13-06-2019

HANA (High Performance Analytic Appliance)

ABAP HANA course --> Designed for ABAP consultants

S/4 HANA → Mainly for Functional consultants

→SAP Business suite for Hana

→ Next generation ERP

Evolution of S/4 Hana:

R/2 ---> R/3 ---> ERP ----> S/4 HANA

1970's ---> 1992 ---> 2004 ---> 2015

Features of S/4 Hana:

1. **Fast in-memory database:** This Technology allows the processing of massive amounts of real time data in a short time. The in-memory Computing engine allows HANA to process data stored in RAM as opposed to reading it from disk.

2. **Supports both Row storage and Column storage tables**

3. **Beautiful user experience (SAP FIORI Screens --> ui5 framework)**

--> can be runned on desktop, tablet & smartphone

4. **Supports 10 LOB's (Lines of Business)**

--> s/4 hana hr,s/4 hana supply chain, s/4 hana manufacturing, s/4 hana sales, s/4 hana sourcing and procurement, s/4 R&D, s/4 hana asset management, s/4 hana service, s/4 hana marketing & commerce

5. **Can be Setup on cloud & on premise**

6. **More simplicity**

Eg: Inventory management --->

ECC --> 26 tables (mstq,mste,mstb,mstqh,msteh,....

S/4 HANA --> only 1 table →'MATDOC' (Material documents)

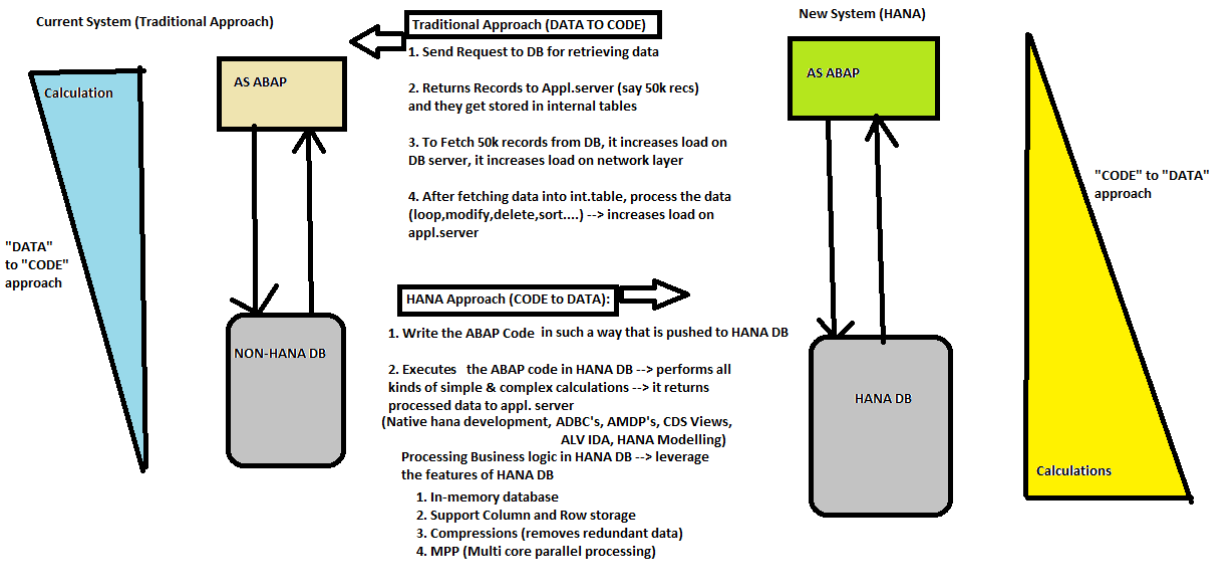
Pre-requisites for ABAP Hana: BASIC ABAP + OOPS Concepts

Traditional ABAP Report development: "DATA" TO "CODE" APPROACH

ABAP HANA Report development: "CODE" TO "DATA" APPROACH

ABAP HANA --> summary of topics covered:

1. Native HANA development
2. ADBC's
3. AMDP's
4. CDS Views
5. ALV IDA
6. HANA Modelling



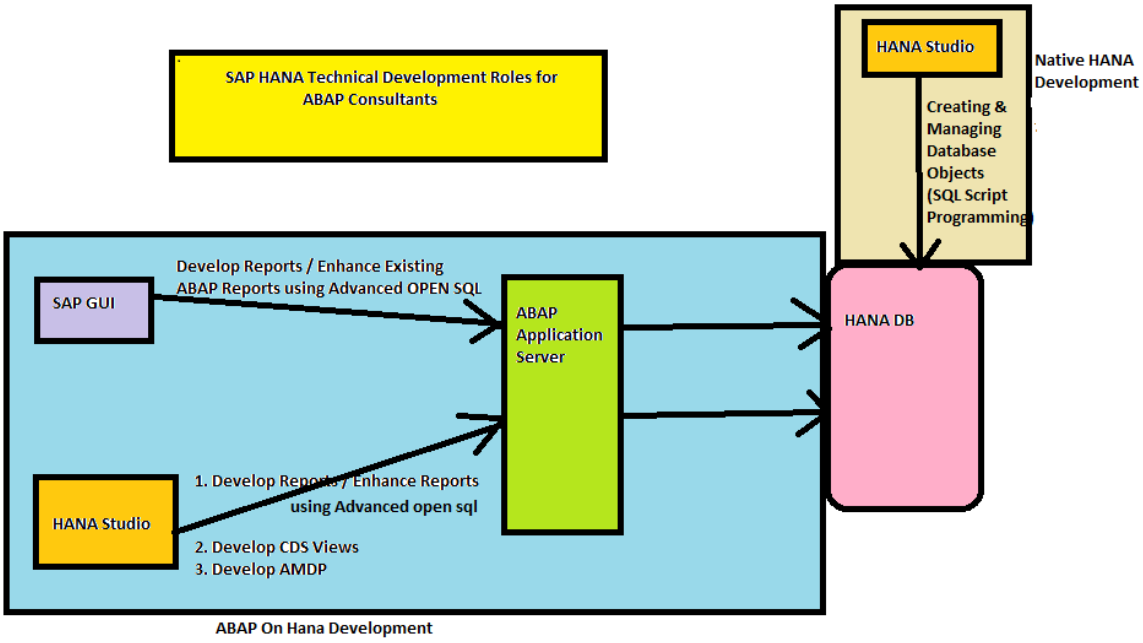
14.06.2019

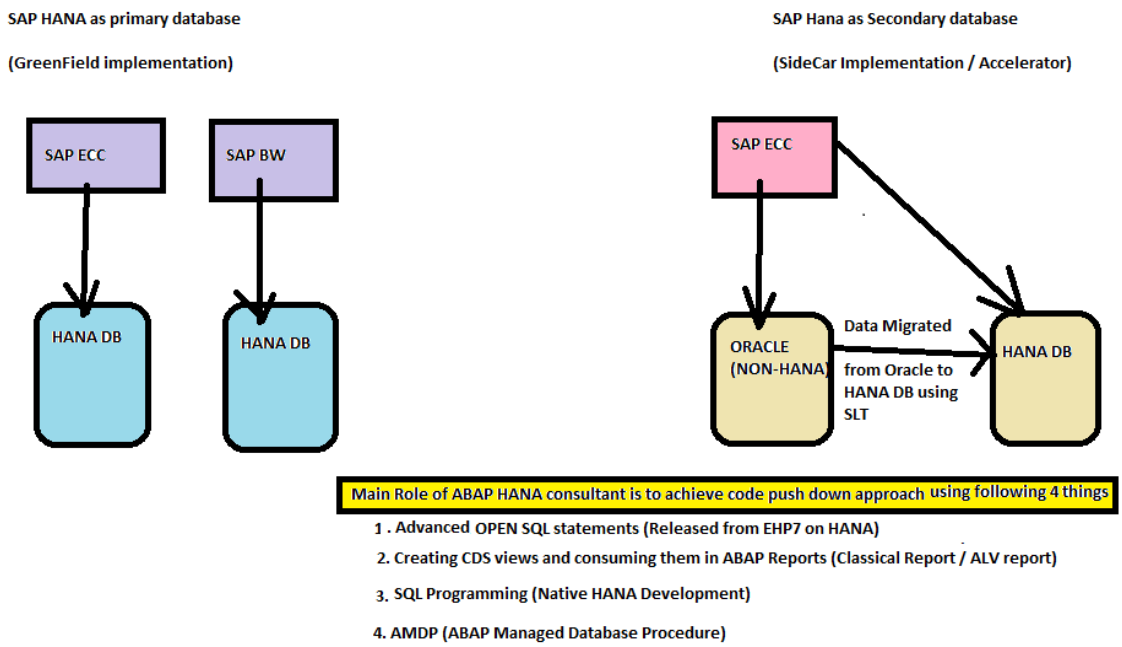
Example: "DATA to CODE" approach vs "CODE to DATA" approach

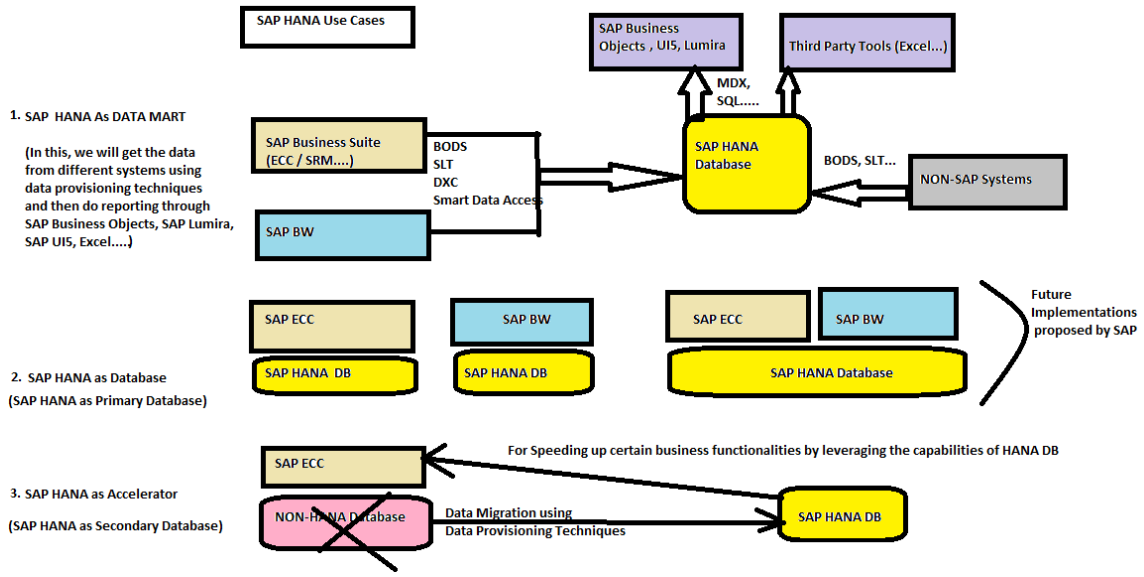
```
REPORT zcode_to_data.

** Data to code approach
*types : begin of ty_vbap,
*            vbeln type vbap-vbeln,
*            posnr type vbap-posnr,
*            matnr type vbap-matnr,
*            netpr type vbap-netpr,
*        end of ty_vbap.
*
*data : t_vbap type table of ty_vbap,
```

```
*       wa_vbap type ty_vbap.
*
*types : begin of ty_final,
*            vbeln type vbap-vbeln,
*            posnr type vbap-posnr,
*            matnr type vbap-matnr,
*            netpr type vbap-netpr,
*            gross type vbap-netpr,
*        end of ty_final.
*
*data : t_final type table of ty_final,
*       wa_final type ty_final.
*
*select vbeln posnr matnr netpr
*       from vbap
*        into table t_vbap
*        where vbeln in ('0000004980','0000004981','0000004982').
*if sy-subrc eq 0.
* loop at t_vbap into wa_vbap.
*      clear wa_final.
*      MOVE-CORRESPONDING wa_vbap to wa_final.
*      wa_final-gross = wa_final-netpr + 100.
*      append wa_final to t_final.
* endloop.
* endif.
*
* if t_final is not initial.
* loop at t_final into wa_final.
*    write :/ wa_final-vbeln,
*             wa_final-posnr,
*             wa_final-matnr,
*             wa_final-netpr,
*             wa_final-gross.
*  endloop.
*else.
*write :/ 'No data'.
*endif.

** Code to Data approach
select vbeln, posnr as itemno, matnr, netpr, ( netpr + 100 ) as
grossamt
   into table @DATA(t_vbap) from vbap
       where vbeln in ( '0000004980','0000004981','0000004982' ).
if sy-subrc eq 0.
  data wa like line of t_vbap.
  loop at t_vbap into wa.
```

```
      write :/ wa-vbeln,
               wa-itemno,
               wa-matnr,
               wa-netpr,
               wa-grossamt.
  endloop.
else.
write :/ 'No data'.
  endif.
```







17.06.2019

HANA Studio: is a tool used for connecting to HANA Database and developing the objects related to HANA DB. This tool is Eclipse BASED tool installed with HANA Plug-in's and SAP renamed it as HANA Studio. As part of this tool, we can use different perspectives for developing applications related to different languages. For HANA Development, SAP has provided different perspectives. Each perspective has its own design view. Following are some of the perspectives related to HANA Development.

1. **HANA Development Perspective**: Used for Native HANA development for designing HANA Database objects and applications specific to HANA

2. **Modeler Perspective**: Used for development of Modeling objects like Attribute View, Analytical View and Calculation View

3. **ABAP Perspective**: used for developing ABAP repository objects and also objects related to HANA like CDS views and AMDP's

4. **Debugger Perspective**: Used for debugging stored procedures and other blocks.

SQL (Standard/Structured Query Language): It is the Standard language to communicate with databases. As Part of HANA Database, We can create and manage different types of database objects and for this we use SQL Script to communicate with HANA DB

Database Objects:

1. Schema
2. Tables
3. Database Views
4. Triggers
5. Sequences
6. Synonyms
7. Indexes
8. Column Views
9. Procedures
10. Functions

For Creating and Managing above Database Objects, Open SQL Console (Right Click on Catalog → Open SQL Console)

Schema: is a logical container of Database Objects. A default schema will be created on behalf of the logged in user with the same name as that of user name. All the System schemas start with '_sys'.

All Database Objects will be qualified (prefixed) by Schema Name.It is recommended to specify Schema name and Object name in double quotes ("").

Table: is a collection of rows and columns

Creating the Table and Inserting the data into the table

```
CREATE TABLE "GENSOFTTEST"
(
```

```
   "EMPID" INTEGER,
   "ENAME" VARCHAR(20)
);  →Creates a Column Store Table in the default schema (Logged in User Schema)

CREATE ROW TABLE "GENSOFTTEST1"
(
   "EMPID" INTEGER,
   "ENAME" VARCHAR(20)
);  →Creates a Row Store Table in the default schema (Logged in User Schema)

CREATE SCHEMA "GENSOFTSQL"; → Creates a New Schema and the owner is Logged in User

CREATE TABLE "GENSOFTSQL"."EMP1"
(
   "EMPNO" INT,
   "ENAME" VARCHAR(20)
);   →Creates a Column Store Table in the schema "GensoftSQL"


INSERT INTO "GENSOFTSQL"."EMP1" VALUES(1,'RAJU');
INSERT INTO "GENSOFTSQL"."EMP1" VALUES(2,'RAMESH');
INSERT INTO "GENSOFTSQL"."EMP1" VALUES(3,'KIRAN');

INSERT INTO "GENSOFTSQL"."EMP1" VALUES(4); → Error, as enough values are not provided

INSERT INTO "GENSOFTSQL"."EMP1"(EMPNO) VALUES(4); → success

CREATE TABLE "GENSOFTSQL"."EMP2"
(
   "EMPNO" INT NOT NULL,
   "ENAME" VARCHAR(20)
);   →Creates a Column Store Table in the schema "GensoftSQL" with not null constraint on EMPNO

INSERT INTO "GENSOFTSQL"."EMP2" VALUES(1,'RAJU');
INSERT INTO "GENSOFTSQL"."EMP2" VALUES(2,'RAMESH');
INSERT INTO "GENSOFTSQL"."EMP2" VALUES(3,'KIRAN');

INSERT INTO "GENSOFTSQL"."EMP2" VALUES(2,'PAVAN'); → Success, because Not null column can contain Duplicate Values
```

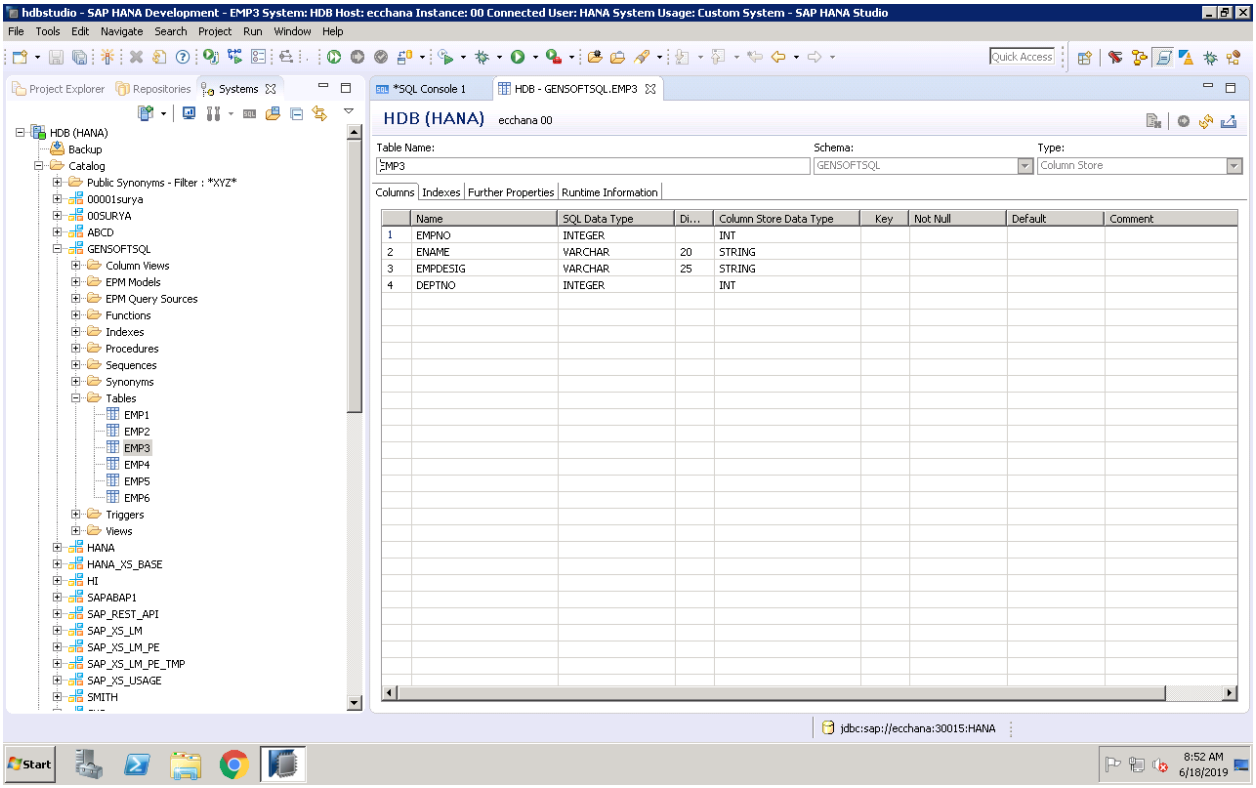
INSERT INTO "GENSOFTSQL"."EMP2"(ENAME) VALUES('SRINIVAS'); → **Error, because Not null column cannot contain Null Values**

18.06.2019

SET SCHEMA "GENSOFTSQL"; → Sets the Schema for the Current Schema, So that we need not prefix the database object with the schema name

Creating the Table Using UI:

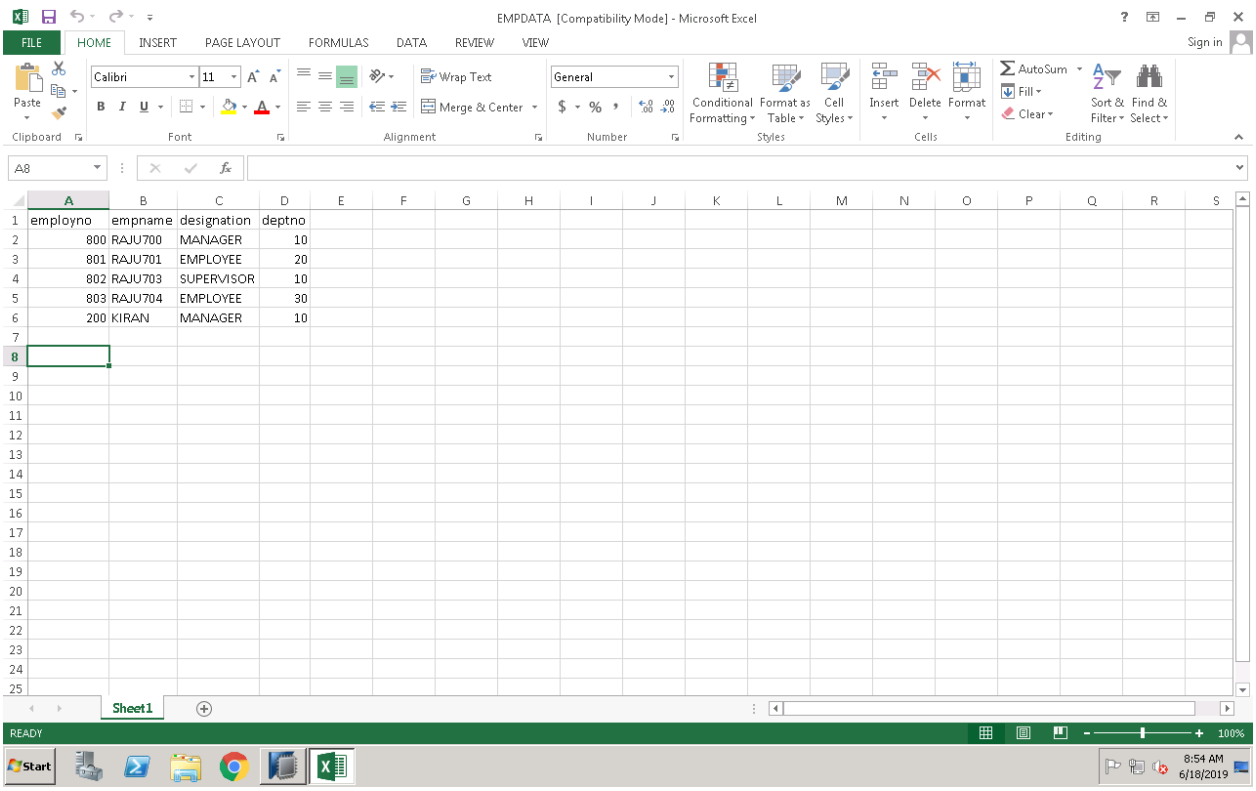
Right Click on Tables → New Table → Provide Table name, type (Row/Column) and provide column names with corresponding datatype



Click on Execute(f8) for creating the table

Uploading the data from .CSV file to the above DB table

1. **Consider a .XLS file created in Excel (empdata.xls)**



employno	empname	designation	deptno
800	RAJU700	MANAGER	10
801	RAJU701	EMPLOYEE	20
802	RAJU703	SUPERVISOR	10
803	RAJU704	EMPLOYEE	30
200	KIRAN	MANAGER	10

2. In HANA Studio, choose the menu 'File' → Import → SAP HANA Content , Data from local file, browse for the file and provide the file properties and also choose the target schema table where the data needs to be uploaded

```
CREATE TABLE "EMP4"
(
   "EMPNO" INT PRIMARY KEY,
   "ENAME" VARCHAR(20)
);   → Creates the Table with Primary Key Constraint on EMPNO column

INSERT INTO "EMP4" VALUES(1,'KIRAN1');
INSERT INTO "EMP4" VALUES(2,'KIRAN2');
INSERT INTO "EMP4" VALUES(3,'KIRAN3');
INSERT INTO "EMP4" VALUES(4,'KIRAN4');
INSERT INTO "EMP4" VALUES(5,'KIRAN5');

SELECT * FROM "EMP4";

INSERT INTO "EMP4" VALUES(4,'KIRAN44'); → Error, as duplicate entry
cannot be inserted for EMPNO because of primary key
```

```
INSERT INTO "EMP4"("ENAME") VALUES('KIRAN44');→ Error, as null value
cannot be inserted for EMPNO because of primary key


SELECT * FROM "SYS"."CONSTRAINTS" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'EMP4'; → Displays the constraints imposed on the table

ALTER TABLE "EMP4" DROP CONSTRAINT "_SYS_TREE_CS_#4426553_#0_#P0"; →
Deletes the constraint from the table

SELECT * FROM "SYS"."CONSTRAINTS" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'EMP4';

INSERT INTO "EMP4" VALUES(4,'KIRAN44'); → Success, as no primary key
INSERT INTO "EMP4"("ENAME") VALUES('KIRAN55'); → Success, as no
primary key

ALTER TABLE "EMP4" ADD CONSTRAINT "EMPNO_PRIKEY" PRIMARY KEY("EMPNO");
→ Error in Creating the primary key constraint on empno column as
duplicate entry already exists

DELETE FROM "EMP4" WHERE "ENAME" = 'KIRAN44'; → deletes the record

ALTER TABLE "EMP4" ADD CONSTRAINT "EMPNO_PRIKEY" PRIMARY KEY("EMPNO");
Creates the primary key constraint on empno column with user defined
constraint name


SELECT * FROM "SYS"."CONSTRAINTS" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'EMP4';

CREATE COLUMN TABLE "EMP5" LIKE "EMP4"; → Creates a new table based on
existing table without copying the data

SELECT * FROM "EMP5";

SELECT * FROM "SYS"."CONSTRAINTS" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'EMP5';

CREATE COLUMN TABLE "EMP6" LIKE "EMP4" WITH DATA; → Creates a new
table based on existing table along with the data

SELECT * FROM "SYS"."CONSTRAINTS" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'EMP6';

SELECT * FROM "EMP6";
```

```
UPDATE "EMP6" SET "ENAME" = 'KIRAN55'; → Updates all records

SELECT * FROM "EMP6";

UPDATE "EMP6" SET "ENAME" = 'KIRAN33' WHERE "EMPNO" = 3; → Updates
specific record

SELECT * FROM "EMP6";
```

19.06.2019

FOREIGN KEY CONSTRAINT:

We create foreign key relationship between two tables to maintain data consistency between the tables. The pre-requisite for creating foreign key relationship is, the two tables should have at-least one logically related field with same data type (field name need not be same). In one of the table, the logically related field should be primary key and this table is considered as parent table and on the other table we create foreign key on logically related field and this table is considered as child table (foreign key table). While defining the foreign key constraint, we can use the additions 'ON DELETE CASCADE' (or) 'ON DELETE RESTRICT'. If no addition is used, the default addition is 'ON DELETE RESTRICT'. Once foreign key relationship is created between two tables, System maintains **Referential Integrity.** According to Referential Integrity, it ensures following things:

1. In case of **'ON DELETE CASCADE'**, if a record is deleted from parent table then the corresponding dependent records will be automatically deleted from child table (Foreign key table)

2. In case of **'ON DELETE RESTRICT'**, if we try to delete a record from parent table then the system checks if any corresponding dependent records are available in child table, if available, it will not allow to delete the record from the parent table.

```
SET SCHEMA "GENSOFTSQL";  //SETS SCHEMA GENSOFTSQL FOR THE WHOLE
SESSION

CREATE COLUMN TABLE "DEPT"
(
   "DEPTNO" INT PRIMARY KEY,
   "DNAME" VARCHAR(20)
);   // PARENT TABLE
```

```sql
CREATE COLUMN TABLE "EMP"
(
   "EMPNO" INT PRIMARY KEY,
   "ENAME" VARCHAR(20),
   "DNO" INT,
   FOREIGN KEY("DNO") REFERENCES "DEPT"("DEPTNO") ON DELETE CASCADE
);   //FOREIGN KEY TABLE / CHILD TABLE

INSERT INTO "DEPT" VALUES(10,'ORACLE');
INSERT INTO "DEPT" VALUES(20,'SAP');
INSERT INTO "DEPT" VALUES(30,'JAVA');
INSERT INTO "DEPT" VALUES(40,'SALESFORCE');

INSERT INTO "EMP" VALUES(1,'RAVI1',50);  // ERROR-NO CORRESPONDING
PARENT RECORD IN DEPT TABLE

INSERT INTO "EMP" VALUES(1,'RAVI1',10);
INSERT INTO "EMP" VALUES(2,'RAVI2',20);
INSERT INTO "EMP" VALUES(3,'RAVI3',30);
INSERT INTO "EMP" VALUES(4,'RAVI4',10);
INSERT INTO "EMP" VALUES(5,'RAVI5',20);
INSERT INTO "EMP" VALUES(6,'RAVI6',30);
INSERT INTO "EMP" VALUES(7,'RAVI7',40);

DELETE FROM "DEPT" WHERE "DEPTNO" = 20; //ALSO DELETES DEPENDENT
RECORDS IN FOREIGN KEY TABLE-EMP

SELECT * FROM "SYS"."CONSTRAINTS" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'EMP'; // DISPLAYS PRIMARY KEY AND UNIQUE CONSTRAINTS

SELECT * FROM "SYS"."REFERENTIAL_CONSTRAINTS" WHERE SCHEMA_NAME =
'GENSOFTSQL' AND TABLE_NAME = 'EMP'; // DISPLAYS FOREIGN KEY
CONSTRAINTS

ALTER TABLE "EMP" DROP CONSTRAINT
"_SYS_CONSTRAINT_4436088_#0_#F0"; //DROPS FOREIGN KEY CONSTRAINT

SELECT * FROM "SYS"."REFERENTIAL_CONSTRAINTS" WHERE SCHEMA_NAME =
'GENSOFTSQL' AND TABLE_NAME = 'EMP';

DELETE FROM "DEPT" WHERE "DEPTNO" = 10; //DELETES THE DATA ONLY FROM
DEPT TABLE

ALTER TABLE "EMP" ADD CONSTRAINT FKEY_EMP_DEPT FOREIGN KEY("DNO")
REFERENCES "DEPT"("DEPTNO"); //ERROR-REFERNTIAL INTEGRITY VIOLATION AS
NO PARENT ROW WITH DEPTNO 10 IN DEPT TABLE
```

```sql
DELETE FROM "EMP" WHERE "DNO" = 10; // DELETES DEPTNO 10 FROM DEPT
TABLE

ALTER TABLE "EMP" ADD CONSTRAINT FKEY_EMP_DEPT FOREIGN KEY("DNO")
REFERENCES "DEPT"("DEPTNO"); // CREATES FOREIGN KEY

SELECT * FROM "SYS"."REFERENTIAL_CONSTRAINTS" WHERE SCHEMA_NAME =
'GENSOFTSQL' AND TABLE_NAME = 'EMP';

DELETE FROM "DEPT" WHERE "DEPTNO" = 30; //ERROR-RESTRICTS THE DELETION
FROM DEPT TABLE BECAUSE DEPENDENT CHILD ROWS FOUND BECAUSE DEFAULT
ADDITION FOR FOREIGN KEY IS ON DELETE RESTRICT

ALTER TABLE "EMP" DROP CONSTRAINT "FKEY_EMP_DEPT"; // DELETES THE
FOREIGN KEY CONSTRAINT

SELECT * FROM "SYS"."REFERENTIAL_CONSTRAINTS" WHERE SCHEMA_NAME =
'GENSOFTSQL' AND TABLE_NAME = 'EMP';

DELETE FROM "DEPT" WHERE "DEPTNO" = 30; //DELETES THE RECORD FROM DEPT
TABLE AS THERE IS NO FOREIGN KEY LINK

ALTER TABLE "EMP" ADD CONSTRAINT FKEY_EMP_DEPT FOREIGN KEY("DNO")
REFERENCES "DEPT"("DEPTNO"); // ERROR IN CREATING FOREIGN KEY BECAUSE
DEPTNO 30 IS NOT FOUND IN PARENT TABLE

DELETE FROM "EMP" WHERE "DNO" = 30; // DELETES DNO 30

ALTER TABLE "EMP" ADD CONSTRAINT FKEY_EMP_DEPT FOREIGN KEY("DNO")
REFERENCES "DEPT"("DEPTNO"); // CREATES FOREIGN KEY

SELECT * FROM "SYS"."REFERENTIAL_CONSTRAINTS" WHERE SCHEMA_NAME =
'GENSOFTSQL' AND TABLE_NAME = 'EMP';

DELETE FROM "EMP" WHERE "DNO" = 40; //DELETES DNO 40 ONLY FROM EMP
TABLE AND NOT FROM DEPT TABLE

ALTER TABLE "EMP" DROP CONSTRAINT FKEY_EMP_DEPT;

SELECT * FROM "SYS"."CONSTRAINTS" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'DEPT';

SELECT * FROM "SYS"."CONSTRAINTS" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'EMP';
```

```
ALTER TABLE "DEPT" DROP CONSTRAINT "_SYS_TREE_CS_#4436081_#0_#P0"; //
DELETES ONLY UNIQUE CONSTRATINT FEATURE OF THE PRIMARY KEY COLUMN AND
NOT THE NOT NULL FEATURE

ALTER TABLE "EMP" DROP PRIMARY KEY; // DELETES BOTH UNIQUE CONSTRATINT
FEATURE AND NOT NULL FEATURE OF THE PRIMARY KEY COLUMN
```

20 june 2019.

```
SET SCHEMA "GENSOFTSQL";

CREATE COLUMN TABLE "EMP"
(
    "EMPNO" INT PRIMARY KEY,
    "ENAME" VARCHAR(20)
);

INSERT INTO "EMP" VALUES(1,'RAJ1');
INSERT INTO "EMP" VALUES(2,'RAJ2');
INSERT INTO "EMP" VALUES(3,'RAJ3');
INSERT INTO "EMP" VALUES(4,'RAJ4');
INSERT INTO "EMP" VALUES(5,'RAJ5');
INSERT INTO "EMP" VALUES(6,'RAJ6');
INSERT INTO "EMP" VALUES(6,'RAJ66'); //ERROR-UNIQUE CONSTRAINT
VIOLATED FOR EMPNO

UPSERT "EMP" VALUES(7,'RAJ7'); //SYNTAX ERROR

UPSERT "EMP" VALUES(7,'RAJ7') WHERE "EMPNO" = 7; //INSERTS NEW RECORD

UPSERT "EMP" VALUES(4,'RAJ44') WHERE "EMPNO" = 4; //UPDATES EXISTING
RECORD
```

Note: Insert always inserts new record if the primary key field value doesn’t exist otherwise generates error if it is already existing whereas UPSERT either inserts or updates the record based on the primary key field value existence compared in where clause.

COMMENTS: are used for providing meaningful descriptions to the table and table columns

```
COMMENT ON TABLE "EMP" IS 'Employee Details'; // PROVIDES COMMENT FOR
THE TABLE IN FURTHER PROPERTIES TAB
```

```
COMMENT ON COLUMN "EMP"."EMPNO" IS 'Employee Number'; //PROVIDES
COMMENT FOR TABLE COLUMN
COMMENT ON COLUMN "EMP"."ENAME" IS 'Employee Name';

UPDATE "EMP" SET ENAME = 'RAJ66' WHERE "EMPNO" = 6; //UPDATES EXISTING
RECORD

UPDATE "EMP" SET ENAME = 'RAJ55' WHERE "EMPNO" = 5; //UPDATES EXISTING
RECORD
```

TRIGGERS:

It is event based database object used for performing follow-up actions. Triggers cannot be created using UI, It should be created only using SQL Editor. It can also be used to maintain referential integrity between the tables.

```
CREATE COLUMN TABLE "AUDIT"
(
    "UNAME" VARCHAR(20),
    "CREATIONDATETIME" TIMESTAMP,
    PRIMARY KEY("UNAME","CREATIONDATETIME")
);

CREATE COLUMN TABLE "EMPTRIGGER"
(
   "EMPNO" INT PRIMARY KEY,
   "ENAME" VARCHAR(20)
);

CREATE TRIGGER "LOGAUDIT"
AFTER INSERT ON "EMPTRIGGER" FOR EACH ROW
BEGIN
   INSERT INTO "AUDIT" VALUES(CURRENT_USER,CURRENT_TIMESTAMP);
END;
```

```
INSERT INTO "EMPTRIGGER" VALUES(1,'RAJU');
```
→ On execution of this insert statement, creates a record in EMPTRIGGER table and also the corresponding record in 'LOGAUDIT' table.

```
INSERT INTO "EMPTRIGGER" VALUES(2,'RAJU2');
INSERT INTO "EMPTRIGGER" VALUES(3,'RAJU3');
INSERT INTO "EMPTRIGGER" VALUES(4,'RAJU4');

DROP TRIGGER "LOGAUDIT";
```

INSERT INTO "EMPTRIGGER" VALUES(5,'RAJU5'); → **On execution of this insert statement, creates a record only in EMPTRIGGER table and not in 'LOGAUDIT' table as the trigger is dropped in the previous execution.**

Implementing Referential Integrity between tables using Triggers

Referential Integrity ensures following:

1. Doesn't allow to insert child row in child table when there is no corresponding parent row in parent table
2. Doesn't allow to delete parent row from parent table when there are dependent child rows in child table

Trigger Definition for Referential Integrity point 1:

```
CREATE COLUMN TABLE "TEMP"
(
   "EMPNO" INT PRIMARY KEY,
   "ENAME" VARCHAR(20),
   "DEPTNO" INT
);

CREATE COLUMN TABLE "TDEPT"
(
  "DEPTNO" INT PRIMARY KEY,
  "DNAME" VARCHAR(20)
);

CREATE TRIGGER "CHECKDEPTNO"
BEFORE INSERT ON "TEMP"
REFERENCING NEW ROW MYNEWROW
FOR EACH ROW
BEGIN
  DECLARE V_CNT INT;
  DECLARE MYEXCEPTION CONDITION FOR SQL_ERROR_CODE 10000;
   SELECT COUNT(*) INTO V_CNT FROM "TDEPT" WHERE DEPTNO
= :MYNEWROW.DEPTNO;
   IF V_CNT = 0 THEN
      SIGNAL MYEXCEPTION SET MESSAGE_TEXT = 'No Corresponding Parent
Record Exists';
   END IF;
END;

INSERT INTO "TDEPT" VALUES(10,'ORACLE');
```

```
INSERT INTO "TDEPT" VALUES(20,'JAVA');
INSERT INTO "TDEPT" VALUES(30,'SAP');

INSERT INTO "TEMP" VALUES(1,'RAJU',10);
INSERT INTO "TEMP" VALUES(2,'RAVI',20);
INSERT INTO "TEMP" VALUES(3,'RANI',10);

INSERT INTO "TEMP" VALUES(4,'RAMESH',40);

INSERT INTO "TEMP" VALUES(5,'RAMESH',30);

INSERT INTO "TEMP" VALUES(4,'RAMESH',50);
```

21 june 2019

```
SET SCHEMA "GENSOFTSQL";

INSERT INTO "TEMP" VALUES(9,'RAJ',40); //ERROR-NO PARENT RECORD EXISTS

SELECT * FROM "TRIGGERS"; // DISPLAYS ALL THE TRIGGERS CREATED ACROSS
SCHEMAS

SELECT * FROM "TRIGGERS" WHERE SCHEMA_NAME = 'GENSOFTSQL'; // DISPLAYS
ALL THE TRIGGERS CREATED IN THE SCHEMA GENSOFTSQL

ALTER TABLE "TEMP" DISABLE TRIGGER "CHECKDEPTNO"; // DISABLES THE
TRIGGER

SELECT * FROM "TRIGGERS" WHERE SCHEMA_NAME = 'GENSOFTSQL';

INSERT INTO "TEMP" VALUES(9,'RAJ',40); //INSERTS THE RECORD BECAUSE
THE TRIGGER IS DISABLED

ALTER TABLE "TEMP" ENABLE TRIGGER "CHECKDEPTNO"; // ENABLE THE TRIGGER

SELECT * FROM "TRIGGERS" WHERE SCHEMA_NAME = 'GENSOFTSQL';
```

Sequence:

It is similar to Number Range Buffer. It is just like generating numbers for Sales Orders, Billing Invoices….
It can be created with UI as well as using SQL Editor.

Sequence Creation using SQL:

```
CREATE SEQUENCE "SEQID"
START WITH 10
INCREMENT BY 1
MINVALUE 10
MAXVALUE 100
CYCLE
CACHE 5; //SEQUENCE WILL BE CREATED

SELECT "SEQID".CURRVAL FROM DUMMY; // DISPLAYS ERROR AS SEQUENCE HAS
NOT STARTED GENERATING THE SEQUENCE NUMBERS

INSERT INTO "TEMP" VALUES("SEQID".NEXTVAL,'KRISHNA',10); // INSERTS
EMP RECORD BY GENERATING SEQUENCE NO FOR EMPNO

INSERT INTO "TEMP" VALUES("SEQID".NEXTVAL,'KRISHNA2',20); // INSERTS
EMP RECORD BY GENERATING SEQUENCE NO FOR EMPNO

SELECT "SEQID".CURRVAL FROM DUMMY; // DISPLAYS CURRENT VALUE OF THE
SEQUENCE

SELECT "SEQID".NEXTVAL FROM DUMMY; // DISPLAYS NEXT VALUE OF THE
SEQUENCE

SELECT "SEQID".CURRVAL FROM DUMMY; // DISPLAYS CURRENT VALUE OF THE
SEQUENCE

INSERT INTO "TEMP" VALUES("SEQID".NEXTVAL,'KRISHNA3',30); // INSERTS
EMP RECORD BY GENERATING SEQUENCE NO FOR EMPNO

INSERT INTO "TEMP" VALUES("SEQID".NEXTVAL,'KRISHNA4',40); // FAILS TO
INSERTS EMP RECORD AS NO PARENT RECORD BUT GENERATES SEQUENCE NO IN
MEMORY

SELECT "SEQID".CURRVAL FROM DUMMY; // DISPLAYS CURRENT VALUE OF THE
SEQUENCE
```

Trigger Definition for Referential Integrity point 2:

```
CREATE TRIGGER "CHECKCHILDDATA"
BEFORE DELETE ON "TDEPT"
REFERENCING OLD ROW MYOLDROW
FOR EACH ROW
BEGIN
   DECLARE V_CNT INT;
   DECLARE MYEXCEPTION CONDITION FOR SQL_ERROR_CODE 10001;
```

```
   SELECT COUNT(*) INTO V_CNT FROM "TEMP" WHERE DEPTNO
= :MYOLDROW.DEPTNO;
   IF V_CNT > 0 THEN
      SIGNAL MYEXCEPTION SET MESSAGE_TEXT = 'Deletion Not Possible,
Dependent Child Rows exists';
   END IF;
END;   //Execute, Trigger Will be created

DELETE FROM "TDEPT" WHERE DEPTNO = 20; //WILL NOT ALLOW TO DELETE AS
DEPENDENT CHILD ROWS EXISTS

ALTER TABLE "TDEPT" DISABLE TRIGGER "CHECKCHILDDATA"; // DISABLES
TRIGGER

DELETE FROM "TDEPT" WHERE DEPTNO = 20; //DELETES DEPTNO 20 ROW

ALTER TABLE "TDEPT" ENABLE TRIGGER "CHECKCHILDDATA"; // ENABLES
TRIGGER

DELETE FROM "TDEPT" WHERE DEPTNO = 10; // WILL NOT ALLOW TO DELETE AS
DEPENDENT CHILD ROWS EXISTS
```

24 june 2019

Procedure: is a reusable database object which is a block of SQL statements defined only once and can be called any no. of times.
A Procedure can be created either with parameters or without parameters.
A Procedure can contain 3 categories of parameters
   1. IN 2. OUT 3. INOUT

IN Parameter is used to pass parameters to Procedure
OUT parameter is used for returning the value from the procedure
INOUT Parameter will act as both IN and OUT Parameter
By Default, Parameter Category is IN Parameter

```
SET SCHEMA "GENSOFTSQL";

CREATE PROCEDURE "SAYHI"
AS
BEGIN
  SELECT 'WELCOME TO HANA PROCEDURES' FROM DUMMY;
END; //PROCEDURE WITHOUT PARAMETERS

CALL "SAYHI"; → Calls the Procedure
```

```
CREATE PROCEDURE "PROC1"
AS
BEGIN
  DECLARE V_X INT;
  V_X=10;
  SELECT V_X AS MYVARIABLE FROM DUMMY;
END;

CALL "PROC1";

CREATE PROCEDURE "PROC2"
AS
BEGIN
  DECLARE V_RES INT;
  V_RES = 100 + 200;
  SELECT V_RES AS "RESULT" FROM DUMMY;
END;

CALL "PROC2";

CREATE PROCEDURE "PROC3"
( IN X INT,IN Y INT )
AS
BEGIN
   DECLARE V_Z INT;
   V_Z = X + Y;
   SELECT V_Z AS "SUM" FROM DUMMY;
END;

CALL "PROC3";  // FAILS TO CALL PROCEDURE AS INPUT PARAMETER VALUES
ARE NOT PASSED

CALL "PROC3"(10);// FAILS TO CALL PROCEDURE AS ONLY ONE INPUT
PARAMETER VALUE IS PASSED

CALL "PROC3"(10,20);

CALL "PROC3"(Y=>35,X=>21);

CREATE PROCEDURE "PROC4"
( X INT,IN Y INT )
AS
BEGIN
   DECLARE V_Z INT;
   V_Z = X + Y;
```

```
   SELECT V_Z AS "SUM" FROM DUMMY;
END;

CALL "PROC4"(34,20);

CREATE PROCEDURE "PROC5"
( IN X INT,IN Y INT,OUT R1 INT,OUT R2 INT )
AS
BEGIN
   R1 = X + Y;
   R2 = X - Y;
END;

CALL "PROC5"(20,10); // ERROR AS OUTPUT PARAMETERS ARE NOT BOUNDED

CALL "PROC5"(30,10,?,?);

CALL "PROC5"(Y=>40,X=>20,R1=>?,R2=>?);

CALL "PROC5"(Y=>40,X=>20,R1,R2=>?); // SYNTAX ERROR

CREATE PROCEDURE "PROC6"
AS
BEGIN
   SELECT * FROM "EMP";
END;

CALL "PROC6";

CREATE PROCEDURE "PROC7"
AS
BEGIN
   ITAB=SELECT * FROM "EMP";
END;

CALL "PROC7";

CREATE PROCEDURE "PROC8"
AS
BEGIN
   ITAB=SELECT * FROM "EMP";
   SELECT * FROM ITAB;
END;    // EXECUTE, GENERATES SYNTAX ERROR

CREATE PROCEDURE "PROC8"
AS
```

```
BEGIN
   ITAB=SELECT * FROM "EMP";
   SELECT * FROM :ITAB;
END;

CALL "PROC8";

CREATE PROCEDURE "PROC9"
(OUT ITAB "EMP")
AS
BEGIN
   ITAB=SELECT * FROM "EMP";
END;

CALL "PROC9"; // ERROR AS OUT PARAMETER ITAB IS NOT BOUND

CALL "PROC9"(?);

CALL "PROC9"(ITAB=>?);

CALL "PROC9"(MTAB=>?); // ERROR AS PARAMETER MTAB NOT FOUND

CREATE PROCEDURE "PROC11"
AS
BEGIN
   CALL "SAYHI";
END;

CALL "PROC11";

CREATE PROCEDURE "PROC12"
AS
BEGIN
  CALL "PROC4"(10,20);
END;

CALL "PROC12";

CREATE PROCEDURE "PROC13"
AS
BEGIN
  CALL "PROC4"(10,20);
  select 'HELLO' FROM DUMMY;
END;

CALL "PROC13";
```

26-06-2019

```
SET SCHEMA "GENSOFTSQL";
```

Cursors: are used for processing the result sets returned by the Select Query.

Steps:

1. Declare the cursor
2. Open the cursor
3. Read the cursor data
4. Close the cursor

1. DECLARE

SYNTAX:

```
declare cursor <cursorname> for <select query>;
```

2. OPEN

SYNTAX:

```
open <cursor name>;
```

3. READ

SYNTAX:

```
fetch <cursor name> into <target variables>;
```

4.CLOSE

SYNTAX:

```
Close <cursor name>;
```

Example: Create a Procedure using Cursors to increment the Manager's salary by 5 % and copy the corresponding records to "EMPLOY_DEPT_MNG" table and Non-Manager's salary by 10% and copy the corresponding records to "EMPLOY_DEPT_NONMNG' table

```
CREATE COLUMN TABLE "EMP_DEPT"
(
    "EMPNO" INT PRIMARY KEY,
    "ENAME" VARCHAR(20),
    "EMPDESIG" VARCHAR(20),
    "EMPSAL" INTEGER,
    "DEPTNO" INTEGER
);

INSERT INTO "EMP_DEPT" VALUES(1,'RAVI1','MANAGER',1000,10);
INSERT INTO "EMP_DEPT" VALUES(2,'RAVI2','SUPERVISOR',2000,20);
INSERT INTO "EMP_DEPT" VALUES(3,'RAVI3','EMPLOYEE',3000,30);
INSERT INTO "EMP_DEPT" VALUES(4,'RAVI4','MANAGER',4000,10);
INSERT INTO "EMP_DEPT" VALUES(5,'RAVI5','SUPERVISOR',5000,30);
INSERT INTO "EMP_DEPT" VALUES(6,'RAVI6','MANAGER',6000,20);
INSERT INTO "EMP_DEPT" VALUES(7,'RAVI7','EMPLOYEE',7000,10);
INSERT INTO "EMP_DEPT" VALUES(8,'RAVI8','MANAGER',8000,30);
INSERT INTO "EMP_DEPT" VALUES(9,'RAVI9','SUPERVISOR',9000,10);
INSERT INTO "EMP_DEPT" VALUES(10,'RAVI10','EMPLOYEE',10000,20);

CREATE TABLE "EMP_DEPT_MNG" LIKE "EMP_DEPT";

CREATE TABLE "EMP_DEPT_NONMNG" LIKE "EMP_DEPT";

CREATE PROCEDURE "INCREMENT_TRANSFER"
AS
BEGIN
  DECLARE V_EMPNO INT;
  DECLARE V_ENAME VARCHAR(20);
  DECLARE V_EMPDESIG VARCHAR(20);
  DECLARE V_EMPSAL INT;
  DECLARE V_DEPTNO INT;

 DECLARE CURSOR C1 FOR SELECT * FROM "EMP_DEPT";

 OPEN C1;

 LOOP
    FETCH C1 INTO V_EMPNO,V_ENAME,V_EMPDESIG,V_EMPSAL,V_DEPTNO;
    IF C1::NOTFOUND THEN
```

```
        CLOSE C1;
        BREAK;
    ELSE
        IF V_EMPDESIG='MANAGER' THEN
            V_EMPSAL = V_EMPSAL + (V_EMPSAL * 5 / 100);
            INSERT INTO "EMP_DEPT_MNG"
VALUES(V_EMPNO,V_ENAME,V_EMPDESIG,V_EMPSAL,V_DEPTNO);
        ELSE
            V_EMPSAL = V_EMPSAL + (V_EMPSAL * 10 / 100);
            INSERT INTO "EMP_DEPT_NONMNG"
VALUES(V_EMPNO,V_ENAME,V_EMPDESIG,V_EMPSAL,V_DEPTNO);
        END IF;
    END IF;
 END LOOP;
END;

CALL "INCREMENT_TRANSFER";
```

Example: Create a Procedure using Cursor for Loop to increment the salaries by 5 % if salary is < 5000 and 2 % if salary is > 5000

```
CREATE PROCEDURE "UPDATE_EMP_DEPT"
(IN V_DNO INT,OUT ITAB "EMP_DEPT")
AS
BEGIN
  DECLARE CURSOR C1 FOR SELECT "EMPNO","EMPSAL","DEPTNO" FROM
"EMP_DEPT" WHERE "DEPTNO"=V_DNO;

  DECLARE V_NEWSAL INT;

  FOR CURROW AS C1 DO
    IF CURROW."EMPSAL" < 5000 THEN
      V_NEWSAL = CURROW."EMPSAL" + (CURROW."EMPSAL" * 5 / 100);
      UPDATE "EMP_DEPT" SET "EMPSAL"=V_NEWSAL WHERE
"EMPNO"=CURROW."EMPNO";
    ELSE
      V_NEWSAL = CURROW."EMPSAL" + (CURROW."EMPSAL" * 2 / 100);
      UPDATE "EMP_DEPT" SET "EMPSAL"=V_NEWSAL WHERE
"EMPNO"=CURROW."EMPNO";
    END IF;
  END FOR;

  ITAB=SELECT * FROM "EMP_DEPT";
END;
```

```
CALL "UPDATE_EMP_DEPT"; //Error

CALL "UPDATE_EMP_DEPT"(10); // Error

CALL "UPDATE_EMP_DEPT"(10,?);
```

Note: Cursor For Loop need not be opened, fetched and closed explicitly.

27-06-2019.

```
SET SCHEMA "GENSOFTSQL";

CREATE COLUMN TABLE "E_DEPT"
(
   "EMPNO" INT PRIMARY KEY,
   "ENAME" VARCHAR(20),
   "DEPTNO" INT,
   "SALARY" DECIMAL(14,2),
   "COMM" DECIMAL(7,2)
);

INSERT INTO "E_DEPT"(EMPNO,ENAME,DEPTNO,SALARY)
VALUES(1,'RAVI1',10,1000);

INSERT INTO "E_DEPT"(EMPNO,ENAME,DEPTNO,SALARY)
VALUES(2,'RAVI2',20,2000);

INSERT INTO "E_DEPT"(EMPNO,ENAME,DEPTNO,SALARY)
VALUES(3,'RAVI3',30,3000);

INSERT INTO "E_DEPT"(EMPNO,ENAME,DEPTNO,SALARY)
VALUES(4,'RAVI4',10,4000);

INSERT INTO "E_DEPT"(EMPNO,ENAME,DEPTNO,SALARY)
VALUES(5,'RAVI5',20,5000);

INSERT INTO "E_DEPT"(EMPNO,ENAME,DEPTNO,SALARY)
VALUES(6,'RAVI6',30,6000);

INSERT INTO "E_DEPT"(EMPNO,ENAME,DEPTNO,SALARY)
VALUES(7,'RAVI7',30,7000);

INSERT INTO "E_DEPT"(EMPNO,ENAME,DEPTNO,SALARY)
VALUES(8,'RAVI8',10,8000);
```

```
INSERT INTO "E_DEPT"(EMPNO,ENAME,DEPTNO,SALARY)
VALUES(9,'RAVI9',20,9000);
```

Requirement: Create a Procedure to calculate the values for "COMM" column based on deptno column (Without using Cursors)

```
CREATE PROCEDURE "UPDATE_COMM"
AS
BEGIN
   DECLARE V_EMPNO INT;
   DECLARE V_DEPTNO INT;
   DECLARE V_SALARY DECIMAL(14,2);
   DECLARE V_COMM DECIMAL(7,2);
   DECLARE V_COUNT INT;
   DECLARE V_LOOPCOUNT INT = 0;

   ITAB=SELECT "EMPNO","DEPTNO","SALARY","COMM" FROM "E_DEPT";

   SELECT COUNT(*) INTO V_COUNT FROM :ITAB;

   IF V_COUNT <> 0 THEN
      LOOP
        IF V_COUNT <= V_LOOPCOUNT THEN
           BREAK;
        END IF;
        SELECT "EMPNO","DEPTNO","SALARY","COMM"
             INTO V_EMPNO,V_DEPTNO,V_SALARY,V_COMM
                FROM :ITAB LIMIT 1 OFFSET :V_LOOPCOUNT;
        IF V_DEPTNO = 10 THEN
            V_COMM = ( V_SALARY * 10 / 100 );
        ELSEIF V_DEPTNO = 20 THEN
            V_COMM = ( V_SALARY * 20 / 100 );
        ELSE
            V_COMM = ( V_SALARY * 30 / 100 );
        END IF;
        UPDATE "E_DEPT" SET COMM = V_COMM WHERE EMPNO = V_EMPNO;
        V_LOOPCOUNT = V_LOOPCOUNT + 1;
      END LOOP;
   END IF;
 END;   // Execute, Procedure gets created

 CALL "UPDATE_COMM"( ); // Calls above procedure successfully by
updating "COMM" column values
```

Consider the following DB Table ‘EMPTRIGGER’:

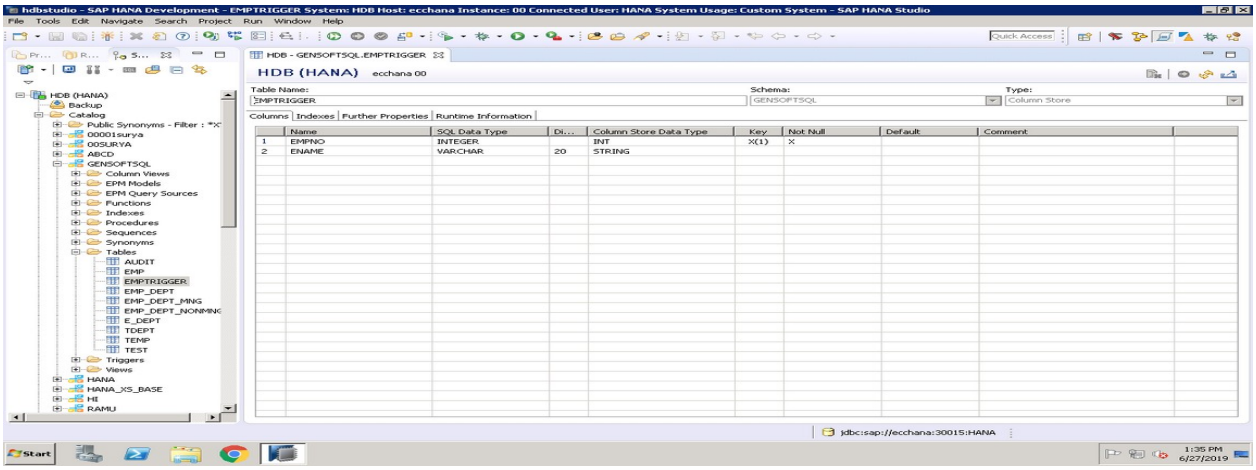
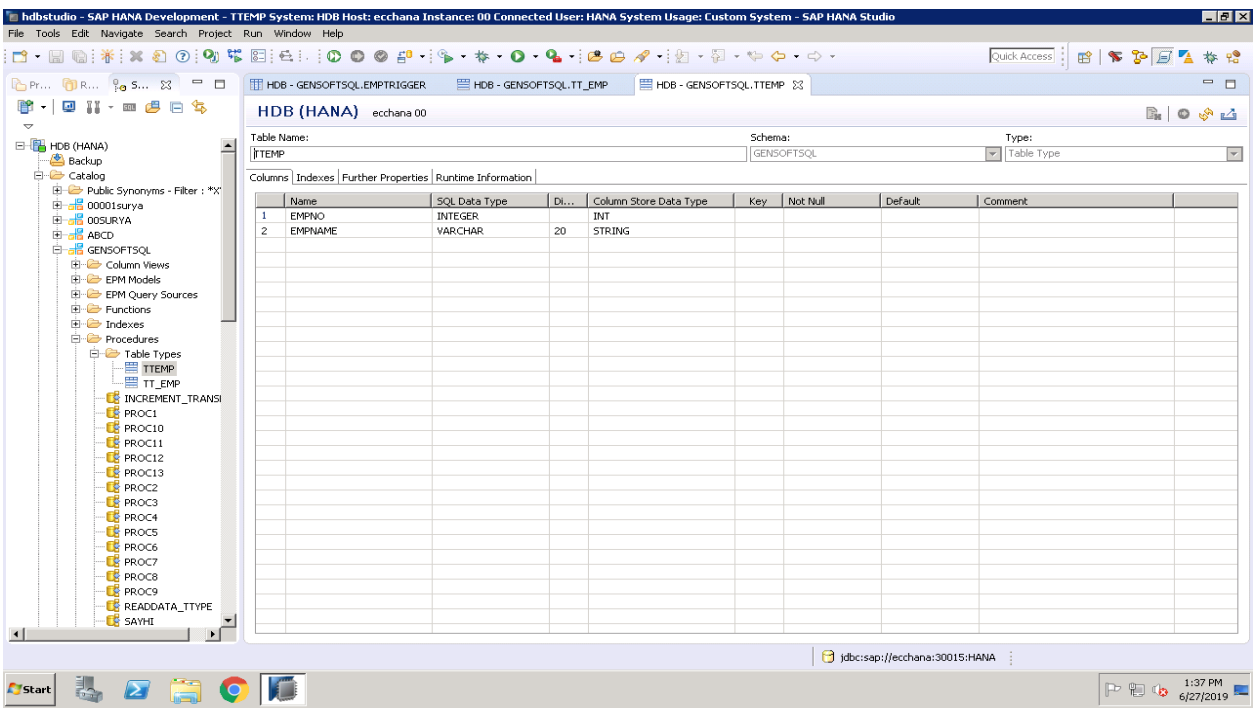


Table Type Creation using UI:



Example: Procedure to refer to above Table type as OUT Parameter

```
CREATE PROCEDURE "READDATA_TTYPE"
(OUT ITAB "TTEMP")
AS
BEGIN
   ITAB=SELECT * FROM "EMPTRIGGER";
END;    // ERROR-AS ONE OF THE ATTRIBUTE NAME IN TABLE TYPE ‘TTEMP’
AND DB TABLE ‘EMPTRIGGER’ ARE NOT SAME
```

Creating Table Type using SQL statement:

```
CREATE TYPE "TT_EMP" AS TABLE
(
   "EMPNO" INT,
   "ENAME" VARCHAR(20)
);   // Execute, Table Type gets created
```

Example: Procedure to refer to above Table type as OUT Parameter

```
CREATE PROCEDURE "READDATA_TTYPE"
 (OUT ITAB "TT_EMP")
 AS
 BEGIN
    ITAB=SELECT * FROM "EMPTRIGGER";
END;

CALL "READDATA_TTYPE"( ); // ERROR as OUT parameter itab not bound

CALL "READDATA_TTYPE"(?); //Calls above procedure successfully
```

28.08.2019

```
SET SCHEMA "GENSOFTSQL";

CREATE COLUMN TABLE "EMP_DEPT"
(
    "EMPID" INT PRIMARY KEY,
    "ENAME" VARCHAR(20),
    "DEPTNO" INT,
    "SALARY" DECIMAL(12,2)
);

INSERT INTO "EMP_DEPT" VALUES(1,'RAJ1',10,12456.45);
INSERT INTO "EMP_DEPT" VALUES(2,'RAJ2',20,54321.12);
INSERT INTO "EMP_DEPT" VALUES(3,'RAJ3',30,17654.23);
INSERT INTO "EMP_DEPT" VALUES(4,'RAJ4',20,76544.14);
INSERT INTO "EMP_DEPT" VALUES(5,'RAJ5',10,15431.45);
INSERT INTO "EMP_DEPT" VALUES(6,'RAJ6',30,21344.78);
```

Example: Create the Procedure to calculate the Tax based on deptno and return the Tax along with the Table columns

```
CREATE TYPE "TT_EMP_DEPT" AS TABLE
(
    "EMPID" INT,
```

```
    "ENAME" VARCHAR(20),
    "DEPTNO" INT,
    "SALARY" DECIMAL(12,2),
    "TAX" DECIMAL(8,2)
);    // Execute, Table type gets created

CREATE PROCEDURE "CALCULATETAX"
( OUT ITAB "TT_EMP_DEPT" )
AS
BEGIN
  ITAB1
   ITAB=SELECT "EMPID","ENAME","DEPTNO","SALARY",
               CASE "DEPTNO"
                   WHEN 10 THEN "SALARY" * 0.10
                   WHEN 20 THEN "SALARY" * 0.20
                   WHEN 30 THEN "SALARY" * 0.30
                   ELSE "SALARY" * 0.40
               END AS "TAX"
            FROM "EMP_DEPT";
 END; // Execute, Creates Procedure

 CALL "CALCULATETAX"( ); // Error-As Out parameter ITAB not bound
 CALL "CALCULATETAX"(?); // Executes procedure successfully

CREATE COLUMN TABLE "SAPABAP1"."EMP_DEPT1"
(
    "EMPID" INT PRIMARY KEY,
    "ENAME" VARCHAR(20),
    "DEPTNO" INT,
    "SALARY" DECIMAL(12,2)
);    // Creates Table in SAPABAP1 schema
```

Example: Read only Procedure

```
CREATE PROCEDURE "INSERT_TDEPT"
(IN IP_DEPTNO INTEGER,IN IP_DNAME VARCHAR(20),OUT OP_MSG VARCHAR(100))
READS SQL DATA
AS
BEGIN
  INSERT INTO "TDEPT" VALUES(IP_DEPTNO,IP_DNAME);
  OP_MSG='RECORD INSERTED SUCCESSFULLY';
END;  // ERROR-BECAUSE DML STATEMENTS ARE NOT SUPPORTED IN READ ONLY
PROCEDURE
```

Example: Procedure returning standard Exception message on entering duplicate value for primary key column

```
CREATE PROCEDURE "INSERT_TDEPT"
(IN IP_DEPTNO INTEGER,IN IP_DNAME VARCHAR(20),OUT OP_MSG VARCHAR(100))
--READS SQL DATA
AS
BEGIN
  INSERT INTO "TDEPT" VALUES(IP_DEPTNO,IP_DNAME);
  OP_MSG='RECORD INSERTED SUCCESSFULLY';
END;

CALL "INSERT_TDEPT"(20,'SALES',?); // SUCCESS

CALL "INSERT_TDEPT"(40,'FINANCE',?); // SUCCESS

CALL "INSERT_TDEPT"(10,'HR',?); // ERROR-UNIQUE CONSTRAINT VIOLATION
for deptno (deptno 10 already existing in the table)
```

Example: Procedure handling standard Exception and returning user defined message

```
CREATE PROCEDURE "INSERT_TDEPT_HANDLE_EXCEPTION"
(IN IP_DEPTNO INTEGER,IN IP_DNAME VARCHAR(20),OUT OP_MSG VARCHAR(100))
AS
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    OP_MSG='RECORD ALREADY EXISTS';
  END;

  INSERT INTO "TDEPT" VALUES(IP_DEPTNO,IP_DNAME);
  OP_MSG='RECORD INSERTED SUCCESSFULLY';
END;

CALL "INSERT_TDEPT_HANDLE_EXCEPTION"(50,'.NET',?);

CALL "INSERT_TDEPT_HANDLE_EXCEPTION"(20,'PRODUCTION',?);
```

01.07.2019

Example: Procedure to Capture Standard Exception Code and Standard Exception Message

```
CREATE PROCEDURE "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"
(IN IP_DEPTNO INTEGER,IP_DNAME VARCHAR(20),OP_MSG VARCHAR(150))
AS
BEGIN
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
     OP_MSG = 'SQL EXCEPTION RAISED,ERROR CODE IS
'||::SQL_ERROR_CODE||' ERROR MESSAGE IS '||::SQL_ERROR_MESSAGE;
   END;
   INSERT INTO "TDEPT" VALUES(IP_DEPTNO,IP_DNAME);
   OP_MSG = 'RECORD INSERTED SUCCESSFULLY';
END;

CALL "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"(70,'SALESFORCE',?); // Error-
as the output parameter message is more than 150 characters

DROP PROCEDURE "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"; // Drops the
Procedure
```

Example: Procedure to Capture Standard Exception code and Standard Exception message

```
CREATE PROCEDURE "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"
(IN IP_DEPTNO INTEGER,IP_DNAME VARCHAR(20),OUT OP_MSG VARCHAR(200))
AS
BEGIN
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
     OP_MSG = 'SQL EXCEPTION RAISED,ERROR CODE IS
'||::SQL_ERROR_CODE||' ERROR MESSAGE IS '||::SQL_ERROR_MESSAGE;
   END;
   INSERT INTO "TDEPT" VALUES(IP_DEPTNO,IP_DNAME);
   OP_MSG = 'RECORD INSERTED SUCCESSFULLY';
END;

CALL "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"(40,'SRM',?);

DROP PROCEDURE "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG";
```

Example: Procedure to Deal with Standard Exceptions and User defined Exceptions

```
CREATE PROCEDURE "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"
(IN IP_DEPTNO INTEGER,IP_DNAME VARCHAR(20),OUT OP_MSG VARCHAR(200))
AS
BEGIN
```

```
   DECLARE MYEXCEPTION CONDITION FOR SQL_ERROR_CODE 10003;

   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
     OP_MSG = 'SQL EXCEPTION RAISED,ERROR CODE IS
'||::SQL_ERROR_CODE||' ERROR MESSAGE IS '||::SQL_ERROR_MESSAGE;
   END;

   DECLARE EXIT HANDLER FOR MYEXCEPTION
   BEGIN
     OP_MSG = 'USER DEFINED EXCEPTION RAISED, ERROR CODE IS
'||::SQL_ERROR_CODE||' ERROR MESSAGE IS '||::SQL_ERROR_MESSAGE;
   END;

   IF IP_DNAME = ' ' THEN
      SIGNAL MYEXCEPTION SET MESSAGE_TEXT = 'DNAME CANNOT BE BLANK';
   END IF;

   INSERT INTO "TDEPT" VALUES(IP_DEPTNO,IP_DNAME);
   OP_MSG = 'RECORD INSERTED SUCCESSFULLY';
END;

CALL "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"(80,'DEVOPS',?); //No
exception raised

CALL "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"(40,'AWS',?); //Raises
Standard Exception

CALL "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"(90,' ',?); // Raises User
defined exception but the control goes to Standard Exit handler block
as it is handled first

DROP PROCEDURE "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG";
```

Example: Procedure to Deal with Standard Exceptions and User defined Exceptions

```
CREATE PROCEDURE "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"
(IN IP_DEPTNO INTEGER,IP_DNAME VARCHAR(20),OUT OP_MSG VARCHAR(200))
AS
BEGIN

   DECLARE MYEXCEPTION CONDITION FOR SQL_ERROR_CODE 10003;

   DECLARE EXIT HANDLER FOR MYEXCEPTION
```

```
   BEGIN
     OP_MSG = 'USER DEFINED EXCEPTION RAISED, ERROR CODE IS
'||::SQL_ERROR_CODE||' ERROR MESSAGE IS '||::SQL_ERROR_MESSAGE;
   END;

   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
     OP_MSG = 'SQL EXCEPTION RAISED,ERROR CODE IS
'||::SQL_ERROR_CODE||' ERROR MESSAGE IS '||::SQL_ERROR_MESSAGE;
   END;

   IF IP_DNAME = ' ' THEN
      SIGNAL MYEXCEPTION SET MESSAGE_TEXT = 'DNAME CANNOT BE BLANK';
   END IF;

   INSERT INTO "TDEPT" VALUES(IP_DEPTNO,IP_DNAME);
   OP_MSG = 'RECORD INSERTED SUCCESSFULLY';
END;

CALL "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"(100,'FICO',?); // No
Exception raised

CALL "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"(40,'AWS',?); // Raises
Standard Exception

CALL "INSERT_TDEPT_CAPTURE_EXCEPTIONMSG"(90,' ',?); // Raises User
defined exception and the control goes to Custom Exit handler block as
it is handled first
```

Example: Procedure with For EndFor

```
CREATE PROCEDURE "FOR_TEST"
AS
BEGIN
  DECLARE V_CNT INT = 1;
  FOR V_CNT IN 1..5 DO
     SELECT V_CNT FROM DUMMY;
  END FOR;
END;

CALL "FOR_TEST";  // Successfully Executes
```

Example: Procedure with Loop Endloop (Without Exit Condition)

```
CREATE PROCEDURE "LOOP_TEST"
AS
```

```
BEGIN
  DECLARE V_CNT INT = 1;
  DECLARE V_Y INT = 4;
  LOOP
     SELECT V_CNT FROM DUMMY;
     V_CNT = V_CNT + 1;
  END LOOP;
END;

CALL "LOOP_TEST"; // Goes for infinite execution
```

Example: Altering Procedure with Loop Endloop (With Exit Condition)

```
ALTER PROCEDURE "LOOP_TEST"
AS
BEGIN
  DECLARE V_CNT INT = 1;
  DECLARE V_Y INT = 4;
  LOOP
     SELECT V_CNT FROM DUMMY;
     IF V_CNT > V_Y THEN
       BREAK;
     END IF;
     V_CNT = V_CNT + 1;
  END LOOP;
END;

CALL "LOOP_TEST"; // Successfully executes
```

02.07.2019

PROCEDURE with Column views

```
CREATE COLUMN TABLE "SALESDATA"
(
    "ORDERID" INT PRIMARY KEY,
    "CUSTOMERFNAME" VARCHAR(25),
    "CUSTOMERLNAME" VARCHAR(25),
    "ORDERDATE" DATE,
    "PRICE" DECIMAL(8,2),
    QUANTITY DECIMAL(10,2)
);    // Execute, Creates the Table

INSERT INTO "SALESDATA" VALUES(1,'KRISHNA','KUMAR','2010-01-
02',19.5,5.0);
```

```
INSERT INTO "SALESDATA" VALUES(2,'MANOJ','KUMAR','2011-01-
03',29.5,9.0);

INSERT INTO "SALESDATA" VALUES(3,'VAMSHI','KUMAR','2012-01-
04',39.5,15.0);

INSERT INTO "SALESDATA" VALUES(4,'SUNIL','KUMAR','2013-02-
05',69.5,4.0);

INSERT INTO "SALESDATA" VALUES(5,'SURAJ','KUMAR','2014-08-
06',79.5,1.0);

INSERT INTO "SALESDATA" VALUES(6,'PAVAN','KUMAR','2015-09-
07',89.5,9.0);

INSERT INTO "SALESDATA" VALUES(7,'PRAVEEN','KUMAR','2016-02-
08',99.5,8.0);

INSERT INTO "SALESDATA" VALUES(8,'PRATAP','KUMAR','2017-03-
09',12.5,2.0);

INSERT INTO "SALESDATA" VALUES(9,'ASHOK','KUMAR','2018-05-
10',21.5,3.0);

INSERT INTO "SALESDATA" VALUES(10,'RAJENDRA','KUMAR','2019-06-
02',65.5,4.0);

CREATE TYPE "TT_SALESDATA" AS TABLE
(
    "ORDERID" INTEGER,
    "CUSTOMERNAME" VARCHAR(50),
    "ORDERDATE" DATE,
    "MONTHNUMBER" VARCHAR(3),
    "MONTHDESC" VARCHAR(15),
    "QUARTERNUMBER" VARCHAR(3),
    "YEARNUM" VARCHAR(4),
    "WEEKNUM" VARCHAR(10),
    "PRICE" DECIMAL(8,2),
    "QUANTITY" DECIMAL(10,2),
    "AMOUNT" DECIMAL(13,3)
);    // Execute, Creates the Table Type
```

Example: Create the Procedure to return the data related to above table type structure and also generate the column view

```
CREATE PROCEDURE "CALCULATESALESDATA"
```

```
(OUT ITAB "TT_SALESDATA")
READS SQL DATA
WITH RESULT VIEW "SALESREP"
AS
BEGIN
   ITAB=SELECT "ORDERID",
               CONCAT(CUSTOMERFNAME,CUSTOMERLNAME) AS "CUSTOMERNAME",
               TO_DATE(ORDERDATE,'YYYY-MM-DD') AS "ORDERDATE",
             MONTH(TO_DATE(ORDERDATE,'YYYY-MM-DD')) AS "MONTHNUMBER",
           MONTHNAME(TO_DATE(ORDERDATE,'YYYY-MM-DD')) AS "MONTHDESC",
         QUARTER(TO_DATE(ORDERDATE,'YYYY-MM-DD')) AS "QUARTERNUMBER",
               YEAR(TO_DATE(ORDERDATE,'YYYY-MM-DD')) AS "YEARNUM",
               WEEK(TO_DATE(ORDERDATE,'YYYY-MM-DD')) AS "WEEKNUM",
               "PRICE",
               "QUANTITY",
               "PRICE" * "QUANTITY" AS "AMOUNT"
               FROM "SALESDATA";
END;  // Execute, Creates Procedure and the column view 'SALESREP'
```

Note: To view the data in Column view, In the Column view folder, right click on the column view 'SALESREP' → Open data preview (Calls the procedure at the background and executes the business logic of the procedure and returns the internal table out parameter data to the column view)

```
CALL   "CALCULATESALESDATA"; // Error-as out parameter not bound

CALL   "CALCULATESALESDATA"(?);   // Calls procedure successfully

SELECT * FROM "SALESREP"; // Returns data from column view
```

Example: Procedure generating Column View with parameter

```
CREATE PROCEDURE "COLVIEWPARAMETER"
(IN IP_ODATE DATE,OUT ITAB "SALESDATA")
READS SQL DATA
WITH RESULT VIEW "SALESREP_PARAM"
AS
BEGIN
   ITAB=SELECT * FROM "SALESDATA" WHERE ORDERDATE >= IP_ODATE;
END;   // Execute, Creates the procedure and the column view
"SALESREP_PARAM"
```

Note: If we try to preview the data from the generated column view, it generates error as the column view is expecting input parameter value

```
SELECT * FROM "SALESREP_PARAM"(PLACEHOLDER."$$IP_ODATE$$"=>'2014-01-01'); //ERROR-as the input parameter name in the column view should be always be referred in lower case

SELECT * FROM "SALESREP_PARAM"(PLACEHOLDER."$$ip_odate$$"=>'2014-01-01');   // Successfully returns the data from the column view by calling the procedure at the background

CALL "COLVIEWPARAMETER"('2014-01-01',?); // Successfully calls procedure

CREATE PROCEDURE "COLVIEWMULTIOUT"
(IN I_X INT,IN I_Y INT,OUT O_R1 INT,OUT O_R2 INT)
READS SQL DATA
WITH RESULT VIEW "MULTIOUTCOLVIEW"
AS
BEGIN
     O_R1 = I_X + I_Y;
     O_R2 = I_X - I_Y;
END; // ERROR- as a procedure with result view cannot contain scalar parameters as OUT parameters

CREATE PROCEDURE "COLVIEWMULTIOUT"
(IN I_X INT,IN I_Y INT,OUT O_R1 INT)
READS SQL DATA
WITH RESULT VIEW "MULTIOUTCOLVIEW"
AS
BEGIN
     O_R1 = I_X + I_Y;
END; // ERROR- as a procedure with result view can contain only OUT Table parameters and not scalar parameters datatypes

CREATE PROCEDURE "COLVIEWPARAMETER_2"
(IN IP_ODATE DATE,OUT ITAB "SALESDATA",OUT ITAB1 "SALESDATA")
READS SQL DATA
WITH RESULT VIEW "SALESREP_PARAM_2"
AS
BEGIN
   ITAB=SELECT * FROM "SALESDATA" WHERE ORDERDATE >= IP_ODATE;
   ITAB1=ITAB;
END;  // ERROR-as a procedure with result view cannot contain multiple OUT Table parameters
```

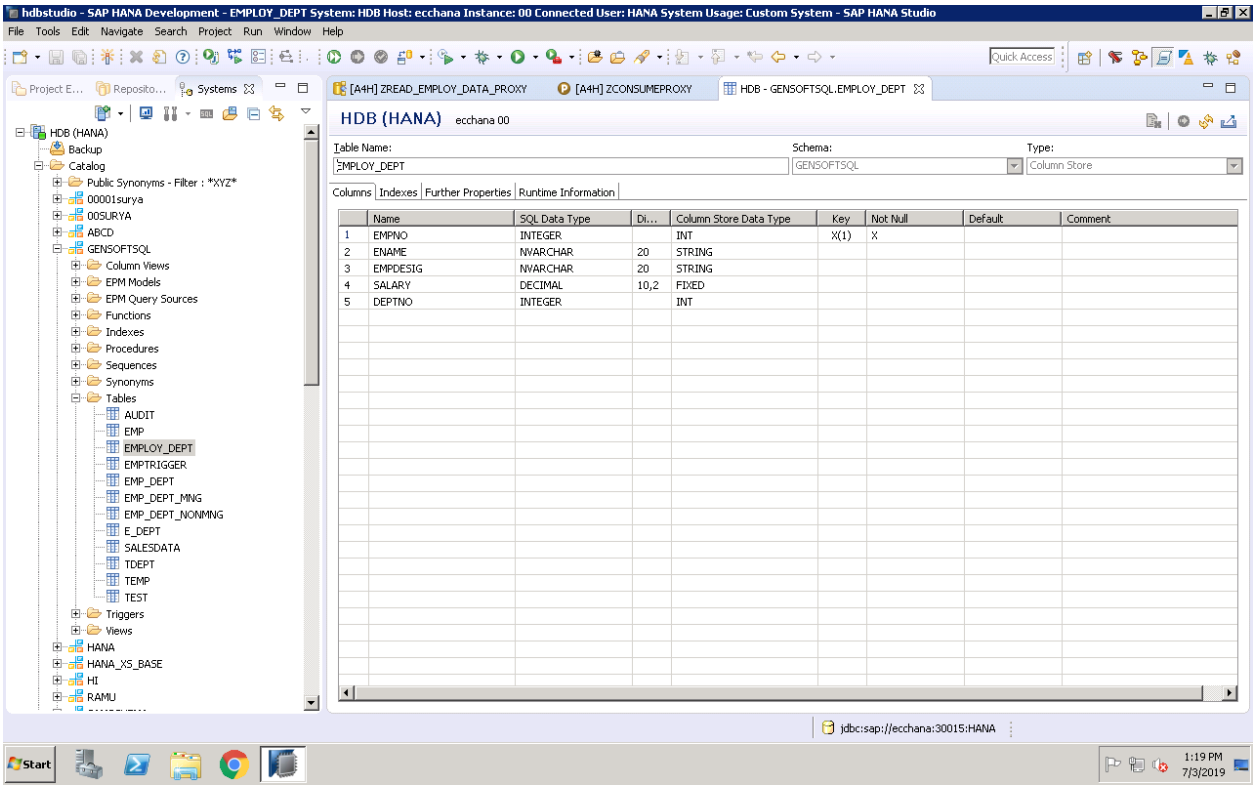
```
CREATE PROCEDURE "COLVIEWPARAMETER_2"
(IN IP_ODATE DATE,OUT ITAB "SALESDATA",OUT ITAB1 "SALESDATA")
READS SQL DATA
AS
BEGIN
   ITAB=SELECT * FROM "SALESDATA" WHERE ORDERDATE >= IP_ODATE;
   ITAB1=SELECT * FROM "SALESDATA" WHERE ORDERDATE >= IP_ODATE;
END; // Successfully creates the procedure as there is no result view
addition
```

03.07.2019

Consuming Stored Procedures from ABAP Reports (ABAP Layer):

Procedure Requirement: To return data from a database table

1. **Consider / Create the following table with no 'VARCHAR' datatype fields**



2. **Insert some data in to the above table**

```
INSERT INTO "EMPLOY_DEPT" VALUES(1,'RAJU1','MANAGER',12000.20,10);
INSERT INTO "EMPLOY_DEPT" VALUES(2,'RAJU2','EMPLOYEE',11000.20,20);
INSERT INTO "EMPLOY_DEPT" VALUES(3,'RAJU3','SUPERVISOR',13000.20,30);
```

```
INSERT INTO "EMPLOY_DEPT" VALUES(4,'RAJU4','EMPLOYEE',8000.20,10);
INSERT INTO "EMPLOY_DEPT" VALUES(5,'RAJU5','SUPERVISOR',9000.20,20);
INSERT INTO "EMPLOY_DEPT" VALUES(6,'RAJU6','MANAGER',7000.20,30);
```

3. **In HANA Development Perspective,** Create Package for storing the Stored Procedure: (Systems Tab, Right Click on Content → New → Package, Provide Package name (ZGENSOFTPACK), Description (….)

4. **In HANA Development Perspective itself,** in the Repositories tab, Right Click on your Package → New → Other, Choose Stored Procedure, Next, Provide the details

   File Name (Name of Procedure) → READ_EMPLOY_DATA

   File Format → Choose XML (.procedure)

   Target Schema → Browse and Choose '_SYS_BIC', Finish, Procedure Template gets created

5. **Define the Procedure as follows**

```
CREATE PROCEDURE _SYS_BIC.READ_EMPLOY_DATA ( OUT ITAB "EMPLOY_DEPT" )
     LANGUAGE SQLSCRIPT
     SQL SECURITY DEFINER
     DEFAULT SCHEMA GENSOFTSQL
     READS SQL DATA AS
BEGIN
   ITAB=SELECT * FROM "EMPLOY_DEPT";
     END;
```

Save (Ctrl+s), Check for Syntax errors (Ctrl+f2), activate (Ctrl+f3).

Note: If any problem in syntax check/activation, execute the following command in SQL console of Systems Tab, and then activate the procedure

GRANT SELECT ON SCHEMA "GENSOFTSQL" TO "_SYS_REPO" WITH GRANT OPTION;

Note: If the procedure is successfully activated, it creates Runtime Object in the Catalog folder under '_SYS_BIC' Schema (Systems Tab)

Runtime object in _SYS_BIC Schema:

```
create procedure "_SYS_BIC"."ZGENSOFTPACK/READ_EMPLOY_DATA" (  out
ITAB "GENSOFTSQL"."EMPLOY_DEPT"  ) language SQLSCRIPT sql security
definer default schema "GENSOFTSQL" reads sql data as
BEGIN
   ITAB=SELECT * FROM "EMPLOY_DEPT";
END;
```

Calling the Procedure from SQL Console:

```
CALL "_SYS_BIC.READ_EMPLOY_DATA"(?); //ERROR-AS WE ARE REFERRING TO
DESIGN TIME OBJECT

CALL "_SYS_BIC"."ZGENSOFTPACK/READ_EMPLOY_DATA"(?);// SUCCESS
```

Note: Now to consume the above Stored Procedure from ABAP Layer (ABAP Reports), we need to expose the stored procedure to Dictionary as Database Procedure Proxy which needs to be created on top of Stored Procedure and consume the same from ABAP Report

Switch to 'ABAP Perspective': (Connect to SAP System by providing Credentials)

Project Explorer Tab:

Right click on Local Objects (Or any Package) → New → Other ABAP repository object, Under Dictionary → choose Database Procedure Proxy

Provide following details:

Name: ZREAD_EMPLOY_DATA_PROXY (ANY DB Proxy Name->User Defined name)

Description: …..

HANA Procedure: ZGENSOFTPACK.READ_EMPLOY_DATA (Press Ctrl + space to browse and select)

Parameter Types Interface: ZIF_ZREAD_EMPLOY_DATA_PROXY (Proposed Automatically by SAP), Next

Displays DB Proxy and Generated Interface, Activate

DB Procedure proxy will be successfully created.

Consuming above DB proxy from ABAP Report:

```
REPORT ZCONSUMEPROXY.

data : t_emp type table of ZIF_ZREAD_EMPLOY_DATA_PROXY=>itab,
       wa_emp type ZIF_ZREAD_EMPLOY_DATA_PROXY=>itab.

call DATABASE PROCEDURE zread_employ_data_proxy
IMPORTING
 itab = t_emp.

 if t_emp is not INITIAL.
  loop at t_emp into wa_emp.
    write :/ wa_emp-empno,
             wa_emp-ename,
```

```
            wa_emp-empdesig,
            wa_emp-salary,
            wa_emp-deptno.
  endloop.
 endif.
```

04.07.2019 & 05.07.2019

Complete Syntax for Creating Procedure:

```
CREATE PROCEDURE <PROCEDURE NAME> ( [ <PARAMETERS> ] )
               [ LANGUAGE <language> ]
               [ SQL SECURITY <MODE> ]
               [ DEFAULT SCHEMA <schema name> ]
               [READS SQL DATA [WITH RESULT VIEW <VIEW NAME>] ]
     AS
      BEGIN
           <PROCEDURE BODY>
      END;
```

Default Language: SQL Script, other language supported is 'R' language

SQL Security Mode: 1. DEFINER 2. INVOKER (Default)

INVOKER: In this, it uses the permissions (rights) of the user who is calling the procedure while executing the procedure

DEFINER: In this, it uses the permissions (rights) of the user whoever has created the procedure (owner) while executing the procedure

Default Schema <schema name>: indicates that all the database objects used inside the procedure body will be picked from specified default schema. If the objects of other schema have to be used, then we need to prefix the object with the schema name

Reads SQL Data indicates that we are not going to perform any data manipulations (i.e NO Insert, NO Delete, NO Update statements), so that the processor will skip some privilege checks on the procedure

Note: We use the option 'WITH RESULT VIEW <view name> whenever we want to return the table types as output parameters when the procedure is executed. When we activate the procedure, it creates column view (similar to materialized view in ORACLE)

Example 2: Consuming HANA Stored Procedure from ABAP Reports

Steps in HANA Studio:

1. Create a DB Table with the following structure as part of the schema (GENSOFTSQL)

EMPID	INTEGER	PRIMARY KEY
ENAME	VARCHAR(20)	
DEPTNO	INTEGER	
SALARY	DECIMAL(12,2)	

2. Maintain Some Data as part of above table

3. In HANA Development Perspective, Create the package under Systems tab by right click on 'Content' Sub node (Package: ZGENPACK)

4. In HANA Development Perspective, Create the stored procedure as part of Repositories tab under the package 'ZGENPACK'

Provide the details as follows:

File Name → READDATA_EMP_DEPT (Name of Procedure)
File Format → .XML
Target Schema → Browse and choose '_SYS_BIC' schema, Finish, Generates the template

Define the procedure as follows

```
CREATE PROCEDURE _SYS_BIC.READDATA_EMP_DEPT ( OUT ITAB "EMP_DEPT" )
     LANGUAGE SQLSCRIPT
     SQL SECURITY INVOKER
     DEFAULT SCHEMA GENSOFTSQL
     READS SQL DATA AS
BEGIN
/****************************
     Write your procedure logic
 ****************************/
 ITAB = SELECT * FROM "EMP_DEPT";
```

```
END;
```

Save, check and activate.

Note: If any problem in syntax check/activation, execute the following command in SQL console of Systems Tab, and then activate the procedure

GRANT SELECT ON SCHEMA "GENSOFTSQL" TO "_SYS_REPO" WITH GRANT OPTION;

Note: If the procedure is successfully activated, it creates Runtime Object in the Catalog folder under '_SYS_BIC' Schema (Systems Tab)

Runtime object in _SYS_BIC Schema:

```
create procedure "_SYS_BIC"."ZGENPACK/READDATA_EMP_DEPT" (  out ITAB
"GENSOFTSQL"."EMP_DEPT"  ) language SQLSCRIPT sql security invoker
default schema "GENSOFTSQL" reads sql data as
BEGIN
/****************************
     Write your procedure logic
 ****************************/
 ITAB = SELECT * FROM "EMP_DEPT";

END;
```

Calling above procedure from SQL Console:

```
CALL "_SYS_BIC"."ZGENPACK/READDATA_EMP_DEPT"(?);
```
→ **successfully calls the procedure and displays the result**

Creating Database procedure proxy:

Now, we need to create the database procedure proxy for the above stored procedure so that we can consume the proxy from ABAP report

a) Open ABAP perspective and connect to Application server by providing login details

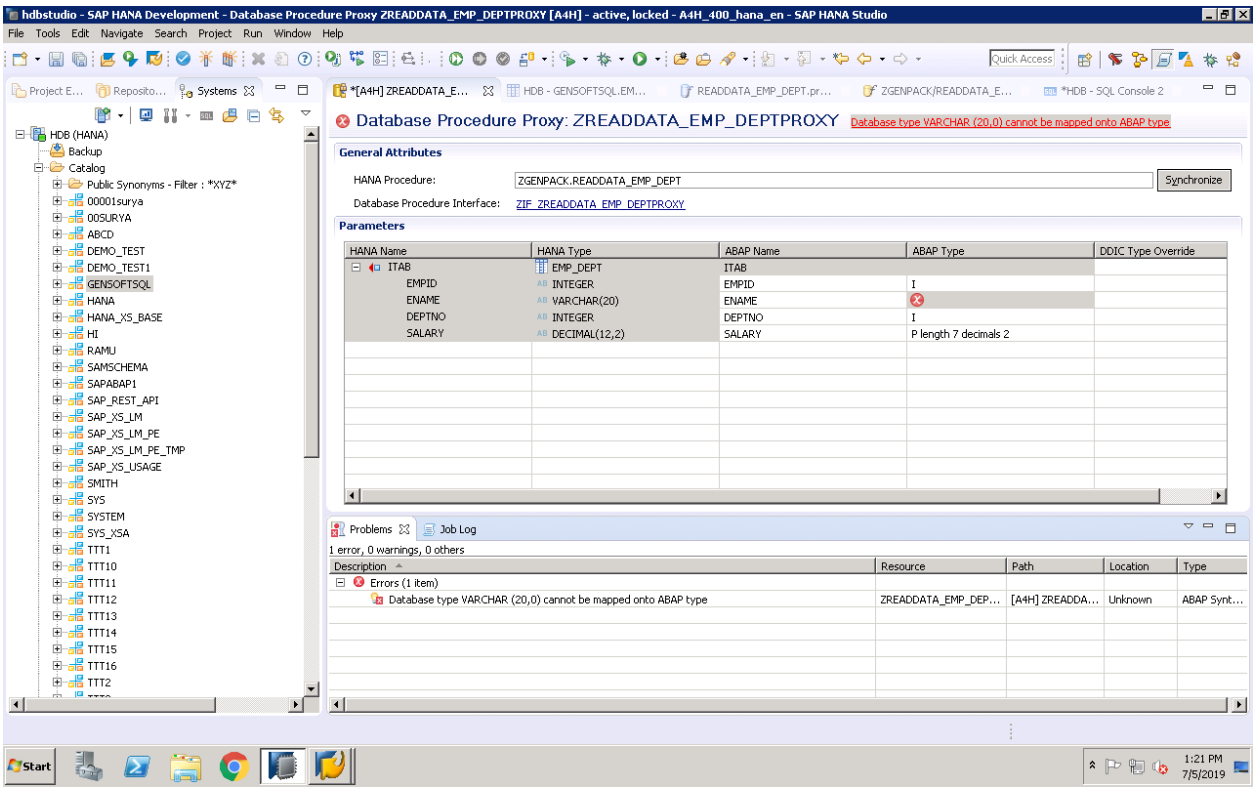
b) Right click on Local Objects / any package → other ABAP repository object → Under SAP HANA → Under Database development → Under Dictionary, choose Database procedure proxy, provide the following details

Name → ZREADDATA_EMP_DEPTPROXY (Name of DB proxy)
Description → …..

HANA Procedure → ZGENPACK.READDATA_EMP_DEPT (Browse by pressing ctrl + space, displays stored procedures from _SYS_BIC schema)

Parameter Type interface → ZIF_ZREADDATA_EMP_DEPTPROXY (Proposed by SAP), Click on Next and then Finish

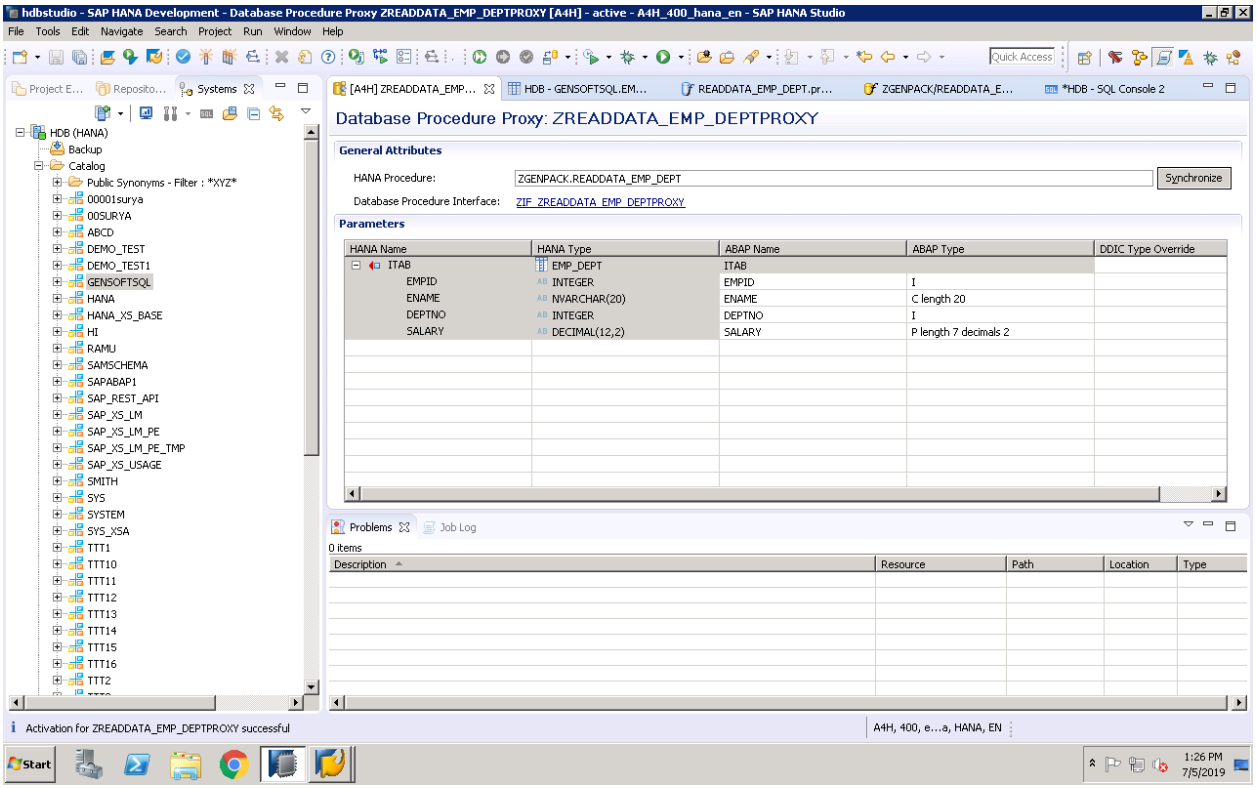
DB Procedure proxy will be created with errors as one of the fields in procedure parameter is varchar data type which is not compatible with ABAP data type



So, Now we need to modify the data type of 'Ename' column from varchar to Nvarchar so that it can be mapped at proxy level

```
ALTER TABLE EMP_DEPT ALTER (ENAME NVARCHAR(20));
```
→ Execute , Modifies the column datatype at table level

Now, We need to synchronize this changes at proxy level. For this, Open the DB procedure proxy created earlier and click on Synchronize button, Now the proxy will be created successfully along with Global interface.



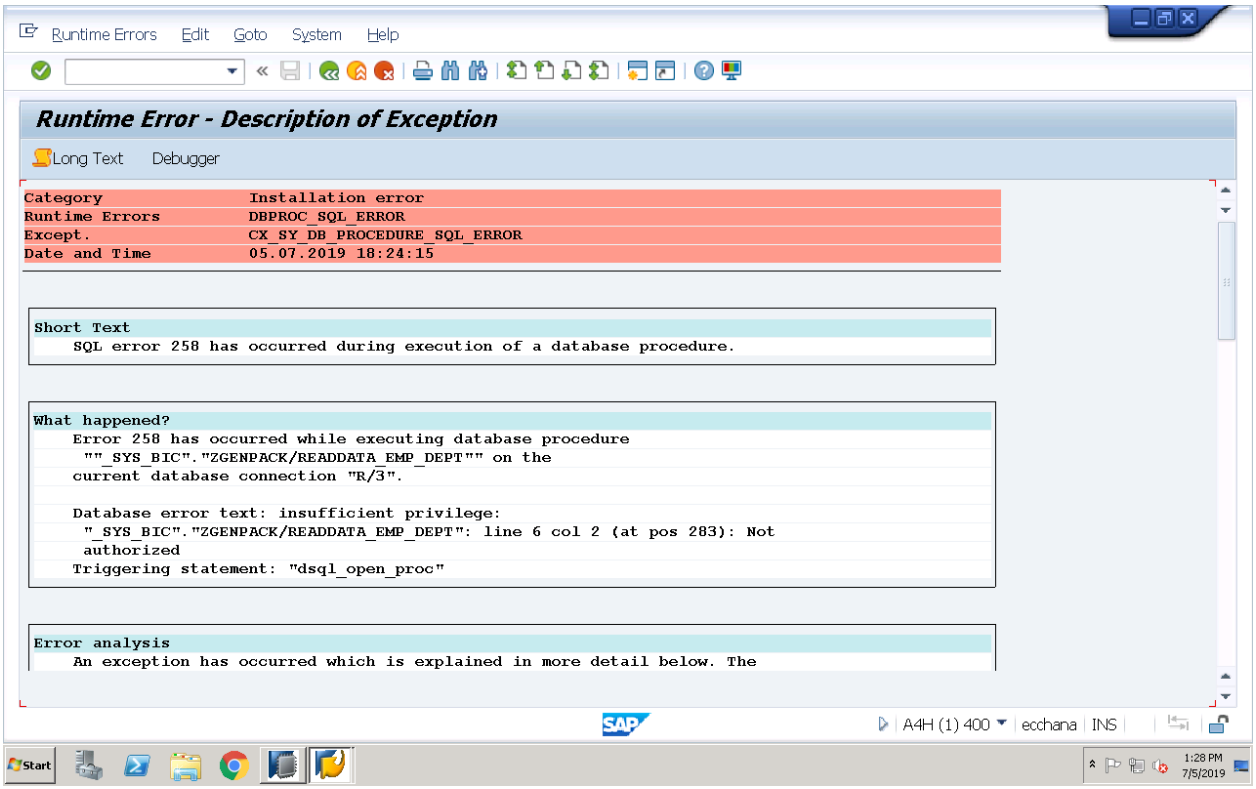
Consuming above Database Procedure Proxy from ABAP report

Create ABAP Executable program (from GUI or HANA Studio)

```
REPORT ZCONSUME_PROXY.

data : t_emp type table of ZIF_ZREADDATA_EMP_DEPTPROXY=>itab,
      wa_emp type ZIF_ZREADDATA_EMP_DEPTPROXY=>itab.

CALL DATABASE PROCEDURE ZREADDATA_EMP_DEPTPROXY
IMPORTING
  ITAB = t_emp.

if t_emp is not INITIAL.
  loop at t_emp into wa_emp.
    write :/ wa_emp-empid,
             wa_emp-ename,
             wa_emp-deptno,
             wa_emp-salary.
   endloop.
else.
  write :/ 'No data'.
endif.
```

Save, check and activate

On executing the above program, it may lead to runtime error 'insufficient privilege' as there is no authorization to execute the procedure.



Now, go back to the stored procedure (design time object) i.e. in Development perspective, in the repositories tab, open the procedure stored in the package (zgenpack) and change the SQL security to DEFINER instead of INVOKER.

```
CREATE PROCEDURE _SYS_BIC.READDATA_EMP_DEPT ( OUT ITAB "EMP_DEPT" )
      LANGUAGE SQLSCRIPT
      SQL SECURITY DEFINER
      DEFAULT SCHEMA GENSOFTSQL
      READS SQL DATA AS
BEGIN
/****************************
      Write your procedure logic
 ****************************/
 ITAB = SELECT * FROM "EMP_DEPT";

END;
```

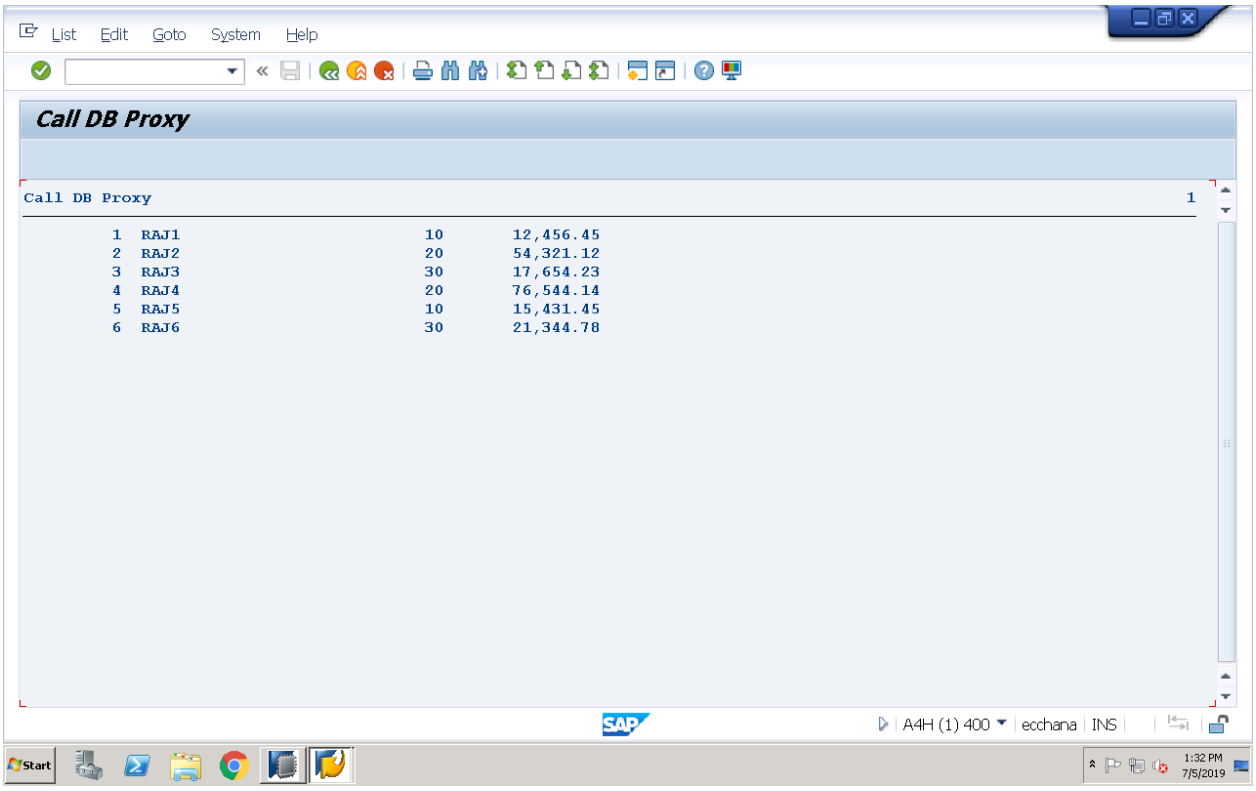
Save, check and activate

Now, SAP re-generates the runtime object of the above stored procedure under '_SYS_BIC' schema

Now, In ABAP perspective, Open the existing DB procedure Proxy and click on 'Synchronize button' so that the changes made to the stored procedure gets synchronized with DB procedure Proxy.

Now, activate the synchronized DB procedure proxy and execute the ABAP report consuming the DB procedure proxy.

Displays following result



ARRAY: is a variable which is collection of multiple values of same data type. It is like a internal table, but Internal table can hold multiple columns with multiple rows whereas Array can hold single column with multiple rows

Example 1: Procedure to transfer column values of a temporary internal table to ARRAY (ARRAY_AGG Function)

```
CREATE PROCEDURE "ARRAY2"( )
AS
```

```
BEGIN
  DECLARE V_DEPTNO INT ARRAY;
  ITAB=SELECT * FROM "EMP_DEPT";
  V_DEPTNO = ARRAY_AGG(:ITAB."DEPTNO");
  SELECT :V_DEPTNO[5] FROM DUMMY;
END;  // Execute, Successfully Creates Procedure

CALL "ARRAY2"( ); //Successfully Calls Procedure by displaying Fifth
row value from Deptno column

CREATE TYPE "T_PRODUCTS" AS TABLE
(
   "PRODUCTID" VARCHAR(100),
   "CATEGORY" VARCHAR(100),
   "PRICE" DECIMAL(12,2)
); // Execute, Creates User defined Table Type
```

Example: Procedure to RETURN MULTIPLE Array Values as a internal table parameter of Procedure (UNNEST Function) (Different no.of rows in each array)

```
CREATE PROCEDURE "ARRAY3"( OUT ITAB "T_PRODUCTS" )
AS
BEGIN
   DECLARE PRODUCTID VARCHAR(20) ARRAY;
   DECLARE CATEGORY VARCHAR(20) ARRAY;
   DECLARE PRICE DECIMAL(12,2) ARRAY;

   PRODUCTID[1] = 'PRODUCT1';
   PRODUCTID[2] = 'PRODUCT2';
   PRODUCTID[3] = 'PRODUCT3';
   PRODUCTID[4] = 'PRODUCT4';

   CATEGORY[1] = 'CATEGORY1';
   CATEGORY[2] = 'CATEGORY2';
   CATEGORY[3] = 'CATEGORY3';

   PRICE[1] = 12.54;
   PRICE[2] = 16.54;
   PRICE[3] = 11.54;

   ITAB=UNNEST(:PRODUCTID, :CATEGORY, :PRICE) AS
              ("PRODUCTID","CATEGORY","PRICE");
 END;     // Execute, Successfully Creates Procedure
```

```
 CALL "ARRAY3"(?);  // Successfully calls procedure by combining
Multiple Array rows into a single out internal table parameter
```

08.07.2019

User Defined Functions: It is a block of statements which is defined only once and can be called any no. of times. All the parameters passed by default are 'IN' parameter and function has to return minimum of one value.

Example 1:

```
CREATE FUNCTION "FUN1"(X INTEGER,Y INTEGER)
RETURNS K1 INTEGER,K2 INTEGER
AS
BEGIN
  K1 = X + Y;
  K2 = X + Y;
END;

CALL "FUN1"( ); // ERROR

CALL "FUN1"(10,20); // ERROR

CALL "FUN1"(20,10,?,?); // ERROR

SELECT "FUN1"(10,20) FROM DUMMY; // ERROR

SELECT "FUN1"(10,20).K1 FROM DUMMY; // SUCCESS

SELECT "FUN1"(10,20).K1 AS "ADDITION" FROM DUMMY; // SUCCESS

SELECT "FUN1"(10,20).K1 AS "ADDITION","FUN1"(20,30).K2 AS "DIFFERENCE"
FROM DUMMY; //SUCCESS
```

Example 2:

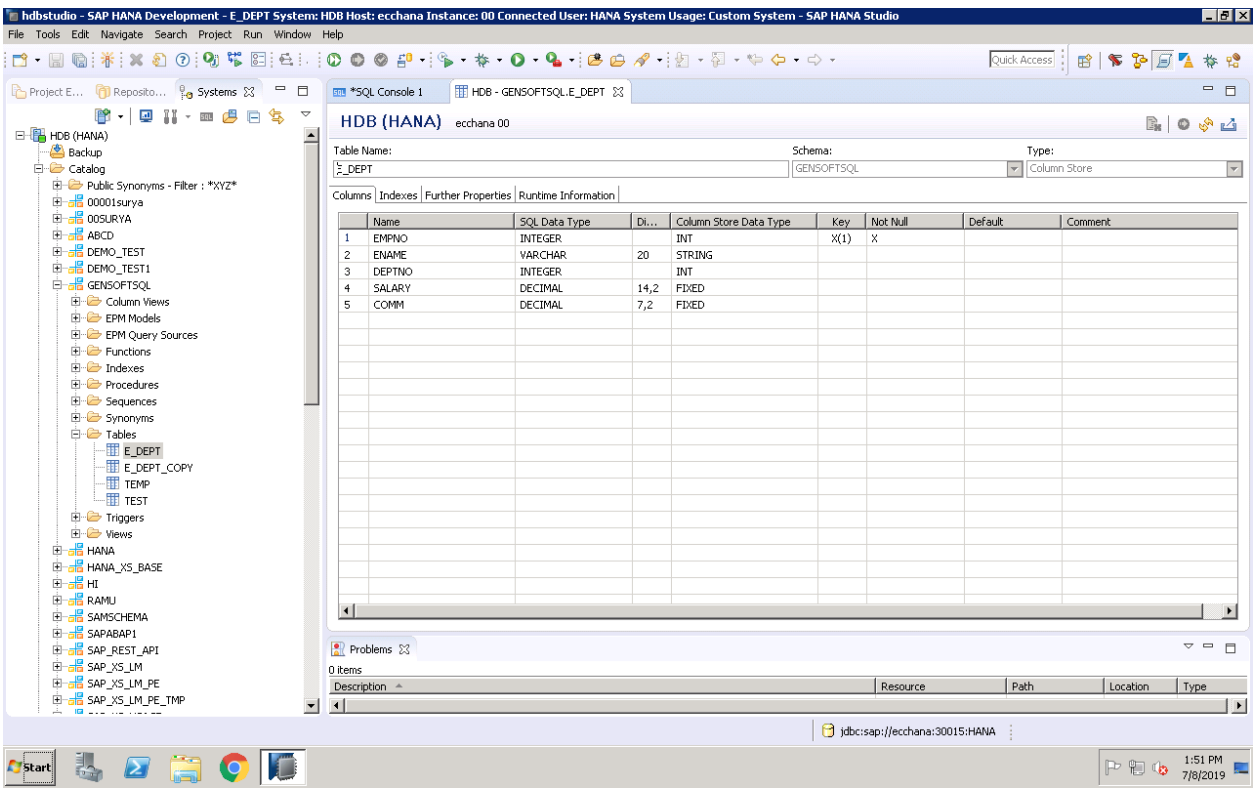
```
CREATE FUNCTION "FUN2"(X INTEGER,Y INTEGER)
RETURNS K1 INTEGER
AS
BEGIN
  K1 = X + Y;
END;

SELECT "FUN2"(10,20) FROM DUMMY; // EXECUTES AND RECEIVES THE ONLY
RETURN VALUE
```

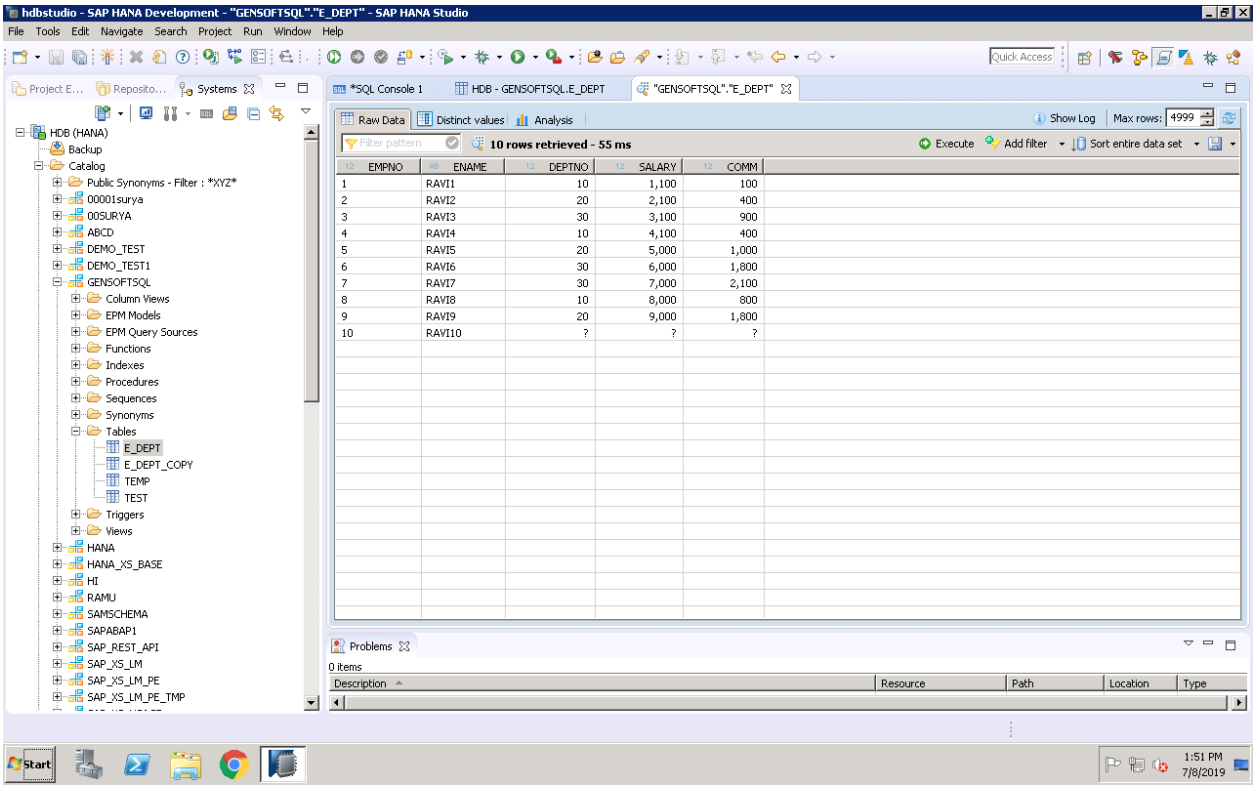
```
CREATE FUNCTION "FUN3"(X IN INTEGER,Y IN INTEGER)
RETURNS K1 INTEGER
AS
BEGIN
  K1 = X + Y;
END;  // EXECUTE, SYNTAX ERROR
```

Requirement:

1. **Create a Function to read the data from a database table**
2. **Call the Function inside a procedure and manipulate the return values using arrays and update the data back to database table**
3. **Also, Insert the whole data of the base table into other db table**

Consider the following DB table with data:






```
CREATE COLUMN TABLE "E_DEPT_COPY" AS
(
    SELECT "EMPNO","ENAME","SALARY" FROM "E_DEPT"
) WITH NO DATA;

CREATE TYPE "TY_E_DEPT" AS TABLE
(
   "EMPNO" INT,
   "ENAME" VARCHAR(20),
   "DEPTNO" INT,
   "SALARY" DECIMAL(14,2),
   "COMM" DECIMAL(7,2)
);

CREATE FUNCTION "GET_E_DEPT"( )
RETURNS "TY_E_DEPT"
AS
BEGIN
    RETURN SELECT * FROM "E_DEPT";
END;   // EXECUTE, CREATES FUNCTION

SELECT * FROM "GET_E_DEPT"( );    //SUCCESSFULLY CALLS ABOVE FUNCTION
```

```
CREATE PROCEDURE "INVOKE_GET_E_DEPT"( )
AS
BEGIN
  DECLARE RES "TY_E_DEPT";

  DECLARE EMPNO INT ARRAY;
  DECLARE ENAME VARCHAR(20) ARRAY;
  DECLARE DEPTNO INT ARRAY;
  DECLARE SALARY DECIMAL(14,2) ARRAY;
  DECLARE COMM DECIMAL(7,2) ARRAY;

  DECLARE LV_IDX INT = 0;

  RES = SELECT * FROM "GET_E_DEPT"( );

  EMPNO = ARRAY_AGG(:RES."EMPNO");
  ENAME = ARRAY_AGG(:RES."ENAME");
  DEPTNO = ARRAY_AGG(:RES."DEPTNO");
  SALARY = ARRAY_AGG(:RES."SALARY");
  COMM = ARRAY_AGG(:RES."COMM");

  FOR LV_IDX IN 1..CARDINALITY(:EMPNO) DO
      IF :SALARY[:LV_IDX] < 5000 THEN
         SALARY[:LV_IDX] = :SALARY[:LV_IDX] + 100;
      END IF;
  END FOR;

  EMPNO[10] := 10;
  ENAME[10] := 'RAVI10';

  RES = UNNEST(:EMPNO,:ENAME,:DEPTNO,:SALARY,:COMM) AS
             ("EMPNO","ENAME","DEPTNO","SALARY","COMM");

  UPSERT "E_DEPT" SELECT * FROM :RES;

  WA_RES = UNNEST(:EMPNO,:ENAME,:SALARY) AS
                ("EMPNO","ENAME","SALARY");

   INSERT INTO "E_DEPT_COPY" SELECT * FROM :WA_RES;
END;          // Execute, Creates Procedure


CALL "INVOKE_GET_E_DEPT"( );  // Success
```

Table: E_DEPT

Fields:

EMPID (PK)
ENAME
DEPTNO
SALARY
COMM

empid: 1,2,3,4,5,6,7,8,9

Table: E_DEPT_COPY

Fields:

EMPID (PK)
ENAME
SALARY

1. Create a Table Function to return the data from E_DEPT table as a table type

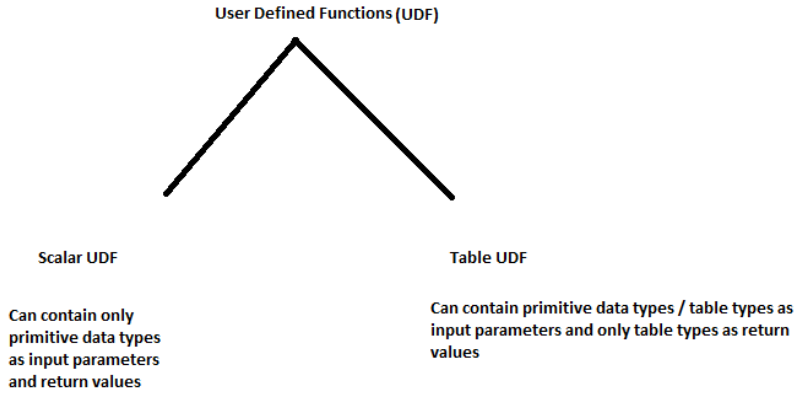
2. Create a procedure to call the above table function

   a) Manipulate the return table type (using Arrays) (Modifying the Internal table)

   b) Update the Manipulated return table type data back to the db table E_DEPT using UPSERT statement (similar to Modify)

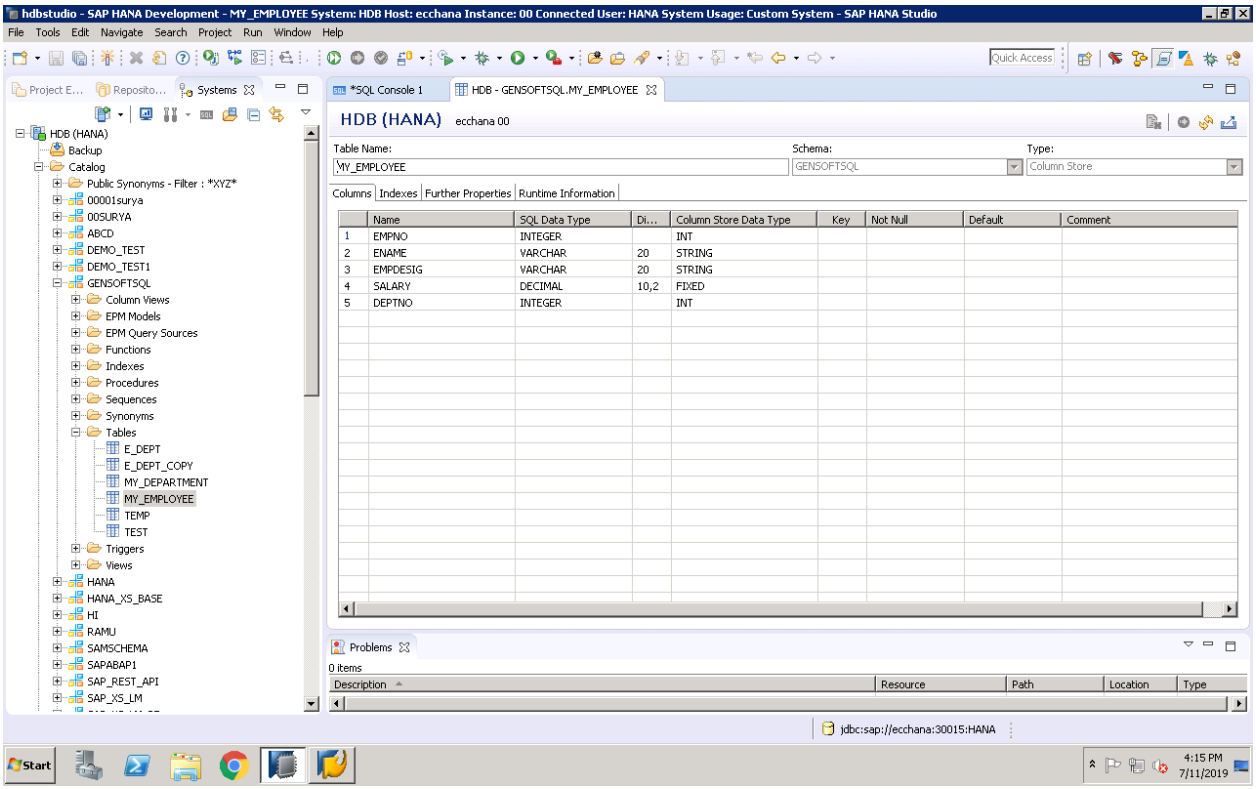
   c) Insert the Manipulated data to the new table E_DEPT_COPY (using INSERT Statement)

STORED PROCEDURES	FUNCTIONS
1. Can be used for READ / WRITE Operations (DML Statements are also allowed)	1. Can be used only for READ Operations (Only Select Queries are allowed)
2. May or May not Return a value	2. Must return a value
3. Can Contain both IN and OUT Parameters	3. By Default, all are IN Parameters



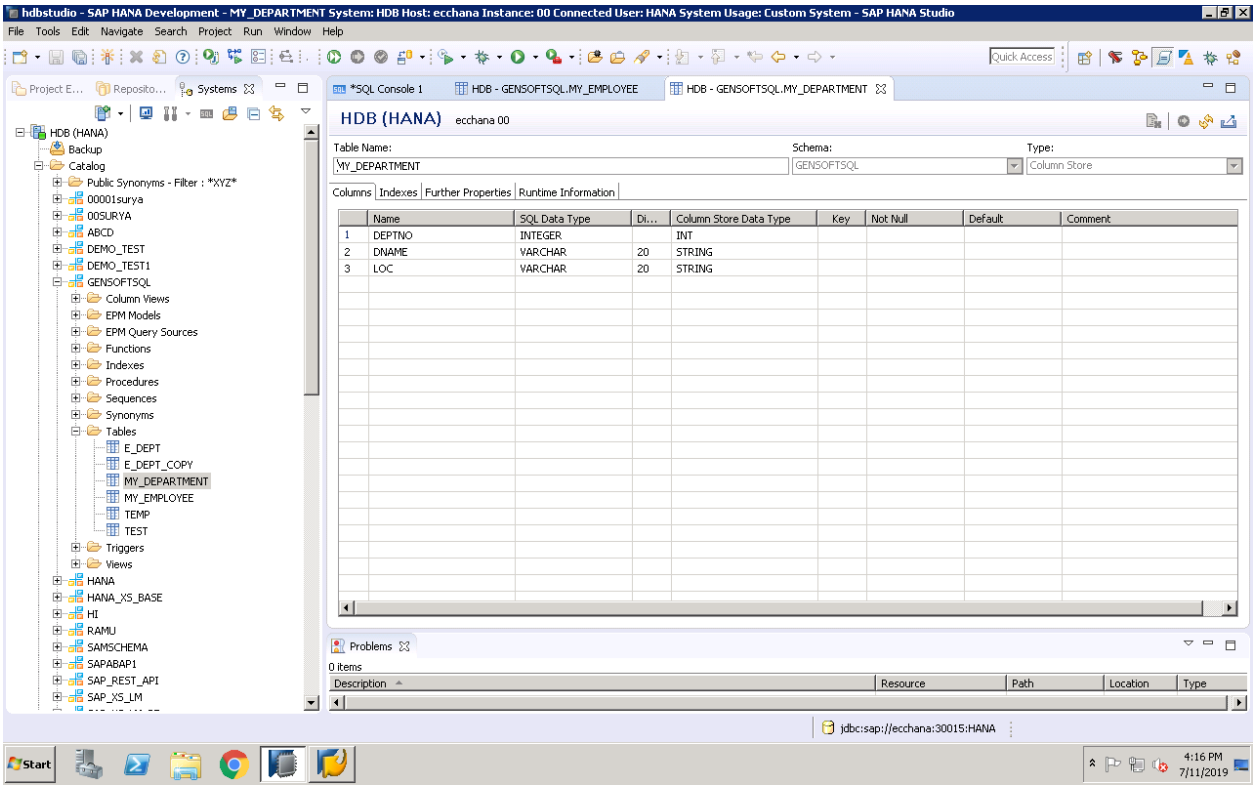
11.07.2019

Create following 2 DB tables with data:




```
INSERT INTO "MY_EMPLOYEE" VALUES(1,'RAJ1','MANAGER',1000,10);
```

```
INSERT INTO "MY_EMPLOYEE" VALUES(2,'RAJ2','EMPLOYEE',2000,20);
INSERT INTO "MY_EMPLOYEE" VALUES(3,'RAJ3','SUPERVISOR',3000,30);
INSERT INTO "MY_EMPLOYEE" VALUES(4,'RAJ4','MANAGER',4000,10);
INSERT INTO "MY_EMPLOYEE" VALUES(5,'RAJ5','EMPLOYEE',5000,20);
INSERT INTO "MY_EMPLOYEE" VALUES(6,'RAJ6','SUPERVISOR',6000,30);
INSERT INTO "MY_EMPLOYEE" VALUES(7,'RAJ7','MANAGER',7000,10);
INSERT INTO "MY_EMPLOYEE" VALUES(8,'RAJ8','EMPLOYEE',8000,20);
INSERT INTO "MY_EMPLOYEE" VALUES(9,'RAJ9','SUPERVISOR',9000,50);
INSERT INTO "MY_EMPLOYEE" VALUES(10,'RAJ10','MANAGER',10000,30);
INSERT INTO "MY_EMPLOYEE" VALUES(11,'RAJ11','EMPLOYEE',11000,30);
INSERT INTO "MY_EMPLOYEE" VALUES(12,'RAJ12','SUPERVISOR',12000,10);
INSERT INTO "MY_EMPLOYEE" VALUES(13,'RAJ13','MANAGER',13000,20);
INSERT INTO "MY_EMPLOYEE" VALUES(14,'RAJ14','EMPLOYEE',14000,50);
INSERT INTO "MY_EMPLOYEE" VALUES(15,'RAJ15','SUPERVISOR',15000,30);
```




```
INSERT INTO "MY_DEPARTMENT" VALUES(10,'SALES','MUMBAI');
INSERT INTO "MY_DEPARTMENT" VALUES(20,'FINANCE','HYDERABAD');
INSERT INTO "MY_DEPARTMENT" VALUES(30,'HR','PUNE');
INSERT INTO "MY_DEPARTMENT" VALUES(40,'MANUFACTURING','CHENNAI');
```

ORDER BY CLAUSE: It sorts the data in ascending / descending order based on the given order by field. Default is ascending order.

```
SELECT * FROM "MY_EMPLOYEE" ORDER BY SALARY;
```

```
SELECT * FROM "MY_EMPLOYEE" ORDER BY SALARY DESC;

SELECT "EMPNO","DEPTNO","SALARY" FROM "MY_EMPLOYEE" ORDER BY 2;

SELECT "ENAME","DEPTNO","SALARY" FROM "MY_EMPLOYEE" ORDER BY
"ENAME","SALARY";

SELECT "ENAME","DEPTNO","SALARY" FROM "MY_EMPLOYEE" ORDER BY
"ENAME","SALARY" DESC;

SELECT "DEPTNO","SALARY" FROM "MY_EMPLOYEE" ORDER BY
"DEPTNO","SALARY";

SELECT "DEPTNO","SALARY" FROM "MY_EMPLOYEE" ORDER BY "DEPTNO","SALARY"
DESC;

SELECT "DEPTNO","SALARY" FROM "MY_EMPLOYEE" ORDER BY "DEPTNO"
DESC,"SALARY" DESC;

SELECT "DEPTNO","SALARY" FROM "MY_EMPLOYEE" ORDER BY "DEPTNO"
DESC,"SALARY";

SELECT "ENAME","SALARY","SALARY" * 1.5 AS "HIKESALARY"
                 FROM "MY_EMPLOYEE"
               WHERE "SALARY" * 1.5 > 5000 ORDER BY "HIKESALARY";

SELECT "ENAME","SALARY","SALARY" * 1.5 AS "HIKESALARY"
                 FROM "MY_EMPLOYEE"
           WHERE "SALARY" * 1.5 > 5000 ORDER BY "HIKESALARY" DESC;
```

GROUP BY CLAUSE: It forms the subsets of the rows based on the given group by field. We generally use Group by Clause whenever we use aggregate functions (MAX,MIN,AVG,COUNT,SUM) in the select query.

```
SELECT SUM("SALARY") FROM "MY_EMPLOYEE";

SELECT SUM("SALARY") FROM "MY_EMPLOYEE" GROUP BY "DEPTNO";

SELECT "DEPTNO",SUM("SALARY") FROM "MY_EMPLOYEE" GROUP BY "DEPTNO";

SELECT "DEPTNO",SUM("SALARY") FROM "MY_EMPLOYEE" GROUP BY "DEPTNO"
ORDER BY "DEPTNO";
```

HAVING CLAUSE: It is used to check the conditions on the aggregated columns in the select query.

```
SELECT "DEPTNO",SUM("SALARY") FROM "MY_EMPLOYEE" GROUP BY "DEPTNO"
WHERE SUM("SALARY") > 25000; // ERROR

SELECT "DEPTNO",SUM("SALARY") FROM "MY_EMPLOYEE" GROUP BY "DEPTNO"
HAVING SUM("SALARY") > 25000;

SELECT "DEPTNO",SUM("SALARY") AS "TOTAL SUM" FROM "MY_EMPLOYEE" GROUP
BY "DEPTNO" HAVING SUM("SALARY") > 25000;

SELECT "DEPTNO",SUM("SALARY") AS "TOTAL SUM" FROM "MY_EMPLOYEE" HAVING
SUM("SALARY") > 25000; // ERROR

SELECT "DEPTNO",SUM("SALARY")
   FROM "MY_EMPLOYEE" GROUP BY "DEPTNO" HAVING SUM("SALARY") > 25000
ORDER BY DEPTNO DESC;

SELECT "DEPTNO",SUM("SALARY")
   FROM "MY_EMPLOYEE" GROUP BY "DEPTNO" HAVING SUM("SALARY") > 25000
ORDER BY DEPTNO;
```

JOINS IN SAP HANA:

Joins are used as part of Select Statements to Query the data from two or more tables

Types of Joins:

1. **INNER JOIN:** This is used whenever we want to return all the values for at-least one match in both the tables. i.e It returns common records between 2 tables

2. **LEFT OUTER JOIN:** This is used whenever we want to return all values from the left table and their corresponding matched values from the right table

3. **RIGHT OUTER JOIN:** This is used whenever we want to return all values from the right table and their corresponding matched values from the left table

4. **FULL OUTER JOIN / FULL JOIN:** This is used whenever we want to return all values from Table A and Table B. i.e combination of left outer and right outer joins

```
SELECT E.EMPNO,E.ENAME,E.EMPDESIG,E.DEPTNO,
       D.DNAME,D.LOC
```

```
        FROM "MY_EMPLOYEE" E INNER JOIN "MY_DEPARTMENT" D
        ON E.DEPTNO = D.DEPTNO;

SELECT E.EMPNO,E.ENAME,E.EMPDESIG,E.DEPTNO,
        D.DNAME,D.LOC
        FROM "MY_EMPLOYEE" E LEFT OUTER JOIN "MY_DEPARTMENT" D
        ON E.DEPTNO = D.DEPTNO;

SELECT E.EMPNO,E.ENAME,E.EMPDESIG,E.DEPTNO,
        D.DNAME,D.LOC
        FROM "MY_EMPLOYEE" E LEFT OUTER JOIN "MY_DEPARTMENT" D
        ON E.DEPTNO = D.DEPTNO
        WHERE D.DEPTNO IS NULL;

SELECT E.EMPNO,E.ENAME,E.EMPDESIG,E.DEPTNO,
        D.DNAME,D.LOC
        FROM "MY_EMPLOYEE" E RIGHT OUTER JOIN "MY_DEPARTMENT" D
        ON E.DEPTNO = D.DEPTNO;

SELECT E.EMPNO,E.ENAME,E.EMPDESIG,E.DEPTNO,
        D.DNAME,D.LOC
        FROM "MY_EMPLOYEE" E RIGHT OUTER JOIN "MY_DEPARTMENT" D
        ON E.DEPTNO = D.DEPTNO
        WHERE E.DEPTNO IS NULL;

SELECT E.EMPNO,E.ENAME,E.EMPDESIG,E.DEPTNO,
        D.DNAME,D.LOC
        FROM "MY_EMPLOYEE" E FULL JOIN "MY_DEPARTMENT" D
        ON E.DEPTNO = D.DEPTNO;

SELECT E.EMPNO,E.ENAME,E.EMPDESIG,E.DEPTNO,
        D.DNAME,D.LOC
        FROM "MY_EMPLOYEE" E FULL JOIN "MY_DEPARTMENT" D
        ON E.DEPTNO = D.DEPTNO
        WHERE E.DEPTNO IS NULL OR
              D.DEPTNO IS NULL;
```

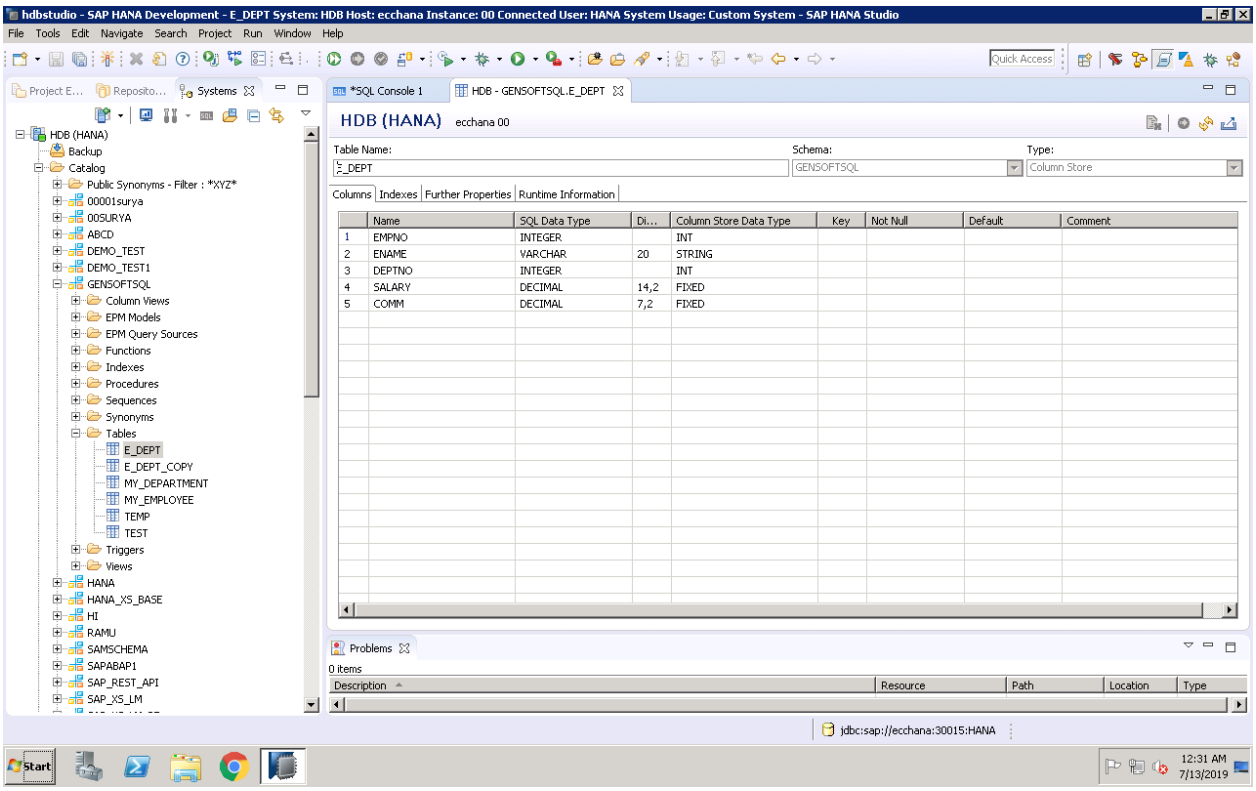
12.07.2019

DATABASE VIEWS: are the virtual tables created on top of one or more database tables. Views are created to implement the security measures like hiding the original table name, original column name, restricted table data access. Views doesn’t hold the data physically, the data will be populated in the runtime by executing the select query on the base table/s of the views. Supports both Projection and Selection.

Projection is a process of giving the access to the required fields and selection is a process of restricting the table data access.

Create / Consider the following DB Tables with data:

Table: E_DEPT



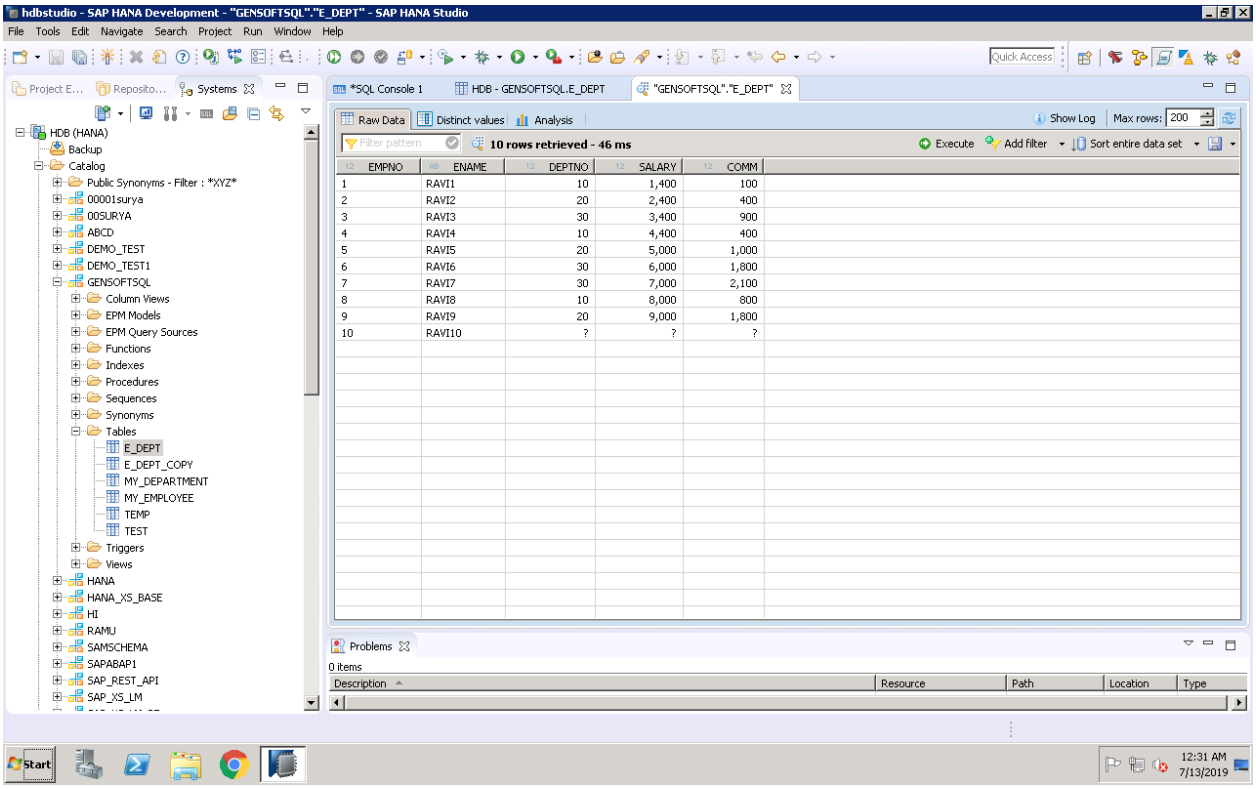
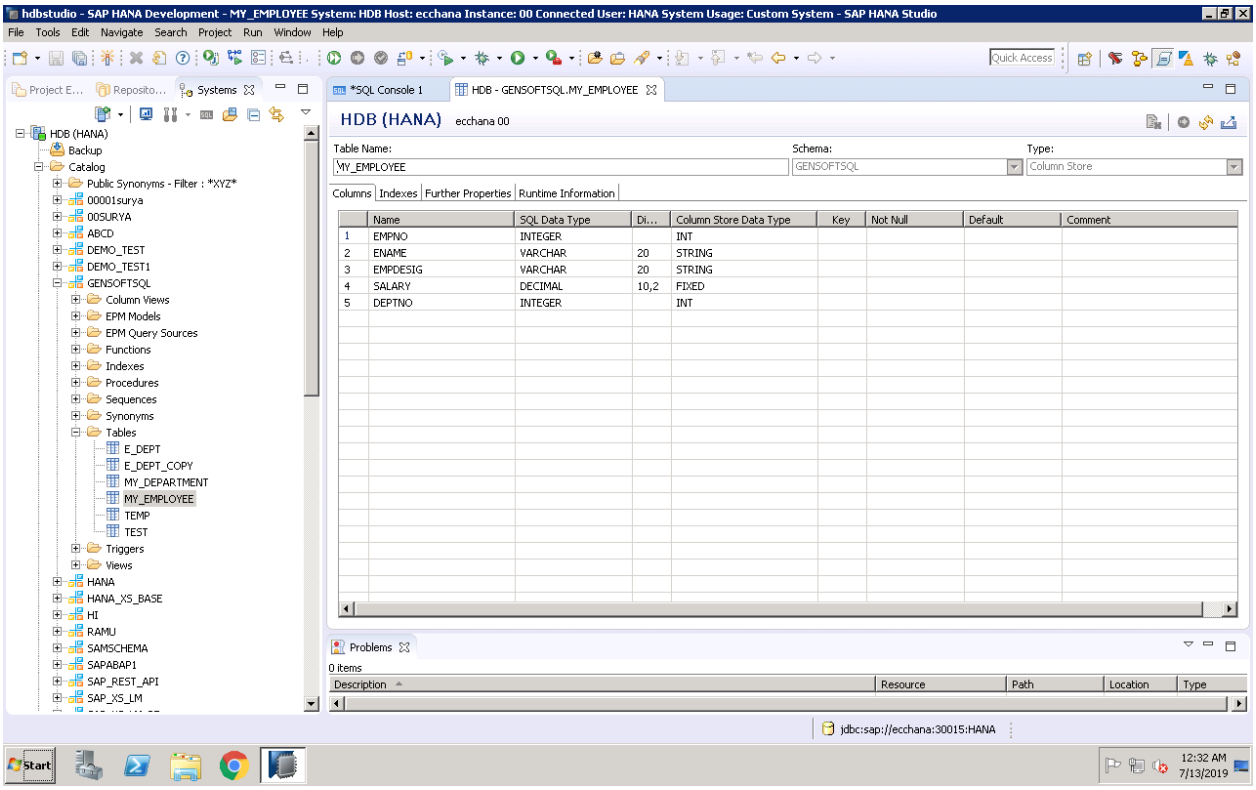


Table: MY_EMPLOYEE



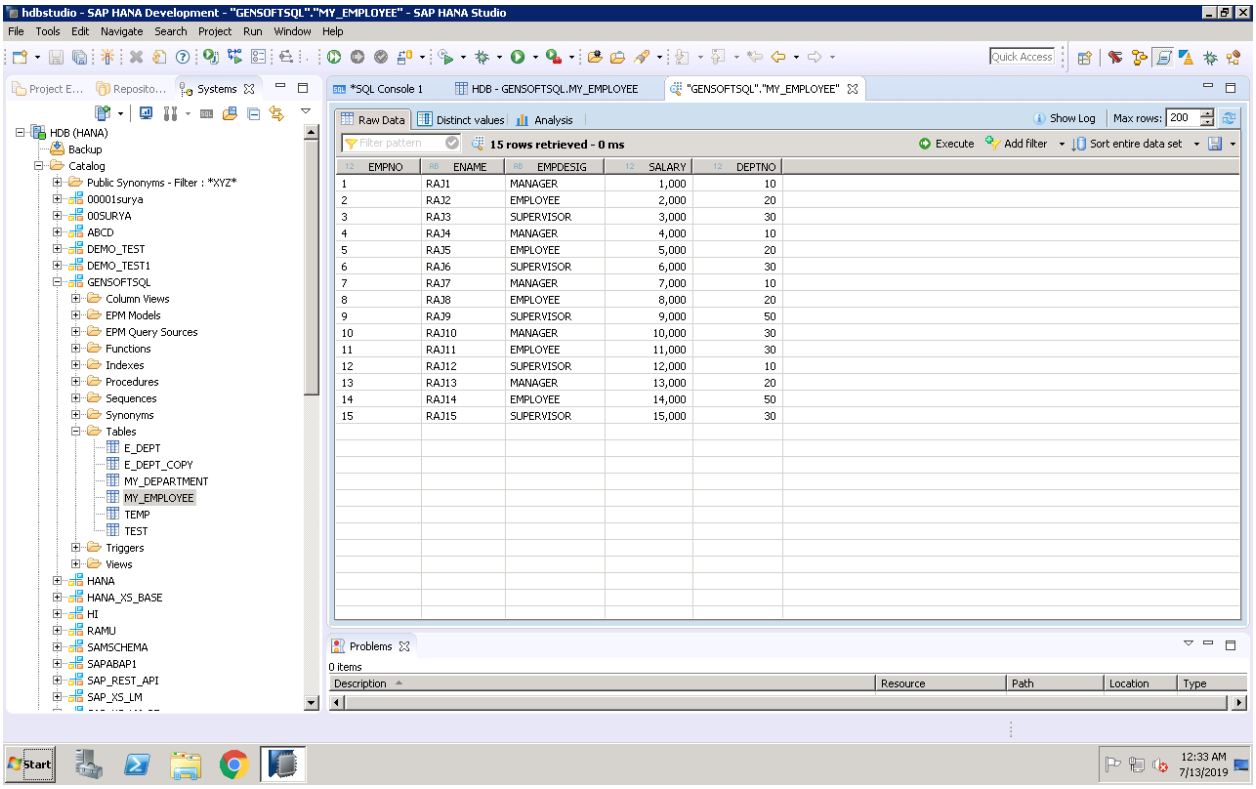
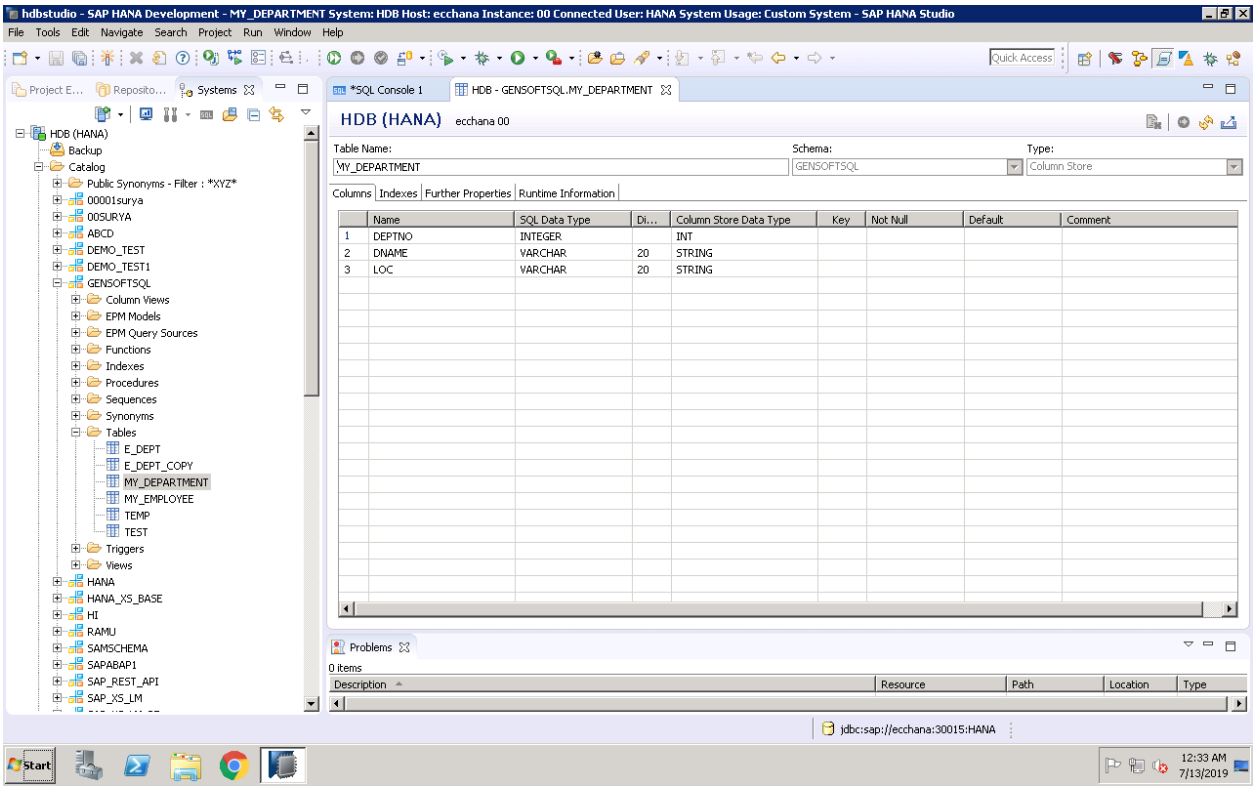
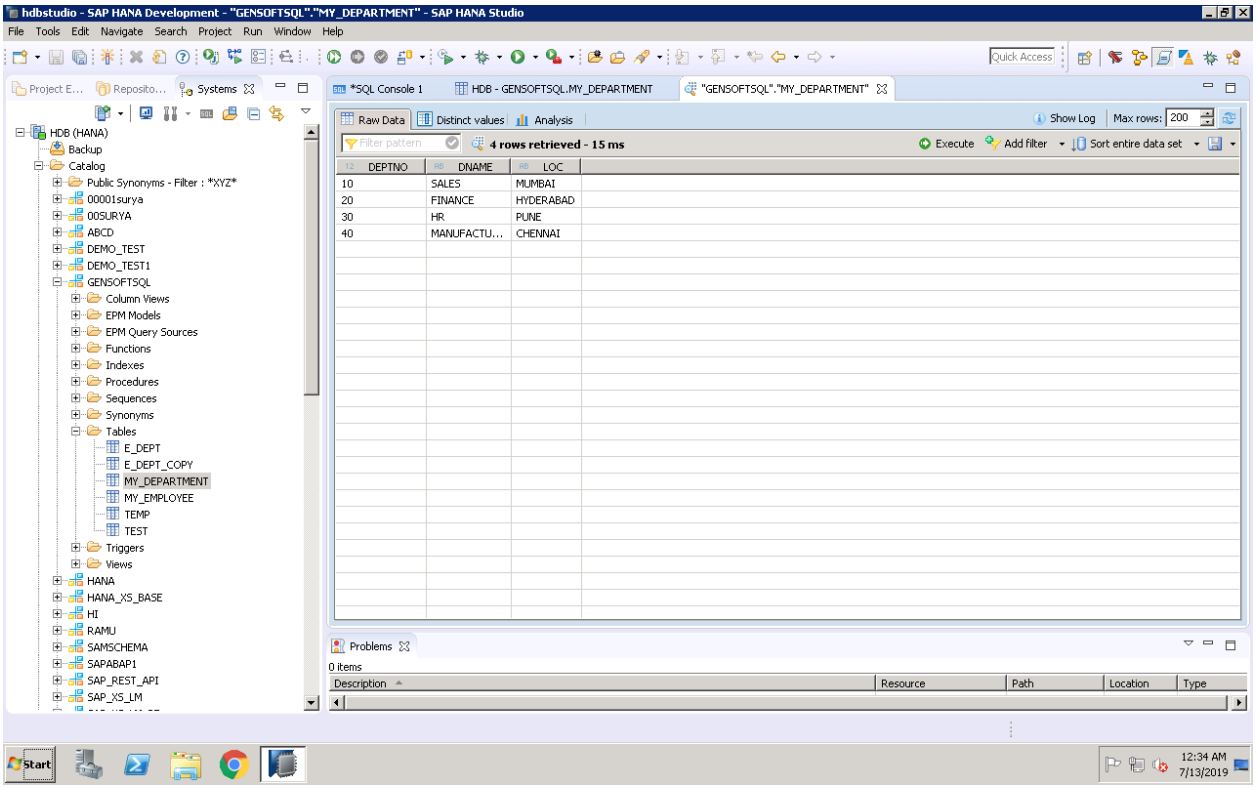


Table: MY_DEPARTMENT






```
CREATE VIEW "EMP_VIEW" AS SELECT * FROM "E_DEPT";

SELECT * FROM "EMP_VIEW";

CREATE VIEW "EMP_VIEW2" AS SELECT "EMPNO","ENAME","DEPTNO" FROM
"E_DEPT";

SELECT * FROM "EMP_VIEW2";

CREATE VIEW "EMP_VIEW3" AS SELECT "EMPNO","ENAME","SALARY" FROM
"E_DEPT" WHERE "SALARY" > 6000;

SELECT * FROM "EMP_VIEW3";

CREATE VIEW "EMP_VIEW4"("EMPID","EMPNAME","EMPSAL") AS SELECT
"EMPNO","ENAME","SALARY" FROM "E_DEPT" WHERE "SALARY" > 6000;

SELECT * FROM "EMP_VIEW4";

CREATE VIEW "EMP_DEPT" AS SELECT "EMPNO","ENAME","SALARY"
                                 "DNAME","LOC"
                                 FROM "MY_EMPLOYEE" AS "E"
                                      INNER JOIN
                                      "MY_DEPARTMENT" AS "D"
```

```
                                        ON E.DEPTNO = D.DEPTNO;

SELECT * FROM "EMP_DEPT";

CREATE VIEW "EMP_DEPT2" AS SELECT "EMPNO","ENAME","SALARY"
                                "DEPTNO","DNAME","LOC"
                                FROM "MY_EMPLOYEE" AS "E"
                                     INNER JOIN
                                     "MY_DEPARTMENT" AS "D"
                                       ON E.DEPTNO = D.DEPTNO;

SELECT * FROM "EMP_DEPT2";

CREATE VIEW "EMP_DEPT3" AS SELECT "EMPNO","ENAME","SALARY",
                                "DNAME","LOC"
                                FROM "MY_EMPLOYEE" AS "E"
                                     INNER JOIN
                                     "MY_DEPARTMENT" AS "D"
                                       ON E.DEPTNO = D.DEPTNO;

SELECT * FROM "EMP_DEPT3";


CREATE VIEW "EMP_DEPT4" AS SELECT "EMPNO","ENAME","SALARY",
                                "DEPTNO","DNAME","LOC"
                                FROM "MY_EMPLOYEE" AS "E"
                                     INNER JOIN
                                     "MY_DEPARTMENT" AS "D"
     ON E.DEPTNO = D.DEPTNO; // ERROR as ambuigity in reading deptno

CREATE VIEW "EMP_DEPT4" AS SELECT "EMPNO","ENAME","SALARY",
                                D.DEPTNO,"DNAME","LOC"
                                FROM "MY_EMPLOYEE" AS "E"
                                     INNER JOIN
                                     "MY_DEPARTMENT" AS "D"
                                       ON E.DEPTNO = D.DEPTNO;

SELECT * FROM "EMP_DEPT4";

DROP VIEW "EMP_DEPT";

DROP VIEW "EMP_DEPT2";

CREATE VIEW "EMP_DEPT5" AS SELECT "EMPNO","ENAME","SALARY",
                                D.DEPTNO,"DNAME","LOC"
                                FROM "MY_EMPLOYEE" AS "E"
```

```
                                        LEFT OUTER JOIN
                                        "MY_DEPARTMENT" AS "D"
                                          ON E.DEPTNO = D.DEPTNO;

SELECT * FROM "EMP_DEPT5";
```

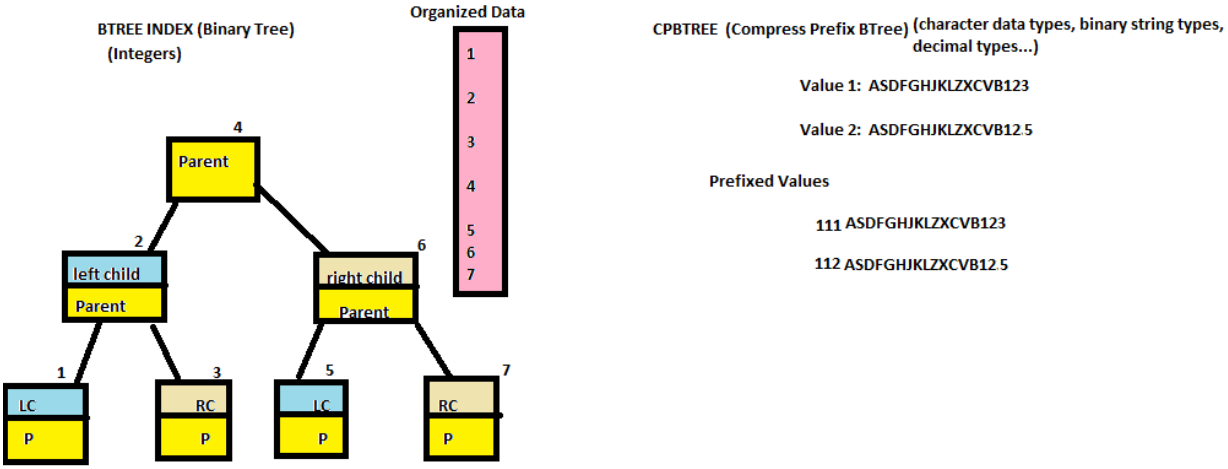
INDEXES: are created on top of database tables to arrange the table data in an order (ascending / descending) which will improve the performance while querying the data

Note: The arrangement of data using indexes is done by using either of the following algorithms depending on the indexed column data types

1. **UNIQUE INDEX** → Will be created by SAP based on primary key constraint
2. **BTREE INDEX (BINARY TREE)** → Will be used for columns having Integer data types
3. **CPBTREE INDEX (COMPRESS PREFIX B TREE)** → Will be used when column data types are character string types, binary string types, decimal types, when the constraint is a composite key, or a non-unique constraint.

```
SELECT * FROM "SYS"."INDEXES" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'E_DEPT';

ALTER TABLE "E_DEPT" DROP PRIMARY KEY;

SELECT * FROM "SYS"."INDEXES" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'E_DEPT';

CREATE UNIQUE INDEX "IDX1" ON "E_DEPT"("EMPNO");

SELECT * FROM "SYS"."INDEXES" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'E_DEPT';

DROP INDEX "IDX1";

SELECT * FROM "SYS"."INDEXES" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'E_DEPT';

CREATE BTREE INDEX "IDX2" ON "E_DEPT"("EMPNO");

SELECT * FROM "SYS"."INDEXES" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'E_DEPT';
```

```
CREATE BTREE INDEX "IDX3" ON "E_DEPT"("DEPTNO");

SELECT * FROM "SYS"."INDEXES" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'E_DEPT';

CREATE BTREE INDEX "IDX4" ON "E_DEPT"("ENAME"); // ERROR AS ENAME IS
STRING DATA TYPE

CREATE BTREE INDEX "IDX4" ON "E_DEPT"("SALARY"); // ERROR AS SALARY
BELONGS TO STRING DATA TYPE FAMILY

CREATE CPBTREE INDEX "IDX4" ON "E_DEPT"("ENAME");

SELECT * FROM "SYS"."INDEXES" WHERE SCHEMA_NAME = 'GENSOFTSQL' AND
TABLE_NAME = 'E_DEPT';
```



15-07-2019

ADBC (ABAP DATABASE CONNECTIVITY)

ADBC (ABAP DATABASE CONNECTIVITY): is an API used for interacting with the databases using Native SQL statements using ABAP Objects.

Steps to use ADBC:

1) Establish the DB Connection (CL_SQL_CONNECTION=>GET_CONNECTION)

2) Instantiate the Statement Object (CL_SQL_STATEMENT)

3) Construct the SQL Query which is compatible with HANA Studio

4) Execute Native SQL Call by calling appropriate methods

i) EXECUTE_QUERY ---> for executing Select Statements

ii) EXECUTE_DDL ---> For executing DDL Statements (CREATE, DROP, ALTER)

iii) EXECUTE_UPDATE --> For executing DML Statements (INSERT, UPDATE, DELETE)

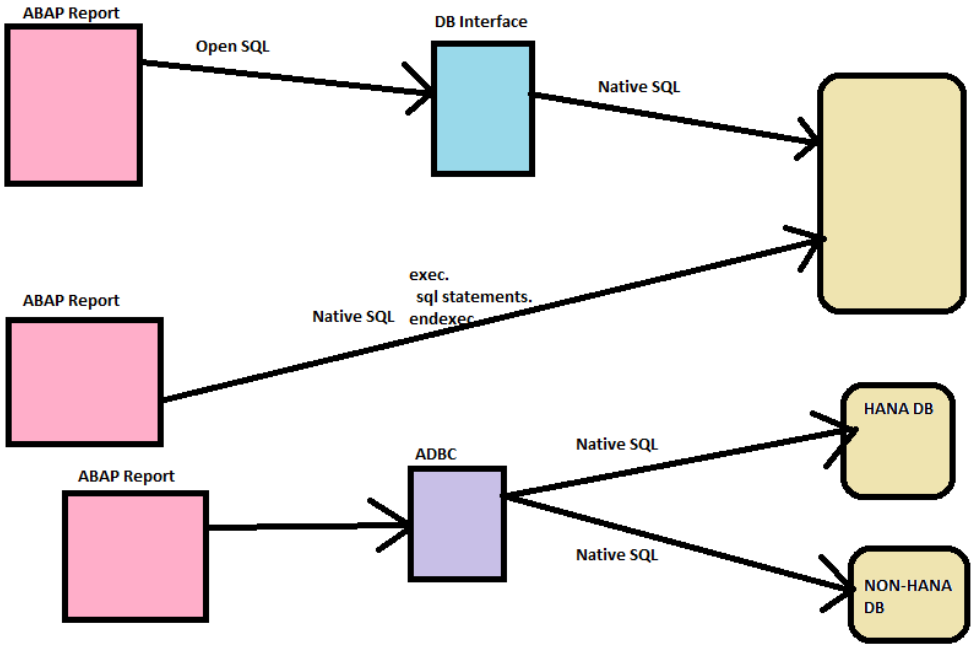
iv) EXECUTE_PROCEDURE --> For executing Stored Procedures

5) Assign Target variable for Result SET (CL_SQL_RESULT_SET--> SET_PARAM _STRUCT/ SET_PARAM / SET_PARAM_TABLE....)

6) Retrieve Result Set (CL_SQL_RESULT_SET=>NEXT_PACKAGE)

7) Close the Query

8) Close database connection



Example: ABAP Report to Display Customer Master Data by executing SQL Query (SELECT) using ADBC

```
REPORT Z6AM_ADBC1.
```

```
* Establish the DB connection (HANA DB)
data : o_con type ref to cl_sql_connection,
      o_sql_excp type ref to cx_sql_exception,
      v_msg type string.

TRY.
CALL METHOD cl_sql_connection=>get_connection
  RECEIVING
    con_ref  = o_con.
CATCH cx_sql_exception into o_sql_excp.
* Capture standard exception long message
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in Establishing DB connection :',v_msg.
ENDTRY.

if o_con is bound. "not initial
  write :/ 'Successfully connected to underlying database (HDB)'.
* Instantiate the statement class on top of DB connection
  data o_stmt type ref to cl_sql_statement.
  CREATE OBJECT o_stmt
    EXPORTING
      con_ref = o_con.
endif.

if o_stmt is bound. "not initial.
* Construct the sql query containing select statement
  write :/ 'Statement object instantiated....'.
data v_sql type string.
v_sql = 'select * from kna1'.

* Execute Select Query
data : o_result type ref to CL_SQL_RESULT_SET,
       o_param type ref to CX_PARAMETER_INVALID.

TRY.
CALL METHOD o_stmt->execute_query
  EXPORTING
    statement   = v_sql
  RECEIVING
```

```
    result_set  = o_result.
CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in Executing select query :',v_msg.
 CATCH cx_parameter_invalid into o_param.
   clear v_msg.
   CALL METHOD o_param->if_message~get_longtext
     RECEIVING
       result            = v_msg.

   write :/ 'Error in Parameter binding ....',v_msg.
ENDTRY.

if o_result is bound. "not initial
write :/ 'Select Query successfully executed...'.

* set the Internal table parameter
data t_kna1 type table of kna1.
data o_data type ref to DATA.
get REFERENCE OF t_kna1 into o_data.

* set the internal table as out parameter
TRY.
CALL METHOD o_result->set_param_table
  EXPORTING
    itab_ref             = o_data.
CATCH cx_parameter_invalid into o_param.
  clear v_msg.
  CALL METHOD o_param->if_message~get_longtext
    RECEIVING
      result            = v_msg.

  write :/ 'Error in setting internal table as OUT parameter ...',v_ms
g.
ENDTRY.

* Retrieve result set
data o_param_type type ref to CX_PARAMETER_INVALID_TYPE.
```

```
TRY.
CALL METHOD o_result->next_package.
 CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

   write :/ 'Error in reading result set :',v_msg.
 CATCH cx_parameter_invalid_type into o_param_type.
   clear v_msg.
   CALL METHOD o_param_type->if_message~get_longtext
     RECEIVING
       result            = v_msg.

write :/ 'Error in parameter type while reading result set :',v_msg.

ENDTRY.

* close the result set
call method o_result->close.

* close the DB connection
TRY.
CALL METHOD o_con->close.
CATCH cx_sql_exception into o_sql_excp.
   clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

   write :/ 'Error in closing DB connection :',v_msg.
ENDTRY.

if t_kna1 is not INITIAL.
  data wa_kna1 type kna1.
  loop at t_kna1 into wa_kna1.
    write :/ wa_kna1-kunnr,
             wa_kna1-land1,
             wa_kna1-name1.
  endloop.
endif.
```

```
endif. "endif for result set
endif. "endif for statement
```

16.07.2019

Example: Binding Parameters to Queries using ADBC

```
REPORT z6am_adbc2.

PARAMETERS p_kunnr TYPE kna1-kunnr.

PARAMETERS : p_fields RADIOBUTTON GROUP grp1,
             p_wa     RADIOBUTTON GROUP grp1,
             p_none   RADIOBUTTON GROUP grp1 DEFAULT 'X'.

DATA : v_kunnr TYPE kna1-kunnr,
       v_land1 TYPE kna1-land1,
       v_name1 TYPE kna1-name1.

TYPES : BEGIN OF ty_kna1,
          kunnr TYPE kna1-kunnr,
          land1 TYPE kna1-land1,
          name1 TYPE kna1-name1,
        END OF ty_kna1.

DATA wa_kna1 TYPE ty_kna1.

* Establish the DB connection to the underlying database
DATA : o_con      TYPE REF TO cl_sql_connection,
       o_sql_excp TYPE REF TO cx_sql_exception,
       v_msg      TYPE string.

TRY.
    CALL METHOD cl_sql_connection=>get_connection
      RECEIVING
        con_ref = o_con.
  CATCH cx_sql_exception INTO o_sql_excp.
    CALL METHOD o_sql_excp->if_message~get_longtext
      RECEIVING
        result = v_msg.

    WRITE :/ 'Error in Establishing DB connection :',v_msg.
ENDTRY.
```

```
IF o_con IS BOUND.
* Instantiate the statement class based on DB connection
  DATA o_stmt TYPE REF TO cl_sql_statement.
  CREATE OBJECT o_stmt
    EXPORTING
      con_ref = o_con.
ENDIF.

IF o_stmt IS BOUND.
* Bind the parameter of select query as IN Parameter
  DATA o_param TYPE REF TO cx_parameter_invalid.
  TRY.
      CALL METHOD o_stmt->set_param
        EXPORTING
*         data_ref = p_kunnr. "syntax error
          data_ref = REF #( p_kunnr ).
    CATCH cx_parameter_invalid INTO o_param.
      CLEAR v_msg.
      CALL METHOD o_param->if_message~get_longtext
        RECEIVING
          result = v_msg.

      WRITE :/ 'Error in binding kunnr as IN Parameter :',v_msg.
  ENDTRY.

* Execute the select query
  DATA o_result TYPE REF TO cl_sql_result_set.
  TRY.
      CALL METHOD o_stmt->execute_query
        EXPORTING
          statement  = `SELECT KUNNR, LAND1, NAME1 ` && `FROM KNA1 ` &
& `WHERE KUNNR = ?`
        RECEIVING
          result_set = o_result.
    CATCH cx_sql_exception INTO o_sql_excp.
      CLEAR v_msg.
      CALL METHOD o_sql_excp->if_message~get_longtext
        RECEIVING
          result = v_msg.

      WRITE :/ 'Error in Executing Query :',v_msg.
```

```
    CATCH cx_parameter_invalid INTO o_param.
      CLEAR v_msg.
      CALL METHOD o_param->if_message~get_longtext
        RECEIVING
          result = v_msg.

WRITE :/ 'Error in executing and binding kunnr as IN Para
meter :',v_msg.
  ENDTRY.

  IF o_result IS BOUND.
    IF p_fields = 'X'. "Read the return values field by field

* Register / set the individual fields as OUT parameters
      TRY.
          CALL METHOD o_result->set_param
            EXPORTING
              data_ref = REF #( v_kunnr ).
        CATCH cx_parameter_invalid INTO o_param.
          CLEAR v_msg.
          CALL METHOD o_param->if_message~get_longtext
            RECEIVING
              result = v_msg.

   WRITE :/ 'Error in registering v_kunnr as OUT parameter :',v_msg.

      ENDTRY.

      TRY.
          CALL METHOD o_result->set_param
            EXPORTING
              data_ref = REF #( v_land1 ).
        CATCH cx_parameter_invalid INTO o_param.
          CLEAR v_msg.
          CALL METHOD o_param->if_message~get_longtext
            RECEIVING
              result = v_msg.

  WRITE :/ 'Error in registering v_land1 as OUT parameter :',v_msg.

      ENDTRY.
```

```
      TRY.
          CALL METHOD o_result->set_param
            EXPORTING
              data_ref = REF #( v_name1 ).
        CATCH cx_parameter_invalid INTO o_param.
          CLEAR v_msg.
          CALL METHOD o_param->if_message~get_longtext
            RECEIVING
              result = v_msg.

 WRITE :/ 'Error in registering v_name1 as OUT parameter :',v_msg.

      ENDTRY.

* Read the record from result set
      DATA o_param_type TYPE REF TO cx_parameter_invalid_type.
      DATA v_count TYPE i.
      TRY.
          CALL METHOD o_result->next
            RECEIVING
              rows_ret = v_count.
        CATCH cx_sql_exception INTO o_sql_excp.
          CLEAR v_msg.
          CALL METHOD o_sql_excp->if_message~get_longtext
            RECEIVING
              result = v_msg.

          WRITE :/ 'Error in Reading Record :',v_msg.

        CATCH cx_parameter_invalid_type INTO o_param_type.
          CLEAR v_msg.
          CALL METHOD o_param_type->if_message~get_longtext
            RECEIVING
              result = v_msg.

          WRITE :/ 'Error in Reading fields ',v_msg.
      ENDTRY.

      IF v_count > 0.
        CLEAR v_msg.
        v_msg = |Result is { v_kunnr } : { v_land1 } : { v_name1 }|.
      ENDIF.
```

```
    ELSEIF p_wa = 'X'. "Read the return values as a work area
* Register Work area (structure) as OUT parameter
      TRY.
          CALL METHOD o_result->set_param_struct
            EXPORTING
              struct_ref = REF #( wa_kna1 ).
        CATCH cx_parameter_invalid INTO o_param.
          CLEAR v_msg.
          CALL METHOD o_param->if_message~get_longtext
            RECEIVING
              result = v_msg.

 WRITE :/ 'Error in registering work area as OUT parameter :',v_msg.
      ENDTRY.

* Read the record from result set
      TRY.
          CALL METHOD o_result->next
            RECEIVING
              rows_ret = v_count.
        CATCH cx_sql_exception INTO o_sql_excp.
          CLEAR v_msg.
          CALL METHOD o_sql_excp->if_message~get_longtext
            RECEIVING
              result = v_msg.

          WRITE :/ 'Error in Reading Record :',v_msg.

        CATCH cx_parameter_invalid_type INTO o_param_type.
          CLEAR v_msg.
          CALL METHOD o_param_type->if_message~get_longtext
            RECEIVING
              result = v_msg.

          WRITE :/ 'Error in Reading fields ',v_msg.
      ENDTRY.

      IF v_count > 0.
        CLEAR v_msg.
 v_msg = |Result is { wa_kna1-kunnr } : { wa_kna1-land1 } : { wa_kna1-name1 }|.
      ENDIF.
    ENDIF.
    MESSAGE v_msg TYPE 'I'.
```

```
* close the result set
  o_result->close( ).
  ENDIF.

* close the db connection
  o_con->close( ).


ENDIF. "STATEMENT
```

17-07.2019

Example: Consuming HANA Stored Procedure from ABAP Reports using ADBC Framework

HANASTUDIO:

1. In Development Perspective, Create the Stored Procedure ‘ADDITION’ as part of ‘SAPABAP1’ schema (Configured Schema with Application Server) in SQL Console

```
CREATE PROCEDURE "SAPABAP1".ADDITION(IN X INTEGER,IN Y INTEGER,OUT Z
INTEGER)
AS
BEGIN
 Z=X+Y;
END; // Execute, Creates Procedure in SAPABAP1 Schema

CALL "SAPABAP1"."ADDITION"(?); // Successfully calls procedure
```

Executable Program: For Consuming above HANA Stored Procedure (SAPGUI or Open SAPGUI in HANASTUDIO)

```
REPORT Z6AM_ADBC3.

PARAMETERS : p_x type i,
             p_y type i.

data v_res type i.

* Establish the DB connection to underlying database
data : o_con type ref to cl_sql_connection,
       o_sql_excp type ref to cx_sql_exception,
       v_msg type string.
```

```
TRY.
CALL METHOD cl_sql_connection=>get_connection
  RECEIVING
    con_ref  = o_con.
CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in connecting to DB :',v_msg.
ENDTRY.

if o_con is bound.
* Instantiate the statement class on top of DB connection
  data o_stmt type ref to cl_sql_statement.
  CREATE OBJECT o_stmt
    EXPORTING
      con_ref = o_con.
endif.

if o_stmt is bound.
* bind the parameters of the procedure
  data o_param type ref to CX_PARAMETER_INVALID.

* bind the first INPUT parameter of the stored procedure TESTADDITION
 TRY.
  CALL METHOD o_stmt->set_param
    EXPORTING
      data_ref = REF #( p_x ).
 CATCH cx_parameter_invalid into o_param.
  clear v_msg.
  CALL METHOD o_param->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in binding first input parameter :',v_msg.
 ENDTRY.

*bind the second INPUT parameter of the stored procedure TESTADDITION
 TRY.
```

```
  CALL METHOD o_stmt->set_param
    EXPORTING
      data_ref = REF #( p_y ).
 CATCH cx_parameter_invalid into o_param.
  clear v_msg.
  CALL METHOD o_param->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in binding second input parameter :',v_msg.
 ENDTRY.

* bind the OUT parameter of the stored procedure TESTADDITION
 TRY.
  CALL METHOD o_stmt->set_param
    EXPORTING
      data_ref = REF #( v_res )
      INOUT    = CL_SQL_STATEMENT=>C_PARAM_OUT.
 CATCH cx_parameter_invalid into o_param.
  clear v_msg.
  CALL METHOD o_param->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in binding OUT parameter :',v_msg.
 ENDTRY.

* Execute procedure
TRY.
CALL METHOD o_stmt->execute_procedure
  EXPORTING
    proc_name      = 'ADDITION'.

write :/ 'Sum is ',v_res.

CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_text
    RECEIVING
      result = v_msg.

  write :/ 'Error in executing stored procedure - short exception  msg
```

```
:',v_msg.  .

  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in executing stored procedure - long exception
msg :',v_msg.  .
ENDTRY.

* close the connection
TRY.
CALL METHOD o_con->close.
CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in closing database connection  :',v_msg.  ..
ENDTRY.

endif.
```

18.07.2019

Example: Consuming HANA Stored Procedure from ABAP Report using ADBC Framework

HANA Studio:

1. In Development Perspective, Under Systems Tab, Right Click on 'Content' folder and create a package (ZADBCPACK)
2. In Development Perspective, Under Repositories Tab, Right Click on Package and Create the stored procedure and store the same in '_SYS_BIC' Schema

Design Time Object:

```
CREATE PROCEDURE _SYS_BIC.GETSALESDOCS ( OUT ITAB "SAPABAP1"."VBAK" )
      LANGUAGE SQLSCRIPT
      SQL SECURITY INVOKER
      DEFAULT SCHEMA SAPABAP1
```

```
     READS SQL DATA AS
BEGIN
   ITAB=SELECT * FROM "VBAK";
                    END; // Execute, Creates the procedure
```

Run Time Object: From '_SYS_BIC' Schema

```
create procedure "_SYS_BIC"."ZADBCPACK/GETSALESDOCS" (  out ITAB
"SAPABAP1"."VBAK"  ) language SQLSCRIPT sql security invoker default
schema "SAPABAP1" reads sql data as
BEGIN
   ITAB=SELECT * FROM "VBAK";
                                    END;
```

Calling Procedure from SQL Console:

```
CALL "_SYS_BIC"."ZADBCPACK/GETSALESDOCS"(?); → Successfully executes
procedure and returns data from the VBAK Table

CALL "_SYS_BIC"."ZADBCPACK/GETSALESDOCS"(?) WITH OVERVIEW; →
Successfully executes procedure and returns the information of the
temporary internal table which holds the return values of the
procedure

SELECT * FROM "HANA"."ITAB_B7EFDADB5A564E4583E13437779DC38E"; →
Returns data from temporary internal table
```

Calling ABOVE Procedure from ABAP Report using ADBC Framework:

```
REPORT Z6AM_ADBC4.


* Establish the DB connection to underlying database
data : o_con type ref to cl_sql_connection,
       o_sql_excp type ref to cx_sql_exception,
       v_msg type string.

TRY.
CALL METHOD cl_sql_connection=>get_connection
  RECEIVING
    con_ref  = o_con.
CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
```

```
      result            = v_msg.

      write :/ 'Error in connecting to DB :',v_msg.
ENDTRY.

if o_con is bound.
  write :/ 'Successfully connected to HANA DB' color 3.
* Instantiate the statement class on top of DB connection
  data o_stmt type ref to cl_sql_statement.
  CREATE OBJECT o_stmt
    EXPORTING
      con_ref = o_con.
endif.

if o_stmt is bound.
write :/ 'Successfully Instantiated Statement Object on Top of DB conn
ection ' color 7.
* Construct the SQL Query
data v_sql type string.
v_sql = 'CALL "_SYS_BIC"."ZADBCPACK/GETSALESDOCS"( null) WITH OVERVIEW
'.

types : begin of ty_overview,
            param type string,
            value type string,
        end of ty_overview.

data : t_overview type table of ty_overview,
      wa_overview type ty_overview.

* Execute Native SQL statement
data : o_result type ref to cl_sql_result_set,
       o_param type ref to cx_parameter_invalid.

TRY.
CALL METHOD o_stmt->execute_query
  EXPORTING
    statement   = v_sql
  RECEIVING
    result_set  = o_result.
CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
```

```
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in executing select query :',v_msg..
CATCH cx_parameter_invalid into o_param.
  clear v_msg.
  CALL METHOD o_param->if_message~get_longtext
    RECEIVING
      result            = v_msg.

write :/ 'Error in parameter binding while executing select query :',v
_msg..
ENDTRY.

if o_result is bound.
write :/ 'Call to procedure is successfully executed ' color 2.
* set the internal table OUT parameter
data o_data type ref to DATA.
get REFERENCE OF t_overview into o_data.

TRY.
CALL METHOD o_result->set_param_table
  EXPORTING
    itab_ref             = o_data.
CATCH cx_parameter_invalid into o_param.
  clear v_msg.
  CALL METHOD o_param->if_message~get_longtext
    RECEIVING
      result            = v_msg.

write :/ 'Error in binding OUT parameter of the result set :',v_msg.
ENDTRY.

* Retrieve Result set
data o_param_type type ref to CX_PARAMETER_INVALID_TYPE.
TRY.
CALL METHOD o_result->next_package.
CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
```

```
      result            = v_msg.

      write :/ 'Error in reading result set :',v_msg..
CATCH cx_parameter_invalid_type into o_param_type.
  clear v_msg.
  CALL METHOD o_param_type->if_message~get_longtext
    RECEIVING
      result            = v_msg.

write :/ 'Error in parameter type while reading result set :',v_msg.
ENDTRY.

* close the result set
call method o_result->close.

* Display the information of temporary internal table
uline.
format color 1.
write :/ 'Information of Temporary internal table retured by procedure
CALL'.
if t_overview is not initial.
  loop at t_overview into wa_overview.
    write :/ wa_overview-param,
             wa_overview-value.
  endloop.
endif.

clear wa_overview.
read table t_overview into wa_overview with key param = 'ITAB'.
if sy-subrc eq 0.
  clear v_sql.
* construct SQL query
  v_sql = ' select * from ' && wa_overview-value.

* execute native sql statement
TRY.
CALL METHOD o_stmt->execute_query
  EXPORTING
    statement   = v_sql
  RECEIVING
    result_set  = o_result.
CATCH cx_sql_exception into o_sql_excp.
```

```
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in executing select query :',v_msg..
CATCH cx_parameter_invalid into o_param.
  clear v_msg.
  CALL METHOD o_param->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in parameter binding while executing select quer
y :',v_msg..
ENDTRY.

types : begin of ty_vbak.
            INCLUDE TYPE vbak.
types end of ty_vbak.

data : t_vbak type table of ty_vbak,
      wa_vbak type ty_vbak.

* set the internal table OUT parameter
get REFERENCE OF t_vbak into o_data.

TRY.
CALL METHOD o_result->set_param_table
  EXPORTING
    itab_ref             = o_data.
CATCH cx_parameter_invalid into o_param.
  clear v_msg.
  CALL METHOD o_param->if_message~get_longtext
    RECEIVING
      result            = v_msg.

write :/ 'Error in binding OUT parameter of the result set :',v_msg.
ENDTRY.

* Retrieve Result set
TRY.
CALL METHOD o_result->next_package.
```

```
CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in reading result set :',v_msg..
CATCH cx_parameter_invalid_type into o_param_type.
  clear v_msg.
  CALL METHOD o_param_type->if_message~get_longtext
    RECEIVING
      result            = v_msg.

write :/ 'Error in parameter type while reading result set :',v_msg.
ENDTRY.

if t_vbak is not INITIAL.
format color 3.
write :/ 'Actual data returned by the procedure...'.
loop at t_vbak into wa_vbak.
  write :/ wa_vbak-vbeln,
           wa_vbak-erdat,
           wa_vbak-erzet,
           wa_vbak-ernam.
endloop.
endif.
endif.
endif. "end of result set
* close the connection
TRY.
CALL METHOD o_con->close.
CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

      write :/ 'Error in closing database connection  :',v_msg.  ..
ENDTRY.

endif. "end of statement
```

19.07.2019

Example: Executing DDL and DML statements using ADBC Framework

```
REPORT Z6AM_ADBC5.

data : o_con type ref to cl_sql_connection,
       o_stmt type ref to cl_sql_statement,
       o_sql_excp type ref to cx_sql_exception,
       o_param type ref to cx_parameter_invalid,
       v_msg type string.

SELECTION-SCREEN begin of block bk1 with FRAME title t1.
  PARAMETERS : p_r1 RADIOBUTTON GROUP grp1,
               p_r2 RADIOBUTTON GROUP grp1,
               p_r3 RADIOBUTTON GROUP grp1,
               p_r4 RADIOBUTTON GROUP grp1 DEFAULT 'X'.
SELECTION-SCREEN end of block bk1.

PARAMETERS p_tname type c length 10.

INITIALIZATION.
   t1 = 'DDL and DML operations using ADBC'.
   perform connect_hdb.

START-OF-SELECTION.
 if o_stmt is bound.
   if p_r1 = 'X'.
     perform createdbtable.
   elseif p_r2 = 'X'.
     perform insertdata.
   elseif p_r3 = 'X'.
     perform droptable.
   endif.
 endif.

FORM connect_hdb .
* Connect to HDB
  TRY.
CALL METHOD cl_sql_connection=>get_connection
  RECEIVING
    con_ref  = o_con.
CATCH cx_sql_exception into o_sql_excp.
* Capture standard exception long message
  CALL METHOD o_sql_excp->if_message~get_longtext
```

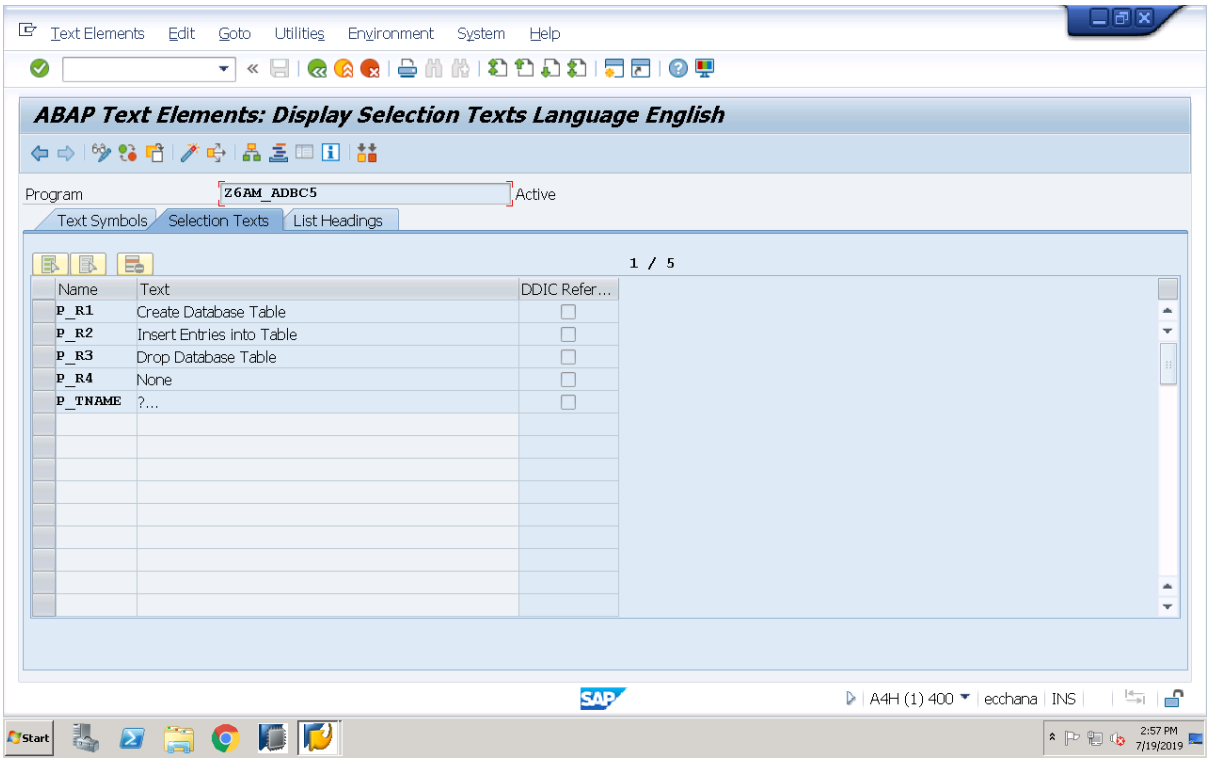
```
    RECEIVING
      result            = v_msg.

      write :/ 'Error in Establishing DB connection :',v_msg.
ENDTRY.

if o_con is bound.
* Instantiate the statement class on top of DB connection
  CREATE OBJECT o_stmt
    EXPORTING
      con_ref = o_con.
endif.
ENDFORM.

FORM createdbtable .
* Create the DB table in the HANA DB
TRY.
  CALL METHOD o_stmt->execute_ddl
    EXPORTING
      statement =  `CREATE TABLE  ` && p_tname &&
                   ` ( empno char(10), ` &&
                   `   ename NVARCHAR(20), ` &&
                   `   PRIMARY KEY (empno) ) `.
   message 'DB Table created Successfully' type 'I'.
CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

  write :/ 'Exception Long msg while creating:',v_msg.

  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_text
    RECEIVING
      result = v_msg.

  write :/ 'Exception short msg while creating:',v_msg.
ENDTRY.
ENDFORM.

FORM insertdata .
```

```
* Read the data from local text file to temp.int table
  types : begin of ty_legacy,
             str(100) type c,
          end of ty_legacy.

   data : t_legacy type table of ty_legacy,
         wa_legacy type ty_legacy.

   CALL FUNCTION 'GUI_UPLOAD'
     EXPORTING
       filename         = 'C:\Users\ABAPHANA3\Desktop\emp.txt'
     tables
       data_tab         = t_legacy.

     if t_legacy is NOT INITIAL.
* Transfer data from temp.int table to final int.table
       types : begin of ty_final,
                   empno(10) type c,
                   ename(20) type c,
               end of ty_final.

        data : t_final type table of ty_final,
              wa_final type ty_final.

        loop at t_legacy into wa_legacy.
          clear wa_final.
          split wa_legacy-str at ',' into wa_final-empno
                                          wa_final-ename.
          append wa_final to t_final.
        endloop.
     endif.

     if t_final is not INITIAL.
* insert each record of final int.table to DB table
       loop at t_final into wa_final.
 TRY.
 CALL METHOD o_stmt->execute_update
   EXPORTING
     statement      = `INSERT INTO ` && P_TNAME && ` ` &&
                      `VALUES('` && wa_final-empno && `','` && wa_fina
l-ename && `')`.
 CATCH cx_sql_exception into o_sql_excp.
```

```
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result            = v_msg.

  write :/ 'Exception Long msg while inserting:',v_msg.

  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_text
    RECEIVING
      result = v_msg.

  write :/ 'Exception short msg while inserting:',v_msg..
 CATCH cx_parameter_invalid into o_param.
  clear v_msg.
  CALL METHOD o_param->if_message~get_longtext
    RECEIVING
      result            = v_msg.

  write :/ 'Exception Long msg related to parameters while inserting:'
,v_msg.

  clear v_msg.
  CALL METHOD o_param->if_message~get_text
    RECEIVING
      result = v_msg.

  write :/ 'Exception short msg related to parameters while inserting:
',v_msg...
 ENDTRY.
       endloop.
     endif.
ENDFORM.

FORM droptable .
* Drop the DB table
  TRY.
  CALL METHOD o_stmt->execute_ddl
    EXPORTING
      statement = `DROP TABLE ` && p_tname.

   message 'Table Dropped successfully' type 'I'.
```

```
 CATCH cx_sql_exception into o_sql_excp.
  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_longtext
    RECEIVING
      result              = v_msg.

  write :/ 'Exception Long msg while dropping table:',v_msg.

  clear v_msg.
  CALL METHOD o_sql_excp->if_message~get_text
    RECEIVING
      result = v_msg.

  write :/ 'Exception short msg while dropping table:',v_msg...
 ENDTRY.
ENDFORM.
```

Program Selection Texts:

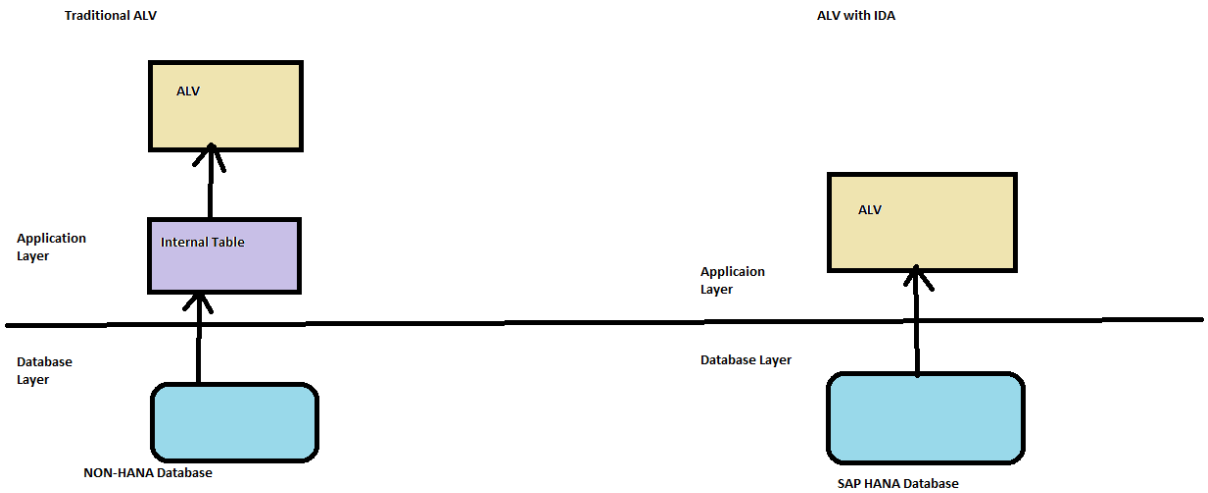


22-07-2019 IDA

1. Get the ALV object by passing DB table name or DDIC view (CREATE method)

2. Using the above ALV object, get the reference of fullscreen mode (FULLSCREEN method)

3. Using the above fullscreen mode object, display the content on the UI (DISPLAY method)



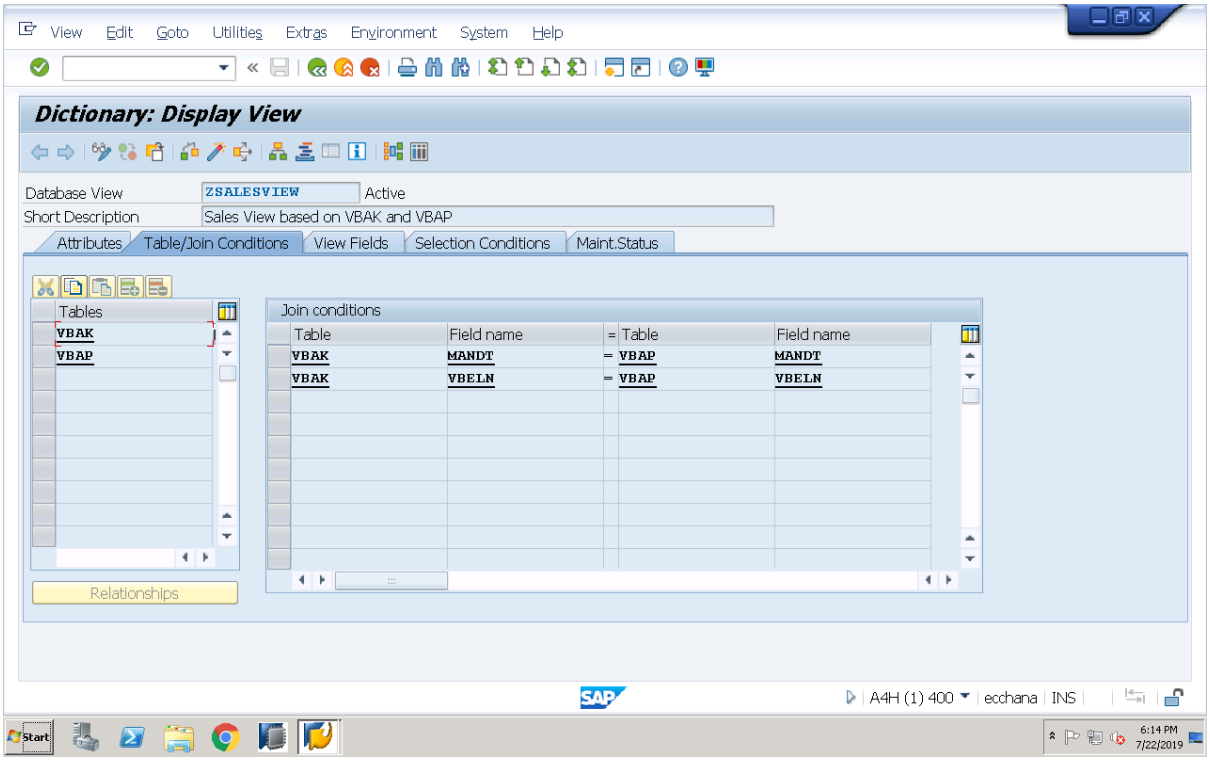
Example: ALV IDA Report to display DB Table entries

```
REPORT Z6AM_ALVIDA1.

* Get the ALV Object
data o_alv type ref to IF_SALV_GUI_TABLE_IDA.
data v_msg type string.

data : o_excp1 type ref to cx_salv_db_connection,
       o_excp2 type ref to cx_salv_db_table_not_supported,
       o_excp3 type ref to cx_salv_ida_contract_violation.

TRY.
CALL METHOD cl_salv_gui_table_ida=>create
  EXPORTING
    iv_table_name          = 'SFLIGHT'
  receiving
    ro_alv_gui_table_ida   = o_alv.
```

```
 CATCH cx_salv_db_connection into o_excp1.
   clear v_msg.
   CALL METHOD o_excp1->if_message~get_longtext
     RECEIVING
       result            = v_msg.

   write :/ 'Error in DB connection :',v_msg.

 CATCH cx_salv_db_table_not_supported into o_excp2.
   clear v_msg.
   CALL METHOD o_excp2->if_message~get_longtext
     RECEIVING
       result            = v_msg.

   write :/ 'Error in DB table not supported :',v_msg.
 CATCH cx_salv_ida_contract_violation into o_excp3.
   clear v_msg.
   CALL METHOD o_excp3->if_message~get_longtext
     RECEIVING
       result            = v_msg.

  write :/ 'Error in IDA Contract Violation :',v_msg.

 ENDTRY.

if o_alv is bound.
* get the reference of fullscreen mode
  data o_mode type ref to IF_SALV_GUI_FULLSCREEN_IDA.
 TRY.
  CALL METHOD o_alv->fullscreen
    RECEIVING
      ro_fullscreen = o_mode.
 CATCH cx_salv_ida_contract_violation  into o_excp3.
   clear v_msg.
   CALL METHOD o_excp3->if_message~get_longtext
     RECEIVING
       result            = v_msg.

  write :/ 'Error in IDA Contract Violation while getting fullscreen m
ode :',v_msg..
 ENDTRY.
endif.
```

```
if o_mode is bound.
* Display the data on the UI
  CALL METHOD o_mode->display.
endif.
```

Example: ALV IDA Report to display DB view entries created on top of 2 DB tables



View Edit Goto Utilities Extras Environment System Help

Dictionary: Display View

Database View: ZSALESVIEW Active

Short Description: Sales View based on VBAK and VBAP

Attributes | Table/Join Conditions | View Fields | Selection Conditions | Maint.Status

Table fields

View field	Table	Field	Key	Data elem.	M...	DTyp	Length	Short description
VBELN	VBAK	VBELN	☑	VBELN_VA	☐	CHAR	10	Sales Document
ERDAT	VBAK	ERDAT	☑	ERDAT	☐	DATS	8	Date on which the recor
ERNAM	VBAK	ERNAM	☑	ERNAM	☐	CHAR	12	Name of Person who Cr
POSNR	VBAP	POSNR	☑	POSNR_VA	☐	NUMC	6	Sales Document Item
MATNR	VBAP	MATNR	☑	MATNR	☐	CHAR	40	Material Number
NETWR	VBAP	NETWR	☑	NETWR_AP	☐	CURR	15	Net Value of the Order I

A4H (1) 400 | ecchana | INS

View Edit Goto Utilities Extras Environment System Help

Dictionary: Display View

Database View: ZSALESVIEW Active

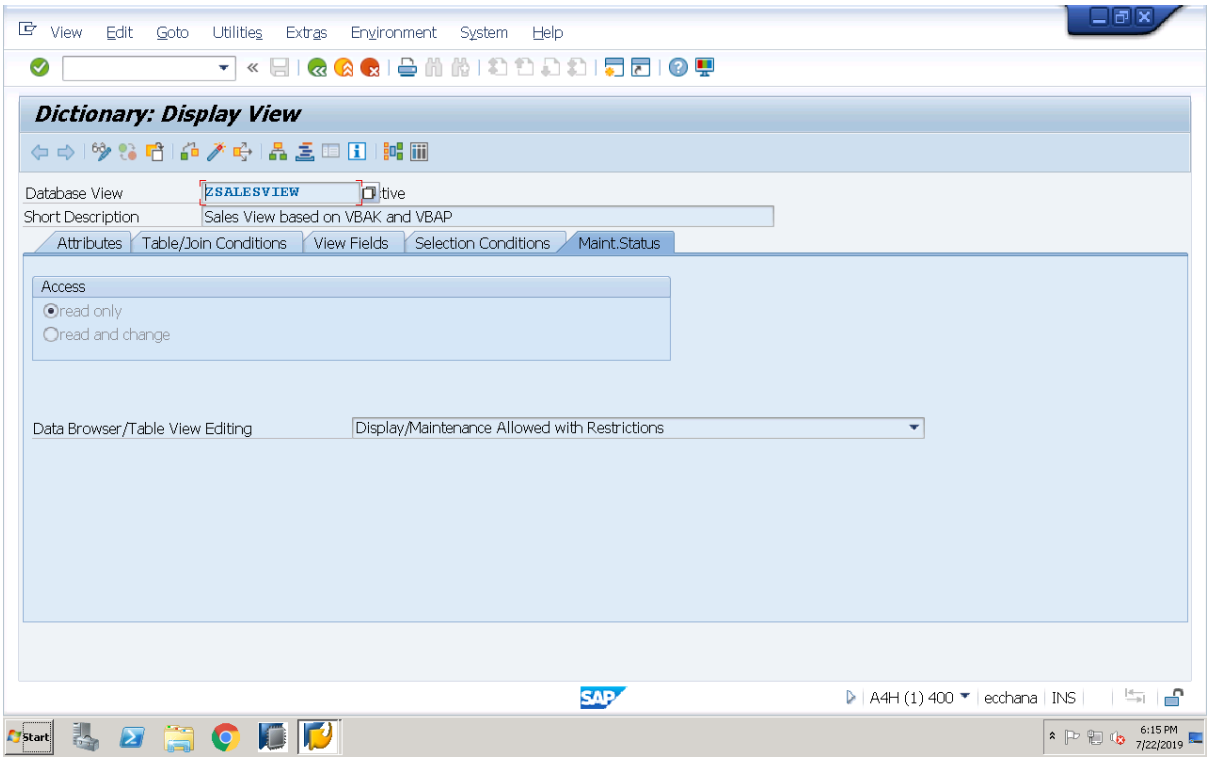
Short Description: Sales View based on VBAK and VBAP

Attributes | Table/Join Conditions | View Fields | Selection Conditions | Maint.Status

Table fields

Table	Field name	Operator	Comparison Value	AND/...
VBAK	VBELN	GE	'0000004980'	AND
VBAK	VBELN	LE	'0000004985'	

A4H (1) 400 | ecchana | INS



Executable Program: To Access above Database view in ALV IDA Report

```
REPORT Z6AM_ALVIDA2.

* Get the ALV Object
data o_alv type ref to IF_SALV_GUI_TABLE_IDA.
data v_msg type string.

data : o_excp1 type ref to cx_salv_db_connection,
       o_excp2 type ref to cx_salv_db_table_not_supported,
       o_excp3 type ref to cx_salv_ida_contract_violation.

TRY.
CALL METHOD cl_salv_gui_table_ida=>create
  EXPORTING
    iv_table_name          = 'ZSALESVIEW'
  receiving
    ro_alv_gui_table_ida   = o_alv.
 CATCH cx_salv_db_connection into o_excp1.
   clear v_msg.
   CALL METHOD o_excp1->if_message~get_longtext
     RECEIVING
       result             = v_msg.
```

```
    write :/ 'Error in DB connection :',v_msg.

  CATCH cx_salv_db_table_not_supported into o_excp2.
    clear v_msg.
    CALL METHOD o_excp2->if_message~get_longtext
      RECEIVING
        result            = v_msg.

    write :/ 'Error in DB table not supported :',v_msg.
  CATCH cx_salv_ida_contract_violation into o_excp3.
    clear v_msg.
    CALL METHOD o_excp3->if_message~get_longtext
      RECEIVING
        result            = v_msg.

   write :/ 'Error in IDA Contract Violation :',v_msg.

  ENDTRY.

 if o_alv is bound.
 * get the reference of fullscreen mode
   data o_mode type ref to IF_SALV_GUI_FULLSCREEN_IDA.
  TRY.
   CALL METHOD o_alv->fullscreen
     RECEIVING
       ro_fullscreen = o_mode.
  CATCH cx_salv_ida_contract_violation  into o_excp3.
    clear v_msg.
    CALL METHOD o_excp3->if_message~get_longtext
      RECEIVING
        result            = v_msg.

   write :/ 'Error in IDA Contract Violation while getting fullscreen m
 ode :',v_msg..
  ENDTRY.
 endif.

 if o_mode is bound.
 * Display the data on the UI
   CALL METHOD o_mode->display.
 endif.
```

23-07-2019

Example: ALV IDA Report for Manipulating Field Catalog and ALV Toolbar

```
REPORT Z6AM_ALVIDA3.

* get the ALV Object referring to DB table
data o_alv type ref to IF_SALV_GUI_TABLE_IDA.

data : o_excp1 type ref to cx_salv_db_connection,
       o_excp2 type ref to cx_salv_db_table_not_supported,
       o_excp3 type ref to cx_salv_ida_contract_violation,
       o_excp4 type ref to CX_SALV_IDA_UNKNOWN_NAME,
       o_excp5 type ref to CX_SALV_CALL_AFTER_1ST_DISPLAy.

data v_msg type string.

TRY.
CALL METHOD cl_salv_gui_table_ida=>create
  EXPORTING
    iv_table_name         = 'KNA1'
  receiving
    ro_alv_gui_table_ida  = o_alv.
CATCH cx_salv_db_connection into o_excp1.
  clear v_msg.
  CALL METHOD o_excp1->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in DB connection :',v_msg.
CATCH cx_salv_db_table_not_supported  into o_excp2.
  clear v_msg.
  CALL METHOD o_excp2->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in DB Table not supported :',v_msg..
CATCH cx_salv_ida_contract_violation  into o_excp3.
  clear v_msg.
  CALL METHOD o_excp3->if_message~get_longtext
    RECEIVING
      result            = v_msg.
```

```
     write :/ 'Error in IDA contract violation :',v_msg..
ENDTRY.

if o_alv is bound.
* get the reference of field catalog
data o_fieldcatalog type ref to IF_SALV_GUI_FIELD_CATALOG_IDA.
CALL METHOD o_alv->field_catalog
  RECEIVING
    ro_field_catalog = o_fieldcatalog.

if o_fieldcatalog is bound.
* set the column heading and tooltip text for NAME1 column
TRY.
CALL METHOD o_fieldcatalog->set_field_header_texts
  EXPORTING
    iv_field_name        = 'NAME1'
    iv_header_text       = 'Customer First Name'
    iv_tooltip_text      = 'Customer Name'
    iv_tooltip_text_long = 'Customer First Name(Name1)'.
 CATCH cx_salv_ida_unknown_name   into o_excp4.
  clear v_msg.
  CALL METHOD o_excp4->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in IDA Unknown name while setting column header t
ext :',v_msg.
 CATCH cx_salv_call_after_1st_display   into o_excp5.
  clear v_msg.
  CALL METHOD o_excp5->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in CALL after 1st display while setting column he
ader texts :',v_msg.
ENDTRY.

* Disable sorting for customer no column
TRY.
CALL METHOD o_fieldcatalog->disable_sort
  EXPORTING
    iv_field_name = 'KUNNR'.
```

```
 CATCH cx_salv_ida_unknown_name   into o_excp4.
  clear v_msg.
  CALL METHOD o_excp4->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in IDA Unknown name while disabling sorting for K
UNNR :',v_msg.
 CATCH cx_salv_call_after_1st_display   into o_excp5.
  clear v_msg.
  CALL METHOD o_excp5->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in CALL after 1st display while disabling sorting
for KUNNR :',v_msg.
ENDTRY.

* Disable Filter for customer no column
TRY.
CALL METHOD o_fieldcatalog->disable_filter
  EXPORTING
    iv_field_name = 'KUNNR'.
 CATCH cx_salv_ida_unknown_name   into o_excp4.
  clear v_msg.
  CALL METHOD o_excp4->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in IDA Unknown name while disabling filtering for
KUNNR :',v_msg.
 CATCH cx_salv_call_after_1st_display   into o_excp5.
  clear v_msg.
  CALL METHOD o_excp5->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in CALL after 1st display while disabling filter
for KUNNR :',v_msg.
ENDTRY.

* Logic to suppress Name2 and SORTL fields
```

```
* get the reference of all fields
CALL METHOD o_fieldcatalog->get_all_fields
  IMPORTING
    ets_field_names = data(gt_field_names).

if gt_field_names is not INITIAL.
* delete NAME2 and SORTL Fields
delete gt_field_names where TABLE_LINE = 'NAME2'.
delete gt_field_names where TABLE_LINE = 'SORTL'.

* Set the avaiable fields back to field catalog
TRY.
CALL METHOD o_fieldcatalog->set_available_fields
  EXPORTING
    its_field_names = gt_field_names.
CATCH cx_salv_ida_unknown_name   into o_excp4.
  clear v_msg.
  CALL METHOD o_excp4->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in IDA Unknown name while setting available field
s in field catalog :',v_msg..
ENDTRY.

endif.
endif.

* Logic for deactivating PRINT BUTTON on alv toolbar
* get the reference of std functions of ALV toolbar
data o_std_fn type ref to IF_SALV_GUI_STD_FUNCTIONS_IDA.
CALL METHOD o_alv->standard_functions
  RECEIVING
    ro_standard_functions = o_std_fn.

if o_std_fn is bound.
* deactivate print button on alv toolbar
  CALL METHOD o_std_fn->set_print_active
    EXPORTING
      iv_active = abap_false.
endif.
```

```
* get the fullscreen interface object
data o_fullscreen type ref to IF_SALV_GUI_FULLSCREEN_IDA.
 TRY.
  CALL METHOD o_alv->fullscreen
    RECEIVING
      ro_fullscreen = o_fullscreen.
 CATCH cx_salv_ida_contract_violation   into o_excp3.
  clear v_msg.
  CALL METHOD o_excp3->if_message~get_longtext
    RECEIVING
      result            = v_msg.

     write :/ 'Error in IDA contract violation while getting fulscreen
object :',v_msg.
 ENDTRY.
endif.

if o_fullscreen is bound.
* Display the content on the UI
  o_fullscreen->display( ).
endif.
```

24-07-2019

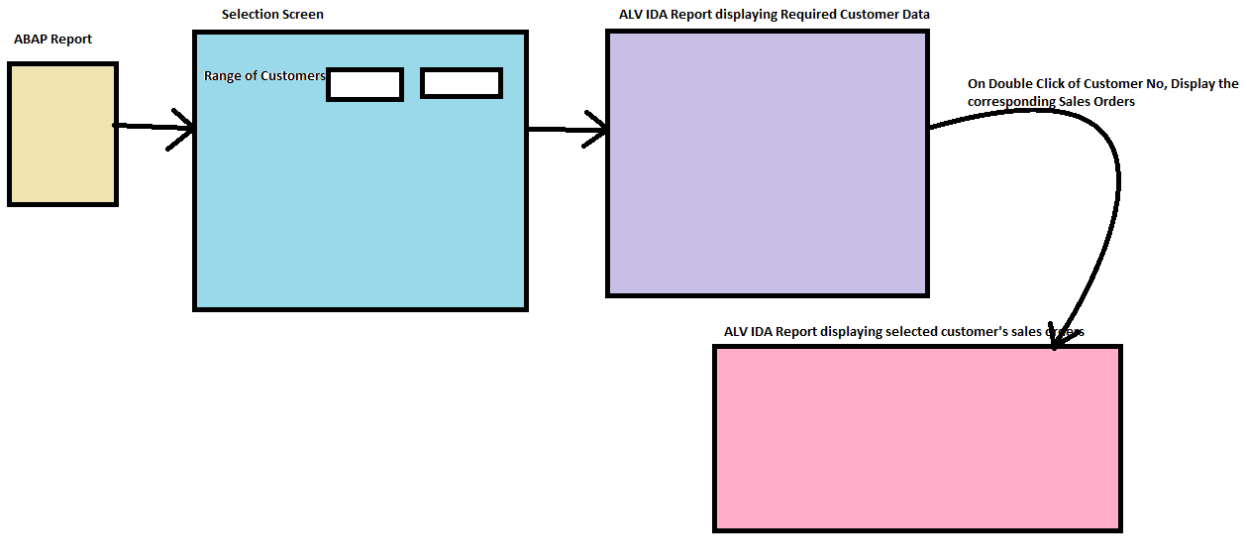
Example: ALV IDA Report implementing Selection Conditions (Parameter and Select Options Functionality for data filtering on Final Data)

```
REPORT Z6AM_ALVIDA4.

parameters p_matnr type matnr.

data v_vbeln type vbap-vbeln.
select-OPTIONS so_vbeln for v_vbeln.

* get the ALV object referring to DB table
data o_alv type ref to IF_SALV_GUI_TABLE_IDA.
TRY.
CALL METHOD cl_salv_gui_table_ida=>create
  EXPORTING
    iv_table_name         = 'VBAP'
  receiving
    ro_alv_gui_table_ida  = o_alv.
 CATCH cx_salv_db_connection .
```

```
   message 'Exception in DB connection' type 'I'.
 CATCH cx_salv_db_table_not_supported .
   message 'DB table Exception' type 'I'.
 CATCH cx_salv_ida_contract_violation .
   message 'IDA Contract violation exception' type 'I'.
ENDTRY.

if o_alv is bound.
* logic for filtering data based on select options
* Instantiate range collector class
      data o_range_collector type ref to CL_SALV_RANGE_TAB_COLLECTOR.
      create object o_range_collector.

      if o_range_collector is bound.
        CALL METHOD o_range_collector->add_ranges_for_name
          EXPORTING
            iv_name   = 'VBELN'
            it_ranges = so_vbeln[].
* Collect the range into int.table
           CALL METHOD o_range_collector->get_collected_ranges
             IMPORTING
               et_named_ranges = data(lt_select_options).
      endif.
* end of logic for filtering based on select options

* logic for filtering based on single field
* get the reference of condition factory interface
data o_cond_factory type ref to IF_SALV_IDA_CONDITION_FACTORY.
CALL METHOD o_alv->condition_factory
  RECEIVING
    ro_condition_factory = o_cond_factory.

 if o_cond_factory is bound.
* create the condition based on parameter field
   data o_cond type ref to IF_SALV_IDA_CONDITION.
  TRY.
   CALL METHOD o_cond_factory->equals
     EXPORTING
       name         = 'MATNR'
       VALUE        = p_matnr
     RECEIVING
       ro_condition = o_cond.
```

```
   CATCH cx_salv_ida_unknown_name .
     message 'Exception in IDA unknown name while constructing conditi
on based on parameter field' type 'I'.
   ENDTRY.
 endif.
* END OF logic for filtering based on parameter field
if lt_select_options is not INITIAL and o_cond is bound.
* set the select options
TRY.
CALL METHOD o_alv->set_select_options
  EXPORTING
    it_ranges    = lt_select_options
    IO_CONDITION = o_cond.
 CATCH cx_salv_ida_associate_invalid .
   message 'Exception in IDA associate invalid while setting select op
tion' type 'I'.
 CATCH cx_salv_db_connection .
   message 'Exception in DB connection while setting select option' ty
pe 'I'.
 CATCH cx_salv_ida_condition_invalid .
   message 'Exception in IDA condition invalid while setting select op
tion' type 'I'.
 CATCH cx_salv_ida_unknown_name .
   message 'Exception in IDA unknown name while setting select option'
type 'I'.
 ENDTRY.
endif.
* get the full screen mode object
  data o_fullscreen type ref to IF_SALV_GUI_FULLSCREEN_IDA.
  TRY.
  CALL METHOD o_alv->fullscreen
    RECEIVING
      ro_fullscreen = o_fullscreen.
  CATCH cx_salv_ida_contract_violation .
    message 'IDA Contract violation exception while getting fullscreen
mode object' type 'I'.
  ENDTRY.
  if o_fullscreen is bound.
* Display the data on the UI
    o_fullscreen->display( ).
  endif.
endif.
```

25-07-2019



Example: Interactive Reporting using ALV IDA (Double_click event), Displaying Sales Orders of the selected customer

```
REPORT Z6AM_ALVIDA5.

data v_kunnr type kna1-kunnr.
select-OPTIONS so_kunnr for v_kunnr.

data : o_alv type ref to IF_SALV_GUI_TABLE_IDA,
       o_range_collector type ref to CL_SALV_RANGE_TAB_COLLECTOR,
       o_fullscreen type ref to IF_SALV_GUI_FULLSCREEN_IDA,
       o_alv_disp type ref to IF_SALV_GUI_TABLE_DISPLAY_OPT,
       o_cond_factory type ref to IF_SALV_IDA_CONDITION_FACTORY,
       o_cond type ref to IF_SALV_IDA_CONDITION,
       wa_kna1 type kna1.

class lcl_eventreceiver DEFINITION.
  PUBLIC SECTION.
    methods handle_double_click
              for event double_click
                of IF_SALV_GUI_TABLE_DISPLAY_OPT
                   IMPORTING ev_field_name eo_row_data.
endclass.

class lcl_eventreceiver IMPLEMENTATION.
  method handle_double_click.
    case ev_field_name.
```

```
      when 'KUNNR'.
*        message 'Double Clicked on Customer No' type 'I'.
* Extract Customer No from Interacted Row
        clear wa_kna1.
       TRY.
          CALL METHOD EO_ROW_DATA->GET_ROW_DATA
            IMPORTING
              ES_ROW  = wa_kna1.
       CATCH CX_SALV_IDA_CONTRACT_VIOLATION .
         message 'Exception IDA Contract Violation while fetching inte
racted row info' type 'I'.
       CATCH CX_SALV_IDA_SEL_ROW_DELETED .
         message 'Exception IDA Selected row deleted while fetching in
teracted row info' type 'I'.
       ENDTRY.

       if wa_kna1 is not INITIAL.
* get the ALV object referring to DB table 'VBAK'
free o_alv.
TRY.
CALL METHOD cl_salv_gui_table_ida=>create
  EXPORTING
    iv_table_name         = 'VBAK'
  receiving
    ro_alv_gui_table_ida  = o_alv.
 CATCH cx_salv_db_connection .
   message 'Exception in DB connection' type 'I'.
 CATCH cx_salv_db_table_not_supported .
   message 'DB table Exception' type 'I'.
 CATCH cx_salv_ida_contract_violation .
   message 'IDA Contract violation exception' type 'I'.
ENDTRY.

if o_alv is bound.
* Logic for filtering data based on selected customer no
* get the reference of condition factory interface
CALL METHOD o_alv->condition_factory
  RECEIVING
    ro_condition_factory = o_cond_factory.

 if o_cond_factory is bound.
* create the condition based on selected customer no
```

```
  TRY.
   CALL METHOD o_cond_factory->equals
     EXPORTING
       name         = 'KUNNR'
       VALUE        = wa_kna1-kunnr
     RECEIVING
       ro_condition = o_cond.
   CATCH cx_salv_ida_unknown_name .
     message 'Exception in IDA unknown name while constructing conditi
on based on selected customer no ' type 'I'.
   ENDTRY.
 endif.
* End of Logic for filtering data based on selected customer no
endif.

    if o_cond is bound.
* set the condition based on interacted customer no
TRY.
CALL METHOD o_alv->set_select_options
  EXPORTING
        IO_CONDITION = o_cond.
 CATCH cx_salv_ida_associate_invalid .
   message 'Exception in IDA associate invalid while setting condition
' type 'I'.
 CATCH cx_salv_db_connection .
   message 'Exception in DB connection while setting condition' type '
I'.
 CATCH cx_salv_ida_condition_invalid .
   message 'Exception in IDA condition invalid while setting condition
' type 'I'.
 CATCH cx_salv_ida_unknown_name .
   message 'Exception in IDA unknown name while setting condition' typ
e 'I'.
 ENDTRY.
    endif.

* get the full screen mode object
    free o_fullscreen.
  TRY.
  CALL METHOD o_alv->fullscreen
    RECEIVING
      ro_fullscreen = o_fullscreen.
```

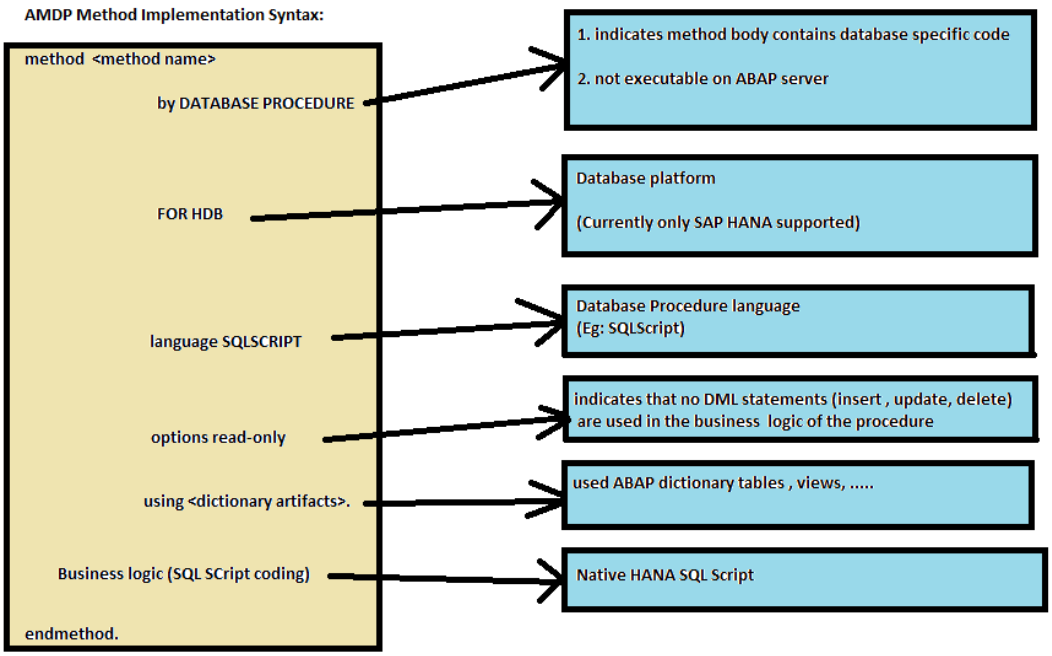
```
  CATCH cx_salv_ida_contract_violation .
    message 'IDA Contract violation exception while getting fullscreen
mode object' type 'I'.
  ENDTRY.

  if o_fullscreen is bound.
* Display the sales orders on UI
    o_fullscreen->display( ).
  endif.
       endif.
      when others.
        message 'Please Double click on Customer No only' type 'I'.
    endcase.
  endmethod.
endclass.

data ob type ref to lcl_eventreceiver.

START-OF-SELECTION.
* get the ALV object referring to DB table 'KNA1'
TRY.
CALL METHOD cl_salv_gui_table_ida=>create
  EXPORTING
    iv_table_name         = 'KNA1'
  receiving
    ro_alv_gui_table_ida  = o_alv.
 CATCH cx_salv_db_connection .
   message 'Exception in DB connection' type 'I'.
 CATCH cx_salv_db_table_not_supported .
   message 'DB table Exception' type 'I'.
 CATCH cx_salv_ida_contract_violation .
   message 'IDA Contract violation exception' type 'I'.
ENDTRY.

if o_alv is bound.
* logic for filtering data based on select options

* Instantiate range collector class
      create object o_range_collector.

      if o_range_collector is bound.
        CALL METHOD o_range_collector->add_ranges_for_name
```

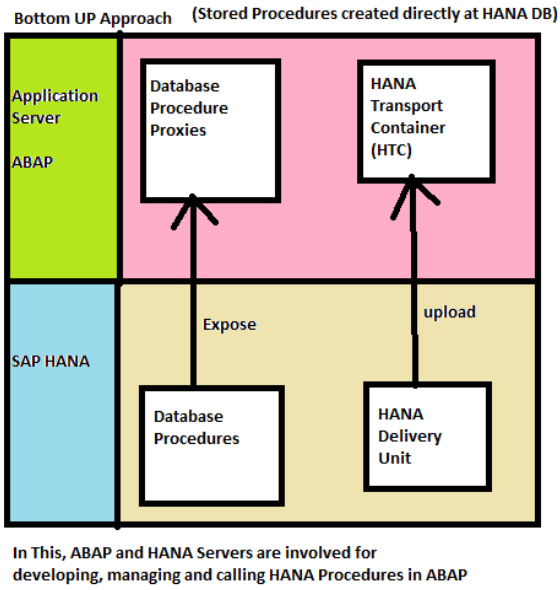
```
          EXPORTING
            iv_name   = 'KUNNR'
            it_ranges = so_kunnr[].
* Collect the range into int.table
           CALL METHOD o_range_collector->get_collected_ranges
             IMPORTING
               et_named_ranges = data(lt_select_options).
      endif.
* end of logic for filtering based on select options

if lt_select_options is not INITIAL.
* set the select options
TRY.
CALL METHOD o_alv->set_select_options
  EXPORTING
    it_ranges    = lt_select_options.
 CATCH cx_salv_ida_associate_invalid .
   message 'Exception in IDA associate invalid while setting select op
tion' type 'I'.
 CATCH cx_salv_db_connection .
   message 'Exception in DB connection while setting select option' ty
pe 'I'.
 CATCH cx_salv_ida_condition_invalid .
   message 'Exception in IDA condition invalid while setting select op
tion' type 'I'.
 CATCH cx_salv_ida_unknown_name .
   message 'Exception in IDA unknown name while setting select option'
type 'I'.
 ENDTRY.
endif.

* get the full screen mode object
  TRY.
  CALL METHOD o_alv->fullscreen
    RECEIVING
      ro_fullscreen = o_fullscreen.
  CATCH cx_salv_ida_contract_violation .
    message 'IDA Contract violation exception while getting fullscreen
mode object' type 'I'.
  ENDTRY.

  if o_fullscreen is bound.
```

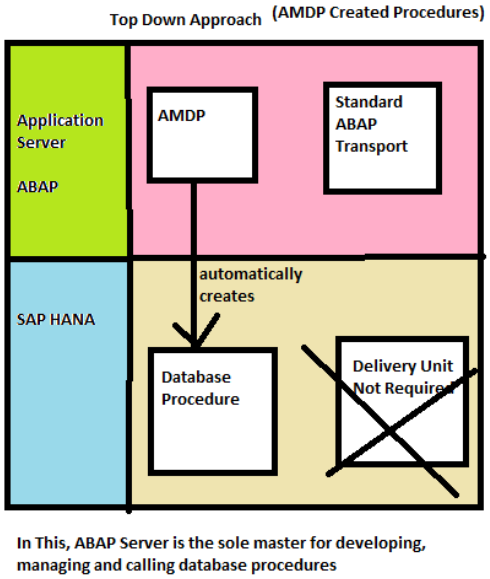
```
* Get the reference of Display options interface
      CALL METHOD O_ALV->DISPLAY_OPTIONS
        RECEIVING
          RO_DISPLAY_OPTIONS = o_alv_disp.

       if o_alv_disp is bound.
* Enable the double click using display options interface
         o_alv_disp->ENABLE_DOUBLE_CLICK( ).
* Register the handler for executing event handler method for double_c
lick event
         create object ob.
         set handler ob->handle_double_click for all INSTANCES.
       endif.

* Display the data on the UI
    o_fullscreen->display( ).
  endif.
endif.
```

AMDP

26-07-2019.







AMDP (ABAP Managed Database Procedure):

AMDP is one of the recommended patterns for use in ABAP Code Optimization within the context of ABAP Development on SAP HANA.

AMDP are implemented as methods of a global ABAP class, the editing environment for AMDP is the ABAP Class Editor. In concrete terms, an AMDP is written in a database specific language such as Native SQL or SQL Script and is implemented within an AMDP method body of an ABAP class.

HANA Stored Procedure: It is developed in HANA Studio, Database Procedure Proxy needs to be created in the ABAP Development tools (HANA Studio). HANA Transport Container and HANA Delivery unit is required for integrating the HANA content in the standard ABAP transport mechanism. In this, ABAP and HANA Servers are involved for developing, managing and calling HANA Procedures in ABAP.

AMDP Created Procedure: In this, ABAP Server is the sole master for developing, managing and calling database procedures. AMDP is implemented as methods of global class marked with tag interfaces (AKA AMDP class). In this, HANA Procedure will be created at the first call of AMDP method. Transport is required for AMDP class. Only ABAP development tools are required. AMDP are the replace technology for database procedure proxies. AMDP Methods can make use of ABAP Dictionary views and tables. AMDP methods are called like regular ABAP methods. Detailed analysis of runtime errors in ST22. AMDP are the replacement technology for database procedure proxies. Database

procedure proxies are still recommended when using a secondary database connection to access SQL Script procedures that exist in a different SAP HANA Database.

AMDP Definition:

1. Definition & Maintenance is done via ADT (ABAP development Tools) in HANA Studio
2. Standard ABAP class method acts as container for DB procedure business logic

AMDP Consumption:

1. Consumption is like any other ABAP class method

Fully integrated into the ABAP infrastructure:

1. Syntax check provided for SQL script coding
2. Detailed analysis of AMDP runtime errors (ST22)
3. Only Transport of ABAP objects required

Procedure to work with AMDP:

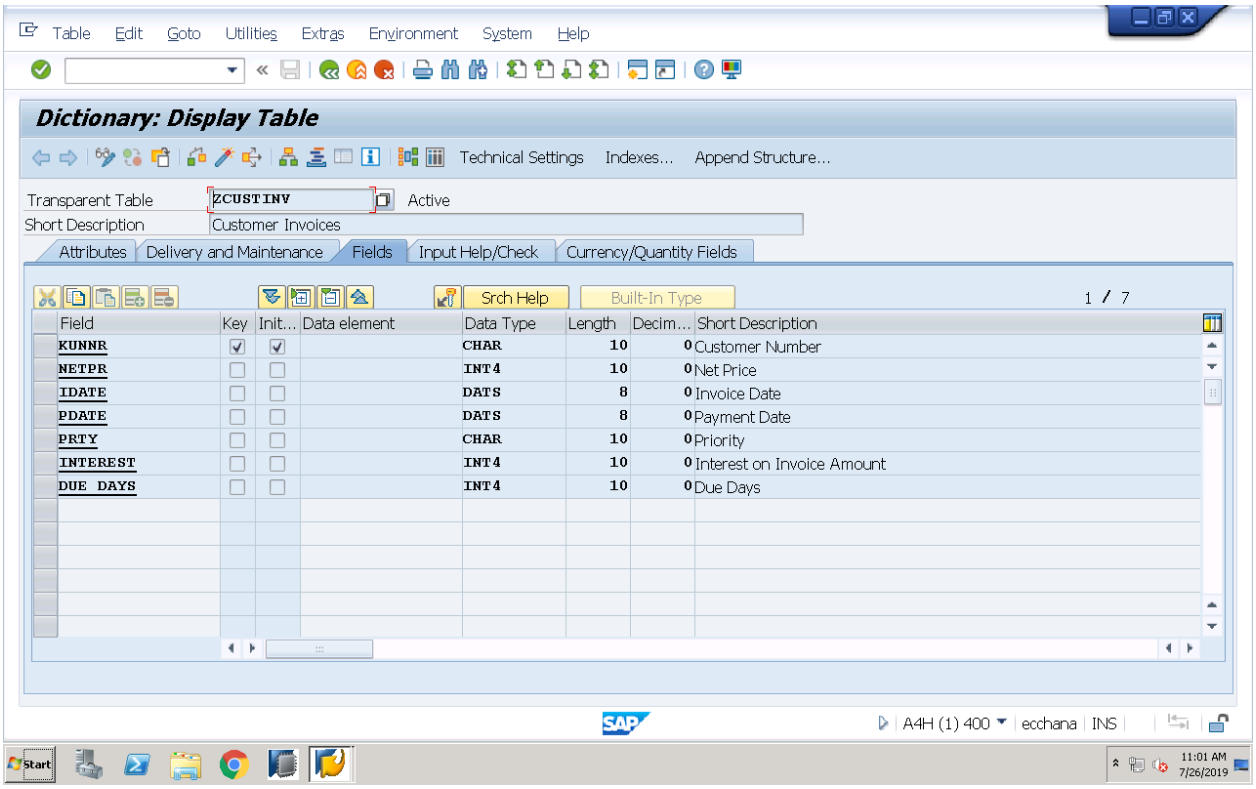
1. Create an AMDP class
2. Create an AMDP Method
3. Implement the AMDP method
4. Create an ABAP report consuming the AMDP method

Pre-requisites for creating AMDP class definition:

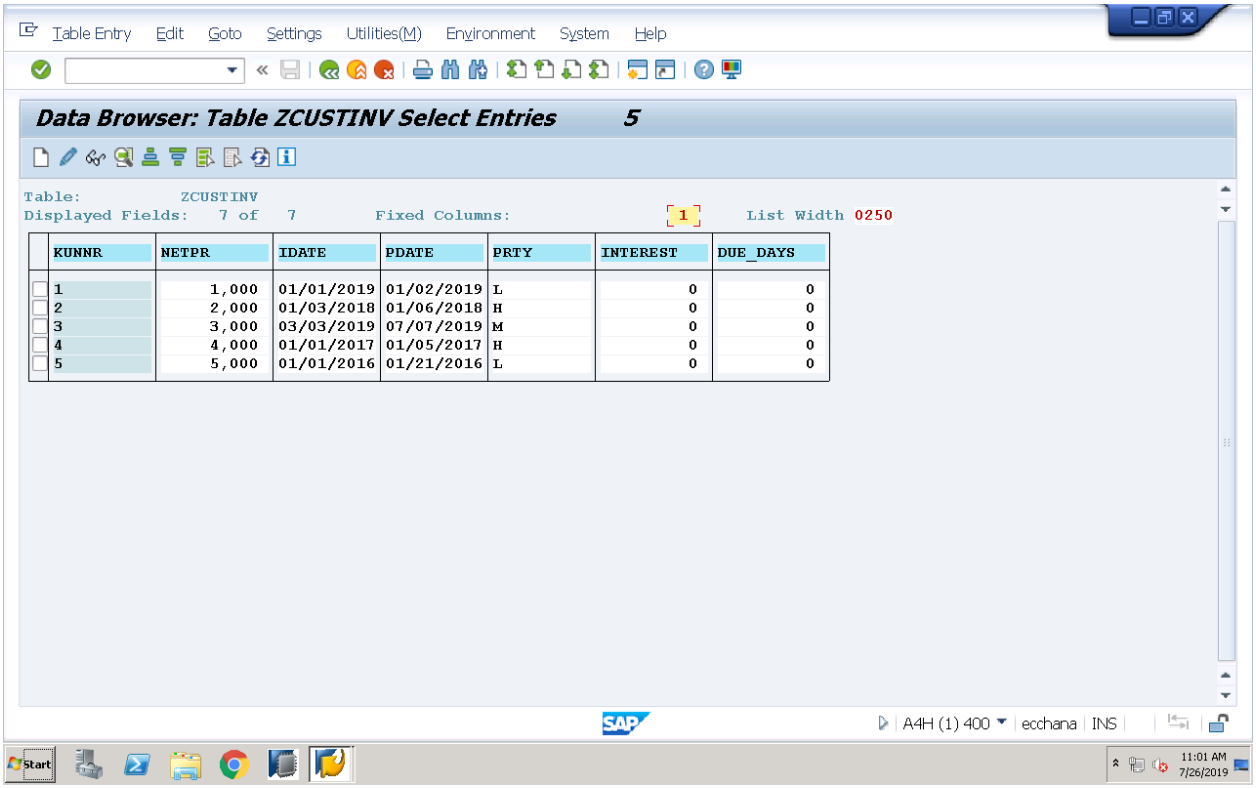
1. AMDP class must use interface 'IF_AMDP_MARKER_HDB'
2. All AMDP method parameters (input/output) must be passed by value (similar to RFC)
3. AMDP returning parameter/s must always be internal table

AMDP Example:

Create the following table in SE11



Maintain some data in the above table



Requirement: Create an AMDP to read data from above database table and construct the values for the fields PRTY, INTEREST and DUE_DAYS and display the same in the report

1. In HANA Studio, Switch to ABAP perspective
2. Create a package (ZMDPPACK) (or) Consider an existing package Right Click on Instance → New → ABAP Package -→ Provide Package name (ZAMDPPACK), Description(….) → Finish
3. Right click on above package → New → ABAP class, provide Name→ZCSTINVOICES, Description → AMDP for Customer Invoices, Next, Finish..

```
class ZCSTINVOICES definition
  public
  final
  create public .

public section.
   TYPES TY_TT_CINV TYPE TABLE OF ZCUSTINV.
   INTERFACES IF_AMDP_MARKER_HDB.
   class-methods GETCUSTOMERINVOICES1 exporting value(ET_CUS_INV) TYPE
TY_TT_CINV.
protected section.
private section.
ENDCLASS.

CLASS ZCSTINVOICES IMPLEMENTATION.
   METHOD GETCUSTOMERINVOICES1 by database procedure for hdb language
sqlscript using zcustinv.
          ET_CUS_INV = SELECT KUNNR, NETPR, IDATE, PDATE,
                        CASE PRTY
                            WHEN 'L' THEN 'LOW'
                            WHEN 'H' THEN 'HIGH'
                            WHEN 'M' THEN 'MEDIUM'
                        END AS PRTY,
                        CASE
                    WHEN DAYS_BETWEEN(  IDATE, PDATE ) BETWEEN 1 AND 30
                                THEN NETPR * 5 / 100
                    WHEN DAYS_BETWEEN( IDATE, PDATE ) BETWEEN 31 AND 60
                                THEN NETPR * 8 / 100
                            ELSE
                                  NETPR * 10 / 100
                        END AS INTEREST,
                        DAYS_BETWEEN( IDATE, PDATE ) AS DUE_DAYS
                        FROM ZCUSTINV;

   endmethod.
```

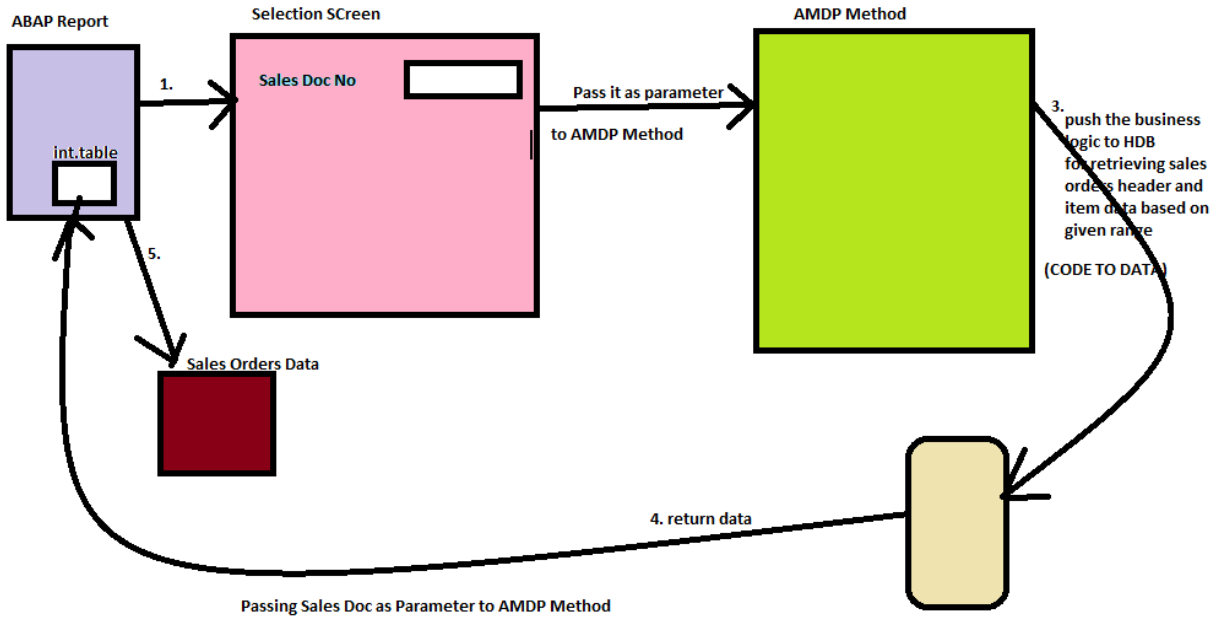
```
ENDCLASS.
```

Consuming AMDP from ABAP Report

```
REPORT ztestprg.

data t_inv type table of zcustinv.

CALL METHOD ZCSTINVOICES=>GETCUSTOMERINVOICES1
  IMPORTING
    ET_CUS_INV = t_inv.

      cl_demo_output=>display( t_inv ).
```

Save, check and activate
Execute the above report, Successfully calls AMDP method with the required result and also creates a stored Procedure with the name of 'AMDP' Method in 'SAPABAP1' Schema (Configured schema for Application server)

29-07-2019



Example: Passing Parameter to AMDP Method

Requirement: AMDP method to get Sales Order Header Data and corresponding Item data based on the sales doc number passed as parameter

HANA Studio:

In ABAP Perspective, Create AMDP class with an AMDP method

```
CLASS ZCL_GET_SALESDATA DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
      INTERFACES IF_AMDP_MARKER_HDB.

      TYPES : BEGIN OF TY_SALES,
                VBELN TYPE VBELN_VA,
                VKORG TYPE VKORG,
                POSNR TYPE POSNR_VA,
                ITEMPRICE TYPE NETWR_AP,
                STATUS TYPE CHAR30,
              END OF TY_SALES.

      TYPES GT_SALES TYPE STANDARD TABLE OF TY_SALES.

      CLASS-METHODS GETSALESDATA IMPORTING VALUE(I_VBELN) TYPE VBELN
                      EXPORTING VALUE(ET_SALES) TYPE GT_SALES.

  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS ZCL_GET_SALESDATA IMPLEMENTATION.

   METHOD GETSALESDATA BY DATABASE PROCEDURE FOR HDB LANGUAGE
SQLSCRIPT OPTIONS READ-ONLY USING VBAK VBAP.


   ET_SALES = SELECT X.VBELN,
                     X.VKORG,
                     Y.POSNR,
                     Y.NETWR AS ITEMPRICE,
                     CASE Y.LFSTA
                        WHEN ' ' THEN 'Not Relevant'
                        WHEN 'A' THEN 'Not Yer Processed'
                        WHEN 'B' THEN 'Partially Processed'
                        WHEN 'C' THEN 'Completely Processed'
                     END AS STATUS
                     FROM VBAK AS X
```

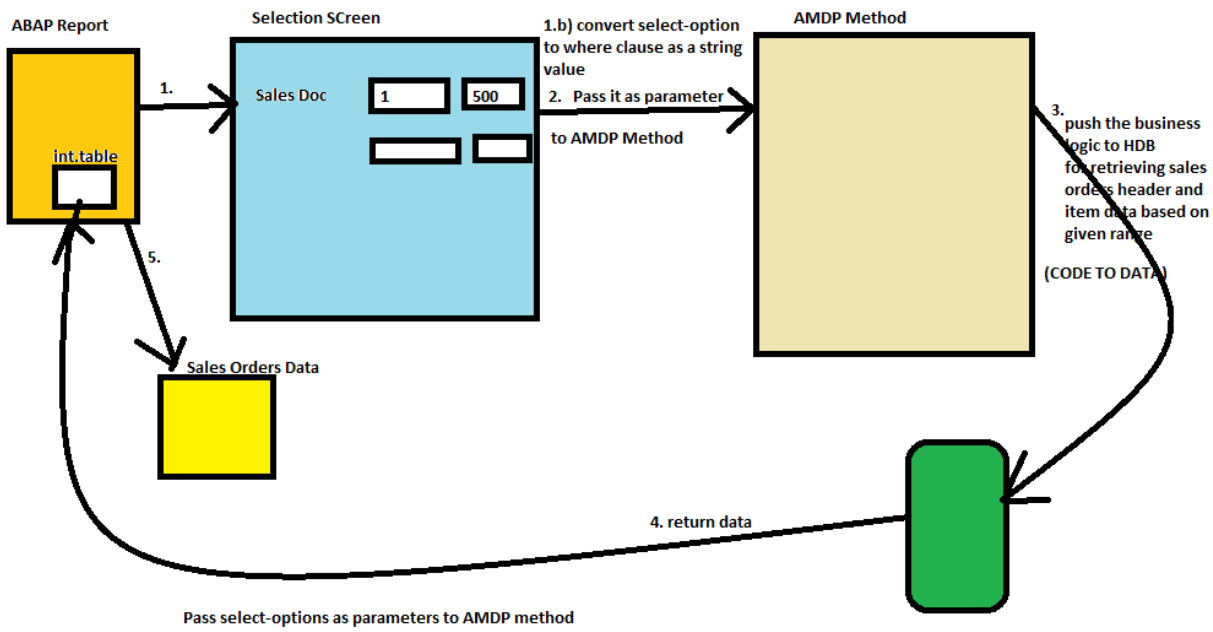
```
                        INNER JOIN VBAP AS Y
                          ON X.VBELN = Y.VBELN
                            WHERE X.VBELN = I_VBELN;

   ENDMETHOD.
ENDCLASS.
```

Save, Check and Activate

Consuming above AMDP from ABAP Report Program:

```
REPORT ZCONSUME_PARAM_AMDP.

PARAMETERS p_vbeln type vbeln.

* Check whether call to AMDP Method feature is supported or not
data v_bool type abap_bool.
TRY.
CALL METHOD CL_ABAP_DBFEATURES=>USE_FEATURES
  EXPORTING
    REQUESTED_FEATURES = value #( ( cl_abap_dbfeatures=>CALL_AMDP_METH
OD ) )
  RECEIVING
    SUPPORTS_FEATURES  = v_bool.
CATCH CX_ABAP_INVALID_PARAM_VALUE .
  message 'Exception in use_features method' type 'I'.
ENDTRY.

if v_bool = 'X'. “call to AMDP method supported
* call AMDP method
  CALL METHOD ZCL_GET_SALESDATA=>GETSALESDATA
    EXPORTING
      I_VBELN  = p_vbeln
    IMPORTING
      ET_SALES = data(gt_order).

  if gt_order is not INITIAL.
*    loop at gt_order into data(wa_order).
*      write :/ wa_order-vbeln,
*               wa_order-vkorg,
*               wa_order-posnr,
*               wa_order-itemprice,
*               wa_order-status.
*    endloop.  "(OR)
```

```
    CALL METHOD CL_DEMO_OUTPUT=>DISPLAY_DATA
      EXPORTING
        VALUE   = gt_order
        NAME    = 'Sales Data'.
  endif.

endif.

30-07-2019.
```



Example: Passing Select-options as Parameter to AMDP Method

Requirement: AMDP method to get Sales Order Header Data and corresponding Item data based on the sales doc number range passed as parameter

HANA Studio:

In ABAP Perspective, Create AMDP class with an AMDP method

```
CLASS ZCL_AMDP_SELECTOPTIONS DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
     INTERFACES IF_AMDP_MARKER_HDB.
```

```
     types : begin of ty_sales,
                     vbeln type vbak-vbeln,
                     erdat type vbak-erdat,
                     ernam type vbak-ernam,
                     netwr type vbak-netwr,
                     posnr type vbap-posnr,
                     matnr type vbap-matnr,
             end of ty_sales.

      types t_temp_sales type table of ty_sales.

 class-methods get_salesorders importing value(i_where) type string
                   exporting value(t_sales) type t_temp_sales.


  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS ZCL_AMDP_SELECTOPTIONS IMPLEMENTATION.

    method get_salesorders by DATABASE PROCEDURE FOR HDB LANGUAGE
SQLSCRIPT options READ-ONLY using vbak vbap.

       t_temp_sales = select x.vbeln,
                             x.erdat,
                             x.ernam,
                             x.netwr,
                             y.posnr,
                             y.matnr
                             from vbak as x
                                 INNER JOIN vbap as y
                                  on x.vbeln = y.vbeln
                                     ORDER BY x.vbeln;

       t_sales = APPLY_FILTER ( :t_temp_sales, :i_where );

    ENDMETHOD.

ENDCLASS.
```

Save, Check and Activate

Consuming above AMDP from ABAP Report Program:

```
REPORT ZCONSUME_SELECTOPTIONS_AMDP.

data v_vbeln type vbak-vbeln.
select-OPTIONS so_vbeln for v_vbeln.

data v_netwr type vbak-netwr.
SELECT-OPTIONS so_netwr for v_netwr.

data v_where type string.

data : t_seltabs type table of if_shdb_def=>ts_named_dref,
      wa_seltabs type if_shdb_def=>ts_named_dref.

clear wa_seltabs.
wa_seltabs-name = 'VBELN'.
wa_seltabs-dref = REF #( so_vbeln[] ).
*wa_seltabs-dref = so_vbeln[]. "syntax error
append wa_seltabs to t_seltabs.

clear wa_seltabs.
wa_seltabs-name = 'NETWR'.
wa_seltabs-dref = REF #( so_netwr[] ).
append wa_seltabs to t_seltabs.

* Convert select options to where clause of type string
TRY.
CALL METHOD CL_SHDB_SELTAB=>COMBINE_SELTABS
  EXPORTING
    IT_NAMED_SELTABS = t_seltabs
  RECEIVING
    RV_WHERE         = v_where.
CATCH CX_SHDB_EXCEPTION.
  message 'Exception in converting select option to where clause of ty
pe string...' type 'I'.
ENDTRY.

if v_where is not INITIAL.
* call AMDP method
  CALL METHOD ZCL_AMDP_SELECTOPTIONS=>GET_SALESORDERS
    EXPORTING
      I_WHERE = v_where
    IMPORTING
```

```
      T_SALES = data(gt_sales).

if gt_sales is not INITIAL.
  WRITE: / '#AMDP Rows ', lines( gt_sales ).
  loop at gt_sales into data(wa_sales).
    write :/ wa_sales-vbeln,
             wa_sales-erdat,
             wa_sales-ernam,
             wa_sales-netwr,
             wa_sales-posnr,
             wa_sales-matnr.
  endloop.
else.
  message 'No Sales Data' type 'I'.
endif.
endif.
```

01-08-2019

CDS (CORE DATA Services)

CDS is used to push down the code to the HANA DB for delegating data intensive calculations to the database layer.

Advantages of CDS:

1. Calculations will be performed on HANA DB itself which increases performance because of HANA's in-memory database, Columnar Storage, Data Compression and massive parallel processing
2. Less data is transferred between Database layer and Application layer so there will be no load on network layer
3. Application layer is only responsible for displaying the data so there will be no load on application layer
4. It is database independent i.e the database view generated on activating CDS view is compatible with all the databases
5. Semantically rich data models (close to conceptual thinking)
6. Completely based on SQL (supports Joins, Expressions, Unions and all built in functions)
7. Easily Extensible

Components of CDS:

1. **DDL Editor:** Text based editor for editing DDL Sources. It is part of ABAP development tools

2. **DDL Sources:** ABAP Development Object that is used to define a CDS view entities. A DDL source is created in ABAP Repository using a wizard of ABAP Development Tools
3. **SQL View:** Projection onto one or multiple relational database tables or other views. An SQL view is generated in the ABAP dictionary after activation of the DDL source. The structure of an SQL view is defined in a CDS entity. SQL view is a technical representation of the entity so that it can be accessed in ABAP

Definition and Consumption of an ABAP CDS View:

1. Definition in an ABAP DDL Source
2. Definition only possible with ADT in Eclipse (Not via SE11)
3. Consumption via a) Open SQL b) Data Preview c) SAP List Viewer

Example: Simple CDS view to read data from a database table 'SNWD_SO'

1. Open HANA Studio, Switch to ABAP Perspective
2. Right Click on any Package / local object → New → Other Repository Object →CORE Data Sources , Choose Data definition, Next → Provide Name: **ZCDSSOVIEW**, provide Description → CDS View to Fetch Sales Orders in EPM, Next, Finish → select the template 'define view', next ➔ Provide the following source code

```
@AbapCatalog.sqlViewName: 'ZCDSSOVIEWSQL'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View to Fetch Sales Orders from EPM'
define view ZCDSSOVIEW as select from snwd_so {
    node_key as node,
    so_id as salesorder,
    buyer_guid as buyerid,
    case billing_status
      when 'P' then 'Paid'
      when ' ' then 'Not Paid'
      else '?'
   end as payment_status
}
```

Note: As part of annotation sqlViewName, provide the DB view name (ZCDSSOVIEW_SQL → in the first line)

Save, Check and activate

Note: On Successful activation, SAP generates a Database view 'ZCDSSOVIEWSQL' in the database. Login to Database (Development Perspective), In the Catalog Folder, Search for the view in the schema 'SAPABAP1'. We can also check this database view in SE11.

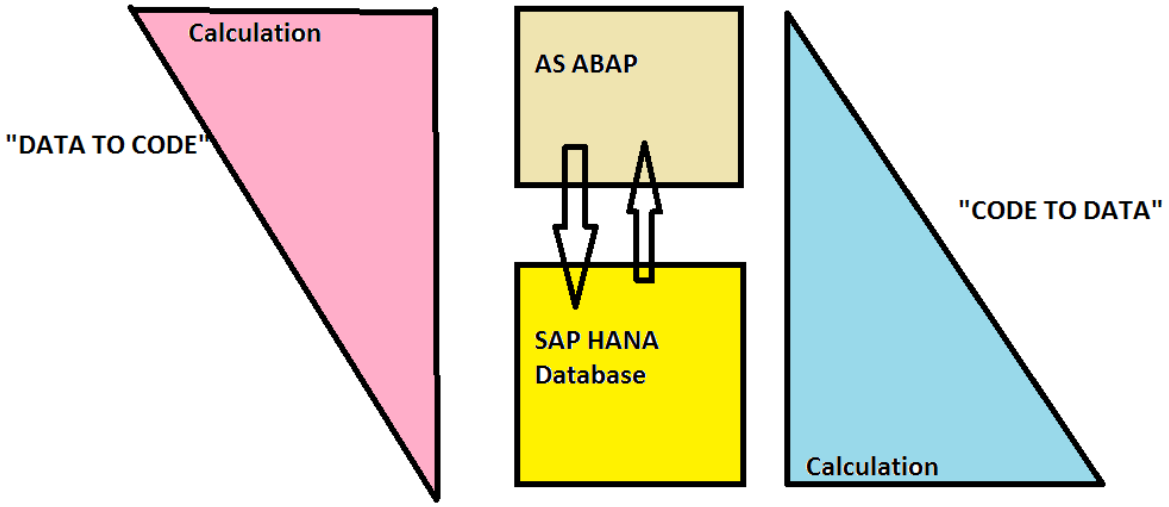
To Preview the data in the CDS view, Right click on CDS view (ZCDSSOVIEW) in the left side hierarchy →open with →data preview

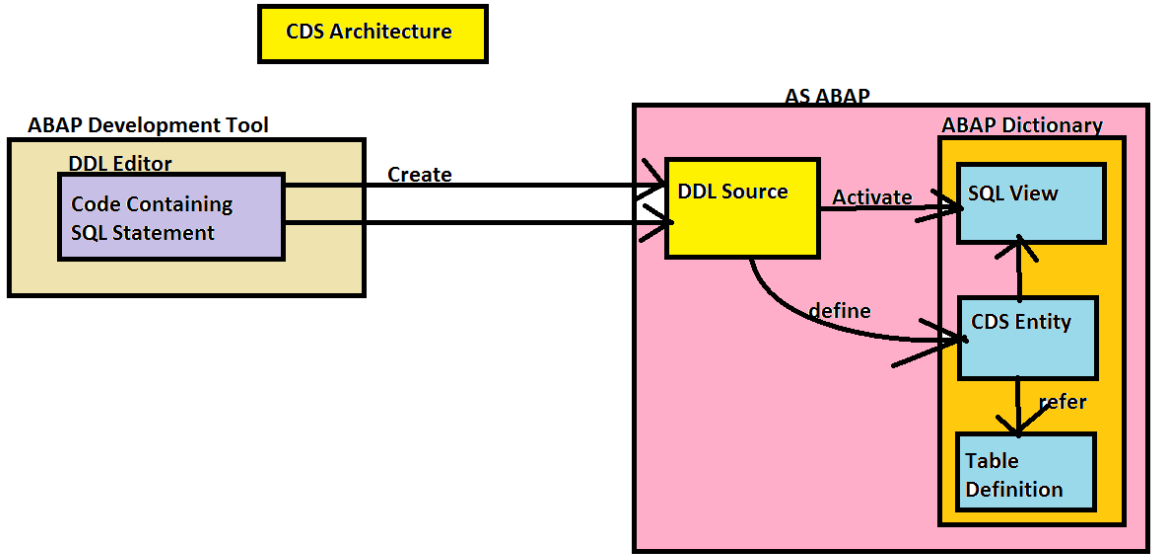
Consuming CDS from ABAP Reports

Create an ABAP program from HANA Studio or from SAP GUI:

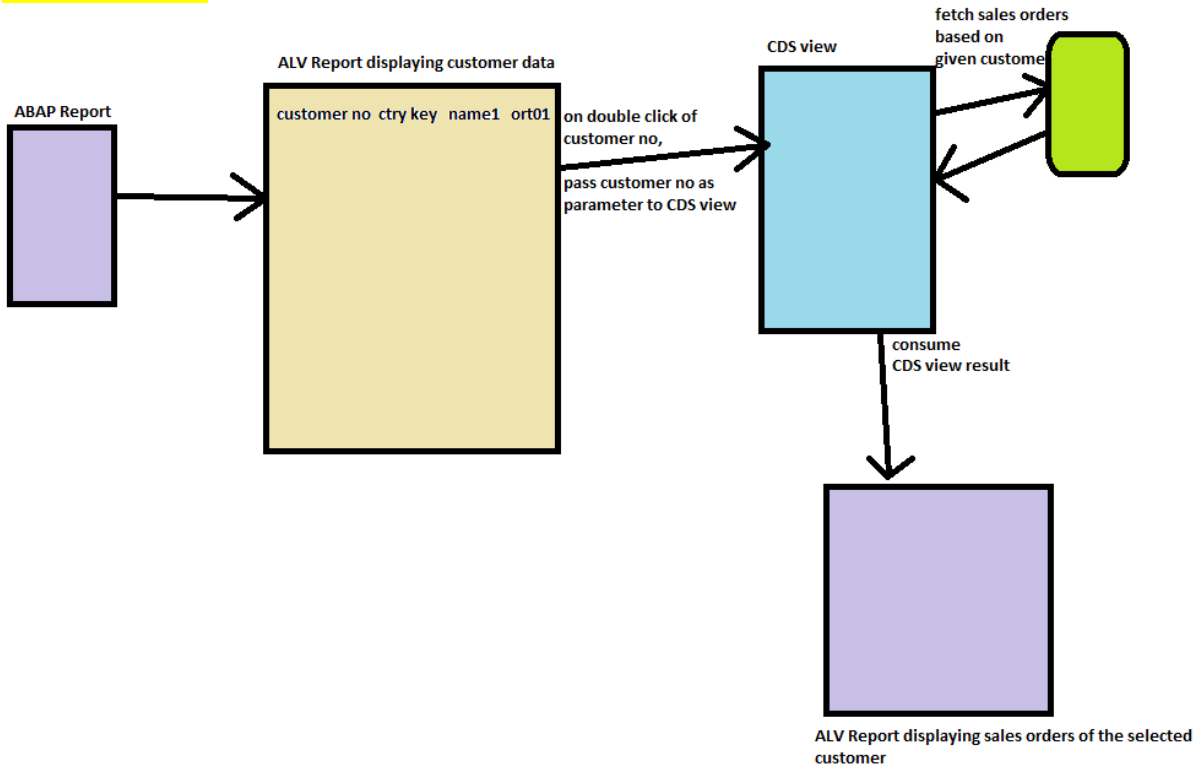
```
REPORT ZCONSUMECDSVIEW.

* Case 1 -Consuming CDS view from Open SQL

*select * from ZCDSSOVIEW into table @data(ITAB).
*data wa like line of itab.
*
*loop at itab into wa.
*write :/ wa-NODE,
*         wa-SALESORDER,
*         wa-PAYMENT_STATUS,
*         wa-BUYERID.
*endloop.

* Case 2 - Consume Generated Dictionary view from OpenSQL
*select * from ZCDSSOVIEWSQL into table @data(ITAB).
*data wa like line of itab.
*
*loop at itab into wa.
*write :/ wa-NODE,
*         wa-SALESORDER,
*         wa-PAYMENT_STATUS,
*         wa-BUYERID.
*endloop.

* Case 3 - Consume CDS View from ALV IDA
data o_Alv type ref to IF_SALV_GUI_TABLE_IDA.
try.
call method cl_salv_gui_table_ida=>CREATE_FOR_CDS_VIEW
```

```
  EXPORTING
    IV_CDS_VIEW_NAME                = 'ZCDSSOVIEW'
  RECEIVING
    RO_ALV_GUI_TABLE_IDA            = o_alv.

  CATCH CX_SALV_DB_CONNECTION.
  CATCH CX_SALV_DB_TABLE_NOT_SUPPORTED.
  CATCH CX_SALV_IDA_CONTRACT_VIOLATION.
  CATCH CX_SALV_FUNCTION_NOT_SUPPORTED.
endtry.

  if o_alv is bound.
    o_alv->fullscreen( )->DISPLAY( ).
  endif.
```



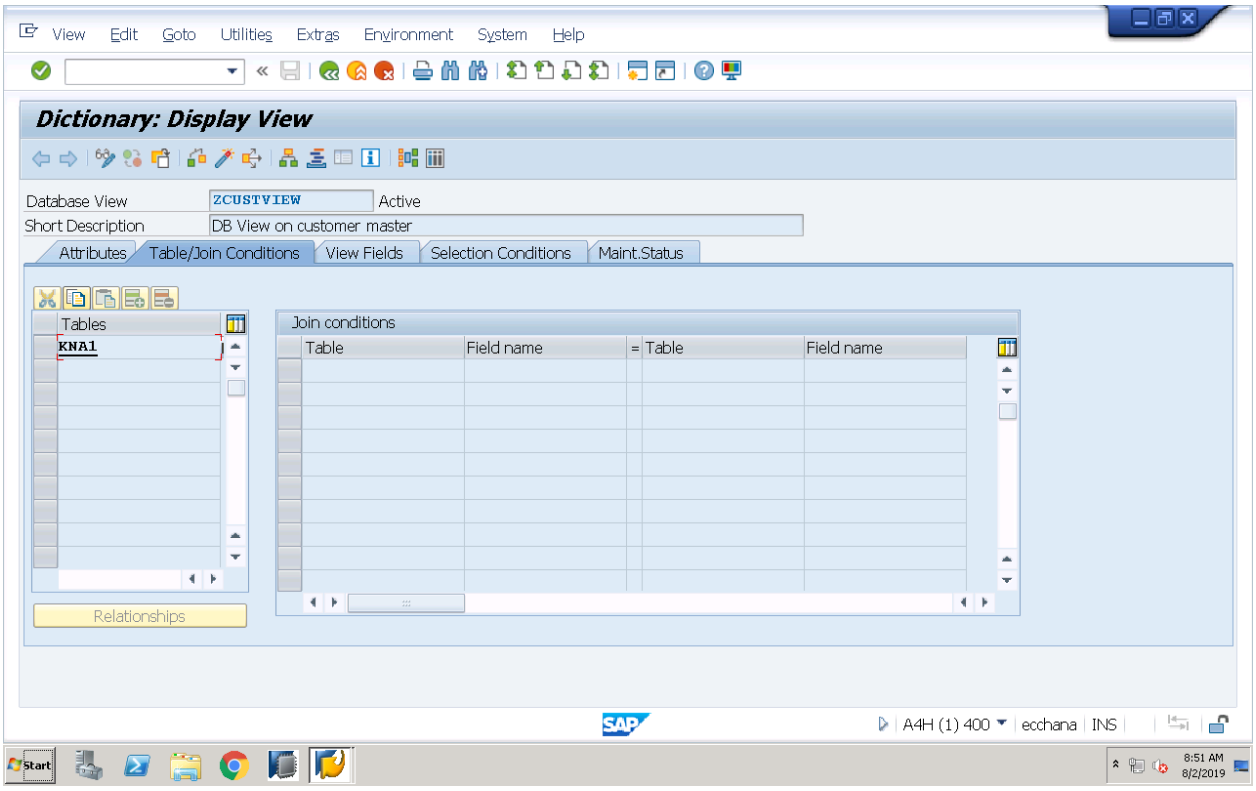


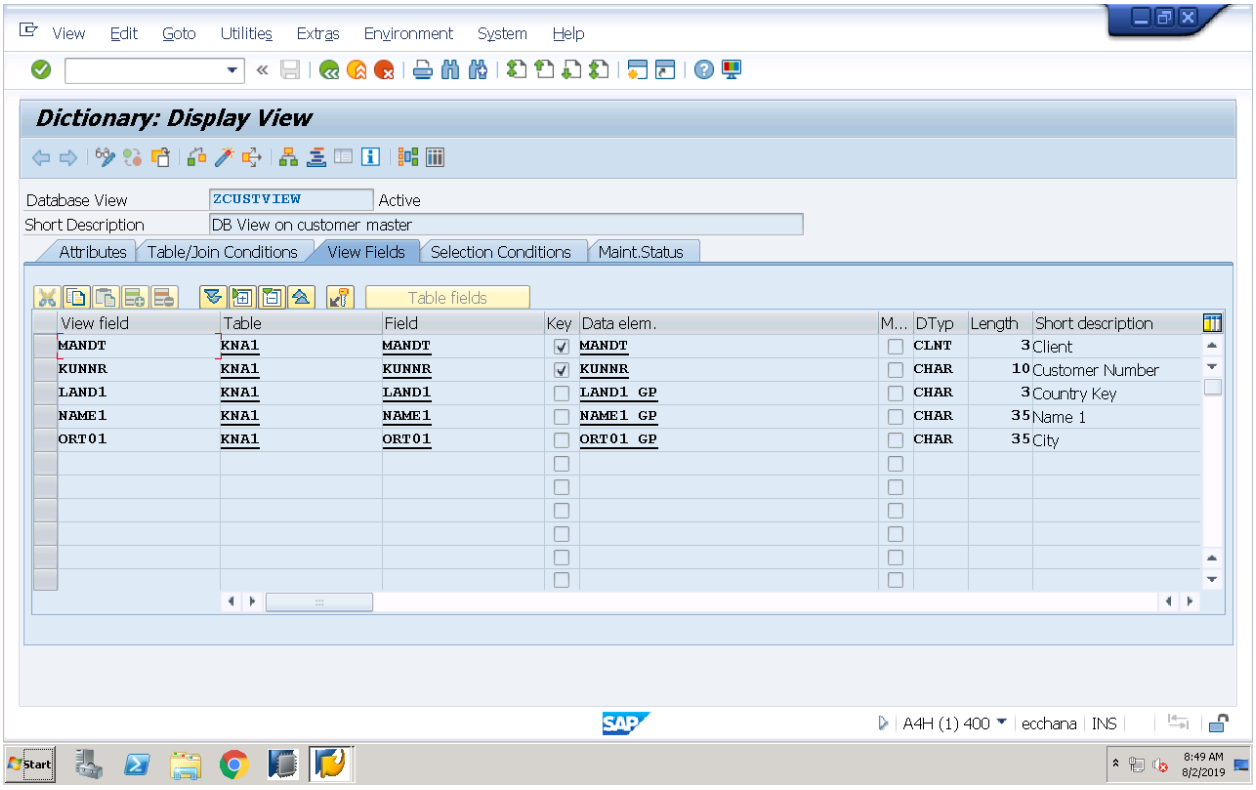
02-08-2019



Example: Interactive ALV IDA and CDS view with parameters

1. Create a Dictionary Database View referring to customer master fields of KNA1 DB Table





Save, check for syntax errors and activate

2. In Hana Studio, create CDS view with parameters

```
@AbapCatalog.sqlViewName: 'ZCDSSODBVIEW'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View to Get Sales Orders of given customer'
define view ZCDSGETSALESORDERS
    with parameters CUSTNO : kunnr
as select from vbak {
    vbeln,
    erdat,
    erzet,
    ernam,
    kunnr
      } where kunnr = $parameters.CUSTNO
```

Save, check and activate

3. Create a executable program for consuming above dictionary database view and also CDS view with parameters (Interactive Reporting using ALV IDA - double_click event)

```
REPORT ZCONSUMECDSVIEWWITHPARAM.

data wa_kna1 type zcustview.
data o_alv type ref to IF_SALV_GUI_TABLE_IDA.

class lcl_eventreceiver DEFINITION.
  public section.
     methods handle_double_click
          for event double_click of IF_SALV_GUI_TABLE_DISPLAY_OPT
            importing EV_FIELD_NAME EO_ROW_DATA.
endclass.

class lcl_eventreceiver IMPLEMENTATION.
  method handle_double_click.
    case ev_field_name.
      when 'KUNNR'.
        message 'Double clicked on customer' type 'I'.
* Extract customer no from interacted row
        clear wa_kna1.
```

```
        TRY.
        CALL METHOD EO_ROW_DATA->GET_ROW_DATA
          IMPORTING
            ES_ROW                = wa_kna1.
       CATCH CX_SALV_IDA_CONTRACT_VIOLATION .
         message 'Exception in IDA contract violation while fetching s
elected row' type 'I'.
       CATCH CX_SALV_IDA_SEL_ROW_DELETED .
         message 'Exception in selected row deleted while fetching it'
type 'I'.
        ENDTRY.

    if wa_kna1 is not INITIAL.
* Consume CDS view
     TRY.
      CALL METHOD CL_SALV_GUI_TABLE_IDA=>CREATE_FOR_CDS_VIEW
        EXPORTING
          IV_CDS_VIEW_NAME      = 'ZCDSGETSALESORDERS'
        RECEIVING
          RO_ALV_GUI_TABLE_IDA  = o_alv.
       CATCH CX_SALV_DB_CONNECTION .
         message 'Exception in DB connection while consuming CDS view'
type 'I'.
       CATCH CX_SALV_DB_TABLE_NOT_SUPPORTED .
         message 'Exception in DB Table not supported while consuming
CDS view' type 'I'.
       CATCH CX_SALV_IDA_CONTRACT_VIOLATION .
         message 'Exception in IDA contract violation while consuming
CDS view' type 'I'.
       CATCH CX_SALV_FUNCTION_NOT_SUPPORTED .
         message 'Exception in Function Not supported while consuming
CDS view' type 'I'.
      ENDTRY.
    endif.

    if o_alv is bound.
* Prepare  parameters for CDS view
      data : t_param type IF_SALV_GUI_TYPES_IDA=>YT_PARAMETER,
             wa_param like line of t_param.

      clear wa_param.
      wa_param-name = 'CUSTNO'.
      wa_param-value = wa_kna1-kunnr.
      append wa_param to t_param.

* set/pass parameters for CDS view
```

```
      o_alv->SET_VIEW_PARAMETERS( t_param ).

* Activate fullscreen and display the CDS view result
    o_alv->fullscreen( )->display( ).
    endif.
       when others.
         message 'Please double click on customer' type 'I'.
    endcase.
  endmethod.
endclass.

data ob type ref to lcl_eventreceiver.

START-OF-SELECTION.
* get the ALV object based on Dictionary Database view
TRY.
CALL METHOD CL_SALV_GUI_TABLE_IDA=>CREATE
  EXPORTING
    IV_TABLE_NAME         = 'ZCUSTVIEW'
  RECEIVING
    RO_ALV_GUI_TABLE_IDA  = o_alv.
 CATCH CX_SALV_DB_CONNECTION .
   message 'EXception in DB connection while getting ALV object based
on dictionary view' type 'I'.
 CATCH CX_SALV_DB_TABLE_NOT_SUPPORTED .
   message 'EXception in DB Table not supported while getting ALV obje
ct based on dictionary view' type 'I'.
 CATCH CX_SALV_IDA_CONTRACT_VIOLATION .
   message 'EXception in IDA Contract violation while getting ALV obje
ct based on dictionary view' type 'I'.
ENDTRY.

if o_alv is bound.
* get the instance of fullscreen mode
  data o_fullscreen type ref to IF_SALV_GUI_FULLSCREEN_IDA.
 TRY.
  CALL METHOD O_ALV->FULLSCREEN
    RECEIVING
      RO_FULLSCREEN = o_fullscreen.
   CATCH CX_SALV_IDA_CONTRACT_VIOLATION .
     message 'EXception in IDA Contract violation while getting ALV fu
llscreen object ' type 'I'.
   ENDTRY.

   if o_fullscreen is bound.
* get the reference of display options interface
```

```
      data o_alv_disp type ref to IF_SALV_GUI_TABLE_DISPLAY_OPT.
      o_alv_disp = o_alv->display_options( ).
      if o_alv_disp is bound.
* Enable the Double click
        o_alv_disp->enable_double_click( ).
* Register the handler for executing event handler method
        create object ob.
        set handler ob->handle_double_click for ALL INSTANCES.
      endif.
* Display the customer master data on the UI
    o_fullscreen->display( ).
   endif.
endif.
```

05-08-2019

Example: CDS View with Multiple Parameters and Currency_Conversion Function

CDS View Definition:

```
@AbapCatalog.sqlViewName: 'ZCDS_FLIGHTS'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View to fetch Flight Details based on
Multiple parameter'
define view ZCDSVIEWFLIGHTDETAILS
    with parameters p_carrid : s_carr_id,
                    p_curr   : abap.cuky(5),
                    p_date   : abap.dats
as select from sflight as f {
    f.carrid,
    f.connid,
    f.fldate,
    currency_conversion( amount => f.price,
                         source_currency => f.currency,
                         target_currency => :p_curr,
                         exchange_rate_date => :p_date ) as amount
} where carrid = $parameters.p_carrid
```

Save, Check and Activate

Executable Program: Consuming above CDS view using OpenSQL statement

```
REPORT ZCONSUMECDS_MULTIPLEPARAM.

PARAMETERS : p_cid type sflight-carrid,
             p_cur type tcurr-tcurr,
```

```
            p_dat type dats.

select * from ZCDSVIEWFLIGHTDETAILS( p_carrid = @p_cid, p_curr =
@p_cur, p_date = @p_dat ) INTO TABLE @DATA(T_FLIGHTS).

call method cl_demo_output=>DISPLAY_DATA
  EXPORTING
    VALUE = t_flights
    NAME  = 'CDS View with Multiuple parameters'.
```

Save, Check, activate and Execute with Input values (AA,EUR,20180124)

Example: CDS View with Optional Parameters

CDS View Definition:

```
@AbapCatalog.sqlViewName: 'ZCDSOPTDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View with optional parameters'
define view ZCDSVIEWOPTIONALPARAM
    with parameters
     @Environment.systemField: #SYSTEM_LANGUAGE
     p1 : spras
as select from makt {
    matnr as MaterialNo,
    :p1 as LanguageKey,
    maktx as MaterialDesc
} where spras = $parameters.p1
```

Save, Check, activate and Execute with different language keys (E,D,F…). Also Check by passing blank value as language key (It returns material and its descriptions maintained in Login Language).

Note: Optional parameters in CDS views are supported in limited way and following are the environment variables which can be used to provide alternate values for the parameters in CDS views

```
  @Environment.systemField = #CLIENT (sy-mandt)

  @Environment.systemField = #SYSTEM_DATE (sy-datum)

  @Environment.systemField = #SYSTEM_TIME (sy-uzeit)

  @Environment.systemField = #SYSTEM_LANGUAGE (sy-langu)

  @Environment.systemField = #USER (sy-uname)
```

08-08-2019

Example: CDS Views with Functions and Arithmetic Expressions

```
@AbapCatalog.sqlViewName: 'ZCDSARITH'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS Views with functions and arithmetic
expressions'
define view ZCDSARITHMETIC as select from vbak {
    vbeln,
    ernam,
// substring function
    SUBSTRING(ernam,2,5) as SHORT_NAME,
    netwr,
// Arithmetic Expressions
   ((netwr * 100)+10) as GROSS_AMT,
   DIV(netwr,10) as DIV_AMT,
// Case Statement
   case when netwr > 10000 then 'HIGH ORDER'
        when netwr between 4000 and 10000 then 'MID ORDER'
        else 'LOW ORDER'
   end as AMT_TYPE,
   auart
}
```

Save, Check and Activate

Preview the data from CDS view by right click

Example: CDS Views with Aggregate Functions, Group by and Having Clauses

```
@AbapCatalog.sqlViewName: 'ZCDS_AGGRVIEW'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View with aggregrate functions,group
by,having clause'
define view ZCDS_AGGR as select from sflight {
    sflight.carrid,
    sflight.connid,
    sum( price ) as Total_price,
    sflight.currency
}group by carrid,connid,currency
  having sum(price) > 10000
```

Save, Check and Activate

Preview the data from CDS view by right click

09-08-2019

ABAP CDS Table Functions Implemented in AMDP

In an ABAP CDS Table Function development, we define an entity with the field structure, parameters (optional) and an association to a class/method. With AMDP we're able to write database procedures directly in the ABAP layer and encapsulate inside the class/method we defined in the Table Function, the call works as the same like any other ABAP methods and we have the following advantages:

- Detailed analysis of runtime errors through ST22;
- Database procedures debug available through HANA Studio;
- Transport identical as to ABAP classes.
- Since AMDP works directly with database scripting some extra steps are necessary like the definition of database and the script language to be used (in SAP HANA the language is the SQL Script). .

Scenario

- Each airline company provides flight connections to different cities around the world, the user wants to see all the cities supported by a specific airline in a single field separate by comma. Since the number of cities can be different for each one of the airlines we need to generate a logic to concatenate all the cities no matter how many entries are returned.
- Through a common ABAP CDS View we could use a CONCAT function, but in this case we would need to define a fixed quantity of fields, since the CDS View can't handle this logic dynamically how should we proceed?
- In ABAP CDS Table Function, we can implement this with a simple database function called STRING_AGG (String Aggregation). This function is available in the SQL Script but currently there is no support through ABAP CDS View.

Procedure:

1. **Create the CDS entity** (ZCDSTABLEFUNCTION) **by choosing the template 'Define Table function with parameters'**

```
@EndUserText.label: 'CDS View Integrated with AMDP Table
Function'
define table function ZCDSTABLEFUNCTION
returns {
```

```
    client            :   abap.clnt;
    airline_code      :   s_carr_id;
    airline_name      :   s_carrname;
    cities_operated   :   abap.string;
  }
  implemented by method ZAMDP_FLIGHTS=>FLIGHT_CONNECTIONS;
```

Save, Check and Activate

2. **Create the AMDP Class and AMDP Method using ADT in ABAP Perspective**

```
CLASS ZAMDP_FLIGHTS DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
       INTERFACES IF_AMDP_MARKER_HDB.
CLASS-METHODS FLIGHT_CONNECTIONS FOR TABLE FUNCTION
ZCDSTABLEFUNCTION.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.


CLASS ZAMDP_FLIGHTS IMPLEMENTATION.

METHOD FLIGHT_CONNECTIONS BY DATABASE FUNCTION FOR HDB LANGUAGE
SQLSCRIPT OPTIONS READ-ONLY USING SFLIGHTS.

  ITAB = SELECT DISTINCT SFLIGHTS.MANDT AS CLIENT,
                         SFLIGHTS.CARRID AS AIRLINE_CODE,
                         SFLIGHTS.CARRNAME AS AIRLINE_NAME,
                         SFLIGHTS.CITYTO AS CITIES_OPERATED
                         FROM SFLIGHTS;

   RETURN SELECT CLIENT, AIRLINE_CODE, AIRLINE_NAME,
          STRING_AGG( CITIES_OPERATED,',' ORDER BY CITIES_OPERATED)
AS CITIES_OPERATED FROM :ITAB
          GROUP BY CLIENT, AIRLINE_CODE, AIRLINE_NAME;

   ENDMETHOD.
   ENDCLASS.
```

Save, Check and Activate

3. **Consuming above CDS using Open SQL in ABAP Report**

```
REPORT ZCONSUME_CDS_AMDP.

select * from zcdstablefunction into table @data(lt_flights).

cl_demo_output=>display_data( EXPORTING value = lt_flights ).
```

airline_code	airline_name	cities_operated
AA	American Airlines	SAN FRANCISCO, NEW YORK
AZ	ALITALIA	FRANK FURT, ROME

CDS View Integrated with AMDP Table Function

12-08-2019

Exposing CDS views as ODATA Service: This can be done in 3 ways

1. **By using the annotation @OData.publish:true**
2. **By referring to DDIC view generated by CDS view**
3. **By referring to CDS view itself**

Example: CDS View without parameter exposed as ODATA Service (using the annotation @OData.publish:true)

```
@AbapCatalog.sqlViewName: 'ZCDSODATADB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
```

```
@EndUserText.label: 'CDS View Exposed as ODATA Service'
@OData.publish: true
define view ZCDSODATA as select from vbak {
    key vbeln as sono,
    erdat,
    erzet,
    ernam
}  Save, check and activate → Creates ODATA Service
```

Logon to SAP System using SAP GUI

1. **Activate the same using the t-code '/IWFND/MAINT_SERVICE'**
2. **Maintain the Service using the t-code '/IWFND/GW_CLIENT'**

Case 1: /sap/opu/odata/sap/ZCDSODATA_CDS/$metadata --> Returns metadata about the service (entity name, entity set name, key properties and non-key properties, data type of properties....)

Case 2: /sap/opu/odata/sap/ZCDSODATA_CDS/ZCDSODATA ---> Returns Multiple records

Case 3: /sap/opu/odata/sap/ZCDSODATA_CDS/ZCDSODATA('18317') --> Returns specific record

Case 4: /sap/opu/odata/sap/ZCDSODATA_CDS/ZCDSODATA('18317')/$count --> Returns the count as '1' as we are reading specific record

Case 5: /sap/opu/odata/sap/ZCDSODATA_CDS/ZCDSODATA/$count --> Returns the count of records returned by the query

CDS View	ODATA Service
ZCDSODATA	ZCDSODATA_CDS

Example: CDS View with parameter exposed as ODATA Service (using the annotation @OData.publish:true)

```
@AbapCatalog.sqlViewName: 'ZCDSODATAPARAMDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View with parameter exposed as ODATA Service'
@OData.publish: true
define view ZCDS_ODATA_PARAM
    with parameters p_vkorg : vkorg
as select from vbak {
    key vbeln as sono,
    erdat,
    erzet,
    ernam,
    vkorg
} where vkorg = $parameters.p_vkorg
```

Save, check and activate → Creates ODATA Service

Logon to SAP System using SAP GUI

1. **Activate the same using the t-code '/IWFND/MAINT_SERVICE'**
2. **Maintain the Service using the t-code '/IWFND/GW_CLIENT'**

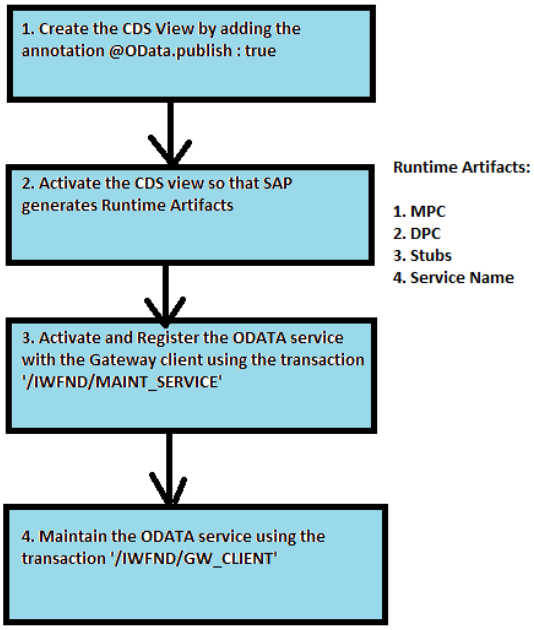
Generates 2 entity types and 2 entity sets as follows:

Entitytype	Entityset	keyproperties
ZCDS_ODATA_PARAMTYPE	zcds_odata_paramset	p_vkorg,sono
zcds_odata_paramparameters	zcds_odata_param	p_vkorg

Navigationproperty --> set

Case 1:
/sap/opu/odata/sap/ZCDS_ODATA_PARAM_CDS/ZCDS_ODATA_PARAMSet(p_vkorg='1010',sono='4343')

Case 2:
/sap/opu/odata/sap/ZCDS_ODATA_PARAM_CDS/ZCDS_ODATA_PARAM(p_vkorg='1010')/Set



13-08-2019

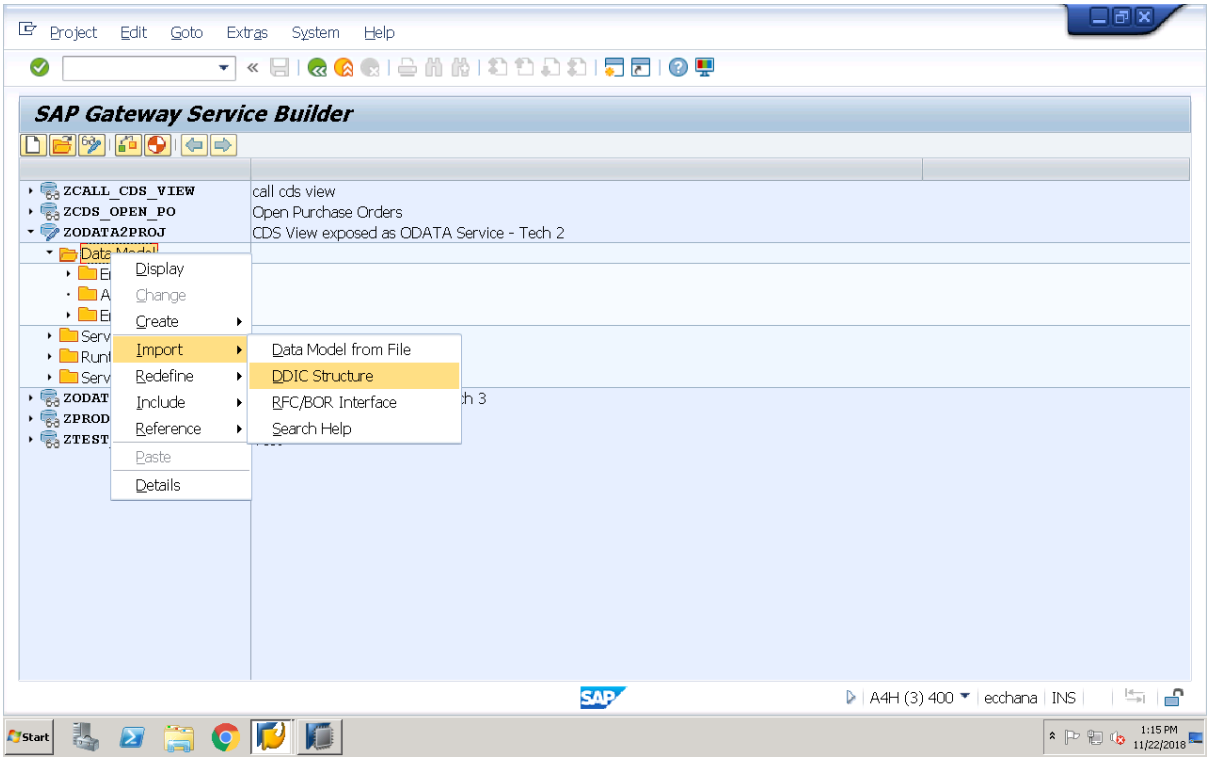
Example: Exposing CDS view as ODATA Service without using the annotation @ODATA.PUBLISH – Technique 2

```
@AbapCatalog.sqlViewName: 'ZODATA2DB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View exposed as odata service wout annot
odata.publish'
define view ZCDSVIEW_ODATA2 as select from vbak {
   key vbeln as sono,
   erdat,
   erzet,
   ernam
}
```

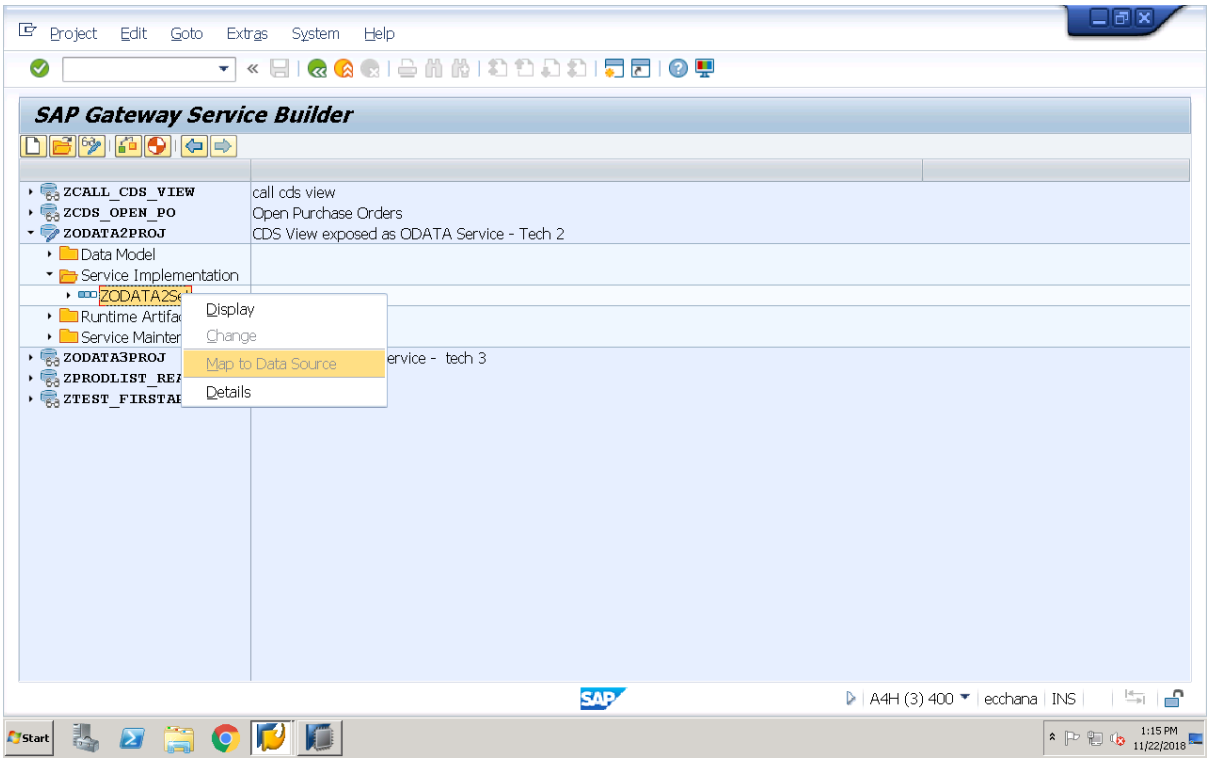
Save, check and activate the CDS View

Now, in SAP GUI, use the t-code 'SEGW' and create the project manually

As part of Project, import the data model referring to the dictionary view generated by above CDS view



Under service implementation, Implement the service by mapping the data source to the above CDS view (CDS View available: CDS~ZCDSVIEW_ODATA2)



Now, register and maintain the service

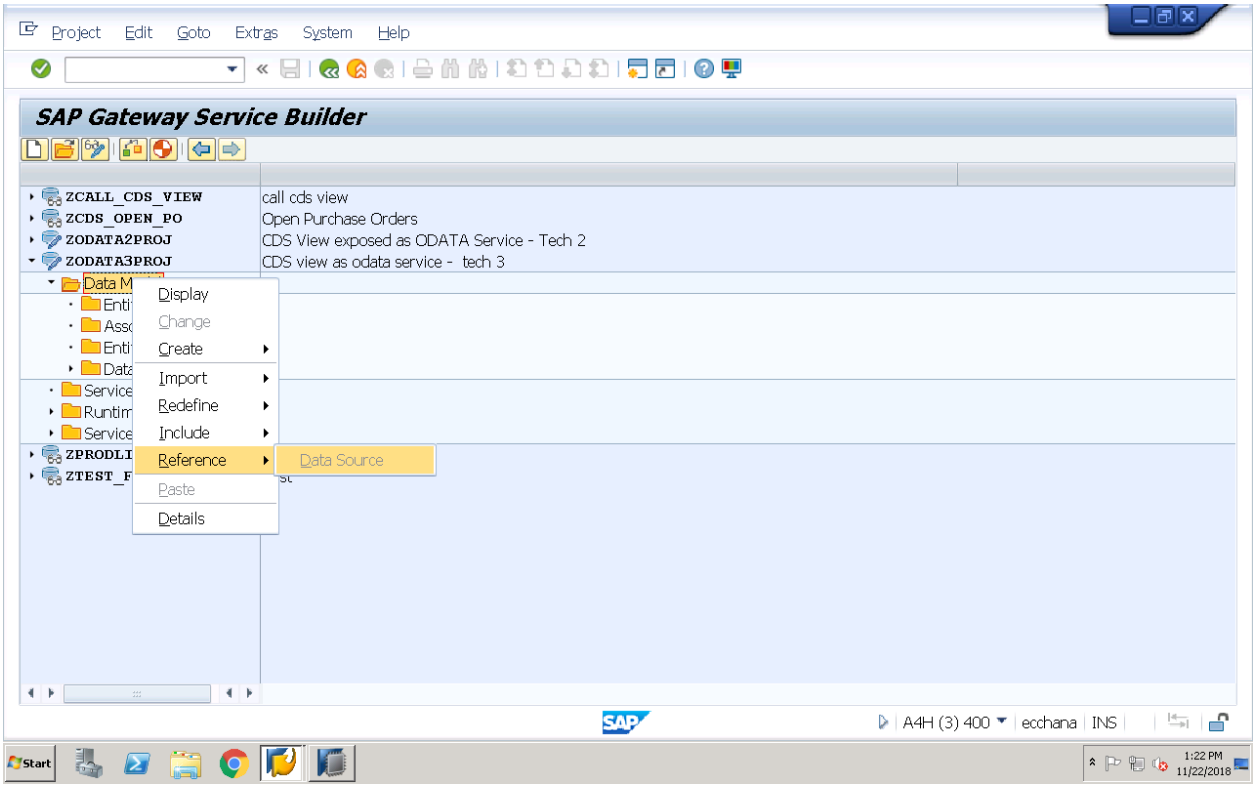
Example: Exposing CDS view as ODATA Service without using the annotation @ODATA.PUBLISH – Technique 3

```
@AbapCatalog.sqlViewName: 'ZODATA3DB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View exposed as odata service -tech 3'
define view ZCDSVIEW_ODATA3 as select from vbak {
 key vbeln as sono,
   erdat,
   erzet,
   ernam
}
```

Save, check and activate the CDS View

Now, in SAP GUI, use the t-code ‘SEGW’ and create the project manually

As part of Project, import the data model referring to the cds view directly



Now, register and maintain the service

14-08-2019

Example: Consuming CDS Views exposed as ODATA Service from UI5 Application in Eclipse IDE

1. **CDS View Creation:**

```
@AbapCatalog.sqlViewName: 'ZCDSODATADB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View Exposed as ODATA Service'
@OData.publish: true
define view ZCDSODATA as select from vbak {
    key vbeln as sono,
    erdat,
    erzet,
    ernam
}
```

Save, Check and Activate the CDS View → Creates ODATA Service with the naming standard 'ZCDSODATA_CDS' (<cds view name>_cds)

2. **Logon to SAP GUI, using the t-code '/IWFND/MAINT_SERVICE' , register the above odata service with the gateway client**

3. **Using the t-code '/IWFND/GW_CLIENT', collect the metadata of the odata service**
   /sap/opu/odata/sap/ZCDSODATA_CDS/$metadata (OR) click on 'call browser' and collect the HTTP URL 'http://ecchana.sapdemo.com:8001/sap/opu/odata/sap/ZCDSODATA_CDS/$metadata'

   EntitysetName: ZCDSODATA_CDS
   Properties: sono, erdat, erzet, ernam

4. **Create the UI5 Application without any initial views (choose 'sap.m')**

5. **Under Webcontent, create a subfolder 'views' and create XML view (MyView)**

MyView.view.xml → Presentation Logic

```
<core:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc"
xmlns="sap.m"
          controllerName="views.MyView"
xmlns:html="http://www.w3.org/1999/xhtml">
<Table id="idSalesOrdersTable" items="{ path: '/ZCDSODATA'}">
          <headerToolbar>
            <Toolbar>
        <Title text="Sales Order Details" level="H2"> </Title>
            </Toolbar>
          </headerToolbar>

          <infoToolbar>
               <OverflowToolbar>
                    <Label text="Sales Orders"/>
               </OverflowToolbar>
          </infoToolbar>

          <columns>

               <Column width="12em">
                    <Text text="Sales Order No" />
               </Column>

              <Column width="12em">
```

```
                <Text text="Creation Date" />
            </Column>

            <Column width="12em">
                <Text text="Creation Time" />
            </Column>

            <Column width="12em">
                <Text text="Created By" />
            </Column>

        </columns>

        <items>

            <ColumnListItem>
                <cells>
                    <Text text="{sono}" />
                    <Text text="{erdat}" />
                    <Text text="{erzet}" />
                    <Text text="{ernam}" />
                </cells>

            </ColumnListItem>
        </items>
    </Table>
                                    </core:View>
```

MyView.controller.js → Business Logic

```
onInit: function() {
debugger;
// Instantiate ODATA Model
var oDataModel=new
sap.ui.model.odata.ODataModel("proxy/http/ecchana.sapdemo.com:8001/
sap/opu/odata/sap/ZCDSODATA_CDS/");

// Set the Model at the view level
this.getView().setModel(oDataModel);

                                        },
```

Index.html → Loading the initial view, setting path of the view, complex binding registration

```
        <script src="resources/sap-ui-core.js"
```

```
                    id="sap-ui-bootstrap"
                    data-sap-ui-libs="sap.m"
               data-sap-ui-resourceroots='{"views" : "./views"}'
                    data-sap-ui-xx-bindingSyntax="complex"
                    data-sap-ui-theme="sap_bluecrystal">
          </script>
          <!-- only load the mobile lib "sap.m" and the
"sap_bluecrystal" theme -->

          <script>
// Load the initial view
sap.ui.view({
     id            :   "idMyView",
     viewName      :   "views.MyView",
     type          :   "XML"
}).placeAt("content");

          </script>

     </head>
     <body class="sapUiBody" role="application">
          <div id="content"></div>
     </body>
                                   </html>
```

16-08-2019

Example: Joins in CDS views

```
@AbapCatalog.sqlViewName: 'ZCDS_JOINSDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Joins on CDS View'
define view ZCDS_JOINS as select from spfli
//inner join scarr
//join scarr
//left outer join scarr
cross join scarr
//    on spfli.carrid = scarr.carrid
    {
        spfli.carrid,
        spfli.connid,
        spfli.countryfr,
        spfli.airpfrom,
        spfli.countryto,
        spfli.cityto,
        scarr.carrname
```

```
}
```

Save, Check and activate the CDS view

Test the CDS view using Data preview by uncommenting different join statements.

Note: For Cross-Join, On statement should not be provided.

Cross-Join is similar to Cartesian product in Database View created in ABAP Dictionary without any join conditions

TABLE A

A	B	C	D
a1	b1	c1	d1
a2	b2	c2	d2
a3	b3	c3	d3
a4	b4	c4	d4
a5	b5	c5	d1
a6	b6	c6	d6

TABLE B

E	F	G	H	I
e1	f1	g1	h1	i1
e2	f2	g2	h2	i2
d1	f3	g3	h3	i3
d3	f4	g4	h4	i4
d2	f5	g5	h5	i5
d3	f6	g6	h6	i6

left
Table A outer join Table B
on A~D = B~E

A	B	C	D	E	F	G	H	I
a1	b1	c1	d1	d1	f3	g3	h3	i3
a2	b2	c2	d2	d2	f5	g5	h5	i5
a3	b3	c3	d3	d3	f4	g4	h4	i4
a3	b3	c3	d3	d3	f6	g6	h6	i6
a4	b4	c4	d4	0	0	null	0	nul
a5	b5	c5	d1	d1	f3	g3	h3	i3
a6	b6	c6	d6	0	0	null	0	nul

TABLE A

A	B	C	D
a1	b1	c1	d1
a2	b2	c2	d2
a3	b3	c3	d3
a4	b4	c4	d4
a5	b5	c5	d1
a6	b6	c6	d6

TABLE B

E	F	G	H	I
e1	f1	g1	h1	i1
e2	f2	g2	h2	i2
d1	f3	g3	h3	i3
d3	f4	g4	h4	i4
d2	f5	g5	h5	i5
d3	f6	g6	h6	i6

Table A inner join Table B
on A~D = B~E

A	B	C	D	E	F	G	H	I
a1	b1	c1	d1	d1	f3	g3	h3	i3
a2	b2	c2	d2	d2	f5	g5	h5	i5
a3	b3	c3	d3	d3	f4	g4	h4	i4
a3	b3	c3	d3	d3	f6	g6	h6	i6
a5	b5	c5	d1	d1	f3	g3	h3	i3

22-08-2019

Example: Associations in CDS views

```
@AbapCatalog.sqlViewName: 'ZCDSVIEWASSOCDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View with Associations'
define view ZCDSVIEWASSOC as select from spfli
association to sflight as _sfli
    on spfli.carrid = _sfli.carrid and
       spfli.connid = _sfli.connid
       association [1..1] to sairport as _sair
--       on spfli.airpfrom = _sair.id
         on $projection.airportfrom = _sair.id
       {
           spfli.carrid,
           spfli.connid,
           spfli.airpfrom as airportfrom,
           _sfli,
           _sair
}
```

Save, Check and Activate the CDS View

For Testing in HANA Studio:

Right Click on CDS view → Open with Data preview → Displays data only source entity (SPFLI Table), Now Right click on any row and choose follow associations, Displays the associations defined, Double Click on the appropriate association, displays the corresponding data

Example: Consuming above CDS view using OPEN SQL without referring to Association Fields

```
REPORT ZCONSUMECDSVIEWASSOCIATION.

select carrid, connid, airportfrom
  from zcdsviewassoc
     into table @data(itab).
if itab is not INITIAL.
  CALL METHOD cl_demo_output=>display_data
    EXPORTING
      value  = itab
      name   = 'Flights'.
endif.
```

Example: Consuming above CDS view using OPEN SQL in sorting sequence of CARRID Field (Order By Clause) without referring to Association Fields

```
REPORT ZCONSUMECDSVIEWASSOCIATION.

select carrid, connid, airportfrom
  from zcdsviewassoc
     into table @data(itab)
  order by carrid.
if itab is not INITIAL.
  CALL METHOD cl_demo_output=>display_data
    EXPORTING
      value  = itab
      name   = 'Flights'.
endif.
```

Example: Consuming above CDS view using OPEN SQL referring to Association Fields (Path Expression)

```
REPORT ZCONSUMECDSVIEWASSOCIATION.

*select carrid, connid, airportfrom, \_sfli-fldate
*  from zcdsviewassoc
*     into table @data(itab). "syntax error

*select carrid, connid, airportfrom, \_sfli-fldate as flightdate, \
_sair-name as airportname
*  from zcdsviewassoc
*     into table @data(itab). "valid

*select carrid, connid, airportfrom, \_sfli-fldate as flightdate, \
_sair-name as airportname
*  from zcdsviewassoc
*     into table @data(itab)
*  order by carrid, connid. "syntax error

select carrid, connid, airportfrom, \_sfli-fldate as flightdate, \
_sair-name as airportname
  from zcdsviewassoc
  order by carrid, connid
     into table @data(itab).
if itab is not INITIAL.
  CALL METHOD cl_demo_output=>display_data
```

```
    EXPORTING
      value   = itab
      name    = 'Flights'.
endif.
```

23-08-2019

Example: CDS view with Association based on Existing CDS view (Refer to 22nd Aug Example for the Existing CDS view)

```
@AbapCatalog.sqlViewName: 'ZCDSVIEWASSOC2DB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View with associations between DB table and a CDS view'
define view ZCDSVIEWASSOC2 as select from scarr
   association to ZCDSVIEWASSOC as _cdsassoc on
       $projection.carrierid = _cdsassoc.carrid
        {
              scarr.carrid as carrierid,
              scarr.carrname,
              scarr.url,
              _cdsassoc
}
```

Save, Check and Activate the CDS view

Check the Data preview

Executable program: Consuming Above CDS view

```
REPORT ZCONSUMECDSVIEWASSOCIATION1.

select carrierid, carrname, url, \_cdsassoc\_sfli-fldate as flightdat
e, \_cdsassoc\_sair-name as airportname
  from ZCDSVIEWASSOC2
  into table @data(itab).
if itab is not INITIAL.
  CALL METHOD cl_demo_output=>display_data
    EXPORTING
      value   = itab
      name    = 'Flight Details'.
endif.
```

Executable program: Consuming above CDS view with Filter Condition

```
REPORT ZCONSUMECDSVIEWASSOCIATION1.

PARAMETERS p_carrid type scarr-carrid.

*select carrierid, carrname, url, \_cdsassoc\_sfli-fldate as flightdat
e,
*  \_cdsassoc\_sair-name as airportname
*  from ZCDSVIEWASSOC2
*  into table @data(itab)
*  where carrierid = @p_carrid. "syntax error

select carrierid, carrname, url, \_cdsassoc\_sfli-fldate as flightdat
e, \_cdsassoc\_sair-name as airportname
  from ZCDSVIEWASSOC2
    where carrierid = @p_carrid
  into table @data(itab).
if itab is not INITIAL.
  CALL METHOD cl_demo_output=>display_data
    EXPORTING
      value  = itab
      name   = 'Flight Details'.
endif.
```

27-08-2019

Example: CDS view with Join on Two Tables

```
@AbapCatalog.sqlViewName: 'ZCDSJAPARAMDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS view with Joins, Associations and Parameters'
define view ZCDS_JOIN_ASSOC_PARAM
      with parameters P_VBELN : vbeln_vf,
                      P_SPRAS : spras
     as select from vbrk
               left outer join vbrp
                  on vbrk.vbeln = vbrp.vbeln
{
      vbrk.vbeln,
      vbrk.erdat,
      vbrk.land1,
      vbrp.posnr,
      vbrp.matnr
}
   where vbrk.vbeln = $parameters.P_VBELN
```

Save, Check and Activate

Testing: Preview the data by providing Billing Document No (0090000001) and Language Key (D) → Returns 2 rows as there are 2 line items

Example: CDS view with Join on Three Tables

```
@AbapCatalog.sqlViewName: 'ZCDSJAPARAMDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS view with Joins, Associations and Parameters'
define view ZCDS_JOIN_ASSOC_PARAM
      with parameters P_VBELN : vbeln_vf,
                      P_SPRAS : spras
     as select from vbrk
               left outer join vbrp
                  on vbrk.vbeln = vbrp.vbeln
               left outer join makt
                  on vbrp.matnr = makt.matnr
{
      vbrk.vbeln,
      vbrk.erdat,
      vbrk.land1,
      vbrp.posnr,
      vbrp.matnr,
      makt.maktx
}
   where vbrk.vbeln = $parameters.P_VBELN
```

Save, Check and Activate

Testing: Preview the data by providing Billing Document No (0090000001) and Language Key (D) → Returns 2 rows with material description as there are 2 line items

Example: CDS view with Join on Three Tables with parameters

```
@AbapCatalog.sqlViewName: 'ZCDSJAPARAMDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS view with Joins, Associations and Parameters'
define view ZCDS_JOIN_ASSOC_PARAM
      with parameters P_VBELN : vbeln_vf,
```

```
                        P_SPRAS : spras
     as select from vbrk
               left outer join vbrp
                  on vbrk.vbeln = vbrp.vbeln
               left outer join makt
                  on vbrp.matnr = makt.matnr
{
      vbrk.vbeln,
      vbrk.erdat,
      vbrk.land1,
      vbrp.posnr,
      vbrp.matnr,
      makt.maktx
}
   where vbrk.vbeln = $parameters.P_VBELN and
         makt.spras = $parameters.P_SPRAS
```

Save, Check and Activate

Testing 1: Preview the data by providing Billing Document No (0090000001) and Language Key (D) → Returns 2 line items with descriptions (No. of entries → 76)

Testing 2: Preview the data by providing Billing Document No (0090000001) and Language Key (E) → Returns 2 line items with descriptions (No. of entries → 76)

Testing 3: Preview the data by providing Billing Document No (0090000001) and Language Key (Z) → Returns No Data as there is no Language Key with 'Z' in MAKT table (No. of entries → 0)

Example: CDS view with Join on Two Tables with parameters and Association with COALESCE function

```
@AbapCatalog.sqlViewName: 'ZCDSJAPARAMDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS view with Joins, Associations and Parameters'
define view ZCDS_JOIN_ASSOC_PARAM
      with parameters P_VBELN : vbeln_vf,
                      P_SPRAS : spras
     as select from vbrk
               left outer join vbrp
                  on vbrk.vbeln = vbrp.vbeln
```

```
--          left outer join makt
       association [1..*] to makt
                  on vbrp.matnr = makt.matnr
{
      vbrk.vbeln,
      vbrk.erdat,
      vbrk.land1,
      vbrp.posnr,
      vbrp.matnr,
      COALESCE(makt[1: spras = :P_SPRAS].maktx,
               makt[1: spras = 'A'].maktx ) as maktx
}
   where vbrk.vbeln = $parameters.P_VBELN
```

Save, Check and Activate

Testing 1: Preview the data by providing Billing Document No (0090000001) and Language Key (D) → Returns 2 line items with descriptions (No. of entries → 76)

Testing 2: Preview the data by providing Billing Document No (0090000001) and Language Key (E) → Returns 2 line items with descriptions (No. of entries → 76)

Testing 3: Preview the data by providing Billing Document No (0090000001) and Language Key (Z) → Returns Material Description Maintained in Arabic for the two line items as there is No Data for the Language Key with 'Z' in MAKT table (No. of entries → 2)

Example: Consuming Above CDS views in ABAP Reports using OPEN SQL

```
REPORT ZCONSUMECDSJOINASSOCPARAM.

PARAMETERS : i_vbeln type vbeln_vf,
             i_spras type spras.

select * from ZCDS_JOIN_ASSOC_PARAM( p_vbeln = @i_vbeln, p_spras = @i_
spras )
     into table @data(itab).
if sy-subrc eq 0.
  CALL METHOD cl_demo_output=>display_data
    EXPORTING
      value  = itab
      name   = 'Billing Document'.
```

```
endif.
```

COALESCE: Returns First Not Null Value from given Expression

Example:

```
SELECT COALESCE('A','B','C') FROM DUMMY; // RETURNS A
SELECT COALESCE(NULL,'B','C') FROM DUMMY; // RETURNS B
SELECT COALESCE(NULL,NULL,'C') FROM DUMMY; // RETURNS C
SELECT COALESCE(NULL,NULL,NULL) FROM DUMMY; // RETURNS NULL (?)
```

28-08-2019

View on View: Processing of creating CDS views referring to existing CDS views. The advantage is instead of creating single CDS view with complex join conditions on many DB tables; we can go for View on View Concept

CDS Simple View: Fetching data from VBAK Table

```
@AbapCatalog.sqlViewName: 'ZCDSSIMPLE'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS Simple View on VBAK'
define view ZCDS_SIMPLEVIEW as select from vbak {
    key vbeln as sono,
       ernam,
       erdat,
       netwr
}
```

Save, Check and Activate

View on View: CDS view referring to above CDS view and VBAP table

```
@AbapCatalog.sqlViewName: 'ZCDSVV'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View on View'
define view ZCDSVIEWONVIEW as select from ZCDS_SIMPLEVIEW as H
  inner join vbap as I
    on H.sono = I.vbeln
{
    H.sono,
    I.posnr,
```

```
    I.matnr,
    I.netpr
}
```

Save, Check and Activate

CDS View Extension: It is a process of enhancing existing CDS views with additional fields and case expression columns. The additional fields will be added as append structure.

Base CDS View:

```
@AbapCatalog.sqlViewName: 'ZCDSBASE'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS Base View'
define view ZCDSBASEVIEW as select from snwd_so
{
    key node_key as Node,
        so_id as SalesOrder,
        buyer_guid as buyer_id,
        gross_amount,
        tax_amount,
        case billing_status
           when 'P' then 'Paid'
           when ' ' then 'Not Paid'
           else '?'
        end as payment_status
}
```

Save, Check and Activate

Extended CDS view: Choose the Template ‘Extend CDS View’

```
@AbapCatalog.sqlViewAppendName: 'ZCDSBASEAPPEND'
@EndUserText.label: 'CDS View Extension'
extend view ZCDSBASEVIEW with ZCDSBASEVIEW_EXT {
    case when gross_amount > 100000 and billing_status != 'P'
       then 'High Impact'
       else 'Fine'
    end as so_status
}
```

Save, Check and Activate

Note: Now, Activate the base view also and we can observe a spiral icon will appear at the beginning of Base view definition indicating that is has been extended.

Now, preview the data from base view or extended view, it shows the complete columns from both base and extended views

View on View / Nested view:

CDS View extension:

ABAP --> Extend standard DB tables

1. 'CI_...' (Customer Include)
2. Append Structure

CDS View name	Dictionary view
cds_original_view (5)	CDSOriginalDB (5 fields)
cds_ext_view (2)	CDSoriginalDB(5) + append structure(2)

ZCDSBASEVIEW	ZCDSBASE
ZCDSBASEVIEW_EXT	ZCDSBASE + APPEND

std. cds view -->SEPM_SDDL_SALESORDER_HEAD

dic view -->SEPM_SDDL_SO_HD

29-08-2019

Identifying Standard CDS views:

Technique 1: In Hana Studio, In ABAP perspective, select the instance and press CTRL + SHIFT + A (or) choose 'Navigate' Menu and select open abap development object, displays the popup, provide search string as 'type:ddls", Displays all CDS views

Technique 2: Open any standard table say 'SNDWD_SO', choose where used list and select 'DDL Definitions' and continue.

Example: Extending Standard CDS view 'sepm_sddl_salesorder_head' related to 'SNWD_SO' table

```
@AbapCatalog.sqlViewAppendName: 'ZEXTSTDDB'
@EndUserText.label: 'Extend Standard View'
extend view sepm_sddl_salesorder_head with ZEXTSTD
{
    overall_status,
    payment_method,
    payment_terms
}
```

Save, Check and activate, preview the data from the original standard CDS view, displays the fields from extension view also

Example: CDS View with explicit name list for the projection elements

```
@AbapCatalog.sqlViewName: 'ZCDSEXPDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View with Explicit Name List'
define view ZCDSEXPLICIT
(sales_doc,creation_date,creation_time,created_by)
as select from vbak
{
   key vbeln,
       erdat,
       erzet,
       ernam
}
```

Note: Above view is not extensible as it contains explicit name list

Extended View for above CDS view:

```
@AbapCatalog.sqlViewAppendName: 'ZCDSEXPEXTDB'
@EndUserText.label: 'Extending CDS view containing Explicit Name list'
extend view ZCDSEXPLICIT with ZCDSEXPLICITEXT
{
    netwr
}
```

Save, Check and activate, Generates error as the original view is having explicit name list / selection list

Changes to above original CDS view for doing extension:

```
@AbapCatalog.sqlViewName: 'ZCDSEXPDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS View with Explicit Name List'
define view ZCDSEXPLICIT
--(sales_doc,creation_date,creation_time,created_by)
as select from vbak
{
   key vbeln,
       erdat,
       erzet,
       ernam
}
```

Note: Above view is extendable as explicit name list is commented

Extended View for above CDS view:

```
@AbapCatalog.sqlViewAppendName: 'ZCDSEXPEXTDB'
@EndUserText.label: 'Extending CDS view containing Explicit Name list'
extend view ZCDSEXPLICIT with ZCDSEXPLICITEXT
{
    netwr
}
```

Save, Check and activate and preview the data from the original view, displays the fields from extended view also

Example: original view

```
@AbapCatalog.sqlViewName: 'ZCDSPRJLISTDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'CDS view'
define view ZCDSPRJLIST
as select from spfli
   inner join scarr on
```

```
   spfli.carrid = scarr.carrid
{
    key spfli.connid as flight,
        spfli.cityfrom as departure,
        spfli.cityto as destination,
        scarr.carrname as carrier
};
```

Save, check and activate

Note: Above CDS view is extendable because the annotation 'viewEnhancementCategory' by default is set to [#PROJECTION_LIST]

Extended view for above CDS view: Extension 1

```
@AbapCatalog.sqlViewAppendName: 'ZCDSPRJLISTEXTDB'
@EndUserText.label: 'Extending CDS view with annotation
viewEnhancementCategory'
extend view ZCDSPRJLIST with ZCDSPRJLISTEXT
{
   spfli.distance,
   spfli.distid as unit
}
```

Save, check and activate

Extended view for above CDS view: Extension 2

```
@AbapCatalog.sqlViewAppendName: 'ZCDSDB1'
@EndUserText.label: 'EXT 1'
extend view ZCDSPRJLIST with ZCDSPRJLISTEXT1 {
    spfli.period
}
```

Save, check and activate

Note: Now, activate the original view and preview the data, displays the fields from both the extension views

Example: Original CDS view

```
@AbapCatalog.sqlViewName: 'ZCDSPRJLISTDB'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@AbapCatalog.viewEnhancementCategory: [ #PROJECTION_LIST ]
--@AbapCatalog.viewEnhancementCategory: [ #NONE ]
```

```
@EndUserText.label: 'CDS view with annotation viewEnhancementCatagory
set to None'
define view ZCDSPRJLIST
as select from spfli
   inner join scarr on
   spfli.carrid = scarr.carrid
{
    key spfli.connid as flight,
        spfli.cityfrom as departure,
        spfli.cityto as destination,
        scarr.carrname as carrier
};
```

Note: In the Above CDS view, if the annotation 'viewEnhancementCategory' is set to [#NONE], then it cannot be extended

Consuming above CDS view from Open SQL:

```
REPORT ZCONSUMECDSVIEW_MULTIPLEXT.

select * from ZCDSPRJLIST into table @data(itab).
if sy-subrc eq 0.
  CALL METHOD cl_demo_output=>display_data
    EXPORTING
      value  = itab
      name   = 'Flight Details'.
endif.
```