第 1 章

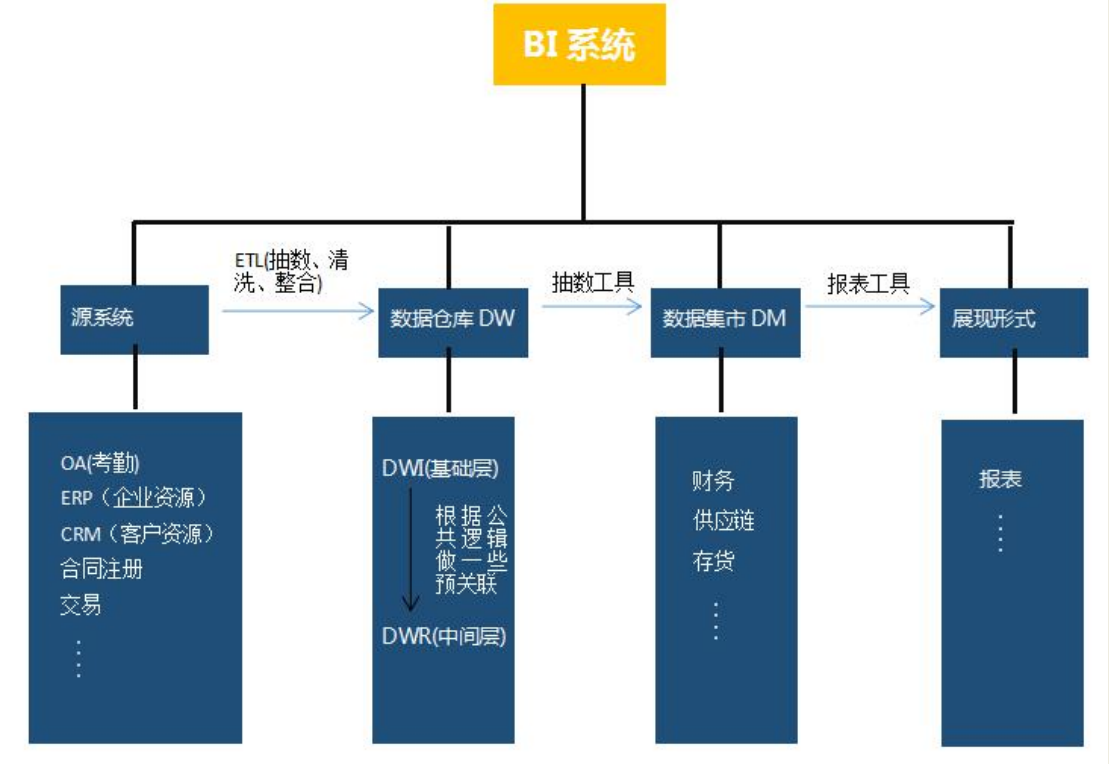
概述

1. BI 介绍

商业智能（BI，Business Intelligence）。商业智能的概念最早在 1996 年提出。当时将商业智能定义为一类由数据仓库（或数据集市）、查询报表、数据分析、数据挖掘、数据备份和恢复等部分组成的、以帮助企业决策为目的技术及其应用。

目前，商业智能通常被理解为将企业中现有的数据转化为知识，帮助企业做出明智的业务经营决策的工具。商务智能系统中的数据来自企业其他业务系统。例如商贸型企业，其商务智能系统数据包括业务系统的订单、库存、交易账目、客户和供应商信息等，以及企业所处行业和竞争对手的数据、其他外部环境数据。而这些数据可能来自企业的 CRM、SCM 等业务系统。

商业智能的关键是从许多来自不同的企业运作系统的数据中提取出有用的数据并进行清理，以保证数据的正确性，然后经过抽取（Extraction）、转换（Transformation）和装载（Load），即 ETL 过程，合并到一个企业级的数据仓库里，从而得到企业数据的一个全局视图，在此基础上利用合适的查询和分析工具、数据挖掘工具、OLAP 工具等对其进行分析和处理（这时信息变为辅助决策的知识），最后将知识呈现给管理者，为管理者的决策过程提供支持。



»源系统也是数据库，可以做简单统计，若要做复杂统计则需要与其他源系统关联多表，为了避免源系统之间的频繁交互，因此需要建立数据仓库。
»中间层：根据公共逻辑，对需要被重复关联的表提前做一些关联
»为什么会有数据集市？因为每个部门关心不一样。用于支撑报表。
»报表用于帮助企业决策

2. Oracle 介绍与安装

Oracle 公司是全球最大的信息管理软件及服务供应商，成立于 1997 年，主要的业务是推动电子商务平台的搭建，Oracle 公司有自己的服务器、数据库、开发工具、编程语言，在行业软件上还有企业资源计划（ERP）软件、客户关系管理（CRM）软件、人力资源管理（HCM）软件等大型管理系统。Oracle 是一家综合性的国际大公司，也是最有实力与微软公司在技术上一较高低的公司之一。

2.1 Oracle 安装

2.2 Oracle 默认用户

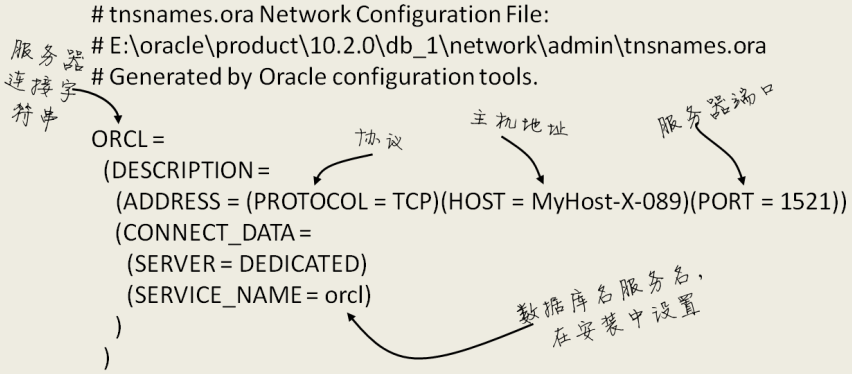
数据库创建完毕后，需要设置数据库的默认用户。Oracle 中为管理员预置了两个用户分别是 SYS 和 SYSTEM。同时 Oracle 为程序测试提供了一个普通用户 scott，初始密码为 TIGER，口令管理中，可以对数据库用户设置密码，设置是否锁定。 Oracle 客户端使用用户名和密码登录 Oracle 系统后才能对数据库操作。

默认的用户中，SYS 和 SYSTEM 用户是没有锁定的，安装成功后可以直接使用，SCOTT 用户默认为锁定状态，因此不能直接使用，需要把 SCOTT 用户设定为非锁定状态才能正常使用。

SCOTT 用户自带的四张表：EMP(员工)、DEPT(部门)、SALGRADE(工资等级)、BONUS(奖金)

2.3 服务器连接

客户端工具可以根据“服务器连接字符串”对服务器进行连接，有了连接字符串后客户端就可以像操作本机一样操作远程数据库，因此“服务器连接字符串”的配置也叫本地网络服务配置，该配置文件在 Oracle 安装目录下的：D:\app\xlovely\product\11.2.0\dbhome_1network/ADMIN/ tnsnames.ora。该文件是一个文本文件，用记事本打开后如下所示：



3. PL/SQL Developer 工具

在实际 Oracle 开发中，经常使用一个功能强大的第三方工具：“PL/SQL Developer”工

具。PL/SQL Developer 基本上可以实现 Oracle 开发中的任何操作。它运行在客户端时必须先安装 Oracle 客户端，并且通过网络配置向导配置网络服务名后才能正常与服务器连接。

图 20 PL/SQL Developer

- **sysdba**： 即数据库管理员，权限包括：打开数据库服务器、关闭数据库服务器、备份数据库、恢复数据库、日志归档、会话限制、管理功能、创建数据库。sys 用户必须用 sysdba 身份才能登录，system 用户可以用普通身份登录。
- **sysoper**：即数据库操作员，权限包括：打开数据库服务器、关闭数据库服务器、备份数据库、恢复数据库、日志归档、会话限制。
- **normal**：即普通用户，权限只有查询某些数据表的数据。默认的身份是 normal 用户。

4. Oracle 用户和权限

4.1 用户和权限

Oracle 中，一般不会轻易在一个服务器上创建多个数据库，在一个数据库中，不同的项目由不同的用户访问，每一个用户拥有自身创建的数据库对象，因此用户的概念在 Oracle 中非常重要。Oracle 的用户可以用 CREATE USER 命令来创建。其语法是：

语法结构：创建用户

```
CREATE USER 用户名 IDENTIFIED BY 口令 [ACCOUNT LOCK|UNLOCK]
```

语法解析：

LOCK|UNLOCK 创建用户时是否锁定，默认为锁定状态。锁定的用户无法正常的登录进行数据库操作。

代码演示：创建用户

```
CREATE USER jerry
IDENTIFIED BY tom
ACCOUNT UNLOCK
```

尽管用户成功创建，但是还不能正常的登录 Oracle 数据库系统，因为该用户还没有任何权限。如果用户能够正常登录，至少需要 CREATE SESSION 系统权限。

Oracle 用户对数据库管理或对象操作的权利，分为系统权限和数据库对象权限。系统权限比如：CREATE SESSION，CREATE TABLE 等，拥有系统权限的用户，允许拥有相应的系统操作。数据库对象权限，比如对表中的数据进行增删改操作等，拥有数据库对象权限的用户可以对所拥有的对象进行对应的操作。

4.2 角色

还有一个概念就是数据库角色（role），数据库角色就是若干个系统权限的集合。下面介绍几个常用角色：

- CONNECT 角色，主要应用在临时用户，特别是那些不需要建表的用户，通常只赋予他们 CONNECT role。CONNECT 是使用 Oracle 的简单权限，拥有 CONNECT 角色的用户，可以与服务器建立连接会话（session，客户端对服务器连接，称为会话）。
- RESOURCE 角色，更可靠和正式的数据库用户可以授予 RESOURCE role。RESOURCE 提供给用户另外的权限以创建他们自己的表、序列、过程（procedure）、触发器（trigger）、索引（index）等。
- DBA 角色，DBA role 拥有所有的系统权限----包括无限制的空间限额和给其他用户授予各种权限的能力。用户 SYSTEM 拥有 DBA 角色。

一般情况下，一个普通的用户（如 SCOTT），拥有 CONNECT 和 RESOURCE 两个角色即可进行常规的数据库开发工作。

可以把某个权限授予某个角色，可以把权限、角色授予某个用户。系统权限只能由 DBA 用户授权，对象权限由拥有该对象的用户授权，授权语法是：

语法结构：授权

```
GRANT角色|权限 TO 用户（角色）
```

代码演示：授权

```
GRANT CONNECT TO jerry;
GRANT CREATE TABLE TO CONNECT;
```

语法结构：其他操作

```
//回收权限
REVOKE 角色|权限 FROM 用户（角色）
//修改用户的密码
ALTER USER 用户名 IDENTIFIED BY 新密码
//修改用户处于锁定（非锁定）状态
ALTER USER 用户名 ACCOUNT LOCK|UNLOCK
```

Oracle 数据库中，默认情况下，所有系统的数据，SQL 关键字等都是大写的，在操作过程中，Oracle 会自动把这些内容转换为大写，因此用户操作时不需考虑大小写问题，一般情况下，为了良好的程序风格，程序中建议关键字用大写，非关键字可以使用小写。

第 2 章

SQL 数据操作和查询

1. SQL 简介

SQL：结构化查询语言(Structured Query Language)简称 SQL，是一种特殊目的的编程语言，是一种数据库查询和 程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统；同时也是数据库脚本文件的扩展名。

在 Oracle 开发中，客户端把 SQL 语句发送给服务器，服务器对 SQL 语句进行编译、执行，把执行的结果返回给客户端。Oracle SQL 语句由如下命令组成：

- 数据定义语言（DDL），包括 CREATE（创建）命令、ALTER（修改）命令、DROP（删除）命令等。
- 数据操纵语言（DML），包括 INSERT（插入）命令、UPDATE（更新）命令、DELETE（删除）命令等。
- 数据查询语言（DQL），包括基本查询语句、Order By 子句、Group By 子句等。
- 事务控制语言（TCL），包括 COMMIT（提交）命令、ROLLBACK（回滚）命令。
- 数据控制语言（DCL），GRANT（授权）命令、REVOKE（撤销）命令。

目前主流的数据库产品（比如：SQL Server、Oracle）都支持标准的 SQL 语句。数据定义语言，表的增删改操作，数据的简单查询，事务的提交和回滚，权限的授权和撤销等，Oracle 与 SQL Server 在操作上基本一致。

2. 查询

2.1 查询结构

数据查询是用 SELECT 命令从数据库的表中提取信息。

SELECT 语句的语法是：

```
SELECT *|列名|表达式
FROM 表名
WHERE 条件
GROUP BY 列名
HAVING 条件
ORDER BY 列名 [ASC/DESC]
```

2.2 查询顺序

1. FROM
2. WHERE
3. GROUP BY
4. HAVING

5. SELECT

6. ORDER BY

语法解析：

1. ***表示表中的所有列。**
2. **列名可以选择若干个表中的列名，各个列表中间用逗号分隔。**
3. **表达式可以是列名、函数、常数等组成的表达式。**
4. **WHERE 子句是查询的条件。**
5. **GROUP BY ----按列分组，与聚合函数联用。**
6. **HAVING----分组结果的筛选条件。**
7. **ORDER BY 要求在查询的结果中排序，默认是升序 ASC，降序为 DESC。**

2.3 聚合函数

Oracle SQL 提供了用于执行特定操作的专用函数。这些函数大大增强了 SQL 语言的功能。函数可以接受零个或者多个输入参数，并返回一个输出结果。包括单行函数、聚合函数、分析函数等。下面介绍聚合函数，单行函数和分析函数在以后的章节细述。

聚合函数：聚合函数同时可以对多行数据进行操作，并返回一个结果。比如 SUM(x)返回结果集中 x 列的总合。

✧ 聚合函数-----与 group by 使用。

聚合函数同时对一组数据进行操作，返回一行结果，比如计算一组数据的总和，平均值等。

名称	作用	语法
AVG	平均值	AVG(表达式)
SUM	求和	SUM(表达式)
MIN、MAX	最小值、最大值	MIN(表达式)、MAX(表达式)
COUNT	数据统计	COUNT(表达式)

表 5 聚合函数

```
--数值类型值，五个聚合函数都可以用
select max(sal),min(sal),sum(sal),avg(sal),count(sal) from emp;
--字符类型值
select max(ename),min(ename),count(ename) from emp;
 -日期类型值
 select max(hiredate),min(hiredate),count(hiredate) from emp;
 select avg(sysdate - hiredate) from emp;
 count
 select count(empno),count(comm),count(1),count(*),count('a') from emp;
```

注意：不使用分组则把整个表作为一组

【例 1】查询 EMP 表中各部门工资大于 2000 的员工人数，并且按人数从高到低排列

```
SELECT E.DEPTNO, COUNT(1) CT1
  FROM EMP E
 WHERE E.SAL > 2000
 GROUP BY E.DEPTNO
 ORDER BY CT1 DESC
```

【例 2】查询 EMP 表中员工人数大于 3 的所在部门的部门编号，及对应的人数

```
SELECT E.DEPTNO, COUNT(E.DEPTNO)
  FROM EMP E
 GROUP BY E.DEPTNO
HAVING COUNT(E.DEPTNO) > 3
```

3. 创建表和约束

3.1 Oracle 常用数据类型

类型	含义
CHAR(length)	存储固定长度的字符串。参数 length 指定了长度，如果存储的字符串长度小于 length，用空格填充。默认长度是 1，最长不超过 2000 字节。
VARCHAR2(length)	存储可变长度的字符串。length 指定了该字符串的最大长度。默认长度是 1，最长不超过 4000 字符。
NUMBER(p，s)	既可以存储浮点数，也可以存储整数，p 表示数字的最大位数（如果是小数包括整数部分和小数部分，p 默认是 38 为），s 是指小数位数。
DATE	存储日期和时间，存储纪元、4 位年、月、日、时、分、秒，存储时间从公元前 4712 年 1 月 1 日到公元后 4712 年 12 月 31 日。

表 1　Oracle 的部分数据类型

各种数据类型的表达形式：

字符型用单引号加字符表示，例如，’ ABC’

数字型直接用阿拉伯数字表示，例如，123

日期型不能直接表示，必须使用函数转换，

例如，DATE’ 2016-12-31’ ，TO_DATE(’ 20161231201237’ ,’ YYYYMMDD 24HH:MI:SS’)

各种数据类型的不同特点：

字符型可以进行拼接，

数字型可以进行算术运算，

日期型也能进行算术运算，但是只能日期减日期，或者日期加减数字

各种数据类型比较大小的方式：

字符型比较第一个字符的 ASCII 码的大小,继续第二个的大小

数字型比较数值的大小

日期型也是直接比较大小，越早的时间越小

3.2 创建表

数据库中的数据是以表的形式存储的，每一个表都被一个模式（或用户）所拥有，因此表示一种最基本的数据库模式对象。创建表时，Oracle 在一个指定的表空间中为表分配存储空间。

Oracle 创建表同 SQL Server 一样，使用 CREATE TABLE 命令来完成。可以直接创建表，也可以根据结果集创建表。

语法结构：直接创建表

```
CREATE TABLE 表名
(列名 数据类型 [,
 列名 数据类型]...
)
```

语法结构：根据结果集创建表

```
CREATE TABLE 表名 AS SELECT 语句
```

使用上面命令创建的新表中，不存在任何约束，并且把查询的数据一起插入到新表中。如果只复制表结构，只需使查询的条件不成立（比如 where 1=2），就不会查询从出任何数据，从而复制一个表结构。

语法结构：复制表结构

```
CREATE TABLE 表名1 AS SELECT * FROM 表名2 WHERE 1=2;
```

语法结构：删除表

```
DROP TABLE 表名
```

语法结构：添加列

```
ALTER TABLE 表名 ADD 列名 数据类型
```

语法结构：修改列类型

```
ALTER TABLE 表名 MODIFY 列名 数据类型
```

语法结构：修改列名

```
ALTER TABLE 表名 RENAME COLUMN 旧列名 TO 新列名
```

语法结构：删除列

```
ALTER TABLE 表名 DROP COLUMN 列名
```

语法结构：修改表名

```
ALTER TABLE 表名 RENAME TO 新表名
```

【例 1】创建一个学生表(student_1),表中包括学号(sno)、姓名(sname)、性别(ssex)、年龄(sage)、出生日期(sbirthday)

```
CREATE TABLE student_1(
sno VARCHAR2(10),
sname VARCHAR2(30),
ssex VARCHAR2(2),
sage NUMBER(3),
sbirthday DATE
)
```

3.3 表的约束

表的约束是 Oracle 数据库中应用在表数据上的一系列强制性规则。当向已创建的表中插入数据或修改表中的数据时，必须满足表的完整性约束所规定的条件。例如，学生的性别必须是“男”或“女”，各个学生的学号不得相同等。在设计表的结构是，应该充分考虑在表上需要施加的完整性约束。表的完整性约束既可以在创建表时制定，也可以在表创建之后再指定。可以对一个或多个字段进行约束。

按照约束用途分类：

1.PRIMARY KEY：主键约束

2.FOREIGN KEY：外键约束

3.CHECK：检查约束

4.UNIQUE：唯一约束

5.NOT NULL：非空约束

表创建完之后

创建约束则使用如下命令：

语法格式：ALTER TABLE 命令

```
ALTER TABLE 表名 ADD CONSTRAINT 约束名 约束内容。
```

语法格式：添加主键约束

```
ALTER TABLE 表名 ADD CONSTRAINT 约束名 PRIMARY KEY(列名 1[,列名 2...])
```

语法格式：添加外键约束

```
ALTER TABLE 主表名 ADD CONSTRAINT 约束名 FOREIGN KEY(列名 1[,列名
2...]) REFERENCES 从表名(列名 1[,列名 2...])
```

语法格式：添加 CHECK 约束

```
ALTER TABLE 表名 ADD CONSTRAINT 约束名 CHECK(条件)
```

语法格式：添加唯一约束

```
ALTER TABLE 表名 ADD CONSTRAINT 约束名 UNIQUE(列名)
```

语法格式：添加非空约束

```
ALTER TABLE 表名 MODIFY 列名 NOT NULL
```

语法格式：删除约束

```
ALTER TABLE 表名 DROP CONSTRAINT 约束名
```

【例 1】为学生表(student_1)学号添加主键约束，姓名添加非空约束，性别添加检查约束

```
ALTER TABLE student_1 ADD CONSTRAINT PK_SNO PRIMARY KEY(sno);
ALTER TABLE student_1 MODIFY sname NOT NULL;
ALTER TABLE student_1 ADD CONSTRAINT CK_SNO CHECK(ssex='男' OR ssex='女');
```

注意：

1. 一张表只有一个主键，主键非空且唯一
2. 外键只能依赖于另一张表的主键（例如 EMP 表中的 DEPTNO 依赖于 DEPT 表中的 DEPTNO）
3. 约束不能修改，只能删除重建

约束也可以在创建表的时候添加

```
CREATE TABLE student_1(
sno VARCHAR2(10) PRIMARY KEY,
sname VARCHAR2(30) NOT NULL,
ssex VARCHAR2(2) CHECK(ssex='男' OR ssex='女'),
sage NUMBER(3),
sbirthday DATE
)
```

3.4 临时表

会话级临时表：因为这这个临时表中的数据和你的当前会话有关系，当你当前 SESSION 不退出的情况下，临时表中的数据就还存在，而当你退出当前 SESSION 的时候，临时表中的数据就全部没有了，当然这个时候你如果以另外一个 SESSION 登陆的时候是看不到另外一个 SESSION 中插入到临时表中的数据的。即两个不同的 SESSION 所插入的数据是互不相干的。当某一个 SESSION 退出之后临时表中的数据就被截断（TRUNCATE TABLE，即数据清空）了。

会话级的临时表创建方法：

```
CREATE GLOBAL TEMPORARY TABLE TABLE_NAME
(COL1 TYPE1,COL2 TYPE2...) ON COMMIT PRESERVE ROWS;
```

举例：

```
CREATE GLOBAL TEMPORARY TABLE STUDENT
(STU_ID NUMBER(5),
CLASS_ID   NUMBER(5),
STU_NAME VARCHAR2(8),
STU_MEMO VARCHAR2(200)) ON COMMIT PRESERVE ROWS ;
```

2）事务级临时表：是指该临时表与事务相关，当进行事务提交或者事务回滚的时候，临时表中的数据将自行被截断，其他的内容和会话级的临时表的一致（包括退出 SESSION 的时候，事务级的临时表也会被自动截断）。

事务级临时表的创建方法：

```
CREATE GLOBAL TEMPORARY TABLE TABLE_NAME
(COL1 TYPE1,COL2 TYPE2...) ON COMMIT DELETE ROWS;
```

举例：

```
CREATE GLOBAL TEMPORARY TABLE CLASSES
(CLASS_ID NUMBER(5),
CLASS_NAME VARCHAR2(8),
CLASS_MEMO VARCHAR2(200)) ON COMMIT DELETE ROWS ;
```

3）两种临时表的异同

相同点：两种表都不能永久的保存记录。他们都是用临时表空间。

不同点：会话级别只有当会话结束临时表中的数据才会被截断，而且事务级临时表则不管是COMMIT、ROLLBACK 或者是会话结束，临时表中的数据都将被截断。

4）什么时候使用临时表

把复杂的逻辑拆分开来，用临时表储存中间结果，以方便后面的逻辑处理。程序执行过程中可能需要存放一些临时的数据，这些数据在整个程序的会话过程中都需要用的等等。

4. 数据操作语言（DML）

数据操纵语言（DML）用于对数据库的表中数据进行添加、修改、删除和 SELECT…For UPDATE(后面专门学习该查询)操作。

4.1 数据插入

用 INSERT 命令完成对数据的插入。

语法结构：数据插入

```
INSERT INTO 表名(列名 1 名 2……) VALUES (值 1，值 2……)
```

语法解析：

列名可以省略。当省略列名时，默认是表中的所有列名，列名顺序为表定义中列的先后顺序。值的数量和顺序要与列名的数量和顺序一致。值的类型与列名的类型一致。

【例】往学生表(student_1)里插一条记录,学号(s001),姓名(SASA),性别(女),出生日期(1995.08.20)

```
INSERT INTO student_1 VALUES（'s001','SASA','女',23,TO_DATE(19950820,'YYYYMMDD'))
```

在 Oracle 中，一个 INSERT 命令可以把一个结果集一次性插入到一张表中。使用的语句是：INSERT INTO 表 SELECT 子句，如下示例：

语法结构：INSERT 向表中插入一个结果集

```
INSERT INTO 表名1(列名1，列名2……) 查询结果集;
```

在这种语法下，要求结果集中每一列的数据类型必须与表中的每一列的数据类型一致，结果集中的列的数量与表中的列的数量一致。

4.2 更新数据

Oracle 在表中更新数据的语法是：

语法结构：UPDATE 操作 1

```
UPDATE 表名 SET 列名 1=值，列名 2=值…… WHERE 条件
--注意：WHERE 条件没下的话则是全表更新
```

【例】在学生表(student_1)里更改一条记录,将学号(s001)的学生性别改为男

```
UPDATE student_1 SET ssex='男' WHERE sno='s001'
```

4.3 删除数据

Oracle 在表中删除数据的语法是：

语法结构：DELETE 操作

```
DELETE FROM 表名 WHERE 条件
```

【例】删除学生表(student_1)里学号为 s001 的记录

```
DELETE FROM student_1 WHERE sno='s001'
```

4.4 TRUNCATE（DDL 命令）

在数据库操作中，TRUNCATE 可以把表中的所有数据一次性全部删除，语法是：

语法结构：TRUNCATE

```
TRUNCATE TABLE 表名
```

TRUNCATE 和 DELETE 都能把表中的数据全部删除，他们的区别是：

1. TRUNCATE 是 DDL 命令，删除的数据不能恢复；DELETE 命令是 DML 命令，删除后的数据可以通过日志文件恢复。
2. 如果一个表中数据记录很多，TRUNCATE 相对 DELETE 速度快。

由于 TRUNCATE 命令比较危险，因此在实际开发中，TRUNCATE 命令慎用。

Oracle 默认安装中，已经创建了一个 SCOTT 用户，该用户下有四张表分别是：雇员表（EMP），部门表（DEPT），工资登记表和奖金表，请参考本章后面的附表。接下来很多操作都是在该用户下完成的。

5. 操作符

Oracle 开发中，依然存在算术运算，关系运算，和逻辑运算。

5.1 算术运算

Oracle 中的算术运算符，只有+、-、*、/四个，其中除号(/)的结果是浮点数。求余运算只能借助函数：MOD(x,y)：返回 x 除以 y 的余数。

【例】每名员工年终奖是 2000 元，请显示基本工资在 2000 元以上的员工的月工资，年总工资。

```
SELECT E.ENAME, E.SAL, (E.SAL * 12 + 2000) FROM EMP E WHERE E.SAL > 2000;
```

5.2 关系运算和逻辑运算

Oracle 中 Where 子句经中经常见到关系运算和逻辑运算。

常见的关系运算有：

运算符	说明	运算符	说明
=	等于	>	大于
<>或者!=	不等于	<=	小于或者等于
<	小于	>=	大于或者等于

表 3　Oracle 的关系运算符

三个逻辑运算符优先级：NOT>AND>OR

5.3 字符串连接操作符（||）

在 Oracle 中，字符串的连接用双竖线（||）表示。

【例】在 EMP 表中，查询工资在 2000 元以上的姓名以及工作。

```
SELECT (ENAME || 'is a ' || JOB) AS "Employee Details"
  FROM EMP
 WHERE SAL > 2000;
```

代码解析：

Oracle 中字符串可以用单引号，存在特殊字符的时候，必须用双引号。

高级查询

6.1 DISTINCT 操作

在 Oracle 查询中结果中，可能出现若干行相同的情况，那么可以使用 DISTINCT 消除重复行。

【例】DISTINCT 消除重复行（GROUP BY 性能较好）

```
SELECT DISTINCT DEPTNO FROM EMP;
```

6.2 NULL 操作

如果某条记录中有缺少的数据值，就是空值（NULL 值）。空值不是 0 或者空格，空值是指未赋值、未知或不可用的值。任何数据类型的列都可以包括 NULL 值，除非该列被定义为非空或者主键。

在查询条件中 NULL 值用 IS NULL 作条件，非 NULL 值用 IS NOT NULL 做条件。

空值的一些特性：

1、 空值跟任何值进行算术运算，得到的结果都为空值

2、 空值跟任何值进行关系运算，得到的结果都为不成立

3、 空值不参与任何聚合运算

4、 排序的时候，空值永远是最大的

【例】查询 EMP 表中没有发奖金的员工。

```
SELECT * FROM EMP E WHERE COMM IS NULL;
```

6.3 IN 操作

在 Where 子句中可以使用 IN 操作符来查询其列值在指定的列表中的行。

【例】查询出工作职责是 SALESMAN、PRESIDENT 或者 ANALYST 的员工。条件有两种表示方法：

```
1. WHERE job = 'SALESMAN ' OR job = 'PRESIDENT ' OR job = 'ANALYST '
2. WHERE job IN ('SALESMAN', 'PRESIDENT', 'ANALYST')
```

代码演示：IN 操作

```
SELECT E.ENAME,E.JOB,E.SAL
FROM EMP E
WHERE E.JOB IN ('SALESMAN', 'PRESIDENT', 'ANALYST');
```

ENAME	JOB	SAL
ALLEN	SALESMAN	1600
WARD	SALESMAN	1250
MARTIN	SALESMAN	1250
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
TURNER	SALESMAN	1500
FORD	ANALYST	3000

对应 IN 操作的还有 NOT IN，用法一样，结果相反。

6.4 BETWEEN…AND…

在 WHERE 子句中，可以使用 BETWEEN 操作符来查询列值包含在指定区间内的行。

【例】查询工资从 1000 到 2000 之间的员工。

可以使用传统方法：

```
WHERE SAL>=1000 AND SAL<=2000
```

也可以使用：

```
WHERE SAL BETWEEN 1000 AND 2000
```

BWTWEEN 操作所指定的范围也包括边界。

代码演示：BETWEEN 操作

```
SELECT E.ENAME, E.JOB, E.SAL FROM EMP E WHERE E.SAL BETWEEN 1000 AND 2000;
```

6.5 LIKE 模糊查询

在一些查询时，可能把握不准需要查询的确切值，比如百度搜索时输入关键字即可查询出相关的结果，这种查询称为模糊查询。模糊查询使用 LIKE 关键字通过字符匹配检索出所需要的数据行。字符匹配操作可以使用通配符“%”和“_”：

- %：表示零个或者多个任意字符。
- _：代表一个任意字符。

通配符表达式	说明
’S%’	以 S 开头的字符串。

'_S%'	第二个字符时 S 的字符串。

表 4　通配符示例

【例】显示员工名称以 J 开头以 S 结尾的员工的姓名、工资和工资。

```
SELECT E.ENAME, E.JOB, E.SAL
FROM EMP E
 WHERE E.ENAME LIKE 'J%S';
```

6.6 集合运算

集合运算就是将两个或者多个结果集组合成为一个结果集。集合运算包括：

- INTERSECT(交集)，返回两个查询共有的记录。
- UNION ALL(并集)，返回各个查询的所有记录，包括重复记录。
- UNION(并集)，返回各个查询的所有记录，不包括重复记录。
- MINUS(补集)，返回第一个查询检索出的记录减去第二个查询检索出的记录之后剩余的记录。

当使用集合操作的时候，要**注意**：

1. 查询所返回的列数以及列的类型必须匹配，列名可以不同。
2. 只有 UNION ALL 不会去重。其他三个都需要排序后去重，性能比较差。

【例】查询出 dept 表中哪个部门下没有员工。只需求出 dept 表中的部门号和 emp 表中的部门号的补集即可。

```
SELECT D.DEPTNO FROM DEPT D
MINUS
SELECT E.DEPTNO FROM EMP E;
```

6.7 连接查询

在实际应用中，单表查询较少，经常是从多张表中查询数据，这就需要多表连接查询。通过各个表之间共同列的关联性，可以查询存放在多个表中的不同实体的信息，将多个表以某个或某列为条件进行连接操作而检索出关联数据的过程称为连接查询。

包括内联接(inner join)、外联接(outer join)。Oracle 中对两个表或者若干表之间的外联接用（+）表示。

6.7.1.内连接(inner join)：inner 可省略

语法结构：内连接写法 1（标准写法）

```
SELECT *|列名|表达式[ AS 列别名]
FROM 表名 1 [表别名] [INNER] JOIN 表名 2 [表别名]……
ON 关联条件
```

语法结构：内连接写法 2（Oracle 写法）

```
SELECT *|列名|表达式[ AS 列别名]
FROM 表名 1 [表别名],表名 2 [表别名]……
WHERE 关联条件
```

注意：标准写法和 Oracle 写法只是写法不同，执行机制相同。

【例 1】查询部门名称为'SALES'的所有员工信息

标准写法：

```
SELECT *
  FROM EMP E
 INNER JOIN DEPT D
  ON E.DEPTNO = D.DEPTNO
 WHERE D.DNAME = 'SALES';
```

Oracle 写法：

```
SELECT *
   FROM EMP E, DEPT D
 WHERE E.DEPTNO = D.DEPTNO
    AND D.DNAME = 'SALES'
```

【例 2】列出 EMP 表所有员工的姓名，工资，以及对应的工资等级

```
SELECT E.ENAME, S.GRADE
   FROM EMP E, SALGRADE S
 WHERE E.SAL BETWEEN S.LOSAL AND S.HISAL
```

发散：关联时，一条记录对应多条就会发散。

统计粒度：数据库中数据的细化和综合程度。细化程度越高，粒度越小。

两个表关联，粒度大的会发散。

6.7.2.外连接(outer join)：outer 可省略

--左外关联(left outer join)

--右外关联(right outer join)

--全外关联（full outer join）

语法结构：左外关联

```
SELECT *|列名|表达式[ AS 列别名]
FROM 表名 1 [表别名] LEFT [OUTER] JOIN 表名 2 [表别名]
ON 关联条件
```

语法结构：右外关联

```
SELECT *|列名|表达式[ AS 列别名]
FROM 表名 1 [表别名] RIGHT [OUTER] JOIN 表名 2 [表别名]
ON 关联条件
```

语法结构：外关联（Oracle 写法）

```
SELECT *|列名|表达式[ AS 列别名]
FROM 表名 1 [表别名],表名 2 [表别名]……
WHERE 关联条件
```

注意：关联条件字段加了（+）的为从表，不加（+）的为主表。

【例】查询各部门名称，及对应的经理名称

标准写法：

```
SELECT D.DNAME, E.ENAME
  FROM DEPT D
  LEFT JOIN EMP E
    ON D.DEPTNO = E.DEPTNO
   AND E.JOB = 'MANAGER'        --WHERE E.JOB='MANAGER'
```

注意：

WHERE 和 ON 的区别：不管是 WHERE 还是 ON，Oracle 都会把能过滤的条件先过滤掉，再关联。但两者区别在于，如果是内关联，两种结果相同，如果是外关联，结果会不同，ON 会保留主表的所有信息，而 WHERE 可能会过滤掉部分主表信息。

Oracle 写法：

```
SELECT D.DNAME, E.ENAME
  FROM DEPT D, EMP E
 WHERE D.DEPTNO = E.DEPTNO(+)
   AND E.JOB(+) = 'MANAGER'
```

注意：从表的每个字段都要加（+）

第 3 章

子查询和常用函数

1. 子查询

子查询在 SELECT、UPDATE、DELETE 语句内部可以出现 SELECT 语句。内部的 SELECT 语句结果可以作为外部语句中条件子句的一部分，也可以作为外部查询的临时表。子查询的类型有：

1. 单行子查询：不向外部返回结果，或者只返回一行结果。
2. 多行子查询：向外部返回零行、一行或者多行结果。

【例 1】查询出销售部（SALES）下面的员工姓名，工作，工资。

例题分析

该问题可以用联接查询实现，由于所需的结果信息都在 Emp 表中，可以先从 Dept 表中查询出销售部对应的部门号，然后根据当前部门号再到 Emp 表中查询出符合该部门的员工记录即可。从销售表中查询出的结果可以作为 Emp 表中查询的条件，SQL 语句实现如下：

```
SELECT E.ENAME, E.JOB, E.SAL
  FROM EMP E
 WHERE E.DEPTNO = (SELECT D.DEPTNO FROM DEPT D WHERE DNAME = 'SALES')
```

代码解析：

① 内部查询的结果作为外部查询的条件。

需要**注意**：

- 如果内部查询不返回任何记录，则外部条件中字段 DEPTNO 与 NULL 比较永远为假，也就是说外部查询不返还任何结果。
- 在单行子查询中外部查询可以使用=、>、<、>=、<=、<>等比较运算符。
- 内部查询返回的结果必须与外部查询条件中的字段（DEPTNO）匹配。
- 如果内部查询返回多行结果则出现错误。

【例 2】查询 EMP 表中每个部门的最低工资的员工信息

```
SELECT E.*
  FROM EMP E
 WHERE E.SAL IN (SELECT MIN(A.SAL) FROM EMP A WHERE E.DEPTNO = A.DEPTNO)
```

2. Oracle 中的伪列

在 Oracle 的表的使用过程中，实际表中还有一些附加的列，称为伪列。伪列就像表中的列一样，但是在表中并不存储。伪列只能查询，不能进行增删改操作。接下来学习两个伪列：ROWID 和 ROWNUM。

2.1 ROWID

表中的每一行在数据文件中都有一个物理地址，ROWID 伪列返回的就是该行的物理地址。使用 ROWID 可以快速的定位表中的某一行。ROWID 值可以唯一的标识表中的一行。由于 ROWID 返回的是该行的物理地址，因此使用 ROWID 可以显示行是如何存储的。

语法结构：删除重复数据，相同数据只保留一条

```
DELETE FROM 表名 别名
WHERE ROWID NOT IN
(SELECT MIN(ROWID) FROM 表名 别名 GROUP BY 列名)
```

【例】删除 EMP 表重复数据

```
DELETE FROM EMP E
WHERE ROWID NOT IN
    (SELECT MIN(ROWID) FROM EMP E GROUP BY EMPNO)
```

2.2 ROWNUM

在查询的结果集中，ROWNUM 为结果集中每一行标识一个行号，第一行返回 1，第二行返回 2，以此类推。通过 ROWNUM 伪列可以限制查询结果集中返回的行数。

【例 1】查询出员工表中前 5 名员工的姓名，工作，工资。

```
SELECT ROWNUM, E.ENAME, E.JOB, E.SAL FROM EMP E WHERE ROWNUM <= 5;
```

在查询条件中，如果查询条件中 ROWNUM 大于某一正整数，则不返还任何结果。

ROWNUM 与 ROWID 不同，ROWID 是插入记录时生成，ROWNUM 是查询数据时生成。ROWID 标识的是行的物理地址。ROWNUM 标识的是查询结果中的行的次序。

3. Oracle 单行函数

单行函数：对每一个函数应用在表的记录中时，只能输入一行结果，返回一个结果，比如：MOD(x,y)返回 x 除以 y 的余数（x 和 y 可以是两个整数，也可以是表中的整数列）。常用的单行函数有：

- 字符函数：对字符串操作。
- 数字函数：对数字进行计算，返回一个数字。
- 转换函数：可以将一种数据类型转换为另外一种数据类型。

- 日期函数：对日期和时间进行处理。

3.1 字符函数

字符函数接受字符参数，这些参数可以是表中的列，也可以是一个字符串表达式。下表列出了常用的字符函数。

函数	说明
ASCII(x)	返回字符 x 的 ASCII 码。
CONCAT(x,y)	连接字符串 x 和 y。
INSTR(x, str [,start] [,n])	在 x 中查找 str，可以指定从 start 开始，第 n 次出现。
LENGTH(x)	返回 x 的长度。
LOWER(x)	x 转换为小写。
UPPER(x)	x 转换为大写。
LTRIM(x[,trim_str])	把 x 的左边截去 trim_str 字符串，缺省截去空格。
RTRIM(x[,trim_str])	把 x 的右边截去 trim_str 字符串，缺省截去空格。
TRIM([trim_str FROM] x)	把 x 的两边截去 trim_str 字符串，缺省截去空格。
REPLACE(x,old,new)	在 x 中查找 old，并替换为 new。
SUBSTR(x,start[,length])	返回 x 的字串，从 staart 处开始，截取 length 个字符，缺省 length，默认到结尾。

表 1　字符函数

示例	示例结果
SELECT ASCII('a') FROM DUAL	97
SELECT CONCAT('Hello', ' world') FROM DUAL	Hello world
SELECT INSTR('Hello world', 'or') FROM DUAL-1	8
SELECT LENGTH('Hello') FROM DUAL	5
SELECT LOWER('hElLO') FROM DUAL;	hello
SELECT UPPER('hello') FROM DUAL	HELLO
SELECT LTRIM('===HELLO===', '=') FROM DUAL	HELLO===
SELECT '=='||LTRIM(' HELLO===') FROM DUAL	==HELLO===
SELECT RTRIM('===HELLO===', '=') FROM DUAL	===HELLO
SELECT '='||TRIM(' HELLO ')||'=' FROM DUAL	=HELLO=
SELECT TRIM('=' FROM '===HELLO===') FROM DUAL	HELLO
SELECT REPLACE('ABCDE','CD','AAA') FROM DUAL	ABAAAE
SELECT SUBSTR('ABCDE',-2) FROM DUAL	DE
SELECT SUBSTR('ABCDE',2,3) FROM DUAL	BCD

表 2　字符函数示例

DUAL 介绍：

dual 是一张虚拟表，只有一行一列，用来构成 select 的语法规则。Oracle 的查询中，必须使用“select 列… from 表”的完整语法，当查询单行函数的时候，from 后面使用 DUAL 表，dual 表在系统中只有一行一列，该表在输出单行函数时为了 select…from 的语法完整性而使用。

【例】查询员工姓名以’S’结尾的员工信息

```
--方法 1
SELECT * FROM EMP E WHERE INSTR(E.ENAME,'S',-1,1)=LENGTH(E.ENAME);
--方法 2
SELECT * FROM EMP E WHERE RTRIM(E.ENAME,'S')<>E.ENAME;
--方法 3
SELECT * FROM EMP E WHERE SUBSTR(E.ENAME,-1,1)='S;
```

3.2 数字函数

数字函数接受数字参数，参数可以来自表中的一列，也可以是一个数字表达式。

函数	说明	示例
ABS(x)	x 绝对值	ABS(-3)=3
MOD(x,y)	x 除以 y 的余数	MOD(8,3)=2
POWER(x,y)	x 的 y 次幂	POWER(2,3)=8
ROUND(x[,y])	x 在第 y 位四舍五入	ROUND(3.456,2)=3.46
TRUNC(x[,y])	x 在第 y 位截断	TRUNC(3.456,2)=3.45

表 3　数字函数

说明：

1. ROUND(X[,Y])，四舍五入。

   在缺省 y 时，默认 y=0；比如：ROUND(3.56)=4。

   y 是正整数，就是四舍五入到小数点后 y 位。ROUND(5.654,2)=5.65。

   y 是负整数，四舍五入到小数点左边|y|位。ROUND(351.654,-2)=400。

2. TRUNC(x[,y])，直接截取，不四舍五入。

   在缺省 y 时，默认 y=0；比如：TRUNC (3.56)=3。

   y 是正整数，就是四舍五入到小数点后 y 位。TRUNC (5.654,2)=5.65。

   y 是负整数，四舍五入到小数点左边|y|位。TRUNC (351.654,-2)=300。

3.3 日期函数

日期函数对日期进行运算。常用的日期函数有：

1. ADD_MONTHS(d,n)，在某一个日期 d 上，加上指定的月数 n，返回计算后的新日期。

   d 表示日期，n 表示要加的月数。

```
SELECT SYSDATE,ADD_MONTHS(SYSDATE,5) FROM DUAL
```

	SYSDATE	ADD_MONTHS(SYSDATE,5)
1	2017/7/3 星期一 14:28:48	2017/12/3 星期日 14:28:48

图 1　ADD_MONTHS 函数示例

2. LAST_DAY(d)，返回指定日期当月的最后一天。

```
SELECT SYSDATE,LAST_DAY(SYSDATE) FROM DUAL
```

	SYSDATE	LAST_DAY(SYSDATE)
1	2017/7/3 星期一 14:33:53	2017/7/31 星期一 14:33:53

图 2　LAST_DAY 函数示例

3. ROUND(d[,fmt])，返回一个以 fmt 为格式的四舍五入日期值，d 是日期，fmt 是格式模型。默认 fmt 为 DDD，即月中的某一天。
   - 如果 fmt 为“YEAR”则舍入到某年的 1 月 1 日，即前半年舍去，后半年作为下一年。
   - 如果 fmt 为“MONTH”则舍入到某月的 1 日，即前月舍去，后半月作为下一月。
   - 默认为“DDD”，即月中的某一天，最靠近的天，前半天舍去，后半天作为第二天。
   - 如果 fmt 为“DAY”则舍入到最近的周的周日，即上半周舍去，下半周作为下一周周日。

图 3　ROUND 函数示例

```
SELECT SYSDATE,
       ROUND(SYSDATE),
       ROUND(SYSDATE, 'DAY'),
       ROUND(SYSDATE, 'MONTH'),
       ROUND(SYSDATE, 'YEAR')
  FROM DUAL
```

	SYSDATE	ROUND(SYSDATE)	ROUND(SYSDATE,'DAY')	ROUND(SYSDATE,'MONTH')	ROUND(SYSDATE,'YEAR')
1	2017/7/3 星期一 14:40:03	2017/7/4 星期二	2017/7/2 星期日	2017/7/1 星期六	2018/1/1 星期一

与 ROUND 对应的函数时 TRUNC(d[,fmt])对日期的操作，TRUNC 与 ROUND 非常相似，只是不对日期进行舍入，直接截取到对应格式的第一天。

TRUNC(date[,fmt])：将 date 截取到 fmt 指定的形式，如果 fmt 省略，则截取到最近的

```
SELECT SYSDATE,
       TRUNC(SYSDATE),
       TRUNC(SYSDATE, 'DAY'),
       TRUNC(SYSDATE, 'MONTH'),
       TRUNC(SYSDATE, 'YEAR')
  FROM DUAL
```

	SYSDATE	TRUNC(SYSDATE)	TRUNC(SYSDATE,'DAY')	TRUNC(SYSDATE,'MONTH')	TRUNC(SYSDATE,'YEAR')
1	2017/7/3 星期一 14:45:05	2017/7/3 星期一	2017/7/2 星期日	2017/7/1 星期六	2017/1/1 星期日

日期。

图 4　TRUNC 函数示例

3.4 转换函数

转换函数将值从一种数据类型转换为另外一种数据类型。常用的转换函数有：

1. TO_CHAR(d|n[,fmt])

把日期和数字转换为制定格式的字符串。fmt 是格式化字符串，日期的格式化字符串前面已经学习过。

代码演示：TO_CHAR 对日期的处理

```
SELECT TO_CHAR(SYSDATE, 'YYYYMMDD') FROM DUAL
```

代码演示：TO_CHAR 对数字的处理

```
SELECT TO_CHAR(123456) FROM DUAL
```

2. TO_DATE(x [,fmt])

把一个字符串以 fmt 格式转换为一个日期类型。

代码演示：TO_DATE 函数

```
SELECT TO_DATE('20170703145533','YYYYMMDD HH24:MI:SS') FROM DUAL
```

3. TO_NUMBER(x[,fmt])

把一个字符串以 fmt 格式转换为一个数字。

代码演示：TO_NUMBER 函数

```
SELECT TO_NUMBER('123456') FROM DUAL
```

3.5 其他常用函数

NVL（列，默认值）	如果列值为 null，则使用默认值表示
NVL2（列，返回值 1，返回值 2）	如果列值不为 null，返回结果 1； 如果列值为 null，返回结果 2
DECODE（列|值，判断值 1，返回值 1，判断值 2，返回值 2，...，默认值） --不是所有数据库都可用	多值判断，如果列值与判断值相同，则显示对应返回值输出，如果没有满足条件，则显示默认值
CASE WHEN 条件 1 THEN 返回值 1 [WHEN 条	用于实现多条件判断，如果都不满足条件，

件 2 THEN 返回值 2 ...] ELSE 默认值 END --所有数据库都支持	则返回默认值
EXISTS（子查询）	用于判断子查询是否有数据返回，如果有则成立，否则不成立。

1. NVL(x,value)

如果 x 为空，返回 value，否则返回 x。

【例 1】对工资是 2000 元以下的员工，如果没有发奖金，每人奖金 100 元。

代码演示：NVL 函数

```
SELECT E.ENAME, E.JOB, E.SAL, NVL(E.COMM, 100)
  FROM EMP E
 WHERE E.SAL < 2000;
```

2. NVL2(x,value1,value2)

如果 x 非空，返回 value1，否则返回 value2。

【例 2】对 EMP 表中工资为 2000 元以下的员工，如果没有奖金，则奖金为 200 元，如果有奖金，则在原来的奖金基础上加 100 元。

代码演示：NVL2 函数

```
SELECT E.ENAME, E.JOB, E.SAL, NVL2(E.COMM, E.COMM + 100, 200)
  FROM EMP E
 WHERE E.SAL < 2000;
```

3. DECODE(列|表达式，值 1，value1，值 2，value2,...,默认值）

当参数的值为判断值 1，则返回 value1……当参数的值匹配不到时，则返回默认值

【例 3】列出 EMP 员工的姓名，以及工作（中文）

代码演示：DECODE 函数

```
    SELECT E.ENAME,
       DECODE(E.JOB,
              'CLERK',
              '业务员',
              'SALESMAN',
              '销售员',
              'MANAGER',
```

```
                '经理',
                'ANALYST',
                '分析员',
                'PRESIDENT',
                '总裁')
  FROM EMP E
```

4. CASE WHEN

功能与 DECODE 相似，DECODE 只用于多值判断，CASE WHEN 适用于多条件判断。

语法格式：CASE WHEN 语法 1

```
    CASE 参数
  WHEN 判断值 1 THEN 返回值 1
  WHEN 判断值 2 THEN 返回值 2
  ……
  ELSE 默认值 END
```

当参数的值为判断值 1，则返回返回值 1……当参数的值匹配不到时，则返回默认值

语法格式：CASE WHEN 语法 2

```
    CASE
  WHEN 条件 1 THEN 返回值 1
  WHEN 条件 2 THEN 返回值 2
  ……
  ELSE 默认值 END
```

当条件成立，则返回对应的返回值，没有条件成立则返回默认值

【例 1】列出 EMP 员工的姓名，以及工作（中文）

代码演示：CASE WHEN 函数

```
  SELECT E.ENAME,
       CASE E.JOB
         WHEN 'CLERK' THEN
          '业务员'
         WHEN 'SALESMAN' THEN
          '销售员'
         WHEN 'MANAGER' THEN
          '经理'
         WHEN 'ANALYST' THEN
          '分析员'
         WHEN 'PRESIDENT' THEN
          '总裁'
       END
  FROM EMP E
```

【例 2】列出 EMP 员工的姓名，工资，以及工资评价

代码演示：CASE WHEN 函数

```
SELECT E.ENAME,
       E.SAL,
       CASE
         WHEN E.SAL > 3000 THEN
          '工资很高'
         WHEN E.SAL > 1000 THEN
          '工资一般'
         ELSE
          '工资很低'
       END
  FROM EMP E
```

注意：这里的 E.SAL>1000 指的是 1000<E.SAL<=3000，ELSE 指的是 E.SAL<=1000

5. EXISTS

EXISTS(查询结果集)：查询结果集有记录则成立，否则不成立
NOT EXISTS(查询结果集)：与 EXISTS 相反

【例 2】列出有员工的部门信息

代码演示：EXISTS 函数

```
SELECT *
  FROM DEPT D
 WHERE EXISTS (SELECT 1 FROM EMP E WHERE D.DEPTNO = E.DEPTNO)
```

由于部门表中部门编号为 40 的记录，在 EMP 表中不能找到与之对应的记录，因此 EXISTS 不成立，部门编号为 40 的部门信息就不能展示出来。

注意：

EXISTS、IN、关联必然可以相互转换。

同理 NOT EXISTS、NOT IN、外关联+从表 IS NULL 也能相互转换

EXISTS、IN 方法不会发散，但关联性能最好

4. Oracle 分析函数

4.1 分析函数介绍

分析函数是 Oracle 专门用于解决复杂报表统计需求的功能强大的函数，它可以在数据中进行分组然后计算基于组的某种统计值，并且每一组的每一行都可以返回一个统计值。

分析函数和聚合函数的不同之处是什么？
◎普通的聚合函数用 group by 分组，
◎每个分组返回一个统计值，
◎而分析函数采用 partition by 分组，
◎并且每组每行都可以返回一个统计值。

语法格式：分析函数语法

```
FUNCTION_NAME(<参数>,…) OVER (<PARTITION BY 表达式,…> <ORDER BY 表达式
  <ASC DESC> )
```

注意：
分析函数是一个整体，不可分割
例如：对求平均值的分析函数做空值转换：
NVL(AVG(SAL)OVER(PARTITION BY DEPTNO ORDER BY 1),0)

4.2 分析函数种类和用法

1.MAX(),MIN(),SUM(),AVG(),COUNT() --加了 ORDER BY 是累计求值

【例】求每个部门工资高于部门平均工资的员工数量占整个部门人数的百分比

```
SELECT A.DEPTNO,
       SUM(CASE
             WHEN A.SAL > A.AVG THEN
              1
             ELSE
              0
           END) / COUNT(1)
  FROM (SELECT E.*, AVG(E.SAL) OVER(PARTITION BY E.DEPTNO) AVG FROM EMP E) A
 GROUP BY A.DEPTNO
```

2.RANK(),DENSE_RANK()与 ROW_NUMBER()：

RANK，DENSE_RANK，ROW_NUMBER 函数为每条记录产生一个从 1 开始至 N 的自然数，
N 的值可能小于等于记录的总数。这 3 个函数的唯一区别在于当碰到相同数据时的排名策略。

①ROW_NUMBER：
ROW_NUMBER 函数返回一个唯一的值，当碰到相同数据时，排名按照记录集中记录的顺序依次递增。
②DENSE_RANK：
DENSE_RANK 函数返回一个唯一的值，当碰到相同数据时，此时所有相同数据的排名都是一样的。
③RANK：
RANK 函数返回一个唯一的值，当碰到相同的数据时，此时所有相同数据的排名是一样的，同时会在最后一条相同记录和下一条不同记录的排名之间空出排名。

【例】查询 EMP 表所有员工姓名,工资以部门分组降序排序

```
SELECT E.ENAME,
       E.SAL,
       ROW_NUMBER() OVER(PARTITION BY E.DEPTNO ORDER BY E.SAL DESC)
  FROM EMP E
```

3.LAG()与 LEAD()：求之前或之后的第 N 行

LAG 和 LEAD 函数可以在一次查询中取出同一字段的前 N 行的数据和后 N 行的值。这种操作可以使用对相同表的表连接来实现，不过使用 LAG 和 LEAD 有更高的效率。

LAG(ARG1,ARG2,ARG3)第一个参数是列名，
第二个参数是偏移的 offset，
第三个参数是超出记录窗口时的默认值。

【例】比较 EMP 表中员工与上一个入职员工晚入职多久

```
SELECT E.ENAME,
       E.HIREDATE - LAG(E.HIREDATE, 1) OVER(ORDER BY E.HIREDATE ASC)
  FROM EMP E
```

4.3 行列转换

在用户制作数据报表时，经常会使用到表数据的行列转换操作。

1.列转行：

【例】有一张表 S，记录了某公司每个月的销售额，如下

Y	Q	AMT
2015	1	100
2015	2	110
2015	3	130
2015	4	100

2016 1 200
2016 2 150
2016 3 100
2016 4 300

（1）用分析函数 lead/lag

```
SELECT S.Y, S.AMT Q1, S.LD1 Q2, S.LD2 Q3, S.LD3 Q4
  FROM (SELECT S.*,
               LEAD(S.AMT, 1) OVER(PARTITION BY S.Y ORDER BY S.Q) LD1,
               LEAD(S.AMT, 2) OVER(PARTITION BY S.Y ORDER BY S.Q) LD2,
               LEAD(S.AMT, 3) OVER(PARTITION BY S.Y ORDER BY S.Q) LD3
          FROM S) S
 WHERE S.Q = 1
```

（2）用 DECODE

```
SELECT S.Y,
       SUM(DECODE(S.Q, 1, AMT, NULL)) Q1,
       SUM(DECODE(S.Q, 2, AMT, NULL)) Q2,
       SUM(DECODE(S.Q, 3, AMT, NULL)) Q3,
       SUM(DECODE(S.Q, 4, AMT, NULL)) Q4
  FROM S
GROUP BY S.Y
```

注意：NULL 值不参于任何计算

（3）部分关联

```
SELECT S.Y, S.AMT Q1, A.AMT Q2, B.AMT Q3, C.AMT Q4
  FROM S,
       (SELECT * FROM S WHERE S.Q = 2) A,
       (SELECT * FROM S WHERE S.Q = 3) B,
       (SELECT * FROM S WHERE S.Q = 4) C
 WHERE S.Y = A.Y
   AND S.Y = B.Y
   AND S.Y = C.Y
   AND S.Q=1
```

2. 行转列：

```
WITH A AS
 (SELECT S.Y,
         SUM(DECODE(S.Q, 1, AMT, NULL)) Q1,
         SUM(DECODE(S.Q, 2, AMT, NULL)) Q2,
         SUM(DECODE(S.Q, 3, AMT, NULL)) Q3,
         SUM(DECODE(S.Q, 4, AMT, NULL)) Q4
    FROM S
  GROUP BY S.Y)      --将结果集命名为 A，把 A 行转列
```

```
SELECT A.Y, 1, A.Q1
  FROM A
UNION ALL
SELECT A.Y, 2, A.Q2
  FROM A
UNION ALL
SELECT A.Y, 3, A.Q3
  FROM A
UNION ALL
SELECT A.Y, 4, A.Q4 FROM A
```

3. 计算同环比

环比=（现阶段-同一周期上一阶段）/同一周期上一阶段
同比=（现阶段-上一周期相同阶段）/上一周期相同阶段

第 4 章

表空间、数据库对象

Oracle 数据库对象

数据库对象是数据库的组成部分，常常用 CREATE 命令进行创建，可以使用 ALTER 命令修改，用 DROP 执行删除操作。前面已经接触过的数据库对象有表、用户等。

今天将学习更多的 Oracle 数据库对象：

- 表空间：所有的数据对象都存在指定的表空间中。
- 同义词：就是给数据库对象一个别名。
- 序列：Oracle 中实现增长的对象。
- 视图：预定义的查询，作为表一样的查询使用，是一张虚拟表。
- 索引：对数据库表中的某些列进行排序，便于提高查询效率。
- 表空间：

1.表空间

在数据库系统中，存储空间是较为重要的资源，合理利用空间，不但能节省空间，还可以提高系统的效率和工作性能。Oracle 可以存放海量数据，所有数据都在数据文件中存储。而数据文件大小受操作系统限制，并且过大的数据文件对数据的存取性能影响非常大。同时 Oracle 是跨平台的数据库，Oracle 数据可以轻松的在不同平台上移植，那么如何才能提供统一存取格式的大容量呢？Oracle 采用表空间来解决。

表空间只是一个逻辑概念，若干操作系统文件（文件可以不是很大）可以组成一个表空间。表空间统一管理空间中的数据文件，一个数据文件只能属于一个表空间。一个数据库空间由若干个表空间组成。如图所示：

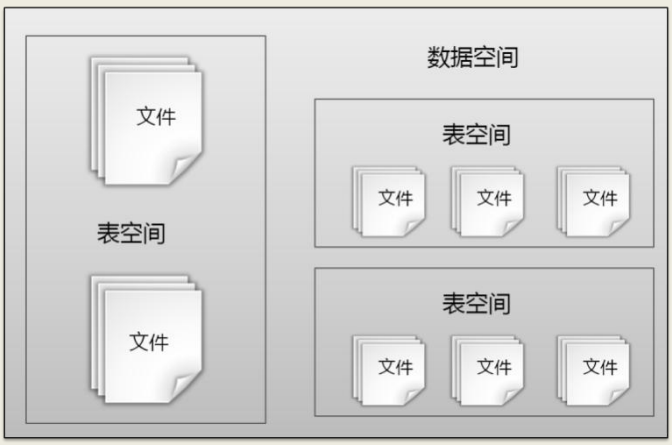


图 1　数据空间、表空间和数据文件

Oracle 中所有的数据（包括系统数据），全部保存在表空间中，常见的表空间有：

- 系统表空间：存放系统数据，系统表空间在数据库创建时创建。表空间名称为 SYSTEM。存放数据字典和视图以及数据库结构等重要系统数据信息，在运行时如果 SYSTEM 空间不足，对数据库影响会比较大，虽然在系统运行过程中可以通过命令扩充空间，但还是会影响数据库的性能，因此有必要在创建数据库时适当的把数据文件设置大一些。
- TMEP 表空间：临时表空间，安装数据库时创建，可以在运行时通过命令增大临时表空间。临时表空间的重要作用是数据排序。比如当用户执行了诸如 Order by 等命令后，服务器需要对所选取数据进行排序，如果数据很大，内存的排序区可能装不下太大数据，就需要把一些中间的排序结果写在硬盘的临时表空间中。
- 用户表自定义空间：用户可以通过 CREATE TABLESPACE 命令创建表空间。

2.同义词

同义词（Synonym）是数据库对象的一个别名，Oracle 可以为表、视图、序列、过程、函数、程序包等指定一个别名。同义词有两种类型：

- 私有同义词：拥有 CREATE SYNONYM 权限的用户（包括非管理员用户）即可创建私有同义词，创建的私有同义词只能由当前用户使用。
- 公有同义词：系统管理员可以创建公有同义词，公有同义词可以被所有用户访问。

创建同义词的语法是：

语法结构：创建同义词

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema.]synonym_name
FOR [schema.]object_name
```

语法解析：

① CREATE [OR REPLACE：]表示在创建同义词时，如果该同义词已经存在，那么就用新创建的同义词代替旧同义词。

② PULBIC：创建公有同义词时使用的关键字，一般情况下不需要创建公有同义词。

③ Oracle 中一个用户可以创建表、视图等多种数据库对象，一个用户和该用户下的所有数据库对象的集合称为 Schema（中文称为模式或者方案），用户名就是 Schema 名。一个数据库对象的全称是：用户名.对象名，即 schema.object_name。

如果一个用户有权限访问其他用户对象时，就可以使用全称来访问。比如：

代码演示：System 用户访问 Scott 用户的 Emp 表

```
SELECT ENAME,JOB,SAL FROM SCOTT.EMP WHERE SAL>2000;
```

代码解析：

① 管理员用户可以访问任何用户的数据库对象，SYSTEM 用户访问 SCOTT 用户的 EMP 表时，必须使用 SCOTT.EMP。

删除同义词使用的语法是：

语法结构：删除同义词

```
DROP [PUBLIC] SYNONYM [schema.]sysnonym_name
```

语法解析：

① PUBLIC：删除公共同义词。

② 同义词的删除只能被拥有同义词对象的用户或者管理员删除。

③ 此命令只能删除同义词，不能删除同义词下的源对象。

3.序列

序列(Sequence)是用来生成连续的整数数据的对象。序列常常用来作为主键中增长列，序列中的可以升序生成，也可以降序生成。创建序列的语法是：

语法结构：创建序列

```
CREATE SEQUENCE sequence_name
[START WITH num]
[INCREMENT BY increment]
[MAXVALUE num|NOMAXVALUE]
[MINVALUE num|NOMINVALUE]
[CYCLE|NOCYCLE]
[CACHE num|NOCACHE]
```

语法解析：

① START WITH：从某一个整数开始，升序默认值是 1，降序默认值是-1。

② INCREMENT BY：增长数。如果是正数则升序生成，如果是负数则降序生成。升序默认值是 1，降序默认值是-1。

③ MAXVALUE：指最大值。

④ NOMAXVALUE：这是最大值的默认选项，升序的最大值是：10^{27}，降序默认值是-1。

⑤ MINVALUE：指最小值。

⑥ NOMINVALUE：这是默认值选项，升序默认值是 1，降序默认值是-10^{26}。

⑦ CYCLE：表示如果升序达到最大值后，从最小值重新开始；如果是降序序列，达到最小值后，从最大值重新开始。

⑧ NOCYCLE：表示不重新开始，序列升序达到最大值、降序达到最小值后就报错。默认 NOCYCLE。

⑨ CACHE：使用 CACHE 选项时，该序列会根据序列规则预生成一组序列号。保留在内存中，当使用下一个序列号时，可以更快的响应。当内存中的序列号用完时，系统再生成一组新的序列号，并保存在缓存中，这样可以提高生成序列号的效率。Oracle 默认会生产 20 个序列号。

⑩ NOCACHE：不预先在内存中生成序列号。

【例】创建一个从 1 开始，默认最大值，每次增长 1 的序列，要求 NOCYCLE，缓存中有 30 个预先分配好的序列号。

代码演示：生成序列号

```
CREATE SEQUENCE MYSEQ
MINVALUE 1
START WITH 1
NOMAXVALUE
INCREMENT BY 1
NOCYCLE
CACHE 30
```

序列创建之后，可以通过序列对象的 CURRVAL 和 NEXTVAL 两个“伪列”分别访问该序列的当前值和下一个值。

代码演示：序列使用

```
--访问下一个值
SELECT MYSEQ.NEXTVAL FROM DUAL;
--访问当前值
SELECT MYSEQ.CURRVAL FROM DUAL;
```

使用 ALTER SEQUENCE 可以修改序列，在修改序列时有如下限制：

1. 不能修改序列的初始值。
2. 最小值不能大于当前值。
3. 最大值不能小于当前值。

使用 DROP SEQUENCE 命令可以删除一个序列对象。

代码演示：序列修改和删除

```
--序列修改
ALTER SEQUENCE MYSEQ
MAXVALUE 10000
MINVALUE -300
--删除序列
```

```
DROP SEQUENCE MYSEQ;
```

4. 视图

视图（View）实际上是一张或者多张表上的预定义查询，这些表称为基表。从视图中查询信息与从表中查询信息的方法完全相同。只需要简单的 SELECT…FROM 即可。

视图具有以下优点：

1. 可以限制用户只能通过视图检索数据。这样就可以对最终用户屏蔽建表时底层的基表，具有安全性。
2. 可以将复杂的查询保存为视图，屏蔽复杂性。

语法结构：创建视图

```
CREATE [OR REPLACE] [{FORCE|NOFORCE}] VIEW view_name
AS
SELECT查询
[WITH READ ONLY CONSTRAINT]
```

语法解析：

1. OR REPLACE：如果视图已经存在，则替换旧视图。
2. FORCE：即使基表不存在，也可以创建该视图，但是该视图不能正常使用，当基表创建成功后，视图才能正常使用。
3. NOFORCE：如果基表不存在，无法创建视图，该项是默认选项。
4. WITH READ ONLY：默认可以通过视图对基表执行增删改操作，但是有很多在基表上的限制（比如：基表中某列不能为空，但是该列没有出现在视图中，则不能通过视图执行 insert 操作），WITH READ ONLY 说明视图是只读视图，不能通过该视图进行增删改操作。现实开发中，基本上不通过视图对表中的数据进行增删改操作。

【例】基于 EMP 表和 DEPT 表创建视图

代码演示：视图

```
--创建视图
CREATE OR REPLACE VIEW EMPDETAIL
AS
SELECT EMPNO,ENAME,JOB,HIREDATE,EMP.DEPTNO,DNAME
FROM EMP JOIN DEPT ON EMP.DEPTNO=DEPT.DEPTNO
WITH READ ONLY
--通过视图查询
SELECT * FROM EMPDETAIL;
```

代码解析：

① 对视图可以像表一样进行查询。该视图中隐藏了员工的工资。

语法结构：删除视图

```
DROP VIEW 视图名
```

5. 索引

5.1 索引介绍

当我们在某本书中查找特定的章节内容时，可以先从书的目录着手，找到该章节所在的页码，然后快速的定位到该页。这种做法的前提是页面编号是有序的。如果页码无序，就只能从第一页开始，一页页的查找了。

数据库中索引（Index）的概念与目录的概念非常类似。如果某列出现在查询的条件中，而该列的数据是无序的，查询时只能从第一行开始一行一行的匹配。创建索引就是对某些特定列中的数据排序，生成独立的索引表。在某列上创建索引后，如果该列出现在查询条件中，Oracle 会比较全表扫描与索引扫描的代价，如果索引扫描代价小，那 Oracle 会自动引用该索引，先从索引表中查询出符合条件记录的 ROWID，由于 ROWID 是记录的物理地址，因此可以根据 ROWID 快速的定位到具体的记录，表中的数据非常多时，引用索引带来的查询效率非常可观。

- ✧ 如果表中的某些字段经常被查询并作为查询的条件出现时，就应该考虑为该列创建索引。
- ✧ 当从很多行的表中查询少数行时，也要考虑创建索引。有一条基本的准则是：当任何单个查询要检索的行少于或者等于整个表行数的 5-10%时，索引就非常有用。

Oracle 数据库会为表的主键和包含唯一约束的列自动创建索引。索引可以提高查询的效率，但是在数据增删改时需要更新索引，因此索引对增删改时会有负面影响。

语法结构：创建索引

```
CREATE [UNIQUE] INDEX index_name ON table_name(column_name[,column_name…])
```

语法解析：

1. UNIQUE:指定索引列上的值必须是唯一的。称为唯一索引。
2. index_name：指定索引名。
3. table_name：指定要为哪个表创建索引。

4. column_name(列名)：指定要对哪个列创建索引。我们也可以对多列创建索引；这种索引称为组合索引。

【例】为 EMP 表的 ENAME 列创建创建唯一索引，为 EMP 表的工资列创建普通索引，把 JOB 列先变为小写再创建索引。

代码演示：创建索引

```
--为 ENAME 创建唯一索引
CREATE UNIQUE INDEX UQ_ENAME_IDX ON EMP(ENAME);
--将 JOB 转成小写再为函数创建索引
CREATE INDEX IDX_JOB_LOWER ON EMP(LOWER(JOB));
```

代码解析：

① 为 SCOTT.EMP 表的 ENAME 列创建唯一索引。

② 在查询中可能经常使用 job 的小写作为条件的表达式，因此创建索引时，可以先对 JOB 列中的所有值转换为小写后创建索引，而这时需要使用 lower 函数，这种索引称为基于函数的索引。

在 select 语句查询时，Oracle 系统会自动为查询条件上的列应用索引。索引就是对某一列进行排序，因此在索引列上，重复值越少，索引的效果越明显。

Oracle 可以为一些列值重复非常多且值有限的列（比如性别列）上创建位图索引。

语法结构：删除索引

```
DROP INDEX 索引名
```

5.2 索引种类

唯一索引（用的最多）

1、何时创建：当某列任意两行的值都不相同

2、当建立 Primary Key(主键)或者 Unique constraint(唯一约束)时，唯一索引将被自动建立

组合索引

1、何时创建：当两个或多个列经常一起出现在 where 条件中时，则在这些列上同时创建

2、组合索引中列的顺序是任意的，也无需相邻。但是建议将最频繁访问的列放在列表的最前面

位图索引

1、何时创建：列中有非常多的重复的值时候。例如某列保存了 “性别”信息。

Where 条件中包含了很多 OR 操作符。较少的 update 操作，因为要相应的跟新所有的 bitmap

基于函数的索引

1、何时创建：在 WHERE 条件语句中包含函数或者表达式时

2、函数包括：算数表达式、PL/SQL 函数、程序包函数、SQL 函数、用户自定义函数。

反向键索引
键压缩索引
索引组织表(IOT)
分区索引

5.3 索引优缺点

创建索引可以大大提高系统的性能。
第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。
第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。
第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。
第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

也许会有人要问：增加索引有如此多的优点，为什么不对表中的每一个列创建一个索引呢？因为，增加索引也有许多不利的方面。
第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。
第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。
第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。
如果一张表 20 个索引，往里面写入 100 万条数据，怎么优化？
先把索引全删了，再把记录写进去，再建立索引。

索引是建立在数据库表中的某些列的上面。
在创建索引的时候，应该考虑在哪些列上可以创建索引，在哪些列上不能创建索引。
一般来说，应该在这些列上创建索引：
在经常需要搜索的列上，可以加快搜索的速度；
在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；
在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；
在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；

同样，对于有些列不应该创建索引。一般来说，不应该创建索引的的这些列具有下列特点：
第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。
第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。
第三，对于那些定义为 text, image 和 bit 数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。
第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能

是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

5.4 索引失效

1. 隐式转换导致索引失效.这一点应当引起重视.也是开发中经常会犯的错误.

由于表的字段 tu_mdn 定义为 varchar2(20),但在查询时把该字段作为 number 类型？以 where 条件传给 Oracle,这样会导致索引失效.

错误的例子：select * from test where tu_mdn=13333333333;

正确的例子：select * from test where tu_mdn='13333333333';

2. 对索引列进行运算导致索引失效,我所指的对索引列进行运算包括(+,-,*,/,! 等)

错误的例子：select * from test where id-1=9;

正确的例子：select * from test where id=10;

3. 使用 Oracle 内部函数导致索引失效.对于这样情况应当创建基于函数的索引.

错误的例子：select * from test where round(id)=10; 说明，此时 id 的索引已经不起作用了

正确的例子:首先建立函数索引,create index test_id_fbi_idx on test(round(id));然后 select * from test where round(id)=10; 这时函数索引起作用了

4. 以下使用会使索引失效，应避免使用;

a. 使用 <> 、not in 、not exist、!=

b. like "%_" 百分号在前（可采用在建立索引时用 reverse(columnName)这种方法处理）

c. 单独引用复合索引里非第一位置的索引列。应总是使用索引的第一个列，如果索引是建立在多个列上，只有在它的第一个列被 where 子句引用时,优化器才会选择使用该索引。

d. 字符型字段为数字时在 where 条件里不添加引号.

e. 当变量采用的是 times 变量，而表的字段采用的是 date 变量时.或相反情况。

6. 表分区

6.1 表空间及分区表的概念

表空间：

是一个或多个数据文件的集合，所有的数据对象都存放在指定的表空间中，但主要存放的是表，所以称作表空间。

分区表：

当表中的数据量不断增大，查询数据的速度就会变慢，应用程序的性能就会下降，这时就应该考虑对表进行分区。表进行分区后，逻辑上表仍然是一张完整的表，只是将表中的数据在物理上存放到一个或多个表空间(物理文件上)，这样查询数据时，不至于每次都扫描整张表。

6.2 表分区的具体作用

Oracle 的表分区功能通过改善可管理性、性能和可用性，从而为各式应用程序带来了极大的好处。通常，分区可以使某些查询以及维护操作的性能大大提高。此外,分区还可以极大简化常见的管理任务，分区是构建千兆字节数据系统或超高可用性系统的关键工具。

分区功能能够将表、索引或索引组织表进一步细分为段,这些数据库对象的段叫做分区。每个分区有自己的名称，还可以选择自己的存储特性。从数据库管理员的角度来看，一个分区后的对象具有多个段，这些段既可进行集体管理，也可单独管理，这就使数据库管理员在管理分区后的对象时有相当大的灵活性。但是，从应用程序的角度来看，分区后的表与非分区表完全相同，使用 DML 命令访问分区后的表时，无需任何修改。

什么时候使用分区表：

1)表的数据量特别大

2)表中包含历史数据，新的数据被增加到新的分区中。

6.3.表分区的优缺点

优点：

1)改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索速度。

2)增强可用性：如果表的某个分区出现故障，表在其他分区的数据仍然可用；

3)维护方便：如果表的某个分区出现故障，需要修复数据，只修复该分区即可；

4)均衡 I/O：可以把不同的分区映射到磁盘以平衡 I/O，改善整个系统性能。

缺点：

分区表相关，已经存在的表没有方法可以直接转化为分区表。

需要维护。

6.4 表分区的几种类型及操作方法

1.范围分区：RANGE
2.列表分区：LIST
3.散列（哈希）分区：HASH
4.组合分区

1.范围分区

范围分区将数据基于范围映射到每一个分区，这个范围是你在创建分区时指定的分区键决定的。这种分区方式是最为常用的，并且分区键经常采用日期。举个例子：你可能会将销售数据按照月份进行分区。

当使用范围分区时，请考虑以下几个规则：

1)每一个分区都必须有一个 VALUES LESS THEN 子句，它指定了一个不包括在该分区中的上限值。分区键的任何值等于或者大于这个上限值的记录都会被加入到下一个高一些的分区中。

2)所有分区，除了第一个，都会有一个隐式的下限值，这个值就是此分区的前一个分区的上限值。

3)在最高的分区中，MAXVALUE 被定义。MAXVALUE 代表了一个不确定的值。这个值高于其它分区中的任何分区键的值，也可以理解为高于任何分区中指定的 VALUE LESS THEN 的值，同时包括空值。

【例】按入职日期进行范围分区

```
CREATE TABLE MYEMP
(
    EMPNO    NUMBER(4)    PRIMARY KEY,
    ENAME VARCHAR2(10),
    JOB VARCHAR2(9),
    MGR NUMBER(4),
    HIREDATE DATE,
    SAL    NUMBER(7,2),
    COMM NUMBER(7,2),
    DEPTNO NUMBER(7,2) ,
     CONSTRAINT EMP2_FK FOREIGN KEY(DEPTNO) REFERENCES DEPT(DEPTNO)
)
PARTITION    BY    RANGE (HIREDATE)
(
      PARTITION    part1 VALUES    LESS    THAN (TO_DATE('1981-1-1','YYYY/MM/DD')), --①
      PARTITION    part2 VALUES    LESS    THAN (TO_DATE('1982-1-1','YYYY/MM/DD')),
      PARTITION    part3 VALUES    LESS    THAN (TO_DATE('1983-1-1','YYYY/MM/DD')),
```

```
    PARTITION    part4 VALUES    LESS    THAN (TO_DATE('1988-1-1','YYYY/MM/DD')),
    PARTITION    part5 VALUES    LESS    THAN (MAXVALUE)
)
```

--①在此逗号前可以此指定此分区的表空间，不写则默认为当前用户下的表空间。

2. 列表分区：

该分区的特点是某列的值只有几个，基于这样的特点我们可以采用列表分区。

【例】按 DEPTNO 进行 LIST 分区

```
CREATE TABLE MYEMP2
(
    EMPNO    NUMBER(4)    PRIMARY KEY,
    ENAME VARCHAR2(10),
    JOB VARCHAR2(9),
    MGR NUMBER(4),
    HIREDATE DATE,
    SAL    NUMBER(7,2),
    COMM NUMBER(7,2),
    DEPTNO NUMBER(7,2),
    CONSTRAINT EMP1_FK FOREIGN KEY(DEPTNO) REFERENCES DEPT(DEPTNO)
)
PARTITION BY LIST (DEPTNO)
(
    PARTITION MYEMP_DEPTNO_10    VALUES (10),
    PARTITION MYEMP_DEPTNO_20    VALUES (20) ,
    PARTITION MYEMP_DEPTNO_30    VALUES (30) ,
    PARTITION MYEMP_DEPTNO_40    VALUES (40)
    )
```

3. 散列分区：

这类分区是在列值上使用散列算法，以确定将行放入哪个分区中。当列的值没有合适的条件时，建议使用散列分区。

散列分区为通过指定分区编号来均匀分布数据的一种分区类型，因为通过在 I/O 设备上进行散列分区，使得这些分区大小一致。

4. 组合分区

这种分区是基于两种分区的组合，分区之中的分区被称为子分区。

【例】按入职日期进行范围分区，再按 DEPTNO 进行 LIST 子分区

```
CREATE TABLE MYEMP3
(
    EMPNO    NUMBER(4)    PRIMARY KEY,
    ENAME VARCHAR2(10),
    JOB VARCHAR2(9),
    MGR NUMBER(4),
    HIREDATE DATE,
    SAL    NUMBER(7,2),
    COMM NUMBER(7,2),
    DEPTNO NUMBER(7,2) ,
      CONSTRAINT EMP3_FK FOREIGN KEY(DEPTNO) REFERENCES DEPT(DEPTNO)
)
PARTITION BY RANGE(HIREDATE) SUBPARTITION BY LIST (DEPTNO)

(

   PARTITION P1 VALUES LESS THAN(TO_DATE('1981-01-01','YYYY-MM-DD'))
            (
                SUBPARTITION P1A VALUES (10) ,
                SUBPARTITION P1B VALUES (20),
                SUBPARTITION P1C VALUES (30),
                SUBPARTITION P1D VALUES (40)
            ),
   PARTITION P2 VALUES LESS THAN (TO_DATE('1982-01-01','YYYY-MM-DD'))
            (
              SUBPARTITION P2A VALUES (10) ,
                SUBPARTITION P2B VALUES (20),
                SUBPARTITION P2C VALUES (30),
                SUBPARTITION P2D VALUES (40)
            ),
   PARTITION P3 VALUES LESS THAN (TO_DATE('1983-01-01','YYYY-MM-DD'))
            (
              SUBPARTITION P3A VALUES (10) ,
                SUBPARTITION P3B VALUES (20),
                SUBPARTITION P3C VALUES (30),
                SUBPARTITION P3D VALUES (40)
            ),
     PARTITION P4 VALUES LESS THAN (TO_DATE('1988-01-01','YYYY-MM-DD'))
            (
              SUBPARTITION P4A VALUES (10) ,
                SUBPARTITION P4B VALUES (20),
                SUBPARTITION P4C VALUES (30),
```

```
                SUBPARTITION P4D VALUES (40)
            )
)
```

6.5 有关表分区的一些维护性操作

1)添加分区

以下代码给 SALES 表添加了一个 P3 分区

```
ALTER TABLE SALES ADD PARTITION P3 VALUES LESS
THAN(TO_DATE('2003-06-01','YYYY-MM-DD'));
```

注意：以上添加的分区界限应该高于最后一个分区界限。

以下代码给 SALES 表的 P3 分区添加了一个 P3SUB1 子分区

```
ALTER TABLE SALES MODIFY PARTITION P3 ADD SUBPARTITION P3SUB1 VALUES('COMPLETE');
```

2)删除分区

以下代码删除了 P3 表分区：

```
ALTER TABLE SALES DROP PARTITION P3;
```

在以下代码删除了 P4SUB1 子分区：

```
ALTER TABLE SALES DROP SUBPARTITION P4SUB1;
```

注意：如果删除的分区是表中唯一的分区，那么此分区将不能被删除，要想删除此分区，必须删除表。

3)截断分区

截断某个分区是指删除某个分区中的数据，并不会删除分区，也不会删除其它分区中的数据。当表中即使只有一个分区时，也可以截断该分区。通过以下代码截断分区：

```
ALTER TABLE SALES TRUNCATE PARTITION P2;
```

通过以下代码截断子分区：

```
ALTER TABLE SALES TRUNCATE SUBPARTITION P2SUB2;
```

4)合并分区

合并分区是将相邻的分区合并成一个分区，结果分区将采用较高分区的界限，值得**注意**的是，不能将分区合并到界限较低的分区。以下代码实现了 P1 P2 分区的合并：

```
ALTER TABLE SALES MERGE PARTITIONS P1,P2 INTO PARTITION P2;
```

5)拆分分区

拆分分区将一个分区拆分两个新分区，拆分后原来分区不再存在。注意不能对 HASH 类型的分区进行拆分。

```
ALTER TABLE SALES SBLIT PARTITION P2 AT(TO_DATE('2003-02-01','YYYY-MM-DD')) INTO
(PARTITION P21,PARTITION P22);
```

6)接合分区(coalesca)

结合分区是将散列分区中的数据接合到其它分区中，当散列分区中的数据比较大时，可以增加散列分区，然后进行接合，值得**注意**的是，接合分区只能用于散列分区中。通过以下代码进行接合分区：

```
ALTER TABLE SALES COALESCA PARTITION;
```

7)重命名表分区

以下代码将 P21 更改为 P2

```
ALTER TABLE SALES RENAME PARTITION P21 TO P2;
```

第 5 章

PL/SQL 程序设计

1. PL/SQL 简介

Oracle PL/SQL 语言（Procedural Language/SQL）是结合了结构化查询与 Oracle 自身过程控制为一体的强大语言，PL/SQL 不但支持更多的数据类型，拥有自身的变量声明、赋值语句，而且还有条件、循环等流程控制语句。过程控制结构与 SQL 数据处理能力无缝的结合形成了强大的编程语言，可以创建过程和函数以及程序包。

PL/SQL 是一种块结构的语言，它将一组语句放在一个块中，一次性发送给服务器，PL/SQL 引擎分析收到 PL/SQL 语句块中的内容，把其中的过程控制语句由 PL/SQL 引擎自身去执行，把 PL/SQL 块中的 SQL 语句交给服务器的 SQL 语句执行器执行。如图所示：

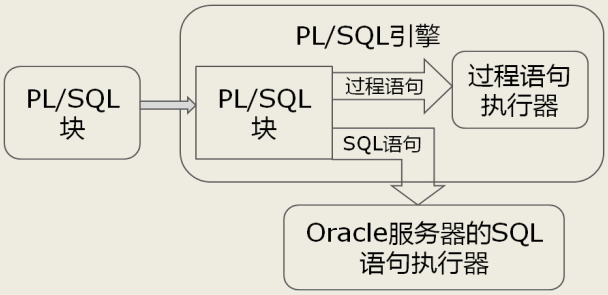


图 1　PL/SQL 体系结构

PL/SQL 的优点：

- **支持 SQL**

SQL 是访问数据库的标准语言，通过 SQL 命令，用户可以操纵数据库中的数据。PL/SQL 支持所有的 SQL 数据操纵命令、游标控制命令、事务控制命令、SQL 函数、运算符和伪列。同时 PL/SQL 和 SQL 语言紧密集成，PL/SQL 支持所有的 SQL 数据类型和 NULL 值。

- **支持面向对象编程**

PL/SQL 支持面向对象的编程，在 PL/SQL 中可以创建类型，可以对类型进行继承，可以在子程序中重载方法等。

- **更好的性能**

SQL 是非过程语言，只能一条一条执行，而 PL/SQL 把一个 PL/SQL 块统一进行编译后执行，同时还可以把编译好的 PL/SQL 块存储起来，以备重用，减少了应用程序和服务器之间的通信时间，PL/SQL 是快速而高效的。

- **可移植性**

使用 PL/SQL 编写的应用程序，可以移植到任何操作系统平台上的 Oracle 服务器，同时还可以编写可移植程序库，在不同环境中重用。

- **安全性**

可以通过存储过程对客户机和服务器之间的应用程序逻辑进行分隔，这样可以限制对 Oracle 数据库的访问，数据库还可以授权和撤销其他用户访问的能力。

2. PL/SQL 基础

PL/SQL 是一种块结构的语言，一个 PL/SQL 程序包含了一个或者多个逻辑块，逻辑块中可以声明变量，变量在使用之前必须先声明。除了正常的执行程序外，PL/SQL 还提供了专门的异常处理部分进行异常处理。每个逻辑块分为三个部分，语法是：

语法结构：PL/SQL 块的语法

```
[DECLARE
    --declaration statements]  ①
BEGIN
    --executable statements  ②
[EXCEPTION
    --exception statements]  ③
END;
```

语法解析：

① 声明部分：声明部分包含了变量和常量的定义。这个部分由关键字 DECLARE 开始，如果不声明变量或者常量，可以省略这部分。

② 执行部分：执行部分是 PL/SQL 块的指令部分，由关键字 BEGIN 开始，关键字 END 结尾。所有的可执行 PL/SQL 语句都放在这一部分，该部分执行命令并操作变量。其他的 PL/SQL 块可以作为子块嵌套在该部分。PL/SQL 块的执行部分是必选的。注意 END 关键字后面用分号结尾。

③ 异常处理部分：该部分是可选的，该部分用 EXCEPTION 关键字把可执行部分分成两个小部分，之前的程序是正常运行的程序，一旦出现异常就跳转到异常部分执行。

PL/SQL 是一种编程语言，与 Java 和 C#一样，除了有自身独有的数据类型、变量声明和赋值以及流程控制语句外，PL/SQL 还有自身的语言特性：

PL/SQL 对大小写不敏感，为了良好的程序风格，开发团队都会选择一个合适的编码标准。比如有的团队规定：关键字全部大些，其余的部分小写。

PL/SQL 块中的每一条语句都必须以分号结束，SQL 语句可以是多行的，但分号表示该语句结束。一行中可以有多条 SQL 语句，他们之间以分号分隔，但是不推荐一行中写多条语句。

PL/SQL 中的特殊符号说明：

类型	符号	说明
赋值运算符	:=	Java 和 C#中都是等号，PL/SQL 的赋值是：=

特殊字符	||	字符串连接操作符。
	--	PL/SQL 中的单行注释。
	/*,*/	PL/SQL 中的多行注释，多行注释不能嵌套。
	..	范围操作符，比如：1..5 标识从 1 到 5
算术运算符	+，-，*，/	基本算术运算符。
	**	求幂操作，比如：3**2=9
关系运算符	>，<,>=,<=,=	基本关系运算符，=表示相等关系，不是赋值。
	<>,!=	不等关系。
逻辑运算符	AND,OR,NOT	逻辑运算符。

表 1　PL/SQL 中的特殊符号和运算符

2.1 声明与赋值

2.1.1 变量声明

PL/SQL 支持 SQL 中的数据类型，PL/SQL 中正常支持 NUMBER，VARCHAR2，DATE 等 Oracle SQL 数据类型。声明变量必须指明变量的数据类型，也可以声明变量时对变量初始化，变量声明必须在声明部分。

声明变量的语法是：

语法格式：声明变量

```
变量名 数据类型[ :=初始值]
```

语法解析：

数据类型如果需要长度，可以用括号指明长度，比如：varchar2(20)。

注意：字符型一定要定义长度，数字型可以不用定义长度。

代码演示：声明变量

```
DECLARE
SNAME VARCHAR2(20) :='JERRY';   ①
BEGIN
SNAME:=SNAME||'AND TOM';   ②                --直接赋值
DBMS_OUTPUT.PUT_LINE(SNAME);   ③
END;
```

代码解析：

① 声明一个变量 sname，初始化值是“jerry”。字符串用单引号，如果字符串中出现单引号可以使用两个单引号（’’）来表示，即单引号同时也具有转义的作用。

② 对变量 sname 重新赋值，赋值运算符是“:=”。

③ dbms_output.put_line 是输出语句。

对变量赋值还可以使用 SELECT…INTO 语句从数据库中查询数据对变量进行赋值。但是

查询的结果只能是一行记录，不能是零行或者多行记录。

代码演示：变量赋值

```
DECLARE
  SNAME VARCHAR2(20);
BEGIN
  SELECT ENAME INTO SNAME FROM EMP WHERE EMPNO = 7934;   --隐式游标赋值
  DBMS_OUTPUT.PUT_LINE(SNAME);
  END;
```

代码解析：

① 使用 select…into 语句对变量 sname 赋值，要求查询的结果必须是一行，不能是多行或者没有记录。

2.1.2 声明常量

常量在声明时赋予初值，并且在运行时不允许重新赋值。使用 CONSTANT 关键字声明常量。

代码演示：声明常量

```
DECLARE
  PI CONSTANT NUMBER := 3.14; --圆周率长值   ①
  R      NUMBER DEFAULT 3; --圆的半径默认值 3   ②
  AREA NUMBER; --面积。
BEGIN
  AREA := PI * R * R; --计算面积
  DBMS_OUTPUT.PUT_LINE(AREA); --输出圆的面积
  END;
```

代码解析：

① 声明常量时使用关键字 CONSTANT，常量初值可以使用赋值运算符（:=）赋值，也可以使用 DEFAULT 关键字赋值。

2.1.2 声明属性数据类型

%ROWTYPE：引用数据库中的一行（所有字段）作为数据类型。

%TYPE：引用数据库中的某列的数据类型或某个变量的数据类型。

【例】找出员工编号为 7934 的员工名称和工资

代码演示：%ROWTYPE

```
DECLARE
  MYEMP EMP%ROWTYPE;
BEGIN
```

```
    SELECT E.ENAME,E.SAL INTO MYEMP.ENAME,MYEMP.SAL FROM EMP E WHERE E.EMPNO =
7934;
    DBMS_OUTPUT.PUT_LINE(MYEMP.ENAME||'的工资是'||MYEMP.SAL);
END;
```

代码演示：%TYPE

```
DECLARE
    SNAME EMP.ENAME%TYPE;
    SSAL    EMP.SAL%TYPE;
BEGIN
    SELECT E.ENAME, E.SAL INTO SNAME, SSAL FROM EMP E WHERE E.EMPNO = 7934;
    DBMS_OUTPUT.PUT_LINE(SNAME || '的工资是' || SSAL);
END;
```

2.2 条件控制

PL/SQL 中关于条件控制的关键字有 IF-THEN、IF-THEN-ELSE、IF-THEN-ELSIF 和多分枝条件 CASE。

IF-THEN	IF-THEN-ELSE	IF-THEN-ELSIF	CASE

IF 条件 THEN
--条件结构体
END IF;
	IF 条件 THEN
	--条件成立结构体
	ELSE
	--条件不成立结构体
	END IF;
		IF 条件 1 THEN
		--条件 1 成立结构体
		ELSIF 条件 2 THEN
		--条件 2 成立结构体
		ELSE
		--以上条件都不成立结构体
		END IF;
			CASE [selector]
			WHEN 表达式 1 THEN 语句序列 1;
			WHEN 表达式 2 THEN 语句序列 2;
			WHEN 表达式 3 THEN 语句序列 3;
			……
			ELSE 语句序列 N;
			END CASE;

➢ IF-THEN

该结构先判断一个条件是否为 TRUE，条件成立则执行对应的语句块。

① 用 IF 关键字开始，END IF 关键字结束，注意 END IF 后面有一个分号。

② 条件部分可以不使用括号，但是必须以关键字 THEN 来标识条件结束，如果条件成立，则执行 THEN 后到对应 END IF 之间的语句块内容。如果条件不成立，则不执行条件语句块的内容。

③ 条件可以使用关系运算符合逻辑运算符。

④ 在 PL/SQL 块中可以使用事务控制语句，该 COMMIT 同时也能把 PL/SQL 块外没有提交的数据一并提交，使用时需要注意。

➢ IF-THEN-ELSE

把 ELSE 与 IF-THEN 连在一起使用，如果 IF 条件不成立则执行就会执行 ELSE 部分的语句。

➢ IF-THEN-ELSIF

PL/SQL 中的再次条件判断中使用关键字 ELSIF。

【例】查询 JAMES 的工资，如果大于 1500 元，则发放奖金 100 元，如果工作大于 900 元，则发奖金 800 元，否则发奖金 400 元。

代码演示：IF-THEN-ELSIF 应用

```
DECLARE
  NEWSAL EMP.SAL % TYPE;
BEGIN
  SELECT SAL INTO NEWSAL FROM EMP WHERE ENAME = 'JAMES';
  IF NEWSAL > 1500 THEN
    UPDATE EMP SET COMM = 1000 WHERE ENAME = 'JAMES';
  ELSIF NEWSAL > 900 THEN
    UPDATE EMP SET COMM = 800 WHERE ENAME = 'JAMES';
  ELSE
    UPDATE EMP SET COMM = 400 WHERE ENAME = 'JAMES';
  END IF;
END;
```

➢ CASE

CASE 是一种选择结构的控制语句，可以根据条件从多个执行分支中选择相应的执行动作。

如果存在选择器 selector，选择器 selector 与 WHEN 后面的表达式匹配，匹配成功就执行 THEN 后面的语句。如果所有表达式都与 selector 不匹配，则执行 ELSE 后面的语句。

如果不使用 CASE 中的选择器，直接在 WHEN 后面判断条件，第一个条件为真时，执行对应 THEN 后面的语句序列。

2.3 循环控制

PL/SQL 提供了丰富的循环结构来重复执行一些列语句。Oracle 提供的循环类型有：

1. 无条件循环 LOOP-END LOOP 语句
2. WHILE 循环语句
3. FOR 循环语句

在上面的三类循环中 EXIT 用来强制结束循环。

➢ LOOP 循环

LOOP 循环是最简单的循环，也称为无限循环，LOOP 和 END LOOP 是关键字。

语法格式：LOOP 循环

```
LOOP
    --循环体
END LOOP;
```

语法格式：

1. 循环体在 LOOP 和 END LOOP 之间，在每个 LOOP 循环体中，首先执行循环体中的语句序列，执行完后再重新开始执行。
2. 在 LOOP 循环中可以使用 EXIT 或者[EXIT WHEN 条件]的形式终止循环。否则该循环就是死循环。

【例】执行 1+2+3+…+100 的值

代码演示：LOOP 循环

```
DECLARE
  COUNTER    NUMBER(3) := 0;
  SUMRESULT NUMBER := 0;
BEGIN
  LOOP
    COUNTER    := COUNTER + 1;
    SUMRESULT := SUMRESULT + COUNTER;
    IF COUNTER >= 100 THEN      --①
      EXIT;
    END IF;
    -- EXIT WHEN COUNTER>=100;    ②
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('RESULT IS :' || TO_CHAR(SUMRESULT));
  END;
```

代码解析：

① LOOP 循环中可以使用 IF 结构嵌套 EXIT 关键字退出循环

② 注释行，该行可以代替①中的循环结构，WHEN 后面的条件成立时跳出循环。

➢ WHILE 循环

先判断条件，条件成立再执行循环体。

语法格式：WHILE 循环

```
WHILE 条件 LOOP
    --循环体
    END LOOP;
```

代码演示：WHILE 循环

```
DECLARE
  COUNTER    NUMBER(3) := 0;
  SUMRESULT NUMBER := 0;
BEGIN
  WHILE COUNTER < 100 LOOP
    COUNTER    := COUNTER + 1;
    SUMRESULT := SUMRESULT + COUNTER;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('RESULT IS :' || SUMRESULT);
  END;
```

- FOR 循环

FOR 循环需要预先确定的循环次数，可通过给循环变量指定下限和上限来确定循环运行的次数，然后循环变量在每次循环中递增（或者递减）。FOR 循环的语法是：

语法格式：FOR 循环

```
FOR 循环变量 IN [REVERSE] 循环下限..循环上限 LOOP LOOP
  --循环体
  END LOOP;
```

语法解析：

循环变量：该变量的值每次循环根据上下限的 REVERSE 关键字进行加 1 或者减 1。

REVERSE：指明循环从上限向下限依次循环。

代码演示：FOR 循环

```
DECLARE
  COUNTER    NUMBER(3) := 0;
  SUMRESULT NUMBER := 0;
BEGIN
  FOR COUNTER IN 1 .. 100 LOOP
    SUMRESULT := SUMRESULT + COUNTER;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('RESULT IS :' || SUMRESULT);
END;
```

2.4 游标

游标是指向查询结果集的一个指针，通过游标可以将查询结果集中的记录逐一取出，并在 PL/SQL 程序块中进行处理。

游标的类型有两种：隐式游标和显示游标。隐式游标是由系统自动创建并管理的游标。PL/SQL 会为所有的 SQL 数据操作声明一个隐式的游标，包括只返回一条记录的查询操作。对于返回多条记录的查询，必须自己创建显示游标。

具体采用 OPEN、FETCH 和 CLOSE 语句来控制游标。OPEN 用于打开游标并使游标指向结果集的第一行，FETCH 会检索当前行的信息并把游标指向下一行，当最后一行被处理完后，CLOSE 就会关闭游标。

语法结构：声明游标

```
CURSOR 游标名[(参数 1 数据类型[, 参数 2 数据类型...])]
IS SELECT 语句;  --游标的声明
```

游标命名规范： C_游标名

语法结构：执行游标

```
OPEN 游标名[(实际参数 1[, 实际参数 2...])];  --打开游标
FETCH 游标名 INTO 变量名 1[, 变量名 2...];
或
FETCH 游标名 INTO 记录变量;  --提取数据
CLOSE 游标名;  --关闭游标（千万别忘了！）
```

游标属性：%FOUND 和%NOTFOUND

%FOUND:
用于判断游标是否从结果集中提取数据。如果提取到数据，则返回值为 TRUE，否则返回值为 FALSE。
%NOTFOUND:
该属性与%FOUND 相反，如果提取到数据则返回值为 FALSE；如果没有，则返回值为 TRUN。

【例】查询 EMP 表中经理的员工编号、姓名、工作和工资

语法结构：方法 1

```
DECLARE
  --类型定义
  CURSOR C_JOB IS
```

```
    SELECT EMPNO, ENAME, JOB, SAL FROM EMP WHERE JOB = 'MANAGER';
  --定义一个游标变量
  C_ROW C_JOB%ROWTYPE;
BEGIN
  OPEN C_JOB;
  LOOP
    --提取一行数据到C_ROW
    FETCH C_JOB
      INTO C_ROW;
    --判读是否提取到值，没取到值就退出
    --取到值 C_JOB%NOTFOUND 是 FALSE
    --取不到值 C_JOB%NOTFOUND 是 TRUE
    EXIT WHEN C_JOB%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(C_ROW.EMPNO || '-' || C_ROW.ENAME || '-' ||
                         C_ROW.JOB || '-' || C_ROW.SAL);
  END LOOP;
  --关闭游标
  CLOSE C_JOB;
END;
```

语法结构：方法 2 FOR 循环

```
DECLARE
  --定义一个游标变量
  CURSOR C_JOB IS
    SELECT EMPNO, ENAME, JOB, SAL FROM EMP WHERE JOB = 'MANAGER';
BEGIN
  FOR X IN C_JOB LOOP       --不需要 OPEN、FETCH、CLESE
    DBMS_OUTPUT.PUT_LINE(X.EMPNO || '-' || X.ENAME || '-' ||
                         X.JOB || '-' || X.SAL);
  END LOOP;
END;
```

语法解析：

游标 FOR 循环不需要事先定义，它会隐式声明一个代表当前的循环索引变量。系统自动打开游标，当所有行都被处理后，就会自动关闭游标，不需要人为操作。

2.5 动态 SQL

在 PL/SQL 程序开发中，可以使用 DML 语句和事务控制语句，但是还有很多语句（比如 DDL 语句）不能直接在 PL/SQL 中执行。这些语句可以使用动态 SQL 来实现。

PL/SQL 块先编译然后再执行，动态 SQL 语句在编译时不能确定，只有在程序执行时把 SQL 语句作为字符串的形式由动态 SQL 命令来执行。在编译阶段 SQL 语句作为字符串存在，

程序不会对字符串中的内容进行编译，在运行阶段再对字符串中的 SQL 语句进行编译和执行，动态 SQL 的语法是：

语法格式：动态 SQL

```
EXECUTE IMMEDIATE 动态语句字符串
[INTO 变量列表]
[USING 参数列表]
```

语法解析：

如果动态语句是 SELECT 语句，可以把查询的结果保存到 INTO 后面的变量中。如果动态语句中存在参数，USING 为语句中的参数传值。

动态 SQL 中的参数格式是：[:参数名]，参数在运行时需要使用 USING 传值。

【例】在过程中复制 EMP 表

```
BEGIN

  EXECUTE   IMMEDIATE 'CREATE TABLE YYY AS SELECT * FROM EMP';
   END;
```

3.创建存储过程 SP（stored procedure）

存储过程是一个命名的程序块，包括过程的名称、过程使用的参数以及过程执行的操作。如果在应用程序中经常需要执行某些特定的操作，那么就可以基于这些操作创建一个特定的存储过程。存储过程经编译后存储在数据库中，所有执行存储过程要比执行存储过程中的封装的 SQL 语句更有效率。

语法格式：创建存储过程

```
CREATE OR REPLACE PROCEDURE 过程名
(参数 1 [IN|OUT|IN OUT] 数据类型，参数 2 [IN|OUT|IN OUT] 数据类型……)
IS|AS
PL/SQL 过程体;
```

①OR REPLACE：表示如果存储过程已经存在，则替换已有的存储过程
②IN 表示传入参数，不可以被赋值，OUT 表示传出参数，可以被赋值，IN OUT 表示传入传出参数，可以传入值，可以被赋值，可以返回值。如果这部分省略，默认表示传入参数。创建函数可以带参数，也可以不带。
③IS/AS：在 IS/AS 后声明变量不要加 DECLARE 语句

命名规范：

存储过程命名规范：SP_目标表名

存储过程传入参数命名规范：P_参数名（P_START_DATE）

存储过程变量命名规范：V_变量名(V_END_DATE)

语法格式：调用存储过程

```
BEGIN
   过程名[(参数)];
END;
```

语法格式：删除存储过程

```
DROP PROCEDURE 过程名;
```

【例】查询 20 部门中的经理的姓名、工资、入职日期。

```
--创建存储过程
CREATE  OR  REPLACE  PROCEDURE SP_MYEMP--存储过程的名字
                              ( P_DEPTNO   IN NUMBER,--参数 1，输入，数据类型 1
                                P_HIREDATE   OUT DATE,--参数 2，输出，数据类型 2
                                P_JOB_ENAME   IN   OUT   VARCHAR2 ) --参数 3，输入/输
                                                                    出，数据类型 3
AS                          ----------参数不能定义长度
V_SAL   NUMBER;
BEGIN
  SELECT   E.ENAME, E.SAL, E.HIREDATE
    INTO   P_JOB_ENAME, V_SAL, P_HIREDATE
    FROM   EMP E
   WHERE   E.DEPTNO = P_DEPTNO
     AND   E.JOB = P_JOB_ENAME;
  DBMS_OUTPUT.PUT_LINE (P_JOB_ENAME || ' ' || V_SAL || ' ' || P_HIREDATE);
END;

--调用存储过程
DECLARE
  V_JOB_ENAME   VARCHAR2(100) := 'MANAGER';
  V_HIREDATE   DATE;
BEGIN
  SP_MYEMP( 20, V_HIREDATE, V_JOB_ENAME );
END;
```

4. 创建自定义函数

Oracle 的自定义函数与存储过程很相似，同样可以接受用户的传递值，也可以向用户返回值，它与存储过程的不同之处在于，函数必须返回一个值，而存储过程可以不返回任何值。

语法格式：创建函数

```
CREATE [OR REPLACE] FUNCTION 函数名(参数 1 数据类型，参数 2，[IN|OUT|IN OUT] 数据类型……)
RETURN 返回的数据类型
IS|AS
PL/SQL 函数体; --里面必须要有一个 RETURN 子句
```

语法格式：删除函数

```
DROP FUNCTION 函数名;
```

Oracle 存储过程（procedure）和函数（Function）的区别：
1. 返回值的区别，函数有 1 个返回值，而存储过程是通过参数返回的，可以有多个或者没有
2. 调用的区别，函数可以在查询语句中直接调用，而存储过程必须单独调用.
函数一般情况下是用来计算并返回一个计算结果而存储过程一般是用来完成特定的数据操作（比如修改、插入数据库表或执行某些 DDL 语句等等）

5. 异常处理

在程序运行时出现的错误，称为异常。发生异常后，语句将停止执行，PL/SQL 引擎立即将控制权转到 PL/SQL 块的异常处理部分。异常处理机制简化了代码中的错误检测。PL/SQL 中任何异常出现时，每一个异常都对应一个异常码和异常信息。

为了 Oracle 开发和维护的方便，在 Oracle 异常中，为常见的异常码定义了对应的异常名称，称为预定义异常，常见的预定义异常有：

异常名称	异常码	描述
DUP_VAL_ON_INDEX	ORA-00001	试图向唯一索引列插入重复值
INVALID_CURSOR	ORA-01001	试图进行非法游标操作。
INVALID_NUMBER	ORA-01722	试图将字符串转换为数字
NO_DATA_FOUND	ORA-01403	SELECT INTO 语句中没有返回任何记录。
TOO_MANY_ROWS	ORA-01422	SELECT INTO 语句中返回多于 1 条记录。
ZERO_DIVIDE	ORA-01476	试图用 0 作为除数。
CURSOR_ALREADY_OPEN	ORA-06511	试图打开一个已经打开的游标

表 6　PL/SQL 中预定义异常

PL/SQL 中用 EXCEPTION 关键字开始异常处理。

语法格式：异常处理

```
BEGIN
    --可执行部分
    EXCEPTION    -- 异常处理开始
        WHEN 异常名 1 THEN
            --对应异常处理
        WHEN 异常名 2 THEN
            --对应异常处理
         ……
        WHEN OTHERS THEN
            --其他异常处理
        END;
```

语法解析：

异常发生时，进入异常处理部分，具体的异常与若干个 WHEN 子句中指明的异常名匹配，匹配成功就进入对应的异常处理部分，如果对应不成功，则进入 OTHERS 进行处理。

6.同步数据

审计字段：即记录数据的创建人、创建时间、修改人、修改时间的字段等，体现在每一张数据库表中。通过审计字段，可以知道数据怎么来的，什么时候同步的，若出现问题也可追溯。（**同步之后就要比对两个表是否是一致，方法 1.返回的行数是否是一致 2.也可以用 pdlink 进行比对等方法） 同步前要清空目标表（必须做的工作）目的为了支持重跑。**

①清空目标表

②逻辑

③ 插入目标表

4.1 全量抽取

将目标表的数据全部删除，再将源系统的数据全部插入目标表。此方法保证了数据的质量，但是对于数据量大的表而言，性能太差。

【例】同步 EMP 表

```
--创建目标表表结构
CREATE TABLE EMPEMP
(EMPNO    NUMBER(4)
```

```
,ENAME    VARCHAR2(10)
,JOB    VARCHAR2(9)
,MGR    NUMBER(4)
,HIREDATE    DATE
,SAL    NUMBER(7,2)
,COMM    NUMBER(7,2)
,DEPTNO    NUMBER(2) ,
 LAST_UPDATE_DATE DATE
)
ALTER TABLE     EMPEMP    ADD CONSTRAINT    PK_EMPNO    PRIMARY KEY(EMPNO)
--创建存储过程
CREATE OR REPLACE PROCEDURE SP_EMPA IS
BEGIN
   EXECUTE IMMEDIATE 'TRUNCATE TABLE EMPEMP';
   INSERT INTO EMPEMP EE
      (EE.EMPNO,
       EE.ENAME,
       EE.JOB,
       EE.MGR,
       EE.HIREDATE,
       EE.SAL,
       EE.COMM,
       EE.DEPTNO,
       EE.LAST_UPDATE_DATE)
      SELECT E.EMPNO,
                  E.ENAME,
                  E.JOB,
                  E.MGR,
                  E.HIREDATE,
                  E.SAL,
                  E.COMM,
                  E.DEPTNO,
                  SYSDATE
           FROM EMP E;
   COMMIT;
END;
--调用
BEGIN
SP_EMPA;
END;
```

注意：

1. 尽量不使用*，*不直观，且有时会使数据插入不对应的字段。
2. 表最好别名，给系统减轻识别负担，不易产生分歧。

3.工作中做全量抽取时，通常先判断源表是否有数据，再执行。如果源表没有数据了，同步之后目标表也没有数据了，只能从硬盘里找回数据，成本代价高。

4.2 增量抽取(以日期为区间)

只需抽取新增的或修改的数据。此方法性能好，但容易遗漏。有时源表更新的字段，在目标表中不存在，则不需要更新。以时间戳取增量，对源表删除的数据无能为力。

--方法一

语法格式：MERGE（不是所有数据库都通用）

```
MERGE INTO 目标表
USING (增量)
ON (匹配字段)
WHEN MATCHED THEN UPDATE SET    --UPDATE 和 SET 之间不需要加表名 （匹配字段不可以更改）
WHEN NOT MATCHED THEN INSERT VALUES
                                --INSERT 和 VALUES 之间不需要加 INTO 表名
```

【例】同步 EMP 表数据

```
CREATE OR REPLACE PROCEDURE SP_EMPC(P_START_DATE VARCHAR2,
                                    P_END_DATE VARCHAR2) IS
  V_START_DATE DATE := TO_DATE(P_START_DATE, 'YYYY-MM-DD HH24:MI:SS');
  V_END_DATE     DATE := TO_DATE(P_END_DATE, 'YYYY-MM-DD HH24:MI:SS');
BEGIN
  MERGE INTO EMPEMP A
  USING (SELECT E.EMPNO,
                E.ENAME,
                E.JOB,
                E.MGR,
                E.HIREDATE,
                E.SAL,
                E.COMM,
                E.DEPTNO
           FROM EMP E
          WHERE E.LAST_UPDATE_DATE >= V_START_DATE
            AND E.LAST_UPDATE_DATE < V_END_DATE) B
  ON (A.EMPNO = B.EMPNO)
  WHEN MATCHED THEN
    UPDATE
       SET A.ENAME    = B.ENAME,
           A.JOB      = B.JOB,
           A.MGR      = B.MGR,
```

```
                A.HIREDATE       = B.HIREDATE,
                A.SAL            = B.SAL,
                A.COMM           = B.COMM,
                A.DEPTNO         = B.DEPTNO,
                A.LAST_UPDATE_DATE = SYSDATE
  WHEN NOT MATCHED THEN
    INSERT
      (A.EMPNO,
       A.ENAME,
       A.JOB,
       A.MGR,
       A.HIREDATE,
       A.SAL,
       A.COMM,
       A.DEPTNO,
       A.LAST_UPDATE_DATE)
    VALUES
      (B.EMPNO,
       B.ENAME,
       B.JOB,
       B.MGR,
       B.HIREDATE,
       B.SAL,
       B.COMM,
       B.DEPTNO,
       SYSDATE);
  COMMIT;
END;

BEGIN
  SP_EMPC('2017-04-05 00:00:00','2017-04-06 00:00:00');
  END;
```

--方法二

先将目标表的记录在增量范围之内的删除，然后再将增量插入

```
CREATE  OR  REPLACE  PROCEDURE SP_EMPB(P_START_DATE VARCHAR2,
                                       P_END_DATE VARCHAR2) IS
  V_START_DATE  DATE := TO_DATE(P_START_DATE, 'YYYY-MM-DD HH24:MI:SS');
  V_END_DATE  DATE := TO_DATE(P_END_DATE, 'YYYY-MM-DD HH24:MI:SS');
BEGIN
  DELETE FROM EMPEMP
   WHERE EMPNO IN (SELECT E.EMPNO
```

```
                                FROM EMP E
                               WHERE E.LAST_UPDATE_DATE >= V_START_DATE
                                 AND E.LAST_UPDATE_DATE < V_END_DATE);
  INSERT INTO EMPEMP A
     (A.EMPNO,
      A.ENAME,
      A.JOB,
      A.MGR,
      A.HIREDATE,
      A.SAL,
      A.COMM,
      A.DEPTNO,
      A.LAST_UPDATE_DATE)
     SELECT E.EMPNO,
            E.ENAME,
            E.JOB,
            E.MGR,
            E.HIREDATE,
            E.SAL,
            E.COMM,
            E.DEPTNO,
            SYSDATE
       FROM EMP E
      WHERE LAST_UPDATE_DATE >= V_START_DATE
        AND LAST_UPDATE_DATE < V_END_DATE;
  COMMIT;
END;

BEGIN
  SP_EMPB('2017-04-05 00:00:00','2017-04-06 00:00:00');
  END;
```

--方法三

使用游标，逐一判断每一条增量记录是需要更新还是插入

```
CREATE  OR  REPLACE  PROCEDURE SP_EMPD(P_START_DATE VARCHAR2,
                                       P_END_DATE VARCHAR2) IS
  V_START_DATE   DATE := TO_DATE(P_START_DATE, 'YYYY-MM-DD HH24:MI:SS');
  V_END_DATE     DATE := TO_DATE(P_END_DATE, 'YYYY-MM-DD HH24:MI:SS');
  CURSOR  C_UPDATE IS
    SELECT  E.EMPNO,
            E.ENAME,
```

```
                E.JOB,
                E.MGR,
                E.HIREDATE,
                E.SAL,
                E.COMM,
                E.DEPTNO
          FROM EMP E
         WHERE E.LAST_UPDATE_DATE >= V_START_DATE
           AND E.LAST_UPDATE_DATE < V_END_DATE;
  CT NUMBER;
BEGIN
  FOR X IN C_UPDATE LOOP
    SELECT   COUNT(empno)   INTO     CT FROM EMPEMP A WHERE A.EMPNO = X.EMPNO;
    IF CT = 1 THEN
      UPDATE EMPEMP A
         SET A.ENAME           = X.ENAME,
             A.JOB             = X.JOB,
             A.MGR             = X.MGR,
             A.HIREDATE        = X.HIREDATE,
             A.SAL             = X.SAL,
             A.COMM            = X.COMM,
             A.DEPTNO          = X.DEPTNO,
             A.LAST_UPDATE_DATE = SYSDATE
       WHERE A.EMPNO = X.EMPNO;
    ELSIF CT = 0 THEN
      INSERT INTO EMPEMP A
        (A.EMPNO,
         A.ENAME,
         A.JOB,
         A.MGR,
         A.HIREDATE,
         A.SAL,
         A.COMM,
         A.DEPTNO,
         A.LAST_UPDATE_DATE)
      VALUES
        (X.EMPNO,
         X.ENAME,
         X.JOB,
         X.MGR,
         X.HIREDATE,
         X.SAL,
         X.COMM,
         X.DEPTNO,
```

```
            SYSDATE);
      END IF;
   END LOOP;
   COMMIT;
END;

BEGIN
   SP_EMPD('2017-04-05 00:00:00','2017-04-06 00:00:00');
   END;
```

--方法四

将增量记录逐一插入目标表，如果违反唯一索引，则更新

```
CREATE   OR   REPLACE   PROCEDURE   SP_EMPG(P_START_DATE VARCHAR2,
                                            P_END_DATE     VARCHAR2) IS
   V_START_DATE DATE := TO_DATE(P_START_DATE, 'YYYY-MM-DD HH24:MI:SS');
   V_END_DATE     DATE := TO_DATE(P_END_DATE, 'YYYY-MM-DD HH24:MI:SS');
   CURSOR C_UPDATE IS
      SELECT E.EMPNO,
             E.ENAME,
             E.JOB,
             E.MGR,
             E.HIREDATE,
             E.SAL,
             E.COMM,
             E.DEPTNO
        FROM EMP E
       WHERE E.LAST_UPDATE_DATE >= V_START_DATE
         AND E.LAST_UPDATE_DATE < V_END_DATE;

BEGIN
   FOR X IN C_UPDATE LOOP
      BEGIN
         INSERT INTO EMPEMP A
           (A.EMPNO,
            A.ENAME,
            A.JOB,
            A.MGR,
            A.HIREDATE,
            A.SAL,
            A.COMM,
            A.DEPTNO,
```

```
            A.LAST_UPDATE_DATE)
        VALUES
          (X.EMPNO,
            X.ENAME,
            X.JOB,
            X.MGR,
            X.HIREDATE,
            X.SAL,
            X.COMM,
            X.DEPTNO,
            SYSDATE);
    EXCEPTION
      WHEN DUP_VAL_ON_INDEX THEN
        UPDATE EMPEMP A
           SET A.ENAME           = X.ENAME,
               A.JOB             = X.JOB,
               A.MGR             = X.MGR,
               A.HIREDATE        = X.HIREDATE,
               A.SAL             = X.SAL,
               A.COMM            = X.COMM,
               A.DEPTNO          = X.DEPTNO,
               A.LAST_UPDATE_DATE = SYSDATE
         WHERE A.EMPNO = X.EMPNO;
    END;
  END LOOP;
END;

BEGIN
 SP_EMPG('2017-04-05 00:00:00','2017-04-06 00:00:00');
END;
```

MERGE 的灵活应用：

若数据量大时，MERGE 删除比 DELETE 好。

```
MERGE INTO EMP E
USING (SELECT * FROM EMP S WHERE S.DEPTNO = 10) S
ON (S.EMPNO = E.EMPNO)
WHEN MATCHED THEN
     UPDATE SET E.COMM = 1 DELETE WHERE 1 = 1;     --SET 随便改，后面接 DELETE
```

如果源表数据删除了怎么办？

源表的数据一般不会被删除，通常表后面会加一个删除标识字段，用来标识这条记录是否失效，即软删除（华为称为打 PD），证明这条记录存在过。

如果源表有数据删除了，可以查看源表的日志，看哪些记录被删除了，在目标表相应记录上加上删除标识，但是此方法代价大，成本高。

两张表 A、B 关联抽数如何抽取？
--方法 1
两张表全量关联，A 表的时间 or B 表的时间在时间范围内
--方法 2
(A 表的增量与 B 表的全量关联)UNION ALL(B 表的增量与 A 表的全量关联)
--方法 3
(A 的增量主键 union all B 增量主键 group by 主键)left join A on 主键 left join B on 主键

7. 创建包

包就是把相关的存储过程、函数、变量、常量和游标等 PL/SQL 程序组合在一起，并赋予一定的管理功能的程序块。

一个程序包由两部分组成：包定义和包体。其中包定义部分声明包内数据类型、变量、常量、游标、子程序和函数等元素，这些元素为包的共有元素。包主体则定义了包定义部分的具体实现。

语法格式：创建包头

```
CREATE [OR REPLACE] PACKAGE 包名
IS|AS
变量、常量及数据类型定义;
游标定义头部;
函数、过程的定义和参数列表以及返回类型;
END [包名];
```

代码演示：

```
CREATE OR REPLACE PACKAGE MYPACKAGE
AS
A NUMBER;
B VARCHAR2;
PROCEDURE MY_SP(P_A IN NUMBER);
FUNCTION MY_FUN(F_A   IN   NUMBER) RETURN NUMBER;
END;
```

语法格式：创建包体

```
CREATE [OR REPLACE] PACKAGE BODY 包名
IS|AS

PROCEDURE 过程名(参数)
```

```
IS|AS
BEGIN
过程体;
END [过程名];

FUNCTION  函数名(参数) RETURN  类型
IS|AS
BEGIN
函数体;
END [函数名];

END;
```

语法格式：调用包

```
BEGIN
包名.变量名|常量名
包名.游标名[(参数)]
包名.函数名[(参数)]|过程名[(参数)]
END;
```

语法格式：删除包

```
DROP PACKAGE  包名
```

8. 创建日志

日志是用来追溯问题的，记录整个程序的运行情况，知道哪个环节报错了，记录每一步花了多少时间，判断哪一步性能不好，从而对程序进行修改和优化。

人为创建的日志区别于 Oracle 系统自带的日志，后者调用的成本比较高。

通常报错的时候，会有很多条报错信息，第一条是真正报错的原因。

一般 SP 里都会有调用日志语句。

日志表一般有哪些字段？

日志 ID	过程名	批次 ID	步骤	完成时间	备注	.....
1	SP_1	1	1	2017/07/04 08：00	步骤 1 完成	
2	SP_1	1	2	2017/07/04 08：20	步骤 2 完成	
3	SP_1	1	3	2017/07/04 08：30	步骤 3 完成	
4	SP_2	2	1	...	...	...

5	SP_2	2	2			
6	SP_2	2	3			
7	SP_1	3	1			
8	SP_1	3	2			
9	SP_1	3	3			

语法格式：创建日志

```
--创建日志表
CREATE TABLE LOG_RECORD
( LOG_ID NUMBER,SP_NAME VARCHAR2(100),CYCLE_ID NUMBER,STEP
NUMBER,FINISH_TIME DATE,REMARKS VARCHAR2(100))

--创建序列用于 LOG_ID
CREATE SEQUENCE SEQ_LOG_ID

--创建序列用于 CYCLE_ID
CREATE SEQUENCE SEQ_CYCLE_ID

--创建存储过程日志记录
CREATE OR REPLACE PROCEDURE SP_LOG(P_SP_NAME    VARCHAR2,
                                   P_CYCLE_ID NUMBER,
                                   P_STEP       NUMBER,
                                   P_REMARKS    VARCHAR2) IS
BEGIN
  INSERT INTO LOG_RECORD
    (LOG_ID, SP_NAME, CYCLE_ID, STEP, FINISH_TIME, REMARKS)
  VALUES
    (SEQ_LOG_ID.NEXTVAL, P_SP_NAME, P_CYCLE_ID, P_STEP, SYSDATE, P_REMARKS);
  COMMIT;
END;
```

语法格式：调用日志

```
CREATE OR REPLACE PROCEDURE SP_EMPD(P_START_DATE VARCHAR2,
                                    P_END_DATE     VARCHAR2) IS
  V_START_DATE DATE := TO_DATE(P_START_DATE, 'YYYY-MM-DD HH24:MI:SS');
  V_END_DATE     DATE := TO_DATE(P_END_DATE, 'YYYY-MM-DD HH24:MI:SS');
  CURSOR C_UPDATE IS
    SELECT E.EMPNO,
           E.ENAME,
           E.JOB,
           E.MGR,
           E.HIREDATE,
```

```
                E.SAL,
                E.COMM,
                E.DEPTNO
          FROM EMP E
         WHERE E.LAST_UPDATE_DATE >= V_START_DATE
           AND E.LAST_UPDATE_DATE < V_END_DATE;
  CT NUMBER;
  CYCLE_ID NUMBER;    --定义变量

BEGIN
  CYCLE_ID := SEQ_CYCLE_ID.NEXTVAL;
  SP_LOG('SP_EMPD', CYCLE_ID, 1, '程序开始');              --调用日志：程序开始
  FOR X IN C_UPDATE LOOP
    SELECT COUNT(1) INTO CT FROM EMPEMP A WHERE A.EMPNO = X.EMPNO;
    IF CT = 1 THEN
      UPDATE EMPEMP A
         SET A.ENAME         = X.ENAME,
             A.JOB           = X.JOB,
             A.MGR           = X.MGR,
             A.HIREDATE      = X.HIREDATE,
             A.SAL           = X.SAL,
             A.COMM          = X.COMM,
             A.DEPTNO        = X.DEPTNO,
             A.LAST_UPDATE_DATE = SYSDATE
       WHERE A.EMPNO = X.EMPNO;
    ELSIF CT = 0 THEN
      INSERT INTO EMPEMP A
        (A.EMPNO,
         A.ENAME,
         A.JOB,
         A.MGR,
         A.HIREDATE,
         A.SAL,
         A.COMM,
         A.DEPTNO,
         A.LAST_UPDATE_DATE)
      VALUES
        (X.EMPNO,
         X.ENAME,
         X.JOB,
         X.MGR,
         X.HIREDATE,
         X.SAL,
         X.COMM,
```

```
                X.DEPTNO,
                SYSDATE);
        END IF;
    END LOOP;
    SP_LOG('SP_EMPD', CYCLE_ID, 2, '同步完成’);      --调用日志：同步完成
    COMMIT;
END;

BEGIN
    SP_EMPD('2017-04-05 00:00:00', '2017-04-06 00:00:00');
END;
```

第 6 章

BI 理论基础

1.数据仓库

什么是数据仓库？

数据仓库（Data Warehouse）是一个面向主题的（Subject Oriented）、集成的（Integrated）、相对稳定的（Non-Volatile）、反映历史变化（Time Variant）的数据集合，用于支持管理决策(Decision Making Support)。

面向主题的：经过 ETL 抽数、清洗、转换加载后，数据按不同主题存放在同一个库中，梳理归类；

集成的：不来来源的数据的集合。

相对稳定的：不会人为改变任何数据，只同步。如果源系统出故障了，数据仓库的数据依然存在。

反应历史变化：源系统数据库一般只保存几个月，定期删除，数据仓库可保存几年后压缩在硬盘里。

在源系统也能做报表，为什么要建立数仓？

因为每个源系统都有自己的功能，可以在这里进行分析或抽其他源系统的数据，但会影响这个系统的功能；其次，应用系统之间会重复抽取数据，为了避免源系统频繁交互，需要构建数仓。

2.维表与事实表

事实表

每个数据仓库都包含一个或者多个事实数据表。事实数据表可能包含业务销售数据，如现金登记事务

所产生的数据，事实数据表通常包含大量的行。事实数据表的主要特点是包含数字数据（事实），并且这些数字信息可以汇总，以提供有关单位作为历史的数据，每个事实数据表包含一个由多个部分组成的索引，该索引包含作为外键的相关性纬度表的主键，而维度表包含事实记录的特性。事实数据表不应该包含描述性的信息，也不应该包含除数字度量字段及使事实与纬度表中对应项的相关索引字段之外的任何数据。

包含在事实数据表中的“度量值”有两中：一种是可以累计的度量值，另一种是非累计的度量值。最有用的度量值是可累计的度量值，其累计起来的数字是非常有意义的。用户可以通过累计度量值获得汇总信息，例如。可以汇总具体时间段内一组商店的特定商品的销售情况。非累计的度量值也可以用于事实数据表，单汇总结果一般是没有意义的，例如，在一座大厦的不同位置测量温度时，如果将大厦中所有不同位置的温度累加是没有意义的，但是求平均值是有意义的。

一般来说，一个事实数据表都要和一个或多个纬度表相关联，用户在利用事实数据表创建多维数据集时，可以使用一个或多个维度表。

维度表

维度表可以看作是用户来分析数据的窗口，纬度表中包含事实数据表中事实记录的特性，有些特性提供描述性信息，有些特性指定如何汇总事实数据表数据，以便为分析者提供有用的信息，维度表包含帮助汇总数据的特性的层次结构。例如，包含产品信息的维度表通常包含将产品分为食品、饮料、非消费品等若干类的层次结构，这些产品中的每一类进一步多次细分，直到各产品达到最低级别。

在维度表中，每个表都包含独立于其他维度表的事实特性，例如，客户维度表包含有关客户的数据。维度表中的列字段可以将信息分为不同层次的结构级。

结论：

1、事实表就是你要关注的内容；
2、维度表就是你观察该事务的角度，是从哪个角度去观察这个内容的。
【例】某地区商品的销量，是从地区这个角度观察商品销量的。事实表就是销量表，维度表就是地区表。
3、上线时，都是先跑维度表，再跑事实表

3.三范式

三范式
第一范式（1NF）：所有字段值都是不可分解的原子值，即不能同行同列出现两个值。
第二范式（2NF）：满足第一范式为前提，一定要有主属性键
，且每一列都和主键相关，而不能只与主键的某一部分相关。
【例】成绩表 SC 中有字段 SNO,CNO,SCORE,SNAME，其中 SNO 和 CNO 为组合主键，SNAME 只依赖于 SNO，因此违反二范式。
第三范式（3NF）：满足第二范式为前提，每一列数据都和主键直接相关，而不能间接相关。
【例】EMP 表中同时存在 DEPTNO 和 DNAME,EMP 表中主属性为 EMPNO,DNAME 直接依赖于 DEPTNO,间接依赖于 EMPNO，因此违反三范式。
第一范式（无重复的列）
定义：数据库表的每一列都是不可分割的原子数据项，而不能是集合，数组，记录等非原子数据项。如果实体中的某个属性有多个值时，必须拆分为不同的属性
通俗解释：一个字段只存储一项信息
eg:班级：高三年 1 班，应改为 2 个字段，一个年级、一个班级，才满足第一范式
不满足第一范式
学号　　姓名　　班级
0001　　小红　　高三年 1 班
改成
学号　　姓名　　年级　　班级
0001　　小红　　高三年　1 班
第二范式（属性完全依赖于主键）
定义：满足第一范式前提，当存在多个主键的时候，才会发生不符合第二范式的情况。比如有两个主键，不能存在这样的属性，它只依赖于其中一个主键，这就是不符合第二范式

通俗解释：任意一个字段都只依赖表中的同一个字段
eg:比如不符合第二范式
学生证 名称 学生证号 学生证办理时间 借书证名称 借书证号 借 书 证 办理
改成 2 张表如下
学生证表
学生证 学生证号 学生证办理时间
借书证表
借书证 借书证号 借书证时间
第三范式（属性不能传递依赖于主属性）
定义：满足第二范式前提，如果某一属性依赖于其他非主键属性，而其他非主键属性又依赖于主键，那么这个属性就是间接依赖于主键，这被称作传递依赖于主属性。
通俗理解：一张表最多只存 2 层同类型信息
eg:爸爸资料表，不满足第三范式
爸爸 儿子 女儿 女儿的小熊 女儿的海绵宝宝
改成
爸爸信息表：
爸爸 儿子 女儿
女儿信息表
女儿 女儿的小熊 女儿的海绵宝宝

4.星型模型与雪花模型

星型模型：所有的维度表都能直接跟事实表关联，存在冗余数据，一般来说性能会更好
雪花模型：一个或多个维度表没有直接跟事实表关联，需要通关其他维度表才能关联到事实表，去除了冗余数据，因为跟维度表要关联多次，所以效率不一定有星型模型好

比较优缺点：

星型模型因为数据的冗余所以很多统计查询不需要做外部的连接，因此一般情况下效率比雪花型模型要高。星型结构不用考虑很多正规化的因素，设计与实现都比较简单。

雪花型模型由于去除了冗余，有些统计就需要通过表的联接才能产生，所以效率不一定有星型模型高。正规化也是一种比较复杂的过程，相应的数据库结构设计、数据的 ETL、以及后期的维护都要复杂一些。

因此在冗余可以接受的前提下，实际运用中星型模型使用更多，也更有效率。

5.代理主键/业务主键

代理主键是与业务不相关的

业务主键
优点：
1.具有更好的检索性能。

2.直观，更好可读和便于理解。
3.数据迁移更加容易。
缺点：
1.关联性能相对不好，占空间。
2.某一业务属性发生变化，会牵连很多表，修改代价大。

代理主键
优点：
1.纯数字，占用空间少，关联性能好。
2.在业务属性发生变化时，减少了对系统的影响范围。
举例：产品编码规则发生变化。此时，产品编码不是主键，所以只需要按照新的编码规则更改产品实体表内的“业务编号”，而不会影响到其他实体。
缺点：
1.数据迁移比较麻烦，存在重复 ID。
2.展现时需要与对应的维表关联，多做一次映射转换的动作。
3.代理主键不能被改变。
对业务主键和代理主键的取舍，更多的是需要从系统、应用环境、实体属性与关系、开发效率、系统性能和维护成本等多方面去思考。

6.保存历史数据的方法

TYPE1:不保存历史数据，直接更新
TYPE2:保存所有的历史数据
TYPE3:保存当前或者上一次的历史数据

Type 1
不记录历史数据。
一切不需要维护的历史数据都可以选择 Type 1 。
假设地理信息中的国家名称发生更改，像这种数据基本上不需要维护的话，那么就直接使用 Type 1 SCD 覆盖旧的国家名称。

Type 2
添加新的数据。
使用的比较常见，基本上除了 Type 1 SCD 之外的情形都会优先考虑 Type 2 SCD。
通常的缓慢变化维（Slowly Changing Dimension）指的就是 type2。

Type 3
添加历史列。
不会追踪所有的历史记录，只会追踪上一次的历史信息。
这种情况往往介于 Type 1 和 Type 2 的时候会考虑，需要记录历史数据，但是又不需要记录那么多。适合单个字段的追踪。
缺点：不知道变化时间

拉链表（做了缓慢变化的表）

员工 KEY	员工 ID	员工姓名	性别	最后更新时间	起始时间	结束时间	有效标识
1	1	郭婉琪	男	6.3	6.3	6.4	0
2	2	覃盛	男	6.4	6.4	9999	1
3	1	郭婉琪	女	6.5	6.5	6.9	0
4	1	郭万祺	女	6.10	6.10	9999	1
...	...	...	...	...	...	...	...

7. 关联机制

在 oracle 中，关联机制有三种方式：嵌套循环、哈希连接、（归并）排序合并连接

嵌套循环关联（NESTED LOOPS JOIN（NL））：是指依次从驱动表中提取一条记录，遍历被探查表，将匹配的记录放入待展示的缓存区中。
优点：适用广，占用内存小，展现快
缺点：需要不停地从硬盘中读取扫描表，性能不好
注意：把两张表最终需要关联的数据对比，大表适合做被探查表，因为可以减少从硬盘读取扫描表的次数。

哈希关联（Hash Join（HJ））：计算出整张被探查表关联字段的哈希值，这些哈希值和整张被探查表一起放入缓存区，然后从驱动表逐条取记录，计算出关联字段对应的哈希值，再与被探查表的哈希值匹配,匹配上了再精准匹配每一条记录。
优点：性能好，匹配次数大大减少
缺点：只适用于等值关联，占用内存较大
注意：把两张表最终需要关联的数据对比，小表适合做被探查表，因为怕缓存不够。如果缓存足够的前提下，大表适合做被探查表。

排序合并连接(Sort Merge Join (SMJ))：是指将关联的 a 表跟 b 表分别进行排序，生成临时的两张表后，随机取一张表逐条抽取记录与另一张表匹配。
优点：适合有索引的两张表或者不等关联
缺点：排序性能消耗大，占用内存大

--怎么看执行计划：
1、看扫描表的方式（全表扫描，索引扫描）
2、看关联方式

查询的执行原理：
1. 提交查询语句
2. 检索缓存区，缓存区会把之前的执行计划暂时保留下来。若语句和缓存区中存在的语句一模一样（多一个空格都不行），则会减少很多解析时间，若缓存区没有，那么执行 3
3. 检查语法
4. 检查表、字段是否存在，检查查询权限

5. 设计执行计划
6. 执行计划

8.HINTS

--常用 HINTS
1、/*+ PARALLEL(表名 1,并行数)[(表名 2,并行数)……] */ --指定开启多少个并行|并发（一般为 2、4、8……）
2、/*+ INDEX(表名,索引名) */ --指定索引
3、/*+ FULL(表名) */ --指定全表扫描
4、/*+ USE_NL(表名 1，表名 2) */ --指定用 NESTED LOOP 连接
5、/*+ USE_HASH(表名 1，表名 2) */ --指定用 HASH 连接
6、/*+ USE_MERGE(表名 1，表名 2) */ --指定用 SORT MERGE JOIN
7、/*+ LEADING(表名 1，表名 2) */ --指定表 1 作为驱动表
8、/*+ APPEND */ --数据直接插入到高水位上面(与 insert 连用)直接往后面插，无视前面的空位置

代码演示：

```
SELECT * /*+ PARALLEL(E,2)(D,2)*/
  FROM EMP E, DEPT D
    WHERE E.DEPTNO = D.DEPTNO
```

9.Shell 介绍

在计算机科学中，Shell 俗称壳（用来区别于核），是指“提供使用者使用界面”的软件（命令解析器）。它接收用户命令，然后调用相应的应用程序。同时它又是一种程序设计语言。作为命令语言，它交互式解释和执行用户输入的命令或者自动地解释和执行预先设定好的一连串的命令；作为程序设计语言，它定义了各种变量和参数，并提供了许多在高级语言中才具有的控制结构，包括循环和分支。

10.项目经验与流程

--项目经验
人员组成
（1）甲方项目经理
（1）乙方项目经理
（2~3）测试人员
（4~5）后台开发人员（数据库开发）

（2~3）前台开发人员（报表开发）
（1~2）设计人员（TA|SA）：表结构设计、源表到目标表的映射关系、抽取规则>>mapping
（1）需求分析人员（BA）：（偏后台的）维度、指标、粒度、取数范围、怎么取增量、跑数频率>>RS 文档（require specification）
维度：看问题的角度（时间+地区+产品　时间+产品+部门）
指标：（衡量的标准）销售额，采购金额
粒度：统计粒度，例如统计人口（统计到户还是统计到人还是村）
取数范围：（重要的保留）过滤不需要的字段

项目经理<<交付经理<<部门经理

项目流程：
需求分析
总体设计
详细设计
开发
测试（UT→SIT→UAT）
上线：把表结构、程序、依赖关系等给上线负责人
运维

UT(单元测试，Unit Test):即自测，首先保证程序不报错，可以不带数空跑，然后根据业务场景造对应的数据，跑出我们需要的结果。
SIT(系统集成测试,System Integration Testing):所有开发放在一起，测试系统兼容性。主要包括功能确认测试、运行测试、强度测试、恢复测试、安全性测试。
UAT(用户可接收测试，User Acceptance Test):向未来的用户表明系统能够像预定要求那样工作。

如何测试优化过的程序？
优化过的程序先在测试环境跑，如果没有语法等问题，进入生产环境，创一个备份表，优化过的程序往备份表里面跑数据，和原来的旧程序同步执行，比较两个表的数据有没有差异，保证数据质量没问题后，执行优化程序 1~2 个月，没有问题，则替换老程序，备份表替换为正式表。

11.工具介绍

SQL 开发工具：PL/SQL Developer
ETL 工具：Datastage、Informatica
调度工具：以前用 Moia、现在用 ConTrol-M
报表工具：Oracle BIEE、BO、水晶报表
建模工具：ER-WIN、PowerDesigner
配置管理工具：Rational ClearCase、RTC、SVN

--ETL 工具

1.什么是 ETL
ETL，是英文 Extract-Transform-Load 的缩写，用来描述将数据从来源端经过抽取（extract）、转换（transform）、加载（load）至目的端的过程。ETL 一词较常用在数据仓库，但其对象并不限于数据仓库。E_JOB>>T_JOB>>L_JOB

2.常用的 ETL 工具有哪些
Datastage、Informatica

3.Datastage 常用的控件有哪些？
COPY STAGE（复制）
Filter Stage（过滤）
Funnel Stage（合并）
Tansformer Stage
Sort Stage（排序）
LookUp Stage(关联)
Join Stage（关联）
Aggregator Stage（聚合）
Remove Duplicates Stage（去重）

4.LOOKUP 和 JOIN 的区别：
lookup 是把从表里的所有记录放到内存中，然后从主表里取数逐一匹配，而 join 是两张表进行排序匹配，可以选择连接方式，lookup 的关联字段名可以不一样，但是 join 无论多少表关联字段名都要一样，如果内存足够的情况下，用 lookup 的性能比较好，但是数据有局限性，数据量太大的话超出内存会报错，所以如果后期的增长量不大则用 lookup，大的话就用 join 比较好。join 有 4 种连接方式：内连接，左外连接，右外连接，全连接。Lookup 只有内连接和左外连接。
LOOKUP：如果从表有重复数据，只随机取一条关联。

5.使用 DS 遇到的一些问题：
1、从别的地方将 SQL 复制到 DS 控件里的时候，如果输入法不一致，中文会变成乱码
2、LOOKUP 控件从表的数据量过大，内存不够的话，会导致报错
3、各个控件中相同字段的字段类型必须一致，否则会报错

--调度工具
自动化、批量
可设置几点跑、先后顺序、出问题发短信、失败重跑、断点续跑、监控图形等。

--建模工具
设计表结构、表与表之间的映射关系（一对多、多对多等），可以把模型直接导成创表语句，可以统一字段（如中文改英文）等。

--配置管理工具
存放需求文档、设计文档、开发 JOB、测试报告等，准备上线。