Oracle学习笔记

使用准备

1. 检查监听和服务（此电脑→管理→服务）
2. 打开cmd，输入命令 `sql /nolog`
3. 测试连接：输入命令 `conn / as sysdba;`
4. 创建用户：`create user 用户名 identified by 密码`
   示例：`create user c##dhee identified by dhee`
5. 对用户授权
   i. Oracle用户授权的主要几种权限：connect，resource，dba
   ii. 命令：`grant dba to c##dhee`
6. 连接新用户：输入命令 `conn c##dhee/dhee`
7. 导表（命令行操作）：输入@，把文件从文件夹拖入命令行窗口内，按回车

基本SELECT语句

SQL：结构化查询语言

- DDL：CREATE,DROP,ALTER
- DML：INSERT,UPDATE,DELETE
- DQL：SELECT,GROUP BY,ORDER BY
- DCL：GRANT,REVOKE
- TCL（事务控制语言）；COMMIT,ROLLBACK

```
SELECT 列名1,列名2,...
FROM 表名;
```

1. 书写规则
   - 关键字大写，其他内容小写
   - 语句结束，分号可加可不加（最好写上）
   - 子句最好换行写
   - `SELECT *` 实际开发不建议使用，影响执行效率
   - 代码适当缩进
   - 字符串由单引号括起
2. 注释

- 单行注释：`-- 注释内容`

- 多行注释：`/*注释内容*/`

1. 数据去重：`DISTINCT`

```
SELECT DISTINCT department_id
FROM employees;
```

4. sql的书写顺序

```
1 SELECT
2 FROM
3 WHERE
```

```
4  GROUP BY
5  HAVING
6  ORDER BY
```

5. 算术运算符

- `+ - * /`：加、减、乘、除
- `nvl(表达式1,表达式2)`：如果表达式1为null，则结果为表达式2；若表达式1不为null，则结果为表达式1

```
SELECT first_name,commission_pct,nvl(commission_pct,0),salary + salary * nvl(commission_pct,0)
FROM employees
```

6. 伪表：`dual`
   Oracle系统内置的一个虚表

```
SELECT 1 + 2
FROM dual;
-- 返回记录：3
```

```
SELECT SYSDATE
FROM dual
--返回记录：当前系统时间
```

```
SELECT 1 + '2'
FROM dual
```

```
SELECT 1 + NULL
FROM dual
--返回记录：null与任何数据类型做运算结果都为null
```

7. 字符串类型：需要先转换成数值型后才可以参与算术运算

```
SELECT '1' + '2'
FROM dual;
```

8. 日期类型：在其后`+1`表示1天
   双引号`""`的作用：特殊含义，需要原样输出字符串时使用

```
SELECT hire_date,hire_date + 30 "DATE"
FROM employees e
```

9. 起别名：直接可在列名、表名后接上，如`employees e`中的`e`就是别名
   - Oracle中不能在这些子句中起别名：`WHERE、GROUP BY、HAVING`
   - Oracle的执行顺序：FROM→WHERE→GROUP BY→HAVING→SELECT→ORDER BY
10. 连接符：`||`

```
SELECT e.first_name,e.last_name,e.job_id ,e.first_name || ' ' || e.last_name NAME
FROM employees e;
```

- 字符串连接：`CONCAT(参数1,参数2)`（需要连接多个字符时，嵌套可实现）

```
SELECT CONCAT(CONCAT(e.first_name,' '),e.last_name) NAME
FROM employees e;
```

```
--示例输出：  e.first_name's job is  e.job_id
SELECT e.first_name || '''s job is' || e.job_id
FROM employees e
```

- `q'[ ]`：表示原样输出括号内的内容

```
SELECT e.first_name || q'['s job is]' || e.job_id  job
FROM employees e
```

WHERE子句

1. 比较运算符：`=  >  >=  <  <=  <>`
   - 不等于有两种写法：`<>`或`!=`

```
SELECT *
FROM employees e
-- WHERE e.employee_id != 200;
WHERE e.employee_id <> 200;
```

- 日期类型也可以使用比较运算符

```
SELECT e.hire_date
FROM employees e
WHERE e.hire_date >= to_date('1990/01/01','yyyy-MM-dd')
```

- null不能使用比较运算符（如下的代码，注释的部分为错误代码）

```
SELECT *
FROM employees e
-- WHERE e.department_id = NULL;
WHERE e.department_id IS NULL;
```

2. `BETWEEN...AND`：包括边界；可比较数值、日期、字符串（比较的是首字母ASCII值）类型

```
-- 日期类型
SELECT e.hire_date
FROM employees e
-- 默认的日期格式是 DD-MON-RR （可以在命令行看到）
WHERE e.hire_date BETWEEN to_date('1990/01/01','yyyy-MM-dd') AND to_date('1998/01/01','yyyy-MM-dd')
```

```
-- 字符串类型
SELECT *
```

```
FROM employees e
WHERE e.first_name BETWEEN 'J' AND 'P'
```

- 匹配字符串时，字符串是区分大小写的

```
SELECT *
FROM employees e
WHERE e.first_name = 'jennifer'  -- 字符串区分大小写
```

3. `IN`

```
-- 使用IN运算显示列表中的值
SELECT *
FROM employees e
WHERE e.department_id IN (60,90,NULL)   -- null:结果忽略了null的条件
```

```
SELECT *
FROM employees e
WHERE e.department_id NOT IN (60,90,NULL)  -- 查询不到结果
```

4. `LIKE`
    i. `%`：匹配0或多个字符

```
SELECT *
FROM employees e
WHERE e.last_name LIKE '%s';
```

2. `_`：匹配任意一个字符

```
SELECT *
FROM employees e
WHERE e.last_name LIKE '_i%';
```

3. 转义字符：可以是字符，也可以是特殊符号 用法：`ESCAPE` 当做转义字符的字符
    v. 逻辑运算符：`AND`、`OR`、`NOT`
4. `AND`：条件同时满足

```
SELECT *
FROM employees e
WHERE e.employee_id >= 150 AND e.employee_id <= 200
-- 等价于
 --SELECT *
 --FROM employees e
 --WHERE e.employee_id BETWEEN 150 AND 200
```

- `OR`：条件满足其一
- `NOT`：条件都不满足

1. 优先级：可用括号改变执行顺序

ORDER BY子句

`asc`：（默认不写）升序；`desc`：降序；null：排序顺序中的最大值
（也可按别名排序、多个列排序）

```
SELECT e.first_name,e.last_name,e.department_id dept, e.salary
FROM employees e
ORDER BY dept,salary DESC
```

修改表数据的两种方式：1) `FOR UPDATE` 2) `ROWID`
1) `FOR UPDATE`

```
SELECT e.*
FROM employees e FOR UPDATE
```

2) `ROWID`

```
SELECT e.*,ROWID
FROM employee e
```

单行函数

单行函数对单行进行操作，每行返回一个结果；可以写在`SELECT`、`WHERE`、`ORDER BY`子句中；函数可以嵌套

字符函数

1. 返回ASCII值：`SELECT ASCII('A') FROM dual; --65`
2. 返回字符：`SELECT CHR(97) FROM dual; --'A'`
3. 大小写处理函数：`LOWER()、UPPER()、INITCAP()`
   - `LOWER()`：全部小写
   - `UPPER()`：全部大写
   - `INITCAP()`：首字母大写
4. 字符处理函数
   - 截取字符串：`SUBSTR(原字符串,起始位置,截取的长度)` *数据库中，序号的起始位置从1开始*
     `SELECT UPPER(SUBSTR('helloworld',2,3)) FROM dual`
   - 返回长度：`LENGTH(字段/字符)`
     `sql SELECT * FROM employee e WHERE LENGTH(e.first_name) = 3 ORDER BY LENGTH(e.first_name)`
   - 返回位置：`INSTR(表达式1,表达式2)`：表达式2在表达式1中的位置
     - 如果找到返回具体位置
     - 找不到则返回0
     - 如果表达式1中有许多表达式2，则返回第一次出现的位置
       `SELECT INSTR('helloworld','w') FROM dual --6`
       `SELECT INSTR('helloworld,'j') FROM dual --0`
       `SELECT INSTR('helloworld','o' FROM dual) --5`
       模糊查询的功能也可通过此函数实现

```
SELECT *
FROM employees e
WHERE e.last_name LIKE '%a%';
-- 等价于
-- SELECT *
-- FROM employee e
-- WHERE INSTR(e.last_name,'a') > 0;
```

- 字符串填充：
  - 左填充：`LPAD(字符,补齐的总长度,填补的内容)`
  - 右填充：`RPAD()`
    `SELECT LPAD('hello',10,'*') FROM dual;`
  - `TRIM()`
    - 去首尾空格
      `SELECT TRIM(' a b c ')|| 'd' FROM dual`
    - 去首尾字符
      `SELECT TRIM('a' FROM 'abcada') FROM dual`
    - 替换字符：`REPLACE(表达式1,要替换的字符,替换后的字符)`
      `SELECT REPLACE('abcada,'a','*') FROM dual`

1. 数值函数
   - `ROUND(数据,保留位数)`：四舍五入
     常规：`SELECT ROUND(31415.926,2) FROM dual` `31415.93`
     不指定保留位数，取整：`SELECT ROUND(31415.926) FROM dual` `31416`
     保留位数是负数：`SELECT ROUND(31415.926,-1) FROM dual` `31420`
   - `TRUNC(数据,保留位数)`：截断
     常规：`SELECT TRUNC(31415.926,2) FROM dual` `31415.92`
     不指定保留位数：`SELECT TRUNC(31415.926) FROM dual` `31415`
     保留位数是小数：`SELECT TRUNC(31315.926,-1)FROM dual` `31410`
   - `mod(除数,被除数)`：取余
2. 日期函数
   日期的模式显示为 `DD-MON-RR`
   `SYSDATE`：返回系统日期和系统时间 `SELECT SYSDATE FROM dual`
   `ROUND()`、`TRUNC()`可用在日期类型上

```
SELECT e.first_name,e.hire_date,e.hire_date + 90, ROUND((SYSDATE-e.hire_date)/365) "日"
FROM employees e;
```

- `MONTH_BETWEEN(日期1,日期2)`：两个日期间相差的月数
  `sql SELECT SYSDATE, e.hire_date,month_between(SYSDATE,e.hire_date) 月份 FROM employees e`
- `ADD_MONTHS(日期,相差的月数)`：向指定日期中加上或减去若干月数
  `sql SELECT e.hire_date,add_months(e.hire_date,6),add_months(e.hire_date,-6) FROM employees e`
- `NEXT_DAY()`：指定日期的下一个日期
  - 数值：1-7（从周日开始）
  - 中文：星期一、星期二、星期三……
    `sql SELECT NEXT_DAY(SYSDATE,1) FROM dual`
    `sql SELECT NEXT_DAY(SYSDATE,'星期一') FROM dual`
- `LAST_DAY()`：当前日期的最后一天
  `sql SELECT last_day(SYSDATE) FROM dual;`

1. 转换函数
   i. `TO_CHAR(数字,格式)`
      - 日期类型转为字符类型
        `sql SELECT e.hire_date 转换前,to_char(e.hire_date,'yyyy/mm/dd') 转换后 FROM employees e;`
      - 数值类型转为字符类型
        `sql SELECT to_char(123456789.1264,'999999999.99') -- 123456789.13 -- 格式的长度要大于等于转换的数值类型的长度，同时四舍五入 FROM dual`
        格式符：
      a. 数字：`9`
      b. 占位：`0`

c. 美元符：`$`
d. 本地货币（￥）：`L`
e. 小数点：`.`
f. 千分符：`,`

ii. `TO_DATE(数字,格式)`：字符类型转为日期类型

```
SELECT *
FROM employees e
WHERE e.hire_date >= to_date('1990-01-01','yyyy-mm-dd')
-- 等价于
--SELECT *
--FROM employees e
--WHERE to_char(e.hire_date,'yyyy-mm-dd') >= '1990-01-01'
```

3. `TO_NUMBER(数字,格式)`：字符类型转为数值类 要求：格式的长度要大于等于转换的数值类型的长度；前面是字符类型的，后面是数值类型的

viii. 通用函数

4. `NVL2(表达式1，表达式2，表达式3)`：如果表达式1不为null ,返回表达式2；为null，返回表达式3

```
SELECT e.commission_pct, NVL2(e.commission_pct,1,2)
FROM employees e
```

2. `NULLIF(表达式1，表达式2)`：如果表达式1和表达式2相等，返回null；不相等，返回表达式1

```
SELECT LENGTH(e.first_name),LENGTH(e.last_name),NULLIF(LENGTH(e.first_name),LENGTH(e.last_name))
FROM employees e
```

3. `COALESCE(n个参数)`：返回第一个不为null的值

```
SELECT COALESCE(NULL,NULL,1,3,NULL)
FROM dual
```

1. 条件表达式

i. `CASE ...WHEN` `sql SELECT e.employee_id, e.first_name, e.salary, CASE e.job_id WHEN 'AD_VP' THEN e.salary * 1.1 WHEN 'IT_PROG' THEN e.salary *1.5 ELSE e.salary END 加薪 FROM employees e /*等价于 SELECT e.employee_id, e.first_name, e.salary, CASE WHEN e.job_id = 'AD_VP' THEN e.salary * 1.1 WHEN e.job_id = 'IT_PROG' THEN e.salary *1.5 ELSE e.salary END 加薪 FROM employees e/*`

b. `DECODE(字段名，值1，处理1，值2，处理2，...,默认值)`
`sql SELECT e.employee_id, e.first_name, e.salary, DECODE(e.job_id,'AD_VP',e.salary * 1.1,'IT_PROG',e.salary *1.5,e.salary) FROM employees e`

多表查询

Oracle自有连接

1. 笛卡尔积：所有表中的所有行和列互相连接
A表20行，B表8行，结果：20*8=160行
2. 等值连接（主外键连接）

i. 别名：定义在 `FROM` 子句中相同字段，要通过表名或表别名标识，如果表有表别名，那么就不能再用表原名了；有效范围为当前语句
```sql
SELECT first_name,d.department_id,l.city
```

FROM employees e ,departments d，locations l
WHERE e.department_id = d.department_id AND d.location_id = l.location_id AND e.department_id = 90

3. 非等值连接

sql
SELECT e.first_name,e.salary,j.grade_level
FROM employees e,job_grades j
WHERE e.salary BETWEEN j.lowest_sal AND j.highest_sal

4. 外连接
   1. 左外连接：查询左表所有数据，加号写在右边

sql
SELECT first_name,d.department_id,d.department_name
FROM employees e ,departments d
WHERE e.department_id = d.department_id(+)

   2. 右外连接：查询右表所有数据，加号写在左边

sql
SELECT first_name,d.department_id,d.department_name
FROM employees e ,departments d
WHERE e.department_id(+) = d.department_id;

2. 自连接
   例：一位员工可能是另一位员工的经理（即表示经理的数据也存在于员工表中），请查出该表中的经理

sql
SELECT e.employee_id,e.first_name,m.employee_id,m.first_name
FROM employees e,employees m
WHERE e.manager_id = m.employee_id

SQL99连接
1. `CROSS JOIN`（等价于笛卡尔积）

sql
SELECT *
FROM employees
CROSS JOIN departments;

2. 自然连接：`NATURAL JOIN`

sql
SELECT e.first_name,department_id,d.department_name -- 自然连接中相同的列不能指定表别名
FROM employees e
NATURAL JOIN departments d;
/* 等价于

SELECT *
FROM employees e,departments d
WHERE e.department_id = d.department_id
AND e.manager_id = d.manager_id;
*/

```
3. 使用`USING`子句创建连接
```

sql
SELECT *
FROM employees e
JOIN departments d
USING (department_id) -- 指定关联的列 （相同的列不能指定表别名）

```
4. 等值连接：`[INNER]JOIN ...ON`
```

sql
SELECT e.first_name,d.department_id,d.department_name,l.city
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id

```
5. 外连接：`[OUTER] JOIN...ON`
   1. 左外连接：`LEFT OUTER JOIN ON`
   2. 右外连接：`RIGHT OUTER JOIN ON`
   3. 满外连接：`FULL OUTER JOIN`
```

sql
SELECT e.first_name,d.department_id,d.department_name
FROM employees e
LEFT OUTER JOIN departments d ON e.department_id = d.department_id

```
## 组函数
`avg()`, `sum()`只能用于数值类型; `min()`,`max()`,`count()` 可以用于任意类型
**WHERE中不可以使用组函数**
```

sql
SELECT e.first_name,e.department_id,e.salary
FROM employees e
WHERE e.salary = MAX(e.salary)

```
1. `avg()`：平均值
```

sql
SELECT avg(e.salary) sal
FROM employees e

```
2. `sum()`：求和
3. `min()`：最小值
```

sql
SELECT min(e.salary) sal
FROM employees e

4. `max()`：最大值

sql
SELECT MAX(e.first_name) NAME -- 字符类型
FROM employees e

```sql
SELECT MAX(e.hire_date) NAME  -- 日期类型
FROM employees e
```

5. `count()`：计数
   组函数里是具体的列名时，结果会忽略NULL

```sql
SELECT COUNT(e.salary) sal  -- 数值类型
FROM employees e

SELECT COUNT(e.first_name) NAME  -- 字符类型
FROM employees e

SELECT COUNT(e.hire_date) NAME  -- 日期类型
FROM employees e

SELECT COUNT(*)
FROM employees e   -- 20

SELECT COUNT(e.department_id) -- 写具体的列名
FROM employees e   -- 19
```

6. `GROUP BY`

```sql
-- 查询各个部门的人数
SELECT e.department_id,COUNT(*)
FROM employees e
GROUP BY e.department_id
```
1. 使用分组函数的时候，**select**子句中的非组函数列，必须出现在**group by** 子句中；而**group by** 中的列，可以不出现在**select**子句中

sql
SELECT e.department_id,COUNT(*)
FROM employees e
GROUP BY e.department_id

2. `**ORDER BY**`中可以出现组函数

sql
SELECT e.department_id, COUNT() *NUM FROM employees e GROUP BY e.department_id ORDER BY COUNT()*

7. `HAVING`
   是在`GROUP BY`之后的过滤；`WHERE`是`GROUP BY`之前的过滤

sql
-- 查询各部门最高工资大于10000的部门信息
SELECT e.department_id,MAX(e.salary) sal
FROM employees e
GROUP BY e.department_id
HAVING MAX(e.salary) > 10000

```
## 子查询
写在括号中；先于主查询执行；可以使用组函数
查询比Abel工资高的员工
```

sql
SELECT e.last_name,e.salary
FROM employees e
WHERE e.salary > (SELECT e.salary
FROM employees e
WHERE e.last_name = 'Abel')

```
查询查询部门跟141号员工同部门的并且薪资大于143号员工工资的员工姓名，部门和薪资
```

sql
SELECT e.last_name,e.department_id,e.salary
FROM employees e
WHERE e.department_id = (SELECT e.department_id
FROM employees e
WHERE e.employee_id = 141)
AND e.salary > (SELECT e.salary
FROM employees e
WHERE e.employee_id =143)

```
查询工资最高的员工信息（姓名，部门，薪资）
```

sql
SELECT e.last_name,e.department_id,e.salary
FROM employees e
WHERE e.salary = (SELECT MAX(e.salary) sal
FROM employees e)

```
### 单行子查询
使用单行操作符：=, !=, >,>=,<,<=
```

sql
SELECT employee_id, last_name
FROM employees
WHERE salary = (SELECT MIN(salary)
FROM employees
GROUP BY department_id);

```
### 多行子查询
使用多行操作符：`in`,`any`,`all`
```

1. `IN`

sql
SELECT employee_id, last_name
FROM employees
WHERE salary IN (SELECT MIN(salary)
FROM employees
GROUP BY department_id);

2. `any`、`all`需要和单行操作符一起使用
   * `<any`：小于最大值
   * `>any`：大于最小值
   * `<all`：小于最小值
   * `>all`：大于最大值

sql
SELECT e.last_name,e.job_id,e.salary
FROM employees e
WHERE e.salary < ANY (SELECT e.salary
FROM employees e
WHERE e.job_id = 'IT_PROG')
AND e.job_id != 'IT_PROG';
/*等价于
SELECT e.last_name,e.job_id,e.salary
FROM employees e
WHERE e.salary < (SELECT MAX(e.salary)
FROM employees e
WHERE e.job_id = 'IT_PROG')
AND e.job_id != 'IT_PROG';
*/

员工的工资大于当前部门的平均工资

sql
SELECT e.last_name,e.job_id,e.salary
FROM employees e
WHERE e.salary < ALL (SELECT e.salary
FROM employees e
WHERE e.job_id = 'IT_PROG')
AND e.job_id != 'IT_PROG';

3. `EXISTS`操作符
   如果在子查询中存在满足条件的行：不在子查询中继续查找，条件返回`TRUE`;
   如果在子查询中不存在满足条件的行：条件返回`FALSE`，继续在子查询中查找
   查询员工表的经理有哪些

sql
-- EXISTS:找到即跳出比较，执行效率高
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
WHERE NOT EXISTS (SELECT 'X' -- ‘X’也可以，字段也可以
FROM employees
WHERE manager_id =

outer.employee_id);
```

# 操作数据

## 数据操作语言

1. `INSERT`： 插入数据
    i. 给出部分值

```
INSERT INTO DEPARTMENTS(department_id,department_name)
VALUES(70,'Helper')
```

2. 给出所有列名，对应插入所有值

```
INSERT INTO DEPARTMENTS
VALUES(40,'development',201,1800)
```

3. 不给出列名，值需全部写出，顺序和表中一致

```
INSERT INTO DEPARTMENTS
VALUES(30,'HR',200,1700)
```

4. 将一个表的结构和数据增加到新创建的表中

```
CREATE TABLE emp
AS
   SELECT *
   FROM employees e;
```

5. 创建表的时候，不带数据，只有表结构

```
CREATE TABLE emp
AS
   SELECT *
   FROM employees e
   WHERE 1 = 2;
```

6. 从一个表拷贝数据到另一个表

```
INSERT INTO emp
   SELECT *
   FROM employees;
```

插入数据时注意：

1. 约束
2. 字段类型（日期类型，可以通过Oracle提供的默认的日期类型插入，也可以使用`to_date()`插入）
```

```
INSERT INTO employees(employee_id,last_name,email,hire_date,job_id)
VALUES(888,'Jerry','123@com','30-9月-22','AD_ASST'); --默认的日期类型
```

```
INSERT INTO employees(employee_id,last_name,email,hire_date,job_id)
VALUES(999,'Tom','456@com',to_date('2022-09-30','yyyy-mm-dd'),'ST_MAN');-- to_date()插入
```

2. `UPDATE`：修改数据
   多列之间逗号分隔，注意where条件(若不写where条件，则修改表所有数据)

```
UPDATE employees e
SET e.department_id = 80,e.salary = 40000
WHERE e.last_name IN ('Jerry','Tom')
```

3. `DELETE`：删除数据
   `DELETE from`可以省略，最好不省略。注意where条件(若不写where条件，则删除表所有数据)

```
SELECT e.* FROM employees e;
DELETE FROM emp e
WHERE e.employee_id = 888
```

4. 使用`ROWID`操作数据
   `ROWID`：伪列；每一行记录都包含着ROWID，表示这一行的唯一地址

```
SELECT ROWID,e.*
FROM employees e
WHERE ROWID = 'AAAShLAAHAAAADLAAC'
```

5. `FOR UPDATE`：行锁，需要增加where条件；如果不加where条件，锁整表

```
SELECT *
FROM employees e
WHERE e.employee_id = 888
FOR UPDATE
```

数据库事务

1. 事务：是由一个或多个SQL语句组成的序列，这些SQL要么全部执行，要么全部不执行
2. 事务特征：ACID
3. 事务的开始与提交
   i. 开始：以DML语句的执行作为开始
   ii. 提交（结束）：`COMMIT`（提交）；`ROLLBACK`（回滚）
4. 设置保存点：`SAVEPOINT`
   i. 定义保存点：`SAVEPOINT 保存点名称`
   ii. 回滚到已定义保存点：`ROLLBACK TO 保存点名称`

表和约束

表

1. 表名和列名的命名规范
   - 必须以字母开头
   - 必须在1~30个字符之间
   - 必须只能包含A–Z, a–z, 0–9, _, $, 和#
   - 必须不能和用户定义的其他对象重名
   - 必须不能是Oracle的关键字
2. 创建表

```
CREATE TABLE 表名
(
   列名 数据类型 约束
   ……
)
```

3. 数据类型
   i. 字符类型
      - `varchar(size)`：可变长度，最小字符数是 1，最大字符数是 3276
        汉字占一个字符
      - `char(size)`：固定长度字符数据，指定了字符长度，不足位右补0；长度的大小以字节为单位，默认和最小字符数为1；最大字符数为 2000
        查询数据时，char类型最好加trim,去空格
        `sql SELECT * FROM my_table WHERE NAME = trim(sex)`
   ii. 数值类型
   `NUMBER(p,s)`：总长度为p，小数位最大为s位 ,NUMBER默认长度38

```
CREATE TABLE my_table(
     ID NUMBER(3),
     NAME VARCHAR(50),
     sex CHAR(6),   -- 字节
     hire_date DATE DEFAULT SYSDATE  --默认值的作用：插入数据时若不给值此处的值将替换插入的值
    )
```

3. 日期和时间类型：`DATE`
   iv. 引用其他用户创建的表：`要使用的用户名.对象名`
   `SELECT * FROM c##dhee.departments;`
   v. `ALTER TABLE`
   - 追加新的列

```
ALTER TABLE table
ADD(column datatype [DEFAULT expr]
 [, column datatype]...);
```

- 修改现有的列

```
ALTER TABLE table
MODIFY(column datatype [DEFAULT expr]
[, column datatype]...);
```

- 为新追加的列定义默认值
- 删除一个列

```
ALTER TABLE table
DROP(column);
```

6. 删除表
   i. `DROP`：删除表结构和数据，彻底删除，不能回滚
   ii. `TRUNCATE`：清空表、清空数据，不能加where，重置表结构；清空空间，不能回滚，执行效率高
   iii. `DELETE`：删除表数据，可以加where，可以回滚

约束

1. not null：非空。只能写在行上(行级)
2. unique：唯一。可以为空，只能有一个
3. primary key：主键。非空且唯一
4. foreign key：外键。级联删除
5. check：检查。自定义，必须为真的条件
   `COMMENT`：给表加注释

其他数据库对象

视图

可通过视图进行DML操作，实现对DML操作（要求：行对应；列存在；符合约束条件）

1. 作用：别名；子查询指定列名

```
CREATE OR REPLACE VIEW view_emp(NAME,dept,sal)
-- REPLACE：如果所创建的视图已经存在，Oracle会自动重建该视图
AS
   SELECT e.last_name,e.department_id,salary * 12 * ( 1 + nvl(e.commission_pct,0)) sal
   FROM employees e
   WHERE e.department_id = 80
```

2. 创建视图

```
CREATE [OR REPLACE][FORCE|NOFORCE]] VIEW 视图名
AS
(
   子查询...
)
[WITH CHECK OPTION] CONSTRAINT 约束名]]
[WITH READ ONLY] CONSTRAINT 约束名]]
```

- `OR REPLACE`：如果当前创建的视图存在，Oracle会自动重建该视图
- `FORCE`：不管基表是否存在Oracle都会创建该视图
- `WITH CHECK OPTION`：保证插入、修改、删除的数据行满足视图定义的约束
- `WITH READ ONLY`：保证在该视图上不能进行DML操作

1. 删除视图
   `DROP VIEW 视图名`

分页

1. `ROWNUM`：伪列，只能执行`<`、`<=`运算(可以`=1`)
   例：将某表中数据（共21条）分成五页，每页五条记录
   思路：

```
/*
   pageNum pageSize Result
      1       5      1~5
      2       5      6~10
      3       5      11~15
      ...
      n       5      n*5 >= count > (n-1)*5
*/
```

```
SELECT *
FROM (
      SELECT ROWNUM rn,emp.*
      FROM (
            SELECT *
            FROM employees e
            ORDER BY e.salary DESC
            ) emp
      WHERE ROWNUM <=  (2 * 5)   -- n * 5(每页的记录数*目标页数)
      ) v_e
WHERE v_e.rn > (2-1) * 5
-- (n-1) * 5(每页的记录数*(目标页数 -1))
```

序列

1. `CURRVAL`： 存放序列当前值
2. `NEXTVAL`： 在 `CURRVAL` 之前给定

```
SELECT emp_seq.currval   -- 序列创建好了之后，不能先执行currval
FROM dual;
```

索引

1. 自动创建：在定义 `PRIMARY KEY` 或 `UNIQUE` 约束后系统自动在相应的列上创建唯一性索引
2. 用户可以在其它列上创建非唯一的索引，以加速查询

同义词

1. 作用：访问其他用户的对象
2. 创建

```
    CREATE SYNONYM 同义词名
    FROM 表名
```

集合运算

1. `UNION`： 返回两个查询的结果集的不重复部分的并集
2. `UNION ALL`： 操作符返回两个查询的结果集的并集以及两个结果集的重复部分（不去重）
3. `INTERSECT`： 操作符返回两个结果集的交集
4. `MINUS`： 操作符返回两个结果集的补集