

# BASEBALL HACKS™

*Tips & Tools for Dissecting and  
Analyzing Statistics*



O'REILLY®

*Joseph Adler*

HACK  
#66

## Measure Hits by Pitch Count

Determine whether pitchers are more effective earlier in games than they are later.

In Game 7 of the 2003 American League Championship Series, Pedro Martinez was dominant. In the first seven innings of the game, Pedro allowed the Yankees to score only two runs. The Red Sox were leading the game, 5 to 2. Pedro got the first batter (Nick Johnson) to pop out to short, but then he allowed Derek Jeter to hit a one-out double. Red Sox fans knew that Pedro became less effective after pitching his first 80 pitches, and, at this point in the game, he was well over that mark. Red Sox manager Grady Little decided to stick with his ace. Bernie Williams hit a single to center, scoring Jeter. Matsui then hit a double, followed by a double from Posada to tie the game.

The rest is history. The game went into extra innings and ended with a walk-off home run by Aaron Boone; the Yankees went on to lose the World Series, and Grady Little lost his job.

Part of the “conventional wisdom” of baseball is that pitchers become less effective as the game progresses. This is partly because pitchers tire out and can’t throw the ball as hard (or can’t follow through as well on their pitches), and partly because batters learn to recognize what the pitcher is throwing. But is this true?

This hack shows you how to use play-by-play data to examine what happens to starting pitchers as a game progresses.

### The Code

To show how a pitcher performs as a game progresses, you need to calculate a cumulative pitch count. The first step in calculating a pitch count is to filter the pitch sequence strings, to remove pickoff throws, stolen bases, and other things that aren’t really pitches. You can do this in SQL using a procedure like this:

```
create view fullpbp2003_w_npitches as
select game_id, res_pitcher, batting_team, event_type, outs,
       event_num, event_text, pitch_sequence,
       length(
         replace(
           replace(
             replace(
               replace(
                 replace(
                   pitch_sequence,
```

```

        '>',''), -- runner going on pitch
        '.'),''), -- play not involving the batter
        '1',''), -- throw to first
        '2',''), -- throw to second
        '3',''), -- throw to third
        'N',''), -- no pitch (on balks and interference calls)
        'V','') -- called ball because pitcher went to mouth
    ) as npitches
from pbp.pbp2k
where substring(game_id,4,4)="2003");

```

The second step is to calculate the cumulative pitch count. We need to make sure not to calculate duplicate pitches, so we include only records that either end the inning or end a plate appearance. There is a trick for doing this in SQL: a self-join. You join a table with itself and add a *where* clause to restrict the set of records with which you join any given record. Here's an example of how to do this:

```

create index fullpbp2003_idx on fullpbp2003(game_id, batting_team,
    event_num, res_pitcher);
select l.game_id, l.res_pitcher, l.batting_team,
    sum(r.npitches) as cum_pitches
from fullpbp2003_w_npitches l inner join fullpbp2003_w_npitches r
on l.game_id=r.game_id and l.res_pitcher=r.res_pitcher and l.batting_team=r.
batting_team
where r.event_num <= l.event_num and
    (-- caught stealing or pickoff with two outs
    (r.event_type IN (6, 8) and r.outs=2) or
    -- ball in play
    r.event_type IN (2,3,14,15,17,18,19,20,21,22,23))
group by l.game_id, l.res_pitcher, l.batting_team;

```

Unfortunately, this method takes a very, very long time to run. (I stopped this query after it ran for 10 minutes.) All of that self-joining takes time, and MySQL isn't very smart about using indexes to optimize operations like this. A faster way to calculate cumulative pitch counts is to use a procedural language. We know that the play-by-play tables are sorted by event number, so we can count just the number of pitches as we read through the table.

To solve this problem, let's use a Perl script. This script reads all of the plays in 2003, determines the starting pitchers, and saves the results in hashes by game and event number. It then calculates the cumulative pitch counts while writing the output results to standard output.

```

use DBI;
use DBD::mysql;

my $dsn = "DBI:mysql:host=localhost;database=pbp";

```

```

my $dbh = DBI->connect($dsn, "jadler", "", {PrintError => 0, RaiseError =>
1});

my $query = "SELECT * FROM fullpbp2003_w_npitches";

my $sth = $dbh->prepare($query);
$sth->execute();
while (my $ref = $sth->fetchrow_hashref ()) {
    $game_id = $ref->{game_id};
    $batting_team = $ref->{batting_team};
    $event_num = $ref->{event_num};
    if (!defined($starting_pitchers{$game_id}[$batting_team])) {
        $starting_pitchers{$game_id}[$batting_team] = $ref->{res_pitcher};
    }
    if ($ref->{res_pitcher} eq $starting_pitchers{$game_id}[$batting_team])
    {
        $pitch_counts{$game_id}{$event_num} = $ref->{npitches};
        $outcomes{$game_id}{$event_num} = $ref->{event_type};
        $batting_teams{$game_id}{$event_num} = $ref->{batting_team};
        $events{$game_id}{$event_num} = $ref->{event_text};
    }
}

print "gameid,starter,pitch_count,event_type,pitches,event_text\n";
foreach $game_id (keys %pitch_counts) {
    $cumulative_pitch_count[0] = 0;
    $cumulative_pitch_count[1] = 0;
    $start[0] = $starting_pitchers{$game_id}[0];
    $start[1] = $starting_pitchers{$game_id}[1];
    foreach $event_num (sort keys %{$pitch_counts{$game_id}}) {
        $batting_team = $batting_teams{$game_id}{$event_num};
        $cumulative_pitch_count[$batting_team]
            = $cumulative_pitch_count[$batting_team] +
              $pitch_counts{$game_id}{$event_num};
        print "$game_id,"
            , $start[$batting_team], ","
            , $cumulative_pitch_count[$batting_team], ","
            , $outcomes{$game_id}{$event_num}, ","
            , $pitch_counts{$game_id}{$event_num}, ","
            , $events{$game_id}{$event_num}, "\n";
    }
}

```

This script outputs a file with a game ID, a player ID, a pitch count at the start of the at bat, the number of pitches during the at bat, an event type, and a description of the event.

## Running the Hack

Save this script in a file called *hits\_by\_pitchcount\_2003.pl* and run it with the following command:

```
% perl hits_by_pitchcount_2003.pl > pitchcounts2003.csv
```

(Incidentally, this script took 12 seconds to run on my iMac G5.) This script output 127,204 lines, which is unfortunate because it means this file is too big to analyze in Excel. To deal with this data, we'll need to use another tool to work with it. Well, at least, we'll need to use a tool to do a little preprocessing so that we can use Excel later because it's going to be easiest to work with the results if we can get them into a pivot table.

Let's start by pulling this data back into MySQL. The easiest way to do this is to use the script from "Load Baseball Data into MySQL" [Hack #20] to generate the import code. You can generate the import code and load the file with the following commands:

```
% perl check_field_sizes.pl -h -s "," -i pitchcounts2003.csv >
pitchcounts2003.sql
% mysql pbp < pitchcounts2003.sql
```

Let's shrink this table by summarizing how each pitcher performed, depending on the pitch count at the start of the at bat. We'll break these pitch counts into bands of 10 (0–9, 10–19, 20–29, etc.) using the `floor` function, and then we'll count the number of hits, walks, and plate appearances.

```
create index pitchcounts2003_idx on pitchcounts2003(starter);

create temporary table names
as select nameLast, nameFirst, pbpID
from master_w_pbpID;
create index names_idx on names(pbpID);

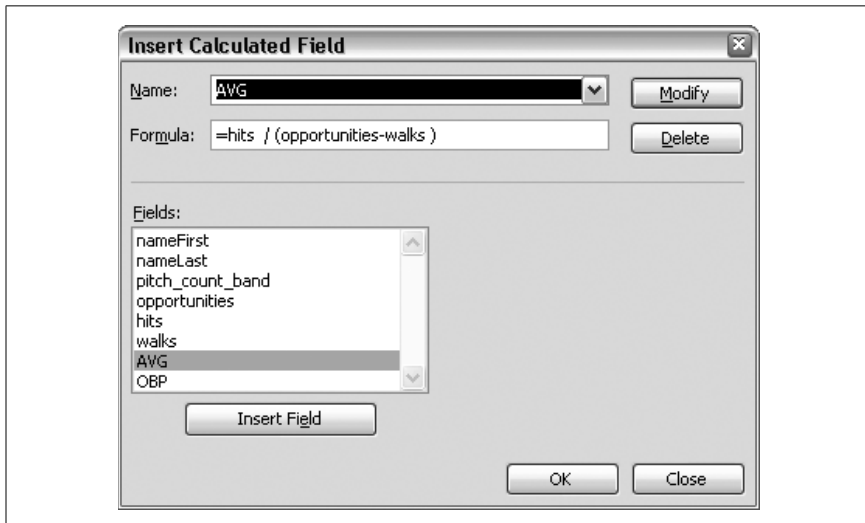
select nameFirst, nameLast,
       floor(pitch_count / 10) * 10 as pitch_count_band,
       count(*) as opportunities,
       sum(if(event_type IN (20,21,22,23), 1, 0)) as hits,
       sum(if(event_type IN (14,15,16), 1, 0)) as walks
from pitchcounts2003 l inner join master_w_pbpID r
on l.starter=r.pbpID
group by nameFirst, nameLast, floor(pitch_count / 10) * 10
;
```

This new table has 3,538 rows of cuts, perfect for pivot tables in Excel. You can run the query in your database query tool (such as your MySQL Query tool), copy the results of the query, and paste them into Excel. To make the analysis easy, select the whole table in Excel and then select PivotTable Report... from the Data menu. Click Finish to end the wizard (just accept

the default values). Next, drag `pitch_count_band` to the “drag rows here” section of the pivot table, and drag `opportunities` onto the “drag data fields here” section of the pivot table.

To make this easier to read, let’s add a couple of calculated fields to this pivot table. Calculated fields are a great feature because they allow you to calculate values, such as averages, dynamically, based upon the counts in the table. (You don’t get the right result if you just calculate averages in MySQL. Using calculated fields gives you the properly weighted answer.)

To add calculated fields, select *Calculated Field...* from the *Formulas* sub-menu of the *PivotTable* menu on the *Pivot Table* palette. An illustration of this dialog box is shown in *Figure 6-6*. (Alternately, you can right-click on the pivot table to reach this dialog box.) Enter an expression on the *Formula* line and a name for the field in the *Name* box, and click the *Insert Field* button to add the field. Click *OK* when finished.



*Figure 6-6. Insert Calculated Field dialog box*

Let’s create two fields: an approximate batting average (hits divided by opportunities minus walks) and OBP (hits plus walks divided by opportunities). If you like, you can change the number format of these fields through the *Field Settings* menu option in the *PivotTable* menu, and change the number format.

Now that we have the data in a pivot table, let’s calculate a few neat things. First, let’s break this down by pitcher. Drag the `nameLast` and `nameFirst`

Measure Hits by Pitch Count

fields onto the “drag page fields here” section of the pivot table. You can now select results for specific pitchers through these menus. Let’s pick Pedro Martinez, to see how his performance changes by pitch count. Figure 6-7 shows Pedro’s performance by pitch count.

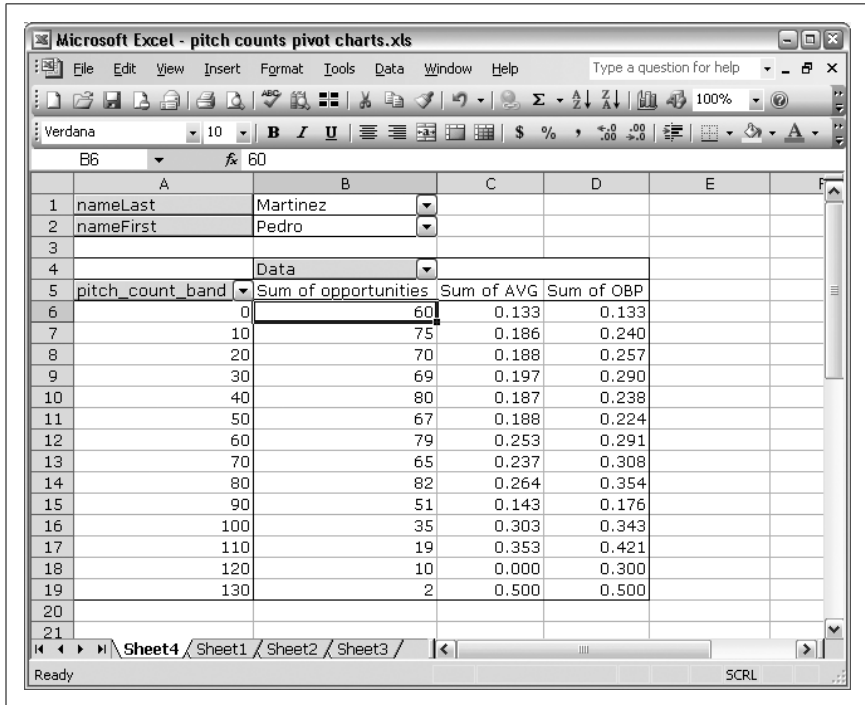


Figure 6-7. Pivot table of Pedro’s performance

As you can see, his performance early in the game was phenomenal: opponents averaged less than .200 against him when his pitch count was less than 60 (pitch\_count\_bands of 0 through 50), but he began to slip as his pitch count increased. There are a couple of exceptions: pitches 90–99 and 120–129. We can explain these partly through statistical significance and partly by noting that Pedro was allowed to pitch deep into games only if he was pitching phenomenally well.

Let’s add a chart to make this table easier to read. First, open the Pivot Table Options dialog box (either by right-clicking on the table or from the Pivot-Table menu) and remove the Grand Total fields (they clutter the charts). Next, select the whole table and click the Chart button to open the Chart wizard. (To select the whole table, try clicking cell A4.) Select a line chart and click Finish to close the wizard and view the chart.

You'll notice a problem with the chart: you can't read the averages (which go up to only .50) and the opportunities (which go up to 82) on the same chart. To fix this problem, we'll move the opportunities to a second axis. Select the opportunities line by clicking on it, and select "format data series" by right clicking on the line. Go to the Axis pan and select Secondary Axis. This will make the graph readable. See Figure 6-8 for an example chart (I changed the formatting so that it would look OK in print). As you can see, opponents did have an easier time hitting off Pedro in later innings.

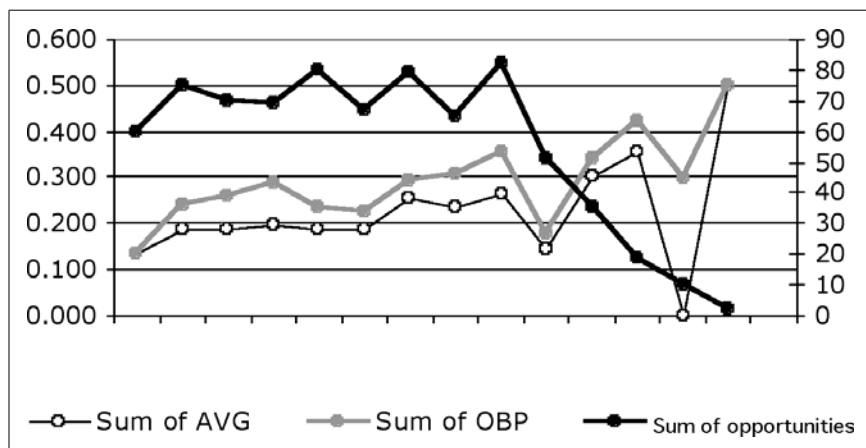


Figure 6-8. Chart showing Pedro's performance by pitch

## Hacking the Hack

Is Pedro's performance typical? To find out, let's show the same chart for all pitchers. You can just deselect the `nameFirst` and `nameLast` values, and the chart will change to reflect this. You should see a chart such as the one shown in Figure 6-9. Somewhat surprisingly, pitcher performance seems flat; opponent averages stay the same (or even decrease) as the number of innings pitched increases. Is this effect real, or did we do something wrong?

One possibility is that these results are being distorted because pitchers are pulled from games as they begin to give up hits, so our figures primarily show pitchers who are performing well. A way to test this hypothesis is to group performances by the maximum number of pitches thrown, not by name.

To figure this out, let's start by creating a temporary table containing the total number of pitches thrown by each starter in each game:

```
create temporary table total_pitches as
select gameid, starter, max(pitch_count + pitches) as totalpitches
from pitchcounts2003 group by gameid, starter;
```

Measure Hits by Pitch Count

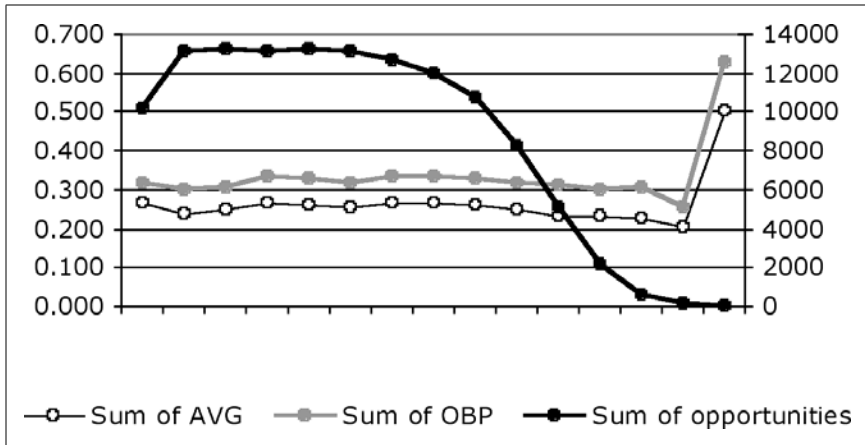


Figure 6-9. Performance by pitch for all pitchers

Then, we'll create two indexes to make the merge fast:

```
create index total_pitches_idx on total_pitches(gameid, starter);
create index pitchcounts2003_gmidx on pitchcounts2003(gameid, starter);
```

Finally, we'll break down performance by pitch count at each point in the game and by total number of pitches thrown:

```
select
    floor(pitch_count / 10) * 10 as pitch_count_band,
    floor(totalpitches / 20) * 20 as total_pitches_band,
    count(*) as opportunities,
    sum(if(event_type IN (20,21,22,23), 1, 0)) as hits,
    sum(if(event_type IN (14,15,16), 1, 0)) as walks
from pitchcounts2003 l inner join total_pitches r
on l.starter=r.starter and l.gameid=r.gameid
group by floor(pitch_count / 10) * 10, floor(totalpitches / 20) * 20
;
```

Run this code in your SQL Query client and then copy the results and paste them into Excel. Select the data and create a pivot table, then create a new calculated field for OBP. Drag `pitch_count_band` to the “drop columns here” part of the pivot table, and `total_pitches_band` to the “drop rows here” part of the pivot table. You should have a table of approximately 15×15 rows, where the lower-left corner is filled with #DIV/0! values. Very few pitchers got beyond 140 pitches in a game, and they'll make the resulting chart unreadable, so you might want to change the field settings for `total_pitches_band` to hide any games with fewer than 20 or more than 140 pitches thrown. Plot this chart as explained earlier. You should see a chart similar to the one shown in Figure 6-10.

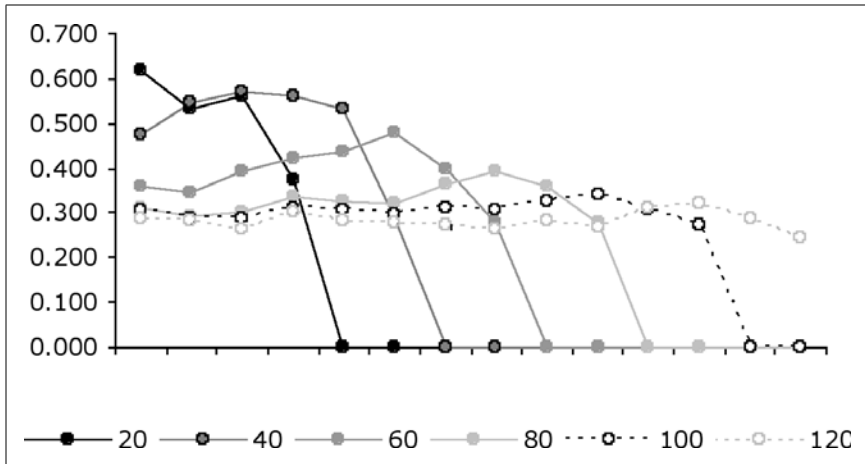


Figure 6-10. Batting average by endurance

Not surprisingly, pitchers who exit games very early have higher on-base percentages. (They are probably taken out of games early because of their ineffectiveness.) But surprisingly, pitchers don't actually seem to perform much worse as their pitch counts increase. Some pitchers get a little less effective (take a look at the 60–79 pitch band) as the game progresses, but it's not a dramatic difference, and it does not appear to be typical.