

PL/SQL 学校课程笔记

ORACLE

徐鸣飞

2023 年 12 月 27 日

目录

第一章 导言	1
1.1 常见数据库	1
1.1.1 关系型数据库	1
1.1.2 NoSQL 数据库	2
1.2 数据库发展趋势	3
1.2.1 1960s-1980s: 层次数据库	3
1.2.2 1980s-2000s: 实体关系数据库	3
1.2.3 2000s-2020s: NoSQL	4
1.2.4 2020s-未知: 图数据库	5
1.3 SQL*Plus 工具	5
1.3.1 介绍	5
1.3.2 交互过程	6
1.3.3 Oracle 数据库连接命令	6
1.3.4 常用编辑命令	7
1.3.5 SQL 文件存取命令	10
1.3.6 输出保存命令	10
第二章 PL/SQL 语法学习	11
2.1 PL/SQL 概述	11
2.1.1 介绍	11
2.1.2 优点	11
2.1.3 执行体系	12
2.2 程序块结构	13
2.3 变量	15

2.3.1	变量命名规则	16
2.3.2	变量声明语法	16
2.3.3	变量类型	16
2.3.4	变量声明建议	18
2.3.5	赋值：字符串分隔符（引号限定符）	18
2.3.6	类型声明：%TYPE 属性	19
2.4	布尔变量与布尔表达式	20
2.4.1	布尔操作符	21
2.5	代替变量和绑定变量	21
2.5.1	代替变量	21
2.5.2	绑定变量	22
2.5.3	不同	23
2.6	LOB 变量	23
2.6.1	LOB 变量类型	23
2.6.2	LOB 变量使用	24

第一章 导言

1.1 常见数据库

422 systems in ranking, September 2023

Rank			DBMS	Database Model	Score		
Sep 2023	Aug 2023	Sep 2022			Sep 2023	Aug 2023	Sep 2022
1.	1.	1.	Oracle +	Relational, Multi-model i	1240.88	-1.22	+2.62
2.	2.	2.	MySQL +	Relational, Multi-model i	1111.49	-18.97	-100.98
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model i	902.22	-18.60	-24.08
4.	4.	4.	PostgreSQL +	Relational, Multi-model i	620.75	+0.37	+0.29
5.	5.	5.	MongoDB +	Document, Multi-model i	439.42	+4.93	-50.21
6.	6.	6.	Redis +	Key-value, Multi-model i	163.68	+0.72	-17.79
7.	7.	7.	Elasticsearch	Search engine, Multi-model i	138.98	-0.94	-12.46
8.	8.	8.	IBM Db2	Relational, Multi-model i	136.72	-2.52	-14.67
9.	↑ 10.	↑ 10.	SQLite +	Relational	129.20	-0.72	-9.62
10.	↓ 9.	↓ 9.	Microsoft Access	Relational	128.56	-1.78	-11.47
11.	11.	↑ 13.	Snowflake +	Relational	120.89	+0.27	+17.39
12.	12.	↓ 11.	Cassandra +	Wide column, Multi-model i	110.06	+2.67	-9.06
13.	13.	↓ 12.	MariaDB +	Relational, Multi-model i	100.45	+1.80	-9.70
14.	14.	14.	Splunk	Search engine	91.40	+2.42	-2.65
15.	↑ 16.	↑ 16.	Microsoft Azure SQL Database	Relational, Multi-model i	82.73	+3.22	-1.69
16.	↓ 15.	↓ 15.	Amazon DynamoDB +	Multi-model i	80.91	-2.64	-6.51

1.1.1 关系型数据库

MySQL

MySQL 是一款开源的关系型数据库管理系统（RDBMS），由瑞典 MySQL AB 公司开发，现由 Oracle 公司维护。以其轻量级、高效、快速的特性而闻名，适用于中小型应用和网站。MySQL 支持多平台运行，包括 Windows、Linux、macOS 等，提供多种存储引擎如 InnoDB、MyISAM 等，具备良好的扩展性和广泛的社区支持。作为开源软件，MySQL 允许用户免费获取、使用和修改其源代码，成为广泛应用于 Web 开发和其他应用场景的可靠数据库解决方案。

Oracle

Oracle 是一款商业性质的关系型数据库管理系统（RDBMS），由 Oracle 公司开发。以其丰富的功能集、高级事务管理、安全性和卓越的性能而著称，适用于大型企业级应用。Oracle 数据库支持复杂查询、分布式数据库、以及高并发环境下的大规模数据处理，具备出色的可伸缩性和性能优化特性。作为商业软件，Oracle 提供了专业的技术支持、认证体系和咨询服务，成为众多大型组织和企业信赖的数据库解决方案。

1.1.2 NoSQL 数据库

文档型数据库：MongoDB

MongoDB 是一种非关系型数据库管理系统（NoSQL DBMS），以其面向文档的存储模型而著称。由 MongoDB 公司开发，采用分布式架构和灵活的模式设计，适用于处理大量非结构化或半结构化的数据。MongoDB 的数据存储形式为 BSON（Binary JSON），支持动态模式，使得数据存储和查询更加灵活。其强大的横向扩展性和自动分片功能使得 MongoDB 适用于大规模数据存储和处理，尤其在 Web 应用、大数据和实时分析等场景中表现出色。由于其开源特性，MongoDB 拥有庞大的社区支持，为开发人员提供了丰富的资源和工具。

搜索引擎：Elasticsearch

Elasticsearch 是一种分布式数据库管理系统，专注于搜索和分析大规模数据。作为开源软件，Elasticsearch 构建在 Apache Lucene 之上，采用文档导向的存储模型，以 JSON 格式存储数据。其核心能力包括实时搜索、结构化查询和复杂分析，使其成为处理实时数据和日志、构建全文搜索引擎以及进行大规模数据分析的理想选择。通过支持分布式架构，Elasticsearch 实现了水平扩展，能够应对高负载和大规模数据存储需求。该系统与 Kibana 等工具的整合形成了 ELK 堆栈，为用户提供了全面的数据管理、搜索和可视化解决方案。

key-value 数据库：redis

Redis 是一款开源、高性能的键值对存储系统，属于 NoSQL 数据库的一种。作为内存数据库，Redis 将数据存储在内存中，提供了快速的读写访问速度，适用于对性能有严格要求的场景。其特色包括支持多种数据结构（如字符串、哈希表、列表、集合等），原子性操作，发布订阅机制等。虽然主要用于缓存、会话管理和实时数据分析等领域，但由于其快速响应和可持久

化存储的能力，也在一些应用中用作主数据库。Redis 的灵活性、简单性和高可用性，使其成为各种实时应用和分布式系统的理想选择。

1.2 数据库发展趋势

1.2.1 1960s-1980s: 层次数据库

20 世纪 60 年代到 80 年代的数据库技术被称为“层次结构”，也可以被称为网状结构，无论标签是什么，这个时代的想法都旨在以树状结构组织数据结构。

换言之，这个时代的数据库技术是将数据存储为相互链接的记录。在层次数据库的模型

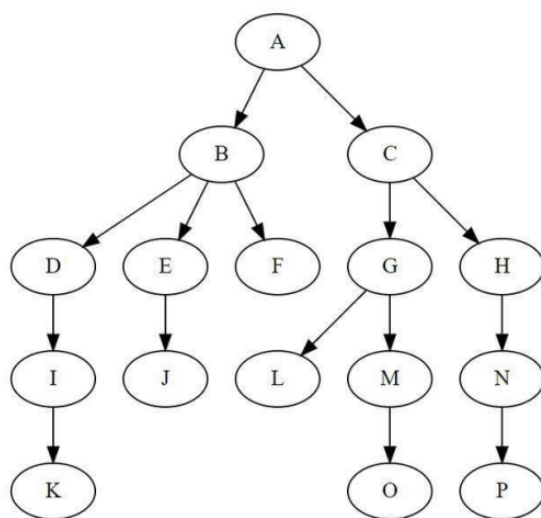


图 1.1: 层次数据库示意图

中，数据以节点的形式存在，每个节点可以包含一个或多个字段，形成父子关系。根节点是顶层节点，而叶节点是没有子节点的底层节点。层次数据库适用于需要表示层次结构信息的场景，如组织结构、文件系统等。尽管在过去曾经流行，但由于关系型数据库的普及，层次数据库在现代数据库系统中的应用相对较少。其特点包括数据重复、路径的唯一性以及专门的查询语言用于数据检索和更新。

1.2.2 1980s-2000s: 实体关系数据库

实体关系数据库（Entity-Relationship Database）是一种基于实体-关系模型的数据库系统，用于存储和管理数据。在这个模型中，数据以实体（Entity）和实体之间的关系（Relationship）为核心。每个实体都有属性，而实体之间的关系描述了这些实体之间的联系和交互。

实体关系数据库的优势在于其规范化的数据结构、ACID 特性（原子性、一致性、隔离性、持久性）以及使用 SQL（Structured Query Language）进行数据操作和查询的能力。这种数据库模型在处理复杂的关联数据和支持事务处理方面表现出色，因此在各种应用场景中广泛应用，包括企业应用、金融系统、医疗信息管理等。

关系型数据库采用了关系代数的概念，数据以表格（表）的形式组织，每个表表示一个实体，表中的行代表实体的具体实例，而列代表实体的属性。实体之间的关系则通过外键来建立。

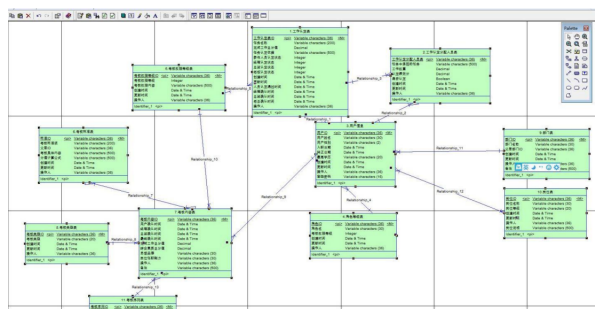


图 1.2: 实体关系数据库示意图

1.2.3 2000s-2020s:NoSQL

NoSQL 为 Not Only SQL 的缩写，是对不同于传统的关系型数据库的数据库管理系统的统称。

网络上的数据本质上不是表格的结构。

NoSQL 常用于超大规模数据的存储（例如谷歌或 Facebook 每天为他们的用户收集万亿比特的数据）。这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

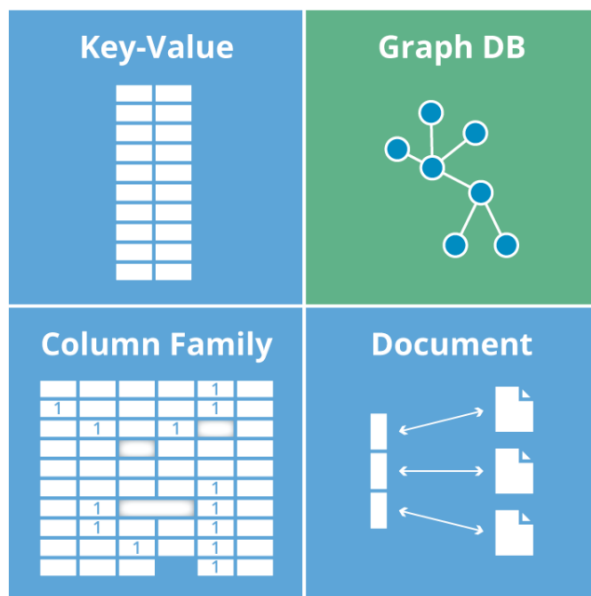


图 1.3: NoSQL 数据库示意图

1.2.4 2020s-未知：图数据库

这个创新时代正在从存储系统的效率转向从存储系统包含的数据中提取价值。

即价值从效率转移到高度连接的数据资产中衍生。

1.3 SQL*Plus 工具

1.3.1 介绍

SQLPlus 是 Oracle 数据库系统中的一种交互式查询工具和脚本处理器。它是一个**命令行工具**，允许用户连接到 Oracle 数据库并执行 SQL 查询、PL/SQL 块以及其他数据库管理任务。

1.3.2 交互过程

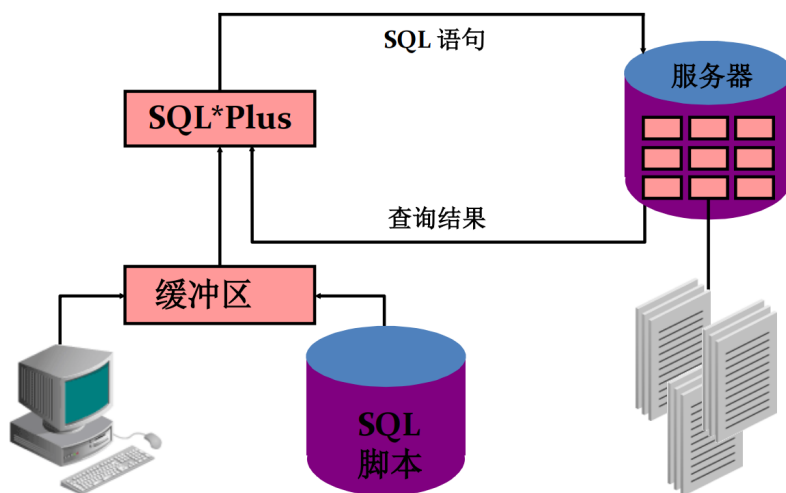


图 1.4: SQL*Plus 交互示意图

参与组件介绍：

用户界面 用户与 SQLPlus 进行交互的地方，可以是命令行界面或其他支持 SQLPlus 的用户界面。用户通过这个界面输入 SQL 查询、PL/SQL 块或 SQL*Plus 命令。

SQL*Plus 引擎 SQLPlus 引擎是一个解释器，负责解释和执行用户输入的 SQLPlus 命令，以及管理与 Oracle 数据库的交互。它解析并执行 SQL 查询、PL/SQL 块，处理输出格式，管理连接和会话等。

OCI (Oracle Call Interface) OCI 是 Oracle 数据库提供的一组 API（应用程序接口），允许应用程序（包括 SQLPlus）与 Oracle 数据库进行通信。**SQLPlus 使用 OCI** 与数据库建立连接、发送 SQL 命令，以及接收执行结果。

Oracle 数据库引擎 这是实际执行 SQL 查询、PL/SQL 块的地方。当 SQLPlus 将命令发送给数据库时，Oracle 数据库引擎负责解析和执行这些命令，然后返回结果给 SQLPlus。

数据缓冲区 SQL*Plus 通常会在本地维护一个数据缓冲区，用于存储从数据库检索到的数据。这使得用户可以在本地对结果进行分析、浏览和编辑。

1.3.3 Oracle 数据库连接命令

在 CMD 中运行 SQLPlus 并连接到 Oracle 服务器的命令为：

sqlplus <用户名> /<密码> @//<数据库 IP>:<Port> /<服务名>

如：

例 1.1. *sqlplus system/system@//192.168.146.132:1521/helowin*

1.3.4 常用编辑命令

DESC DESCRIBE，显示表结构，包括表的列名、数据类型和约束信息，效果见图1.5。

L 列出当前缓冲区中的 SQL 语句（未执行）的命令。

[N] number，选中第 n 行；[n] 后可追加内容 [text] 来替换第 n 行或新建第 n 行。

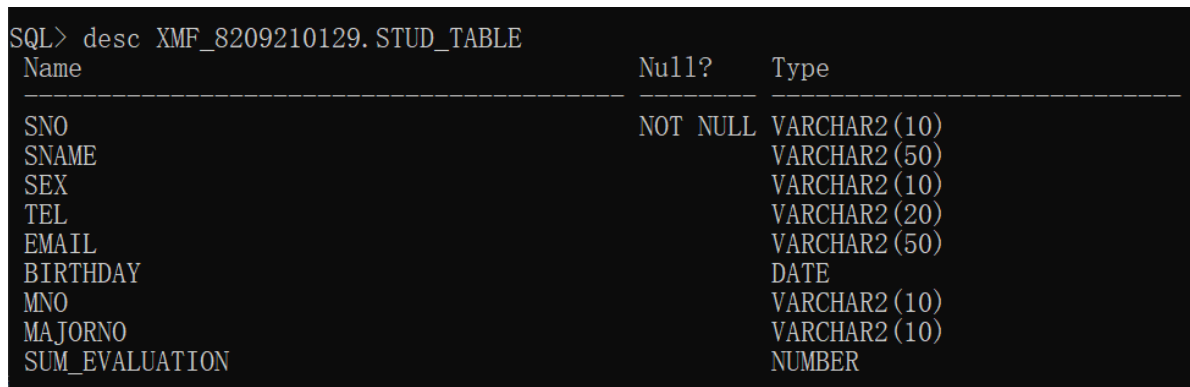
A APPEND，追加文本到缓冲区中的当前选中行。使用 “.” 单独一行表示结束追加。使用前要求缓冲区内存在文本。

C CHANGE，修改缓冲区中的当前选中行的文本，格式为 **C** /< 原文 >/< 新文 >。

DEL DELETE，删除缓冲区中的当前选中行。可追加 [n] 代表删除第 n 行；追加 [n][m] 删除第 n 至 m 行。

/ 运行在 SQL 缓冲区中的 SQL 语句。

CL CLEAR，清除 SQL*Plus 屏幕上的内容，将屏幕滚动条移至顶部。



```
SQL> desc XMF_8209210129.STUD_TABLE
```

Name	Null?	Type
SNO	NOT NULL	VARCHAR2(10)
SNAME		VARCHAR2(50)
SEX		VARCHAR2(10)
TEL		VARCHAR2(20)
EMAIL		VARCHAR2(50)
BIRTHDAY		DATE
MNO		VARCHAR2(10)
MAJORNO		VARCHAR2(10)
SUM_EVALUATION		NUMBER

图 1.5: DESC 效果图

使用编辑缓冲区的命令前，需保证缓存区中存在内容：

```
SQL> SELECT *  
  2 FROM XMF_820921029  
  3 WHERE SNO  
  4 .
```

图 1.6: 输入内容到缓冲区，使用 `enter` 换行，使用单独的. 结束

可使用 `L` 查看缓冲区：

```
SQL> L  
  1 SELECT *  
  2 FROM XMF_820921029  
  3* WHERE SNO
```

图 1.7: `L` 命令效果图

* 号所在行为正在编辑行。可使用数字更改选中行后，使用 `A` 命令添加内容：

```
SQL> L  
  1 SELECT *  
  2 FROM XMF_820921029  
  3* WHERE SNO  
SQL> 2  
  2* FROM XMF_820921029  
SQL> A . STUD_TABLE  
  2* FROM XMF_820921029. STUD_TABLE  
SQL> L  
  1 SELECT *  
  2 FROM XMF_820921029. STUD_TABLE  
  3* WHERE SNO
```

图 1.8: `A` 命令、`n` 命令效果图

可使用 `C` 命令更改指定行（注意原文与新文中无空格）：

```
SQL> L
  1 SELECT *
  2 FROM XMF_820921029.STUD_TABLE
  3* WHERE SNO
SQL> C /SNO /SEX=男
SP2-0023: String not found.
SQL> C /SNO/SEX=男
  3* WHERE SEX=男
SQL> L
  1 SELECT *
  2 FROM XMF_820921029.STUD_TABLE
  3* WHERE SEX=男
```

图 1.9: C 命令效果图

可使用 DEL 命令删除指定行：

```
SQL> L
  1 SELECT *
  2 FROM XMF_820921029.STUD_TABLE
  3* WHERE SEX=男
SQL> DEL 3
SQL> L
  1 SELECT *
  2* FROM XMF_820921029.STUD_TABLE
SQL> _
```

图 1.10: DEL 命令效果图

再次使用 n 命令新建行：

```
SQL> L
  1 SELECT *
  2* FROM XMF_82092029.STUD_TABLE
SQL> 3 WHERE SNO=982
SQL> L
  1 SELECT *
  2 FROM XMF_82092029.STUD_TABLE
  3* WHERE SNO=982
SQL> _
```

图 1.11: N 命令效果图

使用 / 运行（使用后缓冲区不会被清除）：

```
SQL> L
1 SELECT *
2 FROM XMF_8209210129.STUD_TABLE
3* WHERE SNO=982
SQL> /
SNO
-----
SNAME
-----
SEX
-----
TEL
-----
EMAIL
-----
BIRTHDAY      MNO      MAJORNO
-----
SUM_EVALUATION
-----
982
```

图 1.12: 运行命令效果图

1.3.5 SQL 文件存取命令

SAVE 把 SQL 缓冲区的内容存入指定的文件，如 *SAVE D:\SQL\SMAPLE*。

GET 将指定的脚本文件装入 SQL 缓存区，如 *GET D:\SQL\SMAPLE*。

START @ 或 **START**，把指定的脚本文件装入 SQL 缓冲区并运行，如 *@ D:\SQL\SMAPLE.sql*。

1.3.6 输出保存命令

SPOOL，语法为：

SPO[OL] [File_name[.ext]] [[CRE[ATE]]|REP[LACE]]|APP[END]] | OFF | OUT]。

参数说明：

File_name 指定脱机文件的名称，默认的文件扩展名为 LST。

CRE[ATE] 表示创建一个新的脱机文件，这也是 **SPOOL** 命令的默认状态。

REP[LACE] 表示替代已经存在的脱机文件。

APP[END] 表示把脱机内容附加到一个已经存在的脱机文件中。

OFF | OUT 表示关闭 **SPOOL** 输出。

第二章 PL/SQL 语法学习

2.1 PL/SQL 概述

2.1.1 介绍

PL/SQL (Procedural Language/Structured Query Language): 一种过程化编程语言，专门用于 **Oracle** 数据库系统。它将 SQL (Structured Query Language) 语句与结构化程序设计语言（如条件、循环等）相结合，提供了一种强大的方式来处理和操作数据库中的数据。

2.1.2 优点

1. **数据库集成**: PL/SQL 是为数据库设计的，能够直接嵌套在 SQL 中；许多与数据库有关的应用程序功能都已经集成在 PL/SQL 语言中。
2. **模块化开发**: PL/SQL 允许将代码模块化组织，通过存储过程和包的方式来管理和封装代码。这有助于提高代码的可维护性和重用性。
3. **性能优化**: PL/SQL 支持存储过程和函数，可以在数据库中预编译和存储，提高了执行效率。此外，PL/SQL 还支持游标，能够有效地处理大量的数据。

此外，还有其他如**安全性**（PL/SQL 能够帮助限制对数据库的直接访问，从而提高了数据库的安全性）、**灵活性**（PL/SQL 具有丰富的控制结构和数据类型，使其既可以用于简单的 SQL 查询，也可以用于复杂的业务逻辑实现）、**事务控制**（PL/SQL 提供了强大的事务控制功能，支持原子性、一致性、隔离性和持久性（ACID 属性），确保了数据库操作的可靠性和一致性）等优点。

2.1.3 执行体系

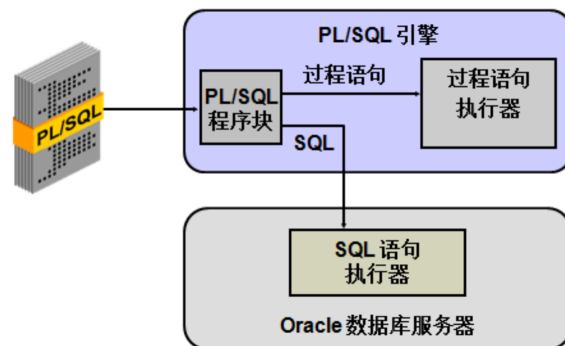


图 2.1: PLSQL 执行体系示意图

从图2.1中这些组件的角度来看 PL/SQL 的执行过程：

PL/SQL 引擎

- **PL/SQL 解释器：**编译后的 PL/SQL 代码由 PL/SQL 引擎的解释器执行。解释器负责逐行执行 PL/SQL 代码，并管理变量、控制流程、处理异常等。
- **数据交互：**在 PL/SQL 执行过程中，可能涉及到与数据库的交互，例如执行 SQL 语句、操作数据等。这些交互通过 PL/SQL 引擎协调实现。

过程语句执行器

- **过程语句解析：**如果 PL/SQL 程序包含了过程性语句，这些语句将被 PL/SQL 引擎的过程语句执行器处理。这可能包括存储过程、函数或触发器的调用。
- **编译过程性代码：**过程语句执行器会编译 PL/SQL 代码，将其转换为可执行的形式。这个编译过程包括语法检查、编译成中间代码等步骤。

SQL 执行器

- **接收 SQL 语句：**当一个包含 PL/SQL 代码的程序被触发时，其中可能包含 SQL 语句。这些 SQL 语句被传递给 SQL 执行器，它负责执行这些语句。
- **解析 SQL 语句：**SQL 执行器解析 SQL 语句，进行语法分析和语义检查。在这个阶段，执行计划被生成，这是一个描述如何执行查询的内部表示。

Oracle 数据库服务器

- **数据存储和检索**：当 PL/SQL 程序执行 SQL 语句时，Oracle 数据库服务器负责实际的数据存储和检索。执行计划告诉数据库服务器如何最有效地执行查询，并从表中检索或修改数据。

当有过程调用或 SQL 语句执行时，流程可能涉及多次从 PL/SQL 引擎到 SQL 执行器，再到数据库服务器的交互。

需要注意的是，Oracle 数据库的内部执行细节是复杂的，并且可能受到优化、缓存、并发控制等多方面的影响。

2.2 程序块结构

PL/SQL 程序块是 PL/SQL 语言中的基本结构，它可以包含变量、常量、游标、异常处理等元素。PL/SQL 程序块有三种主要形式：匿名块、存储过程和存储函数。类型说明：

1. **匿名块 (Anonymous blocks)**：没有命名的程序块。匿名块是在应用程序内部需要的地方声明的，在这个应用程序每次执行时，这些匿名块都会被编译并执行。
2. **过程 (Procedures) 和函数 (Functions)**：过程和函数也统称为子程序 (*Subprograms*)，子程序是对匿名块的补充，子程序就是被命名的 PL/SQL 程序块，而它们可以存储在数据库中。

一个简单的 PL/SQL 匿名块的结构如下：

Listing 2.1: PL/SQL 匿名块示例代码

```
1 DECLARE
2     -- 声明变量和常量
3     variable_name datatype;
4     constant_name CONSTANT datatype := value;
5
6 BEGIN
7     -- 可执行的PL/SQL代码
8     -- 可以包括各种语句，如赋值、条件语句、循环等
9     -- 例如：
10    variable_name := value;
11    IF condition THEN
12    -- do something
```



```
13  END IF;
14
15  EXCEPTION
16      -- 异常处理部分
17      -- 处理可能出现的异常
18      -- 例如:
19      WHEN others THEN
20          -- handle exception
21
22  END;
23  /
```

一个简单的 PL/SQL 函数的示例代码（获取员工薪水）：

Listing 2.2: 获取员工薪水函数示例代码

```
1  CREATE OR REPLACE FUNCTION get_employee_salary(p_employee_id NUMBER)
2      RETURN NUMBER
3      IS
4      -- 声明变量
5      v_salary NUMBER;
6  BEGIN
7      -- 查询员工薪水
8      SELECT salary INTO v_salary
9      FROM employees
10     WHERE employee_id = p_employee_id;
11
12     -- 返回薪水
13     RETURN v_salary;
14 END;
15 /
```

一个简单的 PL/SQL 过程的示例代码（更新员工薪水）：

Listing 2.3: 更新员工薪水过程示例代码

```
1 CREATE OR REPLACE PROCEDURE update_employee_salary(p_employee_id NUMBER,  
    p_new_salary NUMBER)  
2 IS  
3 BEGIN  
4     -- 更新员工薪水  
5     UPDATE employees  
6     SET salary = p_new_salary  
7     WHERE employee_id = p_employee_id;  
8  
9     -- 提交事务  
10    COMMIT;  
11 END;  
12 /
```

程序块结构说明：

DECLARE 可选，声明段，以关键字 DECLARE 开始并以执行段的开始而结束。

BEGIN 必选，执行段，以关键字 BEGIN 开始而以关键字 END 或关键字 EXCEPTION 结束。

EXCEPTION 可选，异常处理段，以关键字 EXCEPTION 开始，以关键字 END 结束。

执行规则：

1. 每一个 PL/SQL 控制语句都是以分号 (;) 结束。
2. 使用正斜杠 (/) 运行 SQL*Plus 内存缓冲区的匿名 PL/SQL 程序块。

2.3 变量

变量，就是内存中一个命名的临时存储区，而变量中所存储的信息就是这个变量的当前值。

在 PL/SQL 中，在使用一个变量之前，必须首先声明这个变量。一旦声明了这个变量，就可以在 SQL 语句和过程化 (PL/SQL) 语句中使用这个变量了。

变量使用步骤：

1. 在声明部分（DECLARE 块）**声明和初始化变量**
2. 在执行部分（BEGIN 块）**为变量赋新值**
3. 通过参数将值传入 PL/SQL 块
4. 通过输出变量来查看结构

2.3.1 变量命名规则

- 必须以**英语字母**开始
- 可以包含**英语字母**或**数字**或**特殊字符**——美元号（\$）、下划线（_）、和井号（#）
- 长度最长为 **30** 个字符
- **不区分大小写**
- 不能是保留关键字
- 不能与数据库的表或列同名

2.3.2 变量声明语法

在引用 PL/SQL 程序块中的变量之前，必须在声明段声明所有的变量（标识符）。在声明变量的同时还可以为变量赋初值，但是在声明变量时赋予初始值是可选的。如果在一个变量声明中引用了其他变量，一定要确保在之前的语句中已经声明了引用的变量。

声明语法：**标识符 [CONSTANT] 数据类型 [NOT NULL] [:= initial_value]**

Listing 2.4: PL/SQL 变量声明语法示例

```
1 DECLARE
2   v_dogid NUMBER(10) NOT NULL := 38;
3   v_name  VARCHAR2(25) := 'White Tiger'
4   c_color CONSTANT VARCHAR2(15) := 'White'
5   v_birthday DATE;
```

2.3.3 变量类型

PL/SQL 支持四大类数据类型（最常用的是标量数据类型，目前熟悉这个即可）：

1. 标量 (Scalar) 数据类型:

- NUMBER: 用于存储数值, 可以是整数或小数。
 - NUMBER(p,s): 数据型数据, p 表示精度 (总位数), s 表示小数部分的位数
 - BINARY_INTEGER: 基本整型数
 - PLS_INTEGER: 基本带符号整数型
 - BINARY_FLOAT: 存储单精度浮点数, 以 IEEE754 格式表示浮点数, 它需要 5 个字节来存储数字
 - BINARY_DOUBLE: 存储双精度浮点数, 以 IEEE754 格式表示浮点数, 它需要 9 个字节来存储数字
- character: 字符型数据类型, 用于存储字符串。
 - VARCHAR2(size): 基本变长字符型数据
 - CHAR(size): 基本定长字符型数据
- DATE: 用于存储日期和时间。
 - DATA: 基本日期和时间数据
 - TIMESATMP(precision): 该数据类型除了日期和时间之外还包括了多达小数点后 9 位秒数
- BOOLEAN: 用于存储布尔值 (TRUE 或 FALSE)。
 - BOOLEAN: 基本逻辑类型, 它只能存储逻辑计算的 3 个可能值之一

2. 组合 (Composite) 数据类型:

- %ROWTYPE: 表示一个表行的结构, 通常在变量声明中使用。
- RECORD: 类似于%ROWTYPE, 但可以自定义结构。
- TABLE: 用于存储同一数据类型的集合。可以是索引表 (INDEX BY) 或关联数组。

3. 引用 (Reference) 数据类型:

- REF CURSOR: 用于存储游标引用, 可用于动态查询结果的处理。
- PL/SQL RECORD: 通过 TYPE 语句定义的记录类型。

4. 大对象 (LOB) 数据类型:

- CLOB (Character Large Object): 用于存储大量字符数据，如文本。
- BLOB (Binary Large Object): 用于存储二进制数据，如图像、音频等。
- NCLOB (National Character Large Object): 类似于 CLOB，但用于存储国家字符集数据。
- BFILE: 用于存储二进制文件的地址引用，文件存储在数据库外部。

2.3.4 变量声明建议

1. 遵守命名的规则，变量的命名规则与 SQL 对象的命名规则完全相同。
2. 必须初始化被指定为非空 (NOT NULL) 和常量 (Constant) 的变量。
3. 一行最好只声明一个标识符以提高代码的易读性和方便代码的维护。
4. 通过使用赋值操作符 (:=) 或默认关键字 (DEFAULT) 来初始化标识符。
5. 变量最好不要与列名重名。
6. 两个 PL/SQL 变量 (对象) 只要在不同的程序块中是可以同名的。
7. 如果没有必要就不要在 PL/SQL 变量上强加非空 (NOT NULL) 约束。
8. 代码中使用了 PL/SQL 游标、BLOB/CLOB Locator 或其他需要手动释放的资源，请确保在不再需要它们时及时释放这些资源，以避免内存泄漏。

2.3.5 赋值：字符串分隔符（引号限定符）

介绍

q 和 p 是引号限定符，用于创建带有引号的字符串字面量。具体而言，q 和 p 用于创建带有引号的标识符或字符串，这样可以在字符串中包含特殊字符，而不必使用转义字符。q 用于创建带有双引号的标识符，而 p 用于创建带有单引号的字符串。

(注意：p 定界符可以使用大多数字符作为定界符，但不能使用双引号”，单引号’，反引号`，右括号)，右中括号]，右大括号 } 和感叹号! 作为定界符。)

演示

不使用字符串分隔符的情况：

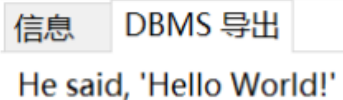
Listing 2.5: 不使用字符串分隔符

```
1 DECLARE
2     v_message VARCHAR2(100);
3 BEGIN
4     -- 使用双单引号进行转义
5     v_message := 'He said, 'Hello World!'';
6
7     -- 打印字符串
8     DBMS_OUTPUT.PUT_LINE(v_message);
9 END;
```

使用字符串分隔符后：

Listing 2.6: 使用字符串分隔符

```
1 DECLARE
2     v_message VARCHAR2(100);
3 BEGIN
4     -- 使用 q'...' 创建原始字符串
5     v_message := q'{He said, 'Hello World!'}';
6
7     -- 打印字符串
8     DBMS_OUTPUT.PUT_LINE(v_message);
9 END;
```



信息 DBMS 导出

He said, 'Hello World!'

图 2.2: 字符串分隔符输出演示

2.3.6 类型声明：%TYPE 属性

为了避免这种变量数据类型和精度的硬编码（即数据类型和精度必须显式定义），PL/SQL 引入%TYPE 属性。程序员（开发人员）可以使用%TYPE 属性按照之前已经声明过的变量或数据库的列来声明一个变量。当存储在一个变量中值来自于数据库中的表时，使用%TYPE 属性

来声明这个变量时再适合不过的了。

使用语法

1. 标识符表名. 列名%TYPE
2. 标识符定义好的变量%TYPE

案例

```
1 DECLARE
2     v_data emp.data%TYPE
3     v_Month_Value NUMBER(6,4) := 3.25
4     v_Year_Value v_Month_Value%TYPE := 4.40
```

好处：

1. 可以不知道 emp 表中的 data 列或 v_Month_Value 的数据类型和精度就直接使用对应类型。
2. 当 emp 的 data 列或者 v_Month_Value 的数据类型和精度发生变化时，v_data 和 v_Year_Value 也会随之变化。

2.4 布尔变量与布尔表达式

布尔变量和布尔表达式在任何程序语言设计上都是非常重要和广泛使用的。在 PL/SQL 程序中，可以在 SQL 语句中也可以在过程化语句中进行变量的比较，这样的比较表达式被称为布尔表达式，它们是由单个表达式或由关系操作符所分隔的复杂表达式所组成。

特性：

- 只有值 **TURE**、**FALSE** 和 **NULL** 可以赋给一个布尔变量。
- 可以通过**逻辑操作符 AND、OR 和 NOT** 对布尔变量进行运算，运算总是产生 TURE、FALSE 或 NULL。
- 数字、字符和日期表达式可以被用来返回一个布尔值。

2.4.1 布尔操作符

操作符	描述
=	等于
>	大于
>=	大于等于
<	小于
<=	小于等于
<>	不等于
BETWEEN...AND...	二个值之间（包括这二个值）
IN(set)	匹配值列表中的任意值
LIKE	匹配某一字符模式
IS NULL	是空值

图 2.3: 布尔操作符

2.5 代替变量和绑定变量

2.5.1 代替变量

因为 PL/SQL 本身没有输入和输出功能，所以必须依赖于执行 PL/SQL 程序的环境变量值传入或传出 PL/SQL 程序块。

在 SQL*Plus 环境中，可以使用 **SQL*Plus 的代替变量**将运行时的值传给 PL/SQL 程序块。在 PL/SQL 程序块中可以使用前导的 & 符号引用代替变量，就像在 SQL 语句中引用 SQL*Plus 的代替变量一样。在 PL/SQL 程序执行前，正文的值被代替进 PL/SQL 程序块中。

例如，一个使用代替变量的查询可以如下所示：

```
1 Select *
2 FROM XMF_8209210129.STUD_TABLE
3 WHERE SNO=&ReplaceVariable1;
```

运行效果：


```
SQL> DEFINE ReplaceVariable1 = 982
SQL> Select * from XMF_8209210129.STUD_TABLE where SNO=&ReplaceVariable1;
old 1: Select * from XMF_8209210129.STUD_TABLE where SNO=&ReplaceVariable1
new 1: Select * from XMF_8209210129.STUD_TABLE where SNO=982

SNO
-----
SNAME
-----
```

图 2.4: 代替变量案例运行结果

在这里，ReplaceVariable1 是一个代替变量，SQL*Plus 会在执行查询之前提示用户输入 ReplaceVariable1 的值，并将其替换到 SQL 语句中。DEFINE ReplaceVariable1 = 982 用于定义 SQLPlus 变量。

2.5.2 绑定变量

绑定变量是在**使用（或调用）PL/SQL 的环境**中创建的，而不是在 PL/SQL 程序的声明段中定义的。在一个 PL/SQL 程序块中声明的所有变量只在执行这个程序块时可以使用。而在这个程序块执行后，这些变量所使用的内存就释放了。然而，绑定变量则不同，在程序块执行后，绑定变量依然存在并允许访问。

绑定变量是在 SQL 语句中使用占位符（通常是冒号: 后跟变量名）来代替实际的数值或表达式。在 PL/SQL 中，通过使用绑定变量，可以减少 SQL 语句的解析次数，提高性能。绑定变量在 SQL 语句执行时被动态绑定，而不是在每次执行时重新解析整个 SQL 语句。

运行演示：

```
SQL> variable v_sno number;
SQL> exec :v_sno := 982;

PL/SQL procedure successfully completed.

SQL> select *
  2  FROM XMF_8209210129.STUD_TABLE
  3  WHERE SNO=:v_sno;

SNO
-----
```

图 2.5: 绑定变量使用案例

variable v_sno number; 定义绑定变量 v_sno

exec :v_sno := 982 给绑定变量赋值为 982

2.5.3 不同

1. 绑定变量 (Bind Variables)

- 绑定变量是用于在 SQL 语句中传递数值或数据的一种机制，通过绑定变量，可以将变量的值绑定到 SQL 语句中，而不是直接在 SQL 语句中使用硬编码的值。
- 这有助于提高性能，因为数据库可以缓存已编译的 SQL 语句，并在执行时仅替换绑定变量的值，而不需要重新编译整个 SQL 语句。

2. 代替变量 (Substitution Variables)

- 代替变量是一种 SQLPlus 工具的功能，它允许在 SQLPlus 中使用变量，这些变量在脚本运行之前由用户手动输入，或者可以通过 DEFINE 命令定义。
- 代替变量在 SQL*Plus 脚本中被替换为相应的值，类似于宏替换。

2.6 LOB 变量

大对象 LOB 是 large object 的缩写，就意味着存储大量的数据，在数据库中，表中的列定义为 LOB 类型（如 CLOB 和 BLOB）。利用 LOB 数据类型，可以在数据库中存储大量的无结构数据块（如正文、图形、声音和影像信息），其存储量可多达 128T（数据量的多少取决于数据块大小）。

2.6.1 LOB 变量类型

CLOB 数据类型 (Character Large Object)

CLOB 用于在数据库中存储字节流类型的大数据对象，如演讲稿、说明书或简历等。

BLOB 数据类型 (Binary Large Object)

用于在数据库中存储大的二进制对象，如照片或幻灯片等。当从数据库中提取这样的数据或向数据库中插入这样的数据时，数据库并不解释这些数据。使用这些数据的外部应用程序必须自己解释这些数据。

BFILE 数据类型 (Binary File)

用于在数据库外的操作系统文件中存储大的二进制对象，如电影胶片等。与其他的 LOB 数据类型不同，BFILE 数据类型是外部数据类型。BFILE 类型是存储在数据库之外的，它们可能是操作系统文件。

NCLOB 数据类型 (National Language Character Large Object)

用于在数据库中存储 NCHAR 类型的单字节或定长多字节的 Unicode 大数据对象。

2.6.2 LOB 变量使用

Listing 2.7: LOB 变量使用示例代码

```
1 DECLARE
2   clob_data CLOB;
3   buffer VARCHAR2(32767);
4   amount NUMBER;
5   offset NUMBER := 1;
6 BEGIN
7   -- 初始化 CLOB 数据
8   clob_data := '这是一个CLOB示例。';
9   -- 读取 CLOB 数据
10  amount := DBMS_LOB.GETLENGTH(clob_data);
11  DBMS_LOB.READ(clob_data, amount, offset, buffer);
12  DBMS_OUTPUT.PUT_LINE(buffer);
13 END;
```

运行结果：

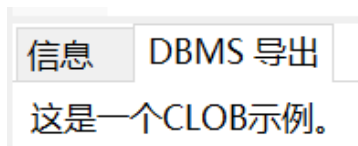


图 2.6: CLOB 示例代码运行结果