一、数据库简介
（一）什么是数据库
（二）数据库分类
（三）数据库安装及用户创建
（四）约束条件
（五）关联关系
（六）DDL、DML、DCL、DQL
（七）注释
二、数据类型
三、DDL
（一）创建表
（二）主外键
（三）修改表
（四）删除表结构
四、DML
（一）插入数据
（二）修改数据
（三）删除表数据
五、DQL
（一）普通语句
（二）条件查询
（三）锁表查询
（四）模糊查询
（五）排序
（六）分组
（七）去重
（八）嵌套查询
（九）分页查询（重点）
（十）关联查询
1、并集
2、交集
3、差集
4、内连接
5、左连接（左外连接）
6、右连接（右外连接）
7、全连接（全外连接）
六、函数
（一）字符函数
（二）数学函数
（三）日期函数
（四）转换类函数
（五）集计函数/统计函数/聚合函数
（六）其他函数
七、数据库对象
（一）视图
（二）索引
（三）序列
八、事务
九、PLSQL（数据库编程语言）
（一）匿名代码块
（二）变量和常量
（三）数据类型
标准类型

%type
record（记录类型）
%rowtype
varray（数组类型）
集合类型
（四）表达式
（五）流程控制
1、判断
2、循环
（五）游标
（六）存储过程
（七）函数
（八）异常
（九）触发器
十、MySQL 对比

一、数据库简介

（一）什么是数据库

```
记录信息的方式：
    龟甲、竹简、纸（东汉蔡伦）、电子存储

信息的保存周期：
    瞬时记忆、长久记忆

数据库(DataBase)：
    数据库是“按照数据结构来组织、存储和管理数据的仓库”。是一个长期存储在计算机内的、有组织的、可共享的、统一管理的大量数据的集合。

数据：
    文字数据、图片数据、音频数据、视频数据

数据换算单位：
    1 KB = 1024 B
    1 MB = 1024 KB
    1 GB = 1024 MB
    1 TB = 1024 GB
    1 PB = 1024 TB
```

（二）数据库分类

```
关系型数据库
    Oracle、MySQL、SQL Server、Postgre、Assess、DB2等

    Oracle版本：9i -> 10G -> 11G -> 12C -> 18C -> 19C -> 21C
    MySQL版本：5.x -> 8.x

非关系型数据库
    Redis、MongoDB、MapReduce等
```

（三）数据库安装及用户创建

1、数据库安装

```
服务端：数据的服务器，有硬件要求，配置较低的不建议安装。
客户端：连接数据的工具，可以为可视化工具提供桥梁
可视化工具：以可视化视图的形式操作数据库。PLSQL Developer、A5M2、Navicat Premium
```

2、数据库的配置

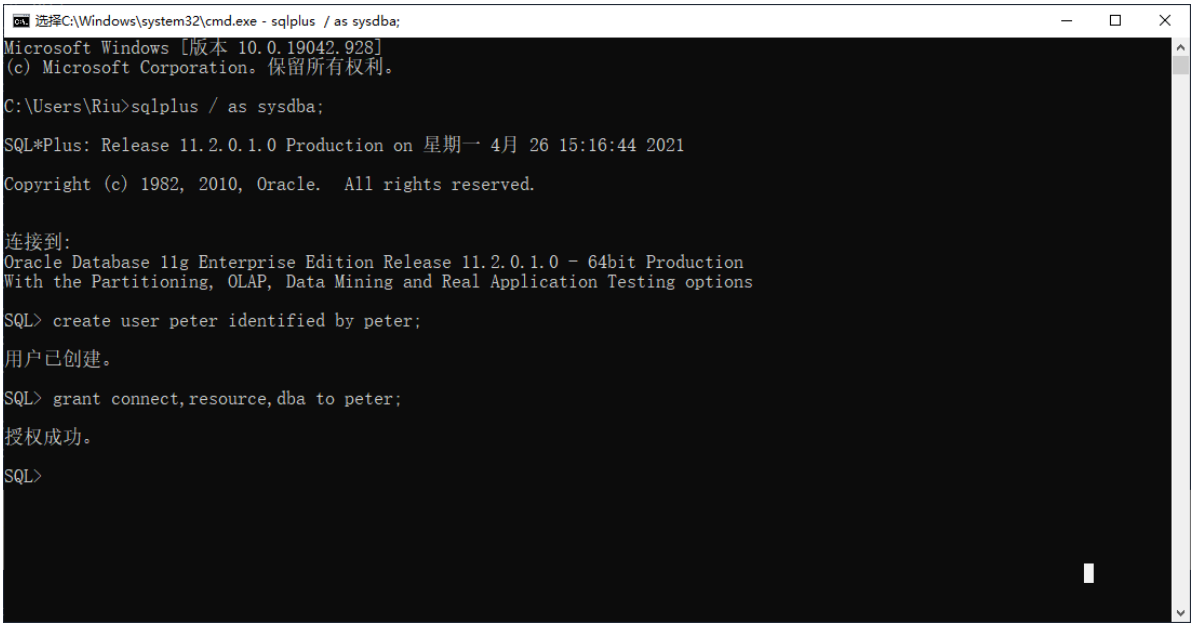
```
连接服务器时需要的信息：
    地址：服务器的IP地址，
        如果服务器在本地（自己的机器），可以填localhost或者127.0.0.1或者本机IP地址（192.168.2.1）
        如果服务器不在本地，必须要填写远程的服务器IP地址（192.168.2.3），项目中远程的地址使用的频率较高
    账号：自己创建的账号，自定义，项目中的账号是有权限限制的
    密码：创建账号是设置的密码，自定义
    端口号：1521
    服务器名称：默认是orcl

orcl =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )

技术开发常见的端口号：
    浏览器端口：80
    Tomcat端口：8080
    Oracle端口：1521
    MySQL端口：3306
    Redis：6379
    FTP：21（传输）、26（管理）
    SMTP：25
```

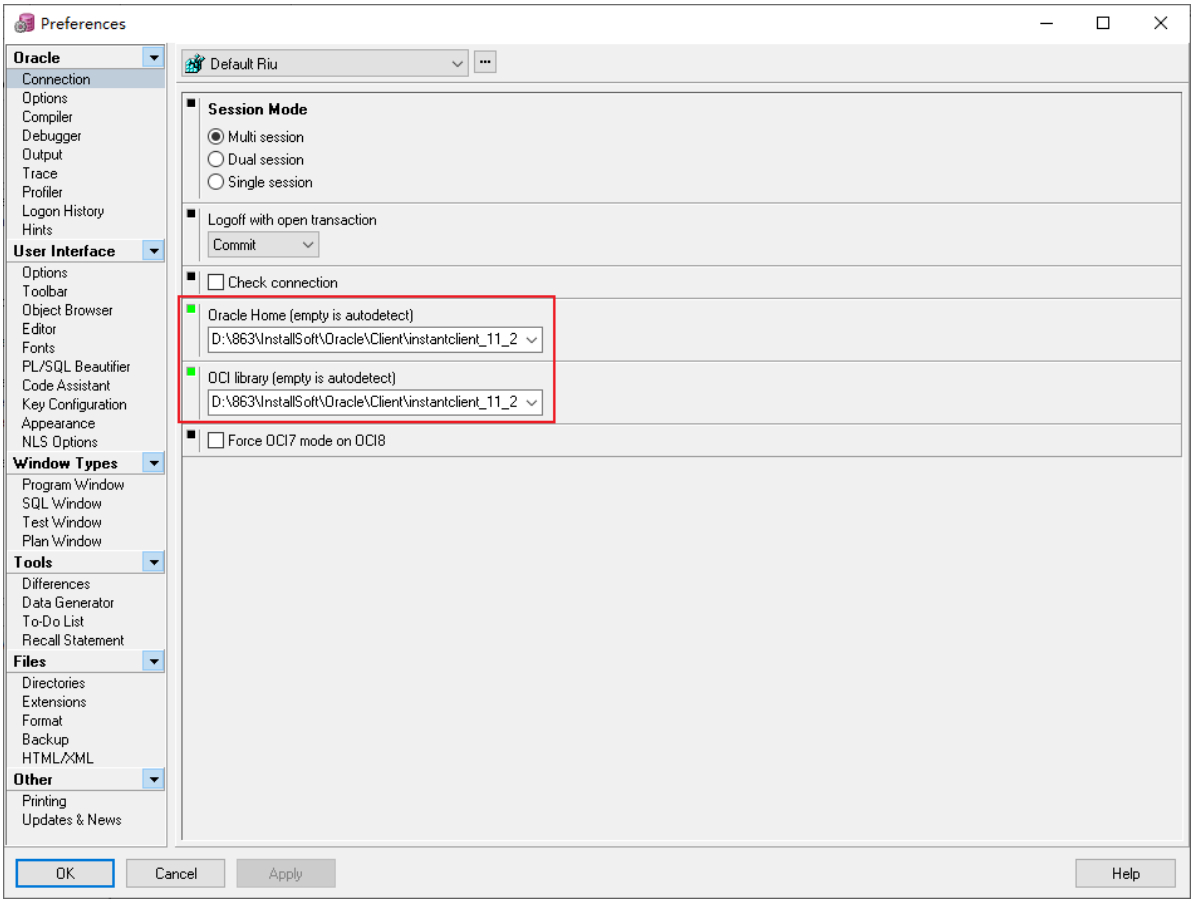
3、创建账号密码、添加权限

```
以下命令统一使用管理员权限：
    1、打开cmd命令提示符，输入【sqlplus / as sysdba;】
    2、输入【create user 用户名 identified by 密码;】创建用户
    3、输入【grant connect,resource,dba to 用户名;】给用户赋权限
```



4、通过可视化工具链接数据库

```
地址：数据库服务器所在的IP地址
用户名：自定义
密码：自定义
```




```
注意：以下两个信息均为客户端安装目录下的路径
    Oracle Home：D:\863\InstallSoft\Oracle\Client\instantclient_11_2
    OCI library：D:\863\InstallSoft\Oracle\Client\instantclient_11_2\oci.dll
```

（四）约束条件

> 可以为一个表列创建约束条件，此时，表中的每一行数据都必须满足约束条件定义所规定的条件

```
主键约束（primary key）：单一主键、联合主键
    主键是表中的一列或多个列。为表定义主键有几个作用，主键包含的列不能输入重复的值，以保证一个表的所有的行的唯一性；主键也不允许定义此约束的列为NULL值；主键在定义此约束的列中创建了唯一性的索引，利用这个索引可更快地检索表中的数据。

默认约束条件（default）
    在表中插入一行数据但没有为列指定值时生成一个在定义表时预先指定的值。

检查约束条件（check）
    该约束条件确保指定列中的值符合一定的条件。

惟一性约束条件（unique）
    用于保证不是主键那些列的惟一性。
    例如：手机号、身份证号码、车牌号

外键约束条件（foreign key）：外键添加在子表中
    该约束条件规定多表间的关系。一个外键使一个表的一列或多列与已定义为主键的表中的一批相同的列相关联。
    例如：学生表（主表：学号）和成绩表（子表：学号）。成绩是依赖学生的

非空约束（not null）
    该约束规定插入的数据不能为NULL。
```

（五）关联关系

```
一对一：一张表中的一条数据在另一张表只有一条对应；例如：一个学生只能对应一个班级；
一对多：一张表中的一条数据在另一张表有多条对应；例如：一个学生有多课成绩；
多对一：一张表中的多条数据在另一张表只有一条对应；例如：多个成绩可以对应一个学生；
多对多：一张表中的多条数据在另一张表中有多条对应；例如：多个学科可以对应多个成绩；
```

（六）DDL、DML、DCL、DQL

> 数据库语言分为：DDL、DML、DCL、DQL
>
> **DDL（Data Definition Language 数据定义语言）**用于操作对象和对象的属性，对象包括数据库本身，以及数据库对象，例如：表、视图等，DDL对这些对象和属性的管理和定义具体表现在**Create（创建）**、**Drop（删除）**和**Alter（修改）**上。
>
> **DML（Data Manipulation Language 数据操控语言）**用于操作数据库对象中包含的**数据**，也就是说操作的单位是记录。
>
> **DQL（Data Query Language 数据查询语句）**用于数据库的数据查询操作。
>
> **DCL（Data Control Language 数据控制语句）**用于数据库对象的权限，这些操作的确定使数据更加的安全。

（七）注释

```
单行注释：
    -- 单行注释
多行注释：
    /*
        多行注释
    */
```

二、数据类型

（一）字符类型

```
char(n)：字符长度是n，如果数据的长度不足n，会以空格补全；最大长度是2000。
    char(10)
        '123' -> '123       '
    特点：
        1、定长
        2、会补空格
        3、最大长度2000

varchar(n)：字符长度是n，不会补空格。最大长度是4000。
    varchar(10)
        '123' -> '123'

    特点：
        1、不定长
        2、不会补空格
        3、最大长度是4000

varchar2(n)：Oracle数据库特有的数据类型，字符长度是n，不会补空格。最大长度是4000；

    varchar2(10)
        '123' -> '123'

clob：大文本类型，最大存储4G数据，用于存储小说、文本、新闻等；
blob：大字节类型，最大存储4G数据，用于存储图片、视频、音频等；
```

varchar和varchar2的区别 1、varchar 是标准的SQL数据类型，varchar2 是 Oracle 提供的特有的数据类型； 2、varchar 对于汉字占2个字节，对于字母和数字占1个字节；varchar2都是2个字节； 3、varchar 对空字符串不处理，varchar2会把空字符串当作NULL处理； 4、varchar 存放固定长度的字符串，最大长度是4000，varchar2 是存放可变长度的字符串，最大长度是4000。

（二）数值类型

```
number(p,s)：
    可以存放数据范围为10^130-10^126（不包含此值），需要1~22字节(BYTE)不等的存储空间；
    p是Precison的英文缩写，即精度缩写，表示有效数字的位数，最多不能超过38个有效数字；从左边第一个不为0的数算起；
    s是Scale的英文缩写，可以使用的范围为-84~127。Scale为正数时，表示从小数点到最低有效数字的位数，它为负数时，表示从最大有效数字到小数点的位数；

    p表示包含小数在内的最大总位数，s表示小数的最大位数；
        s > 0，精确到小数点右边s位，并四舍五入。然后检验有效位是否<=p。
        s < 0，精确到小数点左边s位，并四舍五入。然后检验有效位是否<=p+|s|。
```

```
        s = 0，此时number表示整数。

    number(5, 2)
        123.01

integer：integer是number的子类型，它等同于number(38,0)，用来存储整数。若插入、更新的数值有小数，则会被四舍五入。
```

（三）日期类型

```
date：DATE是最常用的数据类型，日期数据类型存储日期和时间信息。虽然可以用字符或数字类型表示日期和时间信息，但是日期数据类型具有特殊关联的属性。为每个日期值，Oracle 存储以下信息： 世纪、年、 月、 日期、 小时、 分钟和秒。一般占用7个字节的存储空间。

timestamp：这是一个7字节或12字节的定宽日期/时间数据类型。它与DATE数据类型不同，因为TIMESTAMP可以包含小数秒，带小数秒的TIMESTAMP在小数点右边最多可以保留9位
```

三、DDL

（一）创建表

```sql
-- 语法结构
create table 表名(
    字段名称 字段类型 [约束条件],
    字段名称 字段类型 [约束条件],
    字段名称 字段类型 [约束条件],
    ......
);

表名的命名规则：
    tb_表名
```

（二）主外键

```sql
-- 添加主键
-- 1、单一主键（列级）
create table 表名(
    字段名称 字段类型 primary key,
    ......
);

-- 2、联合主键（表级）：联合主键要求所有的值一样才能标识唯一性；
create table 表名(
    字段名1 类型,
    字段名2 类型,
    ...
    constraint pk_主键名称 PRIMARY KEY (字段1, 字段2, ...) -- 联合主键
);

-- 3、创建表之后添加联合主键
alter table 表名 add constraint pk_主键名称 primary key (字段1, 字段2, ...);
```

```sql
-- 添加外键
-- 在创建表时候设置外键约束（列级）
create table 表名1(
    字段名1 类型 references 表名2(字段名)
);

-- 在创建表时设置外键约束（表级）
create table 表名1(
    字段名1 类型,
    constraint fk_外键名称 foreign key(表名1的字段名1) references 表名2(字段名)
);

-- 在创建表之后添加外键约束
alter table 表名1 add constraint fk_外键名称 foreign key(表名1的字段名1)
references 表名2(字段名)
```

（三）修改表

```sql
-- 添加注释
comment on column 表名.字段名 is '注释/备注';
-- 修改表名
alter table 表名 rename to 新表名;
-- 修改字段名称
alter table 表名 rename column 旧列名 to 新列名;
-- 添加字段
alter table 表名 add 字段名称 类型;
-- 修改字段类型
alter table 表名 modify(字段名 新数据类型);
-- 删除字段
alter table 表名 drop column 字段名;
```

（四）删除表结构

```sql
-- 语法结构
drop table 表名;
```

四、DML

CRUD：C（Create）R（Retriieve）U（Update）D（Delete）

（一）插入数据

```sql
-- 语法结构
-- 部分字段插入，值的顺序和字段的顺序对应
insert into 表名(字段名1, 字段名1, ...) values (值1, 值2, ...);
-- 全字段插入，值的顺序和表中字段的顺序对应
insert into 表名 values (值1, 值2, ...);

-- 表的拷贝语句
insert into 表名(
    select * from 表名
    union all
```

```
    select * from 表名
);

-- 插入数据之后一定要点击提交按钮提交，否则查不到数据；
```

（二）修改数据

```
-- 语法结构
update 表名 set 字段 = 值, 字段 = 值, ... [where 字段 = 值 and 字段 = 值 or 字段 = 值 and 字段 <> 值];

-- 修改数据之后一定要点击提交按钮提交，否则数据不能修改成功；
```

（三）删除表数据

```
-- 语法结构
delete [from] 表名 [where 字段 = 值 and 字段 = 值 or 字段 = 值 and 字段 <> 值];
truncate table 表名;

-- 删除数据之后一定要点击提交按钮提交，否则数据不能删除成功；

-- 以上情况称为物理删除
-- 逻辑删除：给表添加一个删除的标识字段（delFlg），
    delFlg = 1表示删除状态；
    delFlg = 0表示非删除状态；
    查询时可以将该字段作为查询条件
```

drop、delete、truncate区别：

```
1、truncate 与不带 where 的 delete ：只删除数据，而不删除表的结构（定义）
2、truncate table 删除表中的所有行，释放空间，但表结构及其列、约束、索引等保持不变。如果想保留标识计数值，请改用delete。
3、如果要删除表定义及其数据，请使用 drop table 语句。
4、执行速度，一般来说: drop > truncate > delete。
5、delete 语句是数据库操作语言(DML)，这个操作会放到 rollback segement 中，事务提交之后才生效；如果有相应的 trigger，执行的时候将被触发。
6、truncate、drop 是数据库定义语言(DDL)，操作立即生效，原数据不放到 rollback segment 中，不能回滚，操作不触发 trigger。
```

五、DQL

（一）普通语句

```
-- 语法结构
-- 全查询（不推荐），项目中全查询需要把子段全部写出来
select * from 表名;
-- 部分字段查询
select 字段名1, 字段名2, ... from 表名;
-- 关键字：dual
dual是Oracle中的伪表，作用就是保证sql的完成性；

select 'a', 'b', 'c' from dual;
```

（二）条件查询

```
条件之间的表达式：
    并且：and
    或者：or
    包含：in(a, b, ...)
    不包含：not in(a, b, ...)
    存在：exists()
    不存在：not exists()
    非：<>(推荐) !=
    大于：>
    小于：<
    大于等于：>=
    小于等于：<=
    等于：=
    空：is null
    非空：is not null
    范围：between ... and ...

-- 有条件查询
select 字段名, 字段名, ... from 表名 where 字段 = 值;
select 字段名, 字段名, ... from 表名 where 字段 > 值;
select 字段名, 字段名, ... from 表名 where 字段 >= 值;
select 字段名, 字段名, ... from 表名 where 字段 < 值;
select 字段名, 字段名, ... from 表名 where 字段 <= 值;
select 字段名, 字段名, ... from 表名 where 字段 <> 值;
select 字段名, 字段名, ... from 表名 where 字段 is [not] null;
select 字段名, 字段名, ... from 表名 where 字段 [not] between 开始值 and 结束值;
select 字段名, 字段名, ... from 表名 where 字段 = 值 and 字段 = 值;
select 字段名, 字段名, ... from 表名 where 字段 = 值 or 字段 = 值;
select 字段名, 字段名, ... from 表名 where 字段 [not] in (值1, 值2, ...);
select 字段名, 字段名, ... from 表名 where 字段 [not] in (子查询);
```

（三）锁表查询

```
select 字段名, 字段名, ... from 表名 for update;
-- for update是锁表语句，语句不要轻易使用，一旦使用，表会被执行者锁定，其他人员将不能操作该表；
-- 一旦使用之后要及时提交；
```

（四）模糊查询

```
-- 关键字：like，百分号在前表示以某个字符结尾，百分号在后，表示以某个字符开头，百分号在两边表示包含某个字符；
select 字段 from 表名 where 字段 like '%a';  -- 以a结尾
select 字段 from 表名 where 字段 like 'a%';  -- 以a开头
select 字段 from 表名 where 字段 like '%a%'; -- 包含a
```

(五) 排序

```
关键字：order by
升序：asc，数据库默认是升序，可以省略不写
降序：desc

select 字段 from 表名 [where 条件] order by 字段, 字段;
select 字段 from 表名 [where 条件] order by 字段 desc, 字段 asc;

购物订单按照什么排序？
    按照下单时间排序
```

(六) 分组

> 分组的使用要求（*）： 1、group by 的字段必须在 select 后面体现； 2、可以使用集计函数/聚合函数； 3、排序的字段必须包含在分组的字段中；

```
分组就是将多条记录中按照字段，值相同的记录归为一类，归类后可以进行统计，比如求最大值、最小值、平均值、和等。
关键字：group by
分组的条件关键字：having

select 字段1 from 表名 [where 条件] group by 字段1 [having 条件] [ordery by];

where、group by、having、order by的顺序：
    where > group by > having > order by
```

(七) 去重

```
关键字： distinct，将多条重复的记录去重，两个记录中所有字段的值一样时才可以去重
select distinct 字段,... from 表名;

通过分组也可以实现去重的操作
```

(八) 嵌套查询

```
嵌套查询（子查询）
从一个sql的结果集中通过另外一个sql进行查询；

select t.字段 from (select 字段 from 表1) t where t.条件
select t.字段 from 表名 t where 条件 in (select 字段 from 表1)
select t.字段 from 表名 t where 条件 exists (select 字段 from 表1)
```

```
-- 要查询有成绩的学生的信息

分析：
    1、查的是学生信息，要从学生表中查询，根据指定的学号(从成绩表中查)
    2、有成绩表示在成绩表中有该学生的学号

    1、select distinct 学号 from 成绩表
    2、select * from 学生表 where 学号 in (select distinct 学号 from 成绩表);
```

- in 和 exists 的区别:

1. 如果子查询得出的结果集记录较少，主查询中的表较大且又有索引时应该用 IN, 反之如果外层的主查询记录较少，子查询中的表大，又有索引时使用 EXISTS。
2. 其实我们区分 IN 和 EXISTS 主要是造成了驱动顺序的改变(这是性能变化的关键)，如果是 EXISTS，那么以外层表为驱动表，先被访问，如果是 IN，那么先执行子查询，所以我们会以驱动表的快速返回为目标，那么就会考虑到索引及结果集的关系了 ，另外 IN 不对 NULL 进行处理。
3. 如果查询语句使用了 NOT IN 那么内外表都进行全表扫描，没有用到索引；而 NOT EXISTS 的子查询依然能用到表上的索引。所以无论哪个表大，用 NOT EXISTS 都比 NOT IN 要快。

(九) 分页查询（重点）

> 关键字：rowid、rownum，都不能直接显示，需要额外查询。 rowid：oracle 中的伪列，用于表示数据的唯一性。 rownum：orcale 中的伪列，用于表示当前数据 **结果集** 中记录的序号。每个结果集的开始序号都是 1；

```
分页查询，就是在数据量非常大的时候通过 SQL 的控制一次只查询固定数量的数据展示给用户。

1、rownum 是伪列，查询时不显示的，先把 rownum 和表中所有的记录都查询出来
    select rownum num, t.* from tb_student t
2、从第1步的结果集中查询数据，并根据 rownum 的范围实现分页

-- 不排序
select *
  from (select rownum num, t.* from tb_student t)
 where num >= startNum
   and num <= endNum;

-- 排序
select *
    from (select rownum num, t.*
          from (select * from tb_student order by 字段, ...) t
         )
 where num >= startNum
   and num <= endNum;


计算分页的各参数：
    1、数据总条数：totalCount
    2、每页显示条数：limit
    3、总页数：totalPage
        totalCount % limit == 0
            totalCount / limit

         totalCount % limit != 0
```

```
            totalCount / limit + 1

    4、当前页：pageNum
    5、结束位置：pageNum * limit
    6、开始位置：结束位置 - (limit - 1)
```

当前页数	开始位置	结束位置
1	1	10
2	11	20
3	21	30

```
-- MySQL分页：limit 关键字
    select * from table_name limit index [,limit]

    index：数据的开始下标；只有index参数时，表示查询前index条
    limit：每页显示条数

例子：
    -- 1、每页显示5条，获取第2页数据
    select * from tb limit 6, 5

    -- 2、获取前10条数据
    select * from tb limit 10
```

（十）关联查询

1、并集

```
-- 关键字： union、union all
-- 将两个结果集合成为一个结果集，union 可以将结果集中重复的数据去掉，union all不会去除重复的数据；

-- 存在不重复的数据（去重）
select 字段1, 字段2, 字段3 ... from 表1
union
select 字段1, 字段2, 字段3 ... from 表2
union
select 字段1, 字段2, 字段3 ... from 表3

-- 存在重复的数据（不去重）
select 字段1, 字段2, 字段3 ... from 表1
union all
select 字段1, 字段2, 字段3 ... from 表2
union all
select 字段1, 字段2, 字段3 ... from 表3

注意点：
    1、要求合并数据的字段数量一致
    2、要求合并数据的对应的字段类型必须一致
    3、最终结果集的列名和第一个结果集的列名一致
```

2、交集

```
关键字：intersect

select 字段1, 字段2, 字段3 ... from 表1
intersect
select 字段1, 字段2, 字段3 ... from 表2
```

3、差集

```
关键字：minus
select 字段1, 字段2, 字段3 ... from 表1
minus
select 字段1, 字段2, 字段3 ... from 表2
```

4、内连接

```
-- 关键字：inner join

-- 语法结构1：
    select 字段1, 字段2, ...
      from 表1 t1
inner join 表2 t2
        on 关联条件
    [where]
    [group by ]
    [order by]

-- 语法结构2：
    select 字段 from 表1 t1, 表2 t2 where 条件

-- 特征：
    多张表能通过关联条件查询到的数据展示，查询不到的不展示，就是多张表的交集；
```

```
-- 要查询有成绩的学生的信息（没有成绩就不显示）

分析：
    1、有成绩也就是在成绩表中有当前学生，也就是有当前学生的学号。
    2、学生的信息包括学号等。
    3、关联条件就是两张表的学号。

     select *
       from 学生表 t1
innert join 成绩表 t2
         on t1.学号 = t2.学号
```

5、左连接（左外连接）

```
-- 关键字：
    left join
    left outer join

-- 语法结构：
    select 字段1, 字段2, ...
      from 表1 t1
 left join 表2 t2
        on 关联条件
    [where]
    [group by]
    [order by]

-- 特征：
    主表（left左边的表）的数据全部展示，子表匹配上的展示，匹配不上的展示为 null
```

```
-- 查询所有学生的成绩信息

分析：
    1、有学生可能没有成绩，但是这个学生的信息还要展示
    2、以学生表为主表，以成绩表为辅表

   select *
     from 学生表 t1
left join 成绩表 t2
       on t1.no = t2.no
```

6、右连接（右外连接）

```
-- 关键字：
    right join
    right outer join

-- 语法结构：
    select 字段
      from 表1 t1
 right join 表2 t2
        on 关联条件
    [where 查询条件]
    [group by ]
    [order by]

-- 特征：
    主表（right右边的）的数据全部展示，子表匹配上的展示，匹配不上的展示为 null
```

7、全连接（全外连接）

```
-- 关键字：
    full join
    full outer join

-- 语法结构：
    select 字段
      from 表1 t1
 full join 表2 t2
        on 关联条件
    [where 查询条件]
    [group by ]
    [order by]

-- 特征：
    完整外部连接返回左表和右表中的所有行，两张表关联上的数据展示，关联不上的数据展示为空
```

六、函数

（一）字符函数

```
-- 字符拼接1
concat(arg1, arg2)
-- 拼接字符2
||

-- Oracle 写法：必须要保证一个语法结构的完整性
select concat('a', 'b') str from dual;
select concat('a', concat('b', concat('c', 'd'))) str from dual;
select 'a' || 'b' || 'c' || 'd' str from dual;

-- MySQL 写法
select concat('a', 'b', 'c', 'd');

-- 大小写转换
lower(arg); 转换为小写
upper(arg); 转换为大写
initcap(arg); 首字母大写

select lower('A'), upper('b') from dual;

-- 字符长度
length(arg); 获取字符长度

select length('abcdefg') length from dual;

-- 去空格
trim(arg); 只能去掉字符两边的空格，中间的空格无法去掉
ltrim(arg); 去掉字符左边的空格
ltrim(arg1, arg2); 去掉arg1左边的arg2
rtrim(arg); 去掉字符右边的空格
rtrim(arg1, arg2); 去掉arg1右边的arg2
```

```
select trim(' abc d ef ') from dual;
select ltrim(' abc d ef ') from dual;
select rtrim(' abc d ef ') from dual;

-- 替换
replace(arg1, arg2); 把arg1中的arg2删掉
replace(arg1, arg2, arg3); 把arg1中的arg2替换成arg3

select replace('abcd', 'a') from dual;
select replace('abcd', 'a', 'A') from dual;

-- 截取字符串
substr(str, index); 依据 str 从 index 的下标位置截取到最后，包含 index 位置的值;
substr(str, idnex, length); 依据 str 从 index 的下标位置截取 length 长度，如果 length 超出了字符的长度，则默认截取到最后一位;
Oracle中的下标从 1 开始；如果写0，也是从1开始；

select substr('abcde', 2) from dual;
select substr('abcde', 2, 3) from dual;
select substr('abcde', 2, 5) from dual;

-- 获取指定值的索引
instr(str, subStr); 返回 subStr 在  str 中的第一次出现的索引位置
instr(str, subStr, count); 返回 subStr 在 str 中的第 count 次出现的索引位置
```

(二) 数学函数

```
-- 绝对值
abs(n);
select abs(-100) from dual;

-- 向上取整
ceil(n);
select ceil(3.14) from dual;

-- 向下取整
floor(n);
select floor(3.14) from dual;

-- 取余
mod(n1, n2);
select mod(10, 3) from dual;

-- 次幂
power(n1, n2);
select power(2, 10) from dual;

-- 保留有效数，四舍五入
round(n1, n2);
select round(3.5555, 3) from dual;

-- 保留有效数，不四舍五入
trunc(n1, n2);
```

```
select trunc(3.5555, 3) from dual;
```

（三）日期函数

- Oracle

```
-- 获取系统时间
sysdate

select sysdate from dual;

-- 月份计算
add_months(d, n); 计算d日期添加n月之后的值

-- 月份的最后一天
last_day(d);
```

- MySQL

```
-- 格式化日期
date(now()), month(now()), day(now()), year(now()), hour(now()),
minute(now())

-- 系统时间
select now(), curdate();
-- 系统时间周的第 n 天
select WEEKDAY(curdate());
-- 系统时间所属月
select MONTH(curdate());
-- 系统时间月的最后一天
select last_day(curdate());
-- 系统时间在月的第 n 天
select DAYOFMONTH(curdate());

-- date_sub 从日期减去指定的时间间隔
-- 本周开始
select date_sub(curdate(), INTERVAL WEEKDAY(curdate()) DAY);
-- 本周结束
select date_sub(curdate(), INTERVAL WEEKDAY(curdate()) - 6 DAY);
-- 本月开始
select date_sub(curdate(), INTERVAL DAYOFMONTH(curdate()) -1 DAY);
-- 本月结束
select last_day(curdate());
```

（四）转换类函数

Oracle：

```
-- 日期转字符串
to_char(date, format);
-- 字符串转日期
to_date(string, format);

```

```
select to_char(sysdate, 'yyyy-MM-DD HH:mi:ss') from dual;
select to_date('20200101', 'yyyy-mm-dd') from dual;

日期格式:
    yyyy/YYYY：年份
    mm/MM：月份
    dd/DD：日
    hh/HH：12小时制
    hh24/HH24：24小时制度
    mi：分钟
    ss：秒

-- 字符串转数值类型
to_number(s);

select to_number('123456') from dual;
select to_number('abc') from dual; // ORA-01722:invalid number
```

MySQL

```
-- 格式化时间，返回字符串
select DATE_FORMAT(curdate(), '%y%m');

-- 字符串转日期
SELECT STR_TO_DATE('2022-05-26 11:30:00','%Y-%m-%d %h:%i:%s');

日期格式:
    %y：2位年份
    %Y：4位年
    %m：数字月份
    %M：英语月份
    %d：日
    %h：12小时制
    %H：24小时制
    %i：分钟
    %s：秒
```

(五) 集计函数/统计函数/聚合函数

```
-- 求记录数，值统计有值的
count();
一般情况下，count中填写主键

-- 最大/小值
max();
min();

-- 平均值
avg();

-- 和
sum();

以上函数经常用在 group by 之后进行统计结果
```

（六）其他函数

```
-- 判断数据字段如果满足条件1，则显示值1，如果满足条件2，则显示值2，如果条件1、2都不满足，则显示默认值
decode(数据字段, 条件1, 值1 , 条件2, 值2, ..., default)

select t.*, decode(t.ssex, '1', '男', '0', '女', '未知') ssex from tb_student t;

-- case-when
-- 语法结构1:
    case 字段
        when 条件 then
            代码块、值
        when 条件 then
            代码块、值
        else
            代码块、值
    end
-- 语法结构2:
    case
        when 条件表达式 then
            代码块、值
        when 条件表达式 then
            代码块、值
        else
            代码块、值
    end

select
    case ssex
        when '1' then '男'
        when '0' then '女'
        else '未知'
    end ssex
from tb_student;

select
    case
        when ssex = '1' then '男'
        when ssex = '0' then '女'
        else '未知'
    end ssex
from tb_student;

-- 如果数据字段为空，则使用值代替，如果不为空，则正常显示
nvl(数据字段, 值)

select nvl(class, '班级未分配') from tb_student ;
```

七、数据库对象

（一）视图

> 视图是由一个或若干个基表（数据库中的表）产生的数据的集合，它不占用存储空间。
>
> 视图犹如数据表的窗户，管理员定义这些“窗户”的位置后，用户就只能查看他可以看到的数据。视图不是数据表，它仅是一些SQL查询语句的集合，作用是按照不同的要求从数据表中提取不同的数据。

```
视图和表的区别：

相同之处：
        1、视图和表一样由列组成，其查询方式与表完全相同。
        2、和表一样，用户也可以在视图中插入、更新或删除数据，在视图中做这些操作时与在表中是一样的。
            （视图只能进行查看，不能进行增删改操作；单表的视图可以进行增删改操作；多表的视图操作可能会决绝；在不给视图增删改权限的时候只能进行查看。）

区别：
        1、与表不同，视图中没有数据，而仅仅是一条SQL查询语句。按此查询语句检索出的数据以表的形式表示。视图中的列可以在一个或多个基本表中找到。视图不使用物理存储位置来存储数据。
        2、视图的定义（列的安排、授予的权限等等）存储在数据字典中。对一个视图进行查询时，视图将查询其基于的表，并且以视图定义所规定的格式和顺序返回值。
        3、由于视图没有直接相关的物理数据，所以不能像表那样被索引。

视图的优点：
    1、提高查询效率，不是执行的速度快慢，简化了复杂SQL重复编写的问题；
    2、安全，只读，不会修改数据；
    3、不占用存储空间；
```

```
关键字：view
创建语法结构：
    create [or replace] [force | noforce] view 视图名称 [alias]
    as subquery
    [with check option [constraint constraint]]
    [with read only];

    语法解析：
    or replace     ：若所创建的试图已经存在，则替换旧视图；
    force：不管基表是否存在oracle都会自动创建该视图(即使基表不存在，也可以创建该视图，但是该视图不能正常使用，当基表创建成功后，视图才能正常使用)；
    noforce  ：如果基表不存在，无法创建视图，该项是默认选项(只有基表都存在oracle才会创建该视图)。
    alias：为视图产生的列定义的别名；
    subquery  ：一条完整的select语句，可以在该语句中定义别名；
    with check option  ：插入或修改的数据行必须满足视图定义的约束；
    with read only ：默认可以通过视图对基表执行增删改操作，但是有很多在基表上的限制
                        (比如：基表中某列不能为空，但是该列没有出现在视图中，则不能通过视图执行insert操作)，
                        with read only 说明视图是只读视图，不能通过该视图进行增删改操作。现实开发中，基本上不通过视图对表中的数据进行增删改操作。

视图名称的名命：v_名称

删除语法结构：
    drop view 视图名称;
```

```
通过更新视图数据可以修改基表数据。但不是所有的视图都可以更新，只有对满足可更新条件的视图，才能进行更新。
    1、没有使用连接函数、集合运算函数和组函数。
    2、创建视图的 SELECT 语句中没有集合函数且没有 GROUP BY, DISTINCT关键字。
    3、创建视图的 SELECT 语句中不包含从基表列通过计算所得的列。
    4、创建视图没有包含只读属性。
```

（二）索引

> 数据库索引好比是一本书前面的目录，能加快数据库的查询速度。算SQL优化的一种。 主键和唯一键会自动创建索引。同时还可以手动根据需求创建索引。

索引是不是越多越好?

```
索引并不是越多越好，适合的才是最好的。

什么样的字段适合添加索引？
    1、经常出现在select后面的字段；
    2、经常出作为条件的字段；
    3、值重复率高的字段不建议添加索引；
```

```
关键字：index
    create index 索引名称
    on 表名(字段[, 字段]);

什么样的情况下会使用索引?
    1、Oracle只知道有什么（等于），不知道没有什么（不等于）；
    2、模糊查询，% 在前不走索引（以某个字符开头会走索引）；

删除索引：
    drop index 索引名称;
```

（三）序列

> 是一种可被多个用户使用的用于自动产生一系列惟一数字的数据库对象。序列是一个共享的对象。通常被用来产生主键值。

```
关键字：sequence
语法结构：
    create sequence 序列名称
    [increment by n]
    [start with n]
    [{maxvalue n | nomaxvalue}]
    [{minvalue n | nominvalue}]
    [{cycle | nocycle}]
    [{cache n | nocache}];

    increment by：每次增加的值
    start with：序列的开始值
    cycle：当序列达到最大值，从头开始
    cache：缓存，一次返回多个值放到缓存中

序列命名规则：seq_序列名称（表名）
```

```
    一个张表对应一个序列

序列中的关键字/伪列：
    currval：获取当前正在使用的序列的值，如果第一次创建序列但还没使用，会报错。
    nextval：获取下一个序列的值。

删除序列：
    drop sequence 序列名称
```

八、事务

> 事务就是用于保证一个（某一个功能）操作的完整性。双方，要么一起成功，要么一方失败，全部回滚到出错前的状态；

- **事务的四大特性：ACID**

  A：原子性，一个操作不拆分； C：一致性，数据保持一致，此消彼长； I：隔离性，两个事务之间是独立的，互不影响； D：持久性，事务的操作是永久性存在；

- **事务的隔离级别：**

  读未提交：读取到了没有提交的数据； 可能会产生的问题：脏读、不可重复读、幻读 **读已提交：**读取到了已经提交的数据； 可能会产生的问题：不可重复读、幻读 **可重复读：**可以重复的读取数据； 可能会产生的问题：幻读 **序列化读：**一个一个的读取；

  效率低

  幻读：数据的增加或者删除； 不可重复读：数据值的修改；

Oracle的隔离级别是读已提交；

> 事务处理的关键字：

```
commit; -- 提交事务
rollback; -- 回滚事务
```

九、PLSQL（数据库编程语言）

> 对于标准化的SQL语言对数据库进行进行各种操作，每次只能执行一条语句，语句以英文的分号“；”为结束标识。这是因为Oracle数据库系统不像VB，VC这样的程序设计语言，侧重在于后台数据库的管理，因此提供的编程能力较弱，而结构化编程语言对数据库的支持能力又较弱。
>
> 在这种要求的驱使下，Oracle公司在标准SQL语言的基础上发展了自己的PL/SQL语言，将变量、控制结构、过程和函数等结构化程序设计的要素引入了SQL语言中，这样就能够编制比较复杂的SQL程序了，利用PL/SQL语言编写的程序也称为PL/SQL程序块

（一）匿名代码块

> 没有名字的一块逻辑处理，不能被其他代码调用，是一次性的。

```
语法结构：
    declare
        变量常量的定义
    begin
        逻辑代码
    [exception]
    end;

输出语句：
    dbms_output.put_line(args);
赋值语句：
    := -> 直接给变量赋值
    into -> 将查询到的值赋给变量
```

```
-- 匿名代码块
declare
   -- 变量、常量名称 数据类型;
   name1 varchar2(100);
   name2 varchar2(100);
begin
   dbms_output.put_line('hello plsql');
   -- 赋值语句:=
   name1 := 'Tom';
   dbms_output.put_line(name1);

   select sname into name2 from tb_student where sno = '110';
   dbms_output.put_line(name2);
end;
```

（二）变量和常量

```
变量语法结构：
    变量标识符 数据类型[:= 值];

常量语法结构：
    常量标识符 constant 数据类型 [not null] := 值;

常量和变量定义的要求：
    1、常量名和变量名都必须以字母开头，在字母后面可以带数字或特殊字符
    2、不能有空格
    3、不能超过30个字符长度
    4、不能和保留字同名
    5、常（变）量名称不分大小写
    6、括号内的 not null 为可选参数，若选用，表明该常（变）量不能为空值。
```

（三）数据类型

标准类型

```
字符类型：char、varchar、varchar2
数值类型：number、integer
日期类型：date
```

%type

可以引用表中的某个字段的类型，作为变量、常量的数据类型；相对于直接使用标准数据类型而言，引用字段类型的方式比较智能；

```
语法结构：
    标识符 表名.字段名%type;

    name tb_student.sname%type; -- name引用的是tb_student表的sname字段的类型；
```

```
-- 匿名代码块
declare
   -- %type
   name3 tb_student.sname%type;
begin
   name3 := 'Jack';
   insert into tb_student values(seq_student.nextval, name3, '男', sysdate,
'95033');
end;
```

record（记录类型）

用于存储一条记录的数据，数据来源可以是表中，可以自定义；

```
语法结构：
    type 记录标识符 is record(
        字段标识符 数据类型,
        ...
    );
```

```
-- 匿名代码块
declare
   -- recode
   type myRecode is record(
        name varchar2(100),
        height number,
        weight number
   );
   r1 myRecode;
begin
   r1.name := 'tom';
   r1.height := 180;
   r1.weight := 140;

   dbms_output.put_line('姓名：' || r1.name || '；身高：' || r1.height || '；体
重：' || r1.weight);
```

```
end;
```

```
-- 匿名代码块
declare
   -- recode
   type myRecode is record(
        sno tb_student.sno%type,
        sname tb_student.sname%type,
        ssex tb_student.ssex%type,
        sbirthday tb_student.sbirthday%type,
        class tb_student.class%type
   );
   r1 myRecode;
begin
   select sno into r1.sno from tb_student where sno = '110';
   select sname into r1.sname from tb_student where sno = '110';
   select class into r1.class from tb_student where sno = '110';

   dbms_output.put_line('学号：' || r1.sno || '；姓名：' || r1.sname || '；班级：' || r1.class);
end;
```

%rowtype

用于存储一条记录的数据，引用表的数据类型；

```
语法结构：
    标识符 表名%rowtype;
```

```
declare
   stu tb_student%rowtype;
begin
   stu.sno := '111';
   stu.sname := 'Lily';

   dbms_output.put_line(stu.sno);
   dbms_output.put_line(stu.sname);
  end;
```

varray（数组类型）

用于存储多个数据

```
语法结构：
    type 数组标识符 is varray(length) of 类型;

    类型：标准类型/记录类型/%type/%rowtype
    length：定义数组的容量最大值；
    数组标识符.count：获取数组已经扩容的长度；

使用步骤：
    1、创建数组类型（可以理解Java中创建了一个类）
    2、初始化数组（依据类创建一个对象）
    3、扩容，extend(n);
    4、使用
```

```
declare
   -- 定义类型
   type myArray is varray(100) of integer;
   -- 根据类型创建数组对象
   array1 myArray := myArray();
begin
   -- 扩容
   array1.extend(5);

   array1(1) := 1;
   array1(2) := 2;
   array1(3) := 3;
   array1(4) := 4;
   -- array1(5) := 5;
   -- array1(6) := 6;

   dbms_output.put_line(array1(1));
   dbms_output.put_line(array1(2));
   dbms_output.put_line(array1(3));
   dbms_output.put_line(array1(4));
   dbms_output.put_line(array1(5));
  end;
```

```
declare
   -- 定义类型
   type myArray is varray(100) of tb_student%rowtype;
   -- 根据类型创建数组对象
   array1 myArray := myArray();
begin
   -- 扩容
   array1.extend(5);

   select * into array1(1) from tb_student where sno = '110';
   select * into array1(2) from tb_student where sno = '101';
   select * into array1(3) from tb_student where sno = '103';

   dbms_output.put_line(array1(1).sno);
   dbms_output.put_line(array1(1).sname);
   dbms_output.put_line(array1(1).ssex);
   dbms_output.put_line(array1(1).sbirthday);
   dbms_output.put_line(array1(1).class);
```

```
  end;
```

集合类型

```
语法结构：
    type 集合的标识符 is table of 类型 [index by binary_integer];

    类型：标准类型/记录类型/%type/%rowtype
    index by binary_integer：自动扩容

使用步骤：
    1、创建集合类型
    2、初始化集合：如果有index by binary_integer，不需要初始化；
    3、扩容：如果有index by binary_integer，不需要扩容；
    4、使用
```

```
declare
  -- 创建类型
  type myList is table of integer;
  -- 初始化集合容器
  list1 myList:=myList();
begin
  -- 扩容
  list1.extend(10);

  -- 使用
  list1(1) := 100;
  list1(2) := 100;
  list1(3) := 100;
  list1(4) := 100;
  list1(5) := 100;
  list1(6) := 100;
  list1(7) := 100;
  list1(8) := 100;
  list1(9) := 100;
  list1(10) := 100;

  dbms_output.put_line(list1(11));

  end;
```

```
declare
  -- 创建类型
  type myList is table of integer index by binary_integer;
  -- 初始化集合容器
  list1 myList;-- :=myList();
begin
  -- 扩容
  -- list1.extend(10);

  -- 使用
  list1(1) := 100;
  list1(2) := 100;
```

```
  list1(3) := 100;
  list1(4) := 100;
  list1(5) := 100;
  list1(6) := 100;
  list1(7) := 100;
  list1(8) := 100;
  list1(9) := 100;
  list1(10) := 100;

  -- 获取集合中内容的个数
  dbms_output.put_line(list1.count);
  dbms_output.put_line(list1(10));

  end;
```

（四）表达式

数值表达式

```
+、-、*、/
```

字符表达式

```
||
```

关系表达式

```
=等于（不是赋值运算符:=)
like类似于
in在. . . 之中
<=小于等于
>=大于等于
<>不等于
Between .. and 在. . . 之间
```

逻辑表达式

```
NOT：逻辑非
OR：逻辑或
AND：逻辑与
```

（五）流程控制

1、判断

```
语法结构：
    if 条件表达式 then
        逻辑代码
    elsif 条件表达式 then
        逻辑代码
    else
        逻辑代码
    end if;
```

```
-- 需求：判定一个年龄是否成年
declare
   age integer := 18;
begin

   age := 17;

   if age < 0 then
     dbms_output.put_line('抱歉，年龄有误！');
   elsif age >= 18 then
     dbms_output.put_line('恭喜，已经成年！');
   else
     dbms_output.put_line('抱歉，未成年！');
   end if;
end;
```

```
-- case-when
语法结构1：
    case 值
        when 条件 then
            代码块、值
        when 条件 then
            代码块、值
        else
            代码块、值
    end
语法结构2：
    case
        when 条件表达式 then
            代码块、值
        when 条件表达式 then
            代码块、值
        else
            代码块、值
    end
```

```
select case ssex
            when '1' then '男'
            when '0' then '女'
            else '未知'
         end ssex
        from tb_student;

select case
            when ssex = '1' then '男'
            when ssex = '0' then '女'
            else '未知'
         end ssex
        from tb_student;
```

2、循环

loop..exit..end loop

```
语法结构：
    loop
        逻辑代码
        exit;
    end loop;
```

```
declare
  num integer := 0;
begin
  loop
    if num > 10 then
      exit;
    end if;
    dbms_output.put_line(num);
    num := num + 1;
  end loop;
end;
```

loop..exit when..end loop

```
语法结构：
    loop
        逻辑代码
    exit when 条件
    end loop;
```

```
declare
  num integer := 0;
begin
  loop
    dbms_output.put_line(num);
    num := num + 1;
    exit when num > 10;
  end loop;
end;
```

while..loop

```
语法结构：
    while 条件 loop
    exit when 条件
    end loop;
```

```
declare
  num integer := 0;
begin
  while num <= 10 loop
    dbms_output.put_line(num);
    /*if num = 8 then
       exit;
    end if;*/
    exit when num = 8;
    num := num + 1;
  end loop;
end;
```

for

```
语法结构：
    for 变量 in 开始条件 .. 结束条件 loop
    exit when 条件
    end loop;

    for后的变量可以省略；
```

```
declare
begin
  for i in 1 .. 100 loop
    if mod(i, 2) = 0 then
       dbms_output.put_line(i || '是偶数');
    else
       dbms_output.put_line(i || '是奇数');
    end if;
  end loop;
end;
```

（五）游标

> 游标是从数据表中提取出来的数据，以临时表的形式存放在内存中，在游标中有一个数据指针，在初始状态下指向的是首记录，利用fetch语句可以移动该指针，从而对游标中的数据进行各种操作，然后将操作结果写回数据表中。

```
语法结构：
    cursor 游标名称 is 查询语句;

游标属性：
    %isopen：测试游标是否打开，如果没有打开游标，用fetch语句并提示错误。
    %found：逻辑值，游标是否找到一条记录。如游标找到记录其值为true，反之为false.
    %notfound：逻辑值，游标没有找到记录，是%found属性的逻辑非。
    %rowcount：返回提取游标记录的行数。

使用步骤：
    1、定义游标
    2、打开游标：open 游标名;
    3、提取数据：
        fetch 游标名 into 变量1, 变量2, ...
        fetch 游标名 into 记录类型
    4、关闭游标

遍历游标：
    1、普通loop、while
    2、for
        for 变量 in 游标名称 loop
            变量.字段;
        end loop;

        通过for遍历游标时，不需要开启和关闭游标;
```

```
declare
  -- 1、创建游标
  cursor myCursor is select sno, sname, ssex, sbirthday, class from tb_student;

  no tb_student.sno%type;
  name tb_student.sname%type;
  sex tb_student.ssex%type;
  birthday tb_student.sbirthday%type;
  class tb_student.class%type;

  stu tb_student%rowtype;
begin
  -- 2、打开游标
  open myCursor;
  -- 3、提取数据
  fetch myCursor into no, name, sex, birthday, class;
  dbms_output.put_line(no || ', ' || name || ', ' || sex || ', ' || birthday || ', '|| class);
  fetch myCursor into no, name, sex, birthday, class;
  dbms_output.put_line(no || ', ' || name || ', ' || sex || ', ' || birthday || ', '|| class);

  dbms_output.put_line('-------------------------');
```

```
22
23    fetch myCursor into stu;
24    dbms_output.put_line(stu.sno || ', ' || stu.sname || ', ' || stu.ssex ||
', ' || stu.sbirthday  || ', '|| stu.class);
25
26    -- 4、关闭游标
27    close myCursor;
28  end;
```

```
declare
  -- 1、创建游标
  cursor myCursor is select sno, sname, ssex, sbirthday, class from
tb_student;
begin
    for stu in myCursor loop
      dbms_output.put_line(stu.sno || ', ' || stu.sname || ', ' || stu.ssex
|| ', ' || stu.sbirthday  || ', '|| stu.class);
    end loop;
end;
```

（六）存储过程

```
语法结构：
    create [or replace] procedure 存储过程名称(参数标识符 in / out 参数类型, ...)
    as/is
        变量、常量定义
    begin
        逻辑代码
    end;

    in：输入参数
    out：输出参数

如果没有输入和输出参数，在定义存储过程时，可以省略括号。但是在调用存储过程时，必须添加括号；

存储过程特点：
    1、没有返回值
    2、有输出参数
```

```
-- 需求：判定一个值是奇数还是偶数
create or replace procedure pro_num(num in integer, result out varchar2)
as
begin
  if mod(num, 2) = 0 then
    dbms_output.put_line('偶数');
    result := '偶数';
  else
    dbms_output.put_line('奇数');
    result := '奇数';
  end if;
end;
```

```

```

```sql
create table account(
    no integer,
    money number
);

insert into account values(1000, 10000);
insert into account values(1001, 10000);

-- 模拟银行在转账
create or replace procedure pro_transfer(i_no_from in account.no%type,
                                         i_money in account.money%type,
                                         i_no_to in account.no%type,
                                         i_flg in integer,
                                         o_msg out varchar2)
as
begin
  -- 当前账户余额减少
  update account set money = money - i_money where no = i_no_from;
  -- 对方账户余额增加
  update account set money = money + i_money where no = i_no_to;

  -- 模拟异常处理
  if i_flg = 1 then
    o_msg := '转账失败! ';
    rollback;
  else
    o_msg := '转账成功! ';
    commit;
  end if;
end;

select * from account;

declare
  msg varchar2(100);
begin
  pro_transfer(1001, 200, 1000, 1, msg);
  dbms_output.put_line(msg);
end;
```

(七) 函数

```
语法结构：
    create [or replace] function 函数名称(标识符 in 参数类型)
    return 返回值类型
    as/is
        常量、变量的定义
    begin
        逻辑代码
        return 值;
    end;

函数特征：
    1、没有输出参数
    2、有返回值
```

```
-- 自定义类似Java的substring()函数
-- substr(a, b) a:操作的字符；b：开始位置；
-- substr(a, b, c) a:操作的字符；b：开始位置；c：截取的长度
-- abcdefg 从第1位，截取到第4位；此时长度是4，4 = 结束位置-开始位置 + 1
-- 1234
create or replace function subString(str in varchar2, startIndex in integer,
endIndex in integer)
return varchar2
as
  -- 截取的长度
  str_length integer;
begin

  if length(str) <= 1 or endIndex <= startIndex or startIndex < 0 or
endIndex < 0 then
    return 'error';
  end if;

  str_length := endIndex - startIndex + 1;

  return substr(str, startIndex, str_length);

end;

select subString('', -1, 1) from dual;

-- 自定义函数startWith()
create or replace function startWith(str in varchar2, subStr in varchar2)
return integer
as
begin
  if length(str) <= 0 or str = '' or str is null
    or length(subStr) <= 0 or subStr = '' or subStr is null
    or instr(str, subStr) <= 0 then
    return -1;
  else
    return 1;
  end if;
end;
```

（八）异常

> 在设计PL/SQL程序时，经常会发生这样或那样的错误，异常处理就是针对错误进行处理的程序段，Oracle中的异常处理分为系统定义异常处理和自定义异常处理两部分。

```
语法结构：
    -- 创建一个自定义异常
    异常名称 exception;
    -- 触发异常
    raise 异常名称;
    -- 处理异常
    exception
        when 异常名称 then
            处理异常的代码;
        when 异常名称 then
            处理异常的代码;

向用户反馈错误信息：
    raise_application_error('错误代码', '错误信息');
    错误代码范围：-20000到-20999
```

```
create or replace function startWith(str in varchar2, subStr in varchar2)
return integer as
  -- 自定义异常
  StrEmptyException exception;
begin
  if length(str) <= 0 or str = '' or str is null
    or length(subStr) <= 0 or subStr = '' or subStr is null
    or instr(str, subStr) <= 0 then
    -- 触发异常
    raise StrEmptyException;
    -- return -1;
  else
    return 1;
  end if;
exception
  -- 处理异常
  when StrEmptyException then
    dbms_output.put_line('StrEmptyException');
    raise_application_error('-20000', '参数错误');
    return -1;
end;
```

（九）触发器

> 当执行某个操作（添加、删除、修改数据）时，触发的额外的一个业务处理；

```
语法结构：
    create [or replace] trigger 触发器名称 [before | after] [insert] [update] [delete]
    on 表名 [for each row ] [when 条件]
```

```
    declare
         << 定义变量 >>
    begin
        << 执行过程代码语句 >>
    exception
         << 异常处理代码语句 >>
    end;
解释：
    create：创建
    replace：修改
    trigger：触发器关键字
    before|after：前置触发|后置触发，二者选择其一
    [insert][update][delete]：触发的动作
    on：针对某一张表/视图 的DML语句进行触发
    for each row：行级触发，包含两个关键字[:new][:old]
        [:new] 添加数据时表示添加的数据，可以通过:new.字段获取值
        [:old] 删除/修改数据时表示修改/删除的数据，可以通过:old.字段获取值

    when：触发条件
```

```
-- 触发器完成修改密码数据备份
create or replace trigger bakData before update
on tb_student for each row
begin
  insert into tb_student_bak(sno, sname, ssex, sbirthday, class, spass)
  values(:old.sno, :old.sname, :old.ssex, :old.sbirthday, :old.class,
:old.spass);
  dbms_output.put_line('触发器执行完毕');
end;
```

十、MySQL 对比

端口：

> 真实项目中为了保证服务的安全问题，都会修改默认的端口号。

- Oracle：默认 1521
- MySQL：默认 3306

语法结构：

- 由于都是关系型数据库，采用标准SQL语法，大部分语法结构相同。
- MySQL 没有虚拟表。
- Oracle 有虚拟表 dual，保证 SQL 的语法完整。
- MySQL 的批量添加和 Oracle 不通用。Oracle 的表拷贝在 MySQL 通用。

数据类型：

- 由于都是关系型数据库，采用标准SQL语法，大部分数据类型相同。
- Oracle 有独有的 varchar2 类型。

主键策略：

- Oracle 采用数据库对象序列方式。

- MySQL 数据库内置自增特点，需要配置主键为 int 类型，且配置自增。

分页：

- Oracle 采用 rownum 关键字，使用嵌套方式分页；

```sql
-- 不排序
select *
  from (select rownum num, t.* from tb_student t)
 where num >= startNum
   and num <= endNum;

-- 排序
select *
    from (select rownum num, t.*
          from (select * from tb_student order by 字段, ...) t
         )
 where num >= startNum
   and num <= endNum;
```

- MySQL 使用 limit 关键字分页。

```sql
-- limit ? 获取前 n 条
select 字段 from 表名 limit ?;
-- limit ?, ? 第一个参数：数据的下标位置；第二个参数：每页显示的数据量
select 字段 from 表名 limit ?, ?;
```

函数：

- 参照《六、函数》

事务隔离级别：

- Oracle 读已提交
- MySQL 可重复读

其他：

- Oracle 收费，MySQL 社区版开源免费。
- Oracle 重型数据库，MySQL 轻量型数据库。
- MySQL 后被 Oracle 甲骨文收购。