Compiled By: Abu Maryam Mohammed Arif
Date: 13th August 2013

How can data from a base table be deleted, while leaving rows that were not fully imported earlier for further examination and correction?

Primary associations can be set using the Primary Flag; for example, in EIM_ACCOUNT there is an ACC_PR_POSTN column that is defaulted to “N”, but when set to “Y”, EIM will attempt to update the Primary Position in the relevant account in S_ORG_EXT.PR_POSTN_ID.

This new functionality enables users to process in one batch one account and its associated positions, and set one position explicitly as primary. It also enables users to update the primary position of an account, if the account already has a primary position.

In Siebel 2000, Enterprise Integration Manager (EIM) now supports explicitly primary fields. To populate the Primary Id field of a base table, set the corresponding EIM table column to 'Y' and populate the required columns and other columns related to the Primary field.

For example:

1. In order to populate the Primary Personal Address Id for a contact, populate all necessary fields to import Contacts and `their addresses thru EIM_CONTACT in the base tables S_CONTACT and S_ADDR_PER. Set the CON_PR_PER_ADDR field to 'Y' in EIM_CONTACT for the record that need to be set as primary.
When multiple addresses are associated with a contact, the address information along with contact data will be populated by EIM in the S_CONTACT and S_ADDR_PER base tables. EIM then defines as primary address for S_CONTACT, the address that has CON_PR_PER_ADDR ='Y' set in EIM_CONTACT.

2. In order to populate the Primary account address for an account, populate all necessary fields to import accounts and their addresses in the base table S_ORG_EXT and S_ADDR_ORG. Set the ACC_PR_ADDR field to 'Y' in EIM_ACCOUNT while importing accounts and their addresses using the base table S_ORG_EXT for the record that need to be set as primary.
When multiple addresses are associated with an account, the address information along with account data will be populated by EIM in the S_ORG_EXT and S_ADDR_ORG base tables. EIM then defines as primary address for S_ ORG_EXT, the address that has ACC_PR_ADDR ='Y' set in EIM_ACCOUNT.
For Siebel v7, please refer to Siebel Enterprise Integration Manager Administration Guide at the following location: Siebel Interface Tables > Interface Table and Column Mappings > Explicit Primary Mappings.

Handling Data Dependencies in EIM using IFB Files

In a typical data migration application using EIM, the IFB file contains a large number of sub-processes, which are called from the main process (SHELL). Often while running the EIM process, there are situations, when we would like to run only a sub set of these processes. Such situations normally arise in testing environments, when we prefer running only those processes for which testing needs to be performed.

For example consider a section of a typical IFB File, which includes the following processes.

```
[Import Position]
TYPE  = IMPORT
BATCH = 11100000-11100005
TABLE = EIM_POSITION
ONLY BASE TABLES = S_PARTY, S_POSTN

[Import Position Organization]
TYPE  = IMPORT
BATCH = 11200000-11200005
TABLE = EIM_POSITION
ONLY BASE TABLES = S_PARTY, S_ACCNT_POSTN
USE INDEX HINTS = TRUE
INSERT ROWS = S_PARTY, FALSE
UPDATE ROWS = S_PARTY, FALSE

;*****************************************************************************
; IMPORT
;*****************************************************************************

[Import Everything]
TYPE = SHELL
INCLUDE = "Import Organization"
INCLUDE = "Import Organization Address"
INCLUDE = "Import Org-Address Relationship"
INCLUDE = "Import Position"
INCLUDE = "Import Position Organization"
INCLUDE = "Employee Position Rel"
```

Suppose the logic for migrating Position Organization relationship has undergone some change. Hence the “Import Position Organization” Process needs to be tested. So, in the IFB file we comment out all other processes and keep only **INCLUDE = "Import Position Organization"** uncommented. We delete the records in the S_ACCNT_POSTN table, which stores the Position Organization Relationship, and then run the EIM process. This works fine.

However, suppose the logic for migrating Position data has undergone some change, which is why the “Import Position” Process needs to be tested. But in this case, if we comment out all other processes and run the EIM only for **“Import Position”**, after deleting records from S_PARTY and S_POSTN, then the Position Organization relationship data records would become orphans. Similarly, records in all other tables, which have a foreign key relationship with S_POSTN, would become orphan records. This is because every time we delete a Base table (either manually or through EIM), and then run the EIM process to migrate records into it, SIEBEL generates new ROW_IDs for each of these records. Hence the foreign key reference that other tables have with this record would be lost.

Hence the thumb rule that should be followed is:
When testing data migration for a Master table (which has foreign key relationship with one or more child tables), run the EIM process not only to import the Master table records, but also to import records to all tables that have foreign key relationship with the Parent table.
In our example, therefore if we need to test the migration of Position Master records into S_POSTN, we would need to run the EIM process for **“Import Position”**, "**Import Position Organization**" and "**Employee Position Rel**" to ensure that foreign key references are not lost.

Using Error and Trace Flags in EIM Error handling:

The settings for the TRACE FLAGS, SQL TRACE FLAGS and ERROR FLAGS can be specified either from the GUI or from the Command Prompt when starting an EIM task.
If the EIM process is being run from the GUI, then the flags can be set in the COMPONENT REQUEST PARAMETERS list in the COMPONENT REQUESTS View of the Enterprise Operations Screen.
If the EIM Process is being run from the command prompt, then the flags can be set within the command for running the EIM task, as shown in the sample below:

```
srvrmgr /g siebelserver /e AspireEnt753 /s SIEBELSERVER /u sadmin /p sadmin
/c "run task for component eim with
config=<<filename.ifb>>,Errorflags=1,Sqlflags=8,Traceflags=3
```

Note: Task parameters can override component-level event logging to create additional information in the log. In Siebel, parameters can be set, from highest to lowest, at Enterprise, Server, Component Definition, Named Subsystem, and Sever Component and Task level. A parameter set at a lower level can override the same parameter set at a higher level.

Error Flags:
By setting the Error Flags to appropriate levels, a detailed explanation of rows that were not successfully processed can be obtained.
Setting the task parameter "Error Flags" = 1 can be used to write a detailed description of the cause of each error to the task log during processing.

SQL Trace Flags:
The SQL Trace flags parameter is used for logging SQL statements that make up the EIM task.
Setting the SQL Trace Flags parameter to 8 creates a log of the summary SQL statements that make up the EIM task. Since it dramatically impedes EIM performance and created a large log file, hence this option is recommended only for test environments. The lower values for SQL Debug Flags (1, 2, and 4) are used for logging at the ODBC level.

Trace Flags:
Trace flags contain logs of various EIM processing steps. Siebel 7.x onwards**, Event Logging** needs to be set for the EIM component, in order to activate Trace Flags.
For setting**Event Logging,** the following sequence needs to be followed:

1. Click the Server Administration screen tab.
2. From the Show drop-down list, select Component.
3. Select Enterprise Integration Manager as the component.
4. Click Component Event Configuration.
5. Perform a query and enter the Log Level values for the following:

EVENT TYPE	LOG LEVEL VALUE
SQL Tracing	4
SQL Summary	4
Task Configuration	4
Component Tracing	3

Trace Flags are bit-based. Available Trace Flags include 1, 2, 4, 8, and 32. To activate multiple trace flags, the Trace Flags parameter has to be set to the sum of individual trace flag numbers. For example, to log trace flags 2 and 4, the Trace Flags parameter needs to be set to 6.

Setting Trace Flags to 1:
Setting the Trace Flags parameter to 1 creates a step-oriented log of the task. This can be used to determine the amount of time EIM spends on each step of the EIM task, or for each interface table processed.

Setting Trace Flags to 2:
Setting the Trace Flags parameter to 2 creates a file log that traces all substitutions of user parameters.

Setting Trace Flags to 4:
Setting the Trace Flags parameter to 4 creates a file log that traces all user-key overrides.

Setting Trace Flags to 8:
Setting the Trace Flags parameter to 8 creates a file log that traces all Interface Mapping warnings.

Setting Trace Flags to 32:
Setting the Trace Flags parameter to 32 creates a file log that traces all file attachment status. The trace file contains four labels, three of which are used to trace file attachment processes as described.

LABEL	DESCRIPTION
Attachment Imported	Indicates whether the file attachment was encoded, compressed, and copied to the Siebel file server with the new name
Attachment (Old) Deleted	This label applies only to updates and indicates whether an existing file was replaced and deleted
Attachment Not Found	Indicates that the file attachment cannot be found in the input directory

Note: Activating flags will have a direct negative effect on performance since, since a lot of information will be recorded in the log file. Typically, activating flags should only be done when testing EIM processes, and should be avoided in a production environment unless absolutely necessary.

How to Resolve Siebel EIM Errors?

When migrating data using the EIM, the process can fail due to several reasons. In certain cases, it is found that the EIM process was successful for migration into some of the Base tables, but failed for some other base tables. In other cases, records get imported into the base tables, but some of the columns fail migration. This section discusses the various causes of EIM failure, how they can be tracked and finally resolved.

After EIM process is complete, the following SQL query has to be run, to check for possible errors:

```
SELECT if_row_batch_num, if_row_stat, if_row_stat_num, count(*)
FROM <<EIM Table Name>>
GROUP BY if_row_batch_num, if_row_stat, if_row_stat_num;
```

Common Errors and Resolution:

The following chart lists the possible values of IF_ROW_STAT, what they indicate and some possible resolutions.

IF_ROW_STAT	Indicates	Resolution
IN_PROGRESS	Integration Manager sets IF_ROW_STAT to this initial value for all rows in the batch. If rows still have this status value after Integration Manager exits, a failure occurred that aborted processing for this table.	One possible reason may be that an extension column in the Base table that the EIM table maps to do not exist. Another reason may be that there is insufficient space in the Siebel Server for the generation of LOG files.
SQL_ERROR	A SQL error occurred during an attempt to import this row. This error occurs for rows processed when transaction logging is TRUE.	In this case, the SQL statement that failed execution is logged in the EIM log file. This statement can be analyzed to deduce the specific error.
IMPORT_REJECTED	A user-specified filter query failed for this row.	The query specified using the FILTER QUERY parameter needs to be analyzed and corrected. The query expression should be a self-contained WHERE clause expression (without the WHERE keyword) and should use only unqualified column names from the interface table or literal values (such as NAME IS NOT NULL).
FOREIGN_KEY	A required foreign key column in the target table could not be resolved.	One reason could be that all the columns in the EIM table that map to the foreign key in the base table have not been populated. Another reason could be that the EIM table columns mapping to the foreign key column, resolve into a ROW_ID that is non-existent in the Foreign Key table.
PICKLIST_VALUES	A required picklist value in the target table could not be resolved. This error occurs for NULL or invalid bounded picklist values.	The error could be resolved by ensuring that a picklist exists for the column that is LOV bounded. The value being imported for this column corresponds to one of the values in the picklist.
REQUIRED_COLS	One or more required columns for the target table were NULL.	Before running the EIM, it should be ensured that the required columns in the base table(s) are

		being populated. Even if these columns are not specified in the EIM table, they could be defaulted to specific values using the DEFAULT or FIXED COLUMN parameters in the IFB file.
DUP_RECORD_EXISTS	The row exactly matches rows that already exist in the destination tables. This error can also occur when the same record (same user key) exists with the same EIM batch number with a lower ROW_ID. In this case, MIN (ROW_ID) is the record processed, and the other records with the same user key are marked as DUP_RECORD_EXISTS.	It should be ensured that in the EIM table, records with the same batch number have distinct unique key combinations.
PARTIALLY_IMPORTED	The row did not fail for the target table (although it may have been a duplicate), but did fail during processing of a secondary base table. This status is set after the import has completed.	In case, a parent-child relationship exists between the target table and the secondary base table, the user keys mapping to the Foreign Key column in the Child table referencing the Parent table must be populated in the EIM table in exactly the same way in which the user key columns resolving to ROW_ID of the parent table are populated.

DUP_RECORD_IN_EIM_TBL	Indicates that there are 2 or more interface records having the same user key values.	To avoid this situation, analyze the input records before beginning the EIM task. If you find duplicate records, you can either combine them into one record, or specify a different batch number for the duplicate record. [4]
NON_UNIQE_UKEYS	This error occurs if there is more than one USERKEY, say <<Base table name_U1>>, <<Base table name_U2>>, etc. defined on a base table. The particular EIM table being used to populate the base table uses the second USERKEY but the user has populated EIM columns corresponding to the columns of the first USERKEY	It is necessary to identify the particular USERKEY that the EIM table uses and populate columns mapping to that USERKEY uniquely.[5]
AMBIGUOUS	This error is usually encountered when there are some columns in the Base table, other than the unique key columns, which also need to be uniquely populated for each record. This is because of unique index database constraint defined on these columns. This error is also encountered when attempting to run DELETE or MERGE processes on records in the base table with	The workaround is to identify all the column combinations in the base table, which participate in the different unique index constraints defined on it, and populate each combination uniquely. [6]

	same unique keys but different conflict_ids.	

Creating Siebel ROW_ID through EIM:

The unique identifier associated with every record in Siebel Enterprise databases is known as a Row ID. The column in which this value is found is ROW_ID and it is present on every table. The ROW_ID is unique for an entity. For example, the ROW_IDs for the same person in S_PARTY, S_CONTACT, and S_CONTACT_X are the same because they each refer to the same person.

Row IDs are used extensively throughout Siebel Enterprises to access specific records. Although users access records by a User Primary Key (such as Opportunity Name), it is more efficient for the Siebel Enterprise to store and access related data via the Row ID.

The Row ID is a base-36 sequence number generated using a confidential, proprietary algorithm that ensures no duplication, thus protecting the referential integrity of the database. The ROW_ID column is VARCHAR(15), which may contain one or two non-numeric symbols (plus signs or hyphens, or both).
The format of the ROW_ID is one of the following:

CP-NS	Records created by the user interface
CP+NP+NS	Records created by Interface Manager (EIM)
CP-NP-NS	Records created by EIM (Starting in Siebel versions 6.2 and higher, and Siebel version 7)

where:

CP	= Corporate Prefix, up to 2 alphanumeric characters
NP	= Next Prefix, up to 6 alphanumeric characters
NS	= Next Suffix, up to 7 alphanumeric characters

The maximum length of the ROW_ID is 15 alphanumeric characters. The corporate prefix will always be unique for any database (main or local). The server maintains its original value, and mobile databases created against that particular server database are always assigned a new, unique value.

The Siebel ROW_ID is a combination of the S_SEQUENCE_S and information from S_SSA_ID table.
All connected users share the same prefix, which is obtained from the table S_SSA_ID on the server database. Remote Users are assigned a unique prefix when they are db-extracted. This value is also stored in the S_SSA_ID table on the local database.

Suffix is generated using an internal algorithm handled by the source code. When a new record is created through the user interface, the Siebel application reads the value of the current NS column from S_SSA_ID table and increments this value by a value more than 1 – for performance reasons, generally 50. The client caches these fifty potential ROW_IDs for future inserts. A new record entered from the user interface may result in many inserts to the underlying tables, depending on the business components used. When the client disconnects, cached ROW_IDs are lost.

The combination of the prefix and suffix generates a unique row_id. Since Siebel does not expose the algorithm behind generation of row ids, the ROW_ID generation is internal to the EIM process. Therefore Siebel provides user keys to map to a unique record in the base table.
During EIM, the EIM table columns, which map to the user keys of the base table, are populated with values so as to map to a unique record. If that record is to be updated by EIM later, the same user key

values have to be populated in the corresponding EIM table columns. Otherwise, it will not resolve into the correct base table record.
For example, PERSON_ID and BU_ID constitute the user keys for S_CONTACT. The corresponding EIM_CONTACT columns are CON_PERSON_UID and CON_BU. When a new record is imported in S_CONTACT using EIM_CONTACT, the CON_PERSON_UID and CON_BU should uniquely identify a Contact record. Now, for updating some fields for this Contact using EIM_CONTACT, CON_PERSON_UID and CON_BU have to be populated with the same set of values that were used during the initial load in order to map to the same Contact record in the base table.

Similarly while loading a child table, which references the row id of the master table, the EIM table columns that map to the foreign-key column of the child table must be populated in the same way the user keys of the master table were populated.

For example, S_CONTACT_XM is a child table of S_CONTACT. The column PAR_ROW_ID of S_CONTACT_XM references the ROW_ID of S_CONTACT. S_CONTACT_XM is loaded by EIM_CON_DTL. The columns of EIM_CON_DTL that need to be populated to generate the PAR_ROW_ID of S_CONTACT_XM are CON_BU, CON_PERSON_UID and CON_PRIV_FLG. These EIM columns should be populated with the same set of values that were used to populate its parent S_CONTACT record through EIM_CONTACT. Note that though PRIV_FLG is not a part of the user key of S_CONTACT, but it is a required column and is part of the foreign-key mapping of its child table.

How to Populate User Keys and Required Columns using Siebel EIM?

It is mandatory to load the required columns of the base tables. If one or more required fields are missed, then the EIM process will fail to load the base table and generate the REQUIRED_COLS error status. To identify which are the required columns for a base table, the following query can be used:

```
SELECT TBL.NAME, COL.NAME
FROM   S_COLUMN COL, S_TABLE TBL
WHERE  COL.TBL_ID=TBL.ROW_ID
AND    COL.REQUIRED='Y'
AND    TBL.NAME = <<give Base Table Name here>>;
```

Similarly to determine the user keys for a base table, the following query can be used:

```
SELECT distinct  TBL.NAME,
    COL.NAME
FROM     S_COLUMN COL,
      S_TABLE TBL
WHERE    COL.TBL_ID=TBL.ROW_ID
AND      TBL.NAME = <<give Base Table Name here>>
AND      COL.USR_KEY_SEQUENCE IS NOT NULL;
```

If the above query does not return any row, then that implies that the corresponding base table does not have any user key.

Is it Possible to Update the User Key using EIM?

Using EIM task user cannot normally update User key attributes of the base table. However, there are some special interface tables EIM_ORG_EXT_UK and EIM_PROD_INT_UK that can be used to update user key columns for some key tables S_ORG_EXT and S_PROD_INT.
Updating user key columns using EIM task with these special tables will not update denormalized columns. For example, updating S_ORG_EXT.LOC will not update S_ACCNT_POSTN.ACCT_LOC AND S_ORG_BU.ORG_LOC.

Handling Siebel Base Tables with no user keys during EIM:

Some base tables like Note tables do not have user keys. Since user keys play a vital role in migration of data using EIM, the EIM behavior is significantly different in case of Base Tables without user keys. For such tables, although IMPORT and EXPORT processes work, but Merge process does not work. The DELETE process, when using DELETE MATCHES and DELETE ALL ROWS parameters, works fine, but does not work with DELETE EXACT parameter. This is because the EIM process cannot identify the rows to delete, in the absence of user keys.

However, Siebel 7.5.x versions have made deletion of data from Notes tables, using a special EIM, EIM_NOTE_DEL. The EIM_NOTE_DEL interface table should only be used to delete notes from the S_NOTE_* tables. To use it, users need to populate ROW_IDs of records to be deleted into corresponding columns in the interface table and then use DELETE EXACT to delete them.
For further details, please refer the Other Issues Relevant to Notes Migration section of the NotesDataMigrationGuide

Choosing the correct interface table for Siebel EIM process:

When migrating data to or from Siebel Base tables, one of the main tasks is to identify the Interface table that would be used in the EIM process. There are cases in which a base table can be loaded from more than one EIM Interface tables. In such cases, the choice of interface table could optimize the EIM performance significantly. ROW_ID and IF_ROW_BATCH_NUM are mandatory columns for EIM processing.

Driving Factors while choosing EIM table:

Here is a list of all factors that should be kept in mind while choosing the EIM Table.
Base table- Identify all the Base tables that need to be populated. This is driven by requirement.
EIM table- Identify all the EIM tables that contain mappings to the Base Table. The query that will help to determine all the EIM tables that populate a particular Base table is:

```
SELECT DISTINCT   T2.NAME BASE,
                  T1.NAME EIM
FROM              S_TABLE T1,
                  S_TABLE T2,
                  S_EIM_TBL_MAP MAP
WHERE             T2.NAME=<<give Base Table Name here>>
AND               T1.ROW_ID=MAP.IF_TBL_ID
AND               T2.ROW_ID=MAP.DEST_TBL_ID
AND               T1.INACTIVE_FLG='N'
```

```
AND                T2.INACTIVE_FLG='N'
AND                MAP.INACTIVE_FLG='N'
ORDER BY           T1.NAME;
```

Target Table- In the IFB file, while specifying the base tables that a particular EIM table will load, it is mandatory to specify the target base table for the EIM table. If the target table is not mentioned, EIM will generate an error. The following query determines the target base table for an EIM table:

```
SELECT DISTINCT    T1.NAME SOURCE,
                   T2.NAME TARGET
FROM               S_TABLE T1,
                   S_TABLE T2
WHERE              T1.TARGET_TBL_ID = T2.ROW_ID
AND                T1.NAME =<<give EIM Table Name here>>
ORDER BY           T1.NAME;
```

EIM Table Selection to load maximum base tables- When more than one related Base tables need to be populated, instead of selecting different EIM tables for each Base table, the optimal choice would be an EIM table that would contain mapping to all the related Base tables that need to be populated. The following query determines the Base tables that can be loaded from a particular EIM table:

```
SELECT      T1.NAME EIM,
      T2.NAME BASE
FROM        S_TABLE T1,
      S_TABLE T2,
      S_EIM_TBL_MAP MAP
WHERE       T1.ROW_ID=MAP.IF_TBL_ID
AND         T2.ROW_ID=MAP.DEST_TBL_ID
AND         T1.INACTIVE_FLG='N'
AND         T2.INACTIVE_FLG='N'
AND         MAP.INACTIVE_FLG='N'
AND         T1.NAME =<<give EIM Table Name here>>
ORDER BY    T1.NAME;
```

For example, consider a situation where both the tables S_PARTY_PER and S_POSTN_CON need to be populated for storing relationships between an Employee and a position. Since employee related data are usually migrated using EIM_EMPLOYEE, this interface table would seem like an obvious choice. But EIM_EMPLOYEE does not have mapping with S_POSTN_CON. So in this case we could

1. Load S_PARTY_PER using EIM_EMPLOYEE and S_POSTN_CON using EIM_CONTACT.
2. Alternatively, we could use only EIM_CONTACT, since this interface table loads both S_POSTN_CON as well as S_PARTY_PER.

The choice ii would be optimal since our purpose is getting solved using a single EIM table.

EIM Table Selection to load maximum columns- Among these Interface tables, select the Interface table that contains mappings to the maximum number of Base table columns. The ideal choice would be the interface table that would have mapping to all the base table columns that have been identified for population.

For example, the base table S_CONTACT can be loaded using both EIM_EMPLOYEE as well as EIM_CONTACT. But if our requirement indicates that a specific column, say, PR_HELD_POSTN_ID of S_CONTACT needs to be populated then we would have to choose EIM_EMPLOYEE because EIM_CONTACT does not have mapping to PR_HELD_POSTN_ID column of S_CONTACT.

However, if our requirement were to populate PR_POSTN_ID of S_CONTACT, then we would have to choose EIM_CONTACT and not EIM_EMPLOYEE because EIM_EMPLOYEE does not have mapping to PR_POSTN_ID column of S_CONTACT.

Siebel EIM Import Order of Entities:

The order in which legacy data is imported is critical to make sure that relationships between dependent data elements are established correctly. Siebel interface tables do not map one-to-one with Siebel target database tables. To make sure that the necessary data is present to establish relationships between data entities, import data in the following recommended order:

1. Administrative
    2. Business Unit
    3. Positions
    4. Accounts
    5. Contacts
    6. Employees
    7. Products
    8. Opportunities
    9. Personal Accounts
    10. Quotes
    11. Documents
    12. Forecasts
    13. Fulfillment
    14. Marketing Campaigns
    15. CPG Promotion Management
    16. CPG Product Movement
    17. Service Requests
    18. Product Defects
    19. Activities and Appointments
    20. Notes
    21. File Attachments
2. This import order reflects most import processes. In some cases, the import order may vary slightly depending on the requirements. A sample recommended order of importing could be found in the Siebel provided default ifb file.

NOTE: An example of administrative data would be a List of Values for Currency or Zip Code.

Component request parameters for running an EIM process:

The component request parameters for running an EIM process are
1. Batch Number: This parameter is used for identifying the rows to be processed by the EIM process. Value 0 specifies the EIM process to use the batch number as specified in the EIM configuration file.
2. Configuration file: This parameter names the configuration file to be used by the EIM process. This file should be placed in the ADMIN subdirectory of the Siebel server.
3. Error Flags, SQL Trace Flags and Trace Flags: These parameters are used to specify whether logging of Error, SQL Statements and other EIM operations would be done in detail or in brief. A detailed discussion on the settings of these flags, and their effect on EIM performance can be found in the Error Handling Section of this guide.

Running the Siebel EIM Process:

The EIM Process consists of executing the following sequence of steps:
1.Populate the columns in the Interface/ EIM table that are required to load the Siebel base
2.Write/Edit the EIM Configuration (IFB) File to define the EIM Process to perform
3.Submit the EIM as a Siebel Server Batch component task
Running the EIM: The EIM Process can be run once the EIM Tables have been prepared, i.e. suitably populated, and the EIM configuration file has been accordingly written. The EIM process can be initiated by running a server task for the Enterprise Integration Manager component. On each pass, EIM processes one interface table and performs a particular action (IMPORT, DELETE, EXPORT, MERGE) on all rows in that table for that batch.

There are two methods for running an EIM process:

1.Running an EIM Process Using the Graphical User Interface (GUI)
2.Running an EIM Process Using the Command-Line Interface
Note: Transaction will be logged during EIM to synchronize mobile clients. During initial data loads the Transaction Logging parameter should be set to FALSE in the System Preferences settings under Application Administration, in order reduce transaction activity to the Siebel docking tables. Once the initial loads are complete it can be set back to TRUE if there are Siebel Mobile Web Clients.

Mystery of Siebel EIM Merge Process Explained:

A merge process deletes one or more existing rows from the base table and ensures that intersecting table rows are adjusted to refer to the remaining rows. Some process parameters like TYPE, BATCH and TABLE are similar to that in case of the above mentioned processes. Upon completion of the merge process, the some row survives and the remaining rows are deleted.
For surviving records, the IF_ROW_MERGE_ID column in the EIM table is set to NULL. For rows to be merged (and subsequently deleted), this column is set to the value of the base table ROW_ID where this row will be merged. All child and intersection table rows that previously pointed to ROW_IDs of the deleted record now point to the merged (survivor) record.
For deleted rows, EIM sets T_MERGED_ROW_ID to the ROW_ID of the row that was merged into (the surviving row) and T_DELETED_ROW_ID to the ROW_ID of the deleted base table row.

UPDATE ROWS: This parameter is necessarily to be set to TRUE in case of Merge process. It specifies whether the foreign key (or keys), which references the merged rows, in the named table need to be adjusted. Valid values are TRUE (the default) and FALSE.

```
[Merge Accounts]
TYPE = MERGE
BATCH = 1
TABLE = EIM_ACCOUNT
UPDATE ROWS = TRUE
```

To set the value to FALSE, the table name on which the parameter setting will act should be mentioned.

```
[Merge Accounts]
TYPE = MERGE
BATCH = 1
```

```
TABLE = EIM_ACCOUNT
UPDATE ROWS = S_ADDR_PER, FALSE
```

EIM can only be used to merge rows from target base tables and not secondary base tables. For example, EIM_ASSET can only be used to merge rows in target base table S_ASSET and not secondary base table S_ASSET_CON.

Siebel EIM Delete Process Parameter Reference:

Some process parameters like TYPE, BATCH, and TABLE are similar to an Import process.

DELETE ROWS: For Delete process the TYPE parameter should be defined as TYPE = DELETE. As a result the DELETE ROWS parameter will be automatically set to TRUE. This parameter can prevent deletions from one table while allowing them in others. For example, the following parameter prevents deletion of rows from the S_ADDR_PER table:
DELETE ROWS = S_ADDR_PER, FALSE

UPDATE ROWS: This parameter specifies if foreign key references can be updated. The default is UPDATE ROWS = TRUE, which affects all tables.
The UPDATE ROWS parameter also prevents updates in one table while allowing them in others. If this parameter is set to FALSE, EIM does not update rows in the specified base table. If you need to specify multiple tables, use one UPDATE ROWS statement for each table.

```
UPDATE ROWS = S_CONTACT, FALSE
UPDATE ROWS = S_ADDR_PER, FALSE
```

DELETE ALL ROWS: It will delete all rows in the named base table including any seed data if it is set to TRUE. The default value is FALSE.

```
[Delete Accounts]
TYPE = DELETE
BATCH = 200
TABLE = EIM_ACCOUNT
DELETE ALL ROWS = TRUE
```

CLEAR INTERFACE TABLE: Valid values are TRUE and FALSE. If it is set to TRUE, the existing rows in the interface table for a given batch number will be deleted. The default setting of this parameter depends on the DELETE EXACT parameter setting. The CLEAR INTERFACE TABLE default is TRUE if DELETE EXACT is set to FALSE and vice versa.

CASCADE DELETE ONLY: (Default = FALSE). Set this parameter to TRUE to delete child records with nullable foreign keys when the parent record is deleted. If FALSE, then when EIM deletes a parent record, it sets the foreign keys of the child records to NULL.
Sometimes cascade delete takes place during delete process even if it is not mentioned specifically in the .ifb file. When a foreign key column that references the deleted record is a required one, the record with the foreign key is deleted. Otherwise, the foreign key column is cleared.
EIM deletion of a parent row causes cascade deletion of child rows only if the foreign key column in the child table is a mandatory column. Otherwise a cascade clear is performed.

DELETE EXACT: This parameter specifies the base table rows to delete by using user key values specified in the interface table. It is recommended to use the DELETE EXACT parameter to delete non-target base tables that contain user keys only. By default, DELETE EXACT = FALSE. If DELETE EXACT is set to TRUE, the ONLY BASE TABLES parameter in conjunction with this parameter needs to be

mentioned to identify the base tables. It should be ensured that any columns that are not part of the user key are NULL in the interface table.

```
TYPE = DELETE
BATCH NUMBER = 100
TABLE = EIM_ACCOUNT
ONLY BASE TABLES = S_ORG_PROD
DELETE EXACT=TRUE
```

DELETE MATCHES: This parameter is used for filtering or more precisely to choose selective base table rows to be deleted. The value is in two parts: the Siebel interface table name and the filter expression that goes against the target base table. An example would be:
DELETE MATCHES = S_ORG_EXT, (LAST_UPD > '2004-07-15' AND LAST_UPD < '2004-07-17')
By default, DELETE MATCHES expressions are not used.

DELETESKIPPRIMARY: This is a special parameter that has been included Siebel 7.x onwards, and is set to either TRUE or FALSE, with TRUE being the default value. The default value results in EIM not performing a cascade update on the primary child column. Setting the parameter to FALSE indicates that EIM will perform the cascade update. Setting DeleteSkipPrimary to FALSE may affect performance, as additional queries will be generated by EIM.

For example,

```
TYPE = DELETE
BATCH = 20
TABLE = EIM_FN_ASSET1
ONLY BASE TABLES = S_ASSET_POSTN
DELETE EXACT = TRUE
```

In the above process, EIM will delete records from S_ASSET_POSTN but the PR_POSTN_ID column in S_ASSET will not be adjusted to reflect the changed relationship. That is, PR_POSTN_ID may still contain the ROW_ID of the old S_POSTN record rather than 'No Match Row Id' as is desired.
The PR_POSTN_ID field in S_ASSET can be reset to reflect the change, by adding the following line in the process:
DELETESKIPPRIMARY = FALSE

Siebel EIM Export Parameter Process Reference:

Some process parameters like TYPE, BATCH, and TABLE are similar to an Import process.
EXPORT ALL ROWS: It implies that all rows in the target base table and data from the related child tables in a specific batch section are to be exported. Valid values are TRUE and FALSE (the default). When it is set to TRUE, the EXPORT MATCHES parameter is ignored, as it is selective export process.

```
[Export Accounts]
TYPE = EXPORT
BATCH = 2
TABLE = EIM_ACCOUNT
EXPORT ALL ROWS = TRUE
```

EXPORT MATCHES: This is used for filtering the base table rows to be exported. The value is in two parts: the Siebel interface table name and the filter expression that goes against the target base table.

```
[Export Accounts]
TYPE = EXPORT
BATCH = 2
TABLE = EIM_ACCOUNT
EXPORT MATCHES = S_ORG_EXT, (LAST_UPD > '2003-01-01')
```

Reference to Siebel EIM Import Parameters:

TYPE: This parameter specifies the type of process being defined (IMPORT, EXPORT, DELETE, MERGE, or SHELL). A shell process uses the INCLUDE statement to invoke a sequence of processes in a single run.

BATCH: Specifies a required batch number for this process. This number is predefined as the user sets this number at the time of EIM table load. It is used to identify the set of rows to load from the interface tables for this specific process. It corresponds to the value in the interface table column IF_ROW_BATCH_NUM and must be a positive integer between 0 and 999999999999999 (up to 15 digits, no commas). To specify multiple batches, use a range or list of batch numbers. BATCH=100-120
To list batches, use the comma-delimited format as shown in this example: BATCH=100,103,104
It is recommended to use batch ranges (BATCH = x-y). This allows the user to run with smaller batch sizes and avoid the startup overhead on each batch. The maximum number of batches that one can run in an EIM process is 1,000.

TABLE: Specifies the name of an EIM table used in this process. From performance aspect the selection of EIM table is vital. The things that should be kept in mind while selecting the EIM table are discussed earlier.

```
TYPE = IMPORT
BATCH = 10000-10100
TABLE = EIM_ACCOUNT
```

ONLY BASE TABLE: Specifies and restricts selected base tables for the import process. Use commas to separate table names. Target table for interface tables must be included. This parameter is used to improve performance when loading some specific base tables. For example:
ONLY BASE TABLES = S_PARTY,S_ORG_EXT,S_ACCNT_POSTN,S_ORG_BU

IGNORE BASE TABLES: Specifies base tables to be ignored by the import process. Use commas to separate table names. Target tables for interface tables cannot be ignored. The default is to not ignore any base tables. Use this parameter to improve performance when loading all but a few tables.

ONLY BASE COLUMNS: Specifies and restricts base table columns for the import process. Use commas to separate column names, which can be qualified with base table names. Include all user key columns and required columns. Use this parameter to improve performance when updating many rows but few columns. The default is to process all interface columns mapped to the base table.
ONLY BASE COLUMNS= S_ORG_EXT.NAME, S_ORG_EXT.LOC, S_ORG_EXT.BU_ID

IGNORE BASE COLUMNS: Specifies base table columns to be ignored by the import process. This is also written with the same manner as ONLY BASE COLUMN. Required columns and user key columns cannot be ignored. Use this parameter to improve performance when loading all but a few columns. The default is to not ignore any interface columns.

IGNORE BASE COLUMNS = S_PROD_INT_BU.PR_FULFL_INVLOC_ID,S_PROD_INT_BU.PR_PROD_LN_ID

DEFAULT COLUMN: Specifies a default value for an interface table column. The syntax is COLUMN NAME, VALUE, as in the following example:
DEFAULT COLUMN = CURCY_CD, "USD"
The given value will be used only if the column is NULL in the interface table.

FIXED COLUMN: Specifies the value for a column from the interface table. The syntax is the same as for DEFAULT COLUMN. The given value will be loaded into the Siebel base table, overriding the value in the interface table column.
FIXED COLUMN = PARTY_TYPE_CD, "Organization"

INSERT ROWS: Specifying INSERT ROWS as FALSE indicates that rows from EIM table are not to be inserted into the specified Siebel base table. The default is INSERT ROWS = TRUE. To change this for all tables, use this syntax:
INSERT ROWS = FALSE

For a Siebel base table, the setting is applied when data is imported from any interface table as in the following example:
INSERT ROWS = S_ORG_BU, FALSE
INSERT ROWS = S_ACCNT_POSTN, FALSE
If the named table is an interface table, as in the example below, the setting applies to all Siebel base tables imported from this interface table.
INSERT ROWS = EIM_ACCOUNT, FALSE

UPDATE ROWS: This parameter specifies if foreign key references can be updated. The default value is UPDATE ROWS = TRUE which affects all tables. To change only for specific table(s), specify the table name as follows:
UPDATE ROWS = S_CONTACT, FALSE
If it is required to set the parameter as FALSE for most tables, and TRUE for only a few, use this method:
UPDATE ROWS = FALSE
UPDATE ROWS = S_CONTACT, TRUE
UPDATE ROWS = S_ADD_ORG, TRUE
By default, when importing information, EIM performs both inserts and updates based on the content of the batch set.
If the named table is an interface table, as in the example below, the setting applies to all Siebel base tables imported from this interface table.
UPDATE ROWS = EIM_ACCOUNT, FALSE
By default, INSERT ROWS and UPDATE ROWS are TRUE

Note: If neither INSERT ROWS nor UPDATE ROWS are set to FALSE, EIM has to perform additional processing (via SQL statements) to determine whether to update or insert. Basically when importing data via EIM, EIM will first look up user key columns in a base table. If it finds the matching user keys, EIM will continue to compare all non-user key columns in order to determine whether it needs to update the record or reject it as duplicate.1

MISC SQL: This parameter is used to set specific explicit or implicit primaries. “Explicit” is when it is required to set specific values as primaries. “Implicit” is when any of a group of values is acceptable. For example, you are importing one account with nine addresses. If any of the addresses is acceptable as being the primary, then set primary to implicit. EIM then selects one of the addresses as primary. If a specific address should be the primary, then set primary to explicit and indicate the primary account by setting its flag column.
For example, set EIM_ACCOUNT.ACC_PR_ADDR to “Y” for explicit Primary. For example,
MISC SQL = EXPR_S_ORG_EXT_PR_ADDR_ID

MISC SQL is intended for initial data loading only (with DOCKING TRANSACTIONS = FALSE). Sometimes the intention is to apply explicit primary when specified and implicit primary otherwise. In that case, we mention the following in the ifb file:
MISC SQL = EXPR_S_ORG_EXT_PR_ADDR_ID, IMPR_ S_ORG_EXT_PR_ADDR_ID

USE INDEX HINTS: The default value for this parameter is TRUE. If it is set to TRUE, EIM generates hints during processing which helps in achieving performance gain. It is recommended for EIM processes to be tested with both TRUE and FALSE settings in order to determine which provides the better performance for each of the respective EIM job.

How is IFB File Structured?

An IFB file consists of two main sections namely:
Header Section: The .ifb file begins with a header section to specify global parameters that apply to all process sections.
Process Section: Following the header section, there must be at least one process section with its associated parameters. Some process section parameters are generic for all EIM processes 'Type', 'Process Name', 'Batch Number', 'EIM Table Selection' etc.

Header Section:

The first nonblank, uncommented line of the header section of any .ifb file must contain the exact information:

[Siebel Interface Manager]
The other parameters that are common to all process types (Import, Export, Delete & Merge) in the header section are:

PROCESS: Identifies the specific process to run during this invocation of EIM. The named process must be defined in the process section of this file.

CONNECT: The ODBC source name for connecting to the database server.

USERNAME: Specifies the database logon name for this process. This parameter is inherited for the EIM component from the Gateway server, so it should already be set. However, you can specify this in the .ifb file if you are running EIM from the Siebel application (not the command line) and if you have not already set this value in the EIM Server Component parameters.

PASSWORD: Specifies the database password for this process. This parameter is inherited for the EIM component from the Gateway server, so it should already be set. However, you can specify this in the .ifb file if you are running EIM from the Siebel application (not the command line) and if you have not already set this value in the EIM Server Component parameters.

Process Section

Unlike header section parameters, the process section parameters are not generic for all the processes (Import, Export, Delete & Merge). But some parameters are common for all.
The first nonblank, uncommented line of each process section is a bracketed string ([]) that specifies the name of the process. This is the name used in the PROCESS parameter in the header section.

What is IFB file?

An IFB file is nothing but a EIM configuration file which is an ASCII or Unicode (Binary) text file of extension type .ifb. IFB stands for Interface Batch. It resides in the Admin directory of Siebel Server and allows the database administrator to define the type of EIM process like Import, Export, Merge and Delete to be performed.
Why IFB is needed?
An IFB file contains specifications for batch jobs for the data element to be operated based on the process type mentioned in it.

Update user key Column for Product Entity, through EIM:

Normally, User key update is not possible through EIM except for few entities, of which Product is one. Here is a useful information to update one of the user key columns for Product (Product NAME for instance) through EIM.

Solution:

We can update Product name through EIM_PROD_INT_UK. EIM_PROD_INT_UK will enable us update a Product's name from "Product 1" to "Product 2".

1. For a Product NAME, BU_ID, VENDR_OU_ID are part of user keys. However, while updating Product Name for instance, through EIM_PROD_INT_UK, we use another column called INTEGRATION_ID, which will be used as secondary user key to identify the record uniquely.
2. Normally the Integration_Id column will be null. Updating the Product user key will include a step which will include a step to update the INTEGRATION_ID column with a unique value that helps in identifying the record uniquely and enable UK updates. We can prepare the record for update by updating the integration_id with the record's row_id and use it for the EIM updation.
3. We need to populate the Integration id column value from base table to the EIM_PROD_INT_UK. INTEGRATION_ID along with other attributes required for the eim process to update the record in the Base table. The new Name, for e.g, should be populated in the EIM_PROD_INT_UK.PROD_NAME column so that it will update the existing value in S_PROD_INT.NAME with the new value.

Here is a sample IFB file for such update which will change the Product Name both in S_PROD_INT and S_PROD_INT_BU

```
[Siebel Interface Manager]
                PROCESS              =UPDATE_PROD_INT

[UPDATE_PROD_INT]
                TYPE      = IMPORT
                TABLE    = EIM_PROD_INT_UK
                BATCH  = 55508

DEFAULT COLUMN = ACTIVE_FLG, "Y"
DEFAULT COLUMN = PROD_BU, "Default Organization"
                ONLY BASE TABLES = S_PROD_INT, S_PROD_INT_BU
                INSERT ROWS = S_PROD_INT, FALSE
                INSERT ROWS = S_PROD_INT_BU, FALSE
```

EIM Tuning – Reference:

What is EIM?

EIM is a data loading tool that reads data from EIM tables (staging table) and Inserts/Updates/Deletes data in/from Siebel base tables.

Why EIM tuning is important

- Data conversion process is often underestimated and becomes bottleneck reaching go-live date.
- The slow initial data conversion could result into large amount of delta updates to catch-up.
- The incremental data loads – if not tuned and tested properly could overrun the nightly batch processing window. Running large batch process with large number of on-line users can hurt on-line user response time.

Myths

- The database tools (Sql*Loader, BCD etc) can be used to load base tables.
- EIM is very slow and we should write our own programs to load Siebel base tables.
- Since EIM can be used to load data in Siebel database, no tuning efforts are required.
- No Database tuning and close DBA support is required since EIM is being used to load data.
- Siebel Server with number of CPUs and large amount of memory is required to run EIM jobs.

Facts

- EIM can load millions of rows per hour if tuned properly.
- EIM satisfies referential integrity requirements for Siebel base tables.
- The data quality greatly effects EIM performance.
- EIM is more Database server intensive than it is Application Server intensive.

What can be tuned

- The initial data loading process (IDL)
- The incremental data loading process (delta)
- Database tuning
- Disk storage tuning
- The EIM process

Tuning Process

- Ensure that legacy/feeder systems are providing the required data.
- Ensure data quality – Most common issue is duplicate data.
- Ensure correct EIM mapping. Engage Siebel ES for EIM mapping review.
- Tune Pre-EIM processing. i.e. getting data from feeder system, data cleansing, ftp/copy and loading data in EIM tables.
- If tuned properly – EIM processing represents not more then 25 – 50% of total data conversion time.
- Don’t forget reconciliation for drop outs.

Critical Resources

- Business Analyst – To verify EIM mappings.
- Database Administrator.
- EIM resource.
- Testing Team.

- Programs from legacy/feeder systems to generate input files for EIM.
- Technical Architect.
- If the amount of data is too high – Siebel Expert Services.

Tuning EIM = Tuning I/O

- By reducing I/O
- By balancing I/O
- By fast read and write
- By reducing operation time

Process Design

- Keep number of rows in EIM tables to a reasonable level (i.e. only those rows that are being currently processed).
- Number of rows in EIM table will depend upon total number of rows that needs to be processed.
- For large EIM jobs (more then 10 million for single EIM table), consider using EIM shadow tables.
- Review and thoroughly test pre-processing.

Reducing I/O in pre-processing

- Use direct insert operation for pre-processing and to load data in EIM tables.
- For initial data loads; consider turning off archive logging (Oracle: Archive logs, DB2: turn off table logging).
- Consider using a different database for pre-processing (Other then loading EIM tables).
- Never do record by record processing between two databases.

Tuning IFB

- Configuration file is read by EIM to determine following
- Process to perform
- Import, Export, Delete, Merge, Update.
- Interface and Base tables used and affected
- Other parameters used for EIM processing
- Default file 'Default.IFB" contains useful example
- Default file is available in Siebel_server\admin directory
- Only Base Tables
- Only Base Columns
- Update Rows = False (When the job is insert only)
- Insert Rows = False (When the job is update only)
- Separate Insert and Update operations
- Use SQLPROFILE parameter to list most expensive SQL statements.

Profiling

The SQLPROFILE lists most expensive SQL statement along with run time and summary. SQLPROFILE is set by specifying output file name in header section of IFB file.

TO Set SQLPROFILE

SQLPROFILE = "C:\temp\EIM_SQL.txt"

SQL Trace

Set SQL trace level to 8 while running EIM Tasks during the tuning phase.

I/O Balancing

- EIM will generate tremendous amount of I/O. The EIM throughput is directly related to I/O capabilities of your database server and storage array.

- Database layout should distribute I/O on multiple disks/spindles to reduce I/O bottleneck.
- Install EIM tables and EIM table's indexes on a separate I/O channel from Siebel base tables and Siebel Base table indexes.
- For initial data load on turn off Archive logs to reduce I/O.
- For initial data load make sure that Docking Transaction for Mobile users is set to FALSE.

Basic Tuning

- Find out the most optimum batch size.
- Generally speaking – for Insert operation start with 10,000 rows per batch. For update, start with 5000 rows per batch and for delete operations, start with 2000 rows per batch.
- The most optimum batch size will differ from table to table.
- Try large batch size for smaller tables (tables with fewer columns) and smaller batch size for larger tables (tables with large number of columns).

Run EIM tasks in parallel.

- Try to load all independent tables in parallel. These are independent tables and should not cause any locking issues.
- Run multiple streams of EIM against same EIM table and same base tables. The number of streams will depends upon RDBMS, database server and disk storage array.
- Some databases may run into locking issues with multiple streams on same EIM table depending upon their locking mechanism.

Database Tuning

- Involving DBA from early stage of data conversion process.
- Share your EIM know how with DBA.
- Plan for several iteration of testing and tuning.
- Have at least one dry run in your project plan.
- Start thinking about automating data loads from the get go.
- Prepare a thorough check list for loading sequence and pre-processing required for EIM data loads.
- For IDL the database needs to be tuned in a different manner then OLTP operations.
- For IDL – Tune the database for large I/O operations.
- Create Siebel base tables and indexes with sufficient PCT_FREE/Fill Factors.

General database tuning

- Involve an experienced database administrator
- Allocate large memory/buffers to the database
- Distribute I/O as much as possible
- Tune OS parameters for Large I/O operations

Tuning Indexes

- Drop all indexes on target Siebel base table except for P1 and U% indexes.
- Run sample EIM job with SQL trace 8 and SQLPROFILE.
- Remember 80 – 20 rule.
- Check execution plan of most expensive SQL statements.
- Create indexes to tune most expensive SQL statement.

Oracle

- Large redo log files
- For IDL – No Archive logs. Take periodic backups.
- Create redo log files on fast devices.

- Rollback segment, Redo logs, Archive redo logs and temp tablespace are going to be very busy during IDL.
- Avoid frequent redo log switches
- Focus on top 5 waiting events
- Set ini_trans to large value (10 – 20) for parallel EIM jobs.
- Set Free Lists to large value for dictionary managed tablespace.
- Row level locking – run EIM tasks in parallel. Depending upon amount of data.
- Use SQL*Loader direct path to load data in EIM tables.
- Cache frequently used tables and indexes (S_LST_OF_ VAL)

DB2

- Use Update Statistics parameter in IFB file till you get optimal set of statistics then turn it off.
- Use larger page size for EIM and large base tables.
- Ensure proper distribution of tablespace containers.
- Use separate large buffer pools for EIM and base tables for IDL.

SQL Server

- Make sure that P1 index (IF_ROW_BATCH_NUM, ROW_ID) on EIM table is a clustered index.
- Test EIM jobs with and without index hints. This can be done by adding "Index Hints = FALSE" parameter in IFB file.
- Avoid fragmentation by allocating appropriate fill factors.
- Periodically de-fragment tables and indexes.
- Use large batch size for insert operations.
- For initial EIM data loads, benchmark your run time with degree of parallelism set to 1.
- If possible – sort your data on base table user keys before loading data in EIM tables.
- Benchmark EIM run time with Index Hints and without index hints. Index hints can be turned off by using "Index Hints = False" in IFB file.
- Create separate file groups for TEMPDB, Data and Log and put them on separate stripe/controller.
- Put P1 index on separate stripe/controller.
- Parallel data load for EIM tables using bcp.

EIM Data Loading: Oracle function - Bulk Collect:

When loading data through EIM process, we normally use Pl/SQL procedures to fetch and process huge volume of records and then load the data to EIM Interface tables. There are scenarios when the normal way of data loading into EIM tables take long time due to the bulk volume of records. Typical usage of SQL queries result in performance bottleneck issues due to the frequent context switches between the SQL and Pl/SQL engine

Implementation :

We have an OOB solution for this by using the BULK COLLECT function provided by Oracle which provide a significant performance boost when dealing with large sets of data. This feature enables to reduce the Turn Around time involved in loading and processing records in EIM table.

Syntax :

FETCH<cursor_name> *BULK COLLECT INTO* <collection_name> *LIMIT*

LIMIT: Specify the # of records to be collected during single fetch.

PS : This function is available in Oracle and works only for Oracle 10g version.

EIM Tuning to improve performance:

Through EIM process we load bulk amount of data from an external system to Siebel base tables. The performance of these EIM jobs has to be maintained at optimum levels in order to reduce the load on the Siebel application and database servers. The following tips will help you in improving the EIM performance. Do a round of validation in development environment before running the EIM jobs on Production systems

- Compute statistics (DBSTATS) and analyze index on the tables periodically. Frequent insert or delete operations on interface tables can cause fragmentation in the table.
- Analyze and figure out the optimal records that can be accommodated in a single batch as per the database capacity
- Avoid hard coding of column values using DEFAULT/FIXED COLUMN IFB parameters
- Set the logging level to minimal values and switch off Siebel transaction logging as per the load type (Initial / Mobile client)
- Run independent EIM jobs in parallel.
- Limit tables and columns to be processed using ONLY BASE TABLES/COLUMNS configuration parameters
- Segregate INSERT and UPDATE records in different batches which in turn will improve the Turn Around Time of the EIM job
- Enable database Optimizer hints and set the USING SYNONYMS parameter as per the account data load type
- During bulk load Drop Index/Constraints on the table which are not mandatory and recreate them after the data load
- After the completion EIM Process, purge the data in the EIM Interface tables
- As a last option set SQLPROFILE parameter in IFB to analyze the long running query and tweak it to optimal values
  Example SQLPROFILE = c:\EIM\eimsql.sql *Note : Though no two Siebel projects can be exactly similar so every project has to be tuned according to the business needs and Infrastructure available*

EIM Data Cleansing : Oracle function – Regular Expression:

When loading data into Siebel through EIM, we do perform Data cleansing/massaging as per our business requirement. The cleansed data from Source is then loaded into the target EIM Interface tables

Scenario:

Convert multiple spaces in address related columns to a single space for all records dynamically. Please refer the example cited below,

SOURCE	TARGET
82 DEVONSHIRE ST	82 DEVONSHIRE ST
2245 OBSERVATORY PL	2245 OBSERVATORY PL
2245 OBSERVATORY PL NW	2245 OBSERVATORY PL NW

Oracle built in functions like replace, decode and translate was not yielding the desired result due to search complexity. Writing a custom procedure / function to do this address cleansing will be tedious and complex process, as it has to cleanse all records dynamically.

Implementation:

We have an OOB solution for this by using the regular expression (REGEXP_REPLACE) function provided by Oracle from 10g version which improves the ability to search and manipulate character data.

Syntax:

```
SELECT ADDR AS SOURCE, REGEXP_REPLACE(ADDR,'( ){2,}', ' ') AS TARGET FROM
SIEBEL.S_ADDR_PER WHERE PER_ID IN ('1-23O0-1625','1-G8FU-133')
```

Table Lock in Oracle:

The challenge was to find out which OS user, through which DB login and from which machine the table has been locked and the type of lock that has been applied.

Situation: Application performance issue; Table Lock

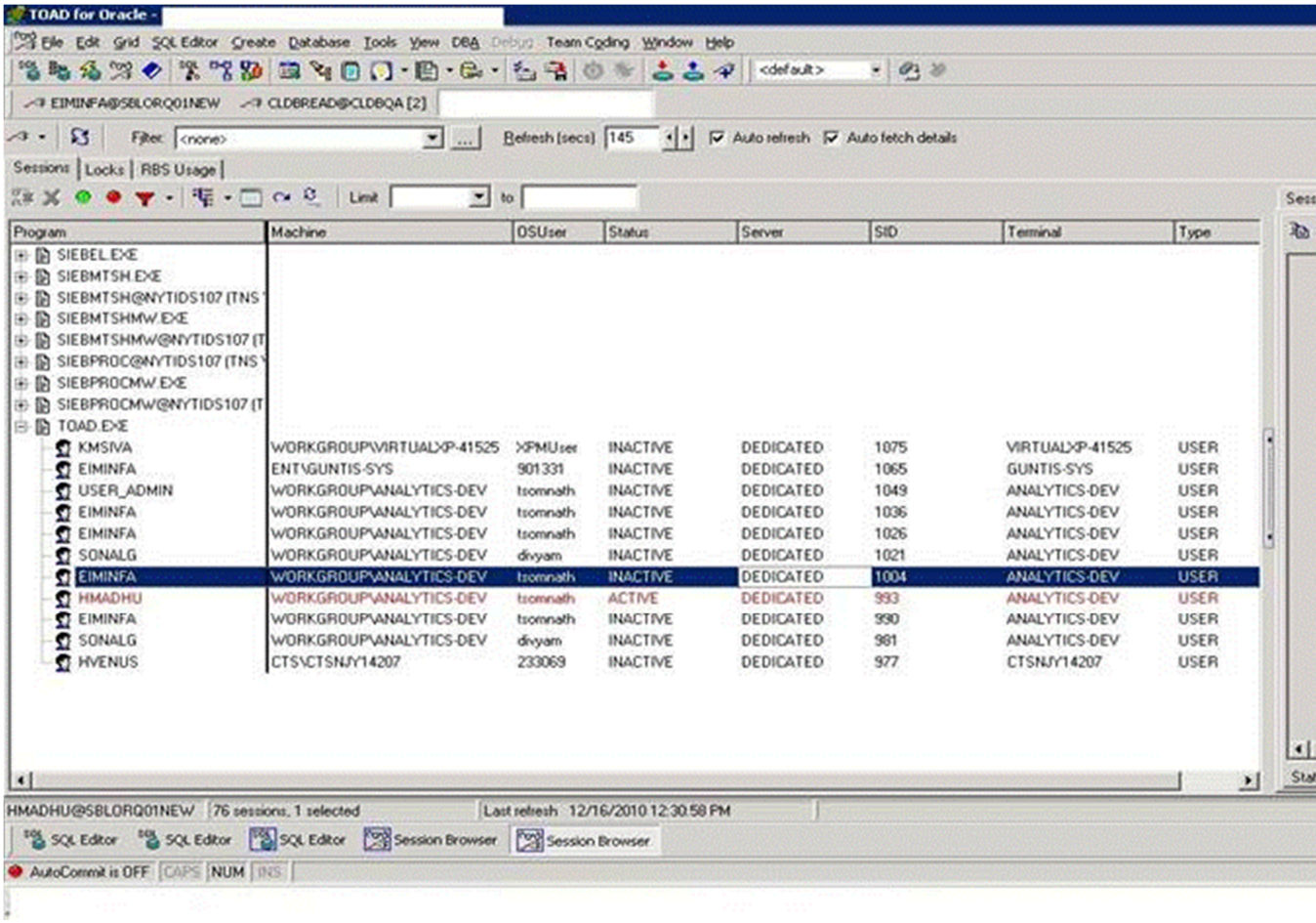
Solution: Two options to find out,

1. Lock query: Sample query and sample output.

```
select a.object_id, b.object_name, a.session_id, a.oracle_username,
a.os_user_name, a.process, a.locked_mode from   sys.V_$LOCKED_OBJECT a,
all_objects b where a.object_id = b.object_id;
```

OBJECT_ID	OBJECT_NAME	SESSION_ID	ORACLE_USERNAME	OS_USER_NAME	PROCESS	LOCKED_MODE
270047	S_CONTACT	1004	SIEBEL	xxxxx	10848:2036	3

2. TOAD > Session Browser: This will display all the sessions and the query fired by the user and the lock mode on them. PFA sample screenshot on the same



PS : Pre-requisite for doing the above is to have read only access on V$session and corresponding tables and it works only for Oracle.

EIM Column Extension in Siebel 8.x:

When extending a custom column in Siebel 8.X which have **case and accent insensitive (CIAI)** enabled, running EIM mapping wizard creates corresponding mapping in EIM table. After which running an EIM job will cause the job to fail with the following error
"SBL-EIM-00205: NULL target table for relation!" followed by "Failed to load the application dictionary"

Root Cause :
This issue is caused by EIM Mapping wizard which creates mapping for the custom column and **also for CI Columns** on the EIM table. EIM mappings should not be exist in the EIM table, as CI columns are not supported by EIM

Solution :
This can be fixed by adopting the following steps

- Delete/inactivate the CI columns under EIM table
- Delete diccache.dat (dictionary) file from SiebSrvr\Bin directory so that fresh file will be created following the deletion of CI column mappings.