The fields that can be stored for each content item are the GUID, category, title, description, publish date, link, slug, networks, state, tags, and compact, medium, and large thumbnail links. The link will not need to be stored because it can be generated in the application layer knowing its base URL (http://www.ign.com/), category, and slug. For a similar reason, the ‘http://’ at the beginning of the thumbnail link and the ‘\_compact.jpg’, ’\_medium.jpg’, or ‘\_large.jpg’ at the end of the thumbnail link will be removed when storing the thumbnail link. This will also remove the need to store three thumbnail links. The GUID will be stored as a 24-byte character, the publish date will be stored as a datetime, and the rest of the fields will be stored as variable characters.

Initially, the database contains one table (Items) with the ID, GUID, category, title, description, publish date, slug, networks, state, tags, and thumbnail link fields. This new ID field will be explained in the next paragraph. To bring this database to the first normal form, fields that can have multiple values must be broken down into their atomic values. The fields that can have multiple values are the networks (assumed because of its plural form) and tags fields. Removing these two fields from the table and creating two new tables, one with the ID and networks fields (ItemNetworks) and the other with the ID and tags fields (ItemTags), will bring the database to the first normal form. The database is already in the second and third normal forms because all non-key fields are directly dependent on the ID field, the primary key.

Using the GUID field as the primary key has a considerable performance impact on the database because a clustered index is automatically created. GUIDs do not receive much benefit from indexing as they are generated randomly, not sequentially. To fix this problem, an ID field will be created. This field will be an auto-increment integer and serve as the primary key instead of the GUID field. Assuming that the database will update as new content is pushed, a non-clustered index is still needed on the GUID field to guarantee that each new record is unique. A practical use for this database is to search and return data from all fields of a record based on date or tags. This means a non-clustered index is needed on the publish date and tags fields.

Pulling records based on tags from the ‘ItemTags‘ table will only return the tags and the ID, which is used to retrieve data from the other fields in the Items table. On the flip side, pulling records based on date from the Items table will return all the fields except for tags and networks, which will also need the ID to be retrieved from the ‘ItemTags‘ and ‘ItemNetworks‘ tables. Because there are more queries with ‘WHERE‘ clauses that involve the ID field, a clustered index is created on the ID field instead of the date or tags fields. A non-clustered index is also created on the ID fields in the ‘ItemTags‘ and ‘ItemNetworks‘ because they are foreign keys referring to the primary key in the ‘Items‘ table.

In my service, each page of the RSS feed is iterated from the first page, and each content item is stored in the database. This will continue until either a content item with an existing GUID in the database or the end of the RSS feed is reached.