-- 数值型

-- NUMBER(4) 不带小数点的数值

-- NUMBER(8,2) 总长8位, 小数点2位

-- 字符型

-- CHAR(2) 定长的字符型

-- VARCHAR(10) 定长的字符型

-- VARCHAR2(20) 变长的字符型

-- 日期型

-- DATE

--create table users(

-- ID NUMBER(4) PRIMARY KEY,

-- NAME VARCHAR(10),

-- PASSWORD VARCHAR2(10),

-- SEX CHAR(2),

-- ADDR VARCHAR2(20),

-- BIRTHDAY DATE,

-- SAL NUMBER(8,2)

--);

-- DROP TABLE USERS;

-- desc users;

-- select \* from emp;

-- 用||把n列合在一起,

-- 连接表达式(||)中间加的自定义内容都是字符数数据, 需要用''包裹

-- 使用了连接表达式之后, null会被处理成''

-- select first\_name||' + '||last\_name from employees;

-- 任何的类型都支持null

-- 空值不等于'', 不等于0.0, 不等于mull

-- 字符型和日期型必须用''包裹

-- 日期型的数据可以使用+,-运算

-- 一个日期+一个数值得到一个新的日期

-- 2个日期型的数据相减, 得到的是2者之间相差的天数

-- SYSDATE(现在的日期)

-- 2个日期型的数据不能相加, 不能乘除

-- 用distinct去除重复的数据

-- SELECT first\_name ||' '||last\_name "姓名", hire\_date "雇佣日期", (hire\_date+90) "转正日期" FROM employees;

-- select first\_name ||' '||last\_name "姓名", (sysdate - hire\_date)/365 "工作年限" from employees;

-- select DISTINCT department\_id from employees

-- where condition中日期和字符型数据必须使用''包裹

-- 字符型数据严格区分大小写

-- select \* from employees where first\_name = 'TIMOTHY';

-- 对于日期型的数据, 默认的格式是"日-月-年"

-- 改变当前会话中的日期格式

-- alter session set nls\_date\_format="YYYY-MM-DD HH:MI:SS"

-- between a and b 包含a和b这样的边界条件

-- select \* from jobs where min\_salary BETWEEN 2000 and 3000;

-- 使用in

-- select \* from jobs where job\_title in ('President', 'Purchasing Clerk')

-- 使用like

-- select \* from employees where last\_name like '%A%'

-- sql注入攻击可以用括号限制顺序解决

-- select \* from users where name = 'admin' or ('x'= 'x' and password = '45456')

-- Oracle的函数必须有返回结果

-- 一个多行函数

-- select first\_name||' '||last\_name, LENGTH(first\_name||' '||last\_name) from employees

-- 一个单行函数

-- select MAX(salary) from employees;

-- 单行函数有: 1.字符函数 2.数字函数 3.日期类型 4.转换函数 5.其他函数

-- 1. 字符函数 LOWER, UPPER. 对于这两种函数, 有用的地方是对于区别大小写的字符类数据, 可以把某个列的值完全转化为大写或者小写. 还有一个initcap, 把首字母变成大写.

-- 2. 合并函数CONCAT(a,b). 和||效果完全一样

-- 3. SUBSTR(a,n1,n2). 截取a的内容, 从n1开始截取, 截取n2个.

-- 4. INSTR(c1,c2,i,j). 从c1中搜索c2, i是开始搜索的位置, j是第几次出现.

-- 5. REPLACE

-- 6. LPAD()

-- 组函数(多行函数, 一次把多行记录变成一条记录), 比如AVG, COUNT, MAX, MIN, SUM

-- AVG, SUM只能在数值型的数据上使用

-- MAX, MIN, COUNT可以针对任何类型的数据

-- 所有组函数自动忽略空值

-- NVL(a, 0)为null的可以被设为零, 防止计算的时候出问题. 但是注意别名一定要用""包裹, 否则会出问题. 被替换的值格式一定要和原来的格式相同.

-- NVL(a, v1, v2)如果a不为null, 就是v1, 为null返回v2.

-- NULLIF(LENGTH(a), LENGTH(b)). 比较2个表达式, 如果相同返回null, 不相同返回第一个表达式.

-- 使用GROUP BY时使用组函数的条件: sql要查询的字段, 要么出现在组函数里, 要么出现在GROUP BY里(也可以只出现在GROUP BY里). 这个规定同样适用于多表查询.

SELECT MAX(e.SALARY), j.JOB\_TITLE from EMPLOYEES e, JOBS j where e.JOB\_ID = j.JOB\_ID GROUP BY j.JOB\_TITLE

-- 分组之后的数据, 要再进行过滤要使用HAVING, 后面还可以接ORDER BY

-- 外键可以为空, 为空也是一组

-- 用case实现if...else

-- SELECT

-- EMPLOYEE\_ID,

-- FIRST\_NAME || ' ' || LAST\_NAME,

-- SALARY,

-- CASE

-- JOB\_ID

-- WHEN 'IT\_PROG' THEN

-- 1.50 \* SALARY

-- WHEN 'SA\_MAN' THEN

-- 1.20 \* SALARY ELSE SALARY

-- END "工资修正"

-- FROM

-- EMPLOYEES

-- 用decode实现if...else

-- SELECT

-- EMPLOYEE\_ID,

-- FIRST\_NAME || ' ' || LAST\_NAME,

-- SALARY,

-- DECODE( JOB\_ID, 'IT\_PROG', 1.5 \* SALARY, 'SA\_MAN', 1.2 \* SALARY, SALARY ) "工资修正"

-- FROM

-- EMPLOYEES

-- 等值查询

-- SELECT E.\*, D.\*, L.\* FROM EMP E, DEPT D, LOCATION L WHERE E.DEPTNO = D.DEPTNO AND D.LOC\_ID = L.LOCID AND E.EMPNO = 6379;

-- 非等值查询

-- SELECT E.EMPNO, E.ENAME, E.SAL, S.GRADE, S.LOSAL, S.HISAL FROM EMP E, SALGRADE S WHERE E.SAL BETWEEN S.LOSAL AND S.HISAL;

-- 自查询, 通过别名讲一个表虚拟成2个表, 然后在这2个表上做等值查询

-- 把7369的领导编号和名字也查出来了

-- SELECT E.EMPNO, E.ENAME, M.EMPNO, M.ENAME FROM EMP E, EMP M WHERE E.MGR = M.EMPNO AND E.EMPNO = 7369;

-- 等值查询只能查询一一对应的情况

-- 左外连接可以把右边表中不满足等值条件的数据查询出来

-- (+)号代表外连接, 放在非主表的一方

-- SELECT E.EMPNO, E.ENAME, E.DEPTNO, E.DEPTNO, D.DNAME, D.LOC\_ID FROM EMP E, DEPT D WHERE D.DEPTNO (+) = E.DEPTNO;

-- 交叉连接 CROSS JOIN 会产生笛卡尔积

-- SELECT E.\*, D.\* FROM EMP E CROSS JOIN DEPT D;

-- 自然连接 在父子表关系上, 自动地匹配2个表中列名完全相同的字段, 在这些相同名称的字段上做等值查询

-- DEPTNO是2个表中相同的字段, 不带表名

-- SELECT E.EMPNO, E.ENAME, DEPTNO, D.DNAME, D.LOC\_ID FROM EMP E NATURAL JOIN DEPT D;

-- 自然连接没有参照列, 会产生笛卡尔积

-- JOIN...USING 在自然连接的基础上, 加以改进, 使用指定的参照列作为等值条件

-- SELECT E.EMPNO, E.ENAME, D.DNAME, D.LOC\_ID FROM EMP E JOIN DEPT D USING (DEPTNO);

-- JOIN...ON, 使用on里面指定的条件作为查询条件(条件可以是任意条件)

-- SELECT E.\*, D.\* FROM EMP E JOIN DEPT D ON (E.DEPTNO = D.DEPTNO)

-- JOIN...ON做非等值查询

-- SELECT E.EMPNO, E.ENAME, E.SAL, S.GRADE, S.LOSAL, S.HISAL FROM EMP E JOIN SALGRADE S ON (E.SAL BETWEEN S.LOSAL AND S.HISAL) WHERE E.EMPNO=7369;

-- OUTER JOIN...ON

-- LEFT OUTER JOIN...ON 可以把左边表中不满足等值条件的数据查询出来

-- SELECT E.EMPNO, E.ENAME, E.DEPTNO, D.DEPTNO, D.DNAME, D.LOC\_ID FROM EMP E LEFT OUTER JOIN DEPT D ON (E.DEPTNO=D.DEPTNO)

-- 使用UNION把2个结果合并成一个结果集

-- 使用查询语句来创建表(复制表及其数据)

-- UNION去重

-- SELECT \* FROM DEPT;

-- UNION

-- SELECT \* FROM DEPT\_BAK;

-- UNION ALL不去重

-- SELECT \* FROM DEPT;

-- UNION ALL

-- SELECT \* FROM DEPT\_BAK;

-- 子查询: 为了给主查询提供条件, 而首先执行的一个查询, 被称为子查询.

-- 子查询首先被执行

-- 主查询使用自查询的查询条件

-- 自查询通常出现在比较运算符的右边, 并且用()包裹, 便于理解

-- SELECT \* FROM EMP WHERE SAL > (SELECT SAL FROM EMP WHERE EMPNO = 7566);

-- 子查询的分类用分析子查询的返回结果来区分

-- 单行单列子查询

-- SELECT SAL FROM EMP WHERE EMPNO = 7566

-- 单行多列子查询

-- SELECT MGR, JOB FROM EMP WHERE EMPNO = 7566

-- 多行多列子查询

-- SELECT MGR, JOB FROM EMP WHERE EMPNO = 7566 OR EMPNO = 7369

-- 子查询的时候用组函数

-- 只要子查询的结果依旧是单行单列, 仍然可以使用单行比较操作符

-- SELECT \* FROM EMP WHERE SAL < (SELECT AVG(SAL) FROM EMP);

-- 子查询返回的结果是多行单列, 就不能使用单行比较操作符

-- SELECT \* FROM EMP WHERE SAL < (SELECT AVG(SAL) FROM EMP GROUP BY DEPTNO);

-- 子查询出现在having中, having对分组结果进行过滤

-- SELECT DEPTNO, MAX(SAL) FROM EMP GROUP BY DEPTNO HAVING MAX(SAL) > (SELECT MAX(SAL) FROM EMP WHERE DEPTNO = 20);

-- 子查询没有结果, 主查询也没有结果

-- 多行单列子查询, 使用多行比较运算符: IN, ALL, ANY

-- SELECT E.EMPNO, E.ENAME, E.JOB, E.SAL FROM EMP E WHERE E.JOB IN (SELECT JOB FROM EMP WHERE SAL > 300);

-- >ALL或者<ALL

-- SELECT \* FROM EMP E WHERE E.SAL > ALL (SELECT SAL FROM EMP WHERE DEPTNO = 30);

-- 多行多列子查询, 可以使用in比较运算符

-- 成对的比较

-- 和7566, 7369同经理同职位的其他同事

-- SELECT EMPNO, ENAME, MGR, JOB FROM EMP WHERE (MGR, JOB) IN (SELECT MGR, JOB FROM EMP WHERE EMPNO = 7566 OR EMPNO = 7369)

-- SELECT EMPNO, ENAME, MGR, JOB FROM EMP WHERE MGR IN (SELECT MGR FROM EMP WHERE EMPNO = 7566 OR EMPNO = 7369) AND JOB IN (SELECT JOB FROM EMP WHERE EMPNO = 7566 OR EMPNO = 7369) AND EMPNO !=7566 AND EMPNO !=7369;

-- 用子查询做一个中间表

-- SELECT \* FROM (SELECT DEPTNO, MAX(SAL), MIN(SAL), AVG(SAL), SUM(SAL) FROM EMP WHERE DEPTNO IS NOT NULL GROUP BY DEPTNO);

-- 数据库对象:

-- 表: 基本的数据存储对象, 以行和列的形式存在

-- 约束: 执行数据校验, 保证数据完整性的对象

-- 视图: 一个或多个表数据的显示

-- 索引: 用于提高查询的速度

-- 同义词: 对象的别名

-- 创建索引

-- CREATE INDEX TEL\_IDX ON STUDENT(TEL);

-- CREATE INDEX COMBINATION\_IDX ON STUDENT(SNAME, GRADE, GENDER);

-- SELECT \* FROM ALL\_INDEXES WHERE TABLE\_NAME = '表名';

-- SELECT \* FROM ALL\_IND\_COLUMNS WHERE TABLE\_NAME = '表名'

-- 视图: 视图是一个虚表, 我们可以从