In this issue:

4  Defining the Content of the Undergraduate Systems Analysis and Design Course as Measured by a Survey of Instructors
   Timothy J. Burns, Ramapo College of New Jersey

18 A Relational Algebra Query Language For Programming Relational Databases
   Kirby McMaster, Weber State University
   Samuel Sambasivam, Azusa Pacific University
   Nicole Anderson, Winona State University

27 The Greening of the Information Systems Curriculum
   Patricia Sendall, Merrimack College
   Li-Jen Shannon, Sam Houston State College
   Alan R Peslak, Penn State University
   Bruce Saulnier, Quinnipiac University

46 Determining the Most Suitable E-Learning Delivery Mode for TUT Students
   Solomon Adeyemi Odunaike Tshwane University of Technology
   Daniel Chuene, Tshwane University of Technology

61 Beyond Introductory Programming: Success Factors for Advanced Programming
   Arthur Hoskey, Farmingdale State College
   Paula San Millan Maurino, Farmingdale State College

71 Systems in the Foundations of Information Systems Course to Retain Students and to Support the IS 2010 Model Curricula
   Gayla Jo Slauson, Colorado Mesa University
   Donald Carpenter, Colorado Mesa University
   Johnny Snyder, Colorado Mesa University

77 Culturally Sensitive IS Teaching: Lessons Learned to Manage Motivation Issues
   Wenshin Chen, Abu Dhabi University

86 Establishing and Applying Criteria for Evaluating the Ease of Use of Dynamic Platforms for Teaching Web Application Development
   Johnson Dehinbo, Tshwane University of Technology

97 Integrating SAP to Information Systems Curriculum: Design and Delivery
   Ming Wang, California State University

105 A Validation Study of Student Differentiation Between Computing Disciplines
   Michael Battig, Saint Michael’s College
   Muhammad Shariq, American University of Afghanistan
The Information Systems Education Journal (ISEDJ) is a double-blind peer-reviewed academic journal published by EDSIG, the Education Special Interest Group of AITP, the Association of Information Technology Professionals (Chicago, Illinois). Publishing frequency is quarterly. The first year of publication is 2003.

ISEDJ is published online (http://isedjorg) in connection with ISECON, the Information Systems Education Conference, which is also double-blind peer reviewed. Our sister publication, the Proceedings of ISECON (http://isecon.org) features all papers, panels, workshops, and presentations from the conference.

The journal acceptance review process involves a minimum of three double-blind peer reviews, where both the reviewer is not aware of the identities of the authors and the authors are not aware of the identities of the reviewers. The initial reviews happen before the conference. At that point papers are divided into award papers (top 15%), other journal papers (top 30%), unsettled papers, and non-journal papers. The unsettled papers are subjected to a second round of blind peer review to establish whether they will be accepted to the journal or not. Those papers that are deemed of sufficient quality are accepted for publication in the ISEDJ journal. Currently the target acceptance rate for the journal is about 45%.

Information Systems Education Journal is pleased to be listed in the 1st Edition of Cabell's Directory of Publishing Opportunities in Educational Technology and Library Science, in both the electronic and printed editions. Questions should be addressed to the editor at editor@isedj.org or the publisher at publisher@isedj.org.

2011 AITP Education Special Interest Group (EDSIG) Board of Directors

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alan Peslak</td>
<td>Penn State University</td>
<td>President 2011</td>
</tr>
<tr>
<td>Wendy Ceccucci</td>
<td>Quinnipiac University</td>
<td>Vice President</td>
</tr>
<tr>
<td>Tom Janicki</td>
<td>Univ of NC Wilmington</td>
<td>President 2009-2010</td>
</tr>
<tr>
<td>Scott Hunsinger</td>
<td>Appalachian State University</td>
<td>Membership Director</td>
</tr>
<tr>
<td>Michael Smith</td>
<td>High Point University</td>
<td>Secretary</td>
</tr>
<tr>
<td>Brenda McAleer</td>
<td>Univ of Maine Augusta</td>
<td>Treasurer</td>
</tr>
<tr>
<td>Michael Battig</td>
<td>Saint Michael's College</td>
<td>Director</td>
</tr>
<tr>
<td>George Nezlek</td>
<td>Grand Valley State University</td>
<td>Director</td>
</tr>
<tr>
<td>Leslie J. Waguespack Jr</td>
<td>Bentley University</td>
<td>Director</td>
</tr>
<tr>
<td>Mary Lind</td>
<td>North Carolina A&amp;T St Univ</td>
<td>Director</td>
</tr>
<tr>
<td>Li-Jen Shannon</td>
<td>Sam Houston State Univ</td>
<td>Director</td>
</tr>
<tr>
<td>S. E. Kruck</td>
<td>James Madison University</td>
<td>JISE Editor</td>
</tr>
<tr>
<td>Kevin Jetton</td>
<td>Texas State University</td>
<td>FITE Liaison</td>
</tr>
</tbody>
</table>
INFORMATION SYSTEMS EDUCATION JOURNAL

Editors

Wendy Ceccucci
Senior Editor
Quinnipiac University

Thomas Janicki
Publisher
Univ NC Wilmington

Don Colton
Emeritus Editor
Brigham Young University
Hawaii

Nita Brooks
Associate Editor
Middle Tennessee State University

Mike Smith
Associate Editor - Cases
High Point University

ISEDJ Editorial Board

Alan Abrahams
Virginia Tech

Mike Battig
Saint Michael's College

Gerald DeHonrdt II
Grand Valley State University

Janet Helwig
Dominican University

Mark Jones
Lock Haven University

Cynthia Martincic
Saint Vincent College

Brenda McAleer
University of Maine at Augusta

Monica Parzinger
St. Mary's University
San Antonio

Doncho Petkov
Eastern Connecticut State Univ.

Samuel Sambasivam
Azusa Pacific University

Mark Segall
Metropolitan State College of Denver

Li-Jen Shannon
Sam Houston State University

Karthikeyan Umapathy
University of North Florida

Laurie Werner
Miami University

Bruce White
Quinnipiac University

Charles Woratschek
Robert Morris University

Peter Y. Wu
Robert Morris University

©2011 EDSIG (Education Special Interest Group of the AITP)
A Relational Algebra Query Language For Programming Relational Databases

Kirby McMaster
mcmaster@weber.edu
CS Dept., Weber State University
Ogden, Utah 84408 USA

Samuel Sambasivam
ssambasivam@apu.edu
CS Dept., Azusa Pacific University
Azusa, California 91702 USA

Nicole Anderson
nanderson@winona.edu
CS Dept., Winona State University
Winona, Minnesota 55987 USA

Abstract

In this paper, we describe a Relational Algebra Query Language (RAQL) and Relational Algebra Query (RAQ) software product we have developed that allows database instructors to teach relational algebra through programming. Instead of defining query operations using mathematical notation (the approach commonly taken in database textbooks), students write RAQL query programs as sequences of relational algebra function calls. The RAQ software allows RAQL programs to be run interactively, so that students can view the results of RA operations. Thus, students can learn relational algebra in a manner similar to learning SQL—by writing code and watching it run.

Keywords: database, query, relational algebra, programming, SQL

1. INTRODUCTION

Most commercial database systems are based on the relational data model. Recent editions of database textbooks focus primarily on the relational model. In this dual context, the relational model for data should be considered the most important concept in an introductory database course.

The heart of the relational model is a set of objects called relations or tables, plus a set of operations on these objects (Codd, 1972).

Coverage of the relational model in database courses includes the structure of tables, integrity constraints, links between tables, and data manipulation operations (data entry and queries).

Classroom discussion of queries and query languages generally leads to a detailed presentation of SQL. Relational algebra (RA) as a query language receives less attention. In a survey of database educators, Robbert and...
Ricardo (2003) found that only 70% included RA in their courses, compared to 92% for SQL. Database textbooks provide substantially more material on SQL than on RA. An extreme case is the textbook by Hoffer, et al (2008), which provides two full chapters on SQL but does not mention RA.

**Why Teach Relational Algebra?**

There is almost universal agreement that SQL is an essential component of an introductory database course. But should we also teach relational algebra? There are several good reasons for doing so.

1. The main reason for teaching RA is to help students better understand the relational model. At the conceptual level, the relational model provides a flexible, adaptable way to query a database. The organization of data into tables, together with RA operations, provides the foundation for this flexibility.

Relational algebra is a query language, not a database design tool. However, an understanding of how RA operations can be performed on tables to extract information should help support database analysis and design decisions.

2. Knowledge of RA facilitates teaching and learning SQL as a query language. The basic syntax of the SQL SELECT statement provides an integrated framework for combining RA operations to express a query.

3. An understanding of RA can also be used to improve query performance. The query-processing component of a DBMS engine translates SQL code into a query plan that includes RA operations. The DBMS query optimizer, together with the database administer, can speed up query execution by reducing the processing time of the RA operations.

**How to Teach Relational Algebra?**

If an instructor decides to include relational algebra as a topic in a database course, a follow-up question is how to present this topic to students? RA coverage in leading database textbooks often takes a mathematical approach. For example, the texts by Connolly and Begg (2009), Elmasri and Navathe (2006), and Silberschatz, et al (2006) present RA concepts using mathematical notation. There are several problems with this form of representation.

Many database students are not comfortable with mathematical notation, such as the use of Greek letters (e.g. σ and π) in a new context. The mathematical approach often mixes *infix* notation (operator name is placed *between* two operands; e.g. table1 union table2) and *functional* notation (operator name is placed *before* the operands; e.g. project table3 cols) when performing multiple RA operations within a single expression. This makes the expressions difficult to interpret, and it disguises the procedural nature of RA.

More importantly, students cannot execute query programs written in the mathematical notation. There is no easy way to verify that the mathematical description of a query is correct.

The mathematical approach contrasts with how *programming* courses are taught. In a programming course, an important part of learning occurs when students write instructions for the computer and watch their code run. Errors in program execution provide feedback, which reduces the gap between a student’s perception of the problem and how the computer interprets the proposed solution.

To demonstrate how computer implementations differ from mathematical models, students need software to experiment with. Students learn mathematical and computational concepts more effectively when they can work with actual computer representations. As with other programming languages, this principle applies when we teach students how to query using relational algebra.

All major relational database products offer SQL as the primary query language. On the other hand, very few computer environments are available for developing and running RA programs. One database system to offer RA as a query language is LEAP (Leyton, 2010). The Rel DBMS (Voorhis, 2010) uses a form of RA called Tutorial D (Date and Darwen, 2007). A third choice is WinRDBI (Dietrich, 2001), which supports queries using RA and other query languages.

Each of the above systems enables RA queries within a specific database system. None allow you to use RA to query desktop databases. In this paper, we introduce a Relational Algebra Query Language (RAQL) and a custom Relational Algebra Query (RAQ) software product that can be used to query relational databases.
We first present a function-based syntax for writing RAQL query programs as sequences of RA operations. We outline the main features of the RAQ software. Next, we demonstrate how to use the software to execute RAQL programs. Finally, we give examples of RA concepts that can be taught using this approach.

2. RELATIONAL ALGEBRA PROGRAMMING

A RAQL query program consists of a set of statements that specify operations to perform on database tables. The statements are executed in a particular sequence to yield a result table that satisfies a query. Each statement might consist of a single relational algebra operation, or several operations can be combined into one "algebraic" expression. Rather than use complex expressions in query programs, we prefer to have each line of code perform a single RA operation. Our coding style reflects 2GL (assembly language) thinking more than 3GL thinking (e.g. Fortran, C).

We provide a library function for each RA operation. A RAQL program is written as a sequence of RA function calls. Each function has one or two input parameters that are tables, plus other input parameters as necessary. The output of each function is another table. Using functions to implement RA operations provides database students with a comfortable programming environment for creating RAQL query programs.

Functions are provided for the following relational algebra operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection</td>
<td>TSelect(Table1,RowCondition)</td>
</tr>
<tr>
<td>projection</td>
<td>TProject(Table1,ColumnList)</td>
</tr>
<tr>
<td>join</td>
<td>TJoin(Table1,Table2,JoinCondition)</td>
</tr>
<tr>
<td>union</td>
<td>TUnion(Table1,Table2)</td>
</tr>
<tr>
<td>intersection</td>
<td>TIntersect(Table1,Table2)</td>
</tr>
<tr>
<td>difference</td>
<td>TMinus(Table1,Table2)</td>
</tr>
<tr>
<td>product</td>
<td>TProduct(Table1,Table2)</td>
</tr>
<tr>
<td>division</td>
<td>TDivide(Table1,Table2)</td>
</tr>
<tr>
<td>rename</td>
<td>TRename(Table1,OldColumnName, NewColumnName)</td>
</tr>
</tbody>
</table>

To illustrate programming using RA functions, we require a sample database. The structure of a simple inventory database is described in the next section.

Relational Database Example

Suppose an INVENTORY database for a manufacturing environment consists of two tables, STOCK and STKTYPE. The diagram in Figure 1 describes the relational model for this database.

This data model assumes that inventory items are divided into categories, or types. Attributes that apply to individual items are recorded in the STOCK table. Attributes that apply to all items of the same type are included in the STKTYPE table. The two tables are linked by a common type code (SType and TType).

In this basic system, when the quantity-on-hand for an item drops below its reorder point, a production run of a predetermined lot-size is scheduled on a specific production line. It is assumed that reorder point, lot-size, and production line depend on the stock type rather than on the individual item. Whenever a production run is scheduled, the OnOrder field for the item is set to 'Y'. This field is reset to 'N' after the order is filled.

RA Query1 Program

Consider the following query for the INVENTORY database.

Query1: List the stock number, name, and quantity-on-hand for all items that are manufactured on production line 3.

A RAQL program for this query takes the form of a sequence of Table 1 function calls. Each function receives one or two tables as arguments and returns a temporary table. The temporary table can be used in later RA
operations. Sample code for this query is shown below:

```
-- RA Query1: Inventory Query - Line 3
T1 = TJoin("STOCK", "STKTYPE", "SType=TType")
T2 = TSelect(T1, "ProdLine=3")
T3 = TProject(T2, "StkNo,StkName,QtyOnHand")
```

An explanation of each line of code for this program follows:

**Line 1:** This is a comment (--)  
**Line 2:** The STOCK and STKTYPE tables are joined. The join condition states that the SType field in the STOCK table must match the TType field in the STKTYPE table. Actual table names are placed in matching single (or double) quotes, since they are fixed string values. The join condition is also placed in quotes. The output of the TJoin function is a cursor (a temporary table in memory). The cursor name is randomly generated and is assigned to variable T1. The name of the cursor is unknown to the programmer, but the cursor can be referred to in later program statements using the variable name.

**Line 3:** Rows of cursor T1 are then selected if they satisfy the condition that the ProdLine (production line) field in T1 equals 3. Quotes are not needed for the number 3. If quotes are needed inside a row condition, then single and double quotes should be nested in pairs (e.g. "OnOrder='N'"). The output cursor name is assigned to variable T2. The T1 argument is not placed in quotes, since T1 is a variable.

**Line 4:** The three columns of cursor T2 specified in the column list are projected as cursor T3, which is the final result table for the query.

### 3. RAQ COMPUTER SOFTWARE

The RAQ software allows us to execute queries written in the format of Query1. Our explanation of how to use RAQ to perform queries is organized according to the controls (textboxes and command buttons) on the RAQ main screen (see Figure 2).

1. **Database File textbox:** Choose a database file. The database must be in a Microsoft Access MDB file. The file can be selected with the file-chooser dialog box, which includes the ability to search in subdirectories. No other actions can be performed in the RAQ software until a valid database file is opened. Once a database is open, it cannot be changed without exiting and then rerunning the RAQ software.

2. **Query Program textbox:** Choose a RAQL query program. The program must be in a text file having a TXT extension. A new query program can be selected at any time during the execution of the RAQ software, but the actions that follow must be repeated.

3. **Display button:** Display the RAQL program code in a window. This command can be selected whenever the Display button is enabled. Press the Escape key to close the window. The display window is read-only. Any changes or corrections to the RAQL program must be made in a separate text editor.

![Database File (click to choose)](image1)

![Query Program (click to choose)](image2)

**Figure 2: RAQ Program Main Screen**

4. **Load button:** Before a RAQL program can be run, it must be loaded. This action can be repeated when you want to restart the program from the beginning.

5. **Step button:** Each click of this button executes one RAQL instruction. This will normally be a single relational algebra operation. Comments in the program code are skipped. For each successful RA operation, the resulting query output cursor is shown on the screen. Hit the escape key to close this view.

If an error occurs while trying to execute an instruction, an error message is displayed in the top right-hand corner of the screen. The error message shows the code for the line that was
just attempted. Clicking the Display button allows the user to see the error in the context of the full RAQL program.

6. **Save button**: When an operation has completed, the current output cursor can be saved to disk. The format of the saved file is an Excel XLS file. The name of the output file is the name of the RAQL program file, followed by the step number. The contents of the XLS file can be easily transferred to a word processing document or other data file.

7. **Exit button**: Click this button to exit the program. You will be prompted to confirm this request before the program ends.

The availability of most RAQ menu choices depends on which actions have already occurred during program execution. Textboxes and command buttons are disabled when their selection would be inappropriate. For example, a result table cannot be saved if the current command fails to execute correctly.

**Running the RA Query1 Program**

The previous discussion of RAQ controls and features was fairly general. To provide a more concrete demonstration, we list below one possible sequence of RAQ actions that could be taken to execute the Query1 program.

1. Load the INVENTORY database file. Sample data for this database is listed in Appendix A.
2. Load the text file that contains Query1. Assume this file is named RAQuery1.txt.
3. Click the Display button to view the query program code (optional).
4. Click the Load button to initialize the program.
5. Click the Step button. The comment line will be skipped, and the TJoin operation will be performed. The joined cursor T1 will appear in a window.
6. Click the Step button again, and the TSelect operation will produce cursor T2.
7. Click Step a third time, and TProject will produce and display cursor T3. The final result for Query1 is shown in Figure 3.
8. To save a result cursor, click the Save button after closing the window showing the cursor. If the final result in this example is saved, the output file name will be RAQuery1-3.xls.

**Figure 3: RA Query1 Final Result**

<table>
<thead>
<tr>
<th>Stkno</th>
<th>Stkname</th>
<th>QtyOnHand</th>
</tr>
</thead>
<tbody>
<tr>
<td>301</td>
<td>Mint-Truffles</td>
<td>116</td>
</tr>
<tr>
<td>303</td>
<td>Almond-Truffles</td>
<td>44</td>
</tr>
<tr>
<td>304</td>
<td>Mocha-Truffles</td>
<td>72</td>
</tr>
<tr>
<td>306</td>
<td>Mixed-Truffles</td>
<td>93</td>
</tr>
<tr>
<td>401</td>
<td>Chocolate-Fudge</td>
<td>145</td>
</tr>
<tr>
<td>404</td>
<td>Marble-Fudge</td>
<td>103</td>
</tr>
</tbody>
</table>

9. Click Exit when you are finished; then confirm when prompted. If you prefer, you can Load and rerun the same RAQL program, or choose a new query program.

Appendix B contains a sample Relational Algebra Project to give students experience writing and running RAQL programs.

### 4. USING RAQ TO TEACH RELATIONAL ALGEBRA CONCEPTS

The RAQ software can be used to teach relational algebra concepts interactively that are usually explained intuitively. The advantage of using RAQ is that students can visualize the concepts when they are implemented as RAQL programs. Some examples of RA concepts that can benefit from this approach are described next.

**Select Before Join**

When select is used before join, the size of the joined table will usually be much smaller than if the join operation is performed first. This will reduce the memory resources required for a query and should decrease processing time. The Query2 RAQL program shown below achieves the Query1 result, but starts with a select operation.

```-- RA Query2: Select before Join
T1 = TSelect(STKTYPE, "ProdLine=3")
T2 = TJoin(STOCK, T1, "SType=TType")
T3 = TProject(T2, "StkNo,StkName,QtyOnHand")
```

This program can be compared to the Query1 program, where the join operation is performed first. The relative size of the two joined cursors (T1 in Query1 and T2 in Query2) highlights the advantage of joining tables "later."

**Set Union and Intersection**

The union of sets A and B consists of all distinct members of A and B. In RA, the union of two tables does not include duplicate rows. This
concept can be illustrated with the following Query3 program.

```plaintext
-- RA Query3: Union and Intersection
TA = TSelect('STOCK', "QtyOnHand<50")
TB = TSelect('STOCK', "SType='C'")
T1 = TUnion(TA, TB)
T2 = TIntersect(TA, TB)

In this program, cursors TA and TB have identical attribute domains (union-compatible). The union cursor T1 does not contain duplicates of rows that satisfy both conditions. The intersection cursor T2 identifies the rows that are in both TA and TB.

Set Intersection and Difference

In set theory, it is known that the relationship between intersection and difference satisfies the equation

\[ A \cap B = A - (A - B) \]

The Query4 program listed below verifies this relationship.

```plaintext
-- RA Query4: Intersection and Difference
TA = TSelect('STOCK', "OnOrder='N'")
TB = TSelect('STOCK', "SType='T'")
T1 = TMinus(TA, TB)
T2 = TMinus(TA, T1)
T3 = TIntersect(TA, TB)

Here, cursor T1 is A - B, T2 is A - (A - B), and T3 is A \( \cap \) B. Students can observe that T2 and T3 are identical.

Product vs. Divide

The RA divide operation is sometimes described as the "inverse" of the product operation, in the sense that for tables A and B,

\[ (A \times B) \div B = A \]

The following Query5 program illustrates the nature of this relationship.

```plaintext
-- RA Query5: Product and Divide
TA = TSelect('STOCK')
TB = TSelect('STKTYPE')
T1 = TProduct(TA, TB)
T2 = TDivide(T1, TB)

In this code, T1 is A \( \times \) B and T2 is (A \( \times \) B) \( \div \) B. Students can note that cursor T2 is the same as cursor TA.

5. SPECIAL CONSIDERATIONS

The RAQ software is not a feature-rich, industrial-strength software product. It was designed for academic use to provide a simple way to teach relational algebra concepts through programming. Users of this software should be aware of certain limitations and special considerations.

1. There is a 100-line maximum for RAQL query programs (not including comments). Each instruction must be on a single line.
2. RAQ provides modest error checking. Error messages are displayed in the upper-right corner of the screen. Messages state the type of error or show the offending line of code.
3. RAQ has limited input options for data. The database must be in an Access MDB (not ACCDB) file. If necessary, convert the ACCDB file to MDB format. More generally, if the database is an ODBC data source (e.g. Oracle, SQL Server, MySQL), then the table structures and data can be imported into an Access file before using RAQ.
4. RAQL query programs must be in a text file with a TXT extension. Programs have to be created and modified with a separate text editor, since RAQ does not provide editing capabilities.
5. RAQ output for query results are shown on the screen. The display of intermediate cursors cannot be skipped, but RAQL programs are usually short. Query output can be saved in XLS files, and the Windows operating system provides various print-screen options.
6. For convenience in expressing RA queries, duplicate field names should be avoided in databases. If you prefer to have duplicate field names in separate tables (e.g. the same name for primary key and foreign key), use the TRename function in RAQL programs. This is a constraint inherent in relational algebra (Date, 2004) and not just in our RAQ software.
7. Nesting of RA function calls within a single statement is permitted but not
recommended. Nested function calls defeat the opportunity to see intermediate result cursors while RA operations are performed. Nesting also disguises the procedural nature of relational algebra.

8. The RAQ software has been tested in Windows XP, Windows Vista, and Windows 7. Administrative privileges may be required for Vista or Windows 7, since RAQ writes some temporary files to the disk.

6. SUMMARY AND CONCLUSIONS

In this paper, we presented arguments for including coverage of relational algebra (RA) along with SQL in database courses. We argued that, in teaching relational algebra to database students, a programming approach is preferable to a mathematical approach. Our recommended programming style is to write query programs in a special Relational Algebra Query Language (RAQL). In this language, query programs are expressed as sequences of function calls, where each call performs one RA operation. Following this format, students gain experience using a procedural query language while learning relational algebra.

Writing query programs improves the educational experience for students, but learning is enhanced if students can execute their query programs. We have developed a custom Relational Algebra Query (RAQ) software environment in which RAQL programs can be run. The RAQ software allows students to see the intermediate results during the sequence of relational algebra operations. With this capability, students can visualize RA concepts and explore performance issues. Thus, they can learn RA in a manner similar to how they learn SQL—by writing code and watching it run. As Knuth might say, students can better understand a problem by teaching a computer how to solve it (Shustek, 2008).

Note: An executable version of the RAQ program, along with runtime files and the database examples in this paper, can be obtained from the lead author.

7. REFERENCES


APPENDIX A: INVENTORY Database – Sample Data

### STOCK Table

<table>
<thead>
<tr>
<th>StkNo</th>
<th>SType</th>
<th>StkName</th>
<th>QtyOnHand</th>
<th>OnOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>B</td>
<td>Prune Basket</td>
<td>65</td>
<td>N</td>
</tr>
<tr>
<td>105</td>
<td>B</td>
<td>Pear Basket</td>
<td>48</td>
<td>N</td>
</tr>
<tr>
<td>107</td>
<td>B</td>
<td>Peach Basket</td>
<td>21</td>
<td>Y</td>
</tr>
<tr>
<td>202</td>
<td>W</td>
<td>Deluxe Tower</td>
<td>54</td>
<td>N</td>
</tr>
<tr>
<td>204</td>
<td>W</td>
<td>Special Tower</td>
<td>29</td>
<td>N</td>
</tr>
<tr>
<td>301</td>
<td>T</td>
<td>Mint Truffles</td>
<td>116</td>
<td>N</td>
</tr>
<tr>
<td>303</td>
<td>T</td>
<td>Almond Truffles</td>
<td>44</td>
<td>Y</td>
</tr>
<tr>
<td>304</td>
<td>T</td>
<td>Mocha Truffles</td>
<td>72</td>
<td>N</td>
</tr>
<tr>
<td>306</td>
<td>T</td>
<td>Mixed Truffles</td>
<td>93</td>
<td>N</td>
</tr>
<tr>
<td>401</td>
<td>F</td>
<td>Chocolate Fudge</td>
<td>145</td>
<td>N</td>
</tr>
<tr>
<td>404</td>
<td>F</td>
<td>Marble Fudge</td>
<td>103</td>
<td>N</td>
</tr>
<tr>
<td>502</td>
<td>C</td>
<td>Berry CheeseCake</td>
<td>73</td>
<td>N</td>
</tr>
<tr>
<td>505</td>
<td>C</td>
<td>Apple CheeseCake</td>
<td>46</td>
<td>N</td>
</tr>
<tr>
<td>506</td>
<td>C</td>
<td>Lemon CheeseCake</td>
<td>18</td>
<td>Y</td>
</tr>
<tr>
<td>508</td>
<td>C</td>
<td>Plain CheeseCake</td>
<td>65</td>
<td>N</td>
</tr>
</tbody>
</table>

### STKTYPE Table

<table>
<thead>
<tr>
<th>TType</th>
<th>TypeName</th>
<th>ReorderPt</th>
<th>LotSize</th>
<th>ProdLine</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Basket</td>
<td>60</td>
<td>90</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>CheeseCake</td>
<td>50</td>
<td>75</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>Fudge</td>
<td>120</td>
<td>180</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>Truffles</td>
<td>90</td>
<td>120</td>
<td>3</td>
</tr>
<tr>
<td>W</td>
<td>Tower</td>
<td>40</td>
<td>60</td>
<td>1</td>
</tr>
</tbody>
</table>
APPENDIX B: Relational Algebra Project

This project uses a Microsoft Access file that contains a Time-and-Billing database for the XFiles group in the FBI. The file is called XFILES.mdb. The database is used to track the number of hours spent by agents on cases. Each agent fills out her/his time card each day, charging up to 8 hours per day on cases.

The database consists of three tables: AGENT, CASES, and TIMECARD. The relational model for this database is shown in the following diagram.

1. Write a Relational Algebra Query Language (RAQL) program for each of the following queries:
   - **Query 1**: List the agent ID, last name, position, and bill rate of all Special Agents that have a bill rate greater than $75 per hour.
   - **Query 2**: List the case number, case title, and budget of all cases that have been worked on by a female agent.
   - **Query 3**: List the agent ID, last name, specialty, and bill rate of all agents that have worked on the Fat-Sucking Vampire case.
   - **Query 4**: List the work date, case title, agent ID, and hours for all time card records where less than 4 hours were charged.
   - **Query 5**: List the last name, first name, and gender of all agents that are female or have worked on the Bermuda Triangle case.
   - **Query 6**: List the last name, first name, and gender of all agents that are female and have worked on the Bermuda Triangle case.
   - **Query 7**: List the agent ID, last name, and specialty of all agents that have not worked on the Dark Matter case.
   - **Query 8**: List the agent ID, last name, and first name of all agents that have worked on every case that Scully has worked on.

2. Use the RAQ software to run your query programs. For each query, save the final output table in an Excel file.

3. Combine your query results in a Word document, grouping together for each query:
   - The word definition of the query.
   - The source code for your RAQL program.
   - The final output table from the query.