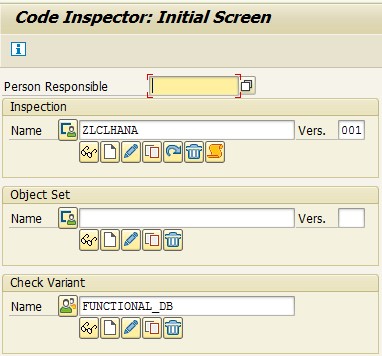
# Remediation for HANA Migration

Before migrating to HANA platform, the custom developments in SAP need to be remediated for HANA compatibility as explained in SAP Note 1912445.

Steps to be followed:

1. Identify the code where modifications are required.

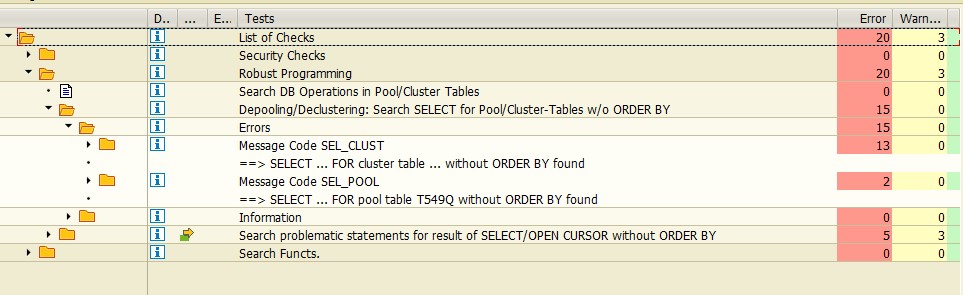
     Execute SCI for custom developments with check variants as ‘FUNCTIONAL\_DB’ and ‘FUNCTIONAL\_DB\_ADDITION’ .

[](https://blogs.sap.com/wp-content/uploads/2016/06/sci_967902.jpg)

          Results of this execution will have different message codes whose handling is explained below.

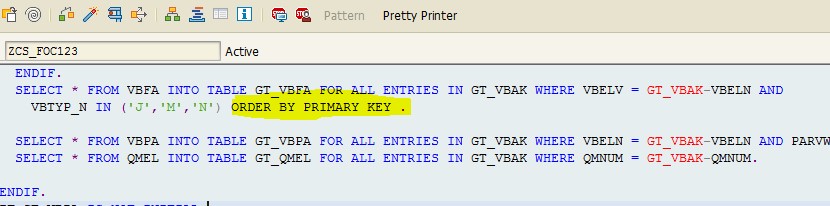
2. Modification required for each message code.

     a. Message code **SEL\_CUST.**

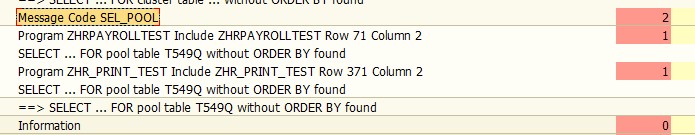
**[](https://blogs.sap.com/wp-content/uploads/2016/06/sel_cust_967903.jpg)**

**Reason** :  These error occur due to implicit sorting on cluster table.

**Solution** : Add explicit sorting to the select query.

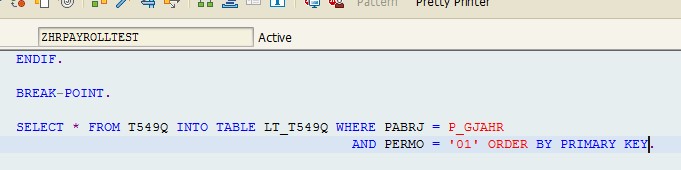
[](https://blogs.sap.com/wp-content/uploads/2016/06/sel_custprg_967916.jpg)

     b. Message code **SEL\_POOL**.

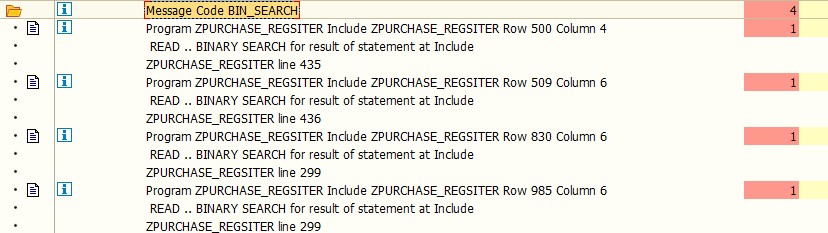
[](https://blogs.sap.com/wp-content/uploads/2016/06/sel_pool_967917.jpg)

**Reason :**These error occur due to implicit sorting on Pool Table.

**Solution:**Add explicit sorting to the query.

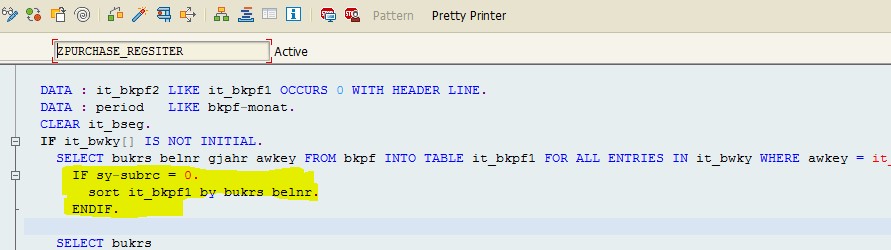
[](https://blogs.sap.com/wp-content/uploads/2016/06/pool_967918.jpg)

     c. Message code **BIN\_SEARCH**

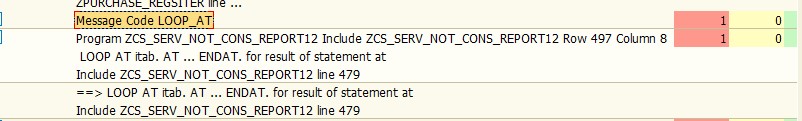
**[](https://blogs.sap.com/wp-content/uploads/2016/06/bin_967919.jpg)**

**Reason :**This error is due to reading unsorted internal table with binary search.

**Solution :**Sort internal table to be read with fields specified in binary search explicitly.

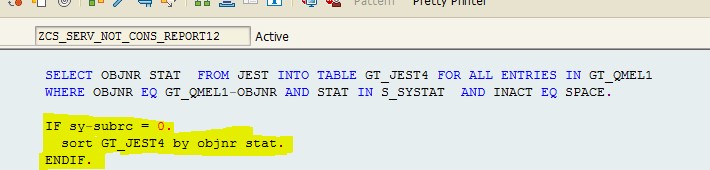
**[](https://blogs.sap.com/wp-content/uploads/2016/06/bins_967920.jpg)**

d. Message code**LOOP\_AT.**

**[](https://blogs.sap.com/wp-content/uploads/2016/06/loop_967921.jpg)**

**Reason:**This error is due to AT NEW/ENDAT operation on unsorted internal table.

**Solution :**Sort internal table on fields specified in AT NEW/ENDAT operation explicitly.

**[](https://blogs.sap.com/wp-content/uploads/2016/06/loopat_967925.jpg)**

SOH : ABAP Code Remediation & Rules to be followed

**1) Issue:**

If you use SELECT SINGLE statement with a non-unique WHERE condition, then result will be ambiguous. (Key Point: SELECT SINGLE)

**Guidelines:**

SELECT SINGLE filedname cases:  
Check if the WHERE clause has a ‘=’ or ‘EQ’ check for all the key fields of the DB table. If all the key fields are present in the WHERE clause with a ‘EQ’ or ‘=’ check then the select is unique and does not require ‘ORDER BY PRIMARY KEY’.

But if the select is not unique then result will be ambiguous and shall be corrected.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
SELECT SINGLE \* cases:  
Check if the WHERE clause has a ‘=’ or ‘EQ’ check for all the key fields of the DB table . If all the key fields are present in the WHERE clause with a ‘EQ’ or ‘=’ check then the select is unique and does not require ‘ORDER BY PRIMARY KEY’.

But if the select is not unique then result will be ambiguous and shall be corrected.

**Recommended Solution:**

**SELECT SINGLE filedname cases:**  
The correction, if required will be:

*SELECT field1 field2 …. FROM dbtable…. INTO wa*  
*WHERE condition1…. ORDER BY PRIMARY KEY.*  
*EXIT.*  
*ENDSELECT.*

**OR**

*SELECT field1 field2 …. UP TO 1 ROWS*  
*INTO CORRESPONDING FIELDS OF TABLE itab*  
*FROM dbtable WHERE condition1….*  
*ORDER BY PRIMARY KEY.*  
*READ TABLE itab into wa INDEX 1.*

*Can’t we directly select 1 record into wa.*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
**SELECT SINGLE \* cases**:  
The correction, if required will be:

*SELECT \* UP TO 1 ROWS…. FROM dbtable…. INTO wa*  
*WHERE condition1…. ORDER BY PRIMARY KEY.*  
*ENDSELECT.***OR**

*SELECT \* UP TO 1 ROWS INTO TABLE itab FROM dbtable*  
*WHERE condition1…. ORDER BY PRIMARY KEY.*  
*READ TABLE itab into wa INDEX 1. .*

*Can’t we directly select 1 record into wa.*

**2) Issue :**

Relying on implicit sorting on SELECT Statements. (Key Point: ORDER BY PRIMARY KEY)

**Guidelines:**

There is a commonly held view that SELECT statements return results implicitly sorted by the used index. Another commonly held view is that SELECT statements on pool / cluster tables will always be returned sorted by primary key.

HANA follows the SQL standard. Implicit behaviors of databases are not guaranteed by the SQL standard. The SQL standard requires an ORDER BY statement to return result sets sorted.

This issue can be found in below patterns:  
–READ …. BINARY SEARCH for result of statement  
–DELETE ADJACENT DUPLICATES for result of statement  
–READ TABLE …. INDEX ….  
–MODIFY/DELETE…INDEX for result of SELECT statement  
–DELETE … FROM / TO for result of statement  
–LOOP AT itab FROM/TO for result of statement  
–LOOP AT itab. AT ENDAT. for result of statement

**Recommended Solution:**

Always perform explicit sorting on the result sets of DB query.

**Example:**  
SELECT \* …. FROM dbtable…. INTO TABLE itab WHERE condition1… ORDER BY PRIMARY KEY.

READ TABLE itab INTO wa WITH KEY condition BINARY SEARCH OR  
DELETE ADJACENT DUPLICATES FROM itab  
COMPARING <field1>.

**3) Issue:**

Minimize the number of data transfers (Key Points: Nested SELECT Loops & SELECT ENDSELECT loop)

**Guidelines:**

This rule refers to instances where a lot of trips happen between database and application server. This hamper the program performance. Such coding instances need to be changed by using JOINs  
a) instead of nested SELECT loops,  
b) avoiding SELECT ENDSELECT loop and  
c) preference to array operations for CRUD (Create, Update, Delete) functionality.

**Recommended Solution**

1. a) Nested SELECT loops:  
   Use Joins / sub-queries instead of nested SELECT loops

b) SELECT-ENDSELECT:  
Usage of SELECT ENDSELECT inside loop results in multiple number of data transfers between DBI (DB interface) and program memory.

Instead of this, use the SELECT … INTO TABLE to fetch the records in an internal table. This results in just one transfer between DBI and program memory.

c) DELETE/UPDATE/INSERT/MODOFY in Loop:  
When using the statements INSERT, UPDATE, MODIFY and DELETE for more than one row of a database table, it is more efficient to use an array operation using an internal table followed by a single Open SQL statement, instead of calling an Open SQL statement for each row.

**4) Issue:**

Minimize the search overhead (Key Points= 1> SELECT SINGLE in LOOP, 2>SELECT without WHERE, 3>MOVE in place of MOVE-CORRESPONDING

**Guidelines:**

1. **a) SELECT SINGLE in LOOP:**  
   Do not use SELECT SINGLE in LOOP when there is a need to read single records. Instead use JOIN between tables and pass on the processing to DB level. This minimizes the number of data transfers.

**B) No use of indexes or keys:**  
Restrict usage of SELECT without any WHERE clause. When WHERE is used, minimize the search overhead by providing index field.

**c) Select With Field(S):**  
Instead of using MOVE-CORRESPONDING, always use MOVE as MOVE-CORRESPONDING has some overheads associated with it.

**d) Control Break:**  
Control level statements like AT END, AT START, AT NEW are used in a loop on internal tables to group and aggregate the data. In HANA the aggregation can be done at DB level and then the aggregated data transferred to application server. This reduces the amount of data transferred and uses the analytical engine of HANA to perform aggregations faster.

**Recommended Solution**

1. **a) SELECT SINGLE in LOOP:**  
   Use INNER JOIN and get the data into an internal table, the number of data transfers is reduces and the join is getting executed in database level, thus increasing the code performance.  
   **b) No use of indexes or keys:**  
   Restrict usage of SELECT without any WHERE clause.  
   **c) Select With Field(S):**  
   Use MOVE Instead of using MOVE-CORRESPONDING.

**5) Issue:**

Minimize the amount of data transfer

(Key Points– 1) Avoiding SELECT \*

(Key Points– 2) Using AGGREGATE Functions )

**Guidelines:**

While reading from the DB, the application should fetch only the data for the further calculation. The conditions on the business logic should be transferred as WHERE conditions or filters to reduce the amount data fetched. This can be achieved by  
a) avoiding SELECT \*,  
b) using aggregate functions like MAX, SUM etc.

**Recommended Solution**

1. **a) avoiding SELECT \* :**  
   Use SELECT <Field List> instead of SELECT \* in order to transfer just the columns you really need. This is very important in case of columnar DB’s as all the columns need to be read in order to fetch the result set.  
   **b) using aggregate functions like MAX, SUM etc :**  
   Use aggregate functions instead of transferring all the rows to application server

**6) Issue:**

ABAP Optimization for HANA:  
Changes at ABAP level –>  
These guidelines suggest optimization at ABAP level without going into the HANA modeler and can reap the benefits out of HANA.

( Key Points = 1) BAPI/FM in LOOP, 2) Table BUFFER Bypassing 3) LOGICAL DB Usage )

**Guidelines:**

1. **a) BAPI/FM in loop:**  
   Using BAPI/FM in loop results multiple access to database.

**b) Bypass table buffer:**  
Buffer must be used wherever possible. As buffer resides in application memory, it is always faster than the accessing it from underlying database layer (HANA or any other DB). Using buffer also ensures that unnecessary load is kept away from DB layer.  
Since now its in-memory database, hence table buffering is not required, implying ‘BYPASSING BUFFER’ is irrelevant now.

**c) Logical DB used in objects:**  
SAP provides many logical db for reading records from the database. Logical databases are ABAP programs that decouple Open SQL statements from application programs and have been optimized by SAP for the best database performance. A logical database has a hierarchical tree-like structure of tables where there is one root node table (the highest level) from which various levels of sub nodes are derived (linked). This is very useful in classical Dbs.  
But in case of HANA, SELECT’s are already optimized within memory and columnar database capabilities of HANA.

**Recommended Solution**

1. **a) BAPI/FM in loop:**  
   BAPI/FM calls should be done after the loop processing.
2. **b) Bypass table buffer:**  
   Buffer should never be bypassed OR ‘BYPASSING BUFFER’ is irrelevant now.
3. **c) Logical DB used in objects:**  
   Using logical databases will result in overhead and might decrease the code performance. Such instances must be changed by replacing logical database usage with SELECTs.

**7) Issue**

ABAP Optimization for HANA:  
Changes at HANA level :  
In order to gain maximum benefits out of HANA, we need to create artifacts on modeler side and it may require changing the code a lot using the stored procedures and artifacts instead of ABAP level application programming.

\*\*Key Points–  
>>Restricted use of FOR ALL ENTRIES  
>>Repeated Hits on same DB tables  
>>Nested SELECTs/LOOPs  
>>AGGREGATION usage  
>>FM for currency Conversion

**Guidelines:**

1. **a) Usage of FOR ALL ENTRIES:**  
   Usage of FOR ALL ENTRIES in most scenarios is always error prone and results multiple and unnecessary data transfers. Such instances can be replaced by a JOIN. Join logic is getting executed in the DB and only the resultant result set is transferred back to the application server.  
   **b) Repeated hits on same DB tables:**  
   Repeated hits on same DB tables within a code module (like FM, subroutine, class methods etc.) can be pushed down to DB level by creating a procedure or a view. This will require changes in flow of the ABAP program.

**c) Nested SELECTS/Loops:**  
Nested SELECT loops where an inner SELECT is executed for each outer SELECT statement, results in multiple number of data transfers.

**d) Usages of aggregation statement COLLECT in loop:**  
HANA has a very powerful OLAP engine. HANA artifacts such as analytical views, calculation views or procedures need to be created at DB level to perform the aggregations.

**e) FM used for currency conversion:**  
Instead of using FM for currency conversion, we can implement the currency conversion directly in HANA DB using an analytic view or CE function. Since this is directly executed in HANA DB, it’s performance is way more better than using FMs.

**Recommended Solution**

1. **a) Usage of FOR ALL ENTRIES:**  
   FOR ALL ENTRIES can be replaced by highly efficient statement with JOIN. Only the resultant result set is returned to application server.  
   We can also create HANA artifact such as a view or procedure for joining the tables and query the artifact in ABAP code. This also moves the pushed the code down to database.  
   **b) Repeated hits on database tables:**  
   Procedures or views can be created to restrict the repeated hits on same DB tables.  
   **c) Nested SELECTS/Loops:**  
   Nested loops must be removed by using proper JOINs.  
   **d) Usages of aggregation statement COLLECT in loop:**  
   All aggregates in ABAP code should be pushed down to database layer to make maximum use of HANA.  
   **e) FM used for currency conversion:**  
   Instead of using FM for currency conversion, we can implement the currency conversion directly in HANA DB using an analytic view or CE function.

**8) Issue**

Keep result set small ( Key Point = Check/Exit in Loops )

**Guidelines:**

Do not retrieve records from database and discard them in the application server using CHECK or EXIT in loops. Retrieve only the necessary records by proper usage of WHERE clause. With this the result set is kept as small as possible and also the amount of data transferred is also reduced.

**Recommended Solution:**

Proper use of ‘Where’ clause is mandatory to restrict the transfer of unnecessary records to application server

**9) Issue**

Access to physical table pools/cluster

**Guidelines:**

After migration to HANA, cluster and pool table are transformed into transparent tables and the relation between them is broken. In such scenario, any operation on cluster table will only be performed on the table named in SQL and not on all the tables that made up the cluster.

**Recommended Solution:**

1. a) Legacy code should be adjusted with separate SQL calls to all tables that previously were in the cluster or pool.  
   B) Restrict direct access to physical clusters/pools.  
   Refer SAP note 1785057 for further details.

**10) Issue**

Usage of native SQL in ABAP

**Guidelines:**

Native SQL used in ABAP via EXEC SQL or ADBC are specific to underline DB only.

**Recommended Solution:**

Use Open SQL as much as possible. Restrict usage of native SQL.

**11) Issue**

Usage of DB hints

**Guidelines:**

DB hints are specific to underlying DB only. These must be manually checked and adapted.

**Recommended Solution:**

Restrict the usage of hints.  
Refer the note 1622681 for supported DBSL hints on SAP HANA.

**12) Issue**

Checking of existence of indices using function modules

**Guidelines:**

The existing ABAP code might contain logic for checking existence of indices using function modules like ‘DB\_EXISTS\_INDEX ‘, ‘DD\_INDEX\_NAME‘.  
During HANA migration, secondary indices are removed as they are not required in HANA environment. These checks need to be removed to make sure that code works correctly.

**Recommended Solution:**

Restrict writing the checks for indices.  
Refer SAP note 1794297 for further details.

https://blogs.sap.com/2016/01/07/abap-on-hana-optimization-step-by-step-remediation/