Application Modernization: Migrating Stored Procedures

Companies need agile data exploration, flexibility in storing diverse data, and reduced complexity for modern applications. Migrating stored procedures into MongoDB enables this transformation, replacing outdated relational schemas with aggregation pipelines for enhanced performance, streamlined management, and modern development practices.

A relic of the past

Stored procedures were crucial in relational databases. However, as the digital landscape evolves, stored procedures present limitations in flexibility, maintainability, and role separation that hinder application modernization.

Need to streamline data management

Modern applications demand a complete reimagining of traditional data models, focusing on efficiency, streamlined management, and diverse data handling.

Multi-phase approach

A successful migration of stored procedures requires refactoring, platform modernization, and performance tuning to support modern database technology.

App Modernization Factory

The App Modernization Factory from MongoDB provides a detailed roadmap for migrating stored procedures and adopting modern development practices.

Efficient processing and analysis

Aggregation pipelines offer a flexible and high-performance alternative to stored procedures.

Drawbacks of Stored Procedures

- Stored procedures are difficult to debug, test, version, and maintain, hindering continuous integration and delivery.
- They create obstacles for database migration and optimization due to their lack of portability.
- The use of stored procedures complicates role management and access rights between developers and DBAs.
- They may suffer from suboptimal execution plans, negatively impacting performance under certain conditions.
- Stored procedures limit the ability to adapt to changing data models and business requirements.

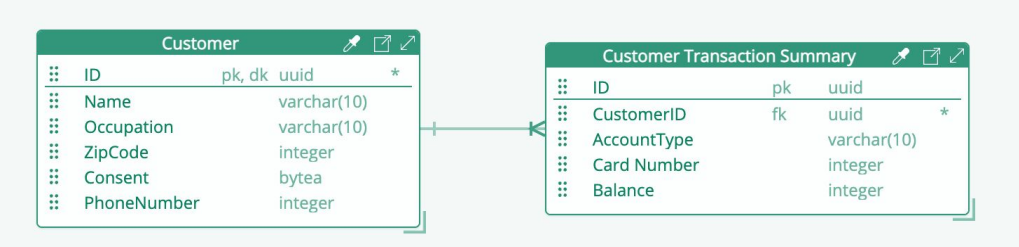
Modernizing with MongoDB

- Aggregation pipelines: Efficient data processing, analysis, and transformation as an alternative to stored procedures.
- Atlas Functions: Server-side JavaScript for app behavior definition and service integration.
- Atlas Triggers: Event-driven or scheduled app and database logic execution on a scalable, serverless layer.
- Change streams: Real-time data change monitoring, with optional aggregation pipeline use for filtering or transforming notifications.
- Atlas HTTPS endpoints: Customizable API routes or webhooks for app-specific integration using serverless functions.

MongoDB.

Implementing a stored procedure with an Aggregation Pipeline

Based on the following relational schema, here's how a non-trivial stored procedure can be easily migrated into MongoDB using the Aggregation Framework.



Stored procedures execute complex logic for different scenarios using conditions, loops, clauses, and transformations. Our example retrieves information for the top 10 customers using dynamic WHERE clause based on input parameters to include the entire dataset or a filtered subset.

Stored Procedure

```
CREATE PROCEDURE TOPCUSTOMERS
  @ZipCode varchar(50) = null,
  @Occupation varchar(50) = null
AS
BEGIN
  IF (@ZipCode IS NOT NULL AND @Occupation IS NOT NULL)
  BEGIN
    SELECT TOP 10 C.Name, C.PhoneNumber,
    REPLACE(LEFT(T.CardNumber, 12), SUBSTRING(T.CardNumber, 1, 4), '****') + RIGHT(T.CardNumber, 4) AS MaskedCardNumber
    FROM Customer C
    INNER JOIN (
      SELECT *
      FROM CustomerTransactionSummary
      WHERE CustomerID IN (
        SELECT ID
        FROM Customer
        WHERE ZipCode = @ZipCode AND Occupation = @Occupation
      )
    ) T ON C.ID = T.CustomerID
    ORDER BY C.Name
  END
  ELSE
  BEGIN
    SELECT TOP 10 C.Name, C.PhoneNumber,
    REPLACE(LEFT(T.CardNumber, 12), SUBSTRING(T.CardNumber, 1, 4), '****') + RIGHT(T.CardNumber, 4) AS MaskedCardNumber
    FROM Customer C
    INNER JOIN CustomerTransactionSummary T ON C.ID = T.CustomerID
    ORDER BY C.Name
  END
END
```

MongoDB Aggregation

```
db.customer.aggregate([
 {
  $match: { Occupation: "Employed",
            ZipCode: "75019"}
 },
 {
  $project: {
   _id: 0,
   Name: 1,
   PhoneNumber: 1,
   CardNumber: {
    $concat: [ "**********",
     {$substr: ["$CardNumber", 12, 4] } ]
   }
  }
 },
 {
  $sort: { Name: 1 }
 },
 {
  $limit: 10
 }
])
```

Note: Customer and Customer Transaction Summary tables are modeled within the same collection on MongoDB. If the aggregation pipeline is being frequently executed, it can be transformed into views. By doing so, we can utilize find queries on these views to efficiently access and analyze the data.

For more information:

- Deep-dive Technical Guide
- Comprehensive Modernization Guide
- Contact us at sales@mongodb.com
- Documentation Aggregation Framework
- Developer Hub mongodb.com/developer

MongoDB.