**Nombre de proyecto: GENERADOR DE CÓDIGO DIAN**

**DIAN CODE GENERATOR**

**Versión 1.0**

| Fecha | Versión | Descripción | Autor |
| --- | --- | --- | --- |
| 2021-03-01 | 1.01 | Versión inicial. | Nelson Javier Hurtado Pinto |
|  |  |  |  |

**CONTENIDO**

[1. Objetivo del documento 5](#_Toc65255421)

[2. DISEÑO DE DIAN CODE GENERATOR 5](#_Toc65255422)

[3. IMPLEMENTACIÓN DE DIAN CODE GENERATOR 6](#_Toc65255423)

[4. CONFIGURACIÓN DE LOS PROYECTOS 7](#_Toc65255424)

[5. GENERACIÓN DEL CÓDIGO FUENTE 9](#_Toc65255425)

**TABLA DE ILUSTRACIONES**

[Ilustración 1 Metamodelo origen 5](#_Toc65255437)

[Ilustración 2 Metamodelo de destino 6](#_Toc65255438)

[Ilustración 3 Cadena de transformación 6](#_Toc65255439)

[Ilustración 4 Importación de proyectos en la instancia 1 de Eclipse 7](#_Toc65255440)

[Ilustración 5 Ejecución de la segunda instancia de Eclipse 8](#_Toc65255441)

[Ilustración 6 archivo modeloContribuyentes.sql 9](#_Toc65255442)

[Ilustración 7 Ejemplo del uso de anotaciones y comentarios 11](#_Toc65255443)

[Ilustración 8 Generación del modelo de entrada a partir del script sql 12](#_Toc65255444)

[Ilustración 9 Ejecución de la tarea de ant que genera el código fuente 13](#_Toc65255445)

[Ilustración 10 Visualización del código generado 14](#_Toc65255446)

# Objetivo del documento

Este documento tiene como objetivo ilustrar al usuario sobre la configuración y uso de la herramienta **Dian Code Generator,** la cual permite genera el código en lenguaje Java, Jsp y Xhtml para las operaciones crear, listar, actualizar y eliminar.

Con esta herramienta se busca agilizar el proceso de desarrollo de software en la Coordinación de Desarrollo de Sistemas de Información, al evitarle al programador el tener que digitar código repetitivo que puede ser generado con herramientas automatizadas.

En este documento se explica cómo se diseñó la herramienta, cómo funciona, qué configuraciones requiere y cómo debe ser ejecutada.

# DISEÑO DE DIAN CODE GENERATOR

Esta herramienta fue construida se base en modelos y utiliza tecnologías basadas en lenguajes Epsilon como Xtext y Sirius. Se diseñaron dos metamodelos, el primero de ellos define la estructura y relaciones que debe tener la entrada la cual corresponde a un modelo de datos entidad relación y se denomina metamodelo origen, como se muestra en la Ilustración 1 Metamodelo origen. El segundo metamodelo define la estructura y relaciones que representan la arquitectura MUISCA, y que se consideran como el modelo de destino del generador, como se muestra en la Ilustración 2 Metamodelo de destino. A continuación, se muestran los diagramas de los metamodelos diseñados:

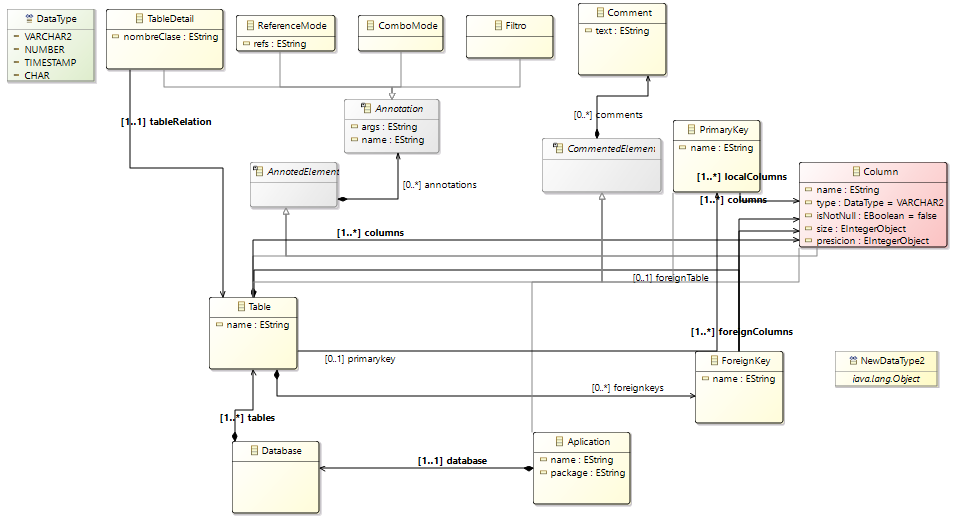


Ilustración 1 Metamodelo origen

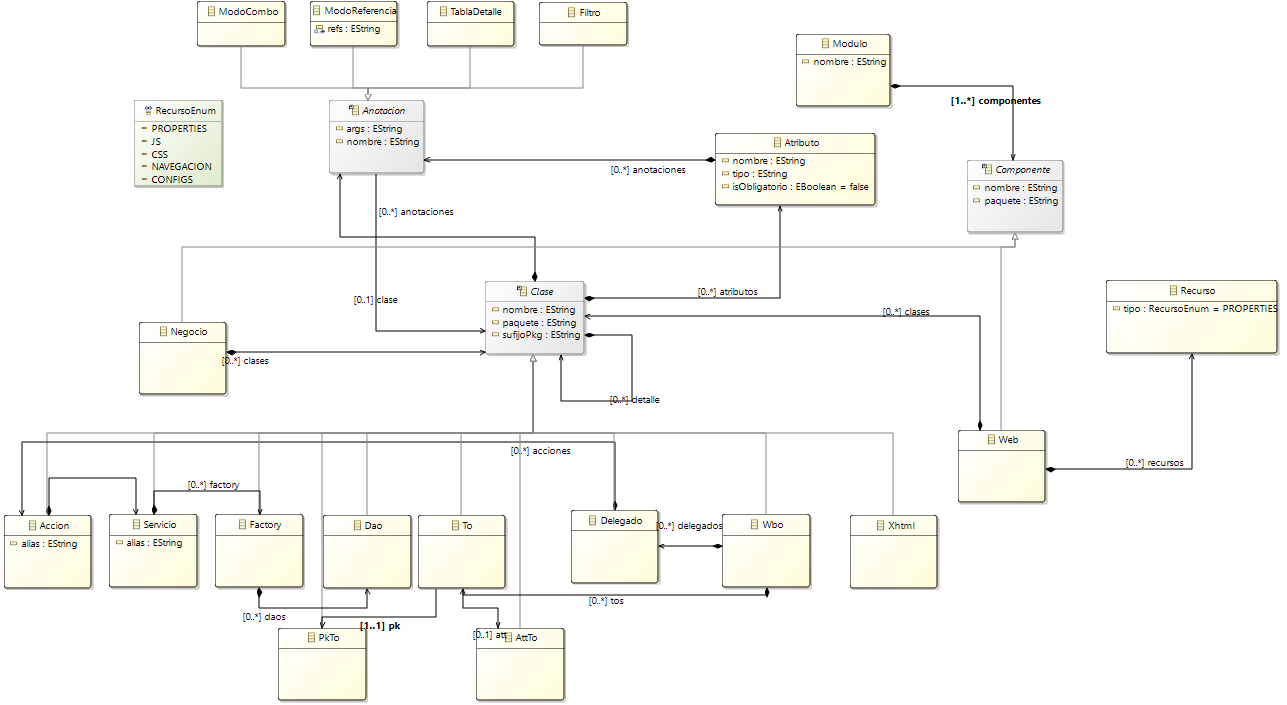


Ilustración 2 Metamodelo de destino

La herramienta Dian Code Generator genera el modelo de entrada asegurando que cumple con el metamodelo definido para ello y su vez lo valida con una gramática. Luego con la ayuda de una transformación de modelo a modelo (M2M) genera un modelo que cumple con el metamodelo destino que en este caso es una representación de la arquitectura Muisca. Una vez se ha obtenido el modelo de destino se aplican una serie de transformaciones de modelo a texto (M2T) que generan los artefactos de código requeridos. En Ilustración 3 Cadena de transformación se muestra el esquema de transformaciones que utiliza Dian Code Generator.

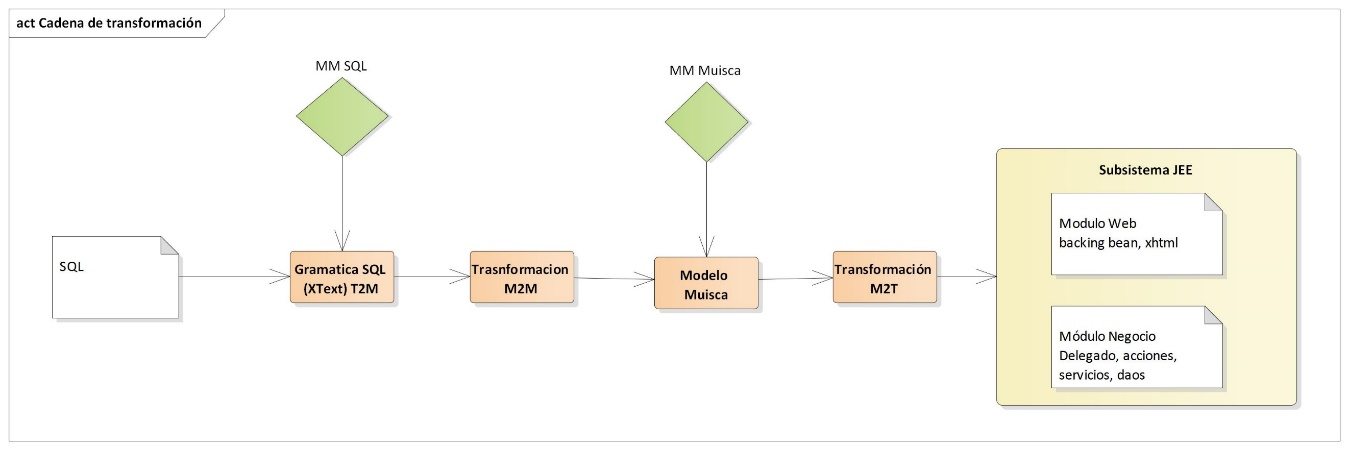


Ilustración 3 Cadena de transformación

# IMPLEMENTACIÓN DE DIAN CODE GENERATOR

Esta herramienta se implementó en el IDE Eclipse configurado con los plugins de modeling y Epsilon. Consta de varios proyectos que especializan las tareas que cada uno de ellos realiza.

# CONFIGURACIÓN DE LOS PROYECTOS

Se abre el Eclipse Epsilon y se importan los proyectos contenidos en la carpeta genedorCodigo, a excepción del que tiene el nombre CadenaArquitectura, como se muestra en la Ilustración 4 Importación de proyectos en la instancia 1 de Eclipse. Una vez se han importado los proyectos en esta primera instancia se da clic derecho sobre el proyecto **co.gov.dian.xtext.arquitecturadian.dsl.sql** como se muestra en la Ilustración 5 Ejecución de la segunda instancia de Eclipse.

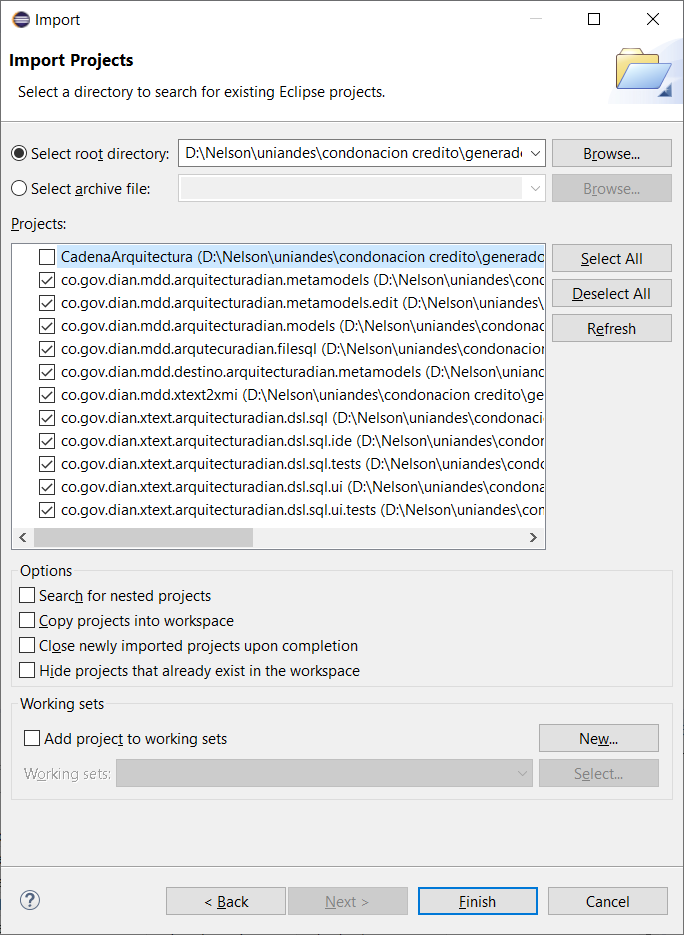


Ilustración 4 Importación de proyectos en la instancia 1 de Eclipse

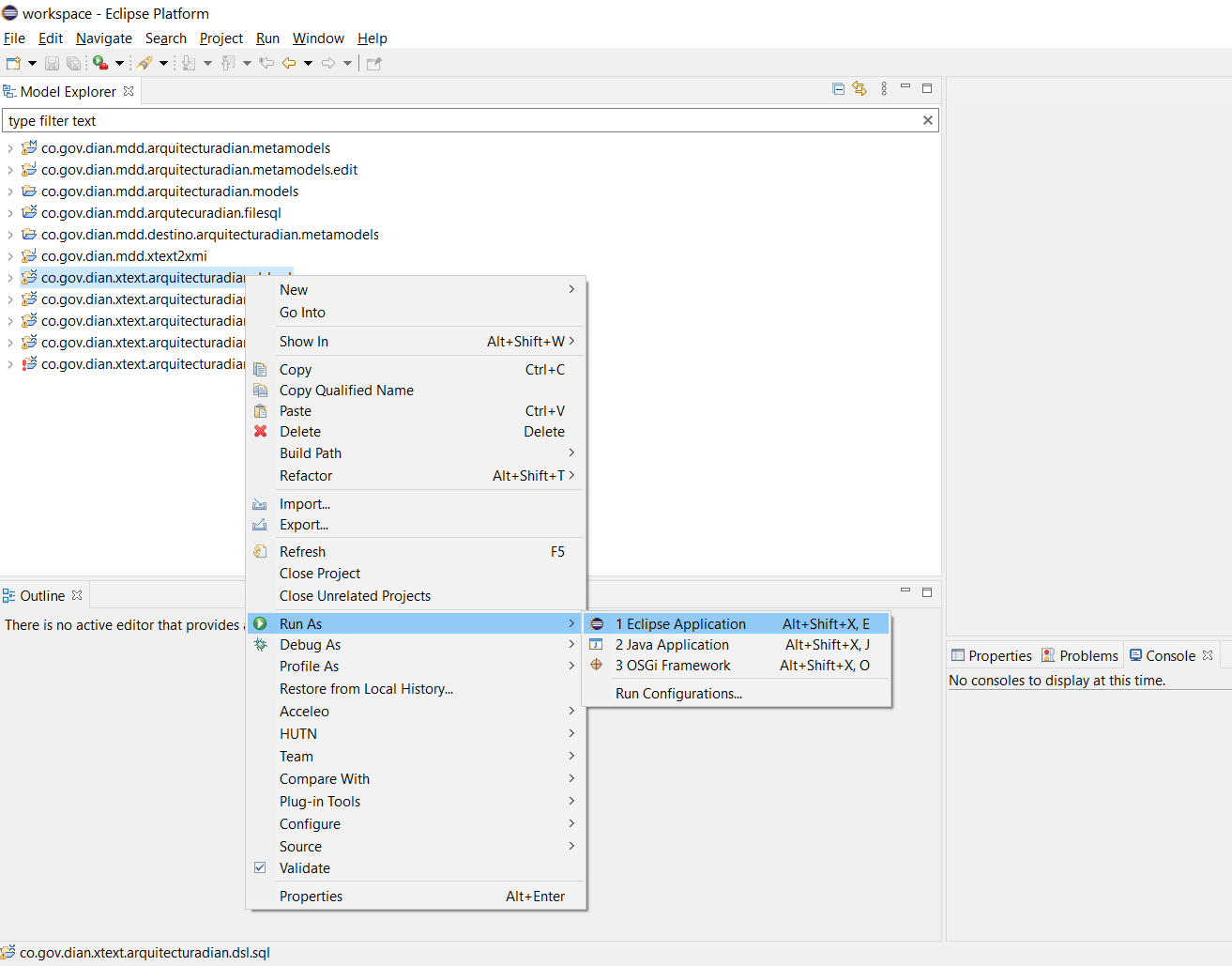
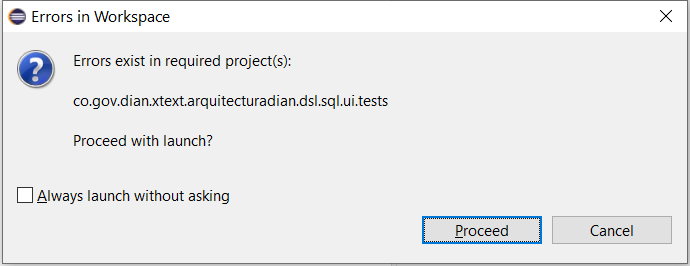


Ilustración 5 Ejecución de la segunda instancia de Eclipse

En la nueva instancia se importa el proyecto CadenaArquitectura que se encuentra dentro de la carpeta generadorCodigo y que no se abrió en la instancia 1. Si aparecen mensajes indicando errores entes de lanzar la segunda instancia se deben obviar:



# GENERACIÓN DEL CÓDIGO FUENTE

En la segunda instancia se debe abrir la perspectiva **Modeling,** luego se abre la carpeta **models** y se ubica el archivo **modeloContribuyentes.sql,** este archivo contiene el script a partir del cual se genera el código, y tiene la estructura que muestra en la Ilustración 6 archivo modeloContribuyentes.sql.

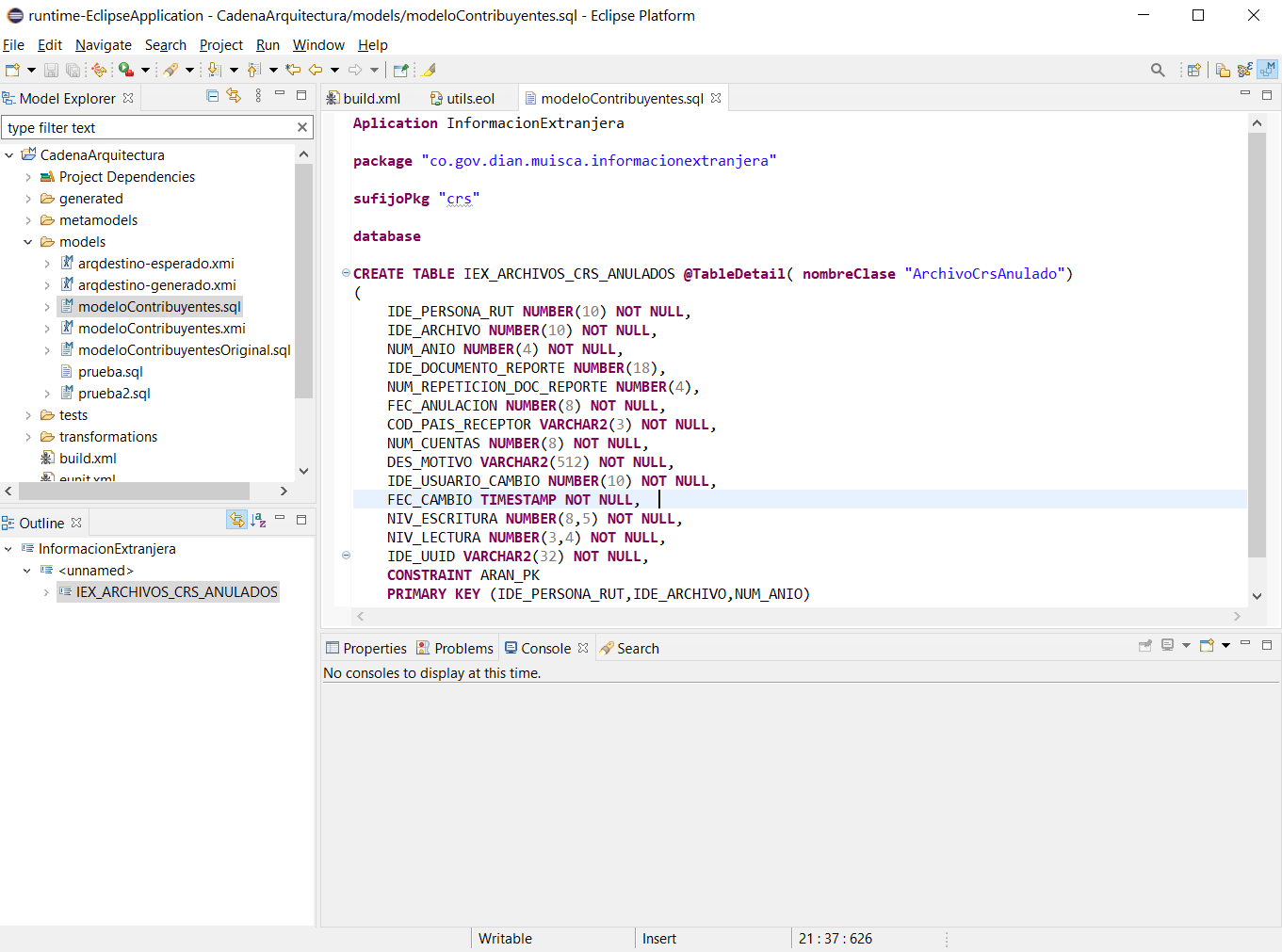


Ilustración 6 archivo modeloContribuyentes.sql

Cuando se edita este archivo se valida contra la gramática y si encuentra caracteres o estructuras no permitidas lo indica marcando la línea que presenta el error.

La estructura que debe tener el archivo es la siguiente:

* ***Aplication*** es la palabra reservada que antecede el nombre del módulo o subsistema.
* ***package***indica el paquete raíz del módulo o subsistema.
* ***sufijoPkg*** indica el nombre del sufijo del paquete. Por ejemplo si el paquete para los servicios es **co.gov.dian.muisca.informacionextranjera.servicios.crs** el valor de **sufijoPkg** es **crs**
* ***database***indica que ahí inicia la estructura que conforma la base de datos.
* ***CREATE TABLE*** son las palabras reservadas que anteceden el nombre de la tabla. En este elemento se pueden agregar anotaciones que permiten capturar información adicional que se requiere para generar correctamente el código. Entre as posibles anotaciones se encuentran:
  + ***@TableDetail*** que tiene entre sus parámetros nombreClase en el cual se le indica al generador el nombre utilizado para nombrar los artefactos, si este nombre no se proporciona el los artefactos se nombrarán quitando el prefijo de la tabla, los caracteres \_ y se utilizará el estilo CamelCase para generarlo. Por ejemplo para la tabla IEX\_ARCHIVOS\_CRS\_ANULADOS el nombre que se asigna es ArchivosCrsAnulados.
  + ***@tableRelation*** Indica que dicha tabla tiene una referencia de llave foránea con otra tabla que se encuentra dentro del mismo script.
* **Columnas.**  Las columnas se adicionan después del carácter ( del elemento ***CREATE TABLE*** y constan de un nombre un tipo de dato, longitud y precisión y la palabra reservada **NOT NULL** si dicha columna es obligatoria. Las columnas también pueden tener anotaciones como las siguientes:
  + **@ComboMode** indica que para dicha columna en los componentes web se debe mostrar una lista de selección.
  + **@ReferenceMode** indica el tipo de información que se muestra en el combo. Por ejemplo @ReferenceMode(refs "LUGAR=108") indica que se mostrará un listado de lugares de tipo 108.
  + **@Filtro** indica que dicha columna se incluye entre los campos utilizados para hacer búsquedas.
* ***CONSTRAINT*** indica que se está definiendo una restricción, puede ser una llave primaria o una llave foránea, estas restricciones deben ir dentro de la misma definción de la tabla, no pueden ir separadas. Debe tener un nombre.
* ***PRIMARY KEY*** indica que se está definiendo una llave primaria. Entre paréntesis las columnas que forman parate de dicha llave.
* ***FOREING KEY*** indica que se está definiendo una llave foránea. Entre paréntesis las columnas que forman parate de dicha llave, y con la palabra reservada ***REFERENCES*** se indica la tabla a la cual hace referencia la llave.
* **Comentarios**. Todos los elementos permiten comentarios estos deben iniciar con la combinación de caracteres /\* y finalizar con \*/. Un ejemplo del uso de estos elementos puede verse en la Ilustración 6 Ejemplo del uso de anotaciones y comentarios.

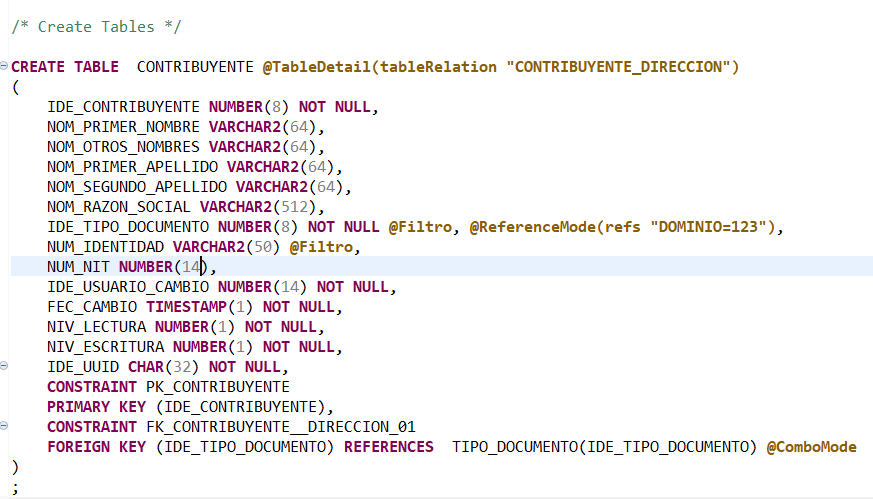
******

Ilustración 7 Ejemplo del uso de anotaciones y comentarios

**Nota**: Las palabras reservadas son sensibles a mayúsculas y minúsculas por lo que se debe tener cuidado con su utilización para evitar errores.

Una vez se ha generado el archivo modeloContribuyentes.sql se da clic derecho sobre el nombre de este archivo y se da clic en la opción Generar Modelo desde archivo Sql, como se muestra en la Ilustración 7 Generación del modelo de entrada a partir del script sql.

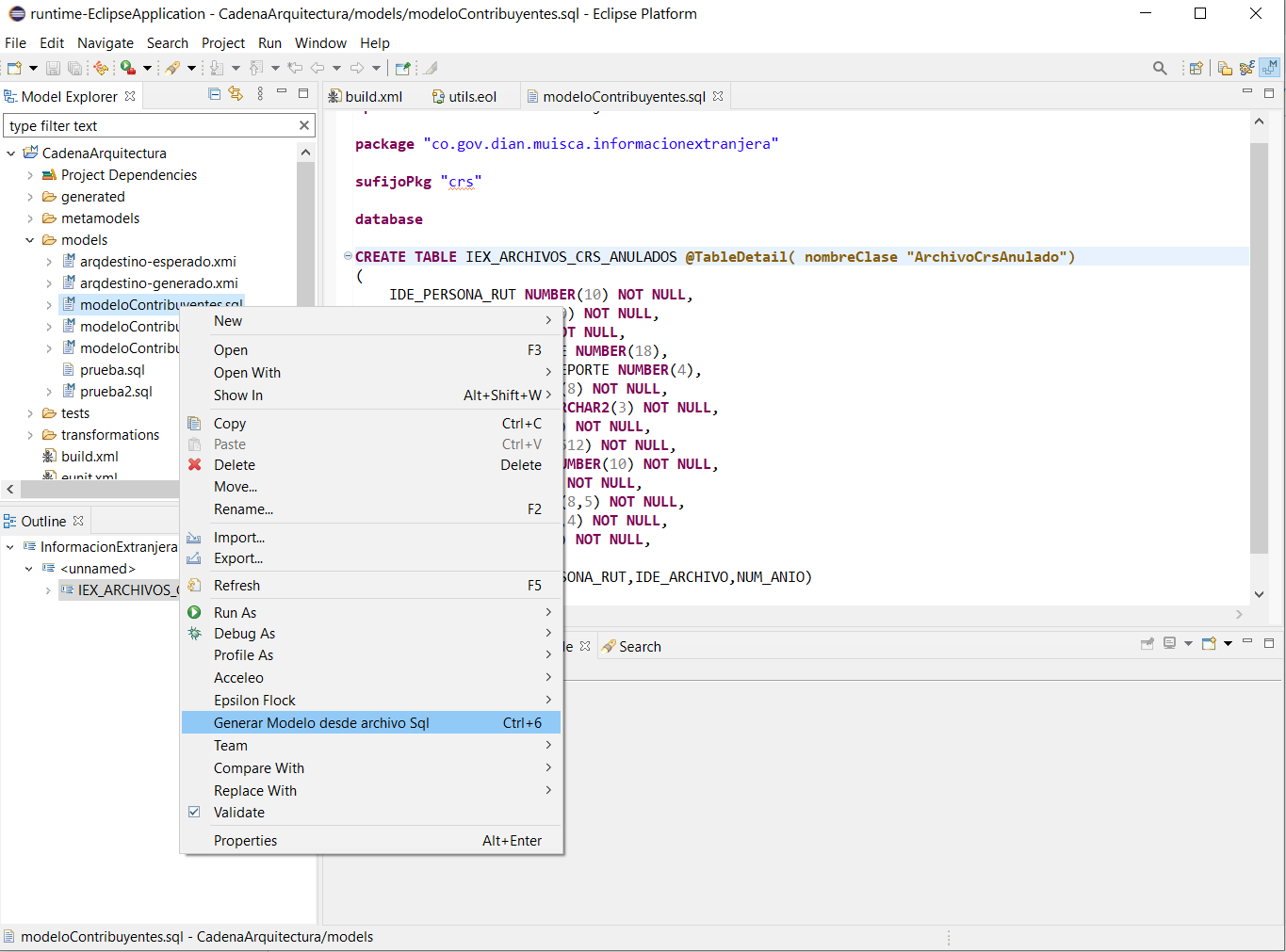


Ilustración 8 Generación del modelo de entrada a partir del script sql

Finalmente, para generar el código fuente se ejecuta la tarea de Ant build.xml como se muestra en la Ilustración 8 Ejecución de la tarea de ant que genera el código fuente.

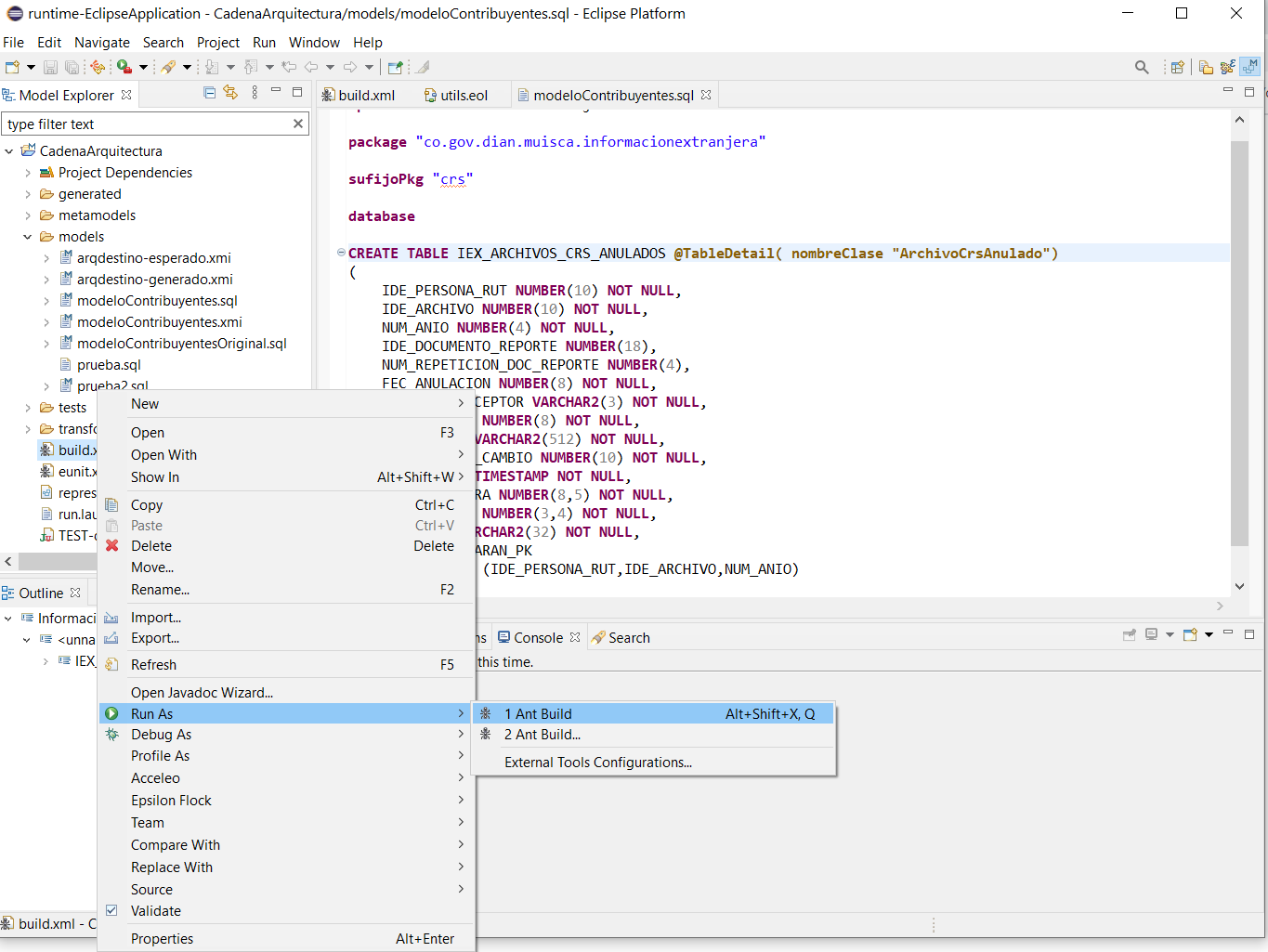


Ilustración 9 Ejecución de la tarea de ant que genera el código fuente

Una vez se ha ejecutado esta tarea, se debe refrescar el directorio generated y dentro de este aparece el código generado como se muestra en la Ilustración 9 Visualización del código generado. En esta carpeta aparen la siguiente estructura:

* **FacesMuisca**: Contiene el WBO, los recursos, la configuración de faces-config-beans.xml y las jsp de listar, crear y actualizar con su respectiva implementación de Muisca faces.
* **PrefBuses**: Donde **Pref** corresponde a las tres primeras letras del nombre del módulo. En esta carpeta se generan las acciones, servicios, DAOs y Tos.
* **WebNombremodulo**: Contiene el WBO, los recursos, el servlet de pruebas, la configuración de faces-config-beans.xml y las páginas xhtml de listar, crear y actualizar con su respectiva implementación Primefaces.

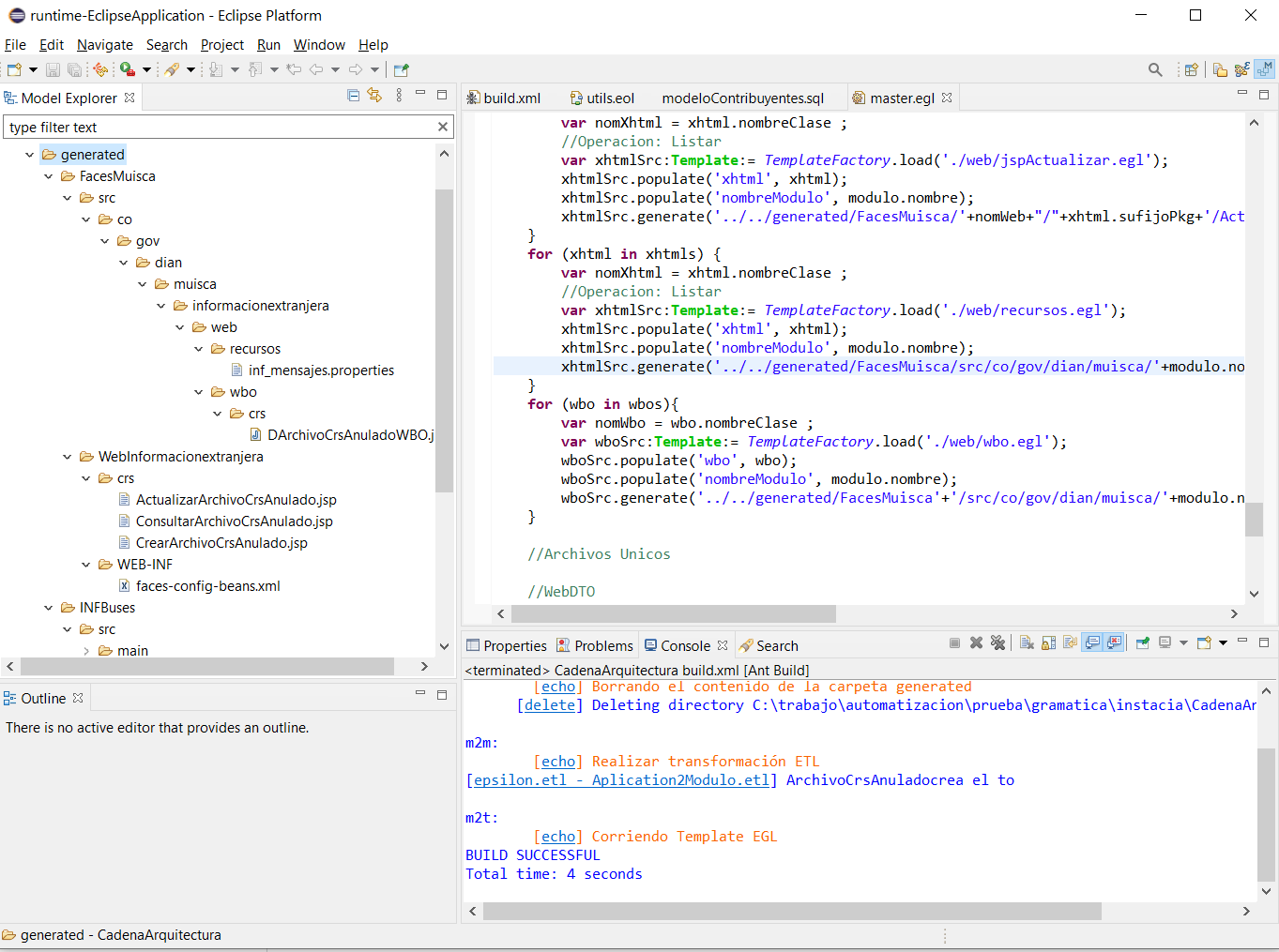


Ilustración 10 Visualización del código generado

En resumen, para la generación de código se requiere:

1. Abrir los proyectos de la primera instancia
2. Ejecutar el plugin de xtext (co.gov.dian.xtext.arquitecturadian.dsl.sql) en la segunda instancia y abrir en ella el proyecto CadenaArquitectura.
3. Crear el script de modeloContribuyentes.sql.
4. Ejecutar la opción Generar Modelo desde archivo Sql
5. Ejecutar la tarea de Ant.