

1-02-326: Database Systems Engineering

Project 3: Hash Joins

Instructor: Michael J. May

Due 25 May 2011

In this assignment you will develop a tool to execute the hash join algorithm as described in class. The purpose of this exercise is two-fold: to gain a better understanding of the hash join algorithm (one that is very amenable to parallel execution) and gain experience using some of the .NET classes which support SQL queries and data presentation.

1 Overview

We studied the hash join algorithm in class in the context of our discussion of parallel query execution. The hash join algorithm takes a standard “divide and conquer” approach commonly found among efficient sorting and searching algorithms. The essential observation of the algorithm is that most of the cost of joins comes during the search part of the job. Dividing the input tables into buckets can help reduce the amount of time spent searching, thereby making the algorithm more efficient than the naive index nested loop join algorithm.

The pseudocode for the hash join algorithm is shown in Figure 1. The two main stages of the algorithm are *partition* where the two tables are divided into buckets based on a hash function (h1) and *probe* where the tuples in each bucket are divided into sub-buckets by a second hash function (h2) and the members of each sub-bucket are compared to find the matches. Figures showing the important steps of the two stages can be found in the text and in the lecture notes for the course.

2 What to do

Your task is to create a tool which will carry out the steps of the hash join algorithm on real tables which are found in a MS SQL Server database. The tool will show the intermediate steps in the algorithm - the buckets from the first hash function h1 and the second hash function h2 - as well as the final join result. **You are to write all of the searching and matching code yourself. Except where noted in the assignment, you may not use SQL, LINQ, or any other built in join or query mechanisms included in .NET or any other prepared packages.**

3 Logging into the Database Server

Your tool should enable the user to login to the MS SQL Server database using the server name, database name, user name, and password as shown in Figure 2. Note that the connection string for the database login varies based on whether you are logging in using SQL Server authentication (as in the Kinneret computer laboratories) or using Windows Authentication (as you will probably be doing when logging into an SQL Server Express instance on your personal computer).

The connection string for SQL Server authentication has the form:

```
Server=ServerName;database=DatabaseName;user=UserName;password=Password
```

```

// Partition R into k partitions
foreach tuple  $r_1$  in R do
    read  $r_1$  and add it to buffer page  $h(r_1)$ ; // flushed to disk as page fills

// Partition S into k partitions
foreach tuple  $s_1$  in S do
    read  $s_1$  and add it to buffer page  $h(s_1)$ ; // flushed to disk as page fills

// Probing phase
for  $l = 1, \dots, k$  do
{
    // Build in-memory hash table for  $R_l$ , using  $h_2$ 
    foreach tuple  $r_1$  in partition  $R_l$  do
        read  $r_1$  and insert into hash table using  $h_2(r_1)$ ;

    // Scan  $S_l$  and probe for matching  $R_l$  tuples
    foreach tuple  $s_1$  in partition  $S_l$  do
    {
        read  $s_1$  and probe table using  $h_2(s_1)$ ;
        for each matching  $R$  tuple  $r_2$ , output  $\langle r_1, s_1 \rangle$ ;

    clear hash table to prepare for next partition;
}
}

```

Figure 1: Pseudocode for Hash Join Algorithm

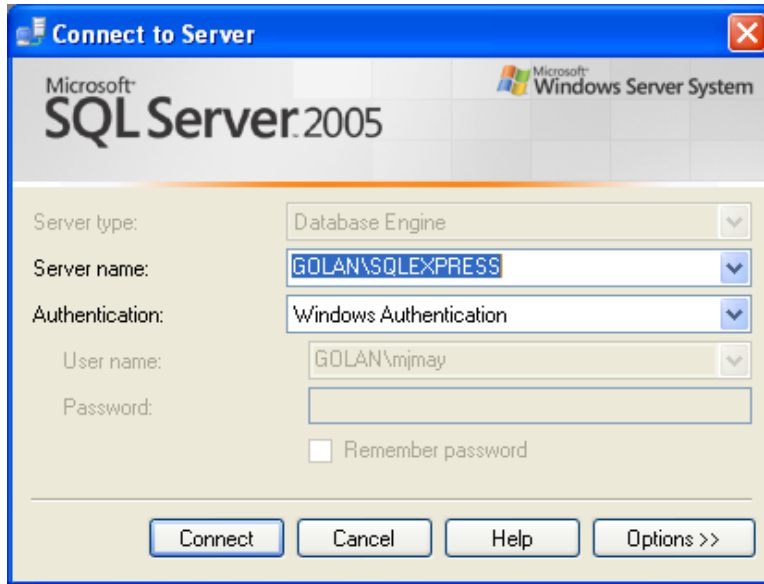
For example, logging into the server “Dugit” and database “targil” with user name “Haim70” and password “12345” you would enter

```
Server=Dugit;database=targil;user=Haim70;password=12345
```

The connection string for Windows authentication is a bit different since you don’t need to supply a password:

```
Server=ComputerName\ServerName;database=DatabaseName;user=UserName;Integrated Security=SSPI
```

If you are using your computer at home with an SQL Server Express installation, your server instance will probably be called “SQLEXPRESS”. You will also need to know the name of your computer (it must have one). To easily find out the name of your computer and its server instance, just open up MS SQL Server Management Studio Express (if you have it installed). It should show you that information in the start up “Connect to Server” window:



Here the name of the computer is “GOLAN”, the name of the server instance is “SQLEXPRESS”, and the user name is “mjmay”. Therefore, to connect to database “targil”, the connection string would be

Server=GOLAN\SQLEXPRESS;database=targil;user=mjmay;Integrated Security=SSPI

Your login window should enable both kinds of logins (so I can test your work at home or at Kinneret). A sample screen shot of the login window is shown in Figure 2.

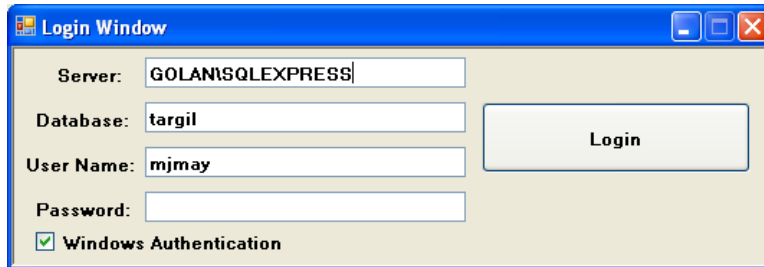


Figure 2: Login Screen

If login fails, show an error window (it should say “Login Failed”) and allow the user to try again.

4 Configuring the Join

The main screen of the application is shown in Figure 3.

We will go over the different parts of the main screen as a guide for how the tool can configure and execute a hash join.

4.1 Choosing Tables

The two ComboBoxes labeled “Table Name:” enable the user to select tables from the list of user tables in the database. Each ComboBox has the full listing of the tables as shown in Figure 4.

Once a table has been selected, it is shown in the DataGridView below, its column names are shown in the ListBox below, and the number of rows in the table is shown in the StatusStrip on the bottom. Figure 5 shows what is displayed after the Sailors and Reserves relations have been selected.

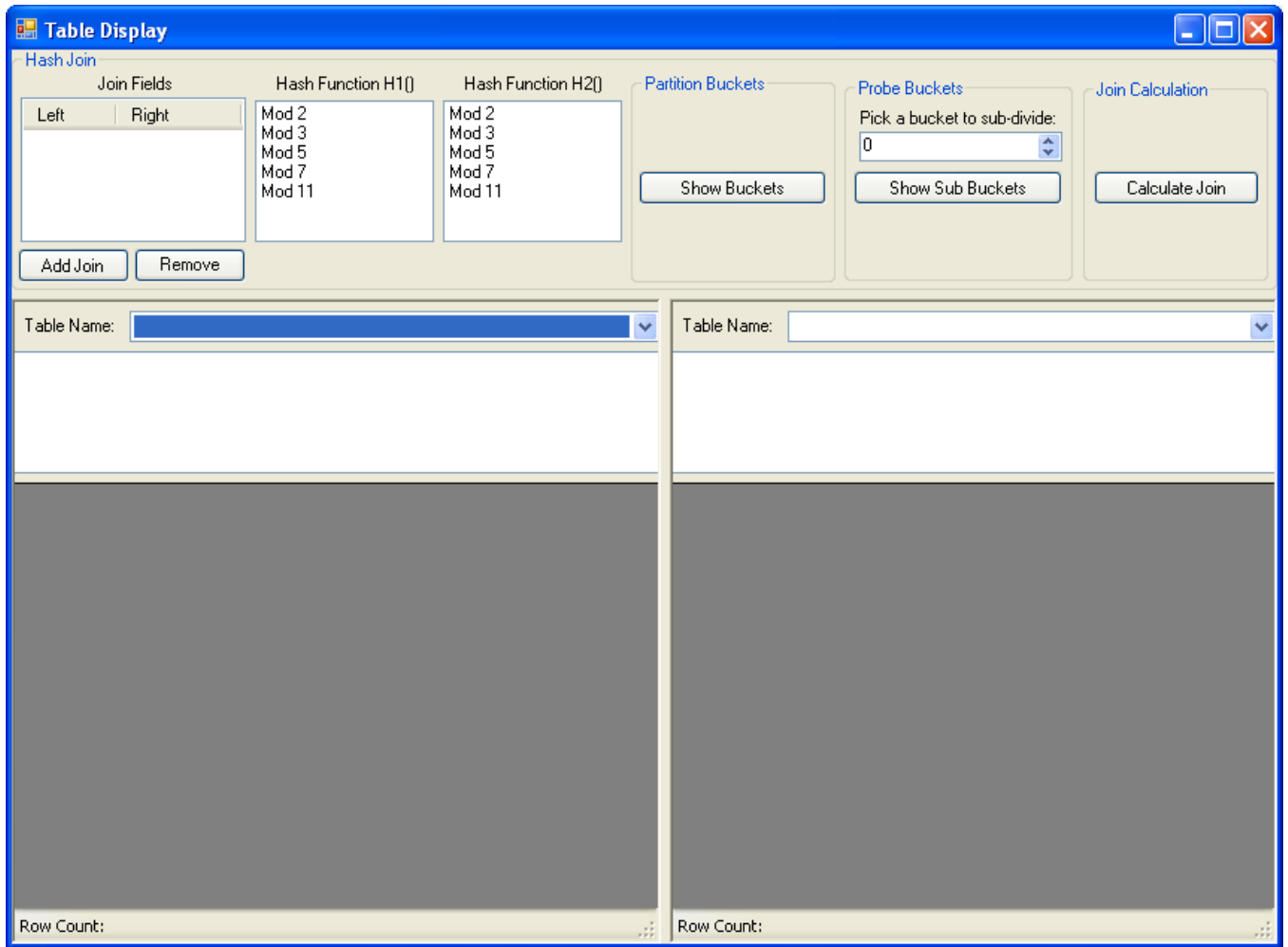


Figure 3: Main Application Screen

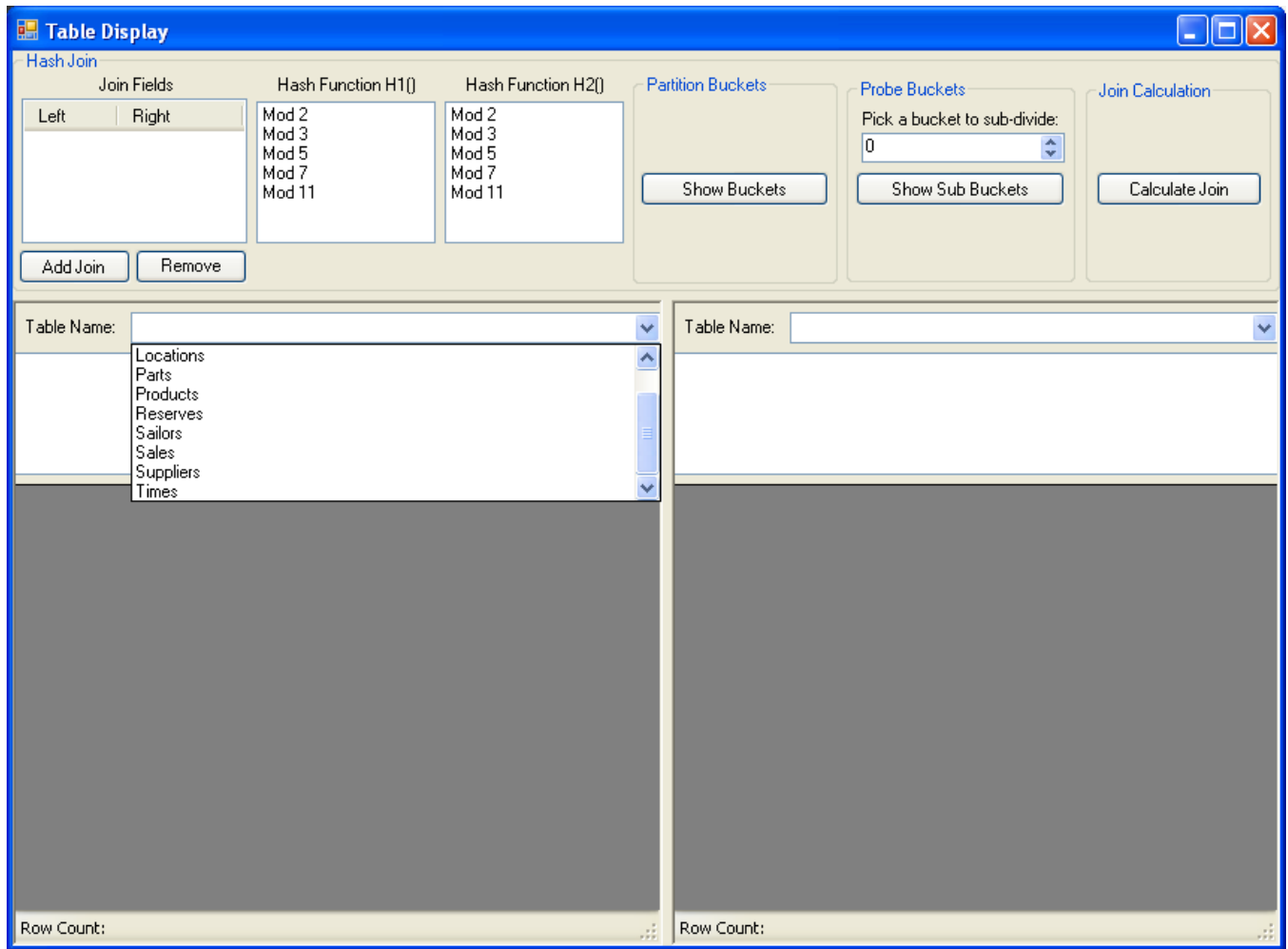


Figure 4: Listing of Database Tables

Hint: You can find the table and column information in the system catalog for your database. Examine the system views called `INFORMATION_SCHEMA.TABLES` and `INFORMATION_SCHEMA.COLUMNS`. **You may use SQL SELECT, WHERE, and joins to retrieve this catalog information from the database.** You may use simple SQL queries (no WHERE) to retrieve the requested table for display in the DataGridView (e.g. `SELECT * FROM Sailors`).

Note If the user subsequently changes the Table Name, the DataGridView, ListBox, and StatusStrip must be updated accordingly.

4.2 Selecting Join Fields

Once the tables have been displayed, we may select one or more *join predicates*. Since we are working with hash join, we will only be considering equijoins (where all fields are to be compared for equality alone), but we do not restrict ourselves to natural joins (i.e. we could join `Reserves.bid = Sailors.age`).

The user may select column names from the ListBoxes on the left and right and click on the “Add Join” button on the top left hand corner to add a new line to “Join Fields”. Note that the user could compare

the same field twice from a table - for example checking `Sailors.age = Reserves.sid AND Sailors.sid = Reserves.sid`.

After adding the join condition `Sailors.sid = Reserves.sid`, the screen appears as in Figure 6.

4.3 Selecting Hash Functions

The user can select which hash functions to use for the hash join using the “Hash Function H1()” and “Hash Function H2()” ListBoxes. For simplicity this tool uses just simple modulus functions (modulo of the primes from 2 – 11).

Modulus is calculated in the usual way on numeric fields. For strings, the modulus is a bit more complex. I used the following algorithm:

- Get the byte array encoding for the string as a Unicode string.
- Sum up the bytes in the byte array
- Perform the modulus on the sum of the bytes

You may choose another encoding scheme you wish so long as it correctly classifies strings and doesn't lump all strings together into the same bucket.

5 Partition Phase

Once the user has selected the tables, join fields, and hash functions, the tool has enough information to calculate the hash join. The tool has the added advantage of enabling you to see the stages of the hash join as it is executed.

During the Partition phase, the tables are divided into buckets based on the function H1. By clicking on the “Show Buckets” button in the “Partition Buckets” GroupBox we can see the buckets produced based on H1().

Figure 7 shows a sample set of buckets for Sailors and Reserves using the hash function Mod 5 for the hash function H1. Note that each bucket is shown in a different numbered tab with its own DataGridView and a StatusStrip showing then number of rows in the bucket currently displayed. Different hash functions will have different numbers of buckets, so you must adjust the number of tabs shown accordingly.

6 Probe Phase

In the Probe phase the buckets prepared in the Partition phase are further divided into sub-buckets using the second hash function. By clicking on the “Show Sub Buckets” button the tool will display the sub-buckets for a given Partition bucket (chosen using the NumericUpDown above it). Note that the hash function H2 is performed on each bucket produced by H1 individually, not on the entire original relation.

Figure 8 shows a sample set of sub-buckets for Sailors and Reserves using the hash function Mod 5 for H1 and the hash function Mod 11 for H2. As in the Partition phase, each sub-bucket is shown on a separate tab with a DataGridView showing the contents of the sub-bucket and a StatusStrip showing the number of rows in the sub-bucket currently displayed.

7 Performing the Hash Join

The button labeled “Calculate Join” in the “Join Calculation” GroupBox calculates the results of the hash join. It does so by invoking a function with the signature

```
public static DataTable CalculateJoin(DataTable[] leftBuckets, string[] leftColumns,
    DataTable[] rightBuckets, string[] rightColumns, HashFunction hf)
```

which is found in a helper class. HashFunction is a custom interface I designed to enable the abstraction of a hash function which the user may select. Its principle functions calculate a numeric hash and return the number of buckets that it produces. The string arrays contain the names of the columns to compare. The two arrays must be of the same length and it is assumed that for each entry i , `leftColumns[i] = rightColumns[i]` will be checked on the provided DataTables.

The hash join is performed by searching for matches within each corresponding sub-bucket in the two tables. This is not precisely the algorithm as shown in the pseudocode in Figure 1 since my code prepares both sub-buckets in memory before performing the matches. You may implement your application to precisely match the pseudocode if you wish.

The results of performing the hash join are shown in Figure 9. Note that the results are shown in a single DataGridView with a StatusStrip showing the total number of rows displayed.

Note that the fields are suffixed with the letters “l” and “r” to indicate their source table. This is an artifact of my using the DataTable class which does not let you have two columns with the same name. Your tool should make it clear which column came from which table in some manner.

8 Experiments to Perform

Using your tool, perform the following experiments:

- Perform a simple hash join on Sailors.sid and Reserves.sid using Mod 2 for H1 and Mod 3 for H2. Note how long it takes for the calculation. Perform it again using Mod 7 for H1 and Mod 11 for H2. Do you notice any difference in the run time? Try using extremely large tables (10,000 rows) to better compare.
- Perform a hash join on the fields Sailors.sname and Boats.bname to see that your tool correctly performs string hash joins. How would you modify your tool to enable LIKE on strings?

Include the answers to the above questions in your assignment report.

9 Table Generation

To assist you in generating large database tables I have prepared a tool called “SailorGenerator” to randomly produce large numbers of rows for Sailors, Reserves, and Boats. The tool is shown in Figure 10. You must provide the tool with a list of Sailor names, Boat names, and Colors. The output will be a series of table inserts for:

```
Sailors(sid int, sname varchar(20), rating int, age int)
Boats (bid int, bname varchar(20), color varchar(10))
Reserves (bid int, sid int, day datetime)
```

The tool is not optimized for speed, so it will take a few seconds to complete the results for the input shown in Figure 10.

I have put a copy of the tool and three sample input files (Sailor names, Boat names, and Colors) on the course web page.

Hint: To perform the second experiment, you can include some of the names in Sailor name file in the Boats name file. This will ensure at least a partial overlap in the names. The sample files given on the web page do not have any overlap between the names, so you will need to modify them.

10 Grading and Submission

The assignment is worth 100 points total. Points will be assigned as follows:

1. Correct login and connection strings. 5 points
2. Correct display of tables and their fields on the main screen. 10 points.
3. Correct calculation and display of the Partition buckets based on H1. 20 points.
4. Correct calculation and display of the Probe buckets based on H1 and H2. 25 points.
5. Correct calculation and display of the join output. 25 points.
6. Correct operation on strings and response to experiments. 10 points.
7. General robustness. 5 points.

You may work in groups of **2 students**. If there are an odd number of students, I will permit one group of three.

You must submit your work via email (to [ise326@gmail](mailto:ise326@gmail.com)) or Telem by 25 May 2011. Keep in mind the submission rules and late penalties listed in the syllabus. Submit your work as a single Zip or RAR file with the following contents:

- The Visual Studio C# project file and all source code.
- A compiled executable file
- A README file which contains the names of the students who worked on the project and their individual contributions, the number of hours spent working on it, and any notes or documentation necessary to understand and use the application.
 - If this information is missing, I will deduct 3 points.

Table Display

Hash Join

Join Fields: Left | Right

Hash Function H1(): Mod 2, Mod 3, Mod 5, Mod 7, Mod 11

Hash Function H2(): Mod 2, Mod 3, Mod 5, Mod 7, Mod 11

Partition Buckets: Show Buckets

Probe Buckets: Pick a bucket to sub-divide: 0, Show Sub Buckets

Join Calculation: Calculate Join

Add Join | Remove

Table Name: Sailors

sid
sname
rating
age

sid	sname	rating	age
0	Zany	4	50
1	Mike	7	89
2	Ely	5	68
3	Zany	2	28
4	Mindy	5	45
5	Horace	4	46
6	Frank	9	57
7	Ilk	5	62
8	Pookie	6	85
9	Valiant	3	89
10	Nancy	4	90
11	Louise	2	11
12	Dima	8	40
13	Beatrice	3	35
14	Zephyer	4	83
15	Randy	7	88
16	Asaf	8	27
17	Windy	8	37
18	Windy	8	44
19	Art	0	59
20	Effie	7	88

Row Count: 200

Table Name: Reserves

bid
sid
day

bid	sid	day
0	12	7/27/2010 12:00...
0	78	10/24/2010 12:00...
0	82	9/8/2010 12:00...
0	170	9/22/2011 12:00...
0	175	6/21/2010 12:00...
1	49	11/4/2010 12:00...
1	72	5/22/2010 12:00...
1	138	1/7/2011 12:00...
1	157	7/20/2010 12:00...
1	191	9/21/2010 12:00...
2	63	9/8/2011 12:00...
3	61	6/24/2010 12:00...
3	86	9/15/2010 12:00...
3	97	6/7/2011 12:00...
3	161	7/10/2011 12:00...
4	68	8/21/2011 12:00...
4	122	6/10/2011 12:00...
4	128	10/12/2010 12:00...
4	143	9/24/2010 12:00...
4	182	8/2/2011 12:00...
5	41	10/5/2010 12:00...

Row Count: 1000

Figure 5: After Selecting Sailors and Reserves

Table Display

Hash Join

Left	Right	Hash Function H1()	Hash Function H2()
sid	sid	Mod 2 Mod 3 Mod 5 Mod 7 Mod 11	Mod 2 Mod 3 Mod 5 Mod 7 Mod 11

Partition Buckets: Show Buckets

Probe Buckets: Pick a bucket to sub-divide: 0 Show Sub Buckets

Join Calculation: Calculate Join

Add Join Remove

Table Name: Sailors

sid	sname	rating	age
0	Zary	4	50
1	Mike	7	89
2	Ely	5	68
3	Zary	2	28
4	Mindy	5	45
5	Horace	4	46
6	Frank	9	57
7	Ilk	5	62
8	Pookie	6	85

Row Count: 200

Table Name: Reserves

bid	sid	day
0	12	7/27/2010 12:00...
0	78	10/24/2010 12:00...
0	82	9/8/2010 12:00:...
0	170	9/22/2011 12:00...
0	175	6/21/2010 12:00...
1	49	11/4/2010 12:00...
1	72	5/22/2010 12:00...
1	138	1/7/2011 12:00:...
1	157	7/20/2010 12:00...

Row Count: 1000

Figure 6: After Selecting Join Fields “sid” and “sid”

Table Display

Hash Join

Left	Right	Hash Function H1()	Hash Function H2()
sid	sid	Mod 2 Mod 3 Mod 5 Mod 7 Mod 11	Mod 2 Mod 3 Mod 5 Mod 7 Mod 11

Partition Buckets:

Probe Buckets: Pick a bucket to sub-divide:

Join Calculation:

Add Join Remove

Table Name: Sailors

First Level Buckets for Tables Sailors and Reserves Using Hash Function Mod 5

Sailors				Reserves					
Bucket 0	Bucket 1	Bucket 2	Bucket 3	Bucket 4	Bucket 0	Bucket 1	Bucket 2	Bucket 3	Bucket 4
sid	sid	sname	rating	bid	sid	day			
0	1	2	3	4	0	170	9/22/2011 12:00...		
5	10	15	20	25	0	175	6/21/2010 12:00...		
30	35	40	45	50	5	95	5/29/2011 12:00...		
55	60				5	125	5/7/2011 12:00...		
					7	155	7/28/2010 12:00...		
					8	155	4/18/2011 12:00...		
					9	65	7/21/2011 12:00...		
					10	20	12/13/2010 12:0...		
					13	120	3/12/2011 12:00...		
					14	25	8/2/2010 12:00...		
					14	115	6/20/2011 12:00...		
					16	75	6/25/2011 12:00...		
					16	115	5/17/2011 12:00...		

Row Count: 200 Number of Rows Shown on Left Tab: 40 Number of Rows Shown on Right Tab: 182

Figure 7: Partition Buckets for Sailors and Reserves using Mod 5

Table Display

Hash Join

Join Fields		Hash Function H1()	Hash Function H2()
Left	Right	Mod 2	Mod 2
sid	sid	Mod 3	Mod 3
		Mod 5	Mod 5
		Mod 7	Mod 7
		Mod 11	Mod 11

Partition Buckets: [Show Buckets]

Probe Buckets: Pick a bucket to sub-divide: 2 [Show Sub Buckets]

Join Calculation: [Calculate Join]

Table Name: Sailors | Table Name: Reserves

Second Level Buckets for Tables Sailors and Reserves Using Hash Function Mod 11. Breakdown of Bucket 2

Sailors					Reserves									
Bucket 1	Bucket 2	Bucket 3	Bucket 4	Bucket 5	Bucket 3	Bucket 4	Bucket 5	Bucket 6	Bucket 7	Bucket 8	Bucket 9	Bucket 10		
	sid	sname	rating	age		bid	sid	day						
	27	Kandy	6	94		18	87	8/12/2010 12:00:00 AM						
	82	Dot	6	27		25	197	10/19/2010 12:00:00 AM						
	137	Tammy	5	6		28	87	6/18/2010 12:00:00 AM						
	192	Xanadu	6	32		28	87	8/29/2011 12:00:00 AM						
						33	197	9/29/2010 12:00:00 AM						
						49	142	7/14/2011 12:00:00 AM						
						55	32	6/5/2011 12:00:00 AM						
						74	197	2/15/2011 12:00:00 AM						
						77	197	10/19/2010 12:00:00 AM						
						78	32	2/23/2011 12:00:00 AM						
						83	32	1/29/2011 12:00:00 AM						

Number of Rows Shown on Left Tab: 4 | Number of Rows Shown on Right Tab: 16

Figure 8: Probe Sub-Buckets for Sailors and Reserves using Mod 5 for H1 and Mod 11 for H2

Table Display

Hash Join

Join Fields		Hash Function H1()	Hash Function H2()	Partition Buckets	Probe Buckets	Join Calculation
Left	Right	Mod 2 Mod 3 Mod 5 Mod 7 Mod 11	Mod 2 Mod 3 Mod 5 Mod 7 Mod 11	Show Buckets	Pick a bucket to sub-divide: 2 Show Sub Buckets	Calculate Join
sid	sid					

Add Join Remove

Table Name: Sailors Table Name: Reserves

Hash Join Results Display

The final hash join display for the tables Sailors and Reserves

	bid_l	sid_l	day_l	sid_r	sname_r	rating_r	age_r
▶	30	110	11/6/2010 12:00...	110	Arby	6	22
	34	165	9/23/2011 12:00...	165	Manny	3	96
	48	165	1/4/2011 12:00:...	165	Manny	3	96
	59	110	1/15/2011 12:00...	110	Arby	6	22
	87	55	1/3/2011 12:00:...	55	Yolonda	6	76
	89	55	6/29/2011 12:00...	55	Yolonda	6	76
	105	165	11/9/2010 12:00...	165	Manny	3	96
	124	165	7/31/2010 12:00...	165	Manny	3	96
	130	165	7/29/2011 12:00...	165	Manny	3	96
	157	110	2/19/2011 12:00...	110	Arby	6	22
	199	110	11/29/2010 12:0...	110	Arby	6	22
	7	155	7/28/2010 12:00...	155	Mandy	1	49
	8	155	4/18/2011 12:00...	155	Mandy	1	49

Number of Rows Shown: 1000

Figure 9: Join Result

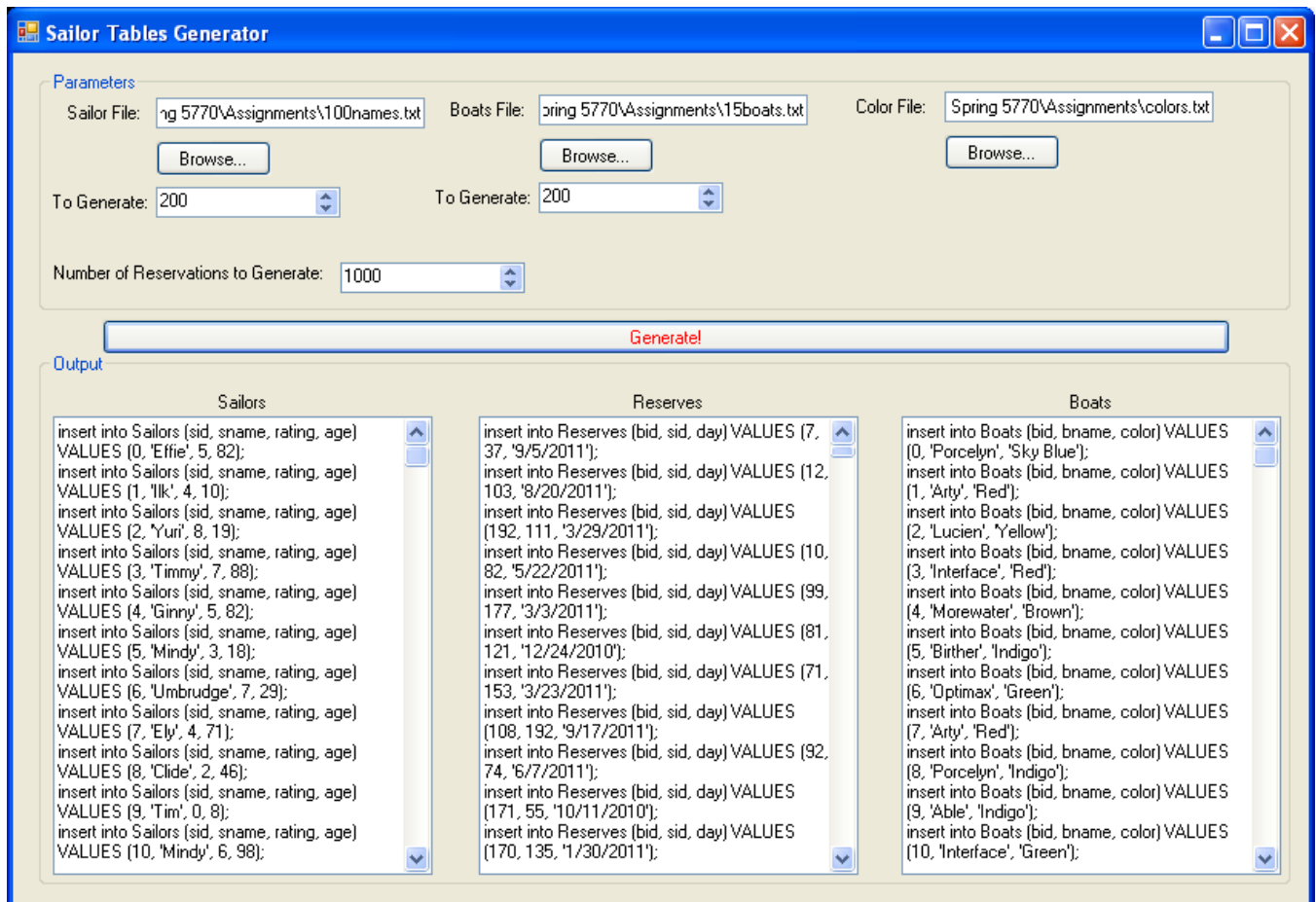


Figure 10: SailorGenerator Tool