WHITEPAPER

Accelerating Sql To NoSql Migration Using erwin Data Modeler by Quest

Authored by:

Ramanan Ramaraj, Selvakumar Thangamuthu, Gayatri Raje

Table of Content

1. Introduction **3**

2. Top Differentiating Features of NoSQL **5**

3. Migrating SQL to NoSQL Database using erwin Converter **6**

I. Preparation for SQL-to-NoSQL Migration 6

II. Migrating SQL to NoSQL Data model 8

a. Option A: Target Database Approach 9

b. Option B: Deriving Model Approach 10

III. Data Migration using LTIMindtree NOSQL Migrator 13

4. Conclusion **14**

5. About the Author(s) **15**

1. Introduction

In last two decades, there has been massive investment in digital technologies by companies worldwide to innovate and disrupt existing business models with an intent to remain relevant in a highly competitive digital economy. They are continuously investing in web, social media, cloud, mobile, IoT, analytics, artificial intelligence, and cognitive technologies to upgrade their business systems. They are leveraging these technologies to:

- Increase sales revenue by providing their customers an enhanced omnichannel experience by seamless integrating their digital customer platforms with their internal systems such as ERP, CRM, PIM, CPQ, Digital Asset Management, etc. They are regularly improving the intuitiveness, responsiveness, and security of their web and mobile applications to provide a superlative user experience for their customers.
- Improve profitability by improving the efficiency and effectiveness of business and finance operations with technology interventions such as robotics automation, IoT, and Cognitive technologies.
- Improve the quality and speed of decision-making at all management levels by leveraging the data produced by humans and things (machines and products).

To achieve these goals, these systems must be designed and developed to:

- Meet changing requirements, add new or remove existing features with agility
- Support a large number of concurrent users who are globally distributed
- Support user performing read and write operations without compromising on the responsiveness
- Manage structured and unstructured data – varied in volume, velocity, variety, and veracity

SQL database are not the best suitable for this purpose, they have fixed data model. It is a manual and time-consuming process to modify model ever-time the requirement changes. It is meant to handle only structured data, often limiting the capability of the business application. It cannot store and manage large data volume – it can be scaled by upgrading the server resources, which proves very costly.

On the other hand, NoSQL Databases are most suitable to meet changing requirements, especially to manage systems or applications that need distributed data stores to store and manage a huge volume of structured and unstructured data. NoSQL databases are built to be flexible, scalable, and capable of rapidly responding to the data management demands of digital businesses.

What is NoSQL?

Traditional RDBMS uses SQL syntax to store and retrieve structured data. NoSQL database system, on the other hand, encompasses a wide range of database technologies that can store structured, semi-structured, unstructured, and polymorphic data. NoSQL database stands for "Not only SQL". It does not require a fixed schema. It avoids joins and is easy to scale. NoSQL database can store and retrieve data using literally "no SQL."

There are primarily four types of NoSQL Databases - Document DB, Key-Value stores, Wide-Column, and Graph DB. A few popular databases in NoSQL Database are as follows:

- Document DB – MongoDB, CouchDB, RavenDB, ArangoDB
- Key-Value DB – Apache Ignite, Redis, Memcached
- Wide-Column DB – Cassandra, Apache HBase, Scylla
- Graph DB – Neo4j, AllegroGraph, Orient DB

2. Top Differentiating Features of NoSQL

- **Multi-Model**

  NoSQL databases offer a great deal of flexibility when working with data. We don't have to specify a schema to work with your application. NoSQL databases do not limit the types of data that can be stored together. New types can be added as our needs change. All of this makes NoSQL a perfect fit for agile development that requires rapid implementation.

- **Easily Scalable**

  NoSQL databases are built using a masterless, peer-to-peer architecture. Data is split and distributed across multiple nodes in the cluster, and aggregated queries are distributed by default. This scalability improves performance, providing continuous availability and extremely high read/write speeds.

- **Distributed**

  NoSQL databases are designed to distribute data globally. It uses multiple locations involving multiple data centers and/or cloud regions for write and read operations.

- **Redundancy and Zero Downtime**

  The master class structure of the NoSQL database lets in a couple of copies of information to be maintained throughout specific nodes. If one node is going down, then some other node will have a duplicate of the information for smooth and speedy access. This ends in 0 downtime withinside the NoSQL database. When one considers the cost of downtime for missing critical applications, that is a big deal.

- **Big Data Applications**

  NoSQL is best- suited for big data applications since it can manage an enormous volume of data quickly. NoSQL databases make sure that when all the other parts of your server-side programme are created to be seamless and quick, data doesn't become an impediment.

In this document, we will demonstrate how to swiftly migrate data from a SQL to NoSQL database. We will use SQL Server (Source SQL) and MongoDB (Target NoSQL) for illustration.

3. Migrating SQL to NoSQL Database using erwin Converter

Most of us in the data analytics industry are familiar with erwin data modeler by Quest (erwin DM). It is a data modelling tool for visualizing metadata and database schema to understand complex data sources and design and deploy new ones. erwin data modeler makes it easier to design, develop and deploy large-scale data warehouses, data lakes, and master data management programs.

erwin DM enables the auto-documentation of schema and auto-transforms that schema for deployment on other DBMS platforms. It facilitates metadata discovery and migration of SQL-to-NoSQL. It also lets business users visualize data from anywhere, irrespective of location. Data architects and integration specialists get a better understanding of the logical and physical data models.

In this document, for the illustration of MS SQL to MongoDB migration, we will use erwin DM schema transformation services to automate the migration of database objects.

Preparation for SQL-to-NoSQL Migration

Before commencing the migration, it is important to under the database objects equivalent between SQL Database and NoSQL Database objects. If there are non-compatible database objects, we need to understand how erwin DM or any other conversion tools is going to handle these exceptions.

The table below provides a mapping of MS SQL DB Objects and MongoDB objects.

SQL Server Database Objects	MongoDB Database Objects	Supported by erwin DM
Database	Database	Yes
Tables	Collection – Collation	Yes
Row & Columns	Document and Fields	Yes
Index	Index	Yes
Relationships	The child table will convert to either array or an embedded document	No
User ID – Roles	User ID – Roles	Yes
User ID - Permissions	Manually we need to give permission	No
View	View	Yes

Similarly, not all data types in SQL Server DB are supported by MongoDB. The table provides a view SQL Server data type equivalent in MongoDB.

SQL Server Data Type	MongoDB Data Type	erwin DM
Char, Varchar, text	String	Yes
Decimal, Numeric	Decimal	Yes
Int	Int	Yes
nchar, nvarchar, ntext	-	No
Binary	Binary Data	Yes
Date	Date	Yes
Datetime	Timestamp	Yes
image	Binary Data	No

Few datatypes in SQL Server DB does not have an equivalent in MongoDB. erwin DM datatype mapper called DSM provides best possible match to facilitate the automated migration and is easily customizable. However, such migration could lead to problems such as memory-out issues. This will be a key consideration during the migration. We need to write a program to store the files on GirdFS that exceed the BSON document size limit of 16 MB. Also, erwin DM Handles other NoSQL Conversion like CouchDB, Arango DB ,Cassandra….etc.

Migrating SQL to NoSQL Data model

erwin DM provides two possibilities to migrate SQL to NoSQL Database:

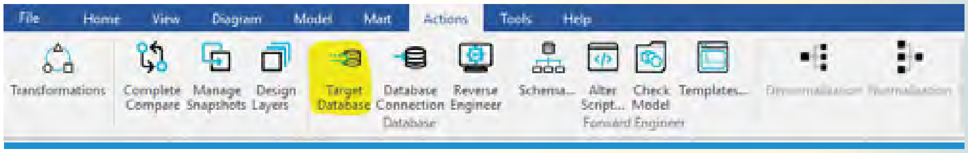
- Target database
- Deriving a model

The target database option only performs migrations, but in derived model option the objects in the source and target models are linked, so you can change the objects in both models and keep the two models in sync. This allows you to maintain the hierarchy of your design layers. Keeping historical information saves the history of each entity, attribute, table, and column in the derived model. You can select a model object from a derived model and see which model object was used to create the object.

We recommend selecting the Target Database option if one-time or big bang approach to migration. When we consider a staggered and agile approach for migration, we recommend the Derived model. Any change applied to the Source SQL will automatically get reflected in Target NoSQL. We can keep this session active until the entire process of migration is completed.

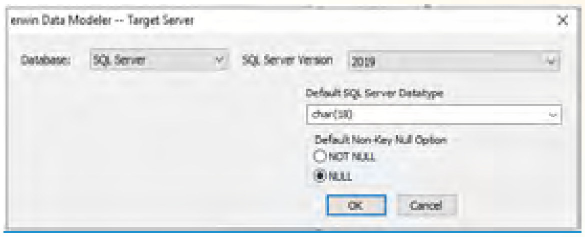
Option A: Target Database Approach

Go to Menu --> Action --> Click on Target Database Icon (Highlighted)



Step 1

Connect to the source database. i.e., Microsoft SQL Server Database



erwin Data Modeler -- Target Server

Database: SQL Server

SQL Server Version 2019

Default SQL Server Datatype

char(18)

Default Non-Key Null Option

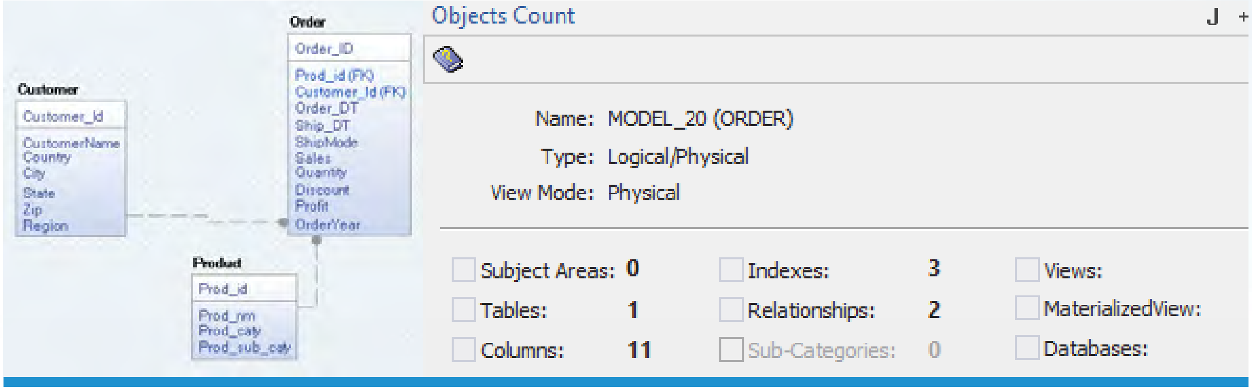
○ NOT NULL

● NULL

Step 2

Open your relational model in erwin DM. Ensure that you are in the physical mode.

For example, the following image uses the SQL-NoSQL. erwin. In the Objects Count pane, note the number of tables, columns, and relationships



Name: MODEL_20 (ORDER)

Type: Logical/Physical

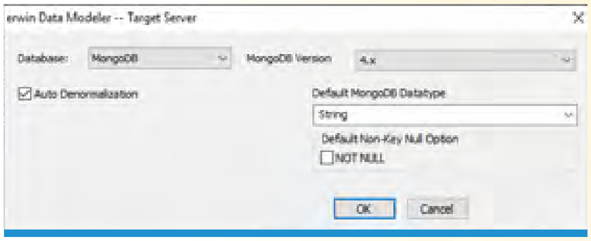
View Mode: Physical

□ Subject Areas:	0	□ Indexes:	3	□ Views:	
□ Tables:	1	□ Relationships:	2	□ MaterializedView:	
□ Columns:	11	□ Sub-Categories:	0	□ Databases:	

Step 3

Select the Target Database. In the Database drop-down list, select MongoDB.

By default, the Auto Denormalization check box is selected. Keep it selected.



erwin Data Modeler -- Target Server

Database: MongoDB

MongoDB Version 4.x

☑ Auto Denormalization

Default MongoDB Datatype String

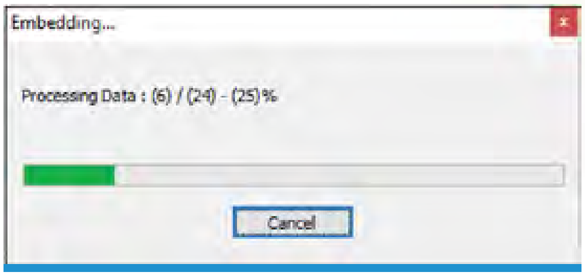
Default Non-Key Null Option

☐ NOT NULL

Step 4

Initiate the migration

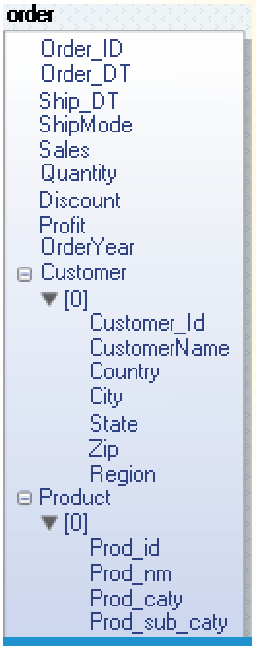
Click OK. The conversion process starts.



Embedding...

Processing Data : (6) / (24) - (25)%

Once the conversion is complete, the existing model in migrated to a NoSQL database can be visualized as below.



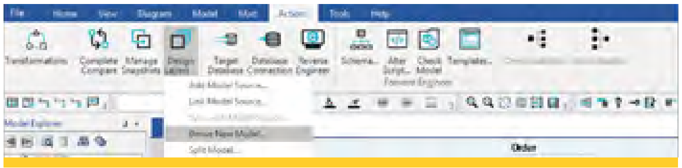
order

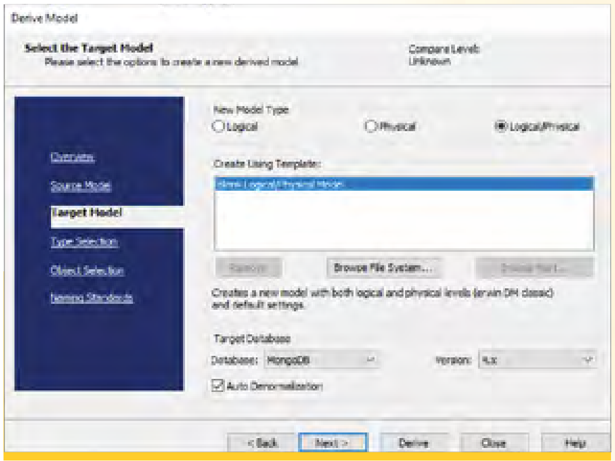
Order_ID
Order_DT
Ship_DT
ShipMode
Sales
Quantity
Discount
Profit
OrderYear
⊟ Customer
▼ [0]
Customer_Id
CustomerName
Country
City
State
Zip
Region
⊟ Product
▼ [0]
Prod_id
Prod_nm
Prod_caty
Prod_sub_caty

Option B: Deriving Model Approach

Open your relational model in erwin DM. Ensure that you are in the Physical mode.

On the ribbon, click Actions > Design Layers > Derive New Model. The Derive Model screen appears. By default, the Source Model is set to your current model





Derive Model

Select the Target Model
Please select the options to create a new derived model

Compare Level:
Unknown

New Model Type
○ Logical ○ Physical ● Logical/Physical

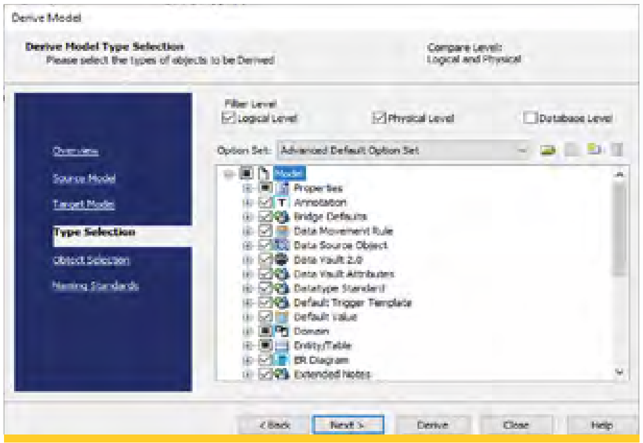
Create Using Template:
Blank Logical/Physical Model

Creates a new model with both logical and physical levels (erwin DM classic) and default settings.

Target Database
Database: MongoDB
Version: 4.x
☑ Auto Denormalization

In the **Database** drop-down list, **select MongoDB**. By default, the Auto Denormalization check box is selected. Keep it selected Click Next.

** Note: If the Type Resolution screen appears, click Finish. The Type Selection section appears.



Derive Model

Derive Model Type Selection
Please select the types of objects to be Derived

Compare Level:
Logical and Physical

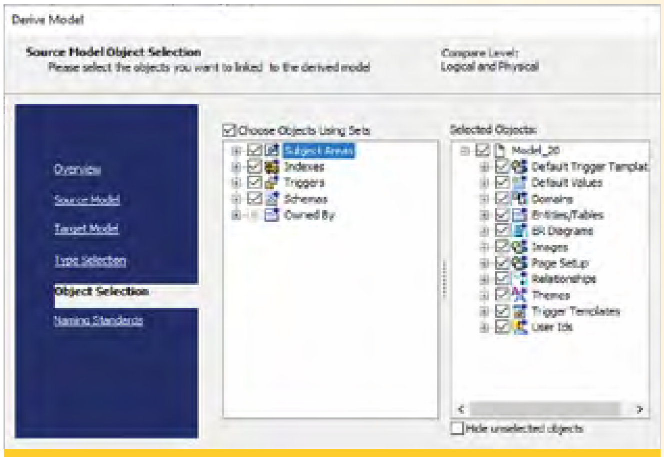
Filter Level
☑ Logical Level ☑ Physical Level □ Database Level

Option Set: Advanced Default Option Set

- Model
  - Properties
  - ☑ Annotation
  - ☑ Bridge Defaults
  - ☑ Data Movement Rule
  - ☑ Data Source Object
  - ☑ Data Vault 2.0
  - ☑ Data Vault Attributes
  - ☑ Datatype Standard
  - ☑ Default Trigger Template
  - ☑ Default Value
  - Domain
  - Entity/Table
  - ☑ ER Diagram
  - ☑ Extended Notes

Select the types of objects that you want to derive into the target MongoDB model.

Click Next. The Object Selection section appears. Based on the object types you selected in Previous Steps; it displays a list of objects.



Derive Model

Source Model Object Selection

Please select the objects you want to linked to the derived model

Compare Level:
Logical and Physical

☑ Choose Objects Using Sets

- ☑ Subject Areas
- ☑ Indexes
- ☑ Triggers
- ☑ Schemas
- □ Owned By

Selected Objects:

- ☑ Model_20
  - ☑ Default Trigger Templat
  - ☑ Default Values
  - ☑ Domains
  - ☑ Entities/Tables
  - ☑ ER Diagrams
  - ☑ Images
  - ☑ Page Setup
  - ☑ Relationships
  - ☑ Themes
  - ☑ Trigger Templates
  - ☑ User Ids

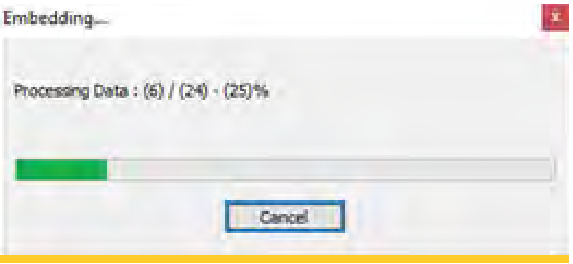
□ Hide unselected objects

Select the objects that you want to derive into the target MongoDB model.

Step 4

Initiate the migration

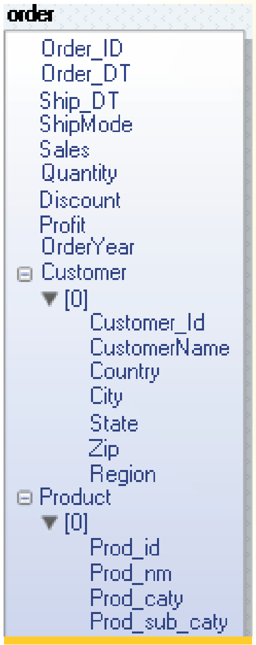
Click **Derive** button to start the model derivation process



Embedding...

Processing Data : (6) / (24) - (25)%

Once the conversion process is complete, the existing model migrated to a NoSQL database can be visualized as below.



```
order
    Order_ID
    Order_DT
    Ship_DT
    ShipMode
    Sales
    Quantity
    Discount
    Profit
    OrderYear
⊟ Customer
    ▼ [0]
        Customer_Id
        CustomerName
        Country
        City
        State
        Zip
        Region
⊟ Product
    ▼ [0]
        Prod_id
        Prod_nm
        Prod_caty
        Prod_sub_caty
```

Data Migration using LTIMindtree NOSQL Migrator

LTIMindtree NoSQL Migrator automates the migration of data from a RDBMS to a NoSQL database post the erwin DB object transformation. LTIMindtree NoSQL Migrator involves a four-step process:

- Configure: Configure the source and target database connections
- Assess: Assess the compatibilities and incompatibilities between the source and target database objects and datatypes
- Migrate: Trigger the data migration
- Validate: Reconcile the data in the source and target

Additionally, once the data is migrated, we can use Toad Data Point by Quest to interrogate the migrated data to validate that the data integrity has been maintained by enabling data comparisons & remediation support. Toad Data Point delivers comprehensive data preparation, profiling, and query support, in a single tool, for SQL and NoSQL databases.

We can also leverage Fog light for DBs by Quest to monitor, analyze and remediate database performance across heterogenous data platforms, including MS SQL Server and MongoDB to ensure that the migrated database is performing optimally and meeting SLAs.

4. Conclusion

There are various methods and tools in the market that facilitate migration from SQL to NoSQL. erwin DM provides a cost-effective and risk-free mechanism to migrate SQL to NoSQL. erwin DM stands out among all options because metadata captured in erwin can be easily integrated with third-party tools and erwin Data Intelligence (DI) by Quest. erwin DI provides a comprehensive metadata-driven data Catalog and data governance capability to ensure visibility, control and insights such as impact analysis and data lineage to increase the literacy and trust in the data and applications migrated through this process. Also, We have an add-on Utility called LTIMindtree NoSQL Migrator that assists in moving data from the Source DB to the Target DB.

If you would like to learn more about how LTIMindtree can help you assess alternatives to your RDBMS and automate the migration of your SQL database to NoSQL database, reach us at info@lntinfotech.com.

4. About the Author(s)

Ramanan Ramaraj

Associate Principal - Data Engineering Solution Architect

Ramanan Ramaraj, Associate Principal – Data Engineering Solution Architect has more than 17 years of experience in developing data and analytics solutions in transportation, supply chain and human resource domain. He is proficient in designing and modeling enterprise scale data warehouse and business intelligence solutions.

Selvakumar Thangamuthu

Senior Specialist - Data Engineering Architect

Selvakumar Thangamuthu is a Senior Specialist – Data Engineering Architect with more than 20 years of experience architecting complex data projects for customers in Insurance and other industries. He is adept in designing applications using SQL and NoSQL databases.

Gayatri Raje

Data Engineer

Gayatri Raje is a Data Engineer with over 2 years of experience in business intelligence & analytics. She is a proficient python programmer with experience in building intuitive analytics applications using Python.

LTIMindtree

info@ltimindtree.com