

## Many to Many Relationships

Suppose you need to design a set of relational tables to track the pitcher statistics for Arizona Diamondbacks. Each game can use many pitchers and each pitcher can pitch in many games. The table below presents the data organized by *game*.

GameNum	GameDate	Opponent	Uniform#	Name	Sequence	Innings	Hits	Runs	Walks	Strikeouts	Pitches
168	10/21/2001	Braves	51	Johnson	1	7.0	7	2	2	8	118
			49	Kim	2	2.0	0	0	1	2	27
169	10/27/2001	Yankees	38	Schilling	1	7.0	3	1	1	8	102
			36	Morgan	2	1.0	0	0	0	0	10
			22	Swindell	3	1.0	0	0	1	1	14
170	10/28/2001	Yankees	51	Johnson	1	9.0	3	0	1	11	111
175	11/4/2001	Yankees	38	Schilling	1	7.1	6	2	0	9	103
			43	Batista	2	0.1	0	0	0	0	1
			51	Johnson	3	1.1	0	0	0	0	1

Recall that each field in a table must be *atomic* (i.e., it must store only a *single* value). Each of the right-most nine columns in the table above violate the atomic requirement since each game can have many pitchers!

Alternatively, you could organize the data by *pitcher*, as illustrated below.

PitcherID	UniformNum	Name	Game	Date	Opponent	Sequence	Innings	Hits	Runs	Walks	Strikeouts	Pitches
1	51	Johnson	168	10/21/2001	Braves	1	7.0	7	2	2	8	112
			170	10/28/2001	Yankees	1	9.0	3	0	1	11	111
			175	11/4/2001	Yankees	3	1.1	0	0	0	1	17
2	49	Kim	168	10/21/2001	Braves	2	2.0	0	0	1	2	27
3	38	Schilling	169	10/27/2001	Yankees	1	7.0	3	1	1	8	102
			175	11/4/2001	Yankees	1	7.1	6	2	0	9	103
4	36	Morgan	169	10/27/2001	Yankees	2	1.0	0	0	0	0	10
5	22	Swindell	169	10/27/2001	Yankees	3	1.0	0	0	1	1	14
6	43	Batista	175	11/4/2001	Yankees	2	0.1	0	0	0	0	1

Notice that a new field named *PitcherID* was created to serve as the table's primary key (indicated by the column's shading). Over the course of several seasons, uniform numbers wouldn't necessarily be unique or static so *UniformNum* would've made a poor choice for a primary key. Notice that this table also violates the atomic requirement since the right-most nine columns each store multiple values since each pitcher can pitch in many games. Thus, this table isn't any better than the previous one!

The solution is to have separate tables for games and for pitchers, and to use a *junction table* to break the *Many-to-Many* relationship into two separate *One-to-Many* relationships. Note that the new table uses a *composite primary key* that consists of the primary key of each of the other tables.

Game		
GameNum	GameDate	Opponent
168	10/21/2001	Braves
169	10/27/2001	Yankees
170	10/28/2001	Yankees
175	11/4/2001	Yankees

Pitcher		
PitcherID	Name	UniformNum
1	Johnson	51
2	Kim	49
3	Schilling	38
4	Morgan	36
5	Swindell	22
6	Batista	43

GamePitcher								
GameNum	PitcherID	Sequence	Innings	Hits	Runs	Walks	Strikeouts	Pitches
168	1	1	7.0	7	2	2	8	118
168	2	2	2.0	0	0	1	2	27
169	3	1	7.0	3	1	1	8	102
169	4	2	1.0	0	0	0	0	10
169	5	3	1.0	0	0	1	1	14
170	1	1	9.0	3	0	1	11	111
175	3	1	7.1	6	2	0	9	103
175	6	2	0.1	0	0	0	0	1
175	1	3	1.1	0	0	0	1	17

## Summary

*Many-to-Many* relationships are engineered by creating a *junction table* to break the *Many-to-Many* relationship into two separate *One-to-Many* relationships. Doing so fulfills the atomic requirement and avoids data redundancy.

## Shorthand Notation and E-R Diagram

**Game**(GameNum, GameDate, Opponent)

**Pitcher**(PitcherID, Name, UniformNum)

**GamePitcher**(GameNum, PitcherID, Sequence, Innings, Hits, Runs, Walks, Strikeouts, Pitches)

Foreign key: GameNum to Game relation

Foreign key: PitcherID to Pitcher relation

