
Item Conversion in a Multiple Inventory Organization Environment

Sugumar Subramani

EXOR Technologies - ERP Division of Sapient Corporation

Introduction

The task of item conversion needs to be done before the data of other modules in Oracle manufacturing are converted. It is critical to get this task done in the shortest time possible with the greatest accuracy. There are several options available to do an item conversion, they are; manual conversion, use of the Oracle open interface, and programmatic conversion. Choosing from these options depend on factors such as volume and complexity of data, and the availability of the Oracle open interface functionality. Manual conversion may be used if the number of items is under 1000. The Oracle open interface may be used if the number of items is under 5000. Programmatic conversion is the best choice if the number of items exceeds 5000 and if the complexity of data is high.

What are the steps involved in doing item conversion? How should a programmatic conversion be approached? How modular should the code be? How much time will a typical programmatic conversion take? How many iterations will a successful item conversion involve? This paper intends to help the implementation project team answer these questions.

Steps Involved in a Programmatic Conversion

The following steps are involved in the transfer of data from legacy systems.

- 1) Extract data from Legacy and other systems into ASCII flat files.
- 2) Transfer data to the hardware platform. For example, flat files are written to a specified UNIX directory.
- 3) Create custom temporary tables, and database objects such as, stored functions and procedures, in the custom schema. This will ensure that these objects are not wiped out during any future upgrades.
- 4) Load flat files into these temporary tables using SQL *Loader. A direct path load may be considered if the volume of data is high and no manipulation is necessary. A direct path load is faster because the data is not written into rollback segments. The downside to this technique is that in case of any errors the data can not be rolled back.

- 5) Create copy rows (templates) for copying standard attributes for the major types of items through the application.
- 6) Modify data from temporary tables, using PL/SQL scripts and load the Oracle tables.
- 7) Use separate scripts to update attributes if the logic depends on information, which does not come from flat files.
- 8) Create a UNIX Shell script, which will sequentially process all the required code.
- 9) Generate error reports, which can be analyzed later.

Functional Aspects of Item Conversion

The logical approach to converting items in a multiple inventory organization environment is to programmatically mimic the creation of an item through the Oracle Applications. When an item is created through the application, it is first defined in the item master and is enabled in the other organizations according to business requirements. This process is mimicked programmatically by inserting one record into the MTL_SYSTEM_ITEMS table for the master organization and one record each for the other organizations where the item is enabled. The combination of inventory_item_id and the organization_id forms the primary key for the MTL_SYSTEM_ITEMS table. Due to the primary key constraint inventory_item_id is the same for an item over all organizations, the organization_id is different for each organization where the item is enabled. In version 10.7 of the applications, the MTL_SYSTEM_ITEMS table has a total of 224 columns. Of these only 44 are mandatory "NOT NULL" columns. From a practical conversion standpoint, all 224 columns do not get populated. Only about 100 columns are populated. These columns are generally grouped under two categories; item_type specific and item specific.

Examples of item_type specific attributes are purchasing_enabled_flag, mrp_planning_code, etc. All the items may be grouped under two categories; manufactured and purchased. Item_type specific attributes depend on whether the item is purchased or manufactured. One approach is to set up two rows for these two types of items through the application. These may be referred to as "COPY ROWS" to distinguish

them from item templates. In the script which is found in appendix B of this paper, segment1 of the MTL_SYSTEM_ITEMS table, which holds all or a segment of the part number, has a value of "MAKE" or "BUY" for these copy rows. Attributes from these rows are copied programmatically for all items which fall under these two categories.

The use of copy rows is a simple yet powerful approach to doing item conversions. All item_type specific attributes can be set for certain categories of items in these copy rows. These attributes can be copied for all the new items that are inserted into the table. Often times, item attributes need to be changed to test certain functionality of modules in the Oracle applications. These changes can be made to the copy rows so the conversion programs need not be modified. By merely running the conversion program again, new item attributes can be set for these items. Copy rows can be set up for any other category of items. For example, all the items can be grouped into different item types such as "assembly", "sub-assembly", "raw-material", "component", or "supplies".

Examples of item specific attributes are; inventory_item_id, segment1, primary_unit_of_measure, etc. Inventory_item_id is assigned from a sequence called MTL_SYSTEM_ITEMS_S, segment1 is directly mapped from the Legacy data and usually contains the part number, primary_unit_of_measure is selected from an Oracle table called MTL_UNITS_OF_MEASURE where the primary unit of measure is defined. All these attribute are mapped in the PL/SQL script.

Technical Aspects of Items Conversion

PL/SQL offers powerful features to modularize the conversion scripts. Employing each of the following techniques will help the conversion scripts become more efficient.

1) PL/SQL records are an advanced feature in PL/SQL. It is a composite data structure and eliminates the need to declare many individual variables. Aggregate assignment of all column values of a row in the table can be done to a PL/SQL record. It resides in memory for the duration of the script and eliminates the need to fetch values from a row of a database table. This results in considerable saving of Disc I/O time. The use of a PL/SQL record makes the code cleaner and modular, which lends itself to easier modification and maintenance. In the item conversion script, which is in appendix B of this paper, copy rows are loaded into two PL/SQL records. These records are defined as MTL_SYSTEM_ITEMS%ROWTYPE. Depending on

the type of item which is processed, aggregate assignment is made from either the MAKE or BUY copy rows to the copy_row_rec. Also another type of PL/SQL record used in the script, is a cursor record of the temporary table. This record contains Legacy data. These records are then passed as parameters into functions and procedures.

2) Functions can greatly simplify the task of item conversion. By writing functions, data mapping can be hidden into black boxes and the time and effort spent in researching the data mapping issues need not be repeated. There are two types of functions that can be written; stored functions that are stored in the database, and local functions that are native to the script. The following criteria may be used in determining which are to be created as stored functions and which are to be created as local functions. If the function can be reused in other conversion scripts or by other programmers, then they need to be created as stored functions. If the functions are used to process a specific task within a script, then they need to be created as local functions. Examples of stored functions are *get_user_id*, *get_user_name*, *get_item_id*, *get_organization_id*, etc. Examples of local functions are *get_primary_uom*, *get_item_type*, *get_planner_code*, etc. All the item specific attributes can be mapped by writing functions. In the conversion script found in appendix B, five local functions and three stored functions are used. Each of these functions has data mapping built into them based upon weeks of functional research.

3) Procedures are yet another powerful PL/SQL construct, which can be used to modularize a conversion script. Tasks like modifying data and inserting values into a table can be done with the use of procedures. Procedures have the advantage of containing all the code necessary to process a certain task and are less CPU time intensive. Procedures lend themselves to easier modification, debugging and maintenance. In the scripts, which are in appendix A of this paper, *set_attributes* is used to assign all the item specific attributes based on different logic. Two PL/SQL records are passed into this procedure as parameters; the cursor record from the temporary table which contains legacy data for an item, and the copy_row_rec record. The PL/SQL record is assigned values from the cursor record. The *insert_items* procedure populates the MTL_SYSTEM_ITEMS table and is called twice in the script; once to insert the item into the item master and once to insert the item into another "enabled" organization.

4) While running the item conversion, many scripts need to be run by the programmer. Several temporary tables also need to be created, these tables must to be

loaded, and several PL/SQL scripts need to be run to insert items and update item attributes. All these scripts can be arranged in a UNIX shell script in the order in which they need to be run. An example of a shell script is found in appendix A of this paper. Scripts are run in a certain order which perform the following tasks; create all temporary tables, load these temporary tables, insert rows into Oracle tables, and update item attributes.

The time it takes to insert 60,000 rows into the MTL_SYSTEM_ITEMS table can vary anywhere from 30 minutes to several hours. There are many factors, which affect the time it takes to convert items, most of these are outside of the control of the programmer. However, the most important factor within the programmer's control is the quality of the conversion scripts. The use of functions, procedures, and PL/SQL records greatly speed up the process of conversion; which the programmer can employ in their scripts to make the conversion process more efficient.

As each organization, which embarks on an Oracle Applications implementation, is different in the way they do business; it is reasonable to expect anywhere from 15 to 30 iterations of item conversion. An adequate number of iterations is necessary to test and implement all functionality which the Oracle Applications provide.

Conclusions

Creation of "copy rows" is essential to proper mapping and functional testing of the item conversion. It is also important to modularize the conversion scripts even as they are developed. As the conversion schedule progresses, the programmer will find that he/she will be called upon to make several modifications to the logic within conversion. The more flexible the PL/SQL code, the easier it will be to modify, debug and maintain. So writing modular code is the key to making the item conversion script efficient and effective.

About the Author

Sugumar Subramani is an application consultant at EXOR Technologies, Inc. Sugumar has over 12 years of experience in the manufacturing industry and has been working with Oracle manufacturing since 1996.

APPENDIX A

```
#!/bin/ksh

echo 'do you want to continue Y/N'
read OPTION
case $OPTION in
# Option N or n - Quit
N|n)clear
exit
;;
#
*)
echo 'Executing Items'
sec/sec

echo " "
date

#####
The following scripts create temporary tables.
#####

echo " "
echo "Creating temporary table for loading Legacy
data"
sqlplus sec/sec @c_item_t.sql

echo " "
echo "Creating temporary table for loading NAFTA
description"
sqlplus sec/sec @c_nafta_des.sql

#####
The following scripts load data into the temporary
tables.
#####

echo " "
echo "Loading temporary tables with items"
sqlldr sec/sec control = items.ctl direct = true

echo " "
echo "Loading NAFTA_DESCRIPTION Table "
sqlldr sec/sec control = naftades.ctl

#####
#The following scripts run the sql scripts.
#####

echo " "
echo "Inserting items in the item master"
sqlplus sec/sec @ins_msi.sql

echo " "
echo "Loading item revisions in the revision table"
sqlplus @upd_rev2.sql
```

```
#####  
The following scripts run the sql update scripts.  
#####
```

```
echo " "  
echo "Update flags for purchased parts in item table"  
sqlplus sec/sec @upd_flags.sql
```

```
echo " "  
echo "Update flags for purchased parts in item table  
from the flat file"  
sqlplus sec/sec @upd_msi_des.sql
```

```
echo " "  
echo "Item conversion process complete !!!"
```

```
echo " "  
date
```

```
exit;  
esac
```

APPENDIX B

```
--*****  
--Program: ins_items.sql  
--Written by Sugumar Subramani  
--Exor Technologies Inc.  
--  
--This program processes  
--Legacy data from a temporary  
--table and inserts items into  
--mtl_system_items table  
--*****
```

```
connect sec/sec  
set serveroutput on
```

```
DECLARE
```

```
v_commit_count NUMBER:=0;  
v_po_ck_item NUMBER;  
  
v_error_msg VARCHAR2(80);  
v_step_cnt VARCHAR2(2);  
v_master_org_item_type VARCHAR2(30);  
v_org1_ins_req_flag VARCHAR2(1);  
  
v_error_occurred BOOLEAN:= FALSE;  
v_item_found_in_msi BOOLEAN:= FALSE;  
v_skip_insert1 BOOLEAN:= FALSE;
```

```
--*****  
-- The following make_cur  
-- cursor returns a single row  
-- that holds all the attributes  
-- for a manufactured part
```

```
--*****
```

```
CURSOR make_cur  
RETURN mtl_system_items%ROWTYPE  
IS  
SELECT * FROM mtl_system_items  
WHERE segment1 = 'MAKE01';
```

```
--*****
```

```
-- The following buy_cur  
-- cursor returns a single row  
-- that holds all the attributes  
-- for a purchased part
```

```
--*****
```

```
CURSOR buy_cur  
RETURN mtl_system_items%ROWTYPE  
IS  
SELECT * FROM mtl_system_items  
WHERE segment1 = 'BUY01';
```

```
--*****
```

```
-- The following PL/SQL record  
-- is used to assign values for an  
-- item from either the make_rec  
-- or the buy_rec cursor records  
-- depending on whether item is  
-- a make part or a buy part
```

```
--*****
```

```
copy_row_rec mtl_system_items%ROWTYPE;
```

```
CURSOR item_cur IS  
SELECT  
rowid  
,partno  
,factory  
,des  
,unit_of_measure_I  
,convert_fact  
,m_planner  
,unit_of_measure_S  
,m_commodity_class  
,m_eco_number  
,m_eco_date  
,c_leadtm_fixP  
,leadtm_varP  
,c_leadtm_fixM  
,leadtm_varM  
,c_leadtm_fixT  
,leadtm_varT  
,m_master_sched_code  
,m_family_code  
,c_goods_in_tm  
,c_safety_tm  
,c_cov_tm_max
```

```

,c_cov_tm_min
,c_cancel_tol
,c_resched_tol
,c_reminder_tol
,c_ord_proc_tm
,over_receipt_tol
,under_receipt_tol
,unit_of_packaging
,abc_category
,procure_method
,preferred_method
,c_fact_add_req
,phantom_code
,safety_stock
,m_item_type_name
,cost_period_start_date
,cost_period_end_date
,standard_price
,routing_code
,min_order_qty
,error_flag
,error_msg
FROM item6_temp item
ORDER BY partno,
      preferred_method,
      factory;

```

```

--*****
-- This procedure inserts
-- errors into the item6_temp table
-- that can be reviewed later for
-- data errors or mapping errors
--*****

```

```

PROCEDURE INSERT_ERROR
(item_rec_in_rowid IN CHAR,
 error_msg_in IN CHAR)
IS
v_error_msg VARCHAR2(80);
BEGIN --Error Procedure--
v_error_msg := substr(sqlerrm,1,80);
UPDATE item6_temp item
SET error_msg =
v_error_msg||'-'||v_step_cnt,
error_flag = 'T'
WHERE item.rowid = item_rec_in_rowid;
END; --Error Procedure--

```

```

--*****
-- This local function maps
-- the uom_code from Legacy
-- data to UOM code that is in
-- the mtl_units_of_measure table
--*****

```

```

FUNCTION GET_UOM_CODE

```

```

(item_rec_in IN item_cur%ROWTYPE)
RETURN VARCHAR2
IS
return_value
VARCHAR2(3):= NULL;
BEGIN
SELECT
DECODE(item_rec_in.unit_of_measure_S,
'GM','G',
'PD','EA',
'SF','FT2',
'XX','EA',
'PK','PKG',
item_rec_in.unit_of_measure_S)
INTO return_value
FROM DUAL;
--dbms_output.put_line
('uom_code--'||return_value);
RETURN return_value;
EXCEPTION
WHEN OTHERS THEN
RETURN NULL;
END;

```

```

--*****
-- This local function returns
-- the unit_of_measure from the
-- mtl_units_of_measure that is
-- used as primary_uom
--*****

```

```

FUNCTION GET_PRIMARY_UOM
(item_rec_in IN item_cur%ROWTYPE)
RETURN VARCHAR2
IS
return_value VARCHAR2(25):= NULL;
v_local_unit_of_measure_S
VARCHAR2(3) := get_uom_code(item_rec_in);
BEGIN
SELECT unit_of_measure
INTO return_value
FROM mtl_units_of_measure
WHERE uom_code = v_local_unit_of_measure_S;
--dbms_output.put_line
('primary_uom----'||return_value);
RETURN return_value;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RETURN NULL;
WHEN OTHERS THEN
RETURN NULL;
END;

```

```

--*****
-- This local function maps
-- the legacy item type to Oracle item type

```

--*****

```
FUNCTION GET_ITEM_TYPE
(item_rec_in IN item_cur%ROWTYPE)
RETURN VARCHAR2
IS
return_value
  VARCHAR2(25):= NULL;
v_kanban_partno
  VARCHAR2(25) := NULL;
BEGIN --H--
SELECT partno
  INTO v_kanban_partno
  FROM kanban_parts
  WHERE partno = item_rec_in.partno;
return_value := 'KANBAN';
EXCEPTION
WHEN NO_DATA_FOUND THEN
  IF item_rec_in.preferred_method = '4'
  THEN
    return_value := 'TRANSFER';
  ELSIF
    substr(item_rec_in.m_item_type_name,1,3)
    = 'DUM'
  THEN
    IF substr(item_rec_in.des,1,3) = 'SCR' OR
    substr(item_rec_in.des,1,3) = 'OBS' OR
    substr(item_rec_in.des,1,3) = 'STA' OR
    substr(item_rec_in.des,1,3) = 'WIR' OR
    substr(item_rec_in.des,1,3) = 'CHA' THEN
      return_value := 'SCREEN';
    ELSIF substr(item_rec_in.des,1,3) = 'LAB'
    THEN
      return_value := 'LABEL';
    ELSIF substr(item_rec_in.des,1,3) = 'NCP'
    THEN
      return_value := 'NCPROG';
    ELSIF substr(item_rec_in.des,1,3) = 'STE'
    THEN
      return_value := 'STENCIL';
    ELSIF substr(item_rec_in.des,1,3) = 'TEM'
    THEN
      return_value := 'TEMPLATE';
    ELSIF substr(item_rec_in.des,1,3) = 'PLA'
    THEN
      return_value := 'PLATING';
    ELSE return_value := 'SCREEN';
    END IF;
  ELSIF substr(item_rec_in.partno,1,2) = '8K'
  THEN
    return_value := 'DISPTOOL';
  ELSIF item_rec_in.phantom_code = 'P'
  THEN
    IF item_rec_in.preferred_method = '1'
    THEN
      return_value := 'PHANTOM';
```

```
ELSIF item_rec_in.preferred_method = '2'
THEN
  return_value := 'FAXBAN';
END IF;
ELSIF
  substr(item_rec_in.m_item_type_name,1,3)
  = 'ASS'
THEN
  IF item_rec_in.preferred_method = '1'
  THEN
    return_value := 'ASSY';
  ELSIF item_rec_in.preferred_method = '2'
  THEN
    return_value := 'RESA';
  END IF;
ELSE
  BEGIN
    SELECT lookup_code
      INTO return_value
      FROM fnd_common_lookups
      WHERE substr(meaning,1,4)
      =
      DECODE(substr(item_rec_in.m_item_type_name,1,4),
        'PIEC','COMP',
        substr(item_rec_in.m_item_type_name,1,4))
        AND lookup_type = 'ITEM_TYPE';
    RETURN return_value;
  EXCEPTION
    WHEN OTHERS THEN
      --dbms_output.put_line('item_type--'||return_value);
      RETURN return_value;
    END;
  END IF;
  --dbms_output.put_line
  ('item_type in the function--'||return_value);
  RETURN return_value;
END; --H--
```

--*****
-- This local function maps
-- item type to mrp_planning_code
--*****

```
FUNCTION
GET_MRP_PLANNING_CODE
(item_rec_in IN item_cur%ROWTYPE)
RETURN NUMBER
IS
return_value NUMBER := null;
BEGIN --Function--
SELECT
  DECODE(item_rec_in.m_item_type_name,
    'SCREEN',6,
    'LABEL' ,6,
    'NCPROG',6,
    'STENCIL',6,
```

```

        'SUPP',6,
        'TEMPLATE',6,3)
    INTO return_value
    FROM DUAL;
--dbms_output.put_line
    ('mrp_planning_code--'||return_value);
    RETURN return_value;
EXCEPTION
    WHEN OTHERS THEN
        RETURN NULL;
END; --Function--

--*****
-- This local function maps
-- Legacy planner code to
-- Oracle planner code
--*****

FUNCTION
    GET_PLANNER_CODE
(item_rec_in IN item_cur%ROWTYPE)
    RETURN VARCHAR2
IS
    return_value VARCHAR2(25) := NULL;
BEGIN
    IF item_rec_in.m_family_code IS NULL
    THEN
        return_value := 'XXX';
        RETURN return_value;
    ELSIF
        item_rec_in.m_family_code = '0' OR
        item_rec_in.m_family_code = '3' OR
        item_rec_in.m_family_code = '5' OR
        item_rec_in.m_family_code = '6' OR
        item_rec_in.m_family_code = '9' THEN
        IF item_rec_in.m_planner = 'AC' OR
        item_rec_in.m_planner = 'AT' OR
        item_rec_in.m_planner = 'LF' OR
        item_rec_in.m_planner = 'LN9' OR
        item_rec_in.m_planner = 'MB' THEN
            return_value := 'XXX';
        ELSIF item_rec_in.m_planner = 'SK9' OR
        item_rec_in.m_planner = 'SKT' OR
        item_rec_in.m_planner = 'SP9' THEN
            return_value := 'JS2';
        ELSIF item_rec_in.m_planner = 'WW0' OR
        item_rec_in.m_planner = 'WW5' OR
        item_rec_in.m_planner = 'WW9' THEN
            return_value := 'WW';
        ELSIF item_rec_in.m_planner = 'JW' THEN
            return_value := 'JW2';
        ELSIF item_rec_in.m_planner = 'XEP' THEN
            return_value := 'SE7';
        ELSIF item_rec_in.m_planner = 'XP2' THEN
            return_value := 'SE2';
        ELSIF item_rec_in.m_planner = 'XP6' THEN

```

```

        return_value := 'SE6';
    ELSIF item_rec_in.m_planner = 'XP9' THEN
        return_value := 'SE9';
    ELSE return_value := item_rec_in.m_planner;
    END IF;
ELSE
    BEGIN
        v_step_cnt := 'H1' ;
        SELECT oracle_planner_code
        INTO return_value
        FROM family_code_temp plan
        WHERE item_rec_in.m_family_code
            = plan.iman_family_code;
--dbms_output.put_line
    ('planner_code---'||return_value);
        RETURN return_value;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            return_value := 'XXX';
            RETURN return_value;
        WHEN OTHERS THEN
            return_value := 'XXX';
            RETURN return_value;
    END;
END IF;
END; --FUNCTION

--*****
-- This local procedure assigns
-- item attributes to the PL/SQL record
--*****

PROCEDURE SET_ATTRIBUTES
(msi_rec_in IN mtl_system_items%ROWTYPE,
item_rec_in IN item_cur%ROWTYPE)
IS
    BEGIN
        msi_rec_in.summary_flag := 'N';
        msi_rec_in.enabled_flag := 'Y';
        msi_rec_in.acceptable_rate_increase := 0;
        msi_rec_in.acceptable_rate_decrease := 0;
        msi_rec_in.shelf_life_code := 1;
        msi_rec_in.planner_code
            := get_planner_code(item_rec_in);
        msi_rec_in.item_type
            := get_item_type(item_rec_in);
        msi_rec_in.acceptable_early_days
            := item_rec_in.c_resched_tol;
        IF item_rec_in.preferred_method = '4' THEN
            msi_rec_in.shippable_item_flag := 'Y';
            msi_rec_in.customer_order_flag := 'Y';
            msi_rec_in.internal_order_flag := 'Y';
            msi_rec_in.customer_order_enabled_flag := 'Y';
            msi_rec_in.internal_order_enabled_flag := 'Y';
            msi_rec_in.so_transactions_flag := 'Y';
            msi_rec_in.planning_make_buy_code := '2';

```

```

msi_rec_in.receiving_routing_id := 3;
msi_rec_in.inspection_required_flag := 'N';
msi_rec_in.fixed_lead_time
:= item_rec_in.c_leadtm_fixP;
msi_rec_in.variable_lead_time
:= item_rec_in.leadtm_varP;
msi_rec_in.postprocessing_lead_time
:= item_rec_in.c_goods_in_tm;
ELSIF item_rec_in.preferred_method = '2'
THEN
msi_rec_in.receiving_routing_id := 2;
msi_rec_in.inspection_required_flag := 'Y';
ELSIF item_rec_in.preferred_method = '1'
THEN
msi_rec_in.fixed_lead_time
:= item_rec_in.c_leadtm_fixM;
msi_rec_in.variable_lead_time
:= item_rec_in.leadtm_varM;
msi_rec_in.receiving_routing_id := 3;
msi_rec_in.inspection_required_flag := 'N';
END IF;
IF msi_rec_in.item_type = 'PHANTOM' THEN
msi_rec_in.fixed_lead_time := 0;
msi_rec_in.variable_lead_time := 0;
msi_rec_in.postprocessing_lead_time := 0;
msi_rec_in.preprocessing_lead_time := 0;
END IF;
IF item_rec_in.phantom_code = 'P' THEN
msi_rec_in.wip_supply_type := 2;
ELSE msi_rec_in.wip_supply_type := 1;
END IF;
EXCEPTION
WHEN OTHERS THEN
NULL;
END; --set_attributes--

--*****
-- This local procedure inserts
-- items into the mtl_system_items table
--*****

PROCEDURE INSERT_ITEMS
(msi_rec_in IN mtl_system_items%ROWTYPE,
item_rec_in IN item_cur%ROWTYPE)
IS
BEGIN
INSERT INTO inv.mtl_system_items
(
inventory_item_id, -- 1
organization_id, -- 2
last_update_date, -- 3
last_updated_by, -- 4
creation_date, -- 5
created_by, -- 6
summary_flag, -- 7
enabled_flag, -- 8
description, -- 9
segment1, -- 10
inspection_required_flag, -- 11
shelf_life_code, -- 12
expense_account, -- 13
primary_uom_code, -- 14
primary_unit_of_measure, -- 15
cost_of_sales_account, -- 16
sales_account, -- 17
planner_code, -- 18
item_type, -- 19
purchasing_item_flag, -- 20
shippable_item_flag, -- 21
customer_order_flag, -- 22
internal_order_flag, -- 23
service_item_flag, -- 24
inventory_item_flag, -- 25
eng_item_flag, -- 26
inventory_asset_flag, -- 27
purchasing_enabled_flag, -- 28
customer_order_enabled_flag, -- 29
internal_order_enabled_flag, -- 30
so_transactions_flag, -- 31
mtl_transactions_enabled_flag, -- 32
stock_enabled_flag, -- 33
bom_enabled_flag, -- 34
build_in_wip_flag, -- 35
pick_components_flag, -- 36
replenish_to_order_flag, -- 37
atp_components_flag, -- 38
atp_flag, -- 39
vendor_warranty_flag, -- 40
serviceable_component_flag, -- 41
serviceable_product_flag, -- 42
preventive_maintenance_flag, -- 43
time_billable_flag, -- 44
material_billable_flag, -- 45
expense_billable_flag, -- 46
prorate_service_flag, -- 47
invoiceable_item_flag, -- 48
invoice_enabled_flag, -- 49
must_use_approved_vendor_flag, -- 50
outside_operation_flag, -- 51
costing_enabled_flag, -- 52
auto_created_config_flag, -- 53
cycle_count_enabled_flag, -- 54
market_price, -- 55
receiving_routing_id, -- 56
revision_qty_control_code, -- 57
taxable_flag, -- 58
allow_item_desc_update_flag, -- 59
receipt_required_flag, -- 60
rfq_required_flag, -- 61
list_price_per_unit, -- 62
price_tolerance_percent, -- 63
lot_control_code, -- 64

```

```

serial_number_control_code, -- 65
restrict_subinventories_code, -- 66
restrict_locators_code, -- 67
location_control_code, -- 68
planning_time_fence_code, -- 69
planning_time_fence_days, -- 70
repetitive_planning_flag, -- 71
bom_item_type, -- 72
wip_supply_type, -- 73
allowed_units_lookup_code, -- 74
inventory_item_status_code, -- 75
inventory_planning_code, -- 76
planning_make_buy_code, -- 77
mrp_safety_stock_code, -- 78
reservable_type, -- 79
mrp_planning_code, -- 80
return_inspection_requirement, -- 81
acceptable_early_days, -- 82
acceptable_rate_increase, -- 83
acceptable_rate_decrease, --84
fixed_lead_time, -- 85
variable_lead_time, -- 86
postprocessing_lead_time, -- 87
preprocessing_lead_time, -- 88
fixed_lot_multiplier, -- 89
minimum_order_quantity, -- 90
segment18, -- 91
segment19, -- 92
attribute6, -- 93
buyer_id -- 94
)
VALUES
(
msi_rec_in.inventory_item_id, -- 1
msi_rec_in.organization_id, -- 2
msi_rec_in.last_update_date, -- 3
msi_rec_in.last_updated_by, -- 4
msi_rec_in.creation_date, -- 5
msi_rec_in.created_by, -- 6
msi_rec_in.summary_flag, -- 7
msi_rec_in.enabled_flag, -- 8
msi_rec_in.description, -- 9
msi_rec_in.segment1, -- 10
msi_rec_in.inspection_required_flag, -- 11
msi_rec_in.shelf_life_code, -- 12
msi_rec_in.expense_account, -- 13
msi_rec_in.primary_uom_code, -- 14
msi_rec_in.primary_unit_of_measure, -- 15
msi_rec_in.cost_of_sales_account, -- 16
msi_rec_in.sales_account, -- 17
msi_rec_in.planner_code, -- 18
msi_rec_in.item_type, -- 19
msi_rec_in.purchasing_item_flag, -- 20
msi_rec_in.shippable_item_flag, -- 21
msi_rec_in.customer_order_flag, -- 22
msi_rec_in.internal_order_flag, -- 23
msi_rec_in.service_item_flag, -- 24
msi_rec_in.inventory_item_flag, --25
msi_rec_in.eng_item_flag, -- 26
msi_rec_in.inventory_asset_flag, -- 27
msi_rec_in.purchasing_enabled_flag, -- 28
msi_rec_in.customer_order_enabled_flag, -- 29
msi_rec_in.internal_order_enabled_flag, -- 30
msi_rec_in.so_transactions_flag, -- 31
msi_rec_in.mtl_transactions_enabled_flag, -- 32
msi_rec_in.stock_enabled_flag, -- 33
msi_rec_in.bom_enabled_flag, -- 34
msi_rec_in.build_in_wip_flag, -- 35
msi_rec_in.pick_components_flag, -- 36
msi_rec_in.replenish_to_order_flag, -- 37
msi_rec_in.atp_components_flag, -- 38
msi_rec_in.atp_flag, -- 39
msi_rec_in.vendor_warranty_flag, -- 40
msi_rec_in.serviceable_component_flag, -- 41
msi_rec_in.serviceable_product_flag, -- 42
msi_rec_in.preventive_maintenance_flag, -- 43
msi_rec_in.time_billable_flag, -- 44
msi_rec_in.material_billable_flag, -- 45
msi_rec_in.expense_billable_flag, -- 46
msi_rec_in.prorate_service_flag, -- 47
msi_rec_in.invoiceable_item_flag, -- 48
msi_rec_in.invoice_enabled_flag, -- 49
msi_rec_in.must_use_approved_vendor_flag, --
msi_rec_in.outside_operation_flag, --51
msi_rec_in.costing_enabled_flag, --52
msi_rec_in.auto_created_config_flag, --53
msi_rec_in.cycle_count_enabled_flag, --54
msi_rec_in.market_price, --55
msi_rec_in.receiving_routing_id, --56
msi_rec_in.revision_qty_control_code, --57
msi_rec_in.taxable_flag, --58
msi_rec_in.allow_item_desc_update_flag, --59
msi_rec_in.receipt_required_flag, --60
msi_rec_in.rfq_required_flag, --61
msi_rec_in.list_price_per_unit, --62
msi_rec_in.price_tolerance_percent, --63
msi_rec_in.lot_control_code, --64
msi_rec_in.serial_number_control_code, --65
msi_rec_in.restrict_subinventories_code, --66
msi_rec_in.restrict_locators_code, --67
msi_rec_in.location_control_code, --68
msi_rec_in.planning_time_fence_code, --69
msi_rec_in.planning_time_fence_days, --70
msi_rec_in.repetitive_planning_flag, --71
msi_rec_in.bom_item_type, -- 72
msi_rec_in.wip_supply_type, --73
msi_rec_in.allowed_units_lookup_code, --74
msi_rec_in.inventory_item_status_code, --75
msi_rec_in.inventory_planning_code, --76
msi_rec_in.planning_make_buy_code, --77
msi_rec_in.mrp_safety_stock_code, --78
msi_rec_in.reservable_type, --79

```

```

msi_rec_in.mrp_planning_code,      --80
msi_rec_in.return_inspection_requirement, --81
msi_rec_in.acceptable_early_days,  --82
msi_rec_in.acceptable_rate_increase, --83
msi_rec_in.acceptable_rate_decrease, --84
msi_rec_in.fixed_lead_time,        --85
msi_rec_in.variable_lead_time,     --86
msi_rec_in.postprocessing_lead_time, --87
msi_rec_in.preprocessing_lead_time, --88
msi_rec_in.fixed_lot_multiplier,   --89
msi_rec_in.minimum_order_quantity, --90
msi_rec_in.segment18,              --91
msi_rec_in.segment19,              --92
msi_rec_in.wip_supply_type,        --93
msi_rec_in.buyer_id                --94
);
    UPDATE item6_temp item
        SET error_flag = 'Z'
        WHERE item.rowid = item_rec_in.rowid;
EXCEPTION
    WHEN OTHERS THEN
--dbms_output.put_line('LOOP INSERT BLOCK
EXCEPTION');
    v_step_cnt := 'M';      /* linh*/
    v_error_msg := substr(sqlerrm,1,80);
    insert_error(item_rec_in.rowid,
        v_error_msg);
END; --Procedure--
BEGIN --main block--
/**** Delete items done in phase 3 ****/
    BEGIN
        DELETE
            FROM mtl_system_items
            WHERE segment1 = '88S5454';
        COMMIT;
    EXCEPTION
        WHEN OTHERS THEN
--dbms_output.put_line
('Error deleting phase 3 items');
    END;
--*****
-- A CURSOR FOR LOOP is used
-- to load make and buy cursor
-- records into memory and the
-- records from the temporary
-- table are processed
-- in the inner LOOP.
--*****
FOR make_rec IN make_cur LOOP
    FOR buy_rec IN buy_cur LOOP
        FOR item_rec in item_cur LOOP
            BEGIN --XXX--

--*****
-- This control is used to
-- commit every 25 rows processed

```

```

--*****
    IF v_commit_count > 10 THEN
        commit;
        v_commit_count := 0;
    ELSE
        v_commit_count
            := v_commit_count + 1;
    END IF;
--*****
-- In the following section
-- variables are reset for
-- every pass of the loop
--*****
        v_skip_insert1:= FALSE;
        v_step_cnt := NULL;
        v_po_ck_item := NULL;
        IF item_rec.preferred_method = '1'
        THEN
            copy_row_rec := make_rec;
        ELSIF
            item_rec.preferred_method = '4'
        THEN
            copy_row_rec := buy_rec;
        ELSIF item_rec.preferred_method = '2'
        THEN
            copy_row_rec := buy_rec;
        END IF;
--*****
-- The following get_user_id
-- function is a stored function
--*****
        copy_row_rec.created_by
            := get_user_id('PHASE3CONV');
        copy_row_rec.last_updated_by
            := get_user_id('PHASE3CONV');
        copy_row_rec.creation_date
            := sysdate;
        copy_row_rec.last_update_date
            := sysdate;
        copy_row_rec.primary_unit_of_measure
            := get_primary_uom(item_rec);
        copy_row_rec.primary_uom_code
            := get_uom_code(item_rec);
        v_error_occurred
            := FALSE;
        v_item_found_in_msi
            := FALSE;
        copy_row_rec.inventory_item_id
            := NULL;
        v_error_msg
            := NULL;
        v_master_org_item_type
            := NULL;

```

```

copy_row_rec.segment1
:= item_rec.partno;
copy_row_rec.receiving_routing_id
:= NULL;
copy_row_rec.inspection_required_flag
:= NULL;
copy_row_rec.receiving_routing_id
:= NULL;
v_org1_ins_req_flag
:= NULL;
copy_row_rec.wip_supply_type
:= item_rec.phantom_code;
copy_row_rec.fixed_lead_time
:= NULL;
copy_row_rec.variable_lead_time
:= NULL;
copy_row_rec.postprocessing_lead_time
:= NULL;
copy_row_rec.preprocessing_lead_time
:= NULL;
copy_row_rec.planner_code
:= NULL;
copy_row_rec.description
:= item_rec.des;
copy_row_rec.fixed_lot_multiplier
:= item_rec.unit_of_packaging;
copy_row_rec.minimum_order_quantity
:= item_rec.min_order_qty;
--*****
-- Purchased parts were
-- converted in phase II conversion and
-- should not be converted in this phase.
-- The following PL/SQL block checks
-- for the existence of the
-- item in the Oracle table
--*****
BEGIN
SELECT
msi.inventory_item_id,
msi.buyer_id,
msi.item_type,
msi.inspection_required_flag,
msi.receiving_routing_id
INTO
copy_row_rec.inventory_item_id,
copy_row_rec.buyer_id,
v_master_org_item_type,
copy_row_rec.inspection_required_flag,
copy_row_rec.receiving_routing_id
FROM mtl_system_items msi
WHERE msi.segment1
= rtrim(upper(item_rec.partno))
AND msi.organization_id = 1;
v_skip_insert1 := TRUE;
EXCEPTION
WHEN NO_DATA_FOUND THEN
v_skip_insert1 := FALSE;
WHEN OTHERS THEN
v_skip_insert1 := FALSE;
END;

IF v_skip_insert1 = TRUE THEN
--*****
-- The following get_org_id
-- function is a stored function
--*****
copy_row_rec.organization_id
:= get_org_id(item_rec.factory);
ELSE copy_row_rec.organization_id
:= get_org_id('SEC');
END IF;
copy_row_rec.item_type
:= get_item_type(item_rec);
copy_row_rec.mrp_planning_code
:= get_mrp_planning_code(item_rec);

IF v_error_occurred = FALSE
AND v_skip_insert1 = FALSE
THEN
v_step_cnt := 'M';
--dbms_output.put_line('FIRST LOOP-'
||copy_row_rec.segment1);
SELECT mtl_system_items_s.nextval
INTO copy_row_rec.inventory_item_id
FROM DUAL;
copy_row_rec.organization_id
:= get_org_id('SEC');
copy_row_rec.buyer_id
:= NULL;
copy_row_rec.item_type
:= NULL;
copy_row_rec.allow_item_desc_update_flag
:= NULL;
--*****
-- Items are inserted into the
-- item master by calling the procedure
--*****
insert_items(copy_row_rec,
item_rec);
END IF;
/* 2nd Insert */
BEGIN
v_step_cnt := 'N';
copy_row_rec.organization_id
:= get_org_id(item_rec.factory);
--dbms_output.put_line('org_id in second loop--'
||copy_row_rec.organization_id);
SELECT msi.inventory_item_id
INTO v_po_ck_item

```

```

        FROM mtl_system_items msi
        WHERE segment1 = item_rec.partno
            AND organization_id =
                copy_row_rec.organization_id;
v_item_found_in_msi := TRUE;
        UPDATE item6_temp item
        SET error_flag = 'I',
            error_msg =
                'Item already in the master Org'
        WHERE item.rowid = item_rec.rowid;
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        v_item_found_in_msi := TRUE;
        UPDATE item6_temp item
        SET error_msg =
            'More than one row in mtl_system_items',
            error_flag = '?'
        WHERE item.rowid = item_rec.rowid;
    WHEN NO_DATA_FOUND THEN
        v_item_found_in_msi := FALSE;
END;
IF v_error_occurred = FALSE
    AND v_item_found_in_msi = FALSE
    THEN
--dbms_output.put_line('item_id --'
    ||copy_row_rec.inventory_item_id);

--*****
-- Items are inserted into
-- the organization where
-- it is enabled with a
-- call to the procedure
--*****

        insert_items (copy_row_rec,
            item_rec);
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND
    THEN
        UPDATE item6_temp item
        SET error_msg =
            'error in the main block',
            error_flag = 'I'
        WHERE item.rowid = item_rec.rowid;
    WHEN OTHERS THEN
        v_error_msg := substr(sqlerrm,1,80);
        insert_error (item_rec.rowid,
            v_error_msg);
        END; -- loop block end--
    END LOOP;
END LOOP;
END LOOP;
END; --main block--
/
exit;

```