Generating Combinations: A Case Study In Database Design, Recursion, VBA, And SQL Programming
Mohammad Dadashzadeh, Oakland University, USA
Padmini Varanasi, 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 problem of designing a database to keep track of university degree programs in, say, business analytics, their required core courses and elective groups, and generating all possible curriculum paths available for graduation. The elegant solution marries data modeling skills with programming skills in recursion, VBA and embedded SQL programming 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: Generating Combinations; Microsoft Access; VBA; SQL; Recursion

STATEMENT OF PROBLEM
Consider the problem of designing a database to keep track of university degree programs in, say, business analytics, their required core courses and elective groups, and generating all possible curriculum paths available for graduation. To fix ideas, Oakland University offers two such programs – 1) a 1-year, cohort-based, half on-line format consisting of 10 required courses and no electives, and 2) a self-paced version requiring four core courses and one elective group of 10 courses to choose six from. In the former program, there is only one curriculum path consisting of 10 courses that the students must take. In the latter, there are a combination of $C(10, 6) = 10! / 6! (10-6)! = 210$ possible curriculum paths, each of which would consist of four required courses and six electives. Some of these can be enumerated for clarity as follows:

- 1-year, cohort-based, half on-line program – only one curriculum path
  - MIS 514, MIS 515, MIS 516, MIS 650, MIS 604, MIS 606, MIS 636, MIS 546, QMM 640, MIS 680
- Self-paced program – showing five of 210 possible curriculum paths
  - MIS 514, MIS 515, MIS 516, MIS 650, MIS 604, MIS 606, MIS 636, MIS 546, QMM 640, MIS 680
  - MIS 514, MIS 515, MIS 516, MIS 517, MIS 650, MIS 604, MIS 606, MIS 636, MIS 546, QMM 640, MIS 680
  - MIS 514, MIS 515, MIS 516, MIS 517, MIS 650, MIS 604, MIS 606, MIS 636, MIS 546, QMM 640, QMM 652
  - MIS 514, MIS 515, MIS 516, MIS 517, MIS 650, QMM 640, QMM 652, MIS 606, MIS 622, MIS 624, MIS 636

Developing a general solution for the problem of generating all possible curriculum paths is made more challenging by the fact that programs can have an arbitrary number of course groups and that the total number of courses required for graduation can also vary. A further complicating factor is that it is possible to allow students to choose between two course groups – a situation that we will exclude from consideration in the solution presented in the following sections.
DATABASE DESIGN

Figure 1 provides the conceptual data model for the problem as an Entity-Relationship Diagram. Given input data about programs, course groups, and courses, we need to create for each program all instances of possible curriculum paths along with the courses each consists of.

![Conceptual Data Model](image1.png)

The physical database, as implemented in Microsoft Access, is shown as a relationship screen in Figure 2. Each entity has been modeled as a table with an auto number primary key. The 1-to-m relationships requires and makes_possible have been collapsed and captured through ProgramID foreign key in the tables CourseGroup and CurriculumPath. The m-to-m relationships are modeled as similarly named tables - Includes and ConsistsOf.

![Physical Data Model as Implemented in Microsoft Access](image2.png)
SOLUTION ARCHITECTURE

For each program, each curriculum path can be envisioned as the juxtaposition of one permissible string of courses from each of the program’s course groups. To fix ideas, assume that a program has three course groups represented by the following sets:

A = { a1, a2, a3, a4 } with three to choose from
B = { b1, b2, b3 } with two to choose from
C = { c1, c2, c3, c4, c5, c6, c7, c8, c9 } with five to choose from

A possible curriculum path can then be strung together from the following strings: <a2, a3, a4>; <b1, b3>; and <c2, c4, c6, c8, c9>. And, all possible curriculum paths can be obtained by the Cartesian Product of three sets, each set consisting of all permissible combinations for each of the three course groups.

The overall architecture for the solution can therefore be sketched as follows:

For each Program P

- For each Course Group CGi belonging to P:
  - Create the set TableNamei of all permissible course combinations in that group.
  - Create ProductSet as the Cartesian Product of all TableNamei’s belonging to CGi.
- For each element in ProductSet:
  - Insert a row into CurriculumPath table and as many rows as needed into ConsistsOf table.

The Appendix presents the entire solution, as implemented in Access VBA. In the following sections, each major step is described in some detail.

Generating All Possible Curriculum Paths

The program utilizes several dynamic arrays that can best be described using examples. The arrayCourse is an array to hold the courses (actually CourseIDs) for a course group such as {MIS 604, MIS 606, QMM 652, MIS 546}. The Boolean arrayUsed reflects a single generated combination of r out of N items from the set {1, 2, 3, …, N} by setting the index position of each chosen item as True. For example, arrayUsed could assume the values {True, False, True, True} reflecting the combination <1, 3, 4> in selecting three out of four items. The CombinationsTable is a two-dimensional table with one row for each possible combination of r out of N items from the set {1, 2, 3, …, N}. Using our example of three out of four courses, CombinationsTable would be populated with C(4, 3) = 4 rows and 3 columns as shown below:

<table>
<thead>
<tr>
<th>CombinationsTable</th>
<th>Reflecting Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>MIS 604, MIS 606, QMM 652</td>
</tr>
<tr>
<td>1 2 4</td>
<td>MIS 604, MIS 606, MIS 546</td>
</tr>
<tr>
<td>1 3 4</td>
<td>MIS 604, QMM 652, MIS 546</td>
</tr>
<tr>
<td>2 3 4</td>
<td>MIS 606, QMM 652, MIS 546</td>
</tr>
</tbody>
</table>

The algorithm used for generating r out of N combinations from the set {1, 2, …, N} is a recursive one based on generating all such combinations including N and then all such combinations excluding N (Knuth, 2005; Rosetta Code, 2013). The following subroutine (see Code Segment in Figure 3) implements this recursion using the arrayUsed and CombinationsTable data structures.

Figure 3: Recursive Subroutine for Generating Combinations of r out of N Items Form the Set {1, 2, …, N}

Sub GenerateCombinations(ByVal N As Integer, ByVal r As Integer)
    'Recursive algorithm to generate all combinations of r out of N items
    'from the set {1, 2, …, N}
    'arrayUsed, CombinationsTable, and RowNumber are declared globally
Dim I As Integer
Dim ColumnNumber As Integer

If r = 0 Then

' We have selected r out of N ...
' Print the combination generated in debug window ...
' Add the generated combination as a row to the CombinationsTable ...

RowNumber = RowNumber + 1
ColumnNumber = 0

For I = 1 To UBound(arrayUsed)
    If arrayUsed(I) = True Then
        Debug.Print I;
        ColumnNumber = ColumnNumber + 1
        CombinationsTable(RowNumber, ColumnNumber) = I
    End If
Next I

Debug.Print

ElseIf (N < 1) Then

' We have no more items to choose from ...
Exit Sub

Else

' Use N in the combination to be generated ...
arrayUsed(N) = True
' Go and generate r-1 out of the remaining N-1 ...
Call GenerateCombinations(N - 1, r - 1)

' Don't use N in the combination to be generated ...
arrayUsed(N) = False
' Go and generate r out of the remaining N-1 ...
Call GenerateCombinations(N - 1, r)

End If

End Sub

The main subroutine (see Code Segment in Figure 4) executing the solution sketched previously utilizes three record sets to iterate through programs, course groups in a specific program, and courses in a particular course group. It calls upon the GenerateCombinations subroutine above, as well as subroutines, to create temporary tables for each CombinationsTable generated and to form the Cartesian Product table that will consist of one row for each possible curriculum path in the program.
Figure 4: The Main Subroutine Implementing the Solution Architecture

Sub GenerateAllCurriculumPaths()

'Generate all possible curriculum paths along with courses each consists of ...

Dim rs As Recordset, rs2 As Recordset, rs3 As Recordset
Dim ProgramID As Integer, CourseGroupID As Integer
Dim NCG As Integer, NCGCount As Integer, N As Integer, r As Integer
Dim I As Integer, K As Integer
Dim strSQL As String

For each program ...
Set rs = CurrentDb.OpenRecordset("Program")

Do While Not rs.EOF

    ProgramID = rs("ProgramID")

    'Number of Course Groups for this program ...
    NCG = DCount("CourseGroupID", "CourseGroup", "ProgramID = " & ProgramID)

    strSQL = "SELECT * FROM CourseGroup "
    strSQL = strSQL & "WHERE ProgramID = " & ProgramID

    Set rs2 = CurrentDb.OpenRecordset(strSQL)
    NCGCount = 0 'Set counter of number of course groups for this program to zero ...

    'For each CourseGroup in this program ...
    Do While Not rs2.EOF

        CourseGroupID = rs2("CourseGroupID")
        r = rs2("RequiredNumberOfCourses")

        'Now get the courses in that group ...
        strSQL = "SELECT * FROM Includes "
        strSQL = strSQL & "WHERE CourseGroupID = " & CourseGroupID

        Set rs3 = CurrentDb.OpenRecordset(strSQL)

        'Re-dimension the global array to hold CourseIDs of courses in this Course Group ...
        ReDim arrayCourse(DCount("CourseID", "Includes", "CourseGroupID =" & CourseGroupID))

        K = 0

        Do While Not rs3.EOF

            K = K + 1
            arrayCourse(K) = rs3("CourseID")

            rs3.MoveNext

        Loop

    'We now have our N courses to select from ...
    N = K

    'We need all possible combinations of r out of N ...
Creating the Table of All Permissible Course Combinations in a Course Group

For each course group with \( N \) courses and \( r \) number of required courses, the algorithm produces - in an in-memory data structure (a two-dimensional array) named \( \text{CombinationsTable} \) - all possible combinations of \( r \) out of \( N \) items to form the set \{1, 2, …, N\}. As such, the \( \text{CombinationsTable} \) has \( C(N, r) \) rows and \( r \) columns. The mapping of a particular row, such as \( <1,2,3,5,6,10> \), to the corresponding CourseIDs in \( \text{arrayCourse} \) (i.e., first, second, third, fifth, sixth, and tenth CourseIDs) and storing those CourseIDs as a row in a temporary table containing all possible combinations of \( r \) courses to choose from the \( N \) courses in that course group, is done in the following subroutine (see Code Segment in Figure 5) which creates a table with \( r \) columns named TableName-F1, …, TableName-Fr, similar to the partial one shown below.
Sub CreateLoadCourseGroupCombinationsTable(ByVal tn As String, ByVal N As Integer)
'
'Produces a table from the CombinationsTable array ...
'tn is TableName ...
'N is NumberOfColumns ...
'
Dim tdf As TableDef
Dim TableExists As Boolean
Dim strSQL As String
Dim I As Integer, J As Integer
'
'DROP TABLE if it already exists ...
For Each tdf In CurrentDb.TableDefs
    If tdf.Name = tn Then
        TableExists = True
        Exit For
    End If
Next

If TableExists Then
    strSQL = "Drop Table " & tn
    CurrentDb.Execute strSQL
End If
'
'Start with CREATE TABLE ...
strSQL = "CREATE TABLE " & tn & ")"
For I = 1 To N
    strSQL = strSQL & ", " & tn & ", " & I & " integer,"
Next
strSQL = Left(strSQL, Len(strSQL) - 1)
strSQL = strSQL & ")"
CurrentDb.Execute strSQL
'
'Now load INSERT INTO ...
For I = 1 To RowNumber

Figure 5: Creating the Table of All Permissible Course Combinations in a Course Group

<table>
<thead>
<tr>
<th>TableName2-F1</th>
<th>TableName2-F2</th>
<th>TableName2-F3</th>
<th>TableName2-F4</th>
<th>TableName2-F5</th>
<th>TableName2-F6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

Table Name: TableName2

Figure 5 cont.

strSQL = "INSERT INTO " & tn & " VALUES("n

For J = 1 To N

'CombinationsTable(I, J)=1 reflects first CourseID in the arrayCourse ...
'CombinationsTable(I, J)=5 reflects fifth CourseID ...
'etc ...

strSQL = strSQL & arrayCourse(CombinationsTable(I, J)) & ","

Next

strSQL = Left(strSQL, Len(strSQL) - 1)
strSQL = strSQL & ")"

CurrentDb.Execute strSQL

Next

End Sub

Creating the Cartesian Product Table ProductSet

Given TableName1, TableName2, ..., TableNameK, the ProductSet table is created using the following SQL statement: SELECT * INTO ProductSet FROM TableName1,TableName2 as part of the following subroutine (see Code Segment in Figure 6). Each row of CourseIDs in the resulting ProductSet table represents a curriculum path.

<table>
<thead>
<tr>
<th>ProductSet</th>
<th>TableName</th>
<th>TableName</th>
<th>TableName</th>
<th>TableName</th>
<th>TableName</th>
<th>TableName</th>
<th>TableName</th>
<th>TableName</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-F1</td>
<td>1-F2</td>
<td>1-F3</td>
<td>1-F4</td>
<td>2-F1</td>
<td>2-F2</td>
<td>2-F3</td>
<td>2-F4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>
Inserting a CurriculumPath Row and Associated ConsistsOf Rows

From each row in the ProductSet table, a CurriculumPath row for the associated program must be created. Furthermore, for as many columns as there are in the ProductSet table, one row must be created in the ConsistOf table for the newly inserted CurriculumPath. In the following subroutine (see Code Segment in Figure 7), the SQL INSERT INTO CurriculumPath(ProgramID) VALUES(1) statement creates the CurriculumPath row allowing Access to supply the auto number value for CurriculumID which is then retrieved using the DMax function and used for inserting the needed rows into ConsistsOf table using SQL: INSERT INTO ConsistsOf(CurriculumID, CourseID) VALUES(1086,2).

Figure 7: Inserting a CurriculumPath Row and Associated ConsistsOf Rows

```vba
Sub InsertFromProductSetTable(ByVal ProgramID As Integer)
'Add a new curriculum path ...
'And, insert the courses it requires ...
Dim strSQL As String
Dim rs As Recordset
Dim I As Integer, AssignedCurriculumID As Integer, CourseID As Integer

'Loop through ProductSet table and insert a record for each CourseID into ConsistsOf ...
Set rs = CurrentDb.OpenRecordset("ProductSet")
Do While Not rs.EOF
  strSQL = "INSERT INTO CurriculumPath(ProgramID) VALUES(" & ProgramID & ")"
  CurrentDb.Execute strSQL
  'Determine the Autonumber ID value assigned ...
  AssignedCurriculumID = DMax("CurriculumID", "CurriculumPath")
  For I = 1 To rs.Fields.Count
    CourseID = rs.Fields(I - 1).Value
    strSQL = "INSERT INTO ConsistsOf(CurriculumID, CourseID) VALUES("
    strSQL = strSQL & AssignedCurriculumID & ","
    strSQL = strSQL & CourseID & ")"
    CurrentDb.Execute strSQL
  Next
  rs.MoveNext
Loop
End Sub
```

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 recursive functions. 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 is remarkable in that it starts as an interesting conceptual modeling problem of designing a database to keep track of all possible curriculum paths in university degree programs, each of which offers its own multiple baskets of core/required courses and elective courses to choose from. As a database design problem, it offers opportunities to review how to model inherently many-to-many relationships and how to decompose those to 1-to-many relationships available in the relational data model.

With the database design discussion out of the way, the challenge of how to populate the database, by automatically generating all possible combinations of courses that comprise a curriculum path, provides a fertile ground for exploration of the limits of a non-procedural language, such as SQL. The recognition that generating these combinations demands some sort of iteration with a termination condition, and that it cannot be accomplished simply with queries, paves the way for a discussion of alternative approaches to DBMS programming. In the context of Microsoft Access, which is the principle DBMS used in our database course, that discussion allows us to compare macro programming versus VBA.

The greatest opportunity for teaching comes in leading students in implementing the easily-understood algorithm. The solution architecture lends itself to intermediate implementation problems to consider and solve - how-to produce a set of strings representing combinations of courses, how-to store each set in a temporary table in the database, how-to handle arbitrary number of such sets for a university program, how-to combine multiple strings into a complete curriculum path using Cartesian product of sets, and how-to accomplish this using the full power of SQL query and update statements. The student reactions to this divide and conquer approach in completing this case study assignment in their database course is positive and supportive of the recognition of the value of programming coverage in their MIS training.

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 course with better problem-solving skills (Dadashzadeh, 2007). This paper has presented an ideal opportunity to provide an integrative case study problem that demands a solution combining data modeling skills with programming skills in recursion, VBA and embedded SQL programming that works remarkably well in teaching students the value of each tool in the toolset they take away from the core courses.

AUTHOR INFORMATION

Mohammad Dadashzadeh serves as Professor of MIS and Chair of Department of Decision and Information Sciences and the coordinator of the 1-year, half on-line program leading to a Master of Science in IT Management focusing in Business Analytics from 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. E-mail: dadashza@oakland.edu (Corresponding author)

Padmini Varanasi earned her Master of Science in IT Management focusing in Business Analytics at Oakland University in 2013. Her capstone project was on sentiment analysis of emergency room patient satisfaction surveys using SPSS Text Analytics. E-mail: pvaranas@oakland.edu

REFERENCES

APPENDIX

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

'Seek program to generate all possible Curriculum Paths ...
Dim arrayCourse() As Integer
Dim arrayUsed() As Boolean
Dim CombinationsTable() As Integer
Dim RowNumber As Integer

Sub GenerateAllCurriculumPaths()
'Seek all possible curriculum paths along with courses each consists of ...
Dim rs As Recordset, rs2 As Recordset, rs3 As Recordset
Dim ProgramID As Integer, CourseGroupID As Integer
Dim NCG As Integer, NCGCount As Integer, N As Integer, r As Integer
Dim I As Integer, K As Integer
Dim strSQL As String

'For each program ...
Set rs = CurrentDb.OpenRecordset("Program")
Do While Not rs.EOF

    ProgramID = rs("ProgramID")

    'Number of Course Groups for this program ...
    NCG = DCount("CourseGroupID", "CourseGroup", "ProgramID = " & ProgramID)

    strSQL = "SELECT * FROM CourseGroup "
    strSQL = strSQL & "WHERE ProgramID = " & ProgramID
    Set rs2 = CurrentDb.OpenRecordset(strSQL)
    NCGCount = 0 'Set counter of number of course groups for this program to zero ...

    'For each CourseGroup in this program ...
    Do While Not rs2.EOF

        CourseGroupID = rs2("CourseGroupID")
        r = rs2("RequiredNumberOfCourses")

        'Now get the courses in that group ...
        strSQL = "SELECT * FROM Includes "
        strSQL = strSQL & "WHERE CourseGroupID = " & CourseGroupID
        Set rs3 = CurrentDb.OpenRecordset(strSQL)

        'Re-dimension the global array to hold CourseIDs of courses in this Course Group ...
        ReDim arrayCourse(DCount("CourseID", "Includes", "CourseGroupID=" & CourseGroupID))
        K = 0

The Clute Institute
Copyright by author(s) Creative Commons License CC-BY
Do While Not rs3.EOF

    K = K + 1
    arrayCourse(K) = rs3("CourseID")

    rs3.MoveNext

Loop

' We now have our N courses to select from ...
N = K

' We need all possible combinations of r out of N ...
' Re-dimension the global arrays to have room ...

    ReDim arrayUsed(N)
    ReDim CombinationsTable(HowManyCombinations(r, N), r)

RowNumber = 0
Call GenerateCombinations(N, r)
Debug.Print RowNumber & " combinations generated."

' Now, create an Access table out of the CombinationsTable array ...
' To hold all course combinations for that course group ...

    NCGCount = NCGCount + 1
    Call CreateLoadCourseGroupCombinationsTable("TableName" & NCGCount, r)

    rs2.MoveNext

Loop

' We have created a separate table of all course combinations for each course group ...
' Now, we need to do Cartesian Product of all of them to make curriculum paths...

    Call CreateProductSetTable(NCGCount)

' Now, we insert new curriculum path rows and what courses each consists of ...
    Call InsertFromProductSetTable(ProgramID)

' We can delete the ProductSet table for next program ...
    strSQL = "DROP TABLE ProductSet"
    CurrentDb.Execute strSQL
    rs.MoveNext 'Move to the next program ...

Loop

MsgBox ("Done!")

End Sub

Sub CreateLoadCourseGroupCombinationsTable(ByVal tn As String, ByVal N As Integer)

' Produces a table from the CombinationsTable array ...
' 'tn is TableName ...
' 'N is NumberOfColumns ...

    Dim tdf As TableDef
Dim TableExists As Boolean
Dim strSQL As String
Dim I As Integer, J As Integer

' DROP TABLE if it already exists ...
For Each tdf In CurrentDb.TableDefs
    If tdf.Name = tn Then
        TableExists = True
        Exit For
    End If
Next

If TableExists Then
    strSQL = "Drop Table " & tn
    CurrentDb.Execute strSQL
End If

' Start with CREATE TABLE ...
strSQL = "CREATE TABLE " & tn & " ("
For I = 1 To N
    strSQL = strSQL & tn & "_F" & I & " integer," Next
strSQL = Left(strSQL, Len(strSQL) - 1)
strSQL = strSQL & " )"
CurrentDb.Execute strSQL

' Now load INSERT INTO ...
For I = 1 To RowNumber
    strSQL = "INSERT INTO " & tn & " VALUES("
    For J = 1 To N
        strSQL = strSQL & arrayCourse(CombinationsTable(I, J)) & ","
    Next
    strSQL = Left(strSQL, Len(strSQL) - 1)
    strSQL = strSQL & " )"
    CurrentDb.Execute strSQL
Next

End Sub
Sub CreateProductSetTable(ByVal NumberOfTables As Integer)
    'Forms the ProductSet table by Cartesian Product of TableName1, TableName2, ...
    Dim strSQL As String, I As Integer
    strSQL = "SELECT * INTO ProductSet FROM "
    For I = 1 To NumberOfTables
        strSQL = strSQL & "TableName" & I & ","
    Next
    strSQL = Left(strSQL, Len(strSQL) - 1)
    CurrentDb.Execute strSQL
End Sub

Sub InsertFromProductSetTable(ByVal ProgramID As Integer)
    'Add a new curriculum path ...
    'And, insert the courses it requires ...
    Dim strSQL As String
    Dim rs As Recordset
    Dim I As Integer, AssignedCurriculumID As Integer, CourseID As Integer
    'Loop through ProductSet table and insert a record for each CourseID into ConsistsOf ...
    Set rs = CurrentDb.OpenRecordset("ProductSet")
    Do While Not rs.EOF
        strSQL = "INSERT INTO CurriculumPath(ProgramID) VALUES(," & ProgramID & ")"
        CurrentDb.Execute strSQL
        'Determine the Autonumber ID value assigned ...
        AssignedCurriculumID = DMax("CurriculumID", "CurriculumPath")
        For I = 1 To rs.Fields.Count
            CourseID = rs.Fields(I - 1).Value
            strSQL = "INSERT INTO ConsistsOf(CurriculumID, CourseID) VALUES(," & AssignedCurriculumID & "," & CourseID & ")"
            CurrentDb.Execute strSQL
        Next
        rs.MoveNext
    Loop
End Sub
Sub GenerateCombinations(ByVal N As Integer, ByVal r As Integer)
   'Recursive algorithm to generate all combinations of r out of N items
   'from the set {1, 2, ..., N}
   'arrayUsed, CombinationsTable, and RowNumber are declared globally ...
   Dim I As Integer
   Dim ColumnNumber As Integer
   If r = 0 Then
      'We have selected r out of N ...
      'Print the combination generated in debug window ...
      'Add the generated combination as a row to the CombinationsTable ...
      RowNumber = RowNumber + 1
      ColumnNumber = 0
      For I = 1 To UBound(arrayUsed)
         If arrayUsed(I) = True Then
            Debug.Print I;
            ColumnNumber = ColumnNumber + 1
            CombinationsTable(RowNumber, ColumnNumber) = I
         End If
      Next I
      Debug.Print
   ElseIf (N < 1) Then
      'We have no more items to choose from ...
      Exit Sub
   Else
      'Use N in the combination to be generated ...
      arrayUsed(N) = True
      'Go and generate r-1 out of the remaining N-1 ...
      Call GenerateCombinations(N - 1, r - 1)
      'Don't use N in the combination to be generated ...
      arrayUsed(N) = False
      'Go and generate r out of the remaining N-1 ...
      Call GenerateCombinations(N - 1, r)
   End If
End Sub

Function Factorial(ByVal N As Single) As Single
   'Returns N!
   If N <= 1 Then
      Fact = 1
   ElseIf N > 1 Then
      Fact = N * Factorial(N - 1)
   End If
End Function
Factorial = 1

Else

    Factorial = N * Factorial(N - 1)

End If

End Function

Function HowManyCombinations(ByVal M As Integer, ByVal N As Integer) As Integer

    'Returns number of combinations of M out of N ...
    'HowManyCombinations = Factorial(N) / (Factorial(M) * Factorial(N - M))

End Function