PL / SQL 培训教程

文档作者:

创建日期:

确认日期:

控制编码:

当前版本: 1.0

Company Confidential - For internal use only

目录

Company Confidential - For internal use only

UNIT1 SQL、PL/SQL 概要

单元培训目标

- 了解 SQL、PL/SQL 的基本概念、功能
- 了解 Oracle 产品集和关联
- 了解 SQL、PL/SQL 的基本命令

LESSON 1 SQL、SQL*Plus 和 PL/SQL 基本概念

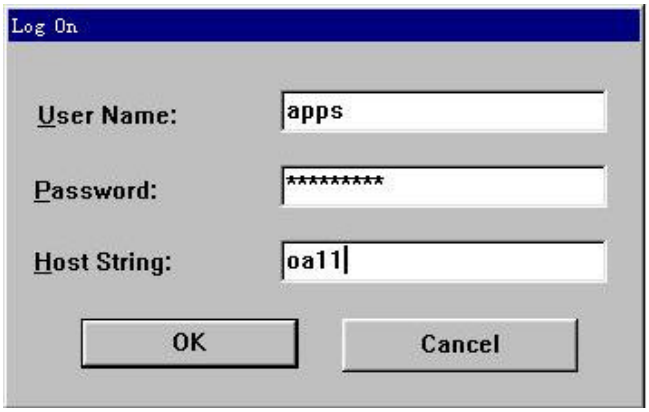
SQL、SQL*Plus 和 PL/SQL 是用来检索和维护存储在 Oracle 数据库上数据的工具或语言。

- Oracle 是一个对象— 关系型数据库管理系统（ORDBMS），它扩展了普通关系型数据库管理方式，支持面向对象的概念，以表的形式存储信息，维护数据的输入、数据的存储和数据查询并处理及对数据的三种基本操作：采集、存储、检索。
- SQL: 结构化查询语言(Structured Query Language)，用来与 Oracle 数据库服务器进行交互。

  例：select name, salary from t_emp

  select sysdate from dual
- PL/SQL: Oracle 扩展的 SQL 语言，具有完整的流程控制定义。
- SQL*Plus: Oracle 常用的工具，用来识别 SQL 语言和编写、执行 PL/SQL 代码，它与 Oracle 数据库管理系统紧密结合。
- SQL*Plus 环境登录：

  （1） windows 环境



  （2） 命令行环境

  如下格式：

Company Confidential - For internal use only

```
Sqlplus    [username [/password          [@database]]]
```

username　　**数据库用户名**

password　　**数据库的密码，此时是可见的**

@database　　**要登录的数据库名**

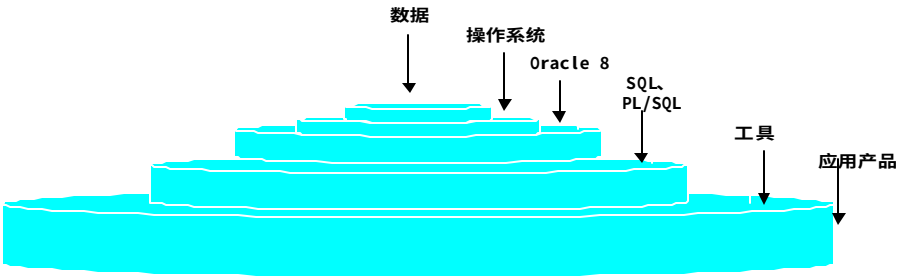
一旦成功登录，系统提示如下：

```
SQL>
```

- SQL **常用命令：**

分类	**命令**
数据检索	Select
数据维护(DML)	insert、update、delete
数据定义(DDL)	create、drop、alert、rename、truncate
事务处理控制	commit、rollback、savepoint
数据控制(DCL)	Grant、revoke

- **数据、操作系统、**Oracle **数据库管理器、** SQL **和** PL/SQL **及工具之间的关系**
- **在** Oracle **的产品中有如下的关系：以数据为核心，**Oracle Server **结合操作系统进行数据传递和通信，外围有** SQL*Plus **等工具提供给用户，通过** SQL、PL/SQL **与** Oracle Server **交互。（如下图：）**
- **代码、缩写和命名标准：由于** SQL **与自然语的相似性，同时为提高代码的可读性，对于检索语句、表列命名和程序代码应采用标准规则，表名和列名既要反映数据的实际意义，又要考虑简洁性、完整性。**Oracle **允许使用最多** 30 **个字符命名表和列，为建立完整的、含义明确的、描述性的名称提供了足够的空间。对于程序代码，更要讲究规范性，如变量、函数名等等，要含义明确，提高程序的可读性。**



Company Confidential - For internal use only

UNIT2 数据查询

单元培训目标

- **理解** Oracle **数据查询的概念和作用；**
- **掌握数据查询的基本方法；**
- **通过函数、子查询的学习，能写一些较复杂的数据查询。**

LESSON 1 简单的数据查询

- **基本结构：**

```
SELECT  [distinct] {*, column [alias], ... ... }
FROM TABLE
```

SELECT: **选择要查询的数据列**

FROM: **数据列所在的表**

*: **选择所有的列**

COLUMN:**选择的列名**

ALIAS: **给选定的列一个别名**

例：SELEC T * FROM s_dept

SELECT last_name, first_name, address FROM s_emp

- **算术表达式：**

加：+

减：-

乘：*

除：/

例：

SELECT last_name, salary * 12, base_salary + salary

FROM tablename

- **字符串处理**

字符串可以做最简单的“加”法，即合并运算：||

例：

SELECT column1||':'||column2 ||'$'

FROM tablename

Company Confidential - For internal use only

注：column1、column2**必须位字符型数据列**

SELECT first_name ||' '|| last_name||','|| title "Employees" FROM s_emp

- **空处理**

NVL: **在数据列的值为空时（并非为零），以真实的值进行替代处理**

例：

NVL (start_date, sysdate)

NVL (name, 'no name')

NVL (base_salary, 2000)

注：任何未被声明为 NOT NULL **或** PRIMARY KEY **的列都可包含空值，注意，空值就是空值，并不是值为零或空格，零是一个数值，而空格是一个字符。**

- **消除重复行：**

例：SELECT item_id from t_onhand

SELECT **DISTINCT** item_id FROM t_onhand

选者多列非重复：

例：SELECT DISTINCT warehoused_id, item_id FROM t_onhand

- **显示表结构：**

格式：

```
DESC[RIBE] tablename
```

Oracle **常用数据类型：**

number(p, s): **浮点类型，最大精度** p**，小数位** s **位**

varchar2(s): **可变长字符类型，最大长度** s**位**

date: **日期型**

char(s): **定长字符类型，长度为** s

例：

```
SQL>desc s_item
```

Name	Null	Type
item_id	NOT NULL	number
item_name	NOT NULL	varchar2(20)
base_price		number,
description		varchar2(100)
class_id		number

Company Confidential - For internal use only

LESSON 2 SQL*Plus 的使用

- SQL*Plus **的使用：**

I. **文件命令**

a) SAVE filename **把当前** SQL **缓冲区的内容存储在文件** filename **中**

b) GET filename **把文件** filename **中的内容写入当前** SQL **缓冲区**

c) START filename **执行存储在** filename **中的内容**

d) @ filename **执行存储在** filename **中的内容**

e) EDIT filename **打开文本编辑器，把当前** SQL **缓冲区的内容写入文件** afiedt.buf

f) SPOOL filename **把查询的数据结果存储在** filename **中**

g) EXIT **退出** SQL*Plus

II. **文本编辑命令**

a) A[PPEND] text

b) C[HANGE] / old / new

c) CL[EAR]buff[ER]

d) DEL

e) I[NPUT] text

f) L[IST] n

g) N text

III. **格式输出**

a) **格式：**

```
COL[UMN] [{column | alias}[option ... ... ]]
```

b) **参数选项：**

CLE[AR]、FOR[MAT]format、HEA[DING]text、JUST[TIFY]{align}、NOPRI[NT]、NUL[L]text、PRI[NT]、TRU[NCATED]、WEA[PPEND]、WOR[D_WRAPPED]

例：

```
COLUMN last_name HEADING 'employee | name' FORMAT A15
COLUMN salary JUSTIFY LEFT FORMAT $99,999.00
COLUMN start_date FORMAT A9 NULL 'Not hired'
```


Company Confidential - For internal use only

- **数据控制输出**

```
SELECT          expr
FROM            table
[WHERE          condition(s)        ]
[ORDER BY expr  ];
```

- **逻辑比较**

=、>=、<=、>、<

between... and...、in、like、is null

and、or、not

例：

SELECT last_name, employee_id FROM t_emp

WHERE start_date>to_date(2001-01-01', 'yyyy-mm-dd)

And last_name like '%H%'

LESSON 3 单行函数的使用

```
Function_name       (column | expression, [arg1, arg2, ... ... ])
```

- **字符函数**

LOWER(col | expr)、UPPER(col | expr)、INITCAP(col | expr)

CONCAT(col | expr1, col | expr2)、SUBSTR(col | expr, m[, n])

LENGTH(col | expr)

NVL(col | expr1, col | expr2)

- **数值函数**

ROUND

TRUNC

MOD

注意：round 和 trunc 函数的不同使用，如下：

例：round(23535.7476, 2) ==> 23535.75

round(23535.7476, 0) ==> 23536

round(23535.7476, -1) ==> 23530

trunc(23535.7476, 2) ==>23535.74

trunc(23535.7476) ==>23535

Company Confidential - For internal use only

trunc(23535, -1) ==> 23530

- **日期函数**

MONTH_BETWEEN、ADD_MONTHS、NEXT_DAY

LAST_DAY、ROUND、TRUNC

日期函数有其特性：

例：

add_months(to_date(2001-05-01','yyyy-mm-dd'), 2) ==>2001-07-01

add_months(to_date(2001-05-01','yyyy-mm-dd'), -2) ==>2001-03-01

两个日期不可相加，但可做减法，结果为 number 型，日期型数据加/减数值型数据结果为日期型，日期型数据间不可做乘除法，其他运算借助于日期函数。

- **转换函数**

TO_NUMBER(number/date, ['fmt'])

TO_NUMBER(char)

TO_DATE(char, ['fmt'])

转换函数即用来变换数据的类型，如把字符串按格式转换成日期型、数值型，八数值型、日期型转换成字符串等等。

在 to_date 的函数中，注意[fmt]的使用，区别 'yyyy'和 RRRR','RR','yy 等。

- **单行函数使用的例子：**

例：

SELECT last_name||':'||first_name, employee_id, manager_id

FROM s_emp

WHERE trunc(start_date) = to_date(2001-05-12','yyyy-mm-dd')

And length(last_name)<10

LESSON 3 多表数据查询

多表数据查询的首要条件是如何把多个数据表按一定的规则连接起来，从而获取多个表的、组织好的信息。连接的方式有多种，可按不同的要求进行选择。

- **连接**

```
SELECT   table.column, table.column
FROM     table1, table2
WHERE    table1.column1=table2.colmn2;
```

- **表别名**

```
SELECT   T1.column, T2.column... ...
FROM     table1      T1,
          table2      T2
WHERE    T1.column1=T2.colmn2;
```


Company Confidential - For internal use only

- **非等号连接**

  即不使用等号来连接两个或两个以上的表

  例如：

  使用：>=、<、between

- **外连接**

  对于两个表，如记录不能一一连接，而又不想漏掉数据，可使用外连接

```
SELECT   T1.column, T2.column... ...
FROM     table1      T1,
          table2     T2
WHERE    T1.column1=T2.colmn2（+）;
```

- **自连接**

  对于某些情况，必须要有表自身的连接，这时可利用表别名的作用，即把表完全看作两个不同的表来处理

  对于已连接好的多表数据查询，如同一张“ 大数据表 ”，进行类似的数据查询。

  例：

```
SELECT t1.item_name, t2.quantity, t3.last_name
FROM  t_item  t1,
        T_sales t2,
        T_emp  t3
WHERE t1.item_id=t2.item_id
    And  t2.emp_id=t3.employee_id
    And  t2.approval_flag=Y ’
```

LESSON 4 组函数

- **格式**

```
SELECT column, group_function
FROM  table
[WHERE condition]
[GROUP BY group_by_expr]
[HAVING group_condition]
[ORDER BY column……]
```

* GROUP BY: **对列进行分组**

* HAVING: **条件限制组查询**

- **常用组函数**

Company Confidential - For internal use only

AVG、COUNT、MAX、MIN

STDDEV、SUM、VARIANCE

组值函数如 sum、avg、stddev **等等，反映的是一组值的某种整体状况、一些统计信息。同时，要注意的是，组值函数忽略** NULL **值，在做计算时将其排除在外，如** avg **函数，如数据中含有较多的空值时，** avg **返回的统计信息偏差较大，而** count **函数出外，它是一种特殊情况，它可以处理** NULL **值，并总是返回一个数值，所求出的值决不会是** NULL**。**

例：

```
SELECT T1.name, T2.item_name, sum(T3.quantity), min(T3.rec_date)
FROM t_inventory    T1,
        T_item          T2,
        T_onhand        T3
Where  T1.warehouse_id=T3.warehouse_id
  And  T3.item_id=T3.item_id
GROUP BY T1.name, T2.item_name
HAVING sum(T3.quantity)>100
```

所有的组值函数都有一个 distinct **或** all **选项，缺省的选项为** all**，**count**函数就是一个说明起作用的例子：**

注意如下查询的结果：

```
SELECT count(distinct item_id), count(item_id), count(*)
FROM t_onhand
```

LESSON 5 子查询

- **格式**

```
SELECT      select_list
FROM        table
WHERE         expr      operator
                (SELECT  select_list
                 FROM   table)
```

注：

I. Expr **与子查询的** select_list **有相应的结构和数据类型**

II. **子查询无** order by **语句**

III. **子查询必须在** operator **的右面**

下名通过一些例子来说明子查询的应用。

Company Confidential - For internal use only

例：

```
SELECT t1.item_id, t1.quantity, t1.rec_date
FROM t_onhand  t1
WHERE rec_date = ( SELECT min(x.rec_date) FROM t_onhand  x
                   WHERE x.item_id = t1.item_id)
```

当然，很多时候子查询也可由组函数实现，比较一下下面查询的结果与前者的情况：

```
SELECT T1.item_id, T1.quantity, T1.rec_date
FROM t_onhand  T1,
     (SELECT item_id, min(rec_date)  rec_date  FROM t_onhand)  T2
WHERE T1.item_id = T2.item_id
  And   T1.rec_date = T2.rec_date
```

可用于 operator **的运算符很多，分为单行子查询和多行子查询，单行子查询即是在自查询的结果中，至多返回一行值，可用于单行子查询的逻辑运算符有：**

=、>、<、>= **或者**<=

多行子查询相对于单行子查询，即在子查询中可返回多行数据结果，可用于多行子查询的逻辑运算有：

in、not in

还有一种存在性相关子查询，即 exists **语句，它与** in **子查询相似，但有两点不同：**

i. **不能匹排一列或多列**

ii. **只能用于相关子查询**

例：

```
SELECT T1.item_name, T1.description
FROM t_item  T1
Where  exists ( SELECT * from t_sales
                WHERE item_id = T1.item_id)
```

UNIT3 数据库

单元培训目标

- **理解** Oracle **数据库的概念和作用；**
- **能创建常用的数据库对象（如表、索引、视图和序列）；**

Company Confidential - For internal use only

- 学会使用数据库字典，记住一些常用视图

LESSON 1 系统开发周期和数据关联

- 系统开发周期

通过使用系统开发周期策略，实现数据库从概念到产品的转变，系统开发周期包含多个开发过程和步骤。这种由上到下、系统的方法可把实际的业务需求变成可操作的数据库概念。

有如下步骤（流程）：

I. 分析，策略： 根据企业的实际业务和将要扩展的业务需求，分析所要提供的信息和企业及系统特性，建立相应系统模型；

II. 设计：数据库设计，把实体模型映射成数据库表之间的关联，外键关联和约束；

III. 开发和创建文档：开发业务流程，建立用户界面，操作流程，并创建相应文档（如帮助文档、操作手册等）

IV. 产品化：集成各模块功能，统一成一个完整产品，并做好各项测试准备；

V. 测试：功能测试、业务流程测试、出错流程测试最后的安装测试等。

在这几步步骤中，第二步尤为重要，直接影响开发的成败和效率，在第二步中，应用开发者重点要要考虑以下一些因数：

效率：最初的设计是最终设计的一个雏形，要充分考虑业务的效率；

系统集成：系统的多个模块间不是孤立的，应有紧密的结合，这种结合并非看起来或感觉上的，是从逻辑上和业务流程上的角度来分析的；

与其他系统的集成：企业与运作时，往往有多个系统的存在，但同时各个系统间是有结合的、联系的，如数据上、业务上的，同时还要考虑未来业务需求的标准接口等等；

- 数据关联

系统开发周期第二步的重点工作就是建立逻辑模型，逻辑模型是业务数据的非常规范化的流程图。懂得数据分解的原因和方法是理解模型的基础，而模型又是建立长期支持业务应用程序的基础。

规范化的过程通常用“ 范式“ 一词来讨论，最常见就是第一、二、三范式，其中第三范式是最高的规范化层次。

I. 第一范式(1NF)：具体做法就是将每一个表中具有相似类型的数据存入不同的表中，并给每个表确定一个主关键字（即是：Primary Key）----- 唯一的标号或标示符，消除重复数据；

II. 第二范式(2NF)：即是表中数据列只依赖主关键字；

III. 第三范式(3NF)：即是找出不只是依赖于主关键字的说有数据列。

- 约束

Company Confidential - For internal use only

为实现数据结构的规范化标准，即是各级范式，可以在定义表结构时，定一各个列和表的约束，从而严格表中的数据，格式如下：

数据列级约束：

```
Column [ CONSTRAINT constraint_name ] constraint_type
```

数据表级约束：

```
Column [ CONSTRAINT constraint_name ] constraint_type
```

约束主要要以下几种：

I. Primary Key(PK)**：由一列或多列构成的唯一标示各行的列；**

II. Foreign Key(FK)**：即由表中一列或列组合来映射同一表或别的表的** PK **或** UK(unique key);

III. NOT NULL

IV. UNIQUE

V. CHECK

对于建立约束的例子，可在建立数据表中参考。

LESSON 2 创建表、序列、视图和索引

- **创建表格式**

```
CREATE TABLE [schema.]table
(column  datatype  [DEFAULT expr] [ column_constraint ],
……
 [table_constraint       ]);
```

Schema	**所有者标识**
Table	**列表名**
Column	**列名**
Datatype	**列的数据类型**
Column_constraint	**列约束**
Table_constr aint	**表约束**

例：

```
CREATE TABLE s_dept
(id                 NUMBER(7)
          CONSTRAINT s_dept_id_pk PRIMARY KEY,
Name                VARCHAR2(25)
          CONSTRAINT s_dept_name_nn NOT NULL,
Region_id           NUMBER(7)
          CONSTRAINT s_dept_region_id_fk REFERENCES
          S_region (id),
          CONSTRAINT s_dept_name_region_id_uk UNIQUE
          (name, region_id));
```


Company Confidential - For internal use only

- **创建序列格式**

```
CREATE SEQUENCE sequence
        [INCREMENT BY     n]
        [START WITH  n]
        [{MAXVALUEn | NOMAXVALUE}]
        [{MINVALUEn | NOMINVALUE}]
        [{CYCLE  |  NOCYCLE}]
        [{CACHE n  |  NOCACHE}]
```

说明：

标示	**说明**
Sequence	**建立的序列名**
INCREMENT BY	**序列数字间的间隔，缺省为** 1
START WITH n	**第一个序列值，缺生为** 1
MAXVALUE n	**序列的最大值**
NOMAXVALUE	**确定序列的最大值，为** 10 **的** 27 **次方**
MINVALUE n	**序列的最小值**
NOMINVALUE	**确定序列的最小值，为** 1
CYCLE	NOCYCLE	**确定序列值达到最大值时，是否重复开始，缺省为** NOCYCEL
CACHE n	NO CACHE	**确定数据库管理器于值和存储的值的个数，缺省为** NOCACHE

序列常用来给一个表的ＰＫ列赋值

序列有 NEXTVAL **和** CURRVAL **方法。**

例：

```
CREATE SEQUENCE t_item_s
        Increment by 1
        Start wih   1
        Maxvalue 999999
        Nocache
        Nocycle;
```

是运行如下命令，看看结果：

```
select t_item_s.nextval from dual;
```


Company Confidential - For internal use only

select t_item_s.currval from dual;

- **创建视图**

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW view
        [(alias[,alias]…)]
AS      subsequery
[WITH CHECK OPTION [constraint]]
[WITH READ ONLY]
```

注：

I. Subquery **可以是非常复杂的** select **语句**

II. Subquery **不能包含** ORDER BY **语句**

实际上，视图是一个基于数据表或其他视图的逻辑数据表，一个视图本身并不包含数据，仅仅是一个可以用来查看数据或更新的窗口，一个视图在数据字典中是以一个 select **语句存储的。**

使用视图有着许多优点：

i. **限制数据表的访问：由于它是一条** select **语句，通过选择的取舍，使用户只能看到或只需看到所要检索的数据列；**

ii. **允许用户通过一个视图检索多个表的信息，而不要去写复杂的** SQL**语句；**

- **创建索引**

I. **自动创建**

在创建表时，若包含 Primary Key **或者其他唯一约束时，一个唯一索引就自动创建立了**

II. **手动创建**

用户可以创建非唯一的索引来提高检索速度

```
CREATE INDEX index
ON table (column[, column]…);
```

III. **何时建立索引：**

列经常在 where**或连接中使用；**

列包含了一个大范围的值；

列包含了大量的空值；

两个或多个列常常在 where **或连接中一起被使用；**

表的数据量大，而常常检索其中的 2-4% **；**

太多的索引并不总是能提高检索速度。

Company Confidential - For internal use only

LESSON 3Oracle 数据字典

Oracle**数据字典是** Oracle **数据库管理系统的重要组件之一，它包含一系列数据表和数据视图，提供只读属性的数据库信息。**

Oracle**数据字典实在数据库建立的时候建立的，又** Oracle **数据库管理器进行更新和维护，提供** Oracle **数据库用户、各个用户的权限、数据库对象、数据表约束等等信息。**

- Oracle**数据字典的四大类视图：**
  - user **用户拥有的对象**
  - all **用户可访问对象**
  - DBA **所有数据对象**
  - v$ **服务器性能对象**
- **其他视图：**
  - DICTIONARY **显示所有数据字典的数据表、视图和同义词**
  - TABLE_PRIVILEGES **显示数据对象的所有者、授权者和权限**
  - IND User_indexes **的同义词**
- **常用视图**

  DICTIONARY、TABLE_PRIVILEGES、IND

  ALL_TABLES、USER_OBJECTS

数据字典使用示例：

例一：
```
SELECT distinct object_name,
                object_type
  FROM usr_objects
```

例二：
```
SELECT *
  FROM user_constraints
  WHERE table_name = T_ITEM '
```

LESSON 4用户访问控制：

在一个多用户数据库环境，必须维护数据访问和使用的权限，Oracle **数据库管理器提供以下安全机制控制：**

I. **控制数据库的访问；**

II. **提供数据库对象的局部访问；**

III. **通过数据字典验证赋予和接收特权；**

IV. **为数据库对象创建同义词。**

Company Confidential - For internal use only

数据库安全分为两类，一是系统安全，二是数据安全。系统安全控制数据库的访问和使用这一层次，例如用户和密码，用户能使用的数据空间，用户允许使用的操作系统等等；数据库安全控制数据库对象的访问和使用，以及能在这些数据库对象上能进行的操作。

权限的概念即是执行特定 SQL**语句的权利。数据库的系统管理员具有最高的权限，普通用户必须有系统管理员赋予权限来访问数据库和数据库对象，当然，用户也可具有赋予其他用户或角色的权利。**

系统权限：

DBA **具有最高级系统权限，典型的系统权限有：**

系统权限	**操作授权**
CREATE USER	**赋予创建其他** Oracle **用户权限**
DROP USER	**删除任何用户**
DROP ANY TABLE	**删除任何束中的表**
BACKUP ANY TABLE	**备份任何束中的表**

创建用户格式：

```
CREATE USER user  ID ENTIFIED    BY password
```

系统权限授权格式：

角色：即是一组相似的权限，可以用来同一授权给一些用户。这样把相似的权限统一起来，易于给用户授权和维护系统权限，同时，一个用户也可以被授予多个角色权限。

用例如下：

```
CREATE ROLE role;
GRANT create table, create view TO manager;
GRANT manager TO userA, userB;
```

```
GRANT privilege [, privilege ... ... ]  TO  user  [, user ... ... ]
```

对象权限：

注意以下几点：

1. DBA **可授权给用户，对特定的对象执行某项操作；**
2. **对象权限因对相的不同类型而有区别；**
3. **对象的所有者拥有对象的所有权限；**
4. **对象的所有者可以对对象授予特定的权限。**

格式如下：

Company Confidential - For internal use only

```
GRANT { object_priv (, object_priv ... ) | ALL } [ ( columns ) ]
ON object
TO { user [, user ... ] | role [PUBLIC]
[WITH GRANT OPTION]
```

例：

```
GRANT update ( last_name, first_name, address )
ON t_emp
TO manager
```

在 Oracle 系统中，可通过如下数据字典查看用户权限情况：

数据字典表	说明
ROLE_SYS_RPIVS	授予角色的系统权限
ROLE_TAB_PRIVS	授予据色的表权限
USER_ROLE_PRIVS	用户可使用的角色
USER_TAB_PRIVS_MADE	用户对象所授予的对象权限
USER_TAB_PRIVS_RECD	用户所授予的对象权限
USER_COL_PRIVS_MADE	用户对象列所授予的对象权限
USER_COL_PRIVS_RECD	用户在特定的列上所授予的权限

权限的回收：

通过使用 revoke 命令可以收回授予其他用户的权限，同时授予用户的 with grant option 也会收回。

格式如下：

```
REVOKE {privilege [, privilege ... ] | ALL }
ON object
FORM { user [,user ... ] | role | PUBLIC }
[CASCADE CONSTRAINTS]
```

例：

```
REVOKE select, insert
ON S_EMP
FORM manager;
```

创建同义词：

为了引用其他用户的某个表，必须使用这个表的全称，即是用户名加表名，通过使用同义词就可避免这个麻烦，可以引用其他用户的表、视图、序列、过程或是其他别的对象。

格式如下：

```
CREATE [PUBLIC] SYNONYM synonym
FOR object;
```


Company Confidential - For internal use only

说明：

PUBLIC	**所有用户都可访问的同义词**
Synonym	**创建的同义词名**
Object	**同义词的引用源**

例：

```
CREATE SYNONYM  T_EMP
FOR    scott.t_tmp;
```

UNIT4 PL/SQL

单元培训目标

- **掌握** PL/SQL**常用语法；**
- **使用** Oracle Procedure Builder**；**
- **能创建包、函数和过程解决一些复杂问题；**
- **掌握出错流程处理**

LESSON 1 简单 PL/SQL

PL/SQL **是** Oracle **扩展的、具有程序设计特色的程序语言，它可以把数据维护和数据检索容入在程序过程当中。**

- PL/SQL **结构**

```
DECLARE  -可选项
           - 变量，常数，游标，用户定义的例外
BEGIN    - Mandatory
           - SQL 语句
           - PL/SQL 控制语句
EXCEPTION  – 可选项
           - 例外处理
END;
```

Declarative 【optional】**：包括所有变量、常量、游标和用户定的例外等**

Executable [Mandatory]**：包括各种** SQL **代码和** PL/SQL **代码等**

Exception Handling [optional]**：对各种例外情况的处理**

通常，每一个 PL/SQL **单元包含一个或多个块结构，这些块之间可以是完全独立的，也可以是一个块在另一个块里面。即是，一个块又能被分为若干个更小的块。**

下面是一些典型的 PL/SQL **程序结构：**

Company Confidential - For internal use only

程序结构	说明	使用位置
匿名块	没有特定名称的 PL/SQL 块，存在于 APPLICATION 中	所有 PL/SQL 环境
数据库 procedure、function	命名 PL/SQL块，能接受参数和重复调用	Oracle database Server
Application Procedure、function	命名 PL/SQL块，能接受参数和重复调用	Developer 2000 的组件，例如 FORMS
Package	命名 PL/SQL模块，包含相关的 Procedure、function 和变量	Oracle database Server
Database trigger	与数据库表联系在一起的 PL/SQL 块，能自动触发	Oracle database Server
Application trigger	与 Application 事件联系在一起的 PL/SQL块，能自动触发	Developer 2000 的组件，例如 FORMS

- **匿名块，过程和函数**

Ⅰ．匿名块

```
[DECLARE]
BEGIN
        -- 语句
[EXCEPTION]
END:
```

II．过程

```
PROCEDURE name
is
BEGIN
        -- 语句
[EXCEPTION]
END;
```

Ⅲ．函数

```
FUNCTION name
RETURN datatype
IS
BEGIN
  -- 语句
  RETURN VALUE
[EXCEPTION]
END;
```


Company Confidential - For internal use only

- PL/SQL 环境

  PL/SQL是一项绑定在 Oracle 数据库服务器上的技术，通过 PL/SQL 引擎进行传输和处理。

  当一段 PL/SQL 块从 Pro* 程序、user-exit、SQL*Plus 或是 Server Manager 中提交后，Oracle数据库服务器上的 PL/SQL 引擎就开始处理他们。PL/SQL引擎首先把块中的 PL/SQL 代码分成独立的若干块，在把他们传送到 SQL 语句执行器。这也意味着尽量采用少的 PL/SQL 块传输到 Oracle 数据库服务器，从而提高客户-服务器网络构造工作的高效性。

  在许多 Oracle 工具中，包括 Developer/2000，拥有他们自己的 PL/SQL 引擎，这是完全独立 Oracle 数据库服务器，且在之前的引擎。

- Oracle Procedure Builder 的使用

  结构化程序设计的最大有点在于能快速、简单的创建和调试代码，Procedure Builder 提供所有必要的功能，以利于成功开发和调试 PL/SQL程序。

  组要构件：

组件	说明
对象浏览器	管理 PL/SQL 结构，执行调试
PL/SQL解释执行器	调试 PL/SQL 代码，执行 PL/SQL代码
程序编辑器	创建和编辑 PL/SQL 代码
数据库程序编辑器 数据库触发器编辑器	创建和编辑服务器端 PL/SQL 代码 创建和编辑数据库触发器

  对象浏览器：

  Program Unit: 能独立被 PL/SQL编译器识别和处理的 PL/SQL结构；

  Program Unit –Sepcification: 程序单元的名称、参数和返回类型；

  Program Unit –Referances: 程序单元引用的过程、函数、匿名块和表；

  Program Unit – Referanced By: 引用程序单元的过程、函数、匿名块和表；

  Libraries: 文件或数据库中PL/SQL 包、过程和函数的集合；

  Attached Libraries: 引用的数据库或文件系统中的库；

  Built-in Packages : 在程序调试中可引用的 PL/SQL 结构单元；

  Debug Actions:在程序单元调试中，可进行的监测、中断的命令；

  Stack: 子程序调用堆栈

  Database Objects: 服务器端存储的过程、库、表和视图。

  PL/SQL解释执行器:

Company Confidential - For internal use only

通过使用 PL/SQL 解释执行器，可以定义、显示、调试和运行 PL/SQL 程序，同时，在 PL/SQL 解释执行器中，可以直接运行过程，以分号结束，如下：

```
Procedure_name  [(argument1  [, argument2,  ... .])];
```

在 PL/SQL 解释执行器中，也可执行 SQL DDL 语句和其他命令语句。

程序编辑器：

1. **创建一个新程序单元：**

   i. **选中程序单元；**

   ii. **点击创建按钮，弹出程序对话框窗口；**

   iii. **出入新建程序单元名称，并选择程序类型，点击 OK 键；新建程序单元显示在对象浏览器上；**

   iv. **在打开的程序单元内，编辑。**

2. **程序编译**

   点击程序编辑窗口中的 compile 按钮，编译程序单元，编译信息在程序编辑器下方显示，包括各种出错信息或编译成功信息。

调试数据库程序：

1. **总体步骤：**

i. **把数据库程序单元载入到对象浏览器中；**

ii. **选中要调试的程序单元；**

iii. **用鼠标将其拖到程序单元中；**

iv. **进行调试工作；**

v. **用鼠标间调试好程序单元拖回到原数据库单元中。**

调试方法：

在 PL/SQL 解释执行器中调试程序，可以双击代码行，设置断点（可以设置多割断点）；同时，还可以设置局部变量和参数，如下：

```
(debug1)PL/SQL>      debug.seti(1,3)
```

- **程序中的参数和变量类型**

I. **参数：**

   参数有如下三种类型：

   IN Argument

   OUT Argument

Company Confidential - For internal use only

IN OUT Argument

在参数格式中，IN **可以省略，它也是缺省的形式；**IN **代表该参数用于向程序中传递值，**OUT **代表由程序单元返回给调用它的函数值，而对于** IN OUT **型参数，代表既可向子程序传递值，同时子程序也可返回值给主程序。**

II. **变量类型**

通用数据类型：

数据类型	**说明**
BINARY_INTEGER	**基本数值整型，**-2147483647 ----- 2147483647
NUMBER[(precision,scale)]	**基本浮点数值型**
CHAR[(maximun_length)]	**固定长度的字符型，最大值为** 32760
LONG	**可变常字符型，最大长度为** 32760
LONG RAW	**二进制型，最大长度为** 32760
VARCHAR2(maximum_length)	**可变长字符型，最大长度为** 32767
DATE	**日期和时间类型**
BOOLEAN	**逻辑型（**TRUE、FALSE **或** NULL**）**

程序单元（过程、匿名块和函数）中变量的声明：

```
Identifier  [CONSTANT] datatype [NOT NULL]
            [:= | DEFAULT expr];
```

注意：具有初始值的常量和变量被认为是 NOT NULL **的，一行最多只有一个常量或是变量的声明**

%TYPE类型：

这是一种动态数据类型，在 PL/SQL **中动态决定器数据类型。它有两大优点：**

1. **决定其数据类型的数据库表列的类型未知；**
2. **决定其数据类型的数据库表列的类型在运行时是可变的。**

为了在 PL/SQL **中保存某个列的数值，必须确保变量与列具有相同的数据类型和精度，否则，常会造成** PL/SQL**错误或是精度的丢失。而** %TYPE **类型是由某一个预定的变量或时数据库表中列决定的，这样就可以确保数据类型的一致。**

例：

Company Confidential - For internal use only

```
...
        l_last_name         t_emp.last_name%TYPE
        l_salary            t_emp.base_salary%TYPE
...
```

复合数据类型：

1. PL/SQL TABLES：

它有两部分组成：I) BINARY_INTEGER **型的** Primary Key;

II) **数据的列范围。**

注意：

i. PL/SQL TABLES **数据类型并不等同于数据库中的表；**

ii. PL/SQL TABLES **数据类型类似一个一维数组；**

iii. PL/SQL TABLES **类型包含两个组件；**

iv. PL/SQL TABLES **类型可动态增长。**

声明如下：

```
TYPE type_name IS TABLE OF scalar_datatype
        [NOT NULL] INDEX BY BINARY_INTEGER;
identifier  type_name;
```

例：

```
...
TYPE name_table_type  IS TABLE OF VARCHAR2(25)
      INDEX BY  BINARY_INTEGER;
First_name_table      name_table_type;
...
first_name_table(1):= 'FRANCK';
```

2. PL/SQL RECORDS：

它包含一个或是一个以上的任何范围、记录、PL/SQL TABLES **类型的组件。**

注意：

i. PL/SQL RECORD **数据类型并不等同于数据库中的表中的一条记录；**

v. PL/SQL RECORD **数据类型类似一个多维数组；**

vi. PL/SQL RECORD **类型包含多个组件；**

vii. PL/SQL RECORD **类型可动态增长；**

viii. PL/SQL RECORD **类型方便地从一个表中获取一行数据，以便于** PL/SQL **的处理。**

Company Confidential - For internal use only

声明如下：

```
TYPE type_name IS RECORD
            (field_name1   field_type
            [NOT NULL { := | DEFAULT} expr],
            (field_name2   field_type
            [NOT NULL { := | DEFAULT} expr], ... );
identifier   type_name;
```

例：

```
...
TYPE emp_record_type  IS RECORD
        (last_name      varchar(25),
         base_salary    number,
        manager_id      number);
employee_record       emp_record_type;
...
employee_record.last_name:= 'FRANCK';
```

LESSON 2 数据获取和游标

- **数据获取**

```
SELECT   select_list
INTO                variable_name | record_name
FROM                table
WHERE   condition;
```

I. INTO **语句时必须的**

II. Select **只有一行数据返回**

III. Select_list **不能为简写**(eg. *)

- **游标**

I. **显式游标**

II. **隐式游标**

例

SQL%ROWCOUNT **检索行数**

SQL%FOUND Boolean **型**

SQL%NOTFOUND Boolean **型**

SQL%ISOPEN

Company Confidential - For internal use only

LESSON 3 PL/SQL 控制流

- IF 语句

```
IF condition            THEN
   语句；
[ELSEIF condition THEN
            语句；]
[ELSE
            语句；]
END IF;
```

- LOOP 语句

```
（1）
LOOP
            语句；
            EXIT [ WHEN condition];
End loop

（2）
FOR index in [REVERSE]
            lower_bound..upper_bound LOOP
            语句
END LOOP
（3）
WHILE condition LOOP
            语句；
END LOOP;
```

LESSON 4 出错处理

- Exception 类型

  I. 数据库预定义

  II. 数据库非预定义

  III. 用户定义

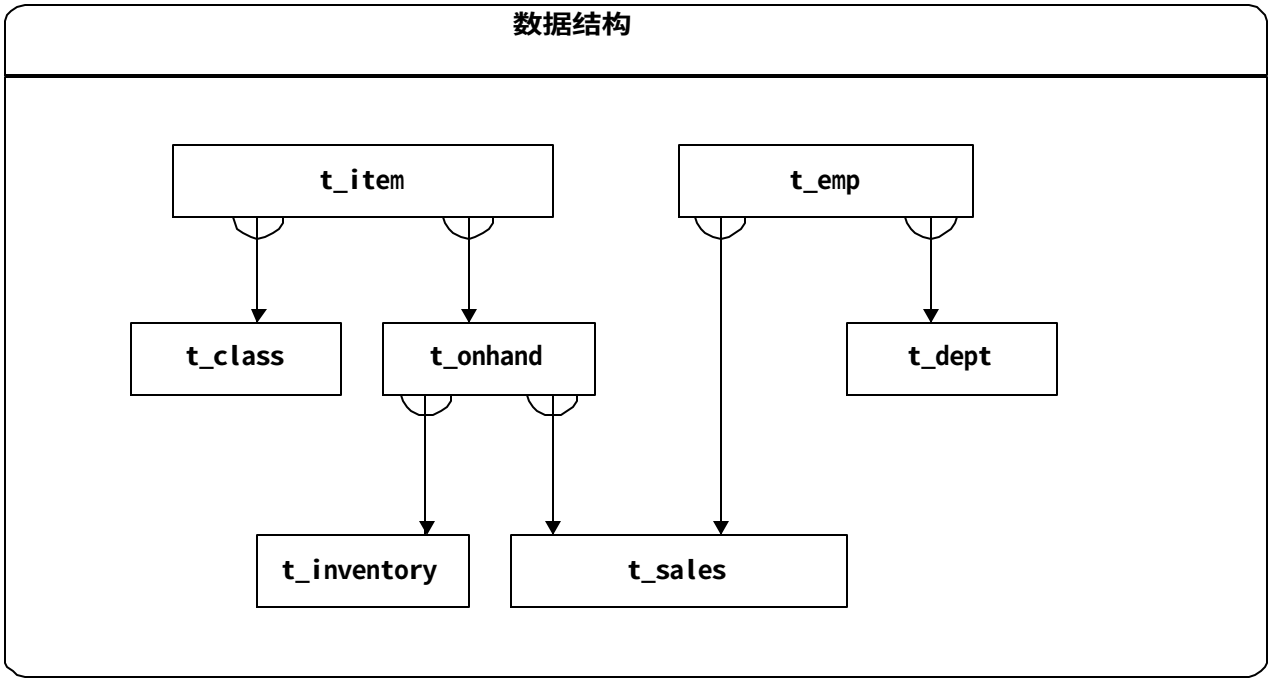
- Exception 处理

```
EXCEPTION
            WHEN exception1 THEN
            语句；
            WHEN exception2 THEN
                        语句；
      [WHEN OTHERS THEN
                        语句；]
```


Company Confidential - For internal use only

附录 A:

数据结构：



描述：

t_item:

item_id	number	PK,
item_name	varchar2(20)	NOT NULL,
base_price	number,	
description	varchar2(100),	
class_id	number	FK

t_inventory:

warehouse_id	number	PK,
name	varchar2(30)	NOT NULL,
description	varchar2(100)	

t_onhand:

warehouse_id	number	FK,

Company Confidential - For internal use only

item_id	number	FK,
quantity	number	NOT NULL,
rec_date	date	NOT NULL

t_sales:

transaction_id	number	PK,
item_id	number	FK,
creation_date	date	NOT NULL,
quantity	number	NOT NULL,
price	number,	
emp_id	varchar2(20)	FK,
approval_flag	varchar2(1),	
shipped_flag	varchar2(1),	
cancel_flag	varchar2(1)	

t_emp:

employee_id	number	PK,
last_name	varchar2(10)	NOT NULL,
first_name	varchar2(10)	NOT NULL,
address	varchar2(100),	
start_date	date	NOT NULL,
department_id	number,	
manager_id	number,	
salary	number	

t_dept:

department_id	number	PK,
name	varchar2(20)	NOT NULL,
description	varchar2(100)	

Company Confidential - For internal use only

t_class:

class_id	number	PK,
name	varchar2(20)	NOT NULL,
description	varchar2(100)	

Company Confidential - For internal use only