Oracle Forms to Oracle APEX Conversion Guide

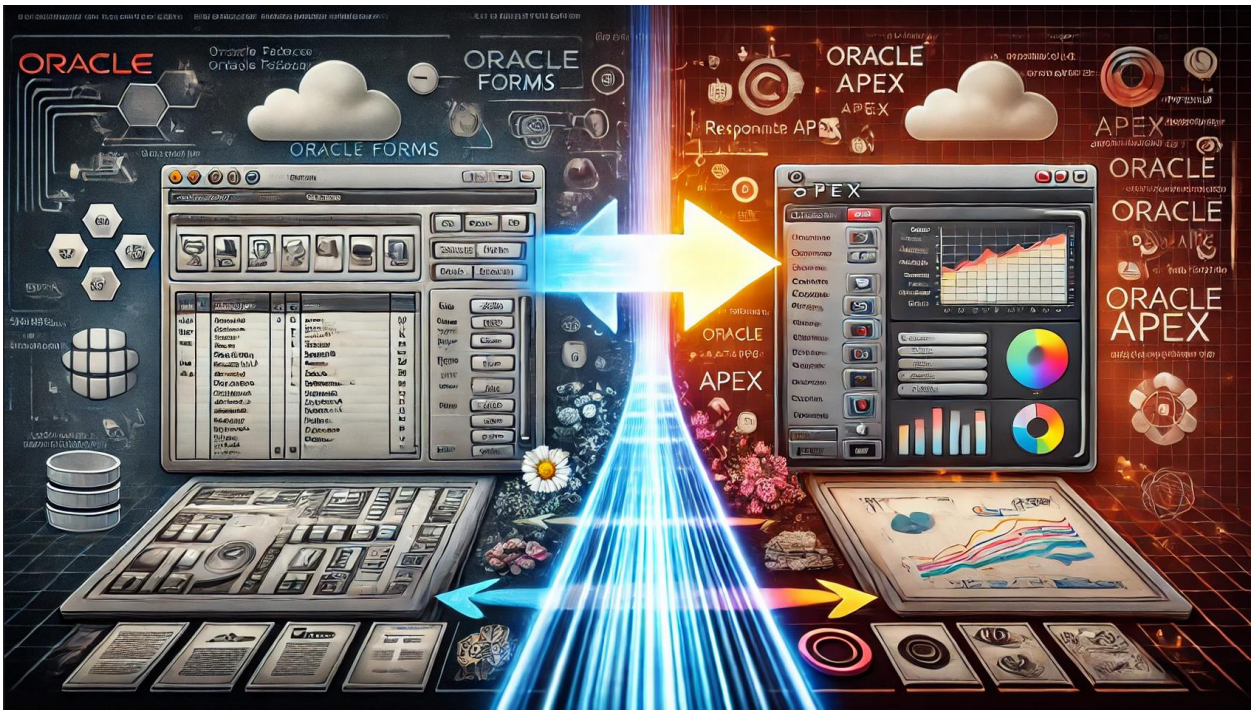

Table of Contents

1 Oracle Forms Elements to Oracle APEX Elements 4

1.1 General Mapping of Oracle Forms Elements to Oracle APEX Elements 4

1.1.1 Canvas → Page 4

1.1.2 Window → Page or Dialog 4

1.1.3 Block → Region 4

1.1.4 Items → Page Items 5

1.1.5 Triggers → Dynamic Actions / Processes 5

1.1.6 LOVs (List of Values) → Shared LOVs 6

1.1.7 Record Groups → SQL Queries 6

1.1.8 Alerts → Dialogs / Notifications 6

1.1.9 Program Units → Database Packages / Shared Components 7

1.1.10 Visual Attributes → Themes / CSS 7

1.1.11 Menus → Navigation Menu / Breadcrumbs 7

1.1.12 Parameters → Page Items / Application Items 7

1.1.13 Other Mappings 8

1.1.14 Summary 8

2 Mapping Oracle Forms Triggers to Oracle APEX 8

3 Database Metadata Tables for Oracle Forms Elements/Objects 11

3.1 forms 11

3.2 blocks 11

3.3 items 12

3.4 triggers 12

3.5 properties 13

3.6 program_units 13

3.7 canvas 14

3.8 windows 14

3.9 alerts 14

3.10 menu_modules 15

3.11 visual_attributes 15

3.12 lovs (List of Values) 16

3.13 record_groups 16

3.14 relations 17

3.15 parameters 17
3.16 coordinates 18
3.17 validation_rules 18
4 Example of Oracle Form XML Extract 18
5 Oracle Form XML to Oracle Tables Parsers 21
5.1 PL/SQL to Parse the XML Extract 21
6 Levels of Effort for Converting Oracle Forms to APEX Pages 24
6.1 Factors Influencing LOE 24
6.1.1 Form Complexity: 24
6.1.2 Number of Elements: 24
6.1.3 Business Logic: 24
6.1.4 Dependencies: 24
6.1.5 Styling Requirements: 24
6.2 Estimated LOE by Complexity 25
6.3 Breakdown of Conversion Tasks 25
6.3.1 Analysis and Planning 25
6.3.2 Recreate UI 25
6.3.3 Business Logic 25
6.3.4 Integrations 25
6.3.5 Testing and Refinement 25
6.4 Typical Conversion Timeline 26
6.4.1 Efficiency Considerations 26
6.4.2 Summary 26

1 Oracle Forms Elements to Oracle APEX Elements

1.1 General Mapping of Oracle Forms Elements to Oracle APEX Elements

1.1.1 Canvas → Page

Oracle Forms Element	APEX Equivalent	Description
Canvas (Content)	Page (Normal)	Represents a primary screen or workspace in Oracle Forms.
Canvas (Stacked)	Region (Static Content)	Used for layered visual elements; converted to APEX regions within the same page.
Canvas (Tab)	Tab Container	Map to APEX tab regions or navigation tabs.
Canvas (Toolbar)	Page Header/Region	Represents toolbar functionality; can be moved to the page header or a separate region.

1.1.2 Window → Page or Dialog

Oracle Forms Element	APEX Equivalent	Description
Window	Page (Modal Dialog)	Forms windows that appear as pop-ups or separate workspaces map to APEX modal dialogs.
Multiple-Document Interface	Multiple Pages	Use APEX pages with navigation to mimic MDI functionality.

1.1.3 Block → Region

Oracle Forms Element	APEX Equivalent	Description
Data Block	Interactive Grid	Used for multi-record data entry and management.
Control Block	Static Content Region	Used for static controls or non-database regions.

1.1.4 Items → Page Items

Oracle Forms Element	APEX Equivalent	Description
Text Item	Text Field	Standard text input.
Display Item	Display Only	Read-only display items.
Checkbox	Checkbox	Boolean inputs.
Radio Button	Radio Group	Choose one option from a group.
List Item (Dropdown)	Select List	Dropdown menus for selecting values.
List Item (Poplist)	Popup LOV	Popup LOVs for selecting values.
Image Item	Image Display / File Browse	For displaying images or uploading files.
Button	Button	Execute actions; map triggers (WHEN-BUTTON-PRESSED) to APEX button actions.
Hidden Item	Hidden Item	Store values that are not displayed on the page.

1.1.5 Triggers → Dynamic Actions / Processes

Oracle Forms Trigger	APEX Equivalent	Description
WHEN-NEW-FORM-INSTANCE	Page Load Process	Initializes the page when loaded.
WHEN-BUTTON-PRESSED	Button Click + DA	Maps to a Dynamic Action or a PL/SQL Process triggered by a button click.
WHEN-VALIDATE-ITEM	Validation Rule	APEX validations for user input.
KEY-NEXT-ITEM	Tab Navigation	APEX handles navigation between items automatically.
WHEN-MOUSE-CLICK	Dynamic Action (Event)	Use Dynamic Actions with JavaScript events.

Oracle Forms Trigger	APEX Equivalent	Description
POST-BLOCK / PRE-BLOCK	Dynamic Actions	Use region-level actions to replicate POST/PRE-BLOCK behavior.

1.1.6 LOVs (List of Values) → Shared LOVs

Oracle Forms LOV	APEX Equivalent	Description
LOV (Static)	Shared LOV	Static LOVs map directly to APEX Shared LOVs.
LOV (Dynamic)	SQL Query LOV	Dynamic SQL queries for LOVs.
LOV (Popup)	Popup LOV	Mimic popup LOV behavior using APEX Popup LOVs.

1.1.7 Record Groups → SQL Queries

Oracle Forms Record Group	APEX Equivalent	Description
Record Group	SQL Query	Convert record groups to SQL queries.
Multi-Query Record Group	APEX Collections	Use collections for multi-query or dynamic datasets.

1.1.8 Alerts → Dialogs / Notifications

Oracle Forms Alert	APEX Equivalent	Description
Alert (Stop)	Dialog Box (Error)	Critical alerts can be displayed as error dialogs.
Alert (Caution)	Warning Notification	Warning messages displayed as inline or dialog warnings.
Alert (Note)	Success Notification	Informational messages displayed as success notifications.

1.1.9 Program Units → Database Packages / Shared Components

Oracle Forms Element	APEX Equivalent	Description
Procedures	Database Package	Convert Forms procedures into PL/SQL procedures in a database package.
Functions	Database Package	Convert Forms functions into database functions.
Triggers	Page Process / DA	Map Forms triggers to server-side processes or dynamic actions.

1.1.10 Visual Attributes → Themes / CSS

Oracle Forms Element	APEX Equivalent	Description
Font, Color, Style	CSS / Theme	Use APEX themes and templates for styling.
Conditional Formatting	CSS / Dynamic Actions	Apply dynamic styles using APEX conditions or CSS classes.

1.1.11 Menus → Navigation Menu / Breadcrumbs

Oracle Forms Menu	APEX Equivalent	Description
Main Menu	Navigation Menu	Map to APEX navigation menus.
Context Menu	Dynamic Actions	Context-sensitive actions handled using dynamic menus or JavaScript.

1.1.12 Parameters → Page Items / Application Items

Oracle Forms Parameter	APEX Equivalent	Description
Global Parameters	Application Items	Application-wide values accessible across sessions.
Block-Level Parameters	Page Items	Page-level items for passing values between regions.

1.1.13 Other Mappings

Oracle Forms Element	APEX Equivalent	Description
LOV Key Mapping	Cascading LOV	Filter LOV values based on other fields.
Query Mode	Report Filters	APEX report filters allow users to query data dynamically.
Validation Units	Validation Rules	Use validation rules for enforcing data integrity.
Built-in Subprograms	PL/SQL API / Shared Components	Implement using APEX APIs or custom PL/SQL.

1.1.14 Summary

- **Pages** in APEX replace **Canvases** and **Windows**.
- **Regions** replace **Blocks**.
- **Page Items** replace Forms **Items**.
- **Dynamic Actions** and **Processes** handle Forms **Triggers**.
- **Themes and Templates** replace Forms **Visual Attributes**.

2 Mapping Oracle Forms Triggers to Oracle APEX

Forms Trigger	APEX Equivalent	Description/Implementation
WHEN-NEW-FORM-INSTANCE	Page Load Process	Use a Page Load Process to initialize data or perform setup tasks when the page is loaded.
WHEN-NEW-BLOCK-INSTANCE	Region Dynamic Action / Before Header Process	Execute actions or logic when navigating to a new block. Use Dynamic Actions or Region-specific logic.
WHEN-NEW-RECORD-INSTANCE	Interactive Grid DA / Row-Level Process	Use Dynamic Actions in Interactive Grids to initialize or validate record-specific logic.

Forms Trigger	APEX Equivalent	Description/Implementation
WHEN-NEW-ITEM-INSTANCE	Dynamic Action (Focus Event)	Use a Dynamic Action with the "Focus" event to handle item-level navigation or initialization.
WHEN-VALIDATE-ITEM	Validation Rule	Use APEX validation rules to validate individual page items.
WHEN-VALIDATE-RECORD	Interactive Grid Validation	Use validations for rows in Interactive Grids to replace record-level validation logic.
WHEN-MOUSE-CLICK	Dynamic Action (Click Event)	Replace with a Dynamic Action triggered by a "Click" event on buttons or regions.
WHEN-BUTTON-PRESSED	Button Action / Process	Map to a button click with an associated Page Process or Dynamic Action.
POST-BLOCK	After Header Process / Region Process	Use an After Header Process for logic to be executed after navigating away from a block.
PRE-BLOCK	Before Header Process / Region Process	Execute logic before entering a block using a Before Header Process or Region-specific Dynamic Action.
POST-QUERY	After Refresh Dynamic Action	Use Dynamic Actions tied to an Interactive Report/Grid refresh event.
PRE-QUERY	Before Header Process	Execute logic before querying data, typically using a Page Process or a PL/SQL Function Returning SQL Query.
KEY-NEXT-ITEM	Automatic Navigation / Dynamic Action	Use APEX's default tab navigation, or create a Dynamic Action for custom navigation logic.
KEY-PREV-ITEM	Automatic Navigation / Dynamic Action	Similar to KEY-NEXT-ITEM but for navigating to the previous item.
KEY-NEXT-BLOCK	Dynamic Action	Handle navigation between blocks with custom PL/SQL in a Dynamic Action.
KEY-PREV-BLOCK	Dynamic Action	Same as KEY-NEXT-BLOCK but for navigating to the previous block.
KEY-COMMIT	Save Button + Process	Replace with an explicit Save button tied to a Page Process for committing data.

Forms Trigger	APEX Equivalent	Description/Implementation
KEY-EXIT	Exit Button + Process	Use an Exit button to redirect users to a different page or end the session.
ON-COMMIT	After Submit Process	Use After Submit Page Processes to execute commit-related logic.
ON-ERROR	Exception Handling in PL/SQL Processes	Handle errors with APEX exception handling or inline PL/SQL exception blocks.
ON-LOGON	Authentication Scheme	Replace with APEX authentication schemes to handle user login and session initialization.
ON-MESSAGE	Custom Notification / Validation Messages	Use APEX validations, notifications, or PL/SQL to manage custom messages.
ON-ROLLBACK	PL/SQL Process	Execute custom logic in response to rollbacks using explicit PL/SQL in processes or button actions.
WHEN-DATABASE-RECORD	Row Process in Interactive Grid	Replace with row-specific Dynamic Actions or server-side PL/SQL processes.
WHEN-IMAGE-PRESSED	Dynamic Action (Click Event)	Use a Dynamic Action triggered by clicking an image item.
WHEN-LIST-CHANGED	Dynamic Action (Change Event)	Use a Dynamic Action triggered by the "Change" event for Select List items.
WHEN-TIMER-EXPIRED	JavaScript Timer + Dynamic Action	Replace timers with JavaScript and trigger Dynamic Actions when conditions are met.
POST-TEXT-ITEM	Dynamic Action (Change Event)	Use the "Change" event to handle logic after the user modifies a text field.
PRE-TEXT-ITEM	Dynamic Action (Focus Event)	Use the "Focus" event to handle logic before the user interacts with a text field.

3 Database Metadata Tables for Oracle Forms Elements/Objects

3.1 forms

Stores metadata for the form.

sql

Copy code

```
CREATE TABLE forms (
  form_id       NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_name     VARCHAR2(255) NOT NULL,
  description   VARCHAR2(4000),
  version       VARCHAR2(50),
  created_date  DATE DEFAULT SYSDATE,
  last_updated  DATE DEFAULT SYSDATE
);
```

3.2 blocks

Represents blocks within the form.

sql

Copy code

```
CREATE TABLE blocks (
  block_id      NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id       NUMBER NOT NULL REFERENCES forms(form_id),
  block_name    VARCHAR2(255) NOT NULL,
  description   VARCHAR2(4000),
  query_data_source VARCHAR2(255),
  created_date  DATE DEFAULT SYSDATE
);
```

3.3 items

Represents items (fields) in blocks.

sql

Copy code

```
CREATE TABLE items (
  item_id       NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  block_id      NUMBER NOT NULL REFERENCES blocks(block_id),
  item_name     VARCHAR2(255) NOT NULL,
  item_type     VARCHAR2(255),
  data_type     VARCHAR2(50),
  length        NUMBER,
  prompt        VARCHAR2(255),
  created_date  DATE DEFAULT SYSDATE
);
```

3.4 triggers

Captures triggers defined in the form, block, or item.

sql

Copy code

```
CREATE TABLE triggers (
  trigger_id    NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  parent_type   VARCHAR2(50) CHECK (parent_type IN ('FORM', 'BLOCK', 'ITEM')),
  parent_id     NUMBER NOT NULL,
  trigger_name  VARCHAR2(255) NOT NULL,
  trigger_event VARCHAR2(255),
  trigger_code  CLOB,
  created_date  DATE DEFAULT SYSDATE
);
```

3.5 properties

Stores properties for forms, blocks, or items.

sql

Copy code

```
CREATE TABLE properties (
  property_id    NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  parent_type    VARCHAR2(50) CHECK (parent_type IN ('FORM', 'BLOCK', 'ITEM')),
  parent_id      NUMBER NOT NULL,
  property_name  VARCHAR2(255) NOT NULL,
  property_value VARCHAR2(4000),
  created_date   DATE DEFAULT SYSDATE
);
```

3.6 program_units

Represents PL/SQL program units defined in the form.

sql

Copy code

```
CREATE TABLE program_units (
  program_unit_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id       NUMBER NOT NULL REFERENCES forms(form_id),
  unit_name     VARCHAR2(255) NOT NULL,
  unit_type     VARCHAR2(50) CHECK (unit_type IN ('PROCEDURE', 'FUNCTION', 'PACKAGE')),
  unit_code     CLOB,
  created_date  DATE DEFAULT SYSDATE
);
```

3.7 canvas

Represents canvases (visual layouts) in the form.

sql

Copy code

```
CREATE TABLE canvas (
  canvas_id     NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id       NUMBER NOT NULL REFERENCES forms(form_id),
  canvas_name   VARCHAR2(255) NOT NULL,
  canvas_type   VARCHAR2(50),
  created_date  DATE DEFAULT SYSDATE
);
```

3.8 windows

Represents windows in the form.

sql

Copy code

```
CREATE TABLE windows (
  window_id     NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id       NUMBER NOT NULL REFERENCES forms(form_id),
  window_name   VARCHAR2(255) NOT NULL,
  created_date  DATE DEFAULT SYSDATE
);
```

3.9 alerts

Stores alert definitions in the form.

sql

Copy code

```
CREATE TABLE alerts (
```

```
  alert_id      NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id       NUMBER NOT NULL REFERENCES forms(form_id),
  alert_name    VARCHAR2(255) NOT NULL,
  alert_message  VARCHAR2(4000),
  alert_type    VARCHAR2(50) CHECK (alert_type IN ('STOP', 'CAUTION', 'NOTE')),
  created_date   DATE DEFAULT SYSDATE
);
```

3.10 menu_modules

Represents menus attached to the form.

sql

Copy code

```
CREATE TABLE menu_modules (
  menu_id       NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id       NUMBER NOT NULL REFERENCES forms(form_id),
  menu_name     VARCHAR2(255) NOT NULL,
  menu_code     CLOB,
  created_date   DATE DEFAULT SYSDATE
);
```

3.11 visual_attributes

Represents visual attributes for items, blocks, or canvases.

sql

Copy code

```
CREATE TABLE visual_attributes (
  attribute_id   NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id       NUMBER NOT NULL REFERENCES forms(form_id),
  attribute_name  VARCHAR2(255) NOT NULL,
```

```
  foreground_color VARCHAR2(50),
  background_color VARCHAR2(50),
  font_name      VARCHAR2(100),
  font_size      NUMBER
);
```

3.12 lovs (List of Values)

Captures LOV definitions in the form.

sql

Copy code

```
CREATE TABLE lovs (
  lov_id        NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id       NUMBER NOT NULL REFERENCES forms(form_id),
  lov_name      VARCHAR2(255) NOT NULL,
  lov_type      VARCHAR2(50),
  query_text    CLOB,
  created_date  DATE DEFAULT SYSDATE
);
```

3.13 record_groups

Represents record groups for LOVs or data sources.

sql

Copy code

```
CREATE TABLE record_groups (
  group_id      NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id       NUMBER NOT NULL REFERENCES forms(form_id),
  group_name    VARCHAR2(255) NOT NULL,
  query_text    CLOB,
```

```
  created_date    DATE DEFAULT SYSDATE
);
```

3.14 relations

Represents relationships between blocks.

sql

Copy code

```
CREATE TABLE relations (
  relation_id     NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  master_block_id NUMBER NOT NULL REFERENCES blocks(block_id),
  detail_block_id NUMBER NOT NULL REFERENCES blocks(block_id),
  join_condition  VARCHAR2(4000),
  created_date    DATE DEFAULT SYSDATE
);
```

3.15 parameters

Represents parameters passed to the form.

sql

Copy code

```
CREATE TABLE parameters (
  parameter_id    NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  form_id         NUMBER NOT NULL REFERENCES forms(form_id),
  parameter_name  VARCHAR2(255) NOT NULL,
  data_type       VARCHAR2(50),
  default_value   VARCHAR2(4000),
  created_date    DATE DEFAULT SYSDATE
);
```

3.16 coordinates

Captures layout coordinates of items, blocks, and canvases.

sql

Copy code

```
CREATE TABLE coordinates (
  coordinate_id   NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  parent_type     VARCHAR2(50) CHECK (parent_type IN ('ITEM', 'BLOCK', 'CANVAS')),
  parent_id       NUMBER NOT NULL,
  x_position      NUMBER,
  y_position      NUMBER,
  width           NUMBER,
  height          NUMBER
);
```

3.17 validation_rules

Represents validation rules for items or blocks.

sql

Copy code

```
CREATE TABLE validation_rules (
  rule_id       NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  parent_type   VARCHAR2(50) CHECK (parent_type IN ('ITEM', 'BLOCK')),
  parent_id     NUMBER NOT NULL,
  rule_expression VARCHAR2(4000),
  error_message   VARCHAR2(255)
);
```

4 Example of Oracle Form XML Extract

```
<?xml version="1.0" encoding="UTF-8"?>
<FormModule version="11.1.2.2.0">
```

```
 <FormProperties>
  <Property name="Name" value="EMPLOYEE_FORM" />
  <Property name="Title" value="Employee Management Form" />
  <Property name="Description" value="Manages employee records, including salary and
department assignments." />
 </FormProperties>

 <Blocks>
  <Block>
   <Property name="Name" value="EMPLOYEE" />
   <Property name="QueryDataSource" value="EMPLOYEES" />
   <Items>
    <Item>
     <Property name="Name" value="EMP_ID" />
     <Property name="DataType" value="NUMBER" />
     <Property name="Canvas" value="EMP_CANVAS" />
     <Property name="Prompt" value="Employee ID" />
     <Triggers>
      <Trigger>
       <Property name="Name" value="WHEN-VALIDATE-ITEM" />
       <Code>
        <![CDATA[
        IF :EMP_ID IS NULL THEN
         RAISE FORM_TRIGGER_FAILURE;
        END IF;
        ]]>
       </Code>
      </Trigger>
     </Triggers>
    </Item>
    <Item>
     <Property name="Name" value="EMP_NAME" />
     <Property name="DataType" value="VARCHAR2" />
     <Property name="Canvas" value="EMP_CANVAS" />
     <Property name="Prompt" value="Employee Name" />
    </Item>
    <Item>
     <Property name="Name" value="DEPT_ID" />
     <Property name="DataType" value="NUMBER" />
     <Property name="Canvas" value="EMP_CANVAS" />
     <Property name="Prompt" value="Department ID" />
    </Item>
   </Items>
  </Block>
  <Block>
   <Property name="Name" value="DEPARTMENT" />
   <Property name="QueryDataSource" value="DEPARTMENTS" />
   <Items>
```

```
    <Item>
     <Property name="Name" value="DEPT_ID" />
     <Property name="DataType" value="NUMBER" />
     <Property name="Canvas" value="DEPT_CANVAS" />
     <Property name="Prompt" value="Department ID" />
    </Item>
    <Item>
     <Property name="Name" value="DEPT_NAME" />
     <Property name="DataType" value="VARCHAR2" />
     <Property name="Canvas" value="DEPT_CANVAS" />
     <Property name="Prompt" value="Department Name" />
    </Item>
   </Items>
  </Block>
</Blocks>

<Canvases>
  <Canvas>
   <Property name="Name" value="EMP_CANVAS" />
   <Property name="Type" value="Content" />
  </Canvas>
  <Canvas>
   <Property name="Name" value="DEPT_CANVAS" />
   <Property name="Type" value="Content" />
  </Canvas>
</Canvases>

<Triggers>
  <Trigger>
   <Property name="Name" value="WHEN-NEW-FORM-INSTANCE" />
   <Code>
    <![CDATA[
    GO_BLOCK('EMPLOYEE');
    ]]>
   </Code>
  </Trigger>
</Triggers>

<ProgramUnits>
  <ProgramUnit>
   <Property name="Name" value="CALCULATE_SALARY" />
   <Property name="Type" value="PROCEDURE" />
   <Code>
    <![CDATA[
    PROCEDURE CALCULATE_SALARY IS
    BEGIN
     -- Sample logic
     :EMPLOYEE.TOTAL_SALARY := :EMPLOYEE.SALARY + :EMPLOYEE.BONUS;
```

```
    END;
    ]]>
   </Code>
  </ProgramUnit>
 </ProgramUnits>

 <VisualAttributes>
  <VisualAttribute>
   <Property name="Name" value="REQUIRED_FIELD" />
   <Property name="ForegroundColor" value="RED" />
   <Property name="FontName" value="Arial" />
   <Property name="FontSize" value="12" />
  </VisualAttribute>
 </VisualAttributes>

 <Relations>
  <Relation>
   <Property name="MasterBlock" value="DEPARTMENT" />
   <Property name="DetailBlock" value="EMPLOYEE" />
   <Property name="JoinCondition" value="DEPARTMENT.DEPT_ID = EMPLOYEE.DEPT_ID" />
  </Relation>
 </Relations>
</FormModule>
```

5 Oracle Form XML to Oracle Tables Parsers

5.1 PL/SQL to Parse the XML Extract

```
DECLARE
  l_xml CLOB := '<your_xml_content>'; -- Replace with the actual XML content or load from a table.
BEGIN
  -- Parse Forms
  INSERT INTO forms (form_name, description, version)
  SELECT form_name, description, version
  FROM XMLTABLE(
    '/FormModule/FormProperties'
    PASSING XMLTYPE(l_xml)
    COLUMNS
      form_name   VARCHAR2(255) PATH 'Property[@name="Name"]/@value',
      description  VARCHAR2(4000) PATH 'Property[@name="Description"]/@value',
      version     VARCHAR2(50) PATH 'Property[@name="Version"]/@value'
  );

  -- Parse Blocks
  INSERT INTO blocks (form_id, block_name, description, query_data_source)
  SELECT f.form_id, block_name, description, query_data_source
  FROM forms f
```

```
CROSS JOIN XMLTABLE(
  '/FormModule/Blocks/Block'
  PASSING XMLTYPE(l_xml)
  COLUMNS
    block_name      VARCHAR2(255) PATH 'Property[@name="Name"]/@value',
    description     VARCHAR2(4000) PATH 'Property[@name="Description"]/@value',
    query_data_source VARCHAR2(255) PATH 'Property[@name="QueryDataSource"]/@value'
);

-- Parse Items
INSERT INTO items (block_id, item_name, item_type, data_type, prompt)
SELECT b.block_id, item_name, item_type, data_type, prompt
FROM blocks b
CROSS JOIN XMLTABLE(
  '/FormModule/Blocks/Block/Items/Item'
  PASSING XMLTYPE(l_xml)
  COLUMNS
    item_name  VARCHAR2(255) PATH 'Property[@name="Name"]/@value',
    item_type  VARCHAR2(255) PATH 'Property[@name="ItemType"]/@value',
    data_type  VARCHAR2(255) PATH 'Property[@name="DataType"]/@value',
    prompt    VARCHAR2(255) PATH 'Property[@name="Prompt"]/@value'
);

-- Parse Triggers
INSERT INTO triggers (parent_type, parent_id, trigger_name, trigger_event, trigger_code)
SELECT 'BLOCK', b.block_id, trigger_name, trigger_event, trigger_code
FROM blocks b
CROSS JOIN XMLTABLE(
  '/FormModule/Blocks/Block/Triggers/Trigger'
  PASSING XMLTYPE(l_xml)
  COLUMNS
    trigger_name  VARCHAR2(255) PATH 'Property[@name="Name"]/@value',
    trigger_event VARCHAR2(255) PATH 'Property[@name="Event"]/@value',
    trigger_code  CLOB        PATH 'Code/text()'
);

-- Parse Properties
INSERT INTO properties (parent_type, parent_id, property_name, property_value)
SELECT 'FORM', f.form_id, property_name, property_value
FROM forms f
CROSS JOIN XMLTABLE(
  '/FormModule/FormProperties/Property'
  PASSING XMLTYPE(l_xml)
  COLUMNS
    property_name  VARCHAR2(255) PATH '@name',
    property_value VARCHAR2(4000) PATH '@value'
);
```

```
-- Parse Program Units
INSERT INTO program_units (form_id, unit_name, unit_type, unit_code)
SELECT f.form_id, unit_name, unit_type, unit_code
FROM forms f
CROSS JOIN XMLTABLE(
  '/FormModule/ProgramUnits/ProgramUnit'
  PASSING XMLTYPE(l_xml)
  COLUMNS
    unit_name  VARCHAR2(255) PATH 'Property[@name="Name"]/@value',
    unit_type  VARCHAR2(50)  PATH 'Property[@name="Type"]/@value',
    unit_code  CLOB        PATH 'Code/text()'
);

-- Parse Canvases
INSERT INTO canvas (form_id, canvas_name, canvas_type)
SELECT f.form_id, canvas_name, canvas_type
FROM forms f
CROSS JOIN XMLTABLE(
  '/FormModule/Canvases/Canvas'
  PASSING XMLTYPE(l_xml)
  COLUMNS
    canvas_name VARCHAR2(255) PATH 'Property[@name="Name"]/@value',
    canvas_type VARCHAR2(50)  PATH 'Property[@name="Type"]/@value'
);

-- Parse Windows
INSERT INTO windows (form_id, window_name)
SELECT f.form_id, window_name
FROM forms f
CROSS JOIN XMLTABLE(
  '/FormModule/Windows/Window'
  PASSING XMLTYPE(l_xml)
  COLUMNS
    window_name VARCHAR2(255) PATH 'Property[@name="Name"]/@value'
);

-- Parse Alerts
INSERT INTO alerts (form_id, alert_name, alert_message, alert_type)
SELECT f.form_id, alert_name, alert_message, alert_type
FROM forms f
CROSS JOIN XMLTABLE(
  '/FormModule/Alerts/Alert'
  PASSING XMLTYPE(l_xml)
  COLUMNS
    alert_name    VARCHAR2(255) PATH 'Property[@name="Name"]/@value',
    alert_message VARCHAR2(4000) PATH 'Property[@name="Message"]/@value',
    alert_type    VARCHAR2(50) PATH 'Property[@name="Type"]/@value'
);
```

```
  -- Continue for the other tables (menu_modules, visual_attributes, lovs, record_groups, relations, parameters, coordinates, validation_rules)
  -- Add XMLTABLE queries with appropriate XPath for each table.

  COMMIT;
END;
```

6 Levels of Effort for Converting Oracle Forms to APEX Pages

The **level of effort (LOE)** to convert an Oracle Form to an Oracle APEX page varies based on the complexity of the form, the experience of the developers, and the specific features being migrated. However, we can estimate an **average LOE** based on typical conversion scenarios.

6.1 Factors Influencing LOE

6.1.1 Form Complexity:

- Simple forms (basic CRUD operations): Lower LOE.
- Complex forms (triggers, custom validations, master-detail blocks): Higher LOE.

6.1.2 Number of Elements:

- Items (fields, buttons, LOVs, etc.): More elements require more time for mapping and implementation.

6.1.3 Business Logic:

- Forms with extensive PL/SQL logic, triggers, and validations require careful reimplementation in APEX.

6.1.4 Dependencies:

- Integration with other forms, custom libraries, or shared codebases adds to the effort.

6.1.5 Styling Requirements:

- If the APEX application needs to replicate the exact UI of Oracle Forms, additional effort is required for themes and templates.

6.2 Estimated LOE by Complexity

Complexity	Example	Estimated LOE
Simple Form	- Single block with a few fields. - Basic CRUD. - No custom triggers or validations.	**4-8 hours**
Moderate Form	- Master-detail relationship. - Several fields. - Validations and some triggers.	**2-4 days**
Complex Form	- Multi-block form with master-detail and query logic. - Custom triggers, LOVs, and validations. - Integration with external systems or PL/SQL libraries.	**1-2 weeks**

6.3 Breakdown of Conversion Tasks

6.3.1 Analysis and Planning

- Understand the form structure and functionality.
- Document the form's components, triggers, and dependencies.
- **LOE**: 1-2 hours per form.

6.3.2 Recreate UI

- Define APEX pages, regions, and items.
- Map Oracle Forms blocks to APEX regions.
- **LOE**:
    - Simple UI: 2-4 hours.
    - Complex UI: 1-2 days.

6.3.3 Business Logic

- Convert triggers to Dynamic Actions, Validations, or Processes.
- Migrate PL/SQL libraries to database packages.
- **LOE**:
    - Basic logic: 2-4 hours.
    - Complex logic: 2-3 days.

6.3.4 Integrations

- Replace Oracle Forms-specific features (e.g., record groups) with SQL or APEX components.
- Handle dependencies on other forms or reports.
- **LOE**: 1-3 days depending on complexity.

6.3.5 Testing and Refinement

- Validate the functionality of the APEX page against the original form.
- Ensure styling and performance meet requirements.
- **LOE**: 1-2 days.

6.4 Typical Conversion Timeline

Number of Forms	Simple Forms Only	Mix of Simple and Complex Forms
10 Forms	~1-2 weeks	~2-4 weeks
50 Forms	~4-6 weeks	~8-12 weeks
100+ Forms	~8-12 weeks	~16+ weeks

6.4.1 Efficiency Considerations

1. **Reusability**:
    - Use Shared Components (e.g., LOVs, templates) to reduce effort for repeated elements.
2. **Experience Level**:
    - Developers familiar with both Oracle Forms and APEX will complete conversions faster.

6.4.2 Summary

- **Simple Forms**: ~4-8 hours per form.
- **Moderate Forms**: ~2-4 days per form.
- **Complex Forms**: ~1-2 weeks per form.

For a project with a mix of form complexities, you can average **2-3 days per form**.