IDERA® | Data Modeling Whitepaper

SEVEN DEADLY SINS OF DATABASE DESIGN

How to avoid the worst problems in database design

INTRODUCTION

Data architects face many challenges on a day-to-day basis.

In the data management arena, you may constantly hear from data professionals that if you don't get the data right, nothing else matters. However, the business focus on applications often overshadows the priority for a well-organized database design. The database just comes along for the ride as the application grows in scope and functionality.

Several factors can lead to a poor database design — lack of experience, a shortage of the necessary skills, tight timelines and insufficient resources can all contribute. In turn, poor database design leads to many problems down the line, such as sub-par performance, the inability to make changes to accommodate new features, and low-quality data that can cost both time and money as the application evolves.

Addressing some simple data modeling and design fundamentals can greatly alleviate most, if not all, of these concerns. Let's focus on seven common database design "sins" that can be easily avoided and suggest ways to correct them in future projects:

- **SIN #1** POOR OR MISSING DOCUMENTATION FOR DATABASE(S) IN PRODUCTION
- **SIN #2** POORLY DEFINED BUSINESS REQUIREMENTS
- **SIN #3** NOT TREATING THE DATA MODEL LIKE A LIVING, BREATHING ORGANISM
- **SIN #4** IMPROPER STORAGE OF REFERENCE DATA
- **SIN #5** NOT USING FOREIGN KEYS OR CHECK CONSTRAINTS
- **SIN #6** NOT USING DOMAINS AND STANDARD ELEMENTS
- **SIN #7** NOT UNDERSTANDING PRIMARY KEYS AND INDEXES PROPERLY

SIN #1 POOR OR MISSING DOCUMENTATION FOR DATABASE(S) IN PRODUCTION

Documentation for databases usually falls into three categories: incomplete, inaccurate, or none at all.

To make matters worse, it is almost never centralized in one place. Without proper, centralized documentation, understanding the impact of a change is difficult at best, and impossible at worst. This causes data stewards, developers, DBAs, architects, and business analysts to scramble to get on the same page. They are left up to their own imagination to interpret the meaning and usage of the data.

Additionally, developers may think that the table and column names are descriptive or self-explanatory enough to understand the usage of the data (more on this with Sin #6). However, as the workforce turns over, if there is no documentation in place, the essential knowledge about the systems can literally walk out the door and leave a huge void that is impossible to fill. Even starting bottom-up with just the physical information can help to alleviate the issue of lack of documentation tremendously.

Also in this day and age, we recognize that not only is data the new oil, but also a poisoned chalice. Certain data such as personal data must be treated with great caution. We need to be able to ask and answer the question, “Where is this data?” at any time. We also need to be able to go to any data asset and ask the question, “What rules apply to this data?”

Ideally we will maintain models of the technical metadata of our data assets in one place. These will be stored as Physical Data Models which clearly show the technical structure of the assets. This structure includes the table and column names, data types, and relationships between tables in RDBMS assets or documents in NoSQL assets. But this is only half of the problem, because we don’t know the meaning of each piece of data. We can document each piece of metadata in a number of ways:

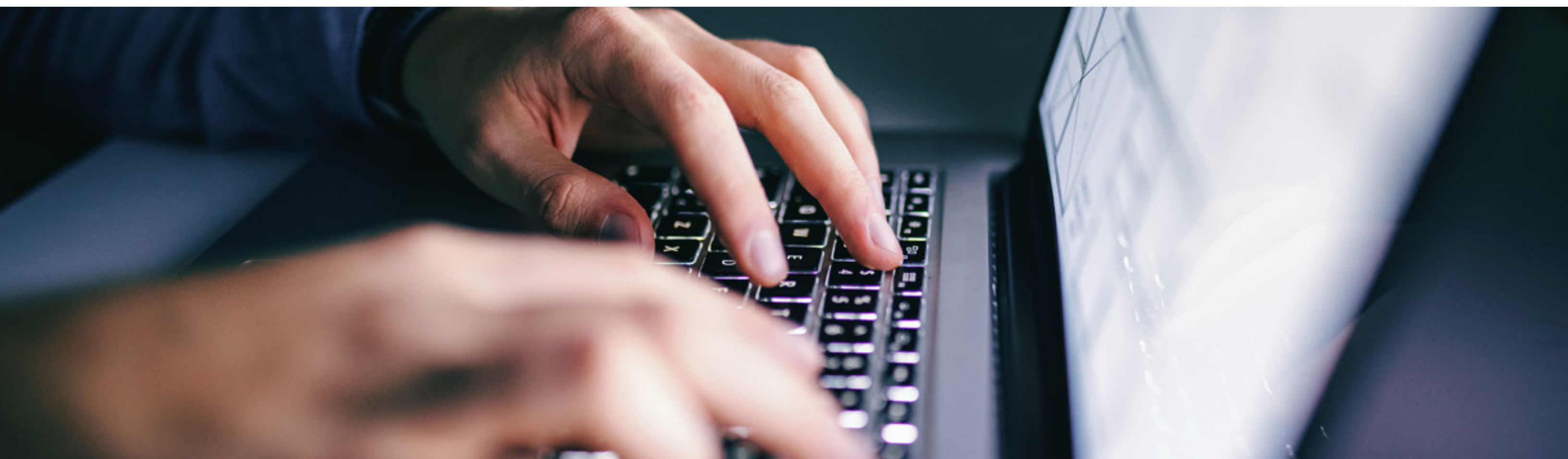
1. Mark up each table and column, etc. with business names and comments, ownership rules, etc.

2. Create a logical data model which shows the structure using business language, or connect to an existing common corporate model.

3. Classify each table, column, or document, etc. against Business Terms from within the Business Glossary.

This single contiguous model can now be interrogated to answer those important questions.

We can then go on to ask more questions about the most important information, such as applying data quality metrics.



SIN #2 POORLY DEFINED BUSINESS REQUIREMENTS

We have all seen the rope swing graphic where what the client asks for differs wildly from what the architect designs and what the developer delivers.

This chain of communication is vital to ensure that the clients needs are met. The initial capture of requirements must be in business language and in a form that the client can understand.

Logical data models are an effective way to show the concepts, their attributes, and relationships of a data asset, independent of the technology that implements the data asset. These logical models can then be transformed into a physical model more closely aligned with the target technology. This physical model will often employ abbreviated, more technical naming conventions which are more difficult for the end user to understand. Using a tool will provide traceability from the logical model through to the final database design in the physical model. This traceability allows testing to be performed.

SIN #3 NOT TREATING THE DATA MODEL LIKE A LIVING, BREATHING ORGANISM

There are numerous examples of customers performing the modeling up front, but once the design is in production, all modeling ceases.

It is always preferable to design first then build. It is proven that errors corrected in the design phase are significantly less costly than those fixed after production.

The sin arises when changes creep into the database due to critical production issues. Inevitably, the model is then left languishing on the side if there is not a process to update the model along with the database. As more changes occur in the database, the model becomes useless.

Undocumented data can also lead to security and compliance risks, poor understanding of future changes and the inability to adapt to future needs of the business. To maintain flexibility and ensure consistency when the database changes, those modifications need to find their way back to the model.

As the model changes, configuration management processes can be used where the requirements for changes are defined within the Logical Data Model, approved for sign-off then a version snapshot cast. Once approved, the physical model can then be updated and again approved and again a version snapshot cast. At this point tooling can help produce scripts to update the target data asset with those changes only. The DBA can view scripts for the asset and provide a final approval before implementation. In this way changes are controlled and documented. This will reduce the opportunity for errors to occur, and if they do, to identify where they came from and fix them.

SIN #4 IMPROPER STORAGE OF REFERENCE DATA

There are two main problems with reference data.

It is either stored in many places or, even worse, embedded in the application code. It is almost never captured in the data model with the rest of the documentation. This causes huge problems when a reference value needs to change.

Reference values provide valuable documentation which should be communicated in an appropriate location. Your best chance is often via the model. It may not be practical to store reference values in the data model if you have large volumes, but it is a best practice to point to them from the model. The key is to have it defined in one place and used in other places.

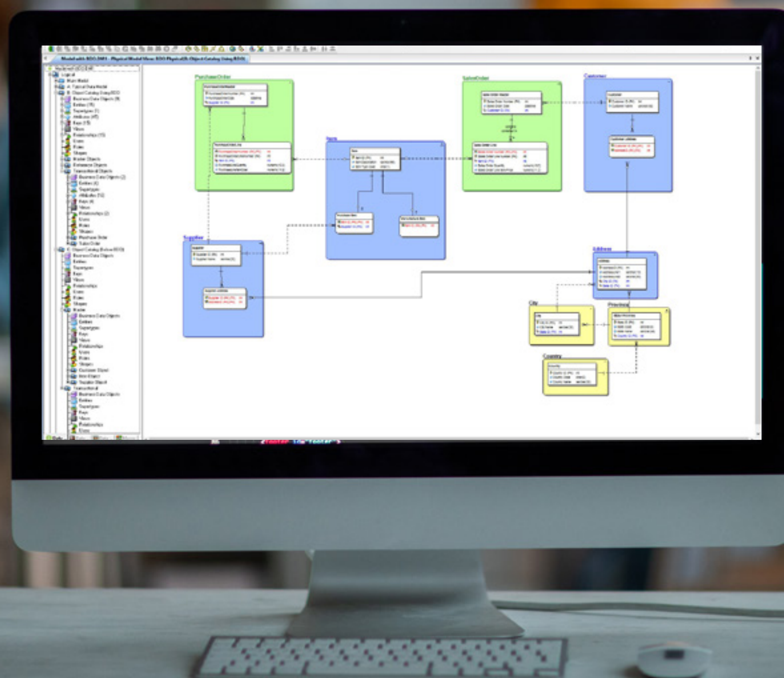
The benefit of this is that any changes can be understood before they are implemented, then properly controlled and documented.

SIN #5 NOT USING FOREIGN KEYS OR CHECK CONSTRAINTS

Customers complain all the time about the lack of referential integrity (RI) or validation checks defined in the database when reverse engineering databases.

For older database systems, it was thought that foreign keys and check constraints slowed performance, thus, the RI and checks should be done via the application. This might have been the case in the past, but DBMSs have come a long way.

The ramifications on data quality if the data is not validated properly by the application or by the database can be significant. If it is possible to validate the data in the database, you should do it there. Error handling will be drastically simplified and data quality will increase as a result.



SIN #6 NOT USING DOMAINS AND STANDARD ELEMENTS

Domains and naming standards are probably two of the most important things you can incorporate into your modeling practices.

Domains allow you to create reusable attributes so that the same attributes are not created in different places with different properties. It is extremely important to have a common set that everyone can use across all models. Naming standards allow you to clearly identify those attributes consistently.

Having a set of standards also ensures consistency across systems and promotes readability of models and code. You don't want short, cryptic names that users need to interpret. Given the advanced nature of the latest vendor releases, the days of limited column length name are over when building new databases. Always have a common set of classwords to identify key types of data and use modifiers as needed.

Reusing data elements from a well-maintained model will also help with Data Governance initiatives. Being able to trace the data in assets back to reusable elements in maintained Business Glossaries or Logical Models helps understand whether policies and data quality or usage rules are being maintained within the asset. Conversely, a model where data assets can be traced to reusable elements helps you find data for impact analysis or sources for Business Intelligence.

SIN #7 NOT UNDERSTANDING PRIMARY KEYS AND INDEXES PROPERLY

A senior data architect once said, "When choosing a primary key, you'd better get it right, because changing it down the line will be a royal pain."

Sure enough, another customer had the un-enviable chore of managing a migration project because a system used Social Security Number as a primary key for individuals. They found out the hard way that SSNs are not always unique and not everyone has one.

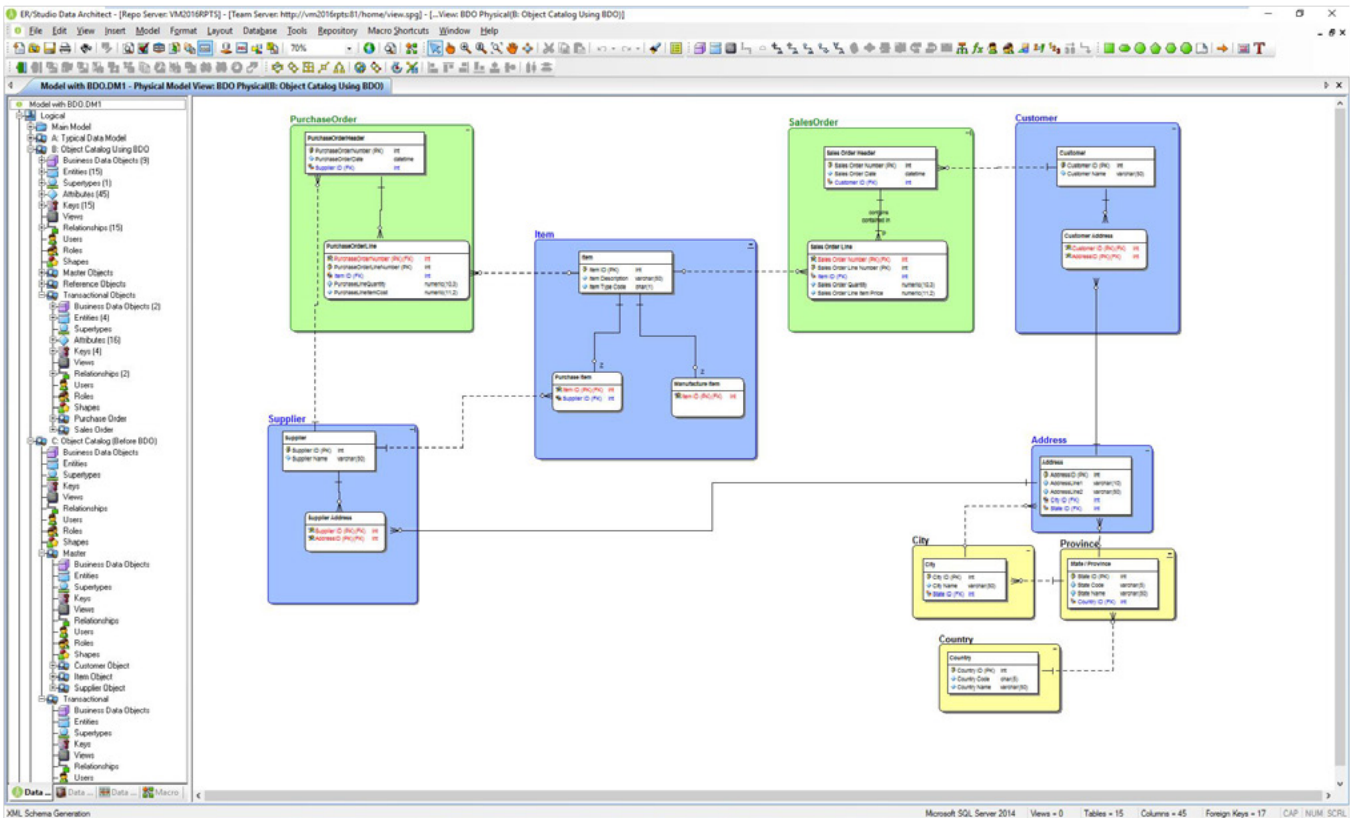
The simplest principle to remember when picking a primary key is SUM: Static, Unique, Minimal. It is not necessary to delve into the whole natural vs. surrogate key debate; however, it is important to know that although surrogate keys may uniquely identify the record, they do not always uniquely identify the data. There is a time and a place for both, and you can always create an alternate key for natural keys if a surrogate is used as the primary key.

This process of understanding how records can be identified early on also provides useful knowledge when optimizing the database later on. Capturing candidate keys during the design process makes defining indexes easier later on. Unique indexes provide optimized methods of retrieving, sorting, and finding data. Plus the data in the primary key may not be available when finding a record.

SIN-FREE DATABASES

It may be a multi-step process to define your strategy to eliminate all of these issues, but it is important to have a plan to get there.

Hopefully this information has helped you to evaluate your data management habits and assess your current database structure. IDERA can help you to address these issues with the **ER/Studio data modeling tools.** ER/Studio enables your business and technical stakeholders to map your complex data landscape, building a business-driven data architecture that serves as a solid foundation for data governance. Create consistent data models, document metadata, and improve data quality for your organization.



Start for FREE

Contact us to learn more.

IDERA