# The Match: A Case Study In Algorithm Analysis Of The National Resident Matching Program

Mohammad Dadashzadeh, Ph.D., Oakland University, USA
Sara Dadashzadeh, Oakland University, USA

## ABSTRACT

*There are rare opportunities when solving an easily-understood problem can bring together application of skills taught in diverse courses in a Computer Science (CS) or Management Information Systems (MIS) program. This paper presents such an opportunity in the typical database management systems course taught at the junior or senior level. Specifically, we describe the case study of solving the classical Hospitals/Residents problem in Microsoft Access. The solution, based on classical Gale-Shapely algorithm for the Stable Marriage problem, offers pedagogical opportunities in data modeling, algorithm and data structure considerations for program development, Visual Basic for Applications (VBA) and embedded SQL (Structured Query Language) programming, and empirical analysis of running time complexity of algorithms that work remarkably well in teaching students the value of each tool in the toolset they take away from required courses as a part of their undergraduate education in CS or MIS.*

**Keywords:** Hospitals Residents Problem; National Resident Matching Program (NRMP); Analysis Of Algorithms; Microsoft Access; VBA; SQL

## INTRODUCTION

National Resident Matching Program (NRMP) is a United States centralized clearinghouse for matching graduating medical school students to prospective residency programs. The resident match process, commonly referred to as The Match, was established in 1952 to address the ineffective decision making caused by fierce competition between hospitals for desired interns and amongst medical students for good internships. Roth (2003) describes the important steps leading to The Match starting with the "Cooperative Plan" adopted by the Association of American Medical Colleges to not release appointment offers before an announced date; establishing a clearinghouse to solicit rank order (preference) lists from students and hospitals and using them to produce a match; and the development of "The Boston Pool Algorithm" for producing a stable match (that is, no applicant and hospital who were *not* matched with one another preferred each other over the matches assigned to them).

The first Main Residency Match® was conducted in 1952 when 10,400 internship positions were available for 6,000 U.S. graduating seniors, while the 2014 Match recorded all-time highs of 26,678 first-year post-graduate positions for 40,394 applicants (NRMP, 2014). The 2014 Match included 3,943 residency programs 407 of which were not able to completely fill their available quotas. The 26,678 positions offered by these programs received a total of 348,065 rankings by the applicants with 25,687 positions being matched. The overall position fill rate of 96.0 percent makes the 2014 Match one of the most successful on record (NRMP, 2014).

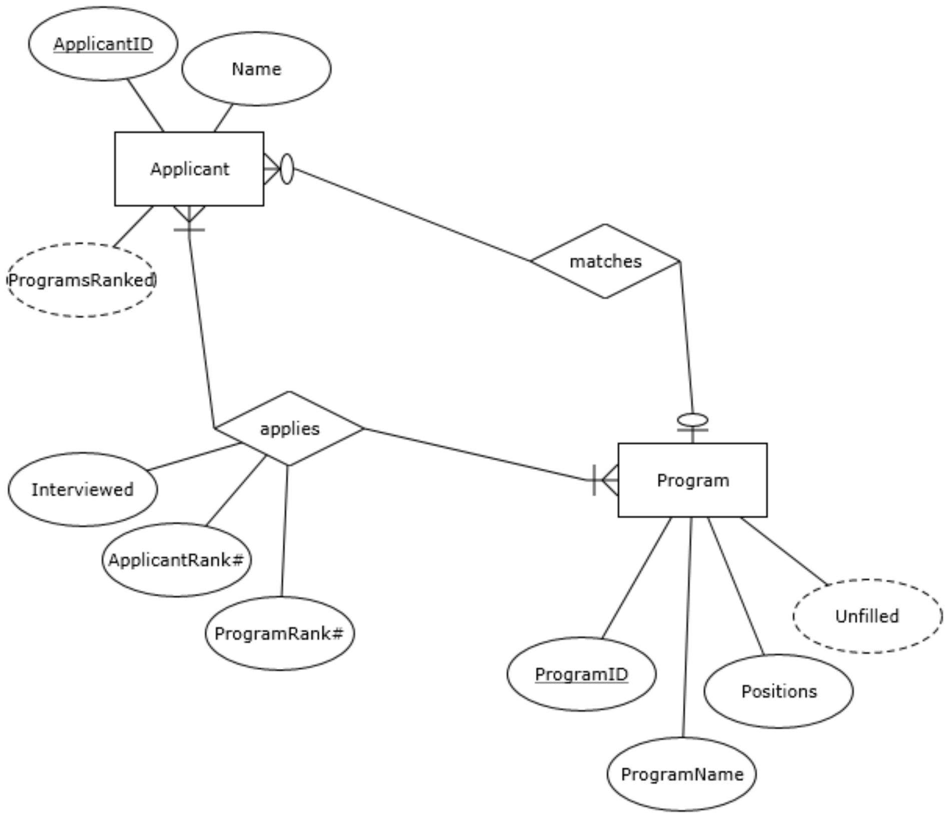
Implementation of The Match represents a real-world problem that can be effectively utilized for problem-based learning in the database course. It engages students' interest and motivates them to research the matching process, the algorithm, and its extensions further. It incorporates the content objectives of the course including data modeling, SQL, DBMS-based application development, while connecting previous knowledge of programming to new concepts such as embedded SQL programming, and connecting new knowledge of empirical analysis of running time complexity of The Match to concepts in other courses such as analysis of algorithms. In the following sections, we formalize the statement of the problem, the implementation of The Match in Microsoft Access, an analysis of the running time complexity of the implementation, and additional teaching experience.

## STATEMENT OF PROBLEM

Consider the problem of designing a database to support The Match by keeping track of residency programs, applicants, and the applicant ranking of each program he/she is invited for an interview, as well as the ranking of each applicant interviewed by each program. Figure 1 provides the conceptual data model for the desired database as an Entity-Relationship Diagram (ERD).

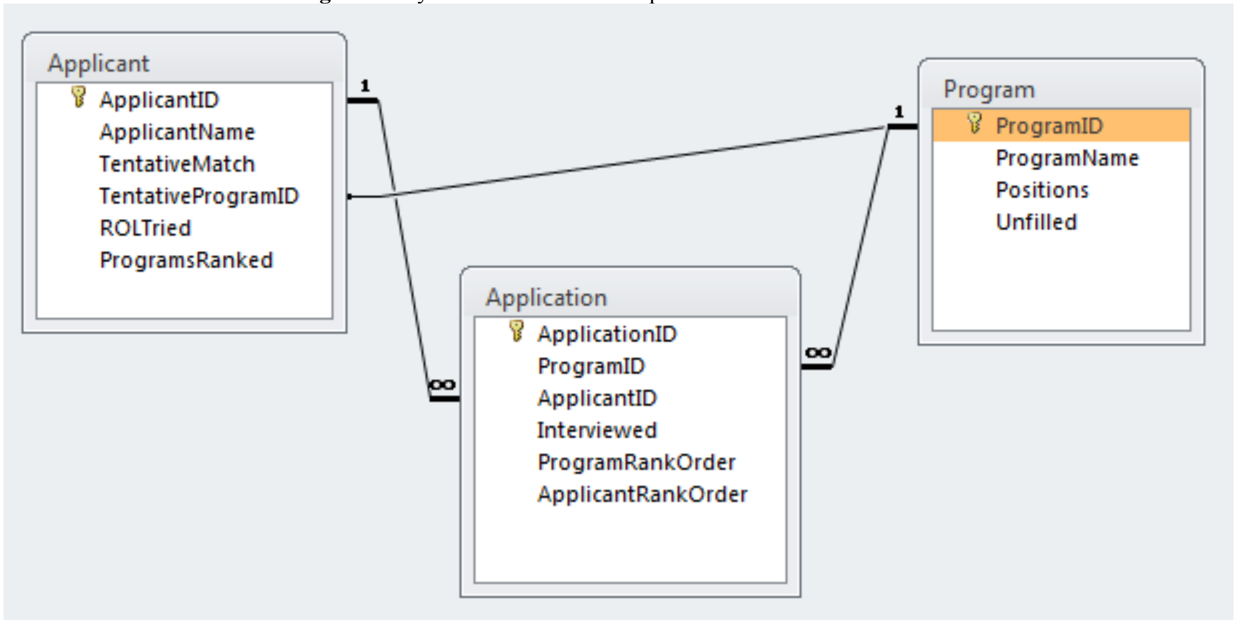
In the ERD, attribute *ApplicantRank#* represent the ranking an applicant gives to a program for matching purposes, while *ProgramRank#* captures the ranking of the applicant by the program. The attribute *ProgramsRanked* depicted in a dashed oval denotes the fact that it is a derivable attribute (since for each applicant *ProgramsRanked* can be calculated from the number of occurrences of *applies* relationship in which that *ApplicantID* appears and the attribute *Interviewed* has a value of true.) Note also that the relationship *matches* is many-to-1 between Applicant and Program entities and optional for both, reflecting the fact that any applicant can match 0 or at most 1 program while a program can match many applicants although it may be possible that it matches none (when no applicant interviewed would rank the program for a desired match.)

**Figure 1.** Conceptual Data Model



The physical database as implemented in Microsoft Access is shown as a relationship screen in Figure 2. The m-to-m relationship *applies* is modeled using the junction table Application with an auto number primary key (named *ApplicationID*) and foreign keys *ApplicantID* and *ProgramID*. The 1-to-m relationships *matches* is captured through the foreign key *TentativeProgramID* in table Applicant. The additional columns *TentativeMatch* and *ROLTried* in Applicant table support the implementation of The Match algorithm to be described in this case study.

**Figure 2.** Physical Data Model As Implemented In Microsoft Access



To fix ideas, let us consider the example of five applicants applying to/ranking three programs each of which has two positions available to match (National Resident Matching Program, 2015). The Program table would appear as shown in Table 1 with the value of the derivable attribute *Unfilled* the same as number of positions available:

**Table 1.** Sample *Program* Table

| Program | | | |
|---|---|---|---|
| **ProgramID** | **ProgramName** | **Positions** | **Unfilled** |
| 1 | City | 2 | 2 |
| 2 | Mercy | 2 | 2 |
| 3 | General | 2 | 2 |

The Applicant table would appear as shown in Table 2 with the value of the derivable attribute *ProgramsRanked* reflecting data from the Application table (see Table 3) on number of programs ranked by each applicant:

**Table 2.** Sample *Applicant* Table

| Applicant | | | | | |
|---|---|---|---|---|---|
| **ApplicantID** | **ApplicantName** | **TentativeMatch** | **TentativeProgramID** | **ROLTried** | **ProgramsRanked** |
| 1 | Anderson | False | | 0 | 1 |
| 2 | Chen | False | | 0 | 2 |
| 3 | Ford | False | | 0 | 3 |
| 4 | Davis | False | | 0 | 3 |
| 5 | Eastman | False | | 0 | 3 |

**Table 3.** Sample *Application* Table

| | | | Application | | |
|---|---|---|---|---|---|
| **ApplicationID** | **ProgramID** | **ApplicantID** | **Interviewed** | **ProgramRankOrder** | **ApplicantRankOrder** |
| 1 | 1 | 1 | True | 2 | 1 |
| 2 | 1 | 2 | True | 3 | 1 |
| 3 | 2 | 2 | True | 1 | 2 |
| 4 | 1 | 3 | True | 5 | 1 |
| 5 | 3 | 3 | True | 3 | 2 |
| 6 | 2 | 3 | True | 2 | 3 |
| 7 | 3 | 4 | True | 4 | 3 |
| 8 | 1 | 4 | True | 4 | 2 |
| 9 | 2 | 4 | True | | 1 |
| 10 | 1 | 5 | True | 1 | 1 |
| 11 | 3 | 5 | True | 1 | 3 |
| 12 | 2 | 5 | True | | 2 |
| 13 | 3 | 1 | True | 2 | |

The above data captures the Rank Order Lists, respectively, by applicants and by programs depicted in Table 4.

**Table 4.** Rank Order Lists Reflected in Sample Tables

| **Anderson** | **Chen** | **Ford** | **Davis** | **Eastman** |
|---|---|---|---|---|
| **1. City** | 1. City | 1. City | 1. Mercy | 1. City |
| | 2. Mercy | 2. General | 2. City | 2. Mercy |
| | | 3. Mercy | 3. General | 3. General |

| **City** | **Mercy** | **General** |
|---|---|---|
| 1. Eastman | 1. Chen | 1. Eastman |
| 2. Anderson | 2. Ford | 2. Anderson |
| 3. Chen | | 3. Ford |
| 4. Davis | | 4. Davis |
| 5. Ford | | |

Given the sample input data as above, The Match algorithm needs to match each applicant to the highest program he/she has ranked provided that:

- the program has an unfilled position
- the programs has also ranked the applicant
- no other applicant ranked higher by the program remains unmatched

For our example case, Table 5 would reflect the expected output of an implementation of The Match algorithm:

**Table 5.** Expected Logical Result of The Match Algorithm for Sample Data

| **Anderson** | **Chen** | **Ford** | **Davis** | **Eastman** |
|---|---|---|---|---|
| 1. City | | | | 1. City |
| | 2. Mercy | 2. General | | |
| | | | 3. General | |

| **City** | **Mercy** | **General** |
|---|---|---|
| 1. Eastman | 1. Chen | |
| 2. Anderson | | |
| | | 3. Ford |
| | | 4. Davis |

This match is reflected in in the database tables as shown in Table 6.

**Table 6.** Expected Output of The Match Algorithm in Sample Database

**Program**

| ProgramID | ProgramName | Positions | Unfilled |
|---|---|---|---|
| 1 | City | 2 | 0 |
| 2 | Mercy | 2 | 1 |
| 3 | General | 2 | 0 |

**Applicant**

| ApplicantID | ApplicantName | TentativeMatch | TentativeProgramID | ROLTried | ProgramsRanked |
|---|---|---|---|---|---|
| 1 | Anderson | True | 1 | 1 | 1 |
| 2 | Chen | True | 2 | 2 | 2 |
| 3 | Ford | True | 3 | 2 | 3 |
| 4 | Davis | True | 3 | 3 | 3 |
| 5 | Eastman | True | 1 | 1 | 3 |

## THE MATCH ALGORITHM IMPLEMENTATION

The algorithm for matching applicants to programs can be described in pseudo code as follows:

```
Do While {an applicant remains unmatched and the applicant's Rank Order List is not exhausted}

    Do

        With the next program on the applicant's Rank Order List
        {
            If {the program has also ranked the applicant) Then

                If (that program has an unfilled position) Then

                    Assign the program as applicant's tentative match

                Else

                    If (another applicant tentatively matched to that program can be unmatched) Then

                        Un-assign the applicant ranked lower by the program

                        Assign the program as applicant's tentative match

                End If

            End If
        }

    Loop Until {the applicant is matched or the applicant's Rank Order List is exhausted}

Loop
```

Appendix A provides the complete implementation of the algorithm in Microsoft Access using Visual Basic for Applications (VBA).

## ALGORITHM ANALYSIS

Analyzing the running time of The Match algorithm as implemented can be approximated as follows. Assuming N applicants and M programs, the outer loop will *at least* iterate once for every applicant, and the inner loop will *at most* iterate M times (reflecting the scenario that each applicant has been interviewed at, and ranked, all programs). Thus, the running time can be approximated as N*M inner loop calculations. Of course, each time that the inner loop voids a tentative match, the outer loop will need to repeat for the candidate whose tentative match was voided. Therefore, if there are *k* such voids, then the overall running time is expected to increase to (N+*k*)*M which reflects a quadratic time performance, O($n^2$).

Table 7 shows the results of running the algorithm for several random data sets.

**Table 7.** Algorithm Analysis Results for Random Data Sets

| Data Set | Outer Loop Iterations | # of Bumps (voiding a tentative match) | Inner Loop Executions |
|---|---|---|---|
| N=5, M=3<br>Range of Programs Ranked: 1, 3<br>Range of Applicants Ranked: 2, 5<br>Range of Positions: 2, 2 | 6 | 1 | 9 |
| **N=100, M=50<br>Range of Programs Ranked: 1, 9<br>Range of Applicants Ranked: 5, 16<br>Range of Positions: 2, 11** | 102 | 3 | 188 |
| N=1,000, M=50<br>Range of Programs Ranked: 1, 9<br>Range of Applicants Ranked: 82, 131<br>Range of Positions: 2, 11 | 1426 | 510 | 4135 |
| **N=10,000, M=100<br>Range of Programs Ranked: 1, 9<br>Range of Applicants Ranked: 447, 552<br>Range of Positions: 2, 11** | 11554 | 1871 | 48345 |

The Match algorithm is based on the seminal work of Gale and Shapely (1962) and the Stable Marriage Problem (SMP) introduced by them. Briefly stated, SMP is to find a matching between men and women considering each person's rank order (preference) list in which the person expresses his/her preference over the members of the opposite gender. The output matching must be stable, which intuitively means that there is no man/woman pair both of whom have incentive to elope (Iwama and Miyazaki, 2008 ). Gale and Shapley (1962) proposed the matching algorithm, which runs in time O($n^2$) and always finds a stable matching. It is important to note that while the pairings found are stable, they are not necessarily optimal from all individual's point of view (Wikipedia, 2015).

A variant of SMP, Stable Marriage Problem with Incomplete preference lists (SMPI), allows each person's preference (rank order) list to be incomplete, i.e., a person can exclude some members whom he/she does not want to be matched with. A slight modification of Gale-Shapely Algorithm can be applied to find a stable matching for SMPI (Iwama and Miyazaki, 2008).

The hospitals/residents (HR) matching problem is a many-to-one extension of SMPI, where we consider men as residents and women as hospitals. Each hospital specifies its quota, i.e., the number of residents it can accept, and rank order (preference) lists are incomplete for both residents and the hospitals (programs). HR is reduced into SMP by replacing each hospital with a quota of q by q copies of it each having the same preference list for residents. It has been shown that most of the results established for SMP hold for HR (Gusfield and Irving, 1989).

## TEACHING EXPERIENCE

By the time students majoring in Computer Science (CS) or Management Information Systems (MIS) reach the database course; they have been exposed to fundamental programming concepts including basic analysis of algorithms.

The limited programming emphasis in the database course, if at all, is typically reserved for event code macros supporting prototyping of graphical user interfaces to back-end databases, for example, in creating forms in Microsoft Access, or in enforcing integrity constraints using triggers and stored procedures. That is, of course, quite appropriate since the database course syllabus is justifiably pre-occupied with data modeling, relational database design, and SQL.

The case study problem presented in this article presents a pedagogical opportunity to allow students apply and extend their programming skills in the database course in solving a non-trivial real-world problem. In a remarkable way, it reinforces the tenet: algorithms + data structures = programs (Wirth, 1978), while highlighting the role that DBMS can play in providing data structure support for program development. Additional teaching opportunities arise in analyzing the running time complexity of the algorithm which, in turn, provides yet additional programming practice for test data generation and empirical algorithm analysis. Extending the solution of the HR problem described in this case study to allow couples to stay together by submitting joint preference lists over pairs of hospitals (McDermid and Manlove, 2010) would be an appropriate advanced assignment in a course on analysis of algorithms. It has been our experience that when an opportunity presents itself to provide a case study problem to use embedded SQL programming when non-procedural SQL alone would not be sufficient, students seem to leave the database course with better problem-solving skills.

## AUTHOR INFORMATION

**Mohammad Dadashzadeh** serves as Professor of MIS at Oakland University. He has authored 4 books and more than 50 articles on information systems and has served as the editor-in-chief of *Journal of Database Management*. (contact author)

**Sara Dadashzadeh** is completing her undergraduate degree in biology at Oakland University. She hopes to participate in The Match in 2020 for residency training.

## REFERENCES

Gale, D., & Shapley, L.S. (1962). College Admissions and the Stability of Marriage. *American Mathematical Monthly*, 69, 9–15.

Gusfield, D., & Irving, R.W. (1989). *The Stable Marriage Problem: Structure and Algorithm.* Boston. MA: MIT Press.

Iwama, K., & Miyazaki, S. (2008). A Survey of the Stable Marriage Problem and Its Variants. In *Proceedings of International Conference on Informatics Education and Research for Knowledge-Circulating Society (ICKS 2008)*, 131-136.

McDermid, E.J., & Manlove, D.F. (2010). Keeping Partners Together: Algorithmic Results for the Hospitals/Residents Problem with Couples. *Journal of Combinatorial Optimization*, 19, 279–303.

National Resident Matching Program. (2014). *Results and Data: 2014 Main Residency Match®*. Washington, DC: National Resident Matching Program.

National Resident Matching Program. (2015). Run A Match. Retrieved from http://www.nrmp.org/wp-content/uploads/2014/05/Run-A-Match.pdf

Roth, A.E. (2003). The Origins, History, and Design of the Resident Match. *Journal of American Medical Association*, 289, 909-912.

Wikipedia. (2015). Stable Marriage Problem. Retrieved from http://en.wikipedia.org/wiki/Stable_marriage_problem

Wirth, N. (1978). *Algorithms + Data Structures = Programs*. Upper Saddle River, NJ: Prentice Hall.

## APPENDIX

This appendix presents the entire Access VBA Module code implementing the solution. A copy of the database and code is available from the authors upon request.

```
Sub DoTheMatch()

Dim strSQL As String
Dim rs As Recordset

Call Initialize

strSQL = "Select * "
strSQL = strSQL & "From Applicant "
strSQL = strSQL & "Where TentativeMatch = False And ROLTried < ProgramsRanked "
strSQL = strSQL & "Order By ApplicantID"

Do While Not Finished

Set rs = CurrentDb.OpenRecordset(strSQL)

ApplicantID = rs("ApplicantID")
'Get the highest Rank Order List # having been considered for the applicant

ROLTried = rs("ROLTried")
'Get the # of programs ranked by the applicant ...

ProgramsRanked = rs("ProgramsRanked"
TentativelyMatched = False

Do While Not TentativelyMatched And ROLTried < ProgramsRanked

ROLTried = ROLTried + 1

rs.Edit
rs("ROLTried") = ROLTried
rs.Update

'Find the program he/she has ranked as ROLTried ...
'See if that program has unfilled position ...
'See if that program has ranked him/her ...
'If so assign as tentative match, otherwise loop for another try ...

ProgramID = DLookup("ProgramID", "Application", "ApplicantID = " & ApplicantID & " And ApplicantRankOrder =
" & ROLTried)

'See if that program has ranked him/her ...
HasBeenRanked = DCount("ApplicantID", "Application", "ProgramID = " & ProgramID & " And ApplicantID = " &
ApplicantID &                    " And ProgramRankOrder Is Not Null")

'How many unfilled positions does the program have?
Unfilled = DLookup("Unfilled", "Program", "ProgramID = " & ProgramID)
```

```
If HasBeenRanked = 1 And Unfilled > 0 Then
'Tentatively match him/her ...
rs.Edit
rs("ROLTried") = ROLTried
rs("TentativeMatch") = True
rs("TentativeProgramID") = ProgramID
rs.Update

'Update Unfilled ...
strSQL2 = "Update Program "
strSQL2 = strSQL2 & "Set Unfilled = [Unfilled]-1 "
strSQL2 = strSQL2 & "Where ProgramID = " & ProgramID
CurrentDb.Execute (strSQL2)

TentativelyMatched = True

End If

If HasBeenRanked = 1 And Unfilled = 0 Then

'See if he/she can bump out someone tentatively matched there ...
MyRank = DLookup("ProgramRankOrder", "Application", "ProgramID = " & ProgramID & " And ApplicanTID = " &
ApplicantID)

strSQL3 = "Select Applicant.ApplicantID As BumpedApplicantID, ProgramRankOrder  "
strSQL3 = strSQL3 & "From Applicant Inner Join Application On ((Applicant.TentativeProgramID =
Application.ProgramID) AND "
strSQL3 = strSQL3 & "(Applicant.ApplicantID = Application.ApplicantID)) "
strSQL3 = strSQL3 & "Where ProgramID = " & ProgramID
strSQL3 = strSQL3 & " And ProgramRankOrder > " & MyRank
strSQL3 = strSQL3 & " Order By 2 DESC"

Set rs3 = CurrentDb.OpenRecordset(strSQL3)

If Not rs3.EOF Then

'Bump the applicant ...
BumpedApplicantID = rs3("BumpedApplicantID")

strSQL4 = "Update Applicant Set TentativeMatch= False, TentativeProgramID = Null"
strSQL4 = strSQL4 & " Where ApplicantID = " & BumpedApplicantID

CurrentDb.Execute (strSQL4)

'Tentatively match our applicant ...
rs.Edit
rs("ROLTried") = ROLTried
rs("TentativeMatch") = True
rs("TentativeProgramID") = ProgramID
rs.Update
```

```
TentativelyMatched = True
End If
End If
Loop 'While Not TentativelyMatched And ROLTried < ProgarmsRanked ...
Loop

MsgBox ("The MATCH is Done!")

End Sub
```

**Sub Initialize()**

```
Dim strSQL As String

'Initialize Unfilled column in Program table ...
strSQL = "Update Program "
strSQL = strSQL & "Set Unfilled = [Positions]"

CurrentDb.Execute (strSQL)

'Initialize columns in Applicant table ...
strSQL = "Update Applicant "
strSQL = strSQL & "Set TentativeMatch= False, TentativeProgramID = Null, ROLTried = 0"

CurrentDb.Execute (strSQL)

'Initialize ProgramsRanked column in Applicant table ...
strSQL = "Update Applicant "
strSQL = strSQL & "Set ProgramsRanked = "
strSQL = strSQL & "DCount('ProgramID', 'Application', 'ApplicantRankOrder Is Not Null AND ApplicantID = ' &
ApplicantID)"

CurrentDb.Execute (strSQL)

End Sub
```

**Function Finished() As Boolean**

```
Dim strSQL As String
Dim rs As Recordset

strSQL = "Select Count(*) As TBD "
strSQL = strSQL & "From Applicant "
strSQL = strSQL & "Where TentativeMatch = False And ROLTried < ProgramsRanked"

Set rs = CurrentDb.OpenRecordset(strSQL)

If rs("TBD") = 0 Then
Finished = True
Else
Finished = False
End If

End Function
```