Oracle笔记

目录

第一章、Oracle基础

Oracle简介

Oracle数据库是Oracle（甲骨文）公司的核心产品，适合于大型项目的开发；银行、电信、电商、金融等各领域都大量使用Oracle数据库。

Oracle数据库是一种对象关系型数据库，在关系型数据库的基础上，引入了一些面向对象的特性。

Oracle基础概念（实例、表空间、用户、表之间关系）

数据库

数据库是数据集合。Oracle是一种数据库管理系统，是一种关系型的数据库管理系统。

实例

一个Oracle实例（Oracle Instance）有一系列的后台进程和内存结构组成。一个数据库可以有n个实例。

用户

用户是在实例下建立的。不同实例可以建相同名字的用户。

Oracle数据库建好后，要想在数据库里建表，必须先为数据库建立用户，并为用户指定表空间。

表空间

表空间是一个用来管理数据存储逻辑概念，表空间只是和数据文件（ORA或者DBF文件）发生关系，数据文件是物理的，一个表空间可以包含多个数据文件，而一个数据文件只能隶属一个表空间。

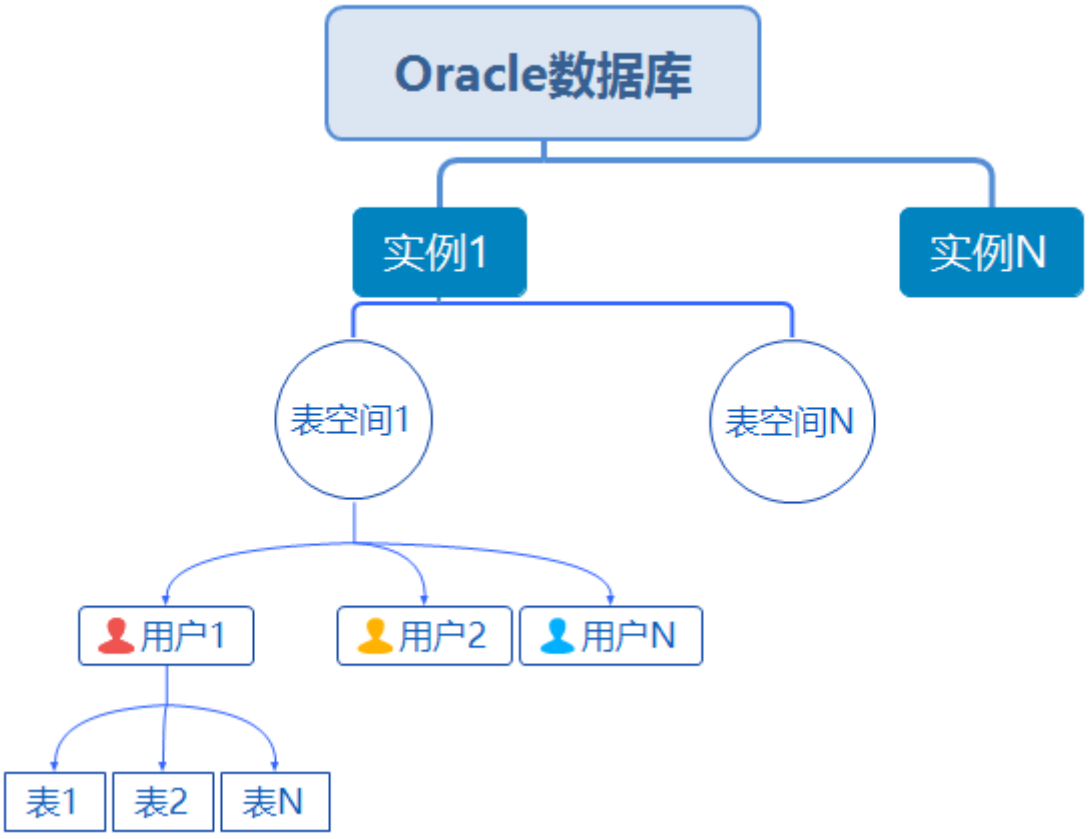
数据文件（dbf、ora）

数据文件是数据库的物理存储单位。数据库的数据是存储在表空间中的，真正是在某一个或者多个数据文件中。而一个表空间可以由一个或多个数据文件组成，一个数据文件只能属于一个表空间。一旦数据文件被加入到某个表空间后，就不能删除这个文件，如果要删除某个数据文件，只能删除其所属于的表空间才行。

> **理解：**表的数据，是有用户放入某一个表空间的，而这个表空间会随机把这些表数据放到一个或者多个数据文件中。
> 由于oracle的数据库不是普通的概念，oracle是由用户和表空间对数据进行管理和存放的。但是表不是由表空间去查询的，而是由用户去查的。因为不同用户可以在同一个表空间建立同一个名字的表！这里区分就是用户了！

Oracle关系图

Oracle数据库可以创建多个实例，每个实例可以创建多个表空间，每个表空间下可以创建多个用户和数据库文件，用户可以创建多个表。



> **解释：**一个表空间（数据库）包含一个或多个数据文件，数据文件通常为*.dbf格式，一个数据库的数据文件包含全部数据库数据（如表、索引等）。
> 一个用户可以使用一个或多个表空间，一个表空间也可以供多个用户使用。用户和表空间没有隶属关系，表空间是一个用来管理数据存储的逻辑概念，表空间只是和数据文件发生关系，数据文件是物理的，一个表空间可以包含多个数据文件，而一个数据文件只能隶属一个表空间。

> **总结：**解释数据库、表空间、数据文件、表、数据的最好办法就是想象一个装满东西的柜子。数据库其实就是柜子，柜中的抽屉是表空间，抽屉中的文件夹是数据文件，文件夹中的纸是表，写在纸上的信息就是数据。

SQLPlus常用语句

显示当前用户名

```
SQL>show user;
```

查看当前用户的角色

```
SQL>select * from user_role_privs;
```

查看当前用户的系统权限和表级权限

```
SQL>select * from user_sys_privs;
SQL>select * from user_tab_privs;
```

查看用户所有表

```
SQL>select * from all_tab_comments -- 查询所有用户的表,视图等。
SQL>select * from user_tab_comments -- 查询本用户的表,视图等。
SQL>select * from all_col_comments  --查询所有用户的表的列名和注释。
SQL>select * from user_col_comments -- 查询本用户的表的列名和注释。
SQL>select * from all_tab_columns --查询所有用户的表的列名等信息。
SQL>select * from user_tab_columns --查询本用户的表的列名等信息。
```

关闭数据库

```
SQL>shutdown immediate;
```

启动数据库

```
SQL>startup open;
```

创建表空间（数据库）

```
SQL>create tablespace //表空间名称
datafile 'E:\路径\文件名.dbf' //表空间数据文件路径
size 100M //表空间初始大小
```

删除表空间

```
SQL>drop tablespace 数据库名;
```

修改表空间：添加文件

```
SQL>alter tablespace 数据库;
add datafile'E:/Oracle/文件名.dbf' //添加文件
size 10M; 文件大小
```

修改表空间：删除文件

```
SQL>alter tablespace //数据库
```

```
drop datafile'E:/Oracle/文件名.dbf'; //删除文件
```

显示当前连接用户

```
SQL>show user
```

查看系统拥有哪些用户

```
SQL>select * from all_users;
```

连接到新用户

```
SQL>conn
```

查看oracle的版本信息

```
SQL>select * from v$version;
```

查询当前用户下所有对象

```
SQL>select * from tab;
```

查询表结构

```
SQL>desc 表名;
```

回滚

```
SQL>roll;
SQL>rollback;
```

提交

```
SQL>commit;
```

退出

```
SQL>exit;
SQL>quit;
```

设置显示效果

```
SQL>set linesize 300;
SQL>set pagesize 300;
SQL>col 列名 for a列宽;
```

Oracle数据类型

数据类型	类型解释
VARCHAR2(length)	字符串类型：存储可变的长度的字符串，length:是字符串的最大长度，默认不填的时候是1，最大长度不超过4000。
CHAR(length)	字符串类型：存储固定长度的字符串，length:字符串的固定长度大小，默认是1，最大长度不超过2000。
NUMBER(a,b)	数值类型：存储数值类型，可以存整数，也可以存浮点型。a代表数值的最大位数：包含小数位和小数点，b代表小数的位数。例子： number(6,2)，输入123.12345，实际存入：123.12 。 number(4,2)，输入12312.345，实际春如：提示不能存入，超过存储的指定的精度。
DATA	时间类型：存储的是日期和时间，包括年、月、日、时、分、秒。例子： 内置函数sysdate获取的就是DATA类型
TIMESTAMP	时间类型：存储的不仅是日期和时间，还包含了时区。例子： 内置函数systimestamp获取的就是timestamp类型
CLOB	大字段类型：存储的是大的文本，比如：非结构化的txt文本，字段大于4000长度的字符串。
BLOB	二进制类型：存储的是二进制对象，比如图片、视频、声音等转换过来的二进制对象

Oracle创建用户、角色、授权、建表

oracle用户的概念对于Oracle数据库至关重要，在现实环境当中一个服务器一般只会安装一个Oracle实例，一个Oracle用户代表着一个用户群，他们通过该用户登录数据库，进行数据库对象的创建、查询等开发。

每一个用户对应着该用户下的N多对象，因此，在实际项目开发过程中，不同的项目组使用不同的Oracle用户进行开发，不相互干扰。也可以理解为一个Oracle用户既是一个业务模块，这些用户群构成一个完整的业务系统，不同模块间的关联可以通过Oracle用户的权限来控制，来获取其它业务模块的数据和操作其它业务模块的某些对象。

用户权限

Oracle数据库用户权限分为：系统权限和对象权限两种。

系统权限：比如：create session可以和数据库进行连接权限、create table、create view 等具有创建数据库对象权限。

对象权限：比如：对表中数据进行增删改查操作，拥有数据库对象权限的用户可以对所拥有的对象进行相应的操作。

一、创建新用户

```
create user student--用户名
  identified by "123456"--密码
  default tablespace USERS--表空间名
  temporary tablespace temp --临时表空间名
  profile DEFAULT --数据文件（默认数据文件）
  account unlock; -- 账户是否解锁（lock:锁定、unlock解锁）
```

```
更改用户：
 alter user STUDENT
  identified by 123456  --修改密码
  account lock;--修改用户处于锁定状态或者解锁状态 （LOCK|UNLOCK ）
```

二、删除用户

```
语法：
 drop user 用户名;
例子：
 drop user test;
若用户拥有对象，则不能直接删除，否则将返回一个错误值。

指定关键字cascade,可删除用户所有的对象，然后再删除用户。
语法：
 drop user 用户名 cascade;
例子：
 drop user test cascade;
```

三、授权角色

角色是一组权限的集合，将角色赋给一个用户，这个用户就拥有了这个角色中的所有权限。

三种标准角色：

1.connect(连接角色)

> connect角色是Oracle用户的基本角色，connect权限代表着用户可以和Oracle服务器进行连接，建立session（会话）。

2.resource(资源角色)

> resouce角色是开发过程中常用的角色。 RESOURCE给用户提供了可以创建自己的对象，包括：表、视图、序列、过程、触发器、索引、包、类型等。

3.dba(数据库管理员角色)

DBA角色是管理数据库管理员该有的角色。它拥护系统了所有权限，和给其他用户授权的权限。SYSTEM用户就具有DBA权限。

提示：
系统权限只能通过DBA用户授权，对象权限有拥有该对象权限的对象授权（不一定是本身对象）！用户不能自己给自己授权！

```
授权命令：
--GRANT 对象权限 on 对象 TO 用户
grant select, insert, update, delete on JSQUSER to STUDENT;
--GRANT 系统权限 to 用户
grant select any table to STUDENT;
--GRANT 角色 TO 用户
grant connect to STUDENT;--授权connect角色
grant resource to STUDENT;--授予resource角色

撤销权限：
-- Revoke 对象权限 on 对象 from 用户
revoke select, insert, update, delete on JSQUSER from STUDENT;
-- Revoke 系统权限  from 用户
revoke SELECT ANY TABLE from STUDENT;
-- Revoke 角色（role） from 用户
revoke RESOURCE from STUDENT;
```

四、创建/授权/删除角色

```
创建角色：
语法：
 create role 角色名;
例子：
 create role testRole;
授权角色：
语法：
 grant select on class to 角色名;
列子：
  grant select on class to testRole;
注：现在，拥有testRole角色的所有用户都具有对class表的select查询权限
删除角色：
语法：
```

```
 drop role 角色名;
例子:
 drop role testRole;
注：与testRole角色相关的权限将从数据库全部删除
```

Oracle约束

在Oracle中，数据完整性可以使用约束、触发器、应用程序（过程、函数）三种方法来实现，在这三种方法中，因为约束易于维护，并且具有最好的性能，所以作为维护数据完整性的首选。

添加约束语法：

```
ALTER TABLE 表名 ADD CONSTRAINT 约束名 约束类型 约束描述
```

约束命名规范:

非空约束	NN_表名_列名
唯一约束	UK_表名_列名
主键约束	PK_表名_列名
外键约束	FK_表名_列名
检查约束	CK_表名_列名
默认约束	DF_表名_列名

主键约束 （ Primary key, 简称 PK)

该约束的定义为：不能重复，不能为null。

```
SQL> alter table 表名 add constraint 约束名 primary key(字段名);
```

非空约束(not null , 简称 NN)

约束该列不能为空值。

```
SQL> alter table 表名 modify 字段名 not null;
```

唯一约束(Unique , 简称 UK)

约束该列数据不能重复，不能相同。

```
SQL> alter table 表名 add constraint 约束名 unique(字段名);
```

检查约束(Check , 简称 CK)

检查自定义条件是否为真，为真就可以插入，更新。

```
SQL> alter table 表名 add constraint 约束名 check(字段名 in('约束的值','约束的值')));
例子:
SQL> alter table emp add constraint CK_stuSex check(sex in('男','女'));
```

```
SQL>alter table 表名 add constraint 约束名 check(字段 between 0 and 100);
--出生日期在1980年1月1日之后
SQL>ALTER TABLE student ADD CONSTRAINT CK_student_borndate CHECK (borndate > TO_date('1980-01-01','yyyy-MM-dd') );
```

外键约束(Foreign key, 简称 FK)

外键约束定义在具有父子关系的子表中，外键约束使得子表中的列对应父表的主键列，用以维护数据库的完整性。

> 外键约束注意以下几点：
> 1、 外键约束的子表中的列和对应父表中的列数据类型必须相同，列名可以不同
> 2、 对应的父表列必须存在主键约束（PRIMARY KEY）或唯一约束（UNIQUE）
> 3、 外键约束列允许NULL值，对应的行就成了孤行了
> 4、 外键约束的表中的列数据必须包含在父表主键字段的数据内，否则会报错：ORA-02298：无法验证- 未找到父项关键字

其实很多时候不使用外键，很多人认为会让删除操作比较麻烦，比如要删除父表中的某条数据，但某个子表中又有对该条数据的引用，这时就会导致删除失败。我们有两种方式来优化这种场景：

第一种方式：简单粗暴，删除的时候，级联删除掉子表中的所有匹配行，在创建外键时，通过 **on delete cascade** 子句指定该外键列可级联删除：

```
SQL> alter table 表名 add constraint 约束名 foreign key(字段名) references 父表名 (父表字段) on delete cascade;
```

第二种方式：删除父表中的对应行，会将对应子表中的所有匹配行的外键约束列置为NULL，通过 **on delete set null** 子句实施：

```
SQL> alter table 表名 add constraint 约束名 foreign key(字段名) references 父表名 (父表字段) on delete set null;
```

默认约束(Default Key,简称 DF)

约束用于向列中插入默认值。

```
SQL> alter table 表名 modify （字段 类型 default 默认值）
例子：
 alter table Student Modify Address varchar(50) default '地址不详';
 alter table Student Modify JoinDate Date default sysdate;
```

Oracle注释

添加表级注释

```
SQL> COMMENT ON TABLE 表名 IS '注释内容';
```

添加列级注释

```
SQL> COMMENT ON COLUMN 表名.字段名 IS '注释内容';
```

查看表级注释

```
SQL> SELECT * FROM USER_TAB_COMMENTS WHERE TABLE_NAME='表名';
```

查看列级注释

```
SQL> SELECT * FROM USER_COL_COMMENTS WHERE COLUMN_NAME='字段名' A
ND COMMENTS IS NOT NULL;
```

删除注释（即，添加空注释）

```
SQL> COMMENT ON TABLE 表名 IS '';
SQL> COMMENT ON COLUMN 表名.字段名 IS '';
```

Oracle序列

序列(SEQUENCE)是序列号生成器，可以为表中的行自动生成序列号，产生一组等间隔的数值(类型为数字)。不占用磁盘空间，占用内存。

其主要用途是生成表的主键值，可以在插入语句中引用，也可以通过查询检查当前值，或使序列增至下一个值。

创建序列

```
创建序列需要CREATE SEQUENCE系统权限。
序列的创建语法如下：
    CREATE SEQUENCE 序列名
    [INCREMENT BY n]
    [START WITH n]
    [{MAXVALUE/ MINVALUE n| NOMAXVALUE}]
    [{CYCLE|NOCYCLE}]
    [{CACHE n| NOCACHE}];

其中：
1)  INCREMENT BY用于定义序列的步长，如果省略，则默认为1，如果出现负值，则代表Oracle序列的值是按照此步长递减的。
2)  START WITH 定义序列的初始值(即产生的第一个值)，默认为1。
3)  MAXVALUE 定义序列生成器能产生的最大值，选项NOMAXVALUE是默认选项，代表没有最大值定义。
4)  MINVALUE定义序列生成器能产生的最小值。选项NOMAXVALUE是默认选项，代表没有最小值定义。
```

```
5)  CYCLE和NOCYCLE 表示当序列生成器的值达到限制值后是否循环。CYCLE代表循环，NOCYCLE代表不循环。如果循环，则当递增序列达到最大值时，循环到最小值;对于递减序列达到最小值时，循环到最大值。
如果不循环，达到限制值后，继续产生新值就会发生错误。
6)  CACHE(缓冲)定义存放序列的内存块的大小，默认为20。NOCACHE表示不对序列进行内存缓冲。对序列进行内存缓冲，可以改善序列的性能。
8)  CURRVAL 中存放序列的当前值,NEXTVAL 应在 CURRVAL 之前指定 ，二者应同时有效。

例子：
SQL> create sequence t1_seq increment by 1 start with 1;
```

使用序列

调用NEXTVAL将生成序列中的下一个序列号，调用时要指出序列名，即用以下方式调用： 序列名.NEXTVAL

CURRVAL用于产生序列的当前值，无论调用多少次都不会产生序列的下一个值。如果序列还没有通过调用NEXTVAL产生过序列的下一个值，先引用CURRVAL没有意义。调用CURRVAL的方法同上，要指出序列名，即用以下方式调用:序列名.CURRVAL

```
SQL> create table t1(id number,qq number,ww number);
SQL> insert into t1 values(t1_seq.nextval,1,1);
```

查看序列

```
SQL> select * from t1;
        ID         QQ         WW
---------- ---------- ----------
         1          1          1
         2          1          1
         3          1          1
         4          1          1
         5          1          1

SQL> select t1_seq.currval from dual;
   CURRVAL
----------
         5
```

```
SQL> select t1_seq.nextval from dual;
   NEXTVAL
----------
         6

SQL> select t1_seq.nextval from dual;
   NEXTVAL
----------
         7
```

第二章、Oracle应用

数据操纵语言 (DML)

DML主要有三种形式：插入(INSERT)、更新(UPDATE)、(删除)DELETE。

插入数据

```
INSERT INTO 表名(字段名1, 字段名2, ……) VALUES ( 值1, 值2, ……);
INSERT INTO 表名(字段名1, 字段名2, ……) SELECT (字段名1, 字段名2, ……) FROM 另外的表名;
```

1.字符串类型的字段值必须用单引号括起来, 例如: 'GOOD DAY'。
2.如果字段值里包含单引号' 需要进行字符串转换, 我们把它替换成两个单引号".
日期字段的字段值可以用当前数据库的系统时间SYSDATE, 精确到秒
或者用字符串转换成日期型函数TO_DATE('2001-08-01','YYYY-MM-DD')。
3.添加数据时如用到从1开始自动增长的序列号, 应该先建立一个序列号
语法:
CREATE SEQUENCE 序列号的名称 (最好是表名+序列号标记) INCREMENT BY 1 START WITH 1 MAXVALUE 99999 CYCLE NOCACHE;
其中最大的值按字段的长度来定, 如果定义的自动增长的序列号 NUMBER(6) , 最大值为999999
INSERT 语句插入这个字段值为: 序列号的名称.NEXTVAL

更新数据

```
UPDATE 表名 SET 字段名1=值1, 字段名2=值2, …… WHERE 条件;
```

删除数据

```
DELETE FROM 表名 WHERE 条件;
```

注意：删除记录并不能释放ORACLE里被占用的数据块表空间，它只把那些已被删除的数据块标成unused，如果确实要删除一个大表里的全部记录, 可以用 TRUNCATE 命

> 令, 它可以释放占用的数据块表空间
> TRUNCATE TABLE 表名;
> 此操作不可回退.

注意事项:

以上SQL语句对表都加上了行级锁，确认完成后, 必须加上事物处理结束的命令 COMMIT 才能正式生效,否则改变不一定写入数据库里，如果想撤回这些操作, 可以用命令 ROLLBACK 复原。

数据查询语言（DQL）

基本语法

```
SELECT 字段名1, 字段名2, …… FROM 表名1, [表名2, ……] WHERE 条件;
字段名可以带入函数例如:
 COUNT(*)
 MIN(字段名)
 MAX(字段名)
 AVG(字段名)
 DISTINCT(字段名),
 TO_CHAR(DATE字段名,'YYYY-MM-DD HH24:MI:SS')
 NVL(EXPR1, EXPR2)函数
```

比较运算符

between ...and ..	between ...and .. 在什么什么之间，包含边界，小值在前大值在后
In(set)	属于值列表中的一个
like	模糊查询
Is null	是否是空值

数据定义语言（DDL）

数据定义语言DDL用来创建数据库中的各种对象---用户、库、表、视图、索引、触发器、事件、存储过程和函数等。

CREATE (创建表, 索引, 视图, 同义词, 过程, 函数, 数据库链接等)

```
创建表:
```

```
create table [<模式名>.]<表名>(
          <字段1> <类型> [约束条件],
          <字段2> <类型> [约束条件],
          ...
   )[tablespace <命名空间>];

创建索引:
CREATE INDEX 索引名ON 表名 ( 字段1, [字段2, ……] );
ALTER INDEX 索引名 REBUILD;

创建视图:
CREATE VIEW 视图名AS SELECT …. FROM …..;
ALTER VIEW视图名 COMPILE;
视图仅是一个SQL查询语句, 它可以把表之间复杂的关系简洁化.

创建同义词:
CREATE SYNONYM同义词名FOR 表名;
CREATE SYNONYM同义词名FOR 表名@数据库链接名;

创建数据库链接 (DATABASE LINK):
CREATE DATABASE LINK 数据库链接名 CONNECT TO 用户名 IDENTIFIED BY
密码 USING ‘数据库连接字符串’;
数据库连接字符串可以用NET8 EASY CONFIG或者直接修改TNSNAMES.ORA里定
义.
数据库参数global_name=true时要求数据库链接名称跟远端数据库名称一样

查询数据库全局名称:
SELECT * FROM GLOBAL_NAME;

查询远端数据库里的表:
SELECT …… FROM 表名@数据库链接名;

```

ALTER (改变表, 索引, 视图等):

```
改变表的名称
ALTER TABLE 表名 TO 新表名;
在表的后面增加一个字段
```

```
ALTER TABLE 表名 ADD 字段名 字段名描述;
修改表里字段的定义描述
ALTER TABLE 表名 MODIFY字段名 字段名描述;
```

DROP (删除表, 索引, 视图, 同义词, 过程, 函数, 数据库链接等)

```
删除表和它所有的约束条件
DROP TABLE 表名 CASCADE CONSTRAINTS;
```

TRUNCATE (清空表里的所有记录, 保留表的结构)

```
TRUNCATE 表名;
```

数据控制语言（DCL)

数据控制语言DCL用来授予或回收访问数据库的某种特权，并控制数据库操纵事务发生的时间及效果，对数据库实行监视等

> 1) GRANT：授权。
> 2) ROLLBACK：回滚。
> 回滚命令使数据库状态回到上次最后提交的状态。
> 3) COMMIT：提交。
> 在数据库的插入、删除和修改操作时，只有当事务在提交到数据库时才算完成。
> 4)SET AUTOCOMMIT ON：设置自动提交
> 若把AUTOCOMMIT设置为ON，则在插入、修改、删除语句执行后，系统将自动进行提交，这就是自动提交。

GRANT 赋于权限

常用的系统权限集合有以下三个:

CONNECT(基本的连接)

RESOURCE(程序开发)

DBA(数据库管理)

常用的数据对象权限有以下五个:

ALL ON 数据对象名

SELECT ON 数据对象名

UPDATE ON 数据对象名

DELETE ON 数据对象名

INSERT ON 数据对象名

ALTER ON 数据对象名

GRANT CONNECT

RESOURCE TO 用户名;

GRANT SELECT ON 表名 TO 用户名;

```
GRANT SELECT, INSERT, DELETE ON 表名 TO 用户名1, 用户名2;
```

REVOKE 回收权限

```
REVOKE CONNECT, RESOURCE FROM 用户名;
REVOKE SELECT ON 表名 FROM 用户名;
REVOKE SELECT, INSERT, DELETE ON表名 FROM 用户名1, 用户名2;
```

Oracle常用SQL语句

```
1、连接SQL*Plus system/manager
2、显示当前连接用户SQL> show user
3、查看系统拥有哪些用户SQL> select * from all_users;
4、新建用户并授权SQL> create user a identified by a;（默认建在SYSTEM表空间下）SQL> grant connect,resource to a;
5、连接到新用户SQL> conn a/a
6、查询当前用户下所有对象SQL> select * from tab;
7、建立第一个表SQL> create table a(a number);
8、查询表结构SQL> desc a
9、插入新记录SQL> insert into a values(1);
10、查询记录SQL> select * from a;
11、更改记录SQL> update a set a=2;
12、删除记录SQL> delete from a;
13、回滚SQL> roll;SQL> rollback;
14、提交SQL> commit;
15、查出当前用户所有表名select unique tname from col;
```

常用函数

Oracle SQL 提供了用于执行特定操作的专用函数。这些函数大大增强了 SQL 语言的功能。

> 1.单行函数：对每一个函数应用在表的记录中时，只能输入一行结果，返回一个结果，
> 比如：MOD(x,y)返回 x 除以 y 的余数（x 和 y 可以是两个整数，也可以是表中的整数列）。
> 常用的单行函数有：
> 字符函数：对字符串操作。
> 数字函数：对数字进行计算，返回一个数字。

转换函数：可以将一种数据类型转换为另外一种数据类型。
日期函数：对日期和时间进行处理。

2.聚合函数：聚合函数同时可以对多行数据进行操作，并返回一个结果。比如 SUM(x)

字符函数

函数	功能	示例	结果
initcap(char)	首字母大写	initcap('hello')	Hello
lower(char)	转换为小写	lower('FUN')	fun
upper(char)	转换为大写	upper('sun')	SUN
ltrim(char,ser)	左剪裁	ltrim('xyzadams','xyz')	adams
rerim(char,set)	右剪裁	rtrim('xyzadams','ams')	xyzad
translate(char,from,to)	按字符翻译	translate('jack','abcd','1234')	j13k
replace(char,search_str,replace_str)	字符串替换	replace('jack and jue','j','bl')	black and blue
instr(char,substr[,post])	查找子串位置	instr('worldwide','d')	5
substr(char,pos,len)	取子字符串	substr('abcdefg',3,2)	cd
length(char)	字符串长度	length('adsfd')	5
concat(char1,char2)	连接字符串	concat('Hello','world')	Helloworld

面试题：请问SUBSTR()函数截取的时候下标从0还是从1开始?
在Oracle数据库之中，SUBSTR()函数从0或1开始都是一样的；
SUBSTR()也可以设置为负数，表示由后指定截取的开始点；

数字函数

函数	功能	示例	结果
abs(n)	取绝对值	abs(-15)	15
ceil(n)	向上取值	ceil(44.778)	45
sin(n)	正弦	sin(1.571)	.999999979
cos(n)	余弦	cos(0)	1
sign(n)	取符号	sign(-32)	-1
floor(n)	向下取整	floor(100.2)	100
power(m,n)	m的n次幂	power(4,2)	16
mod(m,n)	取余数	mod(10,3)	1
round(m,n)	四舍五入	round(100.256,2)	100.26
trunc(m,n)	截断	tranc(100.256,2)	100.25
sqrt(n)	平方根	sqrt(4)	2

```
-- 四舍五入
select round(255,-1) from dual;
-- 舍弃小数位
select trunc(255,-1) from dual;
```

常用日期函数

函数	功能
month_between(date1,date2)	返回两个日期间的月份
add_month(date,n)	返回把月份数n加到日期date上的新日期
next_day(date,week)	返回指定日期后的星期对应的新日期
last-day(date)	返回指定日期所在月的最后一天

常用转换函数

函数	功能	示例	结果
to_char(字符串 | 列, 格式字符串)	转换成字符串类型	to_char(1234.5,'$9999.9')	$1234.5
to_date(字符串，格式字符串)	转换成日期类型	to_date('1980-01-01','yyyy-mm-dd')	01-1月-80
to_number(字符串)	转换成数值类型	to_number('1234.5')	1234.5

```
（1）to_char(x[,format]):将x转化为字符串。 format为转换的格式，可以为数字格式或日期格式
select to_char('12345.67') from dual; --返回结果为12345.67
select to_char('12345.67','99,999.99') from dual; --返回结果为12,345.67

（2）to_number(x [,  format]):将x转换为数字。可以指定format格式
select to_number('970.13') + 25.5 from dual;
select to_number('-$12,345.67', '$99,999.99') from dual;

（3）cast(x as type):将x转换为指定的兼容的数据库类型
select cast(12345.67 as varchar2(10)),cast('05-7月-07' as date),cast(12345.678 as number(10,2)) from dual;

（4）to_date(x [,format]):将x字符串转换为日期
select to_date('2012-3-15','YYYY-MM-DD') from dual
```

常用其他函数

函数	功能
nvl(exp1,exp2)	如果exp1的值为null，则返回exp2的值，否则返回exp1的值
nvl2(exp1,exp2,exp3)	如果exp1的值不为null，则返回exp2的值，否则返回exp3的值
decode(value,if1,then1,if2,then2,.....,else)	如果value的值为if1，则返回then1的值，如果为if2，则返回then2的值，.....，否则返回else的值

```
a、NVL()函数，处理null
示例：如果查询的学生姓名为空时，给默认值（神秘人）
 SELECT
 STUNO,
 NVL(STUNAME, '神秘人'),
 ADDRESS
 FROM
 STU ORDER BY STUNO ASC;

b、DECODE()函数：多数值判断
DECODE()函数非常类似于程序中的if…else…语句，唯一不同的是DECODE()函数
判断的是数值，而不是逻辑条件。
语法如下：
 DECODE(数值 | 列 ，判断值1，显示值1，判断值2，显示值2，判断值3，显示值3，…)
示例：
 SELECT
 STUNO AS 学号,
 STUNAME AS 姓名,
 DECODE( STUSEX, '男','小哥哥','女', '小姐姐') AS 性别
 FROM
 STU ORDER BY STUNO ASC;
DECODE()函数是整个Oracle之中最具特点的函数，一定要将其掌握。
```

统计函数

```
（1）avg(x)：返回x的平均值
select avg(grade) from sc;
（2）count(x)：返回统计的行数
select count(name) from sc;
（3）max(x)：返回x的最大值
select max(grade) from sc;
（4）min(x)：返回x的最小值
select min(grade) from sc;
（5）sum(x)：返回x的总计值
select sum(grade) from sc;
```

第三章、Oracle分组查询、连接查询

分组、排序、过滤

```
语法：
 SELECT [列] 分组函数(列名),...
 FROM table
 [WHERE 条件]
 [GROUP BY 分组字段]
 [HAVING 多行函数筛选]
 [ORDER BY 排序字段];

实例：
 SELECT
 STUCLASSID AS 年级,
 COUNT( 1 ) AS 人数
 FROM
 STUS
 GROUP BY
 STUCLASSID
 ORDER BY
 STUCLASSID;
```

连接查询

笛卡尔集

笛卡尔集会在下面条件下产生:

- 省略连接条件
- 连接条件无效
- 所有表中的所有行互相连接

为了避免笛卡尔集，可以在WHERE加入有效的连接条件。

内连接（INNER JOIN）

结果为两个连接表中的匹配行的连接

左外连接（LEFT JOIN）

结果包括左表（出现在JOIN子句最左边）中的所有行，不包括右表中的不匹配行。

右外连接（RIGHT JOIN）

结果包括右表（出现在JOIN子句最右边）中的所有行，不包括有左表中的不匹配的行。

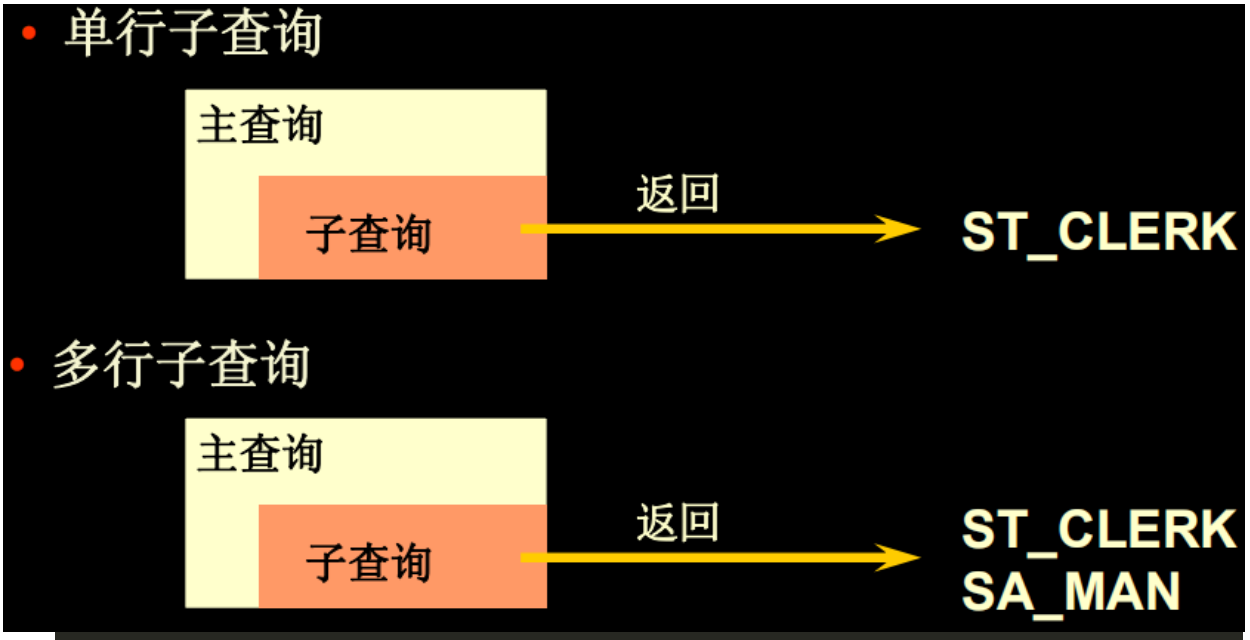
```
语法:
SELECT 表1.列名,表2.列名
FROM 表名1
INNER JOIN 表名2 ON 连接条件
INNER JOIN 表名3 ON 连接条件
```

```
6  WHERE 条件;
```

子查询（嵌套查询）

所谓子查询，即一个select语句中嵌套了另外的一个或者多个select语句，在查询是基于未知的值时应使用子查询，子查询 在主查询执行之前执行。

子查询类型：




```
1  语法：
2   SELECT select_list FROM
3   FROM 表名
4   WHERE 条件 (SELECT select_list FROM FROM 表名);
5
6  示例：
7   SELECT * FROM cj WHERE
8   stuid = ( SELECT stuid FROM xs WHERE stuname = '李思思' );
```

注意事项：

- 子查询要包含在括号内。
- 将子查询放在比较条件的右侧 。
- 除非进行Top-N （获取某一数据集合中的前N条记录）分析，否则不要在子查询 中使用ORDER BY 子句。
- 单行操作符对应单行子查询，多行操作符对应多行子查询。

单行子查询

> 只返回一行。
> 使用单行比较操作符。

操作符	含义
=	等于
>	大于
>=	大于等于
<	小于
<=	小于等于
<>	不等于

多行子查询

主查询与子查询返回的多个列进行比较。

多列子查询中的比较分为两种:

- 成对比较

```
SELECT employee_id, manager_id, department_id
FROM   employees
WHERE  (manager_id, department_id) IN
                       (SELECT manager_id, department_id
                        FROM   employees
                        WHERE  employee_id IN (178,174))
AND    employee_id NOT IN (178,174);
```

- 不成对比较

```
SELECT  employee_id, manager_id, department_id
FROM    employees
WHERE   manager_id IN
                    (SELECT  manager_id
                     FROM    employees
                     WHERE   employee_id IN (174,141))
AND     department_id IN
                    (SELECT  department_id
                     FROM    employees
                     WHERE   employee_id IN (174,141))

AND    employee_id NOT IN(174,141);
```

示例：

```
--查询与张三的年龄和性别相同的学生信息
SELECT * FROM XS
WHERE ( STUAGE, STUSEX ) IN (
SELECT
 STUAGE,
 STUSEX
FROM
 XS
WHERE
  STUNAME = '张三');
```

代码运行效果：

信息 结果 1

STUID	STUNAME	STUAGE	STUSEX	STUADD	STUCLASSID	STUBZ
12	张得美	20	男	洛阳孙村	4	(Null)
7	刘蛋蛋	20	男	洛阳齐全	4	暂无
1	张三	20	男	河南洛阳	1	三好学生

子查询中的 HAVING 子句：

```
思路：
首先查询出学生表中的最小年龄，然后根据查询出的最小年龄作为过滤条件，过滤
大于最小年龄的学生。
SELECT
 MIN( STUAGE ) 年龄
FROM
 XS
GROUP BY
 STUAGE
HAVING
 MIN( STUAGE ) > ( SELECT MIN( STUAGE ) FROM XS WHERE STUAGE = 20);
```

代码运行效果：

年龄
25
21
24
50

在多行子查询中使用 ANY 操作符

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

10 rows selected.

在多行子查询中使用 ALL 操作符

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

EXISTS 操作符

EXISTS 操作符检查在子查询中是否存在满足条件的行

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

```
SELECT STUID,STUNAME
FROM XS
WHERE EXISTS (SELECT 1 FROM XS WHERE STUNAME='张三');
```

相关修改

使用相关子查询依据一个表中的数据更新另一个表的数据

```
UPDATE table1 alias1
SET column = (SELECT expression
FROM table2 alias2
WHERE alias1.column =
alias2.column);
```

示例:

```
UPDATE employees e
SET    department_name =
                (SELECT department_name
                FROM   departments d
                WHERE  e.department_id = d.department_id);
```

相关删除

使用相关子查询依据一个表中的数据删除另一个表的数据

```
DELETE FROM table1 alias1
WHERE column operator
(SELECT expression
FROM table2 alias2
WHERE alias1.column = alias2.column);
```

示例:

```
DELETE FROM employees E
WHERE employee_id =
            (SELECT employee_id
             FROM   emp_history
             WHERE  employee_id = E.employee_id);
```

Oracle分页查询

```
SELECT
 *
FROM
 ( SELECT ROWNUM rnum, t.* FROM STUDENT t )
WHERE
 rnum >= 1 AND rnum <= 5;

分页计算公式:
 第一个rnum公式: (当前页-1) * 页大小 + 1
 第二个rnum公式: 当前页 * 页大小
```

第四章、PLSQL编程基础

什么是PL/SQL?

(1.) PL/SQL体系结构:

PL/SQL引擎用来编译和执行，PL/SQL块或子程序，该引擎驻留在Oracle服务器中。

(2.) PL/SQL块简介

PL/SQL是一种块结构语音，它将一组语句块放在一个快中。

(3.) 运算符和表达式:

PL/SQL语音支持操作符包含关系运算符，一般运算符和逻辑运算符，与SQL语音类似。

(4.) 常量和变量声明

在PL/SQL块的可执行部分引用变量和常量前，必须先对其进行声明。变量和PL/SQL块的部分声明，在PL/SQ块的 可执行部分被使用。

语法如下:

```
DECLARE variable_name [CONSTANT] type [not null][:=value];
声明 变量名称 是否为常量 变量的类型 是否为空 变量初始化
```

```
示例：
 v_stock_count NUMBER(11);
 v_id NUMBER(20):=3;
```

%TYPE方式：定义一个变量，其数据类型与已经定义的某个数据变量（尤其是表的某一列）的数据类型相一致这时可以使用%Type （优点：可以不必知道所引用的数据库列的数据类型。所引用的数据类型可以实现时改变，容易保持一致，不必修改PL/SQL程序）

```
语法：
 变量名 表名.列名%TYPE;
 或
 变量名 其他变量名%TYPE;

示例：
DECLARE -- 声明变量
 v_name PRODUCT."ProductName" %TYPE;
 v_price PRODUCT."Price" %TYPE;
  v_stock_count "PRODUCT"."Stock_Count" %TYPE;
  v_id NUMBER := 1;
BEGIN-- 开始部分
  SELECT
  "ProductName","Price","Stock_Count"
  INTO v_name,v_price,v_stock_count
  FROM
  PRODUCT
  WHERE
  "ProductId" = v_id;
  DBMS_OUTPUT.PUT_LINE ( '商品名称：' || v_name );
  DBMS_OUTPUT.PUT_LINE ( '商品价格：' || v_price );
  DBMS_OUTPUT.PUT_LINE ( '库存数量：' || v_stock_count );
END;--程序结束
```

%ROWTYPE方式: 返回一个记录类型，其数据类型和数据库表的数据结构相一致，这时可以使用%ROWTYPE.（优点：可以不必知道所引用数据库列的个数和数据类型。所引用的数据库中列的个数和数据库类型可以实现改变，容易保持一致，不用修改PL/SQL程序)

```
DECLARE -- 声明变量
 v_pro "PRODUCT" % ROWTYPE;
 v_id NUMBER := 1;
BEGIN-- 开始部分
 SELECT
 "ProductName",
 "Price",
 "Stock_Count"
 INTO
  v_pro."ProductName",
  v_pro."Price",
  v_pro."Stock_Count"
  FROM
  PRODUCT
  WHERE
  "ProductId" = v_id;
  DBMS_OUTPUT.PUT_LINE ( '商品名称: ' || v_pro."ProductName" );
  DBMS_OUTPUT.PUT_LINE ( '商品价格: ' || v_pro."Price" );
  DBMS_OUTPUT.PUT_LINE ( '库存数量: ' || v_pro."Stock_Count" );
END;--程序结束
```

（5.）注释：

在PL/SQL可以使用如下两种注释符号：

```
‘--’双减号
‘/ /’
```

PL/SQL 可用的SQL语句

PL/SQL是ORACLE系统的核心语言，现在ORACLE的许多部件都是由PL/SQL写成。在PL/SQL中可以使用的SQL语句有：

INSERT、UPDATE，DELETE，SELECT INTO，COMMIT，ROLLBACK， SAVEPOINT。

提示：在 PL/SQL中只能用 SQL语句中的 DML 部分，不能用 DDL 部分，如果要在PL/SQL中使用DDL(如CREATE table 等)的话，只能以动态的方式来使用。

PL/SQL的优点或特征

- **有利于客户/服务器环境应用的运行**

> 对于客户/服务器环境来说，真正的瓶颈是网络上。无论网络多快，只要客户端与服务器进行大量的数据交换。应用运行的效率自然就回受到影响。

- **适合于客户环境**

> PL/SQL由于分为数据库PL/SQL部分和工具PL/SQL。对于客户端来说，PL/SQL可以嵌套到相应的工具中，客户端程序可以执行本地包含PL/SQL部分，也可以向服务发SQL命令或激活服务器端的PL/SQL程序运行。

- **过程化**

> PL/SQL是Oracle在标准SQL上的过程性扩展，不仅允许在PL/SQL程序内嵌入SQL语句，而且允许使用各种类型的条件分支语句和循环语句，可以多个应用程序之间共享其解决方案。

- **模块化**

> PL/SQL程序结构是一种描述性很强、界限分明的块结构、嵌套块结构，被分成单独的过程、函数、触发器，且可以把它们组合为程序包，提高程序的模块化能力。

- **运行错误的可处理性**

> 使用PL/SQL提供的异常处理（EXCEPTION），开发人员可集中处理各种ORACLE错误和PL/SQL错误，或处理系统错误与自定义错误，以增强应用程序的健壮性。

- **提供大量内置程序包**

> ORACLE提供了大量的内置程序包。通过这些程序包能够实现DBS的一些低层操作、高级功能，不论对DBA还是应用开发人员都具有重要作用。

PL/SQL块语句

PL/SQL程序由三个块组成，即声明部分（DECLARE）、执行部分（BEGIN）、异常处理部分（EXCEPTION）。

基本结构

```
1 DECLARE
2 --声明部分: 在此声明PL/SQL用到的变量,类型及游标, 以及局部的存储过程和函数
3 BEGIN
```

```
-- 执行部分：过程及SQL 语句 ，即程序的主要部分
EXCEPTION
-- 执行异常部分：错误处理
END;
```

示例：

```
DECLARE -- 声明变量
 v_stock_count NUMBER;
 v_id NUMBER := 3;
BEGIN-- 开始部分
 SELECT
 "Stock_Count" INTO v_stock_count
 FROM
 PRODUCT
 WHERE
  "ProductId" = v_id;
  DBMS_OUTPUT.PUT_LINE ( '库存数量：' || v_stock_count );
EXCEPTION --异常处理
 WHEN others THEN
 DBMS_OUTPUT.PUT_LINE ( '该商品不存在' );
END;--程序结束
```

程序运行结果：

信息 DBMS 导出

库存数量：3000

命名规则

- 变量名首字母必须是英文字母，其后可以是字母，数字或者特殊字符$、# 和下划线_
- 变量名长度不要超过30字符
- 变量名不能有空格

标识符	命名规则	例子
程序变量	V_name	V_name
程序常量	C_Name	C_company_name
游标变量	Cursor_Name	Cursor_Emp
异常标识	E_name	E_too_many
表类型	Name_table_type	Emp_record_type
表	Name_table	Emp
记录类型	Name_record	Emp_record
SQL*Plus 替代变量	P_name	P_sal
绑定变量	G_name	G_year_sal

PL/SQL 变量类型

类型	子类	说　明	范　围	ORACLE限制
CHAR	Character String Rowid Nchar	定长字符串 民族语言字符集	0à32767 可选,确省=1	2000
VARCHAR2	Varchar, String NVARCHAR2	可变字符串 民族语言字符集	0à32767 4000	4000
BINARY_INTEGER		带符号整数,为整数计算优化性能		
NUMBER(p,s)	Dec Double precision Integer Int Numeric Real Small int	小数, NUMBER 的子类型 高精度实数 整数, NUMBER 的子类型 整数, NUMBER 的子类型 与NUMBER等价 与NUMBER等价 整数, 比 integer 小		
LONG		变长字符串	0->2147483647	32,767字节
DATE		日期型	公元前4712年1月1日至公元后4712年12月31日	
BOOLEAN		布尔型	TRUE, FALSE,NULL	不使用
ROWID		存放数据库行号		
UROWID		通用行标识符，字符类型		

PLSQL语句中的SELECT语句

依据select语句返回的记录数。实现分为两类：

当仅返回一条记录的时候：

SELECT select_list INTO {variable_name[,variable_name...]}

FROM table_name

[WHERE condition];

当返回0条或者多条记录的时候:

用cursor指针来实现

若结果是单行单列，into字句后用标量类型。与字段类型同样；

若查询结果为单行多列，into子句后的变量个数，顺序，数据类型和select语句后面的目标匹配，也能够用记录record类型类记录；

PL/SQL控制语句

条件语句 - IF语句

```
语法：
 IF 布尔表达式 THEN
 PL/SQL和SQL语句
 ELSE
 PL/SQL和SQL语句
 END IF;

示例：
DECLARE -- 声明变量
  v_id PRODUCT."ProductId" %TYPE:= 1;
  v_stock_count "PRODUCT"."Stock_Count" % TYPE;
BEGIN-- 开始部分
  SELECT
  "Stock_Count" INTO v_stock_count
  FROM
  PRODUCT
  WHERE
  "ProductId" = v_id;
  IF v_stock_count >0 THEN
  UPDATE PRODUCT SET "Stock_Count"="Stock_Count"-1 WHERE
"ProductId"=v_id;
  COMMIT;
  DBMS_OUTPUT.PUT_LINE ( 'id:' ||v_id|| '库存数量已更新！' );
  ELSIF v_stock_count <0 THEN
  DBMS_OUTPUT.PUT_LINE ( '库存数量<0,数量不正常' );
  ELSE
  DBMS_OUTPUT.PUT_LINE ( 'id:' ||v_id|| '已经没有库存了！');
  END IF;
END;--程序结束
```

表数据：

ProductId	ProductName	Price	Stock_Count
1	小米6	1999.12	-1
2	联想小新笔记本电脑	4800	900
3	外套	80.63	3000
4	裤子	120.99	800
5	卫衣	60	2000

程序运行结果：

信息 DBMS 导出

库存数量<0,数量不正常

条件语句 - CASE语句

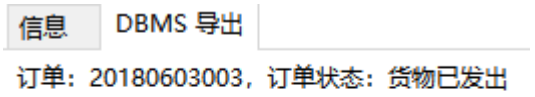
```
语法：
 CASE 条件表达式
 WHEN 条件表达式1 THEN
 语句段1
 WHEN 条件表达式2 THEN
 语句段2
 ELSE
 语句段
 END CASE;

示例：
DECLARE -- 声明变量
 v_id ORDERS."OrderId" %TYPE:= 20180603003;
 v_status ORDERS."Status" %TYPE;
 v_status_name VARCHAR2(20);
BEGIN-- 开始部分
 SELECT
 "OrderId","Status" INTO v_id,v_status
 FROM
 ORDERS
 WHERE
 "OrderId" = v_id;
 CASE v_status
 WHEN '1' THEN
 v_status_name:='订单已提交';
 WHEN '2' THEN
 v_status_name:='已付款';
```

```
28   WHEN '3' THEN
29   v_status_name:='货物已发出';
30   WHEN '4' THEN
31   v_status_name:='已完成';
32   ELSE
33   v_status_name:='未知状态';
34   END CASE;
35   DBMS_OUTPUT.PUT_LINE ( '订单号：' ||v_id|| '，订单状态：'|| v_statu
_name);
36
37  END;--程序结束
```

表数据：

OrderId	ProductId	Amount	Status
20181226001	3	1	1
20190107002	4	2	2
20180603003	1	1	3
20190105004	2	1	4
20190101005	5	6	5

程序运行结果：

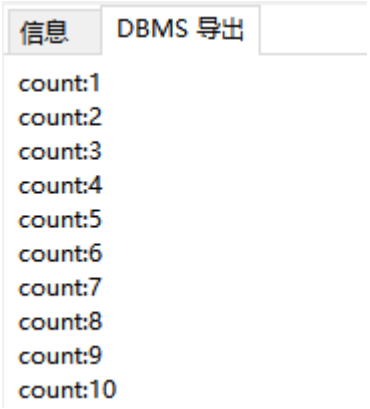


循环控制 - LOOP循环

```
1  语法：
2   LOOP
3   要执行的语句;
4   IF 条件语句 THEN --条件满足时跳出循环
5   EXIT;
6   END IF;
7   END LOOP;
8
9  示例：
10   DECLARE
11   v_count INTEGER :=1;
12   BEGIN
13   LOOP
14   DBMS_OUTPUT.PUT_LINE('count:'||v_count);
```

```
15  IF v_count>=10 THEN
16  EXIT;
17  END IF;
18  v_count:=v_count+1;
19  END LOOP;
20  END;
```

程序运行结果：



循环控制 - WHILE循环

```
1  语法：
2   WHILE 布尔表达式 LOOP
3   要执行的语句;
4   END LOOP;
5
6  示例：
7   DECLARE
8   v_count NUMBER :=1;
9   BEGIN
10   WHILE v_count<=20 LOOP
11   DBMS_OUTPUT.PUT_LINE('count:'||v_count);
12   v_count:=v_count+1;
13   END LOOP;
14   END;
```

程序运行结果：

信息 DBMS 导出

```
count:1
count:2
count:3
count:4
count:5
count:6
count:7
count:8
count:9
count:10
count:11
count:12
count:13
count:14
count:15
count:16
count:17
count:18
count:19
count:20
```

循环控制 - FOR循环

```
语法：
 FOR 循环计数器 IN 下限..上限 LOOP
 要执行的语句;
 END LOOP;

示例：
 BEGIN
 FOR num IN 1..20 LOOP
 DBMS_OUTPUT.PUT_LINE('number:'||num);
  END LOOP;
  END;
```

程序运行结果：

信息 DBMS 导出

```
number:1
number:2
number:3
number:4
number:5
number:6
number:7
number:8
number:9
number:10
number:11
number:12
number:13
number:14
number:15
number:16
number:17
number:18
number:19
number:20
```

PL/SQL异常处理

PL/SQL支持程序员在程序中使用 EXCEPTION 块捕获这些发生错误的条件，并针对错误情况采取适当的措施。

PL/SQL中有两种异常 -

- 系统定义的异常
- 用户定义的异常

语法

使用 **RAISE** 关键字引发异常

使用 **EXCEPTION** 关键字处理异常

```
DECLARE
 exception_name EXCEPTION; ---在这里预定义异常
BEGIN
 IF condition THEN
 RAISE exception_name; ---在这里触发异常
 END IF;
 ..... ---发生异常后不会执行这里代码
EXCEPTION
 WHEN exception_name THEN ---控制被转到异常处理部分，这里代码被执行
  ......
```

```
END;

示例:
DECLARE num NUMBER (11);--用来存储查询结果
 ex_num EXCEPTION;--声明异常
BEGIN
 SELECT COUNT(1) INTO num FROM PRODUCT WHERE "ProductId" = 100;
 IF
 num >= 1 THEN
 RAISE ex_num;--触发异常
 END IF;
 DBMS_OUTPUT.PUT_LINE ('可以注册! ');

EXCEPTION
 WHEN ex_num THEN
 DBMS_OUTPUT.PUT_LINE ('账号已被注册。');
END;
```

程序运行结果：

信息 DBMS 导出

可以注册!

预定义的异常

PL/SQL提供了许多预定义的异常，这些异常在程序违反任何数据库规则时执行。 例如，当 **SELECT INTO** 语句不返回任何行时，会引发预定义的异常 **NO_DATA_FOUND**。下图列出了一些重要的预定义异常情况：

异常	Oracle 错误代码	SQLCODE	描述
ACCESS_INTO_NULL	06530	-6530	当一个空对象被自动分配一个值时会引发它。
CASE_NOT_FOUND	06592	-6592	当没有选择 CASE 语句的 WHEN 子句中的任何选项时，会引发这个错误，并且没有 ELSE 子句。
COLLECTION_IS_NULL	06531	-6531	当程序尝试将 EXISTS 以外的集合方法应用于未初始化的嵌套表或 varray 时，或程序尝试将值分配给未初始化的嵌套表或 varray 的元素时，会引发此问题。
DUP_VAL_ON_INDEX	00001	-1	当尝试将重复值存储在具有唯一索引的列中时引发此错误。
INVALID_CURSOR	01001	-1001	当尝试进行不允许的游标操作(例如关闭未打开的游标)时会引发此错误。
INVALID_NUMBER	01722	-1722	当字符串转换为数字时失败，因为字符串不代表有效的数字。
LOGIN_DENIED	01017	-1017	当程序尝试使用无效的用户名或密码登录到数据库时引发。
NO_DATA_FOUND	01403	+100	当 SELECT INTO 语句不返回任何行时会引发它。
NOT_LOGGED_ON	01012	-1012	当数据库调用没有连接到数据库时引发。
PROGRAM_ERROR	06501	-6501	当PL/SQL遇到内部问题时会引发。
ROWTYPE_MISMATCH	06504	-6504	当游标在具有不兼容数据类型的变量中获取值时引发。
SELF_IS_NULL	30625	-30625	当调用成员方法时引发，但对象类型的实例未初始化。
STORAGE_ERROR	06500	-6500	当PL/SQL用尽内存或内存已损坏时引发。
TOO_MANY_ROWS	01422	-1422	当 SELECT INTO 语句返回多行时引发。
VALUE_ERROR	06502	-6502	当发生算术，转换，截断或者 sizeconstraint 错误时引发。
ZERO_DIVIDE	01476	1476	当尝试将数字除以零时引发。

预定义异常的处理：

这里说两个常见的异常 **no_data_found** 和 **too_many_rows**， 这两个异常多由 **select into** 语句触发。

```
示例：
 DECLARE -- 声明变量
 v_stock_count NUMBER ( 11 );
 v_id NUMBER ( 20 ) := 100;
 BEGIN-- 开始部分
 SELECT
```

```
"Stock_Count" INTO v_stock_count
FROM
PRODUCT
 WHERE
 "ProductId" = v_id;
 DBMS_OUTPUT.PUT_LINE (v_stock_count);
 EXCEPTION --异常处理
 WHEN NO_DATA_FOUND THEN
 DBMS_OUTPUT.PUT_LINE ('该商品不存在');
 END;--程序结束

NO_DATA_FOUND异常:
起因：给一个变量赋值时，查询的结果为空。
说明：如上面例子可以看到，一旦直接抛出异常，就会让过程中断。
```

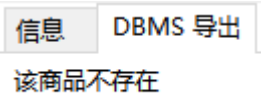
表数据:

OrderId	ProductId	Amount	Status
1	3	1	1
2	4	2	2
3	1	1	3
4	2	1	4
5	5	6	5

程序运行结果:

id存在

id不存在



动态SQL

编译期间SQL语句是不确定的，并且在运行时允许发生变化。

应用场合

- 要执行一个DDL语句时
- 需要增加程序灵活性
- 使用DBME_SQL动态执行SQL语句时

语法

```
Excute immediate 动态SQL语句 using 绑定参数列表 returning into 输出参数列表;
对这一语句作如下说明：
 1)动态SQL是指DDL和不确定的DML（即带参数的DML）
    2)绑定参数列表为输入参数列表在运行时与动态SQL语句中的参数（占位符）进行绑定。
    3)输出参数列表为动态SQL语句执行后返回的参数列表。

示例：
 DECLARE
 i_sql VARCHAR2(500);
  v_ProductId PRODUCT."ProductId"%TYPE:=6;
  v_ProductName PRODUCT."ProductName"%TYPE:='ipad';
  v_Price PRODUCT."Price"%TYPE:=3800;
  v_Stock_Count PRODUCT."Stock_Count"%TYPE:=8000;
  BEGIN
  i_sql:='INSERT INTO PRODUCT VALUES(:1, :2, :3, :4)';
  EXECUTE IMMEDIATE i_sql USING v_ProductId,v_ProductName,v_Price,v_Stock_Count;
  COMMIT;
  END;
```

程序运行结果：

ProductId	ProductName	Price	Stock_Count
1	小米6	1999.12	-1
2	联想小新笔记本电脑	4800	900
3	外套	80.63	3000
4	裤子	120.99	800
5	卫衣	60	2000
6	ipad	3800	8000

游标

游标是指向此上下文区域的指针。PL/SQL通过游标控制上下文区域，游标保存SQL语句返回的行(一个或多个记录)。 游标所在的行集称为活动集。

游标类型

- 隐式游标：返回单行记录
- 显示游标：返回多行记录，逐行读取

语法

```
CURSOR cursor_name [(parammeter [,parammeter].....)]
[RETURN return_type] IS select_sql;

CURSOR: 用于声明一个游标
parammeter: 可选参数，用于指定参数类型，模式等
RETURN可选: 可选，指定游标的返回类型
select_sql: 需要处理的select语句，不能含INTO子句
```

使用显式游标步骤：

1. 声明游标初始化内存

```
CURSOR c_customers IS
SELECT id, name, address FROM customers;
```

2. 打开游标分配内存

```
OPEN c_customers;
```

3. 从游标获取数据

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

4. 关闭游标以释放分配的内存

```
CLOSE c_customers;
```

使用FOR循环读取游标：

```
For 声明记录变量 in 名称 Loop
 Executable_statements
End loop;

示例：
```

```
6  DECLARE
7  CURSOR c_list IS SELECT "ProductId", "ProductName" FROM
 PRODUCT;
8  BEGIN
9  FOR c IN c_list LOOP
10  DBMS_OUTPUT.PUT_LINE(c."ProductId"||'.'||c."ProductName");
11  END LOOP;
12  END;
```

程序运行结果：

信息 DBMS 导出

1.小米6
2.联想小新笔记本电脑
3.外套
4.裤子
5.卫衣
6.ipad

游标的属性

编号	属性	描述
1	%FOUND	如果 INSERT ， UPDATE 或 DELETE 语句影响一行或多行，或者兄 SELECT INTO 语句返回一行或多行，则返回 TRUE ，否则返回 FALSE 。
2	%NOTFOUND	与 %FOUND 的逻辑相反。 如果INSERT，UPDATE或DELETE语句没有影响任何行，或SELECT INTO语句未返回任何行，则返回TRUE。 否则返回FALSE。
3	%ISOPEN	由于Oracle在执行关联的SQL语句后会自动关闭SQL游标，因此总是为隐式游标返回 FALSE 。
4	%ROWCOUNT	返回受 INSERT ， UPDATE 或 DELETE 语句，或者受 SELECT INTO 语句影响的行数。

第五章、视图、事务和索引

视图

视图可以理解为数据库中一张虚拟的表，他是通过一张或者多张基表进行关联查询后组成一个虚拟的逻辑表。查询视图，本质上是对表进行关联查询。

视图的本身是不包含任何数据，只是一个查询结果，当基表的数据发生变化时，视图里面的数据也会跟着发生变化。

为什么要用视图（视图的优点）？

1.简化数据操作：视图可以简化用户处理数据的方式。

2.着重于特定数据：不必要的数据或敏感数据可以不出现在视图中。

3.视图提供了一个简单而有效的安全机制，可以定制不同用户对数据的访问权限。

4.提供向后兼容性：视图使用户能够在表的架构更改时为表创建向后兼容接口。

5.自定义数据：视图允许用户以不同方式查看数据。

6.导出和导入数据：可使用视图将数据导出到其他应用程序。

语法

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view_name

[(alias[, alias]...)]

AS subquery

[WITH CHECK OPTION [CONSTRAINT constraint]]

[WITH READ ONLY]

OR REPLACE ：若所创建的试图已经存在，ORACLE自动重建该视图；
FORCE ：不管基表是否存在ORACLE都会自动创建该视图；
NOFORCE ：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 ：该视图上不能进行任何DML操作。

示例：
-给用户授权创建视图
-- grant create all view to zdgshr; 所有数据库都能创建视图的用户
grant create view to zdgshr;

--创建简单视图
```

```sql
create view temp
as
select * from zd_member_basic_info;
--测试
select * from temp where rownum=1;

--创建简单视图：只读
create or replace view temp1
as
select id,job_number,name,dept_id from zd_member_basic_info
with read only;
--测试
insert into temp1(id,job_number,name,dept_id) values(1,0,'张三',1300);
```

视图的删除：

```sql
DROP VIEW VIEW_NAME。
```

> 删除视图的定义不影响基表中的数据。
> 只有视图所有者和具备DROP VIEW权限的用户可以删除视图。
> 视图被删除后，基于被删除视图的其他视图或应用将无效。

视图分类

- 视图分为简单视图和复杂视图。
- 简单视图只从单表里获取数据；复杂视图从多表里获取数据。
- 简单视图不包含函数和数据组；复杂视图包含函数和数据组。
- 简单视图可以实现DML操作；复杂视图不可以。

事务

在数据库中事务是工作的逻辑单元，一个事务是由一个或多个完成一组的相关行为的SQL语句组成，通过事务机制确保这一组SQL语句所作的操作要么都成功执行，完成整个工作单元操作，要么一个也不执行。

如：网上转帐就是典型的要用事务来处理，用以保证数据的一致性。

事务的四个特性ACID：

1. Atomicity（原子性）：事务中sql语句不可分割，要么都做，要么都不做

2. Consistency(一致性)： 指事务操作前后，数据库中数据是一致的，数据满足业务规则约束（例如账户金额的转出和转入），与原子性对应。

3. Isolation（隔离性）：多个并发事务可以独立运行，而不能相互干扰，一个事务修改数据未提交前，其他事务看不到它所做的更改。

4. Durability（持久性）：事务提交后，数据的修改是永久的。

以下情况之一为事务的结束：

1.显式的结束：执行了commit或是rollback;

2.隐式的提交：执行了DDL,DCL语句，或是exit退出。

3.隐式的回滚：系统异常关闭，死机，断电。

语法：

```
提交：
 COMMIT;
回滚：
 rollback to 事务名; --撤销单个事务
 or
 rollback;    --如果后面不加事务名，则撤销所有存在的事务

示例：
-- 从账户一向账户二转账
DECLARE
 v_money NUMBER(8, 2); -- 转账金额
 v_balance account.balance%TYPE; -- 账户余额
BEGIN
 v_money := &转账金额; -- 输入转账金额
 -- 从账户一减钱
 UPDATE account SET balance = balance - v_money WHERE id=&转出账户
 RETURNING balance INTO v_balance;
 IF SQL%NOTFOUND THEN
 RAISE_APPLICATION_ERROR(-20001, '没有该账户：'||&转出账户);
 END IF;
 IF v_balance < 0 THEN
```

```
22  RAISE_APPLICATION_ERROR(-20002, '账户余额不足');
23  END IF;
24
25  -- 向账户二加钱
26  UPDATE account SET balance = balance + v_money WHERE id=&转入账
户;
27  IF SQL%NOTFOUND THEN
28  RAISE_APPLICATION_ERROR(-20001, '没有该账户：'||&转入账户);
29  END IF;
30
31  -- 如果没有异常，则提交事务
32  COMMIT;
33  DBMS_OUTPUT.PUT_LINE('转账成功');
34
35  EXCEPTION
36  WHEN OTHERS THEN
37  ROLLBACK; -- 出现异常则回滚事务
38  DBMS_OUTPUT.PUT_LINE('转账失败：');
39  DBMS_OUTPUT.PUT_LINE(SQLERRM);
40 END;
```

索引

索引是一种供服务器在表中快速查找一个行的数据库结构。合理使用索引能够大大提高数据库的运行效率。

在数据库中建立索引主要有以下作用：

（1）快速存取数据。

（2）既可以改善数据库性能，又可以保证列值的唯一性。

（3）实现表与表之间的参照完整性

（4）在使用orderby、groupby子句进行数据检索时，利用索引可以减少排序和　　分组的时间。

(5) 索引对用户是透明的，无论表上是否有索引，sql语句的用法不变。

（6）oracle创建主键时会自动在该列上创建索引。

语法：

```
CREATE [UNIQUE] | [BITMAP] INDEX index_name --unique表示唯一索引
ON table_name([column1 [ASC|DESC],column2 --bitmap，创建位图索引
[ASC|DESC],…] | [express])
[TABLESPACE tablespace_name]
[PCTFREE n1] --指定索引在数据块中空闲空间
[STORAGE (INITIAL n2)]
[NOLOGGING] --表示创建和重建索引时允许对表做DML操作，默认情况下不应该使用
[NOLINE]
[NOSORT]; --表示创建索引时不进行排序，默认不适用，如果数据已经是按照该索引顺序排列的可以使用
```

其中：

schema：ORACLE模式，缺省即为当前帐户

index：索引名

table：创建索引的基表名

column：基表中的列名，一个索引最多有16列，long列、long raw列不能建索引列

DESC、ASC：缺省为ASC即升序排序

CLUSTER：指定一个聚簇（Hash cluster不能建索引）

INITRANS、MAXTRANS：指定初始和最大事务入口数

Tablespace：表空间名

STORAGE：存储参数，同create table 中的storage.

PCTFREE：索引数据块空闲空间的百分比(不能指定pctused)

NOSORT：不（能）排序（存储时就已按升序，所以指出不再排序）

注意：

- 一个基表不能建太多的索引；
- 空值不能被索引；
- 只有唯一索引才真正提高速度,一般的索引只能提高30%左右。

修改索引

修改索引的主要任务是修改已存在索引的存储参数适应增长的需要或者重新建立索引。

```
ALTER [UNIQUE] INDEX [user.]index
[INITRANS n]
[MAXTRANS n]
REBUILD
[STORAGE n]
```

其中：

REBUILD是根据原来的索引结构重新建立索引，实际是删除原来的索引后再重新建立。

提示：

DBA经常用REBUILD来重建索引可以减少硬盘碎片和提高应用系统的性能。

删除索引

当不需要时可以将索引删除以释放出硬盘空间。

```
DROP INDEX [schema.]indexname
```

注：当表结构被删除时，有其相关的所有索引也随之被删除。

索引分类

1. B树索引（默认索引，保存讲过排序过的索引列和对应的rowid值）

1）说明：

1.oracle中最常用的索引；B树索引就是一颗二叉树；叶子节点（双向链表）包含索引列和指向表中每个匹配行的ROWID值

2.所有叶子节点具有相同的深度，因而不管查询条件怎样，查询速度基本相同

3.能够适应精确查询、模糊查询和比较查询

2）分类：

UNIQUE,NON-UNIQUE(默认),REVERSE KEY（数据列中的数据是反向存储的）

3）创建例子

```
craete index index_sno on student('sno');
```

4）适合使用场景：

列基数（列不重复值的个数）大时适合使用B数索引

2. 位图索引

1）说明：

1.创建位图索引时，oracle会扫描整张表，并为索引列的每个取值建立一个位图（位图中，对表中每一行使用一位（bit，0或者1）来标识该行是否包含该位图的索引列的取值，如果为1，表示对应的rowid所在的记录包含该位图索引列值），最后通过位图索引中的映射函数完成位到行的ROWID的转换

2)创建例子

```
create bitmap index index_sno on student(sno);
```

3) 适合场景：

对于基数小的列适合简历位图索引（例如性别等）

3.单列索引和复合索引（基于多个列创建）

1) 注意：

即如果索引建立在多个列上，只有它的第一个列被where子句引用时，优化器才会使用该索引，即至少要包含组合索引的第一列

4. 函数索引

1)说明：

1. 当经常要访问一些函数或者表达式时，可以将其存储在索引中，这样下次访问时，该值已经计算出来了，可以加快查询速度

2. 函数索引既可以使用B数索引，也可以使用位图索引；当函数结果不确定时采用B树索引，结果是固定的某几个值时使用位图索引

3. 函数索引中可以水泥用len、trim、substr、upper（每行返回独立结果），不能使用如sum、max、min、avg等

2）例子：

```
create index fbi on student (upper(name));
select * from student where upper(name) ='WISH';
```

5.反向键索引

```
craete index index_sno on student('sno') REVERSE;
```

索引建立原则总结

1、如果有两个或者以上的索引，其中有一个唯一性索引，而其他是非唯一，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。

2、至少要包含组合索引的第一列（即如果索引建立在多个列上，只有它的第一个列被where子句引用时，优化器才会使用该索引）。

3、小表不要建立索引。

4、对于基数大的列适合建立B树索引，对于基数小的列适合简历位图索引。

5、列中有很多空值，但经常查询该列上非空记录时应该建立索引。

6、经常进行连接查询的列应该创建索引。

7、使用create index时要将最常查询的列放在最前面。

8、LONG（可变长字符串数据，最长2G）和LONG RAW（可变长二进制数据，最长2G）列不能创建索引。

9、限制表中索引的数量（创建索引耗费时间，并且随数据量的增大而增大；索引会占用物理空间；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，降低了数据的维护速度）。

Oracle索引什么时候失效

1. 在索引列上使用函数
2. 使用<>、not in 、not exist，对于这三种情况大多数情况下认为结果集很大，一般大于5%-15%就不走索引而走FTS。
3. 单独的>、<。
4. like "%_" 百分号在前。
5. 字符型字段为数字时在where条件里不添加引号。
6. B-tree索引 is null不会走,is not null会走,位图索引 is null,is not null都会走、联合索引 is not null 只要在建立的索引列（不分先后）都会走。

第六章、存储过程、函数

存储过程

什么是存储过程（procedure）？

事先运用oracle语法，写好的一段具有业务功能的程序片段，长期保存在Oracle服务器中语言远程访问，类似于java中的函数。

子程序

命名的PLSQL块，编译并存储在数据库中。

- 子程序的各个部分：

> 声明部分
> 可执行部分
> 异常处理部分

- 子程序的分类：

> 过程 - 执行某些操作
> 函数 - 执行操作并返回值

为什么要用存储过程?

（1）PLSQL每次执行都要整体运行一遍，才有结果

（2）PLSQL不能将其封装起来，长期保存在oracle服务器中

（3）PLSQL不能被其它应用程序调用，例如：Java

过程的结构

- 声明部分：包括类型、变量、游标
- 执行部分：完成功能而编写的SQL语句或则是PLSQL代码块
- 异常处理部分

存储过程与PLSQL是什么关系?

存储过程是PLSQL的一个方面的应用，PLSQL是存储过程的基础

存储过程的概念?

存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的SQL 语句集，存储在数据库中，经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。

存储过程传递参数模式

- IN

用于接受调用程序的值

默认的参数模式

- OUT

用于向调用程序返回值

- IN OUT

用于接受调用程序的值，并向调用程序返回更新的值

创建存储过程语法

```
CREATE [OR REPLACE] PROCEDURE
 过程名 [(参数列表>)]
IS|AS
 PLSQL程序
BEGIN
 executable_section ---包括在存储过程中要执行的语句
[EXCEPTION
 exception_section] ---处理异常
END;

 简单写法:
  CREATE OR REPLACE PROCEDURE 过程名 AS PLSQL程序;

 示例:
  --创建无参存储过程HELLO，无返回值
  CREATE OR REPLACE PROCEDURE HELLO
  AS
  BEGIN
      DBMS_OUTPUT.PUT_LINE('第一个存储过程');
  END;
```

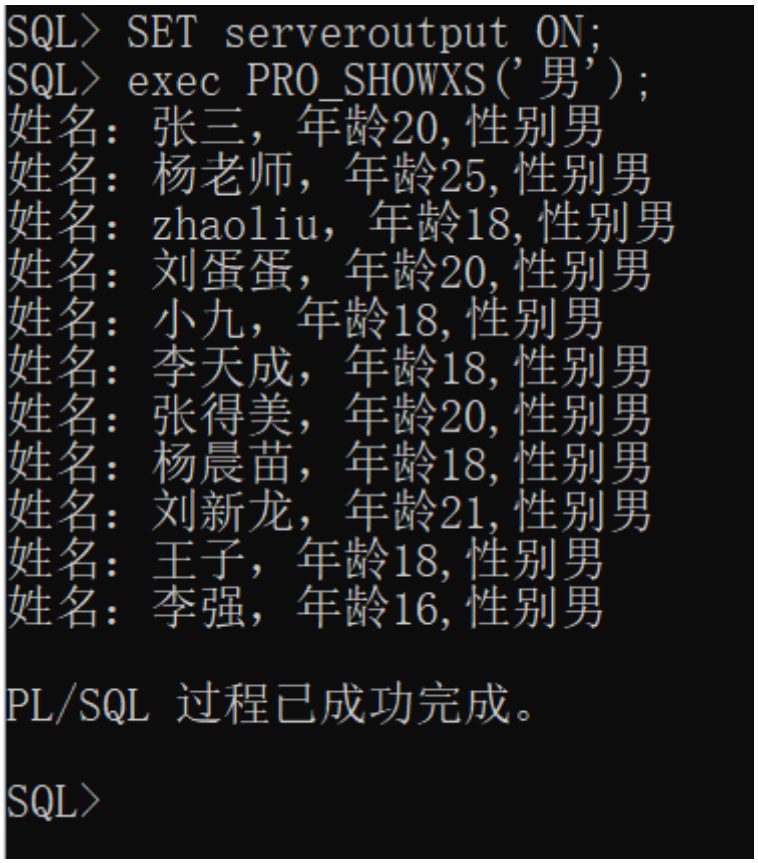
删除存储过程

```
语法:
 DROP PROCEDURE 过程名;
示例:
 DROP PROCEDURE HELLO;
```

调用存储过程方式一：SQLPLUS

```
EXEC 存储过程名;
示例：
 SET serveroutput ON;
 exec PRO_SHOWXS('男');
```

程序运行结果：



调用存储过程方式二：PLSQL程序

```
BEGIN
    HELLO;
END;
```

什么情况下用EXEC调用，什么情况下用PLSQL调用存储过程?

- EXEC适合存储过程没有返回值
- PLSQL适合存储过程含有多个返回值

调用存储过程方式三，JAVA程序

存储函数

与过程函数类似，是一组SQL语句或者PLSQL语句块的集合，同时能够返回执行结果。

函数的结构

过程的结构

- 声明部分：包括类型、变量、游标
- 执行部分：完成功能而编写的SQL语句或则是PLSQL代码块
- 异常处理部分

存储函数与存储过程的区别

一、 存储函数有且只有一个返回值，而存储过程不能有返回值。

二、 函数只能有输入参数，而且不能带in, 而存储过程可以有多个in,out,inout参数。

三、 存储过程中的语句功能更强大，存储过程可以实现很复杂的业务逻辑，而函数有很多限制，如不能在函数中使用 insert,update,delete,create 等语句；存储函数只完成查询的工作，可接受输入参数并返回一个结果，也就是函数实现的功能针对性比较强。

四、 存储过程可以调用存储函数。但函数不能调用存储过程。

五、 存储过程一般是作为一个独立的部分来执行(call调用)。而函数可以作为查询语句的一个部分来调用。

存储过程、函数适合场景

什么情况下【适合使用】存储过程？什么情况下【适合使用】存储函数?

> **【适合使用】**存储过程： 无返回值或者由多个返回值时适合过程。
> **【适合使用】**存储函数：有且只有一个返回值时，适合函数。

什么情况【适合使用】过程函数，什么情况【适合使用】SQL?

【适合使用】过程函数：

- 需要长期保存在数据库中
- 需要被多个用户重复调用
- 业务逻辑相同，只是参数不一样
- 批操作大量数据，例如：批量插入很多数据

【适合使用】SQL：

- 凡是上述反面，都可使用SQL
- 对表，视图，序列，索引，等这些还是要用SQL

存储函数语法

```
CREATE [OR REPLACE] FUNCTION 函数名 ([函数参数[,…]])
RETURN 返回类型
 AS PLSQL程序段
BEGIN
 IF
 RETURN 返回的数据
 ELSE
 RETURN 返回的数据
 END IF;
END;

示例1：
-- 创建有参存储函数findempnameandjobandsal(编号)，查询7788号员工的的姓名(RETURN)，职位(OUT)，月薪(OUT)，返回多个值
create or replace function findSix(pempno in number,pjob out varchar2,psal out varchar2) return varchar2
as
    pename emp.ename%type;
    --pjob emp.job%type;
    --psal emp.sal%type;
begin
    select ename,job,sal into pename,pjob,psal from emp where empno=pempno;
    return pename;
end;

-- 调用
declare
    pename emp.ename%type;
    pjob emp.job%type;
```

```
    psal emp.sal%type;
begin
    pename:=findSix(7788,pjob,psal);
    dbms_output.put_line(pename||pjob||psal);
end;

示例2:
--创建有参存储函数findempincome(编号)，查询7369号员工的年收入，演示IN的用法，默认IN
create or replace function findNumber(pempno in number)
 return number
as
    psal number;
begin
    select sal into psal from emp where empno=pempno;
    return psal;
end;

--调用
declare
    income number;
begin
    income:=findNumber(7369);
    dbms_output.put_line(income);
end;
```

删除存储函数

```
语法:
DROP FUNCTION 函数名;
```

调用存储函数方式一，PLSQL程序

```
declare
    name varchar2(50);
begin
    name:=getName();
    dbms_output.put_line(name);
```

```
6 end;
```

调用存储函数方式二，JAVA程序