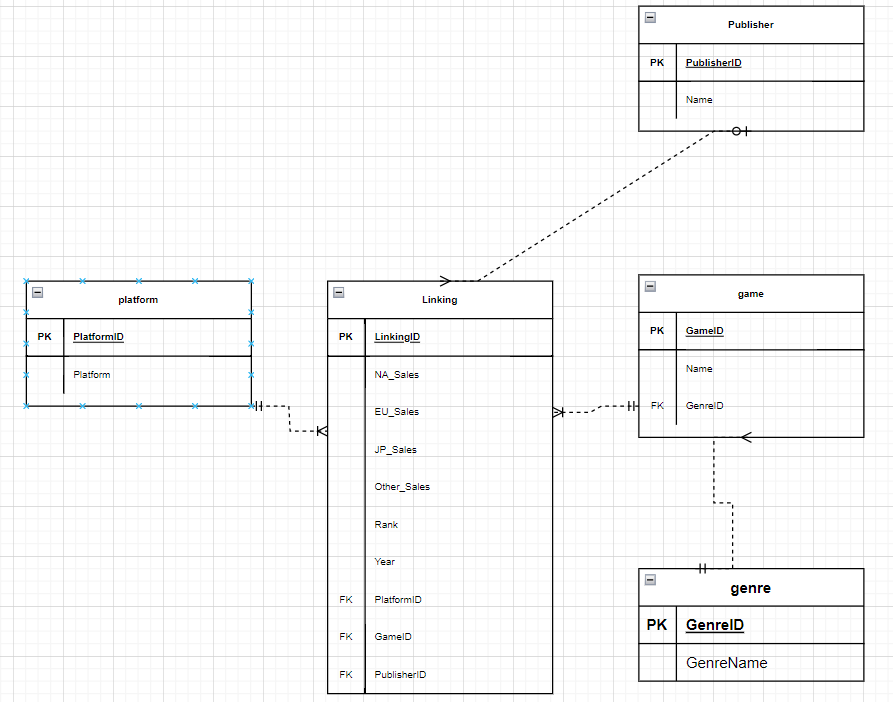
Assignment 2 Report – Database Organization and Management 1

Authors: Sean Wendlandt and Matt Krzewinski

11/13/22



**Crows Foot Diagram:**

While creating our database design we had to do several renditions of the crow’s foot before we decided on the final model. This was largely due to the discovery of fields we initially thought were unique that we discovered were not as we further explored the data and migrated data. Once we fully understood the data and which attributes belonged to each table we decided we needed a large linking table as there were three attributes that were M-M. The three attributes that were M-M were Game (name), Platform, and Publisher. We decided these were M-M due to Game (name) having multiple platforms and multiple publishers. With that same logic, we decided both platform and game are also their own M-M table as they could both have many games associated with them.

In the linking table between these three tables, we included Rank, Year, the surrogate FKs (a surrogate key for each M-M table), and the sales attributes. It is worth noting that we decided to drop global sales from the attributes as it is a calculated field and thus should not be included in our opinion. If we were to include it, and some of the other sales columns were changed, there would need to be a trigger made which updated the global sales column. It seemed like a much better idea to just drop the column and if global sales is needed, just do a calculation. This was by far the largest of all of our tables containing the most attributes, which resulted in some migration struggles which are mentioned in the section below.

The final table we created, which was also the only 1-M relationship in the crow’s foot, was between Game (name) and Genre. This was due to each game only having one genre associated with it in the table, and one genre being associated with many games. This relationship did not change depending on the publisher or platform, so it was best to include a foreign key in our game table. Because of it being a 1-M relationship, we assumed this field only focused on the primary genre of a game. If the data/client showed that there were multiple genres for each game, the database would need to be redesigned.

As we made this design, some of the largest changes came from a further understanding of the attributes of the initial csv table. Specifically, we thought rank was a potential unique attribute that could serve as a key in the linking table. However, with further understanding of the metric, we learned that the ranking is solely based on the sales each game obtained from their respective platform. Due to this, as new data and information would come in, the ranking would change for each game and cause the database to have a lot of issues. We further noted that there weren’t any given fields that could serve as primary keys for our tables, because of this fact, we had to create surrogate keys for each table we created. While we could have created a composite key in the linking table of the game, platform, and publisher id, it seemed easier to just use these as foreign keys for joins.

**Migration and other Issues:**

There were a few issues that came up when trying to migrate the data, primarily with our linking table. While rank and the sales attributes were able to be inserted with a standard select, year was causing issues. First off, a date attribute in mysql requires the format “YYYY-MM-DD,” and the year column in vg\_csv was just the year, so we had to concatenate the year with “01-01” to be a valid date entry. Secondly, there were some “N/A” text fields in the vg\_csv table so these were not able to be converted to date fields, so we needed to use “CASE WHEN year = 'N/A' THEN NULL ELSE CONCAT(year, '-01-01') END” to be able to handle these and the formatting.

The second issue we encountered was inserting the foreign keys in the linking table during the bulk insert. While we believed our nested query was correct, our insert would time out and lose connection to the database. However, after some discussions with the professor, we noticed that we needed to increase the time before the query timed out, because the bulk insert and matching the foreign keys took much more than 30 seconds, just under 200 seconds in most runs.