ISSN: 2167-1435
v29 n1906

Relational Algebra and SQL: Better Together

Kirby McMaster
kmcmaster@weber.edu
CSIS Dept, Fort Lewis College
Durango, CO 81301, USA

Samuel Sambasivam
ssambasivam@apu.edu
CS Dept, Azusa Pacific University
Azusa, CA91702, USA

Steven Hadfield
steven.hadfield@usafa.edu
CS Dept, US Air Force Academy
USAFA, CO 80840, USA

Abstract

In this paper, we describe how database instructors can teach Relational Algebra and Structured Query Language together through programming. Students write query programs consisting of sequences of Relational Algebra operations vs. Structured Query Language SELECT statements. The query programs can then be run interactively, allowing students to compare the results of Relational Algebra and equivalent Structured Query Language commands. In this way, students better understand both Relational Algebra and Structured Query Language—by writing code and watching it run.

Keywords: database, query, relational algebra, structured query language, SQL.

1. INTRODUCTION

Perhaps the most important topic in a first database course is Codd's relational model for data (Codd, 1970). Relational databases implement logical data structures called tables, and provide ways to perform data-entry and data-retrieval operations on the tables. Substantial class time is spent on how to create, maintain, and query a database.

Classroom discussion of query languages generally leads to a detailed examination of Structured Query Language (SQL). Relational Algebra (RA) as a query language usually receives less attention. This emphasis on SQL distorts database history.

When Codd introduced his relational model in 1970, the main focus was on data independence, with no mention of Relational Algebra or SQL. Two years later, Codd (1972) presented a detailed analysis of RA, along with Relational Calculus. IBM developed two prototype databases based on RA in England in the 1970s (Notley, 1972; Todd, 1976). More recent database systems that offer a form of RA as a query language include LEAP (Layton, 2010) and Rel (Voorhis, 2010).

The initial design of SQL (then called SEQUEL) was performed by Chamberlin and Boyce (1974) at IBM in the early 1970s. This led to the development of several IBM relational database

systems based on SQL, including a research prototype System R in the mid-1970s (Chamberlin, et. al, 1981) and the production system DB/2 in 1983. In 1979, Relational Software (now Oracle) introduced the first commercial implementation of SQL.

Why Teach Relational Algebra?

There is widespread agreement that SQL is an essential component of an introductory database course, but there is less support for Relational Algebra (Robbert & Ricardo, 2003). Nevertheless, there are several advantages to including RA in a database course.

1. Teaching RA helps students understand the relational model. The relational model with RA operations provides a consistent, powerful way to query a database. RA is not a database design tool, but it can support database analysis and design decisions.

2. Knowledge of RA facilitates teaching and learning the query portion of SQL. The basic syntax of the SQL SELECT statement provides an integrated way to combine RA operations to express a query.

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

When to Teach Relational Algebra?

Most database textbooks provide more material on SQL than on RA. Table 1 lists six multiple-edition database textbooks that have a Computer Science (CS) or Information Systems (IS) orientation. For each textbook, the table includes the approximate number of pages devoted to RA and to the query features of SQL. The page counts in Table 1 do not include non-query aspects of SQL. All except Date's book have more pages explaining SQL than RA.

Three of the textbooks in Table 1 introduce RA before SQL, while the other three explain SQL first. If an instructor covers RA first, then SQL can be introduced by showing how the RA operations can be performed with SQL. If SQL is presented first, then SQL query statements can be decomposed into a corresponding sequence of RA operations.

Table 1: Database Textbook Summary.

Database Textbook	SQL pages	RA pages	First topic
Connolly & Begg, 5th ed, 2010	34	13	RA
Date, 8th ed, 2004	34	45	SQL
Elmasri & Navathe, 6th ed, 2010	44	29	SQL
Ramakrishnan & Gehrke, 3rd ed, 2002	33	14	RA
Silberschatz, et al, 6th ed, 2010	42	22	SQL
Ullman & Widom, 3rd ed, 2008	48	30	RA

The SQL SELECT statement includes query operations that go beyond RA, such as ordering, grouping, and aggregate functions. If RA has been discussed first, these additional SQL operations can be presented as extensions to RA. If SQL has been covered first, then RA can be described as a subset of the query capabilities provided by SQL. In either case, a student's understanding of the relational model and query languages is improved when RA and SQL reinforce each other.

Other authors have presented innovative ways to teach SQL, such as by using case studies (Caldeira, 2008) or by combining SQL and Java to build web applications (Pereira, Raoufi & and Frost, 2012). Our emphasis in this paper is on teaching Relational Algebra. The advantage of using RA to help students learn SQL is a side benefit of our approach.

2. AN EXAMPLE DATABASE

To illustrate query programming with SQL and Relational Algebra, we define an example database. The logical structure of a Time-and-Billing system for the fictional X-Files group within the FBI consists of three tables: AGENT, CASES, and TIMECARD. The relational model for this XFILES database is shown in Figure 1. Primary keys are marked in bold.

In this data model, agents are assigned to various cases. Several agents often work

ISSN: 2167-1435
v29 n1906

together on a case, and each agent can work on multiple cases. The number of hours charged by an agent to a case is recorded each day in the TIMECARD table. Sample data for the XFILES database is given in the Appendix.

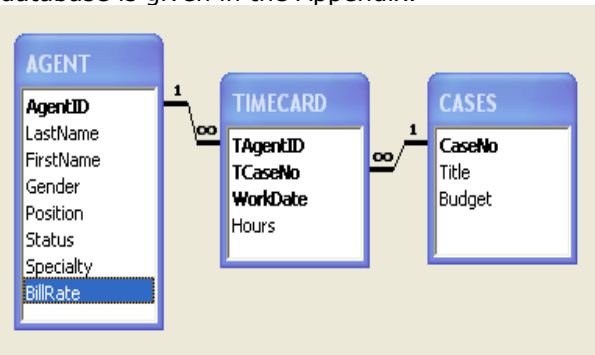

Figure 1: XFILES Database

Consider the following Query1 for the XFILES database.

Query1: List the agent ID and last name of all female agents that have worked on Case 2803.

An SQL statement to perform Query1 is shown below. This SELECT statement combines several RA operations (*select*, *project*, and *join*) into one command.

```
SELECT DISTINCT AgentID, LastName
FROM AGENT, TIMECARD
WHERE AgentID = TAgentID
  AND Gender = 'F'
  AND TCaseNo = 2803
```

3. RELATIONAL ALGEBRA QUERIES

Teaching SQL in a database course is relatively straightforward. Sections of the SELECT command can be covered in the order favored by the instructor or the textbook. When query execution is desired, many database products are available that implement SQL as the primary query language. This includes commercial systems such as Oracle and SQL Server, as well as free software such as MySQL. SQL queries can also be run in Microsoft Access using the SQL View screen.

If an instructor decides to include Relational Algebra in a database course, how should this topic be presented? RA coverage in leading database textbooks often takes a mathematical approach (Elmasri & Navathe, 2010), (Silberschatz, Korth, & Sudarshan, 2010), (Ullman & Widom, 2008). Most database students are not comfortable with a mathematical notation that uses Greek letters in new contexts, along with other strange symbols. A greater problem with the math syntax is that students cannot execute query programs written in the mathematical notation.

The mathematical approach for RA contrasts with the way SQL is taught. With SQL, an important part of learning occurs when students run their query statements. Errors in program execution provide feedback to help students reconcile their understanding of the problem with the proposed solution. Unfortunately, few computing environments are available for running Relational Algebra programs. One current system that does support a form of RA is LEAP (Leyton, 2010)

In this paper, we present a non-mathematical, function-based Relational Algebra language for writing query programs. We then describe a custom RA and SQL (RASQL) software environment that can execute both RA and SQL query programs on Microsoft Access databases.

RA Query Programs

In our Relational Algebra syntax, a query program consists of a sequence of statements that specify operations to perform on database tables. Each statement is a function call that performs one RA operation. Functions are defined for the nine RA operations listed in Table 2. We start each function name with the letter "T" to avoid conflicts with SQL keywords.

ISSN: 2167-1435
v29 n1906

Table 2: RA Query Functions

Operation	Function
selection	TSelect(Table1,RowCondition)
projection	TProject(Table1,ColumnList)
join	TJoin(Table1,Table2,JoinCondition)
union	TUnion(Table1,Table2)
intersection	TIntersect(Table1,Table2)
difference	TMinus(Table1,Table2)
product	TProduct(Table1,Table2)
division	TDivide(Table1,Table2)
rename	TRename(Table1,OldColumnName, NewColumnName)

Each RA function receives one or two tables as input and returns a temporary table. The temporary table can be used in later RA operations. Using function calls for operations provides a familiar programming environment for CS and IS students.

A sample RA program for Query1 using these functions is shown below.

```
-- Query1: XFILES Database
T1 = TJoin('AGENT','TIMECARD',
     "AgentID=TAgentID")
T2 = TSelect(T1,"Gender='F'")
T3 = TSelect(T2,"TCaseNo=2803")
T4 = TProject(T3,"AgentID,LastName")
```

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

Line 1: This is a comment (--)

Line 2: The AGENT and TIMECARD tables are *joined* based on the condition that the AgentID (PK) field matches the TAgentID (FK) field. The output table is assigned to variable T1.

Line 3: Rows of table T1 are then *selected* when the Gender field value is 'F' (female). The output table variable is named T2.

Line 4: Rows of table T2 are *selected* when the TCaseNo field equals 2803. The output table is assigned to variable T3.

Line 5: The two attributes of table T3 specified in the column list are *projected* as table T4 (the result table for the query).

RASQL Software

Our RA and SQL (RASQL) query software allows us to execute queries written in the RA format demonstrated by the Query1 program, as well as SQL SELECT statements. An earlier version of the software provided the ability to run RA query programs, but not SQL.

Our explanation of how to use RASQL is presented in order of the controls that appear on the Main Screen (Figure 2).

1. *Database File* textbox: Select a database. The database must be in an Access MDB (not ACCDB) file. We chose this database format because MDB files are easy to distribute to students.

2. *Query Program* textbox: Choose a query program consisting of a sequence of RA and/or SQL statements. Each RA instruction must be on a single line. SQL statements can span multiple lines. Query programs must be in a text file with a TXT extension.

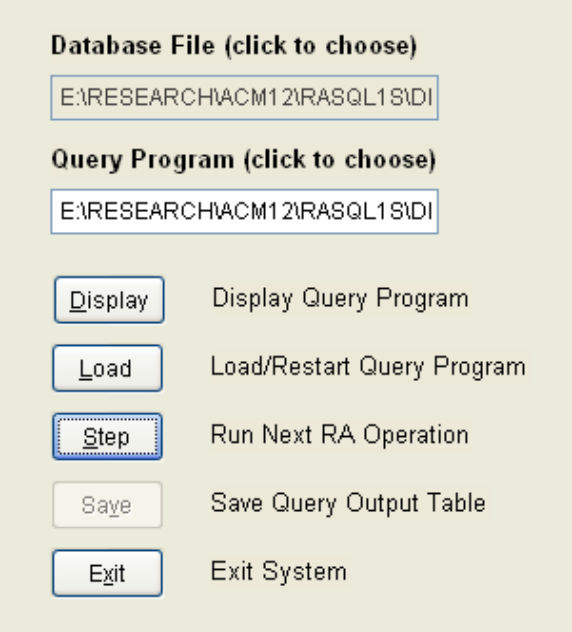

Figure 2: RASQL Program Main Screen

3. *Display* button: Display the query program code in a window (read-only). Use a separate text editor to create and modify the programs.

4. *Load* button: Before a query program can run, it must be loaded (initialized). Repeating

ISSN: 2167-1435
v29 n1906

this action restarts the program from the beginning.

5. *Step* button: Each click of this button executes one RA or SQL instruction. Comments in the program code are skipped. The output table for each step is shown on the screen.

6. *Save* button: When an RA or SQL instruction has successfully completed, the current output table can be saved to disk as an Excel XLS file.

7. *Exit* button: Click this button to exit the RASQL system.

The final output table from the RA program for Query1, using the data in the sample XFILES database, is presented in Figure 3. The result table for the Query1 SELECT statement (with DISTINCT) is identical. Without the DISTINCT keyword, the SQL output lists duplicate rows.

When an SQL SELECT statement is included in a RASQL query program, the statement can extend across multiple lines. The special keyword ENDSQL (not case-sensitive) must be placed at the end of the statement to designate where the statement terminates.

Agentid	Lastname
FB210	Scully
FB340	Reyes

Figure 3: RASQL Query1 Result Table

4. RA AND SQL TOGETHER

The RASQL software can be used to teach Relational Algebra and SQL concepts together. The advantage of mixing SQL and RA in query programs is that it helps students visualize how RA concepts relate to SQL. Some examples of query concepts that can benefit from this integrated approach are described below.

Select Before Join

Relational algebra is a *procedural* language, in that a sequence of operations must be specified for each query. However, different orderings of RA operations can produce identical solutions. Although the end result may be the same, the code versions can vary greatly in terms of resources and performance.

Consider the following revised RA program for Query1. In this version, the *select* operations are performed before the *join*.

```
-- Query1 revised: Select before Join
T1 = TSelect('AGENT',"Gender='F'")
T2=TSelect('TIMECARD',"TCaseNo=2803")
T3 = TJoin(T1,T2,"AgentID=TAgentID")
T4 = TProject(T3,"AgentID,LastName")
```

In the original Query1 program, the join operation is performed first, so table T1 has as many rows as the TIMECARD table. When the select operations are performed before the join, the join table T3 is much smaller, because it is restricted to female agents from the AGENT table. It is also based only on rows for Case 2803 from the TIMECARD table.

Table T3 is the same in both RA versions. The advantage of performing select operations early and join operations later is obvious as the RA program executes step-by-step in RASQL.

In the SQL SELECT statement for Query1, the intermediate operations are unseen by the user. The DBMS query optimizer makes choices among alternative algorithms. Because the intermediate results of the underlying calculations are not visible, SQL appears to be more *non-procedural*.

Product vs. Join

Joining two tables is equivalent to performing a *product* operation followed by a *select* operation. For example, the first line of the Query1 RA program:

```
T1 = TJoin('AGENT','TIMECARD',
     "AgentID=TAgentID")
```

could be split into two operations as follows:

```
T0 = TProduct('AGENT','TIMECARD')
T1 = TSelect(T0,"AgentID=TAgentID")
```

Table T1 and the final result for Query1 (shown in Figure 3) would be unchanged, but the rewritten version requires an extra (relatively large) temporary table T0.

ISSN: 2167-1435
v29 n1906

Union and Union-Compatible

The *union* of tables A and B consists of the combined set of rows of A and B. Because the union is a set, listing duplicate rows is redundant. The union operation in RA follows this convention and eliminates duplicate rows automatically. SQL provides the option to show (UNION ALL) or not show (UNION) duplicate rows. To illustrate the union operation in SQL and RA, consider Query2 stated below.

Query2: List the agent ID and last name of all agents that are male *or* have worked on Case 2801.

An SQL statement for this query, including the ENDSQL keyword required to run in RASQL, is shown below.

```
SELECT AgentID, LastName
FROM AGENT
WHERE Gender = 'M'
UNION
SELECT AgentID, LastName
FROM AGENT, TIMECARD
WHERE AgentID = TAgentID
  AND TCaseNo = 2801
ENDSQL
```

Because this statement contains the UNION keyword (instead of UNION ALL), DISTINCT is not needed, and duplicate rows from the two SELECT sections will not be displayed.

The Query2 result table for this SQL statement, using the XFILES database, is displayed in Figure 4.

Agentid	Lastname
FB160	Skinner
FB180	Mulder
FB210	Scully
FB270	Doggett

Figure 4: RASQL Query2 Result Table

The row "FB270 Doggett", representing the only male agent to work on Case 2801, appears just once.

A comparable RA program for Query2 is listed below. With this query program, duplicate rows in tables T2 and T5 are not repeated in table T6. The result table T6 is identical to the one shown in Figure 4.

```
-- Query2: Union Operation
T1 = TSelect('AGENT',"Gender='M'")
T2 = TProject(T1,"AgentID,LastName")
T3 = TJoin('AGENT','TIMECARD',
     "AgentID=TAgentID")
T4 = TSelect(T3,"TCaseNo=2801")
T5 = TProject(T4,"AgentID,LastName")
T6 = TUnion(T2,T5)
```

The union operation requires the two input tables to be *union-compatible*. That is, they must have the same number of columns, with matching domains for the columns. In this Query2 example, both tables (each SELECT part in the SQL statement, or tables T2 and T5 in the RA program) have the same columns--AgentID and LastName. When two tables are not union-compatible, the union operation is undefined.

Intersection and Difference

Relational algebra includes two additional set operations. The *intersection* of tables A and B is the set of rows that are simultaneously in A and in B. The *difference* A - B includes all rows of A that are not in B. As with union, the intersection and difference operations expect the input tables to be union-compatible. The SQL keyword for the intersection operation is INTERSECT. The difference operation is called MINUS in Oracle and EXCEPT in the SQL standard. In our RA library, the function names are TIntersect and TMinus.

The following Query3 and its SQL and RA programs demonstrate the difference operation.

Query3: List the agent ID and specialty of all agents that have *not* worked on Case 2804.

An SQL statement for this query, including the ENDSQL keyword (RASQL format), is listed next. This SQL statement first collects all rows from AGENT and then removes those for agents who have worked on Case 2804.

```
SELECT AgentID, Specialty
FROM AGENT
```

ISSN: 2167-1435
v29 n1906

```
MINUS
SELECT AgentID, Specialty
FROM AGENT, TIMECARD
WHERE AgentID = TAgentID
  AND TCaseNo = 2804
ENDSQL
```

A corresponding RA program for Query3 is listed below. In this RA program, table T1 includes all agents, and T4 lists those who worked on Case 2804. The difference table T5 consists of all agents who have not worked on that case.

```
-- Query3: Difference Operation
T1 = TProject('AGENT',
     "AgentID,Specialty")
T2 = TJoin('AGENT','TIMECARD',
     "AgentID=TAgentID")
T3 = TSelect(T2,"TCaseNo=2804")
T4 = TProject(T3,"AgentID,Specialty")
T5 = TMinus(T1,T4)
```

Sample output for the Query3 SQL statement and RA program, using the XFILES database, is presented in Figure 5.

Agentid	Specialty
FB160	Management
FB180	Parapsychology
FB210	Medicine
FB480	Computers

Figure 5: RASQL Query3 Result Table

The rows that do not appear in Figure 5 represent the agents in the intersection of the two tables (T1 and T4 in the RA program). These rows are:

FB270	Military Tactics
FB340	Folklore and Mythology

We note that in SQL it is common to use *subqueries* (with IN or NOT IN) to implement queries involving intersections and differences. One reason for this is that some database systems do not support the INTERSECT and MINUS/EXCEPT keywords.

5. RASQL LIMITATIONS

The RASQL software was designed to provide a convenient academic environment for teaching Relational Algebra and SQL together through programming. Several limitations and constraints for using the software are described below.

1. RASQL provides modest error checking. Error messages show the offending line of code but not the reason for the error.

2. Duplicate field names should be avoided in databases. If necessary, use the TRename function in RA programs. This is a constraint inherent in Relational Algebra (Date, 2004). SQL can handle duplicate names using aliases.

3. Nesting of RA function calls within a single statement is permitted but not recommended. Nested function calls defeat the opportunity to see intermediate RA tables. SQL statements hide all but the final query result.

4. Date fields can be included in queries. To specify date constants in row conditions, use the *toDate* function (similar to Oracle's *to_date* function), which returns a date datatype. The format for our toDate function is:

    toDate(year, month, day).

5. The RASQL software has been tested in Windows XP, Windows Vista, and Windows 7. Administrative privileges may be required for Vista or Windows 7.

6. CONCLUSIONS

In this paper, we present reasons for integrating Relational Algebra and SQL in database courses. We suggest that, in teaching RA to database students, a programming approach is preferable to a mathematical approach. Our chosen syntax is to write RA query programs as a sequence of function calls, where each function performs one RA operation. Applying this format, students can gain experience using a procedural query language. Query programs can also be written with non-procedural SQL SELECT statements, or as a combination of RA and SQL instructions.

Learning is enhanced when students can both write and execute query programs. According to Knuth (1974), "... a person does not really understand something until after teaching it to a computer." To ensure that the computer has interpreted our intent correctly, we must be able to run our programs and receive feedback. This is a common learning style in most computing courses.

There are readily available database systems for performing SQL queries, but very few for RA. Because of this, we developed a custom Relational Algebra and SQL (RASQL) software environment in which both RA and SQL programs can run.

The RASQL software allows students to see intermediate results during a sequence of RA and SQL instructions. With this capability, students can visualize how RA operations behave, and relate these operations to SQL statements. Using several example queries, we demonstrated how RA and SQL are interrelated conceptually and functionally. This experience can improve students' understanding of database query languages and the relational model.

We have been using various versions of our Relational Algebra software in database courses for several years. Most of our evidence regarding the utility of the software in teaching RA and SQL has been favorable, but non-empirical. Students are able to effectively write and run Relational Algebra queries. However, no formal assessment of the benefits of this approach in teaching SQL has been established.

In future research, we hope to confirm the symbiotic relationship between learning Relational Algebra and learning SQL. We intend to measure how well a student's understanding of RA improves his/her ability to write SQL query statements, and vice versa. This followup research has been delayed until the authors are again scheduled to teach database courses during the same semester.

Note: An executable version of the RASQL program, runtime files, and the XFILES database and sample RA and SQL programs described in this paper, can be obtained from the lead author.

7. REFERENCES

Caldeira, C. (2008). Teaching SQL: A Case Study. *ITiCSE '08*: Proceedings of the 13th annual conference on Innovation and Technology in Computer Science Education, June 30 - July 2, Madrid, Spain: 340.

Chamberlin, D., and Boyce, R. (1974). SEQUEL: A structured English query language. Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control, pp 249-264.

Chamberlin, D., et. al. (1981). A History and Evaluation of System R. Communications of the ACM, Vol 24, No. 10, pp 632-646.

Codd, E.F. (1970). A relational model of data for large shared data banks. *Comm. ACM* IS, 6.

Codd, E. F. (1972). Relational Completeness of Data Base Sublanguages. In Rustin, Randall (ed.), *Data Base Systems*, Courant Computer Science Series 6. Prentice Hall.

Connolly, T., & Begg, C. (2010). *Database Systems: A Practical Approach to Design, Implementation, and Management* (5th ed). Addison-Wesley.

Date, C. J. (2004). *An Introduction to Database Systems* (8th ed). Addison-Wesley.

Elmasri, R., & Navathe, S. (2010). *Fundamentals of Database Systems* (6th ed). Addison-Wesley.

Knuth, D. (1974). Computer Science and Its Relation to Mathematics. In *The American Mathematical Monthly*, 81, pp 323-343.

Leyton, R. (2010). LEAP RDBMS: An Educational Relational Database Management System. Retrieved from http://leap.sourceforge.net.

Notley, M. (1972). The Peterlee IS/1 System. Rep. UKSC-18, IBM UK Scientific Centre, Peterlee, England.

Pereira, A., Raoufi, M., and Frost, J. (2012). Using MySQL and JDBC in New Teaching Methods for Undergraduate Database Systems Courses. In Lecture Notes in Computer Science, 2012, Volume 6411/2012, pp 245-248.

Ramakrishnan, R., & Gehrke, J. (2002). *Database Management Systems* (3rd ed). McGraw Hill

ISSN: 2167-1435
v29 n1906

Robbert, M., & Ricardo, C. (2003). Trends in the Evolution of the Database Curriculum. *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*. Greece.

Silberschatz, A., Korth, H., & Sudarshan, S. (2010). *Database System Concepts* (6th ed). McGraw Hill.

Todd, S. (1976). The Peterlee Relational Test Vehicle - A System Overview. *IBM Systems Journal*, 15(4), pp 285-308.

Ullman, J., & Widom, J. (2008). *A First Course in Database Systems* (3rd ed). Prentice Hall.

Voorhis, D. (2010). An Implementation of Date and Darwen's Tutorial D Database Language. dappbuilder.sourceforge.net/Rel.php.

ISSN: 2167-1435
v29 n1906

8. APPENDIX: XFILES DATABASE

AGENT table

AgentID	LastName	FirstName	Gender	Position	Status	Specialty	BillRate
FB160	Skinner	Walter	M	Deputy Director	Active	Management	120
FB180	Mulder	Fox	M	Special Agent	Abducted	Parapsychology	90
FB210	Scully	Dana	F	Special Agent	Active	Medicine	96
FB270	Doggett	John	M	Special Agent	Active	Military Tactics	72
FB340	Reyes	Monica	F	Special Agent	Active	Folklore and Mythology	72
FB480	Wildcat	Wendy	F	Student Intern	Temporary	Computers	30

TIMECARD table

TAgentID	TCaseNo	WorkDate	Hours
FB210	2801	10/18/2012	4
FB210	2802	10/18/2012	4
FB270	2801	10/18/2012	4
FB270	2804	10/18/2012	4
FB210	2803	10/19/2012	8
FB270	2803	10/19/2012	8
FB210	2803	10/22/2012	8
FB270	2803	10/22/2012	8
FB340	2802	10/22/2012	4
FB340	2803	10/22/2012	4
FB210	2801	10/23/2012	4
FB210	2802	10/23/2012	4
FB270	2802	10/23/2012	4
FB270	2804	10/23/2012	4
FB340	2802	10/23/2012	4
FB340	2805	10/23/2012	4
FB210	2801	10/24/2012	8
FB270	2805	10/24/2012	8
FB340	2805	10/24/2012	8
FB210	2803	10/25/2012	8
FB270	2801	10/25/2012	2
FB270	2805	10/25/2012	6
FB340	2805	10/25/2012	8
FB210	2801	10/26/2012	2
FB210	2802	10/26/2012	6
FB270	2802	10/26/2012	6
FB270	2804	10/26/2012	2
FB340	2804	10/26/2012	4
FB340	2805	10/26/2012	4

CASES table

CaseNo	Title	Budget
2801	Bermuda Triangle	75000
2802	Dark Matter	40000
2803	Swamp Monster	35000
2804	Alien Cockroaches	25000
2805	Flame-Throwing Aliens	50000
2806	Fat-Sucking Vampire	25000