Changing the Way the Financial World Processes & Utilizes Information

Introduction

Speaker: Samuel Berger

Topic: Data Architecture for the Extreme Data Scientist

Description:

This presentation underscores the importance of creating precise data structures when handling, processing and manipulating mass amounts of data. As data has become key in the operations of virtually all major companies around the world, having the data easily maintained and utilized is pivotal. Companies often live or die in today's hyper-competitive business climate by their ability to advantageously manipulate their data. It is therefore paramount that this enterprise-critical data is housed in well-organized structures that are efficient for developers to work on. The bulk of this presentation offers tips and examples on how as well as the numerical benefits using a large data example.

Background

I am a Fintech entrepreneur and developer as well as a data scientist having worked mostly in mass data projects since 1989. I started in these fields with SBIC, using technology and massive amounts of data to predict the world's largest financial market – FOREX. My systems earned my clients (Daiwa Securities, Bank of Montreal, Julius Bär Group Ltd., Société Générale, royalty and national treasuries, to name a few) returns of over 18% per annum non-compounded over the 5 ½ years we traded. At peak my company traded the equivalent of over $1 billion in a day. I have consulted and developed solutions for E*Trade, VeriSign, Walt Disney Company, SGI, two founding VOIP unified messaging companies, and was the Enterprise Architect at Capital Group Companies (managed over $1.3 trillion with over 3,000 developers at the time). Currently our projects cover the communications, energy, financial, medical and mining industries. We focus on base innovations that make fundamental structural shifts in the way technology flows in those domains in the future.

Discussion Points

I. What is Data Science?
II. Relational Algebra / Normalization Familiarity
III. Brief History
IV. 1st Normal Form Simply Stated
V. Best Practices
VI. Practical Examples
VII. Key Structures
VIII. Performance and Data Space / Maintenance
IX. Overloaded Domains (Columns)
X. Theory Modified Slightly by Practice
XI. Locking
XII. Normalization Conclusion
XIII. Q & A

What is Data Science?

Using data to:

Determine relationships between a possible cause and an outcome – Applications include Medical, Science, Business, Politics, etc.

Determine expected profitability from prior data – Applications include Business Development, Marketing, Sales, Investments, International Trade, etc.

Controlling the narrative – Applications include Marketing, Legal, Pharmaceutical, Investments, Fund Raising, Politics, etc.

Create and enhance technologies that require mass data manipulation to provide products and features that cannot exist without the proper storage and utilization of mass data.

Relational Algebra
Database Normalization

Have you worked on a database that was in at least the First Normal Form (1NF)? Probably not.

At what point in Normalization duplicate rows are no longer allowed anywhere in the entire data schema?

At what point are row column intersections no longer allowed to carry NULL or meaningless values?

Answer: The First Normal Form

Brief History

Relational Algebra was primarily developed by Edger F. Codd from 1969 to 1973, and primarily documented by Chris Date, both IBM employees.

Codd also created his “12 rules” (really 13 as he started from zero) that were used to define the qualifications of a relational database management system (RDBMS).

Codd’s work heavily influenced IBM’s first RDBMS called System R back in 1973. System R was created by Ray Boyce and Don Chamberlin.

System R introduced the Standard Query Language (SQL), originally called SEQUEL while in development, hence the reason we use to refer to SQL Server as “Sequal” Server.

Codd and Boyce later teamed up to create the Boyce-Codd Normal Form, which is one step more confined than the 3rd Normal Form.

1st Normal Form (1NF)

My goal here is not to confuse but to simplify

Key principles:

1) Each row must have at least one unique key also referred to as a Candidate Key (i.e., no duplicate rows). A Candidate Key is the minimum column grouping on a table to create a unique record. Columns that do not help to define uniqueness are attributes of the Candidate Key, or Candidate Keys as the case may be should more than one unique column set exist.

2) Every row column intersection must have a value.

3) Every row column intersection can only contain one value, not a list of values.

4) Every row column intersection must have a valid value from the pool of potential valid values (i.e., a plane parts table cannot have a column for engine parts and then enter into it both engine parts as well as plane max speeds).

5) The functionality of the table is not dependent on the order of the data with respect to the order of rows or columns (i.e., querying the data will determine the column order and the row order of the output).

Best Practices

Modern database terminology uses the term Primary Key as a binder of the data more than as a concept of a unique row identifier based on data properties. As such the Primary Key is now a separate concept from the Primary Candidate Key. The Primary Key should always be an auto-growing integer starting sequentially from row 1, and the server prefers that it is the first column. My naming convention is the table name plus "_ID". A unique index or constraint should be used to enforce the Primary Candidate Key which is the column or set of columns that define a table's unique value(s) for each given row.

Table and column names should always be descriptive even if verbose. Mistakes occur most commonly due to lack of understanding of the data model and the purposes of each table and column container. Never use database keyword names for column names (i.e., typically *name*, *count*, *date*, *etc.)*.

Data should be related for the data's sake and not for the current application requirements. Requirements change, if the data is structured accurately then the data model will remain accurate and fully viable saving time and money.

Audit Columns

Also a component of best practices is to include audit columns to cover the following key pieces of data:

1) The date and time the record was created.
2) The person ID or process name that created the record.
3) The date and time the record was last updated.
4) The person ID or process name that made the update.
5) Update count – an important column for most tables that will be covered a bit later in this discussion.
6) A column indicating whether the record is currently active.

Note: In blockchain projects records are never updated, only added and as such there is a method to handle this requirement while not breaking normalization – will be covered in the webinar series to follow.

Practical Example

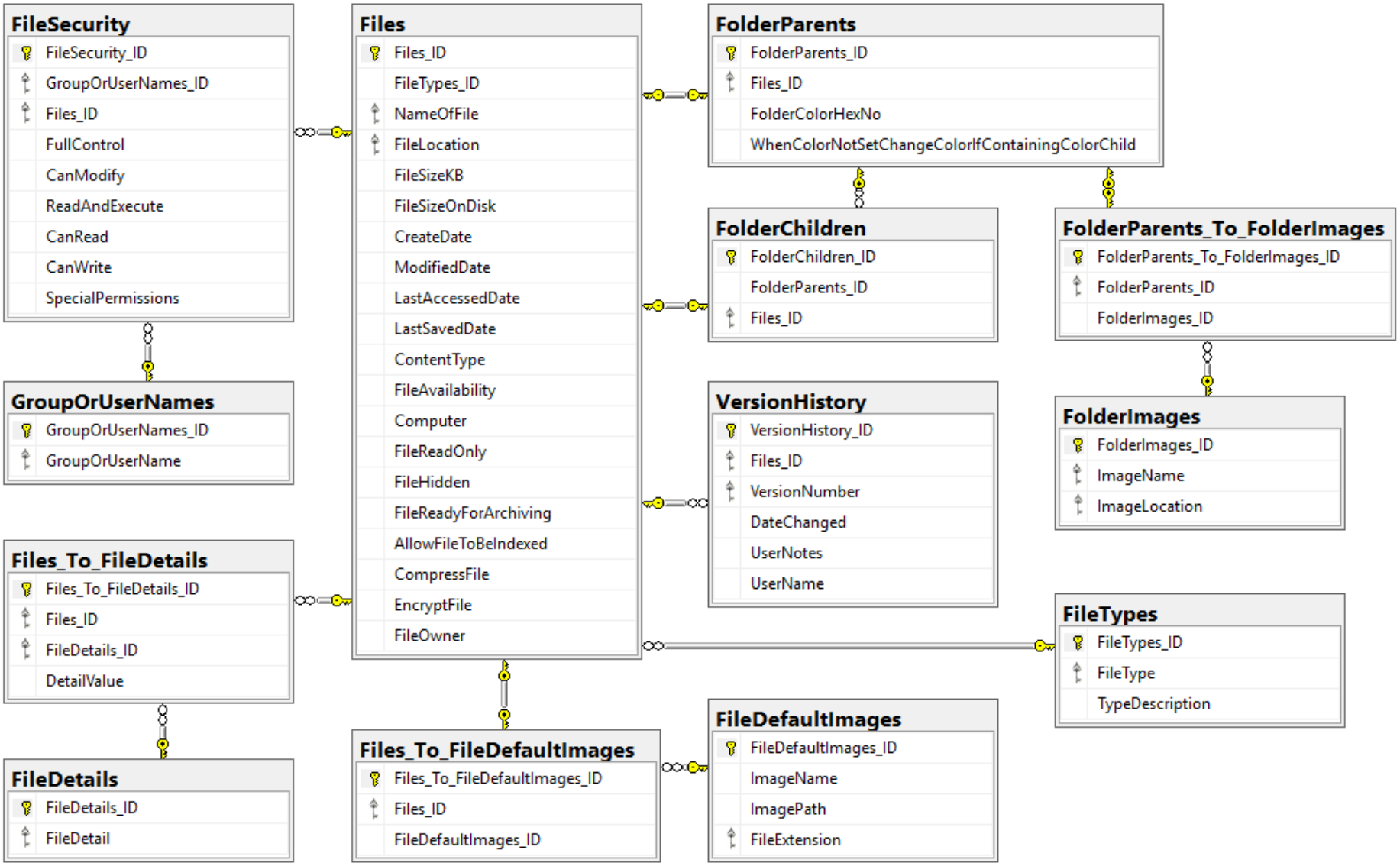
The following example is designed to help illustrate normalization. For this example I created a database schema for how I would build the Microsoft Explorer application from scratch.

I believe everyone here has used Microsoft Explorer and can fully visualize this exercise and see some of the power of the Normalized architectural design.

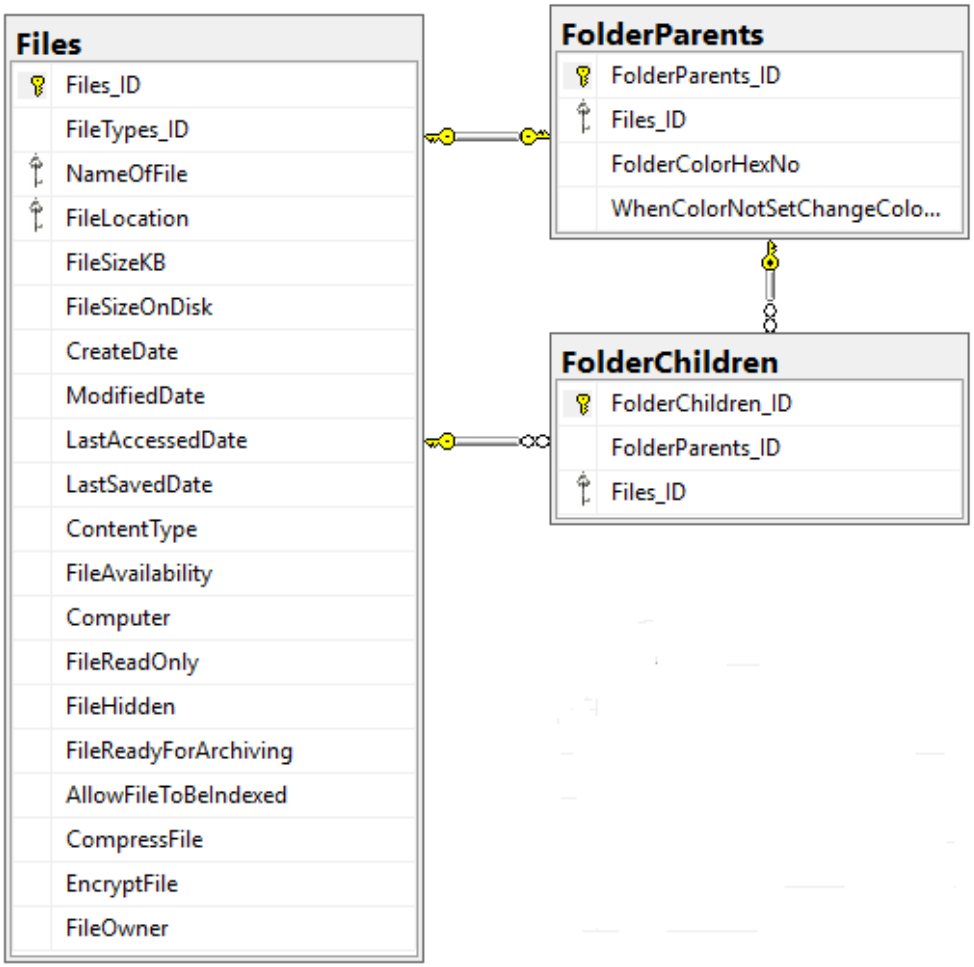
This small database meets the requirements of the Boyce-Codd Normal Form.

NOTES: This is not the actual Microsoft Explorer data model. This is how I would design it. Also, I added silver keys to denote the Candidate Key to each table. Lastly, audit columns do not apply to this simple application.

Explorer Data Model



Key Database Structure



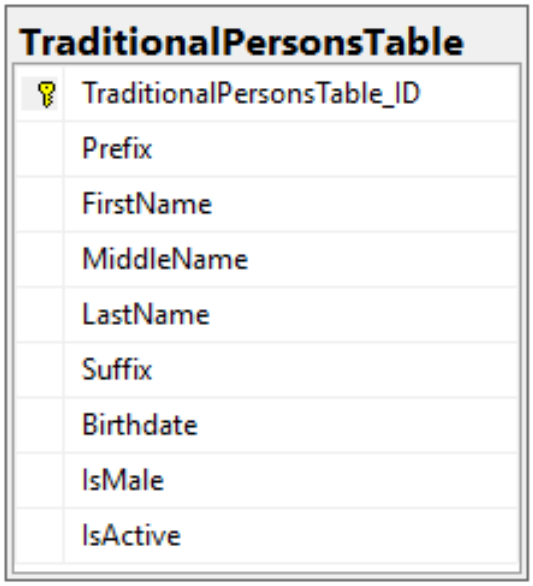
Second Example
“Persons”

This example is far more interesting. I have taken a very common (maybe the single most common) database architecture and restructured it using relational algebra properly. It should be noted I have never seen this level of Normalization used anywhere in the world, much to my surprise. This is particularly true with respect to the overloaded date time fields that are properly broken up in this demonstration. We do use these structures in CLEAR products in many locations (i.e., not just with the Persons information). It is the correct application of the mathematics and has massive benefits when applied to very large data as demonstrated numerically in the following slides.

Data and Architecture Notes

1) Loaded two identical 265 million person name record groups into both structures to give proper time and size comparisons in a big data environment.
2) As there are duplicate records with respect to first, middle and last names along with birthdays, no candidate key is possible in the traditional structure.
3) I used the 2010 US Census data for a list of last names and their frequency. They only included non-concatenated last names, as such I had to create my own concatenated examples.
4) I used a list of 2016 baby names in Scotland as it was the largest first name database that I could locate with a breakout between male and female names.
5) I randomly generated the first and middle names from this list of known names. I also randomly generated the order to enter the names into both table schemas to prevent bias.
6) I did not add the audit columns or display the field specifications for conceptual simplicity.

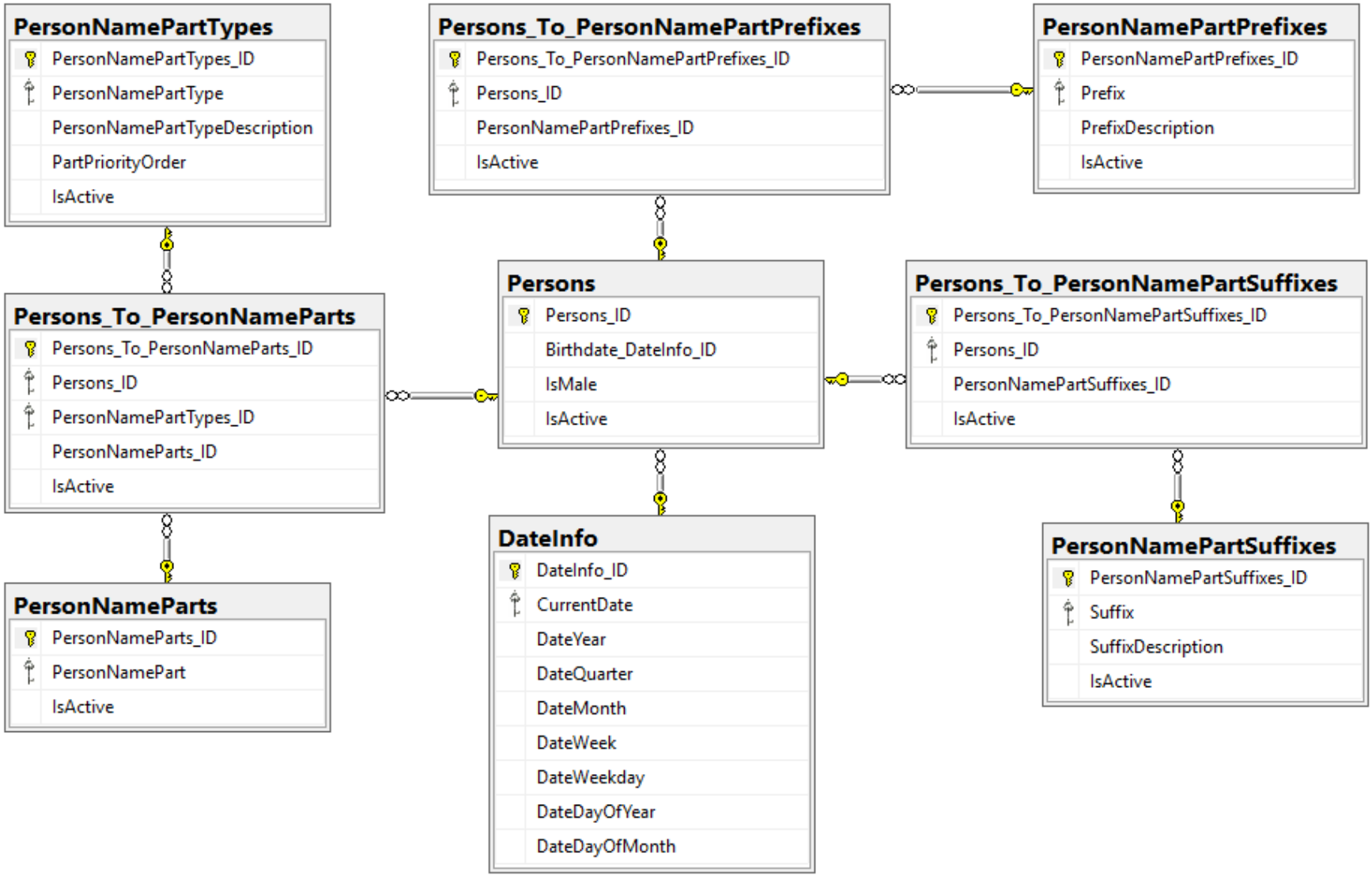
Typical Persons Table



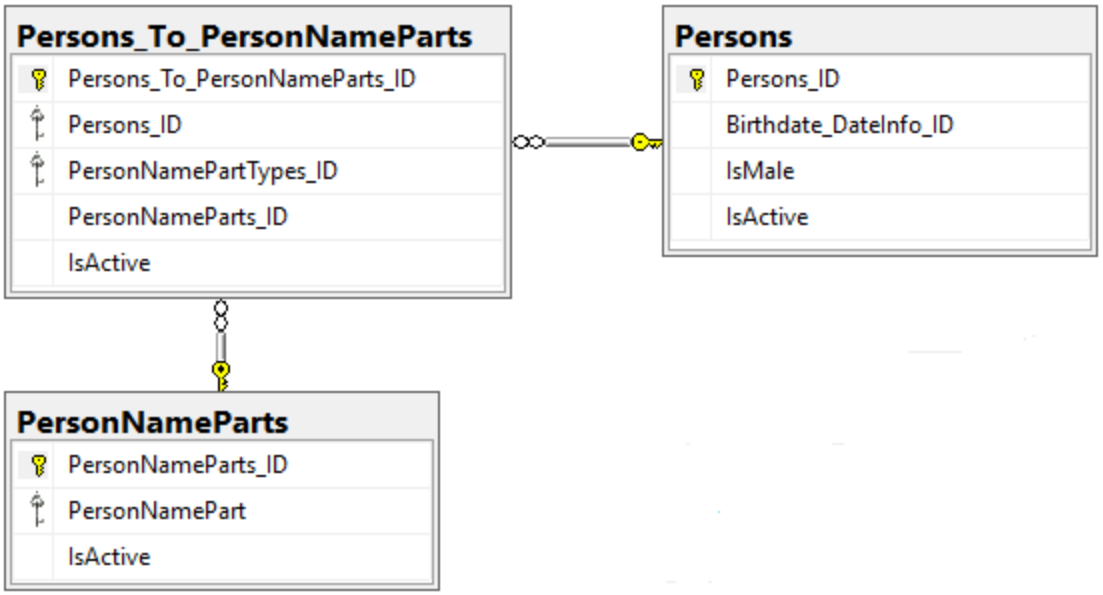
Notes:

1) No way to properly identify a candidate key in spite of important defining data.
2) MiddleName and Suffix will have to allow NULL or absent values.
3) Multiple middle names almost never supported.

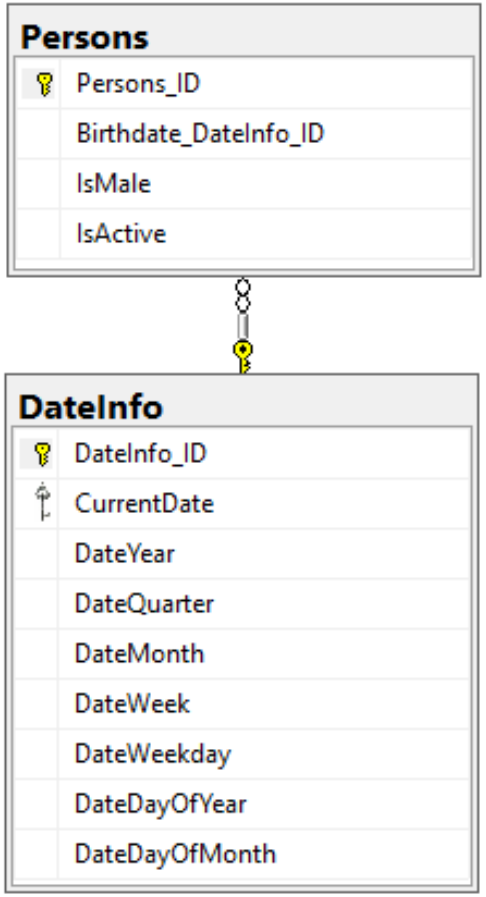
Normalized Persons Table Structure



1st Key Database Structure



2nd Key Database Structure



Statistics – Traditional vs. Normalized

Traditional data structure

Total data size that requires indexing for increased performance:

19.9 gigabytes – also requires multiple columns to be indexed and most likely multiple indexes

Time to count number of people born on May 5 of any year without indexing:

3 minutes 57.4 seconds

Total record count:

729,836

Time to return the ID for 1 person given first, last and middle names plus birthdate without indexing:

50.4 seconds

Statistics – Traditional vs. Normalized (Continued)

Normalized data structures

Total data size that would require indexes for increased performance:

5.2 megabytes – only requires a single column to be indexed

Time to count number of people born on May 5 without indexing:

1.5 seconds

Total records:

729,836

Time to return the ID for 1 person given first, last and middle names, and birthdate without indexing:

1.2 seconds

Statistics – Traditional vs. Normalized (Continued)

Difference

Data requiring indexing for optimization:

3,943.3 times the amount of data!

Time difference to retrieve information by date (Note: the date time field is not a data type that is eligible for indexing, which is a major performance problem):

153.9 times faster!

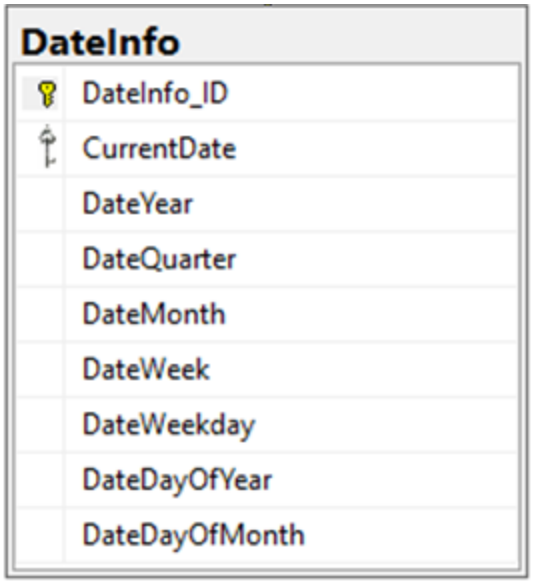
Without indexing either table structure – time to retrieve a single record by customer specific data:

43.1 times faster!

Conclusion: Normalization is always more efficient with respect to both performance and data maintenance when dealing with large quantities of data.

Overloaded Data Column

The First Normal Form requires each column to be a domain. A domain is a column that contains data from the "pool of legal values". Legal values for a ZipCode field are all known zip codes, not, for example, a street name. Columns that contain more than one informational piece are referred to as "overloaded". It can be accurately argued that the datetime data type, which is commonly used throughout databases, is an overloaded column.



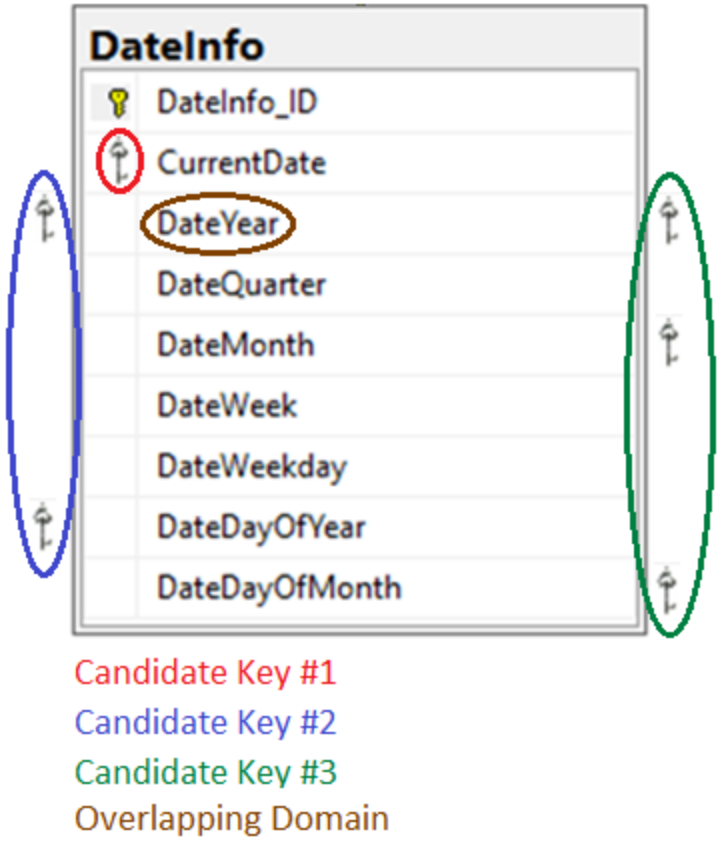
In Microsoft SQL Server the function "DATEPART" allows the following retrievals:

year	weekday	nanosecond
quarter	hour	TZoffset
month	minute	ISO_WEEK
dayofyear	second	
day	millisecond	
week	microsecond	
weekday	nonosecond	

Multiple Candidate Keys
Boyce-Codd Normal Form Slight Flaw

A table must be in third normal form (3NF). Additionally, every domain that determines the value of another domain must be in part or in whole a key (Candidate Key) that has no overlapping domains with another key.

This rule is mostly true. The exceptions primarily are with reference data as the difficulties maintaining data are moot at best. The DateInfo table to the right is an excellent example as the table data will rarely, if ever, change. I typically add records to support multiple centuries (36,525 days per century – tiny amount of data for a table to support). As such my database architectures are always in 3NF and never in Boyce-Codd as all the tables must be in compliance.



Database Table Locking
The Power of Logical Level Locking

1) In most RDBMS the default table locking for select statements is a shared lock.
2) Shared locks easily escalate in high volume production environments leading to poor performance and deadlocks.
3) In the vast majority of cases the lock proves unnecessary.
4) In most cases the locks can be completely avoided without creating concurrency issues by using an UpdateCount field and then applying a NOLOCK (or equivalent table locking hint) when selecting data and logic checks while conducting updates, inserts and deletes from the previously selected data. Also, shared locks are difficult for the system to hold causing disconnected record sets, which are typically both problematic for the system and very annoying for the user, which is solved by logical level locking.
5) In my experience the difference in high volume production environments and very large shared databases are in almost all cases massive.

Normalization Conclusion

1) Proper normalization of the data model can save companies working on big data and high production transaction databases tens of millions in hardware and maintenance expenses over the life of a company.
2) Performance and data integrity (which directly affects accuracy) in a production environment will always be more reliable and significantly faster using relational algebra.
3) No one claiming to be a professional database architect can make that claim without being proficient in relational algebra.
4) Even with data warehousing some data should always be normalized for maximum performance and flexibility. A good example is the date information, which can be used to rapidly slice and dice denormalized data marts efficiently for maximum flexibility with the data.
5) The database can make or break almost all projects. Proper database design, locking schema and efficient database code is always essential.

Intended Future Webinars

For those that are interested in learning more about data architecture as well as data analysis I will be conducting the following webinars over the coming year:

1) A formal and practical look at data architecture including numerous examples and visual displays of the performance differences as well as importance in accuracy and the longevity of products that are properly normalized. Will most likely be covered over 4 one hour sessions.

2) Building a normalized blockchain project. We will build a new crypto currency on the data side only showing how even a blockchain project can be fully normalized to the 3NF. This will be covered over 2 one hour sessions.

3) A true data science project – topic: Global Warming. We will create the data structures and then input all of the temperature data collected since 1850. We will then "clean" the data, which is a critical part of a data scientist job. Then we will look for information that can naturally askew (also referred to as "curve") the data. The goal of the project will be to confirm or deny the hypothesis that the world is getting hotter using pure data science. This will require participation by all and should be both very informative and fun! I anticipate this being covered over 4 one hour sessions.

Thank You

Samuel Berger
Chief Information Officer
sberger@ClearFinTech.com

Changing the Way the Financial World
Processes & Utilizes Information