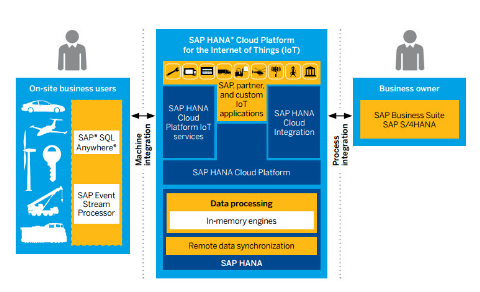
IoT1C05 – Create Database – On Premise

|  |  |
| --- | --- |
| **Product**  HANA Cloud Platform IoT Services  **Level**  Undergraduate/Graduate  Beginner  **Focus**  HANA Cloud Platform  **Author** Ross Hightower | MOTIVATION  In this case you will create the database and services for your IoT data.  **PREREQUISITES**  None |



# Create the Database

In order to implement IoT on HANA we need to create:

1. Database tables to save the data,
2. A database sequence to create a primary key,
3. A procedure to handle data creation in order to invoke the sequence
4. OData services to expose the data

We will use HANA Core Data Services (CDS) to create the database artefacts. CDS is an infrastructure that allows you to model database artefacts using a structured language in text files.

## Create the Tables

Log on to the HANA Web-based Development Workbench. Right-click your package and choose **New→Package** to create a package called **iot** in your package and then create a package called **data** in the iot package.

Right-click the data package and create a new file called **iot.hdbdd.** Paste the code shown below into the file.

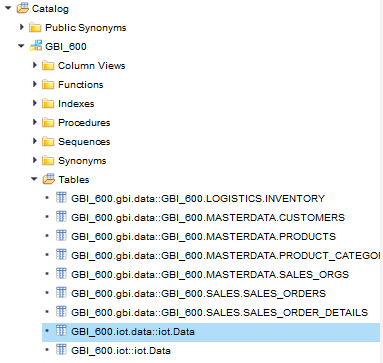
|  |
| --- |
| namespace **"gbi-student-366** **"**.iot.data;  @Schema: **'GBI\_366'**  context iot {  type SDate : UTCTimestamp;  type tt\_error {  HTTP\_STATUS\_CODE: Integer;  ERROR\_MESSAGE: String(100);  DETAIL: String(200);  };    type tt\_details {  ID: Integer;  TIMESTAMP: SDate;  TEMPERATURE: Decimal(9,5);  HUMIDITY: Double;  };  @Catalog.tableType : #COLUMN  Entity Data {  key ID: Integer;  TIMESTAMP: SDate;  TEMPERATURE: Decimal(9,5);  HUMIDITY: Double;  };  }; |

Listing 1

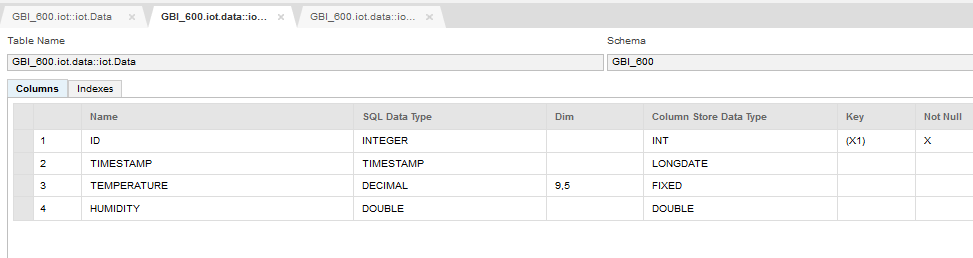
Update the highlighted code to reflect your schema and package. The namespace is the path through the packages to this file. The schema is the database schema created for your user id. The type statements create some data types that are used in the table definition.

The final block of code which begins @Catalog creates a column oriented table called Data. When you save the file, the file is saved and the artefacts are activated in your schema. You can check for errors in the console located below the editor.

If you activation is error free click on the Navigation Links drop-down in the toolbar to open the Catalog editor. Drill into the Tables folder of your schema to find the table.



Double-click the table to open its structure.



## Create a Sequence

In order to fill the ID field, which is the primary key, we need to create a sequence to generate unique values. Create a new file in the data package called iot\_id\_seq.hdbsequence. Insert the code shown below:

|  |
| --- |
| schema="**GBI\_366**";  start\_with=100;  minvalue=1;  cycles=false;  depends\_on\_table="**gbi-student-366**.iot.data::iot.Data"; |

Listing 2

Update the highlighted code. This will create a unique value starting with 100 each time a record is inserted into the GBI\_600.iot.data::iot.Data table. Save the table and make sure it’s activated without errors.

## Crate a Procedure

We can insert records into database tables with a simple OData service but in order to use the sequence we have to create a procedure written in SQLScript. Create a file called newdata.hdbprocedure and paste the code shown below into it.

|  |
| --- |
| PROCEDURE "**GBI\_366**"."**gbi-student-366**.iot.data::newdata" (  IN row "**gbi-student-366**.iot.data::iot.tt\_details",  OUT error "**gbi-student-366**.iot.data::iot.tt\_error" )  LANGUAGE SQLSCRIPT  SQL SECURITY INVOKER  DEFAULT SCHEMA **GBI\_366**  AS  BEGIN  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Write your procedure logic  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  declare lv\_temperature string;  declare lv\_humidity string;  select TEMPERATURE, HUMIDITY into lv\_temperature, lv\_humidity from :row;  if :lv\_temperature = ' ' then  error = select 400 as http\_status\_code,  'invalid date' as error\_message,  'Invalid response from sensor' as detail from dummy;  else  insert into "**gbi-student-366**.iot.data::iot.Data" values ("**gbi-student-366**.iot.data::iot\_id\_seq".NEXTVAL, now(), CAST(lv\_temperature AS decimal(9,5)), CAST(lv\_humidity AS double) );  end if;  END; |

Listing 3

Update the highlighted portions of the code. This code uses the two table types created in the .hdbdd file as input and error structures. It will receive data from the OData service created in the next section and execute the insert SQL statement to insert the record into the table. Note the use of the sequence to generate a value for the ID field. Save the procedure and ensure it is activated without errors.

## Create the OData Service

Create a file called **iot.xsodata** in the iot package. Copy the code below into it.

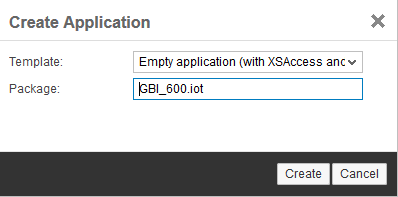
|  |
| --- |
| service namespace "GBI\_366.iot" {    "gbi-student-366.iot.data::iot.Data" as "DATA"  create using "gbi-student-366.iot.data::newdata";  } |

Listing 4

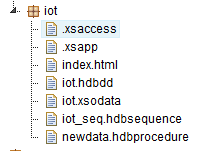
This creates to OData services. The first one has an endpoint called HISTORY and can be used to read the data in the iot\_Data table. The second one will be used to create new records and invokes the procedure created in the previous section.

## Create an Application

Before we can access the services remotely we have to create an application in the iot package. Right-click the iot package and select Create Application.



Click Create. The final iot package looks like this.



To allow cross-site scripting, click the .xsaccess file and change the value for **prevent\_xsrf** from true to false.



With the .xsodata file open in the editor, click the run icon and the service document is shown.



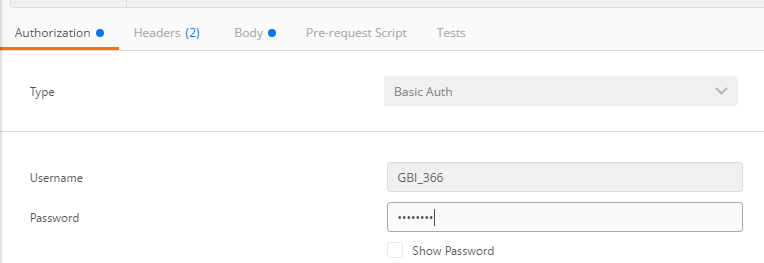
## Test the Service

To test the service, you will need to install a RESTful client like Postman. Postman is an extension you install in Chrome. Change the HTTP method to POST and enter the URL as shown:

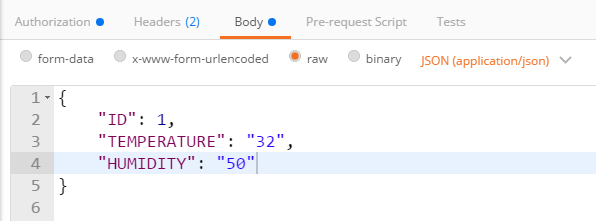


The URL is the same one you get when you run the iot.xsodata file with the addition of /DATA to the end.

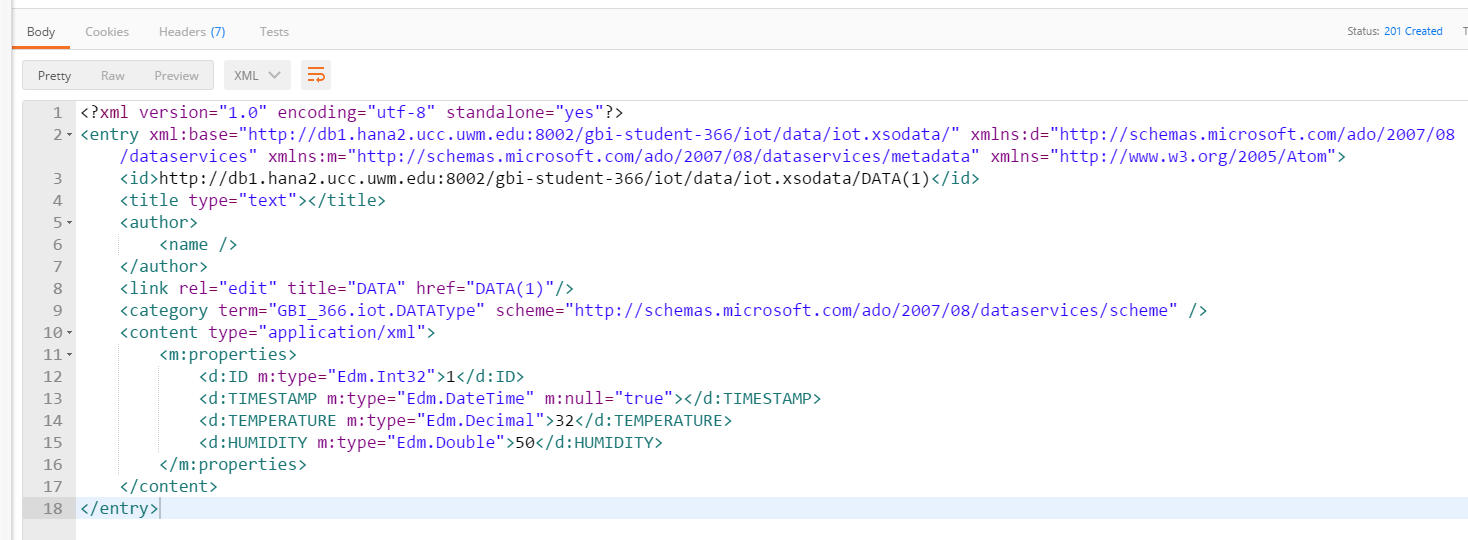
On the Authorization tab, choose Basic Authentication and enter your user id and password.



On the Body tab, enter a message in the format shown:



Click Send and the result is shown. If the service worked you should receive a 201 status and the record created in XML format.



To view the data in the table, open the table in the Catalog editor and view the content.

