1. Using UNION to make tables longer and to append similar tables

(1a) Creating a unified CompactResultsUnion query

Our listing of games contains one column for the Winning Team ID, and one column for the Losing Team ID. Because Team ID's are in two different columns, it makes it trickier to query because you have to check both columns, and sometimes have separate logic for the two cases.

Instead, it makes it easier on the query user to make the data twice as long and to have each game appear twice, so instead of Winning Team ID and Losing Team ID, it is just Team ID and Opponent Team ID, with an additional column called Result that tells you whether Team ID won or not. This way you can just look at Team ID. So a great first step is to build views that do this, using a UNION operation.

```
CREATE VIEW NCAATourneyCompactResultsUnion
  AS
SELECT Season, DayNum, WTeamID as TeamID, WScore as Score, LTeamID as OppTeamID, LScore as OppScore,
1.00 as Result, WLoc as Loc, NumOT
FROM NCAATourneyCompactResults NTCR
  UNION ALL
SELECT Season, DayNum, LTeamID as TeamID, LScore as Score, WTeamID as OppTeamID, WScore as OppScore,
0.00 as Result, CASE WLoc WHEN 'H' THEN 'A' WHEN 'A' THEN 'H' ELSE WLoc END as Loc, NumOT
FROM NCAATourneyCompactResults NTCR
GO

CREATE VIEW RegularSeasonCompactResultsUnion
  AS
SELECT Season, DayNum, WTeamID as TeamID, WScore as Score, LTeamID as OppTeamID, LScore as OppScore,
1.00 as Result, WLoc as Loc, NumOT
FROM RegularSeasonCompactResults NTCR
  UNION ALL
SELECT Season, DayNum, LTeamID as TeamID, LScore as Score, WTeamID as OppTeamID, WScore as OppScore,
0.00 as Result, CASE WLoc WHEN 'H' THEN 'A' WHEN 'A' THEN 'H' ELSE WLoc END as Loc, NumOT
FROM RegularSeasonCompactResults NTCR
GO

CREATE VIEW SecondaryTourneyCompactResultsUnion
  AS
SELECT Season, DayNum, WTeamID as TeamID, WScore as Score, LTeamID as OppTeamID, LScore as OppScore,
1.00 as Result, WLoc as Loc, NumOT
FROM SecondaryTourneyCompactResults NTCR
  UNION ALL
SELECT Season, DayNum, LTeamID as TeamID, LScore as Score, WTeamID as OppTeamID, WScore as OppScore,
0.00 as Result, CASE WLoc WHEN 'H' THEN 'A' WHEN 'A' THEN 'H' ELSE WLoc END as Loc, NumOT
FROM SecondaryTourneyCompactResults NTCR
GO
```

This gives three different queries that are all quite similar – they are all "Compact Results" queries that return the same columns, just for three different types of games (NCAA tourney games, Regular Season games, and secondary tourney games such as the NIT). The UNION operation can be used again, to combine those views together into a single view that returns rows from all three types of games:

```
CREATE VIEW CompactResultsUnion
  AS
SELECT Season, DayNum, TeamID, Score, OppTeamID, OppScore, Result, Loc, NumOT, 'NCAA' as RType FROM
NCAATourneyCompactResultsUnion
```

```
  UNION ALL
SELECT Season, DayNum, TeamID, Score, OppTeamID, OppScore, Result, Loc, NumOT, 'Regular' as RType FROM
RegularSeasonCompactResultsUnion
  UNION ALL
SELECT Season, DayNum, TeamID, Score, OppTeamID, OppScore, Result, Loc, NumOT, 'Secondary' as RType
FROM SecondaryTourneyCompactResultsUnion
GO
```

(1b) Creating a unified DetailedResultsUnion query

This same two-step process (converting from Winning Team and Losing Team, into just Team and Opponent Team, and then appending similar queries one after the other via UNION) can be done with the DetailedResults data views (i.e. with team box scores):

```
CREATE VIEW NCAATourneyDetailedResultsUnion
  AS
SELECT Season, DayNum, WTeamID as TeamID, WScore as Score, LTeamID as OppTeamID, LScore as OppScore,
1.00 as Result, WLoc as Loc, NumOT
, WFGM as OwnFGM, WFGA as OwnFGA, WFGM3 as OwnFGM3, WFGA3 as OwnFGA3, WFTM as OwnFTM, WFTA as OwnFTA,
WOR as OwnOR, WDR as OwnDR, WAst as OwnAst, WTO as OwnTO, WStl as OwnStl, WBlk as OwnBlk, WPF as OwnPF
, LFGM as OppFGM, LFGA as OppFGA, LFGM3 as OppFGM3, LFGA3 as OppFGA3, LFTM as OppFTM, LFTA as OppFTA,
LOR as OppOR, LDR as OppDR, [LAst] as OppAst, LTO as OppTO, LStl as OppStl, LBlk as OppBlk, LPF as
OppPF
FROM NCAATourneyDetailedResults NTCR
  UNION ALL
SELECT Season, DayNum, LTeamID as TeamID, LScore as Score, WTeamID as OppTeamID, WScore as OppScore,
0.00 as Result, CASE WLoc WHEN 'H' THEN 'A' WHEN 'A' THEN 'H' ELSE WLoc END as Loc, NumOT
, LFGM as OwnFGM, LFGA as OwnFGA, LFGM3 as OwnFGM3, LFGA3 as OwnFGA3, LFTM as OwnFTM, LFTA as OwnFTA,
LOR as OwnOR, LDR as OwnDR, [LAst] as OwnAst, LTO as OwnTO, LStl as OwnStl, LBlk as OwnBlk, LPF as
OwnPF
, WFGM as OppFGM, WFGA as OppFGA, WFGM3 as OppFGM3, WFGA3 as OppFGA3, WFTM as OppFTM, WFTA as OppFTA,
WOR as OppOR, WDR as OppDR, WAst as OppAst, WTO as OppTO, WStl as OppStl, WBlk as OppBlk, WPF as OppPF
FROM NCAATourneyDetailedResults NTCR
GO

CREATE VIEW RegularSeasonDetailedResultsUnion
  AS
SELECT Season, DayNum, WTeamID as TeamID, WScore as Score, LTeamID as OppTeamID, LScore as OppScore,
1.00 as Result, WLoc as Loc, NumOT
, WFGM as OwnFGM, WFGA as OwnFGA, WFGM3 as OwnFGM3, WFGA3 as OwnFGA3, WFTM as OwnFTM, WFTA as OwnFTA,
WOR as OwnOR, WDR as OwnDR, WAst as OwnAst, WTO as OwnTO, WStl as OwnStl, WBlk as OwnBlk, WPF as OwnPF
, LFGM as OppFGM, LFGA as OppFGA, LFGM3 as OppFGM3, LFGA3 as OppFGA3, LFTM as OppFTM, LFTA as OppFTA,
LOR as OppOR, LDR as OppDR, [LAst] as OppAst, LTO as OppTO, LStl as OppStl, LBlk as OppBlk, LPF as
OppPF
FROM RegularSeasonDetailedResults NTCR
  UNION ALL
SELECT Season, DayNum, LTeamID as TeamID, LScore as Score, WTeamID as OppTeamID, WScore as OppScore,
0.00 as Result, CASE WLoc WHEN 'H' THEN 'A' WHEN 'A' THEN 'H' ELSE WLoc END as Loc, NumOT
, LFGM as OwnFGM, LFGA as OwnFGA, LFGM3 as OwnFGM3, LFGA3 as OwnFGA3, LFTM as OwnFTM, LFTA as OwnFTA,
LOR as OwnOR, LDR as OwnDR, [LAst] as OwnAst, LTO as OwnTO, LStl as OwnStl, LBlk as OwnBlk, LPF as
OwnPF
, WFGM as OppFGM, WFGA as OppFGA, WFGM3 as OppFGM3, WFGA3 as OppFGA3, WFTM as OppFTM, WFTA as OppFTA,
WOR as OppOR, WDR as OppDR, WAst as OppAst, WTO as OppTO, WStl as OppStl, WBlk as OppBlk, WPF as OppPF
FROM RegularSeasonDetailedResults NTCR
GO

CREATE VIEW DetailedResultsUnion
  AS
SELECT Season, DayNum, TeamID, Score, OppTeamID, OppScore, Result, Loc, NumOT, OwnFGM, OwnFGA, OwnFGM3,
OwnFGA3, OwnFTM, OwnFTA, OwnOR, OwnDR, OwnAst, OwnTO, OwnStl, OwnBlk, OwnPF, OppFGM, OppFGA, OppFGM3,
OppFGA3, OppFTM, OppFTA, OppOR, OppDR, OppAst, OppTO, OppStl, OppBlk, OppPF, 'NCAA' as RType FROM
NCAATourneyDetailedResultsUnion
```

```
  UNION ALL
SELECT Season, DayNum, TeamID, Score, OppTeamID, OppScore, Result, Loc, NumOT, OwnFGM, OwnFGA, OwnFGM3,
OwnFGA3, OwnFTM, OwnFTA, OwnOR, OwnDR, OwnAst, OwnTO, OwnStl, OwnBlk, OwnPF, OppFGM, OppFGA, OppFGM3,
OppFGA3, OppFTM, OppFTA, OppOR, OppDR, OppAst, OppTO, OppStl, OppBlk, OppPF, 'Regular' as RType FROM
RegularSeasonDetailedResultsUnion
GO
```

(1c) Creating a unified WCompactResultsUnion query

This same two-step process can be done with the women's CompactResults data views although there is no SecondaryCompactResults data for the women:

```
CREATE VIEW WNCAATourneyCompactResultsUnion
  AS
SELECT Season, DayNum, WTeamID as TeamID, WScore as Score, LTeamID as OppTeamID, LScore as OppScore,
1.00 as Result, WLoc as Loc, NumOT
FROM WNCAATourneyCompactResults NTCR
  UNION ALL
SELECT Season, DayNum, LTeamID as TeamID, LScore as Score, WTeamID as OppTeamID, WScore as OppScore,
0.00 as Result, CASE WLoc WHEN 'H' THEN 'A' WHEN 'A' THEN 'H' ELSE WLoc END as Loc, NumOT
FROM WNCAATourneyCompactResults NTCR
GO

CREATE VIEW WRegularSeasonCompactResultsUnion
  AS
SELECT Season, DayNum, WTeamID as TeamID, WScore as Score, LTeamID as OppTeamID, LScore as OppScore,
1.00 as Result, WLoc as Loc, NumOT
FROM WRegularSeasonCompactResults NTCR
  UNION ALL
SELECT Season, DayNum, LTeamID as TeamID, LScore as Score, WTeamID as OppTeamID, WScore as OppScore,
0.00 as Result, CASE WLoc WHEN 'H' THEN 'A' WHEN 'A' THEN 'H' ELSE WLoc END as Loc, NumOT
FROM WRegularSeasonCompactResults NTCR
GO

CREATE VIEW WCompactResultsUnion
  AS
SELECT Season, DayNum, TeamID, Score, OppTeamID, OppScore, Result, Loc, NumOT, 'NCAA' as RType FROM
WNCAATourneyCompactResultsUnion
  UNION ALL
SELECT Season, DayNum, TeamID, Score, OppTeamID, OppScore, Result, Loc, NumOT, 'Regular' as RType FROM
WRegularSeasonCompactResultsUnion
GO
```

Thus you will have three main data views on the game results:

CompactResultsUnion
DetailedResultsUnion
WCompactResultsUnion

2. Bring in related fields

In addition, because the overall dataset is structured relationally across many different files/tables, there are many pieces of information (for example, team names, season dates, tourney seeds, coaches, conference affiliations, game locations, etc.) that can be combined with the basic game data via various JOIN operations, into a single query.

```
CREATE VIEW CompactResultsData
  AS
SELECT CRU.Season
, CRU.DayNum
, DATEADD(dd, CRU.DayNum, S.DayZero) as GameDate
, CRU.TeamID
, OwnT.TeamName as TeamName
, CRU.Score
, CONVERT(int,SUBSTRING(OwnSeed.Seed,2,2)) as SeedNumber
, CASE WHEN RIGHT(OwnSeed.Seed,1) IN ('a','b') THEN 1 WHEN OwnSeed.Seed IS NULL THEN NULL ELSE 0 END as
PlayIn
, CRU.OppTeamID
, OppT.TeamName as OppTeamName
, CRU.OppScore
, CONVERT(int,SUBSTRING(OppSeed.Seed,2,2)) as OppSeedNumber
, CASE WHEN RIGHT(OppSeed.Seed,1) IN ('a','b') THEN 1 WHEN OppSeed.Seed IS NULL THEN NULL ELSE 0 END as
OppPlayIn
, CRU.Result
, CRU.Loc
, CRU.NumOT
, CRU.RType
, OwnC.CoachName
, OppC.CoachName as OppCoachName
, OwnTC.ConfAbbrev
, OppTC.ConfAbbrev as OppConfAbbrev
, C.City
, C.State
FROM CompactResultsUnion CRU
  JOIN Seasons S ON CRU.Season = S.Season
  JOIN Teams OwnT ON CRU.TeamID = OwnT.TeamID
  JOIN Teams OppT ON CRU.OppTeamID = OppT.TeamID
  LEFT JOIN NCAATourneySeeds OwnSeed ON CRU.Season = OwnSeed.Season AND CRU.TeamID = OwnSeed.TeamID
  LEFT JOIN NCAATourneySeeds OppSeed ON CRU.Season = OppSeed.Season AND CRU.OppTeamID = OppSeed.TeamID
  JOIN TeamCoaches OwnC ON CRU.Season = OwnC.Season AND CRU.TeamID = OwnC.TeamID AND CRU.DayNum BETWEEN
OwnC.FirstDayNum AND OwnC.LastDayNum
  JOIN TeamCoaches OppC ON CRU.Season = OppC.Season AND CRU.OppTeamID = OppC.TeamID AND CRU.DayNum
BETWEEN OppC.FirstDayNum AND OppC.LastDayNum
  JOIN TeamConferences OwnTC ON CRU.Season = OwnTC.Season AND CRU.TeamID = OwnTC.TeamID
  JOIN TeamConferences OppTC ON CRU.Season = OppTC.Season AND CRU.OppTeamID = OppTC.TeamID
  LEFT JOIN GameCities GC ON CRU.Season = GC.Season AND CRU.DayNum = GC.DayNum
     AND CRU.TeamID = CASE WHEN CRU.Result = 1.0 THEN GC.WTeamID ELSE GC.LTeamID END
     AND CRU.OppTeamID = CASE WHEN CRU.Result = 1.0 THEN GC.LTeamID ELSE GC.WTeamID END
  LEFT JOIN Cities C ON GC.CityID = C.CityID
GO
```

And we can also do this for the detailed results (with team box score data):

```
CREATE VIEW DetailedResultsData
  AS
SELECT DRU.Season
, DRU.DayNum
, DATEADD(dd, DRU.DayNum, S.DayZero) as GameDate
, DRU.TeamID
, OwnT.TeamName as TeamName
```

```
, DRU.Score
, CONVERT(int,SUBSTRING(OwnSeed.Seed,2,2)) as SeedNumber
, CASE WHEN RIGHT(OwnSeed.Seed,1) IN ('a','b') THEN 1 WHEN OwnSeed.Seed IS NULL THEN NULL ELSE 0 END as
PlayIn
, DRU.OppTeamID
, OppT.TeamName as OppTeamName
, DRU.OppScore
, CONVERT(int,SUBSTRING(OppSeed.Seed,2,2)) as OppSeedNumber
, CASE WHEN RIGHT(OppSeed.Seed,1) IN ('a','b') THEN 1 WHEN OppSeed.Seed IS NULL THEN NULL ELSE 0 END as
OppPlayIn
, DRU.Result
, DRU.Loc
, DRU.NumOT
, DRU.OwnFGM, DRU.OwnFGA, DRU.OwnFGM3, DRU.OwnFGA3, DRU.OwnFTM, DRU.OwnFTA, DRU.OwnOR, DRU.OwnDR
, DRU.OwnAst, DRU.OwnTO, DRU.OwnStl, DRU.OwnBlk, DRU.OwnPF
, DRU.OppFGM, DRU.OppFGA, DRU.OppFGM3, DRU.OppFGA3, DRU.OppFTM, DRU.OppFTA, DRU.OppOR, DRU.OppDR
, DRU.OppAst, DRU.OppTO, DRU.OppStl, DRU.OppBlk, DRU.OppPF
, DRU.RType
, OwnC.CoachName
, OppC.CoachName as OppCoachName
, OwnTC.ConfAbbrev
, OppTC.ConfAbbrev as OppConfAbbrev
, C.City
, C.State
FROM DetailedResultsUnion DRU
  JOIN Seasons S ON DRU.Season = S.Season
  JOIN Teams OwnT ON DRU.TeamID = OwnT.TeamID
  JOIN Teams OppT ON DRU.OppTeamID = OppT.TeamID
  LEFT JOIN NCAATourneySeeds OwnSeed ON DRU.Season = OwnSeed.Season AND DRU.TeamID = OwnSeed.TeamID
  LEFT JOIN NCAATourneySeeds OppSeed ON DRU.Season = OppSeed.Season AND DRU.OppTeamID = OppSeed.TeamID
  JOIN TeamCoaches OwnC ON DRU.Season = OwnC.Season AND DRU.TeamID = OwnC.TeamID AND DRU.DayNum BETWEEN
OwnC.FirstDayNum AND OwnC.LastDayNum
  JOIN TeamCoaches OppC ON DRU.Season = OppC.Season AND DRU.OppTeamID = OppC.TeamID AND DRU.DayNum
BETWEEN OppC.FirstDayNum AND OppC.LastDayNum
  JOIN TeamConferences OwnTC ON DRU.Season = OwnTC.Season AND DRU.TeamID = OwnTC.TeamID
  JOIN TeamConferences OppTC ON DRU.Season = OppTC.Season AND DRU.OppTeamID = OppTC.TeamID
  LEFT JOIN GameCities GC ON DRU.Season = GC.Season AND DRU.DayNum = GC.DayNum
     AND DRU.TeamID = CASE WHEN DRU.Result = 1.0 THEN GC.WTeamID ELSE GC.LTeamID END
     AND DRU.OppTeamID = CASE WHEN DRU.Result = 1.0 THEN GC.LTeamID ELSE GC.WTeamID END
  LEFT JOIN Cities C ON GC.CityID = C.CityID
GO
```

And finally we can also do this for the women's compact results, although not all of the associated data is present for the women's data, so the query is a bit simpler:

```
CREATE VIEW WCompactResultsData
  AS
SELECT CRU.Season
, CRU.DayNum
, DATEADD(dd, CRU.DayNum, S.DayZero) as GameDate
, CRU.TeamID
, OwnT.TeamName as TeamName
, CRU.Score
, CONVERT(int,SUBSTRING(OwnSeed.Seed,2,2)) as SeedNumber
, CRU.OppTeamID
, OppT.TeamName as OppTeamName
, CRU.OppScore
, CONVERT(int,SUBSTRING(OppSeed.Seed,2,2)) as OppSeedNumber
, CRU.Result
, CRU.Loc
, CRU.NumOT
, CRU.RType
, C.City
```

```
, C.State
FROM WCompactResultsUnion CRU
  JOIN WSeasons S ON CRU.Season = S.Season
  JOIN WTeams OwnT ON CRU.TeamID = OwnT.TeamID
  JOIN WTeams OppT ON CRU.OppTeamID = OppT.TeamID
  LEFT JOIN WNCAATourneySeeds OwnSeed ON CRU.Season = OwnSeed.Season AND CRU.TeamID = OwnSeed.TeamID
  LEFT JOIN WNCAATourneySeeds OppSeed ON CRU.Season = OppSeed.Season AND CRU.OppTeamID = OppSeed.TeamID
  LEFT JOIN WGameCities GC ON CRU.Season = GC.Season AND CRU.DayNum = GC.DayNum
     AND CRU.TeamID = CASE WHEN CRU.Result = 1.0 THEN GC.WTeamID ELSE GC.LTeamID END
     AND CRU.OppTeamID = CASE WHEN CRU.Result = 1.0 THEN GC.LTeamID ELSE GC.WTeamID END
  LEFT JOIN WCities C ON GC.CityID = C.CityID
GO
```

At the end of all this view creation, you will have three main data views on the game results:

CompactResultsData
DetailedResultsData
WCompactResultsData

Note that for the men's data, it is useful to maintain both a CompactResultsData view on the data (which doesn't have team box scores, but does stretch back to 1985) and a DetailedResultsData view (which does have team box scores, but only goes back to 2003). For a given application, one or the other of those might prove superior. On the women's side, there are no team box scores currently available, and so only a WCompactResultsData view is needed.

3. Using the denormalized views on the data

Armed with the xResultsData views (CompactResultsData, DetailedResultsData, and WCompactResultsData) you can explore lots of possible factors that might be useful in making predictions. Here are several examples of some interesting queries and what they tell us:

(3a) Mike Krzyzewski is the men's coach with the most tournament wins (since 1985):

```
SELECT TOP 10 CoachName, SUM(1) as NCAATourneyWins
FROM CompactResultsData WHERE RType = 'NCAA' AND Result = 1.0
GROUP BY CoachName ORDER BY 2 DESC
```

(3b) Gonzaga has the most women's tourney wins as a double-digit seed (since 1998):

```
SELECT TeamName, COUNT(*) as DoubleDigitSeedWins
FROM WCompactResultsData WHERE Result = 1.0 AND RType = 'NCAA' AND SeedNumber >= 10
GROUP BY TeamName HAVING COUNT(*) >= 3 ORDER BY 2 DESC
```

(3c) Wisconsin in 2011 had the highest men's single-season free-throw percentage (since 2003):

```
SELECT TOP 10 Season, TeamName, CONVERT(float,SUM(OwnFTM))/CONVERT(float,SUM(OwnFTA)) as FTPct
FROM DetailedResultsData GROUP BY Season, TeamName ORDER BY 3 DESC
```

(3d) Vanderbilt in 2002 was the women's #1 seed having the most regular season losses (since 1998):

```
SELECT Season, TeamName, SUM(1) as NumLosses
FROM WCompactResultsData
WHERE SeedNumber = 1 AND RType = 'Regular' AND Result = 0.0
GROUP BY Season, TeamName HAVING SUM(1) > 5 ORDER BY 2 DESC
```

(3e) #1 seeds have better average "Assist:Turnover" ratios in men's NCAA Tourney games than any other seed:

```
SELECT SeedNumber, SUM(1) as NumGames, CONVERT(float,SUM(OwnAst))/CONVERT(float,SUM(OwnTO)) as
AstTORatio
FROM DetailedResultsData WHERE RType = 'NCAA'
GROUP BY SeedNumber ORDER BY 1
```

(3f) The Big East is the conference with the most upsets of #1 seeds (by non-#1-seeds) in NCAA Tourney games, (since 1985):

```
SELECT TOP 10 ConfAbbrev, SUM(1) as UpsetsOf1Seeds
FROM CompactResultsData WHERE RType = 'NCAA' AND Result = 1.0
AND SeedNumber > 1 AND OppSeedNumber = 1
GROUP BY ConfAbbrev ORDER BY 2 DESC
```

(3g) #2 and #3 seeds are both 80-0 in their first-round games in the women's tournaments since 1998:

```
SELECT SeedNumber, OppSeedNumber, SUM(1) as NumGames, AVG(Result) as WinPct
FROM WCompactResultsData
WHERE RType = 'NCAA' AND SeedNumber + OppSeedNumber = 17
GROUP BY SeedNumber, OppSeedNumber ORDER BY 1
```

(3h) Only six times since 2003 has a worse-seeded team won a men's NCAA tournament game despite giving up a shooting percentage above 55% in the game:

```
SELECT Season, TeamName, SeedNumber, OppTeamName, OppSeedNumber
, CONVERT(float,OppFGM)/CONVERT(float,OppFGA) as OppFGPct
FROM DetailedResultsData
WHERE CONVERT(float,OppFGM)/CONVERT(float,OppFGA) > 0.55 AND RType = 'NCAA'
AND Result = 1.0 AND SeedNumber > OppSeedNumber
```

4. Creating a submission file

For a given year, the NCAATourneySeeds listing can be used to generate the full set of all possible matchups in the tournament. Here is a query to produce a very simple set of predictions, namely a 50% prediction for all matchups, for the four men's years 2014-2017 (matching what is required for Stage 1).

```
SELECT CONVERT(varchar,S.Season) + '_' + CONVERT(varchar,NTSLow.TeamID) + '_' +
CONVERT(varchar,NTSHigh.TeamID) as ID
, 0.5 as Pred
FROM Seasons S
  JOIN NCAATourneySeeds NTSLow ON S.Season = NTSLow.Season
  JOIN NCAATourneySeeds NTSHigh ON S.Season = NTSHigh.Season AND NTSLow.TeamID < NTSHigh.TeamID
WHERE S.Season BETWEEN 2014 AND 2017
ORDER BY 1
```

Or for the women's data, a very similar query, that will return fewer records, because there are only 64 tournament teams rather than 68:

```
SELECT CONVERT(varchar,S.Season) + '_' + CONVERT(varchar,NTSLow.TeamID) + '_' +
CONVERT(varchar,NTSHigh.TeamID) as ID
, 0.5 as Pred
FROM WSeasons S
  JOIN WNCAATourneySeeds NTSLow ON S.Season = NTSLow.Season
  JOIN WNCAATourneySeeds NTSHigh ON S.Season = NTSHigh.Season AND NTSLow.TeamID < NTSHigh.TeamID
WHERE S.Season BETWEEN 2014 AND 2017
ORDER BY 1
```

A more reasonable approach would be to use the tournament seed numbers to drive the prediction calculation. We could envision a simple linear relationship between the difference in seed numbers and the predicted winning percentage. For instance, the winning percentage could be 50% plus 3% for each numerical difference in seeds. So for a #3 seed versus a #9 seed, the #3 seed would be given a 68% prediction (0.50 + 6*0.03), whereas for a #12 seed versus a #4 seed, the #12 seed would be given a 26% prediction (0.50 – 8*0.03). This method of prediction is generally called the "Seed Benchmark" in the Kaggle NCAA contests.

```
SELECT CONVERT(varchar,S.Season) + '_' + CONVERT(varchar,NTSLow.TeamID) + '_' +
CONVERT(varchar,NTSHigh.TeamID) as ID
, 0.50 + 0.03 * (CONVERT(int,SUBSTRING(NTSHigh.Seed,2,2)) - CONVERT(int,SUBSTRING(NTSLow.Seed,2,2))) as
Pred
FROM Seasons S
  JOIN NCAATourneySeeds NTSLow ON S.Season = NTSLow.Season
  JOIN NCAATourneySeeds NTSHigh ON S.Season = NTSHigh.Season AND NTSLow.TeamID < NTSHigh.TeamID
WHERE S.Season BETWEEN 2014 AND 2017
ORDER BY 1
```

Or for the women's data:

```
SELECT CONVERT(varchar,S.Season) + '_' + CONVERT(varchar,NTSLow.TeamID) + '_' +
CONVERT(varchar,NTSHigh.TeamID) as ID
, 0.50 + 0.03 * (CONVERT(int,SUBSTRING(NTSHigh.Seed,2,2)) - CONVERT(int,SUBSTRING(NTSLow.Seed,2,2))) as
Pred
FROM WSeasons S
  JOIN WNCAATourneySeeds NTSLow ON S.Season = NTSLow.Season
  JOIN WNCAATourneySeeds NTSHigh ON S.Season = NTSHigh.Season AND NTSLow.TeamID < NTSHigh.TeamID
WHERE S.Season BETWEEN 2014 AND 2017
ORDER BY 1
```

5. Scoring different prediction approaches

A necessary step, when choosing among multiple predictive algorithms for predicting NCAA tournament games, will be trying out those different approaches by making predictions of past against historical data of your candidates, and seeing which one would have predicted best in the past.

For example, let's consider two possible ways to do a Seed Benchmark:
Option A, prediction = 50% + 2%*(seed difference): {20%, 22%, …, 78%, 80%} for seed differences of {-15, -14, …, +14, and +15}
Option B, prediction = 50% + 3%*(seed difference): {5%, 8%, …, 92%, 95%} for seed differences of {-15, -14, …, +14, and +15}

Option A returns an average binomial deviance of 0.585056 across the 252 men's games (excluding play-in games) from 2014-2017 – those are the Stage 1 games:

```
SELECT -AVG(LOG(Pred)) as BinomialDeviance, SUM(1) as NumGamesScored
FROM NCAATourneyCompactResults R
JOIN (SELECT S.Season, NTS1.TeamID as Team1, NTS2.TeamID as Team2
, 0.50 + 0.02 * (CONVERT(int,SUBSTRING(NTS2.Seed,2,2)) - CONVERT(int,SUBSTRING(NTS1.Seed,2,2))) as Pred
FROM Seasons S
  JOIN NCAATourneySeeds NTS1 ON S.Season = NTS1.Season
  JOIN NCAATourneySeeds NTS2 ON S.Season = NTS2.Season
WHERE S.Season BETWEEN 2014 AND 2017) Preds
ON R.Season = Preds.Season AND R.WTeamID = Preds.Team1 AND R.LTeamID = Preds.Team2
WHERE R.DayNum >= 136
```

Option B returns an average binomial deviance of 0.559008 across the 252 men's games (excluding play-in games) from 2014-2017 – those are the Stage 1 games:

```
SELECT -AVG(LOG(Pred)) as BinomialDeviance, SUM(1) as NumGamesScored
FROM NCAATourneyCompactResults R
JOIN (SELECT S.Season, NTS1.TeamID as Team1, NTS2.TeamID as Team2
, 0.50 + 0.03 * (CONVERT(int,SUBSTRING(NTS2.Seed,2,2)) - CONVERT(int,SUBSTRING(NTS1.Seed,2,2))) as Pred
FROM Seasons S
  JOIN NCAATourneySeeds NTS1 ON S.Season = NTS1.Season
  JOIN NCAATourneySeeds NTS2 ON S.Season = NTS2.Season
WHERE S.Season BETWEEN 2014 AND 2017) Preds
ON R.Season = Preds.Season AND R.WTeamID = Preds.Team1 AND R.LTeamID = Preds.Team2
WHERE R.DayNum >= 136
```

So Option B gives a lower binomial deviance and would therefore do better on the Stage 1 leaderboard for the Kaggle men's contest.

6. Find Recent Massey Ordinal Ranks

The Massey Ordinals table provides national rankings for each team across a number of different systems (Pomeroy, Sagarin, RPI, etc.) and a number of different dates, approximately every week. For any given game, the Massey rankings that are most applicable for predictions of that game would be the most recent rankings that occurred prior to when the game was played.

So for instance, if some rankings were released on Day 128, those rankings are based upon data from Day 0-127, and can be used to predict games from Day 128 and later. And if additional rankings were released on Day 133, then those would be even better to use for predicting games on Day 133.

Thus we might decide that for games played on Day 128/129/130/131/132, the appropriate rankings to use in predicting were those released on Day 128. Whereas the games played on Day 133+ (i.e. the NCAA Tournament games) should use the rankings that were released on Day 133.

This SQL query provides this information, and is used in subsequent sections:

```
CREATE VIEW RecentMasseyOrdinalRanks
  AS
SELECT CRU.Season, CRU.DayNum, CRU.TeamID, CRU.OppTeamID, OwnMO.SystemName, OwnMO.RankingDayNum,
OwnMO.OrdinalRank as OrdinalRank, OppMO.OrdinalRank as OppOrdinalRank
FROM (SELECT CRU.Season, CRU.DayNum, CRU.TeamID, CRU.OppTeamID, OwnMO.SystemName,
MAX(OwnMO.RankingDayNum) as LatestRankingDayNum
FROM CompactResultsUnion CRU
  JOIN MasseyOrdinals OwnMO ON CRU.Season = OwnMO.Season AND CRU.DayNum >= OwnMO.RankingDayNum AND
CRU.TeamID = OwnMO.TeamID
  JOIN MasseyOrdinals OppMO ON CRU.Season = OppMO.Season AND CRU.DayNum >= OppMO.RankingDayNum AND
CRU.OppTeamID = OppMO.TeamID AND OwnMO.RankingDayNum = OppMO.RankingDayNum AND OwnMO.SystemName =
OppMO.SystemName
GROUP BY CRU.Season, CRU.DayNum, CRU.TeamID, CRU.OppTeamID, OwnMO.SystemName) CRU
  JOIN MasseyOrdinals OwnMO ON CRU.Season = OwnMO.Season AND CRU.SystemName = OwnMO.SystemName AND
CRU.LatestRankingDayNum = OwnMO.RankingDayNum AND CRU.TeamID = OwnMO.TeamID
  JOIN MasseyOrdinals OppMO ON CRU.Season = OppMO.Season AND CRU.SystemName = OppMO.SystemName AND
CRU.LatestRankingDayNum = OppMO.RankingDayNum AND CRU.OppTeamID = OppMO.TeamID
GO
```

7. Seeing which Massey systems predicted the winner best

With the Massey Ordinals data, we have dozens of publicly-available ranking systems going back many years, and most of those systems published a pre-tournament ranking list right before each year's tournament. Thus for each system we can count up the number of times a better-ranked team beat a worse-ranked team in the NCAA tournament, and this helps us understand which ranking system does best at predicting.

For example, there are nine systems with 15 straight years of pre-tournament rankings like this, which means there were a total of 945 tournament games played among the final 64 teams (that's 63 times 15) that we have rankings for. We can count up how many times the better-ranked team (under each system) beat the worse-ranked team, and sort by the system that predicted best. Here is the query:

```
SELECT SystemName, SUM(CASE WHEN RMOR.OrdinalRank <= RMOR.OppOrdinalRank THEN 1 ELSE 0 END) as
NumCorrect
FROM RecentMasseyOrdinalRanks RMOR
  JOIN NCAATourneyCompactResults NTCR ON RMOR.Season = NTCR.Season
  AND RMOR.TeamID = NTCR.WTeamID AND RMOR.OppTeamID = NTCR.LTeamID
  AND RMOR.DayNum = NTCR.DayNum
WHERE RankingDayNum = 133 AND RMOR.DayNum >= 136
  AND RMOR.Season BETWEEN 2003 AND 2017
GROUP BY SystemName
HAVING COUNT(*) = 945
ORDER BY 2 DESC
```

And here were the results:

SystemName	NumCorrect	PctCorrect	
#1 POM	681	72.06%	(Pomeroy)
#2 SAG	679	71.85%	(Sagarin)
#3-5 MOR	675	71.43%	(Moore)
#3-5 RTH	675	71.43%	(Rothman)
#3-5 WLK	675	71.43%	(Whitlock)
#6 WOL	673	71.22%	(Wolfe)
#7 DOL	670	70.90%	(Dolphin)
#8 COL	667	70.58%	(Colley)
#9 RPI	662	70.05%	(RPI)

By going back fewer than 15 years, we can include more systems (the ones that started appearing more recently), but on the other hand it's fewer games being predicted, so it is a tradeoff. For instance, if we only go back 10 years, we include 18 systems but there are only 630 tournament games predicted. And if we only go back 5 years, we include 35 systems but there are only 315 tournament games predicted.

Also please note that the Kaggle scoring function does not just count up whether you predicted the winner correctly; you are asked to provide a winning percentage and your score is impacted more heavily by more confident predictions (positively impacted for aggressive predictions that were successful, and negatively impacted for aggressive predictions that were unsuccessful). So these calculations don't map directly to success in the Kaggle contest, but they can help to point the way to a successful approach.

8. Picking winner/loser, versus the contest evaluation function

We saw earlier, when checking the nine Massey Ordinal systems that have pre-tournament rankings going back 15 years, that the Pomeroy and Sagarin systems were #1 and #2 and RPI was #9, when we check who did the best at identifying the better team in each tournament game. This calculation is known as the Predictive Misclassification Rate. However, the Kaggle contest does not score directly by checking whether the stronger-ranked team won each game.

Instead, the Kaggle contest uses a "log-loss" scoring approach, where for each game that was played, a winning percentage was predicted, and then the average negative logarithm of each winning team's predicted percentage, are averaged together. The actual calculation is known as the Binomial Deviance.

This hopefully makes intuitive sense. If the Pomeroy system has Arizona ranked #18 and Oklahoma ranked #19, whereas the Sagarin system has Arizona ranked #15 and Oklahoma ranked #13, it is technically true that they disagree about which team is stronger. Nevertheless, in reality both systems would forecast approximately 50-50 winning chances for each team, despite their slight disagreement on who should be the favorite.

And conversely, if one system has Wichita State at #2, and the other has Wichita State at #16, and Wichita State loses to #20 Ohio State, then even though both systems would view this as an upset, it's a much worse prediction result for one system than the other. The log-loss approaches does a better job at distinguishing this type of situation, strongly penalizing aggressive predictions that turned out to be incorrect, and only weakly rewarding aggressive predictions that turned out to be correct.

However, this approach does require a predicted winning percentage for each game. The Massey Ordinals only provide a national ranking (#1, #2, …, up to #351) for each team. There are a number of possible ways to translate from national rankings of two teams into a predicted winning percentage. A straightforward approach, developed informally by Jeff Sonas, is to first convert from national ranking to a "power rating" for each team, providing the predicted margin of victory on a neutral court if you look at the difference between two teams' power ratings. And then you apply a second formula that converts from predicted margin of victory to predicted winning percentage. These are the two formulas:

PowerRating = 100.0 - 2.32*LN(Rank + 1) - Rank/25.3 - (Rank/205.0)^2
WinPct = 1.0 / (1.0 + {POWER(10.0, RatingDiff/12.0)))

There is no theoretical justification for the PowerRating formula; it's just a curve that seems to fit the data well. Obviously you could develop your own formula that might work better.

The SQL query to accomplish this, for the nine systems that are available going back 15 years, is the following:

```
SELECT SystemName
, SUM(CASE WHEN RMOR.OrdinalRank <= RMOR.OppOrdinalRank THEN 1 ELSE 0 END) as NumCorrect
, AVG(CASE WHEN RMOR.OrdinalRank <= RMOR.OppOrdinalRank THEN 1.0 ELSE 0.0 END) as PctCorrect
, AVG(-LOG((1.0000000 / (1.00000000 + POWER(10.00000000, (RMOR.OppPowerRating -
RMOR.PowerRating)/12.00))))) as BDev
, SUM(1) as NumGames
FROM (SELECT *
, 100.00 - 2.32*LOG(OrdinalRank + 1.00) - OrdinalRank/25.3 - (OrdinalRank/205.00)*(OrdinalRank/205.00)
as PowerRating
, 100.00 - 2.32*LOG(OppOrdinalRank + 1.00) - OppOrdinalRank/25.3 -
(OppOrdinalRank/205.00)*(OppOrdinalRank/205.00) as OppPowerRating
FROM RecentMasseyOrdinalRanks) RMOR
  JOIN NCAATourneyCompactResults NTCR ON RMOR.Season = NTCR.Season
  AND RMOR.TeamID = NTCR.WTeamID AND RMOR.OppTeamID = NTCR.LTeamID
  AND RMOR.DayNum = NTCR.DayNum
WHERE RankingDayNum = 133 AND RMOR.DayNum >= 136
  AND RMOR.Season BETWEEN 2003 AND 2017
GROUP BY SystemName
HAVING COUNT(*) = 945
ORDER BY 2 DESC
```

And it yields somewhat different performance across the nine systems:

SystemName	NumCorrect	PctCorrect	Binomial Deviance	
#1 SAG	679	71.85%	0.531905	(Sagarin)
#2 WLK	675	71.43%	0.534104	(Whitlock)
#3 POM	681	72.06%	0.537844	(Pomeroy)
#4 RTH	675	71.43%	0.539562	(Rothman)
#5 MOR	675	71.43%	0.540111	(Moore)
#6 DOL	670	70.90%	0.542924	(Dolphin)
#7 WOL	673	71.22%	0.546965	(Wolfe)
#8 COL	667	70.58%	0.547084	(Colley)
#9 RPI	662	70.05%	0.552366	(RPI)

By this measure, the Sagarin rankings performed best at predicting tournament results, across those 945 games.

Technical note about the above approach – we could certainly have just averaged together the ordinal ranks themselves, rather than first converting to a power rating and only then averaging the power ratings together. The justification for the two-step approach, rather than directly averaging the ordinal ranks, is that the strength gap between teams having adjacent ordinal ranks is not a constant. We will see larger gaps in strength, typically, between the #3 and #4 teams, than we will between the #100 and #101 teams. So for instance, a team that is ranked #3 in one system and #20 in another system, may be slightly better than a team that is ranked #11 in one system and #12 in another system.

9. Benchmarking against the Stage 1 Leaderboard

We could also do this same calculation just across the previous four years (the tournaments from 2014-2017), which only covers 252 tournament games but does include a lot more systems from the Massey Ordinals data. This is exactly the set of 252 (men's) games that are scored in Stage 1 of the contest.

```
SELECT SystemName
, SUM(CASE WHEN RMOR.OrdinalRank <= RMOR.OppOrdinalRank THEN 1 ELSE 0 END) as NumCorrect
, AVG(CASE WHEN RMOR.OrdinalRank <= RMOR.OppOrdinalRank THEN 1.0 ELSE 0.0 END) as PctCorrect
, AVG(-LOG((1.0000000 / (1.00000000 + POWER(10.00000000, (RMOR.OppPowerRating -
RMOR.PowerRating)/12.00))))) as BDev
, SUM(1) as NumGames
FROM (SELECT *
, 100.00 - 2.32*LOG(OrdinalRank + 1.00) - OrdinalRank/25.3 - (OrdinalRank/205.00)*(OrdinalRank/205.00)
as PowerRating
, 100.00 - 2.32*LOG(OppOrdinalRank + 1.00) - OppOrdinalRank/25.3 -
(OppOrdinalRank/205.00)*(OppOrdinalRank/205.00) as OppPowerRating
FROM RecentMasseyOrdinalRanks) RMOR
  JOIN NCAATourneyCompactResults NTCR ON RMOR.Season = NTCR.Season
  AND RMOR.TeamID = NTCR.WTeamID AND RMOR.OppTeamID = NTCR.LTeamID
  AND RMOR.DayNum = NTCR.DayNum
WHERE RankingDayNum = 133 AND RMOR.DayNum >= 136
  AND RMOR.Season BETWEEN 2014 AND 2017
GROUP BY SystemName
HAVING COUNT(*) = 252
ORDER BY 4
```

There were 39 Massey systems that had pre-tournament rankings for all four years, allowing 252 game predictions, with the top five performing like this:

SystemName	NumCorrect	PctCorrect	Binomial Deviance	
#1 DOK	181	71.83%	0.515764	(Dokter Entropy)
#2 TRP	184	73.02%	0.518654	(TeamRankings Pred)
#3 SAG	185	73.41%	0.521680	(Sagarin)
#4 CNG	184	73.02%	0.523743	(Cheong)
#5 SP	181	71.83%	0.525714	(Sagarin Predictor)

And the bottom five looked like this:

SystemName	NumCorrect	PctCorrect	Binomial Deviance	
#35 WOL	179	71.03%	0.560209	(Wolfe)
#36 REW	183	72.62%	0.561600	(Rewards)
#37 RPI	178	70.63%	0.561660	(RPI)
#38 SPW	175	69.44%	0.578119	(Snapper's World)
#39 NOL	170	67.46%	0.589240	(Nolan)

You can use these calculations to get a sense of how your own systems are performing on the Stage 1 Leaderboard. You may recall that the "2% Seed Benchmark" scored 0.585056, and the "3% Seed Benchmark" scored 0.559008.

Note that RPI (which doesn't use margin of victory and has a relatively simple formula) is #37 out of 39 on this list, with a score of 0.561660. Anything that scores worse than 0.56 in Stage 1 is probably not going to predict very well in the real tournament when we reach Stage 2. Conversely, anything that scores better than 0.50 in Stage 1 is very likely suffering from a methodological problem of "leakage", where the actual tournament results themselves have been used to inform the predictions. Such an approach might look superior when scored against Stage 1, but will not work well in Stage 2. Any predictions should only make use of the available information at the time, and should not benefit from information leaked "from the future", such as the results of subsequent games. For example, you should not use "end of the year" ratings (that incorporate tournament game outcomes) to turn around and predict the results of those very tournament games.