第3章

SQL 语言基础

本章将主要介绍 SQL 语言的基础知识。SQL 的全称是结构化查询语言（Structure Query Language），是数据库操作的国际标准语言，也是所有的数据库产品均要支持的语言。因此，要操作数据库一定要掌握好 SQL。

本章通过相关示例，介绍了 SQL 语言的各种知识和语法规范，使读者对 SQL 语言能够全面的掌握。本章的相关示例均来源于 Oracle 附带的示例方案 HR 模式。

3.1 SQL 概述

SQL 语言是在 1974 年由美国 IBM 公司的 San Jose 研究所中的科研人员 Boyce 和 Chamberlin 提出的，然后于 1975～1979 年在关系数据库的管理系统原型 System R 上实现了这种语言。1986 年 10 月，美国国家标准局（American National Standards Institute，ANSI）的数据库委员会批准了 SQL 作为关系数据库语言的美国标准。同年，公布了 SQL 标准文本 SQL_86。1987 年国际标准化组织（International Standards Organization，ISO）将其采纳为国际标准。1989 年公布了 SQL_89，1992 年又公布了 SQL_92（也称为 SQL2）。1999 年颁布了反映最新数据库理论和技术的标准 SQL_99（也称为 SQL3）。

由于 SQL 语言具有功能丰富、简洁易学、使用方式灵活等突出优点，因此倍受计算机工业界和计算机用户的欢迎。尤其自 SQL 成为国际标准后，各数据库管理系统厂商纷纷推出各自的支持 SQL 的软件或与 SQL 接口的软件。这就使得大多数数据库均采用了 SQL 作为共同的数据存取语言和标准接口。但是，不同的数据库管理系统厂商开发的 SQL 并不完全相同。这些不同类型的 SQL 一方面遵循了标准 SQL 语言规定的基本操作，另一方面又在标准 SQL 语言的基础上进行了扩展，增强了一些功能。不同的 SQL 类型有不同的名称。例如，Oracle 产品中的 SQL 称为 PL/SQL，Microsoft SQL Server 产品中的 SQL 称为 Transact-SQL。

3.1.1 SQL 语言的功能

SQL 的主要功能可以分为如下 4 类：

1. 数据定义功能

SQL 的数据定义功能通过数据定义语言（Data Definition Language，DDL）实现，它用来定义数据库的逻辑结构，包括定义基表、视图和索引。基本的 DDL 包括三类，即定义、修改和删除，分别对应 CREATE、ALTER 和 DROP 三条语句。

2. 数据查询功能

SQL 的数据查询功能通过数据查询语言（Data Query Language，DQL）实现，它用来对数据库中的各种数据对象进行查询，虽然仅包括查询一种操作（SELECT 语句），但在 SQL 语言中，它是使用频率最高的语句。查询语句可以由许多子句组成，使用不同的子句便可以进行查询、统计、分组、排序等操作，从而实现选择、投影和连接等运算功能，以获得用户所需的数据信息。

3. 数据操纵功能

SQL 的数据操纵功能通过数据操纵语言（Data Manipulation Language，DML）实现，它用于改变数据库中的数据，数据更新包括插入、删除和修改三种操作，分别对应 INSERT、DELETE 和 UPDATE 三条语句。

4. 数据控制功能

数据库的控制指数据库的安全性和完整性控制。

SQL 的数据控制功能通过数据控制语言（Data Control Language，DCL）实现，它包括对基表和视图的授权、完整性规则的描述以及事务开始和结束等控制语句。

SQL 通过对数据库用户的授权和取消授权命令来实现相关数据的存取控制，以保证数据库的安全性。另外还提供了数据完整性约束条件的定义和检查机制，来保证数据库的完整性。

数据控制功能对应的语句有 GRANT、REVOKE、COMMIT 和 ROLLBACK 等，分别代表了授权、回收、提交和回滚等操作。

3.1.2 SQL 的特点

SQL 语言的主要特点如下：

1. 功能强大

SQL 语言集数据查询、数据操纵、数据定义和数据控制功能于一体，且具有统一的语言风格，使用 SQL 语句就可以独立完成数据管理的核心操作。

2. 集合操作

SQL 采用集合操作方式，对数据的处理是成组进行的，而不是一条一条处理的。通过使用集合操作方式，可以加快数据的处理速度。执行 SQL 语句时，每次只能发送并处理一条语句。若要降低语句发送和处理次数，则可以使用 PL/SQL。

3. 非过程化

SQL 还具有高度的非过程化特点，执行 SQL 语句时，用户只需要知道其逻辑含义，而不需要知道 SQL 语句的具体执行步骤。这使得在对数据库进行存取操作时无需了解

存取路径，大大减轻了用户的负担，并且有利于提高数据的独立性。

4. 语言简洁

虽然 SQL 的语言功能极强，但其语言十分简洁，仅用 9 个动词就完成了核心功能。SQL 的命令动词及其功能如表 3.1 所示。

表 3.1　SQL 的命令动词

SQL 的功能	命令动词
数据定义	CREATE，DROP，ALTER
数据操纵	SELECT，INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

5. 具有交互式和嵌入式两种形式

交互式 SQL 能够独立地用于联机交互，直接键入 SQL 命令就可以对数据库进行操作。而嵌入式 SQL 能够嵌入到高级语言（如 C、FORTRAN、Pascal）程序中，以实现对数据库的存取操作。

无论哪种使用方式，SQL 语言的语法结构基本一致。这种统一的语法结构的特点，为使用 SQL 提供了极大的灵活性和方便性。

6. 支持三级模式结构

SQL 支持关系数据库的三级模式结构，如图 3.1 所示。

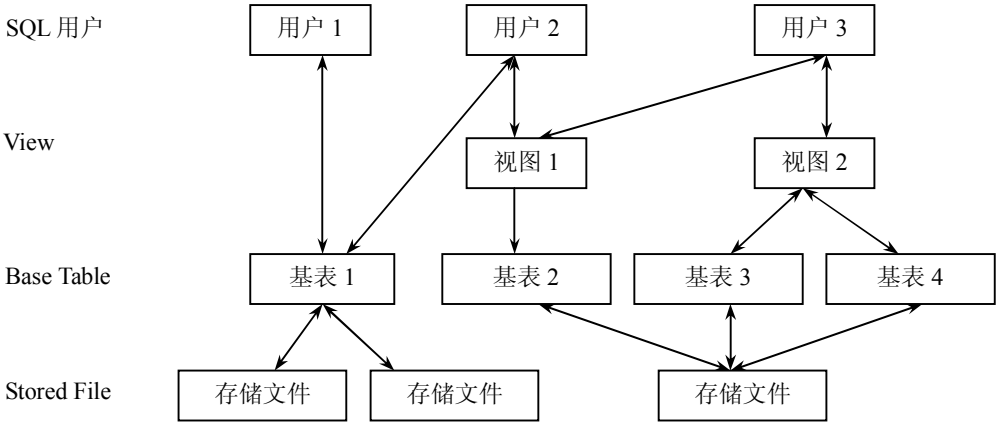

图 3.1　SQL 对关系数据库模式的支持

（1）全体基表构成了数据库的模式

基表（Base Table）是本身独立存在的表，在 SQL 中一个关系就对应一个基表。

（2）视图和部分基表构成了数据库的外模式

视图（View）是从基表或其他视图中导出的表，它本身不独立存储在数据库中，即数据库中只存放视图的定义而不存放视图对应的数据，这些数据仍存放在导出视图的基表中，因此视图是一个虚表。

用户可以用 SQL 语句对视图和基表进行查询等操作。在用户看来，视图和基表是一样的，都是关系。视图是根据用户的需求设计的，这些视图再加上某些被用户直接使

用的基表就构成了关系数据库的外模式。SQL 支持关系数据库的外模式结构。

（3）数据库的存储文件及其索引文件构成了关系数据库的内模式

在 SQL 中，一个关系对应一个表，一个或多个基表对应一个存储文件，一个基表也可以对应多个存储文件，一个表可以带若干索引，索引也存放在存储文件中。每个存储文件与外部存储器上一个物理文件对应。存储文件的逻辑结构组成了关系数据库的内模式。

3.1.3 SQL 语句的编写规则

SQL 关键字不区分大小写，既可以使用大写格式，也可以使用小写格式，或者混用大小写格式。例如：

语句一：

```
SELECT employee_name,salary,job,deptno FROM employee;
```

语句二：

```
select   employee_name,salary,job,deptno from employee;
```

以上两个 SQL 语句是没有区别的。

对象名和列名也不区分大小写，它们既可以使用大写格式，也可以使用小写格式，或者混用大小写格式，例如：

语句一：

```
SELECT employee_name,salary,job,deptno FROM employee;
```

语句二：

```
SELECT employee_name,SALARY,JOB,deptno from employee;
```

以上两个 SQL 语句也没有区别。

字符值和日期值区分大小写。当在 SQL 语句中引用字符值和日期值时，必须要给出正确的大小写数据，否则不能得到正确的查询结果，例如：

语句一：

```
SELECT employee_name,salary,job,deptno FROM employee where employee_name ='SCOTT';
```

语句二：

```
SELECT employee_name,salary,job,deptno FROM employee where employee_name ='scott';
```

以上两个 SQL 语句的执行结果是不一样的，因为在 WHERE 子句中'SCOTT'和'scott'是不一样的两个名称。

在应用程序中编写 SQL 语句时，如果 SQL 语句的文本很短，可以将语句文本放在一行上；如果 SQL 语句的文本很长，可以将语句文本分布到多行上，并且可以通过使用跳格和缩进提高可读性。另外，在 SQL*Plus 中的 SQL 语句要以分号结束。

例如，单行语句文本的书写如下所示：

```
SELECT employ_name,salary FROM employee;
```

多行语句文本的书写如下所示：

```
SELECT a.dept_name, b.employ_name, b.salary, b.job
FROM department a RIGHT JOIN employee b
ON a.dept_no = b.dept_no AND a.dept_no='01';
```

3.2 数据定义

SQL 的数据定义功能是针对数据库三级模式结构所对应的各种数据对象进行定义的，在标准 SQL 语言中，这些数据对象主要包括表、视图和索引。当然，在 Oracle 数据库中，还有各种其他的数据对象，如触发器、游标、过程、程序包等。本节仅以表、视图和索引对数据定义语言进行说明。

SQL 的数据定义语句如表 3.2 所示。

表 3.2 SQL 的数据定义语句

操作对象	操作方式		
	创建	删除	修改
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

注意：在标准的 SQL 语言中，由于视图是基于表的虚表，索引是依附在基表上的，因此视图和索引均不提供修改视图和索引定义的操作。用户若想修改，则只能通过删除再创建的方法。但在 Oracle 中可以通过 ALTER VIEW 对视图进行修改，详细内容见第 6 章。

3.2.1 CREATE

在数据库中，对所有数据对象的创建均由 CREATE 语句来完成，本节仅对使用 CREATE 语句创建表、视图和索引进行描述。

1. 创建表

建立数据库最重要的一项内容就是定义基表。SQL 语言使用 CREATE TABLE 语句定义基表，其一般格式如下：

```
CREATE TABLE<表名>(<列名><数据类型>[列级完整性约束条件]
[，<列名><数据类型>[列级完整性约束条件]]…
[，<表级完整性约束条件>]);
```

其中，<表名>是所要定义的基表的名字，它可以由一个或多个属性（列）组成。

在创建表的同时通常还可以定义与该表有关的完整性约束条件，这些完整性约束条件将被存入系统的数据字典中，当用户操作表中数据时，将由 DBMS 自动检查该操作是否违背这些完整性约束条件。

如果完整性约束条件仅涉及一个属性列，则约束条件既可以定义在列级也可以定义在表级，如果该约束涉及到该表的多个属性列，则必须定义在表级。

【例 3.1】创建一个名为 IT_EMPLOYEES 的表，它由编号 EMPLOYEE_ID、名 FIRST_NAME、姓 LAST_NAME、邮箱 EMAIL、电话号码 PHONE_NUMBER、部门编号 JOB_ID、薪资 SALARY 和部门经理编号 MANAGER_ID 八个属性组成。其中

EMPLOYEE_ID 不能为空，值是唯一的，其创建语句和运行结果如图 3.2 所示。

```
SQL> create table IT_EMPLOYEES
  2  (
  3  EMPLOYEE_ID NUMBER(6) not null unique,
  4  FIRST_NAME VARCHAR2(20),
  5  LAST_NAME VARCHAR2(25) not null,
  6  EMAIL VARCHAR2(25),
  7  PHONE_NUMBER VARCHAR2(20),
  8  JOB_ID VARCHAR2(10),
  9  SALARY NUMBER(8,2),
 10  MANAGER_ID NUMBER(6) ) ;

表已创建。
```

图 3.2　例 3.1 创建表示意图

系统执行 CREATE TABLE 语句后，就在数据库中建立了一个新的空的“雇员”表 IT_EMPLOYEES，并将有关“雇员”表的定义及有关约束条件存放在数据字典中。

提示：定义表的各个属性时需要指明其数据类型及长度。不同的数据库系统支持的数据类型不完全相同，Oracle 支持的数据类型将在下一章进行详细说明。

2. 创建视图

视图是从一个或几个基表（或视图）导出的表，它与基表不同，是一个虚表。数据库中只存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基表中。所以基表中的数据发生变化，从视图中查询出的数据也将发生变化。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中自己感兴趣的数据。

SQL 语言用 CREATE VIEW 命令建立视图，其一般格式为：

```
CREATE   VIEW   <视图名>[(<列名>[,<列名>]...)]
    AS<子查询>
    [WITH CHECK OPTION]
```

其中，子查询可以是不包含 ORDER BY 子句和 DISTINCT 短语的任意复杂的 SELECT 语句。WITH CHECK OPTION 表示对视图进行 UPDATE、INSERT 和 DELETE 操作时，要保证更新、插入或删除的行满足视图定义中的谓词条件（即子查询中的条件表达式）。

在输入组成视图的属性列名时，要么全部省略，要么全部指定，没有第 3 种选择。当省略了视图的各个属性列名时，各个属性列名称隐含在该视图子查询中的 SELECT 子句目标列中，但在下列三种情况下必须明确指定组成视图的所有列名：

- 目标列存在集函数或列表达式时，需要指定列名。
- 多表连接时存在几个同名列作为视图的字段，需要指定不同的列名。
- 某个列需要重命名。

【例 3.2】建立程序员的视图 PROG_EMPLOYEES（JOB_ID='IT_PROG'），其中隐含了视图的列名，如图 3.3（a）所示。

DBMS 执行 CREATE VIEW 语句的结果只是将视图的定义存入数据字典，而并不执行其中的 SELECT 语句。只有在对视图查询时，才会按照视图的定义从基表中将数据查出。

加上了 WITH CHECK OPTION 子句的情况如图 3.3（b）所示，由于在定义

PROG_EM PLOYEES 视图时加上了 WITH CHECK OPTION 子句，以后对该视图进行插入、修改和删除操作时，DBMS 会自动加上条件 JOB_ID='IT_PROG'。

```
SQL> create view prog_employees
  2  as
  3  select employee_id,first_name,last_name,email,
  4  phone_number,salary,manager_id
  5  from it_employees
  6  where job_id='IT_PROG';

视图已创建。

SQL>
```

图 3.3（a） 例 3.2 创建视图示意图

```
SQL> create view prog_employees_1
  2  as
  3  select employee_id,first_name,last_name,email,
  4  phone_number,salary,manager_id
  5  from it_employees
  6  where job_id='IT_PROG'
  7  with check option;

视图已创建。

SQL>
```

图 3.3（b） 例 3.2 中加上 with check option 子句

3. 创建索引

在 SQL 语言中，建立索引使用 CREATE INDEX 语句，其一般格式为：

```
CREATE[UNIQUE][CLUSTER]INDEX<索引名>
ON<表名>(<列名>[<次序>][,<列名>[<次序>]]…);
```

其中，UNIQUE 选项表示此索引的每一个索引值不能重复，对应唯一的数据记录。CLUSTER 选项表示要建立的索引是聚簇索引。<表名>是所要创建索引的基表的名称。索引可以建立在对应表的一列或多列上，如果是多个列，各列名之间需用逗号分隔。<次序>选项用于指定索引值的排列次序，ASC 表示升序，DESC 表示降序，默认值为 ASC。

提示：聚簇索引即指索引项的顺序与表中记录的物理顺序相一致的索引组织。

【例 3.3】执行下面的 CREATE INDEX 语句，创建索引，如图 3.4 所示。

```
CREATE INDEX IT_LASTNAME ON IT_EMPLOYEES(LAST_NAME);
```

```
SQL> create index it_lastname on it_employees(last_name);
索引已创建。
SQL>
```

图 3.4 例 3.3 创建索引示意图

上述语句执行后将会在 IT_EMPLOYEES 表的 LAST_NAME 列上建立一个索引，而且 IT_EMPLOYEES 表中的记录将按照 LAST_NAME 值升序存放。

用户可以在查询频率最高的列上建立聚簇索引，从而提高查询效率。由于聚簇索引是将索引和表记录放在一起存储，所以在一个基表上最多只能建立一个聚簇索引。在建

立聚簇索引后，由于更新索引列数据时会导致表中记录的物理顺序的变更，系统代价较高，因此对于经常更新的列不宜建立聚簇索引。

3.2.2 DROP

当某个数据对象不再被需要，可以将它删除，SQL 语言用来删除数据对象的语句是 DROP。

1. 删除表

当某个基表不再需要时，可以使用 DROP TABLE 语句删除它。其一般格式为：

```
DROP TABLE <表名>;
```

例如，删除 IT_EMPLOYEES 表的语句为：

```
DROP TABLE IT_EMPLOYEES;
```

删除基表定义后，表中的数据、在该表上建立的索引都将自动被删除掉。因此执行删除基表的操作时一定要谨慎。

注意：在有的系统中，删除基表会导致在此表上建立的视图也一起被删除，但在 Oracle 中，删除基表后建立在此表上的视图定义仍然保留在数据字典中，而当用户引用该视图时会报错。

2. 删除视图

删除视图语句的格式为：

```
DROP VIEW<视图名>;
```

视图删除后视图的定义将从数据字典中删除。但是要注意，由该视图导出的其他视图定义仍在数据字典中，不会被删除，这将导致用户在使用相关视图时会发生错误，所以删除视图时要注意视图之间的关系，需要使用 DROP VIEW 语句将这些视图全部删除。同样删除基表后，由该基表导出的所有视图并没有被删除，需要继续使用 DROP VIEW 语句一一进行删除。

【例 3.4】将前文创建的视图 PROG_EMPLOYEES 删除，如图 3.5 所示。

```
SQL> drop view prog_employees;

视图已删除。
```

图 3.5　例 3.4 删除视图示意图

执行此语句后，PROG_EMPLOYEES 视图的定义将从数据字典中删除。如果系统中还存在由 PROG_EMPLOYEES 视图导出的视图，该视图的定义在数据字典中仍然存在，但是该视图已无法使用。

3. 删除索引

建立索引后，将由系统对其进行维护，而不需用户干预。如果数据被频繁地增加删改，系统就会花许多时间来维护该索引。在这种情况下，可以将一些不必要的索引删除掉。

在 SQL 语言中，删除索引使用 DROP INDEX 语句，其一般格式为：

```
DROP INDEX<索引名>;
```

例如，删除 IT_EMPLOYEES 表的 IT_LASTNAME 索引。

```
DROP  INDEX IT_LASTNAME;
```

删除索引后，系统也会从数据字典中将有关该索引的描述进行清除。

3.2.3 ALTER

随着应用环境和应用需求的变化，有时需要修改已建立好的基表，SQL 语言用 ALTER TABLE 语句修改基表，其一般格式为：

```
ALTER TABLE<表名>
[ADD<新列名><数据类型>[完整性约束]]
[DROP<完整性约束名>]
[MODIFY<列名><数据类型>];
```

其中，<表名>表示所要修改的基表，ADD 子句用于增加新列和新的完整性约束条件，DROP 子句用于删除指定的完整性约束条件，MODIFY 子句用于修改原有的列定义，如修改列名和数据类型。

【例 3.5】向 IT_EMPLOYEES 表中增加“雇员生日”列，其数据类型为日期型：

```
ALTER TABLE IT_EMPLOYEES ADD BIRTH_DATE DATE;
```

无论基表中原来是否有数据，增加的列一律为空值。

【例 3.6】将 IT_EMPLOYEES 表的 MANAGER_ID 字段改为 8 位，其语句为：

```
ALTER TABLE IT_EMPLOYEES MODIFY MANAGER_ID NUMBER(8);
```

【例 3.7】删除 IT_EMPLOYEES 表 EMPLOYEE_ID 字段的 UNIQUE 约束，其语句为：

```
ALTER TABLE IT_EMPLOYEES DROP UNIQUE(EMPLOYEE_ID);
```

例 3.5～例 3.7 的运行结果如图 3.6 所示。

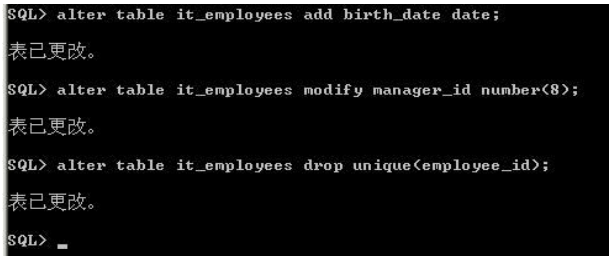

图 3.6 表结构修改示意图

注意：在 SQL 语言中，并没有提供删除属性列的语句，用户只能通过间接的方法实现这一功能。首先将被删除表中要保留的列及其内容复制到一个新表中，然后删除原表，最后再将新表重命名为原表名即可。

3.3 数据查询

在 SQL 语句中，数据查询语句 SELECT 是使用频率最高、用途最广的语句。它由

许多子句组成，通过这些子句可以完成选择、投影和连接等各种运算功能，得到用户所需的最终数据结果。其中，选择运算是使用 SELECT 语句的 WHERE 子句来完成的。投影运算是通过在 SELECT 子句中指定列名称来完成的。

连接运算则表示把两个或两个以上的表中的数据连接起来，形成一个结果集合。由于设计数据库时的关系规范化和数据存储的需要，许多信息被分散存储在数据库不同的表中，但是当显示一个完整的信息时，就需要将这些数据同时显示出来，这时就需要执行连接运算。其完整语法描述如下：

```
SELECT [ALL | DISTINCT] TOP n[PERCENT] WITH TIES select_list
[INTO[new table name]]
[FROM{table_name | view_name}[(optimizer_hints)]
[, {table_name2 | view_name2}[(optimizer_hints)]
[...,table_namel6 | view_namel6][(optimizer hints)]]]
[WHERE clause]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause]
[COMPUTE clause]
[FOR BROWSE]
```

以下将对各种查询方式和查询子句一一进行介绍。

3.3.1 简单查询

仅含有 SELECT 子句和 FROM 子句的查询是简单查询，SELECT 子句和 FROM 子句是 SELECT 语句的必选项，也就是说每个 SELECT 语句都必须包含有这两个子句。其中，SELECT 子句用于标识用户想要显示哪些列，通过指定列名或是用“*”号代表对应表的所有列；FROM 子句则告诉数据库管理系统从哪里寻找这些列，通过指定表名或是视图名称来描述。

下面的 SELECT 语句将显示表中所有的列和行。

```
select * from employees;
```

其中，SELECT 子句中的星号表示表中所有的列，该语句可以将指定表中的所有数据检索出来；FROM 子句中的 EMPLOYEES 表示 EMPLOYEES 表，即整条 SQL 语句的含义是把 EMPLOYEES 表中的所有数据按行显示出来。

大多数情况下，SQL 查询检索的行和列都比整个表的范围窄，用户将需要检索比单个行和列多，但又比数据库所有行和列少的数据。这就是更加复杂的 SELECT 语句的由来。

1. 使用 FROM 子句指定表

SELECT 语句的不同部分常用来指定要从数据库返回的数据。SELECT 语句使用 FROM 子句指定查询中包含的行和列所在的表。FROM 子句的语法格式如下：

```
[FROM{table_rmme | view_name}[(optimizer_hints)]
[, {table_name2 | view_name2}[(optimizer_hints)]
[...,table_namel6 | view_namel6][(optimizer_hints)]]]
```

与创建表一样，登录 SQL*Plus 用到一个用户名。在查询其他角色对应的方案中的表时，需要指定该方案的名称。例如，查询方案 HR 的 COUNTRIES 表中的所有行数据的 SQL 语句如下（该方案和表在安装 Oracle 时就自动创建了）。

```
SELECT * FROM HR.COUNTRIES;
```

可以在 FROM 子句中指定多个表，每个表使用逗号（,）与其他表名隔开，其格式如下所示：

```
SELECT * FROM HR.COUNTRIES, HR.DEPARTMENTS;
```

2. 使用 SELECT 指定列

用户可以指定查询表中的某些列而不是全部，这其实就是投影操作。这些列名紧跟在 SELECT 关键词后面，与 FROM 子句一样，每个列名用逗号隔开，其语法格式如下：

```
SELECT column name_1, ... , colunm_name_n
    FROM table_name_l, ... ,table_name_n
```

利用 SELECT 指定列的方法可以改变列的顺序来显示查询的结果，甚至是可以通过在多个地方指定同一个列来多次显示同一个列。

【例 3.8】创建表 COUNTRIES 时的列顺序为：COUNTRY_ID、COUNTRY_NAME、REGION_ID。通过 SELECT 指定列，可以改变列的顺序，查询显示结果如图 3.7 所示。

```
SELECT REGION_ID，COUNTRY_NAME FROM COUNTRIES;
```

```
SQL> select region_id,country_name from countries;

 REGION_ID COUNTRY_NAME
---------- ----------------------------------------
         2 Argentina
         3 Australia
         1 Belgium
         2 Brazil
         2 Canada
         1 Switzerland
         3 China
         1 Germany
         1 Denmark
         4 Egypt
         1 France

 REGION_ID COUNTRY_NAME
---------- ----------------------------------------
         3 HongKong
         4 Israel
         3 India
         1 Italy
         3 Japan
         4 Kuwait
         2 Mexico
         4 Nigeria
         1 Netherlands
         3 Singapore
         1 United Kingdom

 REGION_ID COUNTRY_NAME
---------- ----------------------------------------
         2 United States of America
         4 Zambia
         4 Zimbabwe

已选择25行。
```

图 3.7　例 3.8 中指定列查询的示意图

3. 算术表达式

在使用 SELECT 语句时，对于数字数据和日期数据都可以使用算术表达式。在 SELECT 语句中可以使用的算术运算符包括加（+）．减（-），乘（*）、除（/）和括号。使用算术表达式的示例如下：

【例 3.9】对 employees 表中薪资进行调整，所有人员的薪资增加 10%，对应的 SQL 语句如下：

```
select employee_id，first_name，last_name，salary*(1+0.1) from employees；
```

上述查询语句的运行结果如图 3.8 所示。

```
SQL> select employee_id,first_name,last_name,salary*(1+0.1) from employees;

EMPLOYEE_ID FIRST_NAME           LAST_NAME                 SALARY*(1+0.1)
----------- -------------------- ------------------------- --------------
        198 Donald               OConnell                            2860
        199 Douglas              Grant                               2860
        200 Jennifer             Whalen                              4840
        201 Michael              Hartstein                          14300
        202 Pat                  Fay                                 6600
        203 Susan                Mavris                              7150
        204 Hermann              Baer                               11000
        205 Shelley              Higgins                            13200
        206 William              Gietz                               9130
        100 Steven               King                               26400
        101 Neena                Kochhar                            18700

EMPLOYEE_ID FIRST_NAME           LAST_NAME                 SALARY*(1+0.1)
----------- -------------------- ------------------------- --------------
        102 Lex                  De Haan                            18700
        103 Alexander            Hunold                              9900
        104 Bruce                Ernst                               6600
```

图 3.8　例 3.9 运行结果部分示意图

在例 3.9 中，显示出了上调所有雇员的薪资 10%以后的薪资。当使用 SELECT 语句查询数据库时，其查询结果集中的数据列名默认为表中的列名。为了提高查询结果集的可读性，可以在查询结果集中为列指定标题。例如，在上面的示例中，将 SALARY 列乘以 1.1 后，计算出上调 10%后的雇员薪资。为了提高结果集的可读性，现在要为它指定一个新的列标题 NEW_SALARY：

```
select employee_id,first_name,last_name,salary*(1+0.1) new_salary from employees；
```

提示：如果列标题中包含一些特殊的字符，例如空格等，则必须使用双引号将列标题括起来。

4. DISTINCT 关键字

在默认情况下，结果集中包含检索到的所有数据行，而不管这些数据行是否重复出现。有的时候，当结果集中出现大量重复的行时，结果集会显得比较庞大，而不会带来有价值的信息，如在考勤记录表中仅显示考勤的人员而不显示考勤的时间时，人员的名字会大量重复出现。若希望删除结果集中重复的行，则需在 SELECT 子句中使用 DISTINCT 关键字。

【例 3.10】在 EMPLOYEES 表中包含一个 DEPARTMENT_ID 列。由于同一部门有多名雇员，相应地在 EMPLOYEES 表的 DEPARTMENT_ID 列中就会出现重复的值。假设现在要检索该表中出现的所有部门，这时我们不希望有重复的部门出现，这时需要在 DEPARTMENT_ID 列前面加上关键字 DISTINCT，以确保不出现重复的部门，其查

询语句如下：

```
select distinct department_id from employees;
```

运行上述语句后的结果如图 3.9 所示。

图 3.9　例 3.10 运行结果示意图

若不使用关键字 DISTINCT，则将在查询结果集中显示表中每一行的部门号，包括重复的部门编号。

3.3.2　WHERE 子句

WHERE 子句用于筛选从 FROM 子句中返回的值，完成的是选择操作。在 SELECT 语句中使用 WHERE 子句后，将对 FROM 子句指定的数据表中的行进行判断，只有满足 WHERE 子句中判断条件的行才会显示，而那些不满足 WHERE 子句判断条件的行则不包括在结果集中。在 SELECT 语句中，WHERE 子句位于 FROM 子句之后，其语法格式如下所示：

```
SELECT column_list
FROM table_name
WHERE conditional_expression
```

其中，CONDITIONAL_EXPRESSlON 为查询时返回记录应满足的判断条件。

1. 条件表达式

在 CONDITIONAL_EXPRESSlON 中可以用运算符来对值进行比较，可用的运算符介绍如下：

- A=B　表示若 A 与 B 的值相等，则为 TRUE。
- A>B　表示若 A 的值大于 B 的值，则为 TRUE。
- A<B　表示若 A 的值小于 B 的值，则为 TRUE。
- A!=B 或 A<>B　表示若 A 的值不等于 B 的值，则为 TRUE。
- A LIKE B　其中，LIKE 是匹配运算符。在这种判断条件中，若 A 的值匹配 B 的值，则该判断条件为 TRUE。在 LIKE 表达式中可以使用通配符。Oracle 支持的通配符为：“%”代表 0 个、1 个或多个任意字符，使用“_”代表一个任意字符。

- NOT <条件表达式>　NOT 运算符用于对结果取反。

【例 3.11】编写一个查询，判断所有 FIRST_NAME 列以"B"开头的雇员。

```
select employee_id,first name,last_name from employees
    where first_name like 'B%';
```

上述查询语句的运行结果如图 3.10 所示。

```
SQL> select employee_id,first_name,last_name from employees
  2  where first_name like 'B%';

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
        104 Bruce                Ernst
        193 Britney              Everett

SQL>
```

图 3.10　例 3.11 运行结果示意图

这里的“%”字符是一个通配符，例 3.11 中的 WHERE 子句相当于告诉 Oracle 查找雇员中所有 FIRST_NAME 以 B 开头，后面可以是由 0 个、1 个或多个字符组成的雇员信息。

2. 连接运算符

在 WHERE 子句中可以使用连接运算符将各个表达式关联起来组成复合判断条件。常用的连接运算符包括：AND 和 OR。使用 AND 连接的运算符只有在 AND 左边和右边的表达式都为 TRUE 时，AND 运算符才返回 TRUE。

【例 3.12】查询出所有属于 IT 部门（DEPARTMENT_ID=60），并且薪资大于 2000 的雇员。

```
select employee_id,first_name,last_name,salary from employees
      where department_id=60 and salary>2000;
```

上述查询语句的运行结果如图 3.11 所示。

```
SQL> select employee_id,first_name,last_name,salary from employees
  2  where department_id = 60 and salary>2000;

EMPLOYEE_ID FIRST_NAME           LAST_NAME                  SALARY
----------- -------------------- ------------------------- ----------
        103 Alexander            Hunold                      9000
        104 Bruce                Ernst                       6000
        105 David                Austin                      4800
        106 Valli                Pataballa                   4800
        107 Diana                Lorentz                     4200
```

图 3.11　例 3.12 运行结果示意图

如果使用 OR 运算符，则只要 OR 运算符左边的表达式或是 OR 运算符右边的表达式中有任一个为 TRUE，那么 OR 运算符就要返回 TRUE。

【例 3.13】在下面的查询中，将选择具有不同部门的雇员信息。

```
select employee_id,first_name,last_name,department_id from employees
    where department_id=60 or department_id=30;
```

上述查询语句的运行结果如图 3.12 所示。

```
SQL> select employee_id,first_name,last_name,department_id from employees
  2  where department_id = 60 or department_id = 30;

EMPLOYEE_ID FIRST_NAME           LAST_NAME                 DEPARTMENT_ID
----------- -------------------- ------------------------- -------------
        114 Den                  Raphaely                             30
        115 Alexander            Khoo                                 30
        116 Shelli               Baida                                30
        117 Sigal                Tobias                               30
        118 Guy                  Himuro                               30
        119 Karen                Colmenares                           30
        103 Alexander            Hunold                               60
        104 Bruce                Ernst                                60
        105 David                Austin                               60
        106 Valli                Pataballa                            60
        107 Diana                Lorentz                              60

已选择11行。
```

图 3.12　例 3.13 运行结果示意图

在复合判断条件中，需要注意运算符的优先级。Oracle 会先运算优先级高的运算符，然后再运算优先级低的运算符，同级别的优先级则从左到右进行运算。这一规则符合人们日常生活中的规定。为了增加可读性，可以使用括号将各个表达式括起来。对上面的查询使用括号界定优先级后的形式如下：

```
select employee_id,first_name,last_name,department_id from employees
    where(department_id=60) or (department_id=30);
```

3. NULL 值

在数据库中，NULL 值是一个特定的术语，用来描述记录中没有定义内容的字段值，通常我们称之为空。在 Oracle 中，判断某个条件的值时，返回值可能是 TRUE、FALSE 或 UNKNOWN。例如，如果查询一个列的值是否等于 20，而该列的值为 NULL，那么就是说无法判断该列是否为 20。如果列值为 NULL，则对该列进行判断时的值就会为 UNKNOWN，它可能等于 20，也可能不等于 20。

【例 3.14】使用 EMPLOYEES 进行 NULL 值的插入和查询。

插入一条记录：

```
insert into departments(department_id,department_name,manager_id)
    values(300,'数据库',NULL);
```

NULL 值是一个特殊的取值，使用“=”对 NULL 值进行查询是无法得到需要的结果的。

```
select department_id,department_name,manager_id
    from departments
    where manager_id = NULL;
```

提示：从查询结果中可以看出，不能使用 manager_id = NULL 这样的判断方式。

Oracle 提供了两个 SQL 运算符，IS NULL 和 lS NOT NULL。使用这两个运算符，可以判断某列的值是否为 NULL：

```
select department_id,department_name,manager_id
    from departments
    where manager_id IS NULL;
```

上述查询语句的运行结果如图 3.13 所示。

```
SQL> insert into departments(department_id,department_name,manager_id)
  2  values(300,'数据库',NULL);

已创建 1 行。

SQL> select department_id,department_name,manager_id from departments
  2  where manager_id = NULL;

未选定行

SQL> select department_id,department_name,manager_id from departments
  2  where manager_id is null;

DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID
------------- ------------------------------ ----------
          120 Treasury
          130 Corporate Tax
          140 Control And Credit
          150 Shareholder Services
          160 Benefits
          170 Manufacturing
          180 Construction
          190 Contracting
          200 Operations
          210 IT Support
          220 NOC
```

图 3.13　例 3.14 运行结果示意图

3.3.3　ORDER BY 子句

在前面介绍的数据检索技术中，只是把数据库中的数据从表中直接取出来。这时，结果集中数据的排列顺序是由数据的存储顺序决定的。但是，这种存储顺序经常不符合我们的查询需求。当查询一个比较大的表时，数据的显示会比较混乱。因此需要对检索到的结果集进行排序。在 SELECT 语句中，可以使用 ORDER BY 子句实现对查询的结果集进行排序。

使用 ORDER BY 子句的语法形式如下：

```
SELECT column_list
FROM table_name
ORDER BY[(order_by_expression[ASC|DESC])…]
```

其中，ORDER_BY_EXPRESSION 表示将要排序的列名或由列组成的表达式，关键字 ASC 指定按照升序排列，这也是默认的排列顺序，而关键字 DESC 指定按照降序排列。

【例 3.15】下面的查询语句中，将使用 ORDER BY 子句对检索到的数据进行排序，该排列顺序是按照薪资从低到高的升序进行的。

```
select employee_id,first_name,last_name,salary
    from employees
    where salary>2000
    order by salary;
```

上述查询语句的运行结果如图 3.14 所示。

从查询结果中可以看出，ORDER BY 子句使用默认的排列顺序，即升序排列，可

以使用关键字 ASC 显式指定。如果想降序排序，可以执行如下语句；

```
select employee_id,first_name,last_name,salary
    from employees
    where salary>2000
    order by salary desc;
```

```
SQL> select employee_id,first_name,last_name,salary from employees
  2  where salary>2000
  3  order by salary;

EMPLOYEE_ID FIRST_NAME           LAST_NAME                 SALARY
----------- -------------------- ------------------------- ----------
        132 TJ                   Olson                       2100
        136 Hazel                Philtanker                  2200
        128 Steven               Markle                      2200
        127 James                Landry                      2400
        135 Ki                   Gee                         2400
        191 Randall              Perkins                     2500
        119 Karen                Colmenares                  2500
        140 Joshua               Patel                       2500
        144 Peter                Vargas                      2500
        182 Martha               Sullivan                    2500
        131 James                Marlow                      2500

EMPLOYEE_ID FIRST_NAME           LAST_NAME                 SALARY
----------- -------------------- ------------------------- ----------
        198 Donald               OConnell                    2600
        199 Douglas              Grant                       2600
        118 Guy                  Himuro                      2600
```

图 3.14　例 3.15 运行结果示意图

如果需要对多个列进行排序，只需要在 ORDER BY 子句后指定多个列名。这样当输出排序结果时，首先根据第一列进行排序，当第一列的值相同时，再对第二列进行比较排序。其他列以此类推。在下面的查询语句中，将首先对 job_id 排序，然后再排序 SALARY。这样便可以使雇员的薪资分工种显示，并能了解到各工种中哪位雇员的薪资最高。

```
select last_name,job_id,salary
    from employees
    where salary>2000
    order by job_id,salary desc;
```

3.3.4　GROUP BY 子句

GROUP BY 子句用于在查询结果集中对记录进行分组，以汇总数据或者为整个分组显示单行的汇总信息。

【例 3.16】以下的查询中，从 EMPLOYEES 表中选择相应的列，分析 JOB_ID 的 SALARY 信息。

```
select job_id,salary from employees order by job_id;
```

上述查询语句的运行结果如图 3.15 所示。

从结果中可以看出，对于每个 JOB_ID 可以有多个对应的 SALARY 值。

```
SQL> select job_id,salary from employees order by job_id;

JOB_ID         SALARY
---------- ----------
AC_ACCOUNT       8300
AC_MGR          12000
AD_ASST          4400
AD_PRES         24000
AD_VP           17000
AD_VP           17000
FI_ACCOUNT       7700
FI_ACCOUNT       8200
FI_ACCOUNT       6900
FI_ACCOUNT       7800
FI_ACCOUNT       9000

JOB_ID         SALARY
---------- ----------
FI_MGR          12000
HR_REP           6500
IT_PROG          6000
IT_PROG          4800
```

图 3.15　例 3.16 部分运行结果示意图

使用 GROUP BY 子句和统计函数，可以实现对查询结果中每一组数据进行分类统计。所以，在结果中每组数据都有一个与之对应的统计值。在 Oracle 系统中，经常使用的统计函数如表 3.3 所示。

表 3.3　常用的统计函数

函数	描述
COUNT	返回找到的记录数
MIN	返回一个数字列或是计算列的最小值
MAX	返回一个数字列或是计算列的最大值
SUM	返回一个数字列或是计算列的总和
AVG	返回一个数字列或是计算列的平均值

【例 3.17】使用 GROUP BY 子句对薪资记录进行分组，使用 SQL 函数计算每个 JOB_ID 的平均薪资（AVG）、所有薪资的总和（SUM），以及最高薪资（MAX）和各组的行数，如图 3.16 所示。

```
select job_id，avg(salary),sum(salary),max(salary),count(job_id) from employees
    group by job_id；
```

上述查询语句的运行结果如图 3.16 所示。

在使用 GROUP BY 子句时，必须满足下面的条件：

- 在 SELECT 子句的后面只可以有两类表达式：统计函数和进行分组的列名。
- 在 SELECT 子句中的列名必须是进行分组的列，除此之外添加其他的列名都是错误的，但是，GROUP BY 子句后面的列名可以不出现在 SELECT 子句中。
- 如果使用了 WHERE 子句，那么所有参加分组计算的数据必须首先满足 WHERE 子句指定的条件。
- 在默认情况下，将按照 GROUP BY 子句指定的分组列升序排列，如果需要重新排序，可以使用 ORDER BY 子句指定新的排列顺序。

```
SQL> select job_id,avg(salary),sum(salary),max(salary),count(job_id)
  2  from employees
  3  group by job_id;

JOB_ID     AVG(SALARY) SUM(SALARY) MAX(SALARY) COUNT(JOB_ID)
---------- ----------- ----------- ----------- -------------
AC_MGR           12000       12000       12000             1
AC_ACCOUNT        8300        8300        8300             1
IT_PROG           5760       28800        9000             5
ST_MAN            7280       36400        8200             5
AD_ASST           4400        4400        4400             1
PU_MAN           11000       11000       11000             1
SH_CLERK          3215       64300        4200            20
AD_VP            17000       34000       17000             2
FI_ACCOUNT        7920       39600        9000             5
MK_MAN           13000       13000       13000             1
PR_REP           10000       10000       10000             1

JOB_ID     AVG(SALARY) SUM(SALARY) MAX(SALARY) COUNT(JOB_ID)
---------- ----------- ----------- ----------- -------------
FI_MGR           12000       12000       12000             1
PU_CLERK          2780       13900        3100             5
SA_MAN           12200       61000       14000             5
MK_REP            6000        6000        6000             1
AD_PRES          24000       24000       24000             1
SA_REP            8350      250500       11500            30
HR_REP            6500        6500        6500             1
ST_CLERK          2785       55700        3600            20

已选择19行。
```

图 3.16　例 3.17 运行结果示意图

【例 3.18】下面是一个错误的查询，由于在 SELECT 子句后面出现了 SALARY 列，而该列并没有出现在 GROUP BY 子句中，所以该语句是一个错误的查询。

```
select job_id,salary,avg(salary),sum(salary),max(salary),count(*)
    from employees group by job_id;
```

上述查询语句的运行结果如图 3.17 所示。

```
SQL> select job_id,salary,avg(salary),sum(salary),max(salary),count(job_id)
  2  from employees
  3  group by job_id;
select job_id,salary,avg(salary),sum(salary),max(salary),count(job_id)
              *
第 1 行出现错误:
ORA-00979: 不是 GROUP BY 表达式
```

图 3.17　例 3.18 的错误运行结果示意图

与 ORDER BY 子句相似，GROUP BY 子句也可以对多个列进行分组。在这种情况下，GROUP BY 子句将在主分组范围内进行二次分组。

【例 3.19】下面的查询是对各部门中的各个工种类型进行分组。

```
select department_id,job_id,avg(salary),sum(salary),max(salary),count(*)
    from employees group by department_id,job_id;
```

在 GROUP BY 子句中还可以使用运算符 ROLLUP 和 CUBE，这两个运算符在功能上非常类似。在 GROUP BY 子句中使用它们后，都将会在查询结果中附加一行汇总信息。

【例3.20】在下面的示例中，GROUP BY子句将使用ROLLUP运算符汇总JOB_ID列。

```
select job_id,avg(salary),sum(salary),max(salary).count(*)
    from employees group by rollup(job_id);
```

上述查询语句的运行结果如图 3.18 所示。

```
SQL> select job_id,avg(salary),sum(salary),max(salary),count(*)
  2  from employees
  3  group by rollup(job_id);

JOB_ID     AVG(SALARY) SUM(SALARY) MAX(SALARY)   COUNT(*)
---------- ----------- ----------- ----------- ----------
AC_ACCOUNT        8300        8300        8300          1
AC_MGR           12000       12000       12000          1
AD_ASST           4400        4400        4400          1
AD_PRES          24000       24000       24000          1
AD_VP            17000       34000       17000          2
FI_ACCOUNT        7920       39600        9000          5
FI_MGR           12000       12000       12000          1
HR_REP            6500        6500        6500          1
IT_PROG           5760       28800        9000          5
MK_MAN           13000       13000       13000          1
MK_REP            6000        6000        6000          1

JOB_ID     AVG(SALARY) SUM(SALARY) MAX(SALARY)   COUNT(*)
---------- ----------- ----------- ----------- ----------
PR_REP           10000       10000       10000          1
PU_CLERK          2780       13900        3100          5
PU_MAN           11000       11000       11000          1
SA_MAN           12200       61000       14000          5
SA_REP            8350      250500       11500         30
SH_CLERK          3215       64300        4200         20
ST_CLERK          2785       55700        3600         20
ST_MAN            7280       36400        8200          5
             6461.68224      691400       24000        107

已选择20行。
```

图 3.18　例 3.20 的运行结果示意图

从查询结果中可以看出，使用 ROLLUP 运算符后，在查询结果的最后一行列出了本次统计的汇总。

3.3.5　HAVING 子句

HAVING 子句通常与 GROUP BY 子句一起使用，在完成对分组结果统计后，可以使用 HAVING 子句对分组的结果做进一步的筛选。如果不使用 GROUP BY 子句，HAVING 子句的功能与 WHERE 子句一样。HAVING 子句和 WHERE 子句的相似之处就是都定义搜索条件，但是和 WHERE 子句不同，HAVING 子句与组有关，而 WHERE 子句与单个的行有关。

如果在 SELECT 语句中使用了 GROUP BY 子句，那么 HAVING 子句将应用于 GROUP BY 子句创建的那些组。如果指定了 WHERE 子句，而没有指定 GROUP BY 子句，那么 HAVING 子句将应用于 WHERE 子句的输出，并且整个输出被看作是一个组，如果在 SELECT 语句中既没有指定 WHERE 子句，也没有指定 GROUP BY 子句，那么 HAVING 子句将应用于 FROM 子句的输出，并且将其看作是一个组。

提示：对 HAVING 子句作用的理解有一个方法，就是记住 SELECT 语句中的子句的处理顺序。在 SELECT 语句中，首先由 FROM 子句找到数据表，WHERE 子句则接收 FROM 子句输出的数据，而 HAVING 子句则接收来自 GROUP BY、WHERE 或 FROM 子句的输入。

【例 3.21】列出平均薪资大于 10000 的统计信息。

```
select job_id,avg(salary),sum(salary),max(salary),count(*)
    from employees group by job_id having avg(salary)>10000;
```

上述查询语句的执行结果如图 3.19 所示。

```
SQL> select job_id,avg(salary),sum(salary),max(salary),count(*)
  2  from employees
  3  group by job_id having avg(salary)>10000;

JOB_ID     AVG(SALARY) SUM(SALARY) MAX(SALARY)   COUNT(*)
---------- ----------- ----------- ----------- ----------
AC_MGR           12000       12000       12000          1
PU_MAN           11000       11000       11000          1
AD_VP            17000       34000       17000          2
MK_MAN           13000       13000       13000          1
FI_MGR           12000       12000       12000          1
SA_MAN           12200       61000       14000          5
AD_PRES          24000       24000       24000          1

已选择7行。
```

图 3.19　例 3.21 的运行结果示意图

从查询结果可以看出，SELECT 语句使用 GROUP BY 子句对 EMPLOYEES 表进行分组统计，然后再由 HAVING 子句根据统计值做进一步筛选。

通常情况下，HAVING 子句与 GROUP BY 子句一起使用，这样可以在汇总相关数据后再进一步筛选汇总的数据。

3.3.6　多表连接查询

到目前为止，大部分查询都集中在 FROM 子句仅使用一个表。但是，在设计数据库时，为了使数据库规范化，常常要把数据分别存放在不同的表中。这样可以消除数据冗余、插入异常和删除异常。但是在查询数据时，为了获取完整的信息就要将多个表连接起来，从多个表中查询数据。例如，为了获知雇员所在部门，可以在 EMPLOYEES 表中获取部门编号 DEPARTMENT_ID，为了得到部门的名称还需要查询 DEPARTMENTS 表。下面将对实现多表查询的方法一一进行介绍。

1. 简单连接

连接查询实际上是通过表与表之间相互关联的列进行数据的查询，对于关系数据库来说，连接是查询最主要的特征。简单连接使用逗号将两个或多个表进行连接，这是最简单、也是最常用的多表查询形式。

（1）基本形式

简单连接仅是通过SELECT子句和FROM子句来连接多个表，其查询的结果是一个通过笛卡尔积所生成的表。所谓笛卡尔积所生成的表，就是由一个基表中每一行与另一个基表的每一行连接在一起所生成的表，查询结果的行数是两个基表行数的积。

【例 3.22】以下的查询操作将 EMPLOYEES 表和 DEPARTMENTS 表相连接，从而生成一个笛卡尔积。

```
select employee_id,last_name,department_name from employees,departments;
```

（2）条件限定

在实际需求中，由于笛卡尔积中包含了大量的冗余信息，这在一般情况下毫无意义。为了避免这种情况的出现，通常是在 SELECT 语句中提供了一个连接条件，过滤掉其

中无意义的数据，从而使得结果满足用户的需求。

SELECT 语句的 WHERE 子句提供了这个连接条件，可以有效避免笛卡尔积的出现。使用 WHERE 子句限定时，只有第一个表中的列与第二个表中相应列相互匹配后才会在结果集中显示，这是连接查询中最常用的形式。

【例 3.23】下面的语句通过在 WHERE 子句中使用连接条件，实现了查询雇员信息，以及雇员所对应的部门信息。

```
select employee_id,last_name,department_name from employees,departments
    wherc employees.department_id=departments.department_id;
```

这次查询返回的结果就有意义了，每行数据都包含了有意义的雇员信息，以及各雇员所在的部门名称信息。

提示：*若希望进一步限定搜索条件，则可以在 WHERE 子句中增加新的限定条件。*

【例 3.24】增加新的限定条件，只显示工作部门为 Shipping 的雇员信息。

```
select employee_id,last_name,department_name
    from employees,departments
    where employees.department_id=departments.department_id
    and departments.department_name='Shipping';
```

上述查询语句的运行结果如图 3.20 所示。

```
SQL> select employee_id,last_name,department_name
  2  from employees,departments
  3  where employees.department_id = departments.department_id
  4  and departments.department_name = 'Shipping';

EMPLOYEE_ID LAST_NAME                 DEPARTMENT_NAME
----------- ------------------------- ------------------------------
        198 OConnell                  Shipping
        199 Grant                     Shipping
        120 Weiss                     Shipping
        121 Fripp                     Shipping
        122 Kaufling                  Shipping
        123 Vollman                   Shipping
        124 Mourgos                   Shipping
        125 Nayer                     Shipping
        126 Mikkilineni               Shipping
        127 Landry                    Shipping
        128 Markle                    Shipping

EMPLOYEE_ID LAST_NAME                 DEPARTMENT_NAME
----------- ------------------------- ------------------------------
```

图 3.20　例 3.24 的运行结果示意图

从多个表中提取信息时，查询所使用表之间应当存在逻辑上的联系。这种联系经常以外键的形式出现，但并不是必须以外键的形式存在。

注意：*在以上示例中，连接的两个表具有同名的列，则必须使用表名对列进行限定，以确认该列属于哪一个表。*

（3）表别名

在以上示例演示中，我们发现，在多表查询时，如果多个表之间存在同名的列，则必须使用表名来限定列。但是，随着查询变得越来越复杂，语句会因为每次限定列时输入表名而变得冗长乏味。因此，SQL 语言提供了另一种机制——表别名。表别名是在 FROM 子句中用于各个表的“简短名称”，它们可以唯一地标识数据源。上面的查询可

以采用如下方式重新编写：

```
select em.employee_id,em.last_name,dep.department_name
    from employees em,departments dep
    where em.department_id=dep.department_id
    and dep.department_name='Shipping';
```

这个具有更少 SQL 代码的查询会得到相同的结果。其中，EM 代表 EMPLOYEES，DEP 代表 DEPARTMENTS。

注意：如果为表指定了别名，那么语句中的所有子句都必须使用别名，而不允许再使用实际的表名。

以下使用表别名的方式是错误的。

```
select employees.employee_id,employees.last_name,dep.department_name
    from employees em,departments dep
    where em.department_id=dep.department_id
    and dep.department_name='Shipping';
```

上述查询语句的运行结果如图 3.21 所示。

```
SQL> select employees.employee_id,last_name,dep.department_name
  2  from employees em,departments dep
  3  where em.department_id = dep.department_id;
select employees.employee_id,last_name,dep.department_name
       *
第 1 行出现错误:
ORA-00904: "EMPLOYEES"."EMPLOYEE_ID": 标识符无效
```

图 3.21 表别名使用错误示意图

出现问题的原因是Oracle编译SQL语句时出现了问题。这里需要介绍一下 SELECT 语句中各子句执行的顺序，从而便可知道出错的真正原因。在 SELECT 语句的执行顺序中，FROM 子句最先被执行，然后就是 WHERE 子句，最后才是 SELECT 子句。当在 FROM 子句中指定表别名后，表的真实名称将被替换。同时，其他的子句只能使用表别名来限定列。在上面的示例中，由于 FROM 子句已经用表别名覆盖了表的真实名称，当执行 SELECT 子句选择显示的列时，将无法找到真实表名称 EMPLOYEES 所限定的列。

2. JOIN 连接

除了使用逗号连接外，Oracle 还支持使用关键字 JOIN 连接。使用 JOIN 连接的语法格式如下：

```
    FROM join_table1 join_type join_table2
    [ON(join_condition)]
```

其中，JOIN_TABLE1 指出参与连接操作的表名；JOIN_TYPE 指出连接类型，常用的连接包括内连接、自然连接、外连接和自连接。连接查询中的 ON(JOIN_CONDITION) 指出连接条件，它由被连接表中的列和比较运算符、逻辑运算符等构成。

（1）内连接

内连接是一种常用的多表查询，一般用关键字 INNER JOIN。其中，可以省略 INNER 关键字，而只使用 JOIN 关键字表示内连接。内连接使用比较运算符时，在连接表的某

些列之间进行比较操作，并列出表中与连接条件相匹配的数据行。

使用内连接查询多个表时，在 FROM 子句中除了 JOIN 关键字外，还必须定义一个 ON 子句，ON 子句指定内连接操作列出与连接条件匹配的数据行，它使用比较运算符比较被连接列值。简单地说，内连接就是使用 JOIN 指定用于连接的两个表，使用 ON 指定连接表的连接条件。若进一步限制查询范围，则可以直接在后面添加 WHERE 子句。

【例 3.25】以下的查询使用内连接查询雇员信息和雇员所在的部门名称。

```
select em.employee_id, em.last_name, dep.department_name
    from employees em inner join departments dep
    on em.department_id=dep.department_id
    where em.job_id='AD_ASST';
```

上述查询语句的运行结果如图 3.22 所示。

```
SQL> select em.employee_id,em.last_name,dep.department_name
  2  from employees em inner join departments dep
  3  on em.department_id = dep.department_id
  4  where em.job_id = 'AD_ASST';

EMPLOYEE_ID LAST_NAME                 DEPARTMENT_NAME
----------- ------------------------- ------------------------------
        200 Whalen                    Administration
```

图 3.22　例 3.25 的运行结果示意图

提示：使用内连接也可以实现两个以上表的查询。

【例 3.26】使用内连接查询雇员的信息、名称以及工作名称。

```
select em.employee_id,em.last_name,dep.department_name,j.job_title
     from employees em inner join jobs
         on em.job_id=jobs.job_id
         inner join departments dep
         on em.department_id=dep.department_id
    where em.job_id='IT_PROG';
```

（2）自然连接

自然连接与内连接的功能相似，在使用自然连接查询多个表时，Oracle 会将第一个表中的那些列与第二个表中具有相同名称的列进行连接。在自然连接中，用户不需要明确指定进行连接的列，系统会自动完成这一任务。

下面的查询语句使用自然连接连接 EMPLOYEES 和 DEPARTMENTS 表。

```
select em.employee_id,em.first_name,em.last_name,dep.departmentname
    from employees em natural join departments dep
    where dep.departmentname='Sales';
```

自然连接在实际的应用中很少，因为它有一个限制条件，即连接的各个表之间必须具有相同名称的列，而这在实际应用中可能和应用的实际含义发生矛盾。

假设 EMPLOYEES 表和 DEPARTMENTS 表都有一个 ADDRESS 列，则在进行自然连接时，Oracle 会尝试使用 EMPLOYEES 和 DEPARTMENTS 的这两个列连接表，这要求对应的 ADDRESS 列相同。但是在应用语义上，毫无疑问这两个 ADDRESS 列代表了完全不同的含义（一个是雇员的居住地址，一个是部门的所在地址），这样的连接

毫无价值。

（3）外连接

使用内连接进行多表查询时，返回的查询结果集中仅包含符合查询条件（WHERE 搜索条件或 HAVING 条件）和连接条件的行。内连接消除了与另一个表中的任何行不匹配的行，而外连接扩展了内连接的结果集，除返回所有匹配的行外，还会返回一部分或全部不匹配的行，这主要取决于外连接的种类。

外连接分为左外连接（LEFT OUTER JOIN 或 LEFT JOIN）、右外连接（RIGHT OUTER JOIN 或 RIGHT JOIN）和全外连接（FULL OUTER JOIN 或 FULL JOIN）三种。与内连接不同的是，外连接不只列出与连接条件相匹配的行，还列出左表（左外连接时）、右表（右外连接时）或两个表（全外连接时）中所有符合搜索条件的数据行。

【例 3.27】演示内连接和外连接的区别。内连接语句及其运行结果如图 3.23 所示。

```
insert into employees(employee_id,last_name,email,hire_date,job_id,department_id)
    values(1000,'blaine','blaine@hotmail.com',to_date('2009-05-01', yyyy-mm-dd'),
    'IT_PROG',null);
select em.employee_id,em.last_name,dep.department_name
    from employees em inner join departments dep
    on em.department_id=dep.department_id
    where em.job_id='IT_PROG';
```

```
SQL> insert into employees
  2  (employee_id,last_name,email,hire_date,job_id,department_id)
  3  values(1000,'blaine','blaine@hotmail.com',
  4  to_date('2009-05-01','yyyy-mm-dd'),'IT_PROG',null);

已创建 1 行。

SQL> select em.employee_id,em.last_name,dep.department_name
  2  from employees em inner join departments dep
  3  on em.department_id = dep.department_id
  4  where em.job_id = 'IT_PROG';

EMPLOYEE_ID LAST_NAME                 DEPARTMENT_NAME
----------- ------------------------- ------------------------------
        104 Ernst                     IT
        103 Hunold                    IT
        107 Lorentz                   IT
        105 Austin                    IT
        106 Pataballa                 IT
```

图 3.23　例 3.27 内连接的运行结果示意图

从上面的查询结果看出，即使向 EMPLOYEES 表添加了一行 JOB_ID 等于 IT_PROG 的雇员信息，在内连接中仍然不会显示该行。因为在新添加记录中 DEPARTMENT_ID 列值不存在于 DEPARTMENTS 中。

外连接语句及其运行结果如图 3.24 所示。

```
select em employee_id,em_last_name,dep.department_name
    from employees em left outer join departments dep
    on em.department_id=dep.department_id
    where em.job_id='IT_PROG';
```

```
SQL> select em.employee_id,em.last_name,dep.department_name
  2  from employees em left outer join departments dep
  3  on em.department_id = dep.department_id
  4  where em.job_id = 'IT_PROG';

EMPLOYEE_ID LAST_NAME                 DEPARTMENT_NAME
----------- ------------------------- ------------------------------
        107 Lorentz                   IT
        106 Pataballa                 IT
        105 Austin                    IT
        104 Ernst                     IT
        103 Hunold                    IT
       1000 blaine

已选择6行。
```

图 3.24　例 3.27 外连接的运行结果示意图

上面查询语句中的 FROM 子句，使用 LEFT OUTER JOIN 指定使用左外连接。从查询结果中看出，左外连接的查询结果集中不仅包含相匹配的行，还包含左表（EMPLOYEES）中所有满足 WHERE 限制的行，而不论是否与右表相匹配。

同样，当执行右外连接时，则表示将要返回连接条件右边表中的所有行，而不管左边表中各行。例如，如果想要对 EMPLOYEES 和 DEPARTMENTS 表进行查询，搜索位于特定位置的所有雇员和部门，也包括没有雇员的部门，其 SQL 语句如下：

```
select em.employee_id,em.last_name,dep.department_name
    from employees em right outer join departments dep
    on em.department_id=dep.department_id
    where dep.location_id=1700;
```

提示：从查询结果可以看出，右外连接查找出了大量的没有雇员的部门，而在内连接和左外连接查询中则没有找到这些记录，读者可以自行验证。

注意：在外连接的左外连接和右外连接中，要特别注意两个表的位置。

此外，还有一种外连接类型即完全外连接。完全外连接相当于同时执行一个左外连接和一个右外连接。完全外连接查询会返回所有满足连接条件的行。在执行完全外连接时，系统开销很大，因为 Oracle 实际上会执行一个完整的左连接查询和右连接查询，然后再将结果集合并，并消除重复的记录行。

使用完全外连接查询的 SQL 语句如下：

```
select em.employee_id,em.1ast_name,dep.department_name
    from employees em full outer join departments dep
    on em.department_id=dep.department_id
    where dep.location_id=1700 or em.job_id='IT_PROG';
```

（4）自连接

有时候，用户可能会拥有自引用式外键。自引用式外键意味着表中的一个列可以是该表主键的一个外键。例如，EMPLOYEES 表的 MANAGER_ID 列可以是另一行的 EMPLOYEE_ID，因为部门经理也是雇员。通过下面的语句可以看出 MANAGER_ID 列和 EMPLOYEES_ID 列的关联：

```
select employee_id,last_name,job_id,manager_id
    from employees
    order by employee_id;
```

上述查询语句的运行结果如图 3.25 所示。

```
SQL> select employee_id,last_name,job_id,manager_id
  2  from employees
  3  order by employee_id;

EMPLOYEE_ID LAST_NAME                 JOB_ID     MANAGER_ID
----------- ------------------------- ---------- ----------
        100 King                      AD_PRES
        101 Kochhar                   AD_VP             100
        102 De Haan                   AD_VP             100
        103 Hunold                    IT_PROG           102
        104 Ernst                     IT_PROG           103
        105 Austin                    IT_PROG           103
```

图 3.25　雇员和经理之间的关系

从中可看出雇员之间的关系，如 King(100)负责管理 Kochhar(101)和 De Haan(102)；而 De Haan(102)负责管理 Hunold(103)等。

通过自连接，用户可以在查询结果的同一行中看到雇员和部门经理的信息。为了实现自连接查询，用户需要在 FROM 子句中指定两次 EMPLOYEES 表为数据源。

【例 3.28】用户通过自连接，在同一行中看到雇员和部门经理的信息。

```
select em1.last_name "manager",em2.last_name "employee"
    from employees eml left join employees em2
    on em1.employee_id=em2.manager_id
    order by em1.employee_id;
```

上述查询语句的运行结果如图 3.26 所示。

```
SQL> select em1.last_name "manager" , em2.last_name "employee"
  2  from employees em1 left join employees em2
  3  on em1.employee_id = em2.manager_id
  4  order by em1.employee_id;

manager                   employee
------------------------- -------------------------
King                      Hartstein
King                      Kochhar
King                      De Haan
King                      Raphaely
King                      Weiss
King                      Fripp
King                      Kaufling
King                      Vollman
King                      Mourgos
King                      Russell
King                      Partners
```

图 3.26　例 3.28 的运行结果示意图

自连接是在 FROM 子句中两次指定了同一个表，为了在其他子句中区分，分别为表指定了表别名。这样 Oracle 就可以将两个表看作是分离的两个数据源，并且从中获取相应的数据。

3.3.7　集合操作

集合操作就是将两个或多个 SQL 查询结果合并构成复合查询，以完成一些特殊的任务需求。集合操作主要由集合操作符实现，常用的集合操作符包括 UNION（并运算）、

UNION ALL、INTERSECT（交运算）和 MINUS（差运算）。

1. UNION

UNION 运算符可以将多个查询结果集相加，形成一个结果集，其结果等同于集合运算中的并运算。即 UNION 运算符可以将第一个查询中的所有行与第二个查询中的所有行相加，并消除其中重复的行形成一个合集。

【例 3.29】下面的示例中，第一个查询将选择所有 LAST_NAME 列以 C 或者 S 开头的雇员信息，第二个查询将会选择所有 LAST_NAME 列以 S 或者 T 开头的雇员信息。其结果是所有 LAST_NAME 列以 C 或者 S 或者 T 开头的雇员信息均会被列出。

```
select employee_id,last_name
    from employees
    where last_name like 'C%'or last_name like 'S%'
    union
    select employee_id,last_name
    from employees
    whefe last_name like 'S%' or last_name like 'T%';
```

上述查询语句的运行结果如图 3.27 所示。

```
SQL> select employee_id,last_name
  2  from employees
  3  where last_name like 'C%' or last_name like 'S%'
  4  union
  5  select employee_id, last_name
  6  from employees
  7  where last_name like 'S%' or last_name like 'T%';

EMPLOYEE_ID LAST_NAME
----------- -------------------------
        110 Chen
        111 Sciarra
        117 Tobias
        119 Colmenares
        138 Stiles
        139 Seo
        148 Cambrault
        150 Tucker
        154 Cambrault
        155 Tuvault
        157 Sully
```

图 3.27　例 3.29 的部分运行结果示意图

注意：UNION 运算会将合集中的重复记录滤除，这是 UNION 运算和 UNION ALL 运算唯一不同的地方。

2. UNION ALL

UNION ALL 与 UNION 语句的工作方式基本相同，不同之处是 UNION ALL 操作符形成的结果集中包含有两个子结果集中重复的行。

```
select employee_id,last_name
    from employees
    where last_name like 'C%'or last_name like 'S%'
    union all
    select employee_id,last_name
```

```
        from employees
        whefe last_name like 'S%' or last_name like 'T%';
```

3. INTERSECT

INTERSECT 操作符也用于对两个 SQL 语句所产生的结果集进行处理。不同之处是 UNION 基本上是一个 OR 运算，而 INTERSECT 则比较像 AND。即 UNION 是并集运算，而 INTERSECT 是交集运算。

【例 3.30】修改例 3.29 的查询语句。使用 INTERSECT 集合操作，在查询结果集中保留 LAST_NAME 以 S 开头的雇员。

```
select employee_id,last_name
        from employees
        where last_name like 'C%'or last_name like 'S%'
        intersect
        select employee_id,last_name
        from employees
        whefe last_name like 'S%' or last_name like 'T%';
```

上述查询语句的运行结果如图 3.28 所示。

```
SQL> select employee_id,last_name
  2  from employees
  3  where last_name like 'C%' or last_name like 'S%'
  4  intersect
  5  select employee_id, last_name
  6  from employees
  7  where last_name like 'S%' or last_name like 'T%';

EMPLOYEE_ID LAST_NAME
----------- -------------------------
        111 Sciarra
        138 Stiles
        139 Seo
        157 Sully
        159 Smith
        161 Sewall
        171 Smith
        182 Sullivan
        184 Sarchand

已选择9行。
```

图 3.28　例 3.30 的运行结果示意图

4. MINUS

MINUS 集合运算符可以找到两个给定的集合之间的差集，也就是说该集合操作符会返回所有从第一个查询中返回的，但是没有在第二个查询中返回的记录。

【例 3.31】以下面的查询语句为例，使用运算符 MINUS 求两个查询的差集。第一个查询会返回所有 LAST_NAME 以 C 或 S 开头的雇员，而第二个查询会返回所有 LAST_NAME 以 S 和 T 开头的雇员。因此，两个查询结果集的 MINUS 操作将返回 LAST_NAME 以 C 开头的那些雇员。

```
select employee_id,last_name
        from employees
        where last_name like 'C%'or last_name like 'S%'
        minus
```

```
select employee_id,last_name
from employees
whefe last_name like 'S%' or last_name like 'T%';
```

上述查询语句的运行结果如图 3.29 所示。

```
SQL> select employee_id,last_name
  2  from employees
  3  where last_name like 'C%' or last_name like 'S%'
  4  minus
  5  select employee_id, last_name
  6  from employees
  7  where last_name like 'S%' or last_name like 'T%';

EMPLOYEE_ID LAST_NAME
----------- -------------------------
        110 Chen
        119 Colmenares
        148 Cambrault
        154 Cambrault
        187 Cabrio
        188 Chung

已选择6行。
```

图 3.29　例 3.31 的运行结果示意图

说明：在使用集合操作符编写复合查询时，其规则包括：第一、在构成复合查询的各个查询中，各 SELECT 语句指定的列必须在数量上和数据类型上相匹配；第二、不允许在构成复合查询的各个查询中规定 ORDER BY 子句；第三、不允许在 BLOB、LONG 这样的大数据类型对象上使用集合操作符。

3.3.8　子查询

子查询和连接查询一样，都提供了使用单个查询访问多个表中数据的方法。子查询在其他查询的基础上，提供一种进一步有效的方式来表示 WHERE 子句中的条件。子查询是一个 SELECT 语句，它可以在 SELECT、INSERT、UPDATE 或 DELETE 语句中使用。虽然大部分子查询是在 SELECT 语句的 WHERE 子句中实现，但实际上它的应用不仅仅局限于此。例如，也可以在 SELECT 和 HAVING 子句中使用子查询。

1. IN 关键字

使用 IN 关键字可以将原表中特定列的值与子查询返回的结果集中的值进行比较，如果某行的特定列的值存在，则在 SELECT 语句的查询结果中就包含这一行。

【例 3.32】使用子查询查看所有部门在某一地区（1700）的雇员信息。

```
select employee_id,last_name,department_id
    from employees
    where department_id in (
        select department_id
        from departments
        where location_id=1700);
```

上述查询语句的运行结果如图 3.30 所示。

```
SQL> select employee_id,last_name,department_id from employees
  2  where department_id in (
  3  select department_id from departments
  4  where location_id = 1700);

EMPLOYEE_ID LAST_NAME                 DEPARTMENT_ID
----------- ------------------------- -------------
        200 Whalen                               10
        205 Higgins                             110
        206 Gietz                               110
        100 King                                 90
        101 Kochhar                              90
        102 De Haan                              90
        108 Greenberg                           100
        109 Faviet                              100
        110 Chen                                100
        111 Sciarra                             100
        112 Urman                               100

EMPLOYEE_ID LAST_NAME                 DEPARTMENT_ID
----------- ------------------------- -------------
        113 Popp                                100
        114 Raphaely                             30
        115 Khoo                                 30
        116 Baida                                30
        117 Tobias                               30
        118 Himuro                               30
        119 Colmenares                           30

已选择18行。
```

图 3.30　例 3.32 的运行结果示意图

该查询语句执行顺序为：首先执行括号内的子查询，然后再执行外层查询。仔细观察括号内的子查询，可以看到该子查询的作用仅提供了外层查询 WHERE 子句所使用的限定条件。

单独执行该子查询则会将 DEPARTMENTS 表中所有 location_id 等于 1700 的部门编号全部返回。

```
select department_id from departments where location_id=1700;
```

这些返回值将由 IN 关键字用来与 EMPLOYEES 表中每一行的 DEPARTMENT_ID 列进行比较，若列值存在于这些返回值中，则外层查询会在结果集中显示该行。

注意：在使用子查询时，子查询返回的结果必须和外层引用列的值在逻辑上具有可比较性。

2. EXISTS 关键字

在一些情况下，只需要考虑是否满足判断条件，而数据本身并不重要，这时就可以使用 EXISTS 关键字来定义子查询。EXISTS 关键字只注重子查询是否返回行，如果子查询返回一个或多个行，那么 EXISTS 便返回为 TRUE，否则为 FALSE。

要使 EXISTS 关键字有意义，则应在子查询中建立搜索条件。

以下查询语句返回的结果与例 3.32 相同。

```
select employee_id,last_name from employees em
    where exists(
        select * from departments dep
        where em.department_id=dep.department_id
        and location_id=1700);
```

在该语句中，外层的 SELECT 语句返回的每一行数据都要由子查询来评估。如果

EXISTS 关键字中指定的条件为真，查询结果就包含这一行；否则该行被丢弃。因此，整个查询的结果取决于内层的子查询。

提示：由于 EXISTS 关键字的返回值取决于查询是否会返回行，而不取决于这些行的内容，因此对子查询来说，输出列表无关紧要，可以使用“*”代替。

3. 比较运算符

如果可以确认子查询返回的结果只包含一个单值，那么可以直接使用比较运算符连接子查询。经常使用的比较运算符包括等于（=）、不等于（<>）、小于（<）、大于（>）、小于等于（<=）和大于等于（>=）。

【例 3.33】查询 EMPLOYEES 表，将薪资大于本职位平均薪资的雇员信息显示出来。

```
select employee_id,last_name,job_id,salary
      from employees
      where job_id='PU_MAN' and
           salary>=(select avg(salary) from employees
           where job_id=' PU_MAN ');
```

上述查询语句的运行结果如图 3.31 所示。

```
SQL> select employee_id,last_name,job_id,salary from employees
  2  where job_id = 'PU_MAN' and
  3  salary >=(select avg(salary) from employees
  4  where job_id = 'PU_MAN');

EMPLOYEE_ID LAST_NAME                 JOB_ID         SALARY
----------- ------------------------- ---------- ----------
        114 Raphaely                  PU_MAN          11000
```

图 3.31　例 3.33 的运行结果示意图

注意：在使用比较运算符连接子查询时，必须保证子查询的返回结果只包含一个值，否则整个查询语句将失败。

提示：子查询的使用相对来说比较复杂，但同时也是最灵活、最强大的一种查询方式，需要多多进行练习，熟练掌握。

3.4　数据操纵

SQL 的数据操纵功能通过数据操纵语言（Data Manipulation Language，DML）实现，用于改变数据库中的数据。数据更新包括插入、删除和修改 3 种操作，对应 INSERT、DELETE 和 UPDATE 三条语句。在 Oracle 11g 中，DML 除了包括 INSERT、UPDATE 和 DELETE 语句之外，还包括 TRUNCATE、CALL、EXPLAIN PLAN、LOCK TABLE 和 MERGE 等语句。在本节中将对 INSERT、UPDATE、DELETE、TRUNCATE 常用语句进行介绍。

3.4.1　INSERT 语句

INSERT 语句用于完成各种向数据表中插入数据的功能，既可根据对列赋值一次插入一条记录，也可根据 SELECT 查询子句获得的结果记录集批量插入指定数据表。

1. 一般 INSERT 语句

INSERT 语句主要用于向表中插入数据。INSERT 语句的语法如下：

```
INSERT INTO [user.]table [@db_link] [(column1[,column2]...)]
VALUES (expressl[,express2]...)
```

其中，table 表示要插入的表名；db_link 表示数据库链接名；columnl，column2 表示表的列名；VALUES 表示给出要插入的值列表。

在 INSERT 语句的使用方式中，最为常用的形式是在 INSERT INTO 子句中指定添加数据的列，并在 VALUES 子句中为各个列提供一个值。

【例 3.34】用 INSERT 语句向 JOBS 表添加一条记录，其结果如图 3.32 所示。

```
insert into jobs(job_id,job_title,min_salary,max_salary)
    values('IT_TEST',测试员',3000.00,8000.00)
```

```
SQL> insert into jobs(job_id,job_title,min_salary,max_salary)
  2  values('IT_TEST','测试员',3000.00,8000.00);

已创建 1 行。
```

图 3.32　例 3.34 的运行结果示意图

在向表的所有列添加数据时，也可以省略 INSERT INTO 子句后的列表清单，使用这种方法时，必须根据表中定义的列的顺序，为所有的列提供数据。可以使用 DESC 命令查看表中定义列的顺序。

【例 3.35】使用 DESC 命令查看 JOBS 表中各列的定义次序，然后省略列表清单向表中添加一行记录，其结果如图 3.33 所示。

```
desc jobs
    insert into jobs values('IT_DBA','数据库管理员',5000.00,15000.00);
```

```
SQL> desc jobs;
 名称                                      是否为空? 类型
 ----------------------------------------- -------- ----------------------------

 JOB_ID                                    NOT NULL VARCHAR2(10)
 JOB_TITLE                                 NOT NULL VARCHAR2(35)
 MIN_SALARY                                         NUMBER(6)
 MAX_SALARY                                         NUMBER(6)

SQL> insert into jobs values('IT_DBA','数据库管理员',5000.00,15000.00);

已创建 1 行。
```

图 3.33　例 3.35 的运行结果示意图

如果上面示例的 VALUES 子句少指定了一个列的值，则在执行时就会收到如下的错误信息：

```
ORA-00947：没有足够的值
```

注意：如果没有按正确的顺序提供各列的插入值，那么插入操作可能失败；也可能插入成功，但在表中则添加了一条错误的数据，这种情况造成的危害更大。因此，推荐使用明确指定插入数据的列名的方式进行 INSERT，它可以有效地避免上述两种错误的发生。

从上例的 DESC 命令的显示结果可以看出，JOB_ID 和 JOB_TITLE 列不能为 NULL，而 MIN_SALARY 和 MAX_SALARY 列则可以接受 NULL 值。也就是说，JOB_ID 和

JOB_TITLE 列被定义了 NOT NULL 约束，在添加数据时，必须为这两个字段提供数据，而其余的两个字段不受此限制。

【例 3.36】建立没有 MAX_SALARY 值的记录，其结果如图 3.34 所示。

```
insert into jobs(job_id,job_title,min_salary)    values('PP_MAN','产品经理',5000.00)；
```

```
SQL> insert into jobs(job_id,job_title,min_salary)
  2  values('PP_MAN','产品经理',5000.00);

已创建 1 行。
```

图 3.34　例 3.36 的运行结果示意图

如果某个列不允许 NULL 值存在，而用户没有为该列提供数据，则会因为违反相应的约束而插入失败。事实上，在定义表的时候为了数据的完整性，常常会为表添加许多完整性约束。例如在 JOBS 表中，为了保证表中每条记录的唯一性，为 JOB_ID 列定义了主键约束。再次尝试将上例运行一次，则因为违反主键约束而失败，运行结果如图 3.35 所示。

```
insert into jobs(job_id,job_title,min_salary)    values('PP_MAN','产品经理',5000.00)；
```

```
SQL> insert into jobs(job_id,job_title,min_salary)
  2  values('PP_MAN','产品经理',5000.00);
insert into jobs(job_id,job_title,min_salary)
*
第 1 行出现错误:
ORA-00001: 违反唯一约束条件 (HR.JOB_ID_PK)
```

图 3.35　由于表的完整性约束插入失败

关于为表定义的完整性约束，将在后面的章节中介绍，这里需要记住的是在向表添加记录时，添加的数据必须符合为表定义的所有完整性约束。

2. 批量 INSERT

SQL 提供了一种成批添加数据的方法，即使用 SELECT 语句替换 VALUES 语句，由 SELECT 语句提供添加的数据，语法如下：

```
INSERT INTO [user.]table [@db_link] [(column1[,column2]...)] Subquery
```

其中，Subquery 是子查询语句，可以是任何合法的 SELECT 语句，其所选列的个数和类型应该与前边的 column 相对应。

【例 3.37】例 3.1 中我们建立了一个名为 IT_EMPLOYEES 的表，下面的示例将从 EMPLOYEES 表提取 department_id 等于“IT”的雇员信息，并保存到 IT_EMPLOYEES 中。

```
insert into IT_EMPLOYEES(
    employee_id,first_name,last_name,email,
    phone_number,job_id,salary,manager_id)
    select em.employee_id,em.first_name,em.1ast_name,em.email,
          em.phone_number,em.job_id,em.salary,em.manager_id
        from employees em,departments dep
        where em.department_id=dep.department_id
        and dep.department_name='IT';
```

上述语句的运行结果如图 3.36 所示。

```
SQL> insert into IT_EMPLOYEES(
  2  employee_id,first_name,last_name,email,phone_number,job_id,salary
  3  ,manager_id)
  4  select em.employee_id,em.first_name,em.last_name,em.email,
  5  em.phone_number,em.job_id,em.salary,em.manager_id
  6  from employees em,departments dep
  7  where em.department_id=dep.department_id
  8  and dep.department_name = 'IT';

已创建5行。
```

图 3.36　例 3.37 的运行结果示意图

从上面的运行结果可以看出，使用 INSERT 和 SELECT 的组合语句一次性为新创建的表添加了 5 行记录。

注意：在使用 INSERT 和 SELECT 的组合语句成批添加数据时，INSERT INTO 指定的列名可以与 SELECT 指定的列名不同，但是其数据类型必须相匹配，即 SELECT 返回的数据必须满足表中列的约束。

3.4.2　UPDATE 语句

当需要修改表中一列或多列的值时，可以使用 UPDATE 语句。使用 UPDATE 语句可以指定要修改的列和修改后的新值，使用 WHERE 子句可以限定被修改的行。使用 UPDATE 语句修改数据的语法形式如下：

```
UPDATE table_name
SET {column1=express1[,column2=express2]
(column1[,column2])=(select query)}
[WHERE condition]
```

其中，各选项含义如下：

- UPDATE 子句用于指定要修改的表名称。需要后跟一个或多个要修改的表名称，这部分是必不可少的。
- SET 子句用于设置要更新的列以及各列的新值。需要后跟一个或多个要修改的表列，这也是必不可少的。
- WHERE 后跟更新限定条件，为可选项。

【例 3.38】使用 UPDATE 语句为所有程序员提高 15%的薪金，其运行结果如图 3.37 所示。

```
update employees
      set salary = salary    * 1.15
      where job_id='IT_PROG';
```

```
SQL> update employees
  2  set salary = salary *1.15
  3  where job_id = 'IT_PROG';

已更新6行。
```

图 3.37　例 3.38 的运行结果示意图

以上使用了 WHERE 子句限定更新薪金的人员为程序员（job_id='IT_PROG'），如果在使用 UPDATE 语句修改表时，未使用 WHERE 子句限定修改的行，则会更新整个表。

同 INSERT 语句一样，可以使用 SELECT 语句的查询结果来实现更新数据。

【例 3.39】使用 UPDATE 语句更新编号为 104 的雇员薪金，调整后的薪金为 IT 程序员的平均薪金。

```
update employees
    set salary=
    (select avg(salary)
        from employees
        where job_id='IT_PROG')
     where employee_id=104;
```

运行上述语句后的结果如图 3.38 所示。

```
SQL> update employees
  2  set salary =
  3  (select avg(salary) from employees where job_id = 'IT_PROG')
  4  where employee_id = 104;

已更新 1 行。
```

图 3.38　例 3.39 的运行结果示意图

注意：在使用 SELECT 语句提供新值时，必须保证 SELECT 语句返回单一的值，否则将会出现错误。

3.4.3　DELETE 语句

数据库向用户提供了添加数据的功能，那么一定也会向用户提供删除数据的功能。从数据库中删除记录可以使用 DELETE 语句来完成。就如同 UPDATE 语句一样，用户也需要规定从中删除记录的表，以及限定表中哪些行将被删除。

```
DALETE FROM table_name
[WHERE condition]
```

其中，关键字 DELETE FROM 后必须要跟准备从中删除数据的表名。

【例 3.40】一个简单的示例，从 IT_EMPLOYEES 表中删除一条记录。

```
delete from it_employees where employee_id=107;
```

上述删除语句的运行结果如图 3.39 所示。

```
SQL> delete from it_employees where employee_id = 107;

已删除 1 行。
```

图 3.39　例 3.40 的运行结果示意图

提示：建议使用 DELETE 语句一定要带上 WHERE 子句，否则将会把表中所有数据全部删除。

3.4.4　TRUNCATE 语句

如果用户确定要删除表中所有的记录，则建议使用 TRUNCATE 语句。使用 TRUNCATE 语句删除数据时，通常要比 DELETE 语句快许多。因为使用 TRUNCATE

语句删除数据时，它不会产生回滚信息，因此执行 TRUNCATE 操作也不能被撤销。

【例 3.41】使用 TRUNCATE 语句删除 IT_EMPLOYEES 表中所有的记录。

```
truncate table it_employees;
select employee_id,last_name from it_employees;
```

运行上述语句后的结果如图 3.40 所示。

```
SQL> truncate table it_employees;
表被截断。
SQL> select employee_id,last_name from it_employees;
未选定行
```

图 3.40　例 41 的运行结果示意图

在 TRUNCATE 语句中还可以使用关键字 REUSE STORAGE，表示删除记录后仍然保存记录占用的空间；与此相反，也可以使用 DROP STORAGE 关键字，表示删除记录后立即回收记录占用的空间。TRUNCATE 语句默认为使用 DROP STORAGE 关键字。使用关键字 REUSE STORAGE 保留删除记录后的空间的 TRUNCATE 语句如下：

```
truncate table it_employees reuse stoage;
```

说明：若使用 DELETE FROM TABLE_NAME 语句，则整个表中的所有记录都将被删除，只剩下一个表格的定义，在这一点上，语句作用的效果和 TRUNCATE TABLE TABLE_NAME 的效果相同。但是 DELETE 语句可以用 ROLLBACK 来恢复数据，而 TRUNCATE 语句则不能。

3.5　数据控制

SQL 定义完整性约束条件的功能主要体现在 CREATE TABLE 语句和 ALTER TABLE 语句中，可以在这些语句中定义主键、取值唯一的列、不允许空值的列、外键（参照完整性）及其他一些约束条件。在 SQL 中，数据控制功能包括事务管理功能和数据保护功能，即数据库的恢复、并发控制、数据库的安全性和完整性控制等。本节将主要介绍 SQL 的安全性控制功能，由于某个用户对某类数据具有何种操作权力是个需求问题而不是技术问题。数据库管理系统的功能是保证这些决定的执行。因此，DBMS 必须具备以下功能：

- 将授权的决定告知系统，这是由 SQL 的 GRANT 和 REVOKE 语句来完成的。
- 将授权的结果存入数据字典。
- 当用户提出操作请求时，根据授权情况进行检查，以决定是否执行操作请求。

3.5.1　GRANT 语句

SQL 用 GRANT 语句向用户授予操作权限，GRANT 语句的一般格式为：

```
GRANT <权限>[，<权限>]…
[ON<对象类型><对象名>]
```

```
TO<用户>[，<用户>]…
[WITH GRANT OPTION]
```

提示：上述语句的语义即将指定操作对象的指定操作权限授予指定的用户。

对于不同类型的操作对象有不同的操作权限，对属性列和视图的操作权限包括查询（SELECT）、插入（INSERT）、修改（UPDATE）、删除（DELETE）以及这 4 种权限的总和（ALLPRIVILEGES）。对基表的操作权限包括查询、插入、修改、删除、修改表（ALTER）和建立索引（INDEX）以及这六种权限的总和。对数据库可以有建立表（CREATETAB）的权限，该权限属于 DBA，可由 DBA 授予普通用户，普通用户拥有此权限后可以建立基表，基表的所有者（Owner）拥有对该表的一切操作权限。

常见的操作权限如表 3.4 所示。

表 3.4　不同对象类型允许的操作权限

对象	对象类型	操作权限
属性列	TABLE COLUMN	SELECT、INSERT、UPDATE、DELETE、ALL PRIVILEGES
视图	TABLE VIEW	SELECT、INSERT、UPDATE、DELETE、ALL PRIVILEGES
基表	TABLE	SELECT、INSERT、UPDATE、DELETE、ALTER、INDEX、ALL PRIVILEGES
数据库	DATABASE	CREATETAB

接受权限的用户可以是一个或多个具体用户，也可以是 PUBLIC，即全体用户。如果指定了 WITH GRANT OPTION 子句，则获得某种权限的用户还可以把这种权限再授予其他的用户。如果没有指定 WITH GRANT OPTION 子句，则获得某种权限的用户只能使用该权限，但不能传播该权限。

以下将通过几个例子来说明 GRANT 语句的使用，由于以下例子中的用户 User1 至 User8 均为用户示意，故不再给出结果示意图，读者可自行创建用户演练。

【例 3.42】把查询 IT_EMPLOYEES 表的权限授给用户 User1。

```
GRANT SELECT
    ON TABLE IT_EMPLOYEES
    TO User1;
```

【例 3.43】把对 IT_EMPLOYEES 表和 JOBS 表的全部操作权限授予用户 User2 和 User3。

```
GRANT ALL PRIVILEGES
    ON TABLE IT_EMPLOYEES,JOBS
    TO User2,User3;
```

【例 3.44】把对表 DEPARTMENT 的查询权限授予所有用户。

```
GRANT SELECT
    ON TABLE DEPARTMENT
    TO PUBLIC;
```

【例 3.45】把查询 IT_EMPLOYEES 表和修改雇员编号的权限授给用户 User4。

```
GRANT UPDATE(EMPLOYEE_ID),SELECT
```

```
    ON TABLE IT_EMPLOYEES TO User4;
```

这里实际上要授予 User4 用户的是对基表 IT_EMPLOYEES 的 SELECT 权限和对属性列 EMPLOYEE_ID 的 UPDATE 权限。授予关于属性列的权限时必须明确指出相应属性列名。

【例 3.46】把对表 DEPARTMENT 的 INSERT 权限授予 User5 用户，并允许将此权限再授予其他用户。

```
GRANT INSERT
    ON TABLE DEPARTMENT
    TO User5 WITH GRANT OPTION;
```

执行此 SQL 语句后，User5 不仅拥有了对表 DEPARTMENT 的 INSERT 权限，还可以传播此权限，即由 User5 用户使用上述 GRANT 命令给其他用户授权。

【例 3.47】User5 将此权限授予 User6。

```
GRANT INSERT
    ON TABLE DEPARTMENT
    TO User6 WITH GRANT OPTION;
```

【例 3.48】User6 将此权限授予 User7。

```
GRANT INSERT
    ON TABLE DEPARTMENT
    TO User7;
```

因为 User6 未给 User7 传播的权限，因此 User7 不能再传播此权限。

【例 3.49】DBA 把在数据库 DB_EMPLOYEES 中建立表的权限授予用户 User8。

```
GRANT CREATETAB
    ON DATABASE DB_EMPLOYEES
    TO User8;
```

由上面的例子可以看到，GRANT 语句可以一次向一个用户授权，如例 3.42 所示，这是最简单的一种授权操作；也可以一次向多个用户授权，如例 3.43、例 3.44 等所示；还可以一次传播多个同类对象的权限，如例 3.43 所示；甚至一次可以完成对基表、视图和属性列这些不同对象的授权，如例 3.45 所示。

注意：授予关于 DATABASE 的权限必须与授予关于 TABLE 的权限分开，这是因为对象类型不同。

3.5.2 REVOKE 语句

授予的权限可以由 DBA 或其他授权者用 REVOKE 语句收回，REVOKE 语句的一般格式为：

```
REVOKE<权限>[，<权限>]…
[ON  <对象类型><对象名>]
FROM<用户> [,<用户>]…;
```

【例 3.50】把用户 User4 修改雇员编号的权限收回。

```
REVOKE UPDATE(EMPLOYEE_ID)
```

```
    ON TABLE IT_EMPLOYEES
    FROM User4;
```

【例 3.51】收回所有用户对表 DEPARTMENT 的查询权限。

```
REVOKE SELECT
    ON TABLE DEPARTMENT
    FROM PUBLIC;
```

【例 3.52】把用户 User5 对 DEPARTMENT 表的 INSERT 权限收回。

```
REVOKE INSERT
    ON TABLE DEPARTMENT
    FROM User5;
```

在例 3.47 中，User5 将对 DEPARTMENT 表的 INSERT 权限授予了 User6，而 User6 又将其授予了 User7。执行例 3.52 的 REVOKE 语句后，DBMS 在收回 User5 对 DEPARTMENT 表的 INSERT 权限的同时，还会自动收回 User6 和 User7 对 DEPARTMENT 表的 INSERT 权限。也就是说，收回权限的操作会级联下去。但如果 User6 或 User7 还从其他用户处获得对 DEPARTMENT 表的 INSERT 权限，则他们仍具有此权限，系统只收回直接或间接从 User5 处获得的权限。

可见，SQL 提供了非常灵活的授权机制，DBA 拥有对数据库中所有对象的所有权限，并可以根据应用的需要将不同的权限授予不同的用户。用户对自己建立的基表和视图拥有全部的操作权限，并且可以用 GRANT 语句把其中某些权限授予其他用户。被授权的用户如果有“继续授权”的许可，还可以把获得的权限再授予其他用户。所有授予出去的权力在必要时又都可以用 REVOKE 语句收回。

3.6 Oracle 常用函数

在 SQL 乃至 SQL 编程中，经常会使用到 DBMS 提供的函数来完成用户需要的功能。针对不同的 DBMS 系统，提供的函数都不尽相同，本小节将对 Oracle 中的一些常用函数进行介绍，如字符类函数、数字类函数、日期类函数、转换类函数、聚集类函数以及其他函数等。

以下主要通过对字符类函数的详细说明来使读者对 Oracle 函数建立一个印象，其他类的函数通过列表的形式给出，读者可以自行演练。

3.6.1 字符类函数

字符类函数是专门用于字符处理的函数，处理的对象可以是字符串常数，也可以是字符类型的列。常用的字符函数如下所示。

1. ASCII(<c1>)

该函数用于返回 c1 第一个字母的 ASCII 码，其中 c1 是字符串。它的逆函数是 CHR()。

【例 3.53】ASCII 函数示例。

```
select ASCII('A') BIG_A, ASCII('a')   SMALL_A   FROM    dual;
```

上述查询语句的运行结果如图 3.41 所示。

```
SQL> select ascii('A') big_a , ascii('a') small_a from dual;

    BIG_A    SMALL_A
--------- ----------
       65         97
```

图 3.41 例 3.53 的运行结果示意图

2. CHR(<i>)

该函数用于求 i 对应的 ASCII 字符，其中 i 是一个数字。

【例 3.54】CHR 函数示例。

```
select CHR(65),CHR(97) FROM dual;
```

上述查询语句的运行结果如图 3.42 所示。

```
SQL> select chr(65),chr(97) from dual;

C C
- -
A a
```

图 3.42 例 3.54 的运行结果示意图

3. CONCAT(cl,c2)

该函数将 c2 连接到 c1 的后面，如果 cl 为 null，将返回 c2；如果 c2 为 null，则返回 c1；如果 c1、c2 都为 null，则返回 null。其中，c1、c2 均为字符串，它和操作符“||”返回的结果相同。

【例 3.55】CONCAT 函数示例。

```
select concat('oracle ','11g') name from dual;
```

上述查询语句的运行结果如图 3.43 所示。

```
SQL> select concat('Oracle ','11g') name from dual;

NAME
-----------
Oracle 11g
```

图 3.43 例 3.55 的运行结果示意图

4. INITCAP(c1)

该函数将 c1 中每个单词的第一个字母大写，其他字母小写返回。单词由空格、控制字符、标点符号限制。其中 c1 为字符串。

【例 3.56】INITCAP 函数示例。

```
select INITCAP('oracle universal installer') name from dual;
```

上述查询语句的运行结果如图 3.44 所示。

```
SQL> select initcap('oracle universal installer') name from dual;

NAME
--------------------------
Oracle Universal Installer
```

图 3.44 例 3.56 的运行结果示意图

5. INSTR(cl,[c2,<i>[,j]])

该函数用于返回 c2 在 c1 中第 j 次出现的位置，搜索从 c1 的第 i 个字符开始。当没有发现需要的字符时返回 0，如果 i 为负数，那么搜索将从右到左进行，但是位置还是按从左到右来计算，i 和 j 的默认值为 1。其中，c1、c2 均为字符串，i、j 为整数。

【例 3.57】INSTR 函数示例 1。

```
select INSTR('Moisossoppo','o',3,3) from dual;
```

【例 3.58】INSTR 函数示例 2。

```
select INSTR('Moisossoppo','o',-2,3)from dual;
```

提示：INSTRB(cl,[c2,<i>[,j]])与 INSTR()函数一样，只是其返回的是字节，对于单字节 INSTRB()等于 INSTR()。

上述查询语句的运行结果如图 3.45 所示。

```
SQL> select instr('Moisossoppo','o',3,3) from dual;

INSTR('MOISOSSOPPO','O',3,3)
----------------------------
                          11

SQL> select instr('Moisossoppo','o',-2,3) from dual;

INSTR('MOISOSSOPPO','O',-2,3)
-----------------------------
                            2
```

图 3.45　例 3.57、例 3.58 的运行结果示意图

6. LENGTH(c1)

该函数用于返回 c1 的长度，如果 c1 为 null，那么将返回 null 值。其中 c1 为字符串。

【例 3.59】LENGTH 函数示例：

```
select LENGTH('Oracle 11g') name from dual;
```

上述查询语句的运行结果如图 3.46 所示。

```
SQL> select length('Oracle 11g')  name from dual;

      NAME
----------
        10
```

图 3.46　例 3.59 的运行结果示意图

7. LOWER(c1)

该函数用于返回 c1 的小写字符，经常出现在 WHERE 子串中。

【例 3.60】LOWER 函数示例。

```
select LOWER(job_id) from JOBS WHERE LOWER(job_id) LIKE 'it%';
```

上述查询语句的运行结果如图 3.47 所示。

```
SQL> select lower(job_id),job_title from jobs where lower(job_id) like 'it%';

LOWER(JOB_ JOB_TITLE
---------- ----------------------------------
it_prog    Programmer
it_test    测试员
it_dba     数据库管理员
```

图 3.47 例 3.60 的运行结果示意图

8. LTRIM(cl,c2)

该函数表示将 c1 中最左边的字符去掉，使其第一个字符不在 c2 中，如果没有 c2，那么 c1 就不会改变。

【例 3.61】LTRIM 函数示例。

```
select LTRIM( 'Moisossoppo','Mois')from dual;
```

上述查询语句的运行结果如图 3.48 所示。

```
SQL> select ltrim('Moisossoppo','Mois') from dual;

LTR
---
ppo
```

图 3.48 例 3.61 的运行结果示意图

9. REPLACE(c1,c2[,c3])

该函数用 c3 代替出现在 cl 中的 c2 后返回，其中 cl、c2、c3 都是字符串。

【例 3.62】REPLACE 函数示例。

```
select REPLACE('feelblue','blue','yellow') from dual;
```

上述查询语句的运行结果如图 3.49 所示。

```
SQL> select replace('feelblue','blue','yellow') from dual;

REPLACE('F
----------
feelyellow
```

图 3.49 例 3.62 的运行结果示意图

10. SUBSTR(c1,<i>[,j])

该函数表示从 cl 的第 i 位开始返回长度为 j 的子字符串，如果 j 为空，则直到串的尾部。其中，cl 为一字符串，i、j 为整数。

【例 3.63】SUBSTR 函数示例：

```
select SUBSTR('Message',1,4)from dual;
```

上述查询语句的运行结果如图 3.50 所示。

```
SQL> select substr('Message',1,4) from dual;

SUBS
----
Mess
```

图 3.50 例 3.63 的运行结果示意图

提示：函数 SUBSTRB(c1,<i>[,j])与 SUBSTR 大致相同，只是 i,j 是以字节计算。此外，函数 TRIM(c1)用于将 c1 的前后空格删除，函数 UPPER(c1)用于返回 c1 的大写字母。

3.6.2 数字类函数

数字函数操作数字数据，执行数学和算术运算。所有函数都有数字参数并返回数字值。需要注意的是所有三角函数的操作数和值都是弧度而不是角度。同时，在 Oracle 中并没有提供内建的弧度和角度的转换函数。

数字类函数的介绍如表 3.5 所示。

表 3.5 数字类函数介绍

序号	函数	意义
01	ABS(n)	用于返回 n 的绝对值
02	ACOS(n)	反余弦函数，用于返回–1～l 之间的数，n 表示弧度
03	ASIN(n)	反正弦函数，用于返回–1～1 之间的数，n 表示弧度
04	ATAN(n)	反正切函数，用于返回 n 的反正切值，n 表示弧度
05	CEIL(n)	用于返回大于或等于 n 的最小整数
06	COS(n)	用于返回 n 的余弦值，n 为弧度
07	COSH(n)	用于返回 n 的双曲余弦值，n 为数字
08	EXP(n)	用于返回 e 的 n 次幂，e=2.71828183
09	FLOOR(n)	用于返回小于等于 n 的最大整数
10	LN(n)	用于返回 n 的自然对数，n 必须大于 0
11	LOG(n1,n2)	用于返回以 n1 为底 n2 的对数
12	MOD(n1,n2)	用于返回 n1 除以 n2 的余数
13	POWER(n1,n2)	用于返回 n1 的 n2 次方
14	ROUND(n1,n2)	用于返回舍入小数点右边 n2 位的 n1 的值，n2 的默认值为 0，这会返回小数点最接近的整数，如果 n2 为负数就舍入到小数点左边相应的位上，n2 必须是整数
15	SIGN()	若 n 为负数，则返回–1，若 n 为正数，则返回 1，若 n=0，则返回 0
16	SIN(n)	用于返回 n 的正弦值，n 为弧度
17	SINH(n)	用于返回 n 的双曲正弦值，n 为弧度
18	SQRT(n)	用于返回 n 的平方根，n 为弧度
19	TAN(n)	用于返回 n 的正切值，n 为弧度
20	TANH(n)	用于返回 n 的双曲正切值，n 为弧度
21	TRUNC(n1,n2)	用于返回截尾到 n2 位小数的 n1 的值，n2 默认设置为 0，当 n2 为默认设置时会将 n1 截尾为整数，如果 n2 为负值，就截尾在小数点左边相应的位上

3.6.3 日期类函数

日期函数操作 DATE 数据类型，绝大多数都有 DATE 数据类型的参数，且其返回值也大都为 DATE 数据类型。

日期类函数的介绍如表 3.6 所示。

表 3.6 日期类函数

序号	函数	意义
01	ADD_MONTHS(d,<i>)	返回日期 d 加上 i 个月后的结果。其中，i 为任意整数。若 i 是一个小数，则数据库将隐式地将其转换成整数，并截去小数点后面的部分
02	LAST_DAY(d)	返回包含日期 d 月份的最后一天
03	MONTHS_BETWEEN(d1,d2)	返回 d1 和 d2 之间月的数目，若 d1 和 d2 的日期都相同，或者都是该月的最后一天，则返回一个整数，否则返回的结果将包含一个分数
04	NEW_TIME(d1,tz1,tz2)	其中，d1 是一个日期数据类型，当时区 tzl 中的日期和时间是 d1 时，返回时区 tz2 中的日期和时间。tzl 和 tz2 是字符串
05	SYSDATE	返回当前日期和时间，该函数没有参数

3.6.4 转换类函数

转换函数用于操作多数据类型，在数据类型之间进行转换。在使用 SQL 语句进行数据操作时，经常使用到这一类函数。

转换类函数的介绍如表 3.7 所示。

表 3.7 转换类函数

序号	函数	意义
01	CHARTORWID(c1)	该函数将 c1 转换为 RWID 数据类型，其中 c1 是一个字符串
02	CONVERT(c1,dset[,sset])	该函数将字符串 c1 由 sset 字符集转换为 dset 字符集，sset 默认设置为数据库的字符集，其中 c1 为字符串，dset、sset 是两个字符集
03	ROWIDTOCHAR()	该函数将 ROWID 数据类型转换为 CHAR 数据类型
04	TO_CHAR(x[,fmt[nlsparm,]])	该函数将 x 转换为字符串
05	TO_DATE(c1[,fmt[nlsparm,]])	该函数将字符串 c1 转换成 date 数据类型，其中 c1 表示字符串，fmt 表示一种特殊格式的字符串。返回按照 fmt 格式显示的 c，nlsparm 表示返回的月份和日期所使用的语言
06	TO_MULTI_BYTE(c1)	该函数将 c 的单字节字符转换成多字节字符，其中 c1 表示一个字符串

续表

序号	函数	意义
07	TO_NUMBER(c1[,fmt[nlsparm,]])	函数将返回 c1 代表的数字，返回值将按照 fmt 指定的格式显示。其中，c1 表示字符串；fmt 表示一个特殊格式的字符串；nlsparm 表示语言
08	TO_SINGLE_BYTE(c1)	将字符串 c1 中的多字节字符转换成等价的单字节字符。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

3.6.5 聚集类函数

聚集类函数也称为集合函数，返回基于多个行的单一结果，行的准确数量无法确定，除非查询被执行并且所有的结果都被包含在内。与单行函数不同的是，在解析时所有的行都是已知的。由于这种差别使聚集类函数与单行函数在要求和行为上有微小的差异。

Oracle 提供了丰富的聚集类函数。这些函数可以在 SELECT 或 SELECT 的 HAVING 子句中使用，当用于 SELECT 子句时常常与 GROUP BY 一起使用。

表 3.8 聚集类函数

序号	函数	意义
01	AVG(x[{DISTINCT|ALL}])	用于返回数值的平均值。默认设置为 ALL
02	COUNT(x[{DISTINCT|ALL}])	用于返回查询中行的数目，默认设置是 ALL，表示返回所有的行
03	MAX(x[{DISTINCT|ALL}])	用于返回选择列表项目的最大值，若 x 是字符串数据类型，则返回一个 VARCHAR2 数据类型；若 x 是一个 DATA 数据类型，则返回一个日期；若 x 是 NUMERIC 数据类型，则返回一个数字。注意 DISTINCT 和 ALL 不起作用，因为最大值与这两种设置是相同的
04	MIN(x[{DISTINCT|ALL}])	用于返回选择列表项目的最小值
05	STDDEV(x[{DISTINCT]ALL}])	用于返回选择列表项目的标准差，所谓标准差是方差的平方根
06	SUM(x[{DISTINCT|ALL})	用于返回选择列表项目的数值的总和
07	VARIANCE(x[{DISTINCTIALL}])	用于返回选择列表项目的统计方差

说明：在 Oracle 基于数据仓库的经营分析系统中，还使用了大量的专门用于分析的分析函数，在本文中不再进行介绍。