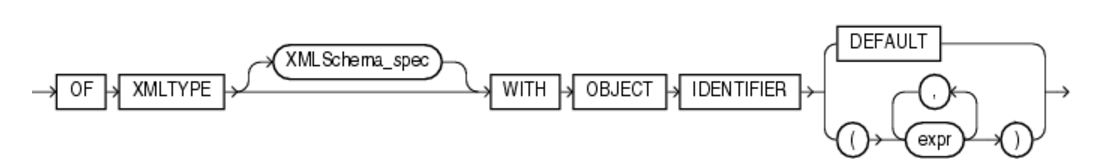
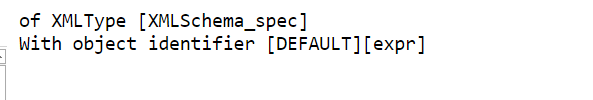
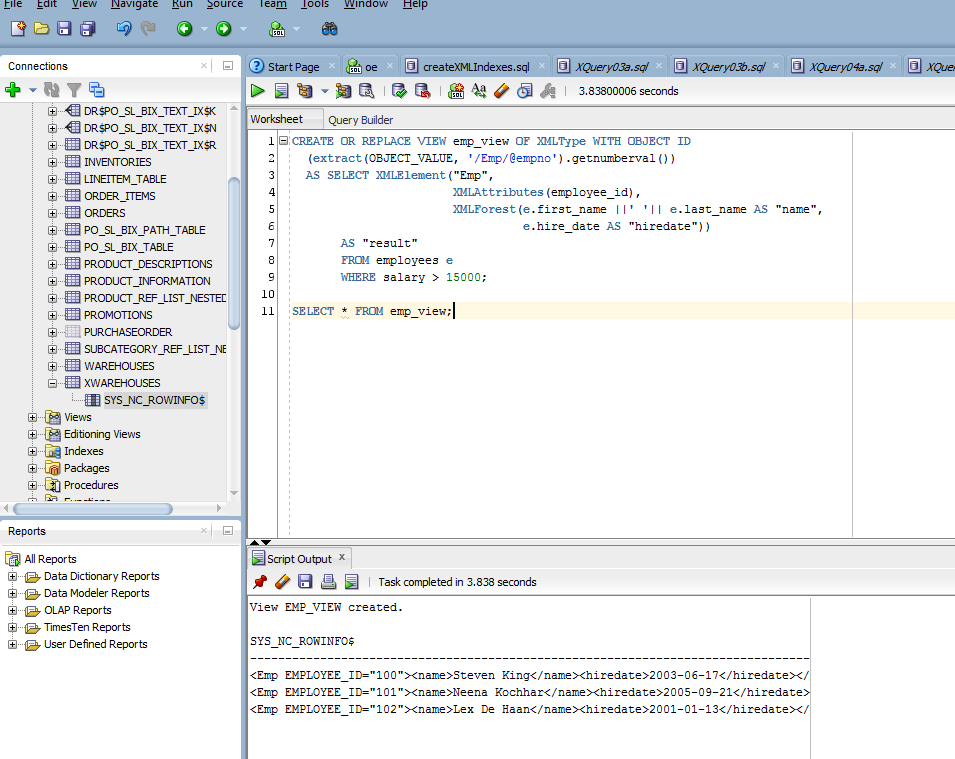
CREATING XMLType VIEWS



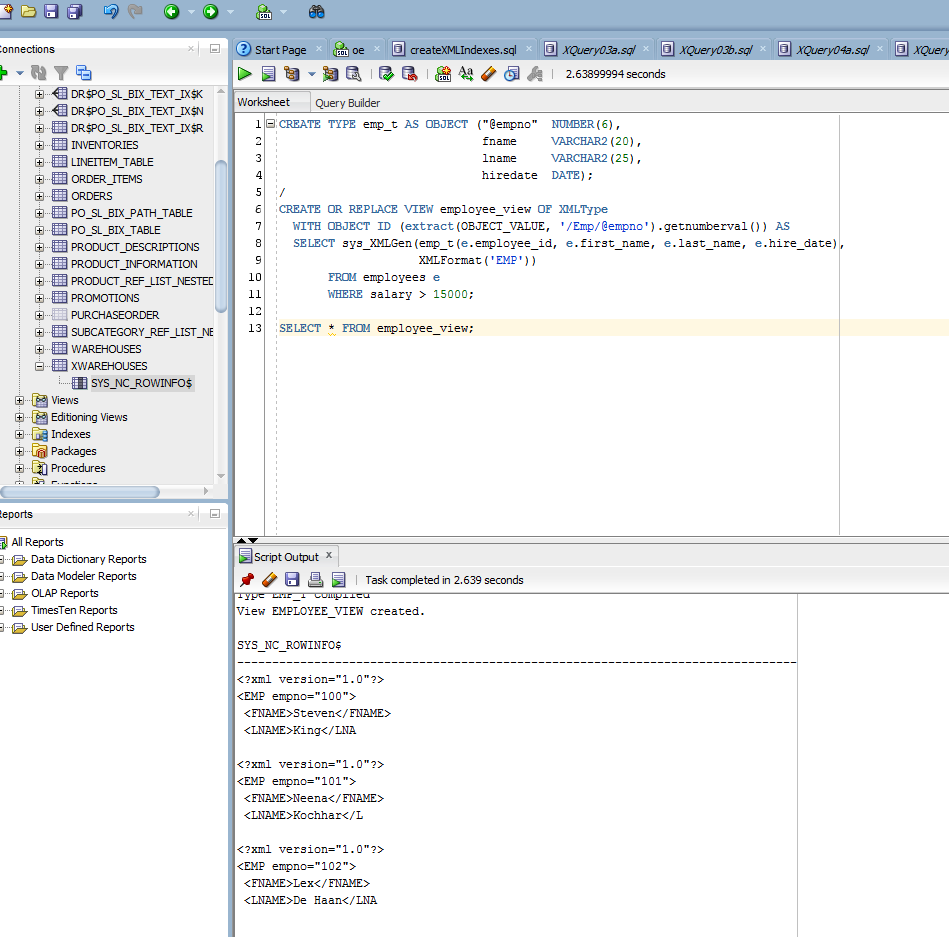


Using SQL/XML Generation Functions to Create Non-Schema-Based XMLType Views

The **empno** attribute in the document will be used as the unique identifier for each row. As the result of the XPath rewrite operation, the XPath**/Emp/@empno** can refer directly to the **empno** column.



### Using Object Types with SYS\_XMLGEN to Create Non-Schema-Based XMLType Views

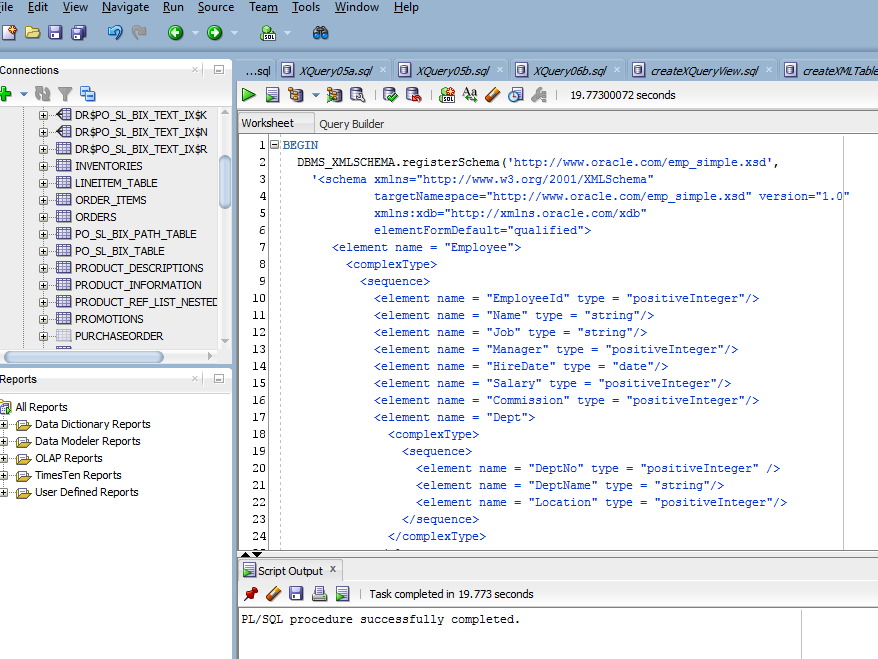


Why doesn’t the date show up in the XML result?

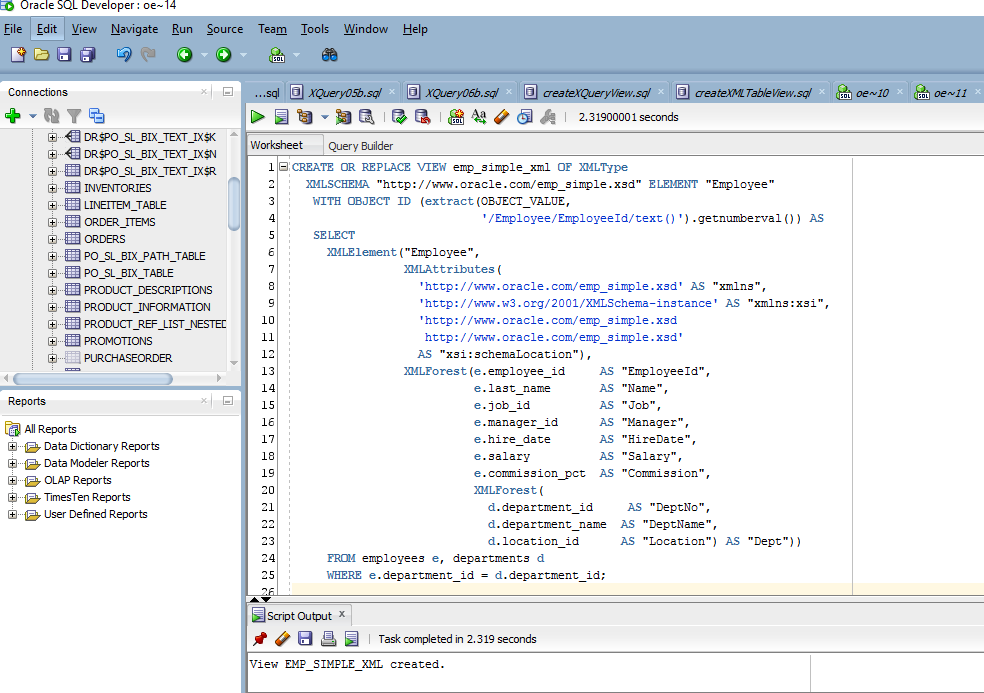
There’s a difference between running SCRIPT and running STATEMENT. What is the difference?

## **Creating XML Schema-Based XMLType Views**

1. Create and register the XML schema document that contains the necessary XML structures. Note that since the **XMLType** view is constructed using SQL/XML generation functions, you do not need to annotate the XML schema to present the bidirectional mapping from XML to SQL object types.
2. Create an **XMLType** view conforming to the XML schema by using SQL/XML functions.

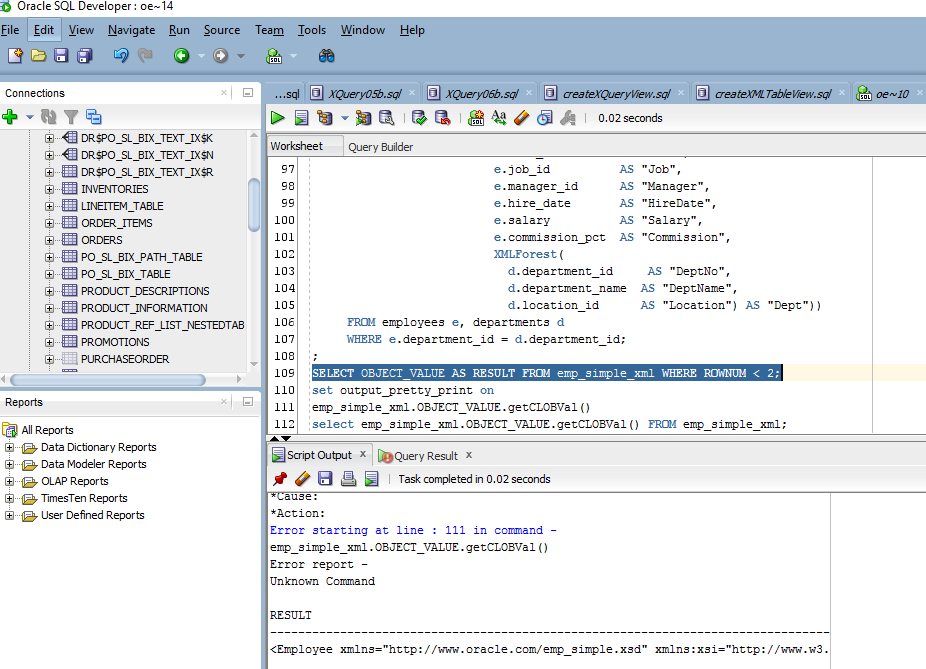


Target location: http://www.oracle.com/emp\_simple.xsd

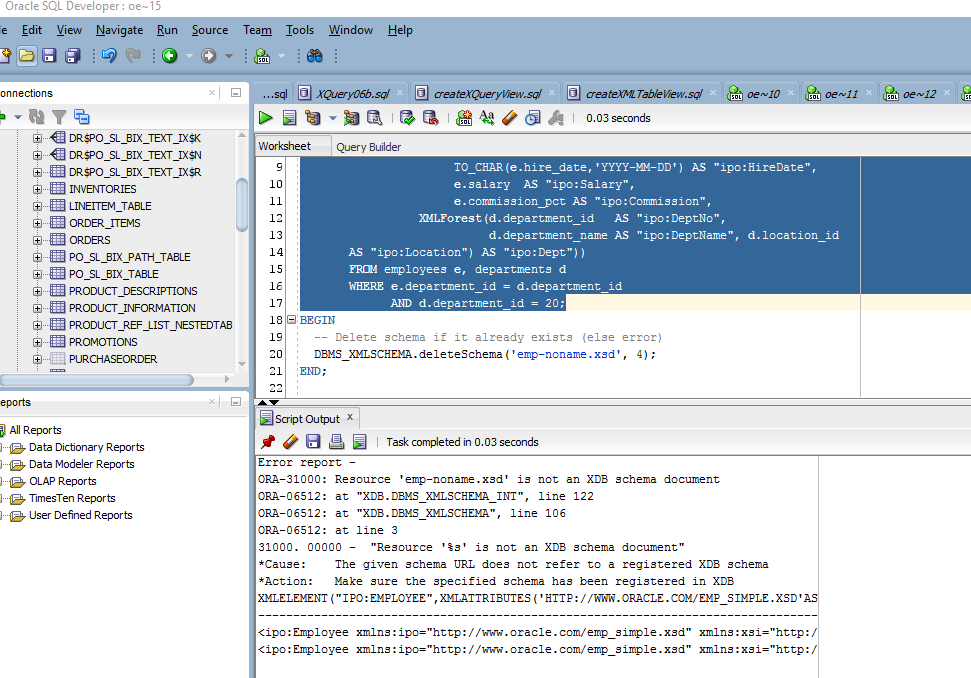


Example 18-4 Creating an XMLType View Using SQL/XML Functions

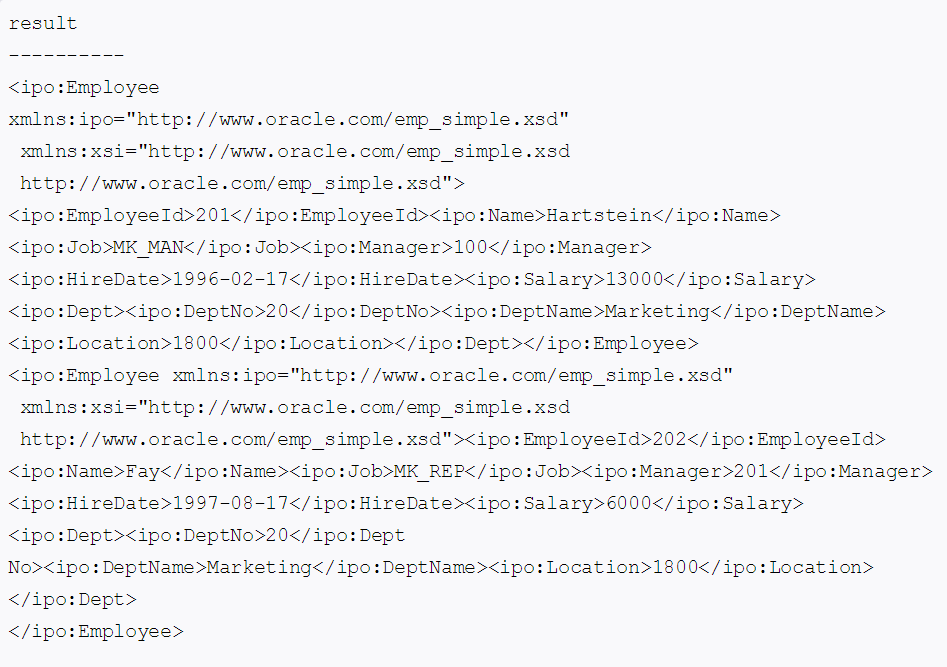
***Example 18-5 Querying an XMLType View***



#### **Using Namespaces With SQL/XML Functions**

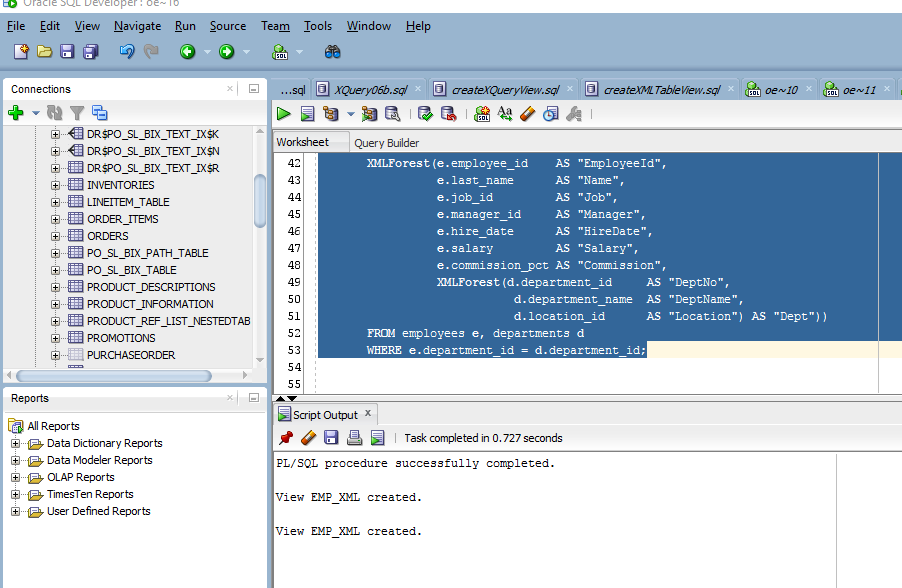


Did not produce the expected results – below is the expected results:



What is <ipo:>?

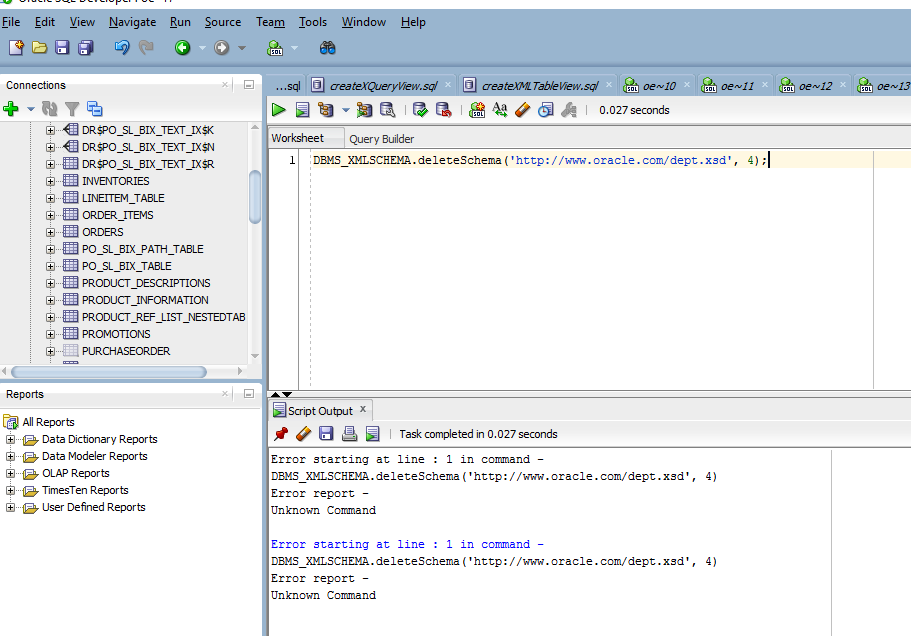
The following statement creates a view that conforms to this XML schema:



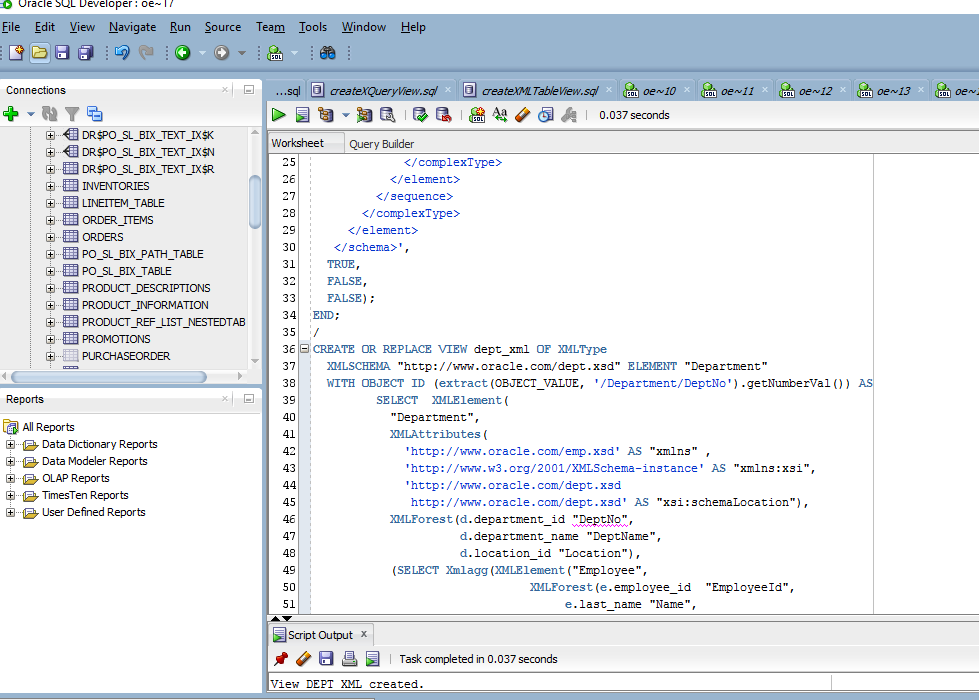
The XMLAttributes clause creates an XML element that contains the noNamespace schema location attribute.

***Example 18-7 Using SQL/XML Generation Functions in Schema-Based XMLType Views***

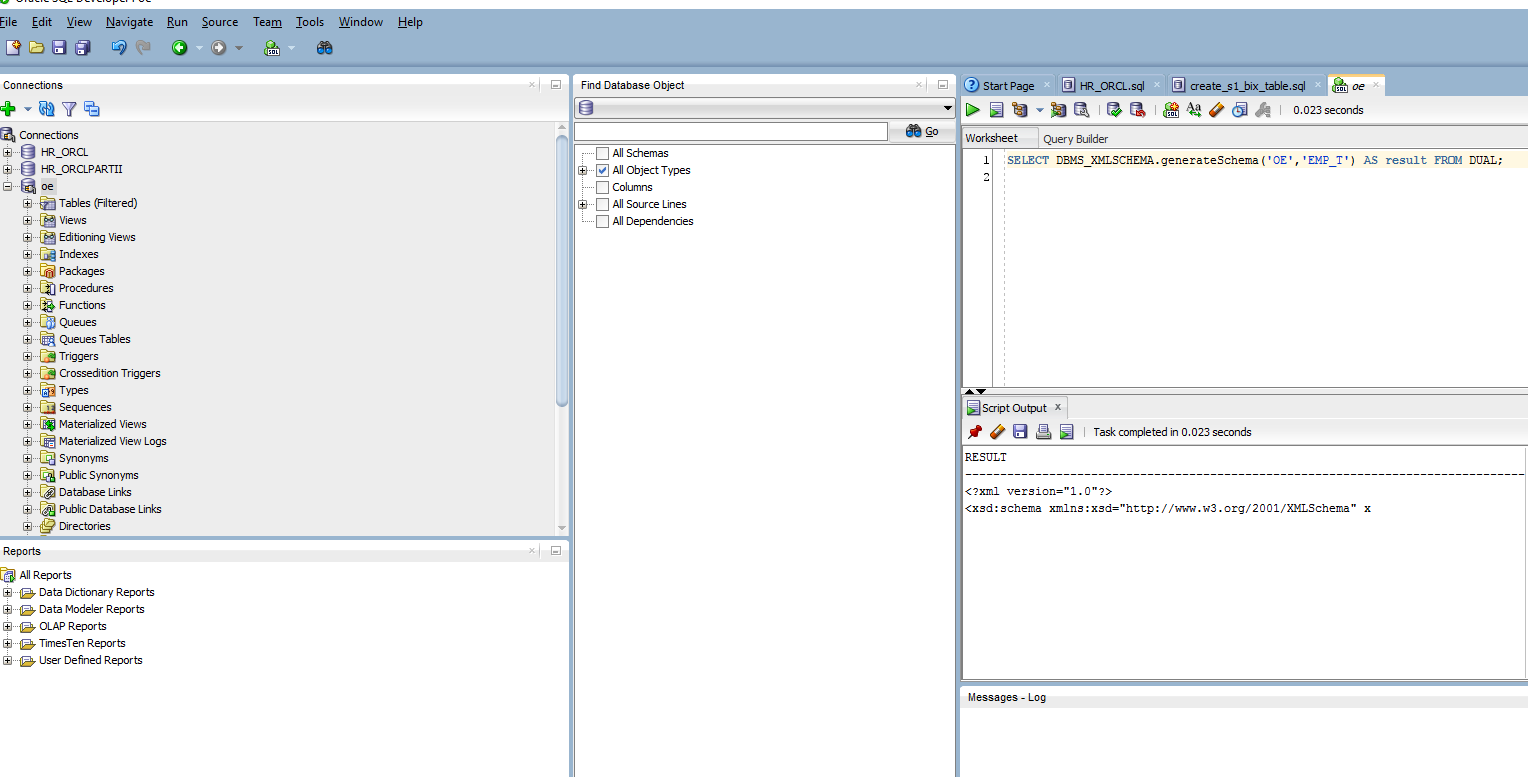
Error; unknown command



***Example 18-7 Using SQL/XML Generation Functions in Schema-Based XMLType Views***



#### **Creating Schema-Based XMLType Views Over Object Views**



Again, the output is not conveniently displayed and is hard for human reading.

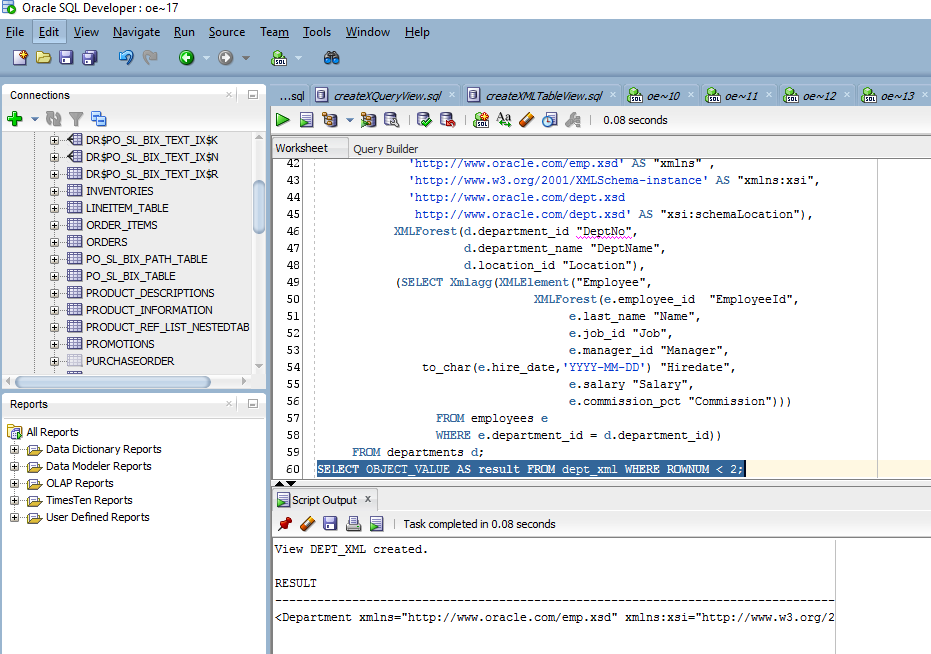
### Using Object Types and Views to Create XML Schema-Based XMLType Views

#### **Creating Schema-Based XMLType Views Over Object Views**

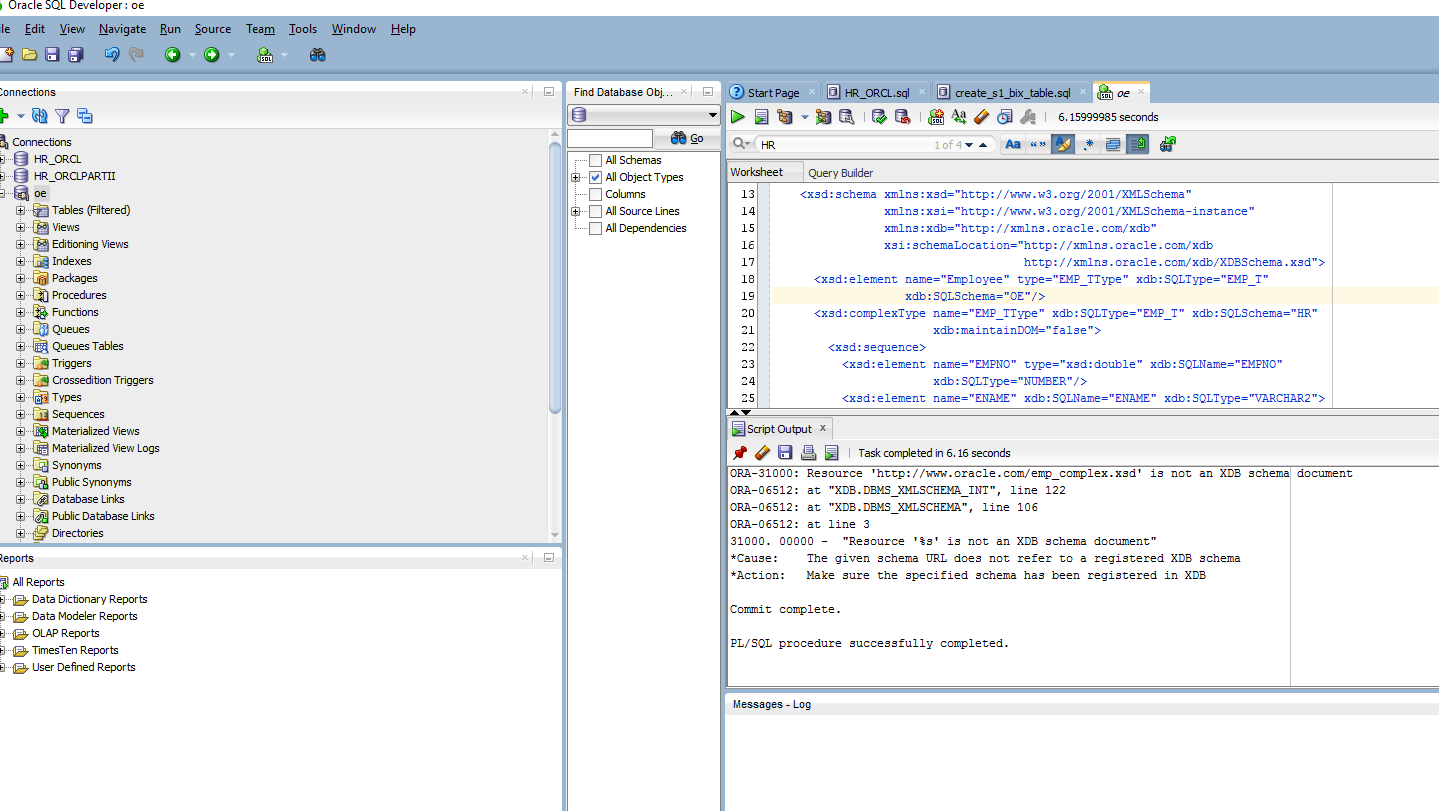
##### **Step 1. Create Object Types**

Step 2. Create or Generate XMLSchema, emp.xsd

Had to make a small change to the given statement. Nowhere in the instructions did Oracle mention switching databases. I’ve been using the ‘oe’ database for the entire assignment and this step calls for the ‘hr’ database.

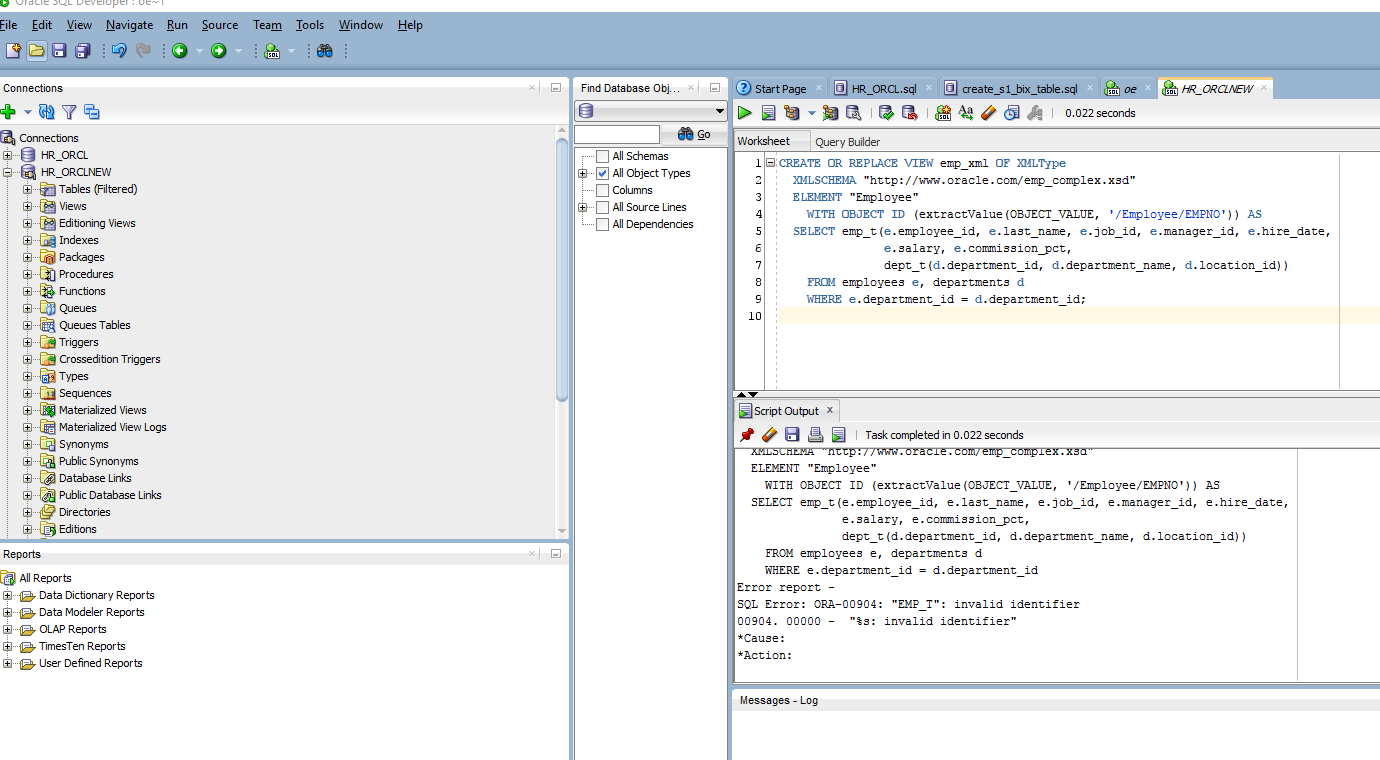


Again, I had to change the database from ‘hr’ to ‘oe’.

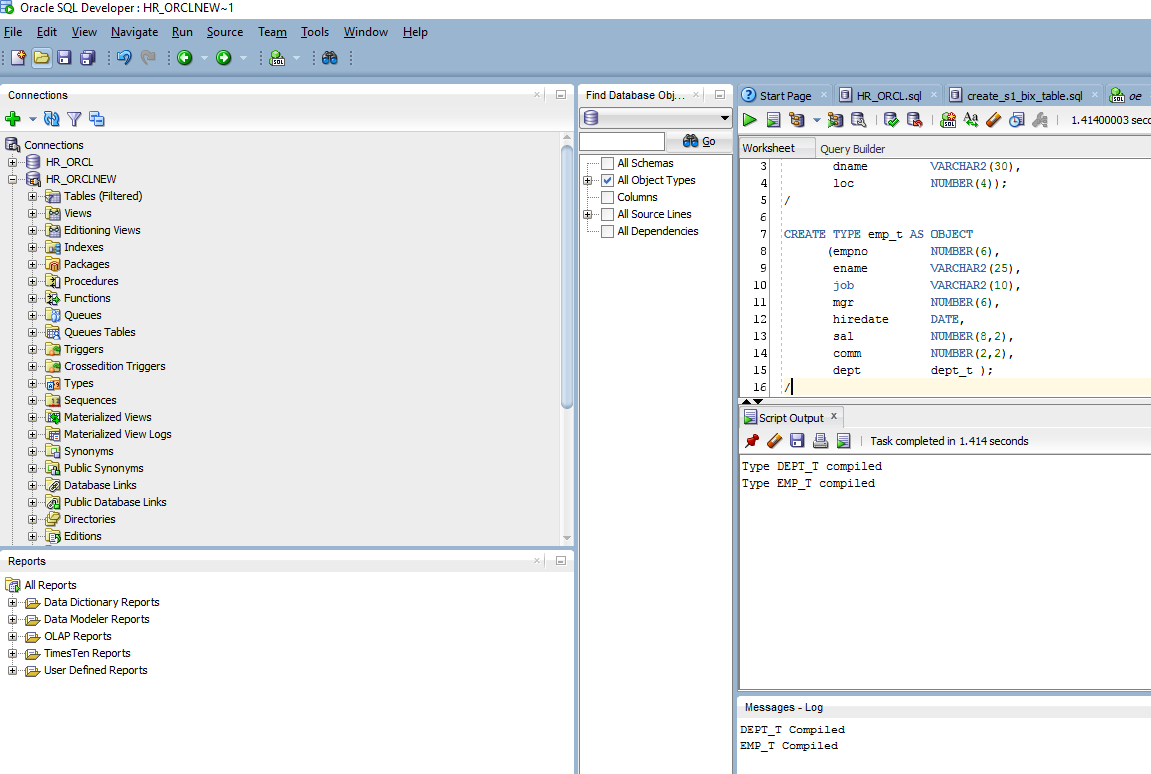


##### **Step 4a. Using the One-Step Process**

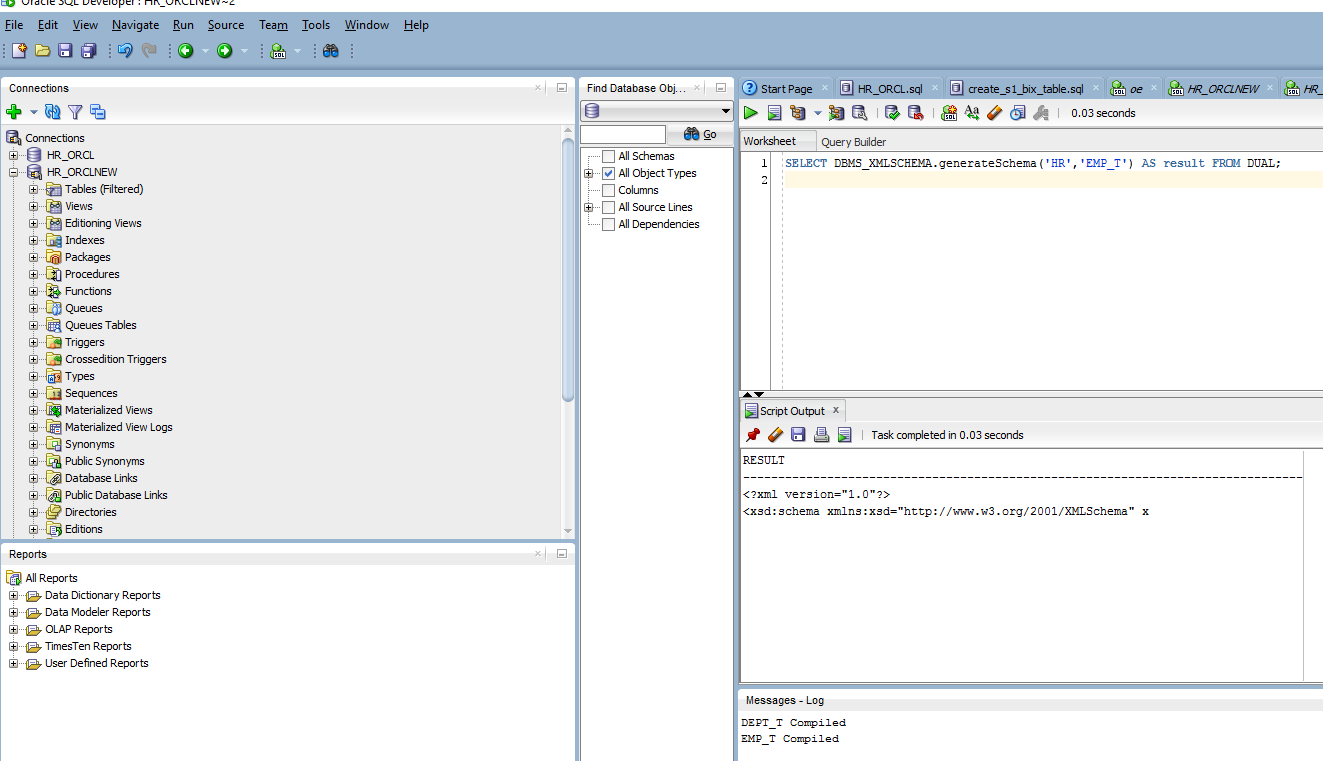
Have to switch databases to ‘hr’ 11:25am (have to redo steps to get emp\_t into hr)



Now, Oracle is working as expected



Script output is appropriate for step 18-9

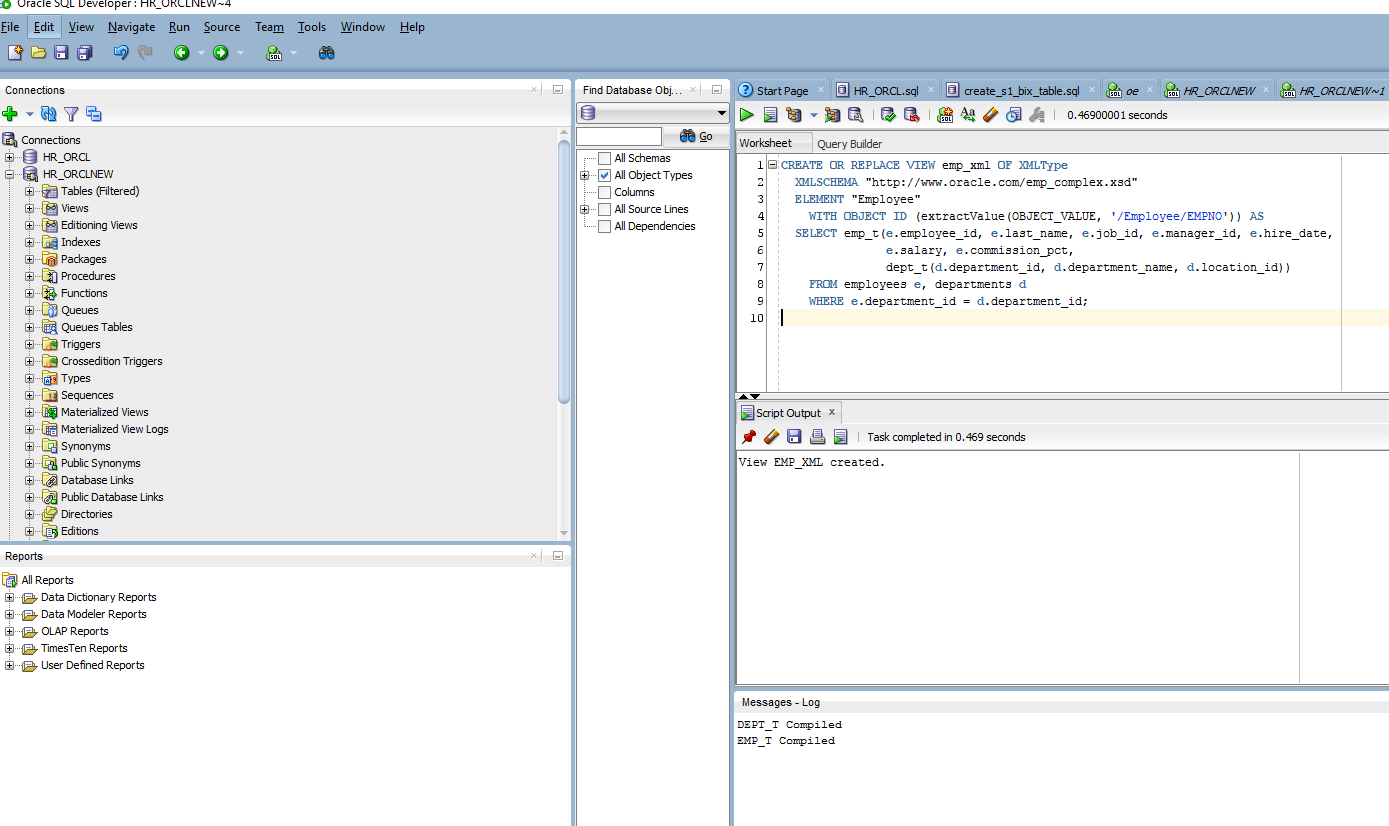


Step 18-10 throws expected error and registers schema at, <http://www.oracle.com/emp_complex.xsd>



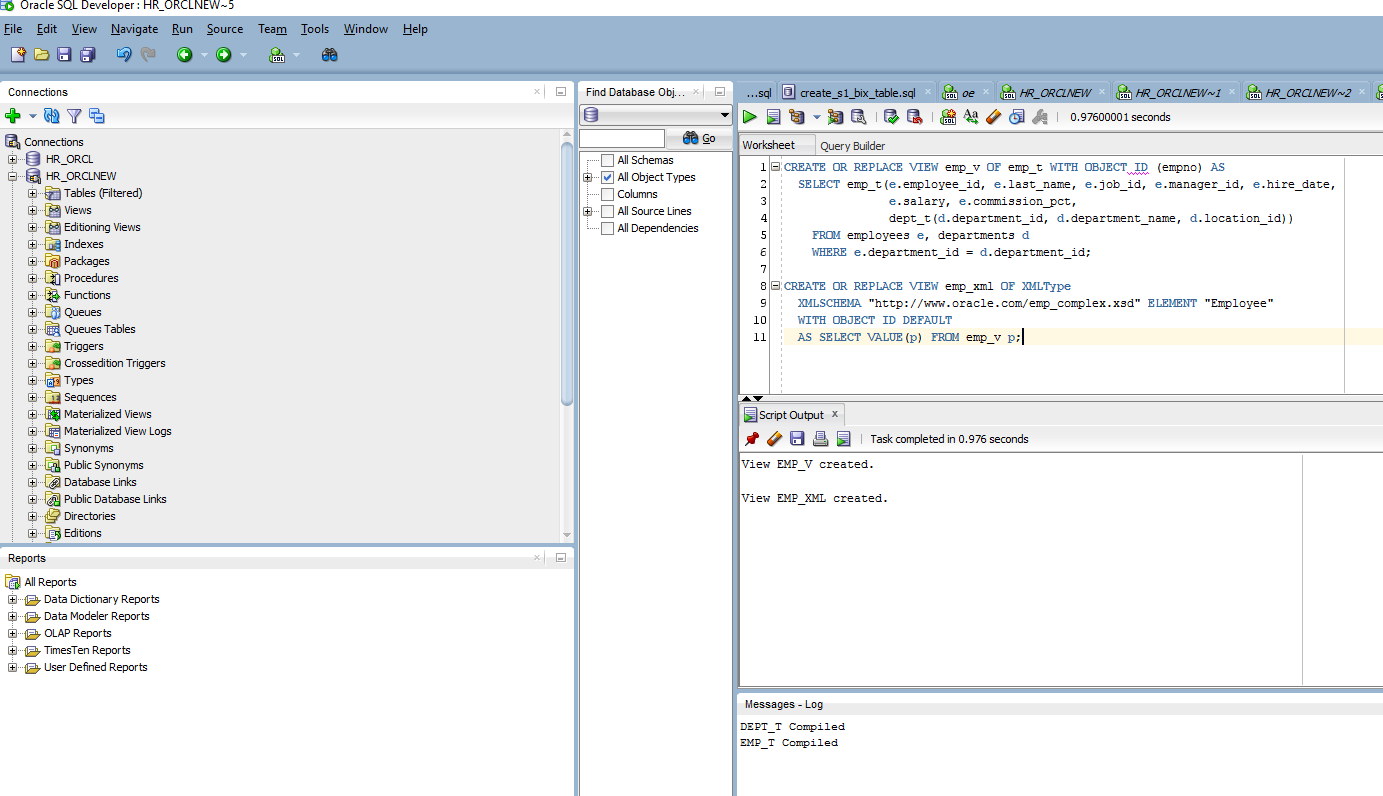
##### **Step 4a. Using the One-Step Process**

***Example 18-11 Creating an XMLType View***



##### **Step 4b. Using the Two-Step Process by First Creating an Object View**

***Example 18-12 Creating an Object View and an XMLType View on the Object View***



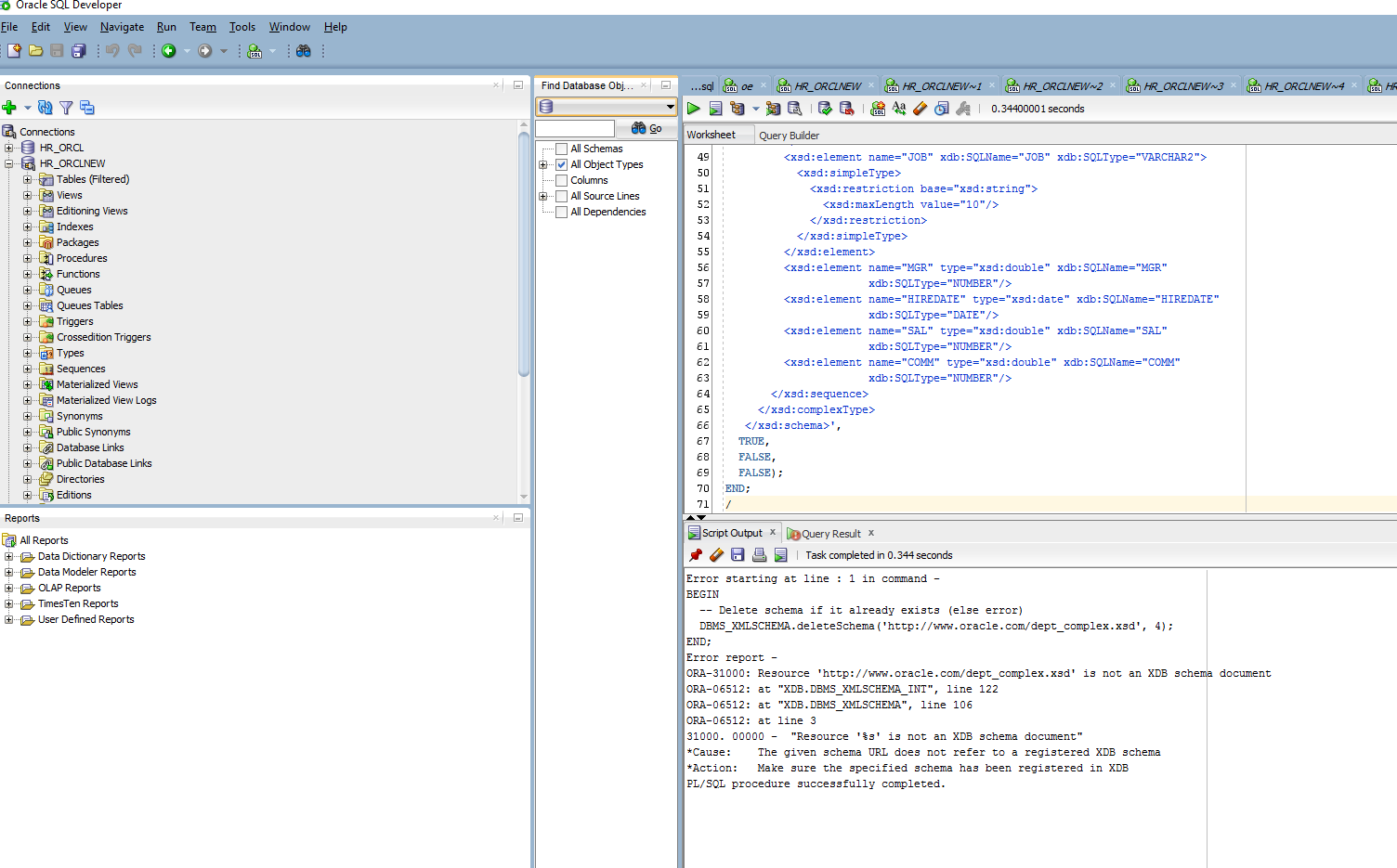
#### **Wrapping Relational Department Data with Nested Employee Data as XML**

##### **Step 1. Create Object Types**

***Example 18-13 Creating Object Types (this was already completed in earlier steps)***

##### **Step 2. Register XML Schema, dept\_complex.xsd**

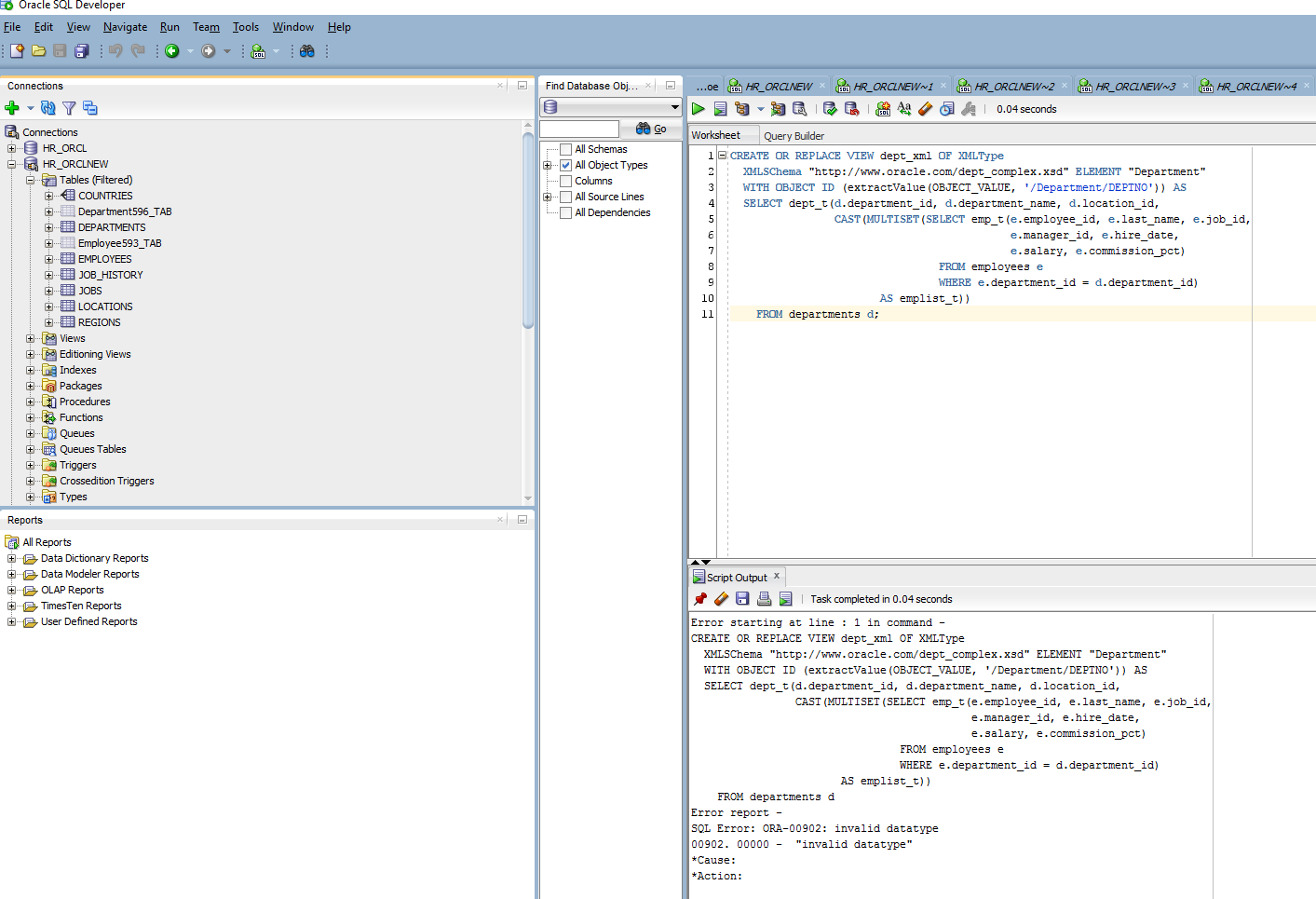
Exception is thrown, as expected and query executes successfully



##### **Step 3a. Create XMLType Views on Relational Tables**

***Example 18-15 Creating XMLType Views on Relational Tables***

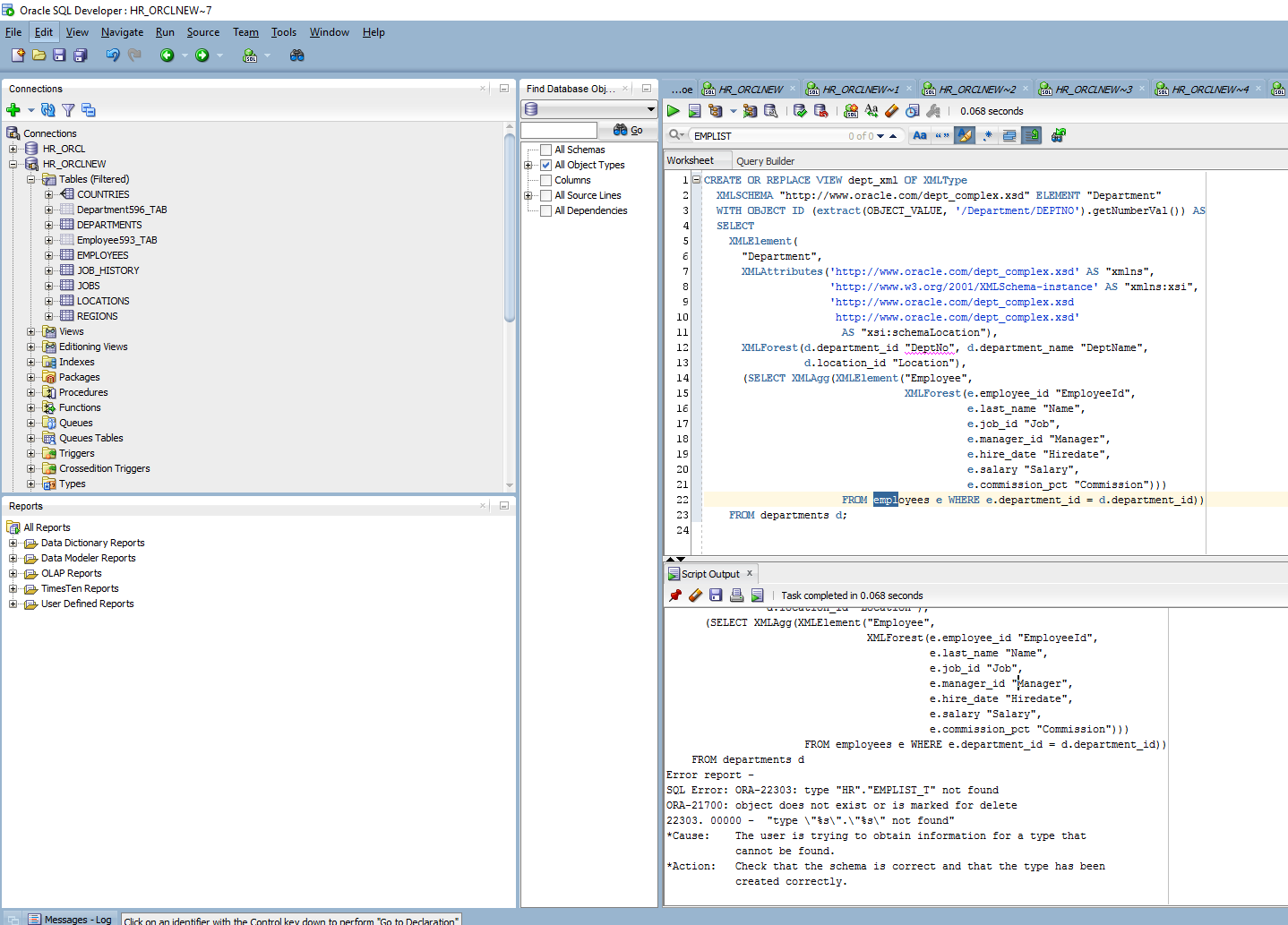
***Did not execute properly or as expected***



##### **Step 3b. Create XMLType Views Using SQL/XML Functions**

***Example 18-16 Creating XMLType Views Using SQL/XML Functions***

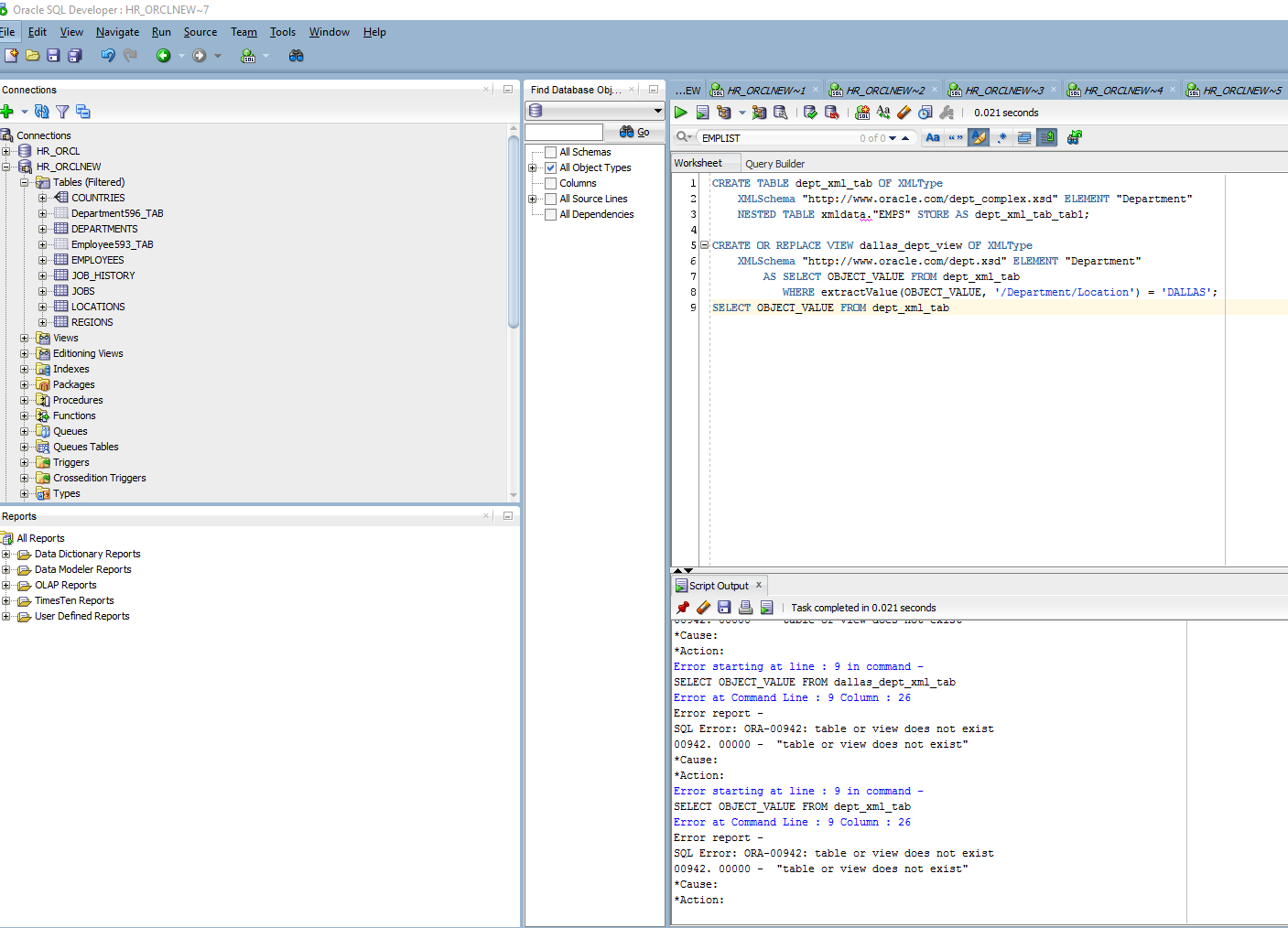
***This is also broken or does not run as expected***



## **Creating XMLType Views From XMLType Tables**

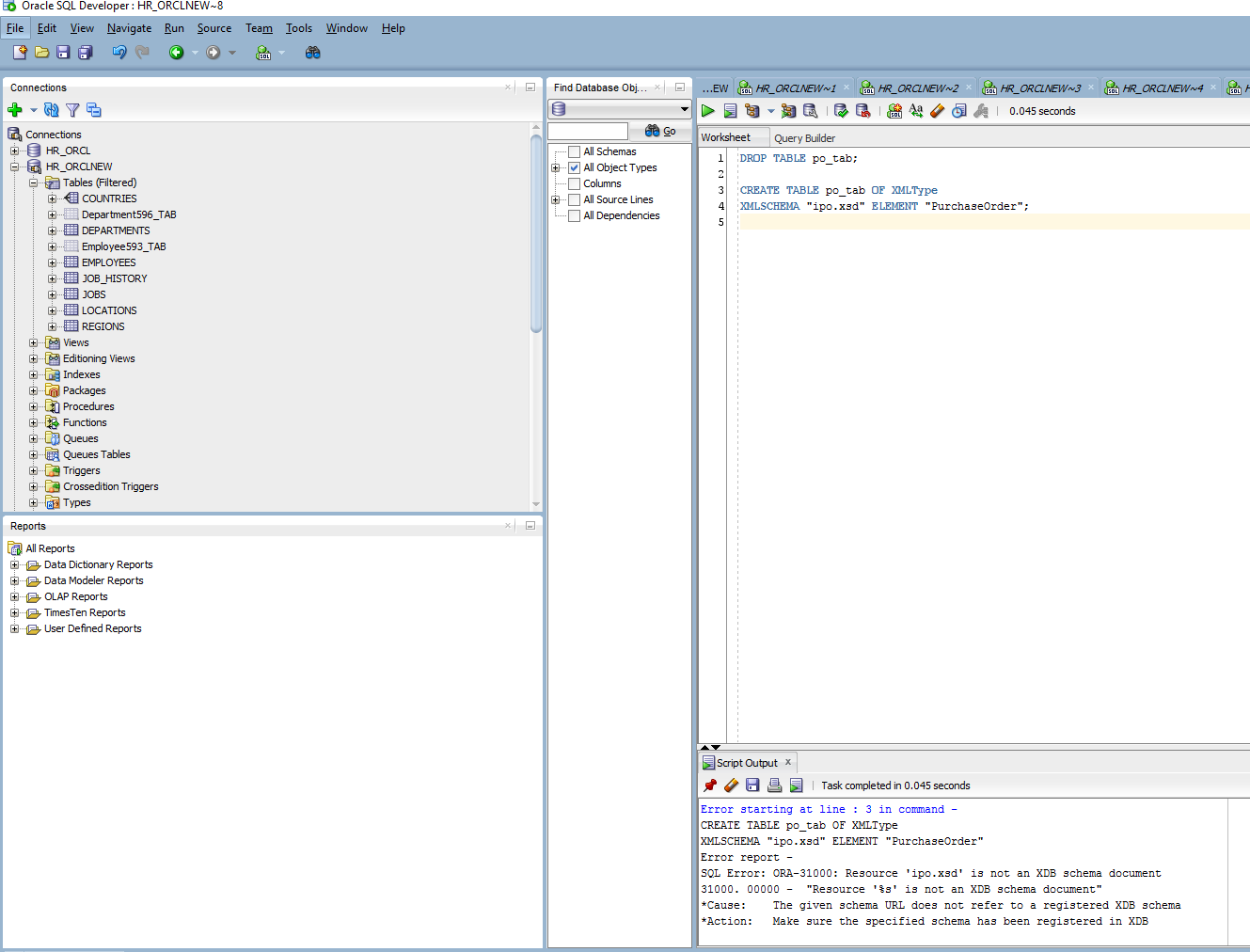
***Example 18-17 Creating an XMLType View by Restricting Rows From an XMLType Table***

***This is also broken or does not run as expected***

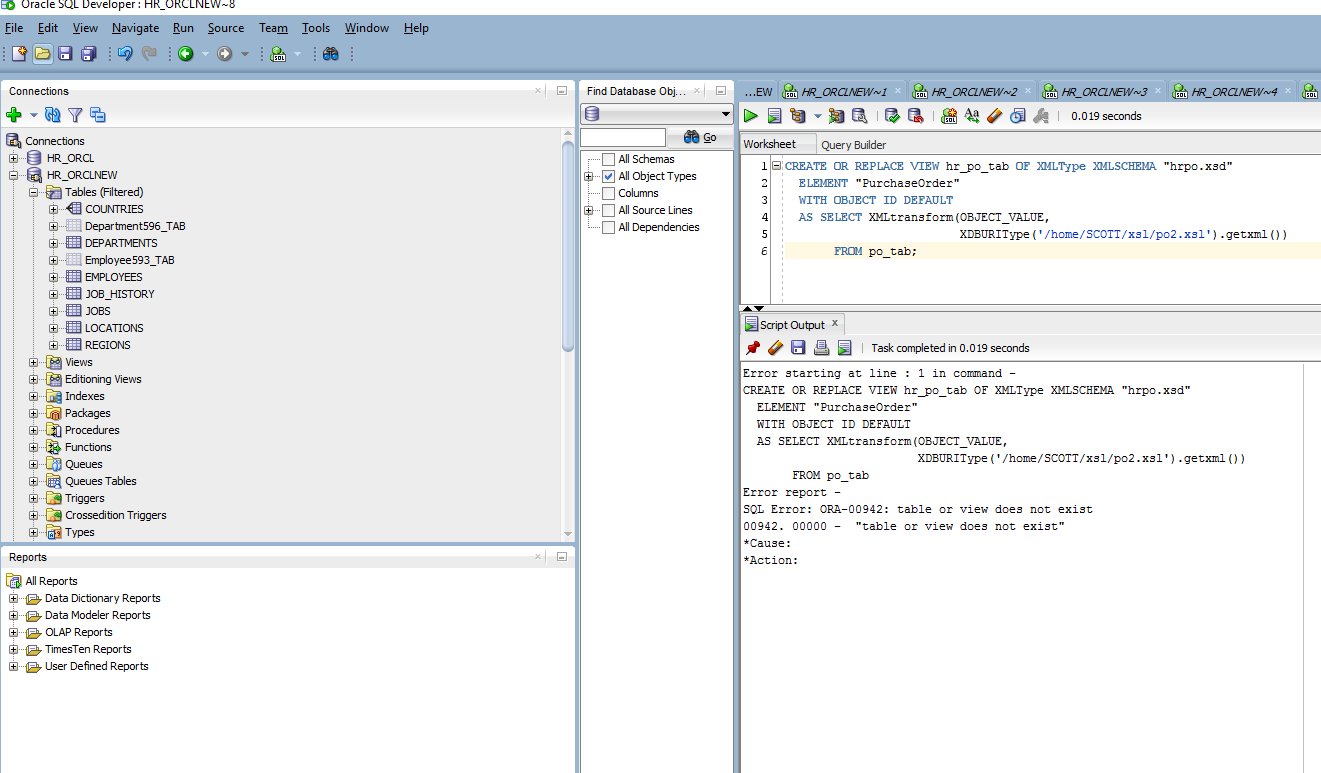


***Example 18-18 Creating an XMLType View by Transforming an XMLType Table***

***This is also broken or does not run as expected***

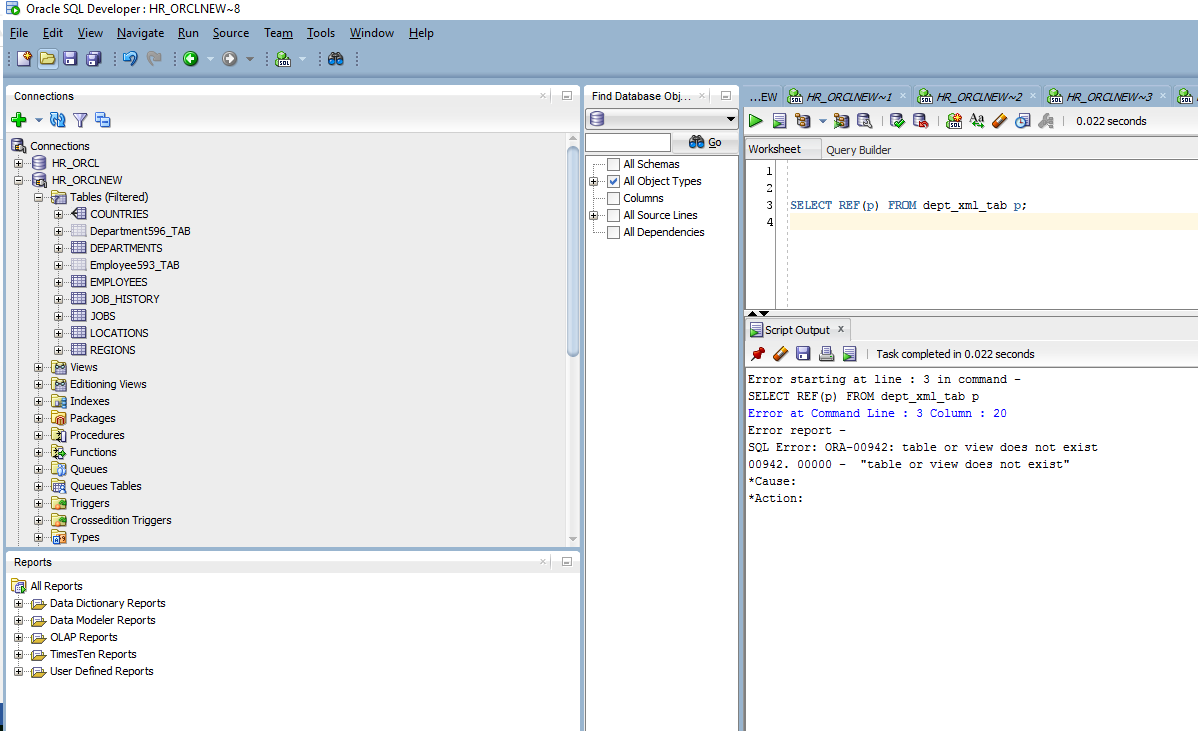


***This is also broken or does not run as expected***



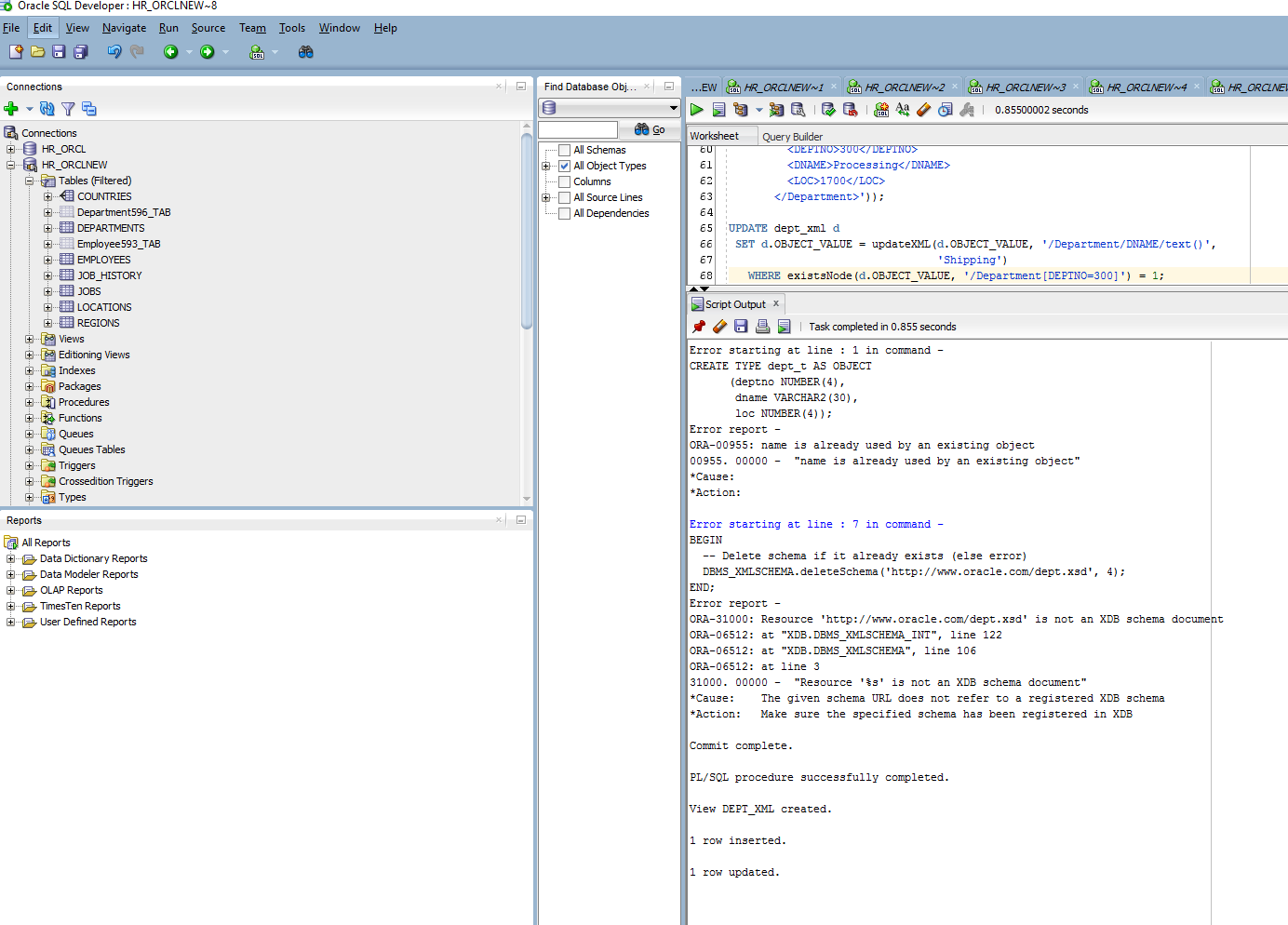
## **Referencing XMLType View Objects Using REF()**

***This is also broken or does not run as expected*** – I’m really disappointed because I was looking forward to learning this material.



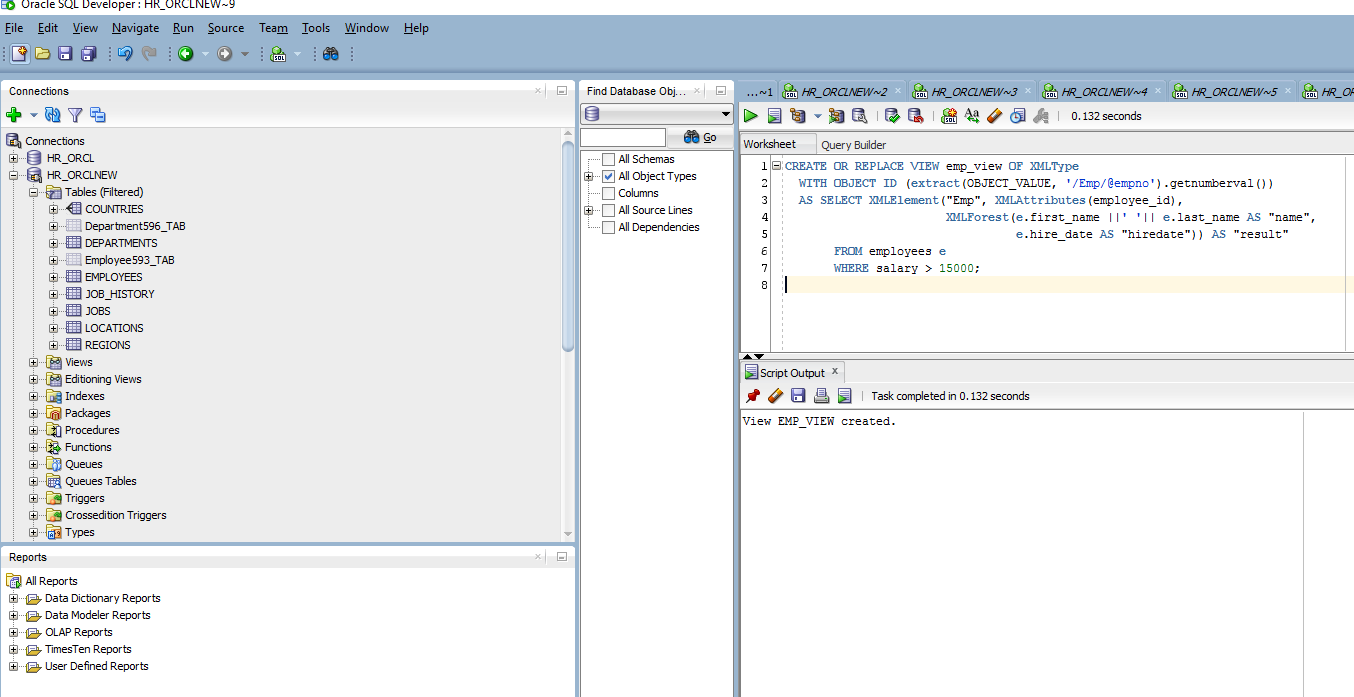
## **DML (Data Manipulation Language) on XMLType Views**

***Example 18-19 Identifying When a View is Implicitly Updatable – very exciting to see tutorial instructions working again***

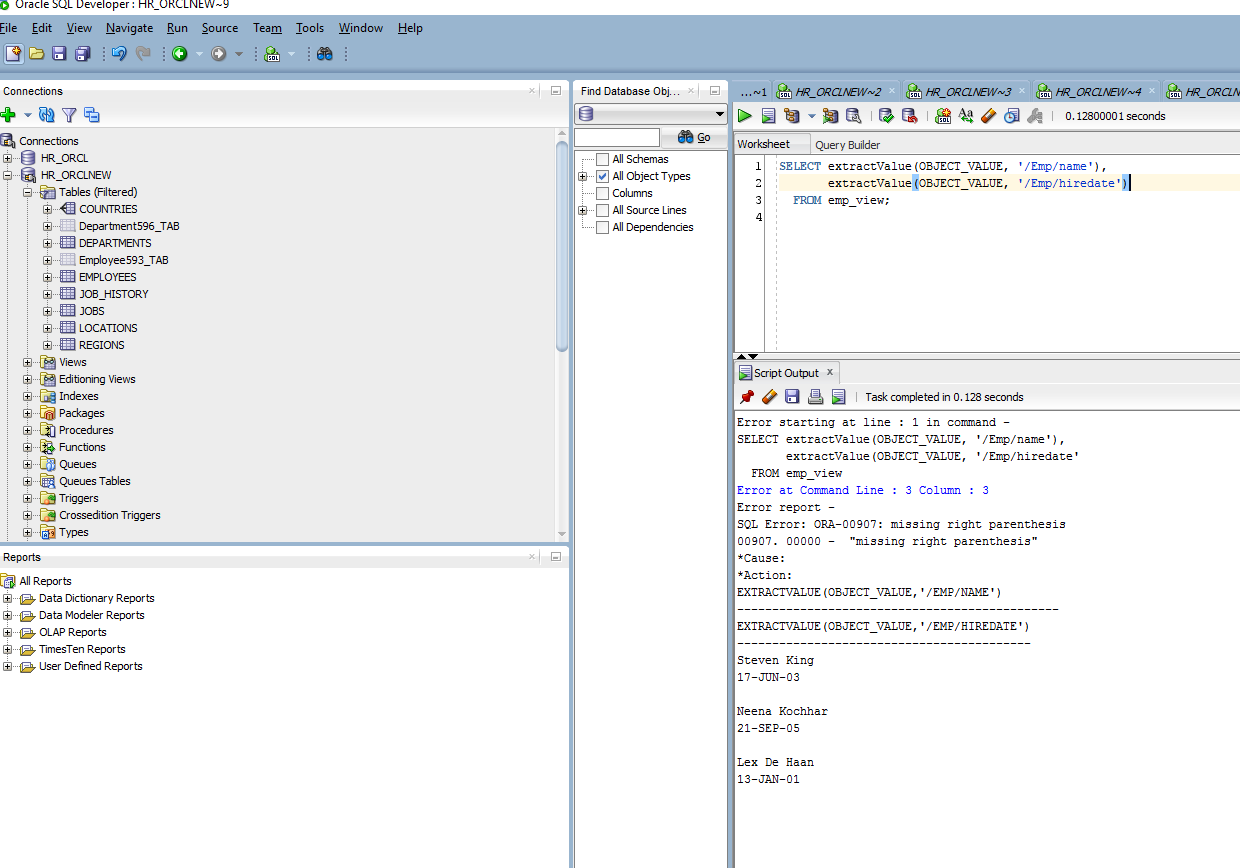


## **XPath Rewrite on XMLType Views**

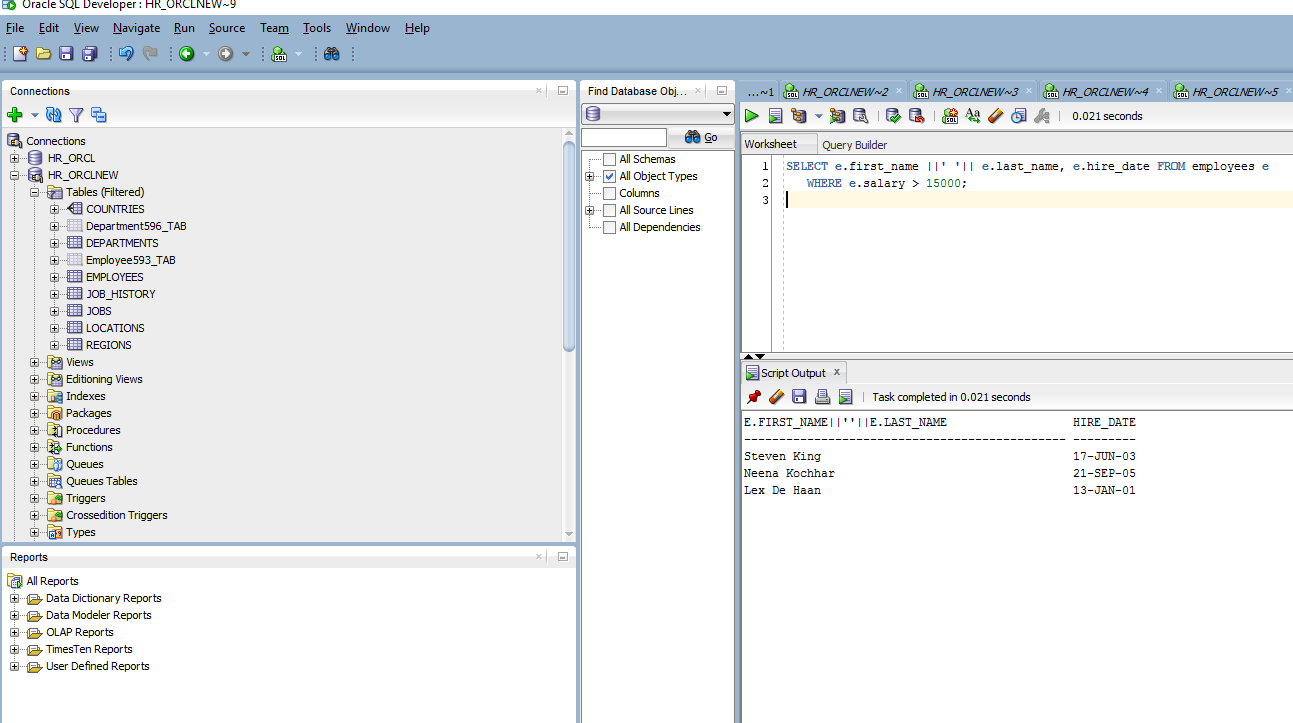
***Example 18-20 Non-Schema-Based Views Constructed Using SQL/XML***



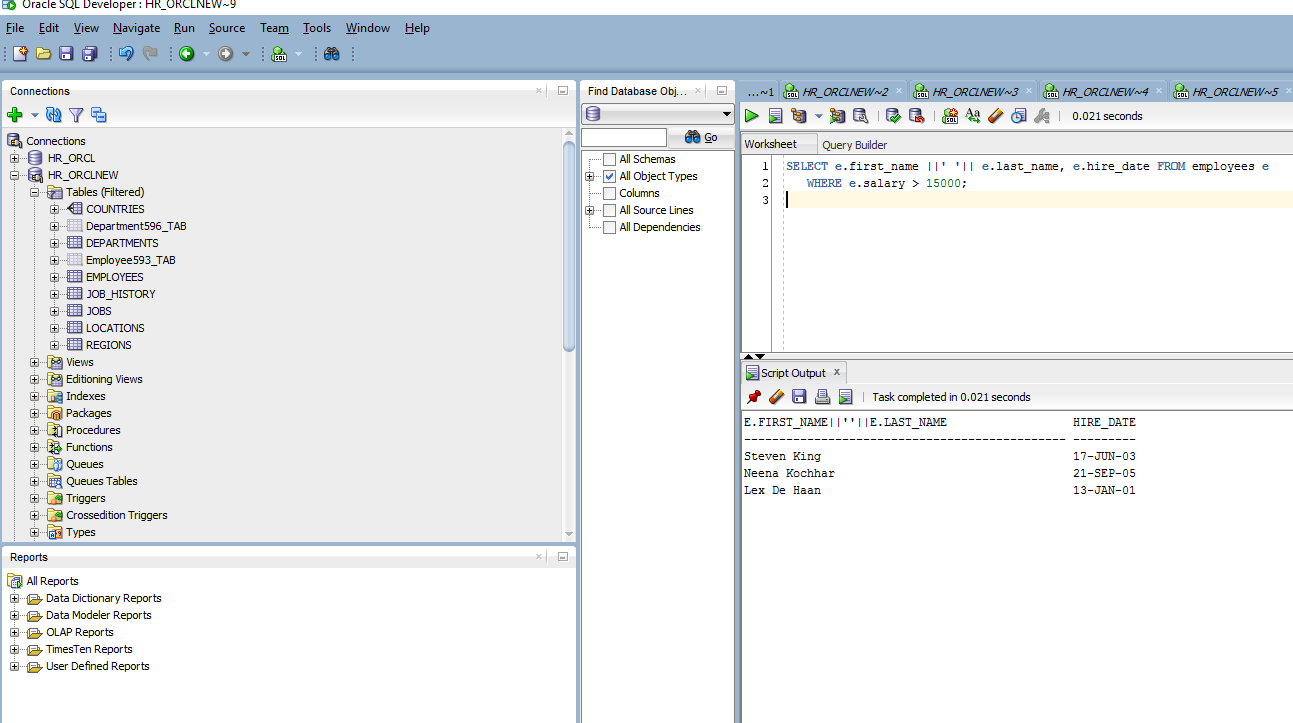
Querying with SWL function extractValue to select rom emp\_view: Tutorial has missing syntax of the last ‘)’.



Query becomes something like the following:

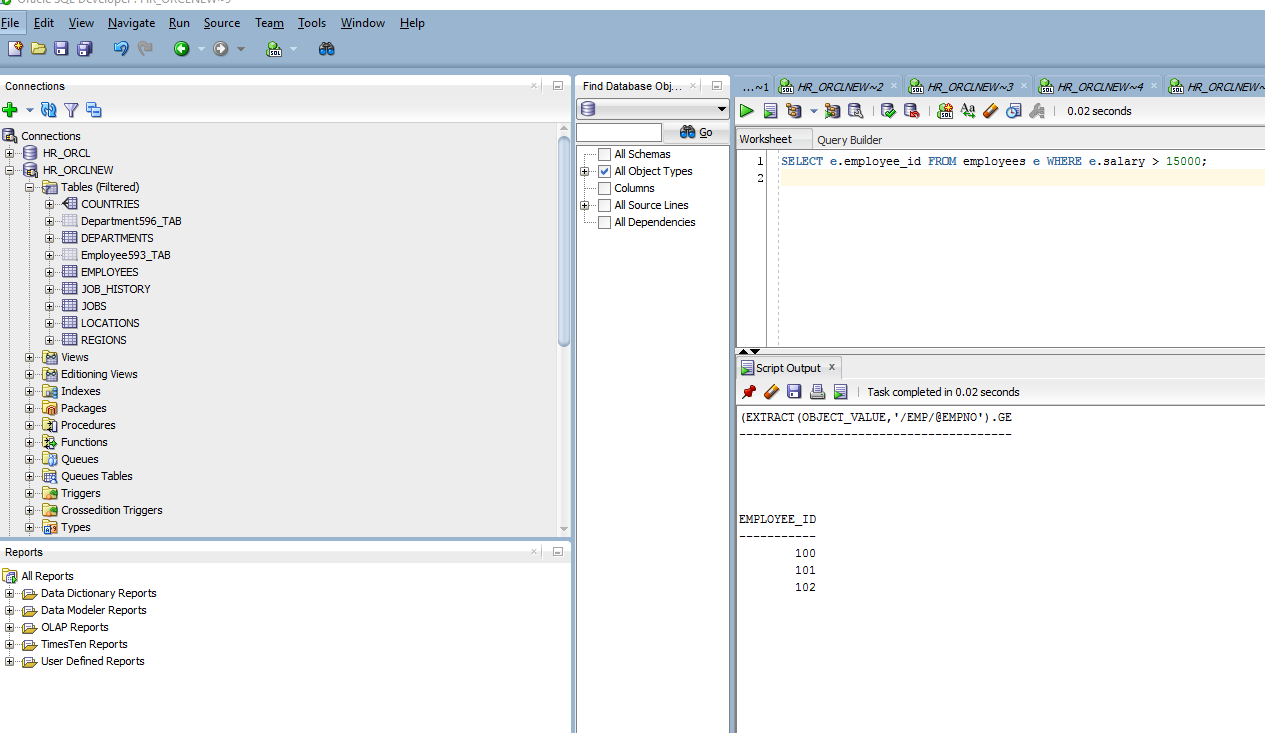


Querying with SQL function **extractValue** followed by method **getNumberVal()** to select from **emp\_view**: unreadable output



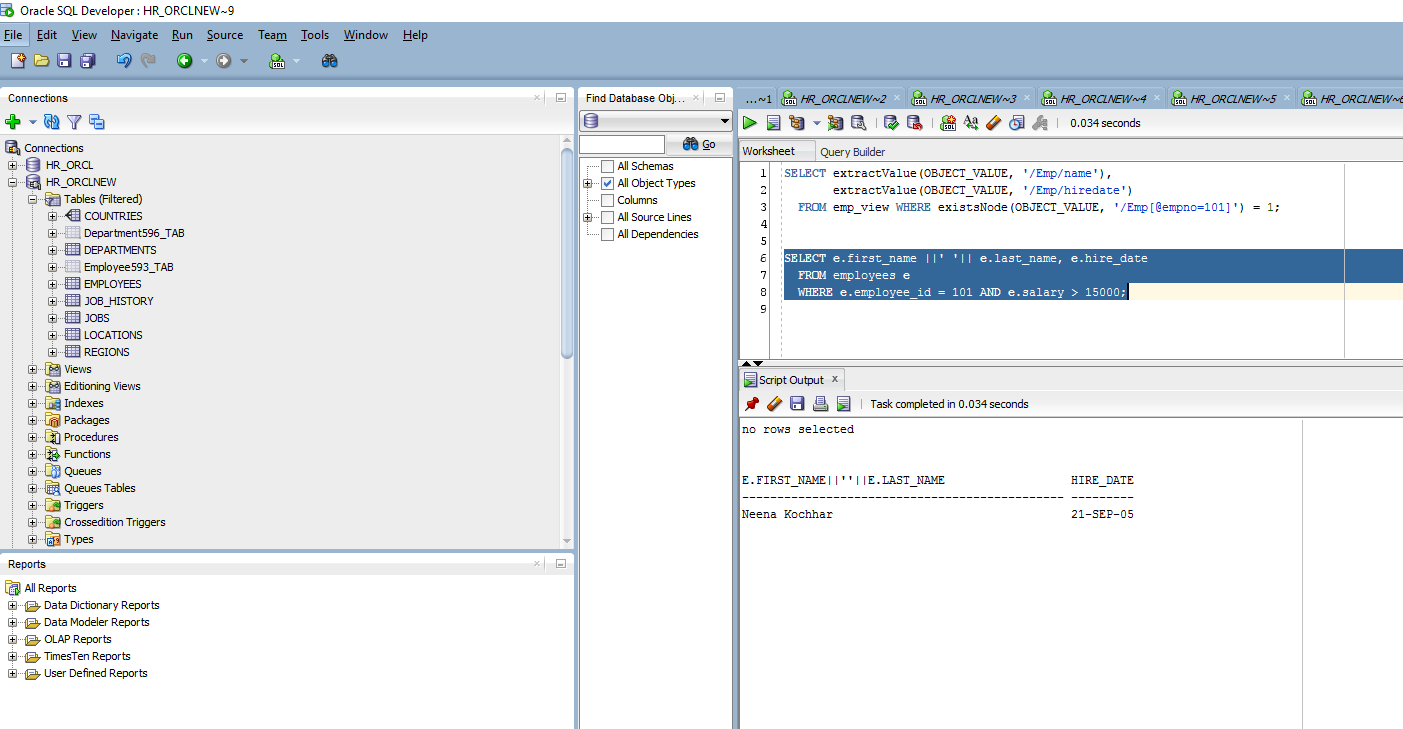
This query becomes something like the following:

SELECT e.employee\_id FROM employees e WHERE e.salary > 15000;

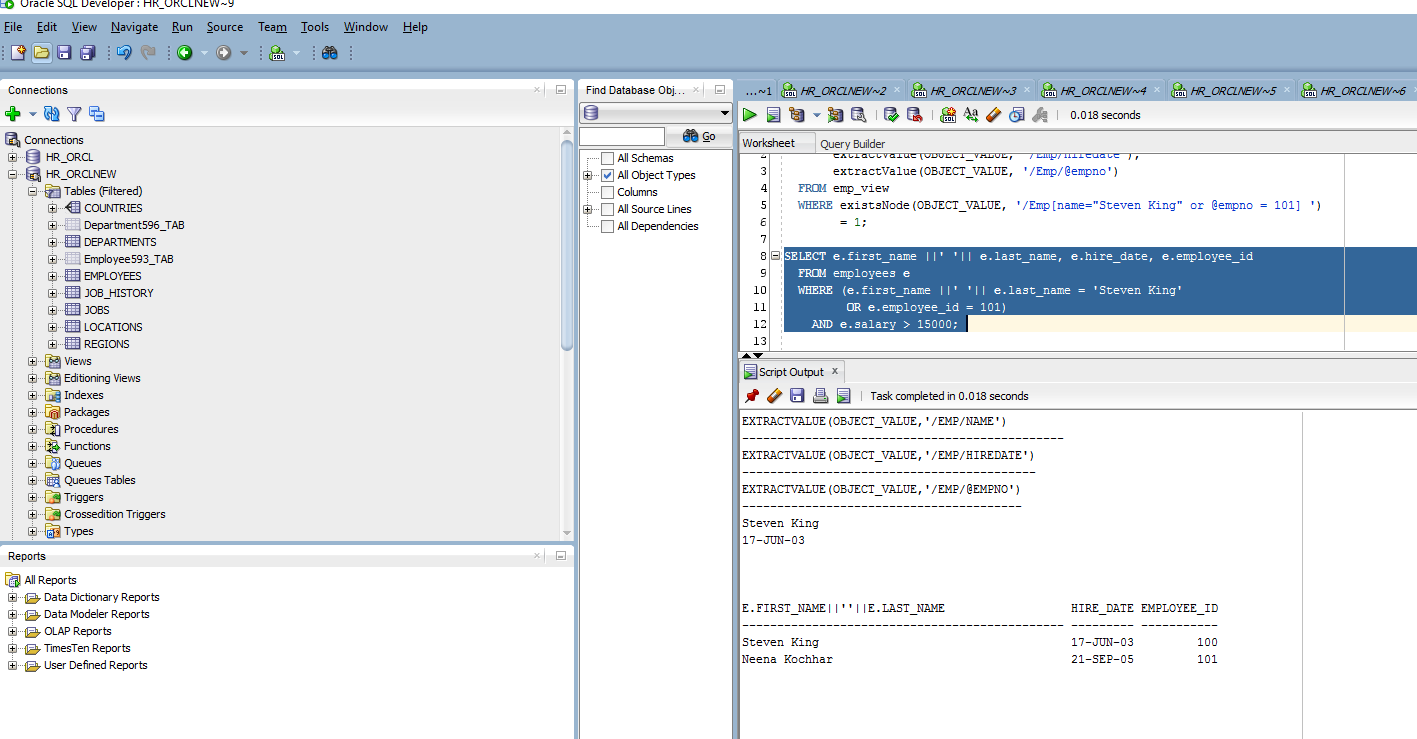


The rewritten query is a simple relational query. The **extract** expression followed by **getNumberVal()** is rewritten down to the relational column access as defined in view **emp\_view**. What does this mean?

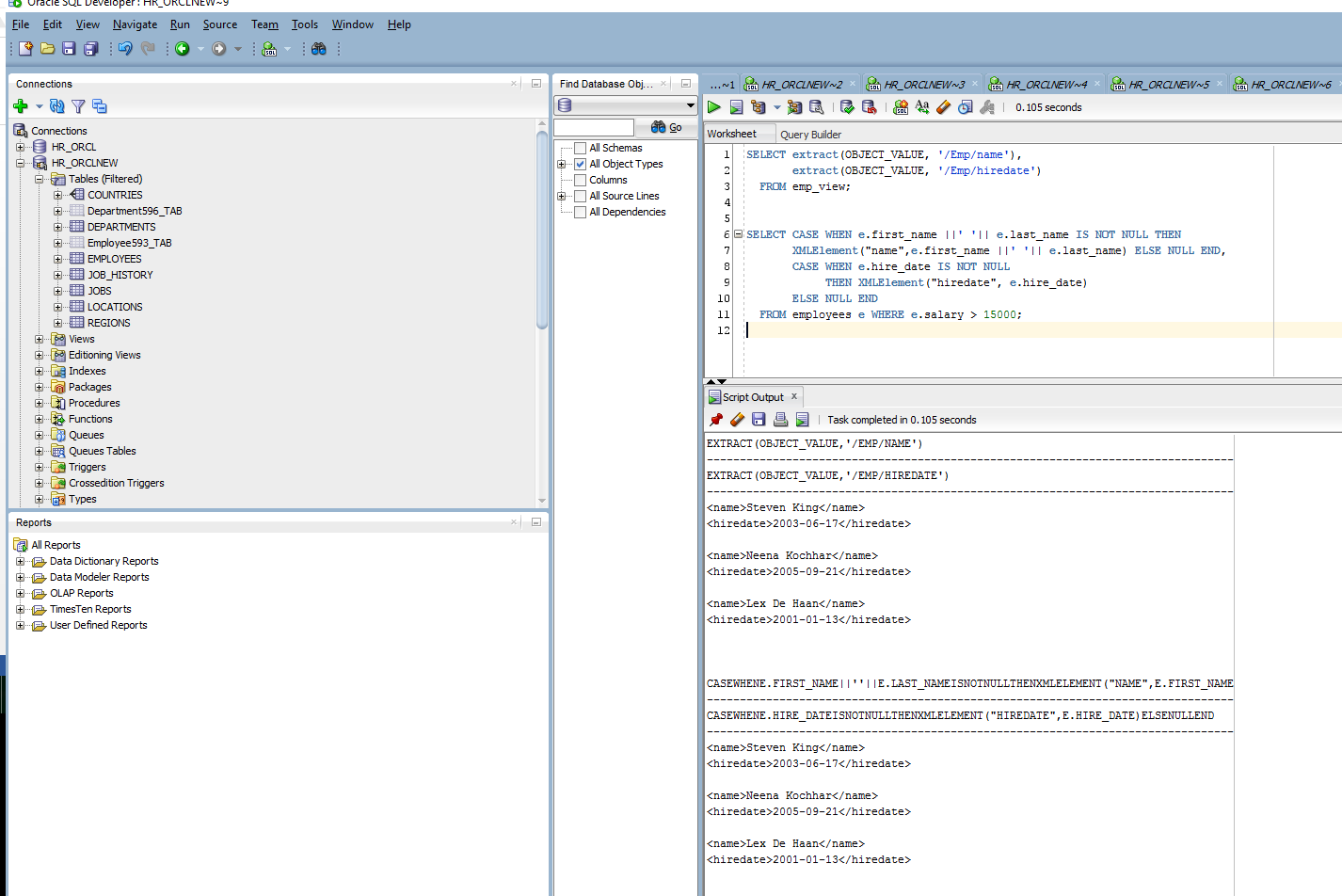
Querying with SQL function **existsNode** to select from view **emp\_view**: non matching query results, when they should be similar – data wise.



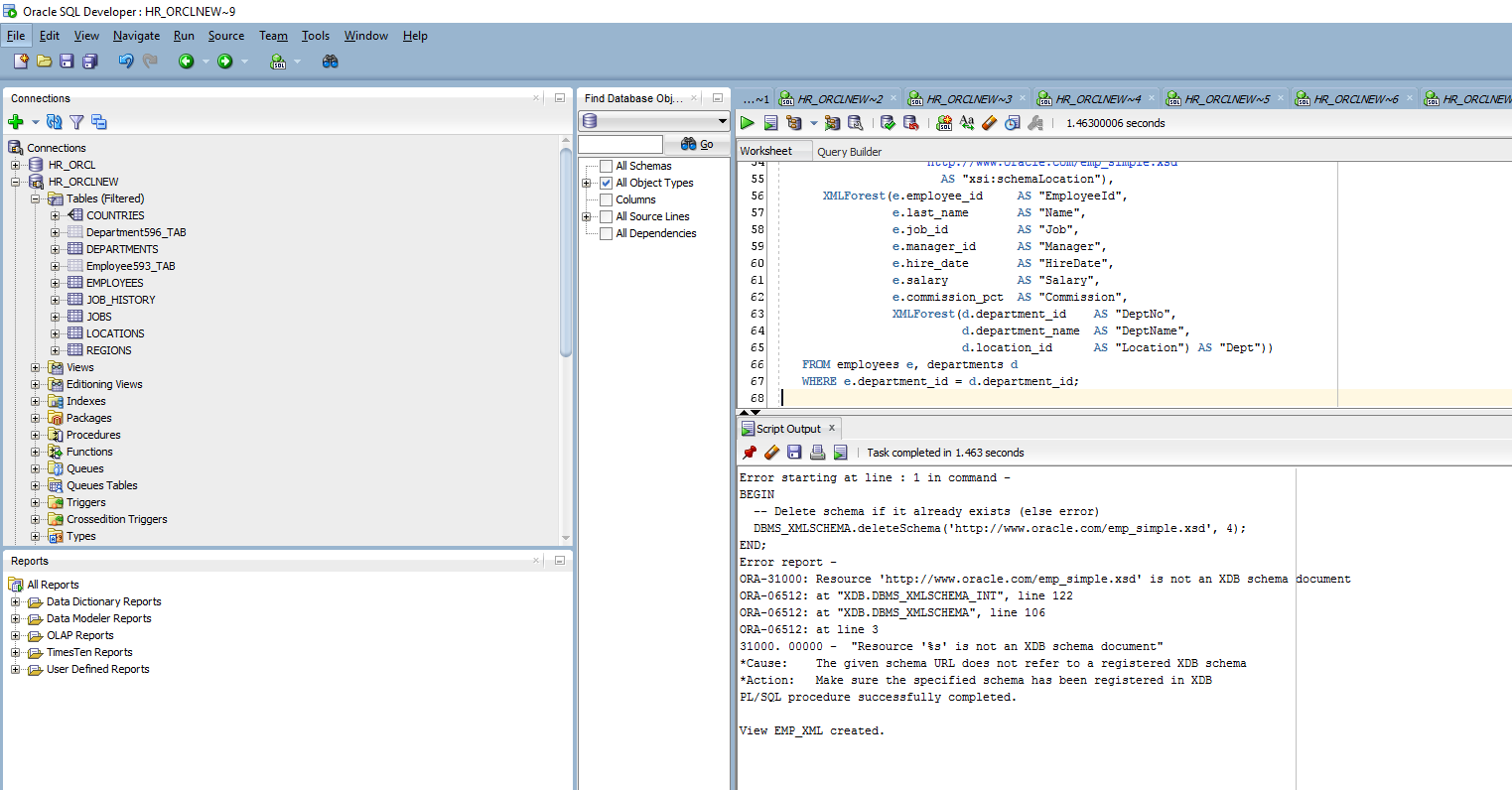
Querying with **existsNode** to select from view **emp\_view**: non matching query results, when they should be similar – data wise.



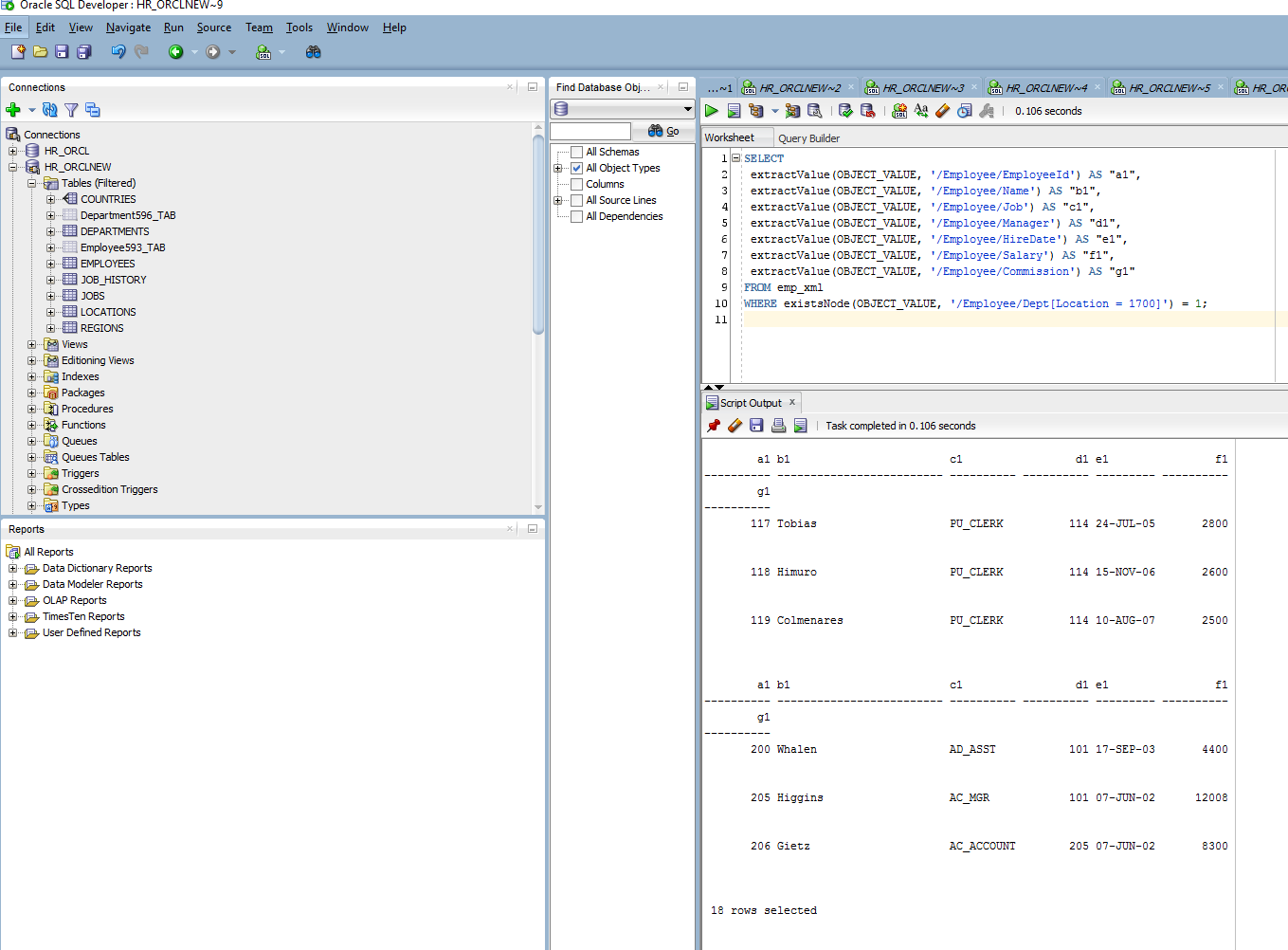
* The rewritten query is a simple relational query. The XPath predicate in the **existsNode** expression is rewritten down to the predicate over relational columns as defined in view **emp\_view**. What does this mean?
* Querying with **extract** to select from view **emp\_view**:



***Example 18-21 XML-Schema-Based Views Constructed With SQL/XML***



A query using the SQL function **extractValue** to select from **emp\_xml**:



This query becomes something like the following:

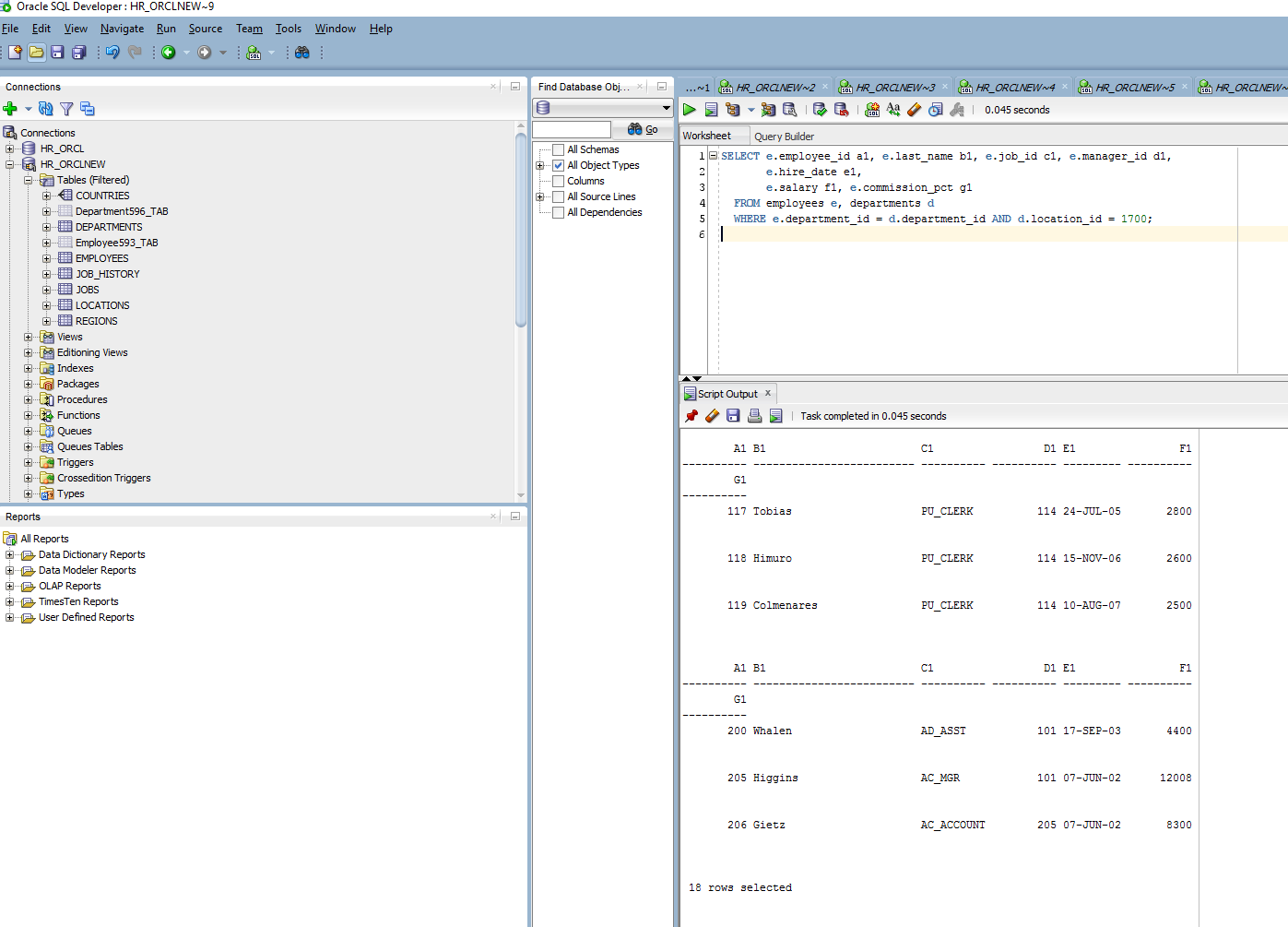
SELECT e.employee\_id a1, e.last\_name b1, e.job\_id c1, e.manager\_id d1,

e.hire\_date e1,

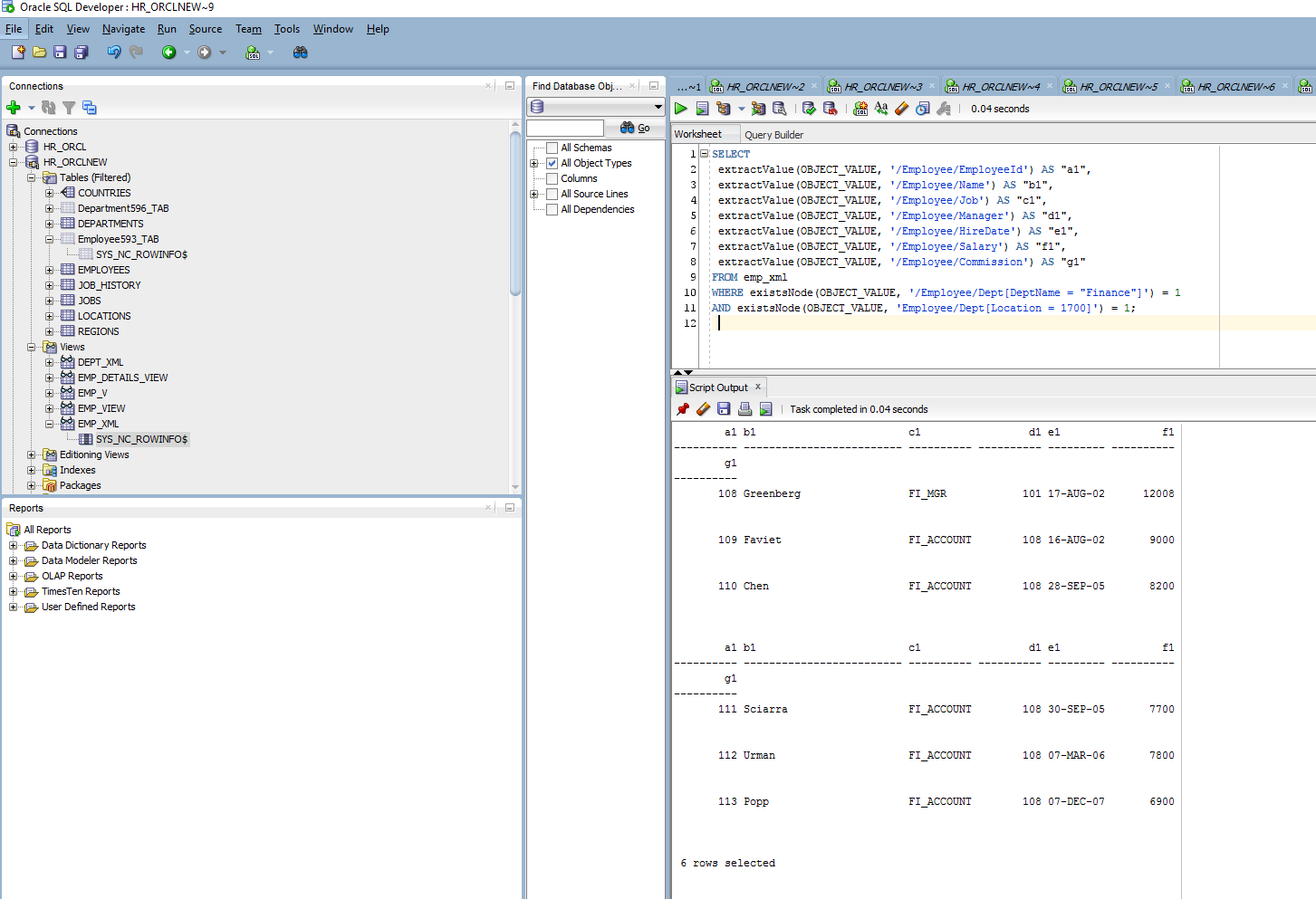
e.salary f1, e.commission\_pct g1

FROM employees e, departments d

WHERE e.department\_id = d.department\_id AND d.location\_id = 1700;



Querying with SQL function **existsNode** to select from view **emp\_xml**: there’s a problem with the WHERE clause. Had to modify the WHERE clause to pass the existsNode() function twice. For some reason, Oracle SQL Developer did not like the AND being used in the second parameter of the existsNode() function.

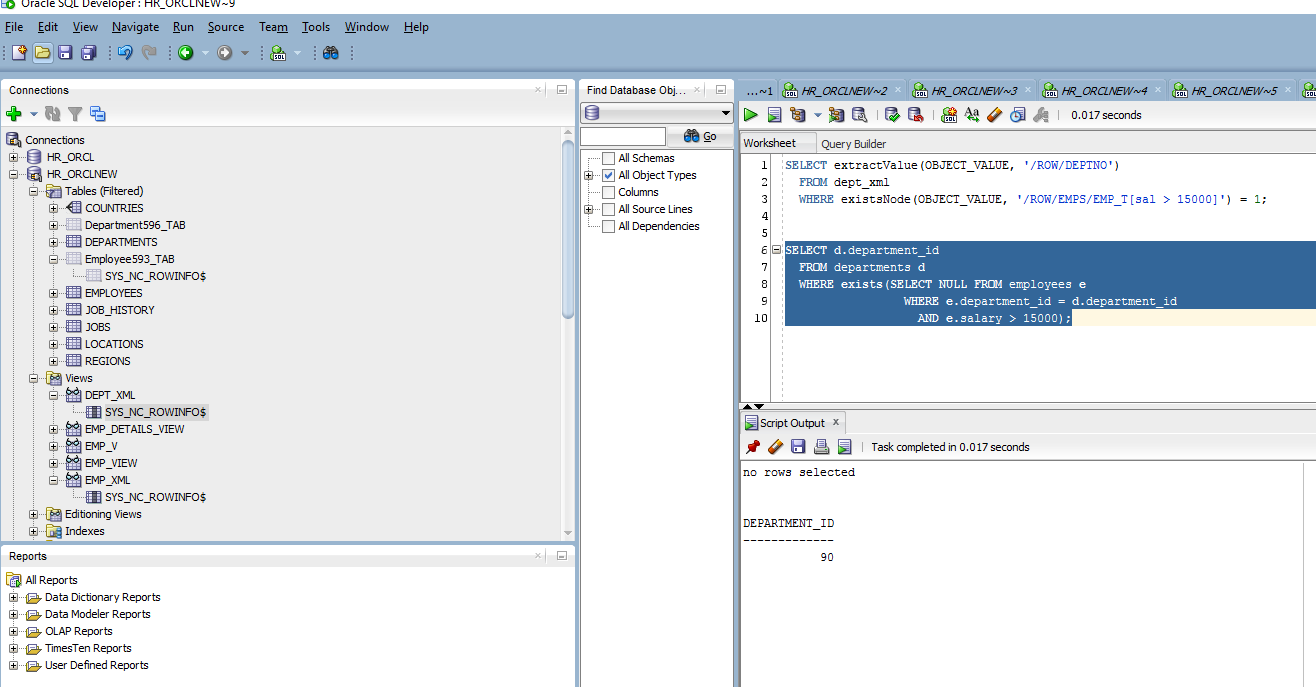


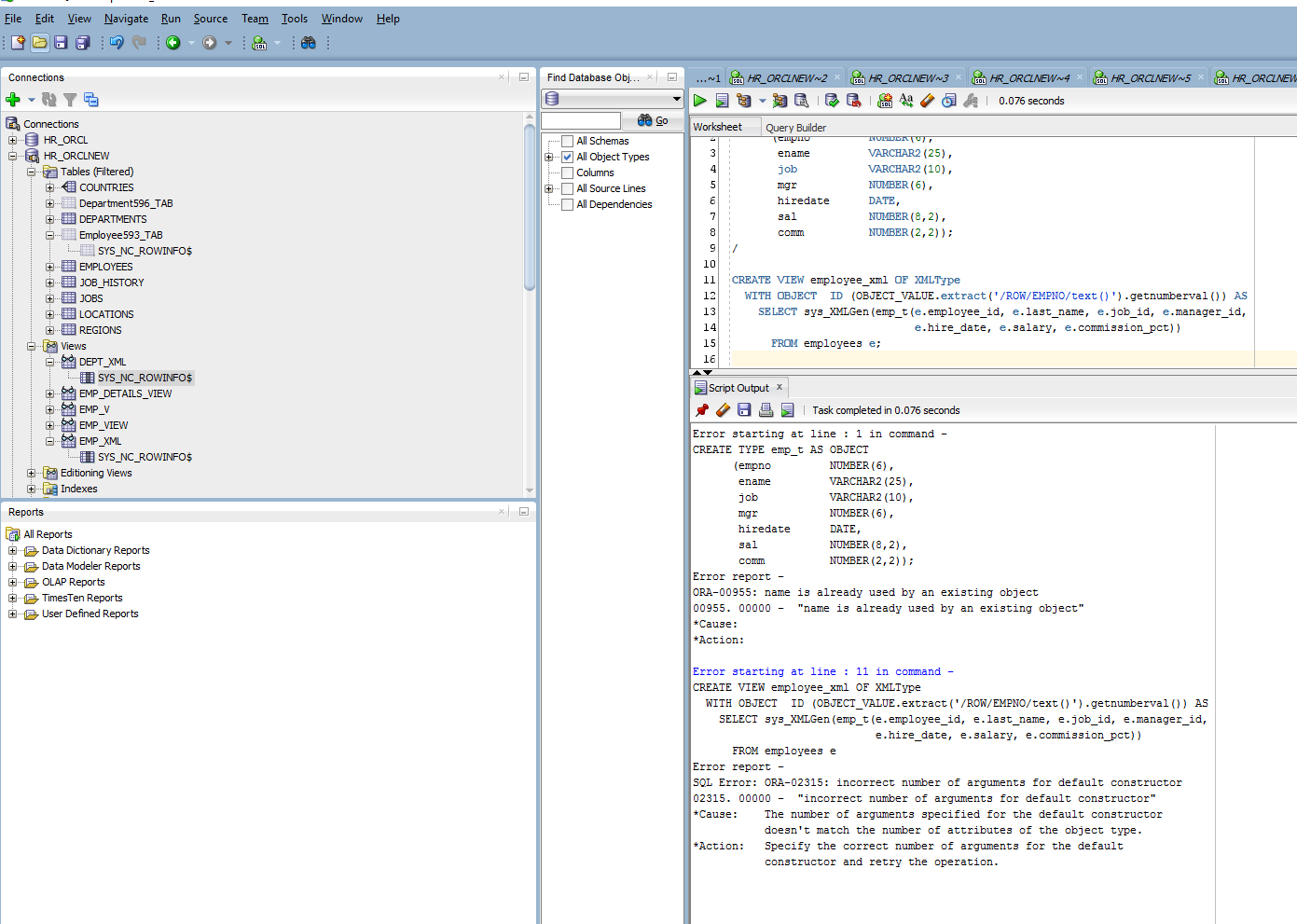
Relational query version of above:

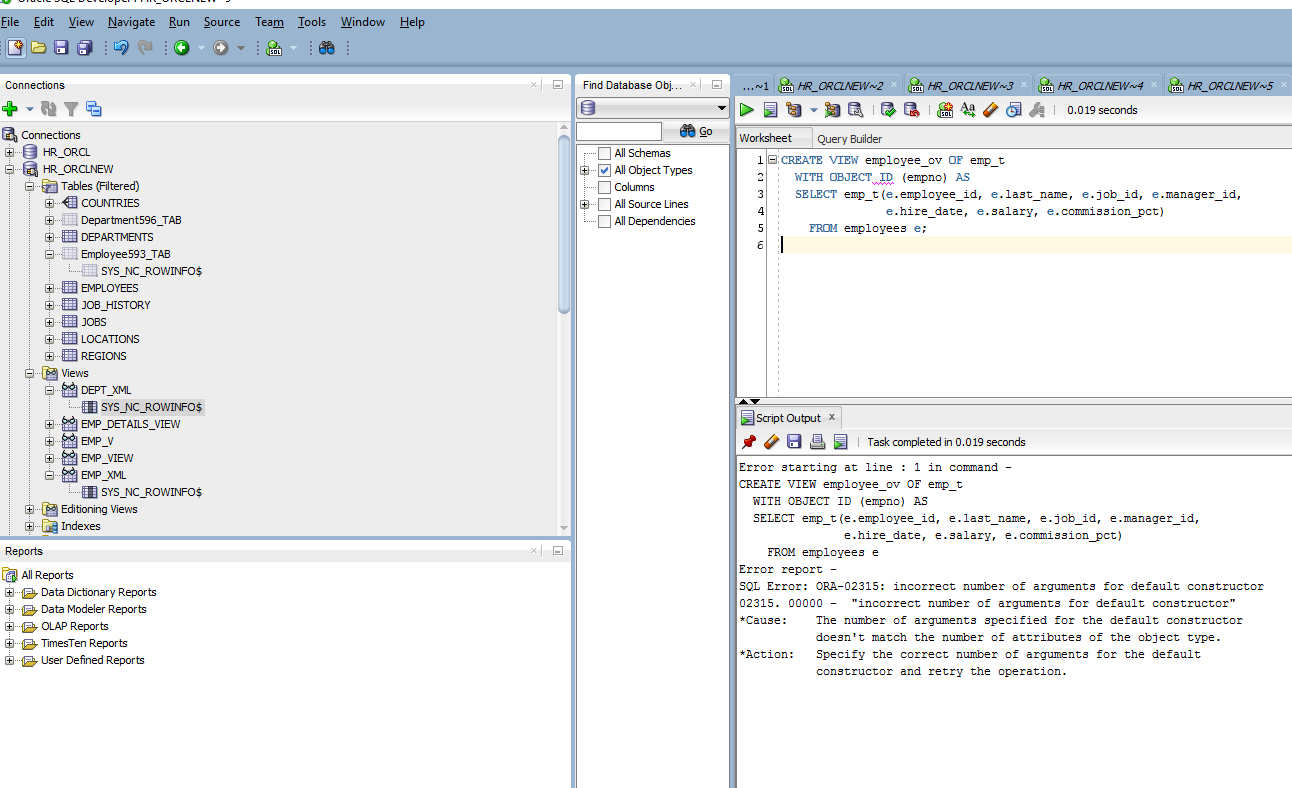


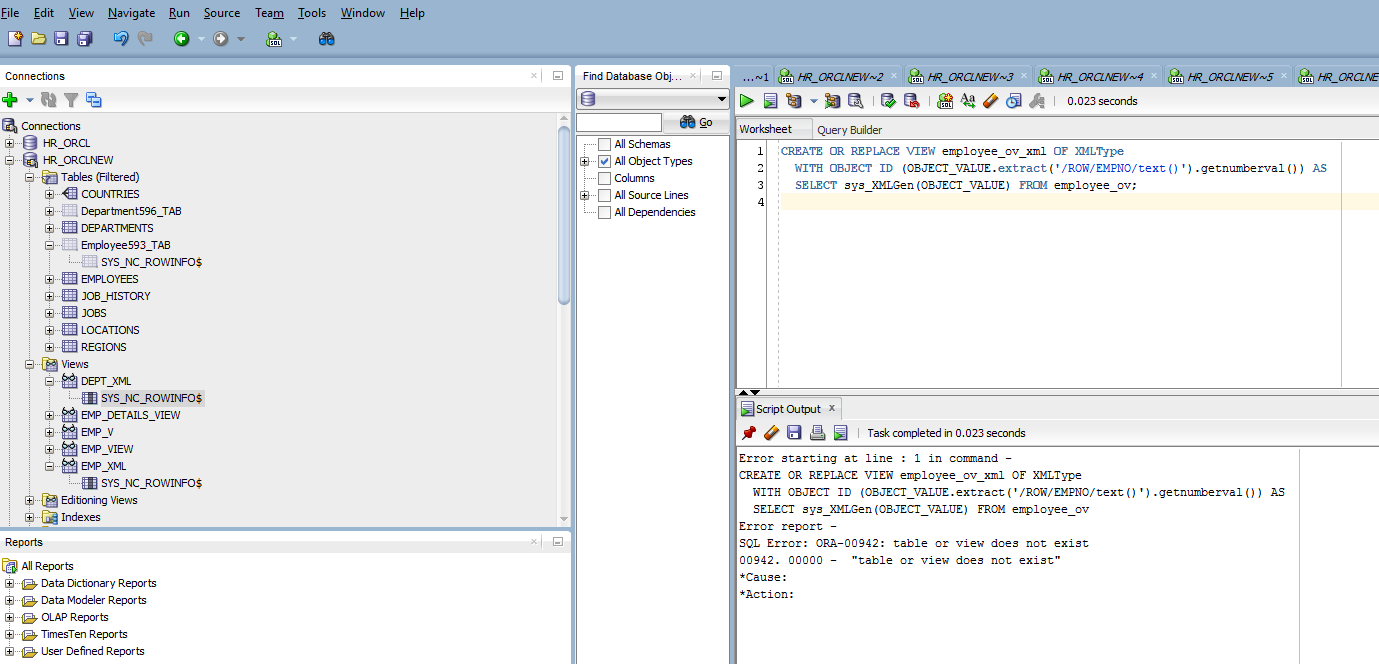
### Views Using Object Types, Object Views, and SYS\_XMLGEN

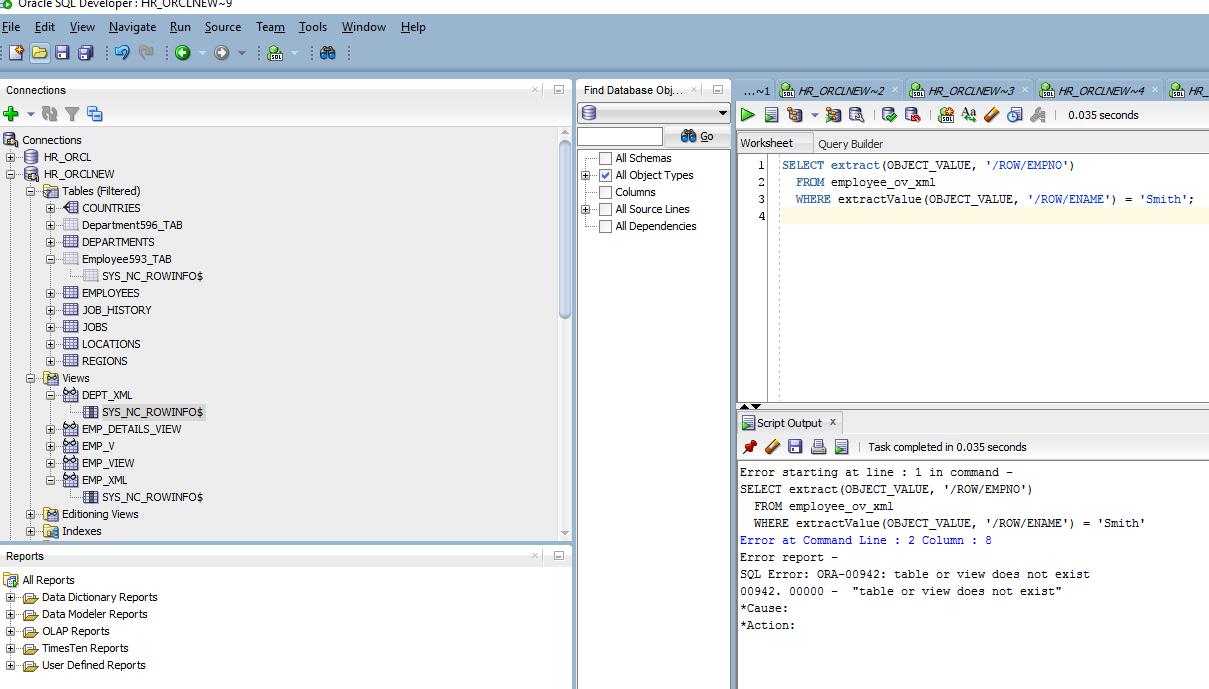
***Example 18-22 Non-Schema-Based Views Constructed Using SYS\_XMLGEN***

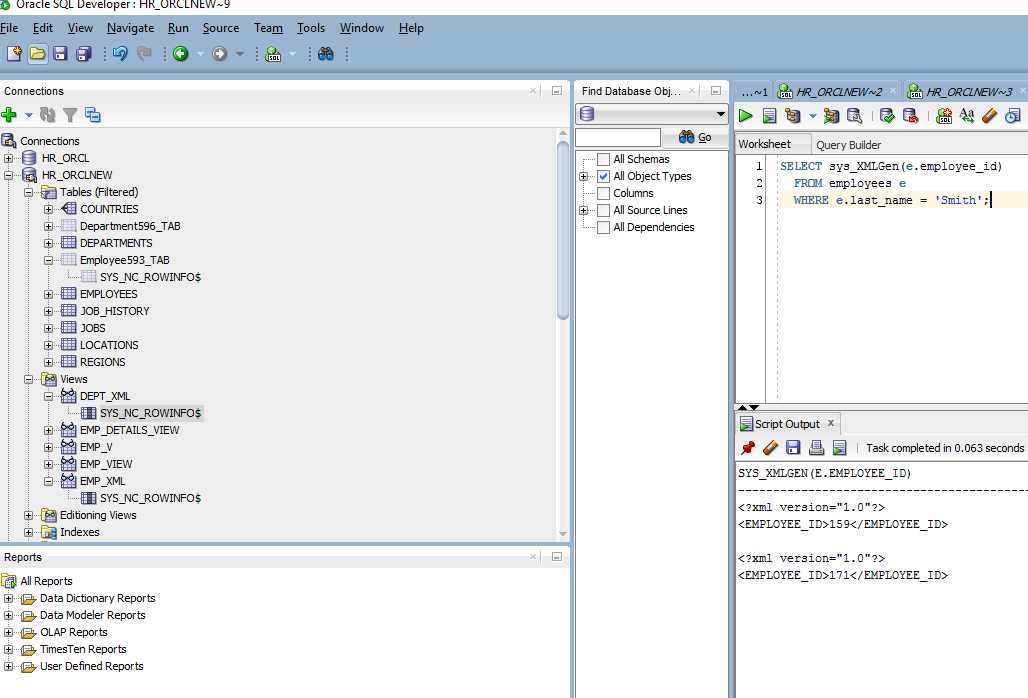


***Example 18-23 Non-Schema-Based Views Constructed Using SYS\_XMLGEN on an Object View***

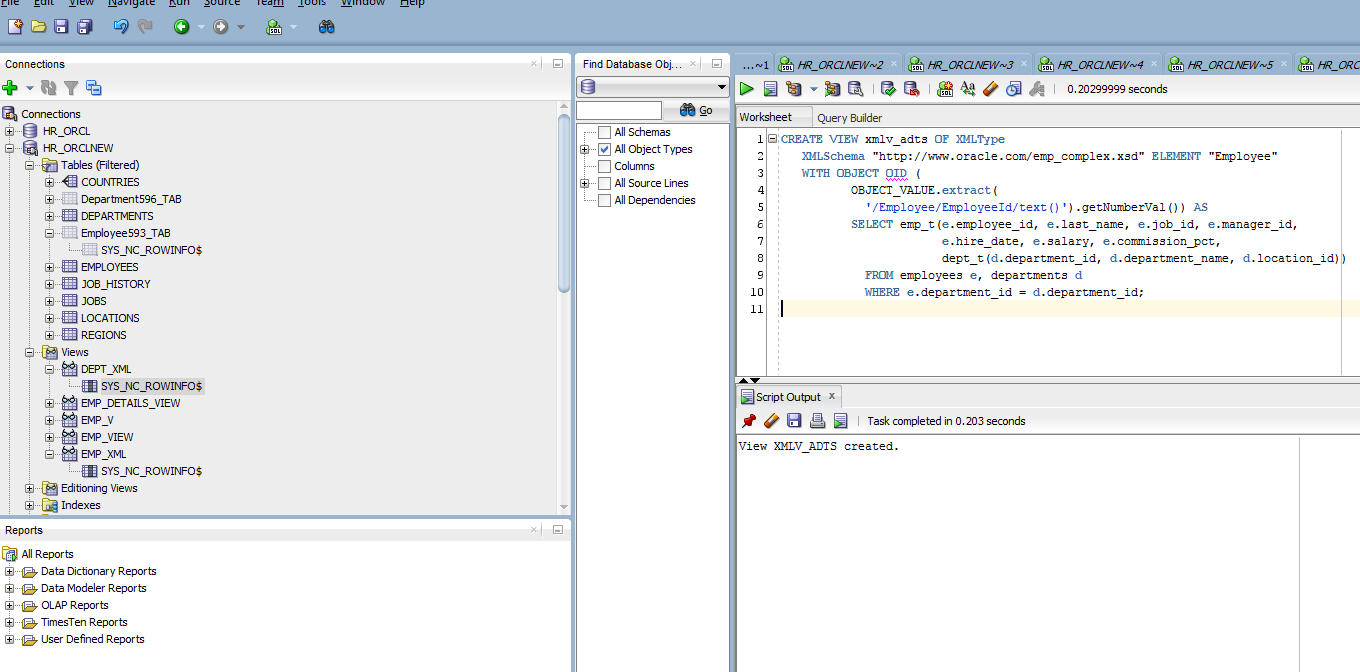


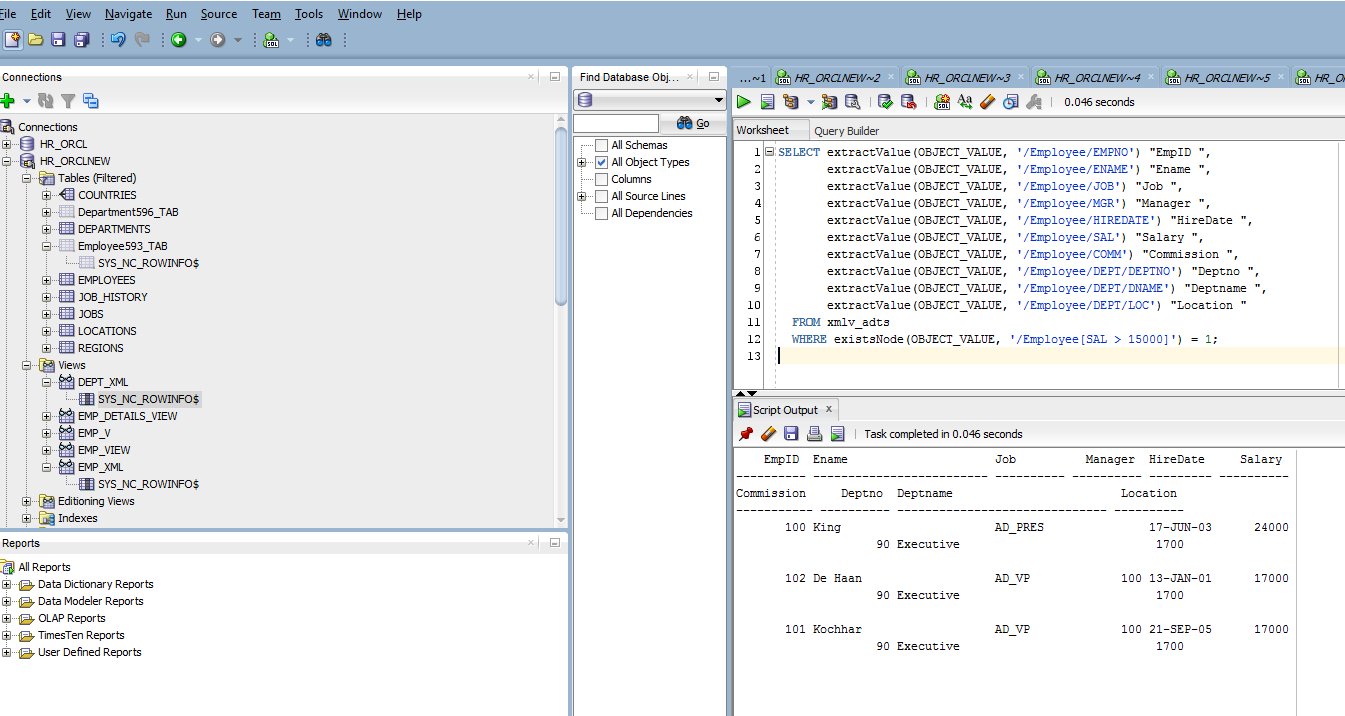


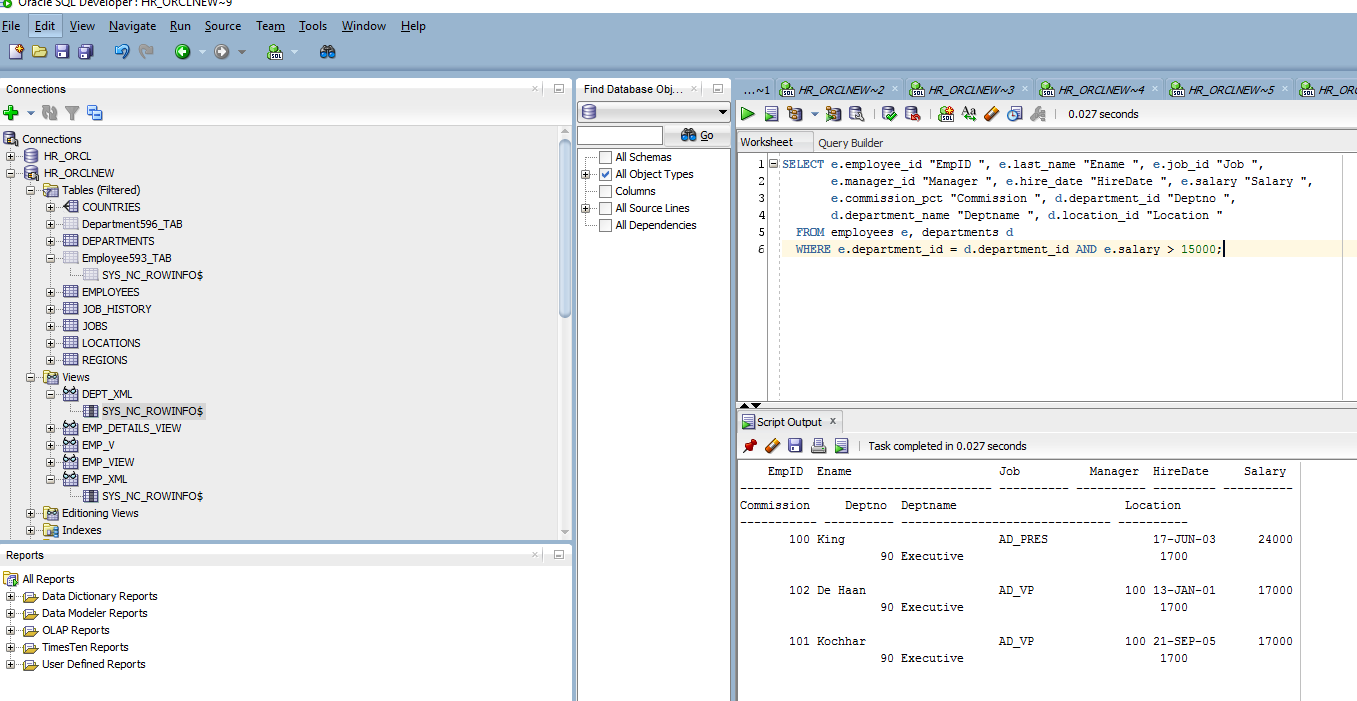




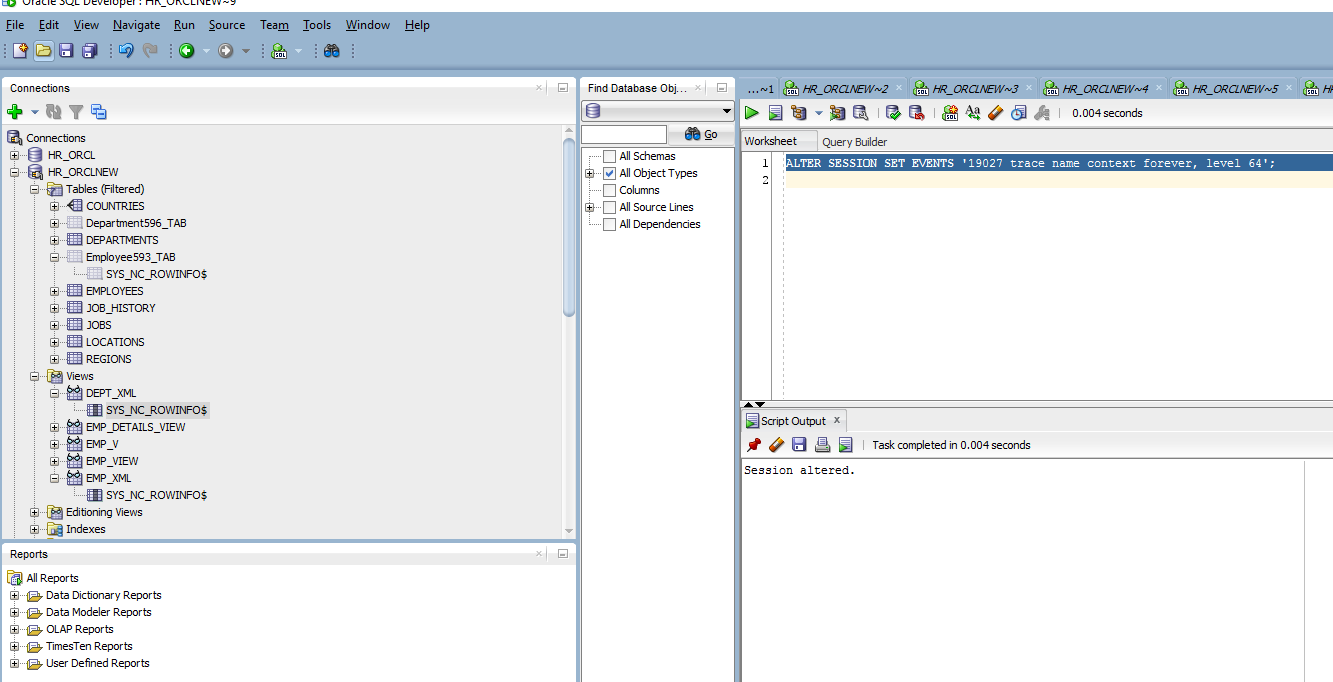
***Example 18-24 XML-Schema-Based Views Constructed Using Object Types***

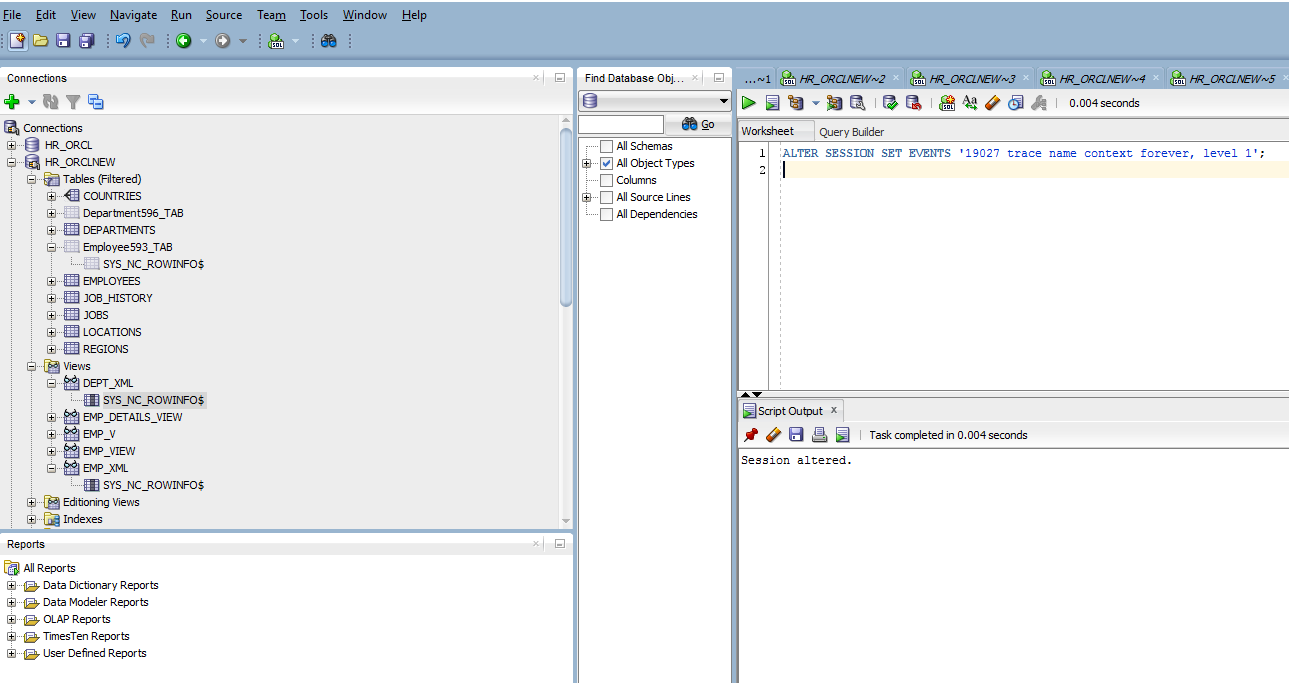


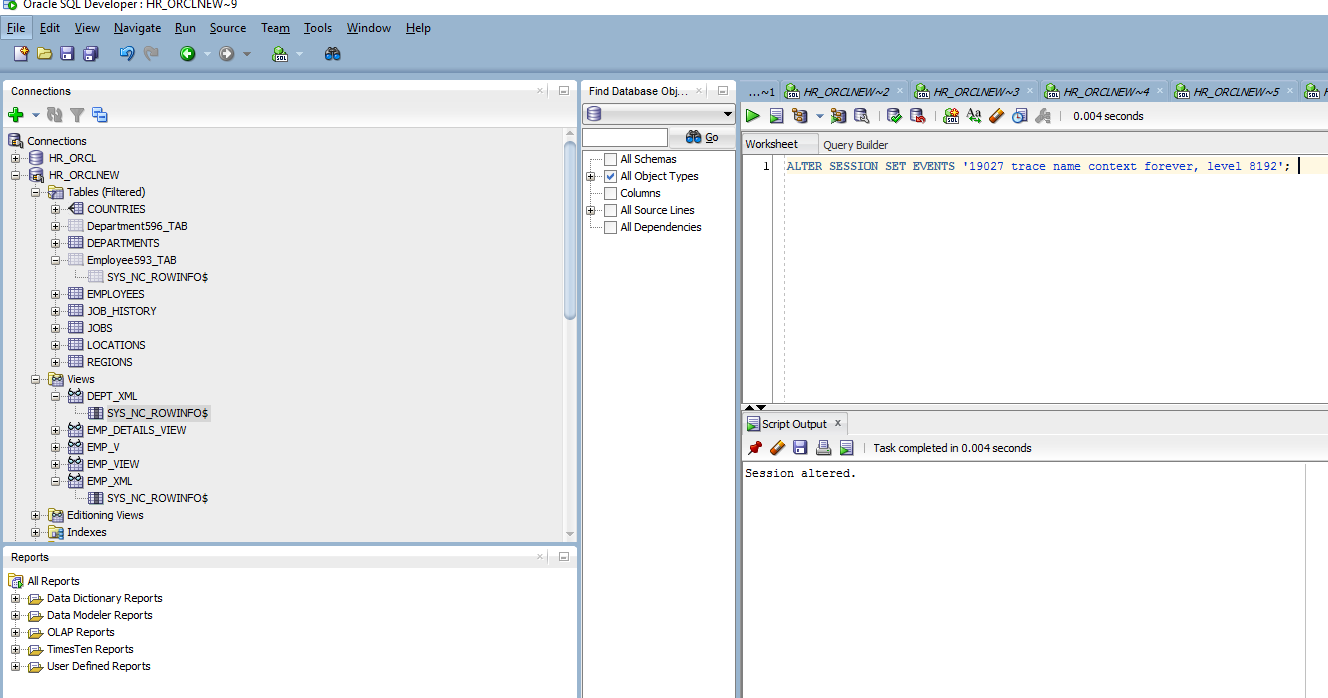




### XPath Rewrite Event Trace







## **Generating XML Schema-Based XML Without Creating Views**

***Example 18-25 Generating XML Schema-Based XML Without Creating Views***

