

```
create user ET identified by etoak default tablespace table_user;
```

### 创建用户未指定表空间：

```
create user ET2 identified by etoak;
```

### 给用户修改表空间：

```
alter user ET2 default tablespace table_user;
```

### 删除表空间：

如果删除用户指定的表空间，该用户仍指向原表空间位置，需要手动指定一个有效的表空间位置

```
drop tablespace table_user including contents and datafiles;
```

## sql语句优化：

- 1.建议少用\*代替列名
- 2.用**exists**代替**in**
- 3.多表连接时，尽量**减少**表的查询次数
- 4.删除全表数据的时候用**truncate**代替**delete**
- 5.合理使用**索引**（详见索引）
- 6.sql语句**尽量大写**，Oracle会默认把小写转换成大写在执行
- 7.在保证语句完整的情况下，**多使用commit**(begin...end)
- 8.**优化group by**,将不需要的数据尽量在分组之前过滤掉
- 9.连表查询的时候**尽量使用表的别名**，减少解析时间
- 10.表连接在where之前，**where条件过滤顺序**，**能够更多的过滤数据的放在前面**

=====

## MySQL

MySQL是一个关系型数据库管理系统，是由瑞典的MySQLAB公司研发，之后被Oracle 公司收购，目前是甲骨文旗下的产品。

MySQL是目前最流行的关系型数据库管理系统之一在小型的web应用中，

MySQL是最好的RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件。

MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

但是注意，MySQL中存在大量的database，用户首先选择database，之后在database中建立表，这一点与Oracle不同。

MySQL同样使用SQL语言，此语言是用于访问数据库的最常用标准化语言。

MySQL软件采用了**双授权政策**，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

由于其社区版的性能卓越，安装简便，搭配Apache旗下的Tomcat等工具可以组成良好的开发环境。

## 1: 使用CMD连接mysql

```
mysql -u root -p etoak
```

\*:在cmd中使用mysql命令时，我们可以直接指定要连接的数据源

```
mysql -u root -p etoak test
```

## 2: 在开始菜单中连接mysql

开始->mysql->Line Client->enter password

### 查看当前有哪些可用的数据源:

```
show databases;
```

### 创建数据源:

```
create database etoak;
```

### 使用某个数据源:

```
use etoak;
```

### 查看当前数据源下有哪些表:

```
show tables;
```

### 查看表结构:

```
desc 表名;
```

```
explain test;
```

```
show columns from test;
```

### mysql中的自动增长:

```
create table test(
tid int primary key auto_increment,
tname varchar(10),
birth date
);
```

### mysql中添加外键:

```
create table test2(
id int primary key auto_increment,
name varchar(10),
tid int,
foreign key(tid) references test(tid));
```

```
alter table test2 add foreign key(tid) references test(tid);
```

### 增加:

```
insert into test(tid,tname,birth) values(null,'葫芦娃','2018-04-16');
insert into test values(null,'金刚','2018-04-16');
insert into test(tid,tname,birth) values(null,'喜洋洋','2018-04-16');
insert into test(tid,tname,birth) values(null,'美羊羊','2018-04-16');
```

```
insert into test(tid,tname,birth) values(null,'懒洋洋','2018-04-16');
insert into test(tid,tname,birth) values(null,'沸羊羊','2018-04-16');
insert into test(tid,tname,birth) values(null,'灰太狼','2018-04-16');
insert into test(tid,tname,birth) values(null,'暖洋洋','2018-04-16');
insert into test(tid,tname,birth) values(null,'慢羊羊','2018-04-16');
insert into test(tid,tname,birth) values(null,'潇洒哥','2018-04-16');
insert into test(tid,tname,birth) values(null,'黑大帅','2018-04-16');
```

#### 删除:

```
delete from test where tid=3;
```

#### 修改:

```
update test set tname = '金刚' where tid = 2;
```

#### 删除表:

```
drop table if exists test;
```

删除数据保留表结构:

```
truncate test;
```

#### 分页:

如果limit之后只有一个值, 从第一条开始数五条数据

```
select * from test limit 5;
```

如果有两个参数: 第一个数为>5开始,第二个数多少个条数

```
select * from test limit 5,5;
```

#### 拿取表中记录数:

```
select count(*) from test;
```

### oracle和mysql

#### 1: 主键

mysql:一般使用自动增长类型, 在创建表的时候只要指定表的主键为auto\_increment, 插入记录时, 不需要再指定该记录的主键值;

oracle:没有自动增长类型, 主键一般使用序列, 插入记录时将序列号的下一个值赋给该字段即可。

#### 2: 分页

mysql:limit开始位置, 记录个数;

oracle:通过rownum字段标明位置进行分页。

#### 3: 分组

mysql:group by 语句可以select 没有被分组的字段;

```
group_concat()
```

```
select tname,group_concat(tid)
```

```
from test group by tname;
```

oracle:select语句后必须有group by后分组的字段。

```
wm_concat()
```

#### 4: 引号

mysql:单引号, 双引号都可以;

oracle:一般不准使用双引号, 用了会报error。

#### 5: 转换数据

mysql:case when;

oracle:decode();

#### 6: 空值处理

mysql:非空字段也有空的内容;

oracle:定义了非空字段就不容许有空的内容。

按照mysql的not null来定义oracle的表结构, 导出数据时候会产生错误;

因此导出数据时要对空字符进行判断, 如果为null或空字符, 需要把它改成一个空格的字符串。

=====

## Oracle面试题

### DB

#### 1.sql\*\*查询数据库重复的数据\*\*

```
select * from emp
```

```
where deptno in (select deptno from emp group by deptno having count(deptno) > 1);
```

删除重复数据:

```
delete from lyric
```

```
where content in (select content from lyric group by content having count(content) > 1) and rowid
not in (select min(rowid) from lyric group by content having count(content)>1);
```

#### 2.\*\*两个表 一个学生信息 一个成绩表 两表信息同时查出来 (级联) \*\*

```
full join on
```

#### 3.\*\*存储过程, 触发器, Oracle的视图\*\*

详见课件

--创建触发器, 记录表删除数据

--创建表

```
CREATE TABLE employeee(
```

```
id VARCHAR2(4) NOT NULL,
```

```
name VARCHAR2(15) NOT NULL,
```

```
age NUMBER(2) NOT NULL,
```

```
sex CHAR NOT NULL
```

```
);
```

--插入数据

```
INSERT INTO employee VALUES('e101','zhao',23,'M');
```

```
INSERT INTO employee VALUES('e102','jian',21,'F');
```

--创建记录表(包含数据记录)

```
CREATE TABLE old_employee AS SELECT * FROM employee;
```

--创建触发器

```
CREATE OR REPLACE TRIGGER TIG_OLD_EMP
```

```
AFTER DELETE ON EMPLOYEE
```

```
FOR EACH ROW --语句级触发，即每一行触发一次
```

```
BEGIN
```

```
INSERT INTO OLD_EMPLOYEE VALUES (:OLD.ID, :OLD.NAME, :OLD.AGE, :OLD.SEX); --:old代表旧值
```

```
END;
```

```
/
```

--下面进行测试

```
DELETE employee;
```

```
DELETE employee where id = 'e101';
```

```
SELECT * FROM old_employee;
```

```
select * from employee;
```

#### **4.Oracle\*\*的分组函数，降序函数\*\***

分组函数：max() min() avg() sum() count()

排序：order by 降序：desc 升序：asc

#### **5.\*\*索引有哪些\*\***

普通索引：normal

唯一索引：unique

位图索引：bitmap 又称分类索引

函数索引：属于普通索引

#### **6.\*\*用sql语句更改列名\*\***

```
alter table test rename column name to name1;
```

#### **7.union 和 union all的区别是什么**

union: 对两个结果集进行并集操作, 不包括重复行, 同时进行默认规则的排序;

union all: 对两个结果集进行并集操作, 包括重复行, 不进行排序;

## 8. 了解过oracle执行计划吗

### 8.1.什么是Oracle执行计划

执行计划是一条查询语句在Oracle中执行过程或者访问路径的描述.

### 8.2.查看Oracle执行计划

执行计划常用的列字段解释

**基数**: 返回的结果集行数

**字节**: 执行该步骤后返回的字节数

**耗费(cust),CPU耗费**: Oracle估计的该步骤的执行成本, 用于说明SQL执行的代价, 理论上越小越好.

### 8.3 看懂Oracle执行计划



<https://images2017.cnblogs.com/blog/966901/201710/966901-20171029170639789-1320421867.png>

#### 8.3.1执行顺序

根据缩进来判断, **缩进最多的最先执行(缩进相同时, 最上面的最先执行)**

## 9.sql\*\*怎么优化\*\*

### 9.1 where字句中的连接顺序

oracle的解析按照从上而下解析, 因此表之间的连接必须写在where条件之前:

例如:

低效:

```
select .. from emp e where sal > 50000 and job = 'manager'
```

```
and 25 < (select count(*) from emp where mgr=e.empno);
```

高效:

```
select .. from emp e where 25 < (select count(*) from emp where mgr=e.empno)
```

```
and sal > 50000 and job = 'manager';
```

### 9.2 通配符\*的使用

Sql在执行带通配符的语句时, **如果%在首位**, 那么在字段上建立的主键或者索引将会失效!

应该避免类似语句的出现

```
Select name from user_info where name='%A';
```

### 9.3 使用truncate代替delete

当删除表时，使用delete执行操作，回滚端用来存放可恢复的信息，当没有提交事务的时候，执行回滚事务，数据会恢复到执行delete操作之前，而当用truncate时，回滚端则不会存放可恢复的信息，减少资源的调用。

#### 9.4 用where子句替换HAVING子句

避免使用 HAVING 子句, HAVING 只会在检索出所有记录之后才对结果集进行过滤. 这个处理需要排序, 总计等操作. 如果能通过 WHERE 子句限制记录的数目, 那就能减少这方面的开销.

#### 9.5 减少对表的查询

低效:

```
Select tab_name from tables where tab_name = (select tab_name from tab_columns where version = 604) and db_ver= (select db_ver from tab_columns where version = 604)
```

高效:

```
select tab_name from tables where (tab_name,db_ver) =
(select tab_name,db_ver from tab_columns where version =604)
```

#### 9.6 用in代替or

低效:

```
Select.. from location where loc_id = 10 or loc_id = 20 or loc_id = 30
```

高效:

```
Select..from location where loc_in in (10,20,30);
```

#### 9.7 删除重复数据

最高效的删除重复记录的方法

```
Delete from ur_user_info a Where a.rowid>(select min(b.rowid) From ur_user_info b Where b.
uid=a. uid);
```

#### 9.8 避免使用耗费资源的操作

带有DISTINCT, UNION, MINUS, INTERSECT, ORDER BY的SQL语句会启动SQL引擎执行耗费资源的排序 (SORT)功能. DISTINCT需要一次排序操作, 而其他的至少需要执行两次排序.

例如, 一个UNION查询, 其中每个查询都带有GROUP BY子句, GROUP BY会触发嵌入排序(NESTED SORT); 这样, 每个查询需要执行一次排序, 然后在执行UNION时, 又一个唯一排序(SORT UNIQUE)操作被执行而且它只能在前面的嵌入排序结束后才能开始执行. 嵌入的排序的深度会大大影响查询的效率.

#### 9.9 自动选择索引

如果表中有两个以上 (包括两个) 索引, 其中有一个唯一性索引, 而其他是非唯一性. 在这种情况下, ORACLE将使用唯一性索引而完全忽略非唯一性索引.

举例:

```
select ename from emp where empno = 2326 and deptno = 20 ;这里, 只有empno上的索引是唯一性的, 所以empno索引将用来检索记录.
```

```
table access by rowid on emp index unique scan on emp_no_idx;
```

#### 9.10 至少要包含组合索引的第一列



如果索引是建立在多个列上, 只有在它的第一个列(leading column)被where子句引用时, 优化器才会选择使用该索引. 当仅引用索引的第二个列时, 优化器使用了全表扫描而忽略了索引。

#### 9.11 避免在索引列上使用函数

WHERE子句中, 如果索引列是函数的一部分. 优化器将不使用索引而使用全表扫描.

低效:

```
select .. from dept where sal * 12 > 25000;
```

高效:

```
select .. from dept where sal > 25000/12;
```

#### 9.12 避免出现索引列自动转换

当比较不同类型的数据时, ORACLE自动对列进行简单的类型转换.

假设user\_no是一个字符类型的索引列.

```
select user_no, user_name, address from user_files where user_no = 109204421;
```

这个语句被ORACLE转换为:

```
select user_no, user_name, address from user_files where to_number(user_no) = 109204421;
```

因为内部发生的类型转换, 这个索引将不会被用到!

#### 9.13 使用DECODE来减少处理时间

例如:

```
select count(*) sum(sal) from emp where dept_no = 0020 and ename like 'smith%';
```

```
select count(*) sum(sal) from emp where dept_no = 0030 and ename like 'smith%';
```

你可以用DECODE函数高效地得到相同结果

```
select count(decode(dept_no, 0020, 'x', null)) d0020_count,
 count(decode(dept_no, 0030, 'x', null)) d0030_count,
 sum(decode(dept_no, 0020, sal, null)) d0020_sal,
 sum(decode(dept_no, 0030, sal, null)) d0030_sal
from emp where ename like 'smith%';
```

#### 9.14 Order by语句

(a). ORDER BY语句决定了Oracle如何将返回的查询结果排序。Order by语句对要排序的列没有什么特别的限制, 也可以将函数加入列中。任何在Order by语句的非索引项或者有计算表达式都将降低查询速度。

(b). order by语句以找出非索引项或者表达式, 它们会降低性能。解决这个问题的办法就是重写order by语句以使用索引, 也可以为所使用的列建立另外一个索引, 同时应**绝对避免在order by子句中使用表达式。**

#### 9.15 用索引提高效率

**索引**是表的一个概念部分,用来提高检索数据的效率, ORACLE使用了一个复杂的自平衡B-tree结构. 通常, 通过索引查询数据比全表扫描要快. 当ORACLE找出执行查询和Update语句的最佳路径时, ORACLE优化器将使用索引. 同样在联结多个表时使用索引也可以提高效率. 另一个使用索引的好处是,它提供了主键(primary key)的唯一性验证. 通常, 在大型表中使用索引特别有效. 当然,你也会发现, 在扫描小表时,使用索引同样能提高效率. 虽然使用索引能得到查询效率的提高,但是我们也必须注意到它的代价. **索引需要空间来存储,也需要定期维护**, 每当有记录在表中增减或索引列被修改时, 索引本身也会被修改. 这意味着每条记录的INSERT , UPDATE将为此多付出4 , 5 次的磁盘I/O . 因为索引需要额外的存储空间和处理,那些不必要的索引反而会使查询反应时间变慢. **定期的重构索引是有必要的。**

#### 9.16 用>= 替代 >

如果DEPTNO上有一个索引。

高效: SELECT \* FROM EMP WHERE DEPTNO >=4;

低效: SELECT \* FROM EMP WHERE DEPTNO >3;

#### 9.17 通过使用>=、<=等, 避免使用NOT命令

例子:

select \* from employee where salary <> 3000;

对这个查询, 可以改写为不使用NOT:

select \* from employee where salary<3000 or salary>3000;

虽然这两种查询的结果一样, 但是第二种查询方案会比第一种查询方案更快些。

第二种查询允许Oracle对salary列使用索引, 而第一种查询则不能使用索引。

#### 9.18 字符型字段的引号

比如有的表PHONE\_NO字段是**CHAR型**,而且创建有索引,

但在WHERE条件中忘记了加引号, 就不会用到索引。

WHERE PHONE\_NO='1392020222'

WHERE PHONE\_NO=1392020222

### 10.\*\*什么情况会使索引失效? \*\*

10.1使用like关键字模糊查询时, % 放在前面索引不起作用, 只有“%”不在第一个位置, 索引才会生效 (like '%文'-索引不起作用)

10.2使用联合索引时, 只有查询条件中使用了这些字段中的第一个字段, 索引才会生效

10.3使用OR关键字的查询, 查询语句的查询条件中只有OR关键字, 且OR前后的两个条件中的列都是索引时, 索引才会生效, 否则索引不生效。

10.4尽量避免在where子句中使用!=或<>操作符, 否则引擎将放弃使用索引而进行全表扫描。

10.5对查询进行优化, 应尽量避免全表扫描, 首先应考虑在where以及order by涉及的列上建立索引。

10.6应尽量避免在 where 子句中对字段进行表达式操作, 这将导致引擎放弃使用索引而进行全表扫描。如:

select id from t where num/2=100

应改为:

```
select id from t where num=100*2
```

10.7尽量避免在where子句中对字段进行函数操作,将导致引擎放弃使用索引而进行全表扫描。

10.8不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

10.9并不是所有的索引对查询都有效，sql是根据表中的数据来进行查询优化的，当索引列有大量数据重复时，sql查询不会去利用索引。

10.10索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，

因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，

若太多则应考虑一些不常使用到的列上建的索引是否有必要。

10.11尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。

这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。


## 11.Oracle\*\*行转列，列转行\*\*

### pivot 列转行

测试数据 (id, 类型名称, 销售数量)，案例：根据水果的类型查询出一条数据显示出每种类型的销售数量。

SQL Code


```
create table demo(id int,name varchar(20),nums int); ---- 创建表
insert into demo values(1, '苹果', 1000);
insert into demo values(2, '苹果', 2000);
insert into demo values(3, '苹果', 4000);
insert into demo values(4, '橘子', 5000);
insert into demo values(5, '橘子', 3000);
insert into demo values(6, '葡萄', 3500);
insert into demo values(7, '芒果', 4200);
insert into demo values(8, '芒果', 5500);
```

 <https://images0.cnblogs.com/blog/180599/201507/131054006573970.png>

分组查询（当然这是不符合查询一条数据的要求的）

SQL Code

```
select name, sum(nums) nums from demo group by name
```

 <https://images0.cnblogs.com/blog/180599/201507/131054016732028.png>

行转列查询

SQL Code

```
select * from (select name, nums from demo) pivot (sum(nums) for name in ('苹果' 苹果, '橘子', '葡萄', '芒果'));
```



<https://images0.cnblogs.com/blog/180599/201507/131054022669211.png>

注意：pivot（聚合函数 for 列名 in（类型）），其中 in("") 中可以指定别名，in 中还可以指定子查询，比如 select distinct code from customers

当然也可以不使用pivot函数，等同于下列语句，只是代码比较长，容易理解

SQL Code

```
select * from (select sum(nums) 苹果 from demo where name = '苹果'),
 (select sum(nums) 橘子 from demo where name = '橘子'),
 (select sum(nums) 葡萄 from demo where name = '葡萄'),
 (select sum(nums) 芒果 from demo where name = '芒果');
```

### **unpivot** 行转列

顾名思义就是将多列转换成1列中去

案例：现在有一个水果表，记录了4个季度的销售数量，现在要将每种水果的每个季度的销售情况用多行数据展示。

创建表和数据

SQL Code

```
create table Fruit(id int,name varchar(20), Q1 int, Q2 int, Q3 int, Q4 int);
insert into Fruit values(1,'苹果',1000,2000,3300,5000);
insert into Fruit values(2,'橘子',3000,3000,3200,1500);
insert into Fruit values(3,'香蕉',2500,3500,2200,2500);
insert into Fruit values(4,'葡萄',1500,2500,1200,3500);
select * from Fruit
```



<https://images0.cnblogs.com/blog/180599/201507/131054041578367.png>

列转行查询

SQL Code

```
select id , name, jidu, xiaoshou from Fruit unpivot (xiaoshou for jidu in (q1, q2, q3, q4));
```

注意：unpivot没有聚合函数，xiaoshou、jidu字段也是临时的变量



<https://images0.cnblogs.com/blog/180599/201507/131054060487524.png>

**12.\*\*乐观锁和悲观锁。悲观锁是列锁还是行锁\*\***

**悲观锁**(Pessimistic Lock), 顾名思义, 就是很悲观, **每次去拿数据的时候都认为别人会修改**, 所以每次在拿数据的时候都会上锁, 这样别人想拿这个数据就会block直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制, 比如**行锁, 表锁等**, 读锁, 写锁等, 都是在做操作之前先上锁。它指的是对数据被外界 (包括本系统当前的其他事务, 以及来自外部系统的事务处理) 修改持保守态度, 因此, 在整个数据处理过程中, **将数据处于锁定状态**。悲观锁的实现, 往往依靠数据库提供的锁机制 (**也只有数据库层提供的锁机制才能真正保证数据访问的排他性**, 否则, 即使在本系统中实现了加锁机制, 也无法保证外部系统不会修改数据)。

**乐观锁**(Optimistic Lock), 顾名思义, 就是很乐观, **每次去拿数据的时候都认为别人不会修改**, 所以不会上锁, 但是在更新的时候会判断一下在此期间别人有没有去更新这个数据, 可以使用版本号等机制。乐观锁**适用于多读的应用类型**, 这样可以提高吞吐量, 像数据库如果提供类似于write\_condition机制的其实都是提供的乐观锁。

两种锁各有优缺点, 不可认为一种好于另一种, 像乐观锁适用于写比较少的情况下, 即冲突真的很少发生的时候, 这样可以省去了锁的开销, 加大了系统的整个吞吐量。但如果经常产生冲突, 上层应用会不断的进行retry, 这样反倒是降低了性能, 所以这种情况下用悲观锁就比较合适。

本质上, 数据库的乐观锁做法和悲观锁做法主要就是解决下面假设的场景, **避免丢失更新问题**:

一个比较清楚的场景

下面这个假设的实际场景可以比较清楚的帮助我们理解这个问题:

假设当当网上用户下单买了本书, 这时数据库中有条订单号为001的订单, 其中有个status字段是'有效', 表示该订单是有效的;

后台管理人员查询到这条001的订单, 并且看到状态是有效的

用户发现下单的时候下错了, 于是撤销订单, 假设运行这样一条SQL: update order\_table set status = '取消' where order\_id = 001;

后台管理人员由于在b这步看到状态有效的, 这时, 虽然用户在c这步已经撤销了订单, 可是管理人员并未刷新界面, 看到的订单状态还是有效的, 于是点击"发货"按钮, 将该订单发到物流部门, 同时运行类似如下SQL, 将订单状态改成已发货:update order\_table set status = '已发货' where order\_id = 001

**简言之: 读取频繁使用乐观锁, 写入频繁使用悲观锁。**

**13.\*\*有100条数据 如何查出重复数据在3条以上的数据数\*\***

```
select deptno, count(*) from emp group by deptno having count(1) >=3;
```

**14.SQL\*\*语句里拼接字符串用什么\*\***

```
concat(a1,a2) ||
```

**15.\*\*常用的分析函数\*\***

15.1分析函数语法

```
function_name(...)over(<partition_Clause><order by_Clause><windowing_Clause>);
```

function\_name(): 函数名称

argument: 参数

**over(): 开窗函数**

partition\_Clause: 分区子句, 数据记录集分组, group by...

order by\_Clause: 排序子句, 数据记录集排序, order by...

windowing\_Clause: 开窗子句, 定义分析函数在操作行的集合, 三种开窗方式: rows、range、Specifying

注: 使用开窗子句时一定要有序子句!!!

## 15.2 分析函数汇总

15.2.1 **count() over()** : 统计分区中各组的行数, partition by 可选, order by 可选

select ename,esex,eage,count(\*) **over()** from emp; --总计数

select ename,esex,eage,count(\*) **over(order by eage)** from emp; --递加计数

select ename,esex,eage,count(\*) **over(partition by esex)** from emp; --分组计数

select ename,esex,eage,count(\*) **over(partition by esex order by eage)** from emp; --分组递加计数

划分

15.2.2 **sum() over()** : 统计分区中记录的总和, partition by 可选, order by 可选

select ename,esex,eage,sum(salary) **over()** from emp; --总累计求和

select ename,esex,eage,sum(salary) **over(order by eage)** from emp; --递加累计求和

select ename,esex,eage,sum(salary) **over(partition by esex)** from emp; --分组累计求和

select ename,esex,eage,sum(salary) **over(partition by esex order by eage)** from emp; --分组递加累计求和

15.2.3 **avg() over()** : 统计分区中记录的平均值, partition by 可选, order by 可选

select ename,esex,eage,avg(salary) **over()** from emp; --总平均值

select ename,esex,eage,avg(salary) **over(order by eage)** from emp; --递加求平均值

select ename,esex,eage,avg(salary) **over(partition by esex)** from emp; --分组求平均值

select ename,esex,eage,avg(salary) **over(partition by esex order by eage)** from emp; --分组递加求平均值

15.2.4 **min() over()** : 统计分区中记录的最小值, partition by 可选, order by 可选

**max() over()** : 统计分区中记录的最大值, partition by 可选, order by 可选

select ename,esex,eage,salary,min(salary) over() from emp; --求总最小值

select ename,esex,eage,salary,min(salary) over(order by eage) from emp; --递加求最小值

select ename,esex,eage,salary,min(salary) over(partition by esex) from emp; --分组求最小值

select ename,esex,eage,salary,min(salary) over(partition by esex order by eage) from emp; --分组递加求最小值

select ename,esex,eage,salary,max(salary) over() from emp; --求总最大值

select ename,esex,eage,salary,max(salary) over(order by eage) from emp; --递加求最大值

select ename,esex,eage,salary,max(salary) over(partition by esex) from emp; --分组求最大值

select ename,esex,eage,salary,max(salary) over(partition by esex order by eage) from emp; --分组递加求最大值

15.2.5 **rank() over()** : 跳跃排序, partition by 可选, order by 必选

select ename,eage,rank() over(partition by job order by eage) from emp;

select ename,eage,rank() over(order by eage) from emp;

15.2.6 **dense\_rank()** : 连续排序, partition by 可选, order by 必选

select ename,eage,dense\_rank() over(partition by job order by eage) from emp;

select ename,eage,dense\_rank() over(order by eage) from emp;

15.2.7 **row\_number() over()** : 排序, 无重复值, partition by 可选, order by 必选

select ename,eage,row\_number() over(partition by job order by eage) from emp;

## **16.\*\*介绍一下oracle的主要特点\*\***

### 16.1 Oracle

优点:

16.1.1 **开放性**: Oracle 能所有主流平台上运行 (包括 windows) 完全支持所有工业标准采用完全开放策略使客户选择适合解决方案对开发商全力支持。

16.1.2 **可伸缩性,并行性**: oracle 并行服务器通过使组结点共享同簇工作来扩展windownt(服务控制管理器)能力提供高用性和高伸缩性簇解决方案windowsNT能满足需要用户把数据库移UNIXOracle并行服务器对各种UNIX平台集群机制都有着相当高集成度。

16.1.3 **安全性**: 获得最高认证级别的ISO标准认证。

16.1.4 **性能**: Oracle 性能高 保持开放平台下TPC-D和TPC-C世界记录。

16.1.5 客户端支持及应用模式: Oracle 多层次网络计算支持多种工业标准用ODBC、JDBC、OCI等网络客户连接。

16.1.6 使用风险: Oracle 长时间开发经验完全向下兼容得广泛应用地风险低。

缺点:



1、对硬件的要求很高；  
价格比较昂贵；  
管理维护麻烦一些；  
操作比较复杂，需要技术含量较高。

## 16.2 MySQL

优点：

16.2.1 体积小、速度快、总体拥有成本低，开源；  
16.2.2 支持多种操作系统；  
16.2.3 是开源数据库，提供的接口支持多种语言连接操作；  
16.2.4 MySQL的核心程序采用完全的多线程编程。线程是轻量级的进程，它可以灵活地为用户提供服务，而不过多的系统资源。用多线程和C语言实现的mysql能很容易充分利用CPU；  
16.2.5 MySQL有一个非常灵活而且安全的权限和口令系统。当客户与MySQL服务器连接时，他们之间所有的口令传送被加密，而且MySQL支持主机认证；  
16.2.6 支持ODBC for Windows，支持所有的ODBC2.5函数和其他许多函数，可以用Access连接MySQL服务器，使得应用被扩展；  
16.2.7 支持大型的数据库，可以方便地支持上千万条记录的数据库。作为一个开放源代码的数据库，可以针对不同的应用进行相应的修改；  
16.2.8 拥有一个非常快速而且稳定的基于线程的内存分配系统，可以持续使用而不必担心其稳定性；  
16.2.9 MySQL同时提供高度多样性，能够提供很多不同的使用者介面，包括命令行客户端操作，网页浏览器，以及各式各样的程序语言介面，例如C+，Perl，Java，PHP，以及Python。你可以使用事先包装好的客户端，或者干脆自己写一个合适的应用程序。MySQL可用于Unix，Windows，以及OS/2等平台，因此它可以用在个人电脑或者是服务器上。

缺点：

1、不支持热备份；  
2、MySQL最大的缺点是其安全系统，主要是复杂而非标准，另外只有到调用mysqladmin来重读用户权限时才发生改变；  
3、没有一种存储过程(Stored Procedure)语言，这是对习惯于企业级数据库的程序员的最大限制；  
4、MySQL的价格随平台和安装方式变化。Linux的MySQL如果由用户自己或系统管理员而不是第三方安装则是免费的，第三方案则必须付许可费。Unix或linux 自行安装 免费、Unix或Linux 第三方安装 收费。

17.\*\*介绍一下你经常使用的时间函数，怎么使用？\*\*

add\_months() 在某个日期上增加多少个月

months\_between() 两个日期之间有多少月份

last\_day() 当前给定月份所在的最后一天

next\_day() 下一个星期几

18.in\*\*和exists的区别，你经常用哪一个，哪个效率高\*\*

一般来说，这两个是用来做两张（或更多）表联合查询用的，in是把外表和内表作hash连接，而exists是对外表作loop循环，假设有A、B两个表，使用时是这样的：



18.1 select \* from A where id in (select id from B)--使用in

18.2 select \* from A where exists(select B.id from B where B.id=A.id)--使用exists  
也可以完全不使用in和exists:

18.3 select A.\* from A,B where A.id=B.id--不使用in和exists

具体使用时到底选择哪一个, 主要考虑查询效率问题:

第一条语句使用了A表的索引;

第二条语句使用了B表的索引;

第三条语句同时使用了A表、B表的索引;

如果A、B表的数据量不大, 那么这三个语句执行效率几乎无差别;

如果A表大, B表小, 显然第一条语句效率更高, 反之, 则第二条语句效率更高;

第三条语句尽管同时使用了A表、B表的索引, 单扫描次数是笛卡尔乘积, 效率最差。

19.\*\*根据一些条件分组后 把数量大于10的数据拿出来 sql语句怎么写\*\*

select job from emp group by job having count(1) >= 11;

## 20.Sql 连表分组查询代码 口述

select job from emp left join dept on emp.deptno = dept.deptno  
group by job having count(1) >= 10;

## 21.sql 商品记录表 商品金额 名称 怎么查出各个商品的总金额和记录条数

Select count(spname),sum(price),spname from sp group by splx;

## 22.db\*\*事务级别\*\*

隔离性问题:

脏读: 读到了其他事务未提交的数据

事务A增加了一条记录, 还未提交, 这个事务B读取到了增加后的数据,  
然后事务A执行回滚操作, 取消了刚才的增加, 所以事务B再次查询  
的行就无效了, 也就是脏数据

不可重复读: 一次读取到记录之后其他事务对这条数据进行了修改, 再次  
读取到的数据不一致

1.事务B读取当前表的数据

2.事务A将对表进行修改操作提交后,

3.事务B再次读取这张表，发现读取到的数据和之前不一样

**幻读**：相同的查询条件首次查询后，其他事务添加或删除了新的数据，

再次查询不一致

1.事务A对表新增并提交以后

2.事务B再次读取这张表，发现数据多了，就像发生了幻觉一样

**隔离级别：**

数据库中隔离级别有4个，由低到高依次为

Read uncommitted、Read committed、Repeatable read、Serializable

这四个级别可以逐个解决脏读，不可重复读，幻读这几个问题。

脏读    不可重复读    幻读

未提交读 Read uncommitted：    会    会    会

提交读 Read committed：    不会    会    会

可重复读 Repeatable read：    不会    不会    会

序列化 Serializable：    不会    不会    不会

\1. ISOLATION\_READ\_UNCOMMITTED：【很少应用 效果很差 效率也没高哪去】

这是事务最低的隔离级别，它允许另外一个事务可以看到这个事务未提交的数据。

这种隔离级别会产生脏读，不可重复读和幻想读。

\2. ISOLATION\_READ\_COMMITTED：【Oracle默认的隔离级别 大多数数据库默认】

保证一个事务修改的数据提交后才能被另外一个事务读取。

另外一个事务不能读取该事务未提交的数据

\3. ISOLATION\_REPEATABLE\_READ：【MySQL默认隔离级别】

这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻读。

它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免不可重复读。

\4. ISOLATION\_SERIALIZABLE：【根治问题 但是牺牲效率】

这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。

除了防止脏读，不可重复读外，还避免了幻像读。

**23.\*\*如果一个表A有值取A, B有值取B (问的case when) \*\***

(case when then else end)

**24.\*\*怎么知道此表有索引? \*\***

select index\_name from user\_indexes where table\_name = 'STUDENT';

**25.\*\*两个表如果都有索引, 那么查询的时候需要注意什么? \*\***

级联关系建立在索引上查询速度会有显著提高

**26.\*\*连表查询用过嘛?说一下有几种连表查询的方式\*\***

内连接 inner join on = join on

外连接 outer join on = join on

左外连接 left join on

右外连接 right join on

全外连接 full join on

**27.\*\*如何替换字符串里边儿的特殊字符\*\***

replace()

**28.Oracle\*\*除了聚组函数之外 还有哪些函数\*\***

聚组函数: max() min() avg() sum() count() group by 分组

单行函数: ceil() floor() round(2) trunc(2) sign() abs() power() sqrt()

日期函数: add\_months() months\_between() last\_day() next\_day()

转换函数: to\_number() to\_char() to\_date()

字符函数: lower() upper() initcap() length() substr(3/2) instr(4/3/2) replace(3) concat()

trim() ltrim() rtrim() lpad() rpad()

通用函数: nvl(a1,a2) nvl2(3)

decode(N)

(case when then else end)

**29.\*\*一个tperson表id, 一个tdetails表pid关联id, 几种方式查询两个表数据\*\***

内连接 外连接 子查询 自查询

**30.\*\*序列是怎么用的\*\***

查看序列的当前值：序列名.currval

查看序列的下一个值：序列名.nextval

**31.\*\*数据库中字符串怎么转换成日期格式，日期格式怎么转换成字符串格式\*\***

to\_date() to\_char()

**32.\*\*批量查询\*\***



```
SELECT * FROM FLB_FLIGHT_LOG_FOC
WHERE FLIGHT_ID IN
```

# {item}



**33.\*\*事物的四个特性\*\***

**事务特性：**

原子性：事务不可再分

一致性：数据类型保持一致

持久性：事务能够将数据持久化到数据库中

隔离性：多个事务之间可能会产生一些隔离性问题

**34.\*\*你在项目中使用什么隔离级别\*\***

详见22题

**35.user\*\*表 字段age code name 根据age 倒序排序查第10-20条数据\*\***

```
select rownum,b.* from (select rownum rn,a.* from (select * from user order by age desc)a where
rownum <= 20)b where b.rn>=10;
```

**36. A\*\*、B两张表，A表4条数据，B表3条数据，select \* from A,B一共几条数据？\*\***

12，笛卡尔基定律，两张表的数据一一对应，自查询

**37.SQL\*\*语句：求男女生平均分\*\***

Select sex,avg(num) from student group by sex;

38.\*\*接上题：再加上平均分大于等于60分这个条件\*\*

```
Select sex,avg(num) from student group by sex having avg(num) >= 60;
```

39.having\*\*用在哪里\*\*

group by之后，没有group by就不要出现having

40.\*\*分表知道吗\*\*

分区表：

当表中的数据量不断增大，查询数据的速度就会变慢，应用程序的性能就会下降，这时就应该考虑对表进行分区。表进行分区后，逻辑上表仍然是一张完整的表，只是将表中的数据在物理上存放到多个表空间(物理文件上)，这样查询数据时，不至于每次都扫描整张表。

表分区的具体作用

Oracle的表分区功能通过改善可管理性、性能和可用性，从而为各式应用程序带来了极大的好处。通常，分区可以使某些查询以及维护操作的性能大大提高。此外，分区还可以极大简化常见的管理任务，分区是构建千兆字节数据系统或超高可用性系统的关键工具。

分区功能能够将表、索引或索引组织表进一步细分为段，这些数据库对象的段叫做分区。每个分区有自己的名称，还可以选择自己的存储特性。从数据库管理员的角度来看，一个分区后的对象具有多个段，这些段既可进行集体管理，也可单独管理，这就使数据库管理员在管理分区后的对象时有相当大的灵活性。但是，从应用程序的角度来看，分区后的表与非分区表完全相同，使用 SQL DML 命令访问分区后的表时，无需任何修改。

什么时候使用分区表：

- 1、表的大小超过2GB。
- 2、表中包含历史数据，新的数据被增加到新的分区中。

表分区的优缺点

表分区有以下优点：

- 1、改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索速度。
- 2、增强可用性：如果表的某个分区出现故障，表在其他分区的数据仍然可用；
- 3、维护方便：如果表的某个分区出现故障，需要修复数据，只修复该分区即可；
- 4、均衡I/O：可以把不同的分区映射到磁盘以平衡I/O，改善整个系统性能。

缺点：

分区表相关：已经存在的表没有方法可以直接转化为分区表。不过 Oracle 提供了在线重定义表的功能。

41.mysql\*\*怎么创建数据库\*\*

```
create database databaseName
```

#### 42.mysql\*\*怎么给字段设置默认值\*\*

```
name varchar(10) default 'elena'
```

#### 43.mysql\*\*时间数据类型有哪些\*\*

**datetime** 的日期范围比较大; **timestamp** 所占存储空间比较小, 只是 datetime 的一半

另外, timestamp 类型的列还有个特性: 默认情况下, 在 insert, update 数据时, timestamp 列会自动以当前时间 (CURRENT\_TIMESTAMP) 填充/更新。“自动”的意思就是, 你不去管它, MySQL 会替你去处理。

```
create table t8 (
 id1 timestamp NOT NULL default CURRENT_TIMESTAMP,
 id2 datetime default NULL);
```

两者之间的比较:

1. timestamp 容易所支持的范围比 datetime 要小。并且容易出现超出的情况
2. timestamp 比较受时区 timezone 的影响以及 MySQL 版本和服务器的 SQL MODE 的影响。

#### 44.\*\*您用过那些数据库\*\*

mysql oracle

#### 45.mysql\*\*数据库获取当前时间\*\*

实现方式:

- 1、将字段类型设为 TIMESTAMP
- 2、将默认值设为 CURRENT\_TIMESTAMP

#### 46.mysql\*\*如果数据量很大怎么解决\*\*

1. 对查询进行优化, 应尽量**避免全表扫描**, 首先应考虑在 where 及 order by 涉及的列上建立索引。
2. 应尽量**避免**在 where 子句中对字段进行 **null 值判断**, 否则将导致引擎放弃使用索引而进行全表扫描, 如: select id from t where num is null 可以在 num 上设置默认值 0, 确保表中 num 列没有 null 值, 然后这样查询: select id from t where num=0
3. 应尽量**避免**在 where 子句中**使用!=或<>操作符**, 否则引擎将放弃使用索引而进行全表扫描。
4. 应尽量**避免**在 where 子句中**使用or** 来连接条件, 否则将导致引擎放弃使用索引而进行全表扫描, 如: select id from t where num=10 or num=20 可以这样查询: select id from t where num=10 union all select id from t where num=20
5. **in 和 not in 也要慎用**, 否则会导致全表扫描, 如: select id from t where num in(1,2,3) 对于连续的数值, 能用 between 就不要用 in 了: select id from t where num between 1 and 3
6. 下面的查询也将导致全表扫描: select id from t where name like '%李%' 若要提高效率, 可以考虑全文检索。

如果在 where 子句中使用参数, 也会导致全表扫描。因为 SQL 只有在运行时才会解析局部变量, 但优化程序不能将访问计划的选择推迟到运行时; 它必须在编译时进行选择。然而, 如果在编译时建立访问计划, 变量的值还是未知的, 因而无法作为索引选择的输入项。如下面语句将进行全表扫描: select id from t where num=@num 可以改为**强制查询使用索引**: select id from t with(index(索引名)) where

num=@num

8.应尽量避免在 where 子句中**对字段进行表达式操作**，这将导致引擎放弃使用索引而进行全表扫描。  
如：select id from t where num/2=100应改为:select id from t where num=100\*2

9.应尽量避免在 where子句中**对字段进行函数操作**，这将导致引擎放弃使用索引而进行全表扫描。如：  
select id from t where substring(name,1,3)='abc'，name以abc开头的id应改为：

```
select id from t where name like 'abc%'
```

10.**不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算**，否则系统将可能无法正确使用索引。

11.在使用索引字段作为条件时，如果该索引是**复合索引**，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

12.不要写一些没有意义的查询，如需要生成一个空表结构：select col1,col2 into #t from t where 1=0

这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：

```
create table #t(...)
```

13.很多时候用 **exists 代替 in 是一个好的选择**：select num from a where num in(select num from b)

用下面的语句替换：

```
select num from a where exists(select 1 from b where num=a.num)
```

14.并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，**当索引列有大量数据重复时，SQL查询可能不会去利用索引**，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。

索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。

17.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

18.尽可能的使用 varchar/nvarchar 代替 char/nchar，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

19.**任何地方都不要使用 select \* from t，**用具体的字段列表代替“\*”，不要返回用不到的任何字段。

20.尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

21.避免频繁创建和删除**临时表**，以减少系统表资源的消耗。

22.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

23.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后 insert。

24.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table，然后 drop table，这样可以避免系统表的较长时间锁定。

25.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

26.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

与临时表一样，游标并不是不可使用。对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

28.在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON，在结束时设置 SET NOCOUNT OFF。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE\_IN\_PROC 消息。

29.尽量避免大事务操作，提高系统并发能力。

30.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

#### 47. MySQL \*\*通配符\_和%的区别\*\*

%：表示任意个或多个字符。可匹配任意类型和长度的字符。

Sql代码

```
select * from user where username like '%huxiao';
```

```
select * from user where username like 'huxiao%';
```

```
select * from user where username like '%huxiao%';
```

另外，如果需要找出u\_name中既有“三”又有“猫”的记录，请使用and条件

```
SELECT * FROM [user] WHERE u_name LIKE '%三%' AND u_name LIKE '%猫%'
```

```
若使用 SELECT * FROM [user] WHERE u_name LIKE '%三%猫%'
```

虽然能搜索出“三脚猫”，但不能搜索出符合条件的“张猫三”。

\_：表示任意单个字符。匹配单个任意字符，它常用来限制表达式的字符长度语句：（可以代表一个中文字符）

Sql代码

```
select * from user where username like '_';
```

```
select * from user where username like 'huxia_';
```

```
select * from user where username like 'h_xiao';
```

#### 48. \*\*数据库中索引是什么\*\*



**索引**是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。如果想按特定职员姓来查找他或她，则与在表中搜索所有的行相比，索引有助于更快地获取信息。索引的一个主要目的就是加快检索表中数据，亦即能协助信息搜索者尽快的找到符合限制条件的记录ID的辅助数据结构。

#### 49.\*\*索引的好处与坏处\*\*

创建索引可以大大提高系统的性能:

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

增加索引也有许多不利的方面:

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

索引是建立在数据库表中的某些列的上面。因此，在创建索引的时候，应该仔细考虑在哪些列上可以创建索引，在哪些列上不能创建索引。一般来说，应该在哪些列上创建索引，例如：

在经常需要搜索的列上，可以加快搜索的速度；

在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；

在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；

#### 50.mysql\*\*和oracle的分页？\*\*

mysql:

select 字段 from 表 limit 起始索引值,显示记录数;

oracle:

select rownum,a.\* from (select rownum rn,表.\* from 表 where rownum <= 20)a where a.rn>=11;

## 51.mysql存储引擎

### InnoDB 与MyISAM

查看MySQL提供的所有存储引擎

```
mysql> show engines;
```

### Mysql 中 MyISAM 和 InnoDB 的区别有哪些？

InnoDB支持事务，MyISAM不支持

对于InnoDB每一条SQL语言都默认封装成事务，自动提交，这样会影响速度，所以最好把多条SQL语言放在begin和commit之间，组成一个事务；

InnoDB支持外键，而MyISAM不支持。

对一个包含外键的InnoDB表转为MYISAM会失败；

InnoDB是聚集索引，数据文件是和索引绑在一起的，必须要有主键，通过主键索引效率很高。

但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。

因此主键不应该过大，因为主键太大，其他索引也都会很大。

而MyISAM是非聚集索引，数据文件是分离的，索引保存的是数据文件的指针。

主键索引和辅助索引是独立的。

InnoDB不保存表的具体行数，执行select count(\*) from table时需要全表扫描。

而MyISAM用一个变量保存了整个表的行数，执行上述语句时只需要读出该变量即可，速度很快；

InnoDB不支持全文索引，而MyISAM支持全文索引，查询效率上MyISAM要高；

### 是否支持行级锁

MyISAM 只有表级锁(table-level locking),

而InnoDB 支持行级锁(row-level locking)和表级锁,默认为行级锁。

### 是否支持事务和崩溃后的安全恢复：

MyISAM 强调的是性能，每次查询具有原子性,其执行速度比InnoDB类型更快，但是不提供事务支持。

但是InnoDB 提供事务支持事务，外部键等高级数据库功能。

具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表。

### 是否支持MVCC：

仅 InnoDB 支持。应对高并发事务, MVCC比单纯的加锁更高效;

MVCC只在 READ COMMITTED 和 REPEATABLE READ 两个隔离级别下工作;

MVCC可以使用 乐观(optimistic)锁 和 悲观(pessimistic)锁来实现;各数据库中MVCC实现并不统一

### 如何选择：

是否要支持事务，如果要请选择innodb，如果不需要可以考虑MyISAM；

如果表中绝大多数都只是读查询，可以考虑MyISAM，如果既有读写也挺频繁，请使用InnoDB

系统崩溃后，MyISAM恢复起来更困难，能否接受；

MySQL5.5版本开始InnoDB已经成为Mysql的默认引擎(之前是MyISAM)，说明其优势是有目共睹的，如果你不知道用什么，那就用InnoDB，至少不会差。

### 查看MySQL提供的所有存储引擎

```
mysql> show engines;
```

### 查看MySQL当前默认的存储引擎

我们也可以通过下面的命令查看默认的存储引擎。

```
mysql> show variables like '%storage_engine%';
```

### 查看表的存储引擎

```
show table status like "table_name";
```

InnoDB（事务性数据库引擎

### InnoDB 逻辑存储结构图

从InnoDB 存储引擎的逻辑存储结构看，所有数据都被逻辑地存放在一个空间中，称之为表空间（tablespace）。

表空间又由段（segment），区（extent），页（page）组成。页在一些文档中有时也称为块（block）。InnoDB 逻辑存储结构图如下：



### 表空间（tablespace）

表空间是InnoDB存储引擎逻辑的最高层，所有的数据都存放在表空间中。

默认情况下，InnoDB存储引擎有一个共享表空间ibdata1,即所有数据都存放在这个表空间中内。

如果启用了innodbfileper\_table参数，需要注意的是每张表的表空间内存放的只是数据、索引、和插入缓冲Bitmap，其他类的数据，比如回滚(undo)信息、插入缓冲检索页、系统事物信息，二次写缓冲等还是放在原来的共享表内的。

### 段（segment）

表空间由段组成，常见的段有数据段、索引段、回滚段等。

InnoDB存储引擎表是索引组织的，因此数据即索引，索引即数据。数据段即为B+树的叶子结点，索引段即为B+树的非索引结点。

在InnoDB存储引擎中对段的管理都是由引擎自身所完成，DBA不能也没必要对其进行控制。

### 区（extent）

区是由连续页组成的空间，在任何情况下每个区的大小都为1MB。

为了保证区中页的连续性，InnoDB存储引擎一次从磁盘申请4~5个区。

默认情况下，InnoDB存储引擎页的大小为16KB，一个区中一共64个连续的区。

## 页 (page)

页是InnoDB磁盘管理的最小单位。

在InnoDB存储引擎中，默认每个页的大小为16KB。

从InnoDB1.2.x版本开始，可以通过参数innodbpagesize将页的大小设置为4K, 8K, 16K。

InnoDB存储引擎中，常见的页类型有：**数据页**，**undo页**，**系统页**，**事务数据页**，**插入缓冲位图页**，**插入缓冲空闲列表页**等。

[回到顶部](#)

InnoDB页结构相关示意图

InnoDB页结构单体图

InnoDB数据页由以下7部分组成，如图所示：



其中File Header、Page Header、File Trailer的大小是固定的，分别为38, 56, 8字节，这些空间用来标记该页的一些信息，

如Checksum，数据页所在B+树索引的层数等。User Records、Free Space、Page Directory这些部分为实际的行记录存储空间，因此大小是动态的。

下边我们用表格的方式来大致描述一下这7个部分：



### 记录在页中的存储流程图

每当我们插入一条记录，都会从Free Space部分，也就是尚未使用的存储空间中申请一个记录大小的空间划分到User Records部分，

当Free Space部分的空间全部被User Records部分替代掉之后，也就意味着这个页使用完了，如果还有新的记录插入的话，就需要去申请新的页了，这个过程的图示如下：



### 不同InnoDB页构成的数据结构图

一张表中可以有成千上万条记录，一个页只有16KB，所以可能需要好多页来存放数据。

不同页其实构成了一条双向链表，File Header是InnoDB页的第一部分，它的FILPAGEPREV和FILPAGENEXT就分别代表本页的上一个和下一个页的页号，即链表的上一个以及下一个节点指针。



## InnoDB 锁类型思维导图



### 加锁机制

乐观锁与悲观锁是两种**并发控制**的思想，可用于解决**丢失更新**问题。

### 乐观锁

每次去取数据，都很乐观，觉得不会出现并发问题。

因此，访问、处理数据每次都不上锁。

但是在更新的时候，再根据**版本号**或**时间戳**判断是否有冲突，有则处理，无则提交事务。

### 悲观锁

每次去取数据，很悲观，都觉得会被别人修改，会有并发问题。

因此，访问、处理数据前就加**排他锁**。

在整个数据处理过程中锁定数据，**事务提交或回滚后才释放锁**。

### 锁粒度

**表锁**：开销小，加锁快；锁定力度大，发生锁冲突概率高，并发度最低；不会出现死锁。

**行锁**：开销大，加锁慢；会出现死锁；锁定粒度小，发生锁冲突的概率低，并发度高。

**页锁**：开销和加锁速度介于**表锁和行锁**之间；会出现死锁；**锁定粒度**介于表锁和行锁之间，并发度一般

### 兼容性

#### 共享锁：

又称**读锁**（S锁）。

一个事务获取了共享锁，其他事务可以获取共享锁，不能获取排他锁，其他事务可以进行读操作，不能进行写操作。

SELECT ... LOCK IN SHARE MODE 显示加共享锁。

#### 排他锁：

又称**写锁**（X锁）。

如果事务T对数据A加上排他锁后，则其他事务不能再对A加任何类型的封锁。获准排他锁的事务既能读数据，又能修改数据。

SELECT ... FOR UPDATE 显示添加排他锁。

### 锁模式

记录锁：在行相应的索引记录上的锁，**锁定一个行记录**

gap锁：是在索引记录间歇上的锁，**锁定一个区间**

next-key锁： 是记录锁和在此索引记录之前的gap上的锁的结合， 锁定行记录+区间。

意向锁 是为了支持多种粒度锁同时存在；

=====