# R12.2 Development and Deployment of Customizations

John Peters

2/19/2014

# About the Presenter

- John Peters, JRPJR, Inc
  - Independent Consultant based in the San Francisco Bay Area
  - Worked with Oracle EBS since 1993
  - OAUG Workflow SIG Coordinator
  - Founding board member of the Northern California OAUG GEO
  - Presented many papers at many conferences: http://jrpjr.com (paper archives)
  - john.peters@jrpjr.com

- Primarily Technology Focus
  - Extension/Customization Design and Development
  - DBA/System Administration

# This is just a cursory introduction

If we were to cover all of the 12.2 Development Details it would take at least a full day.

Please refer to:

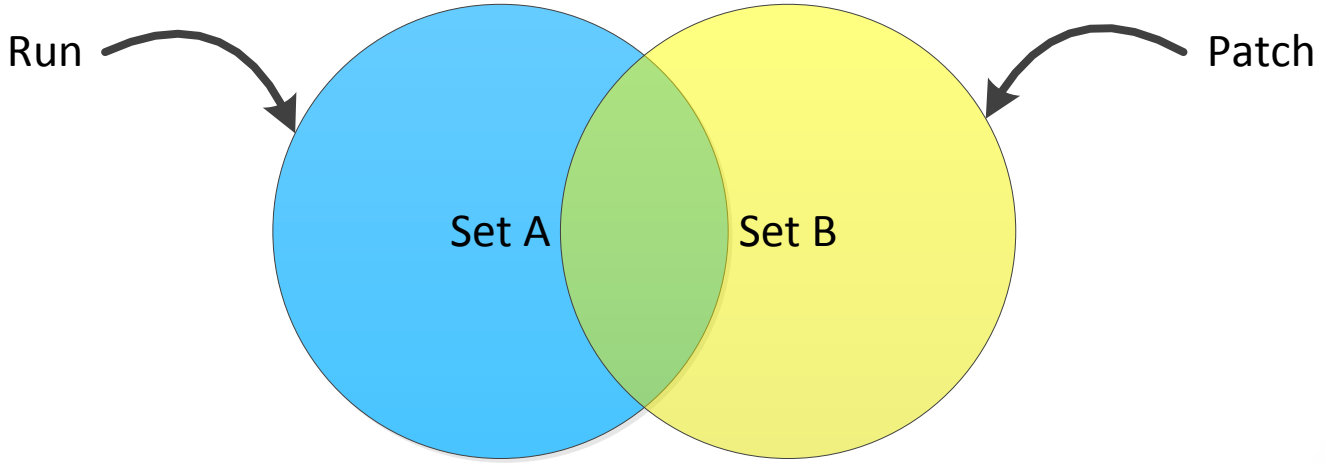
1. Oracle® E-Business Suite Developer's Guide Release 12.2 (Part No. E22961-09)
2. Deploying Customizations in Oracle E-Business Suite Release 12.2 (MOS Doc ID 1577661.1)
3. Using the Online Patching Readiness Report in Oracle E-Business Suite Release 12.2 (MOS Doc ID 1531121.1)
4. Oracle E-Business Suite Release 12.2: Online Patching FAQ (MOS Doc ID 1583902.1)

It is all logical and well thought out

…..And yes it does work…..

# Edition Based Redefinition

- EBR allows us to modify database objects as a “Set”, while in use, thus minimizing downtime.
- Users will see “Set A” while you changing “Set B”.
- You can switch the “Set” reference in the database session which will allow an almost instantaneous cutover.



# Editions Include

For all of these you will have a RUN and PATCH edition:

- Database Objects
- Seed Data, Meta Data
- Filesystem Objects

# EBR in the DB

- Editionable DB Objects are:
  - Synonyms
  - Views
  - All PL/SQL Object Types
    Function, Library, Package Spec/Body, Procedure, Trigger, Type and Type Body

- Everything else in the DB is non-Editionable

- Rules
  - A Non-Edititioned Object cannot depend on an Editioned Object
  - An Editioned  Object (View) cannot be involved in a Foreign Key Constraint
  - An Abstract Data Type cannot be both editioned and evolved

# Editions in the Filesystem

Exact Copy of the Filesystems before you start patching

- Once you patch RUN will contain the code you are currently using
- PATCH will contain the code you are changing

# Determine EBS Editions

- Editions are sync'ed between the OS filesystem and database
- Two Edition Definitions:
  - Runtime
  - Patch
- How to tell your Edition:
- OS:

```
echo $FILE_EDITION
```

- DB:

```
select ad_zd.GET_EDITION_TYPE,
       ad_zd.GET_EDITION
  from dual;
```

# Set EBS Edition

- How to set your Edition:
- OS:

```
source /oracle/ebs122/EBSapps.env run
or
source /oracle/ebs122/EBSapps.env patch
echo $RUN_BASE          - the run filesystem
echo $PATCH_BASE        - the patch filesystem
```

- DB:

```
ad_zd.SET_EDITION('RUN')
or
ad_zd.SET_EDITION('PATCH')
```

# Check EBS Edition Status

- How to check the status of patching
- In the OS:

```
adop -status
```

- From the DB:

```
sqlplus apps @ADZDSHOWED.sql
```

# EBR and Data

- Table Data is maintained through triggers
- Seed Data is handled by striping it with an Edition column
- More on this later

# EBS Global Standards Compliance Checker

Reviews schemas that are registered with the EBS to ensure they will support EBR and On Line Patching

- $AD_TOP/sql/ADZDPSUM.sql
  - Lists schemas containing objects that reference EBS objects that are not editioned.
  - You must manually correct these.
- $AD_TOP/sql/ADZDPMAN.sql
  - Lists objects that violate EBR standards.
  - You must manually correct these.
- $AD_TOP/sql/ADZDPAUT.sql
  - Lists objects that violate Online Patching Enablement standards.
  - Adjusted automatically by the Online Patching Enablement patch, no action required.
- $AD_TOP/sql/ADZDDBCC.sql
  - Lists objects that violate Online Patching Development standards.
  - You must manually correct these.

- Delivered as a standalone patch
  - 16236081:R12.AD.C 12.2.2
  - 16236081:R12.AD.B 12.1
  - 16236081:R12.AD.A 12.0
  - 16236081:11i

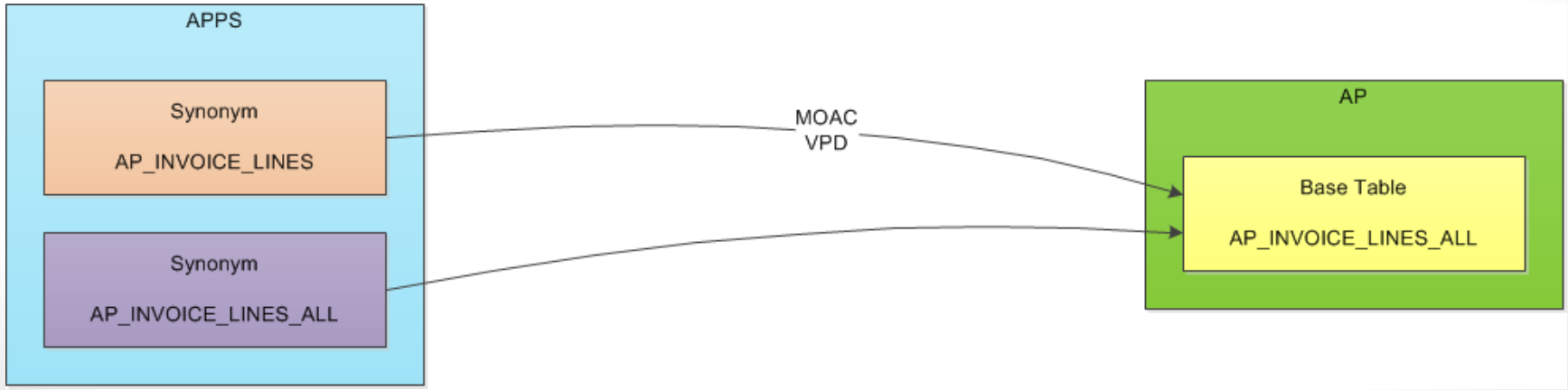
# DB Object Names

- Table Names must be unique at 29 bytes or less
  - The Editioning Views will have suffix of # added to them automatically
  - This is done with a substr(table_name,1,29)
- Column Names must be 28 bytes or less
  - A Revised Column Name has the form:
  - <logical_column_name>#<version_tag>
  - Version_tag is a string of the form: [0-9A-Z]
- Forward Cross-Edition Triggers
  - <table_name>_F<change_number>
- Suffix + are the EBR DB Triggers on Seed Data
- A Materialized View Definition must be stored in an ordinary view called MV_NAME||'#'

# DB Objects pre 12.2

I am using a simple example of AP_INVOICE_LINES_ALL (post 11i)

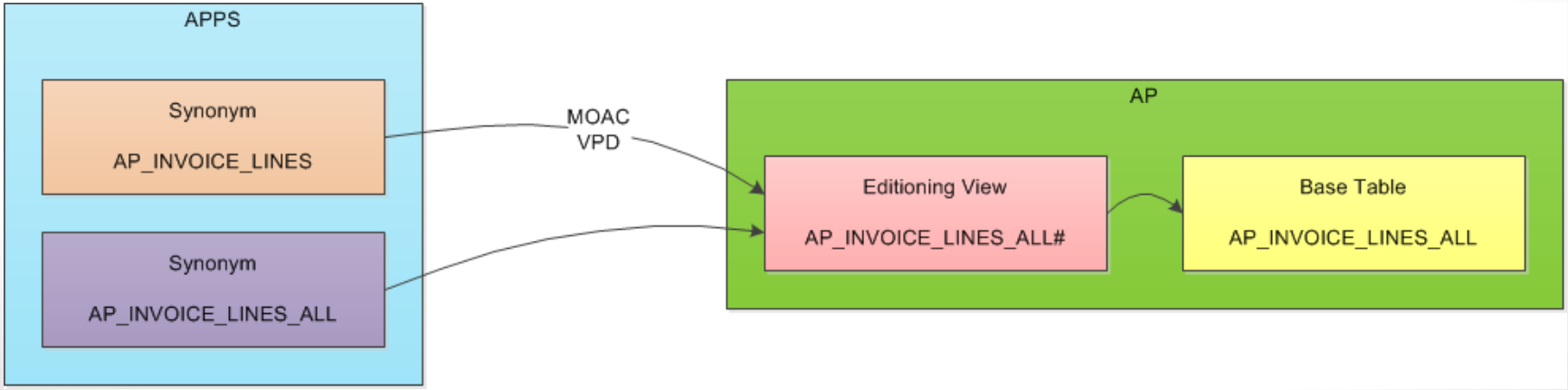
- Base table owned by AP
- Synonym for MAOC object owned by APPS
- Synonym for _ALL object owned by APPS



# DB Objects post 12.2

What changed in 12.2?

- Editioning View added (# suffix)
- APPS synonyms now point to the Editioning View



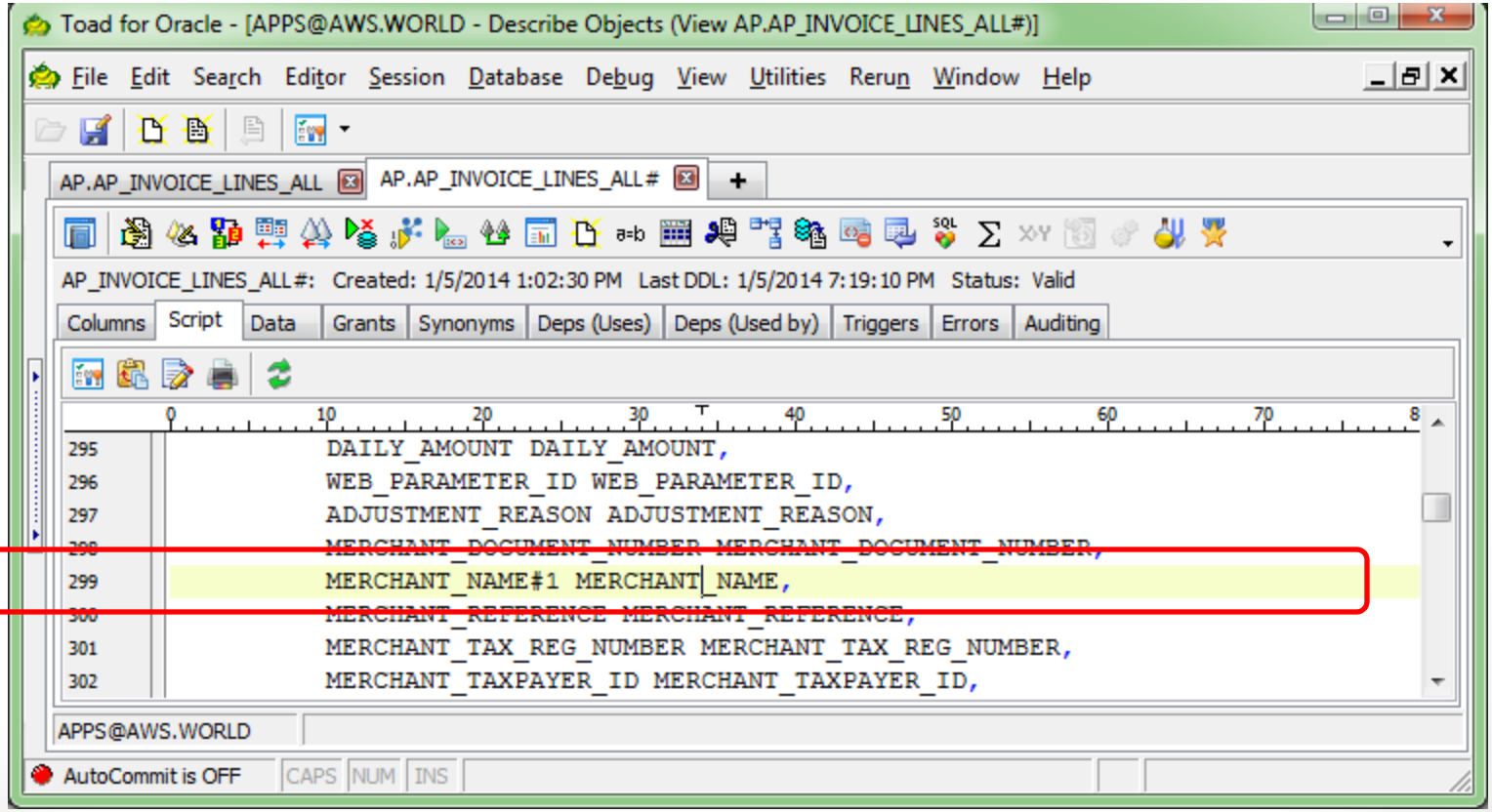
# Where is the Editioning Done?

- A Virtual Private Database (VPD) Policy must be on the Editioning View or Table Synonym, not the base table.

- This provides the filtered representation of the Editioning View to ensure you get a consistent representation of the data.

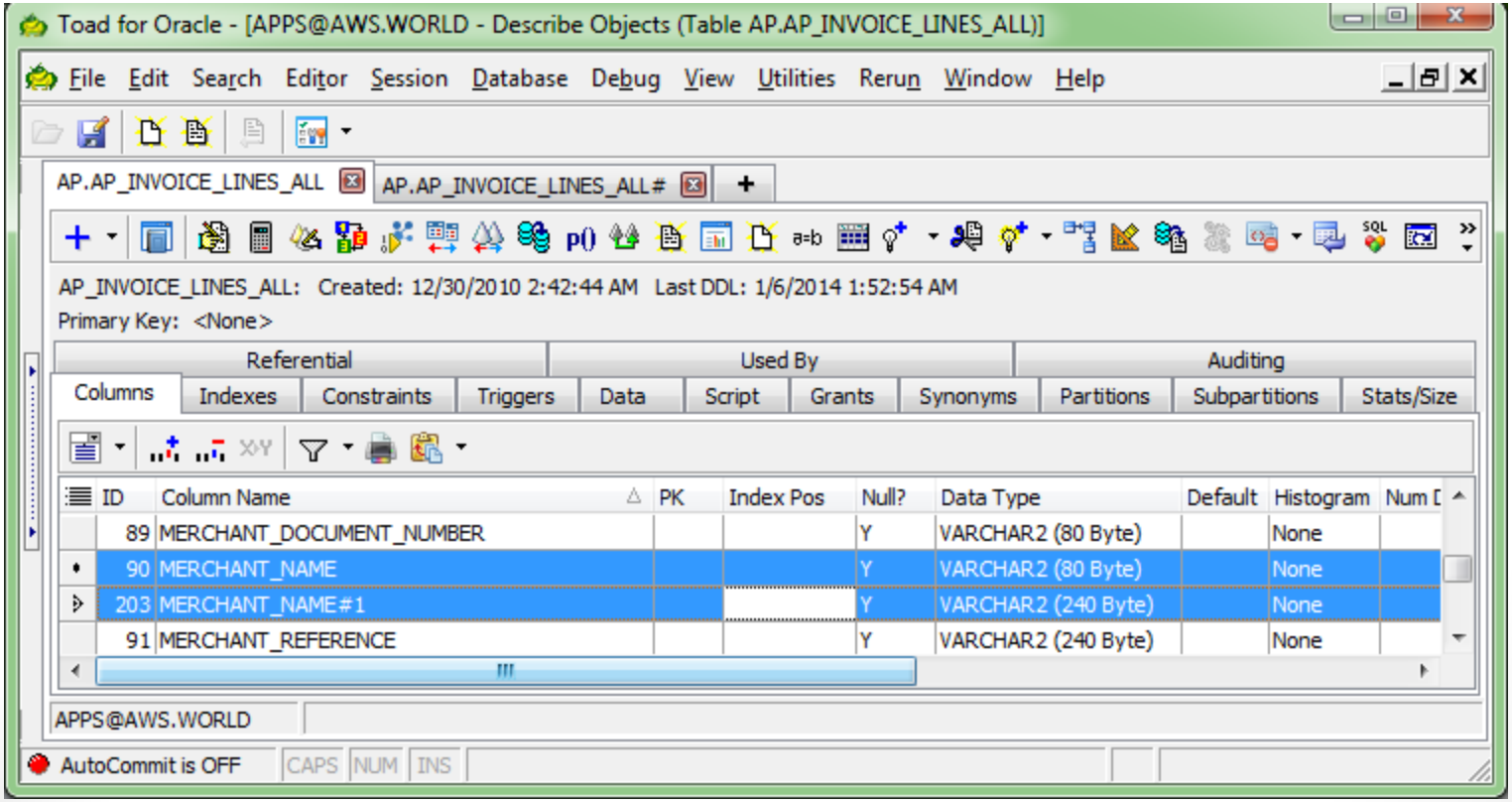
# Why use the Editioning View

- You must use the Editioning View in order to get the correct column representation, that is consistent for your Edition
- We can see below the editioning view's create statement for AP.AP_INVOICE_LINES_ALL#



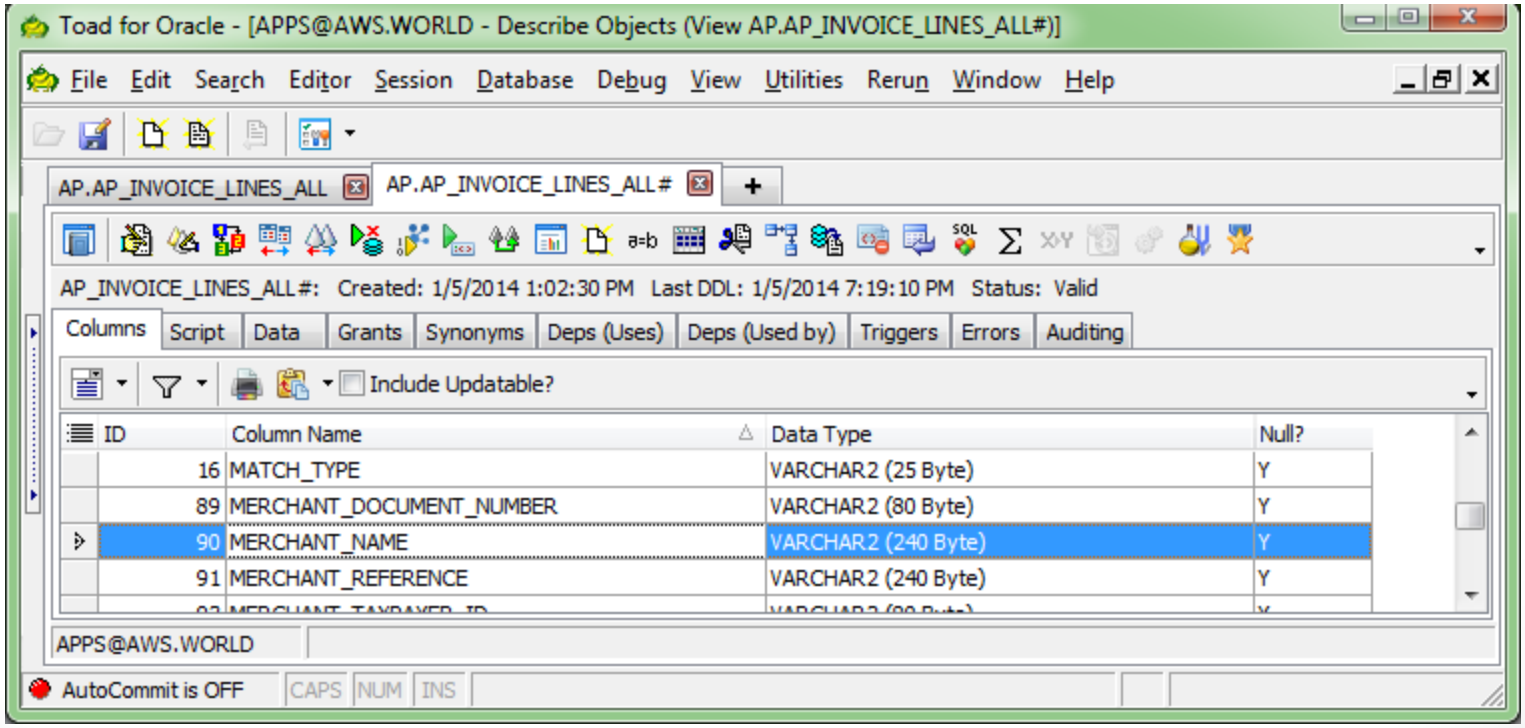
# AP.AP_INVOICE_LINES_ALL

- AP.AP_INVOICE_LINES_ALL has the columns:
  - MERCHANT_NAME varchar2(80)
  - MERCHANT_NAME#1 varchar2(240)



# APPS.AP_INVOICE_LINES_ALL

- APPS.AP_INVOICE_LINES_ALL has one column:
  - MERCHANT_NAME  varchar2(240)



# Development Steps

General Steps

1. Develop customization in the run edition of your non-PROD environment
   - Both DB and OS File System
   - Object specific steps we will talk about below
   - Recompile invalids (ad_zd.compile)
2. Create the patch (manually create patch actions)
3. Test the patch

**Deploying Customizations in Oracle E-Business Suite Release 12.2 (**MOS Doc ID 1577661.1)

# Dev Steps - Tables

An Edition Synonym will point to the correct version of the Table

1. Make your table changes
2. Regenerate the edition view (ad_zd_table.patch)
3. Upgrade table for Edition Storage (ad_zd_seed.upgrade)
4. Create loader LCT for Seed Data
5. Create Forward Crossedition Trigger (FCET)
6. Create Reverse Crossedition Trigger (RCET)
7. Extract updated table defintion (xdfgen.pl)
8. Extract Seed Data (FNDLOAD)
9. Recompile invalids (ad_zd.compile)

# Seed Data

Seed Data tables must include a new column ZD_EDITION_NAME

This is used to present a consistent view of the data based on the Edition.

1. Create initial table definition
   Table must go in APPS_TS_SEED tablespace
2. Upgrade table to support Editioned Storage (ad_zd_seed.upgrade)
   This adds the column ZD_EDITION_NAME
3. Manually insert new Seed Data records into table
4. Create a Loader Control File for Seed Data
   FNDLOAD *.LCT file
5. Extract Seed Data

```
FNDLOAD apps/<apps_pwd> 0 Y DOWNLOAD my_table.LCT my_table.ldt my_table
```

# How is Seed Data Filtered?

- Seed Data is filtered using the ZD_EDITION_NAME by a Virtual Private Database (VPD) Policy

# How to Maintain Table Data


We only have to maintain the table data from Apply to Cutover.

Updating an existing Table Column

- Existing MY_COLUMN has values of (RED,GREEN)
- New MY_COLUMN will have values of (ONE,TWO,THREE)

How is this done

- Create a new MY_COLUMN#1
- Editioning View Maps:

  RUN – MY_COLUMN => MY_COLUMN

  PATCH – MY_COLUMN => MY_COLUMN#1

How is MY_COLUMN#1 populated until cutover is completed?

# Forward Crossedition Trigger

Forward Crossedition Trigger (FCET)

- This is a database trigger the developer has to create
- Maps the old values to new values
  - RED => ONE
  - GREEN => THREE
  - Nothing maps to TWO since this might be a new value with no equivalent meaning

Multiple Updates to the same table

1. F1
2. F2 follows F1
3. F3 follows F2
4. … and so on …

# Forward Crossedition Trigger

Multiple Updates to the same table

1. F1
2. F2 follows F1
3. F3 follows F2
4. … and so on …

```
create or replace trigger XYZ_MY_TABLE_F2
  before insert or update on &1..xyz_my_table
  for each row forward crossedition
  FOLLOWS XYZ_MY_TABLE_F1
  disable
```

# Reverse Crossedition Trigger

Reverse Crossedition Trigger (RCET)

- Used to back populate a Not Null or Unique Index column in a Seed Data table.
- As part of your patching you will be loading new Seed Data and you need to back populate the old column.
- The RCET must populate values that will satisfy a Unique Index, in which case you might need a new sequence to just generate values.

# FCET and RCET Triggers

- It is the developers responsibility to create the FCET and RCET triggers.
- EBS won’t know the exact logic to apply.
- So you as the developer must code that logic.

# Or Another Alternative

- The majority of this complexity around data maintenance during the patching cycle is to support On Line Patching.
- If you do not perform On Line Patching, and you shutdown access to your system as you move from Apply to Cutover then you might not need to handle the data maintenance tasks using triggers.
- However, you will still need to follow the EBS standards for customizations.

# Dev Steps – Global Temp Tables

Use Synonym to point to Global Temp Table

1. Make your table change in a new temp table with a different name
2. Extract updated table defintion (xdfgen.pl)
3. Create Helper Script
   - The Global Temp Table can not be editioned and it can not be changed while users are using it
   - Instead a synonym is created to point to the new temp table
   - This synonym is the permanent logical name
   - During cutover the helper script will change the synonym to point to a new temp table
4. Recompile invalids (ad_zd.compile)

# Dev Steps – Materialized Views

An effectively-editioned Materialized View includes both a Logical View (managed by the developer) and a Materialized View (generated by Online Patching).

The Logical View is an ordinary database view, and is therefore an editioned object that can be changed in the Patch Edition without affecting the Run Edition.

But the generated Materialized View is a non-editioned object, meaning the definition and content of the materialized view is shared across all editions.

In order to avoid breaking the running application during an online patch, the system must defer materialized view regeneration until the cutover phase, when the application is down.

1. Create a Logical View
2. Generate the Materialized View (ad_zd_mview.upgrade)
3. Extract updated table defintion (xdfgen.pl)
4. Recompile invalids (ad_zd.compile)

# Dev Steps - Views

Views are just like normal View Development.

Just create an Edition Synonym to point to the correct view.

# Dev Steps – PL/SQL

PL/SQL Development is same as today.

PL/SQL code is stored in the data dictionary by Edition.

So based on the Edition you will run a specific version.

Everything within a Edition should be consistent.

# Dev Steps – DB Triggers

- If your development includes a new or changed DB Trigger, it must be on the Editioning View, not the Base Table.

- **Tip:** The simplest way to comply with this standard is to create triggers on the APPS table synonym. The resulting trigger will be created on whatever the synonym points to, which will be the editioning view if one exists.

# Dev Steps – Concurrent Programs

- Same as you do today
- You will use FNDLOAD to load the Concurrent Program definition as part of your patching
- Use FNDLOAD to export your Concurrent Program Definition.
- You will use FNDLOAD to import the Concurrent Program Definition as part of your patching

- No more manually entering/updating the program definition

# Dev Steps – Forms

- Same as you do today
- Use FNDLOAD to export your Forms Definition.
- You will use FNDLOAD to import the Forms Definition as part of your patching

# Dev Steps – Application Framework

- Export your modified Framework Personalization Using the XMLExporter or XLIFF Extractor utility

- Patch Action to Import your modified Framework Using the XMLImporter or XLIFFImporter utility

# Dev Steps – Custom Wed ADI

- No PL/SQL API support any longer, use the UI to create the defintion
- Use FNDLOAD to export your Custom Web ADI Definition.
- You will use FNDLOAD to import the Custom Web ADI Definition as part of your patching

# Dev Steps – Workflow

- Use Workflow Definitions Loader or Workflow Builder to export your Custom Workflow Definition.
- You will use Workflow Definitions Loader to import your Custom Workflow Definition as part of your patching.

- Use Workflow XML Loader utility to export your Event and Subscription metadata.
- You will use Workflow XML Loader to import your Event and Subscription metadata as part of your patching.

# Dev Steps – BI Publisher

1. Export your BI Publisher Definition using a combination of FNDLOAD and  XDOLoader utility.
2. Import your BI Publisher Definition using a combination of FNDLOAD and  XDOLoader utility as part of your patching.

# Many, Many More Object Types

- Please refer to the referenced Oracle Documents.
- I hope to revise this presentation as time goes by and I find other tricks and tips.

# Do's and Don'ts

- Don't
  - Call DB objects directly: ap.AP_INVOICE_LINES_ALL
  - Query seed data directly, use the Synonym with the VPD applied.
  - Do not drop an existing table until the Cleanup phase of patch execution.
  - Do not drop an existing column. Columns that are replaced by new revised columns will be dropped automatically at cleanup. Dropping a logical column is not supported.
  - Do not rename an existing table.

# Do’s and Don’ts

- Do’s
  - Take time to learn the new development methodologies.
  - Test your migrations first to a non-DEV instance.
  - You will use Patches to load and modify your customizations using the various utilities Oracle ATG provides.
  - Always use the APPS synonyms for the objects.
  - Unique Index on a Seed Data Table must include ZD_EDITION_NAME.

# What missing?

- Third Party Add-Ons
- Integrations
- Tools like Applications Express
  - Probably Import the APEX Application during the CUTOVER Phase

Thanks for sticking through this, just remember it is logical and try not to get overwhelmed.

This presentation will be posted on the NorCal OAUG Web Site.