第 3 章

关系数据库标准语言 SQL

本章介绍关系数据库标准语言 SQL，包括 SQL 概述、数据定义、基本 SQL 查询、数据更新以及视图。

3.1 SQL 概述

SQL 诞生于 20 世纪 70 年代，最初的名称是 SEQUEL(Structured English Query Language)，后更名为 SQL(Structured Query Language)。SQL 是一种非过程化语言，用户只要说明需要的数据库内容，不必说明存取所需数据的具体过程和操作。SQL 集数据定义、数据查询、数据操纵和数据控制等功能于一体，具有功能综合、风格统一、数据操纵高度非过程化、面向集合操作、语法结构统一和语言简洁等特点。

SQL 标准自诞生以来不断发展完善，目前已有 8 个版本，最新版本为 SQL2016，但 SQL92 已经基本完善。考虑到 SQL92 相对简单，且被众多 RDBMS 实现，本书仍以 SQL92 为基础。读者在上机实现时，应参考所使用的 RDBMS 用户手册。

3.2 数据定义

3.2.1 模式的定义与删除

下面是 SQL92 中关于模式定义的语法。

```
    <schema definition> ::=
CREATE SCHEMA <schema name clause>
[ <schema character set or path> ]
[ <schema element>... ]
```

```
    <schema name clause> ::=
<schema name>
| AUTHORIZATION <schema authorization
identifier>
```

```
| <schema name> AUTHORIZATION <schema authorization identifier>

    <schema authorization identifier> ::=
<authorization identifier>
```

SQL 标准语法描述中，符号“::=”表示对一个元素的定义。不带括号的字符串如“CREATE SCHEMA”表示保留的关键字。带括号的字符串如“<schema name clause>”表示一个元素。符号“[ ]”表示可选内容。

根据语法，创建模式必须包含 CREATE SCHEMA 关键字，后面是模式名称子句（<schema name clause>），以及两个可选内容。模式名称子句中可以指定模式名称或被授权的用户，也可以两者同时指定。

【例 3.1】　为用户张三创建一个模式 Online-Shopping。

```
CREATE SCHEMA "Online-Shopping" AUTHORIZATION "张三"
```

下面是删除模式的语法。

```
<drop schema statement> ::=
            DROP SCHEMA <schema name> <drop behavior>

<drop behavior> ::= CASCADE | RESTRICT
```

其中 CASCADE 和 RESTRICT 两者必选其一。选择了 CASCADE(级联)，表示在删除模式的同时把该模式中所有的数据库对象全部删除。选择了 RESTRICT(限制)，表示如果该模式中已经定义了数据库对象(如表、视图等)，则拒绝该删除语句的执行；只有当该模式中没有任何数据库对象时才能执行 DROP SCHEMA 语句。

【例 3.2】　删除例 3.1 中创建的模式。

```
DROP SCHEMA "Online-Shopping" CASCADE
```

3.2.2　基本表的定义、修改与删除

创建表的基本语法，至少要包含 CREATE TABLE、表的名称（<table name>）和<table element list>。

```
<table definition> ::=
    CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE <table name>
              <table element list>
              [ ON COMMIT { DELETE | PRESERVE } ROWS ]
```

<table element list>的内容是一对圆括号，其中包含一个或多个<table element>，中间由逗号分隔。

```
<table element list> ::=
    <left paren> <table element> [ { <comma> <table element> }... ] <right
paren>
```

＜table element＞可以是＜column definition＞或＜table constraint definition＞。

```
<table element> ::=
    <column definition> | <table constraint definition>
```

＜column definition＞至少要包含＜column name＞，并定义其＜data type＞ 或＜domain name＞。

```
<column definition> ::=
            <column name> { <data type> | <domain name> }
            [ <default clause> ]
            [ <column constraint definition>... ]
            [ <collate clause> ]
```

＜column constraint definition＞至少要包含＜column constraint＞。

```
<column constraint definition> ::=
            [ <constraint name definition> ]
            <column constraint>
              [ <constraint attributes> ]
```

＜column constraint＞可以是 NOT NULL(不允许空值)、＜unique specification＞(唯一性约束)、＜references specification＞(外键约束)或者＜check constraint definition＞(check 约束)。

```
<column constraint> ::=
              NOT NULL
            | <unique specification>
            | <references specification>
            | <check constraint definition>
```

＜table constraint definition＞必须包含＜table constraint＞。

```
<table constraint definition> ::=
            [ <constraint name definition> ]
            <table constraint> [ <constraint attributes> ]
```

＜table constraint＞可以是＜unique constraint definition＞、＜referential constraint definition＞或＜check constraint definition＞中的一个。

```
<table constraint> ::=
              <unique constraint definition>
            | <referential constraint definition>
            | <check constraint definition>
```

【例 3.3】 创建顾客表。

```
CREATE TABLE Customers(
```

```
    CID VARCHAR(32) PRIMARY KEY,
    CName VARCHAR(128) NOT NULL,
    City VARCHAR(128)
);
```

顾客表包含 3 个属性。CID 表示顾客 ID，数据类型是 VARCHAR，表示其为可变长度的字符串，最大长度为 32。PRIMARY KEY 是列级约束条件（<column constraint>），表示这是主码。CName 表示顾客姓名，后面的 NOT NULL 也是列级约束条件，表明其不可为空值。

【例 3.4】 创建供应商表。

```
CREATE TABLE Suppliers(
    SID VARCHAR(32) PRIMARY KEY,
    SName VARCHAR(128) NOT NULL,
    City VARCHAR(128)
);
```

供应商表包含 3 个属性。SID 是供应商 ID，为主码；SName 是供应商名称；City 是供应商所在城市。

【例 3.5】 创建商品表。

```
CREATE TABLE Products(
    PID VARCHAR(32) PRIMARY KEY,
    PName VARCHAR(128) NOT NULL,
    Price DECIMAL,
    Category VARCHAR(128),
    SID VARCHAR(32) NOT NULL,
    FOREIGN KEY (SID) REFERENCES Suppliers(SID)
);
```

商品表包含 5 个属性。PID 是商品 ID，为主码；PName 是商品名称，不可为空值；Price 是商品单价，类型为带小数的数字；Category 是商品类别；SID 是供应商 ID，不可为空值。最后一行定义了一个外码约束，表示 SID 属性是一个外码，引用了 Suppliers 表的 SID 属性。

【例 3.6】 创建订单表。

```
CREATE TABLE Orders(
    OID VARCHAR(32) PRIMARY KEY,
    CID VARCHAR(32),
    CreateTime DATETIME,
    FOREIGN KEY (CID) REFERENCES Customers(CID)
);
```

订单表包含 3 个属性。OID 是订单 ID，为主码；CID 是顾客 ID；CreateTime 是订单创建时间，DATETIME 是时间数据类型。最后一行定义了一个外码约束，表示 CID 属性是一个外键，引用了 Customers 表中的 CID 属性。

【例 3.7】 创建订单项表。

```
CREATE TABLE OrderItems(
    OID VARCHAR(32),
    PID VARCHAR(32),
    Quantity INT,
    Discount DECIMAL,
    PRIMARY KEY(OID, PID),
    FOREIGN KEY (OID) REFERENCES Orders(OID),
    FOREIGN KEY (PID) REFERENCES Products(PID)
);
```

订单项表包含 4 个属性。OID 是订单 ID;PID 是商品 ID;Quantity 是数量,类型为整数;Discount 是折扣。表的主码由 OID 和 PID 共同组成。OID 和 PID 都是外码,分别引用了订单表的 OID 和商品表的 PID。

3.2.3 索引的创建

创建索引需要指定索引的名称、表的名称和属性列的名称。若指定关键词 CLUSTERED,则将创建聚簇索引。聚簇索引要求 RDBMS 根据指定属性列的值存储元组,因此每个关系最多只能有一个聚簇索引。

```
CREATE [CLUSTERED] INDEX  name  ON table_name
    ( { column_name | ( expression ) }
```

【例 3.8】 在顾客表的姓名列上创建索引。

```
CREATE INDEX idx-cname ON Custermers (CName);
```

3.3 基本 SQL 查询

3.3.1 单表查询

查询是 SQL 中语义最为丰富,表达最为灵活的部分之一。下面简单介绍 SQL 标准中的查询部分。

```
<query specification> ::=
            SELECT [ <set quantifier> ] <select list> <table expression>
```

```
<select list> ::=
              <asterisk>
            | <select sublist> [ { <comma> <select sublist> }... ]
```

```
<select sublist> ::=
              <derived column>
            | <qualifier> <period> <asterisk>
```

查询以 SELECT 开始，必须包含＜select list＞和＜table expression＞。

＜select list＞可以是＜asterisk＞(即 *)或＜select sublist＞。＜select sublist＞包含＜derived column＞或 ＜qualifier＞ ＜period＞ ＜asterisk＞。

```
<asterisk> ::= *
```

```
<derived column> ::= <value expression> [ <as clause> ]
```

```
<as clause> ::= [ AS ] <column name>
```

＜derived column＞由＜value expression＞定义，＜as clause＞即 AS 加上自己定义的列名。＜value expression＞指一个表达式，可以是数值型、字符串型、日期型或区间型，可参考后面的例子。

＜table expression＞描述了从哪些表中查询数据，至少包含一个＜from clause＞，可以包含＜where clause＞、＜group by clause＞或＜having clause＞。注意：使用＜having clause＞必须同时使用＜group by clause＞。

```
    <table expression> ::=
            <from clause>
            [ <where clause> ]
            [ <group by clause> ]
            [ <having clause> ]
        <order by clause> ::=
            ORDER BY <sort specification list>
```

查询后面可以加上＜order by clause＞对结果进行排序。

鉴于查询语义的复杂性，为便于读者理解，下面给出查询语句的一般格式。

```
SELECT [ALL|DISTINCT] <目标列表达式> [别名][,<目标列表达式>[别名]]...
FROM <表名或视图名>[别名][,<表名或视图名>[别名]]...
[WHERE <条件表达式>]
[GROUP BY <列名 1> [HAVING <条件表达式>]]
[ORDER BY <列名 2> [ASC|DESC]]
```

【例 3.9】　查询所有顾客信息，结果如表 3.1 所示。

```
SELECT * FROM Customers;
```

表 3.1　例 3.9 查询结果

CID	CName	City
1001	张三	北京
1002	李四	广州
1003	王五	上海

【例 3.10】　查询订单 ID 及其创建的年份，结果如表 3.2 所示。

```
SELECT OID, YEAR(CreateTime) AS 年份;
```

上面的查询语句中，假定 YEAR 是一个函数，其输入为一个 DATETIME 类型数据，输出为对应年份。注意：这是<value expression>的示例。“AS 年份”表示将输出的列更名为“年份”。

表 3.2 例 3.10 查询结果

OID	年　份
O001	2023
O002	2023
O003	2023
O004	2023

【例 3.11】 查询价格低于 1000 元的商品，按价格从高到低排序，结果如表 3.3 所示。

```
SELECT * FROM Products WHERE Price < 1000 ORDER BY Price DESC;
```

上面的查询用到了 WHERE 子句和 ORDER BY 子句，其中“ORDER BY Price DESC”表示按价格从高到低排序。DESC 是降序的意思，默认是 ASC(升序)。

表 3.3 例 3.11 查询结果

PID	PName	Price	Category	SID
P0002	老人专用手机	899	数码产品	S001
P0005	流浪太阳	65	书籍	S003
P0004	数据库教材	48	书籍	S003

上面是关于单表查询的简单示例，GROUP BY 在后面聚集查询时再给例子解释。

3.3.2 连接查询

```
<joined table> ::=
                  <cross join>
              | <qualified join>
              | <left paren> <joined table> <right paren>
```

```
<cross join> ::=
             <table reference> CROSS JOIN <table reference>
```

SQL 标准允许从单个表或多个表中查询，若用到多个表，就涉及连接查询(joined table)。

```
<qualified join> ::=
            <table reference> [ NATURAL ] [ <join type> ] JOIN
              <table reference> [ <join specification> ]
```

连接有两种类型，笛卡儿积(＜cross join＞)和限定连接(＜qualified join＞)。笛卡儿积中的两个表(或是表的引用)用 CROSS JOIN 连接。通常用得最多的是限定连接，语法相对复杂一些。

```
<join type> ::=
              INNER
            | <outer join type> [ OUTER ]
            | UNION
```

连接类型(＜join type＞)可以是内连接(INNER)、外连接(OUTER)，外连接又分左外连接(LEFT)和右外连接(RIGHT)以及并(UNION)。其中 UNION 的用法示例将在 3.3.3 节介绍。

```
<join specification> ::=
              <join condition>
            | <named columns join>
```

```
<named columns join> ::=
            USING <left paren> <join column list> <right paren>
```

```
<join condition> ::= ON <search condition>
```

＜join specification＞描述两个表在哪个属性对上进行连接。

【例 3.12】　用 SQL 描述 Customers 和 Suppliers 的笛卡儿积，结果如表 3.4 所示。

```
SELECT * FROM Customers CROSS JOIN Suppliers;
```

表 3.4　例 3.12 查询结果

CID	CName	Customers.City	SID	SName	Suppliers.City
1001	张三	北京	S001	华北手机厂	北京
1002	李四	广州	S001	华北手机厂	北京
1003	王五	上海	S001	华北手机厂	北京
1001	张三	北京	S002	西北电子厂	西安
1002	李四	广州	S002	西北电子厂	西安
1003	王五	上海	S002	西北电子厂	西安
1001	张三	北京	S003	未央印刷厂	长安
1002	李四	广州	S003	未央印刷厂	长安
1003	王五	上海	S003	未央印刷厂	长安

从上面的结果可以看出，笛卡儿积通常没有实际意义。

【例 3.13】 查询“张三”创建的所有订单，显示 CID、CName、OID，以及 CreateTime，结果如表 3.5 所示。

```
SELECT Customers.CID,CName,OID,CreateTime
    FROM Customers INNER JOIN Orders ON Customers.CID = Orders.CID
    WHERE CName = '张三';
```

表 3.5 例 3.13 查询结果

CID	CName	OID	CreateTime
1001	张三	O001	2023-1-1 18:00:10

说明：在后续 SQL 标准，以及很多 RDBMS 产品实现中，也可以不显式地书写 JOIN，而是在 WHERE 子句中增加一个连接条件。比如，例 3.13 也可以写成如下形式：

```
SELECT Customers.CID,CName,OID,CreateTime
    FROM Customers,Orders
    WHERE Customers.CID= Orders.CID
    AND CName = '张三';
```

【例 3.14】 查询供应商所供应的商品信息，显示供应商的全部信息以及其供应商品的 ID，要求将没有供应商品的供应商也显示出来，结果如表 3.6 所示。

```
SELECT Suppliers.*, PID
      FROM Suppliers LEFT OUTER JOIN Products ON Suppliers.SID =
      Products.SID;
```

因为要显示没有供应商品的供应商，上面采用了左外连接。假设存在一个 ID 为 S004，位于延安，名为宝塔山印刷厂的供应商，其没有供应任何商品。

表 3.6 例 3.14 查询结果

SID	SName	City	PID
S001	华北手机厂	北京	P0002
S002	西北电子厂	西安	P0003
S003	未央印刷厂	长安	P0004
S003	未央印刷厂	长安	P0005
S004	宝塔山印刷厂	延安	NULL

3.3.3 集合查询

```
<non-join query expression> ::=
        <non-join query term>
        |<query expression> UNION [ALL][<corresponding spec>]<query term>
        |<query expression> EXCEPT[ALL][<corresponding spec>]<query term>
```

可以使用 UNION 或 EXCEPT 将两个查询的结果合并起来，此时要求两个查询结果有相同数量的列，且对应列的数据类型相同。UNION 表示将两个查询的结果合并，EXCEPT 则表示从第一个查询结果中删去第二个查询的结果。

【例 3.15】　查询价格低于 50 或价格高于 1500 的商品信息，结果如表 3.7 所示。

```
SELECT * FROM Products WHERE Price <50
    UNION
SELECT * FROM Products WHERE Price > 1500
```

表 3.7　例 3.15 查询结果

PID	PName	Price	Category	SID
P0004	数据库教材	48	书籍	S003
P0001	智能手机	1999	数码产品	S001
P0003	平板电脑	1688	数码产品	S002

3.3.4　空值查询

判断一个属性是否为空，不能用等号或不等号，而必须使用 IS NULL 或 IS NOT NULL。

【例 3.16】　查询类别为空值的商品。该查询结果为空，如表 3.8 所示。

```
SELECT * FROM Products WHERE Category IS NULL;
```

表 3.8　例 3.16 查询结果

PID	PName	Price	Category	SID

3.3.5　聚集查询

在查询定义的＜value expression＞中，可以使用聚集函数，定义如下。

```
<set function specification> ::=
                COUNT <left paren> <asterisk> <right paren>
              | <general set function>
```

```
<general set function> ::=
          <set function type>
             <left paren> [ <set quantifier> ] <value expression> <right
             paren>
```

```
<set quantifier> ::= DISTINCT | ALL
```

根据定义，SQL 标准包含 5 个聚集函数，分别是 AVG、MAX、MIN、SUM 和 COUNT。可以指定 DISTINCT，去掉重复值。默认是 ALL，即所有值参与运算。

```
<set function type> ::=
            AVG | MAX | MIN | SUM | COUNT
```

【例 3.17】 查询最高的商品价格，结果如表 3.9 所示。

```
SELECT MAX(Price) AS 最高价格 FROM Products;
```

表 3.9 例 3.17 查询结果

最高价格
1999

【例 3.18】 查询不同种类的商品的数量，结果如表 3.10 所示。

```
SELECT Category, COUNT( * ) AS 数量
    FROM Products
    GROUP BY Category;
```

表 3.10 例 3.18 查询结果

Category	数　量
数码产品	3
书籍	2

注意：本例用到了 GROUP BY，先对结果分组，再将聚集函数分别作用于各组。

3.4 数据更新

数据更新指向数据库中插入数据，修改已有数据或删除已有数据，分别通过 INSERT、UPDATE 或 DELETE 实现。

3.4.1 插入数据

```
<insert statement> ::=
            INSERT INTO <table name>
              <insert columns and source>
```

```
<insert columns and source> ::=
              [ <left paren> <insert column list> <right paren> ]
            <query expression>
    | DEFAULT VALUES
```

```
<insert column list> ::= <column name list>
```

插入语句以 INSERT INTO 开始，后面是表的名称，之后可以加上“(＜insert column list＞)”，即插入的属性列。后面跟一个查询表达式，或是由 VALUES 引出的一组插入的值(即一个元组)。

【例 3.19】　在供应商表中新增一条记录，其供应商 ID 为 S004，名称为宝塔山印刷厂，城市为延安。

```
INSERT INTO Suppliers VALUES('S004','宝塔山印刷厂','延安')
```

上例中，由于省略了<insert column list>，后面 VALUES 包含的元组必须包含 Suppliers 的全部属性，且按照定义该关系时所给属性列的顺序列出。

【例 3.20】　新增一个关系 TotalPrice，记录各订单的总价，然后根据 Products 和 OrderItems 关系查询所有订单的总价。

```
CREATE TABLE TotalPrice(
    OID VARCHAR(32) PRIMARY KEY,
    TotalPrice DECIMAL
);

INSERT INTO TotalPrice (OID, TotalPrice)
SELECT OID, SUM(Quantity * Price * Discount)
    FROM Products INNER JOIN OrderItems
    ON Products.PID = OrderItems.PID
    GROUP BY OID;
```

上例中，先是创建了一个关系 TotalPrice，包含 OID 和 TotalPrice 两个属性。接下来，执行一个插入语句，该语句将里面的子查询结果插入 TotalPrice 中。该子查询在 Products 和 OrderItems 之间做了连接，并按 OID 进行分组，最后计算每组（即每个订单）的数量乘以价格再乘以折扣之和，即订单总价。

3.4.2　修改数据

```
<update statement: searched> ::=
            UPDATE <table name>
              SET <set clause list>
              [ WHERE <search condition> ]
```

UPDATE 语句有两类：一类是 positioned 版本，与游标相关；另一类是 searched 版本。这里只介绍 searched 版本。UPDATE 语句必须指明要更新的表名，然后用 SET 引出修改动作，后面可以加一个 WHERE 子句，指明对哪些元组进行更新。

【例 3.21】　由于数码产品降价，将所有类别为数码产品的商品价格调整为原价的 90%。

```
UPDATE Products
    SET Price = Price * 0.9
    WHERE Category = '数码产品';
```

3.4.3　删除数据

```
<delete statement: searched> ::=
            DELETE FROM <table name>
```

```
[ WHERE <search condition> ]
```

同样，DELETE 语句也有 positioned 版本，与游标相关。此处只介绍 searched 版本。DELETE 语句很简单，只需要指明删除数据所在的表，可以加上 WHERE 语句，指定要删除的元组。

【例 3.22】 从 OrderItems 表中删除所有 OID 为'O001'的数据。

```
DELETE FROM OrderItems
WHERE OID = 'O001';
```

3.5 视图

3.5.1 定义视图

```
<view definition> ::=
            CREATE VIEW <view name> [ <left paren> <view column list>
            <right paren> ]
              AS <query expression>
              [ WITH [ <levels clause> ] CHECK OPTION ]
```

定义视图以 CREATE VIEW 开始，后面视图名称，之后可以加上选择的属性列表。若没有指定属性列表，则视图包含的属性就是之后的查询结果。之后由 AS 关键字引出一个查询表达式。最后如果指定 WITH CHECK OPTION，则视图应该是可以更新的。通过该视图插入或更新后的元组需要满足视图定义。默认是 CASCADED，即该视图和其所依赖的视图都应满足此条件。若指定 LOCAL，则只检查当前视图。

```
<levels clause> ::=
            CASCADED | LOCAL
      <view column list> ::= <column name list>
```

【例 3.23】 定义一个视图，存放订单的总价。

```
CREATE VIEW view_TotalPrice
    AS
    SELECT OID, SUM(Quantity * Price * Discount) AS TotalPrice
       FROM Products INNER JOIN OrderItems ON Products.PID = OrderItems.PID
       GROUP BY OID;
```

上例中，定义了一个名为 view_TotalPrice 的视图，没有指定属性列表，因此其包含后面查询的两个列，即 OID 和总价。注意：该视图包含了分组聚集的结果，因此是不能更新的。

【例 3.24】 定义一个视图，里面只存放所有类别是“数码产品”的商品信息。

```
CREATE VIEW view_DigitalProducts
    AS
```

```
    SELECT *
        FROM Products
        WHERE Category = '数码产品'
        WITH CHECK OPTION;
```

上例中，定义了一个名为 view_ DigitalProducts 的视图，没有指定属性列表，因此其包含后面查询的所有列。注意：该视图可以更新，也指定了 WITH CHECK OPTION。

3.5.2　查询视图

查询视图与查询基础表的语法是相同的。

【例 3.25】　根据已定义的视图 view_TotalPrice，查询编号为'O001'的订单总价，结果如表 3.11 所示。

```
SELECT *
    FROM view_TotalPrice
    WHERE OID = 'O001'
```

表 3.11　例 3.25 查询结果

OID	总　　价
O001	3148.15

3.5.3　更新视图

对于可更新的视图，其更新语法与更新基础表相同。

【例 3.26】　根据视图 view_ DigitalProducts 进行更新，修改 ID 为 P0001 的商品，将其价格降价 100 元，更新后查询视图 view_ DigitalProducts，其结果如表 3.12 所示。

```
UPDATE view_DigitalProducts
SET Price = Price - 100
WHERE PID  ='P0001';
```

表 3.12　例 3.26 查询结果

PID	PName	Price	Category	SID
P0001	智能手机	1899	数码产品	S001

3.6　本章小结

本章主要介绍了 SQL，包括使用 SQL 定义数据、查询数据和更新数据，此外也介绍了如何定义和使用视图。SQL 可以认为是 RDBMS 提供的标准接口，是用户与 RDBMS 交互的最重要工具。因此本章是全书重点之一，建议读者仔细阅读 SQL 语法，并在实际的 RDBMS 系统中反复练习，熟练掌握。

本章介绍的语法主要基于 SQL92 标准，RDBMS 在具体实现时与该标准多少存

在一定差异。读者在进行上机练习时，一定要认真阅读 RDBMS 提供的 SQL 参考手册。另外后续还有若干 SQL 标准，增加了很多新的特性，例如派生表等。读者可以根据实际需要，学习相关资料。限于篇幅，本书不对新的特性予以展开讲解。

本章难点在于查询语句。SQL 查询非常灵活，可以有 WHERE、ORDER BY、GROUP BY 等多个子句。读者可以结合关系代数理解每个子句的语义，并通过上机实验大量练习。

习题

1. 了解 TPCC 性能测试，在关系数据库中创建一个名为 TPCC 的模式。

2. 在 TPCC 模式中，使用 SQL 语句建立 TPCC 场景所包含的多个关系。

3. 对 2.1.3 节介绍的“网上商城”数据库，使用 SQL 语句完成下面的查询。

(1) 查询所有价格大于 500 元的数码产品，结果显示商品 ID、商品名称和价格，并按价格降序排列。

(2) 查询所有来自北京的顾客所下的订单，结果显示订单 ID、顾客姓名。

(3) 查询所有数码产品的销量，结果显示商品 ID、商品名称、总销量。其中一个商品的总销量表示其在所有订单中的数量之和。

(4) 查询不同类别商品的种数，结果显示类别和该类别商品的种数。

(5) 查询商品及供应商信息，结果显示商品名称、供应商 ID 和供应商名称。

(6) 查询所有订单的实际销售额，结果显示订单 ID、顾客 ID 和实际销售额。其中实际销售额计算方式为订单中每种商品的数量×价格×折扣之和。

4. 修改“网上商城”数据库商品信息，将所有商品的价格下调 10%。

5. 建立一个视图，包含订单 ID、顾客 ID 和订单总金额。

实验

读者可以选择一个合适的关系数据库管理系统，包括但不限于 Oracle、SQL Server、MySQL、OpenGauss 以及 KingBaseES，认真阅读其 SQL 参考手册，完成上述习题内容。读者也可以比较所选择 RDBMS 所支持的 SQL 语法与标准 SQL 语法的差异。