ORACLE APPLICATIONS

1. INTRODUCTION

ERP: Resource Planning with in an Enterprise. ERP is a term that covers whole Product line. ERP means integration of different module. Any business will greatly benefits by adapting this feature because u can customize it or integrate it with other Packages to satisfy unique requirements.

BENEFITS OF ERP: 1). Flow of Information Effectively
2). Maintaining Standardizations

Q: What is Oracle Applications?

Ans: Oracle Applications are an ERP Package. The Key Feature of the entire oracle-Application module is Data Integration.

Master data is Integrated: All the application share common files of customers, suppliers, employee, items and other entities that are used by multiple applications.

Transaction data is Integrated: Oracle automatically bridge transactions from one system to another.

Financial data is integrated: Financial data is carried in a common format, and financial data is transmitted from one application to another.

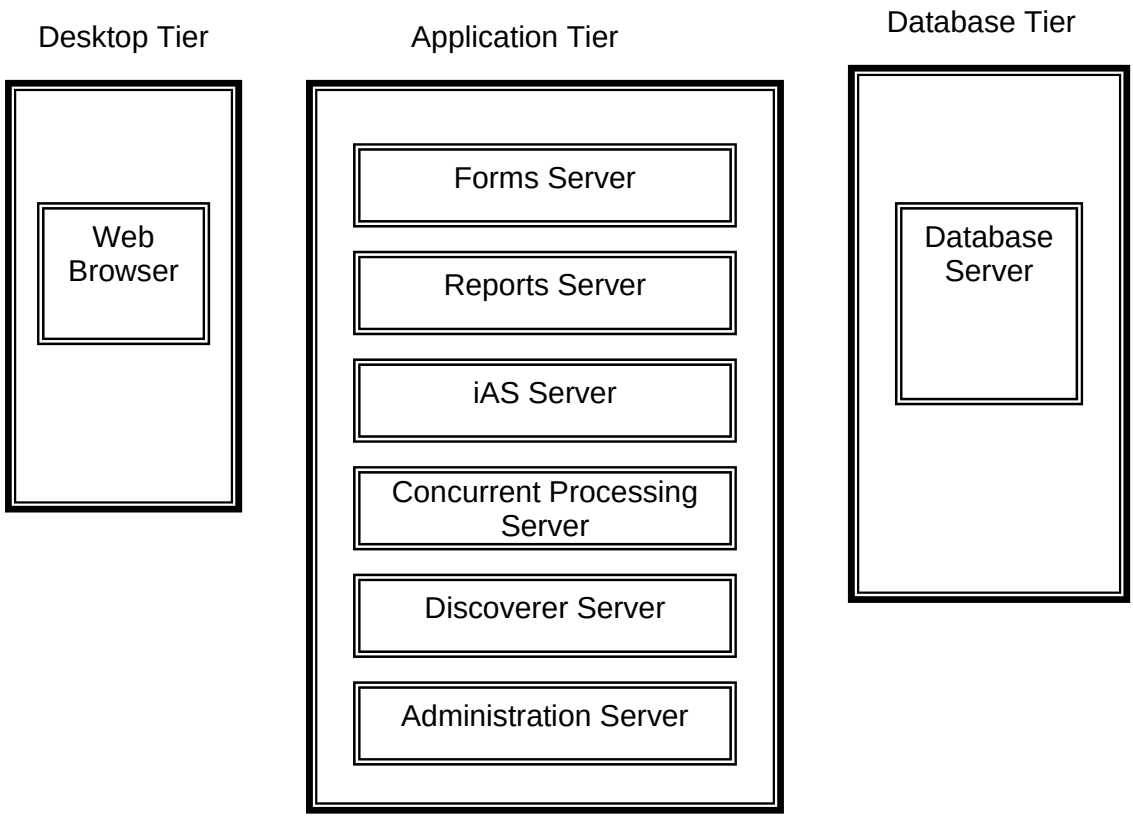
2. ORACLE APPLICATIONS

Oracle Applications is one of the Enterprise Resource Planning (ERP) Business Application packages. It comprises of various Modules, Libraries, Forms, Reports, etc. Oracle Applications is designed on the basis of Generally Acceptable Accounting Principles (GAAP). It is used to chatter the Business needs of an Organization. It consists of 256 different types of modules, 4000 Forms, 5000 Programs etc. Any Organization can adopt this package and use the supplied modules with the customization as per their business requirements.

Oracle works on 3 Tier Architecture (i.e. Client, Application and Database). It supports Form based Interface (i.e. FIN, MFG, MM & HR) and HTML based Interface (i.e. CRM, SSWA).

Note: SSWA – Self Service Web Architecture

Applications 3 – Tier Architecture



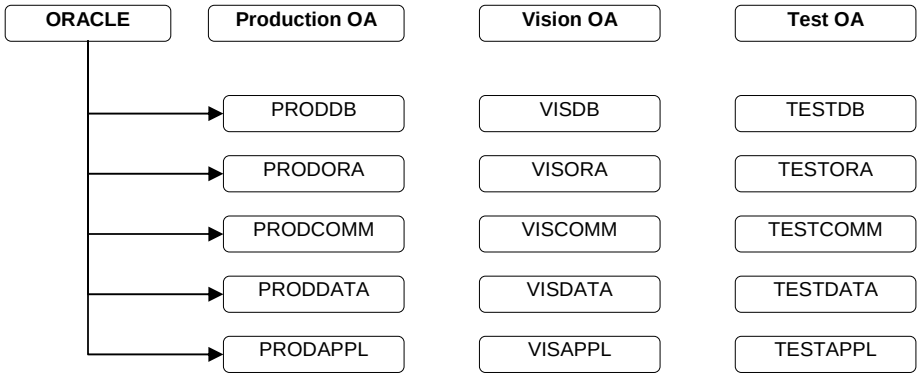
3. PRODUCT DIRECTORY STRUCTURE

When Oracle Applications installs in a machine it creates a base directory with the name ORACLE. This is the base directory for accessing the application.

Oracle Applications supplied in three flavors

1. **Production Oracle Applications:** used for implementing in any organization.
2. **Vision Oracle Applications:** used for Demonstration or Training purpose.
3. **Test Oracle Applications:** used for R&D purpose

Product Directory Structure of Oracle Applications



VISDB: Consists of Oracle 8i / Oracle 9i / Oracle 10g Database executable files.

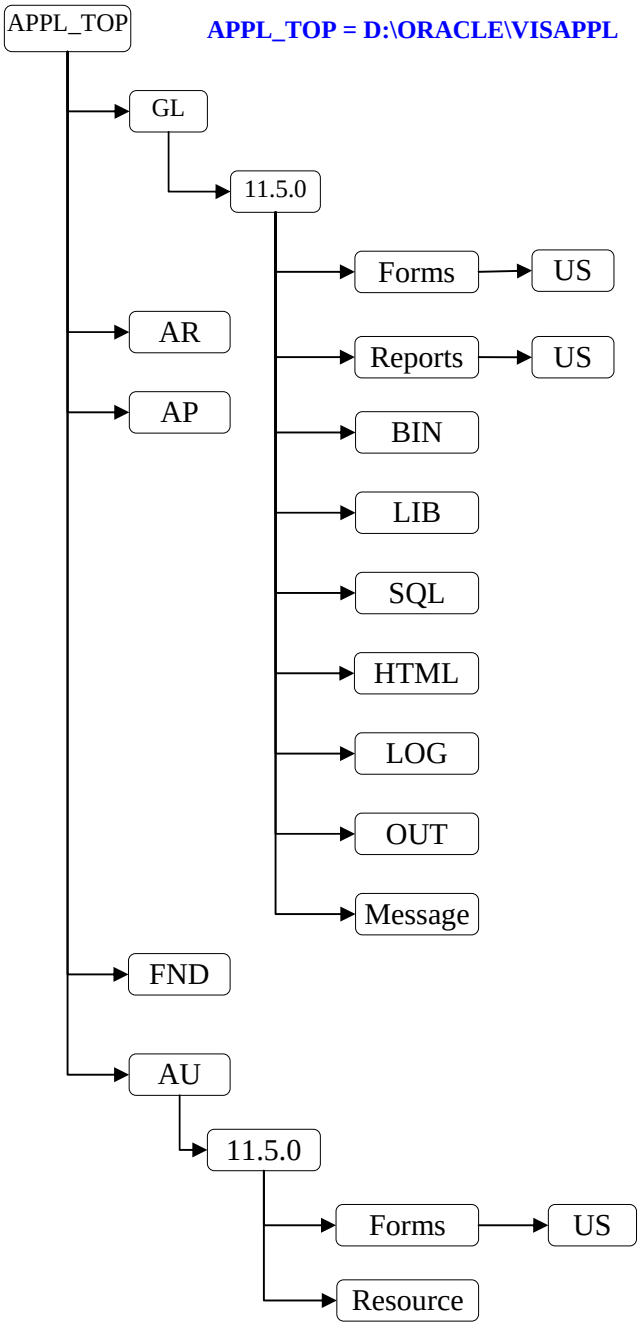
VISORA: Captures 8.0.6 / 9iAS / 10giAS middleware Database executable files.

VISCOMM: Captures Control Script files, which are used to Start or Stop the Application Services.

VISDATA: Captures all .DBF files.

VISAPPL: Captures all Module Specific Directories and Some Special Directories.

APPL_TOP (i.e. VISAPPL) Directory Structure



GL_TOP: (APPL_TOP/GL/11.5.0) is one of the Module Directory of Oracle Applications. It consists of a release directory (i.e. 11.5.0) under which Forms, Reports, BIN, LIB, SQL, etc.,

GL_TOP/11.5.0/Forms/US is forms directory to store all .FMX (Compiled) Form files of a specific module.

GL_TOP/11.5.0/Reports/US is a reports directory to capture all the .RDF (Compiled) Report files of a specific module directory.

US is a language specific directory.

GL_TOP/11.5.0/BIN will capture the C, C++, PRO*C, SQL *LOADER etc., files.

GL_TOP/11.5.0/LIB will capture all .OBJ files of C, C++ or JAVA CLASS files.

GL_TOP/11.5.0/Message will capture all .MSB message body files.

GL_TOP/11.5.0/LOG will capture all .LOG files.

GL_TOP/11.5.0/OUT will capture all .OUT files.

GL_TOP/11.5.0/SQL will capture all .SQL script files.

GL_TOP/11.5.0/HTML will capture all .HTML, .HTM web files.

GL_TOP/11.5.0/FND is a Special Directory known as Application Object Library (AOL) directory. It is used to capture all Modules Application Executable Files.

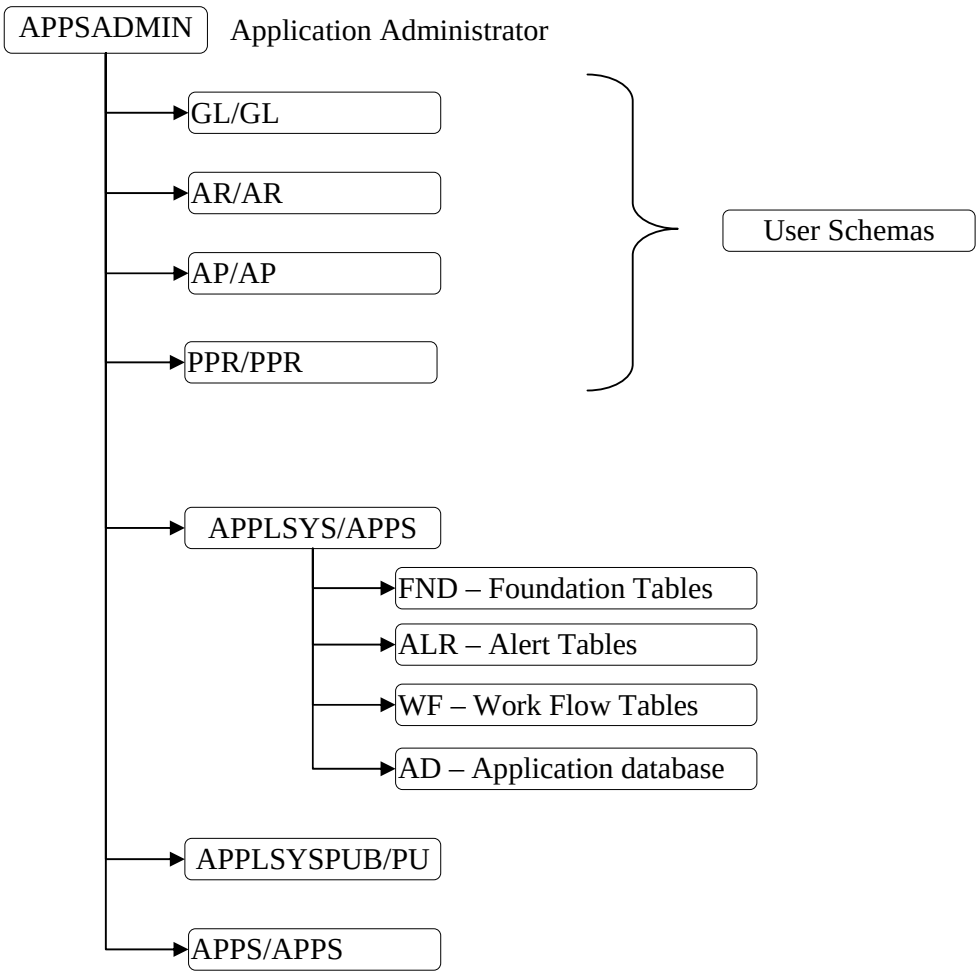
GL_TOP/11.5.0/AU is an Application Utility (AU) Directory. It consists of a application release Sub-Directory, which consists of Forms and Resource directories. It is used to store all .FMB and .PLL and PL/SQL Library files.

GL_TOP/11.5.0/AU/11.5.0/Forms/US will capture .FMB Form files of all Modules.

GL_TOP/11.5.0//AU/11.5.0/Resource is used to store .PLL and .PL/SQL Library files of Oracle Applications.

4. DATA MODEL

When we install Oracle Database by default system will creates SYS and SYSTEM schemas. These consist of all Data Dictionary Tables. Like this if we install Oracle Applications System will automatically creates schemas of all Modules (i.e. GL, AR, AP, etc.) with the respective module name as User and Password. Along with these schemas some special Schemas i.e. APPS, APPLSYS, APPLSYSPUB will be created for special purpose.



APPS Schema: It is Public Schema. It consists of a collection of public synonym of all the objects of all the schemas in the Application database. All the Procedures, Functions and Packages created must be stored in this Schema.

APPLSYS Schema: This is a special Schema consists of the files starts with FND, ALR, WF and AD.

APPLSYSPUB Schema: This schema is a collection of public synonyms of all FND Tables, which are used for User verification. This is the Gate Way User ID of Oracle Applications.

Note: 1. When we are changing the APPS Schema password, first we have to change in the backend for both APPS and APPLSYS Schemas.

2. Password for both APPS and APPLSYS should be same.
3. Change the password of both the Schemas in Front-End and Back-End.
4. Drop the Concurrent Manager Services and re-create the Concurrent Manager Service with the Password as APPS Password.

5. RESPONSIBILITIES AND USER

Responsibility: It is a role authorized to access specific Forms and Programs of a particular Module. A responsibility is a collection of Menus, Request Groups and Data Groups. Menus and Data Groups are mandatory to a responsibility.

Menu: A menu is a collection of Sub-Menus and Functions.

Request Group (RG): It is a collection of concurrent Programs. It is used to request programs from the responsibility.

Data Group (DG): It is a collection of Modules used to integrate one or more Modules for cross application transfer of data, cross application reporting and cross application reference. If we want to get data from other Modules we need to define those modules in the Data Group.

Functions: A function is a part of an application's functionality that is registered under a unique name for the purpose of assigning it to, or excluding it from, a menu (and by extension, a responsibility).

An Oracle Applications feature that let's you control user access to certain functions and windows. By default, access to functionality is *not* restricted; your system administrator customizes each responsibility at your site by including or excluding functions and menus in the Responsibilities window. There are several types of functions: form functions, sub-functions, and non–form functions.

These are two types of functions

1. **Form Functions:** are used to secure Forms from the responsibilities.
2. **Non-Form Functions:** are used to secure Layout Items with in the Form.

Advantages of Form Functions

If you want to open the Form in different modes without creating the copies, we can create a Form Function and pass the parameters based on the requirement. The parameter, which is passed in the form function, must be already defined in the form while designing the Form.

A menu can be assigned to more than one Responsibility. If you want to restrict some of the Forms from a particular responsibility, we can include Form Function of those Forms in Menu Exclusions of the Responsibility.

Pre-Defined Responsibilities

S.No	Module	Responsibility
1	Application Object library	Application Developer
2	System Administration	System Administrator
3	Oracle General Ledger	General Ledger Super User
4	Oracle Public Sector Payables	Payables Manager
5	Oracle Receivables	Receivables Manager
6	Oracle Alerts	Alerts Manager

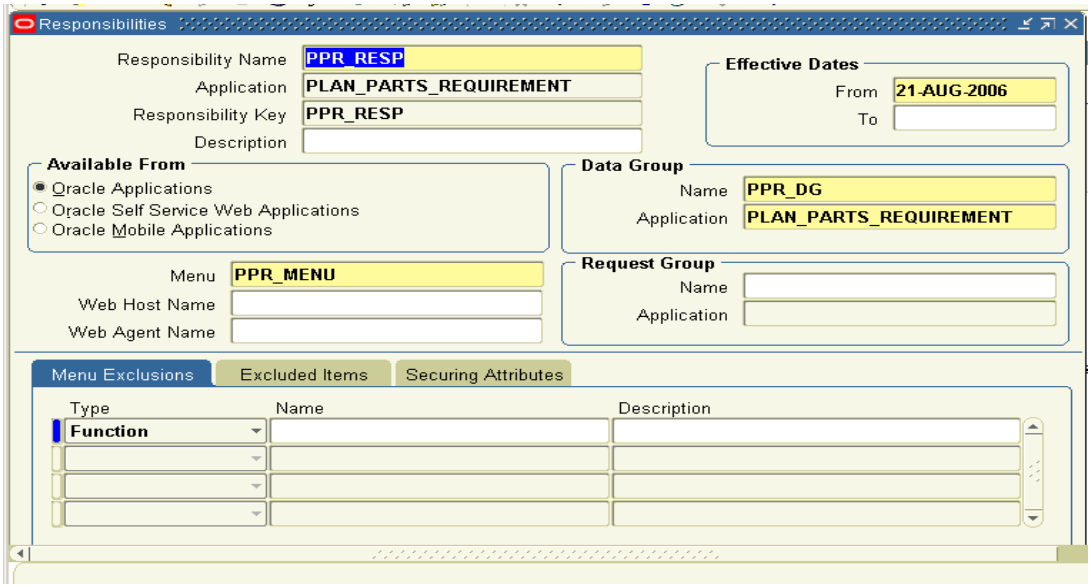
How to Create a Responsibility?

Step 1: Connect to Oracle Application with APPSADMIN/APPSADMIN User.

Step 2: Go to Application Administrator.

Step 3: Go to Security then Responsibility then Define

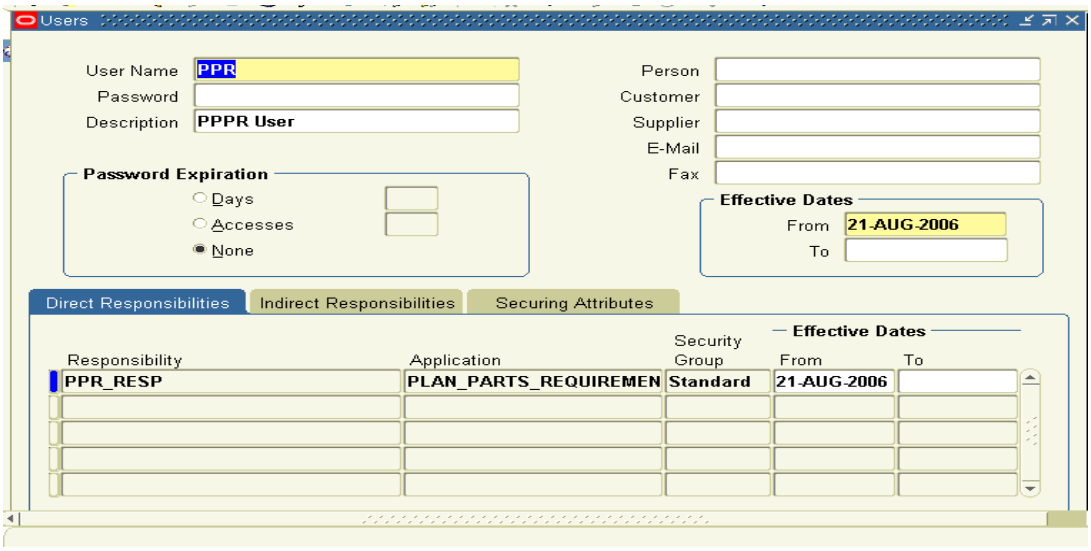
Step 4: Enter the Data in the opened Window.



Note: Above Information of Responsibility is stored in FND_RESPONSIBILITY Table.

How to Create a User?

Step 1: Connect to Oracle Application with APPSADMIN/APPSADMIN User.
Step 2: Go to Application Administrator.
Step 3: Go to Security then User then Define
Step 4: Enter the Data in the opened Window.

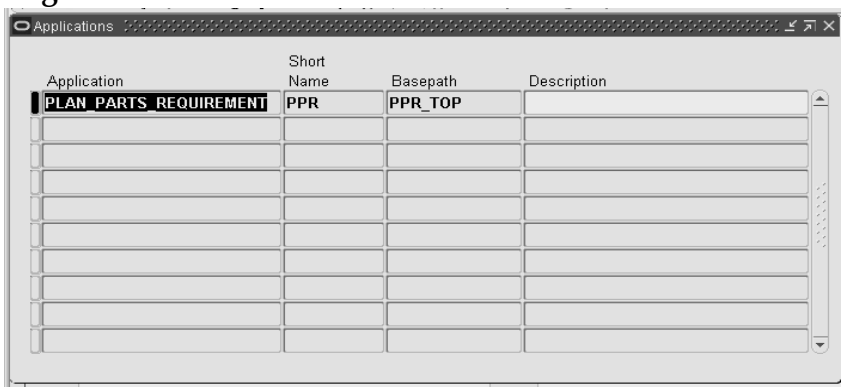


Note: Above Information of User is stored in FND_USER Table.
User Responsibilities stored in FND_USER_RESP_GROUPS Table.

6. NEW MODULE DEVELOPMENT

Steps required for developing a New Module?

1. **Register the New Module with AOL module.**



Save the Entries.

2. Create a Schema/User for the New Module and Grant Rolls to it from SQL Prompt.

Login as apps/apps@vis.
SQL>CREATE USER PPRS IDENTIFID BY PPRS;
SQL>GRANT CONNECT, RESOURCE TO PPRS;
SQL>

3. Create product Directory Structure to the New Module at Server end.

\\Orafin8\d$\appl\pprs\11.5.0\forms\us
\\Orafin8\d$\appl\pprs\11.5.0\reports\us
\\Orafin8\d$\appl\pprs\11.5.0\bin
\\Orafin8\d$\appl\pprs\11.5.0\sql etc.,

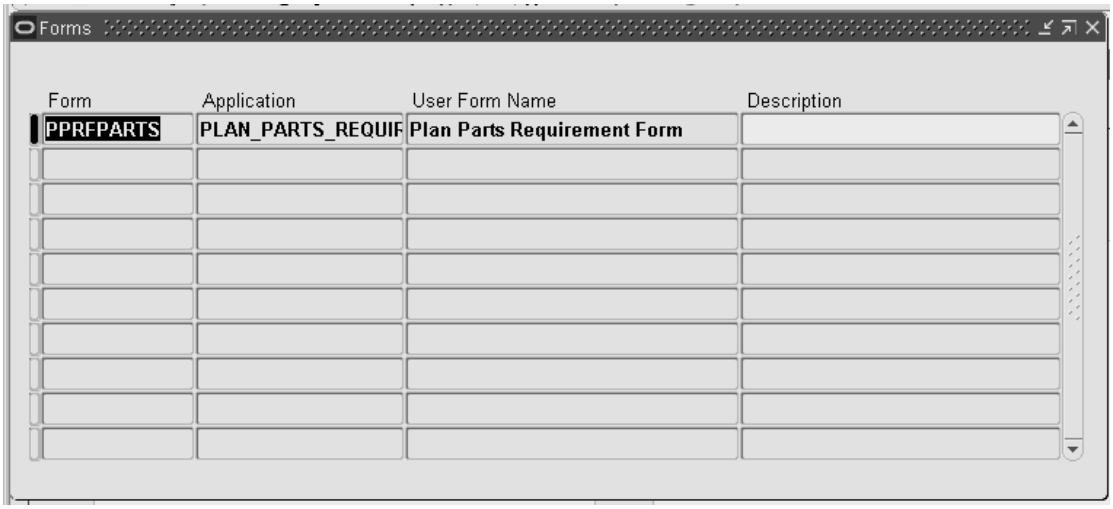
4. **Create the Form Using Form Builder with all the coding standards required according to Oracle Applications.**

5. **Compile the Form and Copy .FMX file in Module Specific (New) Directory.**

6. **Copy some .fmx Form files from any existing modules to new module directory for registration demo.**

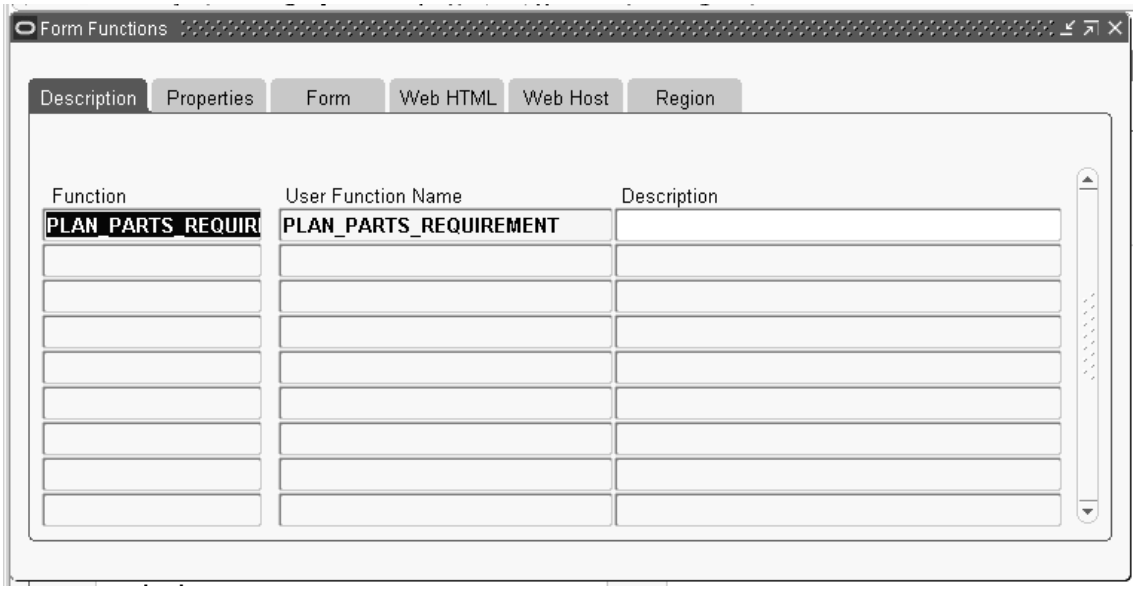
7. **Register the form with Application Object Library (AOL) Module.**
   Connect to APPSTECH/APPSTECH
   Go to Application Developer

Application
Form



Save & Close

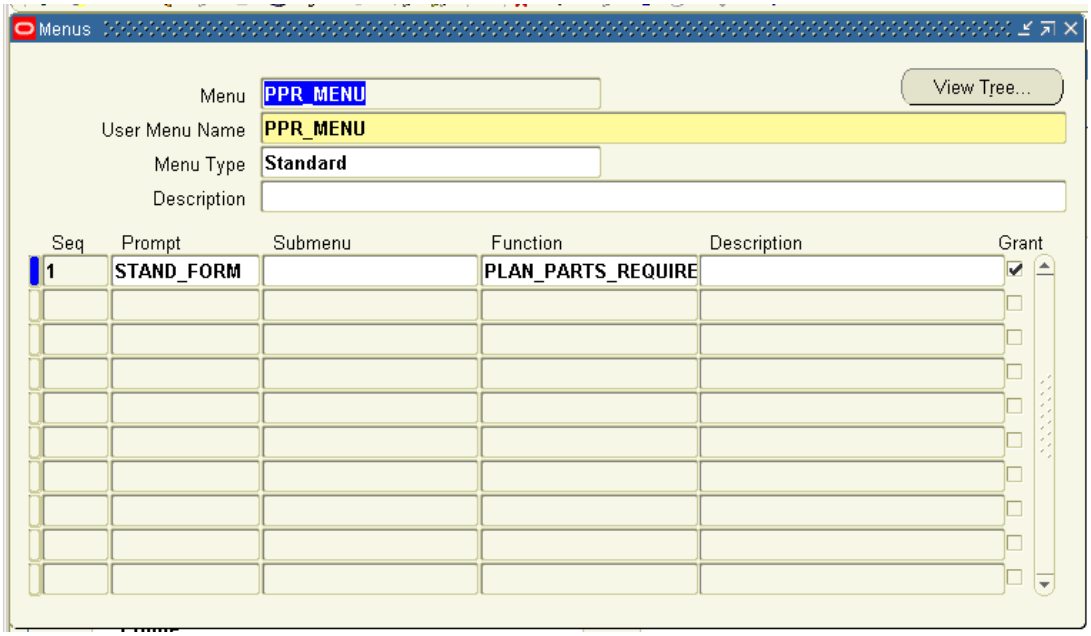
8. **Create Form Function for all the Form, which are registered with our module.**
   Connect to APPSTECH/APPSTECH
   Go to Application Developer
   Application
   Function



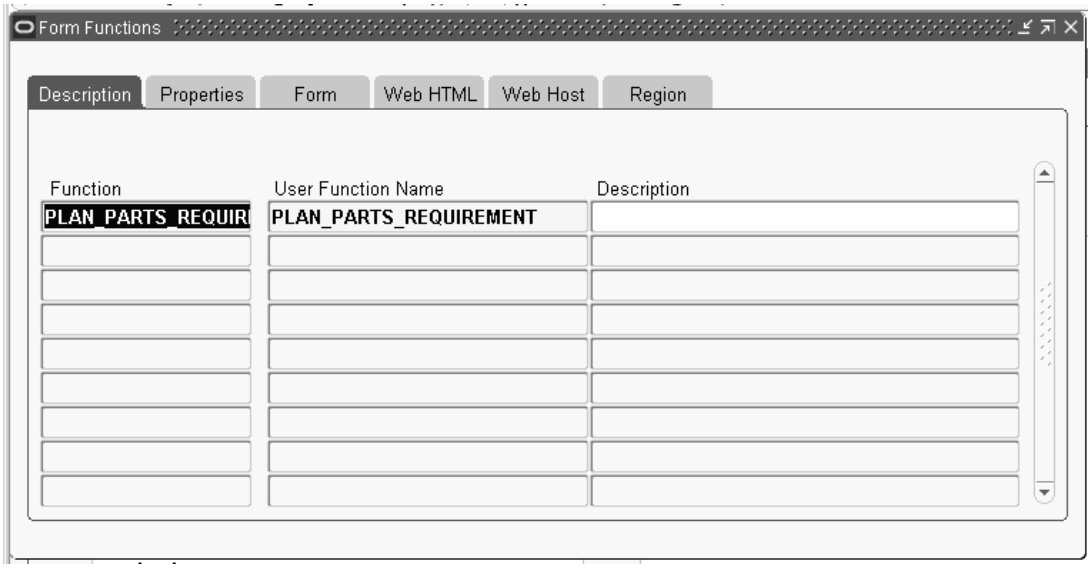
Save & Close

9. **Define a menu for our new module using AOL Module.**
   Connect to APPSTECH/APPSTECH

Go to Application Developer
Application
Menu

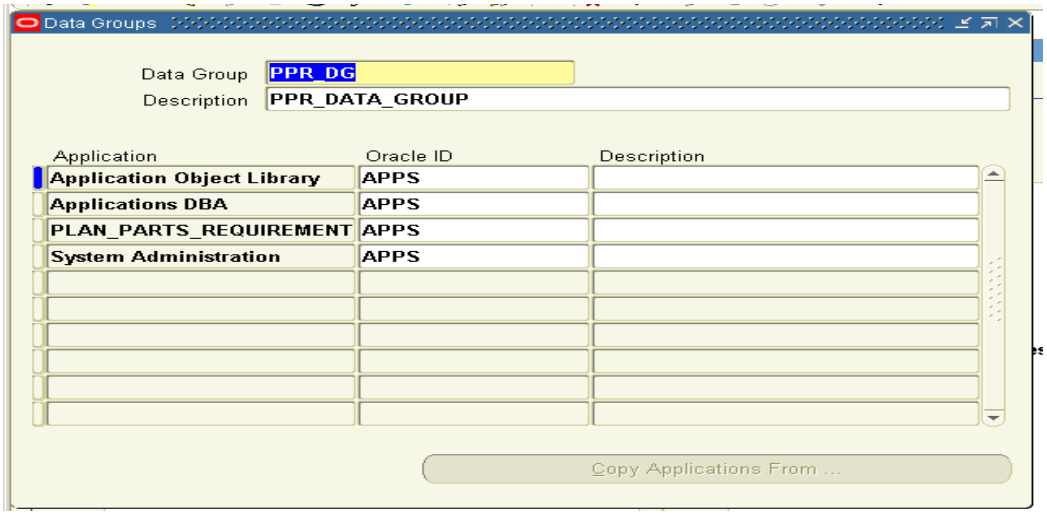


10. Assign the required Form Functions to the menus.



11. Define Data Group (DG) using System Administration Module.

System Administrator
Security
Oracle
Data Group



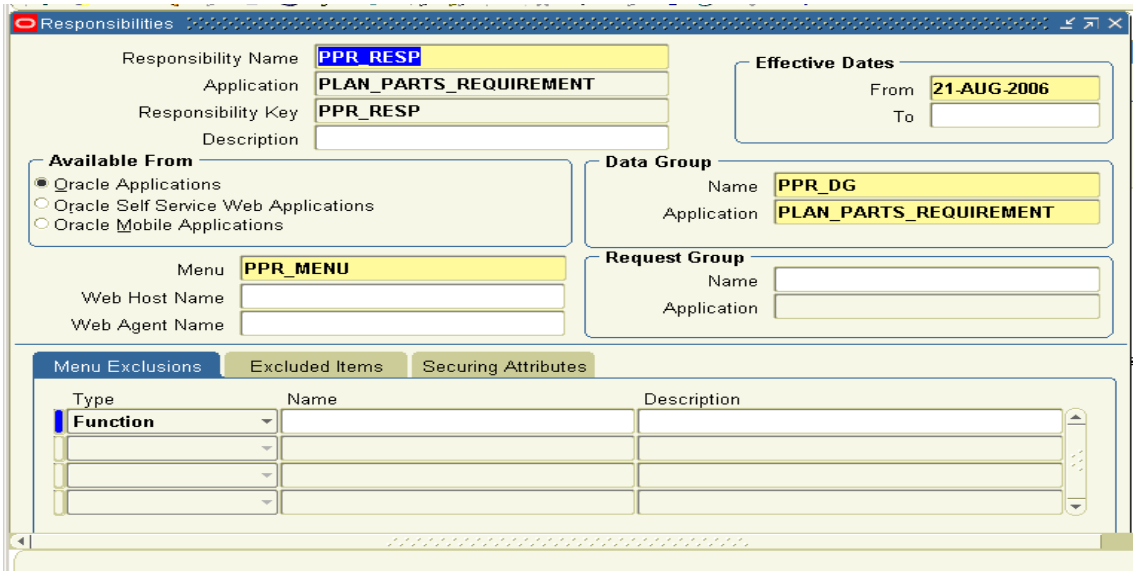
11. Define a Responsibility for the new module.

System Administrator

Security

Responsibility

Define



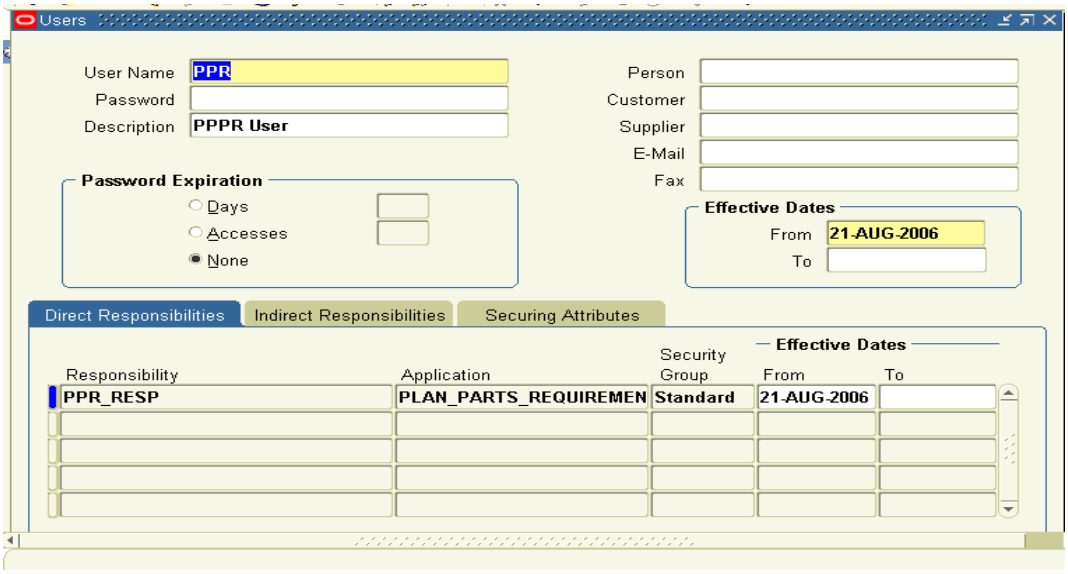
12. Assign Responsibility to the User.

System Administrator

Security

User

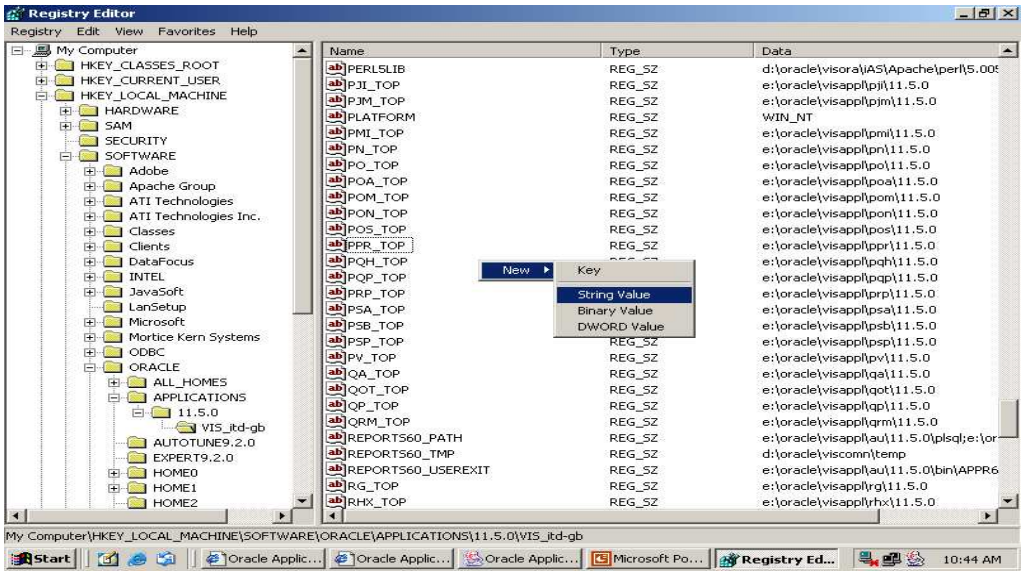
Define



13. Setup the base path of the New Module at the Server end.

PPRS_TOP is the base path

At Run: regedit



Create a new String

Rename the New String as PPRS_TOP

PPRS_TOP: \\Oracle\visappl\pprs\11.5.0

7. TABLE REGISTRATION

Q: Why we have to register a Table?

Ans: To define the Special Objects at the time of definition of a Business Applications.
Special Objects are Key Flex Fields (KFF) and Descriptive Flex Fields (DFF)

In Applications Tables are classified into three categories

1. **Transaction Data Tables:** are normal tables, which are used to store the data in all the modules in which we can perform any DML operations.
2. **Seed Data Tables:** The data for these tables created at the time of installation. The records present in these tables are read only.
3. **Interim Data Tables:** are the temporary tables, which are used for validation purpose. These tables are used when we are transferring the data from external applications to the system tables.

Steps Required for registering a Table

1. Create a Table in Module Specific Schema.

Connect as PPRS/PPRS@VIS

```
SQL>  CREATE TABLE PPRS_PARTS (
        PART_CODE  NUMBER(5),
        PART_NO    VARCHAR2(50),
        PART_NAME VARCHAR2(100),
        QPS    NUMBER(3),
        PPART_CODE NUMBER(5));
```

2. Create a Public Synonym in APPS Schema.

Connect to APPS/APPS@VIS

SQL> CREATE PUBLIC SYNONYM PPRS_PARTS FOR PPRS_PARTS;

Note: Public Synonym Name should be same as Table Name;

3. Register the Table with AOL Module.

Connect to APPSTECH/APPSTECH
Application Developer
Functions
Application
Database
Table

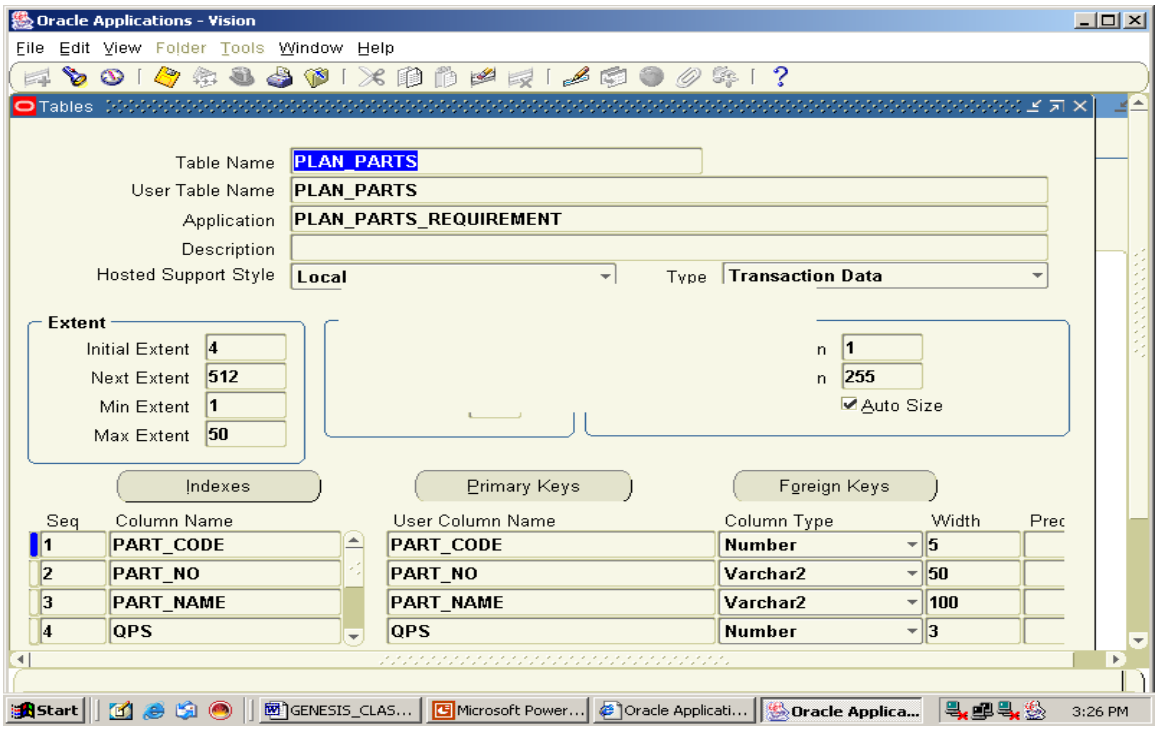


Table Registration with Application Interface (API)

We can also register the Table using Application DBA Data Definitions (AD_DD) Package from the Back End. You can also use the AD_DD API to delete the registrations of tables and columns from Oracle Application Object Library tables. To alter a registration you should first delete the registration, and then reregister the table or column. You should delete the column registration first, then the table registration.

```
procedure register_table (p_appl_short_name in varchar2,
        p_tab_name in varchar2, p_tab_type in varchar2,
        p_next_extent in number default 512, p_pct_free in number default 10,
        p_pct_used in number default 70);

procedure register_column (p_appl_short_name in varchar2,
        p_tab_name in varchar2, p_col_name in varchar2,
        p_col_seq in number, p_col_type in varchar2,
        p_col_width in number, p_nullable in varchar2,
        p_translate in varchar2, p_precision in number default null,
        p_scale in number default null);

procedure delete_table (p_appl_short_name in varchar2, p_tab_name in varchar2);

procedure delete_column (p_appl_short_name in varchar2,
        p_tab_name in varchar2, p_col_name in varchar2);
```

```
SQL> ed
Wrote file afiedt.buf

  1  BEGIN
  2   AD_DD.REGISTER_TABLE('PPR','PLAN_PARTS','T');
  3* END;
SQL> /

PL/SQL procedure successfully completed.

SQL> ed
```

```
BEGIN
ad_dd.register_column('PPR', 'PLAN_PARTS','PART_CODE', 1, 'NUMBER', 5, 'N', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','PART_NO', 2, 'VARCHAR2', 50, 'N', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','PART_NAME', 3, 'VARCHAR2', 100, 'N', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','QPS', 4, 'NUMBER', 3, 'N', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','PPART_CODE', 5, 'NUMBER', 5, 'N', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','SEGMENT1', 6, 'VARCHAR2', 30, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','SEGMENT2', 7, 'VARCHAR2', 30, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','SEGMENT3', 8, 'VARCHAR2', 30, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','SEGMENT4', 9, 'VARCHAR2', 30, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','SEGMENT5', 10, 'VARCHAR2', 30, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','STRUCTURE_ID', 11, 'NUMBER', 15, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','CCID', 12, 'NUMBER', 15, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','ATTRIBUTE1', 13, 'VARCHAR2', 150, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','ATTRIBUTE2', 14, 'VARCHAR2', 150, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','ATTRIBUTE3', 15, 'VARCHAR2', 150, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','ATTRIBUTE4', 16, 'VARCHAR2', 150, 'Y', 'N');
ad_dd.register_column('PPR', 'PLAN_PARTS','CONTEXT', 17, 'VARCHAR2', 25, 'Y', 'N');
END;
/
```

Note: Table can be register from Front End by Application Developer or Back End by Database Administrator.

Type of Table Suffixes and their meaning

_TL	Translation Tables, Used to Store the Information of Language Specific
No Suffix	Are Normal Tables
_V	Are Views
_VL	Are Views based on Translation Tables
_All	Indicates Multi Organization Tables

Applications Storages Tables

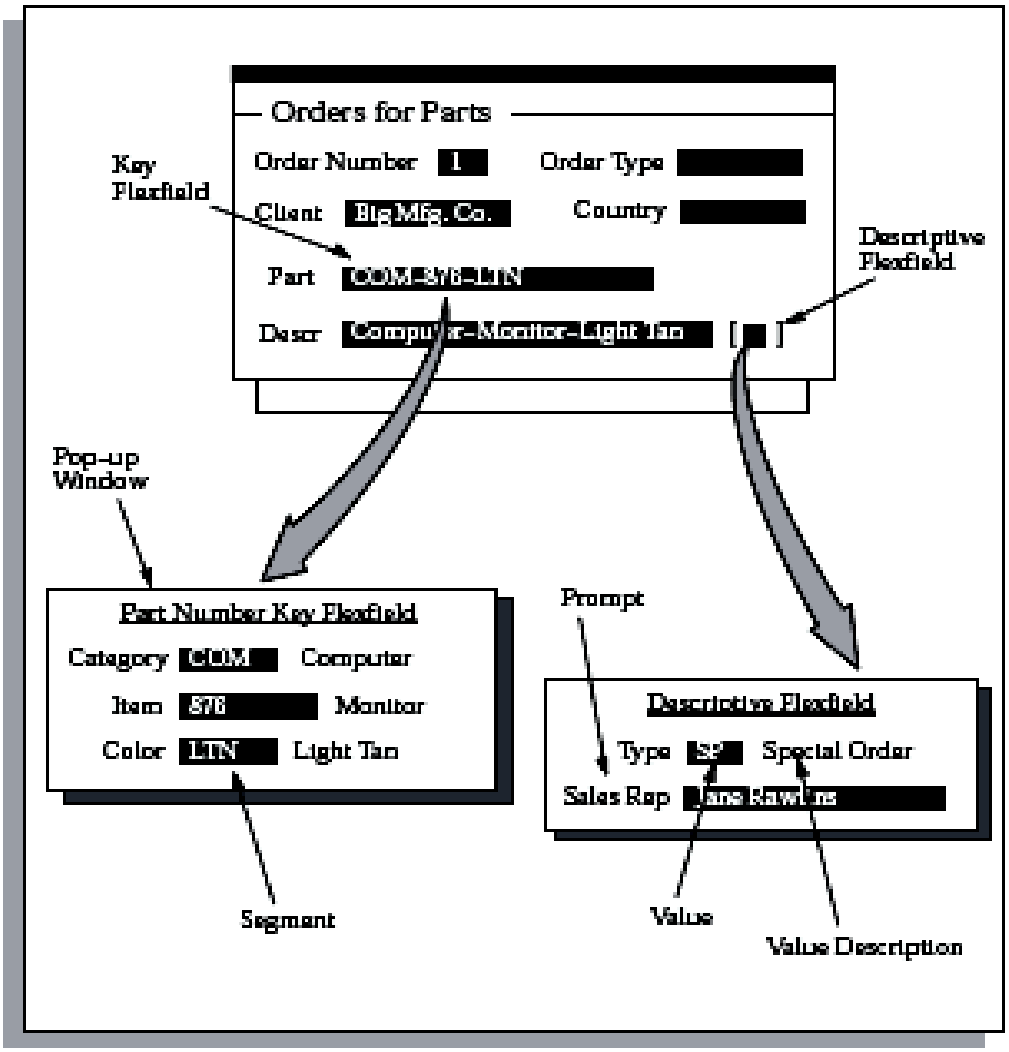
Table Name	Storage Purpose
FND_APPLICATION	New Module Information
FND_FORM	Forms Information
FND_FORM_FUNCTIONS	Information of Functions of a Form
FND_MENU	Menu Information
FND_MENU_ENTRIES	List of Functions Assigned to the Menu Information
FND_DATAGROUP	Data Group Information
FND_DATAGROUP_UNIT	List of Modules Assigned to the Data Group
FND_RESPONSIBILITY	All Responsibility Information
FND_RESP_GROUPS	All responsibility groups information
FND_TABLES	All Tables Information
FND_COLUMNS	All Table Columns Information
FND_USER	All Users Information
FND_USER_RESP_GROUPS	All Users Responsibility Information
FND_ID_FLEXS	All Key Flex Fields Information
FND_ID_FLEX_STRUCTURES	All Structures Information
FND_ID_FLEX_SEGMENTS	All the Segments Information
FND_FLEX_VALUE_SETS	Each Segment's Value Set Information
FND_FLEX_VALUES	Each Value Codes of a Value Set of a Segment
FND_FLEX_VALUE_TL	Each Value Description of a Value Set of a Segment

8. FLEXFIELDS

A flexfield is a field made up of sub–fields, or segments. There are two types of flexfields: key flexfields and descriptive flexfields. A key flexfield appears on your form as a normal text field with an appropriate prompt. A descriptive flexfield appears on your form as a two–character–wide text field with square brackets [ ] as its prompt.
Flex Fields are used to capture the Business Information of the Organization. The organization can use the Flex Fields based on their Business Structure.

KEY FLEX FIELDS (KFF)

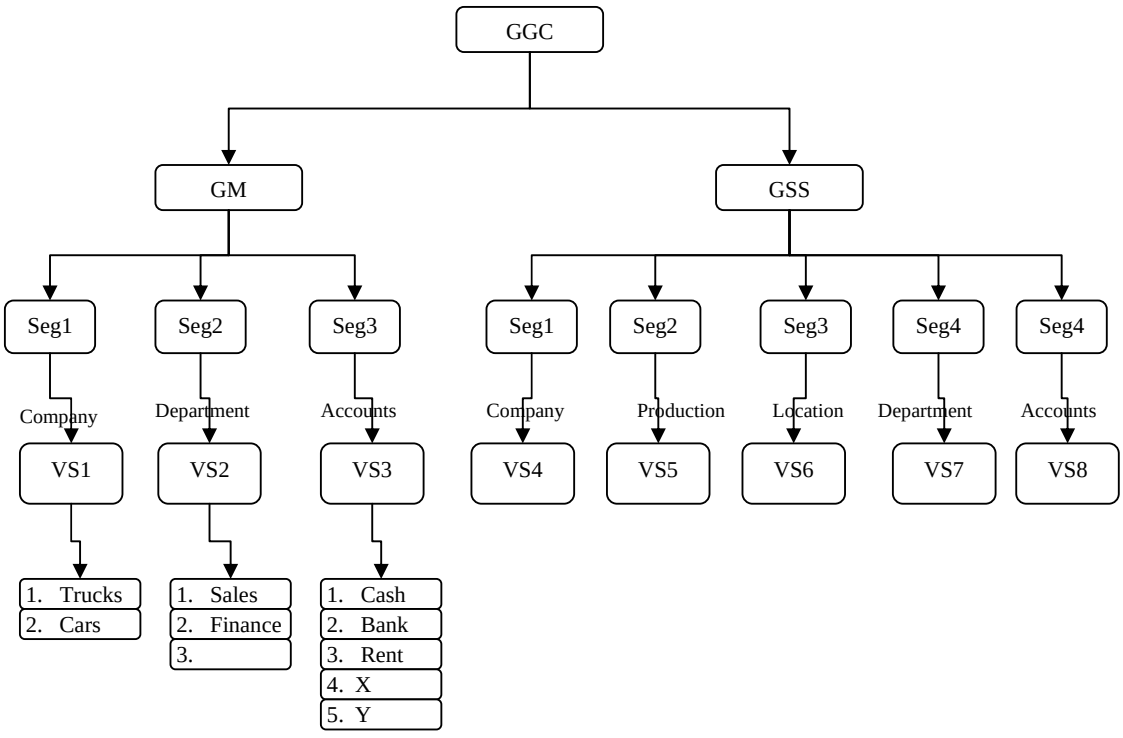
KFF are used to capture mandatory or Key Business information of the Organization. Each Key Flex Field is having its won base Table.



Q: How many KFF are supplied with the Package?
Ans: There are 31 KFF.

S.No	Type of Flex Field	Module	Table
1	Accounting FF	GL	GL_CODE_COMBINATION
2	Job FF	HR	PER_JOB_DEFINITIONS
3	Position FF	HR	PER_POSITION
4	Grade FF	HR	PER_GRADE
5	Location FF	FA	FA_LOCATION
6	Asset FF	FA	FA_KEYWORDS
7	Sales Tax FF	AR	AR_SALEX_TAX
8	Territory FF	AR	AR_TERRITORY
9	System Items FF	INV	MTL_SYSTEM_ITEM
10	Item Category FF	INV	MTL_ITEM_CATEGORY
…	….		
31			

Accounting Company KFF Structure



Code Combination ID of KFF is GGC_GM_VS1_1

GGC - Company Name
GM - Branch Name
VS1 - Segment 1
1 - Trucks

Value Set: is a collection of properties like Length, Data Type, Minimum Value, Maximum Value, Alignment and Value Validation etc.

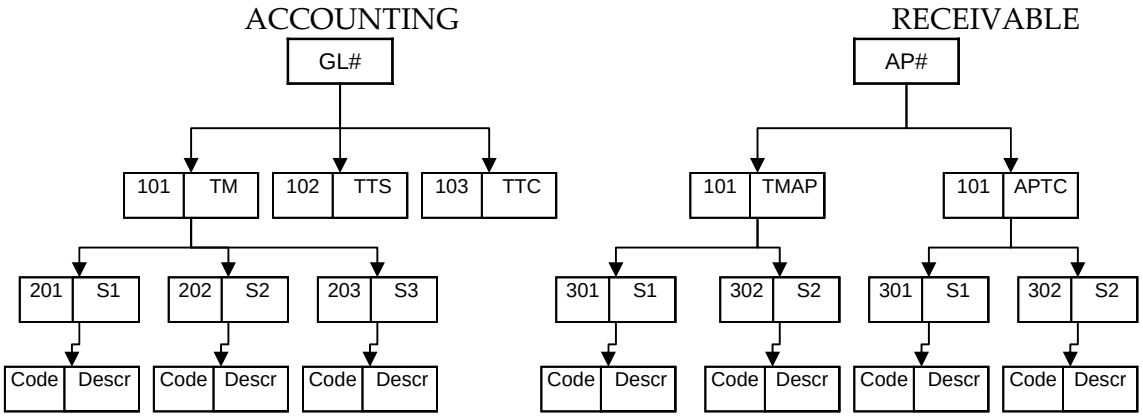
Flexfields consists of Structures
Structures consists of Segments
Segments consists of Value Set
Value Set consists of Parameters.

FND_ID_FLEXS Table captures the information of all the Key FlexFields.

APPL_ID - Column consists of Application ID
ID_FLEX_CODE - Column KFF Code

Example of APPL_ID: AR, AP, GL etc.
ID_FLEX_CODE: AR#, AP#, GL#, etc.

STRUCTURES



FND_ID_FLEX_STRUCTURES Table captures the information of all the structures.

Each Structure is uniquely identified by
APPLICATION_ID (Module Code),
ID_FLEX_CODE (Code of KFF)
ID_FLEX_NUM (Number of a Structure)

FND_ID_FLEX_SEGMENTS Table captures the information of Segments.

Each Segment is Uniquely identified by
APPL_ID
ID_FLEX_CODE
ID_FLEX_NUM
SEG_NUM
FLEX_VALUE_SET_ID

FND_FLEX_VALUE_SETS Table captures the information of each Segment's Value Set.

Each Value Set is Uniquely identified by FLEX_VALUE_SET_ID as Foreign Key of FND_ID_FLEX_SEGMENTS Table.

FND_FLEX_VALUES Table captures the information each Value codes of a Value Set of a Segment.

Each Value Code is uniquely identified by
FLEX_VALUE_SET_ID
FLEX_VALUE_ID

FND_FLEX_VALUE_TL Table captures the information of each Value Description of a Value Set of a Segment.

Each Value Description is uniquely identified by
FLEX_VALUE_ID

If we accept a KFF in the form, the information will be stored first in GL_CODE_COMBINATION Table (i.e. Mater Table) then in Transaction Table i.e. Details table).

Form to Accept KFF

Responsibility: R1 (TM) / R2 (TCS) Journal Form Journal: Date: Currency:			
Form	Application	User Form Name	Description
Name1	Function1		
Name2	Function2		

GL_CODE_COMBINATION Table (Master Table)

S1	S2	S3	S4	S5	….	Structure ID	CCID
1	1	1				TM	2001

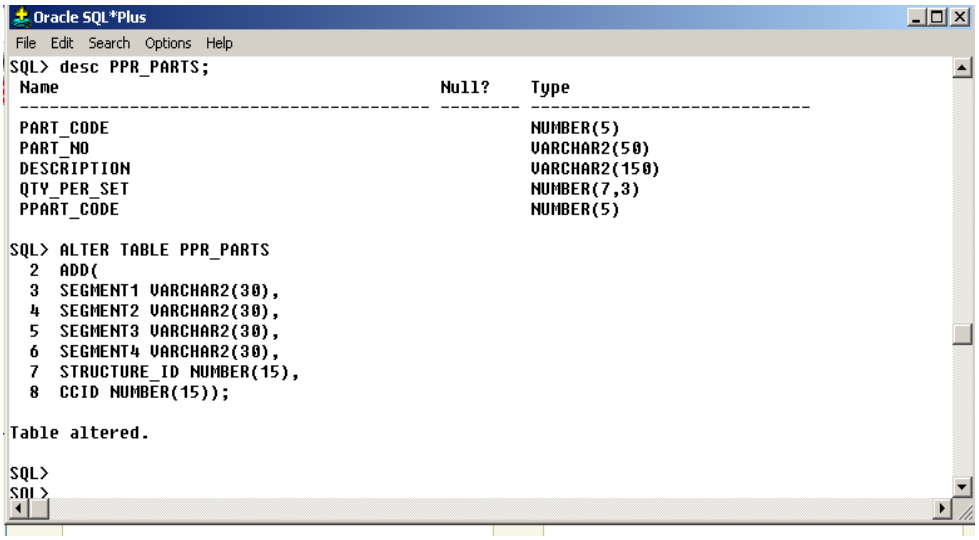
Transaction Table (Details Table)

SN	Date	Debt	Credit	…	….	CCID
1	1	1				2001

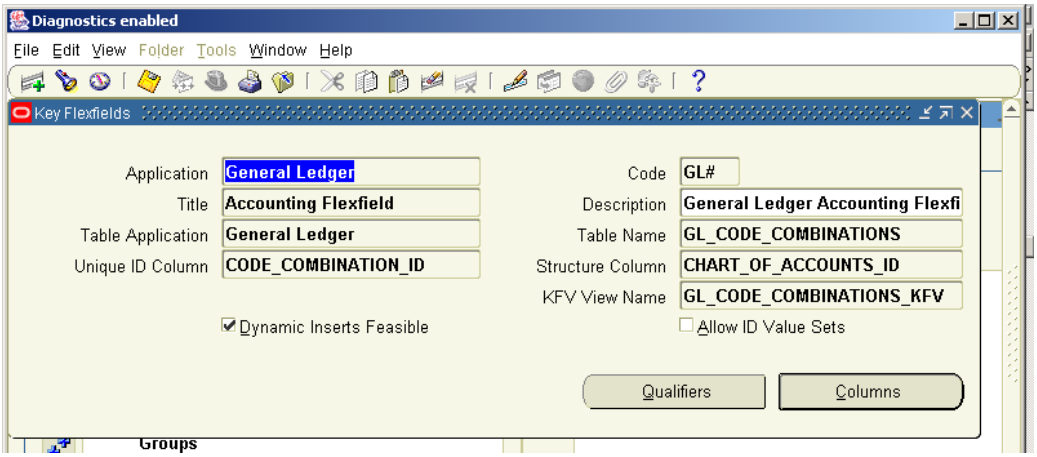
This Code is generated by DataBase

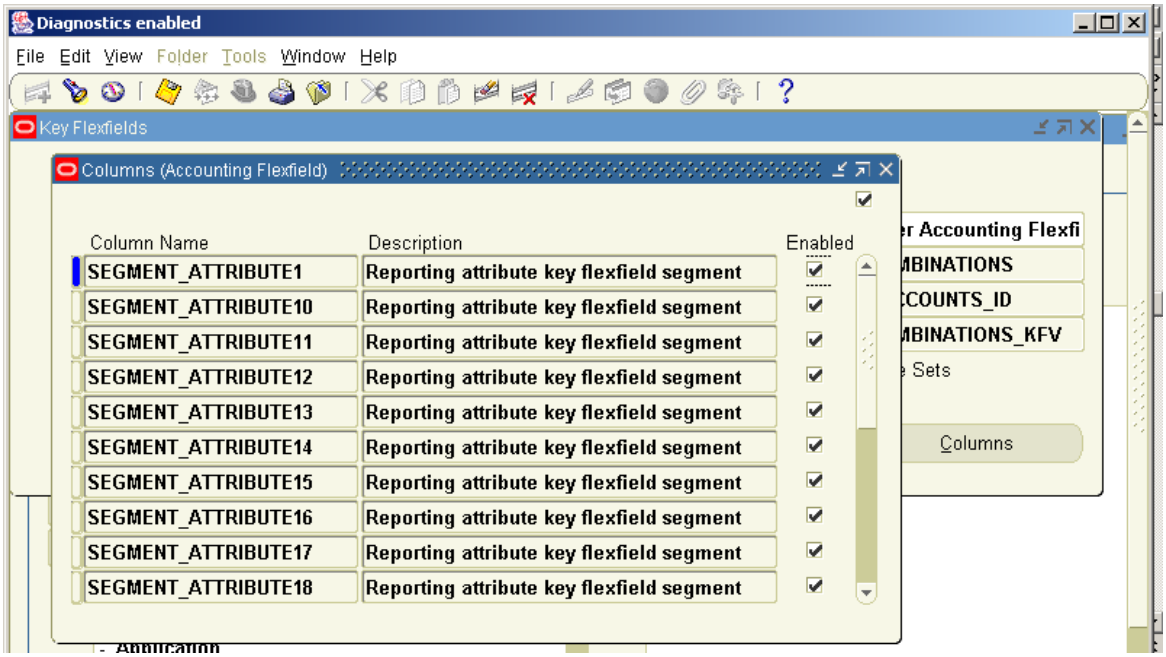
Steps required to registration of New Key Flex field (KFF)

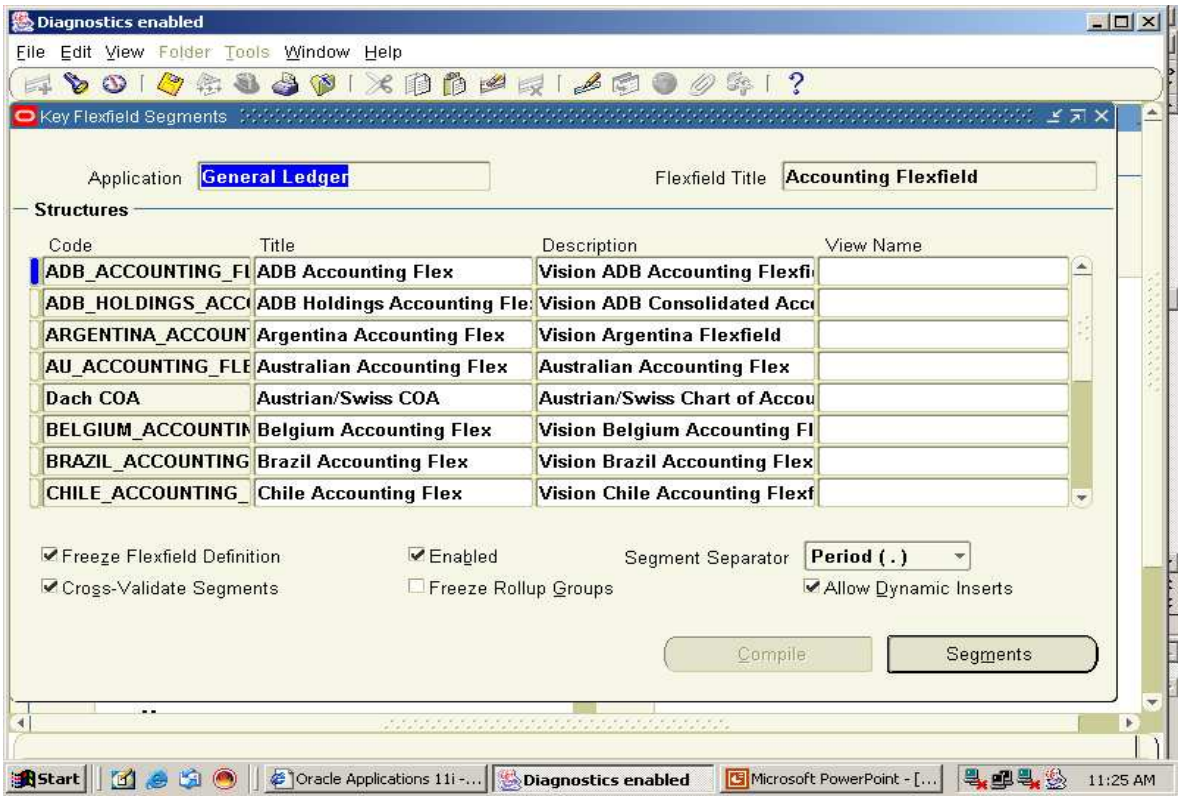
1. **Create a KFF Table in Module Specific Schema.**
   Connect to PPRS/PPRS@VIS

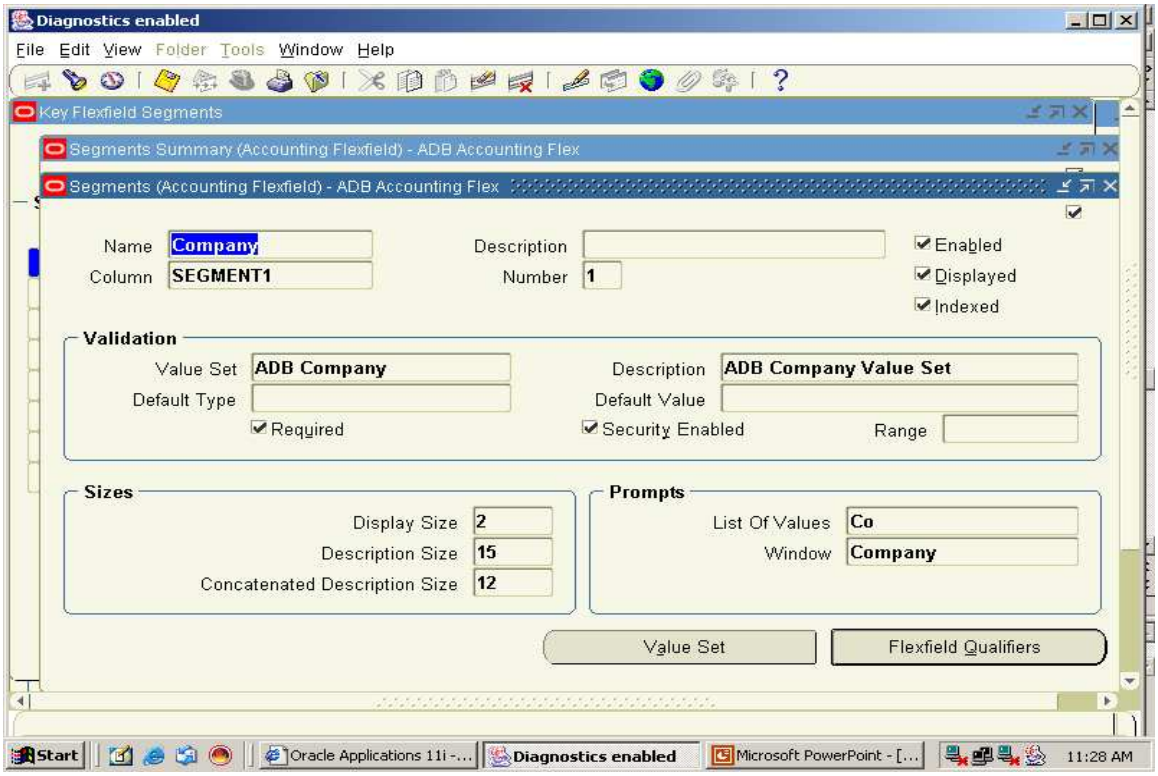


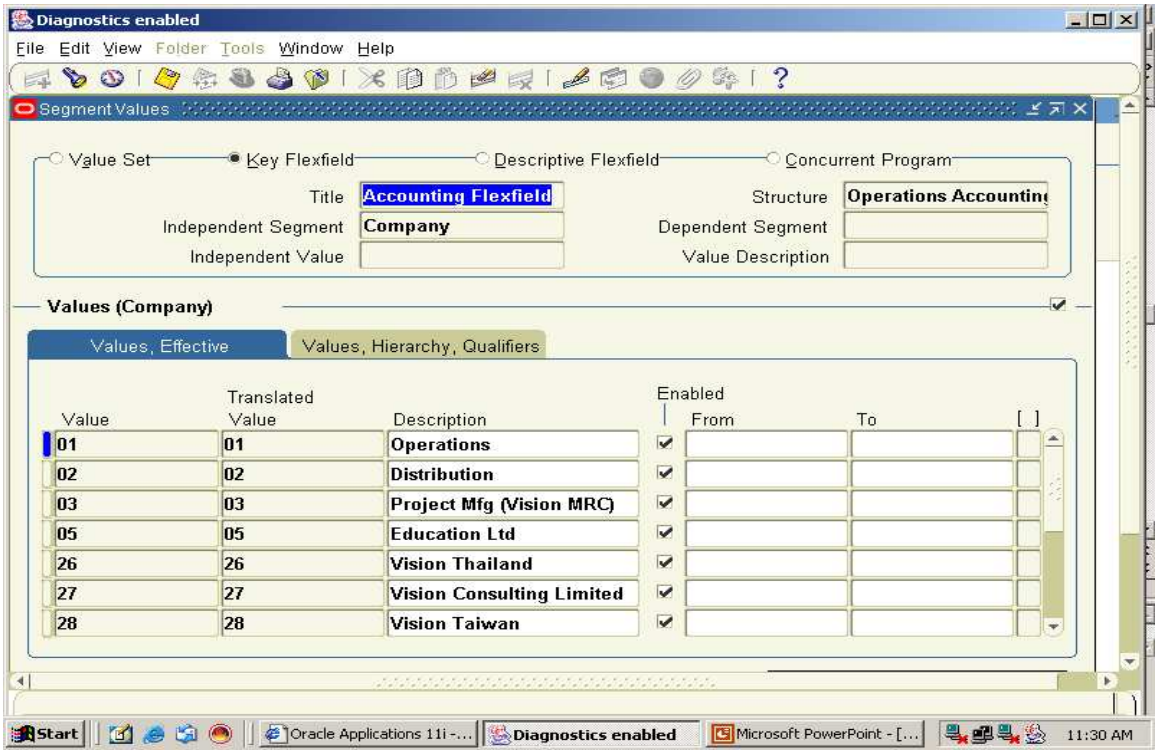
2. **Create a Public Synonym in APPS Schema.**
   Connect to APPS/APPS@VIS
   SQL> CREATE PUBLIC SYNONYM PPRS_PARTS FOR PPRS_PARTS;
3. **Register the Table with AOL Module.**
   PPRS_PARTS Table already registered.
4. **Register the KFF with AOL Module.**
   Connect to APPSTECH/APPSTECH
   Go to FlexField → Key → Registration

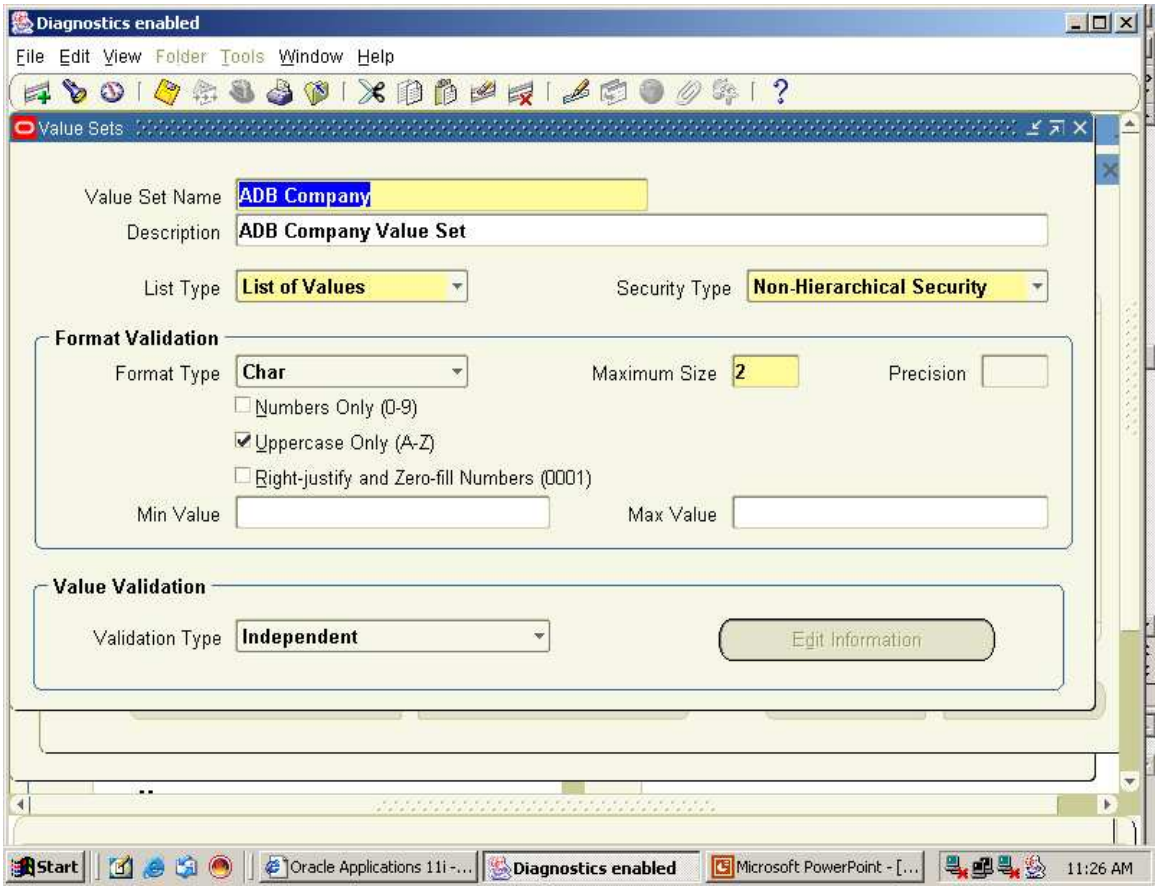



Diagnostics enabled
File Edit View Folder Tools Window Help
Key Flexfields
Columns (Accounting Flexfield)
Column Name
Description
Enabled
SEGMENT_ATTRIBUTE1
SEGMENT_ATTRIBUTE10
SEGMENT_ATTRIBUTE11
SEGMENT_ATTRIBUTE12
SEGMENT_ATTRIBUTE13
SEGMENT_ATTRIBUTE14
SEGMENT_ATTRIBUTE15
SEGMENT_ATTRIBUTE16
SEGMENT_ATTRIBUTE17
SEGMENT_ATTRIBUTE18
Reporting attribute key flexfield segment
Columns


Diagnostics enabled
File Edit View Folder Tools Window Help
Key Flexfield Segments
Application
General Ledger
Flexfield Title
Accounting Flexfield
Structures
Code
Title
Description
View Name
ADB_ACCOUNTING_FI
ADB Accounting Flex
Vision ADB Accounting Flexfi
ADB_HOLDINGS_ACC
ADB Holdings Accounting Fle
Vision ADB Consolidated Acc
ARGENTINA_ACCOUN
Argentina Accounting Flex
Vision Argentina Flexfield
AU_ACCOUNTING_FLE
Australian Accounting Flex
Australian Accounting Flex
Dach COA
Austrian/Swiss COA
Austrian/Swiss Chart of Accou
BELGIUM_ACCOUNTIN
Belgium Accounting Flex
Vision Belgium Accounting Fl
BRAZIL_ACCOUNTING
Brazil Accounting Flex
Vision Brazil Accounting Flex
CHILE_ACCOUNTING_
Chile Accounting Flex
Vision Chile Accounting Flexf
Freeze Flexfield Definition
Enabled
Segment Separator
Period (.)
Cross-Validate Segments
Freeze Rollup Groups
Allow Dynamic Inserts
Compile
Segments
Start
Oracle Applications 11i -...
Diagnostics enabled
Microsoft PowerPoint - [...
11:25 AM


Diagnostics enabled
File Edit View Folder Tools Window Help
Key Flexfield Segments
Segments Summary (Accounting Flexfield) - ADB Accounting Flex
Segments (Accounting Flexfield) - ADB Accounting Flex
Name Company
Column SEGMENT1
Description
Number 1
Enabled
Displayed
Indexed
Validation
Value Set ADB Company
Default Type
Required
Description ADB Company Value Set
Default Value
Security Enabled
Range
Sizes
Display Size 2
Description Size 15
Concatenated Description Size 12
Prompts
List Of Values Co
Window Company
Value Set
Flexfield Qualifiers
Start
Oracle Applications 11i -...
Diagnostics enabled
Microsoft PowerPoint - [...
11:28 AM


Diagnostics enabled
File Edit View Folder Tools Window Help
Segment Values
Value Set
Key Flexfield
Descriptive Flexfield
Concurrent Program
Title Accounting Flexfield
Independent Segment Company
Independent Value
Structure Operations Accountin
Dependent Segment
Value Description
Values (Company)
Values, Effective
Values, Hierarchy, Qualifiers
Value
Translated Value
Description
Enabled
From
To
01 01 Operations
02 02 Distribution
03 03 Project Mfg (Vision MRC)
05 05 Education Ltd
26 26 Vision Thailand
27 27 Vision Consulting Limited
28 28 Vision Taiwan
Start
Oracle Applications 11i -...
Diagnostics enabled
Microsoft PowerPoint - [...
11:30 AM


Diagnostics enabled
File Edit View Folder Tools Window Help
Value Sets
Value Set Name
ADB Company
Description
ADB Company Value Set
List Type
List of Values
Security Type
Non-Hierarchical Security
Format Validation
Format Type
Char
Maximum Size
2
Precision
Numbers Only (0-9)
Uppercase Only (A-Z)
Right-justify and Zero-fill Numbers (0001)
Min Value
Max Value
Value Validation
Validation Type
Independent
Edit Information
Start
Oracle Applications 11i -...
Diagnostics enabled
Microsoft PowerPoint - [...
11:26 AM

DESCRIPTIVE FLEX FIELDS (DFF)

DFF are used to capture the additional or extra Business information of the organization. DFF are used to add extra accounts, these are varry from one business to another business. All DFF columns are defined as Attribute Columns. All these columns are defined in the transaction table itself. There are around 5000+ DFF available with the package.

DFF have two different types of Segments

1. Global DFF
2. Context Sensitive DFF

Q: How to know that a form is having DFF?

Ans: Go to Help in Menu Bar
Diagnostic
Examine
Enter Oracle Password : APPS

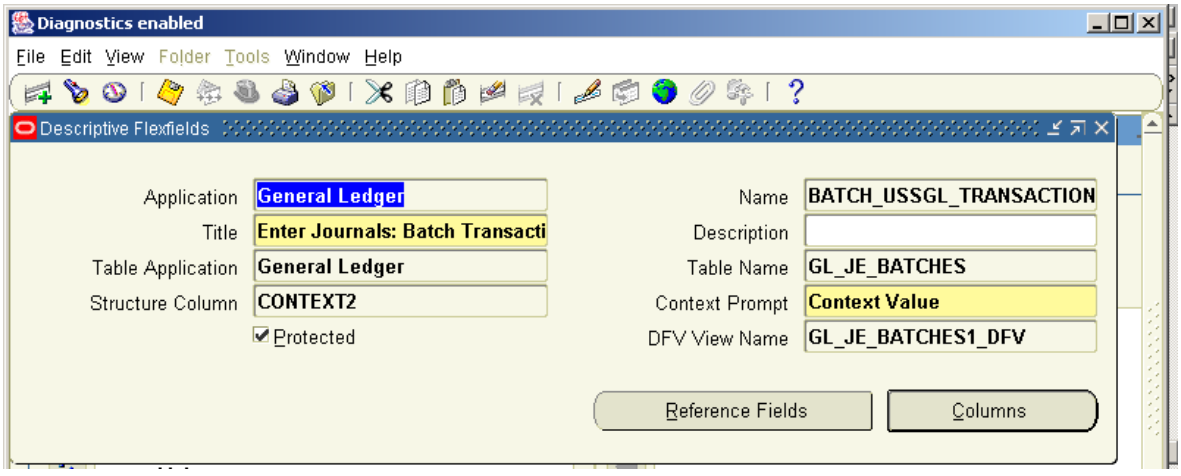
Block Name: $Descriptive Flexfields (Change the field like this))

Field: PPRS FLEX

If the Form having the button with square brackets [ ] then we can say that the form is having DFF. A form can have multiple DFF but a Block can have only one DFF. To reference a DFF use Block_Name.DFF_Name.

Q: How to enable a DFF?

To enable DFF switch responsibility to
Application Developer
FlexField
Descriptive
Segments
Enter Totle: PPRS DFF
Click on Segment Button at the bottom of the Pop-Up Window.



Example:

Table T1

Column1
Column2
Column3
Column4
…..
Attributes, A1..A20
Attribute Category or Context Column AC1, Ac2, Ac3, AC4 ….

FORM

Block1
Column1 → DFF1 → AC1 → Attribute A1 .. A5

Block2
Column2 → DFF2 → AC2 → Attribute A6 .. A15

Block3
Column3 → DFF3 → AC3 → Attribute A16 .. A20

We can register more than one DFF on a table. How many number of DFF that we want to register on a table, we must have those many number of attribute categories or Context columns are to be used to group a set of attribute column. The attribute category and attribute columns mapped with on DFF can't be mapped with other DFF.

When we register a DFF, System internally generates the default Context field value as global data elements (GDE). The values for Attribute Category column or Context Column are null, when we are enabling the fields as global data elements.

C1 → AC1 → A1 .. A20 | Column | DFF |

Global Data Elements A1,A2

INDIA	(A3 .. A4)
UK	(A3 .. A5)
US	(A3 .. A6)

Country Values will be stored in Attribute Category Column (OR) Context Column. We can also define the field, which is enabled, as global data elements are common for all the user defined context field values. That is the reason the attribute columns, which are mapped with global data elements, are not mapped with user defined data elements.

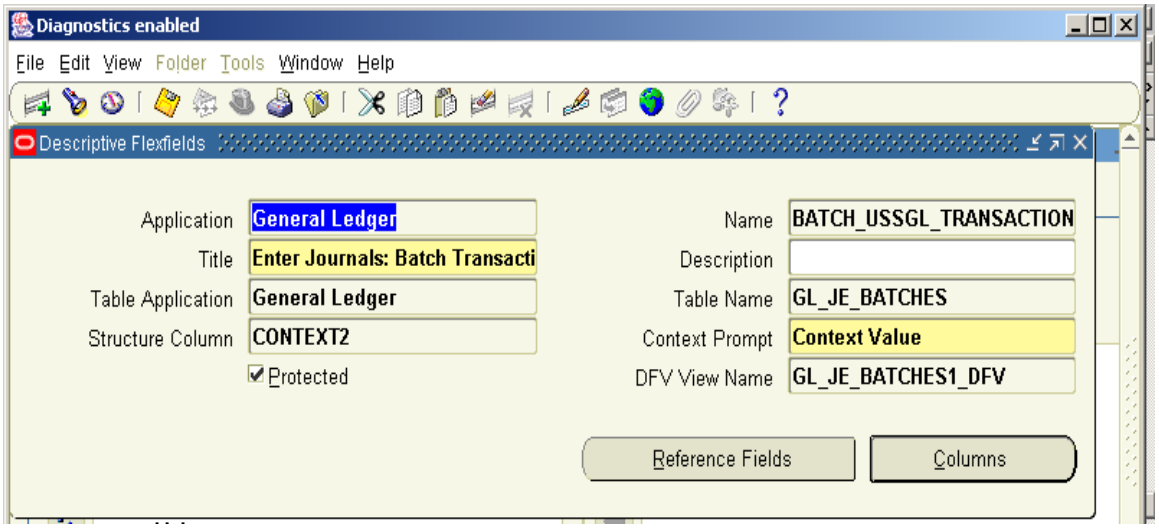
Steps required for registering a DFF

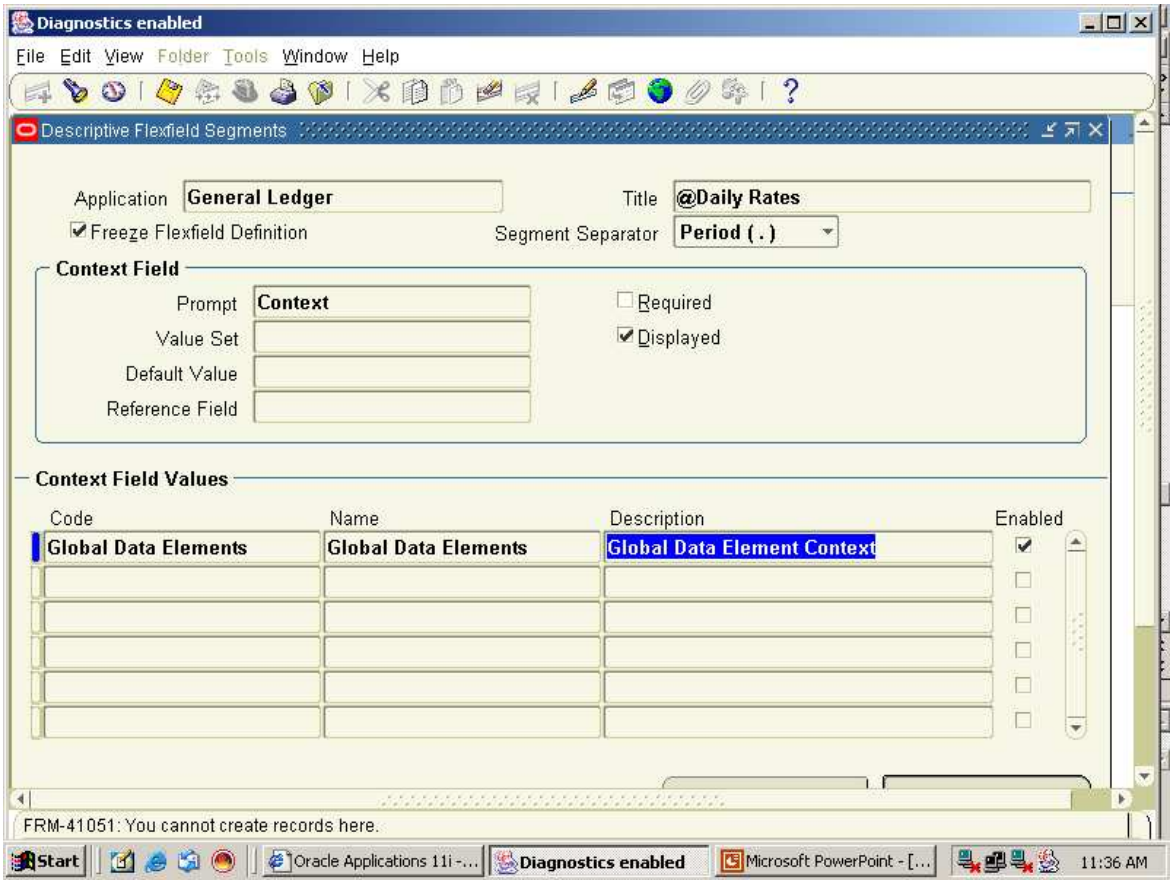
1. **Create a DFF Table in Module Specific Schema.**
   Connect to PPRS/PPRS@VIS
   SQL> ALTER TABEL PPRS_PARTS
   ADD (
   Attribute1 Varchar2(150),
   Attribute2 Varchar2(150),
   Attribute3 Varchar2(150),
   Attribute4 Varchar2(150),
   Context Varchar2(25));

2. **Create a Public Synonym in APPS Schema.**
   Connect to APPS/APPS@VIS
   SQL> CREATE PUBLIC SYNONYM PPRS_PARTS FOR PPRS_PARTS;

3. **Register the Table with AOL Module.**
   PPRS_PARTS Table already registered.

6. **Register the KFF with AOL Module.**
   Connect to APPSTECH/APPSTECH
   Go to FlexField
   Descriptive
   Register




Diagnostics enabled
File Edit View Folder Tools Window Help
Descriptive Flexfield Segments
Application General Ledger
Title @Daily Rates
Freeze Flexfield Definition
Segment Separator Period (.)
Context Field
Prompt Context
Value Set
Default Value
Reference Field
Required
Displayed
Context Field Values
Code
Name
Description
Enabled
Global Data Elements
Global Data Elements
Global Data Element Context
FRM-41051: You cannot create records here.
Start
Oracle Applications 11i -...
Diagnostics enabled
Microsoft PowerPoint - [...
11:36 AM

9. NEW FORM DEVELOPMENT

To enable the end user to operate the Forms user friendly we need to standardize the Forms. To design or develop the form with Applications Standards, Package provided a Standard Form called APPSTAND.fmb

APPSTAND.fmb: is a collection of the entire standard object Groups used in the Oracle Applications. Form should inherit the following object Group Properties

1. **STAND_PC_AND_VA:** This consists of Windows, Canvases, Blocks and Items used for Property Classes and Visual Attributes.

2. **STAND_TOOLBAR:** This consists of Windows, Canvases, Blocks and Items used for Tool Bars.

3. **STANDARD_CALENDER:** This consists of Windows, Canvases, Blocks and Items used for Calendars.

4. **STANDARD_FOLDER:** This consists of Windows, Canvases, Blocks and Items used for Data Restriction and Changing the Layout of the Form at run time. And this is not used for Custom Forms).

5. **QUERY_FIND:** This consists of Windows, Canvases, Blocks and Items used for implementing find windows or Search methods.

6. **LIBRARIES:**

APPCORE.PLL: is the collection of Procedures, Functions and Packages used to define Menus and Tool Bars.

FNDSQF.PLL: is the collection of Procedures, Functions and Packages used to define Special Objects like KFF, DFF, Form Functions and Non-Form functions.

APPDAYPK.PLL: is the collection of Procedures, Functions and Packages used to define Calendars.

CUSTOM.PLL: is the collection of Procedures, Functions and Packages used to customize the Forms with out modifying the code of Oracle Applications.

Note: The Built-ins of CUSTOM.PLL are internally called by the Built-ins of APPCORE.PLL and the Built-ins of APPCORE.PLL are called by few Triggers like

WHEN-NEW-FORM-INSTANCE,
WHEN-NEW-RECORD-INSTANCE,
WHEN-NEW-ITEM-INSTANCE,
PRE-BLOCK,
PRE-RECORD,
PRE-ITEM

APPCORE2.PLL: is a Duplicate/Replica of APPCORE.PLL. It has the same Built-ins as APPCORE.PLL and this is used in CUSTOME.PLL

GLOBE.PLL: is used to change the regional settings of the Package with out modifying the code of Applications. GLOBE.PLL internally calls the Built-ins of JE.PLL, JL.PLL and JA.PLL Libraries.

JE.PLL: is used to change the regional settings of Middle East Countries.

JA.PLL: is used to change the regional settings of Asia Pacific Countries.

JL.PLL: is used to change the regional settings of Latin America.

FORM TEMPLATE: It is used to develop the new Form of a Business Application.

1. TEMPLATE.fmb inherits all the Object Groups of APPSTAND.fmb.

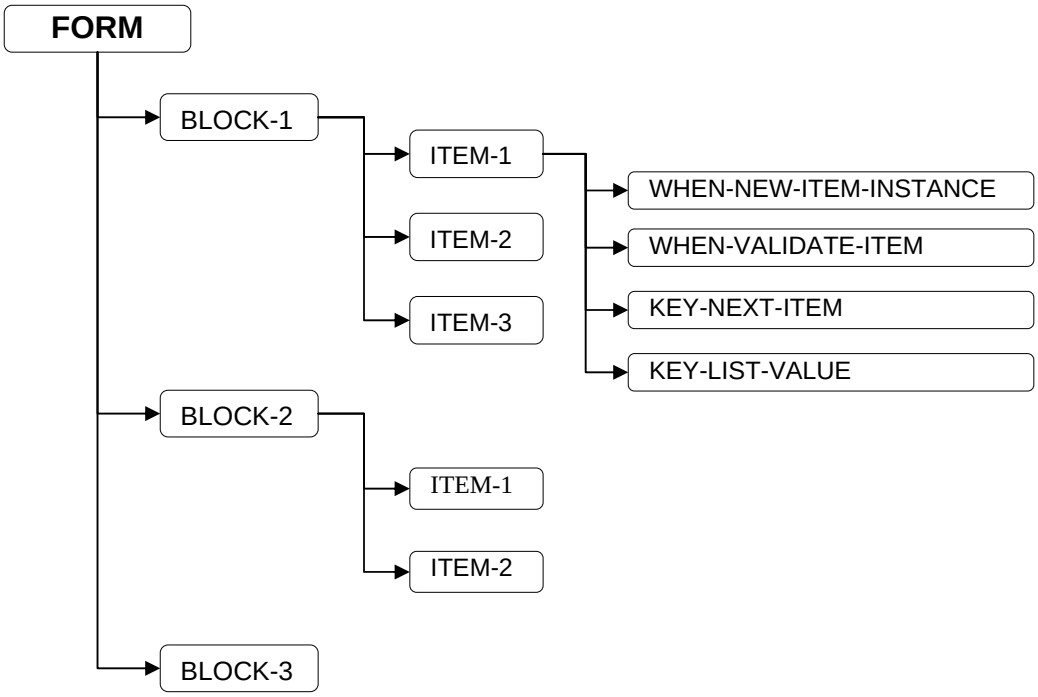
2. Attaches all the Libraries.

3. Attaches Standard Menu called FNDMENU.mmb.

4. Creates Form level Triggers. Some of these Triggers are read only and some are read and write. Read only triggers are marked with RED Colour and Read and Write triggers are marked with BLUE Colour.

5. APPS_CUSTOM Package is a collection of OPEN_WINDOW(WND IN VARCHAR2) and OPEN_WINDOW(WND IN VARCHAR2) Procedures. These are created by default when we open the TEMPLATE.fmb Form.

Example: If we want to open a window in the following structure.



Write the following code in the OPEN_WINDOW Procedure of the APP_CUSTOM Package to open the navigation of the first window and subsequent windows, which is called by the PRE-FORM trigger.

```
BEGIN
   IF (WND = 'W1') THEN
            Statement 1;
            Statement 2;
  ELSIF (WND = 'W2') THEN
            Statement 1;
            Statement 2;
    ELSIF (WND = 'W3') THEN
            ----
            ----
    ELSE
            -----
            -----
    END IF;
```

6. Creates Dummy BLOCKS with name Block Name, Detail Name, Dummy CANVAS with the Name Block Name and Dummy WINDOW with the name Block Name. Purpose of these dummies is only to understand the creation of Blocks, Canvases and Windows in a Form.

Naming Conventions to be followed in the Forms Development

1. Name of the Form Module should be same as .fmb.
2. Name of the Data Block should be same as Base Table Name.
3. Name of the Canvas and Window is same as Block name.
4. Name of the LOV should be same as Item Name.
5. Name of the Package should be same as Block Name.
6. Name of the Procedure should be same as Item Name.
7. Name of the ROW LOV should be same as Block_Name_QF.
8. Name of the FND_WINDOW should be same as Block_Name_QF.

Steps Required to Developing a New Form

1. Open the TEMPLATE.fmb Form from Form Builder and Save as with Module specific form name.
2. Delete the default Blocks with names BLOCK_NAME and DETAIL_BLOCK, Canvas with name BLOCK_NAME and Window with name BLOCK_NAME.
3. Create a new Window and assign Window property class to it.
4. Create Canvas and assign Canvas property class to it.
5. Assign Canvas to Window and Window to Canvas.
6. Create Data Block using wizard and name it as Table Name. Assign Text Item Property class to all the Items in the Data Block.
7. Modify Pre-Form Trigger at Form Level.
8. Modify APP_CUSTOM Package.
9. Change Form Module Level Properties.
10. Save and Compile the Form and copy the .FMX file in the Module specific directory.
11. Perform all the pre requisites to register the Form.
    a. Register the form with AOL module.
    b. Define a form function
    c. Assign form function to the Menu.
    d. Assign the menu to the Responsibility.
    e. Assign Responsibility to the User.

Note: Check the following attachments when you open the TEMPLATE.fmb Form.

1. Check the required Libraries attached or not.
2. Check the required Object Groups attached or not.
3. Check the required Visual Attributes attached or not.
4. Check the required Property Classes attached or not.
5. Check the required Form Level Triggers created or not.

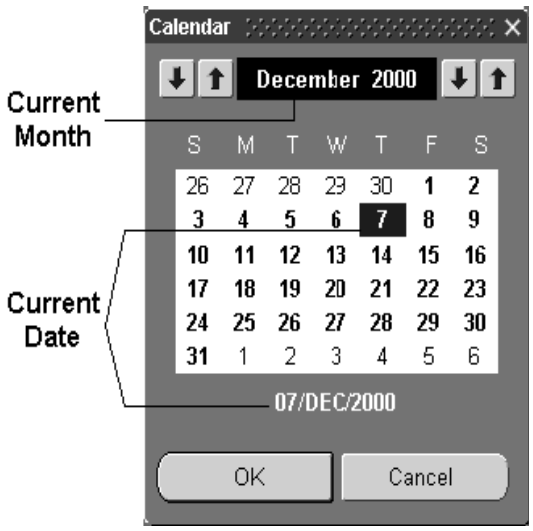
10. DEFINING CALENDARS

Calendars are special objects are used to select the date value in user-friendly manner. When you navigate to a date field it displays an LOV icon.

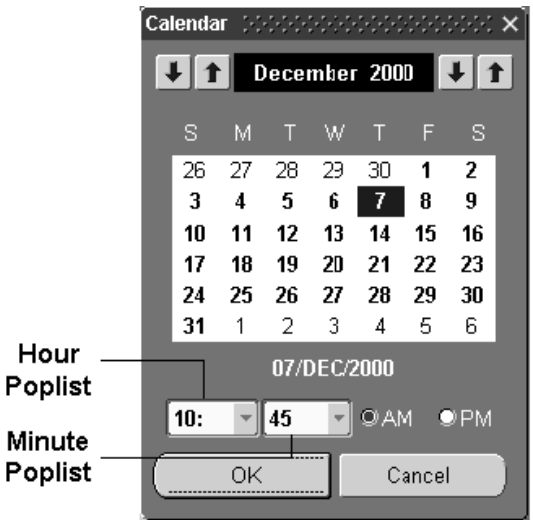
The Calendar lets you satisfy the following business needs:

- Represent dates and time graphically so that you can make a quick date and time selection without additional typing.
- Select only those dates or date ranges that are appropriate for the current field.

You can use the Calendar window, as shown in Figure 3 – 1, to help you enter a date.



Calendar for Date Selection



Calendar for Date and time Selection

Steps required to defining the Calendar

1. Change the following properties of the date item

a. Change the Sub-Class to Text_Item_Date property Class.
b. LOV Enable List Lamp
c. Validate from list property set to NO.

Note: Enable List Lamp is a dummy LOV, which is going to come with TEMPLATE.fmb. The record Group associated with this LOV is SELECT NULL FROM DUAL; we can use this LOV whenever we want to define some Special Objects on the Text Item.

2. **Create KEY-LISTVAL Trigger at Item Level.**
   Date:

   Change the Properties of date field as
   1. Text_Item_Date
   2. Enable_List_Lamp
   3. Key_Listval Trigger

   Write Code: CALENDAR.SHOW;
3. **Save and Compile the Form.**
4. **Perform all the Form registration process**.

11. WHO COLUMNS

WHO columns are used to track the information updated or inserted by the users against the tables. FND_STANDARD package is used for this purpose. FND_STANDARD.SET_WHO Procedure is used to update the WHO columns in a Table when a DML operation s (i.e. INSERT, UPDATE) performed.

Steps required to Track WHO Information in our form

1. **ALTER the table by adding WHO columns in the table.**
2. **Open TEMPLATE.fmb in form Builder.**
3. **Perform all the pre-requisites to develop a form.**
4. **Include all the WHO columns in the Data Block with NULL Canvas.**
5. **Call FND_STAND.SET_WHO procedure in Pre-Insert and Pre-Update triggers at Block Level.**
6. **Save and Compile the Form.**
7. **Perform all the pre-requisites to register the form**

Note: 1. WHO Columns need not register with the table registration option.

2. Table registration will be done only for KFF and DFF columns.

3. For Form development table registrations not required.

To Show the Module Form Directory Path

1. In PRE-FORM Trigger change the module name from FND to specific module short name (i.e. EX, INC, PPR, etc.,).
2. Save and Compile the form.

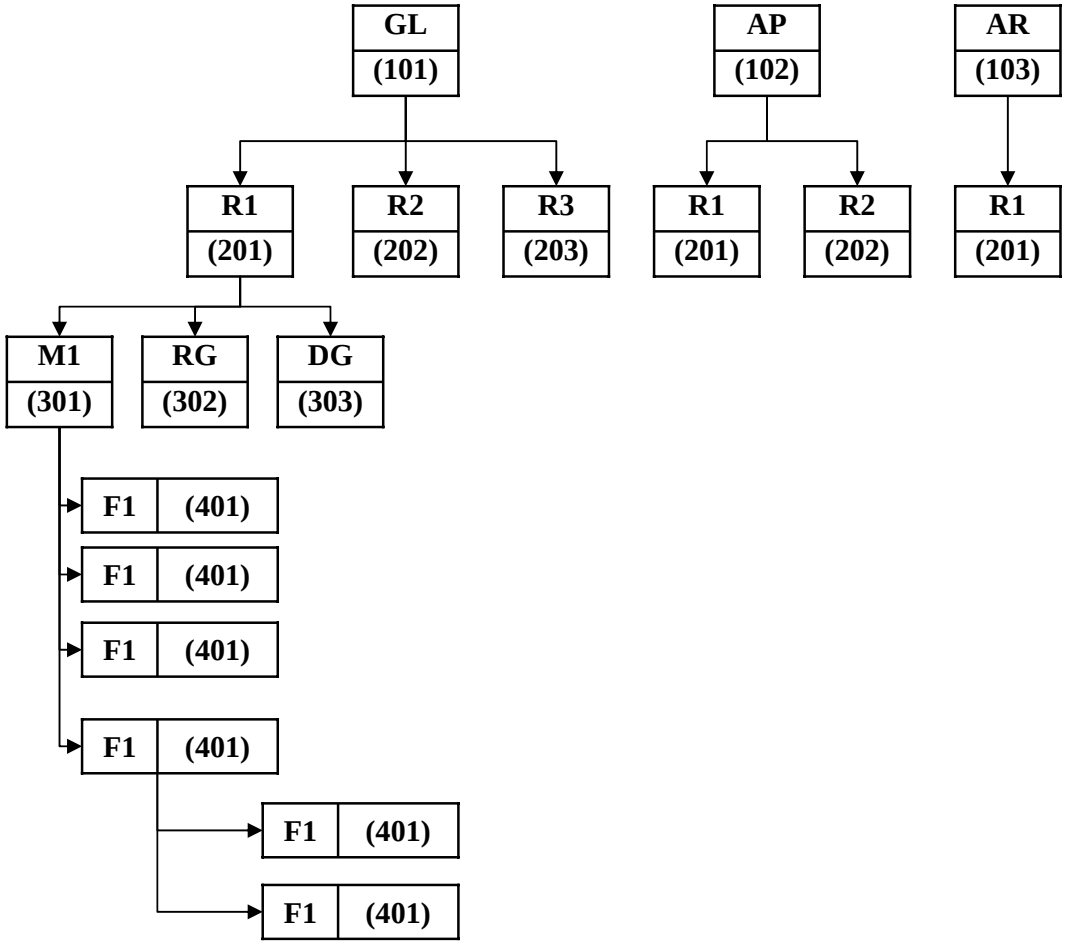
12. NON – FORM FUNCTIONS

Non-form functions (sub functions) are also known as Layout Items. Non-form functions allow access to a particular subset of form functionality from this menu.

Q: How the function security works?

Ans: Whenever we select responsibility information, it selects the Responsibility_ID, Menu_ID, Request_Group_ID and Data_Group_ID values. Using MENU_ID identifies Menu information in FND_MENUES. Retrieves all its Menu_ID entries from FND_MENU_ENTRIES. Before displaying the prompt of these entries in the navigator Window application checks whether any functions and sub-menus excluded for the above responsibility along with the Application_ID in FND_RESP_FUNCTIONS. If any entries found then it suppresses the display of the prompts of functions and sub-menus in the navigator window.

STRUCTURES



Details of the above Structure

GL, AR, AP	are Application Users information stored in FND_USERS
101, 102, 103	are Application_IDs information stored in FND_APPLICATIONS
R1, R2, R3	are Responsibilities information stored in FND_RESPONSIBILITY
201, 202, 203	are Responsibility_IDs information stored in FND_RESP_GROUPS
M1, RG, DG	are Menus information stored in FND_MENUES
301, 302, 303	are Menu_ID information stored in FND_MENU_ENTRIES
F1, F2, F3, M2	information stored in FND_FORM_FUNCTIONS

Steps Required For Coding Non-Form functions

1. **Open TEMPLATE.fmb in Form Builder.**
2. **Perform all the pre-requisites to develop a Form.**
3. **Create the PUSH BUTTON in the data Block and change the following properties.**

   Name : BOOK
   Sub Class : BUTTON
   Label : BOOK
   Canvas : Canvas Name

4. **Create a Non-Form Function for this Layout Item in AOL Module.**
5. **Assign this Non-Form Function to the Menu leaving the prompt empty.**
6. **Modify PRE-FORM Trigger at Form Level.**

```
IF FND_FUNCTION.TEST('NON_FORM_FUNCTION_NAME') THEN
   /* Retrieves Function ID */
   APP_ITEM_PROPERTY.SET_PROPERTY('EX_ORDERS.BOOK',
    'ENABLED','PROPERTY_ON');
ELSE
    APP_ITEM_PROPERTY.SET_PROPERTY('EX_ORDERS.BOOK',
    'ENABLED','PROPERTY_OFF');
END IF;
```

7. **Save and Compile the Form.**
8. **Perform all the pre-requisites to register the form.**

13. SEARCH METHODS

There are two search methods available with APPS Package

1. **ROW LOV** Type Search Method.
2. **FIND WINDOW** Type Search Method.

To Open a Search Window

Go to Menu Bar View Find

1. ROW LOV

It is used to search a record based on Primary Key Column.

Steps Required to Implement ROW LOV

1. **Open Form in Form Builder.**
2. **Perform all the prerequisites to develop the form**
3. **Create a user defined Parameter and change the Data Type and width of the column, same as Primary Key Column.**
4. **Create a LOV using wizard and assign the return value of the LOV to the Parameter.**
5. **ROW LOV is based on Record Group.**
6. **SELECT Order_No, Customer FROM Ex_Orders;**

   P_Order_No (Parameter_Name)

   Look Up Return Item
7. **Create QUERY_FIND user defined Trigger at Block Level.**

   Note: LOV Name should be Block_Name_QF

   Trigger: QUERY_FIND (User Defined Trigger)

```
APPS_FIND.QUERY_FIND('LOV');
IF QUERY_FIND IS NOT NULL THEN
        -----
END IF;
IF G_QUERY_FIND := 'TRUE' THEN
        Execute_Query;
END IF;
```

8. **Create PRE_QUERY Trigger at Block Level.**

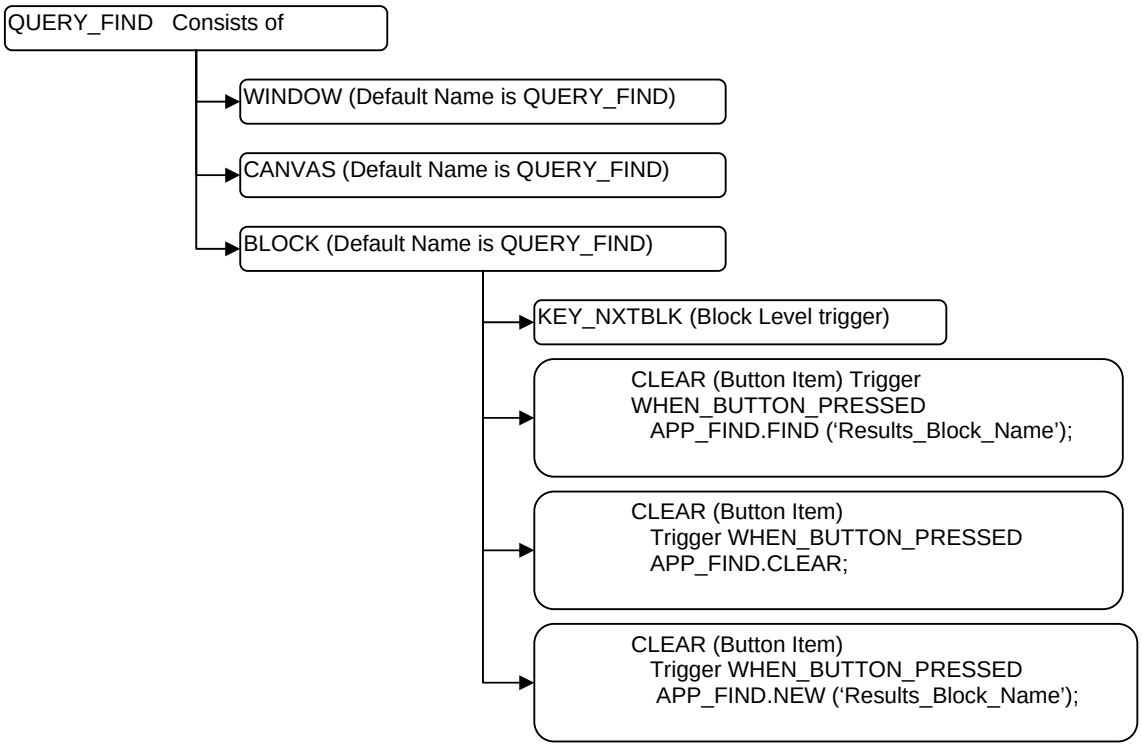
```
IF PARAMETER.G_QUERY_FIND = 'TRUE' THEN
    :EX_ORDERS.ORDER_NO := :PARAMETER.P_ORDER_NO;
    :PARAMETER.G_QUERY_FIND := 'FALSE';
END IF;
```

Note: In latest versions of Applications Parameter_Query_Find will be implicitly
assigned by the compiler after Query Execute. We need not assign it explicitly as FALSE.

10. **Save and Compile the form and copy .fmx file in to module specific forms directory.**

2. FIND WINDOWS

Query Find is used to search the matching records based on value entered in specified fields by the user. It consists of Window, Canvas, Block, Text Items and three default layout Items (i.e. CLEAR, NEW, FIND).



Steps Required To Create a FIND WINDOW

1 **Open TEMPLATE.fmb form in Form Builder.**

2 **Perform all the prerequisites to develop the form.**

3 **Open APPSTAND.fmb form in Form Builder.**

4 **Open QUERY_FIND Object Group from APPSTAND.fmb to TEMPLATE.fmb.**

5 **Rename QUERY_FIND Window, Canvas and Block names with the name of the Results Block name (i.e. EX_ORDERS) and QF suffix**

Note: Name of the QUERY_FIND looks like EX_ORDERS_QF.

6 **Modify the FIND_BLOCK by adding the Text Items as per the requirement.**

7 **Modify the coding in KEY-NXTBLK trigger at Block Level.**

8 **APP_FIND.FIND ('Results_Block_Name');**

9 **Modify the coding in the WHEN-BUTTON-PRESSED Trigger associated with the NEW Button**.

```
APP_FIND.FIND ('Results_Block_Name');
```

10 **Modify the coding in the WHEN-BUTTON-PRESSED Trigger associated with the FIND Button.**

```
APP_FIND.FIND ('Results_Block_Name');
```

11 **Create QUERY_FIND user defined trigger at Results Block Level.**

```
APP_FIND.FIND('Results_Window_Name', 'Find_Window_Name',
    'Find_Block_Name');
```

12 **When we press the FIND button in FIND_WINDOW, it will first evaluates G_QUERY_FIND = 'TRUE' then fires PRE-QUERY Trigger then Fetches the results based on matches then fires POST-QUERY Trigger.**

13 **Create PRE-QUERY Trigger at Results Block Level.**

```
IF :PARAMETER.G_QUERY_FIND = 'TRUE' THEN
      COPY(:FIND_BLOCK.ITEM_NAME1, 'RESULTS_BLOCK,ITEM_NAME1');
      COPY(:FIND_BLOCK.ITEM_NAME2, 'RESULTS_BLOCK,ITEM_NAME2');
      COPY(:FIND_BLOCK.ITEM_NAME3, 'RESULTS_BLOCK,ITEM_NAME3');
APP_FIND.QUERY_FIND (:FIND_BLOCK.START_DATE,
:FIND_BLOCK.END_DATE,'RESULTS_BLOCK.ORDER_DATE');
END IF;
```

14 **Save and Compile the form and copy .fmx file in to module specific forms directory.**

15 **Perform all the prerequisites to register the form.**

Note: APP_FIND.QUERY_RANGE (:Query_Find_Block_Name.START_DATE, :Query_Find_Block_Name.END_DATE, 'Results_Block_Name.Order_Date');

This code is to be written in PRE-QUERY Trigger at Results Block Level and is used to search the values based on the range of values.

Steps Required To Invoke Descriptive Flex Fields (DFF)

1 **Create a DFF Table in Module Specific Schema.**

2 **Create a Public Synonym in APPS Schema.**

3 **Register the Table with AOL Module.**

4 **Register DFF with AOL Module.**

5 **Open TEMPLATE.fmb Form in Form Builder.**

6 **Perform all the prerequisites to develop the form.**

7 **Include all the DFF columns in Data Block with NULL Canvas.**

8 **Create a Text Item in the Data Block and change the following properties.**

Name	:	DFF
SubClass	:	Text_Item_Desc_Felx
Database Item	:	NO
LOV	:	Enable_List_Lamp
Validate From List	:	NO
Canvas	:	Canvas_Name

9 **Create a Package Specification in Program Unit with a DFF Procedure and the name of the package is same as Block Name.**

```
PROCEDURE DFF (EVENT IN VARCHAR2);
```

10 **Create Package Body in Program Unit.**

```
PROCEDURE DFF (EVENT IN VARCHAR2) IS
        BEGIN
        IF EVENT = 'WHWN-NEW-FORM-INSTANCE' THEN
        FIND_DESCR_FLEX.DEFINE ('EX_ORDERS','DFF','EX','EXDFF');
        --- FIND_DESCR_FLEX.DEFINE ('Block_Name','Item_Name',
        --- 'Application_Short_Name','Name_Of_DFF');
        END IF;
END;
```

11 **Call the DFF procedure in WHWN-NEW-FORM-INSTANCE Trigger at Form Level and passing the event as WHEN-NEW-FORM-INSTANCE.**

12 **Call FND_FLEX.EVENT in WHEN-NEW-ITEM-INSTANCE Trigger at form level.**

Note: FND_FLEX.EVENT is a common Built-In used to invoke DFF and KFF.

13 **Save and Compile the form and copy .fmx file in to module specific forms directory.**

14 **Perform all the prerequisites to register the form.**

To Enable the DFF
Go to Flex Field
Descriptive
Segments

To see the Enable DFF
Go to Forms
Click on DFF Button
Open the DFF Window with Invoked Fields

Steps Required Invoke Key Flex Fields from the Form

1 **Create a KFF Table in Module Specific Schema.**
2 **Create a Public Synonym in APPS Schema.**
3 **Register the Table with AOL Module.**
4 **Register KFF with AOL Module.**
Note: When ever we register a KFF, System internally generates a Structure with the same name as Title of the KFF and the Structure_ID.
5 **Create the Structure of KFF.**
6 **Associate Segments to the Structure.**
7 **Define Value Sets.**
8 **Assign Value Sets to Segments.**
9 **Define Values to each Segment based on Value Sets.**
10 **Open TEMPLATE.fmb Form in Form Builder.**
11 **Perform all the prerequisites to develop the form.**
12 **Include all the KFF Columns in Data Block with NULL Canvas.**
13 **Create a Text Item in the Data Block and change the following properties.**

Name	:	KFF
SubClass	:	Text_Item
Database Item	:	NO
LOV	:	Enable_List_Lamp
Validate From List	:	NO
Canvas	:	Canvas_Name

14 **Create a Package Specification in Program Unit with a KFF Procedure and the name of the package is same as Block Name.**
PROCEDURE KFF (EVENT IN VARCHAR2);

15 **Create Package Body in Program Unit.**

```
PROCEDURE KFF (EVENT IN VARCHAR2) IS
BEGIN
        IF EVENT = 'WHWN-NEW-FORM-INSTANCE' THEN
        FIND_DESCR_FLEX.DEFINE ('EX_ORDERS','KFF','EX',
        'Code_Of_KFF','Structure_Code_No');
        --- FIND_DESCR_FLEX.DEFINE ('Block_Name','Item_Name',
        --- 'Application_Short_Name', Code_Of_KFF','Structure_Code_No');
        END IF;
END;
```

16 **Call the KFF procedure in WHWN-NEW-FORM-INSTANCE Trigger at Form Level and passing the event as WHEN-NEW-FORM-INSTANCE.**

17 **Call FND_FLEX.EVENT in WHEN-NEW-ITEM-INSTANCE Trigger at form level.**

Note: FND_FLEX.EVENT is a common Built-In used to invoke DFF and KFF.

18 **Save and Compile the form and copy .fmx file in to module specific forms directory.**

19 **Perform all the prerequisites to register the form.**

14. PROFILES

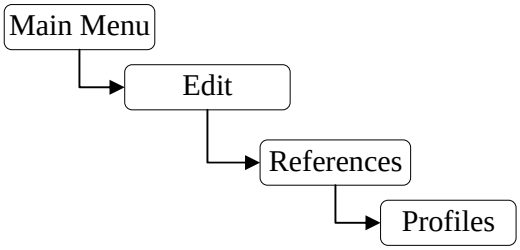
Profiles are used to determine the behavior of Oracle Application Forms and Programs. The registration information of all the users will be stored in FND_PROFILES_OPTION table. The dictionary of all the Profile Options is maintained by AOL module. To set the values of the Profile Options, we use System Administrator Module. Profile Options are used to pass the values to the variables declared in our forms and programs, by which we can determine the behavior of Oracle Application Forms.

Note: There are 5316 types of profiles options are going to come with the package.

Each Profile Option has access at 3 levels.

1 User Access Level.
2 Program Access Level.
3 System Administrator Level

USER LEVEL: If the Profile is having access at this level any front end login user can change the values of the Profile Options.



PROGRAM LEVEL: If the Profile is having access at Program Level then we can use them in our Programs.

SYSTEM ADMINISTRATOR LEVEL: If the Profile is having access at this level then we can use Profile Options at the following four sub levels.

1 SITE Level
2 APPLICATION Level
3 RESPONSIBILITY Level
4 USER Level

SITE LEVEL: Site is a collection of Modules. A Module is a collection of Responsibilities. Responsibility is assigned to different users.

If we set the Profile Options at Site level then these Profile Options get affected to all the Forms and Programs of all the Modules, which are accessed from our site.

APPLICATION LEVEL: If we set the Profile Options at this level then the Profile Options get affected to all the forms of a particular module.

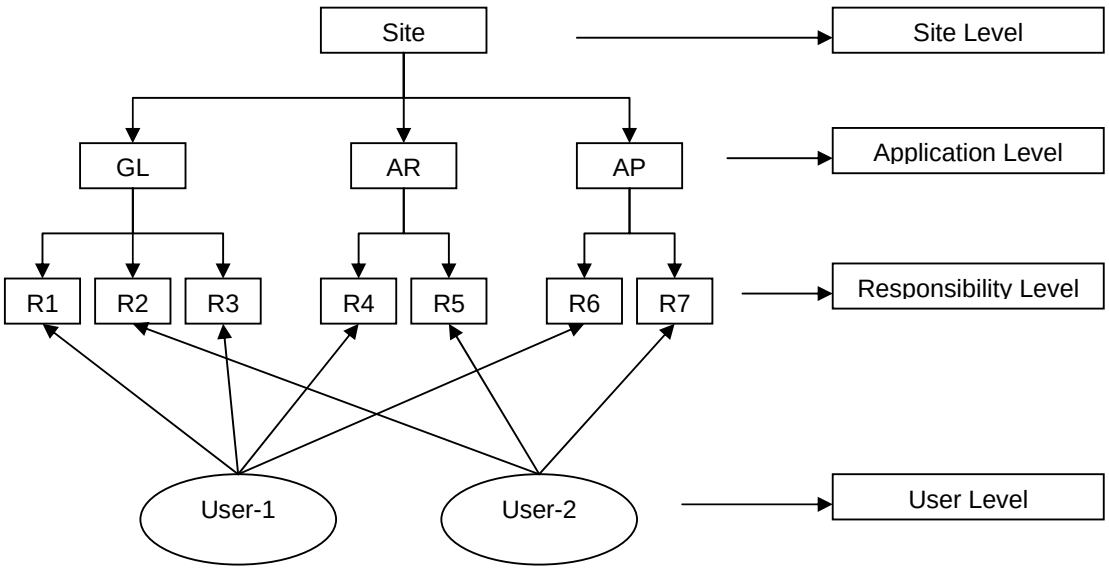
RESPONSIBILITY LEVEL: If we set the Profile Options at this level then the Profile Options get affected to all the Forms and Programs, which are assigned to a particular Responsibility.

USER LEVEL: When we set the Profile Options at this level then the Profile Options get affected to all the Forms and Programs of all the Responsibilities, which are assigned to a particular User.

When we set the Profile Options at all the 4 level then

1. User Level will overwrite Responsibility Level
2. Responsibility Level will overwrite the Application Level
3. Application Level will overwrite the Site Level

The Built-In that is used to get the value of the Profile Option is FND_PROFILE.GET, which takes two arguments i.e. Name of the Profile Option and the Output Variable. This Built-In first searches for the value at User Level and it will get the USER_ID, RESPONSIBILITY_ID and APPLICATION_ID of Form where the Form is called.



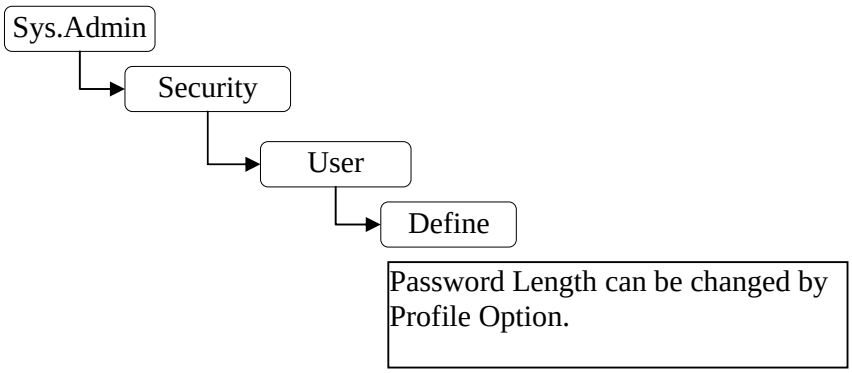
When we call the Built-In, it will first searches at User Level

1 If the value is not found at User Level then it will search at Responsibility Level.

2 If the value is not found at Responsibility Level then it will search at Application Level.

3 If the value is not found at Application level then it will search at Site Level. This is the default level of Profile Option.

Q: How to set the values for existing Profile Options?



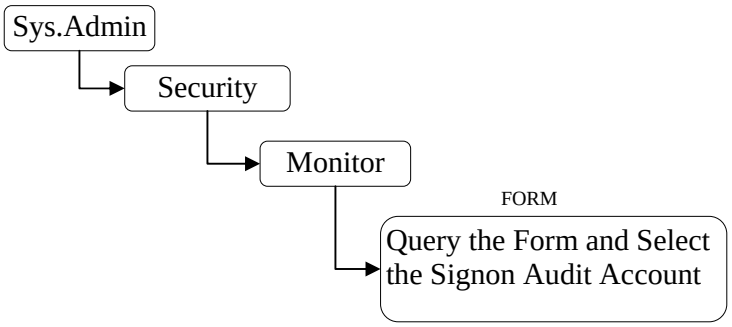
Built-In to change the length of the password is called SIGNON Password Length.

Note: Whenever we set the Profile Option at Site Level, we have to re-login into Applications.

The different Signon Audit Accounts are

- NONE
- USER
- RESPONSIBILITY and
- FORM.

Q: How to know the WHO the users are currently connected to the Server for Audit Accounts?



LOV based on Record Group, Record Group Based on Select Statement.

Profiles will be changed on the basis of Profile Options Record Group.

Profile	Profile Type	Status
P1	No LOV	SQL Validation is not required
P2	Static/Fixed Values	SQL Validation is required Defined in FND_LOOKUPS Table
P3	Variable	SQL Validation is required Validation is based on any table.

Steps required creating a profile on tables

1. Define a profile.
2. Setting a new profile.
3. Adding code into Forms Trigger.

15. CUSTOMIZATION OF FORMS

CUSTOM.PLL is used to customize the forms with out modifying the code of Oracle Applications. There is only one CUSTOM.PLL exists for all the Forms in Oracle Applications. CUSTOM.PLL is going to be in use when ever we start Form Server Service. In CUSTOM.PLL we have a Package by name CUSTOM. Under this package we have the built-ins as ZOOM AVAILABLE, EVENT and STYLE.

ZOOM AVAILABLE: is the function which returns a BOOLEAN value (TRUE or FALSE). By default it returns FALSE. If this function is going to return TRUE, the ZOOM Menu Item from the main menu is enabled. The ZOOM menu item is used to call one form from another form.

EVENT: is a procedure which is used to handle specific events in which we are going to add our custom code. The main events that we can handle with this procedure are

WHEN-NEW-FORM-INSTANCE Trigger
WHEN-NEW-BLOCK-INSTANCE Trigger
WHEN-NEW-RECORD-INSTANCE Trigger
WHEN-NEW-ITEM-INSTANCE Trigger
WHEN-VALIDATE-RECORD Trigger
PRE-FORM Trigger
POST-FORM Trigger

All these triggers, which are created at form level, which are identified with blue colors. Event procedure by default performs NULL.

STYLE: is a function. It returns an integer ranging from 1 to 4

1	indicate	BEFORE
2	indicate	AFTER
3	indicate	OVERWRITE
4	indicate	STANDARD

These are Module Specific events. The built-ins of CUSTOM.PLL are called by the built-ins of APPCORE.PLL in all the form level triggers which are going to come with TEMPLATE.fmb form.

CUSTOM.PLL (Library Name)
    CUSTOM (Package Name)
        EVENT (Procedure Name)

Whenever we create or develop a form in applications these are some form level triggers, which will be created, and in all the form level triggers built-ins of

APPCORE.PLL are called. The built-ins of APPCORE.PLL are going to call the built-ins of CUSTOM.PLL.

Restrictions in CUSTOM.PLL

1. We should not attach and call any built-ins of APPCORE.PLL in CUSTOM.PLL to avoid recursion. Instead we can use APPCORE2.PLL in CUSTOM.PLL.
2. APPCORE2.PLL is a duplicate of APPCORE.PLL. All the built-ins of APPCORE2 are suffixed with 2.
3. We can't use SQL statements directly in CUSTOM.PLL. In order to use SQL statements we can make use of Packages and Procedures.

Note: 1. To get the information about current form name in use
Go to help in main menu
Open Oracle Applications

2. To get the Block Name and Item Name of the current form
Go to help in main menu
Diagnostic
Examine

How to Implement ZOOM Availability?

1. **First identify calling form and called form details.**
2. **Open called form in form builder.**
3. **Create a parameter.**
4. **Modify WHEN-NEW-INSTANCE trigger at form level.**

```
IF :Parameter.P_Vendor_Name IS NOT NULL THEN
        GO_BLOCK('VNDR');
        DO_KEY('EXECUTE_QUERY'):
        :Parameter.P_Vendor_Name := NULL;
ELSE
        /* Existing Statements */
END IF;
```

5. **Modify PRE-QUERY trigger at Block level.**
6. **Save and Compile the form and copy .fmx file in module specific directory.**
   Note: By default ZOOM option in the Menu bar is disabled for all the Forms.
7. **Open the CUSTOM.PLL library in Oracle Form Builder.**
8. **Modify the code in ZOOM_AVAILABLE Function.**

```
FORM_NAME := NAME_IN ('SYSTEM.CURRENT_FORM');
BLOCK_NAME := NAME_IN ('SYSTEM.CURSOR_BLOCK');

BEGIN
    IF (FORM_NAME = 'POXPOEPO' AND
        BLOCK_NAME = 'PO_HEADERS') THEN
                RETURN TRUE;
    ELSE
                RETURN FALSE;
     END IF;
```

9. **Handle ZOOM Event in EVENT Procedure of CUSTOM.PLL.**
   Built-ins of calling or Opened Form are CALL_FORM, OPEN_FORM and NEW_FORM.

```
FND_FUNCTION.EXECUTE (FUNCTION_NAME = ' ',
        OPEN_FORM = 'Y',
        SESSION_FLAG = 'N',
        OTHER_PARAMETERS = 'P_VENDOR_NAME');
```

16. CONCURRENT PROCESSING (CP)

Concurrent Programming is a processing requests simultaneously and producing different results. Different types of programs supported by oracle applications are Oracle Reports, PL/SQL, JAVA, JAVA Concurrent Process, SQL Scripts, SQL * Loader, PRO *C, C, C++, HOST etc.

To process all the Concurrent Programs in applications is Application User Concurrent Manager Server. In applications we can run only those programs which are assigned to the Request Group of our Responsibility. To run any Concurrent Program in application we use SRS (Standard Request Submission) Form. Actual executable name of the SRS Form is FNDRSRUN.fmb. We can call this SRS From directly from the main menu or we can create a form function for this form and assign this to our Menu Responsibility.
Based on the executable name CP identifies the actual executable name, executable methods and for which module it is assigned.

Reports Server stores Oracle Reports
Database Server stores JAVA, C, etc.
Concurrent Manager Server Stores Pro *C, Pro * Cobol, etc.

The information of the Concurrent Programs are stored in FND_CONCURRENT_PROCESSES table.

Columns of this table are
- Request_ID
- User
- Program
- Executable
- Arguments
- Phase → Pending, Running, Completed
- Status → Standby, Normal, Errors
  (or) Normal, Paused, Warnings Normal

Whenever we submit a program SRS Form first the request information goes to Internal Concurrent Manager.

Internal Concurrent Manager has two components

1. Response Component
2. Identification Component

Response component generates a unique Request_ID for each program submitted for SRS form and updates the request information in FND_CONCURRENT_REQUEST table, then forwards the request to Identification component. Identification Component

searches for the program in FND_CONCURRENT_REQUEST table and executes output format and output style based on the executable name it identifies the actual executable name, execution method and module name. Based on the module name and executable method application identifies the executable in Module specific directory and maps the parameter values to the executable from FND_CONCURRENT_REQUEST and updates the program information in FND_CONCURRENT_PROCESSES. Once the process is completed it generates the .log and .out files in module specific directory.

Note: In Forms Form Functions are used to pass the parameters at run time, where as in reports programs are used to set the different output formats.

Steps Required for Registering a Simple Report

1. Create a Report using Report Builder.
2. Compile and copy .RDF file in module specific directory.
3. Register the executable with System Administrator Module.
4. Define the Concurrent Program.
5. Assign the executable to Concurrent Program.
6. Assign the Concurrent Program to Request Group.
7. Assign the Request Group to the Responsibility.
8. Assign the Responsibility to the User.

FND_EXECUTABLES Table captures the information of all the executables, which are registered.

FND_CONCURRENT_PROGRAMS Table stores the information of Concurrent Programs.

Q: How to Run the Report from Oracle Applications?

Ans: Open VIEW in main menu
Requests
Submit a request by pressing the button

Q: What are the different types of parameters available in Reports?

Ans: There are two types of parameters

1. BIND PARAMETERS
2. LEXICAL PARAMETERS.

The parameters, which we create at design time, are called Actual Parameters. The parameters defined at concurrent program level are called Formal Parameters. Validation of a parameter is to be done to the formal parameters.

Registering Parametric Reports

1. Create a Report using Report Builder with parameters.
2. Compile and copy .RDF file in module specific directory.
3. Register the executable with System Administrator Module.
4. Define the Value set to validate the parameters.
5. Define the Concurrent Program.
6. Assign the executable to Concurrent Program.
7. Define Parameters.
8. Assign Value Set to the Parameters.
9. Assign bind parameter of yours to the TOKEN.

   Note: Token is used to map bind parameters with the formal parameters of the Concurrent Program.
10. Assign the Concurrent Program to Request Group.
11. Assign the Request Group to the Responsibility.
12. Assign the Responsibility to the User.

To reference the values of Prior Parameters of a particular program into the values of other parameter is based on the Value Set. The value of the 1st parameter is to be referenced in where clause of the 2nd Parameter.

WHERE Deptno = :$FLEX$.First_Parameter

Note: 1. To call SRS Form from our own Menu register SRS Form (FNDRSRUN) in our Module.

2. We can assign only one Request Group for a Responsibility.

3. We can call the programs of multiple Request Groups from a Request Group.

4. For each Request Group we can create a function.

To run the programs of the Request Group from the Responsibility it is not compulsory to assign the Request Group to the Responsibility. We can also assign a Form function to the Menu it is defined on FNDRSRUN with Request Group code, Request Group application short name and Title as parameters.

17. FLEX FIELD FEPORTS

USER EXITS: These are the 3rd party programs (i.e. Java, Pro * C, etc.) that we can link with our reports to perform a particular task. We can link 3rd party programs with our reports with SRW.USER_EXIT('Name_of_the_User_Exit'). We can reference the input parameters in user exits with SRW.REFERENCE(:input_parameters).

FND SRWINIT: This User Exit is used to initialize the report in the memory.

FND SRWEXIT: This User Exit is used to release the memory occupied by the program.

FND FLEXSQL: This User Exit is used to tailor the SELECT statement which returns a string. It is a concatenation of the Segment columns of a particular structure.

Parameters:
Application Short Name: "SQLGL"
Code: "GL#"
Number: "P_STRUCTURE_ID"
Output: "P_FLEXDATA"

Note: Bind Parameters can be mapped with only one column where as Lexical Parameters is a placeholder column which can be used to substitute string in the parameter.

Example:

```
SELECT P_FlexData CflexData,Journal_Name,…,…,
FROM KFF, TRANS
WHERE KFF.CCID = TRANS.CCID
AND STRUCT_ID = :P_STRUCT_ID;
```

FND FLEXID_VAR: is user to retrieve Value_Set_IDs of all the Segments based on a parameter Structure and also retrieves values and descriptions based on Value_IDs and Value_Set_IDs.

Application Short Name	:	"SQLGL"
Code	:	"GL#"
Number	:	"P_STRUCTURE_ID"
Data	:	"C_FLEXDATA"
Description	:	":C_DESC"

It concatenates the Value Set IDs of Segments.

Steps Required to create a Simple Flex Field Report

1. **Open the Report in Report Builder.**
2. **Create User defined parameters as per our requirement.**

Name	Data Type	Width	Initial Value	Notes
F_CONC_REQUEST_ID	Number	15	0	Always Create
P_FLEXDATA	Character	600	Long String	Cumulative width more than expected width required to hold the data (i.e. concatenated value of Segments 1..n)
P_STRUCT_NUM	Character	15	101	Contains Structure Number

3. **Call FND SRWINIT User Exit in Before Report Trigger.**

```
BEGIN
        SRW.USER_EXIT ('FND SRWINIT');
        RETURN (TRUE);
END;
```

4. **Call FND SRWEXIT User Exit in After Report trigger.**

```
BEGIN
        SRW.USER_EXIT ('FND SRWEXIT');
        RETURN (TRUE);
END;
```

5. **Call FND FLEXSQL User Exit in Before Report Trigger.**

```
BEGIN
        SRW.USER_EXIT ('FND SRWEXIT');
   BEGIN
        SRW.REFERENCE (':P_STRUCT_NUM');
        SRW.USER_EXIT ('FND FLEXSQL
                        APPL_SHORT_NAME = "SQLGL"
                        CODE = "GL#"
                        NUM = ":P_STRUCT_NUM"
                        OUTPUT = ":P_FLEXDATA"
                        MODE = "SELECT"
                        DISPLAY = "ALL" ');
    END;
        RETURN (TRUE);
```

```
END;
```

Note: When we are calling multiple user exits in a trigger we should call these User Exits in separate Blocks (i.e. BEGIN .. END).

6. **Create a report query in the Data Model.**

```
SELECT &P_FLEXDATA C_FLEXDATA
FROM GL_CODE_COMBINATION
WHERE CHART_OF_ACCOUNT_ID = &P_STRUCT_NUM;
```

7. **Create two Formula Columns C_VALUE and C_DESC_ALL.**
8. **Call FND FLEXIDVAL in the PL/SQL formula of C-VALUE to reference Value Code Combination.**

To retrieve the concatenated flexfield segment values and description you incorporate the AOL user exits in these columns

```
BEGIN
    SRW.REFERENCE (':C_FLEXDATA');
    SRW.FND (:P_STRUCT_NUM);
    SRW.USER_EXIT ('FND FLEX_ID_VAL
                APPL_SHORT_NAME = "SQLGL"
                CODE = "GL#"
                NUM = ":P_STRUCT_NUM"
                OUTPUT = ":P_FLEXDATA"
                VALUE = ":C_VALUE"
                DISPLAY = "ALL" ');
    RETURN (:C_VALUE);
END;
```

9. **Call FND FLEXIDVAL in the PL/SQL formula of C_DESC_ALL to retrieve Description Code Combination.**

```
BEGIN
    SRW.REFERENCE (':C_FLEXDATA');
    SRW.FND (:P_STRUCT_NUM);
    SRW.USER_EXIT ('FND FLEX_ID_VAL
                APPL_SHORT_NAME = "SQLGL"
                CODE = "GL#"
                NUM = ":P_STRUCT_NUM"
                OUTPUT = ":P_FLEXDATA"
                DESCRIPTION = ":C_DESC_ALL"
                DISPLAY = "ALL" ');
    RETURN (:C_DESC_ALL);
END;
```

10. **Run the report with default Layout.**
11. **Compile and copy .RDF in module specific directory.**
12. **Perform all the pre-requisites to register the report.**

System Administrator

Concurrent

Program

Executable

Note: All the FND Tables are stored in Application Object Library (AOL).

18. QUALIFIERS

Qualifiers are used to identify the Segments with specific Property and the value of the Segments while designing the reports. Qualifiers are classified in to two categories.

1. Flex Field Qualifiers (FFQs)
2. Segment Qualifiers (SQs)

Flex Field Qualifiers (FFQs): are used to assign a specific property to the Segment Qualifier. These are based on Key Flex Fields (KFF). FFQs are varry from one KFF to another KFF and it is not compulsory that all the KFF should have FFQs.

Segment Qualifiers (SQs): are used to assign a specific property to the value of a Segment. SQs is based on FFQs and it is not compulsory that all the FFQs should have SQs.

Note: You must define Flexfield Qualifier before you define Segment Qualifier.

For example according to Accounting Flex Fields

1. **Balancing Segment Property (User Name)**
   GL_BALANCING (Actual Name)
   a. **Global Flexfield Qualifier:** applies to all segments.
   b. **Unique Flexfield Qualifier:** applies to only one Segment.
   c. **Required Flexfield Qualifier:** applies to at least one Segment.

   Write Code before Execution of the Report

```
FND FLEXSQL
    APPL_SHORT_NAME = "SQLGL"
    CODE = "GL#"
    NUM = "P_STRUCT_NUM"
    OUTPUT = ":P_COMPANY"
    MODE = "SELECT"
    DISPLAY = "GL_BALANCING"
```

2. **Natural Accounting Segment Property (User Name)**
   GL_ACCOUNT (Actual Name)
   a. Global Flexfield Qualifier
   b. Unique Flexfield Qualifier
   c. Required Flexfield Qualifier

Write Code before Execution of the Report

```
FND FLEXSQL
APPL_SHORT_NAME = "SQLGL"
CODE = "GL#"
NUM = "P_STRUCT_NUM"
OUTPUT = ":P_ACCOUNT"
MODE = "SELECT"
DISPLAY = "GL_ACCOUNT"
```

3. Cost Center Segment Property (User Name)
   GL_COST_CENTER (Actual Name)
   a. This is not Unique and is not compulsory

19. INTERFACES

Interfaces are the Tables, which are act as a medium to transfer the data from one module to another module or to transfer the data from legacy system into Oracle Applications. There are 352 tables provided by the Oracle Package. Each module is having its own Interface Tables.

Interfacing:

It is the process of converting the records from one format to another format.

The main components of this interfacing are

- Transfer Program
- Interface Table and
- Import Program

Transfer Program:

If the source modules data are implemented in Oracle Applications then the Transfer Programs are going to come with the Package. If the source modules are implemented in external system (i.e. other than Oracle Applications) then we have to develop our own Transfer Programs. Generally these Transfer Programs are developed using PL/SQL, JAVA or SQL Loader.

The Logic of the Transfer Programs is

- It maps the columns of source table with the columns of Interface Tables.
- It performs Row Level and Column Level validations.
- It transfers the data from Source to the Interface Table.

Interface Tables:

The columns of these tables are classified into 4 types.

1. Mandatory Columns.
2. Conditionally Required Columns.
3. Optional Columns.
4. Internal Processing Columns.

Mandatory Columns:

These are the Normal columns in the Interface Tables. These are the main columns which are required in the destination tables (i.e. Oracle Application Module Tables).

Note: With the help of mandatory columns only the Import Program will converts the records from source to destination.

Conditionally Required Columns:

The values for these columns are based on the values of Mandatory columns.

Example: When we are converting foreign currency transactions to INR then it as compulsory to provide conditionally required columns like Currency conversion rate, Conversion Time and Conversion Date.

Optional Columns:

These are used when a client wanted to transfer some additional information from source to destination. These are based on client's requirement.

Internal Processing Columns:

Status and Error Message columns are called Internal Processing Columns. These are specific only to Interface Table. These columns are going to be used by the Import Program to update the status and error message, if the record fails its validation while importing from Interface Table to the Destination Table.

Import Program:

For all Interface Tables, Oracle Application Package is going to provide Import Programs. These are generally registered with destination modules. These Import Programs are designed using PL/SQL, JAVA, C, C++, etc.

The logic of the Import Program is

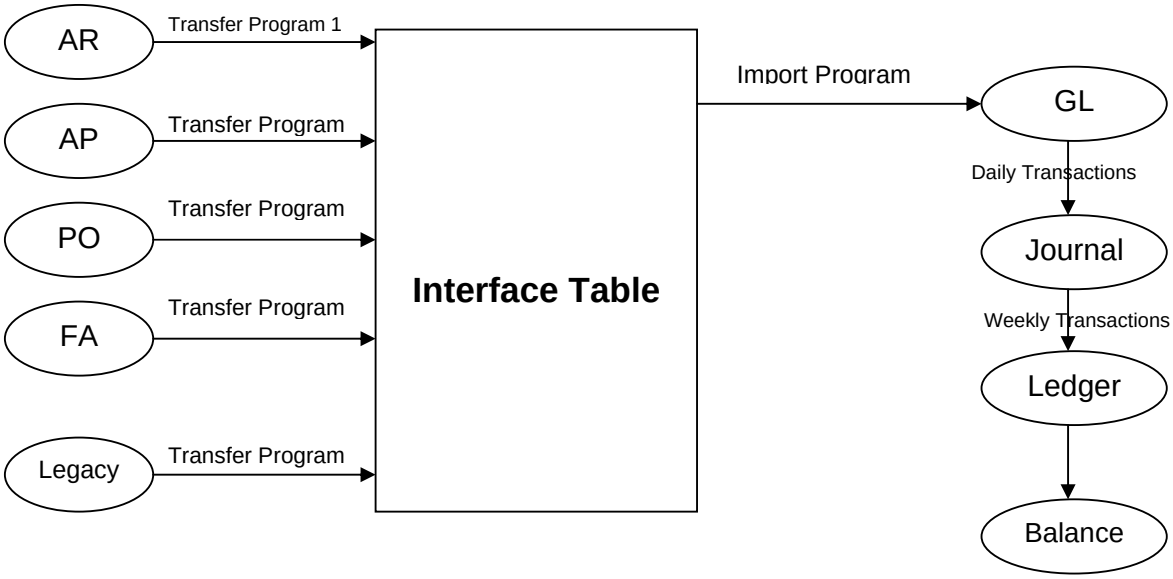
- It maps the columns of the Interface Table with one or more columns in the destination table.

- It performs row level and column level validation.
- It imports the data from Interface Table to the Destination tables, if the records validated successfully.
- It deletes all the successfully validated records from Interface Table.
- If the record fails its validation then the Import Program will update the status and error message columns of Interface Table.

Q: What is the difference between Interface and Application Program Interface (API)?

Ans: Interfaces are used to transfer the data from legacy system to Oracle Application system where as API is used to convert the data from one form to another form with in the Oracle Application Module.

Example:



Scenario-1: Interface Tables are provided and the integrating modules are implemented in Oracle Applications. In this scenario the package itself is going to provide a separate transfer program for each integrating module. **See figure for details.**
Indent: is specifies the details like, the list of materials required, how much quantity is required, when the material is required and quality of the items.

Scenario-2: Interface Tables are provided and the integrated modules are implemented in external system with the same database as Oracle Applications. In this scenario front end is different and database is same.

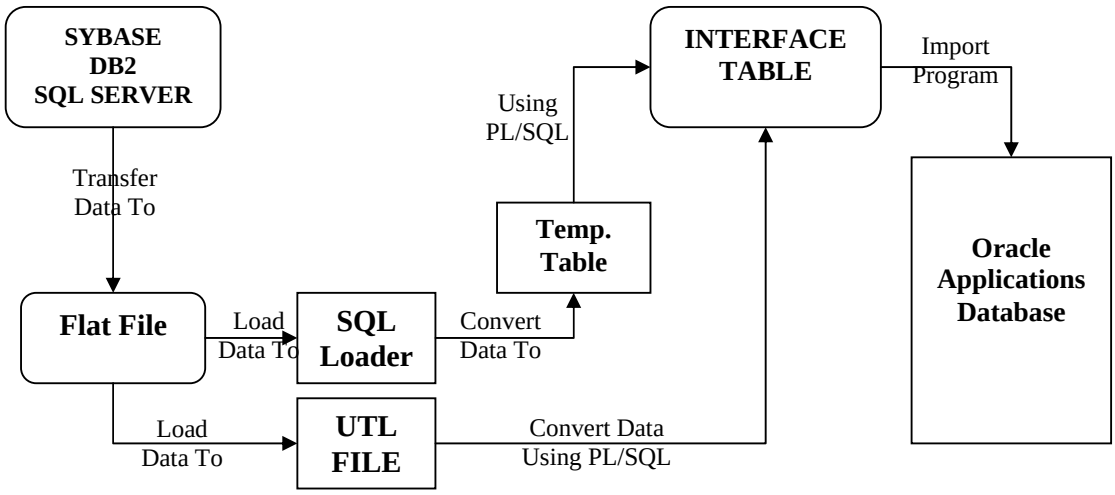
Take for example there are 40 columns according to transaction table of client out of which 23 are mandatory columns and 17 are optional columns. But as per applications interface table is having 20 mandatory columns. While transferring the data from existing system to applications we have to search for matching columns in transaction table with the interface mandatory columns. Matching columns may be of mandatory or optional columns of transaction table of client. Assume that there are 17 columns are found matching columns out of 20 mandatory columns. If we want to insert the record into interface table we need 20 mandatory columns compulsorily. In this example since there are 17 matching columns out of 20 columns we have to provide default values to the non-matching (i.e. 3) columns with the authentication of the client. If the client is having mandatory columns of his own, which are not mandatory in interface table, then we map those columns with the optional columns of interface table.

Logic of the Transfer Program in this scenario

1. Create the database links between the two databases.
2. Search for the matching columns of the interface table with source tables or external tables.
3. Provide default values for the mandatory columns of interface table for the columns where the matching columns are not found in source table.
4. Perform row level and column level validations.
5. Incase if the client's requirement is to transfer all the mandatory columns of the source to the destination, we can map the remaining mandatory columns of the client are going to be mapped with optional columns of the interface table.
6. Create a PL/SQL procedure and map the columns of Interface Table columns.
7. Transfer the data from source table to the Interface Table.

Scenario–3: Interface Tables are provided and the integrating modules are implemented in external system with different database (i.e. Sybase, DB2, SQL Server, etc.). In this scenario we have to convert the data into flat file (i.e. .dat). Then we write the transfer program using PL/SQL script.

Scenario-3: Transfer the Data from Legacy System to Oracle Apps.



GL_DAILY_RATES_INTEAFACE Table is used to store the conversion rates on daily basis. It consists of the following fields

From_ Currency	To_ currency	From_ Conversion_ Date	To_ Conversion_ Date	User_ Conversion_ Type	Conversion_ Rate	Mode_ Flag
USD	INR	01-SEP-2006	30-SEP-2006	Corporate	44	I
ASD	INR	01-SEP-2006	15-Sep-2006	Spot	28	I

This Interface table consists of Database Trigger, which fires the input program whenever we insert the record in this. This will create no of records into GL_DAILY_RATES Table of GL module based on the From_Conversion_Date and To_Conversion_Date period.

Example: In the above table there are 30 days in the first record.

This DB Trigger updates total 60 records in GL_DAILY_RATES i.e.

30 records for conversion from USD to INR and

30 records for conversion from INR to USD.

Validations Required according to this interface table are

1. From currency should exists in GL_CURRENCY Table and it should be active.
2. To currency should also exists in GL_CURRENCY Table and it should be active.
3. From Conversion Date: All date validations applied for this date.
4. To Conversion Date: Same validations required as from currency but it should be greater or equal to From Conversion Date.
5. Conversion Type: This conversion type should exist in GL_CONVERSION_TYPE.
6. Conversion Rate: It should be a number.
7. Mode Flag: It is the status column in GL_DAILY_RATES interface table.

I	stand	for Insert
U	stands	for Update
D	stands	for Duplicate.

Note: From Currency, To Currency and Conversion Type are Case sensitive.

Steps required for transferring data from legacy system to oracle applications

1. Create daily.dat flat file and save it in server directory D:\Oracle\Visdb\plsql\temp (this is the default directory for UTL script files. To know the path of this directory open the initvis.ora file.

From_ Currency	To_ currency	From_ Conversion_ Date	To_ Conversion_ Date	User_ Conversion_ Type	Conversion_ Rate	Mode_ Flag
USD	INR	01-SEP-2006	30-SEP-2006	Corporate	44	I
ASD	INR	01-SEP-2006	15-Sep-2006	Spot	28	I

2. Create Transfer program using SQL and PL/SQL to upload data from flat file to the Interface Table.

Write the code at SQL * PLUS prompt (i.e. SQL>)

```
DECLARE
  Fp        UTL_FILE.FILE_TYPE;
  F_Curr    Varchar2 (10);
  T_Curr    Varchar2 (10);
  F_Date    Date;
  T_Date    Date;
  C_Type    Varchar2 (9);
  C_Rate    Varchar2 (2);
  M_Flag    Varchar2 (1);
  My_Line   Varchar2 (100);
BEGIN
   Fp:=UTL_FILE.FOPEN ('D:\Oracle\Visdb\9.2.0\plsql\temp','daily.dat','r');
   UTL_FILE.GET.LINE(Fp,My_Line);
```

```
LOOP
      UTL_FILE.GET.LINE(Fp,My_Line);
      F_Curr := SUBSTR(My_Line,1,3);
      T_Curr := SUBSTR(My_Line,1,3);
      F_Date := SUBSTR(My_Line,1,3);
      T_Date := SUBSTR(My_Line,1,3);
      C_type := TRIM(SUBSTR(My_Line,1,3));
      C_Rate := SUBSTR(My_Line,1,3);
      M_Flag:= SUBSTR(My_Line,1,3);
      INSERT INTO GL.GL_DAILY_RATES_INTERFACE(
      FROM_CURRENCY,  TO_CURRENCY,  FROM_CONVERSION_RATE,
      TO_CONVERSION_RATE,  USER_CONVERSION_RATE,
      CONVERSION_DATE,  M_FLAG) VALUES
      (F_Curr, T_Curr, F_Date, T_Date, C_type, C_Rate , M_Flag);
 END LOOP;
EXCEPTION WHEN NO_DATA_FOUND THEN
     TL_FILE.FCLOSE (FP);
     MESSAGE ('END OF THE FILE');
END;
```

20. BISINESS COMPONENTS

The main components that are required to any business transaction are DATE, CURRENCY, ACCOUNTS and ENTITIES.

DATES: To validate these date components we need to define calendar based on period type. This period type is to specify how many number of periods required for maintaining the balances of the organization (i.e. daily, weekly, monthly, quarterly, etc.). This period types are stored in GL_PERIOD_TYPES table. Based on these period types we define calendars. These calendars information is stored in GL_CALENDARS table. For each calendar we define periods depending upon the number of periods assigned to period types. This period's information is stored in GL_PERIODS table. By default the status of all these periods is closed. This period statuses are stored in GL_PERIOD_STATUSES table.

ACCOUNTS: To define accounting structure in General Ledger or in Finance Module we use Flexfield Structures. The structures defined using accounting KFF are called as Chart Of Accounts. Chart Of Accounts ID uniquely identifies each Structure.

ENTITIES: entity is nothing but a module. Supplier Entity (AP), Customer Entity (AR), Employee Entity (HR), etc., are some examples.

CURRENCY: There are 200 types of currencies come with the package. All the currency information will be stored in GL_CURRENCIES table. The currency, which is assigned to our Set Of Books, is called Functional Currency and the remaining all the currencies are called as Foreign Currencies.

There are 3 types of currencies

1. Functional Currency
2. Foreign Currency
3. Static Currency

Functional Currency: in which we are going to maintain the balances.
Foreign Currency: All the currencies other than functional currencies are called foreign currency.
Statistical Currency: This is used to record the usage factor for various measurement units.

Example: If 3 departments are going to share the same premises and rent for the premises is 20000. This 20000 has been shared proportionally to all the departments depending up on their usage factor.

CHART OF ACCOUNTS: is the structure defined on accounting Key Flex Field.

SET OF BOOKS (SOB): These are used to secure a Journals Transactions of a particular company and are a collection of components Calendar, Currency and Chart Of Accounts. Assign these SOB to a responsibility to the GL_RESPONSIBILITY using GL_SET_OF_BOOKS profile option. When we open the Journal from the responsibility the default status of this period is closed. To record a transaction from our responsibility we have to open the periods. Depending upon the first opening period system assigns never open status to its prior period and future entry status period to next periods depending upon the number of future period sets in our Sets Of Books.

Period Status	Journal Entry	Journal Posting	Inquiries/ Reports
Never Open	NO	NO	NO
Open	YES	YES	YES
Closed	NO	NO	YES
Future Entry	YES	NO	NO
Permanently Closed	YES	NO	YES

Set Of Books information stored in GL_DETS_OF_BOOKS table. Its Set_of_Books_ID uniquely identifies each set of book.

Q: How do we can find the Set Of Books, Flex Fields, Open/Close Periods, Category and Sources?

In AOL Module Setup Financials Books Define	In AOL Module Setup Flex Fields Define
In AOL Module Setup Open Close Latest Open Periods	In AOL Module Setup Journal Category
In AOL Module Setup Journal Sources	

GL_INTERFACE: It is used to convert the transactions of sub-ledger into Journals.

Validations required for GL_INTERFACE table are

1. **STATUS:** should be always new when we are transferring the data from sub-ledger to the status of general ledger. Import program changes the status to hold if the record is not validated successfully. The status should be in sentence case.
2. **SET_OF_BOOKS_ID:** this should exist in GL_SETS_OF_BOOKS table.

```
DECLARE
   CHK VARCHAR2 (1);
BEGIN
   SELECT 'X' INTO CHK FROM APPS.GL_SETS_OF_BOOKS
   WHERE SET_OF_BOOKS_ID = &SOB_ID;
   DBMS_OUTPUT.PUT_LINE ('THIS SOB_ID VALID YOU CAN PROCEED');
EXCEPTION WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE ('THIS SOB_ID IS NOT VALID');
END;
/
```

3. **ACCOUNTING_DATE:** is the date when exactly we can post the journals. Tables required to validate the accounting date are GL_SETS_OF_BOOKS, GL_PERIOD_TYPES, GL_PERIOD_SETS, GL_PERIODS and GL_PERIOD_STATUSES.
4. **CURRENCY_CODE:** This code should be same as the currency assigned to the Sets o f Books. This code should exist in GL_CURRENCY table. If the currency

code is different from Sets of Books then it is mandatory for us to provide values for the conditional columns like USER_CURRENCY, USER_TYPE, CURRENCY_CONCERSION_RATE and CURRENCY_CONVERSION_DATE.

Note:

5. **DATE_CREDITED:** is the transaction date or Journal date. This date should be in open or future entry periods. The tables affected for this same as accounting date. Here we have to give the condition for open and future periods. Date should be between starting date of latest open period and ending date of future entry period.
6. **CREATED_BY:** This is front-end login USER_ID should exist in FND_USER table and this user should also have the responsibility to record the Journals.
7. **ACTUAL_FLAG:** in general ledger we are going to maintain 3 types of balances.
    a. Actual Balances (flag is A)
    b. Budget Balances (flag is B)
    c. Encumbrance Balances (flag is E)

    To update actual balances the general has to be created as actual journal. Budget journals are going to affect budget balances. Encumbrance journals are going to affect encumbrance balances.
8. **USER_JE_CATEGORY_NAME:** it is used to describe the purpose of the journal entry. This category is going to be stored in GL_JE_CATEGORIES.
9. **USER_JE_SOURCE_NAME:** it is used to store the origin of the journal entry. This source is stored in GL_JE_SOURCES.
10. **BUDGET_VERSION_ID:** This column is required when we are going to record Budget Journals. This information is stored in GL_BUDGET_ENTITIES.
11. **ENCUBRANCE_TYPE_ID:** This is required when we are going to required Encumbrance Journals.
12. **SEGMENT_COLUMNS:** We have to provide the values for the segment columns depending upon howmany number of segment columns associated with the Chart_Of_Accounts_ID or Structure_ID associated with our Set_Of_Books.

    The tables used to validate the segments are

    FND_ID_FLEXES,

    FND_ID_FLEXE_STRUCTURES,

    FND_ID_FLEXE_SEGMENTS,

    FND_ID_FLEXE_VALUE_SETS,

    FND_ID_FLEXE_VALUES,

    FND_ID_FLEXE_VALUE_TL.

13. **ENTER_DR and ENTER_CR:** Columns are used to enter the debit amount ar Credit amount.

Sample Code Required to Upload the Data from Legacy System

```
LOADDATA
  INFILE *
  INTO TABLE GL.GL_INTERFACE
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ' " '
  (STATUS, SET_OF_BOOKS_ID, ACCOUNTING_DATE, CURRENCY_CODE, DATE_CREATED,
   ACTUAL_FLAG, USER_JE_CATEGORY_NAME, USER_JE_SOURCE_NAME, SEGMENT1, SEGMENT2,
   SEGMENT3, SEGMENT4, SEGMENT5, ENTER_DR, ENTER_CR)
BEGINDATA
  - - Record-1
  "new", 1, "25-MAY-2001", "USD",  "25-MAY-2001", 1001239,"A","Sales Invoice", "Receivables", 03, 110, 1570,
  0000, 110, 1947, ""
    - - Record-2
  "new", 1, "25-MAY-2001", "USD",  "25-MAY-2001", 1001239,"A","Sales Invoice", "Receivables", 03, 110, 6160,
  1100, 120, "", 1947
```

Save the above code with the file name GLINT.CTL in the path Oracle\visdb\8.1.6\bin at server end.

Then run the following command at command prompt at Server end to upload the data into various database tables.

SQLLDR APPS/APPS@VIS control=glint.ctl

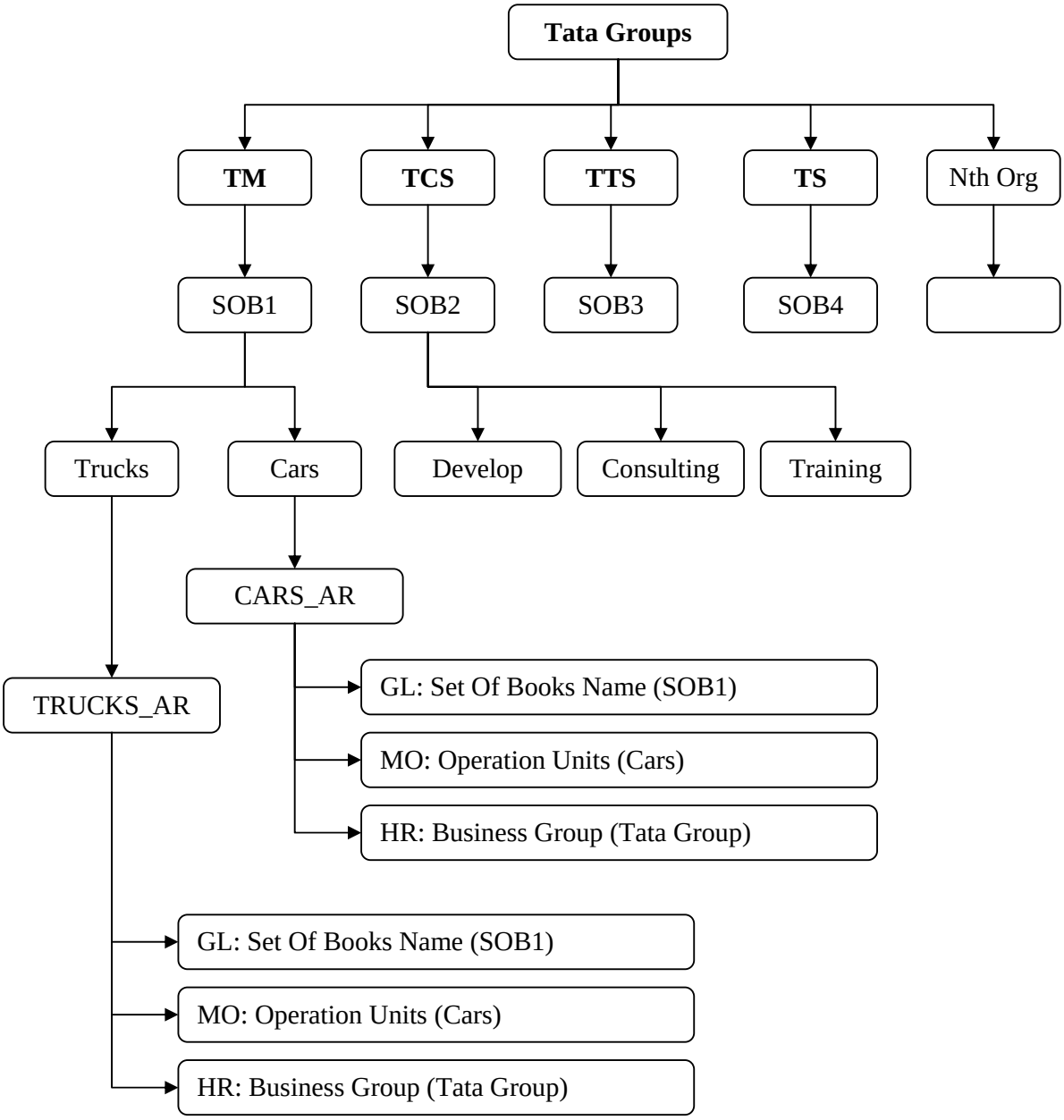
Journals are going to be maintained in 3 levels

1. Batches: is a collection of Headers. This information will be stored in GL_JE_BATCHES table.
2. Headers: is a collection of Lines. Headers are based on Category and Currency. This information will be stored in GL_JE_HEADERS table.
3. Lines: This information will be stored in GL_JE_LINES table.

21. MULTI ORGANIZATION

The ability to define multiple organizations and the relationships among them within a single installation of Oracle Applications is called multi organization. Multi Organization is the future used to store the data of multiple organizations in a single Database.

Structure of Multi Organization



The main components of the multi organization are

1) Business Group (BG)
2) Legacy Entity
3) Set Of Books (SOB)
4) Operating Units or Divisions
5) Organizations

Business Group (BG): The business group represents the highest level in the organization structure, such as the consolidated enterprise, a major division, or an Operation Company. A BG is used to secure human resources information like generation of employee numbers, generation of applicants, position flex fields, Job flex fields, Grade Flex field, Fiscal year, etc.

Legacy Entity: It is a legal business, which is going to have its own legislation code or tax registration number. At this level we are going to prepare all our financial statements.

Set Of Books (SOB): A SOB is a collection of Currency, Calendar and Chart of Accounts (COA). Oracle General Ledger is used to secure Journal transactions (such as journal entries and balances) of a company by set of books. For each organization of the Business Group we need to define a set of Book.

Operation Unit: An operating unit is a division or a Business unit of the legal entity. At this level we are going to maintain the information of sub-ledgers. We are going to maintain the ledgers at Legal Entity level. Receivable, Payables, Assets, etc. are comes under Operation Unit level.

Organization: The different types of organizations are Inventory Organization, Asset Organization, Human Resource Organization, etc. These organizations are going to be maintained at operating unit level.

Note: In case if we are working with single organization we can define an entity as a Business Group, Legal Entity, Operating Unit and Organizations.

The individual Accounts (i.e. Application Receivables) of an organization (i.e. Tata Motors) of multi organization (Tata Group) are identified by

First by Module then Responsibility then Profile Option then Set of Books

In the above example:

Module	Responsibility	Profile Option	Set Of Books
GL	TM_GL	GL_Set_Of_Books	SOB1
GL	TCS_GL	GL_Set_Of_Books	SOB2
GL	TTS_GL	GL_Set_Of_Books	SOB3
GL	TS_GL	GL_Set_Of_Books	SOB4
GL	….	GL_Set_Of_Books	….
GL	Nth_GL	GL_Set_Of_Books	SOBn

Data will be secured for every organization by using Set Of Books ID.

The following are the tables are going to be effected for all the Organizations of multi organization.

GL_JE_BATCHES

GL_JE_HEADERS

GL_JE_LINES

To record the sales transactions of all the organizations will be stored in the AR_TRANSACTION_ALL table. Extra columns required to identify the individual sales transactions are Set_Of_Books_ID, Org_ID (Operating Unit) and Organization_ID (Business Group).

22. ALERTS

Oracle Alert facilitates the flow of information within your organization by letting you create entities called *alerts.* Oracle Alert will send messages or perform predefined actions in an action set when important events occur.

Alert is a mechanism that checks your database for a specific exception condition. *Alerts are used* to monitor your business information and to notify you of the information you want.

There are two types of Alerts

1. An *event alert*
2. A *periodic alert*.

An *event alerts:* Event alerts immediately notifies the activity (i.e. an insert or an update to a table) in the database as it happens/ occurs.

To create an event alert, you perform the following tasks in the order listed:

- Define the database events that will trigger your alert
- Specify the details for your alert
- Define actions for your alert
- Create action sets containing the actions you want your alert to perform

Example of an event alert: Select statement that reports the creation of new users:

```
SELECT user_name, :MAILID INTO &NEWUSER, &USER
FROM fnd_user
WHERE rowid = :ROWID;
```

A *periodic alerts*: Periodic alerts periodically report key information according to a schedule you define.

For example, you can define a periodic alert for Oracle Purchasing that sends a message to the Purchasing Manager listing the number of approved requisition lines that each purchasing agent placed on purchase orders. You can define this alert to run weekly, and provide performance measurement on a consistent and timely basis.

To create a periodic alert, you perform the following tasks in the order listed:

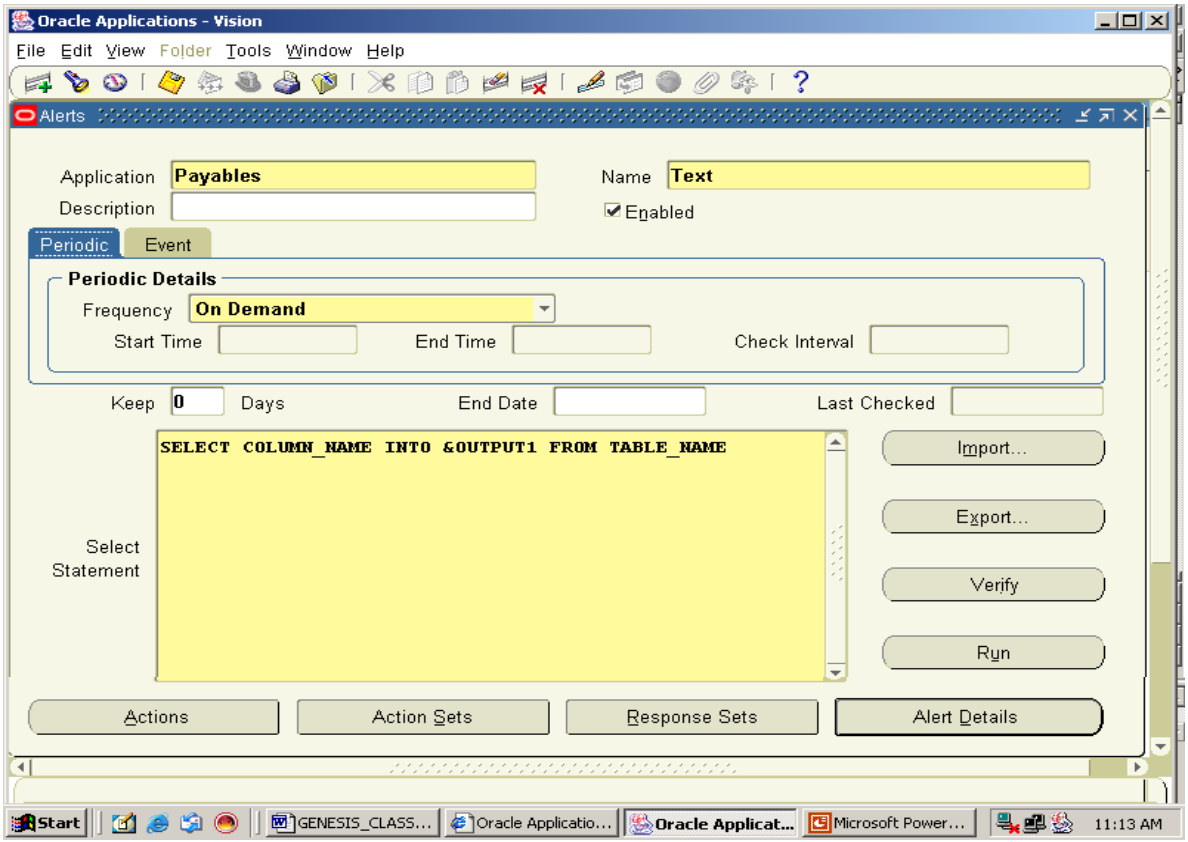
- Define your periodic alert and specify its frequency
- Specify the details for your alert
- Define actions for your alert
- Create action sets containing the actions you want your alert to perform

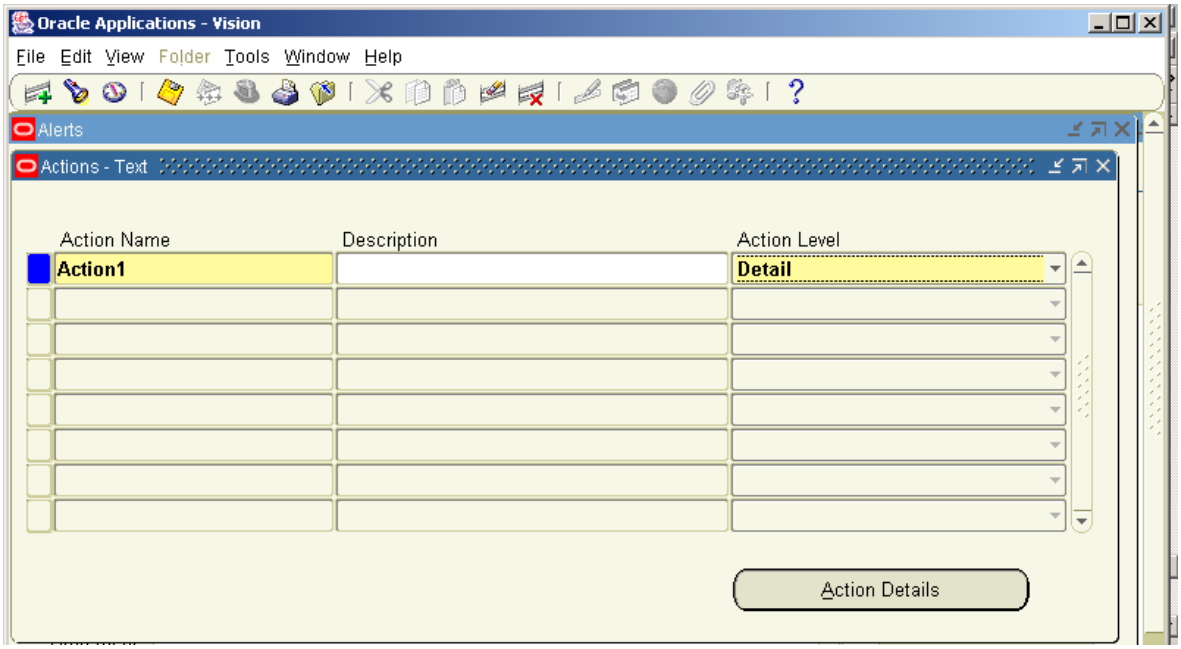
Example of a periodic alert: Select statement that looks for users who have not changed their passwords within the number of days specified by the value in :THRESHOLD_DAYS.:

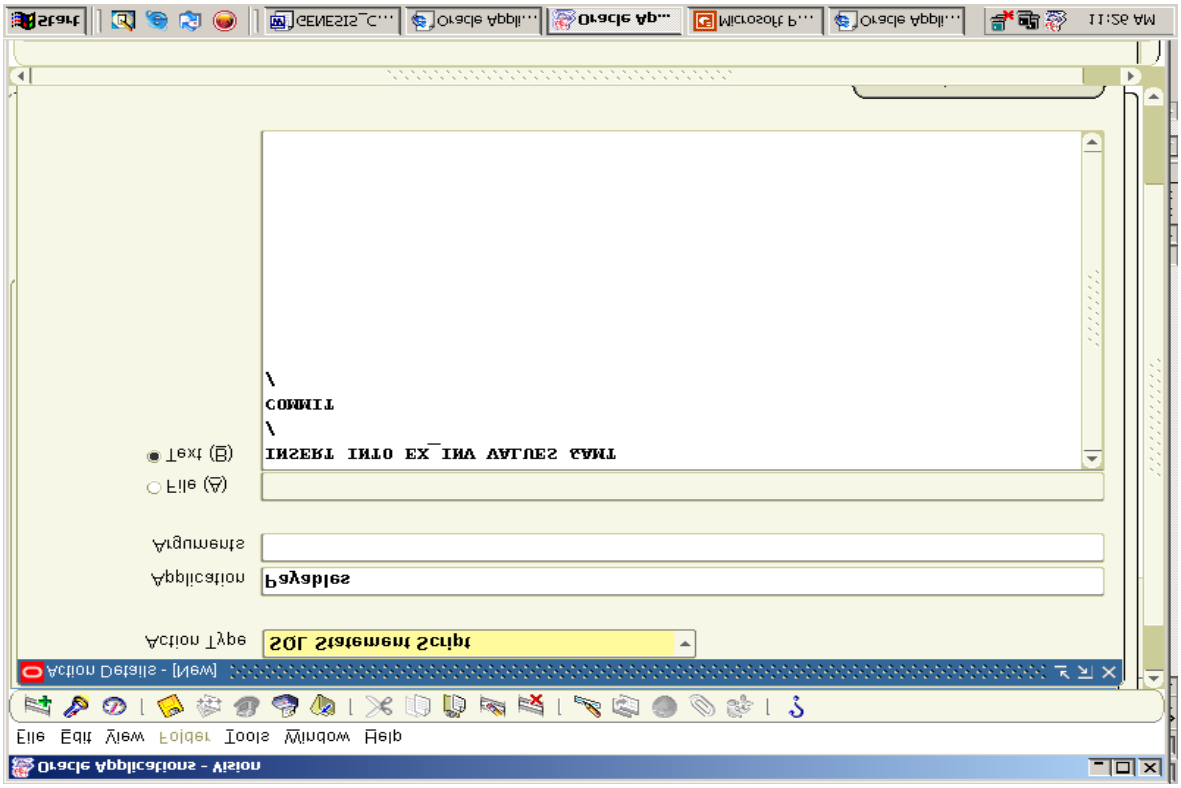
```
SELECT user_name, password_date, :THRESHOLD_DAYS
INTO &USER, &LASTDATE, &NUMDAYS
FROM fnd_user
WHERE sysdate = NVL(password_date, sysdate) + :THRESHOLD_DAYS
ORDER BY user_name;
```

Alert Action: An action you want your alert to perform. An alert action can depend on the output from the alert. An action can include sending an electronic mail message to a mail ID, running an Oracle Applications program, running a program or script from your operating system, or running a SQL script to modify information in your database. You can have more than one action for an alert, and an action can incorporate the output of the alert.

Action Set: A sequence of alert actions that are enabled for a particular alert. You can assign a sequence number to each action you include in an action set to specify the order in which the actions are performed.




Oracle Applications - Vision
File Edit View Folder Tools Window Help
Alerts
Actions - Text
Action Name
Description
Action Level
Action1
Detail
Action Details


Oracle Applications - Vision
File Edit View Folder Tools Window Help
Action Details - [New]
Action Type
SQL Statement Script
Application
hrxapps
Arguments
File (A)
Text (B)
INSERT INTO EX_INV_VALUES CAMT
\
COMMIT
\

Message: This action is used to send a message to application user email_id using Electronic mail server.

Concurrent Program: Using this action we can run any concurrent program, which is registered in oracle applications when an exception is raised.

SQL Statement Script: This option is used to run the SQL statements when an exception is raised.

OS Script: This option is used to run a Shell Script or Batch file when an exception is raised.

Responsibility is used to define the Alerts are Alerts Manager in production services. Alerts Manager Vision Enterprises in vision database.

Note: History table is used to store the exceptions raised and actions taken by alerts information. Alert (i.e. whether the alert is fired or not) status is found in Alerts History option.

Details: Validates of the entire details summary. Any one of the conditions is satisfied is enough to raise an exception.

Action Sets: is the collection of actions. This option calls the actions when an exception is raised.

Note: Vision Applications comes under Operating Unit

Oracle ID is APPS and Operation Unit is VISION

23. DISCOVERER

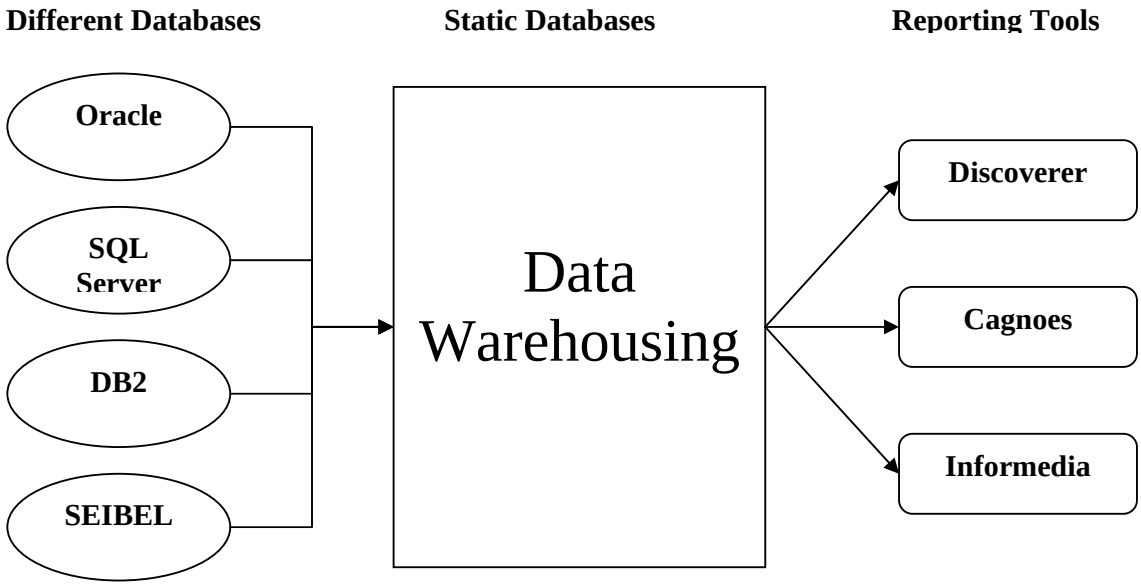
Oracle Discoverer is a decision support product that enables you to perform ad hoc queries on a database, analyze and format the results of the query, prepare the results for presentation, and manage data in a way that is meaningful for your business situation. It separates the more difficult database administration tasks from the simpler querying and reporting tasks so that analysts, managers, and other information workers can easily get their work done without having to know about either databases or SQL.

Oracle Discoverer has three parts:

1. **Administration Edition** - enables you to create a layer of metadata, called the End User Layer that hides the complexity of the database from users and reflects the business areas specific to your company data.
2. **Discoverer Plus** - enables you to easily query a database, save query results in workbooks, analyze the results, and format a report.
3. **Discoverer Viewer** - enables you to view workbooks created in Discoverer Plus using nothing more than a web browser.

Business Intelligence: is software it is going to give solutions, Business Solutions and predict future trends.

Discoverer: is one of the business intelligence tool used to answer business queries by creating ad-hoc queries with out depending on IT specialists (i.e. Developers). The other Business Intelligence tools are Cagnos, Informatica, etc.



The main tools of discoverer are

1. Discoverer Administrator
2. Discoverer Desktop
3. Discoverer Viewer
4. Discoverer Plus

From release 11.5.9 onwards discoverer is fully integrated with application server. All the above tools are based on end user layer.

End User Layer: is a Meta Data (i.e. Data about data) or a logical database, which is used to hide the complexicity of the database. End user layer can be defined on database users or non-database users. Non-database users are also called as application users. One database or application user can have only one end user layer. End user layer defined on end user can also give grant access to other database users. An end user layer is a collection of one or more business areas.

Discoverer Administrator: functions are

- Gathers the requirements of end user
- Creates end user layer
- Creates Business Area
- Load Data in to Business Area
- Give Control access to Business Area
- Creates Joins and Conditions
- Send to Users

Business Area: is a logical group of tables and views of a particular end user layer. B.A is a collection of folders. A folder is a table loaded fro m the database. Folder is a collection of items. Each item represents a column in the table. We can also define special items for calculation purpose.

Discoverer Desktop: This tool is going to be used by the end user in standalone machine. It is used to access end user layers. Using desktop we can create worksheets. We can analyze the data. We can create charts (Graphs).

Discoverer Viewer: This tool is going to access the end user layer at client site on the Web in HTML format only. Using this tool we can analyze the data and create graphs.

Discoverer plus: is used to access end user layer on the Web in applet format. Using this tool we can create work sheets, analyze data and create charts.