Capgemini

SAP HANA

Lesson Name: ABAP Database connectivity

Lesson Objectives

After completing this lesson, participants will be able to -

- Understand ADBC
- Use ADBC to execute Native SQL statements

ADBC

- ADBC stands for ABAP Database Connectivity
- ADBC is an object based API
- It is used in ABAP applications where SAP HANA is installed as a secondary database side-by-side with the ABAP system.
- For such side-by-side systems, it is recommended to use ADBC API.
- It is used for native SQL calls in ABAP.
- As an API, it allows for the determination of where native SQL calls are used.
- It also supports exception handling.

Business Requirement

ADBC is flexible, object-oriented, and not difficult to use, as only two-three main classes are relevant in most cases

- It allows native SQL access providing
  - Flexibility
  - Where used list
  - Error Handling

- Main Classes are –
  - CL_SQL_CONNECTION
  - CL_SQL_STATEMENT
  - CL_SQL_RESULT_SET

ADBC

ABAP Database Connectivity = ADBC

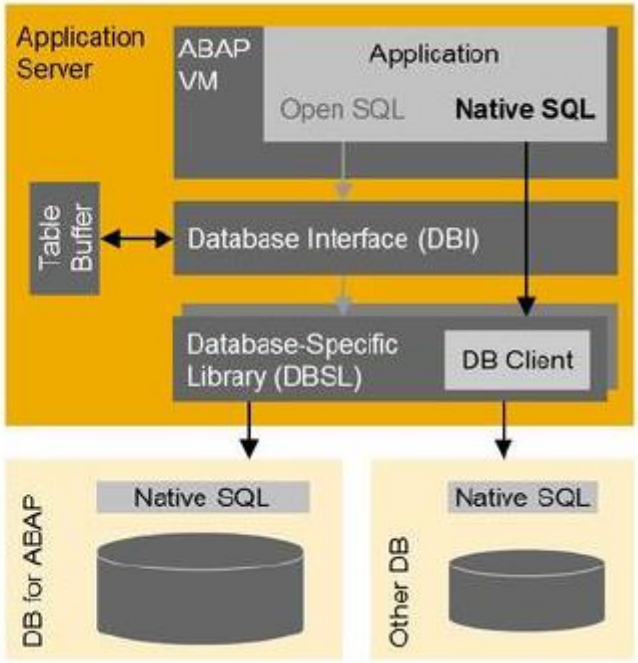
ABAP API for Native SQL calls

- Object-based
- Flexible
- Supports error handling
- Supports where-used list

  → **Recommended instead of older EXEC SQL statement**

Main classes:

- CL_SQL_CONNECTION
- CL_SQL_STATEMENT
- CL_SQL_RESULT_SET



Sequence for Reading Data with ADBC

1.	Choose database connection (only when accessing secondary DB)	Call method *get_connection()* of class CL_SQL_CONNECTION
2.	Create a statement object	Instantiation of class CL_SQL_STATEMENT
3.	Fill string variable with SQL syntax	Use either CONCATENATE or string templates/string expressions
4.	Issue native SQL call	Call method *execute_query()* of class CL_SQL_STATEMENT
5.	Assign target variable for result set	Call method *set_param()* or *set_param_table()* of class CL_SQL_RESULT_SET
6.	Retrieve result set	Call method *next_package()* of class CL_SQL_RESULT_SET
7.	Close query and release resources	Method *close()* of class CL_SQL_RESULT_SET

ADBC

In short, the steps in the previous slide can be summarized as below

- Choose database connection
  - cl_sql_connection=>get_connection
- Instantiate the statement object
- Construct the SQL (check with SQL Console for syntax)
- Issue Native SQL Call
- Assign target variable for result set
- Retrieve Result set
- Close the query and release resources

Disadvantages of ADBC

No hashed or sorted tables allowed as target

- Use standard table

No automatic client handling

- Specify MANDT in where condition

No guaranteed release of allocated resources on DB

- Close the query

ADBC

Coding Example: ABAP Database Connectivity (ADBC)

```
DATA: lo_con    TYPE REF TO cl_sql_connection,
      lo_sql    TYPE REF TO cl_sql_statement,
      lv_sql    TYPE string,
      lo_result TYPE REF TO cl_sql_result_set,
      lr_data   TYPE REF TO data,
      lt_flight TYPE STANDARD TABLE OF sflight.
TRY.
  lo_con = cl_sql_connection=>get_connection( 'HANA' ).

  CREATE OBJECT lo_sql
       EXPORTING
         con_ref = lo_con
         table_name_for_trace = 'SFLIGHT'.

  lv_sql = `SELECT ...        `.

  lo_result = lo_sql->execute_query( lv_sql ).

  GET REFERENCE OF lt_flight INTO lr_flight.
  lo_result->set_param_table( lr_flight ).
  lo_result->next_package( ).

  lo_result->close( ).
CATCH cx_sql_exception INTO ... .
      ...
ENDTRY.
```

Prepare native SQL call
- Specify secondary DB connection
- And info for SQL trace

Define native SQL syntax

Issue native SQL call

Retrieve result of native SQL call – in packages if needed

Summary

In this lesson, you have learnt:

- How to use ADBC to execute Native SQL statements

Review Question

Capgemini

Thank you