# 简易关系数据库管理系统

## 系统简介

关系数据库管理系统（DBMS）是通过SQL语言对数据库进行查询和修改操作的系统。将数据组织成表的形式呈献给用户的形式称为关系（relation）。在本项目中，我们通过文本文件来模拟数据库，要求实现一个简化了的关系数据库管理系统。

SQL语言可以对关系数据库中的数据进行**查询（包括简单查询、投影、模式匹配、比较查询、查询结果排序、查询的并、交、差等），连接，消除重复，删除，修改等操作。这些语句支持嵌套，SQL语言也包含循环结构。**

## SQL语言基本知识

W3C为SQL提供了一个非常易读的在线教程，条理清晰，易于学习，推荐各位同学参考：<http://www.w3school.com.cn/sql/index.asp>。为了完成本项目的基本要求，仅学习第一部分“SQL基础”即可。下面将集中地、较为简短地展示各个语句的功能和写法。

为了方便描述，我们有如下几个数据表：

**Movies(title, year, length, genre, studioName, producerC#)**

**StarsIn(movieTitle, movieYear, starName)**

**MovieStar(name, address, gender, birthdate)**

**MovieExec(name, address, cert#, netWorth)**

**Studio(name, address, pressC#)**

例如Studio(name, address, pressC#)表示这个数据表有三列，三列的标题分别是name, address和pressC#。随后的每行中都有三列，每一列由空格隔开，保证各列内容中不包含空格，每行表示一个Studio的情况。

**SQL对大小写不敏感**。

SQL语句的标准分割使用分号完成的，但是在一些系统中分号不是必须的，我们的测试中保证每一个语句都有分号。

### 简单查询

简单查询是找出表中特定条件的一部分，使用三个保留字SELECT，FROM和WHERE来表示。如在关系Movies中找出Disney在1990年出品的电影，SQL语句为：

**SELECT \***

**FROM Movies**

**WHERE studioName = ‘Disney’ AND year = 1990;**

查询使用这样的select-from-where模式。FROM给出查询所引用的关系，这里是Movies，WHERE是一个条件子句，查询匹配的元组必须满足WHERE指定的条件，SELECT决定满足条件的元组的哪些属性应该在结果中列出，\*表示列出所有属性。

再看两个例子：

**SELECT title, length**

**FROM Movies**

**WHERE studioName = ‘Disney’ AND year = 1990;**

结果是一个两列的表，两个列的标题分别是title和length。

**SELECT title AS name, length AS duration**

**FROM Movies**

**WHERE studioName = ‘Disney’ AND year = 1990;**

结果也是一个两列的表，表的内容和上一个例子相同，但两个列的标题变成了name和duration。SQL语言中的内容可以进行运算，例如如果想把length的分钟数改为小时数，可以把SELECT改写为**SELECT title AS name, length\*0.166667 AS lengthInHours。**SELECT语句中也可以是常量，例如**SELECT title AS name, length\*0.166667 AS lengthInHours, ‘hrs.’ AS inHours**。生成一个三列的关系，第三列的内容全部为’hrs.’。

WHERE中的条件也可以是不等式，例如**WHERE(year > 1970 OR length < 90) AND studioName = ‘MGM’)**。字符串做比较的时候比较的是字典序。同时WHERE也支持模式匹配，用LIKE实现，如**WHERE title LIKE ‘Star \_\_\_\_’**表示查询电影名是9个字符的，前五个字符是Star和一个空格后面四个字符任意的电影。可以找出Star Wars和Star Trek等。LIKE语句的语法较为复杂，包含转义字符等内容，在此不进行详细介绍。

查询结果也可以基于某一个属性来排序，并且将其他属性跟在其后进行约束。在select-from-where语句后加上如下子句用于排序：

**ORDER BY <list of attributes>**

默认是升序，可以加上给某个属性加上**保留字DESC**按照降序排列。保留字ASC表示按升序排列，ASC可以省略。

**SELECT \***

**FROM Movies**

**WHERE studioName = ‘Disney’ AND year = 1990**

**ORDER BY length, title;**

上面代码将电影按照长度排序，如果长度相同按照电影名字排序。在排序的时候Movies里面所有的属性都可以用，不论这个属性是否被SELECT包含。ORDER BY后面也可以跟表达式，例如**ORDER BY A+B DESC;**

为了去除重复，在SELECT中使用DISTINCT关键字即可。如**SELECT DISTINCT \* FROM Movies;**将取出Movies中的所有不同的条目。

### 插入

**INSERT INTO 表名称 VALUE (值1,值2,…)**的功能是向指定的表中插入一行数据。例如向Person表中插入一个人：**INSERT INTO Persons VALUES ('Gates', 'Bill', 'Xuanwumen 10', 'Beijing')**。也可以在指定的列中插入数据，例如：**INSERT INTO Persons (LastName, Address) VALUES ('Wilson', 'Champs-Elysees')**表示将值插入一行，这一行只有LastName和Address两列具有指定的数据。

### 更新

**UPDATE 表名称 SET 列名称 = 新值 WHERE 约束条件;**。其中的列名称=值的格式都可以像前面的语句一样用逗号分开一次性修改多个值。

### 删除

格式为：**DELETE FROM 表名称 WHERE 约束条件;**

### 内连接

以下内容引自W3SCHOOL。

数据库中的表可通过键将彼此联系起来。主键（Primary Key）是一个列，在这个列中的每一行的值都是唯一的。在表中，每个主键的值都是唯一的。这样做的目的是在不重复每个表中的所有数据的情况下，把表间的数据交叉捆绑在一起。

请看 "Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id\_P** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

请注意，"Id\_P" 列是 Persons 表中的的主键。这意味着没有两行能够拥有相同的 Id\_P。即使两个人的姓名完全相同，Id\_P 也可以区分他们。

接下来请看 "Orders" 表：

|  |  |  |
| --- | --- | --- |
| **Id\_O** | **OrderNo** | **Id\_P** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 65 |

请注意，"Id\_O" 列是 Orders 表中的的主键，同时，"Orders" 表中的 "Id\_P" 列用于引用 "Persons" 表中的人，而无需使用他们的确切姓名。

"Id\_P" 列把上面的两个表联系了起来。

我们可以通过引用两个表的方式，从两个表中获取数据：谁订购了产品，并且他们订购了什么产品？

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons, Orders

WHERE Persons.Id\_P = Orders.Id\_P

结果集：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **OrderNo** |
| Adams | John | 22456 |
| Adams | John | 24562 |
| Carter | Thomas | 77895 |
| Carter | Thomas | 44678 |

除了上面的方法，我们也可以使用关键词 JOIN 来从两个表中获取数据。

如果我们希望列出所有人的定购，可以使用下面的 SELECT 语句：

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons

INNER JOIN Orders

ON Persons.Id\_P = Orders.Id\_P

ORDER BY Persons.LastName

结果集：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **OrderNo** |
| Adams | John | 22456 |
| Adams | John | 24562 |
| Carter | Thomas | 77895 |
| Carter | Thomas | 44678 |

### 语句的嵌套

SQL的各个语句都支持嵌套，以SELECT为例：**SELECT name FROM bbc WHERE region = (SELECT region FROM bbc WHERE name = 'Brazil');**是一个嵌套的例子。如果内层的SELECT取出的只有一个元素则不会出现问题，否则会出现问题。此时应当把region=改为region IN即可。可对上述语句进行组合嵌套。

## 实验要求

按照难度，要求分为三类，请根据自身情况选择实现。

1. 能够读入数据库和指令，执行上述中的查询、插入、更新、删除、内连接的基本操作（必选）；
2. 根据上述语句说明，参照附件中Mini SQL语句的BNF定义，实现Mini SQL语句的集合。可以使用使用词法和语法分析工具来处理，也可以不使用。
3. 在实现上两项的基础上，应用B树或B+树实现外排序，对大规模数据（文件比内存要大）的查询和结果输出操作进行支持（此时输出应以文件方式进行）。有能力的可以实现大规模数据的插入、更新、删除、内连接操作。有余力的可以支持更多SQL语句或实现其他拓展功能。

**实验报告和提交文件的要求包括：**

1. 报告中给出所涉及到的SQL语句和符号的说明（必选）；
2. 给出实现基本操作的算法描述和算法复杂度分析，其中算法描述不要使用源代码加注释的方法描述，可以采用自然语言、流程图等方式（必选）；
3. 针对所实现的查询系统，设计测试用例，测试用例包括两部分，一部分是SQL语句，另一部分是存储数据的文本文件，数据规模自由调整。若某些数据规模大于内存，请说明数据生成方法和格式，不必提交具体数据。（必选）；
4. 对所实现的系统进行优化，并给出优化说明，说明所使用的数据结构和算法，优化程度会影响评分（必选）；
5. Readme文件中给出编译和执行方法，如果按照文件说明不能执行，将会扣分（必选）；
6. 实现其他的SQL语句操作，如SQL函数、SQL IN、SQL Primary Key等（可选）；
7. 不需要实现图形界面，但是需要对输入、输出以及数据格式进行说明（必选）；
8. 实现SQL语句检错，包括语法错误检测和运行时错误报告（可选）；
9. 其他自主实现的功能（可选）。

## 实验示例

数据文件（格式可以自行定义，应当考虑一个文件中可以有多个表）：

Persons 3,4

LastName FirstName OrderNo

Adams John 22456

Adams John 24562

Carter Thomas 77895

Carter Thomas 44678

SQL语句输入：

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons

INNER JOIN Orders

ON Persons.Id\_P = Orders.Id\_P

ORDER BY Persons.LastName

输出（格式可以自行定义）：

LastName FirstName OrderNo

Adams John 22456

Adams John 24562

Carter Thomas 77895

Carter Thomas 44678