MySQL- or MariaDB-based Implementation of Bike Configurator Showcase

Sumit Ghanty, Rainer Brendle

The purpose of this this project is just establish for building application using Single-Page Web Applications techniques like SAPUI5 based on Service APIs. The purpose is also that you can have this without even have to have bought SAP already, but a customer may just consider at a future time. We do not talk loudly about using MySQL or MariaDB or similar open source-based approaches,] as the initial data storage engine, but eventually we can do the same on SAP Hana. The Service API, which connects to the database via a web server and HTTP on a REST protocol interface and JSON is the typical wire data encoding..

## Basic Assumptions

Every service API call is atomic, i.e. it is a single transaction. We have no auto-commit fro the web server to the database. Instead we have a clean transaction bracket around every service API call, which either completes successfully or rolls back completely on failure. In that sense any REST operation can be seen similar as a “stored procedure” in the database, which starts with

**START TRANSACTION;**

and ends with

**COMMIT;**

or

**ROLLBACK;**

A REST service API call is a message to be send to a resource, a resource, which is to be described via a URL. This also means that the transaction is limited to the resource and the addressed resource is forming a unit of integrity and atomicity. But this also means that relations between objects of different resources are to be described via like between URLs. i.e as unidirectional hyper links, not classical SQL-style bidirectional relations.

More complex transactions over several objects are managed thru a concept called **“eventual consistency”** then.

This gives another opportunity for simplicity. When SAP was created 3 decades go, customers needed to provide own customization codes for any attribute, which then ended up in quite complicated process of managing and maintaining a “golden tenant copy”, which actually was different at every installation creating some substantial complexity. On the other side e-Commerce has moved forward and today for almost any kind of value sets and master data, we have international standards and services well established. A today’s beginner would not start reinventing the wheel. A new beginner will start we ISO codes and standardized product and customer data instead. This gives us an interesting opportunity to encode data actually as URLs between objects for our demo. ISO 80000 for measures, GTIN for product data, country and currency codes (ISO 3166, ISO 4217), but also address validation services for customer data like Google Maps or also USPS in the US or the DUNS numbering system for businesses are a better starting point today, then what SAP had some decades ago. Since these codes started from EDI, they have values, which can be described similar as a URL when using some naming conventions. We can start from there and map to SAP later. And since we only do a demo, we can leave the complexity of old SAP installations out of scope.

## Unit Of Consistency

We can make the assumption that every business transaction, which is based on (stateless) services goes thru steps. which all can be represented via own URLs for the addressed resource.

A business transaction, which wants to create or change something, always starts with the filling out of a form. The data of the form form a document and then itself is a resource. The result of the form is not final, but actually is a work-in-progress. It becomes final, when successfully submitted.

In that sane the content of a form is described as a “case”, which is owned by one or several users. The form’s content describes the wish or intention of an object to be created or changed. The form’s content is an own URL, which can be accessed via GET and PUT services. The abstract data model of the content of a form can be described as following in a SQL schema description:

Let us assume we describe an business entity as a table:

CREATE TABLE

<entity>\_table(

<entity>\_id INT NOT NULL AUTO\_INCREMENT,

<attributes) ...,

PRIMARY KEY ( <entity>\_id )

);

or as ER Diagram - may be as a master-detail document, which fits most type of entities:

<Entity>Master

EntityID

Attributes

<Entity>Detail

EntityID

ItemNo

Attributes

Only on that level we have a ER diagram style of relationship. Between objects we have URLs instead. Most attributes must eventually be transaltable into URLs, text or measures.

## A Form to Create or Change Entities

In a stateless web-based transaction we need to have a form describing the a business transaction. a form, which has its own state. It is a form to ask for a change.

The form data to create or change the entries of the table can and should be very much the same, but with a primary key associated to some kind of “case”, which relates to a user eventually (or to many users actually with collaboration

).

(Question: should a user be able to have several entity forms in the same case - for simplicity probably not. So the case-to-form relationship is 1:1, probably, but the case may have several users associated. Basically we are adding the CaseID as an additional primary key,

Form<Entity>Master

EntityID

Attributes

Form<Entity>Detail

EntityID

ItemNo

Attributes

Case

CaseID

CaseID

CaseID

UserID

It depends of the nature of the action, which the form is describing, if the form can have the same attributes then the entity, where is acting on, i.e. its final state. If it is about setting values only, then the structure of entity and the form can be identical. If the action includes computations (like adding a value), then we need to change the structures to guarantee “idempotent” execution of the form against the real entity. In general however, you want to avoid this situation. Assignment is the most effective idempotent operation.

## Final Transaction State

We need to make a difference for the final transaction state, when the form is submitted. If it is just about the demo we can simulate a table like described. Talking to SAP then means to call a BAPI or Service from the SAP gateway or the frontend server of S4/Hana. Here the mapping has to happen. If we will have a new customer, the important step in a project then will mean to have value sets and master data set up in a way that it corresponds to eCommerce standards.

For connecting to SAP data here we basically need to map keys and we need to handle a mapping to the ENQUEUE locking concepts of SAP, which is easy, if you already have a persistent document for the form. It basically means just to repeat the POST action on failure, since the POST action has no parameters then and so is idempotent.

## Historic Data

We also at some point in time we should be able to show that we can handle historic data the same. The best way to handle historic data, is basically to archive them in an insert-only manner, i.e. we keep copies of immutable documents and arrange them over creation time. We arrange them as documents (i.e. as master-detail-data structures) with related data, which may be able to change already embedded and copied with (typical example: the customer address data in an order. the archive needs to keep it together with the order, we can denormalize historic immutable data). This is only necessary for data which can change over time, but not for data, which are stable like attachments, but also GTIN data or ISO codes are stable. To do so we need to attach a time stamp to every object into the primary key, the same as we did it above with the CaseID, but now with time.

The most appropriate way on an ANSI SQL database would be to use a SEQUENCE as a kind of a “wall clock” and so create, what is called a “Lamport Time”. This construct does not exist in MySQL or MariaDB (they exist in PostgreSQL and SQL Server, why Amazon uses it in Amazon Redshift, a PostgreSQL clone, and Microsoft uses it in their Azure Cosmos DB).

A similar construct is an auto increment column, which exists in MySQL. An auto-increment column however needs an insert-operation to trigger it, so we can only assign it per table, but not globally. For creating the idea it may be sufficient. We cannot solve all problems from the beginning. You probably need to go for either real physical time stamps additionally or for a vector time data structure, there are many options, when on MariaDB or MySQL.

## Designing The Entities

When we design the entities, what we need from SAP area basically the basic entity names and the core business processes, which for manufacturing clearly is going from material catalog to BOM to Quote and to Order. Let us use standard eCommerce attributes like ISO codes and GTIN identifiers for this, then the data structures are pretty simple. A BOM is a list of materials with measures, a quote is a BOM with added prices and customer data, an order is the same as a a quote from a data structure perspective and we only have to care about this, when we talk about service APIs and not the inner workings.

MATERIAL

GTIN Number in various formats.

BOM

Master: Name, string

Detail:

GTIN Number : string

CodeOfMeasure: ISO 80000, string

Measure : value, integer or double

QUOTE

Master : Name, string

CustomerID

Name and Address, string

Detail:

Like BOM Detail, plus Prices (CurrencyCode, Value)

ORDER Like Quote

You will add some status information like “APPROVED”, “SEND”, “PAID” etc.

And you will add some standard information about time: “CREATED”, “CHANGED” , “READ BY for monitoring” etc. basically to everything.

PS: There is no need for absolute perfection, we just talk about a demo.

## 