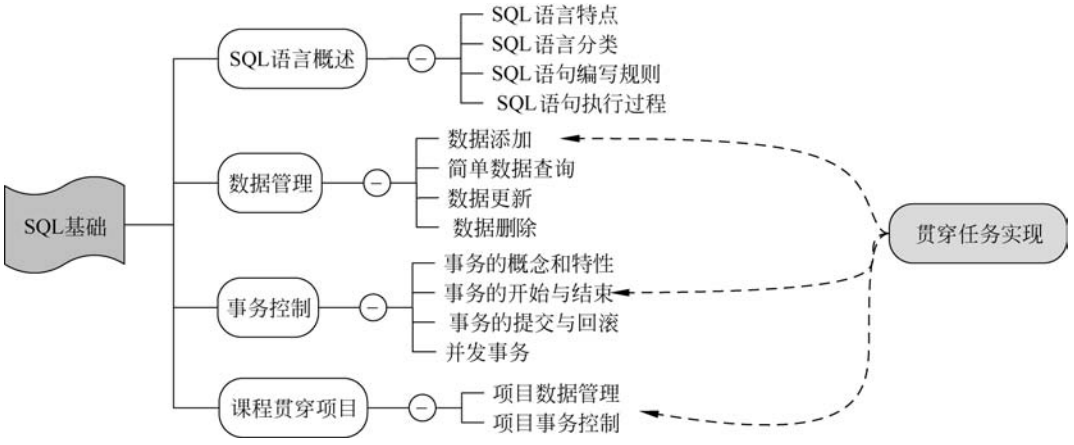
第5章 SQL基础

任务驱动

本章任务完成 Q_MicroChat 微聊项目的数据管理及事务控制功能。具体任务分解如下：

- 【任务 5-1】 项目数据管理
- 【任务 5-2】 项目事务控制

学习导航/课程定位



本章目标

知 识 点	Listen(听)	Know(懂)	Do(做)	Revise(复习)	Master(精通)
SQL 语言分类	★	★	★	★	★
SQL 语言编写规则	★	★	★	★	★
SQL 语句执行过程	★	★	★	★	★
数据管理	★	★	★	★	★
事务控制	★	★	★	★	★
并发事务	★	★	★	★	

5.1 SQL 语言概述

SQL(Structured Query Language,结构化查询语言)是关系数据库的标准语言,是Oracle 数据库中定义和操作数据的基本语言,是用户与数据库之间交互的接口。

SQL 语言于 1974 年由 Boyce 和 Chamberlin 提出,并在 IBM 公司研制的关系数据库原型系统 System R 上实现。1986 年 10 月美国国家标准学会颁布了 SQL 语言的美国标准,1987 年 6 月国际标准化组织将其采纳为国际标准,SQL 语言成为关系数据库的标准语言。随着数据库技术的发展,SQL 标准也在不断地进行扩展和修正,国际标准化组织(ISO)相继颁布了 SQL-89 标准、SQL-92 标准和 SQL-99 标准。Oracle 数据库完全遵循 SQL 标准,将最新的 SQL-99 标准集成到了 Oracle 产品中,并进行了部分功能扩展。

SQL 语言是关系数据库操作的基础语言,将数据查询、数据操纵、数据定义、事务控制、系统控制等功能集于一体,使得数据库应用开发人员、数据库管理员等都可以通过 SQL 语言实现对数据库的访问和操作。

5.1.1 SQL 语言特点

SQL 语言之所以能成为关系数据库的标准语言,并得到广泛应用,主要在于 SQL 语言具有以下特点。

- SQL 是一种一体化的语言。尽管设计 SQL 的最初目的是查询,数据查询也是其最重要的功能之一,但 SQL 绝不仅仅是一个查询工具,它集数据定义、数据查询、数据操纵和数据控制功能于一体,可以独立完成数据库的全部操作。
- SQL 是一种高度的非过程化的语言。执行 SQL 语句时,用户只需要知道其逻辑含义,而不需要关心 SQL 语句的具体执行步骤,这大大降低了对用户的技术要求。
- SQL 采用集合的操作方式,对数据的处理是成组进行的,不仅操作对象、查找结果可以是记录集合,而且一次插入、删除、更新操作的对象也可以是记录的集合。通过使用集合操作方式,极大地提高了对数据操作的效率。
- SQL 使用方式多样,既可以直接以命令方式交互使用,也可以将 SQL 语句嵌入高级语言中执行。
- SQL 非常简洁。SQL 语言命令数量有限,语法简单,接近于英语自然语言,容易学习和掌握。

5.1.2 SQL 语言分类

根据 SQL 语言实现功能的不同,Oracle 数据库中的 SQL 语言可以分为以下 6 类。

- 数据定义语言(Data Definition Language,DDL):用于定义、修改、删除数据库模式对象、进行权限管理等。它包括创建模式对象语句 CREATE、修改模式对象语句 ALTER、删除模式对象语句 DROP、重命名模式对象语句 RENAME、删除表中所有行但不删除表的语句 TRUNCATE、管理权限的语句 GRANT 和 REVOKE、审核数据库语句 AUDIT 和 NOAUDIT、为表或列添加注释的语句 COMMENT 等。

- 数据查询语言(Data Query Language,DQL):用于数据检索,包括数据检索语句SELECT。
- 数据操纵语言(Data Manipulation Language,DML):用于改变数据库中的数据。它包括数据插入语句INSERT、数据修改语句UPDATE、数据删除语句DELETE等。
- 事务控制语言(Transaction Control Language,TCL):用于将一组DML操作组合起来,形成一个事务并进行事务控制。它包括事务提交语句COMMIT、事务回滚语句ROLLBACK、设置保存点语句SAVEPOINT和设置事务状态语句SET TRANSACTION。
- 系统控制语言(System Control Language):用于设置数据库系统参数,只有一条语句ALTER SYSTEM。
- 会话控制语言(Session Control Language):用于设置用户会话相关参数。它包括设置会话属性语句ALTER SESSION、切换角色语句SET ROLE。

本章将主要介绍数据查询、数据操纵和事务控制的SQL语句在Oracle数据库中的应用。

5.1.3 SQL语句编写规则

SQL语句在编写时具有以下特性:

- 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。
- 对象名和列名不区分大小写,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。
- 字符值和日期值区分大小写。
- SQL语句以分号结束。在编写时如果SQL语句很长,可以使用回车和缩进将语句分布到多行,从而提高语句的可读性。

下述示例中的SQL语句均符合SQL语言的编写规则。

【示例】 SQL语言的编写规则1。

```
SQL> select empno,ename,sal from emp where lower(ename) = 'smith';
```

【示例】 SQL语言的编写规则2。

```
SQL> Select EmpNO,EnaMe,Sal
FroM Emp
Where Lower(ENAME) = 'smith';
```

通过上述示例可以看出,SQL这种不区分大小写的编写规则,虽然灵活方便,但同时会使写出来的代码变得毫无规律、不易阅读。另外,由于SQL语句在执行期间时,会首先检查系统全局区SGA中是否已缓存完全相同的SQL语句,如果没有,则将此SQL语句存入共享池中,因此如果SQL语句没有按统一的编写规则编写,还将导致共享池中缓存的SQL语句不能重复使用而造成性能的损失。为此Oracle建议用户按照以下大小写规则编写代码:

- SQL关键字使用大写字母,如SELECT、INTO、UPDATE、WHERE等。

- 预定义类型使用大写字母，如 NUMBER、VARCHAR2、BOOLEAN、DATE 等。
- 内置函数使用大写字母，如 SUBSTR、COUNT、TO_CHAR 等。
- 数据库对象使用小写字母，如数据库表名、列名、视图名等。
- PL/SQL 保留字使用大写字母，如 BEGIN、DECLARE、LOOP、ELSEIF 等。
- 用户声明的标识符使用小写字母。

按照上述 SQL 语言编写规则，规范后的示例代码如下。

【示例】

```
SQL > SELECT empno,ename,sal
FROM emp
WHERE LOWER(ename) = 'smith';
```

在实际开发中，通过设计项目中 SQL 的代码规则并遵守这些规则，使待被处理的语句与共享池中的相一致，将会大大提高项目的开发效率以及提升数据库对 SQL 语句的执行性能。

5.1.4 SQL 语句执行过程

用户执行 SQL 语句时，通过数据库连接，先将该语句发送到 Oracle 服务器，再由服务器进程处理该语句。Oracle 服务器进程处理 SQL 语句的过程包括：语句解析、语句执行、返回结果。

1. 语句解析

服务器进程在接到客户端传送过来的 SQL 语句时，不会直接去数据库查询，而是会先在数据库的高速缓存区中查找是否存在相同的 SQL 语句及其执行计划。如果存在，则服务器进程会直接执行此 SQL 语句；如果不存在，则会依次执行对该 SQL 语句的语法及语义的检查、给对象加解析锁、访问权限的核对等操作。此过程中，如果语法不正确，就将语法错误消息返回给客户机；如果语法正确，就通过查询数据字典，检查该 SQL 语句的语义，以确定表名、列名是否正确。如果表名和列名不正确，就将语义错误消息返回给客户机。当语法、语义都正确后，系统就会给所操作的对象加解析锁，以防止在解析期间其他用户改变这些对象的结构或删除这些对象。接下来，服务器进程还会检查当前操作用户是否具有访问该对象的相应权限，如果没有相应权限，会将权限不足的消息返回给客户机；如果具有相应的权限，则由 SQL 语句的优化器来确定该 SQL 语句的最佳执行计划，为该 SQL 语句在数据高速缓存区中分配空间，将该 SQL 语句及其执行计划装入其中，以便执行。具体语句解析过程如图 5-1 所示。

2. 语句执行

语句执行指服务器进程按照 SQL 语句的执行计划执行 SQL 语句。在此期间，服务器进程执行如下操作：

- 确定被操纵对象的数据所在的数据块是否已经被读取到 SGA 区的数据高速缓存区中了，如果数据块在数据高速缓存中，则直接在其中操作。

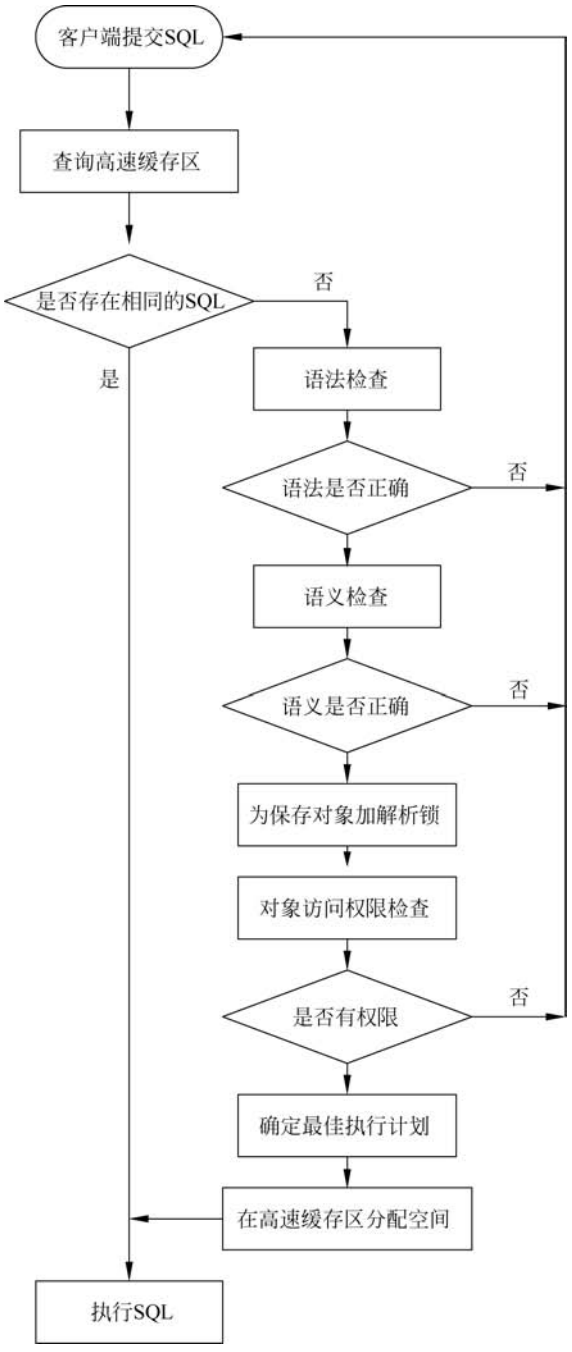


图 5-1　SQL 语句解析过程

- 如果数据块不在数据高速缓存中，则从数据文件所对应的物理存储设备中读取该数据块，并在数据高速缓存中寻找空闲数据块，将读入的数据放入。
- 对于 UPDATE 和 DELETE 语句，将需要修改或删除的行锁住，以便在事务结束之前相同的行不会被其他进程修改。对于 SELECT 和 INSERT 语句，因为不会修改数据，所以不需要锁住行。

3. 返回结果

语句执行完成之后，查询到的数据还是在服务器进程中，还没有被传送到客户端的用户进程。所以，在服务器端的进程中，有专门负责数据提取的代码，它的作用就是把查询到的数据结果返回给用户端进程，从而完成整个查询动作。

对于 DBA 或者基于 Oracle 数据库的开发人员，如果能够了解整个 SQL 语句的处理过程，那么在数据库应用开发中对于涉及的 SQL 语句的开发与调试是非常有帮助的，可以大大减少排错的时间。

5.2 数据管理

使用数据操纵语言(DML)和数据查询语言(DQL)可以对表中数据进行管理，包括数据的添加、数据的查询、数据的更新以及数据的删除等操作。

5.2.1 数据添加

在 Oracle 数据库中，使用 INSERT 语句向表中插入数据。数据的插入方式主要有两种，一种是每次插入单行记录；另一种是利用子查询插入查询到的结果集。

1. 插入单行记录

INSERT INTO...VALUES 语句用于向表中插入单行记录，语法如下。

【语法】

```
INSERT INTO table_name [(column1[,column2...])]
VALUES(value1[,value2,...]);
```

其中：

- 如果 INTO 子句中没有指明任何列名，则 VALUES 子句中列值的个数、顺序、类型必须与表中列的个数、顺序、类型相匹配。
- 如果在 INTO 子句中指明了列名，则 VALUES 子句中提供的列值的个数、顺序、类型必须与指定列的个数、顺序、类型按位置对应。

本小节将以下述示例创建的部门表和员工表为例，进行数据管理的讲解。

【示例】 创建部门表 department 和员工表 employee。

```
SQL> CREATE TABLE department(
  2  deptno NUMBER(2) PRIMARY KEY,
  3  dname VARCHAR2(14),
  4  loc VARCHAR2(13)
  5  )
  6  TABLESPACE USERS;
表已创建。
SQL> CREATE TABLE employee(
  2  empno NUMBER(4) PRIMARY KEY,
```

```
  3  ename VARCHAR2(10),
  4  job VARCHAR2(20),
  5  mgr NUMBER(4),
  6  hiredate DATE,
  7  sal NUMBER(7,2),
  8  comm NUMBER(7,2),
  9  deptno NUMBER(2) CONSTRAINT fk_emp_dept REFERENCES department(deptno)
 10  )
 11  TABLESPACE USERS;
表已创建。
```

对表进行数据添加时，如果插入的数据为字符型或日期型，需要加单引号，对于日期型数据需要按系统默认格式输入或使用 TO_DATE()函数进行日期转换。如下述示例所示。

【示例】 向部门表插入一行记录。

```
SQL> INSERT INTO department(deptno,dname,loc) VALUES(50,'研发部','青岛');
已创建 1 行。
```

【示例】 向员工表插入一条记录。

```
SQL> INSERT INTO employee VALUES(
  2  7210,'Jenny','PROGRAMMER',null,TO_DATE('20150302','yyyy-MM-dd'),
  3  3000,null,50);
已创建 1 行。
```

在进行记录添加时，需要注意记录插入后不能违反被插入表的完整性约束，如主键不能重复、引用的外键必须存在等，否则系统会报完整性约束异常，如下述示例所示。

【示例】 向员工表插入一条记录，部门号为一个不存在的编号。

```
SQL> INSERT INTO employee VALUES(
  2  7211,'Fanny','PROGRAMMER',7210,TO_DATE('20140522','yyyy-MM-dd'),
  3  2000,500,60);
INSERT INTO employee VALUES(
*
第 1 行出现错误:
ORA-02291: 违反完整约束条件 (TEST.FK_EMP_DEPT) - 未找到父项关键字
```

2. 利用子查询插入数据

利用子查询可以将子查询的结果集一次插入一个表中，语法如下。

【语法】

```
INSERT INTO table_name [(column1[,column2...])] subquery;
```

其中：

- 子查询中列的个数、顺序和类型必须与 INTO 子句中指定的列的个数、顺序和类型相匹配。

【示例】 将 scott 模式的 dept 表的记录插入 department 表中。

```
SQL> INSERT INTO department SELECT * FROM scott.dept;
已创建 4 行。
```

【示例】 将 scott 模式的 emp 表的记录插入 employee 表中。

```
SQL> INSERT INTO employee SELECT * FROM scott.emp;
已创建 12 行。
```

在上述示例中,除了要求子查询的结果记录插入后不能违反被插入表的完整性约束外,还需要操作用户具有访问 SCOTT 模式对象的权限。

5.2.2 简单数据查询

数据查询是 SQL 语言中最常用,也是最复杂的操作。本节先介绍使用 SELECT 语句对表数据进行简单查询,较复杂的查询请查看本书第 6 章。

SELECT 语句进行简单数据查询的语法如下。

【语法】

```
SELECT * | column_name [,column_name2...]
FROM table_name
[WHERE condition];
```

其中:

- "*"表示查询该表中的所有列。
- WHERE 用于指定查询条件。如果不省略,则在执行 SELECT 语句时,系统会首先根据 WHERE 子句的条件表达式 condition 从 FROM 子句指定的基本表中查找满足条件的记录,再按 SELECT 子句中的目标列或目标表达式形成结果表。

【示例】 查询部门表中所有列的记录。

```
SQL> SELECT * FROM department;
    DEPTNO   DNAME                        LOC
----------  ---------------------  ---------------
        50   研发部                    青岛
        10   ACCOUNTING               NEW YORK
        20   RESEARCH                 DALLAS
        30   SALES                     CHICAGO
        40   OPERATIONS               BOSTON
```

【示例】 查询符合条件的员工的部分列的信息。

```
SQL> SELECT empno,ename,job,sal,comm
  2  FROM employee
  3  WHERE comm IS NULL;
    EMPNO ENAME                JOB              SAL             COMM
----------------------  -------------  --------------  -------------
     7210 Jenny             PROGRAMMER       3000
```

```
      7369 SMITH            CLERK         800
      7566 JONES            MANAGER       2975
      7698 BLAKE            MANAGER       2850
      7782 CLARK            MANAGER       2450
      7839 KING             PRESIDENT     5000
      7900 JAMES            CLERK         950
      7902 FORD             ANALYST       3000
      7934 MILLER           CLERK         1300
已选择 9 行。
```

5.2.3 数据更新

表中数据的修改使用 UPDATE 语句，可以一次修改一条记录，也可以一次修改多条记录，语法如下。

【语法】

```
UPDATE table_name
SET column1 = value1[,column2 = value2...]
[WHERE condition];
```

【示例】 将员工表中所有员工的奖金更新为 200 元。

```
SQL> UPDATE employee SET comm = 200;
已更新 13 行。
```

【示例】 将工号为 7210 的员工的工资更新为原有工资与奖金的和。

```
SQL> UPDATE employee SET sal = sal + comm WHERE empno = 7210;
已更新 1 行。
SQL> SELECT empno,ename,job,sal,comm FROM employee;
     EMPNO ENAME                JOB          SAL          COMM
---------- -------------------- ---------- ---------- ----------
      7210 Jenny                PROGRAMMER   3200         200
      7369 SMITH                CLERK        800          200
      7499 ALLEN                SALESMAN     1600         200
      7521 WARD                 SALESMAN     1250         200
      7566 JONES                MANAGER      2975         200
      7654 MARTIN               SALESMAN     1250         200
      7698 BLAKE                MANAGER      2850         200
      7782 CLARK                MANAGER      2450         200
      7839 KING                 PRESIDENT    5000         200
      7844 TURNER               SALESMAN     1500         200
      7900 JAMES                CLERK        950          200
      7902 FORD                 ANALYST      3000         200
      7934 MILLER               CLERK        1300         200
已选择 13 行。
```

5.2.4 数据删除

DELETE 语句用于删除数据库中的一条或多条记录，语法如下。

【语法】

```
DELETE FROM table_name
[WHERE condition];
```

【示例】 依据条件子句删除记录。

```
SQL> DELETE FROM employee WHERE empno = 7210;
已删除 1 行。
```

【示例】 删除表中所有记录。

```
SQL> DELETE FROM department;
DELETE FROM department
*
第 1 行出现错误:
ORA-02292: 违反完整约束条件 (TEST.FK_EMP_DEPT) - 已找到子记录
```

在上述示例中,由于 EMPLOYEE 表创建了引用 DEPARTMENT 表主键的外键约束,因此,此时 DEPARTMENT 表的记录不能删除。如果需要删除,可以先删除 EMPLOYEE 表的外键约束或所有记录,然后再删除 DEPARTMENT 表的记录。如下述示例所示。

【示例】 删除有外键约束引用的表中的所有记录。

```
SQL> DELETE FROM employee;
已删除 12 行。
SQL> DELETE FROM department;
已删除 5 行。
```

使用 DELETE 语句删除数据时,实际上数据库会将数据标记为 UNUSED,同时将操作过程写入日志文件,此时并不释放空间,因此 DELETE 操作可以回滚数据。但是,如果要删除的数据量非常大,此过程将会非常耗时,为此,Oracle 数据库提供了另一种删除数据的命令 TRUNCATE。TRUNCATE 语句执行时将释放存储空间,并且不写入日志文件,因此也不能回滚操作,但执行效率较高。TRUNCATE 语句的语法如下。

【语法】

```
TRUNCATE TABLE table_name;
```

【示例】 使用 **TRUNCATE** 语句删除有外键约束引用的表中的所有记录。

```
SQL> INSERT INTO department SELECT * FROM scott.dept;
已创建 4 行。
SQL> ALTER TABLE employee DROP CONSTRAINT fk_emp_dept;
表已更改。
SQL> TRUNCATE TABLE department;
表被截断。
SQL> ROLLBACK;
回退已完成。
SQL> SELECT * FROM department;
未选定行
```

5.3 事务控制

数据库中使用约束保证数据的完整性之外，保证数据的一致性同样十分重要。数据一致性是指数据库的数据在每一时刻都是稳定且可靠的状态，而事务是保证数据一致性的主要手段。

5.3.1 事务的概念和特性

事务(Transaction)是由一系列相关的 SQL 语句组成的最小逻辑工作单元。Oracle 系统以事务为单位来处理数据，用来保证数据的一致性。对于事务中的每一个操作要么全部完成，要么全部不执行。例如，用户 A 向用户 B 转账，其中包括两条 SQL 语句：一条 UPDATE 语句负责在用户 A 的账户中减去转账额度，另一条 UPDATE 语句负责在用户 B 的账户中增加转账额度。这两条减少和增加用户账户额度的 SQL 语句必须全部执行成功，被永久性地记录到数据库中才算成功完成此次转账操作，否则如果任何一条语句执行异常，转账就会出现问题，此时需要做的就是同时取消减少和增加这两条 SQL 语句操作，恢复到转账前的初始账户状态。

事务具有四个特性：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)和持久性(Durability)。这四个特性简称为 ACID 特性，具体说明如下。

- 原子性：事务是原子的，事务中包含的所有 SQL 语句组成一个不可分割的工作单元，要么所有 SQL 语句全部操作成功，要么全部操作失败。例如，用户转账事务中，用户 A 的转账操作和用户 B 的接账操作便是一个不可分割的工作单元，是一次原子性的操作。
- 一致性：事务执行的结果必须是使数据库从一个一致性状态转变到另一个一致性状态，不存在中间的状态。例如，用户转账事务中，如果用户 A 和用户 B 转账成功，则它们将保持转账后账户的一致性，如果用户 A 和用户 B 转账失败，则保持转账操作前账户的一致性，即用户 A 的钱不会减少，用户 B 的钱不会增加。
- 隔离性：数据库中一个事务的执行不受其他事务干扰，每个事务都感觉不到还有其他事务在并发执行。例如，如果用户 A 向用户 B 转账的同时用户 C 也向用户 B 转账，则 A 向 B 转账的事务和 C 向 B 转账的事务会并发执行，执行过程中两个事务之间也无法互访，只有当两个事务都完成最终操作时，才可以看见操作结果。
- 持久性：一个事务一旦提交，则对数据库中数据的改变是永久性的，以后的操作或故障不会对事务的操作结果产生任何影响。例如，用户转账事务被提交后，用户账户数据便会被永久性地保存在磁盘中，此次事务的操作结果即已生效。

5.3.2 事务的开始与结束

数据库中，每个事务都有清晰的开始与结束事件点。当下列事件发生时，事务开始。

- 第一个 DML 语句(INSERT、UPDATE、DELETE)执行时事务开始。
- 前一个事务结束后，又输入一条 DML 语句执行时新事务开始。

遇到下面事件之一时，事务结束。

- 执行 COMMIT(事务提交)或 ROLLBACK(事务回滚)语句后，事务结束。
- 执行一条 DDL 语句(如 CREATE TABLE)时，会自动执行 COMMIT 语句提交事务，事务结束。
- 使用 EXIT 命令正常退出 SQL * Plus 时，会自动执行 COMMIT 语句提交事务，事务结束。如果 SQL * Plus 被意外终止，则自动执行 ROLLBACK 回滚事务，事务结束。

下述示例演示数据库事务的开始与结束。

【示例】 事务的开始与结束。

```
SQL> CONNECT test/QSTqst2015;
SQL> CREATE TABLE account(
  2     account_id VARCHAR2(16),
  3     account_name VARCHAR2 (10),
  4     account_balance NUMBER(16,3),
  5     CONSTRAINT pk_accountid PRIMARY KEY(account_id)
  6   );
表已创建。
SQL> INSERT INTO account VALUES('1001','张三',1000);
SQL> -- 第一个 DML 语句执行,事务 A 开始
已创建 1 行。
SQL> INSERT INTO account VALUES('1002','李四',1);
已创建 1 行。
SQL> SELECT * FROM account;
ACCOUNT_ID                     ACCOUNT_NAME          ACCOUNT_BALANCE
------------------------------ --------------------  ----------------
1001                           张三                          1000
1002                           李四                             1
SQL> COMMIT;
提交完成。
SQL> -- 事务提交,事务 A 结束
SQL> UPDATE account SET account_balance = account_balance - 1000
  2  WHERE account_id = '1001';
已更新 1 行。
SQL> -- 执行 DML 语句,新的事务 B 开始
SQL>  UPDATE account SET account_balance = account_balance + 1000
  2  WHERE account_id = '1002';
已更新 1 行。
SQL> ALTER TABLE account
  2  ADD CONSTRAINT ck_accountbalance CHECK(account_balance >= 0);
表已更改。
SQL> -- 执行 DDL 语句,事务 B 自动提交,事务 B 结束
SQL> SELECT * FROM account;
ACCOUNT_ID                     ACCOUNT_NAME          ACCOUNT_BALANCE
------------------------------ --------------------  ----------------
1001                           张三                             0
1002                           李四                          1001
SQL> DELETE FROM account WHERE account_id = '1001';
已删除 1 行。
```

```
SQL> -- 新的事务 C 开始
SQL> EXIT; -- 正常退出 SQL Plus,事务 C 被自动提交,事务 C 结束
```

5.3.3 事务的提交和回滚

事务的成功完成,以事务被提交后事务中 SQL 语句的结果被永久性地记录为标志。当事务提交后,用户对数据库修改操作的日志信息由日志缓冲区写入重做日志文件中,事务所占据的系统资源和数据库资源被释放。此时,其他用户可以看到该事务对数据库的修改结果。

下述示例演示当一个用户的数据操作事务未提交时,数据库中数据的变化以及该用户和其他用户对数据的查询现象。首先开启一个 SQL * Plus 连接,执行一条 DML 语句,对部门表数据进行更新,但不进行事务提交,然后开启另一个 SQL * Plus 连接,查看部门表的数据结果,如图 5-2 所示。

图 5-2 通过不同窗口查看未提交的事务数据

从上述示例结果可以看出,左边 SQL * Plus 连接会话中通过执行 INSERT 语句开启一个事务,然后在该事务中可以查看到对部门表的数据更新结果,但此时事务仍没有被提交,因此新增的数据并没有被永久性地写入数据库中,这样,在右边开启的另一个 SQL * Plus 会话中便不能查看到部门表更新后的结果。

在 Oracle 数据库中,事务提交有两种方式:一种方式是执行 COMMIT 命令显式提交;另一种方式是执行特定操作(如 DDL 语句)时系统自动提交,即隐式提交。

通过 COMMIT 命令显式提交的语法示例如下所示。

【示例】 通过 COMMIT 命令显式提交事务。

```
SQL> CREATE TABLE department(
  2  deptno NUMBER(2) PRIMARY KEY,
  3  dname VARCHAR2(14),
  4  loc VARCHAR2(13)
  5  )
  6  TABLESPACE USERS;
表已创建。
SQL> INSERT INTO department VALUES(50,'研发部','青岛');
已创建 1 行。
```

```
SQL> COMMIT;
提交完成。
SQL> SELECT * FROM department;
    DEPTNO   DNAME                        LOC
----------  ---------------------  -------------
        50   研发部                        青岛
SQL> INSERT INTO department VALUES(60,'产品部','青岛');
已创建 1 行。
SQL> UPDATE department SET loc = '高新区' WHERE deptno = 60;
已更新 1 行。
SQL> COMMIT;
提交完成。
SQL> SELECT * FROM department;
    DEPTNO   DNAME                        LOC
----------  ---------------------  -------------
        50   研发部                        青岛
        60   产品部                        高新区
```

在事务执行过程中，如果需要取消已执行的 SQL 语句的结果，可以使用事务回滚命令 ROLLBACK。执行该命令后，事务中的所有操作都将被取消，数据库恢复到事务开始之前的状态，同时事务所占用的系统资源和数据库资源被释放，如下述示例所示。

【示例】 通过 ROLLBACK 命令回滚单条 DML 语句事务。

```
SQL> SELECT * FROM department;
    DEPTNO   DNAME                        LOC
----------  ---------------------  -------------
        50   研发部                        青岛
        60   产品部                        高新区
SQL> DELETE FROM department WHERE deptno = 60;
已删除 1 行。
SQL> ROLLBACK;
回退已完成。
SQL> SELECT * FROM department;
    DEPTNO   DNAME                        LOC
----------  ---------------------  -------------
        50   研发部                        青岛
        60   产品部                        高新区
```

【示例】 通过 ROLLBACK 命令回滚多条 DML 语句事务。

```
SQL> INSERT INTO department VALUES(70,'市场部','青岛');
已创建 1 行。
SQL> UPDATE department SET loc = '高新区' WHERE deptno = 70;
已更新 1 行。
SQL> ROLLBACK;
回退已完成。
SQL> SELECT * FROM department;
    DEPTNO   DNAME                        LOC
----------  ---------------------  -------------
        50   研发部                        青岛
        60   产品部                        高新区
```

从上述示例可以看出，在由多个 SQL 语句组成的事务中，默认情况下，ROLLBACK 命令意味着全部的操作都要回滚，而在多数情况下，事务中大部分的工作都是必须执行的，仅有少量的工作有可能出现异常，并需要回滚。此时，则可以使用 SAVEPOINT 命令，通过设置保存点的方式，使事务仅回滚到指定的保存点处。SAVEPOINT 的操作流程如图 5-3 所示。

图 5-3　回滚保存点的执行流程

SAVEPOINT 命令设置保存点的语法如下。

【语法】

```
SAVEPOINT savepoint_name;
```

【示例】 设置和回滚保存点。

```
SQL> SELECT * FROM department;
    DEPTNO   DNAME                        LOC
---------- ---------------------- -------------
        50   研发部                        青岛
        60   产品部                        高新区
SQL> UPDATE department SET loc = 'QING DAO' WHERE deptno = 50;
已更新 1 行。
SQL> SAVEPOINT sp1;
保存点已创建。
SQL> DELETE department WHERE deptno = 50;
已删除 1 行。
SQL> SAVEPOINT sp2;
保存点已创建。
SQL> ROLLBACK TO sp1;
回退已完成。
SQL> SELECT * FROM department;
    DEPTNO   DNAME                        LOC
---------- ---------------------- ------------------
        50   研发部                        QING DAO
        60   产品部                        高新区
SQL> ROLLBACK;
回退已完成。
SQL> SELECT * FROM department;
    DEPTNO   DNAME                        LOC
---------- ---------------------- -------------
        50   研发部                        青岛
        60   产品部                        高新区
```

通常情况下，回滚操作是非常耗费时间和资源的，因此，回滚往往被用于处理异常，而不用作终端用户的可操作选项。因为如果终端用户经常性地使用回滚操作，那么将为数据库带来非常大的负担。所以应让终端用户在提交事务之前进行确认，以此来代替允许用户执行回滚，从而实现提交与回滚操作的平衡。

5.3.4 并发事务

在一般的数据库应用中，不太可能出现在一个时刻有且只有一个事务在操作数据库的情况。对于大多数数据库应用来说，往往都会出现两个或两个以上事务试图修改数据库中的同一个数据的情况，这种情况就被称为事务的并发(Concurrent Transaction)。

并发能力是衡量数据库性能的重要指标之一，数据库允许并发数量越大，标志着该数据库的性能越好。另一方面，并发会带来数据库不一致性的风险。数据库事务的并发运行，可能导致下列 3 类问题。

- **脏读**：一个事务对数据的修改在提交之前被其他事务读取，如表 5-1 所示的示例。

表 5-1 事务并发问题之读脏数据

时间	转账事务 A	取款事务 B
T1		开始事务
T2		查询余额为 1000 元
T3		取款 200 元，更新余额为 800 元
T4	开始事务	
T5	查询余额为 800 元	
T6		撤销事务，余额恢复为 1000 元
T7	汇入 100 元，更新余额为 900 元	
T8	提交事务	

- **不可重复读**：在某个事务读取一次数据后，其他事务修改了这些数据并进行了提交，当该事务重新读取这些数据时就会得到与前一次不一样的结果，如表 5-2 所示的示例。

表 5-2 事务并发问题之不可重复读

时间	取款事务 A	取款事务 B
T1		开始事务
T2	开始事务	
T3		查询余额为 1000 元
T4	查询余额为 1000 元	
T5		取款 200 元，更新余额为 800 元
T6		提交事务
T7	查询余额为 800 元，与 T4 查询时不同	

- **幻读**：同一查询在同一事务中多次进行，由于其他提交事务所做的插入或删除操作，每次返回不同的结果集，如表 5-3 所示的示例。

表 5-3　事务并发问题之幻读

时间	统计事务 A	开户事务 B
T1		开始事务
T2	开始事务	
T3	统计余额总额为 10000 元	
T4		新增账户,初始余额为 1000 元
T5		提交事务
T6	统计余额总额为 11000 元,与 T3 不同	

为了处理这些可能出现的并发问题,数据库提供了不同级别的事务隔离性,以防止并发事务相互影响。SQL92 标准定义了以下几种事务隔离级别,按照隔离性级别从低到高依次如下。

- READ UNCOMMITTED:称为未授权读取或读未提交。表示即使一个更新操作的事务没有提交,其他事务仍然可以读取到改变的数据。
- READ COMMITTED:称为授权读取或读提交。表示在一个更新操作的事务提交以后,其他的事务才能读取到改变的数据。
- REPEATABLE READ:称为可重复读取。表示在同一事务里面先后执行同一个查询语句时,得到的结果集相同。
- SERIALIZABLE:称为序列化。提供严格的事务隔离,要求事务序列化执行,即事务只能一个接着一个地执行,不能并发执行。

这 4 种事务的隔离级别对并发事务有可能产生的脏读、不可重复读和幻读这 3 类问题的处理程度如表 5-4 所示。

表 5-4　事务隔离级别与并发问题

隔离级别 SQL92 标准	脏　读	不可重复读	幻　读
READ UNCOMMITTED	会出现	会出现	会出现
READ COMMITTED	不会出现	会出现	会出现
REPEATABLE READ	不会出现	不会出现	会出现
SERIALIZABLE	不会出现	不会出现	不会出现

由上述 4 种隔离级别可以看出,隔离级别越高,越能保证数据的完整性和一致性。但同时隔离级别也同数据库的并发性能成反比,隔离级别越高,数据库的并发性能越低。对于多数应用程序,可以优先考虑把数据库系统的隔离级别设为 READ COMMITTED,它能够避免脏读,而且具有较好的并发性能。但它会导致不可重复读、幻读和第二类丢失更新这些并发问题,在可能出现这类问题的个别场合,可以由应用程序端进行解决。

SQL92 标准定义的上述 4 种隔离级别中,Oracle 数据库仅支持 READ COMMITTED 和 SERIALIZABLE 两种事务隔离级别,因此 Oracle 数据库的查询永远不会读取脏数据(未提交的数据)。在 Oracle 支持的这两种事务隔离级别中,READ COMMITTED 隔离级别是 Oracle 默认使用的事务隔离级别。另外,Oracle 还增加了一个非 SQL92 标准的 READ ONLY(只读)事务模式,在 READ ONLY 事务模式下,只读事务只能看到事务执行前就已经提交的数据,且事务中不能执行 INSERT、UPDATE 及 DELETE 操作,因此

READ ONLY 事务模式可以避免脏读、不可重复读和幻读问题。

数据库管理员及应用程序的设计开发者可以依据应用程序的需求及系统负载而为不同的事务选择不同的隔离级别。用户可以在事务开始时使用以下语句设定事务的隔离级别。

【示例】 设置数据库事务为 SERIALIZABLE 隔离级别。

```
SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
事务处理集。
```

【示例】 设置数据库事务为 READ COMMITTED 隔离级别。

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
事务处理集。
```

【示例】 设置数据库事务为 READ ONLY 隔离级别。

```
SQL> SET TRANSACTION READ ONLY;
事务处理集。
```

如果数据库中具有大量并发事务,并且应用程序的事务处理能力和响应速度是关键因素,则 READ COMMITED 隔离级别比较合适。如果数据库中多个事务并发访问数据的概率很低,并且大部分的事务都会持续执行很长时间,则应用程序适合使用 REALIZABLE 隔离级别。事务隔离级别修改的示例如下。

【示例】 修改数据库事务的隔离级别为 SERIALIZABLE。

```
SQL> ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
会话已更改。
```

【示例】 修改数据库事务的隔离级别为 READ COMMITTED。

```
SQL> ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED;
会话已更改。
```

下述实例演示 Oracle 在使用默认事务隔离级别 READ COMMITTED 时,通过两个 SQL * Plus 的会话对同一表数据进行操作时出现的幻读现象。

【实例 1】 创建一个事务测试表 tran_test。

```
SQL> CONN test/QSTqst2015;
已连接。
SQL> CREATE TABLE tran_test(num NUMBER);
表已创建。
```

【会话 1】 插入一条记录,不提交。

```
SQL> CONN test/QSTqst2015;
已连接。
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
事务处理集。
SQL> INSERT INTO tran_test VALUES(10);
已创建 1 行。
SQL> SELECT * FROM tran_test;
```

```
       NUM
----------
        10
```

【会话 2】 插入一条记录并提交。

```
SQL> CONN test/QSTqst2015;
已连接。
SQL> INSERT INTO tran_test VALUES(20);
已创建 1 行。
SQL> COMMIT;
提交完成。
```

【会话 1】 再次进行表数据查询。

```
SQL> SELECT * FROM tran_test;
       NUM
----------
        10
        20
```

上述【实例 1】中，在【会话 1】中可以看到【会话 2】中已提交的数据。在该隔离级别中，只要其他事务已提交(即使其他事务在该事务之后才开始)，该事务就能看到其他事务对数据操作的结果，因此 READ COMMITTED 隔离级别不能阻止不可重复读和幻读。

下述【实例 2】演示在事务隔离级别为 SERIALIZABLE 时，执行和【实例 1】相同的事务操作时数据的读取现象。

【实例 2】 清空表中记录。

```
SQL> DELETE FROM tran_test;
已删除 2 行。
SQL> EXIT;
```

【会话 1】 插入一条记录，不提交。

```
SQL> CONN test/QSTqst2015;
已连接。
SQL> ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
会话已更改。
SQL> INSERT INTO tran_test VALUES(10);
已创建 1 行。
SQL> SELECT * FROM tran_test;
       NUM
----------
        10
```

【会话 2】 插入一条记录并提交。

```
SQL> CONN test/QSTqst2015;
已连接。
SQL> INSERT INTO tran_test VALUES(20);
已创建 1 行。
```

```
SQL> COMMIT;
提交完成。
```

【会话 1】 再次进行表数据查询。

```
SQL> SELECT * FROM tran_test;
       NUM
----------
        10
```

从上述【实例 2】可以看出,【会话 1】中的事务仅能看见在本事务中所做的更改。这种隔离级别查询的结果已经在事务启动的时候确定,事务启动后其他事务对数据的改变,对本事务的查询没有影响,因此 SERIALIZABLE 隔离级别不会出现不可重复读和幻读。

下述【实例 3】演示在事务模式为 READ ONLY 时,执行和【实例 1】相同的事务操作时数据的读取现象。

【实例 3】 清空表中记录。

```
SQL> DELETE FROM tran_test;
已删除 2 行。
SQL> EXIT;
```

【会话 1】 插入一条记录,不提交。

```
SQL> CONN test/QSTqst2015;
已连接。
SQL> SET TRANSACTION READ ONLY;
事务处理集。
SQL> INSERT INTO tran_test VALUES(10);
INSERT INTO tran_test VALUES(10)
*
第 1 行出现错误:
ORA-01456: 不能在 READ ONLY 事务处理中执行插入/删除/更新操作
```

从上述【会话 1】可以看出,READ ONLY 事务模式中不允许在本事务中进行 DML 操作。此时在【会话 1】中查询当前表中的数据结果集如下所示。

【会话 1】 查询此时表中的数据。

```
SQL> SELECT * FROM tran_test;
未选定行
```

【会话 2】 插入一条记录并提交。

```
SQL> CONN test/QSTqst2015;
已连接。
SQL> INSERT INTO tran_test VALUES(20);
已创建 1 行。
SQL> COMMIT;
提交完成。
```

【会话 1】 再次进行表数据查询。

```
SQL > SELECT * FROM tran_test;
未选定行
```

由【实例 3】可以看出，READ ONLY 事务模式也可以避免不可重复读和幻读，它和 SERIALIZABLE 隔离级别唯一的不同是它不允许在本事务中进行 DML 操作。

5.4 课程贯穿项目

5.4.1 【任务 5-1】 项目数据管理

本阶段需要添加一些测试数据到系统中供功能测试使用，同时根据业务需求对一些数据进行更新、删除和简单查询操作。

1）为项目各表添加测试数据

执行下述测试数据添加脚本 qmicrochat_testdata. sql，为项目各表添加测试数据。

【任务 5-1】 数据添加脚本 qmicrochat_testdata. sql。

```
-- TB_USERS 表测试数据
INSERT INTO tb_users(
  user_id,username,userpwd,nickname,uprofile,sex,telephone,
  email,address,signature,note)
VALUES(1,'张三','123456','张小三',null,'男','15515436543',
  'zhangs@test.com','青岛','数据库学习','');
INSERT INTO tb_users(
  user_id,username,userpwd,nickname,uprofile,sex,telephone,
  email,address,signature,note)
VALUES(2,'李四','234567','李小四',null,'男','15367894325',
  'lisi@test.com','青岛','学习加油','');
INSERT INTO tb_users(
  user_id,username,userpwd,nickname,uprofile,sex,telephone,
  email,address,signature,note)
VALUES(3,'王五','345678','王小五',null,'女','13189654354',
  'wangw@test.com','北京','走自己的路','');

-- TB_FRIENDS 表测试数据
INSERT INTO tb_friends VALUES(1,3);
INSERT INTO tb_friends VALUES(2,3);

-- TB_PERSONAL_DYNAMICS 表测试数据
INSERT INTO tb_personal_dynamics
VALUES(1,1,to_date('2012-12-01 03:03:02','yyyy-mm-dd hh24:mi:ss'),
  '青岛','现在是凌晨 3 点',1,1);
INSERT INTO tb_personal_dynamics
VALUES(2,1,to_date('2016-12-29 00:00:00','yyyy-mm-dd hh24:mi:ss'),
  '青岛','新年快乐',2,1);
INSERT INTO tb_personal_dynamics
VALUES(3,2,to_date('2016-06-01 05:03:02','yyyy-mm-dd hh24:mi:ss'),
```

```
  '青岛','今天是六一儿童节',2,1);

--TB_PHOTOS_DYNAMICS 表测试数据
INSERT INTO tb_photos_dynamics(photo_id,dynamic_id ,display_order)
VALUES(1,1,1);
INSERT INTO tb_photos_dynamics(photo_id,dynamic_id ,display_order)
VALUES(2,1,2);

--TB_ARTICS_DYNAMICS 表测试数据
INSERT INTO tb_artics_dynamics(article_id,dynamic_id,article_url,reading_num)
VALUES(1,2,'http://www.itshixun.com',10);
INSERT INTO tb_artics_dynamics(article_id,dynamic_id,article_url,reading_num)
VALUES(2,3,'http://book.moocollege.cn',50);

--TB_COMMENT 表测试数据
INSERT INTO tb_comment
VALUES (1,1,3,'拍的不错',
  to_date('2014-07-29 21:00:00','yyyy-mm-dd hh24:mi:ss'));
INSERT INTO tb_comment
VALUES(2,2,3,'好文章!', to_date('2015-07-29 21:00:00','yyyy-mm-dd hh24:mi:ss'));
INSERT INTO tb_comment
VALUES(3,3,3,'点个赞',to_date('2016-07-29 21:00:00','yyyy-mm-dd hh24:mi:ss'));

--TB_COMMENT_REPLY 表测试数据
INSERT INTO tb_comment_reply VALUES (1,1,1,'谢谢夸奖');
INSERT INTO tb_comment_reply VALUES (2,3,1,'哈哈');

--TB_USER_CHAT 表测试数据
INSERT INTO tb_user_chat
VALUES(1,1,3,'在干嘛?',to_date('2015-07-29 21:00:00','yyyy-mm-dd hh24:mi:ss'));
INSERT INTO tb_user_chat
VALUES 2,3,1,'学习呢!',to_date('2015-07-29 21:00:07','yyyy-mm-dd hh24:mi:ss'));
INSERT INTO tb_user_chat
VALUES(3,2,3,'你好?',to_date('2015-07-29 21:01:00','yyyy-mm-dd hh24:mi:ss'));

--TB_GROUPS 表测试数据
INSERT INTO tb_groups(
  group_id ,group_name,user_id,creation_time,max_person_num,real_person_num)
VALUES (1,'数据库技术交流群',1,
  to_date('2015-06-29 21:00:00','yyyy-mm-dd hh24:mi:ss'),500,100);
INSERT INTO tb_groups(
  group_id ,group_name,user_id,creation_time,max_person_num,real_person_num)
VALUES(2,'Java 技术分享群',1,
  to_date('2016-06-29 21:00:00','yyyy-mm-dd hh24:mi:ss'),300,200);
INSERT INTO tb_groups(
  group_id ,group_name,user_id,creation_time,max_person_num,real_person_num)
VALUES(3,'大数据技术交流群',2,
  to_date('2013-06-29 21:00:00','yyyy-mm-dd hh24:mi:ss'),500,400);

--USERS_GROUPS 表测试数据
INSERT INTO users_groups VALUES(1,1,'无敌',0,0);
INSERT INTO users_groups VALUES(3,1,'西伯利亚',1,0);
```

```
INSERT INTO users_groups VALUES(2,3,'寒冷',0,1);

--TB_GROUP_CHAT 表测试数据
INSERT INTO tb_group_chat
VALUES(1,1,1,to_date('2016-05-12 20:00:00','yyyy-mm-dd hh24:mi:ss'),'大家好!');
INSERT INTO tb_group_chat
VALUES(2,1,1,to_date('2016-05-14 17:00:00','yyyy-mm-dd hh24:mi:ss'),'欢迎大家');
INSERT INTO tb_group_chat
VALUES(3,1,3,to_date('2016-05-17 20:00:00','yyyy-mm-dd hh24:mi:ss'),'我来啦!');
```

【任务 5-1】 执行脚本 qmicrochat_testdata.sql。

```
SQL> CONN qmicrochat_admin/admin2015;
已连接。
SQL> START E:\qmicrochat_testdata.sql
1 行已插入。
…
```

2）数据的简单查询

根据以下任务需求，进行数据的简单查询。

- 查询项目所有注册用户的编号、姓名、昵称、性别和电话。
- 查询编号为 1 的用户创建的所有群信息。
- 查询 2015 年以前创建的所有群的编号、群名、创建人、创建时间。

【任务 5-1】（1）查询项目所有注册用户的编号、姓名、昵称、性别和电话。

```
SQL> SELECT user_id,username,nickname,sex,telephone FROM tb_users;

   USER_ID     USERNAME     NICKNAME    SEX    TELEPHONE
----------  ----------  ----------  ----  -----------------
         1     张三         张小三       男     15515436543
         2     李四         李小四       男     15367894325
         3     王五         王小五       女     13189654354
```

【任务 5-1】（2）查询编号为 1 的用户创建的所有群信息。

```
SQL> SELECT * FROM tb_groups WHERE user_id=1;

GROUP_ID GROUP_NAME        GROUP_LOGO  USER_ID CREATION_TIME   MAX_PERSON_NUM  REAL_PERSON_NUM
------ --------------  ---------  ----------------  ------------  ------------
1        数据库技术交流群           1  29-6月 -15         500           100
2        Java 技术分享群            1  29-6月 -16         300           200
```

【任务 5-1】（3）查询 2015 年以前创建的所有群的编号、群名、创建人、创建时间。

```
SQL> SELECT group_id,group_name,user_id,creation_time
  2     FROM tb_groups
  3     WHERE creation_time <= '1-1月 -15';
  GROUP_ID     GROUP_NAME           USER_ID     CREATION_TIME
----------  -----------------  ---------  -------------------
         3     大数据技术交流群      2          29-6月 -13
```

3）数据的修改

根据以下任务需求，进行数据的修改操作。

- 重置编号为 3 的用户的密码为“000000”。
- 将当前所有阅读次数为 0 的动态文章的初始值设为 100，已有阅读次数的在现有阅读次数基础上增加 100。
- 将编号为 2 的用户在 3 号群的昵称修改为“学霸”。

【任务 5-1】（4）重置编号为 3 的用户的密码为“000000”。

```
SQL> UPDATE tb_users SET userpwd = '000000' WHERE user_id = 3;
已更新 1 行。
```

【任务 5-1】（5）修改阅读次数。

```
SQL> UPDATE tb_artics_dynamics SET reading_num = reading_num + 100;
已更新 2 行。
```

【任务 5-1】（6）将编号为 2 的用户在 3 号群的昵称修改为“学霸”。

```
SQL> UPDATE users_groups SET group_nickname = '学霸'
  2     WHERE user_id = 2 AND group_id = 3;
已更新 1 行。
```

4）数据的删除

根据以下任务需求，进行数据的删除操作。

- 解除编号为 1 的用户与编号为 3 的用户的好友关系。
- 删除编号为 1 的用户与编号为 3 的用户的所有私聊信息。
- 删除编号为 1 的用户群的所有聊天信息。

【任务 5-1】（7）解除用户的好友关系。

```
SQL> DELETE FROM tb_friends WHERE user_id = 1 AND friend_id = 3;
```

【任务 5-1】（8）删除用户间的私聊信息。

```
SQL> DELETE FROM tb_user_chat
  2     WHERE send_user_id = 1 AND receive_user_id = 3
  3        OR send_user_id = 3 AND receive_user_id = 1;
```

【任务 5-1】（9）删除某个群的所有聊天信息。

```
SQL> DELETE FROM tb_group_chat WHERE group_id = 1;
```

为保证本项目测试数据的完整性，此处的删除任务仅给出 SQL 语句，不执行。读者可自行添加测试数据进行练习。

5.4.2 【任务 5-2】 项目事务控制

通过数据库事务控制，完成 Q_MicroChat 微聊项目的以下需求。

1）编号为 3 的用户发表相册动态

【任务 5-2】（1）发表相册动态。

```
SQL> --3 号用户添加个人动态,并指定动态类型为相册
SQL> INSERT INTO tb_personal_dynamics
  2  VALUES(4,3,to_date('2015-12-12 08:03:02','yyyy-mm-dd hh24:mi:ss'),
  3    '青岛','看看我拍的照片',1,1);
已创建 1 行。
SQL> --为 3 号用户添加的个人动态添加相册动态
SQL> INSERT INTO tb_photos_dynamics(photo_id,dynamic_id,display_order)
  2  VALUES(3,4,1);
已创建 1 行。
SQL> COMMIT;
提交完成。
```

2）编号为 3 的用户发表文章动态

【任务 5-2】（2）发表文章动态。

```
SQL> --3 号用户添加个人动态,并指定动态类型为文章
SQL> INSERT INTO tb_personal_dynamics
  2  VALUES(5,3,to_date('2015-10-23 12:03:02','yyyy-mm-dd hh24:mi:ss'),
  3    '青岛','鸡汤分享',2,1);
已创建 1 行。
SQL> --为 3 号用户添加的个人动态添加文章动态
SQL> INSERT INTO tb_artics_dynamics(article_id,dynamic_id,article_url)
  2  VALUES(3,5,'http://book.moocollege.cn/oracle.html');
已创建 1 行。
SQL> COMMIT;
提交完成。
```

3）删除编号为 5 的个人动态

【任务 5-2】（3）删除某条个人动态。

```
SQL> --删除 5 号个人动态对应的文章动态
SQL> DELETE FROM tb_artics_dynamics WHERE dynamic_id=5;
已删除 1 行。
SQL> --删除 5 号个人动态
SQL> DELETE FROM tb_personal_dynamics WHERE dynamic_id=5;
已删除 1 行。
SQL> COMMIT;
提交完成。
```

4）删除编号为 3 的用户发表的所有个人动态，包括该动态对应的相册动态及对该动态的所有评论和评论回复

【任务 5-2】（4）添加对该动态的评论和评论回复测试数据。

```
SQL> --为 4 号个人动态添加评论
SQL> INSERT INTO tb_comment
```

```
VALUES(4,4,1,'好技术',to_date('2015-12-12 08:30:10','yyyy-mm-dd hh24:mi:ss'));
已创建 1 行。
SQL> -- 为 4 号评论添加评论回复
SQL> INSERT INTO tb_comment_reply VALUES (3,4,3,'相机好');
已创建 1 行。
```

【任务 5-2】（5）删除某人的所有个人动态。

```
SQL> -- 删除对 4 号评论的所有回复
SQL> DELETE FROM tb_comment_reply WHERE comment_id = 4;
已删除 1 行。
SQL> -- 删除 4 号个人动态对应的评论
SQL> DELETE FROM tb_comment WHERE dynamic_id = 4;
已删除 1 行。
SQL> -- 删除 4 号个人动态对应的相册动态
SQL> DELETE FROM tb_photos_dynamics WHERE dynamic_id = 4;
已删除 1 行。
SQL> -- 删除 3 号用户发表的所有个人动态
SQL> DELETE FROM tb_personal_dynamics WHERE user_id = 3;
已删除 1 行。
SQL> COMMIT;
提交完成。
```

5）删除群编号为 4 的群，包括解散该群的用户，以及删除所有群聊记录

【任务 5-2】（6）添加 4 号群及群用户、群聊测试数据。

```
SQL> -- 新增 4 号群
SQL> INSERT INTO tb_groups(
  2  group_id ,group_name,user_id,creation_time,max_person_num,real_person_num)
  3  VALUES(4,'Oracle 技术交流群',3,
  4  to_date('2015-12-29 21:00:00','yyyy-mm-dd hh24:mi:ss'),500,400);
已创建 1 行。
SQL> -- 添加 4 号群的参与用户
SQL> INSERT INTO users_groups VALUES(3,4,'甲骨文',0,0);
已创建 1 行。
SQL> INSERT INTO users_groups VALUES(2,4,'SCOTT',0,0);
已创建 1 行。
SQL> -- 添加 4 号群的群聊记录
SQL> INSERT INTO tb_group_chat
  2  VALUES(4,4,3,to_date('2015-12-29 21:10:10','yyyy-mm-dd hh24:mi:ss'),
  3  '欢迎加入!');
已创建 1 行。
SQL> INSERT INTO tb_group_chat
  2  VALUES(5,4,2,to_date('2015-12-29 21:11:20','yyyy-mm-dd hh24:mi:ss'),
  3  '我捧场!');
已创建 1 行。
```

【任务 5-2】（7）解散 4 号群。

```
SQL> -- 删除 4 号群的群聊记录
SQL> DELETE FROM tb_group_chat WHERE group_id = 4;
```

```
已删除 2 行。
SQL> -- 删除 4 号群的参与用户
SQL> DELETE FROM users_groups WHERE  group_id = 4;
已删除 2 行。
SQL> -- 删除 4 号群信息
SQL> DELETE FROM tb_groups WHERE group_id = 4;
已删除 1 行。
SQL> COMMIT;
```

本章小结

小结

- SQL 语言是关系数据库操作的基础语言，将数据查询、数据操纵、数据定义、事务控制、系统控制等功能集于一体，使得数据库应用开发人员、数据库管理员等都可以通过 SQL 语言实现对数据库的访问和操作。
- 根据 SQL 语言实现功能的不同，Oracle 数据库中的 SQL 语言可以分为：数据定义语言(DDL)、数据查询语言(DQL)、数据操纵语言(DML)、事务控制语言、系统控制语言、会话控制语言。
- 遵守 SQL 的代码规则，可以使要处理的语句与共享池中的相一致，会大大地提高项目的开发效率以及提升数据库对 SQL 语句的执行性能。
- 约束保证数据的完整性，事务保证数据的一致性。
- 事务(Transaction)是由一系列相关的 SQL 语句组成的最小逻辑工作单元。
- 事务具有四个特性：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)。
- 数据库事务的并发运行可导致脏读、不可重复读、幻读这 3 类问题。
- 为了处理可能出现的并发问题，SQL92 标准定义了 READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ、SERIALIZABLE 4 种事务隔离级别。
- Oracle 数据库仅支持 READ COMMITTED 和 SERIALIZABLE 两种事务隔离级别，同时增加了一个非 SQL92 标准的 READ ONLY(只读)事务模式。
- 如果数据库中具有大量并发事务，并且应用程序的事务处理能力和响应速度是关键因素，则应用程序适合 READ COMMITED 隔离级别；如果数据库中多个事务并发访问数据的概率很低，并且大部分的事务都会持续执行很长时间，则应用程序适合使用 REALIZABLE 隔离级别。

Q&A

1. 问：SQL 有何作用？它的产生有何重要意义？

答：SQL 即结构化查询语言，它是关系数据库的标准语言。在 SQL 规范推出之前，各

种数据库有着各自的数据操作方式，SQL 的出现，统一了不同数据库间的数据操作问题，很大程度上解决了程序开发人员的困难。SQL 功能强大，同时又简洁易学，集数据查询、数据操作、数据定义和数据控制等功能于一体，被用户和业界所青睐，并成为国际标准。

2. 问：Oracle 数据库中有哪几种常用的 SQL 语言？

答：Oracle 数据库中常用的 SQL 语言类别有：数据定义语言(DDL)，用于定义、修改、删除数据库模式对象，如 CREATE、ALTER、DROP 等；数据查询语言(DQL)，用于数据检索，如 SELECT；数据操纵语言(DML)，用于改变数据库中的数据，如 INSERT、UPDATE、DELETE 等；事务控制语言(TCL)，用于将一组 DML 操作组合起来，形成一个事务并进行事务控制，如 COMMIT、ROLLBACK、SAVEPOINT 等。

3. 问：Oracle 数据库为何要引入事务？

答：数据一致性是指数据库的数据在每一时刻都是稳定且可靠的状态，而事务是保证数据一致性的主要手段。Oracle 系统以事务为单位来处理数据，以保证数据的一致性。对于事务中的每一个操作要么全部完成，要么全部不执行。

4. 问：Oracle 数据库如何保障事务的并发运行？

答：Oracle 数据库通过支持 READ COMMITTED、SERIALIZABLE 两种级别的事务隔离性以及 READ ONLY 事务模式，防止并发事务的相互影响。其中，READ COMMITTED 隔离级别是 Oracle 默认使用的事务隔离级别，保障 Oracle 数据库的查询永远不会读取脏数据(未提交的数据)；SERIALIZABLE 隔离级别可以避免脏读、不可重复读和幻读问题；READ ONLY 事务模式同样可以避免脏读、不可重复读和幻读问题。

章节练习

习题

1. 要建立一个语句向 Types 表中插入数据，这个表只有两列，T_ID 和 T_Name 列。如果要插入一行数据，这一行的 T_ID 值是 100，T_Name 值是 FRUIT，应该使用的 SQL 语句是(　　)。

A. INSERT INTO Types Values(100,'FRUIT')

B. SELECT * FROM Types WHERE T_ID=100 AND T_Name='FRUIT'

C. UPDATE SET T_ID=100 FROM Types WHERE T_Name='FRUIT'

D. DELETE * FROM Types WHERE T_ID=100 AND T_Name='FRUIT'

2. 用(　　)语句修改表的一行或多行数据。

A. UPDATE　　B. SET　　C. SELECT　　D. WHERE

3. 要建立一个 UPDATE 语句更新表的某列数据，且更新的数据为表统计的结果，则需要在 UPDATE 语句中使用(　　)语句。

A. UPDATE　　B. SET　　C. SELECT　　D. WHERE

4. DELETE 语句中用(　　)语句或子句来指明表中所要删除的行。

A. UPDATE　　B. WHERE　　C. SELECT　　D. INSERT

5. 使用(　　)命令可以清除表中所有的内容。

A. INSERT　　B. UPDATE　　C. DELETE　　D. TRUNCATE

6. 下列(　　)语句会终止事务。

A. SAVEPOINT　　B. ROLLBACK TO SAVEPOINT

C. END TRANSACTION　　D. COMMIT

7. 假如当前数据库中有两个并发的事务,其中,第一个事务修改表中的数据,第二个事务在将修改提交给数据库前查看这些数据。如果第一个事务执行回滚操作,则会发生(　　)。

A. 假读　　B. 非重复读取　　C. 错读　　D. 重复读

8. 在 SQL 语句中,用于向表中插入数据的语句是_____。

9. 如果需要向表中插入一批已经存在的数据,可以在 INSERT 语句中使用_____语句。

10. 事务的 ACID 特性包括_____、一致性、_____和永久性。

11. 在众多的事务控制语句中,用来撤销事务的操作的语句为_____,用于持久化事务对数据库操作的语句是_____。

12. 简述 DELETE 语句与 TRUNCATE 语句的差异。

13. 简述事务的四个属性及其作用。

上机

训练目标:数据管理的 SQL 语句。

培养能力	熟练掌握数据管理 SQL 语句		
掌握程度	★★★★★	难度	中等
代码行数	11	实施方式	重复编码
结束条件	独立编写,运行不出错		

参考训练内容:

(1) 根据表 5-5 所示的表结构创建数据表 pet。

(2) 将表 5-6 中的记录插入 pet 表中。

(3) 使用 UPDATE 语句将名称为 Fang 的狗的主人改为 Kevin。

(4) 将没有主人的宠物的 Owner 字段值都改为 Duck。

(5) 删除已经死亡的宠物记录。

(6) 删除所有表中的记录

表 5-5　pet 表结构

字　段　名	字段说明	数 据 类 型	主键	外键	非空	唯一
Name	宠物名称	VARCHAR2(20)	否	否	是	否
Owner	宠物主人	VARCHAR2(20)	否	否	否	否
Species	种类	VARCHAR2(20)	否	否	是	否
Sex	性别	VARCHAR2(20)	否	否	是	否
Birth	出生日期	DATE	否	否	是	否
Death	死亡日期	DATE	否	否	否	否

表 5-6 pet 表记录

Name	Owner	Species	Sex	Birth	Death
Fluffy	Harold	Cat	f	2003-10-12	2010-8-12
Claws	Gwen	Cat	m	2004-8-10	NULL
Buffy	NULL	Dog	f	2009-8-11	NULL
Fang	Benny	Dog	m	2000-5-15	NULL
Bowser	Diane	Dog	m	2003-4-16	2009-11-12
Chirpy	NULL	Brid	f	2008-5-19	NULL