Natural versus Surrogate KEYS – The “Key” to Success in Database Modernization

For most developers involved in high volume commercial OLTP software technology, the concept of using a key to access a record in a database table rings familiar bells. Several different keys made up of various data element combinations provide different ways to access the underlying data, with one of those keys designated as the table’s primary key. The design of what fields to use as the primary key and how to construct it usually falls to database and software architects (we will use the term DBE {Database Engineer} henceforth). In recent years, new theories in this area have resulted in significant debate over the use of natural keys (actual business data elements) versus surrogate keys (artificially generated unique identification fields with absolutely no business meaning).

Each type of database key possesses certain advantages and disadvantages, so neither provides a perfect solution. Extreme zealots exist on both sides of the argument, advocating exclusive use of one or the other. A more objective view suggests both have a place, depending on usage circumstances.

This discussion does not intend to cover the pros and cons of each exhaustively; many well-written articles already cover that ground (see some citations at the end of this document for additional consideration). Instead, we want to examine the use of each type of key in conjunction with a database modernization project that likely involves restructuring and normalizing existing tables. In many cases, this introduces new tables (PF’s), new keys, and new access plans (read logical constructs = LF’s, Views, Indexes); the construction of those keys can influence the timeline and effort required for modernization.

As mentioned, a natural key uses one or more actual business data elements in a table to provide a unique key: a customer number, an order number, a product description, or the combination of a person’s first and last name.

A surrogate key, on the other hand, provides a unique way to distinguish records separate and distinct from the business data: a sequence number, a random number with ensured uniqueness, or one of several algorithmically generated identifiers. Many relational database management systems (RDBMS) provide a feature for defining a column in a table as an “identity” element where the system automatically generates a value for each new record, providing a simple method to create surrogate keys. In DB2 for the IBM i, the field is designated using the keywords “GENERATED ALWAYS AS IDENTITY.”

Applications generally hide surrogate keys from end-users, even though they use them to establish relationships within the database. While some natural keys seem to fit the surrogate description of artificially generated unique identification fields (such as customer number, order number, etc.), these differ in that they represent real data presented to end-users.

The primary key for a given table often becomes a field in other tables, referred to as a foreign key. In some cases, a foreign key may even comprise part of the primary key in another table. The relationships between tables provide much of the advantage and power of databases (as opposed to data-store in flat files) but also introduces complexities.

Natural keys offer the advantage of easy comprehension by the business users because they hold real meaning. However, this also opens them to significant pitfalls if the business meaning or the data itself changes. A typical example involves using a person’s name as a key; if the person changes

Document Version: 2020-1
Publication Date: 2020/02/27

their name (for example, when their marital status changes), all instances of the key in all tables where the key exists require updates to maintain all the proper records linkages between all the tables.

Surrogate keys do not suffer the same issue. If a record with a surrogate identifier of “56” represents “Mary Smith,” changing her name to “Mary Jones” involves changing the single table containing the actual name value. All foreign key references in other tables use “56”, which remains unchanged. Surrogate keys may also simplify situations where the natural key for a table would require multiple fields to ensure uniqueness; the surrogate suffices with a single field.

However, in many circumstances, surrogate keys hide otherwise inherent understanding. Customer numbers carry meaning in many companies. An order record containing a customer number may provide a seasoned user instant information of who placed the order since they know many of the numbers by heart. However, if the customer file uses a surrogate key instead of the customer number, then the foreign key in the order record provides no information without also accessing the customer file to determine the name of the customer.

The above barely scratches the surface of the arguments for one or the other. Many proponents of surrogate keys maintain the ability to update business data at will without affecting foreign keys renders surrogate keys superior. Others counter that the rise of the business analyst role and the increase in “ad hoc” queries and reporting to provide business leaders more significant insight into business operations makes the use of natural keys essential to maintain the business meaning of the keys. Again, many well-written and extensive comparisons exist (see a short introductory list of links at the end of this document).

We wish to focus on the impact of each key type, specifically on database modernization. Modernizing the database underlying a 20 to 30-year-old (or older) application generally involves changes to the structure of the tables making up the database and the data stored in each table. Often these older databases grew “organically” over time and contain redundant elements repeated in numerous tables. A modernization effort highly likely includes “normalizing” this data, isolating these elements into one discrete table, so the data exists in only one place, and other tables reference the data through a foreign key relationship with the primary table.

A modernization project must also consider the software development cycle, and here the type of key used for a given file (and potentially the method used to generate the key) begins to impact the project. Realistically, such projects take considerable time, so integrating them into the typical development cycle often provides the most significant impact for the least cost (ROI). Programmers implement the table changes gradually over time as part of other enhancement efforts (see our suggested “***Optimal Authentic Modernization of Heritage IBM i Software Assets***” document). As such, multiple application environments likely exist; -one for current development, one or more for quality assurance testing, and the primary production environment.

Because each environment is discrete, the same natural keys can typically exist in each environment without concern. Customer number 24680 “Joe’s Plumbing” may exist in the customer table in all the environments. Actual production data copied into the test environments (sanitized for privacy where necessary) often reveals oversights and flaws more readily than artificially constructed test data. Identical surrogate keys can also exist in various environments, depending on the generation method. For example, if the process of normalizing a product table leads to the creation of a separate table of available colors, the DBE (Database Engineer) may choose to use a simple sequential number surrogate key for the latter table. This way, the business can easily update the color descriptions without impact through a simple insert/update/delete (BREAD or CRUD) interface

Document Version: 2020-1
Publication Date: 2020/02/27

that internally handles maintenance and uniqueness of the sequence number. After verification in the testing environments, "promoting" the color table to production may involve copying the new table (or at least the data therein after creating the new table structure) and using relatively straightforward SQL scripts to update existing affected tables.

Generating surrogate keys through an identity column introduces additional complexities to the above scenario. External processes cannot write to identity columns. Because of this, the surrogate keys for a given table may not match between the development, test, and production environments. This makes copying production data to the test environments more delicate and can complicate the promotion process as well. Loading data from the existing database system into updated versions of current tables becomes more difficult and may require extra scripts to validate or update existing foreign keys.

Many other RDBMS have facilities to address this latter shortcoming, providing capabilities to temporarily suspend the identity aspect of the column to allow the system to write specific values. Unfortunately, DB2 for IBM i only allows permanent removal of the identity attribute from the column.

Choosing the correct type of primary key for any given table requires careful thought and analysis. A well-chosen natural key remains a reliable choice, based on the business meaning it carries for ad-hoc analysis. Choosing data elements unlikely to change under normal circumstances mitigates the concern over using actual business data. Likewise, for ancillary data tables meant to encapsulate individual aspects (such as the color table example), a simple surrogate key makes an excellent choice for isolating data possibly subject to change. Both types of keys definitely have their uses.

Due to the nature of commercial OLTP applications, the primary users thereof, and especially as a result of massive developments in the areas of data science and cognitive computing, we believe that a well-chosen natural key will likely facilitate improved end-user experience. This is especially true with the gluttony of extraction and analysis tools at the disposal of most end-users, who demand to be able to "slice and dice" transactional data in any way they deem fit.

The preceding paragraph clearly demonstrates the impact as soon as referential constraint relationships are implemented between related entities (tables). Using surrogate keys between associated tables hides the natural relationship, which complicates looking at the data from both an end-user as well as a developer perspective. As soon as well-chosen natural keys are used, the relationship between entities is immediately self-evident from a business perspective.

The main impact for a database modernization effort then lies less with the primary key type and more with the use of RDBMS identity columns. Surrendering control of the key creation to the RDBMS requires careful consideration, as this can add significant overhead that actually exceeds the effort that creating a simple application-driven methodology entails. The latter, combined with simple uniqueness constraints, offers all the benefits that the RDBMS functionality provides while maintaining the control and flexibility to modernize the database in an orderly, incremental fashion.

Additional background reading:

1. http://www.agiledata.org/essays/keys.html
2. http://www.agiledata.org/essays/keys.html#KeyTips
3. http://www.agiledata.org/essays/databaseRefactoring.html
4. https://databaserefactoring.com/
5. https://www.databasejournal.com/features/mssql/article.php/3922066/SQL-Server-Natural-Key-Verses-Surrogate-Key.htm
6. http://www.dbjournal.ro/archive/4/6_Dragos_Pop.pdf
7. https://www.drdobbs.com/refactoring-for-fitness/184414821
8. https://www.refactoring.com/
9. https://en.wikipedia.org/wiki/Natural_key
10. https://en.wikipedia.org/wiki/Surrogate_key

Document Version: 2020-1
Publication Date: 2020/02/27