*ADBC (ABAP DataBase Connectivity) is not the proprietary feature of HANA*. This property is database independent. Years ago even before we heard about HANA, *ABAPer used to connect to underlying database explicitly using native SQL* and perform the needful activity.

|  |  |
| --- | --- |
| 1  2  3  4 | EXEC SQL.  <Native SQL statements to perform the requirement>  ENDEXEC. |

Why was there the need to use Native SQL?  
*The most generic reason why Native SQL was used is, the database tables were not available in SAP Data Dictionary.* In such cases, native SQL can be used.

Some salient features of Native SQL  
1. Native SQL allows us to use database-specific SQL statements in an ABAP program  
2. No Syntax check of the SQL statement is performed. If there is an issue, we come to know only at runtime.  
3. The data is transported between the database table and the ABAP program using host variables. For the sake of clarity:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | EXEC SQL.  SELECT matnr mtart bismt  INTO  :wa\_mara  FROM  mara  WHERE matnr = :p\_matnr  ENDEXEC. |

The above example does not justify the usage of native SQL, as MARA should reside at both places. Just replace MARA with something like ORA\_INV\_MGT table which is not available in SE11. The above code will not work.

Recap: salient features of Open SQL  
1. Open SQL provides a uniform syntax and semantics for all of the database systems supported by SAP. Therefore, it is called Open. Open to all Database.

2. Open SQL statements can only work for database tables that have been created/replicated in the ABAP Dictionary

If native SQL was already doing what Open SQL could not do,

then what was the need of introducing ADBC?

ADBC is definitely better than native SQL as explained below.

*ADBC is an object base API.* This API *determines where native SQL calls have been made* and supports exception handling better. Technically, *ADBC writes native SQL* which would be executed at the database layer. But, ADBC makes the process of connecting to the database and transferring the native SQL code to be executed at database layer smoother and organized. In simple terms, the object-oriented approach is used by ADBC to connect to the database and perform the needed task.

Salient feature of ADBC

1. Just like native SQL, syntax checker cannot catch issues in the code which the underlying database is expecting. We need to handle the exceptions properly (usually cx\_sql\_exception is implemented).

2. Hashed and Sorted tables are not allowed as the target. So, the standard table must be used as target.

There are 8 generic steps performed in an ADBC call

1. *Set the database connection* (CL\_SQL\_CONNECTION=>GET\_CONNECTION)  
2. *Instantiate the statement object* (CL\_SQL\_STATEMENT)  
3. *Construct the SQL using Concatenate syntax or string operatio*n.

4. *Issue Native SQL Call* (EXECUTE\_QUERY, EXECUTE\_DDL, EXECUTE\_UPDATE)  
There are three methods to execute SQL statements.  
EXECUTE\_QUERY – For Queries (SELECT statements). An instance of CL\_SQL\_RESULT\_SET is returned as the result of the query.  
EXECUTE\_DDL – For DDL (CREATE, DROP, or ALTER). No returning parameter.  
EXECUTE\_UPDATE – For DML (INSERT, UPDATE, or DELETE). Returns the number of table rows processed in ROWS\_PROCESSED.  
5. *Assign Target variable for result set* (CL\_SQL\_RESULT\_SET, methods SET\_PARAM(), SET\_PARAM\_TABLE())  
6. *Retrieve Result set* (CL\_SQL\_RESULT\_SET=>NEXT\_PACKAGE)  
7. *Close the query and release resources*(CL\_SQL\_RESULT\_SET method CLOSE())  
8. *Close database connection* (CL\_SQL\_CONNECTION; method CLOSE())

Important Classes in ADBC  
The above 8 steps help us narrow down to three important classes in ADBC.

1. CL\_SQL\_CONNECTION  
2. CL\_SQL\_STATEMENT  
3. CL\_SQL\_RESULT\_SET

Error handling is one of the important advantages of ADBC so CX\_SQL\_EXCEPTION is the fourth important class in ADBC.