Oracle 从入门到精通

一、SQL

1.1、基本概念：

数据库对象：
1. 表
2. 约束条件：保证数据完整性。
3. 视图：虚表，命名的查询语句。
4. 索引：加速查询（加快查询的速度）。
5. 序列：一串连续递增或递减的数字，步长相同，（代理键）。
6. 同义词：一个对象的另外一个叫法（对象的别名）。
7. 存储过程：用于操作
8. 函数：用作复杂运算的。用于计算。
9. 触发器：由事件触发的存储过程。
10. 包

1.2、数据库安全：

1、用户
2、方案或模式(Schema)：是用户所对应的对象的集合。用户名等于方案名
3、权限
4、角色：权限组，一组权限。
5、配额(quota)：允许被使用的空间。用户可以在表空间上可以使用的空间。
端口：2030
环境变量
-ORACLE_BASE 基本目录
-ORACLE_NAME 当前的主目录
-ORACLE_NLS33 使用 US7ASCLL 字符集时不用设
-PATH 路径

1.3、基本的 SQL SELECT 语句

口令中的第一个字符不能为数字。
语句类型：
（一）查询：SELECT
数据操作语句：DML（数据的插入 INSERT、删除 DELETE、修改 UPDATE、合并 MERGE）
（二）合并：把一个表中的数据合并到另一个表中去，如果数据在原表中存在做 UPDATE，否则 INSERT（9I 独有）。
（三）事务控制语句：COMMIT 提交、ROLLBACK 回滚、
SAVEPOINT 存储点（与 ROLLBACK 搭配使用）在回滚的时候可以回滚到某个存储点上。否则回滚到最初起点上。
（四）数据定义语句：对对象操作。TRUNCATE 清除表中所有数据 /CREATE 创建 /DROP 删除 /ALTER

修改
（五）权限控制语句（DCL）：GRANT 授予权限 /REVOKE 移除权限

1.4、SELECT 语句

1、语法：

SELECT 查询列表 FROM 数据源；
& SQL 命令必须加分号。
ALTER USER HR IDENTIFIED BY HR ACCOUNT UNLOCK;
修改用户 解锁
给 HR 解锁
CONNECT(conn) HR/HR（密码） 用 HR 用户连接数据库。
&（不是 SQL 命令 是 SQLPLUS 命令）不用加分号
DESCRIBE(desc) DEPARTMENTS
关键字 表名
描述表命令（SQLPLUS 命令）

2、SQL 语句说明：

（1）语句文本的书写不区分大小写。（但字符串在作为值的时候要注意大小写）
（2）语句可以写单行也可以写多行。
（3）关键字不能缩写或跨行。
（4）语句通常被分多行书写。
（5）缩进被用于提高语句的可读性。

3、数字和日期都可以使用数学运算符建立表达式。

+，-，*，/ <>不等于
日期可以加减数字，数字默认为天。
日期不能加日期，但日期可以减日期。
字符不能加减。

4、定义空（NULL）值

空值出现在表达式中会导致整个表达式的值为空。
NVL（字段名，将要赋予的值）函数
作用：将空值转换成其他有 ASCLL 码的值。
annual_salary 年薪

5、别名

可以加中文的字段别名。
如果想强制地改变列名的大小写，可以在别名的定义时加上双引号,列名有空格时也要在列名上加双引号。
例：select lastname as "employees name" from employees;

6、spool +路径

保存命令（将显示保存）

7、连接操作符： ||

```
select lastname || 'work in' || department_id from tablename;

select last_name || '''s salary is ' || salary 员工月薪 from employees;
                    ~~~
```

注：在单引号中还要使用单引号的话，就必须使用两个单引号来实现一个单引号的功能。

8、文本字符串

*可以代表字符、数字或是日期。
*当代表字符或日期的时候用单引号括起来，数字不需要。

9、DISTINCT

在查询时默认显示所有的行，包括有重复值的行。
DISTINCT 消除重复行关键字，放在整查询列表的最前面。
作用范围：整个查询列表的组合。
消除重复行后会按字段的特性，做升序排列。（执行过程：先排序，再消除重复）

```
select distinct department_id,job_id from employees;
```

1.5、SQLPLUS 与 SQL 的关系

SQL *是一种语言
*ANSI 标准
*关键字不能缩写
*用于操作数据库中的数据和表的定义

1、SQLPLUS 命令的功能：

*描述表的结构
*编辑 SQL 语句

2、查询 SQLPLUS 命令

```
help + 命令
```

3、SQLPLUSW 在 WINDOWS 下运行的分析器。

登陆 ISQLPLUS
（1）先到服务中启动 OracleOraHome92HTTPServer
（2）在浏览器中输入：http://wnj:7778/isqlplus
URL(网页中的地址)

4、SQLPLUS 命令：

4.1、与文件相关的命令：

4.1.1、 spool + 路径

.

.

spool off

4.1.2、save

把当前内存中的语句保存为文本文件。

4.1.3、run 或 /

运行当前内存中的语句

4.1.4、clear buffer（cl buff）

清空当前内存中的语句

4.1.5、start @ 读取并执行

4.1.6、get 读取不执行

4.2 编辑命令：

- List 列出一条语句

*表示当前行

- Change 修改命令

原来 c/jj/kk

c/jjj/xxx

- input 在当前行之后插入一行新的数据
- append 在当前行中插入新的东西

n 写数字显示对应行

- delete

del + 回车 删除当前行

del 1 3 删除第一到第三行

- edit l,c,i,a,n,d,e
- 查看当前用户

SHOW USER

- 默认的日期格式：DD-MON-RR 日-月-年
- 日期可以进行比较；
- 字符可以进行比较（以字母的 ASCLL 码比较）；
- IN（set）或 NOT IN 匹配任何列表中的值；
- LIKE 模糊匹配字符串值；
- IS NULL 是否空值；
- IS NOT NULL 是否不为空；
- BETWEEN 可以做数字、日期和字符的比较。
- 通配符 %、_

S_mith

WHERE first_name like 's/_%'escape'/';

解释这个符号后的下划线为正常的字符。如果不加，将被视为通配符作用的下划线。

'_'只能通配一个字符

主要用于通配固定位数的字符。例如查询月收入五位数以上的员工。五位就可以用'_'来查询。

- 逻辑操作符（用在 WHERE 子句中）
  1. AND
  2. OR
  3. NOT

先执行 NOT ，再执行 AND 最后执行 OR。

- ORDER BY

ORDER BY 子句在 SELECT 语句的最后。

- ASC：升序 DESC：降序
- 空值作为无穷大来处理。
- rownum 显示行数量约束的关键字（在结果中可以做代理键使用）；

可以按照查询列表中序号进行排序。

系统在用户写出查询列表的同时就赋予每个列名一个序号，升序赋予。

例： SELECT name,phone,adress from.....;

1 2 3

1.6、单行函数

单行函数：对单行数据进行计算并返回一个值的函数。

*修改数据项

*接受参数返回一个值。

*对每行进行操作。

*每行返回一个结果。

*可以修改数据类型

*可以嵌套

1、character 字符类型函数：

- LOWER()强制小写
- UPPER()强制大写
- INITCAP()每个单词首字母大写,可以用在 WHERE 子句中。
- CONCAT（‘’, ‘’）连接函数
- SUBSTR(string,a[,b])返回 string 的一部分，a 和 b 以字符为单位。
- SUBSTRB(string,a[,b])返回 string 的一部分，a 和 b 是以字节为单位。
- SUBSTRC(string,a[,b])返回 string 的一部分，a 和 b 是以 UNICODE 完全字符为单位。
- SUBSTR2(string,a[,b])返回 string 的一部分，a 和 b 是以 UCS2 代码点为单位。
- SUBSTR4(string,a[,b])返回 string 的一部分，a 和 b 是以 UCS4 代码点为单位。

以上函数都是返回 string 的一部分，从字符位置 A 开始，长为 B 个字符。如果 A 是 0，那它就被认为是 1（字符串的开始位置）。如果 A 是正数，那么字符从左边开始数。如果是负数，则从 STRING 的末尾开始，从右边数。如果 B 不存在，那么缺省是整个字符串。如果 B 小于 1，将返回 NULL。如果 A 或 B 使用了浮点数，那么该数值首先被节取成一个整数，返回类型与 STRING 相同。

- LENGTH(string)
- LENGTHB(string)
- LENGTHC(string)
- LENGTH2(string)

- LENGTH4(string)

以上函数返回 string 的长度。因为 CHAR 类型的值是填充空格的，所以如果 string 是 CHAR 数据类型，那么末尾的空格算在长度之内。如果 string 是 NULL，函数返回 NULL。

- INSTR(string1,string2[,a][,b]) 返回 string1 中包含 string2 的位置。a 和 b 以字符为单位。
- INSTRB(string1,string2[,a][,b]) 返回 string1 中包含 string2 的位置。a 和 b 是以字节为单位。
- INSTRC(string1,string2[,a][,b])返回 string1 中包含 string2 的位置。a 和 b 是以 UNICODE 完全字符为单位。
- INSTR2(string1,string2[,a][,b]) 返回 string1 中包含 string2 的位置。a 和 b 是以 UCS2 代码点为单位
- INSTR4(string1,string2[,a][,b]) 返回 string1 中包含 string2 的位置 a 和 b 是以 UCS4 代码点为单位。

以上函数返回 string1 中包含 string2 的位置。从左边开始扫描 string1,起始位置是 A。如果 A 为负数那么从右边开始扫描。第 B 次出现的位置将被返回。A 和 B 缺省都为 1，即返回在 string1 中第一次出现 string2 的位置。如果 string2 在 A 和 B 的规定下没有找到那么就返回 0。位置的计算是相对于 string1 的开始位置的，而不关 A 和 B 的取值。

- LPAD（列名,数字,‘要补上的字符’）左补位
- RPAD（列名，数字,‘要补上的字符’）右补位
- TRIM（‘child_str’ FROM ‘parents_str’）将连续子串（只能有一个字符）从主串的两边截取出来，区分大小写。默认为截取空格。
- LTRIM（）左截取
- RTRIM（）右截取
- ascii(x)函数，返回’X’字符的十进制数，即 X 的 ASCII 码值。
- chr(x)函数，返回 ASCII 码为 X 的字符。
- length(x)函数，求串 X 的长度，与之相似的是 lengthb(x)函数，用在多字节字符中。
- replace(x,y[,z])函数，返回值为将串 X 中的 Y 串用 Z 串替换后的结果字符串。若省略 Z 参数，则将串 X 中为 Y 串的地方删除。
- soundex(x)函数，返回串 X 的语音描述，这个描述由 4 个字符组成，说明串 X 的声音表示形式发音，有时在只知道一个名字的发音而不知道拼写情况下或许能用到。

例：select soundex(’smith’) from dual; 返回值为：S530.

- translate(x,y,z)函数，返回将 X 串中每个字符按它在 Y 串中出现的位置翻译成 Z 串中相应位置的字符后的结果，相当与替换。

例：select translate(’this is an example’,’my is’,’@#$%^&’) from dual;

- NLS 函数

除了 NCHR，这些函数都是以字符类型为参数返回字符类型值。

- CONVERT(string,dest_charset[,source_charset])

将输入 string 转换为指定字符集 dest_charset。source_charset 是输入值的字符集——如果它没有被指定，则缺省为数据库字符集。输入值可以是 CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB 和 NCLOB 类型。返回值为 VARCHAR2 类型。如果 dest_charset 中没有输入字符串中的一个字符，将会使用一个代替字符（由 dest_charset 定义）

- NCHR(X)

返回数据库国家字符集中值为 X 的字符。NCHR(X) 等价于 CHR(x USING NCHAR_CS).

- NLS_CHARSET_DECL_LEN(byte_width,charset)

返回一个 NCHAR 值的声明宽度(以字符为单位).byte_width 是该值以字节为单位的长度 charset 是该值的字符集 ID。

- NLS_CHARSET_ID(charset_name)

返回指定字符集 charset_name 的数字 ID。为 charset_name 指定“CHAR_CS”将返回数据库字符集的 ID，为 charset_name 指定“NCHAR_CS”将返回数据库国家字符集的 ID。如果 charset_name 是一个无效字符集名，将返回 NULL。NLS_CHARSET_ID 和 NLS_CHARSET_NAME 是互为反函数。

- NLS_CHARSET_NAME([charset_id])

返回指定字符集 ID charset_id 的名字。如果 charset_id 是一个无效字符集 ID，将返回 NULL;

- NLS_INITCAP(string[,nlsparams])

以字符串中每个单词第一个字符大写而单词中其余字母小写的形式返回 string.nlsparams 指定了一个与该会话缺省的不同的排序次序。如果没有指定该参数，NLS_INITCAP 与 INITCAP 相同。nlsparams 应该采取下面的形式：

'NLS_SORT=sort',其中 sort 是一个语言排序序列。

- NLS_UPPER(string[,nlsparams])

以大写形式返回 string,不是字母的字符不受影响。如果没有指定 nlsparams,NLS_UPPER 与 UPPER 相同。

- NLS_LOWER

以小写形式返回 string,不是字母的字符不受影响，如果没有指定 nlsparams，NLS_LOWER 与 LOWER 相同。

- NLSSORT(string[,nlsparams])

返回用于排序 string 的字符串字节。所有值都被转换为字节字符串，这样在不同数据库之间就保持了一致性。

如果没有指定 nlsparams,那么就会使用会话中缺省排序序列。

- TRANSLATE(string USING {CHAR_CSINCHAR_CS})

TRANSLATE...USING 将输入 string 参数转换为数据库字符集（指定 CHAR_CS）或数据库国家字符集（指定 NCHAR_CS）.string 可以是 CHAR、VARCHAR2、NCHAR、NVARCHAR2 类型。如果指定 CHAR_CS，返回类型为 VARCHAR2，如果指定 NCHAR_CS,返回类型为 NVARCHAR2。TRANSLATE...USING 是 CONVERT 功能的子集。

如果输入值包含 UCS2 字符或反斜线符号要使用 UNSTR 函数。

例：

```
SQL> select translate('asd' using NCHAR_CS) from dual;--数据库国家字符集
TRANSL
------
asd
SQL> select translate('asd' using CHAR_CS) from dual;--数据库字符集
TRA
---
asd
```

- UNSTR(s)

返回转换为数据库 UNICODE 字符集的字符串。s 可包含 escaped UCS2 代码点字符。它由一个反斜线符号加上十六进制代码点数字组成。因此，要在字符串中包含一个反斜线符号就必须使用双反斜线符号(\\).

UNISTR 与 TRANSLATE...USING 相似，差别是它仅能转换为 UNICODE，而且可以包含 escaped 字符。

```
general
//number
//conversion
//date
```

多行函数：对多行数据（一组数据）进行计算并返回一个值的函数。

2、number 数字类型函数

- ABS(x) 函数，此函数用来返回一个数的绝对值。
- ACOS(x)函数，返回 X 的反余弦值。X 范围从 1 到-1，输入值从 0 到派，以弧度为单位。
- ASIN(x)函数，返回 X 的反正弦值。X 范围从 1 到-1，输入值从-PI/2 到 PI/2，以弧度为单位。
- ATAN(x)函数，返回 X 的反正切值。输入值从-PI/2 到 PI/2，以弧度为单位。
- BITAND(x,y)函数,返回 X 和 Y 的与结果。X 和 Y 必须为非负整数。注意没有 BITOR 函数，但是在 UTL_RAW 包中有用于 RAW 值的位操作符。
- CEIL(x)函数，用来返回大于或等于 X 的最小整数。
- COS(x)函数，返回 x 的余弦值。x 是以弧度表示的角度。
- COSH(x)函数，返回 X 的双曲余弦。
- EXP(x)函数，与 power(x,y)函数类似，不过不用指明基数，返回 E 的 X 次幂。E=2.71828183...
- FLOOR(x)函数，用来返回小于或等于 X 的最大整数。
- LN(x)函数，返回 x 的自然对数。x 必须大于 0。
- LOG(x,y)函数，返回以 X 为底 Y 的对数。底必须是不为 0 和 1 的正数，Y 是任意正数。
- MOD(被除数，除数)求余函数，如果除数为 0，则返回被除数。
- POWER(x,y)函数，返回 X 的 Y 次幂。底 X 和指数 Y 都不必是正整数，但如果 X 是负数的话，Y 必须是整数。
- ROUND(x[,y])函数，返回舍入到小数点右边 Y 位的 X 值。Y 缺省为 0，这将 X 舍入为最接近的整数。如果 Y 是负数，那么舍入到小数点左边相应的位上，Y 必须为整数。
- SIGN(x)函数，此函数用来返回一个数的正负值，若为一个正数则返回 1，若为一个负数则返回-1，若为 0 则仍返回 0，有点像把模拟量数字化的意思。
- SIN(x)函数，返回 X 的正弦。x 是以弧度表示的角度。
- SINH(x)函数，返回 x 的双曲正弦。
- SQRT(x)函数，返回 x 的平方根，x 不能是负数。
- TAN(x)函数，返回 x 的正切。x 是以弧度表示的角度。
- TANH(x)函数，返回 x 的双曲正切。
- TRUNC(x[,y])截取值函数，Y 缺省为 0，这样 X 被截取成一个整数。如果 Y 为负数，那么截取到小数点左边相应位置
- WIDTH_BUCKET(x,min,max,num_buckets) 只能在 SQL 语句中使用。

使用 WIDTH_BUCKET 可以根据输入参数创建等长的段。范围 MIN 到 MAX 被分为 num_buckets 节，每节有相同的大小。返回 X 所在的那一节。如果 X 小于 MIN，将返回 0，如果 X 大于或等于 MAX，将返回 num_buckets+1.MIN 和 MAX 都不能为 NULL，num_buckets 必须是一个正整数。如果 X 是 NULL，则返回 NULL。

3、时间类型函数：（date）

内部存储格式：世纪、年、月、日、小时、分钟、秒
默认格式是：DD-MON-RR。
SYSDATE 返回当前的系统时间。
SELECT SYSDATE FROM DUAL;

3.1、对日期的数学运算

SELECT （SYSDATE-HIRE_DATE）/7 FROM TABLENAME WHERE ROWNUM;

SYSDATE-HIRE_DATE:数字列

ADD_MONTHS(date,x)函数，返回加上 X 月后的日期 DATE 的值。X 可以是任意整数。如果结果的月份中所包含的 日分量少于 DATE 的月份的日分量，则返回结果月份的最后一天。如果不小于，则结果与 DATE 的日分量相同。时间分量也相同。

CURRENT_DATE 以 DATE 类型返回会话时区当前的日期。这个函数同 SYSDATE 相似，除了 SYSDATE 不管当会话时区。

CURRENT_TIMESTAMP[(precision)] 以 TIMESTAMP WITH TIMEZONE 类型返回会话时区当前的日期。如果指定 precision，它指返回秒数的精度，缺省为 6。

DBTIMEZONE 返回数据库的时区。

LAST_DAY（日期） 指定日期所在月份的最后一天的日期，这个函数可用来确定本月还有多少天。

LOCALTIMESTAMP[(precision)] 以 TIMESTAMP 类型返回会话时区的当前日期。如果指定 precision，它指返回秒数的精度，缺省为 6 。

MONTHS_BETWEEN(离当前比较近的日期 date1,以前的日期) 两个日期之间相差的月数（以日作为最小单位来计算的）。返回是相差的月数。如果 date1 和 date2 的日分量相同，或者这两个日期都分别是所在月的最后一天，那么返回结果是个整数。否则，返回结果包含一个分数，以一个月 31 天计算。

NEW_TIME(d,zone1,zone2)函数，当时区 zone1 中的日期和时间是 D 的时候，返回时区 zone2 中的日期和时间。返回类型为 DATE。zone1 和 zone2 是字符字符串，另外的时区可在 ORACLE9I 中通过查询 V$TIMEZONE_NAMES 得到。

NEXT_DAY （日期，星期几） 指定日期后将要遇到的后七天的某一天的日期。

ROUND（日期，‘MONTH/YEAR’） 四舍五入得到新的日期。 保留位置是月和年

SESSIONTIMEZONE 返回当前会话的时区。返回类型是一个时区偏移或时区片名的字符字符串。如果指 定格式，则与 ALTER SESSION 语句中的格式相同。

SYS_EXTRACT_UTC(datetime) 从提供的 DATETIME 中以 UTC(Coordinated Universal Time)返回时间。DATETIME 必须包含一个时区。

SYSTIMESTAMP 以 TIMESTAMP WITH TIMEZONE 返回当前的日期和时间。当在分布式 SQL 语句中使用的时候，返回本地数据库的日期和时间。

TRUNC（日期，‘MONTH/YEAR’） 截取

TZ_OFFSET(timezone) 以字符字符串返回提供的 timezone 和 UTC 之间的偏移量。timezone 可以被指定为时区名或’+/-HH:HI’格式表示的偏移量。也可使用 SESSIONTIMEZONE 和 DBTIMEZONE 函数，返回格式为’+/-HH:HI’。

字符字符串	时区
AST	大西洋标准时
ADT	大西洋夏令时
BST	白令标准时
BDT	白令夏令时
CST	中央标准时
CDT	中央夏令时
EST	东部标准时
EDT	东部夏令时
GMT	格林威治平均时
HST	阿拉斯加夏威夷标准时

HDT	阿拉斯加夏威夷夏令时
MST	Mountain 标准时
MDT	Mountain 夏令时
NST	纽芬兰标准时
PST	太平洋标准时
PDT	太平洋夏令时
YST	YuKon 标准时
YDT	YuKon 夏令时

3.2、日期和日期时间算术

运算 返回类型
结果

d1-d2 NUMBER
返回 D1 和 D2 之间相差的天数。该值是一个数值，其小数部分代表一天的几分之几。
dt1-dt2 INTERVAL
返回 DT1 和 DT2 之间的时间间隔。
i1-i2 INTERVAL
返回 i1 和 i2 之间的差距。
d1+d2 N/A
非法——仅能进行两个日期之间的相减。
dt1+dt2 N/A
非法——仅能进行两个日期之间的相减。
i1+i2 INTERVAL
返回 i1 和 i2 的和。
d1+n DATE
在 D1 上加上 N 天作为 DATE 类型返回。N 可以是实数，它包含一天的几分之几。
d1-n DATE
从 D1 上减去 N 天作为 DATE 类型返回。N 可以是实数，它包含一天的几分之几。
dt1+i1 DATETIME
返回 DT1 和 I1 的和。
dt1-i1 DATETIME
返回 DT1 和 I1 之间的差距。
i1*n INTERVAL
返回 I1 的 N 次方。
i1/n INTERVAL
返回 I1 除以 N 的值。

表中注：
D1 和 D2 指日期值；
DT1 和 DT2 指日期时间值；
I1 和 I2 指时间间隔值；
N 指数字值。

3.3、显示转换：（conversion）

TO_NUMBER(char[,'format_model']) 字符转换到数字类型
TO_DATE(char[,'format_model']) 字符转换到日期类型
格式说明符：要与前边要转换的字符串的格式要相同才能转换（匹配问题：格式和位数）。
TO_CHAR(date[,'format_model'[,nlsparams]])
第二个参数可以省略，不指定格式，按系统默认格式输出。
区分大小写。
使用 FM（在格式控制符前添加）符号可以去掉空格或是首位的零。
如果指定了 NLSPARAMS，则它控制返回字符串的月和日分量所使用的语言。格式为：
'NLS_DATA_LANGUAGE=language',language 指需要的语言。
例：select to_char(sysdate,'FMyyyy-mm-dd') from dual;

5.格式控制符的类型：

YYYY 四位的年
YEAR 年的拼写
MM 2 位数字的月
MONTH 月的全名
MON 月名的前三个字符
DY 星期名的前三个字符
DAY 星期名的全称
DD 2 位的天

6.时间格式控制符：

HH24:MI:SS AM
HH12:MI:SS PM

7.通过“”来实现加入特殊字符的格式控制符。
SELECT TO_CHAR(SYSDATE,'FMyyyy"年"mm"月"dd"日"') from dual;
DDSPTH
~~
DD 是格式控制符。
TH 是序数词，将日期转换成英文的序数词拼写。
SP 是基数词，将日期转换成英文的基数词拼写。

TO_CHAR(NUM[,'format_model'[,nlsparams]])转换数字
将 NUMBER 类型参数 NUM 转换成 VARCHAR2 类型。如果指定 FORMAT，它会控制整个转换。
如果没有指定 FORMAT，那么结果字符串中将包含和 NUM 中有效位的个数相同的字符。NLSPARAMS 用来指定小数点和千分符及货币符号。它的格式可为：'NLS_NUMERIC_CHARS=' dg '
'NLS_CURRENCY=' 'string' ' 。 d 和 g 分别代表小数点和千分符，STRING 代表货币符号。

数字格式控制符：

9	代表一位数字（替换符。有，数字显示；没有。不什么都显示。）
0	代表一位数字（有数字，显示；没有，强制显示 0。）
$	美圆符号
L	本地货币

. 小数点
, 千分符
B 当整数部分为 0 时，将整数部分填充为空格。 例：B999
MI 返回带有后继符号而不是前导负号的负数值，正数值将带有后继的空格。999MI
S 返回一个前导符号或后继符号，正数为+，负数为-。 S9999 或 9999S
PR 使用尖括号返回负数。正数将有前导或后继空格。999PR
D 在指定位置返回一个小数点。两侧的 9 的个数指定了最大的位数。99D9
G 在指定位置返回千分符，G 可以在 FORMAT_model 中出现多次。9G999G9
C 在指定位置返回 ISO 货币符号。C 可以在 FORMAT_model 中出现多次。C99
L 在指定位置上返回本地货币符号。 L99
V 返回一个被乘以 10 的 N 次方的数值，这里 N 是 V 后边 9 的个数。99V99
EEEE 使用科学记数法返回该数值。9.99EEEE
RM 使用大写的罗马数字表示返回该数值。 RM
rm 使用小写的罗马数字表示返回该数值。 rm
FM 返回不含前导和后继空格的数值。 FM99.09

格式控制符位数一定要大于或等于 NUMBER 的位数，不能小于。

用 RR 解决跨世纪问题： 小于 50 的认为是 1950-2050
大于 50 的认为是 1951-1999

数字和日期是不能相互转换的。

ASCIISTR(string)

返回只包含有效的 SQL 字符和斜线的字符串。string 中的任何无效的字符将被转换为一个相当的数字，在之前加上斜线。

BIN_TO_NUM(num[,num]...)

将一位矢量转换位相当的数字。它的参数是一系列逗号隔开的 NUMS，每一个都必须是 0 或 1。例如 BIN_TO_NUM(1,0,1,1)将返回 11，因为 11 的二进制表示是 1011。当使用分组集合和 GROUP BY 子句时该函数很有用。

CHARTOROWID(x)函数，

将字符串转换成一个 ROWID 类型的值，注意格式必须采用 ROWID 数据类型格式，即“数据块号:行序号:数据文件号”。

COMPOSE(string)

以相同字符集中完全规格化 Unicode 形式返回 string.string 可以是 CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB 或 NCLOB 类型。

DECOMPOSE(string)

返回一个 Unicode 字符串。它是 string 的规范分解。string 可以是 CHAR、VARCHAR2、 NCHAR、NVARCHAR2、CLOB 或 NCLOB 类型。

FROM_TZ(timestamp,timezone)

返回一个 TIMESTAMP WITH TIMEZONE 类型值。它将 TIMESTAMP(没有时区信息)和提供的 TIMEZONE 组合在一起。

HEXTORAW(string)

将由 STRING 表示的二进制数值转换为一个 RAW 数值。STRING 应该包含十六进制值。STRING 中的每两个字符表示结果 RAW 中的一个字节。HEXTORAW 和 RAWTOHEX 互为反函数。

NUMTODSINTERVAL(x,unit)

将 X 转换为 INTERVAL DAY TO SECOND 值，X 应该是一个数字。UNIT 是一个字符字符串（可以是 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2），且是'DAY'、'HOUR'、'MINUTE'、'SECOND'之一。unit 是不区分大小写的，返回值的缺省精度为 9。

NUMTOYMINTERVAL(x,unit)

将 X 转换成 INTERVAL YEAR TO MONTH 值，X 应该是一个数字。UNIT 是一个字符字符串（可以是 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2），且是'YEAR'或'MONTH'之一。unit 是不区分大小写的，返回值的缺省精度为 9。

REFTOHEX(refvalue)

返回一 REF refvalue 的十六进制表示。

RAWTOHEX(rawvalue)

将 RAW 类型值 rawvalue 转换为一个十六进制表示的字符串。rawvalue 中的每个字节转换为一个双字符的字符串。

RAWTONHEX(rawvalue)

将 RAW 类型值 rawvalue 转换为一个十六进制表示的字符串。rawvalue 中的每个字节转换为一个双字符的字符串。RAWTONHEX 返回值是 NVARCHAR2 类型而不是 VARCHAR2 类型。

ROWIDTOCHAR(rowid)函数，将 ROWID 类型值转换成字符串。与 CHARTOROWID 互为反函数。

ROWIDYONCHAR(rowid) 与 ROWIDTOCHAR 类似，返回类型是 NCHAR，而不是 CHAR。

TO_CLOB(string)

将 string 转换为 CLOB。string 可以是文字或另一个 LOB 列。如果参数包含 NCHAR 数据，它被转换为数据库字符集

TO_DSINTERVAL(string[,nlsparams])

将 string(可以是 CHAR、VARCHAR2、NCHAR、NVARCHAR2)转换为 INTERVAL DAY TO SECOND 类型。如果选定 nlsparams,则 nlsparams 只能包含小数点和千分位字符的 NLS_NUMERIC_CHARARCTERS 表示。

TO_LOB(long_column)

将 long_column 转换成 LOB。这个函数用于将 LONG 和 LONG RAW 分别转换为 CLOB 和 LOB。

TO_MULTI_BYTE(string)

返回将所有单字节字符替换为等价的多字节字符的 STRING。该函数仅当数据库字符集同时包含单字节和多字节字符时才使用。否则，STRING 不会进行任何处理而被返回，与 TO_SINGLE_BYTE 互为反函数。

TO_NCHAR

和 TO_CHAR 相似，结果是属于国家字符集而不是数据库字符集。

TO_NCLOB(string)

将 STRING 转换为 NCLOB。STRING 可以是文字或另一 LOB 列。

TO_SINGLE_BYTE(string)

返回将所有双字节字符替换为等价的单字节字符的 STRING。。该函数仅当数据库字符集同时包含单字节和多字节字符时才使用。否则，STRING 不会进行任何处理而被返回，与 TO_MULTI_BYTE 互为反函数。

TO_TIMESTAMP(string[,format[,nlsparams]])

将其参数 CHAR 或 VARCHAR2 类型 string 转换成 TIMESTAMP 类型。

TO_TIMESTAMP_TZ(string[,format[,nlsparams]])

将其参数 CHAR 或 VARCHAR2 类型 string 转换成 TIMESTAMP WITH TIMEZONE 类型。

TO_YMINTERVAL(string)

将 string(可以是 CHAR、VARCHAR2、NCHAR、NVARCHAR2)转换为 INTERVAL YEAR TO MONTH 类型。TO_YMINTERVAL 与 TO_DSINTERVAL 相似,除了它不能使用 NLS 参数作为参数并返回 YEAR TO MONTH 时间间隔而不 DAY TO SECOND 时间间隔。

TO_label(x[,y])函数，按照格式 Y 将字符串 X 转换成 MLSLABEL 类型的一个值，若默认格式为 Y，则按照默认格式进行转换。
dump(w,[x[,y[,z]]])函数,用来返回字符串 EXPR 的数据类型，内部的存储位置和字符长度。
dump(expr,return_datatype,start_position,length).
return_datatype 是指定返回返回位置用什么方式表示，可以为 8、10、16、17，分别表示用八进制、十进制、十六进制和字符类型。
例：

```
select dump(last_name,8,3,2) ,dump(last_name,10,3,2) ,
dump(last_name,16,3,2) ,dump(last_name,17,3,2) from employees
where lower(last_name) = 'smith';
```

greatest(x,y,...)函数，返回参数列表中的最大值。其参数的类型是由第一个参数决定的，可以为数值型、日期型、和字符型等，后面的参数被强制转换成此种数据类型。进行字符串的比较时，其大小由字符在字符集中的数值决定，在字符集中的数值大，则此字符就大，对于字符串，此函数返回 VARCHAR2 类型。
least(x,y,.....)函数，返回列表参数中的最小值。

与上两个函数类似的有：
greatest_lb(x,y,....)函数和 least_lb(x,y,....)函数，分别求出列表中的标签的最大下限和最小上限，其参数必须为
MLSLABEL 类型，返回值为 RAW MLSLABEL 类型。

user 函数，返回当前用户的数据库用户名。
uid 函数，返回唯一标识当前用户的整数。
这两个函数在完整性约束检查时会用到，可以当作引用变量一样引用它们。

userenv(x)函数，返回当前会话的一些信息，由 X 指定返回何种信息。在写一个指定应用的审计测试表或决定为当前会话指定哪种语言时会用到，但完整性约束时不能用。
参数：

Entryid	返回有效的审计条目标识
Label	返回当前会话的标签
Language	以“语言.字符集”形式返回所用的语言和字符集
Sessionid	返回正在使用的审计会话号
Terminal	返回当前会话终端所用的操作系统

1.7、嵌套函数：

单行函数可以嵌套任意层；
嵌套函数从最深层开始执行。

1. 通用函数：

BFILENAME(directory,file_name)
返回操作系统中与物理文件 file_name 相关的 BFILE 位置指示符。directory 必须是数据字典

中的一个 DIRECTORY 类型对象。
COALESCE（,,,, 可以多个参数）返回从左到右的第一个非空的表达式。如果所有表达式都为 NULL，则返回 NULL。
EMPTY_BLOB/EMPTY_CLOB
返回一个空的 LOB 位置指示符。EMPTY_CLOB 返回一个字符位置指示符，EMPTY_BLOB 返回一个二进制位置指示符。
EXISTSNODE(XMLType_instrance,Xpath_string)
使用 Xpath_string 中的路径，确定由 XMLType_instrance 标识的 XML 文档的 TRAVELSAL 是否返回任何节点。这个函数将返回一 个 NUMBER 值，如果没有节点则为 0，如果有节点则为大于 0。
EXTRACT(XMLType_instrance,Xpath_string)
应用 Xpath_string 之后，返回由 XMLType_instrance 标识的 XML 文档的一部分。
GREATEST(expr1[,expr2]...)
返回其参数中最大的表达式。在进行比较之前，每个表达式都被隐式转换为 EXPR1 的类型，如果 EXPR1 是字符类型，则使用非填充空格字符比较，返回结果为 VARCHAR2 类型。
LEAST(expr1[,expr2]...)
返回其参数中最小的表达式，其余同上。
NVL(EXPR1,EXPR2)
类型必须匹配，如果 EXPR1 是 NULL，则返回 EXPR2，否则返回 EXPR1。返回值与 EXPR1 类型相同，除非 EXPR1 是字符类型，在这种情况下将返回 VARCHAR2 类型。这个函数用于确保查询记录集中不包含 NULL 值。
NVL2(EXPR1,EXPR2,EXPR3)
如果 EXPR1 是 NULL，则返回 EXPR2，否则返回 EXPR3。返回值与 EXPR2 类型相同，除非 EXPR2 是字符类型，在这种情况下将返回 VARCHAR2 类型。
SYS_CONNECT_BY_PATH 返回列值的从根到结点的路径，它仅在层次查询中有效。
SYS_CONTEXT(namespace,parameter[,length])
返回与 namespace 的内容相关联的 patameter 的值。使用 DBMS_SESSION.SET_CONTEXT 过程设置参数和 namespace.返回值是 VARCHAR2 类型，如果没有指定 length,则最大长度是 255 字节。
SYS_DBURIGEN
产生一个 URL 用于从数据库中提取 XML 文档。
SYS_GUID
以 16 位 RAW 类型值形式返回一个全局唯一的标识符。
SYS_TYPEID(object_type)
返回指定类型 object_type 的类型 ID。
SYS_XMLAGG
将几个 XML 文档或文档片段组合为一个文档。
SYS_XMLGEN
返回一个基于数据库中数据的 XML 文档片段。
TREAT(expr AS [REF] [schema.]type)
TREAT 用于改变一个表达式的声明类型。仅可以将声明类型改变为给定表达式的子类型或超类型。以类型[schema.]type 返回 expr，如果指定了 REF，则返回 REF。
UID
返回一个唯一标识当前数据库用户的整数，UID 没有参数。
VSIZE(x)返回 X 内部表示的字节数。
NULLIF(a,b)如果 A 等于 B 返回 NULL，如果不等于返回 B。
DUMP(expr[,number_format[,start_position][,length]])
返回一个包含 EXPR 内部表示信息的 VARCHAR 值，如果没有指定 NUMBER_FORMAT，则返回结果以

十进制形式返回。如果指定了 start_position 和 length，则返回从 start_position 开始，长为 length 字节的字符串，缺省是返回整个表达式。所返回的数据类型是内部数据类型编码的对应数字。

NUMBER_FORMAT
格式 返回结果
8 8 进制符号
10 10 进制符号
16 16 进制符号
17 单字符

编码	数据类型	有效于
1	VARCHAR2	ORACLE7
2	NUMBER	ORACLE7
8	LONG	ORACLE7
12	DATE	ORACLE7
23	RAW	ORACLE7
24	LONG RAW	ORACLE7
69	ROWID	ORACLE7
96	CHAR	ORACLE7
112	CLOB	ORACLE8
113	BLOB	ORACLE8
114	BFILE	ORACLE8
180	TIMESTAMP	ORACLE9i
181	TIMESTAMP WITH TIMEZONE	ORACLE9i
182	INTERVAL YEAR TO MONTH	ORACLE9i
183	INTERVAL DAY TO SECOND	ORACLE9i
208	UROWID	ORACLE9i
231	TIMESTAMP WITH LOCAL TIMEZONE	ORACLE9i

USERENV[option]
基于 option 返回包含有关当前会话信息的 VARCHAR2 值。

函数的行为

选项值	USERENV(option)的行为
'OSDBA'	如果当前会话将 OSDBA 角色的设置打开了，则返回'TRUE',否则返回'FALSE',注意返回值是 VARCHAR2 类型，而不是 BOOLEAN 类型。
'LABEL'	仅对 TRUSTED ORACLE 中有效，返回当前会话标志。
'LANGUAGE'	返回当前会话所使用的语言和地域，以及数据库字符集，这是 NLS 参数，返回形式是 LANGUAGE_TERRITORY.CHARACTERSET.
'TERMINAL'	返回当前会话所使用终端的操作系统标识符。对于分布式的 SQL 语句，返回的是本地会话的标识符。
'SESSIONID'	如果初始化参数 AUDIT_TRAIL 被设置为 TRUE，那么将返回审计会话标识符。

在分布式 SQL 语句中，USERENV('SESSIONID')是无效的。

'ENTRYID' 如果初始化参数 AUDIT_TRAIL 被设置为 TRUE，那么将返回可用的审计项标识符。在分布式 SQL 语句中 USERENV('ENTRYID')是无效的。

'LANG' 返回语言名称的 ISO 缩写符号。它的格式比 USERENV('LANGUAGE')要短。

例：

```
select USERENV('TERMINAL'),USERENV('LANGUAGE') from dual;

USERENV('TERMINA USERENV('LANGUAGE')
--------------- ------------------------------------------------
WNJ             SIMPLIFIED CHINESE_CHINA.ZHS16GBK
```

2. 条件表达式：

CASE 表达式（简单 CASE）

语法：

```
CASE 表达式 WHEN  条件 1 THEN 返回值 1
     WHEN  条件 2 THEN 返回值 2
  .
  .
  .
     WHEN  条件 n THEN 返回值 n
ELSE  返回值
END
```

DECODE 函数

语法：

```
DECODE（
  条件,比较值 1，返回值 1
       比较值 2，返回值 2
              .
              .
              .
       比较值 n，返回值 n
       返回值（不满足条件时）
    )

select last_name,salary,
  decode( trunc(salary/2000,0),//条件
      0, 0.00,//比较值 1，返回值 1
      1, 0.09，
      2, 0.20，
      3, 0.30，
      4, 0.40，
```

```
        5, 0.42,
        6, 0.44,
         0.45
          )   TAX_RATE
from employees
where department_id=80;
```

3. 从多表中显示数据：

SQL（老版本的）

等值查询
SELECT TABLE1.COLUMN,TABLE2.COLUMN FROM TABLE1,TABLE2 WHERE
TABLE1.COLUMN1=TABLE2.COLUMN2;//自然连接使用 AND 操作符增加查询条件
使用表的别名来简化查询，提高查询功能。

SELECT E.ID，D.ID FROM EMPL E，DEP D WHERE E.NAME=D.NAME;
E、D：表别名
多表等值连接查询
为了连接 N 个表，至少需要 N-1 个连接条件。

非等值查询
使用 BETWEEN AND 查询近似值作为连接条件的多表结果。
WHERE E.SALARY BETWEEN J.LOW AND J.HIGH

外连接查询
SELECT T1.COL,T2.COL FROM WHERE T1.COL(+)=T2.COL;左外连接所有 T2 的 T1 信息。
SELECT T1.COL,T2.COL FROM WHERE T1.COL=T2.COL(+);右外连接所有 T1 的 T2 信息。
为了看到与连接条件不匹配的数据，就必须得用外连接。

自连接
通过表的别名来创建虚拟逻辑表,进行自连接查询。

```
select worker.last_name || 'work for' || manager.last_name
from employees worker,employees manager
where worker.manager_id=manager.employee_id;
```

9I 适应性连接：

```
select t1.col,t2.col
from table
cross join t2 //交叉连接
natural join t2//自然连接:把两表中所有等值的字段都作为连接条件（但这些连接条件不用写）。
```

从两个表中选出连接列的值相等的所有行。
如果两个列的名称相同，但数据类型不同；或是类型相同，意义不同都会出错。

join t2 using (column_name);基于自然连接，只有在 USING 中出现的，才作为连接条件（在 USING 中列名前一定不能加前缀）。

join t2 on (t1.col=t2.col);基于 ON 的自然连接。等值、非等值或自连接都可以实现。

left|right|full outer join t2 on(t1.col=t2.col);

```
select e.last_name,d.department_name,l.city
from employees e
left outer join departments d on e.department_id=d.department_id
      right outer join locations l  on d.location_id=l.location_id;
```

& 可以连续做左连接或右连接的操作。
full outer join 忽略连接条件，把要查询的列的所有行全显示出来。

笛卡尔乘积（多表查询容易产生的错误）形成原因：
*、忽略连接条件；
*、连接条件不正确；
*、笛卡尔乘积是由第一个表的所有行和第二个表的所有行联合形成的；
*、为了避免笛卡尔乘积的产生，一定要在 WHERE 条件中正确写出连接条件。
set linesize 160;设置显示行的行数。

1.8、用字函数产生的总计

对多行的计算产生单行的结果。

组函数用语对每个组的行集进行运算，每个组产生一个结果。

AVG([DISTINCT/ALL]col)只能用与数字。只能对多行的数据进行运算，不能在这个函数中做单行的数学运算。

CORR(x1,x2)
返回表达式 X1 和 X2 组成的集合的相关系数。在保证所有行中的 X1 和 X2 都不为 NULL 之后结果通过
COVAR_POP(x1,x2)/(STDDEV_POP(x1)*STDDEV_POP(x2))得到。
COUNT([DISTINCT/ALL]col)所有非空字段的行数。
COVAR_POP(x1,x2)返回表达式 x1 和 x2 组成的集合的人口协方差结果通过
(SUM(x1*x2)-SUM(x2)*SUM(x1)/n)/n 得到，n 是没有 NULL 项的集合的数目。
COVAR_SAMP(x1,x2)返回表达式 X1 和 X2 组成的集合的相同协方差。
CUME_DIST 返回一组值中一个值的累积分布。
DENSE_RANK 返回有序分组的行中一行的秩，秩是从 1 开始的连续的整数。
GROUP_ID()返回一个唯一数字值用于在 GROUP BY 字句中辨别组。
GROUPING_ID 返回一个数字对应于一行的 GROUPING 位矢量。

MAX([DISTINCT/ALL]col)可以用于任何类型，当用于日期类型时代表最晚。忽略空值。字符类型时候，比较字符串首字母的 ASCLL 值。
MIN([DISTINCT/ALL]col)可以用于任何类型，当用于日期类型时代表最早。忽略空值。字符类型时候，比较字符串首字母的 ASCLL 值。

PERCENTILE_CONT 这个函数是一个反分布函数，它假设了一个连续分布模式。
PERCENTILE_DISC 一个反分布函数，它假设了一个离散分布模式。
RANK 返回给定行的秩。秩不必是连续的，因为相同的行有相同的秩。
REGR 这些函数
(REGR_SLOPE,REGR_INTERCEPT,REGR_COUNT,REGR_R2,REGR_AVGX,REGR_AVGY,REGR_SXX
REGR_SYY,REGR_SXY)得到了双集合的普通最小衰减线。
SUM([DISTINCT/ALL]col)返回选择列表项目的总和，只能用于数字。
STDDEV([DISTINCT/ALL]col) 标准方差
STDDEV_POP(col)计算人口标准差并返回人口方差的平方根。
STDDEV_SAMP(col)计算累计标准差并返回例子方差的平方根。
VAR_POP(x)返回提系列数字在去除了 NULL 值之后的人口不同。由
(SUM(x*x)-SUM(x)*SUM(X)/COUNT(x))/COUNT(x)得到。
VAR_SAMP(x)返回一系列数字在去 NULL 值之后的范例不同。由
(SUM(x*x)-SUM(x)*SUM(X)/COUNT(x))/(COUNT(x)-1)得到。
VARIANCE([DISTINCT/ALL]col)偏移方差，返回 COL 的方差。

语法：
select col,group function(col) from table where 条件 group by col;
GROUP BY
必须：出现在查询列表中的一个字段，但没有出现在函数中，那么这个字段必须要出现在 GROUP BY 中。
可以：出现在 GROUP BY 子句中的字段可以不出现在查询列表中。
先排列，再运算。

WHERE 子句中不能使用 group function。
限制组必须使用 HAVING 子句。

语法：
select col,group function from table
where 条件//可以没有条件限制
group by col
having group_condition //组过滤，在过滤以后，再进行分组计算。
order by col;

组函数嵌套最多只能有两层。
select max(avg(salary))
from employees
group by employee_id;

select * from tab;查询一个用户中的所有表。

1.9、子查询：

语法：
select col from table
where expr operator (select col from table);
子查询在主查询执行前执行一次。
子查询的结果被用于主查询。

使用规则：
在 WHERE 和 HAVING 子句中都可以使用子查询。
*、子查询必须用括号扩起。
*、子查询应该在比较条件的右边。
*、在子查询中的 ORDER BY 子句不需要，除非执行 TOP-N 分析。
TOP-N 分析：（在一些表里求出最怎么怎么怎么样（最好、做多...）的几个人）。
*、对单行子查询使用单行比较操作符，多行子查询使用多行比较操作符。
可以在子查询中使用组函数。

子查询的分类：
单行单列子查询：一定返回一行
　　单行操作符：>,<,=，>=,<=,<>
多行单列子查询
单行多列子查询：返回零行或多行
　多行操作符：
　　in 等于列表中的任何值。（不能用 NOT IN）
　　ANY 与子查询返回的每个值进行比较。(小于是小于最大的，大于是大于最小的)
　　select employee_id,last_name,job_id,salary from smployees
　　where
　　salary < any(select salary from employees where job_id='IT_PROG')
　　and job_id <> 'IT_PROG';
　　ALL（小于是小于最小的，大于是大于最大的）
select e.employee_id,e.last_name,e.salary
from employees e,
(select department_id,min(salary) m from employees
group by department_id)d
where e.department_id=d.department_id
and e.salary=d.m;
查询每个部门薪水最少的员工的资料。

4. 多行多列子查询

2.0、替换变量：

临时存储值：

1.&

生命周期：单次引用中，不需要声明。如果替换字符或日期类型，最好用单引号扩起。
使用范围：
where
order by
列表达式
表名
整个 SELECT 语句中。

2.&&

生命周期：整个会话（session 连接）,不需要声明。
define（生命周期）：整个会话，预先声明，使用时用&引用声明的变量。
define column_name(变量名) 查看变量命令。
undefine 变量名 清除变量
define variable=用户创建的 CHAR 类型的值：define 变量名=值；

accept（生命周期）：整个会话，预先声明，可以客户化提示信息，使用时用&引用声明的变量。
定义：accept 变量名 number/char/date prompt ’提示信息内容’
ACC[EPT] variable [NUM[BER] | CHAR | DATE] [FOR[MAT] format]
[DEF[AULT] default] [PROMPT text | NOPR[OMPT]] [HIDE]

例：accept a char prompt ’请输入员工的雇佣时间(yyyy-mm-dd):’

hide
例：accept a char prompt ’input a:’ hide
set verify(环境变量) off;关闭调试命令（关掉替换过程）
set verify(环境变量) on;打开调试命令(可以看到替换过程)

2.1.环境变量：

ECHO 显示回显
HEADING {OFF/ON}是否显示列标题；
ARRAYSIZE{20/n}每一次从查询得到的返回量的大小。
FEEDBACK{OFF/ON}回馈，反馈信息。
LONG{80/n}on/text} LONG 类型
LINESIZE 行的宽度。
SET LINESIZE n(最好是在 200 之内)
PAGESIZE ：设置页的大小。SET PAGESIZE N

wrap{off/on} 折行
SET 修改
SHOW 显示

2.2 格式化命令：

COLUMN[column option]可以设置字段或字段别名的格式。
COLUMN last_name HEADING employee|name ’|’代表换行。
col 字段名 查看命令
CLE[AR]：清除列的格式
HEA[DING] TEXT：设置列标题
FOR[MAT] FORMAT：格式化显示列的值，对字符和数字有效，对日期无效。
column salary justify left format $999,999.00
justify left:左对齐。
col manager_id format 999999999
限制字符串的长度有 A+数字限制
限制数字的长度有 9，有几为 9 就限制成几位。
NOPRINT/PRINT NOPRINT：把一个字段从输出上屏蔽掉（返回但不显示）。
col 字段名 noprint/print.

NULL 如果有 NULL 值，显示什么。
col name null ’on employee’

TTITLE[text/off/on]设置报表的表头
BTITLE[text/off/on]设置报表的表尾
做报表的时候要先想好 PAGESIZE 的大小。

BREAK ON [REPORT_ELEMENT]
压制重复值的显示。只能跟一个字段名才有效。
例：

```
select department_id,last_name
from employees
where rownum<30
order by 1,2;

break on department_id
otn.oracle.com/cn
www.oracle.com/cn
www.itpub.net 入门与认证版 ora-600
www.oracle.com.cn
www.cnoug.org  ora-600
```

2.3 做脚本文件的过程：

变量定义 accept
环境变量设置 SET
格式控制命令
SPOOL
使用变量的 SQL
SPOOL OFF
清除格式控制
重置环境变量
释放变量

2.3 数据操作语句：

1. 插入

INSERT INTO TABLE（字段 1，字段 2....）VALUES（值 1，值 2....）
一次插入只插入一行。字符和日期值需要单引号扩起。
插入空值：
方法一：隐示插入，插入时省略列名系统就会默认省略的列为 NULL。
方法二：显示插入，在插入时指定列的值为空。
注：（1）值的个数不能少于列名的个数。
（2）注意非空属性的列，不能插入空值。
WITH CHECK OPTION 视图约束。
UID 当前数据库用户 ID
在插入日期的时候最好用 TO_DATE 来控制输入格式。
可以创建一个脚本用 &变量名 的形式来用一个插入语句实现多行的插入（在值列表里用 &变量名）。
插入中的子查询：将另一个表中的内容都插入被插入的表中。

```
insert into sales_reps(id,name,salary)
select employee_id,last_name,salary from employees where employee_id>100;
```

可以在子查询中做运算后插入到被插入表中。
不要使用 VALUES 子句。
在子查询中的列数必须匹配 INSERT 语句中的列数。

2. 删除

DELERT FROM TABLE WHERE 条件;
删除所有符合 WHERE 条件的行。
基于子查询的删除。
注意及联删除。

3. 更新

UPDATE TABLE SET 列名 1=值 1，列名 2=值 2.....WHERE 条件;

如果更新错误，要用 ROLLBACK 回滚。
利用子查询更新另外表中的数据，在 SET 后和 WHERE 后都可以利用子查询语句。
更新的时候要注意参照完整性约束。
子表的外键字段值必须是父表主键字段值的真子集。

DEFAULT+‘’ 默认值
在创建表的时候用，在第三个参数的位置上。
例：

```
CREATE TABLE A
（    C1 CHAR（10） DEFAULT
      .
      .
      .
）
```

在修改的时候，如果先给 DEFAULT 赋值的话，可以直接用 列名=DEFAULT，使列名回复为默认值。

4. MERGE 语句

提供了对表根据条件进行插入或者更新的能力。
如果行存在则执行 UPDATE，如果不存在则执行 INSERT。

避免了单独的修改。
提高了性能，更便于使用。
对于数据库应用很有益。

语法：别名 AL

```
MERGE INTO table_name（目的表） table_alias USING (table/view/sub_query)//数据来源
（可以用子查询） alias(别名)
on(连接条件)
WHEN MATCHED THEN UPDATE SET(关键字) (修改目的表)
目的表 AL.col1=原表 AL.col_var1
目的表 AL.col2=原表 AL.col2_var2
WHEN NOT MATCHEN THEN（关键字）
INSERT (目的表 AL.COL_LIST)
VALUES(原表 AL.COL_VARS);（插入原表）
create table newtable_name(新表) as select * from oldtable_name(原表) where 1=0;
 将原表中的结构复制到新表中，但具体的数据项不进行复制。
```

5. 事务（transaction）：

由被逻辑组织在一起的多个 DML 语句的构成。
COMMIT：提交。
ROLLBACK：回滚。
SAVEPOINT：存储点，只在事务执行过程中有效，事务结束即被释放。
事务的组成：

一组相同改变特性的 DML 语句；
一个 DDL：数据定义语句；
一个 DCL：权限控制语句；

建立存储点：
例：SAVEPOINT A；
.
.
.
ROLLBACK TO A；

事务的开始：
开始于第一个 DML SQL 语句执行时开始
结束的时候是在：
*、一个 COMMIT 或 ROLLBACK 被执行的时候。
、一个 DDL 或 DCL 语句被执行（自动提交）注意&*!（隐式）
*、用户退出 SQLPLUS（隐式）
*、系统崩溃（隐式）

语句级回滚：
ORACLE 服务器执行隐式的存储点。

2.4 创建和管理表

1、表（TABLE）基本的存储单位，由行和列组成。

1. 规则：
1.表名和列名（使用规则）：
2.必须是字母开头；
3.必须是 1-30 的字符长度；
4.只能包括 A-Z，a-z,0-9,_,$,#；
5.在同一个用户下不能头重名的对象；
6.不能是 ORACLE 的保留字；

创建需求
必须有： 1. CREATE TABLE 权限；
2.足够的存储空间；
语法：
CREATE TABLE [SCHEMA（方案）.]TABLE
（COL DATATYPE [DEFAULT 默认值][]）；

当前用户所有的表
select table_name from user_tables;
当前用户所有的对象：

desc user_objects
当前用户对象的别名：
select * from cat;

字段类型：
VARCHAR(size) 变长字符串类型
CHAR(size) 定长字符串类型
NUMBER(p,s)p 位整数，s 位小数
DATE
DATETIME 秒级最多可以到小数点后的 9 位
TIMESTAMP 带有小数秒的日期
TIMESTAMP WITH TIME ZONE 带时区的类型
TIMESTAMP WITH LOCAL TIME ZONE 带时区的并会进行时区转换的类型（同一时间在不同地区看到的时间）
INTERVAL YEAR TO MONTH 按年和月的间隔存储的类型
INTERVAL '123-2' TEAR(3) TO MONTH
INTERVAL DAY TO SECOND 按天、小时、分和秒的间隔存储的类型
INTERVAL
LONG 变长的长字符串类型
CLOB 字符类型 4GIGABYTES
RAW 二进制类型与 CHAR 对应
LONG RAW 二进制类型与 LONG 对应
BLOB
BFILE 以文件的形式存储在操作系统中
ROWID 表中行的唯一地址（行地址）

2、方案： 一个用户所有对象的命名集合。

如果想访问其他用户或方案的表要加上用户或方案作为前缀。

必须指明：
表名称；
列名，列类型和长度；
用户表：
被用户创建和维护的一些表；
包括了用户自己的信息；
数据字典表：
被 ORACLE 数据库创建和维护的一些表；
包括了数据库的信息；

3、CTAS（子查询建表）：

CREATE TABLE table_name

[(col,coltype,..)]
as subquery(子查询);

创建的表的列的数目匹配子查询的列的数目。
使用子查询的列的名字和默认值定义表。
注：
*、被创建表的字段名要遵循如果没有字段别名和子查询中没设置别名的话，使用子查询中的列名；
　如果有别名，使用别名；如果有字段列表（[(col,coltype,..)]），在被创建的表中使用字段列表；
*、有字段列表与子查询的列要匹配。
*、当没有字段列表的时候，而在子查询中有表达式的时候一定要在表达式后要加上别名。
*、只会把属性当中的非空属性复制过来，其他的比如约束条件、关联...都不会复制过来。
使用 ALTER TABLE 语句可以：
*、在表中增加一个新列
语法：ALTER TABLE table add (col datatype [default],...,....);
新增加的字段一定是放在表的最后。
*、修改表字段的类型和长度
ALTER TABLE table modify (col datatype [default],...,....);
对默认值的修改只会影响到新插入的行。
如果字段下有值的话，类型的修改成功率很小（要修改数据类型，要修改的列必须为空，即没有数据项）。
CHAR 类型不能修改长度。
*、删除表字段
ALTER TABLE table DROP COLUMN (COLUMN_NAME_LIST);
9I2 版可以修改列名
*、SET UNUSED 设置字段为不可用。
原理：清楚掉字典信息（撤消存储空间），不可恢复。
可以使用 SET UNUSED 选项标记一列或者多列不可用。
使用 DROP SET UNUSED 选项删除被被标记为不可用的列。

语法：
ALTER TABLE table SET UNUSED (COLlist 多个) 或者　ALTER TABLE table SET UNUSED COLUMN col 单个;
ALTER TABLE table DROP UNUSED COLUMNS;
删除表：
删除关联：drop table table_name cascade;
改对象名：
　RENAME 对象原名 TO 要改的对象名;
注：　必须是对象的所有者才能进行改名的操作。

4、截取：

不能回滚；
删除表中所有数据；

释放存储空间；
语法：
TRUNCATE TABLE 表名称；
DELETE 也可以删除所有行，但：
可以回滚。
不释放存储空间。

5、给表加注释：COMMENT

comment on table table_name is '注释内容';

6、约束条件：

如果经常用到约束条件的话，最好自己命名。
当定义约束的时候可以将定义的语句作为 CREATE TABLE 中的参数的一部分来完成。
表级别约束定义：
CONSTRAINT 约束名 约束条件（字段名）
约束在表上强制了规则。
如果有参照的花，约束防止表的删除。
ORACLE 支持的约束条件：
1.NOT NULL 非空
特点：唯一一个只能在列级定义的约束条件。
2.UNIQUE 唯一
允许有空值（空值不做比较）；
特点：当创建约束的时候，系统会自动创建对应其的索引。
3.PRIMARY KEY 主键
特点：当创建约束的时候，系统会自动创建对应其的索引。
在一个表中只允许一个主键。
4.FOREIGN KEY 外键
外键参照的一定是主表的主键或唯一键；
保证子表外键字段的值一定是主表中的被参照字段值的真子集；
当主表字段被参照的时候，其值不允许被直接删除。
5.CONSTRAINT 约束名 FOREIGN KEY （外键字段名） REFERENCES 主表名（主表字段名）；
如果在字段列表中定义外键就可以不写 FOREIGN KEY 关键字。
如下格式：CONSTRAINT 约束名 REFERENCES 主表名（主表字段名）；
6.ON DELETE CASCADE 当主表的行被删除的时候，要删除子表中参照主表的行。
ALTER TABLE TABLE_NAME DROP (PK) CASCADE CONSTRAINTS;把作为主键的字段也同时删除了。
7.ON DELETE SET NULL 当主表的行被删除的时候，转换子表中的参照值为空。

CHECK
定义一个每行都必须满足的条件。
CREATE TABLE table_name
(....

```
salary number(10,2),
CONSTRAINT 约束名 CHECK(SALARY>0),
  ....
);
```

约束的使用：
约束的命名：给约束命名或者 ORACLE 服务器将使用 SYS_Cn 的格式为约束命名。

创建时期：在创建表的同时或者在建表之后。
定义级别：
可以在表级定义或列级定义。
在数据字典中可以查看约束。

使用 ALTER TABLE 语句：
*、添加或者删除约束条件，但是不能修改约束条件。　就算列名上已经有约束条件，还可以继续添加约束条件的。
添加：ALTER TABLE table_name ADD [CONSTRAINT] 约束名 约束条件(column);
删除：ALTER TABLE table_name drop constraint 约束名;
ALTER TABLE table_name PRIMARY KEY CASCADE;删除主键的时候，不用约束名。
*、启动或禁用约束条件
  ALTER TABLE table_name Disable constraint 约束名; 禁用
  ALTER TABLE table_name ENABLE constraint 约束名; 启用
*、通过 MODIFY 添加 NOT NULL 约束条件（因为 NOT NULL 为列级约束，只能用 MODIFY 添加）。
ALTER TABLE table_name MODIFY(col type NOT NULL);

查看约束条件：
//desc user_constraints
OWNER 拥有者；
CONSTRAINT_NAME 约束名称
CONSTRAINT_TYPE 约束类型
SEARCH_CONDITION check 的条件
select constraint_name, constraint_type,search_condition,status
from user_constraint where table_name='b';

2.5.视图 （VIEW）

一个或多个表的数据集的逻辑表示（虚表，不存储数据）
视图不能提高查询的性能。
分类：
简单
　　数目：一个
　　函数：不包含
　　分组数据：不包含
　　可以做 DML 操作
复杂

数目：一个或多个
函数：包含
分组数据：包含
不一定能做 DML 操作

视图也可以用 DESC 描述。
创建视图：

```
CREATE [or replace(修改视图)] [force/noforce] VIEW view_name(col coltype ,.....)
as
subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY[CONSTRAINT constraint]];
```

USER_VIEWS 关于视图的字典
修改视图：

```
CREATE OR REPLACE 原视图名 （字段列表）
AS 子查询;
```

包含：
GROUP BY 、DISTINCT、ROWNUM
不能对视图进行删除操作；

包含：
GROUP BY
DISTINCT
ROWNUM
通过表达式定义的列不能对视图进行修改操作；

包含：
GROUP BY
DISTINCT
ROWNUM
通过表达式定义的列
在视图中没有包含基表中的 NOT NULL 列，不能对视图进行插入操作；

使用视图的原因；
为了限制对数据的访问；
为了使复杂的查询变得简单；
提供了数据的独立性；
提供了对相同数据的不同显示；

使用 WITH CHECK OPTION 子句创建视图
创建视图时通过 WITH CHECK OPTION 子句确保执行的 DML 语句不会引起数据不出现在视图上。
在对视图做 DML 操作的时候，一定要符合 WHERE 子句中的条件。

```
CREATE OR REPLACE VIEW empvu20 as select * from employees
where check option constraint [empvu20_ck];
```

WITH READ ONLY
不可以进行 DML 操作；

删除视图：
DROP VIEW view_name;

行内视图：是一个在 SQL 语句中使用的带有别名的子查询，该子查询放在 FROM 之后；

TOP-N:
select [col_list],rownum rank(排名)
from (select [col_list] from table_name order by top-n_col)
where rownum<=n;

序列（SEPUENCE）产生的顺序数字，单向递增或单向递减，且步长相同。
索引（INDEX）用于提高查询性能。
同义词（SYNONYM）对象的别名。
create public synonym e for hr.employees;
user_synonyms;

创建同义词要有权限，访问的时候也需要权限。

2.6、序列：

自动产生的唯一值；
一个共享的对象；
典型的用法是作为主键的值；
insert into 给主键提供值。
替代了应用的代码；
通过将序列 CACHE（预先生成一部分序列号，放入到内存中）到内存中，可以加速对序列的访问。

CREATE SEQUENCE sequence_name
[increment by n]//步长
[start with n]//起始点
[maxvalue n/nomaxvalue]//递增
[minvalue n/nominvalue]//递减
[cycle/nocycle]//循环
[cache n/nocache];//n 为预先生成序列号的个数，默认为 20。

查询序列：
user_sequences
last_number 序列将要产生的下一个号是多少；
select sequence_name,min_value,max_value,increment_by last_number from
user_sequences;
伪列：NEXTVAL 引用下一个可用的序列值，不同的用户每次引用都会获得一个唯一的值。

CURRVAL 得到当前的值（刚被领走的号）。
在 CURRVAL 执行前必须先通过 NEXTVAL 得到一个初始的值。
序列名.NEXTVAL/CURRVAL
序列发生间隙是正常的，保证唯一即可。

序列的修改：

```
ALTER SEQUENCE sequence_name
  increment by
  maxvalue
  cycle
  cache;
```

start with 不能修改。

删除序列：

```
drop  SEQUENCE sequence_name;
```

2.7、索引：

一个方案中的对象；
被 ORACLE 服务器用来加速对表的查询；
通过使用快速路径访问方法快速定位数据；
与表独立存放；
被 ORACLE 服务器使用和维护。
一定是 WHERE 条件的才有可能使用索引。

手动创建索引:

```
CREATE INDEX index_name on table_name (col_name);
```

考虑创建索引的情况：
*、包含了大量不同值的列；
*、包含了大量空值的列；
*、一个或者多个列经常被一起出现在 WHERE 条件中或者作为连接的条件出现；
*、表的数据量很大，而且对表的查询经常是得到表中数据的 2%到 4%（少量数据）.
不应该创建索引的情况：
*、一个很小的表；
*、列很少被用于查询的条件；
*、表上的大多数查询是得到大量数据的；
*、表中的数据经常发生变动；
*、要被索引的列被作为条件表达式的一部分。
查看：
user_indexes 得到索引的定义和唯一性。
user_ind_columns 得到索引的名称，表名和列名。

```
select ic.index_name,ic.column_name,ic.column_position,    ic.uniquenes
from user_indexes ix,user_ind_columns ic
where ic.index_name=ix.index_name and ic.table_name='table_name';
```

删除索引：
DROP INDEX index_name;
为了删除索引，必须拥有索引或者拥有 DROP ANY INDEX 权限。

2.8 控制用户的访问

1.数据库的安全性

系统安全性：
系统权限(system privilege)，获得访问数据库的能力。
超过一百个
创建新用户：
CREATE USER user_name IDENTIFIED BY password;
删除用户
删除表
授予权限：
GRANT priv_list TO user/public/role(角色);
grant create session ,create table,create sequence to user_name;
在授予建表权限的同时也应该赋予存储空间。
分配配额：ALTER USER user_name QUOTA nM ON space_name;

数据安全性：
对象权限(object privilege)，获得维护数据库的能力。
每种对象的权限都不相同。
对象的所有者拥有对象的所有权限。
对象的所有者可以将自己的对象权限赋予其他人。
GRANT object_priv_list [(col_list)]
ON owner.object TO user/role/public
[WITH GRANT OPTION];--将权限授予用户的同时，该用户也拥有了授予其他用户对象权限的功能。(及联授予）会导致及联移除。

移除权限：
REVOKE priv_list/all ON object FROM user;

方案：数据库对象的集合，包括表、视图、序列.....。

2.角色：

CREATE ROLE role_name;
GRANT priv_list TO role_name;
GRANT role_name TO user_list/role_list;
修改口令：
方法 （1）ALTER USER user_name IDENTIFIED BY password;
（2）password + 回车

USER_SYS_PRIVS 当前用户的系统权限。
USER_ROLE_PRIVS 当前用户的角色权限。
USER_tab_privs_made 用户对象被授予的他人的信息。

3.使用集合操作

```
UNION
select employee_id,job_id from employees
union
select employee_id,job_id from job_history;
```

两个表的并集，但不显示重复行。执行的时候要先排序再剔重，所以结果集是有序的。
union all
也是两个表的并集，而且显示重复行。
语法同上。

```
intersect
select employee_id,job_id from employees
intersect
select employee_id,job_id from job_history;

minus
select employee_id,job_id from employees
minus
select employee_id,job_id from job_history;
```

e-j=e-e 与 j 的交集；
j-e=j-j 与 e 的交集；

注：在 select 列表中的表达式必须有同样的数目和类型。

```
匹配 SELECT 语句
select employee_id,job_id,salary from employees
union
select employee_id,job_id,0      from job_history;
```

括号可以用拉修改序列的执行顺序。

4.ORDER BY 子句：

只能在整个集合的最后出现；
可以按照第一个 SELECT 语句中的列名，别名或者位置号排序。

5.GROUP BY 子句的增强

CUBE 操作符的 GROUP BY
在 GROUP BY 子句中使用 ROLLUP 或者 CUBE 来产生分组小计；
ROLLUP 分组产生包括规则的分组结果和小计的结果的组合；
GROUP BY [ROLLUP]（col_name_list）
ROLLUP:

```
a       ab    abc

      abc
  ab  ab
a a   a
all   all all
```

CUBE 分组产生包括 ROLLUP 产生的结果和交叉分组小计。
cube:

```
a ab  abc

a ab  abc
all   a   ab
  b   ac
  all bc
      a
      b
      c
      all
```

6.GROUPING 函数

参数一定是在 CUBE 或 ROLLUP 里进行分组排序的字段或表达式之一。
通过 1 或 0 来判断结果集中的空值是由于本身列的值是空的，还是由于使用 CUBE 或 ROLLUP 产生的空值。
1 代表是由于分组产生的空值，没有参与分组。
0 代表是由于列本身产生的空值，参与了分组，但分组中没有包含它。
GROUPING SETS
可以使用 GROUPING SETS 在同一个语句中定义多个组集。
只需要访问一次基表。
不需要写很复杂的 UNION 语句。
GROUPING SETS 子句中组合的元素越多，语句的执行性能就越好。
group by GROUPING SETS((abc),(ab),(bc),(a),(b))
组合列：
是一个列的组合，在分组计算时被作为一个单元处理。

2.9 高级子查询

1. 成对子查询：

行内视图的性能比成对子查询的性能高。

相关子查询：

主查询的字段在子查询里做条件（特征）。

主查询先执行，取出第一条数据，把该数据传入子查询做比较，返回查询结果给主查询，主查询根据这个结果再做查询

依次类推

直到主查询中没有可查询列为止。

EXISTS 操作符

EXISTS 操作符测试子查询的结果是否存在；

返回 TRUE 或 FALSE

查询机制：

如果一个子查询找到了结果：

在内部子查询中不在继续执行，条件被设为 TRUE

如果一个子查询没有找到结果：条件被设为 FALSE

```
select col_list from table_name tab_alias
where exists (select 'x' from table_name where col=tab_alias.col);
```

用的是相关子查询

NO EXISTS 操作符

和 NOT IN 相对应，速度要快，性能好。

UPDATE 中的相关子查询

```
update emp e
set department_name in(select d.department_name from departments d where
e.department_id=d.department_id);
```

delete 中的相关子查询

2.层次查询

```
select [level],column,expr from table [where condition]
[start with]起点（自底向上/自顶向下）
[connect by prior + 主键/外键=外键/主键]//看你往哪个方向查
```

自顶向下 左边放主键，右边放外键

```
select employee_id,last_name,salary,job_id,manager_id
from employees
start with manager_id is null
connect by prior employee_id=manager_id;
```

自底向上 右边放主键，左边放外键

level（伪列）

层次的级别：不固定值。

使用 level 和 LPAD 层次化格式的显示
修剪分支

Oracle 9i 对 DML 和 DDL 语句的扩展
多表插入的 INSERT 语句
insert .... select 语句可以被用来在单个 DML 语句中向多个表插入数据。
多表插入语句：
无条件 INSERT
条件 ALL INSERT
条件 FIRST INSERT
轮巡 INSERT

二、Management：

1.Oracle 的构件和组件

instance 实例/例程
database 数据库
SGA 系统全局区
shared pool 共享池
library cache 库高速缓存区
data dictionary cache 字典高速缓存区
database buffer cache 数据库高速缓存区
redo log buffer 重做日志缓冲区
java pool java 池
large pool 大池
PMON 进程监视进程
SMON 系统监视进程
DBWR 数据库写进程
LGWR 日志写进程
CKPT 检查点进程
data file 数据文件
control file 控制文件
redo log file 重做日志文件
parameter file 初始化参数文件
password file 口令文件
archived log file 归档日志文件
user process 用户进程
server process 服务进程
PGA 程序全局区
tablespace 表空间

2.数据库的物理结构：

1.控制文件

2. 数据文件

3. 重做日志文件

4. data file 数据文件：

5. 作用：存放数据。

6. 数据文件大小可以扩展。

7. tablespace 表空间：一个或多个数据文件的逻辑组成。

一个数据库最少有一个 system 表空间，一个表空间最少有一个数据文件。

8. redo log file 重做日志文件

重做日志：在数据库中发生的所有改变，改变的每一条信息都叫做一条重做日志信息。
存放重做日志信息的文件就叫做重做日志文件。
作用：利用日志记录可以恢复损坏的数据库，保证数据的可恢复性。
重做日志组：包含一个或多个日志文件。
特点：
一个数据库最少有两个组，一个组最少有一个组成员。
重做日志的大小是固定的。
写入方式是顺序写入，按时间。
切换（一个写满往下一个写），循环（都写满了就重新回到头组写）。

9. control file 控制文件

数据库中的核心文件。
存放内容：
数据库的名称和编号，数据库的结构信息（数据文件和重做日志文件的地址）
当前的 SCN system change number 系统改变号：
给系统的每一次改变编号（最后一次同步的 SCN 号）。
控制文件的 SCN 号和所有存放数据文件的头部的 SCN 相同。
启动的时候，一样启动，不一样恢复。
特点：一个数据库最少需要一个控制文件，但一般情况下都会有复用/副本。

10. parameter file 初始化参数文件

非必要文件
跟实例有关。

11. password file 口令文件

非必要文件。
作用：做特权身份认证。
sysdba/sysoper/普通身份

12. archived log file 归档日志文件

非必要文件。
内容：
重做日志文件（redo log file）内容的备份。
作用：使数据库所有的日志都会保留下来。
特点：每个重做日志内容都会对应一个归档文件。
LOG SEQUENCE 日志序列号：重做日志内容的序列号。
非归档日志：
没有归档日志文件；
性能要好一些；
可以选择归档模式或非归档模式。

3. instance 实例/例程

启动的步骤：
（1）启动实例；
实例包含：
1. SGA
2. 后台进程
一个实例只有一个 SGA，可以有多个 PGA，建立一次连接(会话)就产生一次 PGA。

ORACLE 的内存结构：
SGA 在实例启动的时候分配，是 ORACLE 实例的基本组件； 大小可以动态改变（内部大小）
通过 SGA_MAX_SIZE 定义大小（总大小是静态的）
PGA 程序全局区(私有区域)：在服务进程启动的时候分配。

PGA 程序全局区
是为每个用户进程连接 ORA 数据库提供的内存区域；
当进程创建时分配；
当进程结束时释放；
一个 PGA 只被一个进程使用；
包含：有连接的信息

SGA 系统全局区
所有数据库的后台进程和服务进程的共用内存区域。
包含：
1. shared pool 共享池

设置大小的参数：SHARED_POOL_SIZE
包含：
library cahce 库高速缓存区
大小由共享池决定。
包含：SQL 语句和 SQLPLUS 文本，分析代码，执行计划
存储目的：与性能有关。
当两个语句相同的时候不需要重新分析，使用相同的执行计划即可。
管理方法：采用 LRU （最近使用算法）least recent used 当空间不足的时候，使用 LRU。
结构： SQL 区域、PL/SQL 区域
data dictionary cache 字典高速缓存区
内容：最近最多使用的数据字典信息
作用：为了能够在分析的时候所需要的字典信息能在内存中找到，避免使用物理 I/O。
管理方法：采用 LRU （最近使用算法）

2.database buffer cache 数据库高速缓存区
单位：块。
内容：最近最常数据块
作用：减少物理 I/O，提高性能
参数：DB_BLOCK_SIZE 决定主数据块的大小。
会影响性能；
设置后不允许修改；
管理方法：采用 LRU （最近使用算法）
由独立的自缓冲区组成：
DB_CACHE_SIZE
CB_KEEP_CACHE_SIZE
DB_RECYCLE_CACHE_SIZE
可以通过 ALTER SYSTEM 命令动态增加或收缩：
ALTER SYSTEM SET DB_CACHE_SIZE =96M;

redo log buffer 重做日志缓冲区
记录了对数据块所作的所有修改。
作用：数据库的恢复
大小由 LOG_BUFFER 决定。
顺序单条写入，批量写出到文件。

java pool java 池
可选
java_pool_size 大小
large pool 大池
可选
减轻共享池的负担。
被用于： 1.在共享服务器中的会话的内存需求（UGA）；
2.I/O 辅助；
3.备份恢复操作或 RMAN；
4.并行执行的信息缓冲区；
large_pool_size 大小

4、进程结构

1. 用户进程： 开始于数据库用户请求连接数据库

2. 服务进程： 与 ORA 实例连接，开始于用户会话的建立。

分为： 专用服务进程、共享服务进程
性能专用更好。
利用资源方面共享更好（网站方面）。

3. 后台进程： 当 ORA 实例启动时启动

后台进程包含：
1.PMON 进程监视进程
监视用户进程（客户端连接服务器的进程）到服务进程（在服务器端响应用户进程的进程）的连接。
创建会话。
监视会话是否异常中断，如果中断：PMON 会回滚事务、解锁、释放资源。

2.SMON 系统监视进程

任务：会检测 SCN 号，相等：启动
不相等，实例恢复（1）前滚将日志应用
（2）打开数据库
（3）恢复数据库
恢复的起点是走后一次 CHECKPOINT 的位置。
每 3 秒合并空闲空间
释放临时段
临时段：暂时存放在排序时中没有空间的字段值。
在排序中产生的。
数据量大的时候，排序是分成若干块执行的，当字段值排好序之后就放到临时段中。

1. DBWR 数据库写进程

/DBW0/DBWn n：0-9
将脏数据写回到数据文件中。

当发生以下情况执行：
当发生检查点事件的时候，checkpoint

脏块达到极限值；
没有空间的缓冲空间
超时
RAC PING 请求
表空间离线
表空间只读
在表执行 DROP 或 TRUNCATE
表空间上执行 BEGIN BACKUP

2. LGWR 重作日志写进程

把重做日志缓冲区的内容写出到日志文件（顺序写出，按时间）。
执行条件：
事务提交
先写日志后写数据的好处：
最快地保证数据不丢失
快
重作日志缓冲区三分之一满
每 3 秒
有一条超过 1MB 的重作日志记录
DBWn 进程操作前

6.CKPT 检查点进程

先执行 DBWR，写数据，再进行同步。
提供数据库同步性；
在执行检查点时通知 DBWn 执行写操作。

7.ARCn 归档进程（可选）

保存数据库的所有修改记录
当数据库在 ARCHIVELOG 模式的时候自动归档当前的重做日志记录。

8.LOGICAL STRUCTURE 逻辑结构

表明了物理空间的使用情况。
由表空间(tablespace)，段(segment)，区(extent)，和数据块(blocks)组成。

*&*cmd+回车
DBCA 命令
创建和删除数据库命令

/nolog 登陆 SQLPLUS 但不连接数据库
在用户名提示框中出入
可以有选择地登陆库和用户

5、OEM ORACLE 企业管理器

sysman 是 OEM 这个软件的用户
配置 OEM 的资料档案库：（repository）就是创建一个数据库用户的过程。
建的是在数据库中的创建一个数据库用户并把 OEM 的表放到这个用户下。
& emca 命令进入
管理企业中多个数据库
OMS Oracle Management server

◆ OEM 的注意事项：

（1）注意这些服务：
OracleOraHome92Agent
OracleOraHome92HTTPServer
OracleOraHome92ManagementServer
Performance Logs and Alerts
OracleServiceWNJ
OracleOraHome92TNSListener
（2）导出相当与把数据库中的数据复制到文件中

6.管理 ORA 实例

主要任务是了解数据库启动和关闭的过程。
初始化参数文件：
启动实例的时候用到。
连接 DBA 用户：CONNECT / AS SYSDBA

◆ STARTUP 启动命令

两种设置参数的方法：

1. 隐式：在文件中出现记录
2. 显式：在文件中没有记录，但是使用了 ORA 的默认值。

多个参数文件可以用于同一个数据库以便在不同的情况下优化数据库性能。

◆ 静态参数文件：PFILE

查看路径：D:\oracle\admin\wnj\pfile
文本文件；
可以使用编辑器进行修改，对当前实例无效，只能对下一次启动有效；
手动修改这个文件；
兼容性：compatible=9.2.0.0.0

◆ 稳固参数文件：SPFILE

WINDOWS 默认为 DATABASE 文件夹
查看路径：D:\oracle\ora92\database
UNIX 默认 DBS 文件夹

◆ create pfile from spfile;

用默认的 pfile 创建 默认的 spfile；

可以反方向执行：
create spfile from pfile;
修改：
alter system set 参数名=值 scope=both/memory/pflie
both 当前实例和 PFILE
memory 当前实例

7.启动过程：

1. NOMOUNT 实例启动阶段

◆数据库操作：
读取参数文件
根据参数设置分配空间
启动后台进程
打开报警文件，记录启动过程
◆ 用户操作：
创建数据库：创建文件
重建控制文件

2. MOUNT 数据库装载阶段

2. 数据库操作：
读取控制文件：得到数据库信息
将数据库和实例关联在一起
3. 用户操作：
操作数据库的归档和非归档模式
修改控制文件，重做数据文件的名称和路径
数据库的备份和恢复
改变归档模式

3. OPEN 打开数据库

4. 数据库操作：
读取文件的 SCN 号，判断数据一致性，是否打开或回滚。
检查数据文件状态
5. 用户操作：
查看数据
....
普通身份的用户是不能在 OPEN 之前登陆的。
SYSDBA 是通过口令文件和操作系统验证，可以在 OPEN 之前登陆。

8.启动命令：

STARTUP 命令

直接到 OPEN 阶段

1. 在关闭状态下执行

```
STARTUP open
STARTUP nomount
STARTUP mount
```

2. 切换命令： 不能跳级切换

```
alter database database_name open  [read only/read wirte]
alter database database_name nomount
alter database database_name mount
```

3. 关闭过程与启动逆向；

◆ 关闭数据库

shutdown 模式
shutdown normal 等待所有连接都断开才关闭（慢（爬））。
shutdown abort 放弃性关闭（最快（飞毛腿）），不可取。
shutdown transactional 事务性关闭（比较快（小跑））
shutdown immediate 立即性关闭，不等待事务，保证数据库的一致性（挺快了（冲刺）），一般都用。

以上模式执行后，新的连接允许形成。

9.监视诊断文件：

ALERTSID.LOG FILE 告警文件
对数据库的操作记录。
一个数据库只有一个
记录着数据库操作中的命令
主要事件的结果
被用于记录日常的操作
被用于数据库错误的诊断
每条记录都有相关的时间与其对应
文件位置：BACKGROUND_DUMP_DEST
查看命令：SHOW BACKGROUND_DUMP_DEST
最新的信息放在最后，最老的信息在最上边。

10.BACKGROUND TRACE FILES 后台进程跟踪文件

后台进程跟踪文件提供了后台进程检测出的错误信息。
被用于诊断和解决故障。

当后台进程遇到错误时生成。
以进程名称命名；
文件位置：BACKGROUND_DUMP_DEST

11.user TRACE FILES 用户跟踪文件

由用户进程产生
也可以由服务器产生
包括了用户 SQL 语句的执行统计信息，用来分析 SQL 语句的性能，从而进行调整。
包括了用户的错误信息
当用户遇到了会话错误时产生。
位置：USER_DUMP_DEST
大小：MAX_DUMP_FILE_SIZE 默认 10M

◆ 启动用户跟踪：
SQL_TRACE 为 TRUE 启动 为 FALSE 紧用

◆ 会话级：
ALTER SESSION SET SQL_TRACE = TRUE;
DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION

◆ 实例级：（不可取）
SQL_TRACE = TRUE

12.创建数据库

1. 创建前的准备：

*、一个具有以下权限的用户：（只有 SYSDBA && administrator 能操作）
- ◆ 操作系统认证
- ◆ 使用口令文件

*、启动实例所需要的足够内存；
*、满足计划数据库所需的足够磁盘空间；

◆ 使用口令文件：
WINDOWS 默认为 DATABASE 文件夹
查看路径：D:\oracle\ora92\database
UNIX 默认 DBS 文件夹

◆ 使用口令工具创建口令文件
$ orapwd file = $ oracle_home/dbs/orapwU15
password = admin entries = ;

◆ 在初始化参数文件中设置；

REMOTE_LOGIN_PASSWORDFILE 为 EXCLUSIVE
增加用户到口令文件中；
赋予合适的权限；
grant sysdba to user_name;

2. 创建方法：

（1）OUI Oracle Universal Installer
（2）Oracle 数据库配置助手
dbca 命令调出
图形化接口
基于 JAVA 语言
可以被 OUI 调用
也可以作为独立的应用来使用
可以建库
可以建脚本
可以建模板

（3）CREATE DATABASE 命令

13、UNIX 操作系统环境变量

ORACLE_BASE ORACLE 软件的基础目录（所有主目录都在基础目录之下，HOME 是 BASE 的子目录）
ORACLE_HOME ORACLE 软件的主目录（ORACLE 产品每一个软件的主目录）
ORACLE_SID （数据库编码）设置当前的数据库
ORA_NL32 us7ascll 语言环境支持
PATH 命令收缩路径
LD_LIBRARY_PATH ORA 中的 JAVA 所需要的库
字符集 语言环境支持 标准：us7ascii

14、手动创建数据库：

确定唯一的数据库名称和实例名
选择数据库的字符集
设置操作系统的环境变量
编辑/创建初始化参数文件
启动数据库实例
执行 CREATE DATABASE 命令
执行脚本创建数据字典并完成后续的创建步骤
根据需要创建其他的表空间

15、使用数据字典和动态性能视图

1. 数据字典

只能在启动到 OPEN 阶段可以访问
内建的数据库对象：
每个数据库的中心
对用户而言是只读的表，不能修改。

◆ 内容：

数据库的物理和逻辑结构
对象的定义和空间的分配
完整性约束条件

用户
角色
权限
审计

◆ 三种用法：

ORA 能做的：
服务器用来查询相关的信息
执行 DDL 的时候，服务器修改
用户能做的：用户查询

2. 数据字典的分类：

基表

◆ 字典视图

DBA：所有方案包含的信息 ；
ALL：用户可以访问的信息；
被授予对象权限的对象信息和用户自己的对象信息
USER：用户方案的信息；

```
select count(*) from user-table;
```

◆ 通用的：

DICTIONARY(dict) 数据字典的字典

```
select table_name from dict where table_name like '_%users%';
```

查询 user 下的字典视图

3. 动态性能表：

在启动的三个阶段都可以访问
虚表
当数据库在操作的时候，动态性能视图被不断地更新。
包含了来自内存和控制文件的信息

DBA 使用动态性能视图监视和调优数据库
动态性能视图被 SYS 用户拥有
使用 V$开头的同义词
在 V$FIXED_TABLE 中可以查到

◆ PL/SQL 包
包：一组相关的存储过程的集合。
数据库事件触发器

◆ 管理控制文件
一个二进制文件；
定义了当前的数据库的状态信息；
维护数据库的一致性；
- 需要：
  在数据库启动到 MOUNT 状态时
  在数据库操作的时候需要

只与一个数据库相关联；
丢失了控制文件的数据库需要恢复；
大小（由特性参数决定）在创建数据库的时候被初始化；
- 内容：
  数据库的名称和标识
  数据库创建时的时间
  表空间的名字
  数据文件、重做日志文件的名称和位置
  检查点的信息
  重作日志归档信息
  撤消段的起用和停用时间

重建控制文件在启动的 NOMOUNT 阶段。
路径：D:\oracle\oradata\wnj\CONTROL01.CTL

◆ 复用：将复用文件放在不同的磁盘上。
使用 SPFILE 复用控制文件：
修改参数文件 control_files
关闭数据库(除了 abort)
复制控制文件
启动数据库,show parameter control_files
◆ 使用 PFILE：
关闭数据库(除了 abort)
复制控制文件
修改参数文件 control_files
启动数据库,show parameter control_files

16、维护重做日志文件

组大小最好相等

分别把组的大小均等的两部分别放在两个磁盘中备以复用。

◆ 在线重做日志的特征：

1. 记录了所有数据的改变（存储对象的改变）
2. 为实例故障或介质故障提供必要的恢复机制
3. 重做日志文件被组织成日志组
4. ORA 数据库至少需要两个重做日志组。

重做日志文件也会有复用需求。

通过给一个组增加更多的组员文件来达到复用目的。

一个库三以上个组，一个组两个以上组员文件，分别放在两个不同的磁盘上（可以通过指定路径放在不同的磁盘上）。

◆ 建立多个组的好处：

b) 提高性能。

c) 将一次写的循环时间拉长，减少等待（等待的原因：检查点没执行完或归档没有完成）。

d) 给归档提供更长的时间。

建议：少做复用，多做组。

select * from v$log; 查看重做日志组。

select * from v$logfile; 查看重做日志组员。

日志文件大小只能通过删除重建才能修改。

◆ 强制切换

步骤：

(1)alter system switch logfile;

强制切换检查点

(2)alter system checkpoint;

检查点被强制执行的参数：

fast_start_mttr_target=600(秒)

◆ 增加新的重做日志组

```
alter database add logfile gruop 3
('$home/oradata/u01/log3a.rdo',
 '$home/oradata/u02/log3b.rdo'
)size 1M;
```

每个文件 1M。

◆ 为日志组增加新的重做日志文件

```
alter database add logfile member
'$home/oradata/u01/log1c.rdo' to group 1,
'$home/oradata/u01/log2c.rdo' to group 2,
'$home/oradata/u01/log3c.rdo' to group 3;
```

新增加的组员初始化状态为 invalid

◆ 删除日志组

当前和激活状态下的不能删除。

删除的只是文件的内容，文件本身得手动删除。

文件路径：D:\oracle\oradata\wnj\REDO01.LOG

◆ 重命名重定位日志文件

步骤：

（1）关闭数据库

（2）拷贝日志文件到新的位置

（3）数据库启动到 MOUNT 状态

（4）alter database rename file

'$home/oradata/u01/log3a.rdo'to'$home/oradata/u01/log1c.rdo';

（5）正常打开数据库。

增加新的文件并删除旧的日志文件

◆ 重做日志的清除：

alter database clear logfile group n;

初始化日志文件。

alter database clear unarchived logfile group n;

可以避免归档已经损坏的日志文件。

◆ 归档重作日志文件

写满的重作日志文件可以被归档

使用的好处：

i. 恢复上:一个备份的数据库，如果备份了所有的重作日志文件和归档文件就保证了数据库所有的提交了的事务都可以被恢复。

ii. 备份上：可以在数据库打开的状态下备份。

默认数据库状态的（NOARCHIVELOG）非归档模式。

◆ 归档日志文件：

归档日志文件被 ARCn（归档进程）自动产生或手动通过 SQL 语句产生

成功归档后：

在控制文件中记录了归档信息

记录：归档文件名，日志序列号，以及被归档的日志文件的最高和最底的 SCN 号

被写满的日志文件在完成下列事件之前不能被重用：

执行完检查点；

被 ARCn 进程归档

可以复用，必须被 DBA 管理

17、管理表空间和数据文件

ORA 的数据逻辑上是保存在表空间里，物理上是保存在数据文件中。

◆ 表空间：

1. 只能属于一个数据库；
2. 由一个或多个数据文件组成；
3. 更进一步被分成更细的逻辑单位存储（段、区、块）；

◆ 数据文件：

1. 只能属于一个数据库的一个表空间；

2. 存放方案对象的仓库；

◆表空间的类型

1. SYSTEM 表空间：
   只有一个；
   在创建数据库的时候创建；
   包含了数据字典；
   包含了 SYSTEM 的撤消段；
   最好不要存放对象（表、索引....）；
2. NON-SYSTEM 表空间：
   存放独立的段；
   易于执行空间管理；
   可以控制分配给用户的空间；

◆ 创建表空间：

CREATE TABLESPACE userdata DATAFILE（永久类型）
’路径+文件名 1.dbf’SIZE nM,’路径+文件名 2.dbf’SIZE nM,.....
EXTENT MANAGEMENT LOCAL [UNIFORM SIZE 128K];//注明为本地管理

永久类型表空间：存放固有的存储对象。

◆ 临时类型表空间：
   可以删除；
   存放排序时所用到的临时段。
   用于排序操作
   被多个用户共享(一个临时段的不同区)
   不包含任何永久对象
   CREATE TEMPORARY TABLESPACE temp_name
   TEMPFILE ’路径+文件名.dbf’ size nM EXTENT MANAGEMENT LOCAL [UNIFORM SIZE 128K];

◆ 默认的临时表空间：
   不允许被删除；
   不能被离线；
   不能将一个默认的临时表空间指定到一个永久类型的表空间上；
   设置数据库级的默认的临时表空间；
   减少使用 SYSTEM 表空间存储临时数据的需求；
   创建方法：
      CREATE DATABASE 的时候指定；
      ALTER DATABASE
         先创建一个临时表空间
         再指定 ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp_name;

◆ 查看临时表空间：select * from database_properties;

◆ 撤消类型表空间：
   只能存放回滚段（撤消段）。
   只能使用本地管理；
   CREATE UNDO TABLESPACE undo_name

DATAFILE ’路径+文件名.dbf’ size nM;

18、表空间的空间管理（区的管理）：

1、本地管理：

9i 默认管理方式；
性能好；
空闲区的信息记录在位图区中；
位图用于记录空闲空间（连续的 0 就表示有空闲）；
每一位相当于一个数据块或一组数据块；
位的值代表空闲或被使用；
空间管理方式：
自动分配空间；
用户分配空间，空间中每个区的大小等同；

2、数据字典管理表空间：

8i 以前的默认管理方式；
空闲区信息记录在数据字典中；
当区被分配和释放的时候，特定的表被更新；

3、存储参数：

initial 初始化大小
next 下一个区大小
pctincrease 区大小增量
minextents 最小区数（本地管理）
maxextents 最大区数（本地管理）

4、表空间状态：

- 读写(read write)：
- 只读(read only)：
  ALTER TABLESPACE space_name READ ONLY;
  引起检查点
  数据只能进行查询
  对象可以从表空间删除
  删除对象步骤：
  删除字典信息
  释放空间
- 联机(online)
- 脱机(offline)：
  离线的表空间不能访问到其包含的数据。

系统表空间不能脱机；
有激活回滚的表空间不能脱机；
默认的临时表空间不能脱机；

先修改读写或只读，再修改在线或离线；

5、查看表空间信息：

- 表空间信息：DBA_TABLESPACES
  V$TABLESPACE
- 数据文件信息：DBA_DATA_FILES
  V$DATAFILE
- 临时文件信息：DBA_TEMP_FILES
  V$TEMPFILE

6、重定义表空间的大小

*、改变数据文件的大小：
使用 AUTOEXTEND 选项设置自动；
CREATE TABLESPACE space_name DATAFILE 'file_name.dbf' SIZE nM
AUTOEXTEND ON NEXT nM MAXSIZE nM;
不利于性能；
查看 DBA_DATA_FILES
使用 ALTER DATABASE 手动修改；
ALTER DATABASE DATAFILE '文件名' RESIZE nM;
缩小是用限制的；
*、使用 ALTER TABLESPACE 命令增加数据文件；
ALTER TABLESPACE space_name ADD DATAFILE '文件名' SIZE nM;
均衡 I/O；
不能增加得太多；
只能增加不能删除；

7、操作表空间：

文件大小先设置为自动；
有计划地增加数据文件；
监控表空间；
按需求手动增加或减少表空间的大小；

8、移动数据文件：

*、 OPEN 状态下执行；
表空间必须离线；
目标数据文件必须存在；
ALTER TABLESPACE RENAME DATAFILE 'old_name'TO'new_name';

*、 MOUNT 状态下执行；
目标文件必须存在；
ALTER DATABASE RENAME FILE 'old_name'TO'new_name';

9、删除表空间：

不能删除的状态：是 SYSTEM 表空间；
有激活回滚段的表空间；
DROP TABLESPACE space_name INCLUDING CONTENTS AND DATAFILES [CASCADE CONSTRAINTS];

19、存储结构和关系

数据块
区：空间扩展的单位
段和数据文件的关系：
组成段的所有区必须在段表空间的数据文件上；

1、段类型：

1. 表
2. 表分区：
   分区表(partitioned table)：分了多个区的表，单表多段，存储海量数据；
   特点：
   容量大；性能好；
   分类：
   范围
   散列
   列表：按单点的值进行分区；
3. 簇表(cluster)：多表单段；
   表和表有共同的字段，公用字段只存储一次；
   减少存储空间，查询速度快；
   全表扫描的时间增长；
4. 索引
5. 索引组织表(index-organized table)：将字段索引和字段值放在一个表中，没有物理的 rowid。
   查找索引字段值非常快，其他字段的查询却非常慢；
   单表单段；
6. 索引分区
7. 回滚段：保证事务回滚（存放原始数据）；
8. 临时段
9. 大对象段
10. 签到表：多维表才会用到；
11. 引导段：初始化数据字典信息；

2、区：

是段在表空间上使用的连续空间
当段执行下列操作的时候分配区：
Created 段创建
Extended 段扩展
Altered 手动分配
当段执行下列操作的时候释放区：
Droped
Altered
Truncated

3、数据库块

最小的 I/O 单位
有一个或多个操作系统块组成
可以在表空间创建时分配
DB_BLOCK_SIZE 参数设置默认块的大小

4、9I 提供非标准块

数据库可以使用一个标准块和四个非标准块创建；
快的大小可以是 2KB 到 32KB 之间的任意一个 2 的 N 次方数；
好处：对数据的操作特性上，在做查询的时候减少 I/O 使用；

5、标准块大小

在数据库创建的时候通过 DB_BLOCK_SIZE parameter 参数设置。在数据库创建以后不能被修改。
SYSTEM 和 TEMPORARY 临时表空间必须使用
DB_CACHE_SIZE 设置了标准块对应的数据高速缓存中 DEFAULT 池的大小
DEFAULT 池的大小：
最小是 4M 或 16M 默认为 48M

6、非标准块的大小

使用下列参数配置对应的数据高速缓存：
DB_2K_CACHE_SIZE FOR 2 KB BLOCKS
DB_4K_CACHE_SIZE FOR 4 KB BLOCKS
DB_8K_CACHE_SIZE FOR 8 KB BLOCKS
DB_16K_CACHE_SIZE FOR 16 KB BLOCKS
DB_32K_CACHE_SIZE FOR 32 KB BLOCKS
DB_nK_CACHE_SIZE 不允许使用与标准块对应的参数。

7、数据块的内容：

1. 块头：
   自顶向下递增
   事务槽：通过事务槽来表示锁定的事务，如果想执行事务必须先获得事务槽。
2. 空闲空间：没有碎片
3. 数据：自底向上递增

8、块的空间利用参数：

INITRANS 初始事务数（事务槽的个数：平常并发事务的个数）
MAXTRANS 最大事务数（最大事务槽数）
PCTFREE 为 UPDATE 保留空间的百分比；如果没有 UPDATE 操作设置为零。
PCTUSED 标识数据块什么时候可以成为可用状态（FREELIST）
FREELIST 可用数据块状态
当块没有空间后，会构建新的块，并在原块里保留指针制向新块，但性能下降（行迁移）。

9、数据块管理：

设置好之后不能更改。
1、自动空间管理：（默认）
管理数据库的段中空闲的一种方法：
使用 bitmap 用 0 或 1 标识不可用或可用状态。
只能在表空间级上才能设置：
在 CREATE TABLESPACE 语句最后加上 SEGMENT SPACE MANAGEMENT AUTO;
限制：不能用于包含了 LOB(大对象)对象的表空间。
2、手动管理：
可以使用下面的参数手动配置数据块：
PCTFREE PCTUSED FREELIST
得到存储信息：
DBA_EXTENTS 查看区信息
DBA_SEGMENTS 查看段信息
DBA_TABLESPACES 查看表空间信息
DBA_DATA_FILES 查看数据文件信息
DBA_FREE_SPACE 查看空闲空间信息

10、管理回滚段（Undo）的数据

- 管理方法：
  1. 自动 Undo 管理
  2. 手动 Undo 管理
- 回滚段的原理：

用来暂时保存事务中的原始数据，至少保存到事务结束，保留到事务结束后回滚段中的空间被其他事务覆盖之前；

- 作用：

1. 事务回滚；
2. 事务恢复；
3. 读一致性；

在事务提交后查询，是在被查询表中查询的；

如果在事务执行中并发查询，数据库会在回滚段中取得数据，但正在并发查询的时候事务提交就会产生读一致性错误的问题。

◆ 特性：

1. 最少需要两个区；
2. 使用区是以循环的方式使用；
3. 一个事务只能使用一个回滚段（事务不能跨回滚段）；
4. 在一个回滚段上可以有多个事务（多个事务可以共享一个回滚段），每个事务使用不同的区；
5. 在一个区中可以同时有多个事务的数据，但只能有一个活动的事务。
6. 事务对区的使用是连续的，当他将要写的下一个区有活动的事务时，它就会执行扩展区操作；

长时间执行事务不提交，会造成阻碍会话。

减小事务大小，提交次数高一些，避免区扩展操作；

20、Undo 段的类型：

1.NON-SYSTEM 类型：

◆ 自动模式：需要一个 UNDO 类型的表空间。（默认）设计为最少 200M，一般是几个 G。

配置参数： UNDO_MANAGEMENT = AUTO/MANUAL 自动/手动

UNDO_TABLESPACE = tablespace_name

提供足够大的 UNDO 类型表空间；

可以创建多个回滚类型的表空间，但只有一个是被使用的，当做切换操作的时候才需要多个表空间。

切换：正在运行的事务可以切换。

可以在创建数据库的 CREATE DATABASE 命令中增加一个子句创建 UNDO 表空间：

```
CREATE DATABASE database_name
.......
undo tablespace space_name datafile 'filepath' size nM autoextend on
```

或

使用 CREATE UNDO TABLESPACE 命令创建：

```
create undo tablespace space_name datafile 'filepath' size nM;
```

◆ 手动模式：

需要一个永久类型的表空间

私有：被单个实例使用

公有：被任意实例使用

DEFERRED ：当表空间被修改为 OFFLINE ,IMMEDIATE，TEMPORARY，RECOVERY 状态时候出现。

2.SYSTEM 类型：

用于 SYSTEM 表空间的对象。

◆ 修改一个 UNDO 表空间：

alter tablespace space_name add datafile 'filepath' size nM autoextend on;

◆ 切换 UNDO 表空间：

可以从一个 UNDO 表空间切换到另一个。

在实例中一次只能使用一个 UNDO 表空间。

多个 UNDO 表空间可以存放在一个实例中，但只有一个是激活的。

使用 ALTER SYSTEM 命令动态切换 UNDO 表空间。

ALTER SYSTEM set UNDO_TABLESPACE = space_name;

◆ 删除 UNDO 表空间：

drop tablespace space_name;

UNDO 只是在当前实例没有使用它的时候才可以删除。

为了删除一个激活的 UNDO 表空间：

切换到一个新的 UNDO 表空间；

当所有表空间上的当前事务结束后删除表空间；

3.自动 UNDO 段管理的其他参数：

UNDO_SUPPRESS_ERRORS parameter

设置为 TRUE，可以压制在 AUTO 模式下执行手动管理命令时的错误。

UNDO_RETENTION parameter

控制为了保证读一致性而保留在提交后回滚段中数据的时间。

◆ 查看 UNDO 段信息：

SELECT... FROM V$UNDOSTAT;回滚段的使用频度

DBA_ROLLBACK_SEGS

21、管理表

rowid 格式：伪列。

◆ 扩展型(extended)

组成：数据对象号(data object number)、相对文件号(relative file number)、块号(block number)、行号(row number)（每个号 6 个字节）

由 18 个字母组织的，存成 60 个字节，代表一行数据的绝对地址

◆ 限制型(restrcted)

组成：块号(block number)、行号(row number)、文件号（file number）

exec DBMS_ROWID 包（backage）

SELECT ...DBMS_ROWID. ...FROM table_name;

1.创建表提示：

将表创建在独立的表空间中；

使用本地管理；

表使用标准的区大小来避免在表空间上产生碎片；

在创建各种文件的时候最好都有它们独立的表空间支持。

2.创建临时表

```
CREATE GLOBAL TEMPORARY TABLE
.......
```

◆ 特征：

用户创建的临时表是用户的会话独占的；
表在事务或会话过程中包含数据；
只会有结构而不会有存储；
在数据上不会得到 DML 锁；
对该表所做的 DML 操作不会写到日志上；
不用做 DELETE 或 TRUNCATE 操作，该表在内存中，当事务结束或会话结束时释放；
可以在临时表上创建索引、视图、触发器；

3.修改存储参数和块空间利用参数：

```
alter table owner.table_name
pctfree 30//块参数
pctused 50//块参数
storage(next 500k minextents 2 maxextents 120);//存储参数
```

4.手动分配区：

```
alter table hr.employees allocate extent(size 500k datafile 'filepath');
```

5.非分区表的重组

```
alter table owner.table move tablespace space_name;
```

当表被重组后，它的结构被重组，但内容没有影响；
约束条件不变，但与其对应的索引不存在了，所以重组之后的索引必须重建；
可以被用于移动表到新的表空间或者重组区；

6.删除列：

```
alter table owner.table_name drop
column col_name cascade constraints checkpoint 1000;
```

◆ checkpoint 1000 每一千行 执行一次检查点

检查点：完成同步，将写入日志的改变对应的数据写入数据文件中；
从每行中删除掉列占据的长度和数据，释放在数据块中占用的空间。
删除大表中的一列将花费比较长的时间。

7.重命名表中的一列：

```
alter table owner.table_name
```

```
rename column old_name to new_name;
```

8.标记列不再使用：

```
alter table owner.table_name
set unused column col_name cascade constraints;
```

9.删除不使用的列：

```
alter table owner.table_name
drop unused columns checkpoint 1000;
```

10.继续列的删除操作：

```
alter table owner.table_name
drop columns continue checkpoint 1000;
```

11.得到表的信息：

```
DBA_TABLES  DBA_OBJECTS
```

22、管理索引（index）

关键字+ROWID，排序!!!

1.索引的分类：

- 逻辑上：

Single column 单行索引
Concatenated 多行索引
Unique 唯一索引
NonUnique 非唯一索引
Function-based 函数索引
Domain 域索引

- 物理上：

Partitioned 分区索引
NonPartitioned 非分区索引
B-tree：
    Normal 正常型 B 树
    Rever Key 反转型 B 树
Bitmap 位图索引

2.索引结构：

B-tree：
适合与大量的增、删、改（OLTP）；

不能用包含 OR 操作符的查询；
适合高基数的列（唯一值多）
典型的树状结构；
每个结点都是数据块；
大多都是物理上一层、两层或三层不定，逻辑上三层；
叶子块数据是排序的，从左向右递增；
在分支块和根块中放的是索引的范围；

Bitmap:

适合与决策支持系统；
做 UPDATE 代价非常高；
非常适合 OR 操作符的查询；
基数比较少的时候才能建位图索引；
树型结构：

索引头
开始 ROWID，结束 ROWID（先列出索引的最大范围）
BITMAP

每一个 BIT 对应着一个 ROWID，它的值是 1 还是 0，如果是 1，表示着 BIT 对应的 ROWID 有值；

3.存储参数：

initial 初始化大小
next 下一个区大小
pctincrease 区大小增量
minextents 最小区数（本地管理）
maxextents 最大区数（本地管理）

4.创建 B-TREE 索引：

```
CREATE INDEX owner.index_name
ON owner.table_name(col_list)
PCTFREE n
STORAGE(INITAL nK NEXT nK PCTINCREASE n MAXEXTENTS n )
TABLESPACE space_name;
```

5.索引 PCTFREE 的变化：

索引不存在 PCTUSED；
PCTFREE 的含义是为新的插入预留的空间，在索引创建时保留；
在索引中是没有 UPDATE 的。

6.创建索引的提示：

平衡查询与 DML 操作的需求，不要建太多的索引；
将索引放在单独的表空间；
使用统一的区大小：5 块的整倍数或表空间的 MINIMUMEXTENT 的大小；

对于大的索引考虑使用 NOLOGGING（不产生日志文件，插入时速度快，索引不需要产生日志）；
索引的 INITRANS 应该比相应表的 INITRANS 设置的高一些；

7.创建位图索引：

参数： create_bitmap_areasize 在创建位图索引之前为其分配区域。

```
CREATE BITMAP INDEX owner.index_name
ON owner.table_name(col_list)
PCTFREE 30
STORAGE(INITAL 200K NEXT 200K PCTINCREASE 0 MAXEXTENTS 50)
TABLESPACE space_name;
```

8.改变索引参数:

```
alter index index_name storage(NEXT 400K MAXEXTENTS 50);
```

高水平线（HWM）：曾经被使用过的最后一块的位置，当对存储对象进行 TRUNCATE 时，
高水平线才会回到起点。
对全表扫描的时候，读到高水平线为止；

9.重建索引：

移动索引到另外的表空间；
通过清除删除掉的索引记录提高了空间的利用率，改善 PCTFREE 设置不良带来的问题；

```
alter index index_name rebuild tablespace space_name;
```

10.在线重建索引：（建议不使用）

```
alter index index_name rebuild online;
```

11.合并索引：

```
alter index index_name coalesce;
```

只能把同一个分支下的空间合并。

12.删除索引：

在数据做大量数据装载之前删除索引，之后再重建索引；
删除那些很少使用的索引，在需要的时候再创建（月底报表）；
删除并重建失效的索引；
删除不需要的索引（备选）；

```
drop index owner.index_name;
```

13.确定未使用的索引：

开始监视索引的使用：ALTER INDEX index_name MONITORING USAGE;

停止监视索引的使用：ALTER INDEX index_name NOMONITORING USAGE;

14.查看索引信息：

dba_indexes 提供索引的信息
dba_ind_columns 提供索引列的信息

23、管理口令安全和资源

- profiles：是命名的口令管理和资源限制的集合；
  不是物理上的文件；
  profiles 通过 CREATE USER 或 ALTER USER 命令分配给用户；
  可以被启用或禁用；
  可以使用 DEFAULT profile；
- profiles 提供的口令管理：
  口令历史，保证在规定的时间和次数以后重用以有的口令；
  允许最多的错误次数；
  口令的过期失效；
  口令的效验规则；
- 启动口令管理：
  通过使用 profiles 建立口令管理并分配给用户；
  通过 CREATE USER 或 ALTER USER 命令可以使用户的状态为 lock、unlock 或是 expired;
  口令的限制总是强制的；
  为了启动口令管理，以 SYS 用户身份执行 utlpwdmg.sql 脚本；
  存放脚本路径：oracle/ora92/rdbms/admin

ALTER USER user_name ACCOUNT LOCK; 手动锁定。
ALTER USER user_name ACCOUNT UNLOCK; 手动解锁。
ALTER USER user_name PASSWORD EXPIRED; 设置口令过期。

1.口令帐户锁定：

failed_login_attempts 在试图登陆多少次失败后，帐户被锁定。
password_lock_time 在由登陆失败的帐户被锁定的天数

2.自动锁定，可以手动解锁

3.口令的到期和过期：

password_life_time 口令到期的天数
password_grace_time 在口令到期后，从用户第一次成功登陆开始允许修改口令的天数；

4.口令历史：

password_reuse_time 口令在可以重用前必须经过的天数
password_reuse_max 口令在可以重用前必须被修改过的次数

这两个参数是互相排斥的。

5.口令的校验：

password_verify_function + function_name
在口令修改被确认前通过 PL/SQL 函数执行口令的复杂性校验。

6.用户提供的校验函数：

必须在 SYS 用户下创建，必须遵循下面的声明：

```
function_name
(   userid_parameter in varchar2(30)
    password_parameter in varchar2(30)
    old_password_parameter in varchar2(30)
)
return boolean
```

7.口令校验函数：

verify_function

8.创建 profile 口令设置：

```
create profile profile_name LINIT
failed_login_attempts 3
password_lock_time unlinited --无限
password_life_time 30
password_grace_time 5
password_reuse_time 30
password_verify_function verify_function;
```

9.修改 profile ：口令设置

```
alter profile DEFAULT limit
password_life_time 30
password_grace_time 5
password_reuse_time 30;
```

10.删除 profile： 口令设置

```
defaule profile 不能被删除;
cascade 从被授予的用户上移走 profile
drop profile profile_name;
drop profile profile_name cascade;
```

24、资源管理：

资源管理限制可以强制在会话级和调用级或者都可以强制限制;
可以通过 CREATE PROFILE 命令定义在 profiles 上;

1.启动资源限制通过：

resource_limit 初始化参数
alter system 命令
alter system resource_limit = true/false;

2.会话级参数：

cpu_per_session 总的 CPU 时间（单位：百分之一秒）
sessions_per_user 每个用户允许的并发会话数
connect_time 每个用户的连续会话时间（单位：分）
idle_time 挂起状态时间（单位：分）
logical_reads_per_session 每个用户能够使用的数据块数
private_sga 在 SGA 的私有空间数（单位：字节，只对共享服务器有效）

3.调用级参数：

cpu_per_call 每次调用的 CPU 时间
ogical_reads_per_call 每次调用读的数据块数（物理读和逻辑读）

4.创建 profile：资源配制

create profile profile_name limit
cpu_per_session 10000
sessions_per_user 2
idle_time 60
connect_time 480;

5.查看：

dba_users
dba_profiles

24、管理用户

1. 用户：

操作系统权限
用户口令
用户状态：锁定、未锁定
默认表空间（创建时设置）可选位置，不一定能放东西。
临时表空间
表空间配额（永久类型表空间才有）

如果用户创建对象的时候只是指定了默认表空间而不指定表空间配额的话，对象是不能被创建的。

2.数据库的方案：

是命名的对象集合；

一个用户被创建，方案也同时被创建；
一个用户只能用于一个方案关联；
用户名经常和方案交替使用；

3.创建用户的步骤：

确定用户用于存放对象的表空间；
确定每个表空间的配额；
指定默认表空间和临时表空间；
创建用户；
为用户分配权限和角色；

4.创建一个新的用户：数据库认证

```
create user user_name identified by password
DEFAULT tablespace space_name
temporary tablespace temp_table_name
quota nM on data
quota nM on users
quota nM on index
.....
password expire;
```

5.改变用户的表空间配额：

在下列情况下需要修改：
用户的表异常增长；
应用被增强并需要更多的表和索引；
对象被重组并被放在不同的表空间上；

```
ALTER USER user_name quota nM on user_name;
ALTER USER user_name quota 0 on user_name;
```

为 0：
不能再使用空间了；
原有空间可以使用；

6.删除用户：

```
drop user user_name cascade;
```

删除方案中包含的所有对象；
正在连接服务器的用户不能删除；

7.查看：

```
dba_users   dba_ts_quotas
```

25、管理权限

start restrict(启动到限制模式)
unlimited tablespace 无限空间配额

只能直接授予，不能传递授予

1. 两种用户权限：

系统：使用户可以在数据库中执行特定的任务；
对象：使用户可以操作访问特定的对象；

2.系统权限：

在权限中的 ANY 关键字表示用户在所有方案中都有权限
GRANT 将权限授予一个用户或一组用户
REVOKE 将权限从用户身上移走

3.授予系统权限：

grant priv_list to users with admin option;不及联删除

4.授予对象权限：

grant priv_list to users with grant option;及联删除
sysoper
startup
shutdown
alter database open | mount 备份控制文件
alter datadase backup controlfile to recover database 恢复数据库
alter database archivelog
restricted session
sysdba
sysoper privileges with admin option
create datadase
alter tablespace begin/end backup
restricted session
recover database until
07_dictionary_accessibility 参数：
控制 SYSTEM 权限的限制；
如果设置为 TRUE，可以访问 SYS 方案中的对象；
默认问 FALSE：确保能访问其他方案的用户不能访问 SYS 方案中的对象；

5.移除系统权限：

使用 REVOKE 命令从用户上去除系统权限；
具有 ADMIN OPTION 选项的用户可以移除系统权限；
只有用 GRANT 命令授予的权限可以被移除；
revoke create table from emi;

6.对象权限：

VIEW 没有 ALTER 权限；

7.移除对象权限：

移除对象权限的用户必须是要移除对象权限的对象的拥有者；
revoke select on owner.object from user_name;

8.查看：

SESSION_PRIVS 当前会话拥有的通过角色授予的权限信息；
DBA_SYS_PRIVS 关于被授予用户或角色的系统权限信息；
dba_tab_privs
dba_col_privs

26、管理角色

- 角色的好处：

简化了权限的管理
实现了动态权限管理
可以选择可用的权限

- 管理过程：

1.创建角色：

CREATE ROLE role_name;
CREATE ROLE role_name IDENTIFIED BY password;
创建的角色不属于任何一个用户；

2.赋予角色权限：

GRANT priv_list TO role_name;

3.将角色赋予用户；

grant role to user with admin option;
grant role_name1 to role_name2; 将角色赋予角色

4.设置用户的默认角色在需要的时候启用或禁用角色；

一个用户可以被分配许多角色
一个用户可以被指定默认角色
可以限制用户的默认角色
ALTER USER user_name DEFAULT ROLE role_name1,role_name2;
ALTER USER user_name DEFAULT ROLE all;
ALTER USER user_name DEFAULT ROLE all except role_name;除了这个角色其他的都授予
ALTER USER user_name DEFAULT ROLE none;

启用和禁用：
禁用角色将从一个用户身上暂时移走角色；

默认角色自动启用；
启用角色的时候可能需要口令；
启用角色暂时将角色授予用户；
set role role_name;
set role all except role_name;
set role all;
set role role_name1 identified by password,role_name2,role_name3;

5.移除角色；

从用户上移除角色需要 ADMIN OPTION 或者 GRANT ANY ROLE 权限；
revoke role_name from user_name;

6.删除角色；

从所有被授予的用户和角色上移除角色
从数据库中移除；
需要 ADMIN 或者 DROP ANY ROLE 权限；
DROP role role_name;

7.预定义角色：

connect,resource,dba 为了满足向后兼容性而保留
连接 资源 管理员
当授予 resource 角色给用户的时候，数据库自动把 unlimited tablespace 这个系统权限授予用户；
exp_full_database 导出数据库权限
imp_full_database 导入数据库权限
delete_catalog_role 在数据字典上的删除权限
execute_catalog_role 在数据字典包上的 execute 权限 ？
select_catalog_role 在数据字典上的查询权限

8.查看：

session_roles 查看当先会话可用的角色；
dba_roles 所有存在于数据库的角色
dba_role_privs 授予用户或角色的角色
role_rol_privs 授予角色的角色
dba_sys_privs 授予用户和角色的系统权限
role_sys_privs 授予角色的系统权限
role_tab_privs 授予角色的对象权限

高权限身份用户的权限跟角色无关，它是靠身份得到权限的；
带口令的角色

27、使用全球化支持

◆ 全球化支持特性：

语言支持
区域支持：地区支持，对不同地区的特定规则的支持
字符集支持
语言排序
信息支持
日期和时间格式
数字格式
货币格式

- 不同类型的编码方案：
  单字节字符集：
  7-bit 8-bit
  变长多字节字符集
  定长多字节字符集
  Unicode(AL32UTF8,AL16UTF16,UTF8)
  全球统一字符集：国际标准组织，能兼容大多数国家的字符集。
- 数据库字符集：VARCHAR VARCHAR2 CHAR
- 国家字符集： NVARCHAR、NVARCHAR2、NCHAR

- 设置服务器基于语言的行为：
  基于数据库服务器设置
  NLS_LANGUAGE SPECIFIES
  信息的显示语言
  天和月份的名称
  A.D,B.C,A.M,P.M 的符号
  默认的排序方式（二进制）
  NLS_TERRITORY SPECIFIES

28、基本的 ORA 网络服务器端配置

- 连接字符串：HOST：1521：SID
- 监听器：conn hr/hr@服务命名
- 服务命名：连接字符串的命名
- 服务端： 配置监听器 listener.ora
- 客户端： 1.tnsnames.ora 命名方法选择的配置文件（用什么方法来解析连接字符串）
  2.sqlnet.ora
- 监听进程：

特点：监听多种网络协议；
单个监听器可以监听多个数据库的连接；
多个监听器可以监听单个数据库；
监听器是有名称的，在同一台主机上的监听器不能重名；
不管有几个监听器，都只有一个监听文件存储它们的内容。

- 配置监听器：
  1.静态配置：
  由于 8I 以前的版本；
  需要配置 listener.ora；

使用 OEM 连接数据库必须使用静态配置；
listener.ora 的默认设置：
listener name listener
port 1521
pro*—*

2.动态的服务注册：
不需要配置 listener.ora 文件
监听器依赖于 PMON 进程

- 监听器处理连接的方法：
  1.传递会话：（专用会话）
  会话请求到监听器
  监听器判断如果没有问题，监听器通知数据库

  2.重定位会话：（多线程服务器：共享服务器）
  预先生成调度器进程和服务进程放到监听器中
  连接建立的时候服务进程才启用

- 服务名：是数据库对外的名称。
- 主机名称和 IP 地址会影响监听器。

- 共享服务器：共享连接/专用连接
- 专用服务器：专用连接

- 故障解决方法：
  1. 检查物理连接 PING
  2. 服务器端做本地连接
  3. lsnrctl status 检测监听器配置运行是否正常
  4. 客户端执行 TNSPING （命令：TNSPING +主机字符串）服务器名，检测服务器命名是否能连通
  5. 检测 TNSNAMES.ORA 配置是否正常。

三、PL/SQL

pl/sql program language 是能够进行一定程度控制的程序语句。将 SQL 语句嵌入到 ORA 程序语句中。
pl/sql developer4.1 pl/sql 开发工具
PRO*C ORA 提供的 C 语言的编辑器
SQLJ ORA 提供的 JAVA 的编辑器
Declaring Variables

- pl/sql 的存储程序单元：命名的 pl/sql 块，作为数据对象存储在数据字典中。
- 匿名的存储程序单元：临时的 pl/sql 语句，只对当前有效。
- 存储过程：一定执行某个操作，意味着数据或数据对象的改变。
- 函数：做计算，不能包含任何的数据操作，只能出现 SELECT 语句。

- 包：逻辑上相关的一组存储过程和函数的集合体。
- 触发器：当事件发生，就会触发，仍然是用 pl/sql 语句。

- 结构：
  1. DECLARE（可选）
     定义标识符，标识符：变量，常量，游标
  2. BEGIN：开始执行主体（必须）
     SQL 语句
     PL/SQL 语句
  3. EXCEPTION（可选）
     异常处理
  4. END：结束执行主体（必须）

- 程序头定义：

1. 匿名块：

```
[declare]
begin
statements
....
[exception]
end;
```

2. 存储过程：

```
PROCEDURE name
IS
begin
statements
....
[exception]
end;
```

3. 函数：

```
FUNCTION name
RETURN data_type
IS
begin
statements
RETURN value
....
[exception]
end;
```

1、创建 PL/SQL 语句的过程：

选择开发环境
写程序
编辑（语句）
编辑（逻辑）

执行

- 变量类型：
  - 单值变量（Scalar）
    - BOOLEAN :true, false, null.
  - 组合变量
  - 大对象
  - 参照变量
- 输出变量值：
  - 1.建表输出；2.使用包输出；
- 可以使用替换变量或绑定变量。
  - 1.绑定变量使用时： “: var_name”。
  - 2.替换变量使用时：“&var_name”。
- 变量定义规则：
  1. 命名规则；
  2. 如果使用 NOT NULL 必须给变量或常量赋值；
  3. 每行只能定义一个标识符；
  4. 赋值操作符 “:=”
  5. 变量名称在同一块内不能重名；
  6. 变量名称不要跟查询中的字段名称相同；

- %type 属性（也是一种声明单值变量的方法）
  - 已经声明好的属性
  - 以字段名称或声明好的变量作为前缀。
  - 保证变量的匹配关系。

- 显示变量输出命令

print var_name

在使用了 DBMS_OUTPUT.PUT_LINE()的时候，用 set serveroutput on。

例：

```
define manager_sal = 1000/var manager_sal =1000
declare
wc_sal  employees.salary %type;
begin
select salary into wc_sal from employees where manager_id is null;
wc_sal:=wc_sal+&/:manager_sal;
dbms_output.put_line(wc_sal);
end;
```

书写正确的执行语句，每一条语句必须有分号。

在语句中：

- 不可以使用组函数和 DECODE 语句；
- 其他的字符函数和转换函数都可以使用；

CHR（ASCII）将 ASCII 值转换成其对应的字符

赋值语句

条件判断语句：可以有 NOT AND OR 来连接单行比较操作符（>,<,=)。
循环控制语句
注释语句：1./* */ 2.--

2、PL/SQL 中的 SQL 语句

1. 查询语句：可以直接使用，语法和规则有改变。

```
select select_list into variable_name_list/record_name from table where condition;
```

◆ 规则：一个查询必须并且只能返回一条数据，无返回或多条数据都会出现异常；
如果想要得到多条记录就必须使用游标；

DML 操作和事务控制：可以像 SQLPLUS 一样使用
DDL 语句不能被直接执行，不可以直接使用

◆ 执行的使用方法：

通过 DBMS_SQL 包使用,适合动态 SQL；

```
execcute immediate 'DDL 语句';
```

2. 循环控制：

◆ Baisc loop

```
loop
  statement1;
.....
exit [when condition];//退出循环约束
end loop;
```

◆ While loop

```
while condition loop
statement1;
statement2;
.....
end loop;
```

◆ For loop

```
for counter in [reverse]//取反
lower_bound .. upper_bound loop
statement1;
statement2;
....
end loop;
```

counter 不需要声明，不能在 FOR 循环中为其赋值，但他能在循环中参加运算或赋予其他变量 counter 值。

例：

```
declare
empl_name employees.last_name %type;
empl_id employees.employee_id %type :=100;
empl_sal employees.salary %type;
```

```
begin
for i in 1 .. 10 loop
select last_name,salary into empl_name,empl_sal from employees
where employee_id=empl_id;
empl_id :=empl_id + i;
dbms_output.put_line (empl_name ||chr(32) ||empl_sal);
exit when empl_sal<5000;
end loop;
end;

declare
empl_name employees.last_name %type;
empl_id employees.employee_id %type :=100;
flag number :=1;
begin
while flag<=10 loop
select last_name into empl_name from employees
where employee_id=empl_id;
flag :=flag+1;
empl_id :=empl_id + flag;
dbms_output.put_line (empl_name);
end loop;
end;
```

◆嵌套循环要使用标签 << >>

records 组合类型

每一个域都可以是不同的数据类型；

使用其来记录表中一行的数据；

```
type type_name is record
 (field1,field2,field3,.....);
```

◆ %rowtype 参照一个表的结构。

```
declare
 type_name table_name %rowtype;
```

例：

```
declare
depart_type departments %rowtype;
begin
select * into depart_type from departments where department_id = 10;
dbms_output.put_line(depart_type.department_id||chr(32)||depart_type.department_na
me);
end;
```

index by tables 类型

◆ 组成：

（一）起主键作用的 BINARY_INTEGER 类型的字段。

（二）简单类型或 records 类型的字段。

不需要定义长度,但最大长度是由 BINARY_INTEGER 的最高值来决定(-32767~32768)。

◆ 语法：

```
TYPE type_name IS TABLE OF
col_type / variable %type / table.col %type[not null]
/table %rowtype
INDEX BY BINARY_INTEGER;
identifier type_name;
```

3. index by tables 中的方法：

identifier.functions 引用形式。

exists	(下标) 判断该下标位置有没有值
count	计算该数组中有值的个数
first	数组中第一个有值的下标
last	数组中最后一个有值的下标
prior	前一个
next	后一个
trim	截取数组
delete	删除

```
declare
type depart_sum_type is table of
departments %rowtype
INDEX BY BINARY_INTEGER;
depart_sum depart_sum_type;
begin
for i in 1 .. 20 loop
select * into depart_sum(i) from departments
where department_id=i*10;
if i<=80 then
dbms_output.put_line(depart_sum(i).department_name);
else
dbms_output.put_line(depart_sum(i).department_id);
end if;
end loop;
end;
```

4. SQL Cursor

一段私有的 SQL 内存区域；

◆ 类型：1、隐式游标 2、显式游标：

定义：CURSOR cursor_name is select_statement;

打开：OPEN cursor_name;

FETCH：

FETCH cursor_name INTO [var1,var2,....] / [record_name];

个数，顺序和类型都必须匹配；

关闭：CLOSE cursor_name;

如果想再次利用，必须重新打开。

◆游标的属性：

1. sql%rowcount SQL 语句影响到的行数，在显式里是指 FETCH 出的行数。
2. sql%found SQL 语句是否访问到数据的属性，在显式里是指 FETCH 是否访问到数据：返回 TRUE/FALSE(while loop)
3. sql%notfound SQL 语句是否没访问到数据的属性， 在显式里是指 FETCH 是否没访问到数据：返回 TRUE/FALSE(Basic loop)
4. sql%isopen 对显式有效，在隐式中始终为 FALSE

if not cursor_name %isopen then open cursor_name;

◆显式引用： cursor_name %属性，在关闭状态下也可以查看。

◆ 隐式引用： SQL%属性，如果有多条的 SQL 语句，该属性取出的是离 SQL%属性最近的那条 SQL 语句。

例：

```
declare
cursor empl_dep is
select last_name,department_name from employees,departments
where employees.department_id=departments.department_id;
empl_name employees.last_name %type;
depart_name departments.department_name %type;
begin
open empl_dep;
fetch empl_dep into empl_name,depart_name;
dbms_output.put_line(empl_name ||chr(32)||'work in'||chr(32)||depart_name);
while empl_dep%found loop
fetch empl_dep into empl_name,depart_name;
dbms_output.put_line(empl_name ||chr(32)||'work in'||chr(32)||depart_name);
end loop;
if empl_dep%isopen then
dbms_output.put_line(empl_dep%rowcount);
end if;
close empl_dep;
end;
```

5.FOR 循环的游标使用：

```
for record_name in cursor_name loop
statement1;
statement2;
......
end loop;
```

游标不需要打开，FETCH 和关闭，直接在循环中使用 record_name.col_name(子查询中的

col_name)就可以。

```
declare
cursor cur_empl_dep is
select last_name,department_name from employees,departments
where employees.department_id=departments.department_id;
begin
for re_empl_dep in cur_empl_dep loop
dbms_output.put_line(re_empl_dep.last_name||' work in
'||re_empl_dep.department_name);
end loop;
end;

begin
for re_empl_dep in (select last_name,department_name from employees,departments
where employees.department_id=departments.department_id) loop
dbms_output.put_line(re_empl_dep.last_name||' work in
'||re_empl_dep.department_name);
end loop;
end;
```

6.带参数的游标：

处理串行化数据（一个执行完才能执行下一个）。

```
CURSOR  cursor_name [(parameter_name datatype,.....)]
IS
select_statement;
OPEN cursor_name [(parameter_name datatype,.....)];
```

◆ 限制：不能实现多个游标的并行打开。

例：

```
declare
cursor dep_cur
(dep_id employees.department_id %type)
is
select employee_id,last_name,department_id from employees
where department_id=dep_id;
begin
for j in 1 .. 5 loop
for i in dep_cur(j*10) loop
dbms_output.put_line(i.employee_id||' : '||i.last_name||' in '||i.department_id);
end loop;
end loop;
end;
```

◆ 在查询的同时锁定数据：

select .... from .... for update [of column reference] [nowait];

[nowait] 如果得不到访问资源就立即返回。

如果要使用游标修改数据的话，一定要在游标定义中 SELECT 语句中加上 FOR UPDATE。

在游标执行过程中的 UPDATE 语句：
UPDATE....SET...WHERE CRRENT OF cursor_name;

7.异常处理

异常：在执行过程中出现的错误。
错误触发：语句执行错误（规则）。
显式触发：人为认定的异常，人为规定出现异常的数据范围（逻辑）。

8.预定义异常：

有编号，有名称。使用异常名称扑捉；
NO_DATA_FOUND
TOO_MANY_ROWS
INVALID_CURSOR
ZERO_DIVIDE
DUP_VAL_ON_INDEX 在索引上出现重复数；

```
exception
when NO_DATA_FOUND then
statement1;
statement2;
........;
when TOO_MANY_ROWS then
statement1;
statement2;
..........;
when others then
statement1;
statement2;
..........;
```

9.非预定义异常：

有编号，没名称。需要先定义名称再扑捉；
(1)在 DECLARE 部分定义一个异常的名称； exc_name EXCEPTION;
(2)把名称和异常编号连接起来； PARGMA EXCEPTION_INIT(exc_name,-exc_number);

3、函数：

SQLCODE :返回错误编号
SQLERRM :返回错误信息
写入错误日志的内容： 用户，时间，对象，操作，错误编号，错误信息。

◆ 用户定义异常：
（1）命名；
（2）raise 关键字在 BEGIN 后触发；

（3）捕捉；

◆ 语法：

```
exception
when exception1 or exception2 then
statement1;
statement2;
........;
when exception3 or exception4 then
statement1;
statement2;
...........;
when others then
statement1;
statement2;
...........;
```

能够判断出的异常放到 WHEN 语句中，将那些无法预测的异常放在 OTHERS 中做成错误日志表。在程序内部的在程序内部捕捉，程序外部的程序外部捕捉，只有都没捕捉到，才会传到环境中。

◆ 抛出异常：

RAISE_APPLICATION_ERROR

RAISE_APPLICATION_ERROR(自定义错误编号，自定义信息);

不会被 EXCEPTION 捕捉到，而是直接回显到界面上给用户看到。

在 EXCEPTION ...WHEN ...THEN 之前或之中都可以使用。

◆ 函数：

不能独立出现，只能作为表达式来使用。

```
create [or replace] function fun_name
[(parameter1 [mode] datatype,parameter2 [mode] datatype, ...)]
return datetype
is/as
pl/sql block;
```

只能返回一个且必须有一个返回类型。

在 pl/sql block 中必须有一个可以执行的 RETURN 子句。

只允许接受 IN 模式的参数。

函数可以返回 BOOLEAN 类型，但不能将返回 BOOLEAN 类型的函数应用到 SQL 语句中。

◆ 函数的限制：

只能出现查询语句；

不允许出现 DML 操作；

如果在对表 T 进行 DML 语句操作调用函数 F，该函数 F 不能对编辑表 T 进行查询；

不允许出现 DDL 语句；

例：

```
create or replace function sal_comm_fun(empl_sal number) return number is
Result number;
begin
```

```
    if empl_sal <= 1000 then
      return(empl_sal * 0);
    elsif  empl_sal <= 2000 then
      return(empl_sal * 0.1);
    elsif  empl_sal <= 5000 then
      return(empl_sal * 0.15 + empl_sal * 0.1);
    elsif empl_sal <= 20000 then
      return(empl_sal * 0.2 + empl_sal * 0.15 + empl_sal * 0.1);
    else
      return(empl_sal * 0.3 + empl_sal * 0.2 + empl_sal * 0.15 +
             empl_sal * 0.1);
    end if;
  end sal_comm_fun;
```

4、存储程序单元

存储过程(Procedure)

```
  create [or replace] procedure pro_name
  [(parameter1 [mode] datatype,parameter2 [mode] datatype, ...)]
  is/as
  .....//声明部分
  begin
  ....//函数主题
  exception
  ....//异常处理
  end;
```

◆ mode:

1. in 传入(形参)默认，可以有默认值；

```
create or replace procedure empl_sal_pro(empl_id employees.employee_id %type)
is
begin
update employees set salary = 170000 where employee_id = empl_id;
end empl_sal_pro;
exec empl_sal_pro(empl_id=> 100)
```

2. out(实参)先将初始值传入经过处理后在传出,必须定义名称；

先在 SQLPLUS 中定义 VAR 变量，然后执行 EXEC 将定义好的 VAR 变量 传入到存储过程中的参数位置。

```
SQL> create or replace procedure empl_info(empl_id
employees.employee_id%type,
  2  empl_name out employees.last_name%type,
  3  empl_sal  out employees.salary%type) is
   4  begin
  5  select last_name,salary into empl_name,empl_sal from employees
  6  where employee_id=empl_id;
  7  end empl_info;
  8  /
```

```
过程已创建。
SQL> exec empl_info(100,:name,:sal)
PL/SQL 过程已成功完成。
SQL> print name
NAME
--------------------------------
King

SQL> print sal

   SAL
----------
    24000


begin
 -- Call the procedure
 query_emp(p_id => :p_id,
       p_name => :p_name,
       p_salary => :p_salary,
           p_comm => :p_comm);
  end;
```

3. inout 先传入后传出，可以直接赋予默认值;

编辑并存储;

SHO ERR 查看编辑时错误的命令;

exec 执行存储过程命令;

删除：DROP procedure;

5、管理 PL/SQL 程序块：

管理 PL/SQL 程序块:

在用户自己的方案下有 CREATE 权限

对于其他用户的方案有 CREATE ANY 权限

存储过程，函数和包需要执行权限，触发器不需要。

当用户使用其他用户的存储过程或函数去访问其他用户的对象，

能否访问到对象取决与该存储过程或函数的拥有者的权限。

TUTHID CURRENT_USER 表示存储过程不依赖他的拥有者，而依赖与当前调用他的用户。

在 CREATE PROCEDURE 的最后添加。

◆ 查看信息:

user_objects;

user_source;

user_errors;

query_emp;对过程的描述。

6、包（package）

一组相关类型的变量，常量，游标，存储过程，函数的集合。

1.组成：

包头：声明部分,只声明 PUBLIC PROCEDURE/VARIABLE;

包体:程序实体,包含 PRIVATE PROCEDURE/VARIABLE 、PUBLIC PROCEDURE/VARIABLE 和 LOCAL VARIABLE;

包头没有包体是可以独立存在的。

◆ 包头：

```
CREATE [OR REPLACE] PACKAGE package_name
IS
public_type
PROCEDURE procdeure_name (parameter.... );
END package_name;
```

◆ 包体：

```
CREATE [OR REPLACE] PACKAGE BODY package_name
IS
private type and item declarations
subprogram bodies
end package_name;
```

2.构建没有包头的包：

为了在整个会话中的全局变量设计。

如果其他会话访问的话，和定义时候的回话得到的包的值有可能是不一样的。

包体的部分可以通过 WRAP 程序加密。

只有局部过程和被打包的过程才能 OVERLOAD（重载）。

◆ 前项声明：

将过程或函数的名称和参数放在包体的头部声明；

◆ 建立一次性的过程：

就是在包体中插入了一个匿名块，在包体执行的时候，该匿名块一定会先执行一次该匿名块，然后再执行包体中的其他部分。

3.SQL 中使用包函数的限制

函数中不能包含影响当前事务的语句；

如果在对表 T 进行 DML 语句操作调用函数 F，该函数 F 不能对编辑表 T 进行查询；

包中变量的稳定性：

PUBLIC VARIABLE 发生改变的情况：

重新建立会话
符合规则

4.与开发相关的系统包：

- DBMS_SQL
  （1）能过生成动态的 SQL 语句；
  （2）能过执行 DDL 语句；
- 包使用的主要步骤：
  OPEN_DURSOR
  PARSE
  BIND_VARIABLE
  EXECUTE
  FETCH_ROWS
  CLOSE_CURSOR

EXECUTE IMMEDIATE

```
EXECUTE IMMEDIATE dynamic_string
    [INTO {define_variable
        [, define_variable] ... | record}]
[USING [IN|OUT|IN OUT] bind_argument
            [, [IN|OUT|IN OUT] bind_argument] ... ];
```

- DBMS_DDL
  ALTER_COMPILE(object_type,owner,object_name);强制编辑
- DBMS_JOB
  设置用户的任务；
  在某个指定的时刻执行一定的操作；
  手动强制执行；
  挂起任务；
  提交
  删除
  修改
  要执行什么，可以跟语句或过程
  下一次的执行时间
  任务执行的时间间隔
  挂起
  手动强制执行
  查看：
  USER_JOBS;
- DBMS_OUTPUT
  PUT 输出多行数据
  NEW_LINE 起一个新行
  PUT_LINE 输出一行数据
  GET_LINE 得到一行信息
  GET_LINES 得到多行信息
- UTL_FILE
  对操作系统文件进行操作；

- DBMS_LOB
  对 LOB 对象的读和写
- UTL_HTTP
  可以把指定的网页的内容摘取下来

- 静态的 SQL 语句的执行过程：

分析　PARSE
绑定　BIND
执行　EXECUTE
取操作　FETCH

用户可以通过 DML 语句对 LONG 类型的数据可以直接访问；

7、触发器

触发器：
不能直接调用，必须得事件触发，一般情况下是与对象、数据库和方案有关。

1. 数据库触发器
2. 应用触发器
   当一个操作与另一个操作有密切关系的时候；
   触发器不要太大，如果代码量大，要把触发器放到一个可执行的存储过程中；
   不要在一个对象上建立太多的触发器；
3. 表触发器
   BEFORE 做数据校验
   AFTER 在操作以后对操作和数据进行记录
4. 视图触发器
   INSTEAD OF 对视图的操作替换成对视图的基表的操作
5. 行级触发器
   对多行的操作，每操作一行都会触发触发器。
6. 语句级触发器
   当一条语句执行的时候触发，执行一次触发一次，与操作的行数无关。
   WHEN CLAUSE 触发器执行条件

1.语句级：

```
CREATE [OR REPLACE] TRIGGER trigger_name
    timing(before/after)
      event1 [or event2 or event3] ON table_name
trigger_body
```

在同一个方案下不允许重名。

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OR DELETE ON employees
BEGIN
 IF (TO_CHAR (SYSDATE, 'HH24') NOT BETWEEN '08' AND '12') AND DELETING
 THEN
     RAISE_APPLICATION_ERROR (-20502,'You may delete from EMPLOYEES table
              only during business hours.');
```

```
   ELSIF (TO_CHAR (SYSDATE, 'HH24') NOT BETWEEN '12' AND '18') AND INSERTING THEN

     RAISE_APPLICATION_ERROR (-20500,'You may insert into
       EMPLOYEES table only during business hours.');

   ELSIF (TO_CHAR (SYSDATE, 'HH24') NOT BETWEEN '8' AND '18') AND UPDATING ('SALARY')
THEN

     RAISE_APPLICATION_ERROR (-20503,'You may update
         SALARY only during business hours.');
   ELSE
     RAISE_APPLICATION_ERROR (-20504,'You may update
           EMPLOYEES table only during normal hours.');
 END IF;
END;
```

2.行级触发器：

```
  CREATE [OR REPLACE] TRIGGER trigger_name
  timing
         event1 [OR event2 OR event3]
         ON table_name
      [REFERENCING OLD AS old / NEW AS new]
  FOR EACH ROW
      [WHEN (condition)] //执行条件
  trigger_body
```

:OLD.字段名/:NEW.字段名 引用格式。

例：

```
CREATE OR REPLACE TRIGGER restrict_salary
  BEFORE  INSERT OR [UPDATE OF] salary ON employees
    FOR EACH ROW
 WHEN (new.salary is not null)    //在出发器头里，new.salary 相当与本身的变量
    BEGIN
      IF  :NEW.salary>2*:old.salary   //在 TRIGGER_BODY 里，:NEW.salary 相当与一个绑
定变量
          or :NEW.salary <0.5*:old.salary
      THEN
        RAISE_APPLICATION_ERROR (-20202,'Employee cannot earn this amount');
     END IF;
END;
```

◆ 自建的日志表：

```
CREATE OR REPLACE TRIGGER empl_table
 AFTER DELETE OR INSERT OR UPDATE ON employees
 FOR EACH ROW
BEGIN
```

```
    INSERT INTO empl_table (user_name, timestamp,
       id, old_last_name, new_last_name, old_title,
       new_title, old_salary, new_salary)
    VALUES (USER, SYSDATE, :OLD.employee_id,
          :OLD.last_name, :NEW.last_name, :OLD.job_id,
          :NEW.job_id, :OLD.salary, :NEW.salary );
  END;
  /

  CREATE OR REPLACE TRIGGER derive_commission_pct
    BEFORE INSERT OR UPDATE OF salary ON employees
    FOR EACH ROW
    WHEN (NEW.job_id = 'SA_REP')
  BEGIN
    IF  INSERTING
       THEN :NEW.commission_pct := 0;
    ELSIF :OLD.commission_pct IS NULL
       THEN :NEW.commission_pct := 0;
    ELSE
      :NEW.commission_pct := :OLD.commission_pct + 0.05;
    END IF;
  END;
  /
```

3.INSTEAD OF TRIGGER：替换类型触发器

```
  CREATE [OR REPLACE] TRIGGER trigger_name
    INSTEAD OF
      event1 [OR event2 OR event3]
        ON view_name
     [REFERENCING OLD AS old | NEW AS new]
  [FOR EACH ROW]
  trigger_body
```

不允许在触发器中进行事务的操作。

4.DDL 触发器：

```
  CREATE [OR REPLACE] TRIGGER trigger_name
   timing//
      [ddl_event1 [OR ddl_event2 OR ...]]
      ON {DATABASE|SCHEMA}
      trigger_body
```

当用户在数据库中对方案执行 CREATE ALTER DROP 操作时触发。

5.系统事件触发器：

```
  CREATE [OR REPLACE] TRIGGER trigger_name
```

```
    timing
         [database_event1 [OR database_event2 OR ...]]
        ON {DATABASE|SCHEMA}
  trigger_body
```

在数据库启动后，和关闭前触发。
对数据库权限进行 CREATE ANY，ALTER ANY.....操作时触发。
如果是 ON SCHEMA ，该触发器是当前方案下的触发器，不能在连接其他用户登陆或断开时触发，ON DATABASE 可以。
sys_context('userenv','ip_address'/'isdba'/'HOST'/'os_user');
对日期格式的操作只能通过修改环境变量或用 TO_CHAR 来实现；
当触发器代码量大的时候，把触发器体整体放入一个存储过程中。

```
  CREATE [OR REPLACE] TRIGGER trigger_name
    timing
      event1 [OR event2 OR event3]
        ON table_name
    [REFERENCING OLD AS old | NEW AS new]
  [FOR EACH ROW]
    [WHEN condition]
  CALL procedure_name;

  CREATE OR REPLACE TRIGGER log_employee
  BEFORE INSERT ON EMPLOYEES
    CALL log_execution
  /
```

如果能用数据库的功能就用，不能的时候才考虑使用触发器。

```
 CREATE OR REPLACE TRIGGER owner_tri
 BEFORE DELETE OR INSERT OR UPDATE ON EMPLOYEES
 BEGIN
  if sys_context('userenv','HOST') not in ('305\wnj') then
  RAISE_APPLICATION_ERROR(-20202,'You cat not do anything on this SCHEMA!');
 end if;
  if user<>'HR' then
  RAISE_APPLICATION_ERROR(-20202,'You cat not do anything on this table!');
 end if;
 end;
```

8、审计

```
  AUDIT INSERT, UPDATE, DELETE
    ON  departments
    BY ACCESS
  WHENEVER SUCCESSFUL;
```

每一次访问都会产生审计信息，记录用户对数据库对象的操作，操作时间和次数。但不会记录对哪些数据进行操作，
使用触发器完成记录对哪些数据进行操作。

CHECK 约束只能做确切值的比较；

9、数据同步：

```
stream
CREATE SNAPSHOT emp_copy AS  SELECT * FROM employees@ny;
```

四、backup and recover 备份与恢复

1、备份与恢复概论：

◆ 备份恢复的相关问题：
*、防止数据库在运行中可能发生的各种故障
*、增加数据库的连续可用时间 Mean-Time-Between-Failures (MTBF)
*、减低数据库的恢复用时间 Mean-Time-To-Recover (MTTR)
*、最小化数据的丢失
◆ 故障类型：
*、语句失败
*、用户进程失败
*、实例失败
*、用户错误
*、介质故障
*、网络故障
◆ 引起语句失败的原因：
*、应用的逻辑错误
*、试图在表中存放不合法的数据
*、试图执行无权限的操作
*、试图创建表,但是已经超过了空间配额的限制
*、试图对表执行 INSERT 或者 UPDATE 操作,导致新区的分配,但是在表空间上已经没有足够的自由空间
◆ 解决语句的失败：
*、修改程序的逻辑流.
*、修改并重新执行 SQL 语句.
*、提供必要的数据库权限.
*、使用 ALTER USER 命令改变用户的配额.
*、为表空间增加新的空间.
*、Oracle9i 给用户了当因空间不足导致应用挂起时,通过分配空间而使挂起程序继续执行的能力.
◆ 用户进程失败的原因：
*、用户异常的断开了会话.
*、用户会话被非正常终止.
*、用户的程序触发了地址异常,导致了会话的终止.
◆ 解决用户进程的失败：

*、PMON 进程自动检测用户进程的非正常终止.
*、PMON 自动回滚事务并释放所有被用户占用的资源和锁.

◆ 用户错误：

就是连接到数据库的用户有意或无意地做了删除了某些不该删除的数据或更新了某些不该更新的数据等等不该做的事。

◆ 解决用户错误：

*、培训数据库用户.
*、从以前的备份中恢复.
*、从导出的文件中导入表.
*、使用 LogMiner 确定出现错误的时间.
*、执行基于时间的恢复.
*、使用 LogMiner 执行对象级的恢复.
*、使用 FlashBack 察看并修复历史数据.

◆ 介质故障的原因 ：

*、磁盘驱动器头损坏
*、读写数据文件时遇到了物理故障
*、文件被意外的删除了

◆ 解决介质故障：

*、恢复策略依赖于选择了哪种备份方法以及那些文件需要恢复.
*、如果可能,使用从上一次备份开始产生的所有归档重做日志文件执行恢复.

2、定义一个备份、恢复策略：

*、商业需求：
恢复所需时间最少(Mean-Time-To-Recover)
连续无故障时间最长(Mean-Time-Between-Failure)
备份策略的制定是一个持续的过程，需要不断的调整
*、操作需求：
24-hour 的连续操作
能不能在数据库运行时执行策略的测试,有没有完善的备份恢复文档
数据库的数据是不是不断变化的
*、技术上的考虑：
资源方面：硬件，软件，电源，以及时间
执行操作系统级数据文件的物理影像拷贝
执行数据库对象的逻辑拷贝
数据库的配置
事务的大小也影响着备份的频率
*、灾难恢复的问题：
如果遇到天灾人祸,对具有备份策略的数据库有多大的影响?
地震，洪水，火灾
机器完全丢失
软硬件的存储故障
数据库的关键人物不在了，例如数据库管理员
能不能周期性的测试备份恢复策略

根据以上四大方面的各种因素以及各个的不同情况来定义一个备份与恢复的策略。

3、数据库的同步:

*、所有的数据文件(除了离线状态和只读状态的)必须同步（数据文件的 SCN 号与控制文件中 SCN 号相同）数据库才可以打开.
*、同步是基于当前的检查点的.
*、应用重做日志文件中的修改记录以同步数据文件.
*、重做日志文件在实例恢复时被 Oracle 服务器自动使用.

◆ 实例恢复的过程 ：

1、出现数据不同步；
2、前滚(redo)；
3、在数据文件中存在提交的数据和未提交的数据；
4、回滚(undo)；
5、文件中只剩下提交的数据；

配置数据库的归档模式

◆ 数据库的两种模式：

非归档模式(noarchivelog) 如果对数据操作非常频繁的话一定要选择非归档模式。

归档模式(archivelog) 不需要关闭数据库就可以恢复数据库，也可以实现数据库的不完全恢复。

◆ 改变归档模式：

1、SHUTDOWN IMMEDIATE
2、STARTUP MOUNT
3、ALTER DATABASE ARCHIVELOG/[NOARCHIVELOG];
4、ALTER DATABASE OPEN;
5、在切换模式后做数据库的完全备份;

◆ 自动和手动归档：

自动：LOG_ARCHIVE_START=TRUE
手动：LOG_ARCHIVE_START=FALSE

◆ 设置多个 ARCn 进程:

动态参数 LOG_ARCHIVE_MAX_PROCESSES 控制在实例启动时启动的归档进程数.

最多可以设置 10 个 ARCn 归档进程.
ARCn 进程数可以使用 ALTER SYSTEM 命令改变.
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES = 3;
ALTER SYSTEM ARCHIVE LOG START [TO ‘?/dbs/arch’]; 切换为自动归档模式

使用 LOG_ARCHIVE_DEST_n 最多可以设置 10 个归档目的
使用 LOG_ARCHIVE_DEST_n 选项：

*、可以设置归档目的为 MANDATORY （强制的，归档不成功就 REOPEN）或者 OPTIONAL（可选的，默认）.
*、定义失败时重试等待的时间.

log_archive_dest_1="LOCATION=/archive MANDATORY REOPEN"
log_archive_dest_2="SERVICE=standby_db1MANDATORY REOPEN=600"
log_archive_dest_3="LOCATION=/archive2 OPTIONAL"

Log_archive_dest_1 必须为 MANDATORY，其他随意

设置在本地成功归档的最少个数：

LOG_ARCHIVE_MIN_SUCCEED_DEST 参数

ALTER SYSTEM LOG_ARCHIVE_MIN_SUCCEED_DEST = n [scope = both];

一个在线重做日志组仅仅在下面的条件满足时可被重用:

所有被设置为 mandatory 的目的都已成功归档

本地成功归档的目的数大于或等于 LOG_ARCHIVE_MIN_SUCCEED_DEST 参数

控制归档的目的:

*、归档目的可以通过 LOG_ARCHIVE_DEST_STATE_n 动态参数禁用.

ALTER SYSTEM SET log_archive_dest_state_3 = DEFER

*、也可以再次启用归档目的.

ALTER SYSTEM SET log_archive_dest_state_3 = ENABLE

使用 LOG_ARCHIVE_FORMAT 设置归档文件的命名格式，日志序列号和线程号是默认命名的一部分.

设置归档文件的命名格式:

/ORADATA/archive/ arch%s.arc

其中：

LOG_ARCHIVE_DEST_n 控制 /ORADATA/archive/

LOG_ARCHIVE_FORMAT 控制 arch%s.arc

◆ 得到关于归档的信息：

*、V$ARCHIVED_LOG 已经成功的归档日志

*、V$ARCHIVE_DEST 归档目的的信息

*、V$LOG_HISTORY 日志历史

*、V$DATABASE

```
Select log_mode from V$DATABASE;
```

*、V$ARCHIVE_PROCESSES

◆ 命令行：

```
ARCHIVE LOG LIST;
show parameter archive --AS SYSDBA connected
```

4、数据库的备份

1.物理备份与逻辑备份：

ORACLE 提供的 exp 和 imp 实用程序可以处理 ORACLE 数据库的逻辑备份和恢复。

exp 用与逻辑备份；

imp 负责恢复这些逻辑备份；

对于逻辑备份来说，时间点恢复是不可能的。

下边介绍的备份和恢复的方法都是物理备份与恢复,对于此处的了解会有助与更好的使用 RMAN。

◆ 术语：

1. 完整数据库备份 -- Whole database backup
   目的数据库可能是打开或者关闭状态
   备份所有的数据文件和控制文件
2. 部分数据库备份 -- Partial database backups
   表空间
   数据文件
   控制文件
3. 一致的备份 -- Consistent backups

在数据库没有启动的情况下的备份又称为冷备份，SCN 号相同。

4. 不一致的备份 -- Inconsistent backups

在数据库打开状态下备份又称为热备份，SCN 号不同。

◆ 察看视图以得到数据文件的信息：

V$DATAFILE
V$CONTROLFILE
V$LOGFILE
DBA_DATA_FILES

◆ 执行一致的完整数据库备份(NOARCHIVELOG 模式)：

1、SHUTDOWN IMMEDIATE
2、操作系统 COPY 所有的数据库数据文件、控制文件和联机重做日志文件；
3、重新启动数据库；

◆ 在两种模式(NOARCHIVELOG 和 ARCHIVELOG)下都可以进行脱机备份；

1. 打开数据库的备份的优势：
   *、维护了数据库的高可用性
   *、可以在表空间或数据库级执行
   *、提供了对不停顿的商业需求的支持
2. ARCHIVELOG 模式下的物理备份：冷备：
   1、SHUTDOWN IMMEDIATE
   2、操作系统 COPY 所有的数据库数据文件
   3、重新启动数据库
   4、使用 ALTER SYSTEM SWITCH LOGFILE 命令强制执行一个联机重做日志的切换。一旦归档了联机重做日志，那么就备份所有的归档日志；
   5、使用 ALTER DATADBASE BACKUP CONTROL FILE TO TRACE 命令和 ALTER DATABASE BACKUP CONTROLFILE TO ’file_path’ 命令创建一个控制文件的备份。

◆ 备份表空间和数据文件：

1、alter tablespace space_name begin backup;
2、backup db_file.dbf(操作系统 COPY)
3、alter tablespace space_name end backup;
4、使用 ALTER SYSTEM SWITCH LOGFILE 命令强制执行一个联机重做日志的切换。
5、一旦完成了日志的切换并归档了当前的联机重做日志，就备份左右的归档重做日志。

需要注意的是：日志切换和归档日志的备份是必须的，这是因为恢复操作必须应用在备份期间生成的所有重做。

SCN 冻结的表空间（BACKUP 状态）仍然可以对数据进行操作。

◆ 备份状态的信息：

V$BACKUP

◆ 只读表空间备份的问题：

*、当表空间被切换到只读状态后,只需要备份一次.

*、在只读表空间切换回读写状态后,重新加入到正常备份的策略中.

*、控制文件必须能够正确的表示出表空间的只读状态,否则必须执行数据库的恢复操作.

◆ 手工执行控制文件的备份：

创建二进制映像

ALTER DATABASE BACKUP CONTROLFILE TO ‘control1.bkp`;打开状态下创建文本跟踪文件

ALTER DATABASE BACKUP CONTROLFILE TO TRACE;

2.数据库的恢复

1.在 NOARCHIVELOG 模式下的恢复：

1. 完全依赖手动备份。
2. 必须恢复的文件：
   数据库数据文件；
   控制文件；
3. 可选的恢复文件：
   重做日志文件；
   参数文件；
   口令文件；

可以简单的恢复所有的数据库数据文件、控制文件和联机重做日志文件，然后在启动数据库。这种恢复只能恢复到恢复到最后备份的时间点，而不能恢复这个备份时间点后的任何更改。任何数据库中的一个数据文件损坏都必须恢复所有的数据文件。

2.在非归档方式不使用备份的重作日志进行恢复：

1.关闭数据库.

2.从最近的完整数据库备份中回复数据文件和控制文件.

3.执行基于 cancel(放弃)的恢复.

4.使用 RESETLOGS 选项打开数据库.

3.ARCHIVELOG 模式下的完全恢复：

概念：

◆ 完全恢复：
   使用重作日志记录或者增量备份以更新数据库到最接近当前的时间
   应用所有的重作日志记录

◆ 不完全恢复：

使用备份和重作日志记录以生成一个非当前版本的数据库

◆ 完全恢复的优点和缺点：

优点：

只需要回复丢失的文件

恢复所有数据到数据库故障的那一刻

恢复时间是回复丢失文件的时间与应用所有归档日志文件时间的总和

缺点：

必须有从备份开始的所有归档日志文件

◆ 决定哪个文件需要被恢复：

*、察看 V$RECOVER_FILE 视图以决定哪个数据文件需要被恢复

*、察看 V$ARCHIVED_LOG 以得到数据库的所有归档日志文件列表

*、察看 V$RECOVERY_LOG 得到所有在恢复中需要的归档日志文件

◆ 在恢复过程中使用归档日志文件：

1. 为了改变归档的位置：

使用 ALTER SYSTEM ARCHIVE LOG. . .命令.

2. 为了自动应用归档日志文件:

在开始介质恢复前执行 SET AUTORECOVERY ON 命令

当提示输入归档日志文件时输入 auto

使用 RECOVER AUTOMATIC. . . 命令.

假顶数据库故障至少没有损坏当前每个联机重做日志组中的一个成员和没有备份的任何归档重做日志，我们可以在 ARCHIVELOG 模式恢复故障点的数据库。

如果丢失了归档的重做日志或联机重做日志就需要执行某种形式的时间点恢复；

如果丢失了当前控制文件的所有副本，就需要恢复控制文件并执行不完全恢复；

1. 完全恢复步骤：

1、从备份中还原所有的数据库数据文件；

2、还原所以备份的归档的重做日志；

3、加载数据库(start mount)；

4、恢复数据库(recover database)；

5、ORACLE 提示应用归档的重做日志中的重做，在提示符下简单地输入 AUTO，ORACLE 会自动应用 所有重做日志；

6、一旦应用了所有的重做日志，就可以打开恢复的数据库(alter database open)；

ARCHIVELOG 模式下的表空间恢复和数据文件恢复：

2. 在数据库加载或数据库打开可执行表空间恢复和数据文件恢复。

2.1 在打开阶段执行表空间恢复步骤：

1、使表空间脱机(alter tablespace offline)；

2、还原与要恢复的表空间相关联的所有数据文件;

3、恢复表空间(recover tablespace)；

4、一旦完成了恢复，使表空间联机(alter tablespace online)；

2.2 恢复数据文件步骤：

1、使数据文件脱机(alter database datafile 'file_path' offline)；

2、还原所有要恢复的数据文件;

3、恢复数据文件(recover datafile);
4、一旦完成了恢复，就可以是数据文件联机(alter database datafile 'file_path' online);

3.恢复一个从未备份过的数据文件：

*、丢失的数据文件从未备份过
*、如果丢失的是一个系统表空间的数据文件,则不能使用这种方法

◆ 步骤：
1、是数据文件或表空间 OFFLINE;
2、应用归档日志重新创建数据文件; ???
3、open database;
4、是表空间或数据文件 ONLINE;

◆ 控制文件的丢失：

如果在下列情况,可能需要重建控制文件:
所有的控制文件由于故障而丢失
数据库的名字需要被改变
控制文件的当前设置需要被改变

◆ 恢复控制文件：

恢复丢失的控制文件的方法:
使用当前的控制文件
创建一个新的控制文件

四种情况下重建控制文件：
数据库一些特性参数要改变的时候;
数据库需要重命名的时候;
控制文件全部丢失的时候;
使用一个备份的控制文件

4、ARCHIVELOG 模式下的不完全恢复：

◆ 执行不完全恢复的原因：

由于归档文件丢失导致完全恢复失败.
所有的控制文件丢失.
所有未归档的重作日志文件和数据文件丢失.
用户错误
一个重要的表被删除.
不正确的数据被提交.

◆ 执行不完全恢复的注意事项：
必须仔细的遵循步骤执行.
在恢复的前后都要执行备份.
在恢复完成后一定要检验恢复是否成功.
备份数据库,删掉以前的归档日志文件.

注意这些的原因主要在与如果不完全恢复执行的不好很可能造成数据库的不要一致从而不能正常地启动数据库.

1. 用户管理的执行不完全恢复的过程 ：

1、关闭数据库, 执行完整的数据库备份.必须备份控制文件和重作日志文件.

2、还原所有数据文件. 不要回复控制文件,重作日志文件,口令文件,或者参数文件.
3、装载数据库.
4、恢复数据文件到失败的时间点.RECOVER DATABASE UNTIL TIME ‘error_date’[using backup controlfile];
5、使用 RESETLOGS 选项打开数据库.ALTER DATABASE OPEN RESETLOGS;
6、校验.
7、执行关闭数据库的备份.

◆ 丢失了当前的重做日志文件：
1、如果数据库是关闭的:
2、尝试打开数据库.
3、找到当前的日志序列号.
4、恢复数据库直到 cancel.
5、如果必要删除、重建日志文件.
6、使用 RESETLOGS 选项打开数据库.
7、执行完整数据库备份.

2. ARCHIVELOG 模式下的时间点恢复：

如果要恢复某个时间点的表空间，我们需要恢复相同时间点的整个数据库(除非执行表空间的时间点恢复，但那是另一种恢复形式)
执行步骤(简述):
1、从备份中恢复所有数据库数据文件，这个备份在要恢复数据库的时间点之前结束。
2、使用如 recover database until time ’10-10-2002 21:00:00’的命令并且应用所需的重做日志，恢复选定时间点的数据库；

3. 使用 SCN 号恢复数据库：

1、从备份中恢复所有数据库数据文件，这个备份在要恢复数据库的时间点之前结束。
2、使用如 recover database until change ’221122’ 的命令并且应用所需的重做日志，恢复选定 SCN 的数 据库；
3、一旦完成恢复，打开数据库；

◆ 应用数据库的更改并在应用指定的归档的重做日志之后手动取消进程：
1、从备份中恢复所有数据库数据文件，这个备份在要恢复数据库的时间点之前结束。
2、使用 recover database until cancel 命令并且应用所需的重做日志，恢复选定时间点的数据库。应用了最后一个归档的重做日志后，可以简单的执行 cancel 命令来结束日志应用;
3、一旦完成恢复，打开数据库；

执行时间点恢复(或有关这一问题的任何恢复)时一定要牢记数据库一致性概念.