**EPIC**: MTA

Contents

[Story: Database management 2](#_Toc377744800)

[Task: Schema migration 2](#_Toc377744801)

[Using DTS to transfer data 3](#_Toc377744802)

[Task: Database Encryption 6](#_Toc377744803)

[Encryption using TSQL (CRU-operations) 6](#_Toc377744804)

[Encryption of Column (permanent) 7](#_Toc377744805)

[Encrypted Database Connections 7](#_Toc377744806)

[Real-time Encryption 7](#_Toc377744807)

[Other links 7](#_Toc377744808)

[Third party components 7](#_Toc377744809)

[Story: Patch management 8](#_Toc377744810)

[General Design 8](#_Toc377744811)

[POC Details 10](#_Toc377744812)

[Task: Strategy for Shared DB, Shared Schema 12](#_Toc377744813)

[Task: Strategy for Shared DB, Shared Schema 13](#_Toc377744814)

[Task: Strategy for Individual DB 14](#_Toc377744815)

[Task: Build Admin Console for customization of XML Schema 14](#_Toc377744816)

[Sub-Task: Create new customization 14](#_Toc377744817)

[Sub-Task: Update existing customization 14](#_Toc377744818)

[Sub-Task: Delete existing customization 14](#_Toc377744819)

[Logic Flow in MTA 15](#_Toc377744820)

# Story: Database management

## Task: Schema migration

**Background**: Each tenant has its own dedicated schema in the shared database.

**Description**: Tenant wants to migrate their data on to a different server, hence extract all relevant DB objects

**Input**: Tenant’s schema name

**Output**: Script to create the given schema’s copy somewhere and a mechanism to transfer all data and then delete the tenant’s component in source DB

**Processs**:

1. Create users in new environment
2. Create schema in new environment
3. Get all tables in given schema and generate their script.
4. Get all procedures in given schema and generate their script
5. Create DTS package for data movement

Scripts (available in DB MultiTenantPOC)

* sp\_ScriptTable: generates the create script for a table, given the schema name and table name. The table names are retrieved from sys.tables as given below

|  |
| --- |
| SELECT '['+SCHEMA\_NAME(schema\_id)+'].['+name+']'  AS SchemaTable  FROM sys.tables where SCHEMA\_NAME(schema\_id) = 'Dominos' |

* The migration for stored procedures can be created using following script

|  |
| --- |
| SELECT  pr.name ,  REPLACE(mod.definition, 'CREATE PROCEDURE [base].', 'CREATE PROCEDURE [' + @newClientName + '].') as FinalSP  FROM sys.procedures pr INNER JOIN sys.sql\_modules mod  ON pr.object\_id = mod.object\_id  WHERE pr.Is\_MS\_Shipped = 0 and SCHEMA\_NAME(schema\_id) = 'base') |

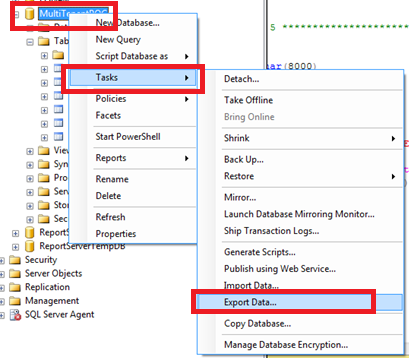
**Data transfer**: Using Data Transformation Services (DTS) tool in SQL server

### Using DTS to transfer data

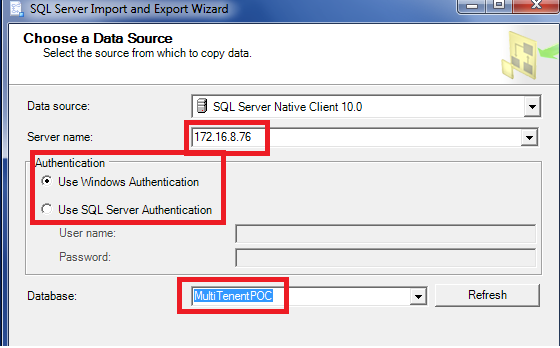
**Notes**:

Upon selection of source tables, DTS automatically identifies the schema associated with the table, so creates the same on destination DB if it doesn’t exist out there. It also creates the tables and can change the mapping if it requires customization based on tenant’s requirement.

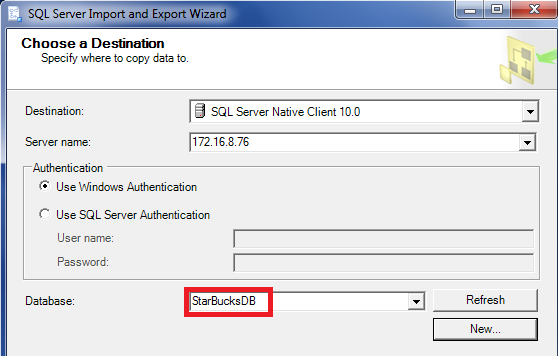
Step 1: Launch DTS wizard



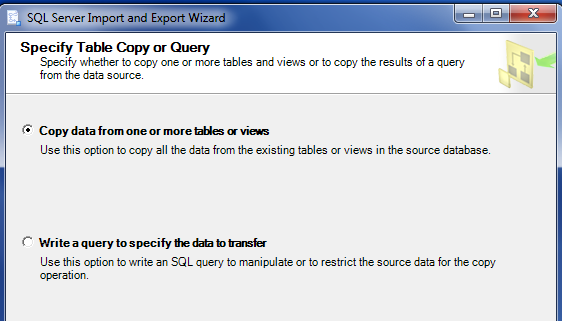
Step 2: Select source database and authentication method



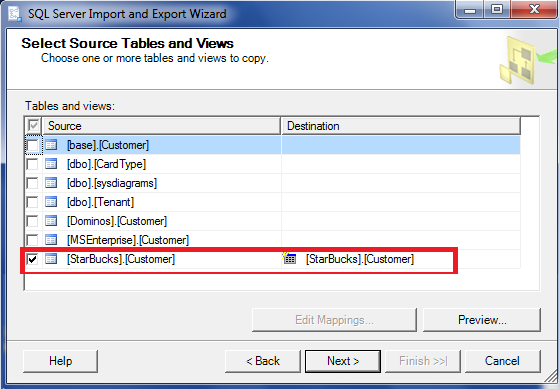
Step 3: Select the destination server and database. In this case, we are selecting a new database to be created for StarBucks



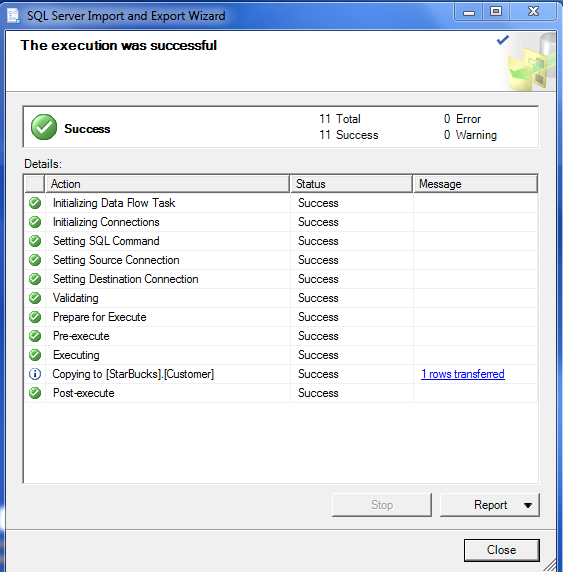
Step 4: Select the method of copy



Step 5: Select the tables to copy



Step 6: Continue on ‘Next’ till the success screen



## Task: Database Encryption

**Background**: If all tenant’s data is stored in a single table, identified using Tenant-id; any user who has access to the table can read any record. So each record needs to be encrypted using tenant’s own key.

**Description**: Each tenant has its private key using which the encryption of sensitive columns must happen so other no other user can access the data even if it reaches wrong hands

**Input**: Tenant’s schema name and tenant’s key (assume user-id for POC)

**Output**:

* Possibility POC
* Encryption methodology and possible algorithm choices to use.

**Processs**:

Study

### Encryption using TSQL (CRU-operations)

Specific column level encryption can be done using “passphrase”. For this, the underlying column has to be varbinary and hence no index can be setup on such a column.

We can use different passphrase on different sets, however the one used for reading has to be same as the one used for writing.

SQL function [**EncryptByPassPhrase**](http://technet.microsoft.com/en-us/library/ms190357(v=sql.105).aspx) and [**DecryptByPassPhrase**](http://technet.microsoft.com/en-us/library/ms188910(v=sql.105).aspx) are for this.

In our scenario, the passphrase will be the user-id. In actual scenario, this will be the tenant’s GUID, which itself will be encrypted at database level (hidden from all users altogether)

1. Add a varbinary column Name which will get encrypted

|  |
| --- |
| alter table [StarBucks].[Customer]  add Name varbinary(100) |

1. Insert a record without the Name column and get the primary key’s value

|  |
| --- |
| insert into [StarBucks].[Customer] (Id, cardTypeId, createdOn, updatedOn, createdBy, updatedBy)  values (newid(), 1, GETDATE(), GETDATE(), 13434, 13434 ) |

1. Now update this record, using passphrase

|  |
| --- |
| update [StarBucks].[Customer]  set Name = EncryptByPassPhrase(@passphrase, 'Rimba', 1, convert(varbinary, Id))  where id = '51F23C08-0205-4B9A-BD2E-4D7D673032AC' |

1. Do a select on this row, and see the column as encrypted

|  |
| --- |
| select id, Name from [StarBucks].[Customer]  where id = '51F23C08-0205-4B9A-BD2E-4D7D673032AC' |
|  |

1. Now use the passphrase to get the decoded value back. Since the actual column is in varbinary, we need to convert the column value back to varchar as well.

|  |
| --- |
| select convert(varchar,  DecryptByPassPhrase (@passphrase, Name, 1, convert(varbinary, Id)) )  from [StarBucks].[Customer]  where id = '51F23C08-0205-4B9A-BD2E-4D7D673032AC' |
|  |

### Encryption of Column (permanent)

* <http://technet.microsoft.com/en-us/library/ms179331.aspx>

### Encrypted Database Connections

* <http://technet.microsoft.com/en-us/library/ms191192.aspx>

### Real-time Encryption

* <http://technet.microsoft.com/en-us/library/bb934049.aspx>
* www.youtube.com/watch?v=oHmz5q0YtD8
* www.youtube.com/watch?v=TnFdfUl39Hs‎

### Other links

* sqlmag.com/database-security/sql-server-encryption-options
* <http://blog.sqlauthority.com/2009/04/28/sql-server-introduction-to-sql-server-encryption-and-symmetric-key-encryption-tutorial-with-script/>
* <http://www.mssqltips.com/sqlservertip/3081/using-views-to-expose-encrypted-data-in-sql-server/>
* <http://www.sqlperformance.com/2013/10/io-subsystem/native-backup-encryption>

### Third party components

* <http://www.vormetric.com/products/encryption/database-encryption/microsoft-sql-server-data-encryption>
* www.dbdefence.com

# Story: Patch management

**Description**: We can have following DB-Server situations:-

1. Single DB, shared schema (SDSS)
2. Single DB, individual schema (SDIS)
3. Individual DB (ID)

In such cases, if we have changes to application we need to patch existing deployments. For 2&3, any table changes will need to consider if tenant’s own structure has changed due to their specific business requirements. For 1, any DB patch will be made available to ALL the tenants.

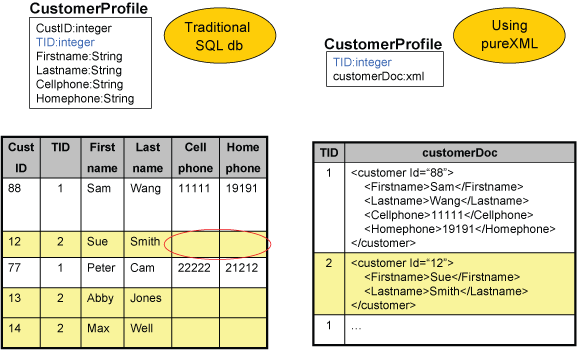
In such cases, we need to ensure that there is no breaking change. Any column removal will need to be certified through dependency check that the corresponding impacts are validated in corresponding stored procedures.

IMPORTANT: Any DB related changes have to be validated parallel in the ORM (if used). This presents a conflict or special case.

## General Design

(ref: [IBM Dev Works](http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/))

Customization for a given tenant can add application and administration complexity. This section describes how customization can be handled by using XML.



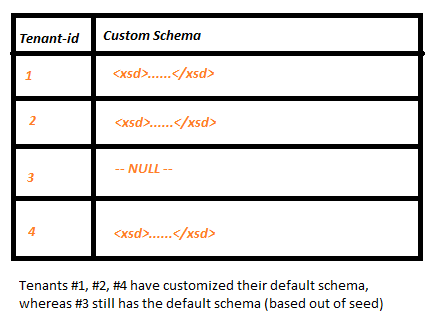
The figure shows that a tenant has many clients, and each client has a different profile. The tenant ID (TID) column in both tables is used to identify the tenant. Rows 1 and 3 in the left table belong to one tenant with a TID of 1; rows 2, 4, and 5 belong to another tenant with TID of 2.

Suppose tenant No. 2 (TID = 2) has a business rule that indicates that clients cannot input telephone information, so storing information about telephone numbers would not be applicable to this tenant. However, in a multi-tenant environment where tables are shared (case 1), the SaaS company needs to consider that other tenants do want to include telephone information. Using a traditional SQL database (left table), the SaaS vendor can create one fixed-column table, which includes columns for all possible cases of telephone numbers (cellphone number, home phone). Even if tenant No. 2 doesn't allow phone numbers in customer profiles, the column is still included. Therefore, there will be a lot of "holes" and dispersed data, as highlighted by the circles in Figure.

In addition, say tenant No. 1 (TID = 1) wants to change its requirements so it doesn't store only cell and home numbers but also work numbers. In this situation, you may have to alter the table. However, if you follow the normalization rules for database design, you actually need to create a separate PHONE table. Then you'd have to move the data and change your applications so your SQL queries point to the new table PHONE and use join operations. This method is not flexible.

On the right side of Figure is the suggested method of handling customizations. The table in this case has only two columns, where the second column is defined with the XML data type. Using XML, it is a lot more flexible to handle changes in the database schema.

In such design, the actual schema of xml will get stored as an xsd for the meta-data value



## POC Details

Environment

* Base customer schema available to all tenants
* XSD validation of new field
* Tenant application to process new fields along with existing

Things to do:

* How to retrieve different fields as stored in DB
* Entity level changes

Strategy:

* Store the entire set of custom fields in a dedicated xml column. This column will always be present for each transaction table.
* If tenant wants to add fields to existing set for a given entity, they need to define the schema mapping for that. This schema will get stored in tenant metadata as “XML Schema Collection”.
* Here all xsd validation can be enforced, like:-
  + Field name
  + Datatype
  + Required
  + Maximum items and minimum items (like for phone number)
* The XSD will be created per-tenant and per table basis
* Each stored procedure for Create / Update will have an input field for new fields mandatorily. Before inserting / updating such fields, the xml will get validated using the given schema
* If no customization has been done, both the schema and input xml will get passed as empty
* The XML schema once created can also be made to set as part of the column design or used at runtime while inserting or updating. This will depend on the DB/Schema strategy as below

Links:

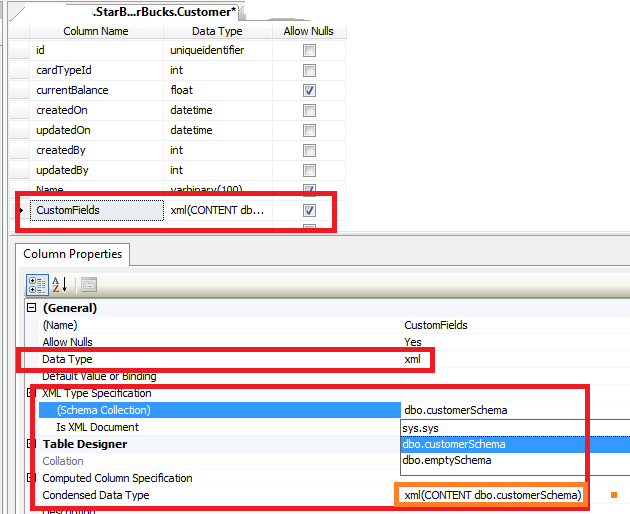
* <http://technet.microsoft.com/en-us/library/ms176009.aspx>
* <http://technet.microsoft.com/en-us/library/ms190665.aspx>
* <http://technet.microsoft.com/en-us/library/ms187856.aspx>

Create schema method

|  |
| --- |
| -- have a schema as xml (just to validate it)  -- the schema will have "null", "type", min,max etc  declare @customXsd xml  set @customXsd = '<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  <xsd:element name="Custom">  <xsd:complexType>  <xsd:sequence>  <xsd:element name="Age" nillable="true" type="xsd:int" />  <xsd:element name="State" nillable="true" type="xsd:string" />  </xsd:sequence>  </xsd:complexType>  </xsd:element>  </xsd:schema>'  create xml schema collection customerSchema as @customXsd |
| Empty schema  declare @emptyXsd xml  set @emptyXsd = '<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"></xsd:schema>'  create xml schema collection emptySchema as @emptyXsd |
| Validation using schema  declare @validatedField xml (customerSchema)  select @validatedField = @CustomFields  If schema matches, then the variable will have the resultant xml, else it will give errors |

## Task: Strategy for Shared DB, Individual Schema

Since we have individual tables per tenant (within the tenant’s schema), we can put the XML Schema as part of table design itself. This schema will be managed from Tenant’s own admin-console wherein they can update the Custom fields.



In this, we have done following changes to normal customer table:-

1. Added a CustomFields as XML column
2. Created schemas (one as empty and another customer specific). The empty schema will validated an empty field in a record, while *customerSchema* will validate new fields in customer entity which tenant requires; this can have appropriate nullability checks as well

## Task: Strategy for Shared DB, Shared Schema

If we are storing all tenant’s data in a single customer table, then we will need to create tenant’s specific XML schema with elements in custom namespaces created for the tenant.

We will create the schema once when tenant requests customization for the first time, post that we will “ALTER” that schema. The schema name will reflects the tenant’s identifier, as there is single schema in whole database.

Any corresponding stored procedures will require *regeneration* post any such 1st time creation, post that only on schema deletion. No SP change will be required for any schema changes.

|  |
| --- |
| -- have a schema as xml (just to validate it)  -- the schema will have "null", "type", min,max etc  declare @customXsd xml  set @customXsd = '<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  ……. [[[[ schema contents as above ]]]  </xsd:schema>'  create xml schema collection customerSchema\_tenant\_958 as @customXsd |
| create procedure UpdateCustomer (  @id uniqueidentifier,  @CustomFields xml = null  )  as  declare @validatedField xml (customerSchema\_tenant\_958)  select @validatedField = @CustomFields  update StarBucks.Customer  set CustomFields = @validatedField  where id = @id |
|  |

## Task: Strategy for Individual DB

An individual DB is equivalent to a logical DB in this case; hence we can set the XML schema to be part of table’s definition.

No specific consideration is needed.

## Task: Build Admin Console for customization of XML Schema

The admin for the site will need an interface to customize the custom fields, so the admin console has to provide that feature.

Following use cases will be handled in this

* Create new customization
* Update existing customization
* Delete existing customization

### Sub-Task: Create new customization

Steps:

The strategy to be used {SDSS, SDIS, ID} and tenant’s name will be system decided based on case to case basis.

1. User selects the table to have custom fields
2. User provides details of new fields { Name, datatype, nullable etc... }
3. Upon submit, the schema gets created in DB/tenant metadata table and corresponding SP’s are updated for this parameter
4. Post this, any insert/update of table uses this schema to validate the field’s values

### Sub-Task: Update existing customization

Steps:

The strategy to be used {SDSS, SDIS, ID} and tenant’s name will be system decided based on case to case basis.

1. User selects the table to have its customization modified
2. System returns existing custom field structure
3. User add/update fields and submits
4. The schema gets updated in DB/tenant metadata table. No SP needs any modification

### Sub-Task: Delete existing customization

Steps:

The strategy to be used {SDSS, SDIS, ID} and tenant’s name will be system decided based on case to case basis.

1. User selects the table to have its customization modified
2. User deletes the customization and submits
3. Upon submit, the schema gets deleted from DB/tenant metadata table and corresponding SP’s are updated for this parameter

# Logic Flow in MTA

