PROYECTO DATA WAREHOUSE

DATA WAREHOUSE

ANDRES SANCHEZ

LUIS FELIPE VELASCO TAO
JUAN DAVID GONZALEZ

UNIVERSIDAD DE SAN BUENAVENTURA

BOGOTÁ

11 DE AGOSTO

2021

Contenido

Bona Health EPS 5
Contexto de la Bona Health EPS 5
Misión 5
Visión 5
KPI 5
Requerimientos de negocio 9
Fuentes elegidas 10
Mortalidad VIH 2010 a 2016 10
Mortalidad cáncer de pulmón 2010 a 2016 11
Mortalidad cáncer de mama 2010 a 2016 12
Departamentos y municipios de Colombia 20
Modelo Entidad Relación 22
DISEÑO CONCEPTUAL 24
Diseño de Dimensiones y Medidas 24
Modelo CMDM: relaciones dimensionales 25
Modelo CMDM - Defunciones por fecha 25
Modelo CMDM – Asistencia médica 25
Modelo CMDM – Defunciones por municipio 26
Modelo CMDM – Muertes 27
Modelo Multidimensional 27
Modelos DF 28
Modelo DF – Defunciones por fecha 28
Modelo DF – Asistencia médica 29
Modelo DF – Defunciones por municipio 29
Modelo DF – Muertes 30
DISEÑO LOGICO 31
Modelo ROLAP 31

Bona Health EPS
Cuidado De Ti

DDL del Data Ware House 32
ETL 35
Diseño de ETL 35
Secuencia de ETL 37
Modelo relacional ODS 39
Programa ETL 45
Muestra de datos del Data Warehouse 56
Fact_asistencia_medica 57
Fact_defunciones_fecha 57
Fact_defunciones_municipio 58
Fact_muertes 58
CREACIÓN Y PUBLICACIÓN DEL CUBO 59
Entorno de trabajo 59
Pentaho Server 59
Pentaho Schema Workbench 63
Pentaho Report Desginer 64
Creación del cubo 66
Dim_fecha 69
Dim_Municipio 73
Dim_Enfermedad 76
Ejemplo de creacion de una medida 78
Asistencia_medica 80
Defun_por_municipio 80
Defunciones_fecha 81
Prueba 82
Publicación del cubo 84
Dashboard del cubo en Pentaho Server 86
Conexión al Data Warehouse 86

Bona Health EPS
Cuidando De Ti

Configuración del Schema 90
Despliegue del Dashboard 91
Creación de reportes 93
Reporte 1 93
Reporte 2 106
Reporte 3 112
Tabla de ilustraciones 113

Bona Health EPS
Cuidando De Ti

BONA HEALTH EPS

A continuación, se van a definir los elementos corporativos de la empresa y se definirán las fuentes de datos elegidas para el desarrollo de la actividad.

Contexto de la Bona Health EPS

Bona Health EPS es una entidad promotora de salud colombiana la cual tienen presencia a nivel nacional, principalmente en las ciudades capitales, en las cueles nuestros afiliados acceden a los servicios de salud prestados por medio de las IPS asociadas, entre las cuales se tienen hospitales, centro de optometría, odontología, radiografía y de atención para distintas afecciones medicas de los colombianos. Nuestra empresa nació en el 2015, teniendo como punto de partida la ciudad de Bogotá, pasando año tras año a tener presencia en Cali, Medellín, Cartagena, Bucaramanga y demás ciudades principales del país. En la actualidad, debido a la pandemia, Bona Health EPS ha tenido inconvenientes con los procesos de afiliación y especialmente con el manejo de citas y tratamientos de nuestros afiliados con enfermedades como VIH, cáncer y demás problemas de salud, llegando hasta el limite de notar cierto aumento en la mortalidad de muchos afiliados con ciertas enfermedades.

Bona Health EPS
Cuidando De Ti

Misión

Asegurarles a los colombianos desde el primer día que se encuentren afiliados el acceso a servicios de salud de calidad, a tiempo y siempre buscando su bienestar.

Visión

Para el 2030, Bona Health EPS espera ser una EPS que tenga una cobertura del 80% en municipios apartados en el país con el fin de acercar a más a los pueblos y a sus habitantes servicios de salud de calidad, sea medio del régimen contributivo o subsidiado.

KPI

Con el fin de proveer un servicio que sea capaz de atender a su debido tiempo las necesidades de los colombianos, es necesario determinar la tasa de mortalidad de determinados padecimientos en un intervalo de tiempo, y la frecuencia de defunciones en un lugar con respecto a dicho padecimiento.

Es por esto por lo que se establecieron los siguientes indicadores para determinar el tipo de servicio que debe ofrecerse en los departamentos más afectados por un determinado padecimiento.

1. **Tasa de defunciones por VIH, cáncer de mama y cáncer de pulmón**. A continuación, se muestran los datos de las defunciones presentadas en el periodo del 2010 a 2016 con el fin de definir medidas para la prevención y atención oportuna de dichas enfermedades.

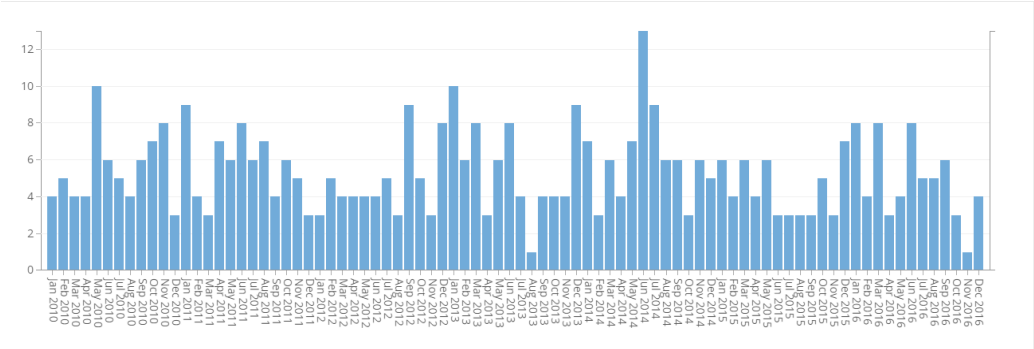


Ilustración 1 Defunciones por VIH

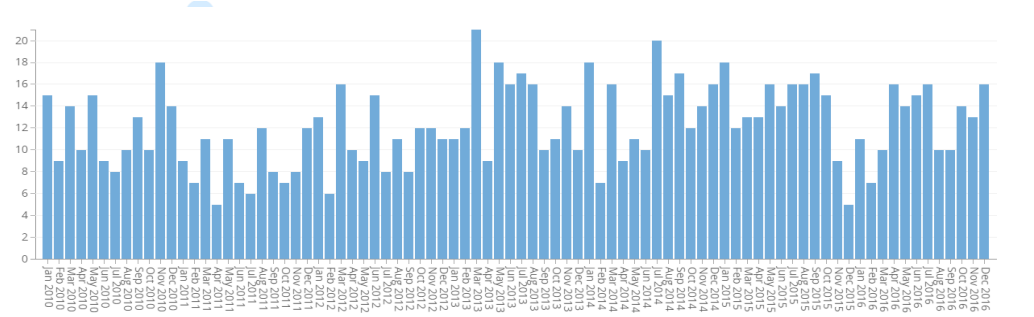


Ilustración 2 Defunciones por cáncer de mama

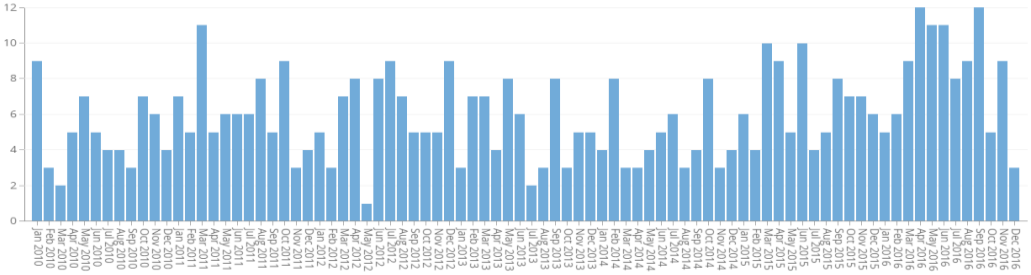
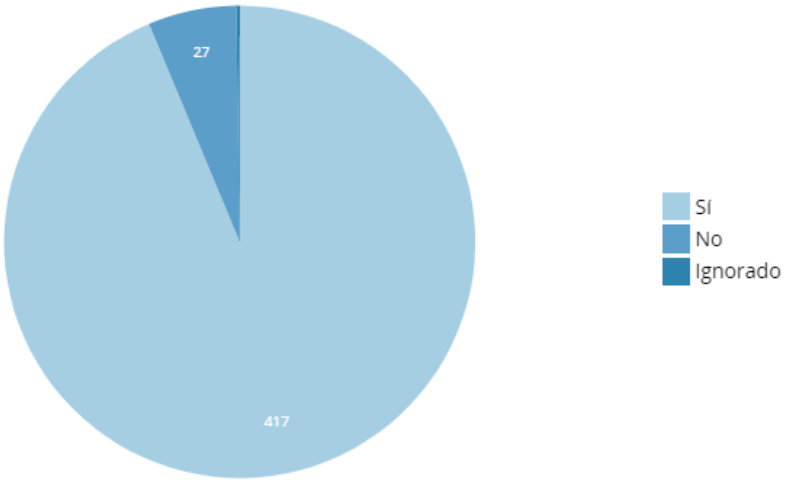


Ilustración 3 Defunciones por cáncer de pulmón

2. **Tasa de atención médica:** en las defunciones registradas. Se quiere definir si los fallecidos tuvieron o no atención medica en el momento, de modo tal se pueda definir acciones que mejoren la atención medica en futuros casos.



Bona Health EPS

Ilustración 4 Asistencia medida prestada a los fallecidos por VIH

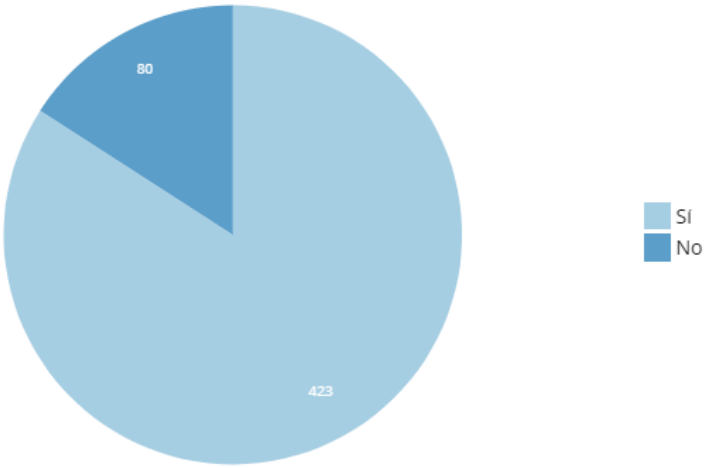


Ilustración 5 Atención medica prestada a los fallecidos por cáncer de mama

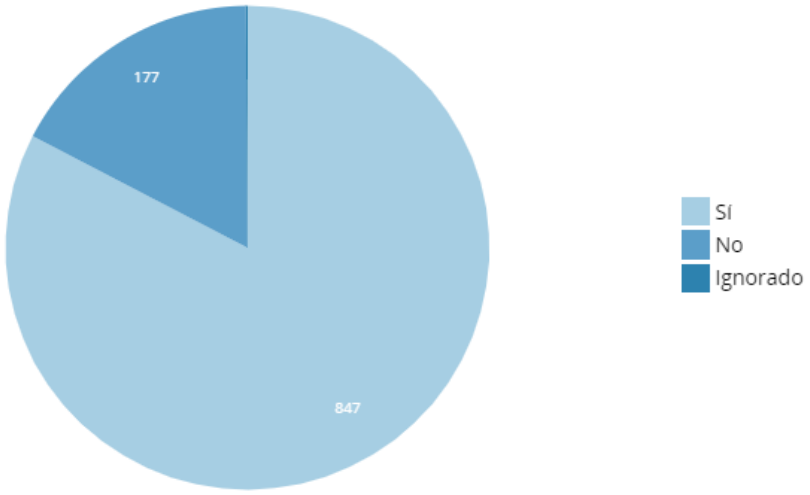


Ilustración 6 Atención medica prestada a los fallecidos por cáncer de pulmón

3. **Defunciones por municipio**: con el fin definir medidas y mecanismos de acción, se quiere medir cuantas personas fallecieron en cada uno de los municipios registrados.

Bona Health EPS

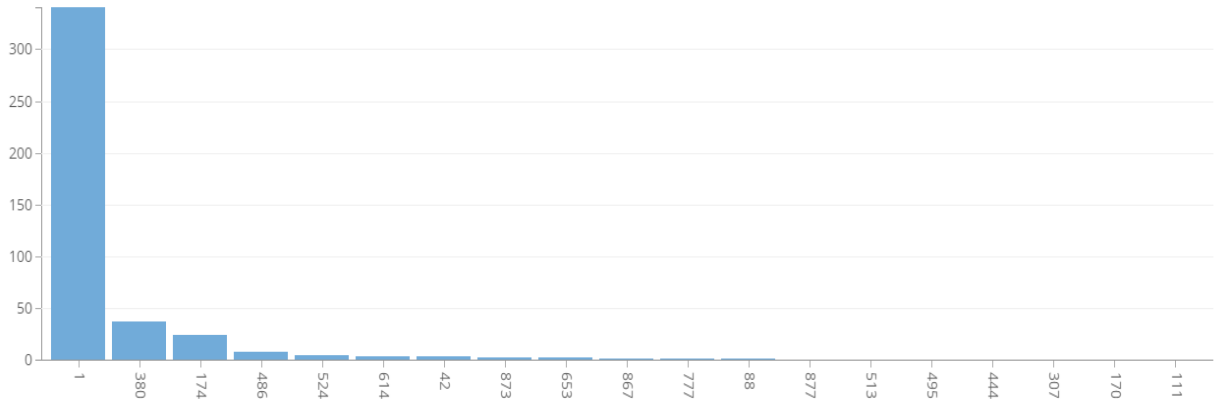


Ilustración 7 defunciones por VIH clasificadas por municipio

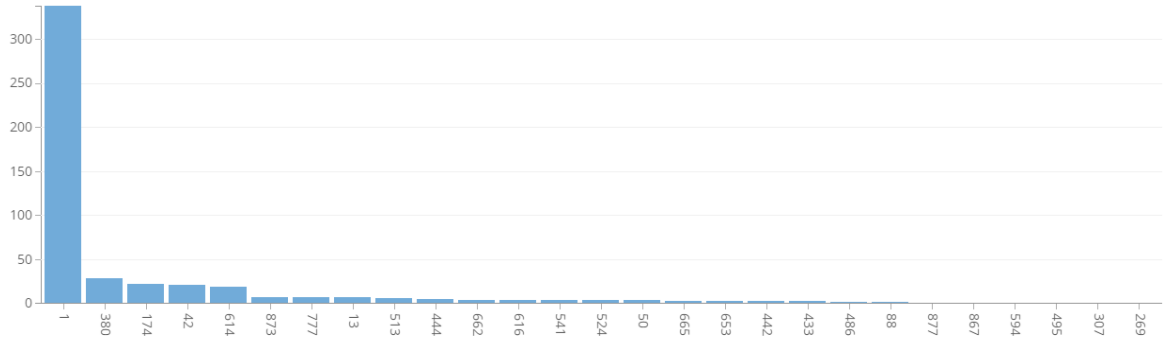


Ilustración 9 defunciones por cáncer de mama clasificadas por municipio

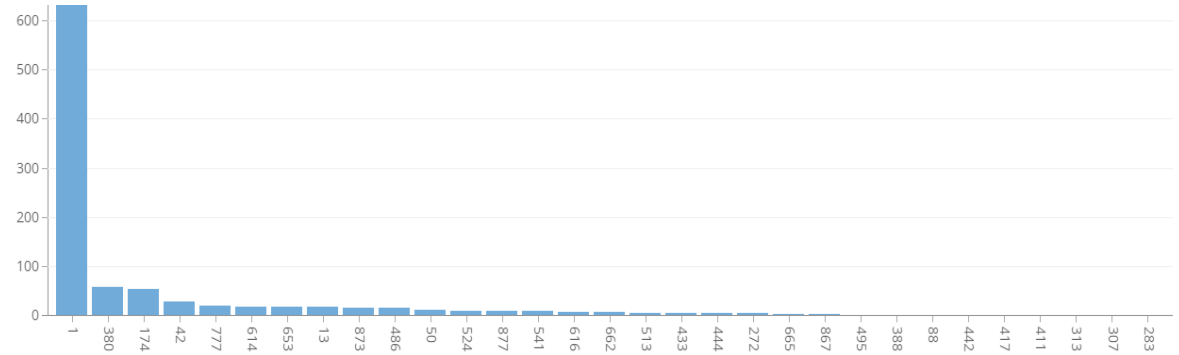


Ilustración 8 defunciones por cáncer de pulmón clasificadas por municipio

Requerimientos de negocio

Bona Health EPS busca ofrecer un servicio accesible que cubra las zonas con mayor tasa de mortalidad por padecimientos clínicos en el país, tales como el VIH, el cáncer de Mama y el cáncer de pulmón. Con el fin de realizar estos objetivos, se plantean los siguientes requerimientos.

1. Analizar la tasa de mortalidad por padecimiento clínico con respecto a los departamentos del país para proveer cobertura en los municipios en dónde se requieran los servicios de la entidad.

2. Establecer un histórico de defunciones por padecimientos clínicos en cada departamento para la asignación del personal adecuado en las instalaciones que tengan cobertura de la entidad.

3. Determinar el número de difuntos que recibieron atención médica en una institución clínica con el fin de poder asignar los recursos correspondientes a las instituciones cubiertas por la entidad.

Fuentes elegidas

Para la correcta toma de decisiones con relación a la mejora en la prestación de servicios de salud que prevengan la mortalidad de los afiliados con enfermedades criticas como el VIH, el cáncer de mama y de pulmón se seleccionaron las siguientes fuentes de datos, las cuales son los reportes emitidos de defunción relacionados con los afiliados con estas enfermedades, además de requerir de cierta información para definir apropiadamente el lugar (departamento y municipio) donde falleció la persona:

Mortalidad VIH 2010 a 2016

Este dataset es suministrado por Observatorio Social de Salud Pública en el cual se entregan datos relacionados con las defunciones de las personas con VIH en el periodo del 2010 a 2016, en el cual se pueden ver los datos relacionados con el lugar en donde falleció la persona, como lo es municipio, departamento institución en la que falleció, también se presentan datos básicos del fallecido como su sexo y condición socioeconómica y finalmente se define una descripción de la condición médica de la persona al fallecer, incluyendo sus antecedentes de salud y datos relacionados con el estado de salud de la persona.

Fecha de creación: 21 de abril de 2017

Sector: Salud y protección social

Cantidad de registros:445

Cantidad de columnas: 105

COD_...	COD_...	A_DE...	COD_...	SIT_D...	OTRS...	COD_...	NOM...	TIPO...	FECH...	ANO	MES	HOF
17	1	Cabecera...		Hospital/c...		170.010.0...	ESE HOSP...	2	2012 Sep ...	2.012		
17	1	Cabecera...		Hospital/c...		170.010.1...	IPS CAPR...	2	2012 Mar...	2.012	3	
17	1	Cabecera...		Hospital/c...		170.010.1...	INSTITUT...	2	2011 Oct ...	2.011	10	
17	42	Cabecera...		Casa/do...		170.420.0...	ESE HOSP...	2	2012 Nov ...	2.012	11	
17	1	Cabecera...		Hospital/c...		170.010.1...	INSTITUT...	2	2012 Jan ...	2.012	1	
17	1	Cabecera...		Hospital/c...		170.010.0...	ESE HOSP...	2	2013 Dec ...	2.013	12	
76	1	Cabecera...		Hospital/c...		760.010.4...	CLINICA R...	2	2013 Feb ...	2.013	2	
17	42	Cabecera...		Hospital/c...		170.420.0...	ESE HOSP...	2	2013 Dec ...	2.013	12	
17	1	Cabecera...		Hospital/c...		170.010.1...	INSTITUT...	2	2013 Nov ...	2.013	11	
17	42	Cabecera...		Hospital/c...		170.420.0...	ESE HOSP...	2	2013 Oct ...	2.013	10	
17	42	Cabecera...		Hospital/c...		170.420.0...	ESE HOSP...	2	2013 Sep ...	2.013	9	
63	1	Cabecera...		Hospital/c...		630.010.0...	EMPRESA...	2	2013 May...	2.013	5	
17	1	Cabecera...		Hospital/c...		170.010.0...	CAJA DE ...	2	2014 Apr ...	2.014	4	

Ilustración 10 Muestra de datos del dataset de mortalidad por VIH

Fuente: Mortalidad VIH 2010 A 2016 | Datos Abiertos Colombia

Mortalidad cáncer de pulmón 2010 a 2016

Este dataset es suministrado por Observatorio Social de Salud Pública en el cual se entregan datos relacionados con las defunciones de las personas con cáncer de pulmón en el periodo del 2010 a 2016, en el cual se pueden ver los datos relacionados con el lugar en donde falleció la persona, como lo es municipio, departamento institución en la que falleció, también se presentan datos básicos del fallecido como su sexo y condición socioeconómica y finalmente se define una descripción de la condición médica de la persona al fallecer, incluyendo sus antecedentes de salud y datos relacionados con el estado de salud de la persona.

Fecha de creación: 20 de abril de 2017

Sector: Salud y protección social

Cantidad de registros:1025

Cantidad de columnas: 105

COD_...	COD_...	A_DE...	COD_...	SIT_D...	OTRS...	COD_...	NOM...	TIPO...	FECH...	ANO	MES
17	13	Cabecera...		Casa/do...				2	2010 Jan ...	2.010	
17	13	Cabecera...		Hospital/c...		170.130.0...	ESE HOSP...	2	2010 Sep ...	2.010	
17	13	Rural disp...		Hospital/c...		170.130.0...	ESE HOSP...	2	2010 Nov ...	2.010	
17	1	Cabecera...		Hospital/c...		170.010.0...	ESE HOSP...	2	2010 Jul 1...	2.010	
17	13	Centro po...	1	Centro/pu...		170.130.0...	ESE HOSP...	2	2011 May...	2.011	
17	13	Rural disp...		Casa/do...		170.130.0...	ESE HOSP...	2	2011 Mar...	2.011	
17	13	Centro po...	16	Casa/do...		170.130.0...	ESE HOSP...	2	2011 Mar...	2.011	
17	13	Centro po...	1	Casa/do...		170.130.0...	ESE HOSP...	2	2011 Mar...	2.011	
17	1	Cabecera...		Hospital/c...		170.010.1...	INSTITUT...	2	2011 Jan ...	2.011	
17	13	Cabecera...		Casa/do...		170.130.0...	ESE HOSP...	2	2012 Dec ...	2.012	
5	1	Cabecera...		Hospital/c...		50.010.21...	E.S.E. HO...	2	2012 Sep ...	2.012	
76	1	Cabecera...		Hospital/c...		760.010.0...	IPS COMF...	2	2012 May...	2.012	
17	13	Cabecera...		Casa/do...		170.130.0...	ESE HOSP...	2	2012 Jan ...	2.012	

Ilustración 11 Muestra de datos del dataset de mortalidad por cáncer del pulmón

Fuente: Mortalidad Cáncer de Pulmón 2010 A 2016 | Datos Abiertos Colombia

Bona Health EPS
Cuidando De Ti

Mortalidad cáncer de mama 2010 a 2016

Este dataset es suministrado por Observatorio Social de Salud Pública en el cual se entregan datos relacionados con las defunciones de las personas con cáncer de mama en el periodo del 2010 a 2016, en el cual se pueden ver los datos relacionados con el lugar en donde falleció la persona, como lo es municipio, departamento institución en la que falleció, también se presentan datos básicos del fallecido como su sexo y condición socioeconómica y finalmente se define una descripción de la condición médica de la persona al fallecer, incluyendo sus antecedentes de salud y datos relacionados con el estado de salud de la persona.

Fecha de creación: 20 de abril de 2017

Sector: Salud y protección social

Cantidad de registros:503

Cantidad de columnas: 105

Vista previa de la tabla

Explorar los datos | Crear visualización

COD_...	COD_...	A_DE...	COD_...	SIT_D...	OTRS...	COD_...	NOM...	TIPO...	FECH...	ANO	MES
17	1	Cabecera...		Hospital/c...		170.010.0...	SERVICIO...	2	2010 Apr ...	2.010	
17	13	Rural disp...		Casa/do...		170.130.0...	ESE HOSP...	2	2010 Oct ...	2.010	
17	1	Cabecera...		Hospital/c...		170.010.1...	INSTITUT...	2	2010 Nov ...	2.010	
17	1	Cabecera...		Hospital/c...		170.010.0...	SERVICIO...	2	2010 Apr ...	2.010	
17	1	Cabecera...		Hospital/c...		170.010.1...	INSTITUT...	2	2011 Oct ...	2.011	
17	1	Cabecera...		Hospital/c...		170.010.0...	ESE HOSP...	2	2011 Mar...	2.011	
17	13	Cabecera...		Casa/do...		170.130.0...	ESE HOSP...	2	2011 Mar...	2.011	
17	13	Cabecera...		Hospital/c...		170.130.0...	ESE HOSP...	2	2012 Jul 0...	2.012	
66	1	Cabecera...		Hospital/c...		660.010.1...	CORPORA...	2	2013 Sep ...	2.013	
17	13	Centro po...	16	Casa/do...		170.130.0...	ESE HOSP...	2	2014 Sep ...	2.014	
17	1	Cabecera...		Hospital/c...		170.010.0...	ESE HOSP...	2	2014 Apr ...	2.014	
17	13	Centro po...	10	Casa/do...		170.130.0...	ESE HOSP...	2	2015 Dec ...	2.015	
17	13	Cabecera...		Casa/do...		170.130.0...	ESE HOSP...	2	2015 Jul 1...	2.015	

Ilustración 12 Muestra de datos del dataset de mortalidad por cáncer de mama

Bona Health EPS
Cuidando De Ti

Fuente: Mortalidad Cáncer Mama 2010 A 2016 | Datos Abiertos Colombia

Debido a que los datasets se encuentran estructurados bajo las mismas columnas, se realizó un análisis de las columnas de los datasets y se brinda una descripción de los siguientes datos relevantes.

Nombre de Columna	Descripción	Tipo
COD_DPTO	Identificador del departamento donde falleció la persona	Número
COD_MUNIC	Identificador del municipio donde falleció la persona. Este valor se puede repetir, ya	Número

	que, dos o más municipios pueden tener el mismo código, la diferencia es el código del departamento	
A_DEFUN	Área del municipio donde falleció la persona (rural, cabecera municipal, etc.)	Texto simple
SIT_DEFUN	Sitio en donde la persona falleció, como su casa o un hospital/clínica	Texto simple
COD_INST	Identificador de la institución prestadora de servicios de salud donde falleció la persona. (Si la persona fallece en su casa, este campo se encuentra vacío)	Número
NOM_INST	Nombre de la institución prestadora de servicios de salud en donde la persona halla fallecido.	Texto simple
FECHA_DEF	Fecha y hora en la cual falleció la persona,	Fecha y hora

	expresada de la siguiente forma: 2012 Sep 14 12:00:00 AM	
ANO	Año en el que falleció la persona	Número
MES	Mes en el que falleció la persona	Número
HORA	Hora en la que falleció la persona	Número
MINUTOS	Minuto en que falleció la persona	Número
SEXO	Cadena de caracteres con el sexo de la persona (Masculino/Femenino)	Texto simple
FECHA_NAC	Fecha (y hora la cual no es un dato relevante) de nacimiento del fallecido	Fecha y hora
EST_CIVIL	Estado civil del fallecido	Texto simple
EDAD	Edad del fallecido	Número

NIVEL_EDU	Nivel educativo del fallecido	Texto simple
MUERTEPORO	Campo de texto en donde se define si la persona murió o no por omisión	Texto simple
OCUPACION	Ocupación o profesión del fallecido.	Texto simple
CODPTORE	Nombre del departamento donde residía el fallecido	Texto simple
CODMUNRE	Nombre del municipio donde residía el fallecido	Texto simple
AREA_RES	Área del municipio donde residía la persona (rural, cabecera municipal, etc.)	Texto simple
BARRIOFAL	Nombre del barrio donde residía el fallecido	Texto simple
SEG_SOCIAL	Tipo de seguridad social por medio de la cual se	Texto simple

	le presto los servicios de salud al fallecido.	
IDADMISALU	Identificador de la EPS (este dato se obvia para la naturaleza del ejercicio)	Número
IDCLASADMI	EPS a la cual el fallecido estaba afiliado (este dato se obvia para la naturaleza del ejercicio)	Texto simple
PMAN_MUER	Tipo de muerte	Texto simple
MU_PARTO	Campo en donde se define si la muerte fue en un parto o no	Texto simple
T_PARTO	Si la muerte fue por un parto, se define el tipo de parto. Se definen otros datos relacionados con muertes en partos	Texto simple
ASIS_MED	Campo en donde se define si la persona tuvo asistencia médica en el momento de su muerte	Texto simple

Los siguientes datos son descripción de la condición médica del fallecido		
N_DIR1 **C_DIR1** **C_DIR12** **N_ANT1** **C_ANT1** **C_ANT12** **N_ANT2** **C_ANT2** **C_ANT22** **N_ANT3** **C_ANT3** **C_ANT32** **N_PAT1** **C_PAT1** **N_PAT2** **C_PAT2**	Datos sobre antecedentes, signos vitales y condición médica del fallecido	Texto simple
N_BAS1	Enfermedad de base del fallecido	Texto simple

TOTAL, DE REGISTROS A MANIPULAR: 1975

Con base al análisis a cada una de las fuentes de datos, la descripción brindada a las columnas de interés y la definición de los requerimientos y KPIS, se definirá un único modelo entidad relación en el cual se puedan estructurar los datos vistos.

Bona Health EPS
Cuidando De Ti

Departamentos y municipios de Colombia

Este dataset es suministrado por el Departamento Administrativo Nacional de Estadística, por sus siglas, DANE de entregan los datos de codificación o identificadores de los municipios y departamentos del país, los cuales son empleados por las entidades publicas y privadas en sus procesos de registro de actividades departamentales.

Fecha de creación: 11 de mayo de 2017

Sector: estadísticas

Cantidad de registros:1123

Cantidad de columnas: 5

REGION	CÓDIGO DANE DEL DE...	DEPARTAMENTO	CÓDIGO DANE DEL MU..	MUNICIPIO
Region Eje Cafetero - Antioq...	5	Antioquia	5.001	Medellin
Region Eje Cafetero - Antioq...	5	Antioquia	5.002	Abejorral
Region Eje Cafetero - Antioq...	5	Antioquia	5.004	Abriaqui
Region Eje Cafetero - Antioq...	5	Antioquia	5.021	Alejandria
Region Eje Cafetero - Antioq...	5	Antioquia	5.030	Amaga
Region Eje Cafetero - Antioq...	5	Antioquia	5.031	Amalfi
Region Eje Cafetero - Antioq...	5	Antioquia	5.034	Andes
Region Eje Cafetero - Antioq...	5	Antioquia	5.036	Angelopolis
Region Eje Cafetero - Antioq...	5	Antioquia	5.038	Angostura
Region Eje Cafetero - Antioq...	5	Antioquia	5.040	Anori
Region Eje Cafetero - Antioq...	5	Antioquia	5.044	Anza
Region Eje Cafetero - Antioq...	5	Antioquia	5.045	Apartado
Region Eje Cafetero - Antioq...	5	Antioquia	5.051	Arboletes

Ilustración 13 Muestra de datos del dataset de departamentos y municipios de Colombia

Fuente: Departamentos y municipios de Colombia

A continuación, se mostrara una tabla con la identificación de los datos presentados en este dataset:

Nombre de Columna	Descripción	Tipo

REGIÓN	Nombre de la región a la que pertenece cada municipio y departamento	Texto simple
CÓDIGO DANE DEL DEPARTAMENTO	Código identificador para cada uno de los departamentos de Colombia	Número
DEPARTAMENTO	Nombre del departamento asociado al código anterior	Texto simple
CÓDIGO DANE DEL MUNICIPIO	Código identificador para cada uno de los municipios de Colombia, conformado por el código del departamento al que pertenece y los últimos tres dígitos hacen referencia al identificador del municipio dentro del departamento	Numero
MUNICIPIO	Nombre del municipio asociado al código anterior	Texto simple

Modelo Entidad Relación

Debido a como las tres fuentes de datos se encuentran estructuradas bajo las mismas columnas, se obtuvo un único modelo entidad relación en el cual se definieron las siguientes tablas:

- **Lugar**: se definen los datos del lugar donde la persona falleció, y se estructura por medio de los siguientes datos:
    - Id_lugar: dato identificador del lugar*
    - Tipo_lugar: se define si la persona murió o no en un hospital o clínica.
    - Departamento: este dato se obtendrá de la tabla departamento, y se definiría el departamento en el cual falleció la persona.
    - Municipio: este dato se obtendrá de la tabla municipio, y se definiría el municipio en el cual falleció la persona.
    - Institución: con base al tipo lugar, se define la institución obtenida de la tabla del mismo nombre.
- **Fecha**: se definen los datos del la fecha y hora registrada del fallecimiento de la persona, esto por medio de los siguientes datos:
    - Id_fecha: fecha y hora formateada del fallecimiento de la persona.
    - Año: valor numérico del año del fallecimiento de la persona.
    - Mes: valor numérico del mes del fallecimiento de la persona.
    - Día: valor número del día del fallecimiento de la persona.
    - Hora: valor numérico de la hora de fallecimiento de la persona
    - Minutos: valor numérico del minuto de fallecimiento de la persona
- **Persona**: en esta tabla se definen los datos básicos de la persona, de modo tal se definen los siguientes datos:
    - Id_persona: identificador de la persona.
    - Fecha_nacimiento: fecha de nacimiento del fallecido
    - Edad: valor numero de la edad del fallecido
    - Sexo: cadena de caracteres con el sexo de la persona.

Bona Health EPS
Cuidando De Ti

- Seguridad_social: cadena de caracteres en donde se define el régimen por medio del cual se le prestó atención a la persona (contributivo, subsidiado)

- **Enfermedad**: En esta tabla solo se definirá el nombre de la enfermedad de base que produjo el fallecimiento de la persona, por lo que se define los siguientes datos:
  - Nombre_enfermedad: nombre de la enfermedad de base que genero el fallecimiento de la persona.
  - Observación: observación brindada por los médicos al momento del fallecimiento de la persona.

A continuación, se muestra el modelo resultante:

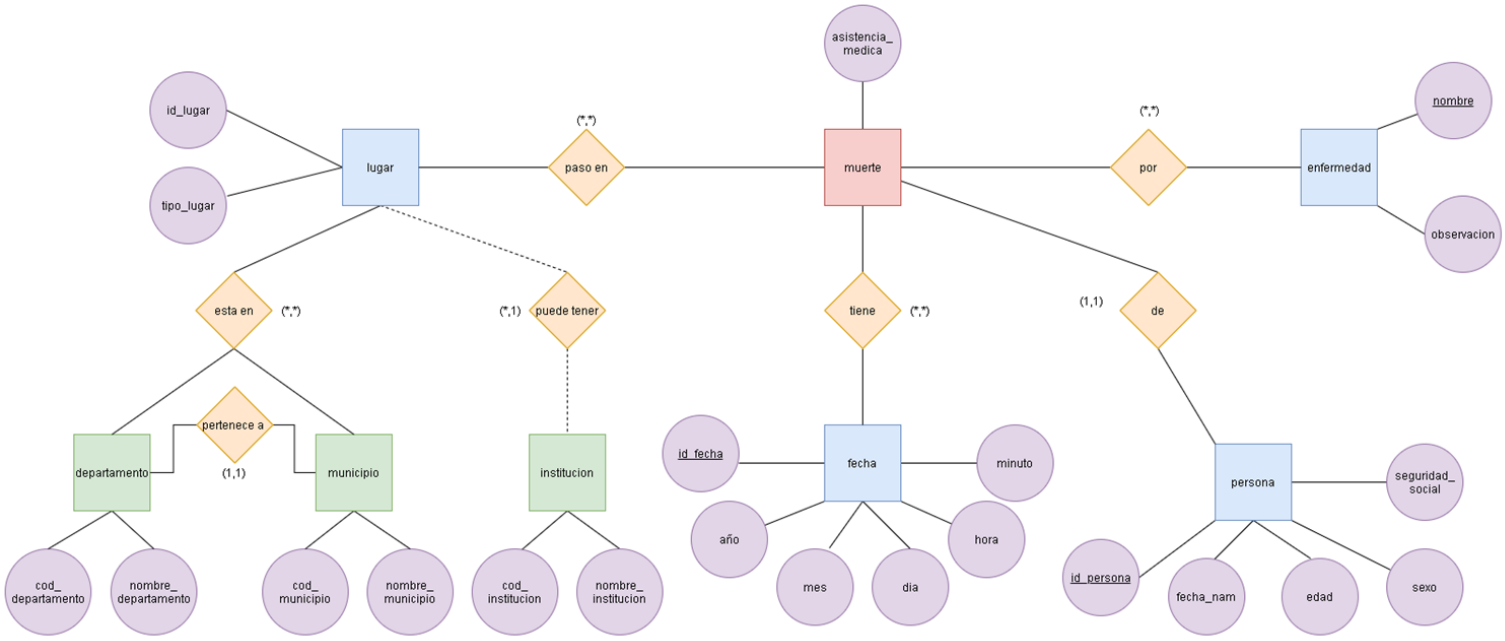


Ilustración 14 Modelo entidad relación de las fuentes de datos

DISEÑO CONCEPTUAL

En esta fase se definen los elementos de dimensiones, hechos, medidas y la forma en que estas se relacionaran en el Data Warehouse, todo esto contemplando los requerimientos del negocio y las fuentes de datos a emplear en el desarrollo e implementación del almacén de datos. Para esta fase se contemplará el desarrollo de los diagramas de dimensiones y medidas, de relaciones y los modelos multidimensionales y de dimensiones y hechos.

Diseño de Dimensiones y Medidas

Aquí se definen las dimensiones por medio de las cuales se obtendrá la información requerida por el negocio, en las cuales se definen los niveles y cierta jerarquía por medio de la cual se podrá ordenar los elementos de cada dimensión. Además, se deben definir las medidas, los datos de especial interés por parte de la empresa, por medio de los cuales se definirán más adelante las tablas de hechos.

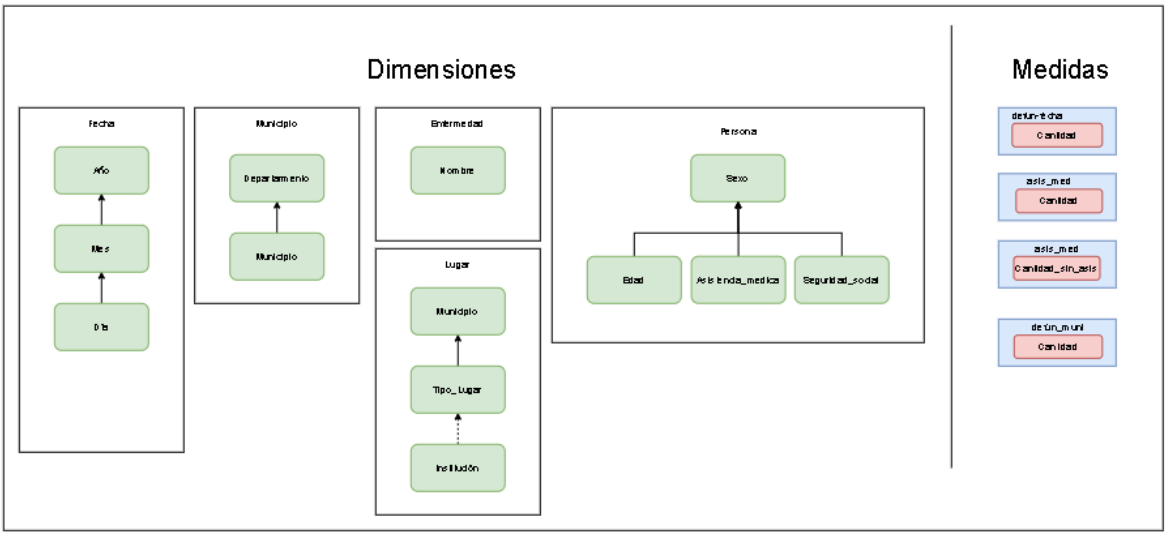


Ilustración 15 Diseño de dimensiones y medidas

Bajo los requerimientos del negocio, en la imagen anterior, se definieron las siguientes dimensiones compuestas por sus respectivas jerarquías y miembros:

- **Fecha**: Se tienen los datos para conocer la fecha de defunción de una persona, necesaria para hacer los análisis dispuestos por los KPI
- **Municipio**: con el fin de saber el municipio donde falleció cada persona se define esta dimensión con los datos de departamento y municipio, a su vez. Esta dimensión será empleada por la dimensión de lugar.
- **Lugar**: esta dimensión determina el lugar con municipio, tipo de lugar y institución en donde falleció una persona, este último dato dependerá de si la persona pudo ser o no atendida al momento de su fallecimiento.

- **Enfermedad**: Es la causa de muerte de los difuntos que contempla el sistema. Cada enfermedad tiene un **nombre** que tendrá un identificador para ser reconocida.
- **Persona**: Son los datos pertinentes al difunto, entre los que se incluyen el **sexo, la edad, su seguridad social** y si recibió **asistencia médica**.

Modelo CMDM: relaciones dimensionales

Con base a las dimensiones y medidas previamente especificadas, se definieron los siguientes cubos por medio de las relaciones multidimensionales que le den forma a cada uno de los hechos. A continuación, se especificarán cada uno de los modelos.

Modelo CMDM - Defunciones por fecha

Para conocer los datos con respecto a las defunciones generadas por cada enfermedad, se definió el siguiente vivo el cual usa las dimensiones de enfermedad y fecha, con el fin de saber cuántos fallecimientos se ocasionaron por cada enfermedad en cada fecha.

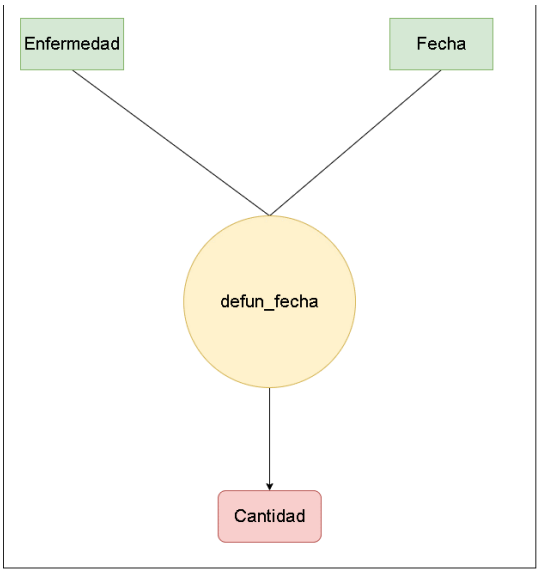


Ilustración 16 Modelo CMDM – Defunciones por fecha

Modelo CMDM – Asistencia médica

Con el fin de conocer la asistencia medica brindada en cada uno de los fallecimientos registrados se tomará en cuenta las dimensiones de lugar/municipio, fecha y enfermedad, de modo tal se pueda conocer cuántos de los fallecimientos

registrados de cada enfermedad en cada fecha recibieron asistencia medica y cuantos no la pudieron recibir. Esto con el fin de saber si los servicios de salud de cada lugar responder de forma óptima.

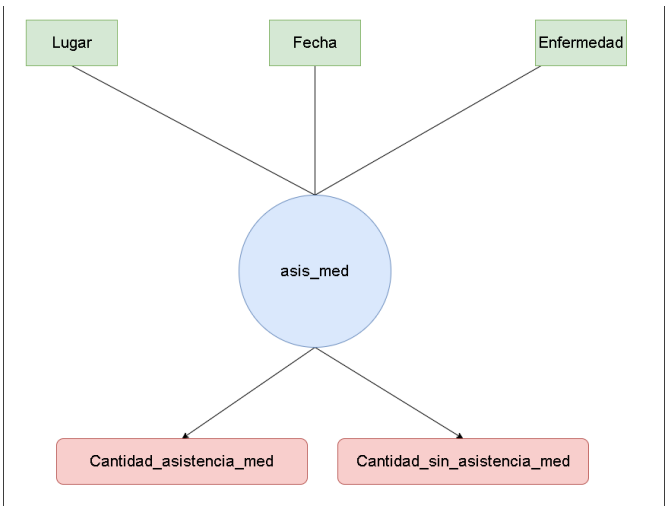


Ilustración 17 Modelo CMDM – Asistencia médica

Modelo CMDM – Defunciones por municipio

Para tener presente los municipios con más o menos casos de defunciones de las enfermedades a trabajar en el Data Warehouse, se definen las dimensiones de enfermedad, lugar/municipio y fecha para poder conocer la cantidad de muertes registradas en cada municipio, fecha y por cada enfermedad

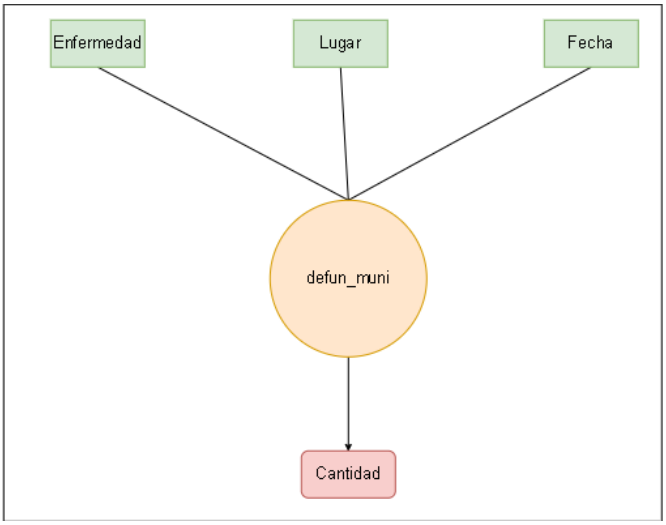


Ilustración 18 Modelo CMDM – Defunciones por municipio

Modelo CMDM – Muertes

En este cubo se contemplarán las dimensiones de persona, lugar, fecha y enfermedad, esto con el fin de permitir el posterior análisis de los datos de la forma que se requiera. Aquí no se contará con ninguna medida.

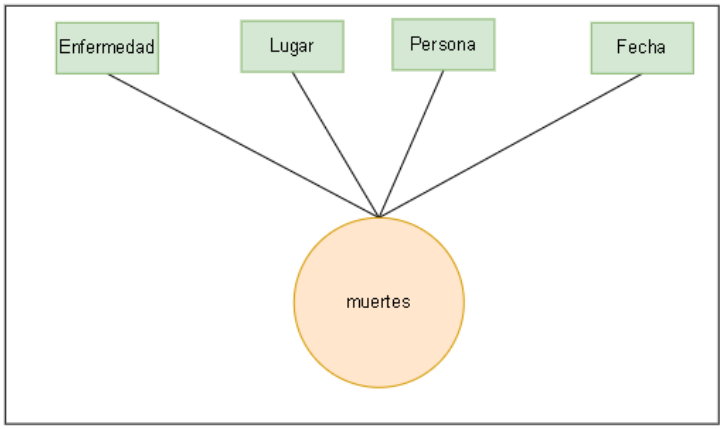


Ilustración 19 Modelo CMDM – Muertes

Modelo Multidimensional

Al tener presente cada una de las definiciones de las dimensiones involucradas en cada uno de los cubos, se realizo el siguiente modelo multidimensional el cual permite ver las dimensiones y hechos definidos.

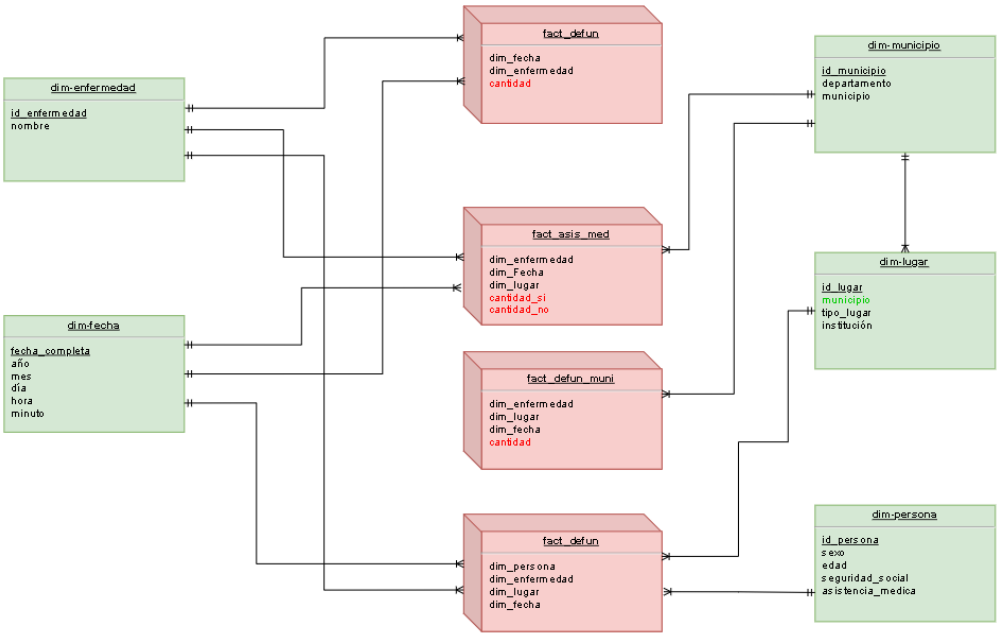


Ilustración 7 Modelo Multidimensional

En la anterior imagen se puede ver las relaciones entre los hechos, los cuales se encuentran en el siguiente orden:

- **Fact_defuncion**: se define la relación entre las dimensiones de fecha y enfermedad, y se deja como medida la cantidad de muertes registradas de cada enfermedad en cada fecha.
- **Fact_asis_medica**: en este hecho se ve la relación entre las dimensiones de fecha, municipio y enfermedad, y teniendo como medidas la cantidad de fallecimientos que si recibieron asistencia médica y aquellos que lastimosamente no pudieron recibir asistencia médica.
- **Fact_defunciones_municipio**: en este hecho se vuelven a emplear las dimensiones de fecha, municipio y enfermedad, enfocándose en generar como medida la cantidad de defunciones registradas en cada municipio y fecha de cada una de las tres enfermedades a analizar.
- **Fact_muerte**: en este hecho se verán todos los registros de fallecimientos, pero enfocados en las dimensiones de lugar, fecha, enfermedad y persona, esto en el caso que se requiera analizar particularidades presentadas con los fallecimientos y las personas.

Modelos DF

En estos modelos se busca definir las jerarquías presentes en cada una de las dimensiones, de modo tal se conozca como se ira avanzando por los miembros de cada una de estas, dando un mapa de guía para las futuras operaciones que se van a realizar con el Data Warehouse, además de definir las tablas de hechos y las medidas que harán parte de cada una de estas. A continuación, se definirán los modelos resultantes en base a cada uno de los cubos visualizando la jerarquía de cada una de las dimensiones que los componen.

Modelo DF – Defunciones por fecha

En este modelo se puede ver la jerarquía de las dimensiones involucradas en el hecho fact_defun, de modo tal se pueda conocer la forma en que podemos realizar las consultas sobre el cubo y sus dimensiones. Este hecho involucra las dimensiones de fecha, en donde se ve como esta tiene en su nivel más bajo el día y en el más alto el año del fallecimiento, por otro lado, se contempla la dimensión de enfermedad que solo se compone por el nombre de la enfermedad. No se debe

olvidar la medida de este hecho que es la cantidad de fallecimientos registrados asociados a cada enfermedad y fecha.

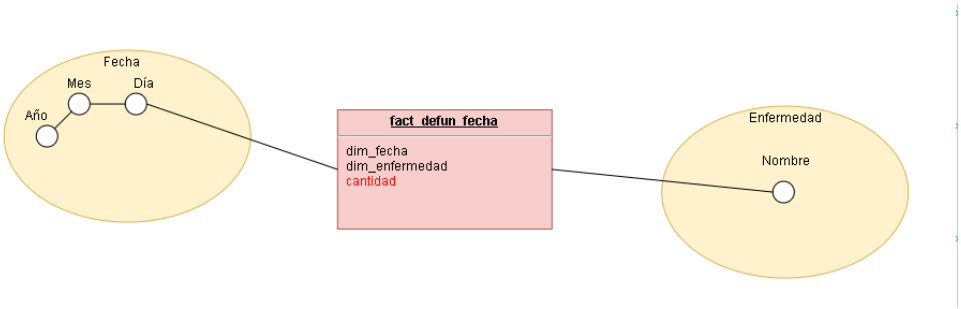


Ilustración 8 Modelo DF – defunciones_fecha

Modelo DF – Asistencia médica

En base a la definición de miembros de las dimensiones de Fecha, Lugar y Enfermedad, se obtuvieron las siguientes jerarquías para cada una de las dimensiones y se define la tabla de hechos fact_asis_med en la cual se encuentra la medida de cantidad de asistencias médicas prestadas.

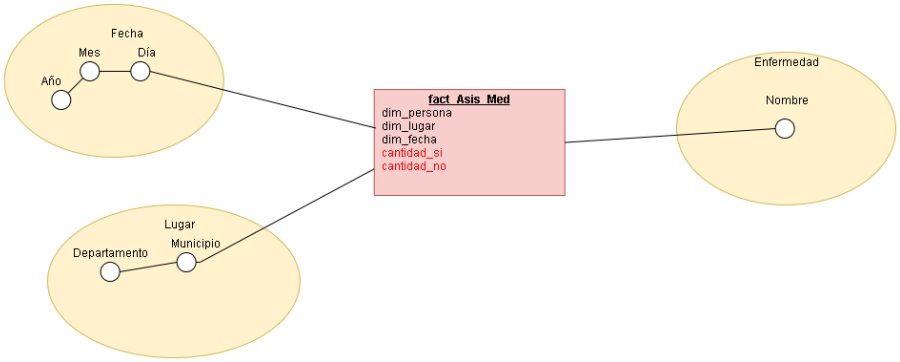


Ilustración 9 Modelo DF – asistencia_medica

Modelo DF – Defunciones por municipio

En base a la definición de miembros de las dimensiones de Fecha, Lugar y Enfermedad, se definieron las siguientes jerarquías en base a las dimensiones previamente definidas, visualizando finalmente la relación entre estas en la tabla de hechos de defun_municipio, la cual cuenta como medida la cantidad de fallecimientos registrados en cada municipio y fecha de las tres enfermedades empleadas en este ejercicio.

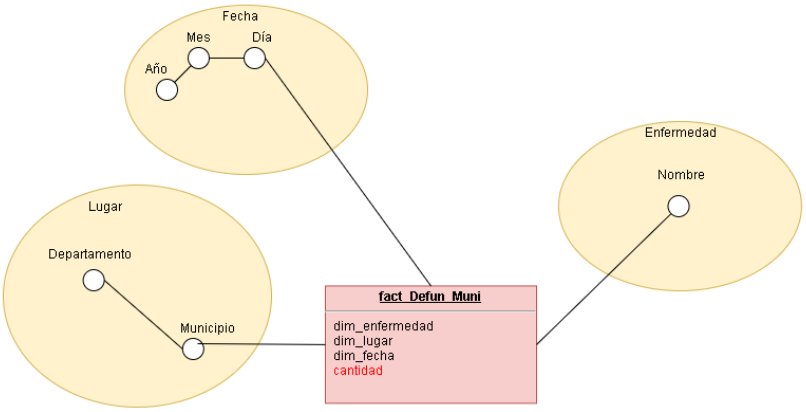


Ilustración 10 Modelo DF – Defunciones_municipio

Nota: se define la dimensión de lugar, hasta este momento, con los miembros de departamento y municipio por practicidad, ya que igualmente hacen referencia a la dimensión de municipio definida en el modelo multidimensional.

Modelo DF – Muertes

En este ultimo modelo se define el cubo de muertes el cual contempla las dimensiones de persona, fecha, luchar y enfermedad, sin definir nunca medida, esto con el fin de poder analizar cada uno de los fallecimientos registrados y sus particularidades.

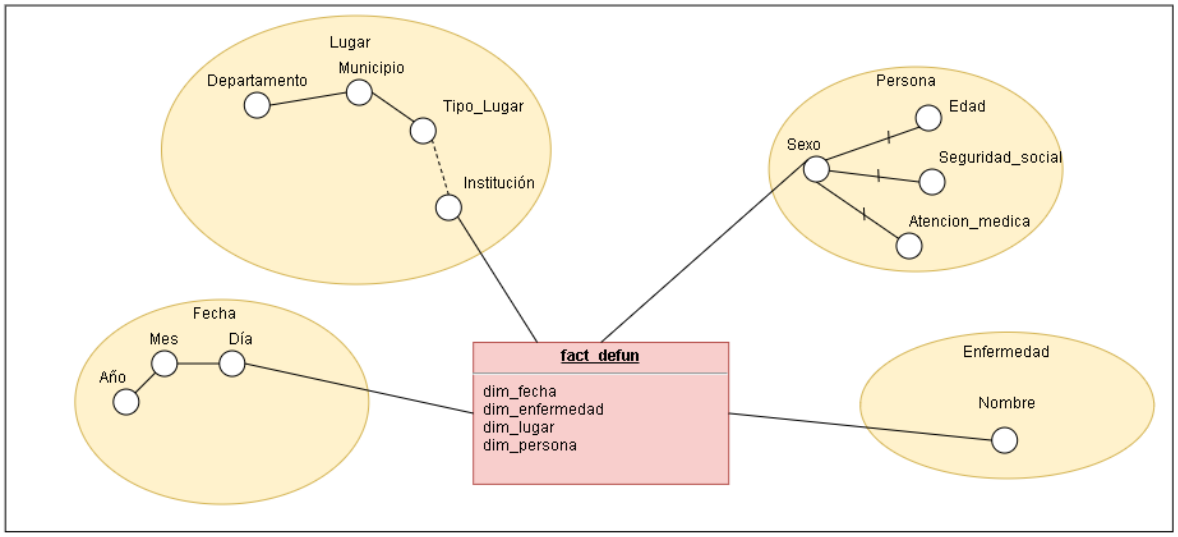


Ilustración 20 Muertes

DISEÑO LOGICO

Con todos los modelos que nos permitan conocer mucho más a fondo los elementos que compondrían al Data Warehouse destinado a satisfacer las necesidades de la empresa, se diseñara el modelo ROLAP, el cual permitirá la posterior creación de la base de datos.

Modelo ROLAP

A continuación, se expone el modelo relacional del Data Warehouse el se realiza en base a todo el proceso de diseño conceptual desarrollado previamente.

se expone el modelo relacional del Data Warehouse el se realiza en base a todo el proceso de diseño conceptual desarrollado previamente.

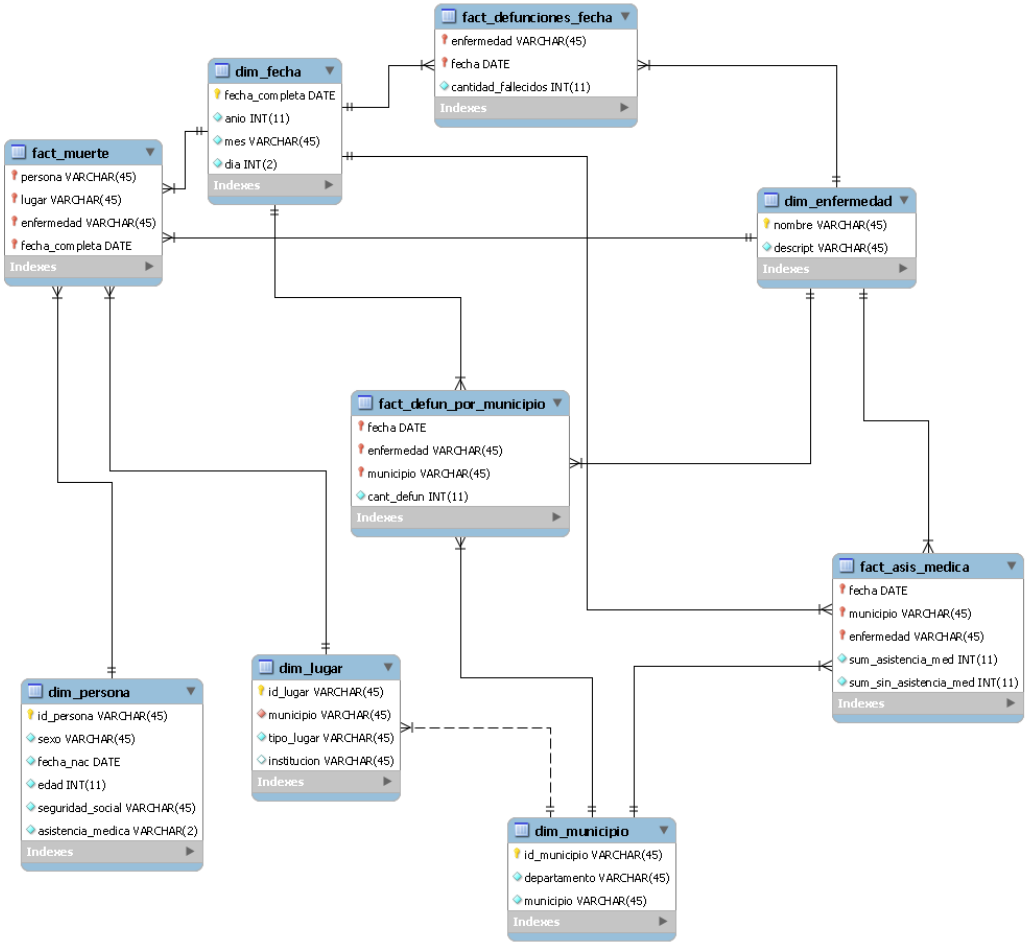


Ilustración 21 Modelo ROLAP

DDL del Data Ware House

Por medio de MySQL Workbench se generó el siguiente script en el cual se encuentran las sentencias DDL para cada una de las tablas definidas en el modelo:

```
-- -----------------------------------------------------
-- Schema Bona_Health_DW
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `Bona_Health_DW` DEFAULT CHARACTER SET utf8 ;
-- -----------------------------------------------------
-- Schema ods_bona_health
-- -----------------------------------------------------
USE `Bona_Health_DW` ;

-- -----------------------------------------------------
-- Table `Bona_Health_DW`.`dim_enfermedad`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Bona_Health_DW`.`dim_enfermedad` (
  `nombre` VARCHAR(45) NOT NULL,
  `descript` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`nombre`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Bona_Health_DW`.`dim_fecha`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Bona_Health_DW`.`dim_fecha` (
  `fecha_completa` DATE NOT NULL,
  `anio` INT NOT NULL,
  `mes` VARCHAR(45) NOT NULL,
  `dia` INT(2) NOT NULL,
  PRIMARY KEY (`fecha_completa`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Bona_Health_DW`.`fact_defunciones_fecha`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Bona_Health_DW`.`fact_defunciones_fecha` (
  `enfermedad` VARCHAR(45) NOT NULL,
  `fecha` DATE NOT NULL,
  `cantidad_fallecidos` INT NOT NULL,
  PRIMARY KEY (`enfermedad`, `fecha`),
  INDEX `fk_Defun_Fecha1_idx` (`fecha` ASC) ,
  CONSTRAINT `fk_Defun_Enfermedad1`
    FOREIGN KEY (`enfermedad`)
    REFERENCES `Bona_Health_DW`.`dim_enfermedad` (`nombre`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Defun_Fecha1`
    FOREIGN KEY (`fecha`)
```

```
    REFERENCES `Bona_Health_DW`.`dim_fecha` (`fecha_completa`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Bona_Health_DW`.`dim_municipio`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Bona_Health_DW`.`dim_municipio` (
  `id_municipio` VARCHAR(45) NOT NULL,
  `departamento` VARCHAR(45) NOT NULL,
  `municipio` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id_municipio`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Bona_Health_DW`.`fact_asis_medica`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Bona_Health_DW`.`fact_asis_medica` (
  `fecha` DATE NOT NULL,
  `municipio` VARCHAR(45) NOT NULL,
  `sum_asistencia_med` INT NOT NULL,
  `sum_sin_asistencia_med` INT NOT NULL,
  PRIMARY KEY (`fecha`, `municipio`),
  INDEX `fk_Asis_med_Fecha1_idx` (`fecha` ASC) ,
  INDEX `fk_fact_asis_medica_dim_municipio1_idx` (`municipio` ASC) ,
  CONSTRAINT `fk_Asis_med_Fecha1`
    FOREIGN KEY (`fecha`)
    REFERENCES `Bona_Health_DW`.`dim_fecha` (`fecha_completa`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_fact_asis_medica_dim_municipio1`
    FOREIGN KEY (`municipio`)
    REFERENCES `Bona_Health_DW`.`dim_municipio` (`id_municipio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-----------------------------------------------------
-- Table `Bona_Health_DW`.`fact_defun_por_municipio`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Bona_Health_DW`.`fact_defun_por_municipio` (
  `fecha` DATE NOT NULL,
  `enfermedad` VARCHAR(45) NOT NULL,
  `municipio` VARCHAR(45) NOT NULL,
  `cant_defun` INT NOT NULL,
  PRIMARY KEY (`fecha`, `enfermedad`, `municipio`),
  INDEX `fk_Defun_muni_Enfermedad1_idx` (`enfermedad` ASC) ,
  INDEX `fk_fact_defun_por_lugar_dim_municipio1_idx` (`municipio` ASC) ,
  CONSTRAINT `fk_Defun_muni_Fecha`
```

```
    FOREIGN KEY (`fecha`)
    REFERENCES `Bona_Health_DW`.`dim_fecha` (`fecha_completa`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Defun_muni_Enfermedad1`
    FOREIGN KEY (`enfermedad`)
    REFERENCES `Bona_Health_DW`.`dim_enfermedad` (`nombre`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_fact_defun_por_lugar_dim_municipio1`
    FOREIGN KEY (`municipio`)
    REFERENCES `Bona_Health_DW`.`dim_municipio` (`id_municipio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Bona_Health_DW`.`dim_persona`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Bona_Health_DW`.`dim_persona` (
  `id_persona` VARCHAR(45) NOT NULL,
  `sexo` VARCHAR(45) NOT NULL,
  `edad` INT NOT NULL,
  `seguridad_social` VARCHAR(45) NOT NULL,
  `asistencia_medica` VARCHAR(2) NOT NULL,
  PRIMARY KEY (`id_persona`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Bona_Health_DW`.`dim_lugar`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Bona_Health_DW`.`dim_lugar` (
  `id_lugar` VARCHAR(45) NOT NULL,
  `tipo_lugar` VARCHAR(45) NOT NULL,
  `institucion` VARCHAR(45) NULL,
  `municipio` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id_lugar`),
  INDEX `fk_dim_lugar_dim_municipio1_idx` (`municipio` ASC) ,
  CONSTRAINT `fk_dim_lugar_dim_municipio1`
    FOREIGN KEY (`municipio`)
    REFERENCES `Bona_Health_DW`.`dim_municipio` (`id_municipio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Bona_Health_DW`.`fact_muerte`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Bona_Health_DW`.`fact_muerte` (
  `persona` VARCHAR(45) NOT NULL,
  `lugar` VARCHAR(45) NOT NULL,
```

```
  `nombre` VARCHAR(45) NOT NULL,
  `fecha_completa` DATE NOT NULL,
  INDEX `fk_fact_muerte_dim_persona1_idx` (`persona` ASC) ,
  INDEX `fk_fact_muerte_dim_lugar1_idx` (`lugar` ASC) ,
  INDEX `fk_fact_muerte_dim_enfermedad1_idx` (`nombre` ASC) ,
  INDEX `fk_fact_muerte_dim_fecha1_idx` (`fecha_completa` ASC) ,
  PRIMARY KEY (`persona`, `lugar`, `nombre`, `fecha_completa`),
  CONSTRAINT `fk_fact_muerte_dim_persona1`
    FOREIGN KEY (`persona`)
    REFERENCES `Bona_Health_DW`.`dim_persona` (`id_persona`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_fact_muerte_dim_lugar1`
    FOREIGN KEY (`lugar`)
    REFERENCES `Bona_Health_DW`.`dim_lugar` (`id_lugar`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_fact_muerte_dim_enfermedad1`
    FOREIGN KEY (`nombre`)
    REFERENCES `Bona_Health_DW`.`dim_enfermedad` (`nombre`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_fact_muerte_dim_fecha1`
    FOREIGN KEY (`fecha_completa`)
    REFERENCES `Bona_Health_DW`.`dim_fecha` (`fecha_completa`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

ETL

Con la creación del Data Warehouse por medio del script expuesto anteriormente, a continuación, se expondrán los procesos ETL por medio de los cuales se dejará finalmente el Data Warehouse cargado y listo para ser usado. Se mostrarán las actividades relacionadas con el diseño de los ETL, secuencia de los procesos, creación de la base de datos ODS y el programa desarrollado en Python en el cual se integrará el diseño previamente expuesto.

Diseño de ETL

Antes de desarrollar el componente de software destinado para los procesos de extracción de la información de los archivos CSV en donde se encuentra los datos relacionados con las defunciones, transformación o limpieza de los datos y carga de los datos, se realizo el siguiente diagrama en el cual se exponen los elementos involucrados en el proceso, definiendo las fuentes de datos, la forma en que se extraerán los datos de estas, métodos y clases requeridos para la conexión y ejecución de sentencias SQL y un listado previo de las trasformaciones a realizar.

En este diagrama se hace un vistazo a lo que se podría implementar en el programa, pero puede que algunos métodos o elementos no se implementen, esto ya dependerá exclusivamente de los inconvenientes que se presenten sobre el desarrollo.

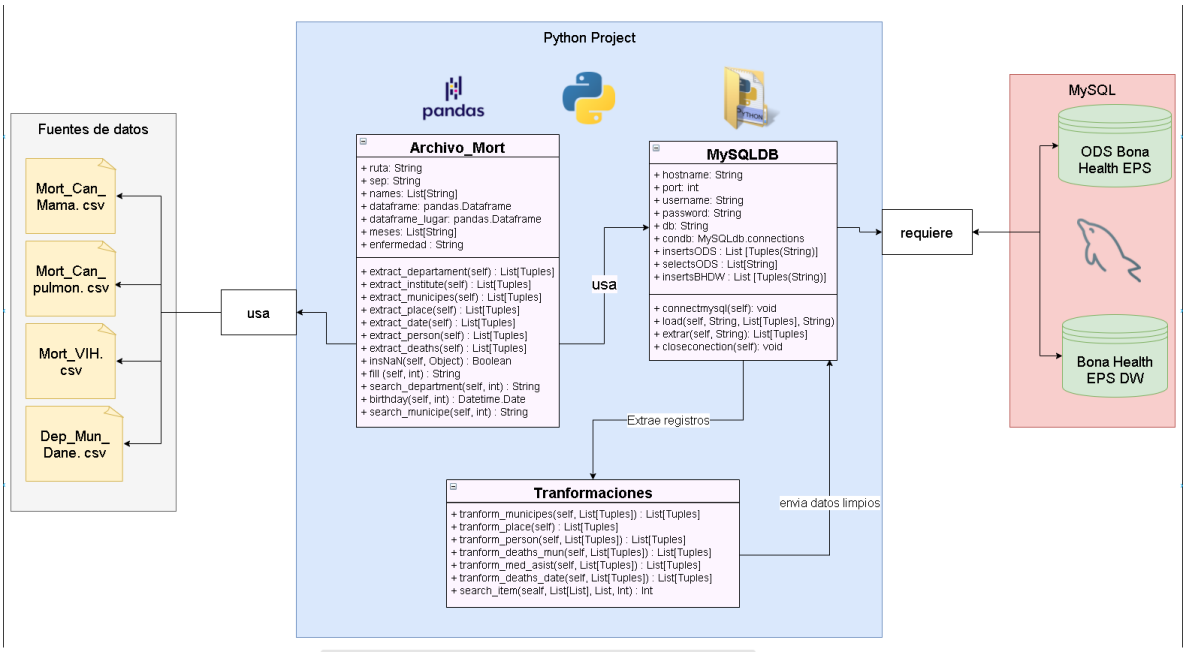


Ilustración 22 Diagrama de Clases y componentes del sistema

A continuación, se definen algunos elementos presentados en el anterior diagrama:

- Fuentes de datos: en este apartado se define la carpeta en la cual se encontrarán los archivos en formato csv con los datos de las defunciones y los códigos de municipios y departamentos requeridos para este ejercicio.
- Proyecto en Python: en este apartado se definen las clases y librerías a emplear en el programa. Para este ejercicio se empleo a Python en su versión 3.9.
    - Pandas: librería enfocada en la manipulación de datagramas los cuales son estructuras de datos las cuales almacenan los datos extraídos de cada uno de los archivos csv a usar.
    - MySQLdb: librería destinada para la conexión a bases de datos MySQL.
    - Archivo_Mort: clase la cual se encargará de la lectura de los archivos csv y la extracción de los datos para cada una de las tablas definidas en la base de datos ODS.

- MySQLDB: clase encargada de la creación de las conexiones a las bases de datos MySQL, además de contener métodos para la extracción y carga de datos.
- Transformaciones: clase con los de limpieza y transformación de los datos extraídos de la base de datos ODS, para la posterior inserción en cada una de las tablas definidas para el Data Warehouse.

- MySQL: en este apartado se encuentran las dos bases de datos que serán el destino de los datos manipulados dentro del programa en Python.
  - ODS Bona Health EPS: base de datos creada en base al modelo entidad – relación de las fuentes de datos a emplear en este ejercicio.
  - Bona Health EPS DW: base de datos para el Data Warehouse de este ejercicio.

Secuencia de ETL

Con los componentes definidos a emplear en este proyecto, ahora se realizara un listado o secuencia de las actividades a realizar para la extracción de los datos de los archivos csv, carga de estos en la base de datos ODS, la extracción de los datos del ODS, la transformación de estos y su posterior carga al Data Warehouse.

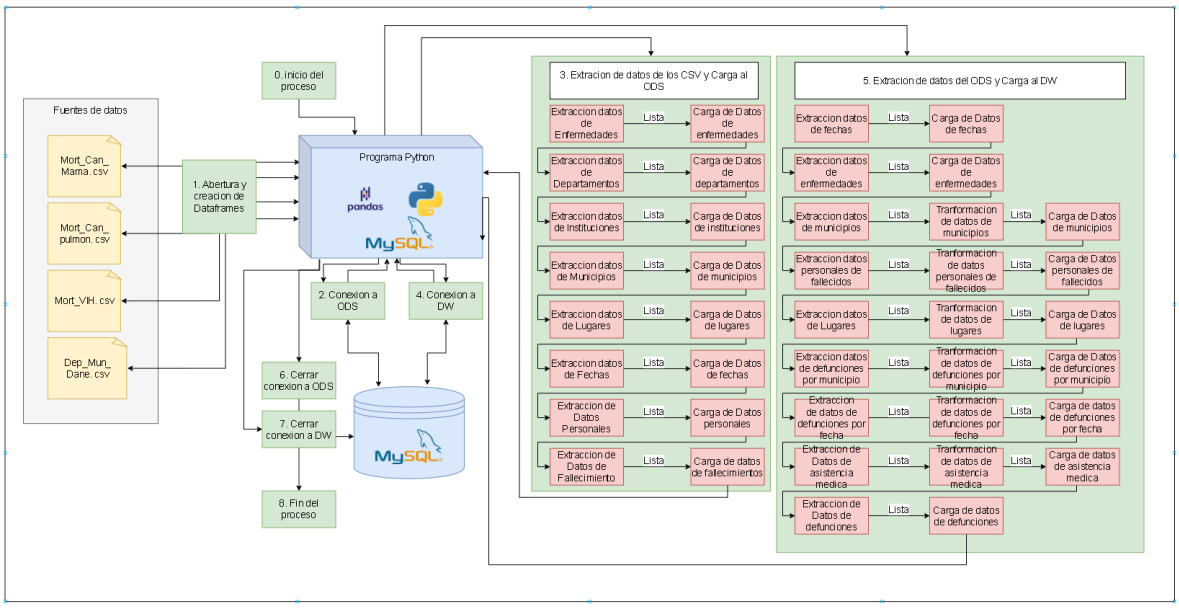


Ilustración 23 Secuencia de procesos ETL

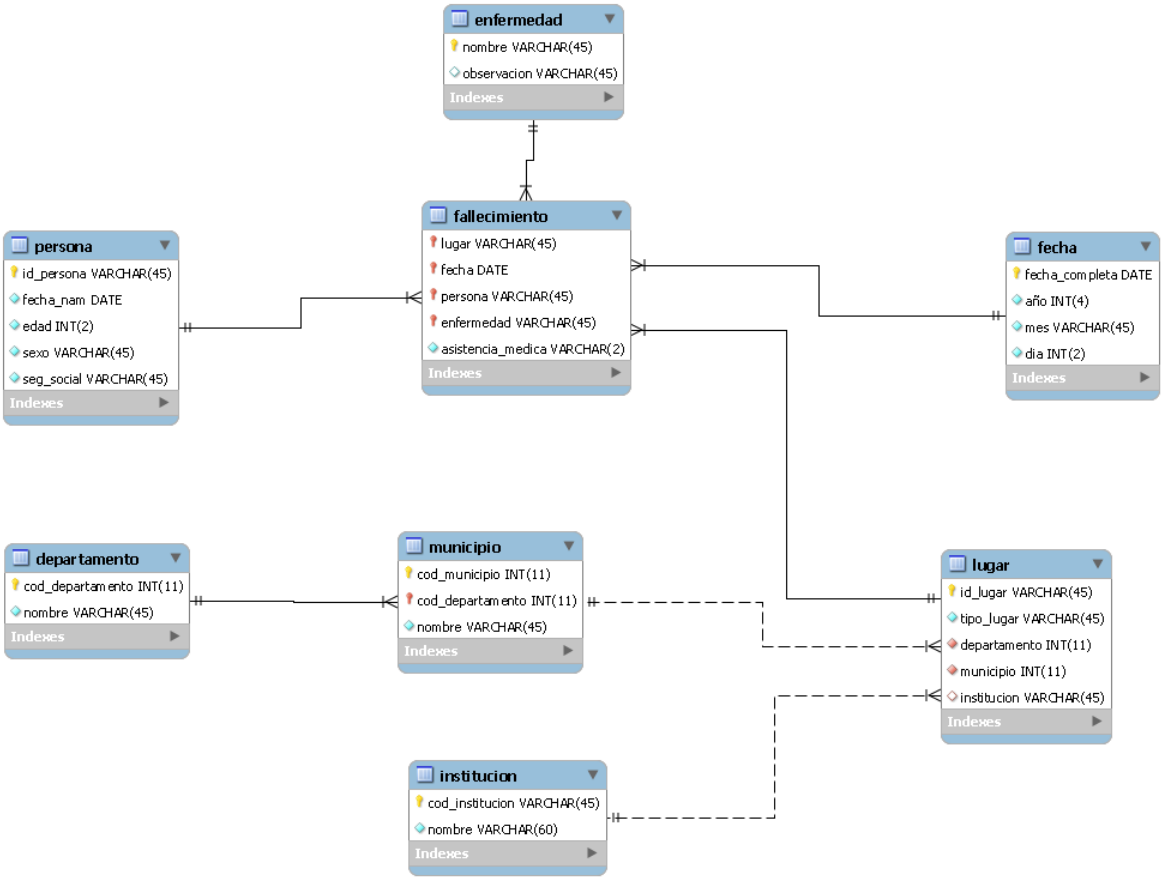
A grandes rasgos, a continuación, se expondrán los principales procesos definidos en el diagrama:

1. **Abertura y creación de Dataframes**: este proceso contempla la lectura de los archvis csv y su conformación como Dataframes manipulables en el programa. Este proceso se realizará con cada uno de los archivos, contemplando que la extracción de los datos se realizara de forma individual con los archivos que contienen los datos sobre las defunciones, y se prevé que se requiera el uso de un Dataframe en donde se encuentre la información de los departamentos y municipios.
2. **Conexión al ODS**: este proceso contempla el ingreso de los datos requeridos para la conexión a la base de datos ODS en la cual se almacenarán los datos extraídos de los archivos.
   Si este proceso no resulta exitoso, es decir, con la conexión al ODS correctamente, se detiene todo, con el fin de evitar errores en la ejecución.
3. **Extracción de datos de archivos CSV y carga al ODS**: este proceso contemplara un conjunto de subprocesos, cada uno destinado para la extracción de un conjunto de datos en específico, el curado de los datos extraídos y la posterior carga de estos a la base de datos ODS. En este proceso se contemplan las primeras trasformaciones las cuales facilitaran posteriormente la carga de los datos al Data Warehouse. El orden de estos subprocesos se basa en las dependencias de tablas del ODS.
4. **Conexión al Data Warehouse**: este proceso contempla el ingreso de los datos requeridos para la conexión a la base de datos definida para el Data Warehouse en la cual se almacenarán los datos extraídos de la base de datos ODS en la cual se encuentran los datos extraídos d ellos archivos csv.
   Si este proceso no resulta exitoso, es decir, con la conexión al Data Warehouse correctamente, se detiene todo, con el fin de evitar errores en la ejecución.
5. **ETL (Extracción, trasformaciones y carga al Data Warehouse)**: este proceso contemplara un conjunto de subprocesos secuenciados en los cuales se realizará la extracción de los datos contenidos en el ODS, la transformación de estos que permita su posterior inserción en el Data Warehouse. El orden en que se cargan cada uno de los datos se hace en base a las dimensiones definidas, la dependencia entre estas y dejando siempre al final la carga de los datos en las tablas de hechos, esto con el fin de evitar conflictos de claves foráneas. Debido a que, al momento que se cargaron los datos de los archivos csv al ODS se realizaron ya algunas transformaciones y conociendo la estructura de las tablas del ODS y el data Warehouse, se prevé que, para la carga de los datos de ciertas tablas de dimensiones y hechos no se requería implementar métodos para que los datos encajen con la estructura definida en estas tablas.

6. 7. **Cierre de conexiones**: en estos dos procesos se cerrarán las dos conexiones empleadas para la extracción y carga de datos en el ODS y el almacén de datos.

Modelo relacional ODS

Antes del desarrollo del componente de software encargado de la extracción, transformación y carga de datos, se realiza el modelado de la base de datos ODS en la cual se almacenarán los datos que se encuentran en los archivos csv.



Con este modelo, en el cual se definen las tablas para conocer la ubicación en donde ocurrió el fallecimiento, la fecha donde ocurrió, los datos personales de la persona fallecida y la enfermedad que causo el descenso, se genero el siguiente script con las sentencias DDL para la creación de la base de datos y las tablas:

```
--

-- Base de datos: `ods_bona_health`

--
```

```
CREATE SCHEMA IF NOT EXISTS ` ods_bona_health` DEFAULT CHARACTER SET utf8 ;

-- -----------------------------------------------------

-- Schema ods_bona_health

-- -----------------------------------------------------

USE ` ods_bona_health ` ;

-- -------------------------------------------------------



--

-- Estructura de tabla para la tabla `departamento`

--

CREATE TABLE `departamento` (

  `cod_departamento` int(11) NOT NULL,

  `nombre` varchar(45) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- -------------------------------------------------------



--

-- Estructura de tabla para la tabla `enfermedad`

--

CREATE TABLE `enfermedad` (

  `nombre` varchar(45) NOT NULL,

  `observacion` varchar(45) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- -------------------------------------------------------



--

-- Estructura de tabla para la tabla `fallecimiento`

--

CREATE TABLE `fallecimiento` (

  `lugar` varchar(45) NOT NULL,

  `fecha` date NOT NULL,
```

```
  `persona` varchar(45) NOT NULL,
  `enfermedad` varchar(45) NOT NULL,
  `asistencia_medica` varchar(2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
-- -------------------------------------------------------


--
-- Estructura de tabla para la tabla `fecha`
--
CREATE TABLE `fecha` (
  `fecha_completa` date NOT NULL,
  `año` int(4) NOT NULL,
  `mes` varchar(45) NOT NULL,
  `dia` int(2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
-- -------------------------------------------------------


--
-- Estructura de tabla para la tabla `institucion`
--
CREATE TABLE `institucion` (
  `cod_institucion` varchar(45) NOT NULL,
  `nombre` varchar(60) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


-- -------------------------------------------------------
--
-- Estructura de tabla para la tabla `lugar`
--
CREATE TABLE `lugar` (
  `id_lugar` varchar(45) NOT NULL,
```

```
  `tipo_lugar` varchar(45) NOT NULL,
  `departamento` int(11) NOT NULL,
  `municipio` int(11) NOT NULL,
  `institucion` varchar(45) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
- -------------------------------------------------------
--
-- Estructura de tabla para la tabla `municipio`
--

CREATE TABLE `municipio` (
  `cod_municipio` int(11) NOT NULL,
  `cod_departamento` int(11) NOT NULL,
  `nombre` varchar(45) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
-- --------------------------------------------------------
--
-- Estructura de tabla para la tabla `persona`
--
CREATE TABLE `persona` (
  `id_persona` varchar(45) NOT NULL,
  `fecha_nam` date NOT NULL,
  `edad` int(2) NOT NULL,
  `sexo` varchar(45) NOT NULL,
  `seg_social` varchar(45) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
--
-- Índices para tablas volcadas
--

--
```

```
-- Indices de la tabla `departamento`
--
ALTER TABLE `departamento`
  ADD PRIMARY KEY (`cod_departamento`);
--
-- Indices de la tabla `enfermedad`
--
ALTER TABLE `enfermedad`
  ADD PRIMARY KEY (`nombre`);
--
-- Indices de la tabla `fallecimiento`
--
ALTER TABLE `fallecimiento`
  ADD PRIMARY KEY (`lugar`,`fecha`,`persona`,`enfermedad`),
  ADD KEY `fk_fallecimiento_lugar1_idx` (`lugar`),
  ADD KEY `fk_fallecimiento_fecha1_idx` (`fecha`),
  ADD KEY `fk_fallecimiento_persona1_idx` (`persona`),
  ADD KEY `fk_fallecimiento_enfermedad1_idx` (`enfermedad`);
--
-- Indices de la tabla `fecha`
--
ALTER TABLE `fecha`
  ADD PRIMARY KEY (`fecha_completa`);
--
-- Indices de la tabla `institucion`
--
ALTER TABLE `institucion`
  ADD PRIMARY KEY (`cod_institucion`);
--
-- Indices de la tabla `lugar`
--
```

```
ALTER TABLE `lugar`
  ADD PRIMARY KEY (`id_lugar`),
  ADD KEY `fk_lugar_institucion1_idx` (`institucion`),
  ADD KEY `fk_lugar_municipio1_idx` (`municipio`,`departamento`);
--
-- Indices de la tabla `municipio`
--
ALTER TABLE `municipio`
  ADD PRIMARY KEY (`cod_municipio`,`cod_departamento`),
  ADD KEY `fk_municipio_departamento_idx` (`cod_departamento`);
--
-- Indices de la tabla `persona`
--
ALTER TABLE `persona`
  ADD PRIMARY KEY (`id_persona`);
--
-- Restricciones para tablas volcadas
--
--
-- Filtros para la tabla `fallecimiento`
--
ALTER TABLE `fallecimiento`
  ADD CONSTRAINT `fk_fallecimiento_enfermedad1` FOREIGN KEY (`enfermedad`) REFERENCES
`enfermedad` (`nombre`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk_fallecimiento_fecha1` FOREIGN KEY (`fecha`) REFERENCES `fecha`
(`fecha_completa`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk_fallecimiento_lugar1` FOREIGN KEY (`lugar`) REFERENCES `lugar`
(`id_lugar`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk_fallecimiento_persona1` FOREIGN KEY (`persona`) REFERENCES `persona`
(`id_persona`) ON DELETE NO ACTION ON UPDATE NO ACTION;
--
-- Filtros para la tabla `lugar`
```

```
--

ALTER TABLE `lugar`

  ADD CONSTRAINT `fk_lugar_institucion1` FOREIGN KEY (`institucion`) REFERENCES
`institucion` (`cod_institucion`) ON DELETE NO ACTION ON UPDATE NO ACTION,

  ADD CONSTRAINT `fk_lugar_municipio1` FOREIGN KEY (`municipio`,`departamento`) REFERENCES
`municipio` (`cod_municipio`, `cod_departamento`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--

-- Filtros para la tabla `municipio`

--

ALTER TABLE `municipio`

  ADD CONSTRAINT `fk_municipio_departamento` FOREIGN KEY (`cod_departamento`) REFERENCES
`departamento` (`cod_departamento`) ON DELETE NO ACTION ON UPDATE NO ACTION;

COMMIT;
```

Programa ETL

Con la base de datos ODS, el Data Warehouse, las clases previamente definidas y la secuencia de procesos creada, se realizo el siguiente programa en Python, el cual cuenta con comentarios para comprender su funcionamiento. Este programa se entregará en un archivo comprimido, con el fin de prevenir problemas con las librerías implementadas. Además, dentro del directorio del programa, se encontrara la carpeta DATASETS en donde estarán los archivos csv con los datos a manipular por el programa.

```
from datetime import datetime  # Libreria para la manipulacion de fechas

import MySQLdb.connections as con  # Libreria para la conexion con bases MySQL
import pandas as pd  # Libreria para la manipulacion de archivos CSV
from dateutil.parser import parse  # Metodo para formatear fechas en Strings sin
indicar el formato de destino
from dateutil.relativedelta import relativedelta  # Metodo para encontrar la
diferencia entre dos fechas


# Clase destinada para la conexion a bases de datos MySQL
# von los metodos que permiten la carga y extraccion de datos
class MySQLCon:

    # Contructor que recibe los datos necesarios para la conexion
    # a una base de datos con sus respectivas credenciales
    # y que a su vez inicia la conexion a la base de datos
    # Para las consultas e inserciones se crean listas con las sentencias
requeridas
    def __init__(self, hostname, port, user, password, db):
        self.hostname = hostname
        self.port = port
        self.username = user
```

```
        self.password = password
        self.database = db
        self.condb = None
        self.connect()
        self.insertsODS = [("ods_bona_health.departamento
(cod_departamento,nombre) VALUES (%s, %s)", "DEPARTAMENTO"),
                           ("ods_bona_health.institucion (
cod_institucion,nombre) VALUES (%s, %s)", "INSTITUCION"),
                           ("ods_bona_health.municipio
(cod_municipio,cod_departamento,nombre) VALUES (%s, %s, %s)",
                            "MUNICIPIO"),
                           ("ods_bona_health.lugar
(id_lugar,tipo_lugar,departamento,municipio,institucion) VALUES (%s, %s, %s,%s,
%s)",
                               "LUGAR"),
                           ("ods_bona_health.fecha (fecha_completa,año,mes,dia)
VALUES (%s, %s, %s,%s)", "FECHA"),
                           ("ods_bona_health.persona
(id_persona,fecha_nam,edad,sexo,seg_social) VALUES (%s, %s, %s,%s,%s)",
                               "PERSONA"),
                           (
                               "ods_bona_health.fallecimiento
(lugar,fecha,persona,enfermedad,asistencia_medica) VALUES (%s,%s, %s,%s,%s)",
                               "FALLECIMIENTO")]
        self.selectsODS = ["* FROM fecha;", "* FROM enfermedad;",
                           "municipio.*, departamento.nombre from municipio INNER
JOIN departamento ON municipio.cod_departamento =
departamento.cod_departamento;",
                           "persona.*, fallecimiento.asistencia_medica from
persona INNER JOIN fallecimiento ON persona.id_persona = fallecimiento.persona;",
                           "* FROM lugar;",
                           "fallecimiento.fecha, fallecimiento.enfermedad,
lugar.departamento, lugar.municipio FROM fallecimiento INNER JOIN lugar ON
fallecimiento.lugar = lugar.id_lugar;",
                           "fallecimiento.fecha, fallecimiento.enfermedad,
lugar.departamento, lugar.municipio FROM fallecimiento INNER JOIN lugar ON
fallecimiento.lugar = lugar.id_lugar;",
                           "fallecimiento.fecha, fallecimiento.enfermedad,
lugar.departamento, lugar.municipio,fallecimiento.asistencia_medica FROM
fallecimiento INNER JOIN lugar ON fallecimiento.lugar = lugar.id_lugar;",
                           "fallecimiento.persona, fallecimiento.lugar,
fallecimiento.enfermedad, fallecimiento.fecha FROM `fallecimiento`;"]
        self.insertBHDW = [("dim_fecha (fecha_completa,anio,mes,dia) VALUES
(%s,%s,%s,%s);", "DIM FECHA"),
                           ("dim_enfermedad (nombre, descript) VALUES (%s,%s);",
"DIM ENFERMEDAD"),
                           ("dim_municipio (id_municipio,departamento,municipio)
VALUES (%s,%s,%s);", "DIM MUNICIPIO"),
                           (
                               "dim_persona
(id_persona,sexo,fecha_nac,edad,seguridad_social,asistencia_medica) VALUES
(%s,%s,%s,%s,%s,%s);",
                               "DIM PERSONA"),
                           ("dim_lugar
(id_lugar,municipio,tipo_lugar,institucion) VALUES (%s,%s,%s,%s);", "DIM LUGAR"),
                           ("fact_defun_por_municipio
(fecha,enfermedad,municipio,cant_defun) VALUES (%s,%s,%s,%s);",
                            "FACT_DEFUN_POR_MUNICIPIO"),
```

```
                            ("fact_defunciones_fecha (fecha,enfermedad,
cantidad_fallecidos) VALUES (%s,%s,%s);",
                             "FACT_DEFUN_POR_FECHA"),
                            (
                                "fact_asis_medica
(fecha,municipio,enfermedad,sum_asistencia_med,sum_sin_asistencia_med) VALUES
(%s,%s,%s,%s,%s);",
                                "FACT_ASIS_MEDICA"),
                            ("fact_muerte
(persona,lugar,enfermedad,fecha_completa) VALUES (%s,%s,%s,%s);",
                             "FACT_MUERTE")]

    # Metodo para la conexion a la base de datos con la libreria
    # MySQLdb con los datos almancenados en el cosntructor de la instancia
    def connect(self):
        try:
            self.condb = con.Connection(host=self.hostname, port=self.port,
                                        user=self.username,
                                        passwd=self.password,
                                        db=self.database)
            cursor = self.condb.cursor()
            cursor.execute("SET lc_time_names = 'es_CO';")
            print("Conexión exitoso a " + self.database)
        except Exception as e:
            print("Error de conexión", e)

    # Metodo para la carga de datosen la BD, el cual recibe parte de la sentencia
para la carga de los datos,
    # una lista con la tuplas de los datos a cargar y una cadena con el nombre de
la tabla. La carga de los datos se
    # realiza con el metodo executemany, el cual usa la sentencia SQL y la lista
que iterara para cargar los datos
    def load(self, sql, data, table):
        cursor = self.condb.cursor()
        try:
            cursor.executemany("INSERT IGNORE INTO " + sql, data)
            print("TABLA ", table, " CARGADA CON ", len(data),
                  " REGISTROS (Los registros pueden ser repetidos y por ende
haber sido ignroados)")
            self.condb.commit()
        except Exception as e:
            print("Error de insercion", e)
            self.condb.rollback()

    # Metodo para la extraccion de datos de la BD, el cual un String para
complementar la sentencia SELECT
    # y retornara los datos obtenidos por el cursor, los cuales vienen en una
lista de tuplas
    def extract(self, sql):
        cursor = self.condb.cursor()
        try:
            cursor.execute("SELECT " + sql)
            return cursor.fetchall()
        except Exception as e:
            print("Error de consulta", e)

    # Metodo encargado de cerrar la conexion con la base de datos,
    #  conexion que se encuentra alamcenada en la variable condb
```

```
    def closeconection(self):
        try:
            self.condb.close()
            print('Desconexión exitosa a ' + self.database)
        except Exception as e:
            print("Error de desconexión", e)


## Clase encarga de la manipulacion de los archivos csv
# haciendo uso de la libreria Pandas
class ArchivoMort:

    # Contructor de la clase, el cual recibe el nombre del archivo ubicado en la
carpeta
    # DATASETS y el nombre de la enfermedad asociada a este. Dentro de este
metodo se incluyen
    # parametros como separador de datos, nombres de las columnas y se crea el
dataframe a manipular
    def __init__(self, nombre, enfermedad):
        self.ruta = 'DATASETS/' + nombre
        self.sep = ';'
        self.names = ['COD_DPTO', 'COD_MUNIC', 'A_DEFUN', 'COD_INSP',
'SIT_DEFUN', 'OTRSITIODE',
                      'COD_INST', 'NOM_INST', 'TIPO_DEFUN', 'FECHA_DEF', 'ANO',
'MES', 'HORA', 'MINUTOS',
                      'SEXO', 'FECHA_NAC', 'EST_CIVIL', 'EDAD', 'NIVEL_EDU',
'ULTCURFAL', 'MUERTEPORO',
                      'SIMUERTEPO', 'OCUPACION', 'IDPERTET', 'IDPUEBIN',
'N_IDPUEBIN', 'CODPRES', 'CODPTORE',
                      'CODMUNRE', 'AREA_RES', 'BARRIOFAL', 'COD_LOCA', 'CODIGO',
'VEREDAFALL', 'SEG_SOCIAL',
                      'IDADMISALU', 'IDCLASADMI', 'PMAN_MUER', 'CONS_EXP',
'MU_PARTO', 'T_PARTO', 'TIPO_EMB',
                      'T_GES', 'PESO_NAC', 'EDAD_MADRE', 'N_HIJOSV', 'N_HIJOSM',
'EST_CIVM', 'NIV_EDUM',
                      'ULTCURMAD', 'EMB_FAL', 'EMB_SEM', 'EMB_MES', 'MAN_MUER',
'COMOCUHEC', 'CODOCUR',
                      'CODMUNOC', 'DIROCUHEC', 'LOCALOCUHE', 'C_MUERTE',
'C_MUERTEB', 'C_MUERTEC', 'C_MUERTED',
                      'C_MUERTEE', 'ASIS_MED', 'N_DIR1', 'T_DIR1', 'M_DIR1',
'C_DIR1', 'C_DIR12', 'N_ANT1', 'T_ANT1',
                      'M_ANT1', 'C_ANT1', 'C_ANT12', 'N_ANT2', 'T_ANT2',
'M_ANT2', 'C_ANT2', 'C_ANT22', 'N_ANT3',
                      'T_ANT3',
                      'M_ANT3', 'C_ANT3', 'C_ANT32', 'N_PAT1', 'T_PAT1',
'M_PAT1', 'C_PAT1', 'N_PAT2', 'C_PAT2',
                      'N_BAS1',
                      'C_BAS1', 'N_MCM1', 'C_MCM1', 'CAUSA_666', 'IDPROFCER',
'DD_EXP', 'MM_EXP', 'FECHA_EXP',
                      'FECHAGRA',
                      'CAU_HOMOL', 'GRU_ED1', 'GRU_ED2', 'HORA_SE']
        # Dataframe de los datos del csv relacionados con la mortalidad de una
enfermedad
        self.dataframe = pd.read_csv(self.ruta, sep=self.sep, header=None,
names=self.names)
        # Dataframe con los codigos de los municipios y departamentos del pais
        self.dataframe_lugar =
pd.read_csv("DATASETS/Departamentos_y_municipios_de_Colombia.csv", sep=self.sep,
```

```
                                                header=None,
                                                names=['REGION', 'CÓDIGO DANE DEL
DEPARTAMENTO', 'DEPARTAMENTO',
                                                       'CÓDIGO DANE DEL MUNICIPIO',
'MUNICIPIO'])
        # Tupla con los nombres de los meses en español
        self.meses = (
            "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
"Agosto", "Septiembre", "Octubre",
            "Noviembre",
            "Diciembre")
        self.enfermedad = enfermedad

    # Metodo para la verificación de valores vacios extraidos del dataframe
    def isNaN(self, num):
        return num != num

    # Metodo para el llenado de digitos de los numeros
    def llenar(self, param):
        return str(('0' * (3 - len(str(param)))) + str(param))

    # Metodo encargado de la extraccion del nombre de un departamento con base en
su codigo
    def search_departament(self, clave):
        for i in self.dataframe_lugar.index:
            if self.dataframe_lugar['CÓDIGO DANE DEL DEPARTAMENTO'][i] == clave:
                return self.dataframe_lugar['DEPARTAMENTO'][i]
        else:
            return "SIN NOMBRE"

    # Metodo encargado de la extraccion de la fecha de nacimiento de una persona,
permitiendo la limpieza
    # de los valores y haciendo uso de los valores de fecha de defuncion y edad
en los casos requeridos
    # en donde no se tenga la fecha de nacimiento de la persona, además de
arreglar problemas de digitación
    def fecha_nam(self, i):
        clave, fechad, edad = self.dataframe['FECHA_NAC'][i],
self.dataframe['FECHA_DEF'][i], self.dataframe['EDAD'][i]
        if type(clave) is str:
            if not self.isNaN(clave):
                dt2 = parse(fechad)
                return dt2 + relativedelta(years=-int(edad))
            else:
                dt, dt2 = parse(clave), parse(fechad)
                if dt > dt2:
                    dt = dt + relativedelta(years=-100)
                return dt.date()
        else:
            dt2 = parse(fechad)
            return dt2 + relativedelta(years=-int(edad))

    # Metodo encargado de la extraccion del nombre de un municipio con base en su
codigo
    def search_municipe(self, clave):
        for i in self.dataframe_lugar.index:
            if self.dataframe_lugar['CÓDIGO DANE DEL MUNICIPIO'][i] == clave:
                return self.dataframe_lugar['MUNICIPIO'][i]
```

```
        else:
            return "SIN NOMBRE"

    # Metodo para la extraccion de los datos de los departamentos en donde
ocurriendo las defunciones
    # en donde se crean tuplas con el codigo y nombre del departamento, para ser
almacenados en
    # la lista. Se controlan los duplicados con una lista auxiliar en donde se
almacenen los
    # codigos de los departamentos
    def extract_departament(self):
        lista, pk = [], []
        for i in self.dataframe.index:
            clave = self.dataframe['COD_DPTO'][i]
            if clave not in pk:
                pk.append(clave)
                aux = (clave, self.search_departament(clave))
                lista.append(aux)
        return lista

    # Metodo para la extraccion de los datos de las instituciones en donde
ocurriendo las defunciones
    # en donde se crean tuplas con el codigo y nombre de la institucion, para ser
almacenados en
    # la lista. Se controlan los duplicados con una lista auxiliar en donde se
almacenen los
    # codigos de las intituciones. Esta columna tienen valores vacios, que se
cnontrolar con el metodo isNaN
    def extract_institutes(self):
        lista, pk = [], []
        for i in self.dataframe.index:
            if not self.isNaN(self.dataframe['COD_INST'][i]):
                clave = str(self.dataframe['COD_INST'][i]).replace(';', ".")
                if clave not in pk:
                    pk.append(clave)
                    aux = (clave, self.dataframe['NOM_INST'][i])
                    lista.append(aux)
        return lista

    # Metodo para la extraccion de los datos de los departamentos en donde
ocurriendo las defunciones
    # en donde se crean tuplas con el codigo del municipio, nombre del municipio
y codigo del departamento, para ser almacenados en
    # la lista. Se controlan los duplicados con una lista auxiliar en donde se
almacenen los una clave compuesta por el codigo
    # del departamento y del municipio
    def extract_municipes(self):
        lista, pk = [], []
        for i in self.dataframe.index:
            clave = int(str(self.dataframe['COD_DPTO'][i]) +
(self.llenar(self.dataframe['COD_MUNIC'][i])))
            if clave not in pk:
                pk.append(clave)
                aux = (self.dataframe['COD_MUNIC'][i],
self.dataframe['COD_DPTO'][i], self.search_municipe(clave))
                lista.append(aux)
        return lista
```

```
    # Metodo para la extraccion de los lugares de defuncion de las personas, en
donde se crean tuplas con una clave compuesta
    # por el codigo del dep, municipio, y si ocurrio en una institucion, esta se
añade. Se realiza un control del valor
    # vacio de la institucion y de los egistros duplicados por medio de la clave
compuesta.
    def extract_place(self):
        lista, pk, ins = [], [], ""
        for i in self.dataframe.index:
            if not self.isNaN(self.dataframe['COD_INST'][i]):
                ins = str(self.dataframe['COD_INST'][i]).replace(';', ".")
                clave = str(self.dataframe['COD_DPTO'][i]) + "-" +
str(self.dataframe['COD_MUNIC'][i]) + "-" + ins
            else:
                ins = None
                clave = str(self.dataframe['COD_DPTO'][i]) + "-" +
str(self.dataframe['COD_MUNIC'][i])
            if clave not in pk:
                pk.append(clave)
                aux = (
                    clave, self.dataframe['SIT_DEFUN'][i],
self.dataframe['COD_DPTO'][i],
                    self.dataframe['COD_MUNIC'][i],
                    ins)
                lista.append(aux)
        return lista

    # Metodo para la extracion de las fechas de las defunciones registradas, en
donde se crean tuplas con la fecha completa,
    # el año, nombre del mes y día, se controlan los valores duplicados por medio
de una lista auxiliar con las fechas
    # completas
    def extract_date(self):
        l, pk = [], []
        for i in self.dataframe.index:
            fechad = self.dataframe['FECHA_DEF'][i]
            dt2 = parse(fechad).date()
            if dt2 not in pk:
                aux = (dt2, dt2.year, self.meses[dt2.month - 1], dt2.day)
                pk.append(dt2)
                l.append(aux)
        return l

    # Extraccion de los datos personas de los fallecidos, creacion tuplas con ña
fecha de nacimiento, edad, sexo y seguridad
    # social de la persona, además de una clave compuesta usando todos los datos
anteriomente expuestos
    def extract_person(self):
        l, pk = [], []
        for i in self.dataframe.index:
            f, edad, sexo, segsoc = self.fecha_nam(i), self.dataframe['EDAD'][i],
self.dataframe['SEXO'][i], \
                                        self.dataframe['SEG_SOCIAL'][i]
            clave = str(f.year) + str(f.month) + str(f.day) + "-" + str(edad) +
"-" + sexo[0] + "-" + segsoc
            if clave is not pk:
                pk.append(clave)
                aux = (clave, f, edad, sexo, segsoc)
```

```
                l.append(aux)
        return l

    # Metodo encargado de la extraccion de los datos principales del
fallecimiento de una persona, contemplando
    # la fecha, creacion de la clave compuesta para el lugar y la persona y la
cinluscion del nombre de la enfermedad
    # y el dato de si la persona recibio o no asistencia medica
    def extract_deaths(self, enf):
        lista, pk, lugar = [], [], ""
        for i in self.dataframe.index:
            if not self.isNaN(self.dataframe['COD_INST'][i]):
                ins = str(self.dataframe['COD_INST'][i]).replace(';', ".")
                lugar = str(self.dataframe['COD_DPTO'][i]) + "-" +
str(self.dataframe['COD_MUNIC'][i]) + "-" \
                        + ins
            else:
                lugar = str(self.dataframe['COD_DPTO'][i]) + "-" +
str(self.dataframe['COD_MUNIC'][i])
            fechad = self.dataframe['FECHA_DEF'][i]
            df = parse(fechad).date()
            f, edad, sexo, segsoc = self.fecha_nam(i), self.dataframe['EDAD'][i],
self.dataframe['SEXO'][i], \
                                    self.dataframe['SEG_SOCIAL'][i]
            persona = str(f.year) + str(f.month) + str(f.day) + "-" + str(edad) +
"-" + sexo[0] + "-" + segsoc
            am = str(self.dataframe['ASIS_MED'][i]).upper()
            if am == "IG" or am == "SÍ":
                am = "SI"
            aux = (lugar, df, persona, enf, am)
            lista.append(aux)
        return lista


# Clase encargada de las tranformaciones necesarias para la carga de los datos
extraidos de la ODS
# para el DW.
class Tranformaciones:

    # Constructora sin parametros
    def __init__(self):
        pass

    # Método encargado de la búsqueda de la posición de un item en una lista de
tuplas, de modo tal se comparar las
    # Posiciones de la tupla ingresada con las posiciones de cada una de las
tuplas de la lista. Se retorna la posición
    # de la tupla con el numero K de posiciones de la tupla. Se envía -1 si no se
encuentra la tupla en la lista
    def search_item(self, l, c, lim):
        x = 0
        for i in l:
            k = 0
            for j in range(len(i) - lim):
                if i[j] == c[j]:
                    k = k + 1
                else:
                    break
```

```
            if k == len(i) - lim:
                return x
            x = x + 1
        return -1

    # Metodo para la trabformacion de los valores de municipios en donde se crea
una tupla con una clase compuesta por
    # el cod del departamento y municipio (al igual que en el listado del DANE) y
se incluye el nombre del departamento
    # y municipio. No se controlan duplicados
    def transform_municipe(self, lista):
        l = []
        for i in lista:
            clave = str(i[1]) + "." + str(i[0])
            aux = (clave, str(i[3]), str(i[2]))
            l.append(aux)
        return l

    # Metodo encargado de la creación de las tuplas ordenadas de los daros
personales de los fallecidos
    def transform_person(self, lista):
        l = []
        for i in lista:
            aux = (i[0], i[3], i[1], i[2], i[4], i[5])
            l.append(aux)
        return l

    # Metodo encargado de la tranformación de las tuplas con los datos de los
lugares de fallecimiento de las personas
    # en donde se congirua la clave del municipio y se organizan los datos
extraidos del ODS
    def transform_place(self, lista):
        l = []
        for i in lista:
            clave = str(i[2]) + "." + str(i[3])
            aux = (i[0], clave, i[1], i[4])
            l.append(aux)
        return l

    # Metodo para la tranformacion de los datos de las defunciones por muncipio,
en donde se contempla la creacion de
    # listas con el codigo del municipio, fecha y enfermedad, en donde se
controlan los valores duplicados y se acunulan
    # los registros duplicados en la ultima posicion de cada lista.
    def transform_fact_def_mun(self, lista):
        l = []
        for i in lista:
            mun = str(i[2]) + "." + str(i[3])
            aux = [i[0], i[1], mun, 1]
            pos = self.search_item(l, aux, 1)
            if pos == -1:
                l.append(aux)
            else:
                a = l[pos]
                a[3] = a[3] + 1
                l[pos] = a
        return l
```

```
    # Metodo para la tranformación de los datos de defunciones por fecha, creando
listas con la fecha y enfermedad,
    # controlando los valores duplicados y acumulandolos en la ultima posicion de
la lista
    def transform_fact_def_fecha(self, lista):
        l = []
        for i in lista:
            aux = [i[0], i[1], 1]
            pos = self.search_item(l, aux, 1)
            if pos == -1:
                l.append(aux)
            else:
                a = l[pos]
                a[2] = a[2] + 1
                l[pos] = a
        return l

    # Metodo para la tranformacion de los datos de asistencia medica, creando
listas con los valores de fecha, muncipio y
    # enfermedad, controlando si la persona recibio o no asistencia medica en las
dos ultimas posiciones de cada lista.
    def transform_fact_asis_medica(self, lista):
        l = []
        for i in lista:
            mun = str(i[2]) + "." + str(i[3])
            aux = [i[0], mun, i[1], 0, 0]
            pos = self.search_item(l, aux, 2)
            if pos == -1:
                if i[4] == "SI":
                    aux[3] = aux[3] + 1
                else:
                    aux[4] = aux[4] + 1
                l.append(aux)
            else:
                a = l[pos]
                if i[4] == "SI":
                    a[3] = a[3] + 1
                else:
                    a[4] = a[4] + 1
                l[pos] = a
        return l


if __name__ == '__main__':
    # Creacion de la conexion con ods_bona_health
    ods = MySQLCon('localhost', 3306, 'root', 'password', 'ods_bona_health')
    if ods.condb is not None:
        # Creacion de los dataframes para los csv con los datos de mortalidad
        arc1, arc2, arc3 = ArchivoMort("Mort_Can_Mama.csv", "Cancer de Mama"),
ArchivoMort("Mort_Can_Pulmon.csv",

"Cancer de Pulmón"), ArchivoMort(
            "Mort_VIH.csv", "VIH")
        print("INICIO CARGA DE DATOS ODS", datetime.today().strftime('%Y-%m-%d
%H:%M:%S'))
        # Lista de enfermedades a cargar
        lista = [(arc3.enfermedad, "Virus de inmunodeficiencia humana"),
(arc1.enfermedad, ""), (arc2.enfermedad, "")]
```

```
        # carga de la tabla enfermedad del ODS
        ods.load("ods_bona_health.enfermedad (nombre,observacion) VALUES (%s,
%s)", lista, "ENFERMEDAD")
        # Ciclo for para la extracion de datos de los csv y su porterior carga al
ODS
        for i in range(7):
            if i == 0:
                lista = arc1.extract_departament() + arc2.extract_departament() +
arc3.extract_departament()
            elif i == 1:
                lista = arc1.extract_institutes() + arc2.extract_institutes() +
arc3.extract_institutes()
            elif i == 2:
                lista = arc1.extract_municipes() + arc2.extract_municipes() +
arc3.extract_municipes()
            elif i == 3:
                lista = arc1.extract_place() + arc2.extract_place() +
arc3.extract_place()
            elif i == 4:
                lista = arc1.extract_date() + arc2.extract_date() +
arc3.extract_date()
            elif i == 5:
                lista = arc1.extract_person() + arc2.extract_person() +
arc3.extract_person()
            elif i == 6:
                lista = arc1.extract_deaths("Cancer de Mama") +
arc2.extract_deaths(
                    "Cancer de Pulmón") + arc3.extract_deaths("VIH")
            # Carga de los datos a la tabla especificada para cada iteración
            ods.load(ods.insertsODS[i][0], lista, ods.insertsODS[i][1])
        print("FIN DE CARGA DE DATOS ODS", datetime.today().strftime('%Y-%m-%d
%H:%M:%S'))
        # Creacion de la conexion a bona_health_dw
        bonaheathl_dw = MySQLCon('localhost', 3306, 'root', 'password',
'bona_health_dw')
        if bonaheathl_dw.condb is not None:
            # Instancia de la clase con las tranformaciones
            tranform = Tranformaciones()
            print("INICIO DE CARGA DE DATOS A BONA_HEALTH_DW: ",
datetime.today().strftime('%Y-%m-%d %H:%M:%S'))
            # ciclo for para la extracion de los datos con las sentencias Select
para el ODS
            # se definen condiciones para los casos de tranformaciones requeridas
            for i in range(9):
                lista = ods.extract(ods.selectsODS[i])
                if i == 2:
                    lista = tranform.transform_municipe(lista)
                elif i == 3:
                    lista = tranform.transform_person(lista)
                elif i == 4:
                    lista = tranform.transform_place(lista)
                elif i == 5:
                    lista = tranform.transform_fact_def_mun(lista)
                elif i == 6:
                    lista = tranform.transform_fact_def_fecha(lista)
                elif i == 7:
                    lista = tranform.transform_fact_asis_medica(lista)
                # Carga de los datos a la tabla especificada para cada iteración
```

```
                bonaheathl_dw.load(bonaheathl_dw.insertBHDW[i][0], lista,
bonaheathl_dw.insertBHDW[i][1])
            # Cierre de las conexiones al ODs y DW
            bonaheathl_dw.closeconection()
            ods.closeconection()
            print("FIN DE CARGA DE DATOS A BONA_HEALTH_DW: ",
datetime.today().strftime('%Y-%m-%d %H:%M:%S'))
        else:
            print("NO SE PUDO CONECTAR A ", bonaheathl_dw.database)
    else:
        print("NO SE PUDO CONECTAR A ", ods.database)
```

El programa, el cual se desarrollo en Pycharm, genera la siguiente salida:

```
main
C:\Users\user\PycharmProjects\ETLBonaHealth\venv\Scripts\python.exe C:/Users/user/PycharmProjects/ETLBonaHealth/main.py
Conexión exitoso a ods_bona_health
INICIO CARGA DE DATOS ODS 2021-10-05 21:09:26
TABLA  ENFERMEDAD  CARGADA CON  3  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  DEPARTAMENTO  CARGADA CON  26  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  INSTITUCION  CARGADA CON  215  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  MUNICIPIO  CARGADA CON  95  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  LUGAR  CARGADA CON  250  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  FECHA  CARGADA CON  1705  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  PERSONA  CARGADA CON  1973  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  FALLECIMIENTO  CARGADA CON  1973  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
FIN DE CARGA DE DATOS ODS 2021-10-05 21:09:28
Conexión exitoso a bona_health_dw
INICIO DE CARGA DE DATOS A BONA_HEALTH_DW:  2021-10-05 21:09:28
TABLA  DIM FECHA  CARGADA CON  1388  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  DIM ENFERMEDAD  CARGADA CON  3  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  DIM MUNICIPIO  CARGADA CON  42  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  DIM PERSONA  CARGADA CON  1973  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  DIM LUGAR  CARGADA CON  133  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  FACT_DEFUN_POR_MUNICIPIO  CARGADA CON  1870  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  FACT_DEFUN_POR_FECHA  CARGADA CON  1705  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  FACT_ASIS_MEDICA  CARGADA CON  1870  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
TABLA  FACT_MUERTE  CARGADA CON  1973  REGISTROS (Los registros pueden ser repetidos y por ende haber sido ignroados)
Desconexión exitosa a bona_health_dw
Desconexión exitosa a ods_bona_health
FIN DE CARGA DE DATOS A BONA_HEALTH_DW:  2021-10-05 21:09:30
```

Ilustración 24 Salida en consola del programa ETL

Muestra de datos del Data Warehouse

A continuación, se mostrarán pantallazos los cuales evidencian que el Data Warehouse destinado para este proyecto se encuentra cargado, presentando las tablas de hechos por medio de PHPMyAdmin, en donde se evidencia por medio del color de los datos que estos están relacionados con las tablas de dimensiones:

Fact_asistencia_medica

✔ Mostrando filas 0 - 24 (total de 1870, La consulta tardó 0,0051 segundos.) [sum_sin_asistencia_med: 2... - 1...]

```
SELECT * FROM `fact_asis_medica` ORDER BY `sum_sin_asistencia_med` DESC
```

☐ Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

1 > >> | Número de filas: 25 Filtrar filas: Buscar en esta tabla Sort by key

+ Opciones

←T→			fecha	municipio	enfermedad	sum_asistencia_med	sum_sin_asistencia_med
☐ Editar	Copiar	Borrar	2012-04-10	17.1	Cancer de Mama	0	2
☐ Editar	Copiar	Borrar	2012-02-29	17.486	Cancer de Pulmón	0	2
☐ Editar	Copiar	Borrar	2014-07-02	17.1	Cancer de Pulmón	0	2
☐ Editar	Copiar	Borrar	2010-05-29	66.1	Cancer de Pulmón	0	1
☐ Editar	Copiar	Borrar	2010-11-25	17.1	VIH	0	1
☐ Editar	Copiar	Borrar	2010-10-08	17.873	Cancer de Pulmón	0	1
☐ Editar	Copiar	Borrar	2010-08-04	17.877	Cancer de Pulmón	0	1
☐ Editar	Copiar	Borrar	2011-01-02	17.1	Cancer de Mama	0	1
☐ Editar	Copiar	Borrar	2010-12-26	17.1	Cancer de Pulmón	0	1
☐ Editar	Copiar	Borrar	2010-06-06	17.1	Cancer de Pulmón	0	1
☐ Editar	Copiar	Borrar	2010-09-13	17.444	Cancer de Pulmón	0	1
☐ Editar	Copiar	Borrar	2010-05-10	17.380	Cancer de Pulmón	0	1
☐ Editar	Copiar	Borrar	2010-11-24	17.614	VIH	0	1

Ilustración 25 Muestra de datos Fact_asistencia_medica

Fact_defunciones_fecha

✔ Mostrando filas 0 - 24 (total de 1705, La consulta tardó 0,0014 segundos.)

```
SELECT * FROM `fact_defunciones_fecha`
```

☐ Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

1 > >> | Número de filas: 25 Filtrar filas: Buscar en esta tabla

+ Opciones

←T→			enfermedad	fecha	cantidad_fallecidos
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-01-08	3
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-01-13	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-01-16	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-01-17	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-01-18	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-01-28	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-01-30	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-02-04	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-02-19	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-02-25	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-03-20	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-03-28	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-04-02	1
☐ Editar	Copiar	Borrar	Cancer de Mama	2010-04-15	1

Ilustración 26 Muestra de datos Fact_defunciones_fecha

Fact_defunciones_municipio

✔ Mostrando filas 0 - 24 (total de 1870, La consulta tardó 0,0007 segundos.)

```
SELECT * FROM `fact_defun_por_municipio`
```

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

1 > >> | Número de filas: 25 Filtrar filas: Buscar en esta tabla Sort by key: Ningun

+ Opciones

←T→				fecha	enfermedad	municipio	cant_defun
☐	Editar	Copiar	Borrar	2010-01-01	Cancer de Pulmón	17.1	1
☐	Editar	Copiar	Borrar	2010-01-01	Cancer de Pulmón	17.13	1
☐	Editar	Copiar	Borrar	2010-01-04	Cancer de Pulmón	17.433	1
☐	Editar	Copiar	Borrar	2010-01-05	Cancer de Pulmón	17.1	2
☐	Editar	Copiar	Borrar	2010-01-05	VIH	17.1	1
☐	Editar	Copiar	Borrar	2010-01-07	Cancer de Pulmón	17.174	1
☐	Editar	Copiar	Borrar	2010-01-08	Cancer de Mama	17.1	2
☐	Editar	Copiar	Borrar	2010-01-08	Cancer de Mama	17.174	1
☐	Editar	Copiar	Borrar	2010-01-09	Cancer de Pulmón	17.1	1
☐	Editar	Copiar	Borrar	2010-01-09	VIH	17.1	1
☐	Editar	Copiar	Borrar	2010-01-11	Cancer de Pulmón	17.1	1
☐	Editar	Copiar	Borrar	2010-01-11	VIH	17.1	1
☐	Editar	Copiar	Borrar	2010-01-13	Cancer de Mama	17.1	1
☐	Editar	Copiar	Borrar	2010-01-15	Cancer de Pulmón	17.1	1

Ilustración 27 Muestra de datos fact_defunciones_municipio

Fact_muertes

✔ Mostrando filas 0 - 24 (total de 1973, La consulta tardó 0,0015 segundos.)

```
SELECT * FROM `fact_muerte`
```

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

1 > >> | Número de filas: 25 Filtrar filas: Buscar en esta tabla Sort by key: Ninguna

+ Opciones

←T→				persona	lugar	enfermedad	fecha_completa
☐	Editar	Copiar	Borrar	192611-84-M-Subsidiado	17-1	Cancer de Pulmón	2010-01-01
☐	Editar	Copiar	Borrar	194011-70-F-Contributivo	17-13	Cancer de Pulmón	2010-01-01
☐	Editar	Copiar	Borrar	195714-53-M-Subsidiado	17-433-174.330.052.301	Cancer de Pulmón	2010-01-04
☐	Editar	Copiar	Borrar	193115-79-F-Contributivo	17-1-170.010.048.801	Cancer de Pulmón	2010-01-05
☐	Editar	Copiar	Borrar	197515-35-M-Subsidiado	17-1-170.010.087.301	VIH	2010-01-05
☐	Editar	Copiar	Borrar	197715-33-M-No asegurado	17-1-170.010.081.707	Cancer de Pulmón	2010-01-05
☐	Editar	Copiar	Borrar	192017-90-F-Contributivo	17-174	Cancer de Pulmón	2010-01-07
☐	Editar	Copiar	Borrar	191318-97-F-Subsidiado	17-174	Cancer de Mama	2010-01-08
☐	Editar	Copiar	Borrar	195218-58-F-Contributivo	17-1	Cancer de Mama	2010-01-08
☐	Editar	Copiar	Borrar	195218-58-F-Subsidiado	17-1-170.010.005.301	Cancer de Mama	2010-01-08
☐	Editar	Copiar	Borrar	194419-66-M-Subsidiado	17-1	Cancer de Pulmón	2010-01-09
☐	Editar	Copiar	Borrar	195319-57-F-No asegurado	17-1-170.010.087.301	VIH	2010-01-09
☐	Editar	Copiar	Borrar	1941111-69-F-Contributivo	17-1-170.010.089.201	Cancer de Pulmón	2010-01-11
☐	Editar	Copiar	Borrar	1981111-29-M-No asegurado	17-1-170.010.081.702	VIH	2010-01-11

Ilustración 28 Muestra de datos fact_muertes

CREACIÓN Y PUBLICACIÓN DEL CUBO

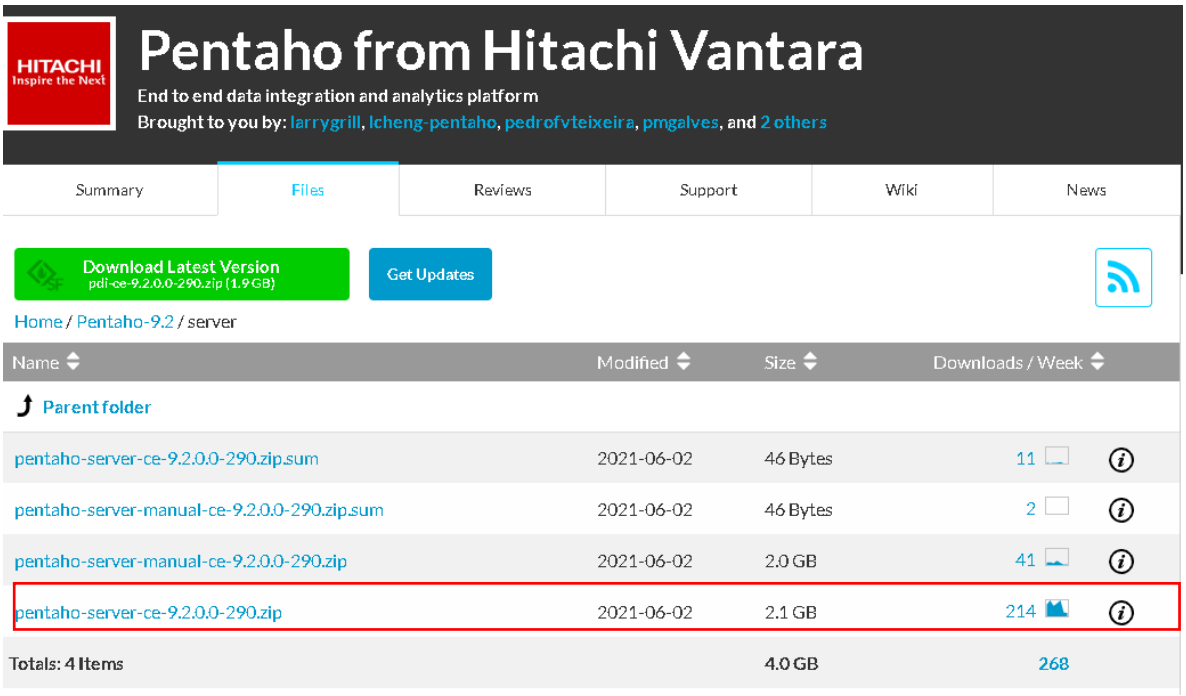
En esta sección se explicara todo el proceso realizado para el despliegue de un cubo OLAP basado en el Data Warehouse creado previamente.

Entorno de trabajo

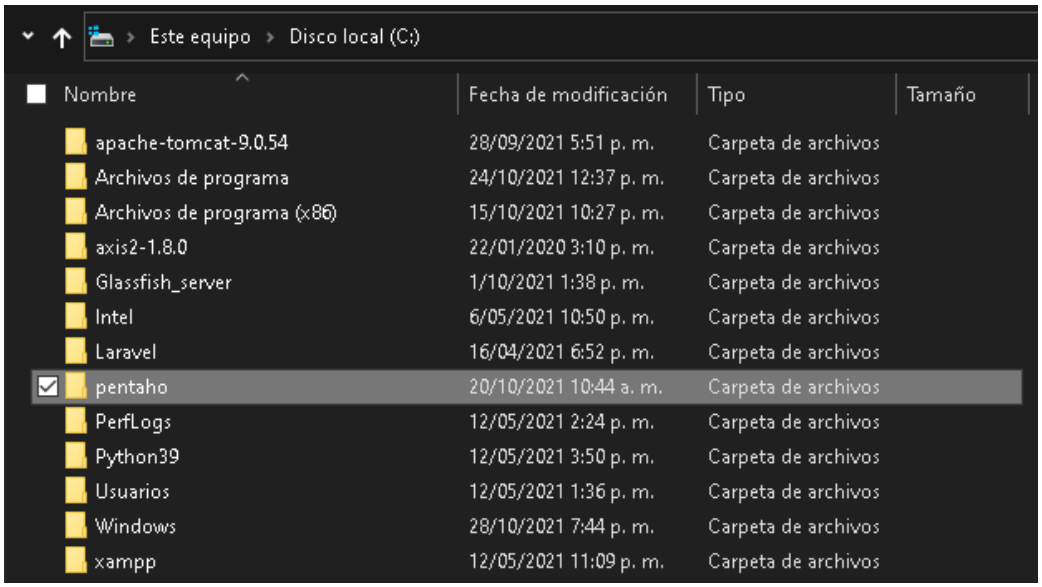
Para la realización de esta actividad, se emplearon herramientas de software presentadas por Pentaho, las cuales están destinadas para la gestión de los cubos, y como parte de esta actividad, se mostrará el proceso de descarga, despliegue y configuración de dichas herramientas.

Pentaho Server

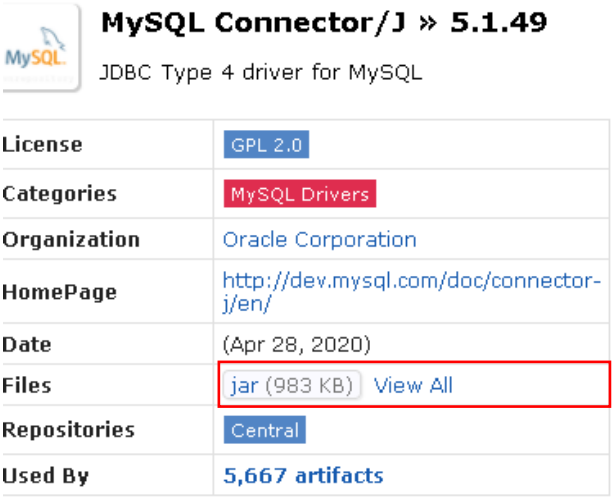
Esta herramienta nos permitirá la gestión de los cubos publicados y la manipulación de ciertas consultas. Esta herramienta esta desarrollada en Java y será manipulada por medio de un cliente al que se tendrá acceso por medio del navegador web. Para la descarga de esta herramienta, se debe acceder a la pagina de Source Force en la cual se encuentran los archivos destinados para la instalación de estas herramientas, en el caso de Pentaho Server, se debe acceder al siguiente enlace el cual descargara la herramienta en formato zip. (https://sourceforge.net/projects/pentaho/files/Pentaho-9.2/server/pentaho-server-ce-9.2.0.0-290.zip/download)



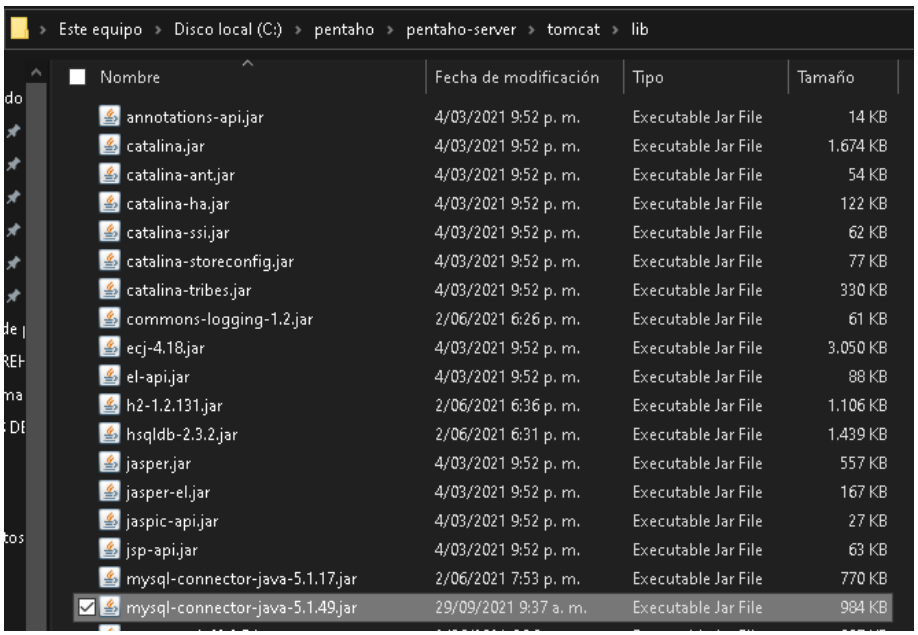
Con este archivo comprimido descargado, se creará una carpeta en el directorio raíz del disco duro principal en la cual se almacenaran todas las herramientas de Pentaho, en este caso, se creo una carpeta llamada Pentaho y se descomprimirá el archivo dentro de esta carpeta:



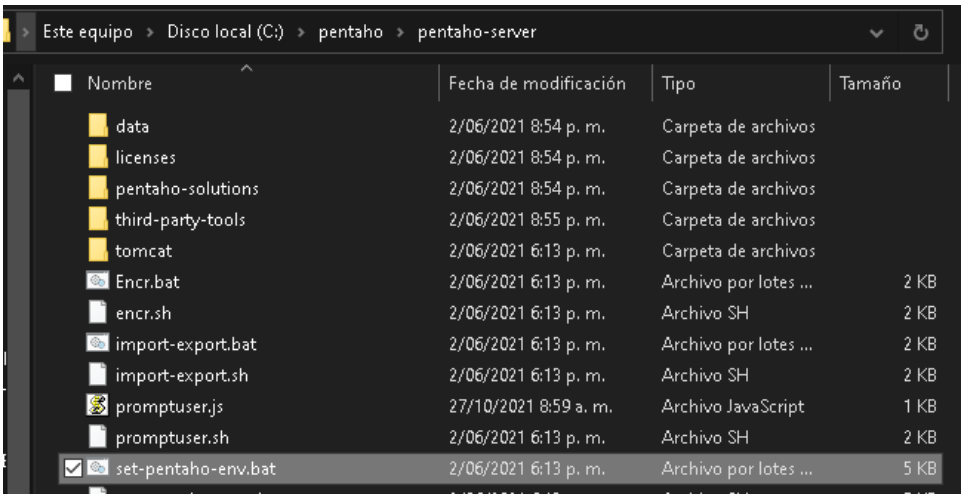
Para asegurar que la herramienta se conecte apropiadamente a MySQL y la base de datos correspondiente al Data Warehouse, se requiere contar con el conector en formato Jar, en este caso, se empleó al archivo mysql-connector-java-5.1.49 descargado desde Maven Repository en el siguiente enlace: https://repo1.maven.org/maven2/mysql/mysql-connector-java/5.1.49/mysql-connector-java-5.1.49.jar



Con este archivo descargado, se debe incluir en la carpeta Tomcat/ Lib dentro de Pentaho Server. Este conectar se empleará e la adecuación de las siguientes herramientas.



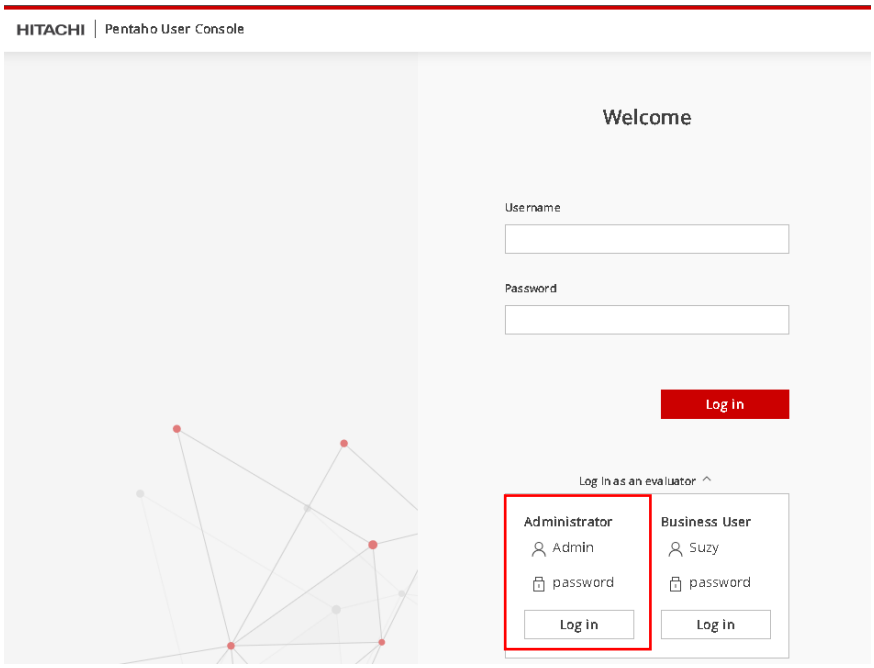
Antes de iniciar el servidor, se requiere contar con las variables de entorno, para esto, nos dirigimos a la carpeta principal de Pentaho Server y se ejecuto el archivo **set-pentaho-env.bat** el cual se encargará de implementar las variables de entorno requeridas.



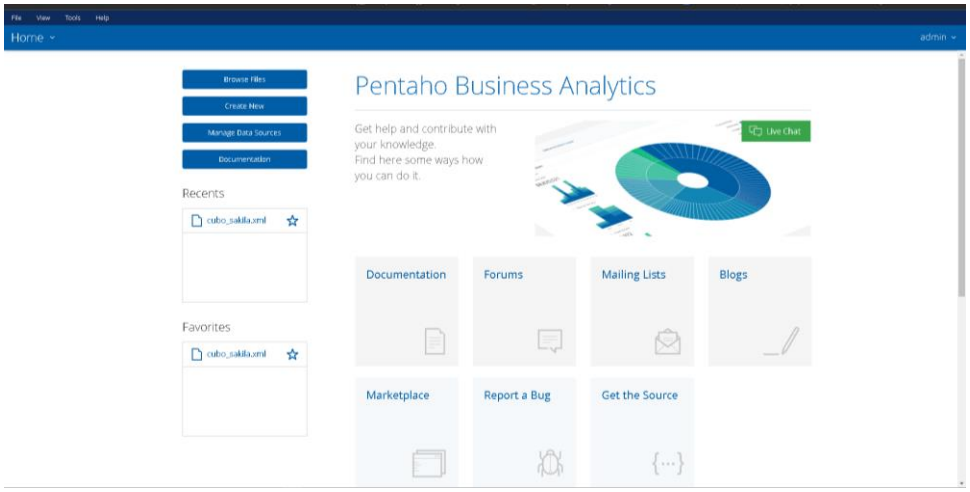
Finalmente, para iniciar el servidor y teniendo en cuenta todas las actividades previas para la configuración del entorno de este, se ejecutará el archivo **start-**

pentaho.bat el cual desplegará una instancia de la consola en la cual se podrá evidenciar el inicio del servidor. Este proceso es demorado, pero hay que prestar atención a como este se comporta y las posibles alertas de error en su ejecución. Verifique que no cuente con otro servicio que este operando en el puerto 8080.

Para acceder al servidor, solo se requiere ingresar al siguiente enlace http://localhost:8080/pentaho/Login en el cual se encuentra el Login del servidor, y se elegirán las credenciales de acceso como administrador.



Finalmente, nos encontraremos en el Dashboard del servidor, lo cual indica, hasta el momento, que el proceso de despliegue de la herramienta se realizó correctamente. Se recomienda acceder al servidor solo después de que, en la ventana de consola, nos presente el mensaje **org.apache.catalina.startup.Catalina.start Server startup in [n] milliseconds.**

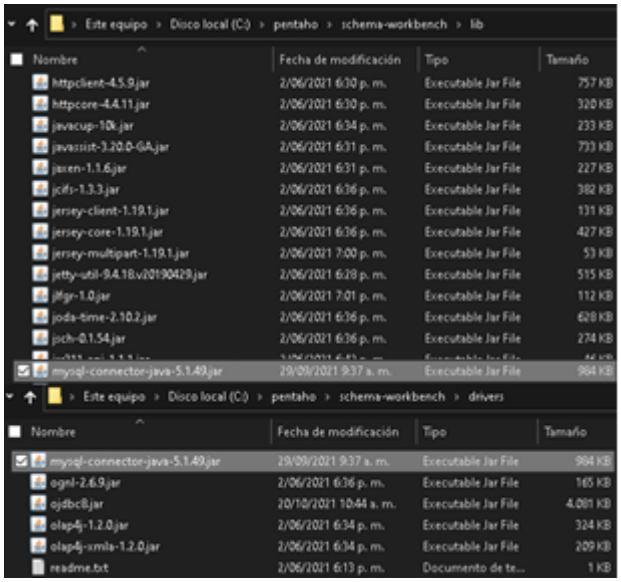


Pentaho Schema Workbench

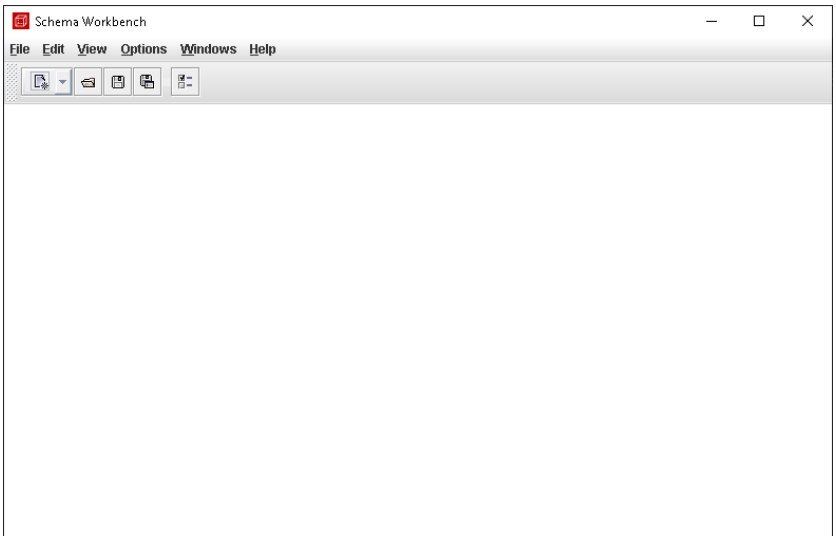
Esta herramienta de Pentaho será la encarga de permitirnos el modelado de los cubos a publicar en el servidor, por lo que esta herramienta requiere de la conexión con la base de datos correspondiente al Data Warehouse. Para la descarga de esta herramienta, nos dirigimos a la página correspondiente a las herramientas de Pentaho y descargamos el archivo comprimido desde el siguiente enlace https://sourceforge.net/projects/pentaho/files/Pentaho-9.2/client-tools/psw-ce-9.2.0.0-290.zip/download .

pdi-ce-9.2.0.0-290.zip	2021-06-02	1.9 GB	6,484	
pme-ce-9.2.0.0-290.zip	2021-06-02	1.7 GB	25	
pad-ce-9.2.0.0-290.zip	2021-06-02	28.4 MB	24	
prd-ce-9.2.0.0-290.zip	2021-06-02	1.7 GB	78	
pentaho-big-data-plugin-9.2.0.0-290.zip	2021-06-02	520.4 MB	28	
psw-ce-9.2.0.0-290.zip	2021-06-02	26.4 MB	65	

El archivo comprimido descargado también se debe de descomprimir en la carpeta Pentaho previamente creada en el directorio raíz del disco duro principal, y como actividad final para el adecuamiento de la herramienta, se debe incluir el conector de MySQL dentro de las carpetas lib y drivers.



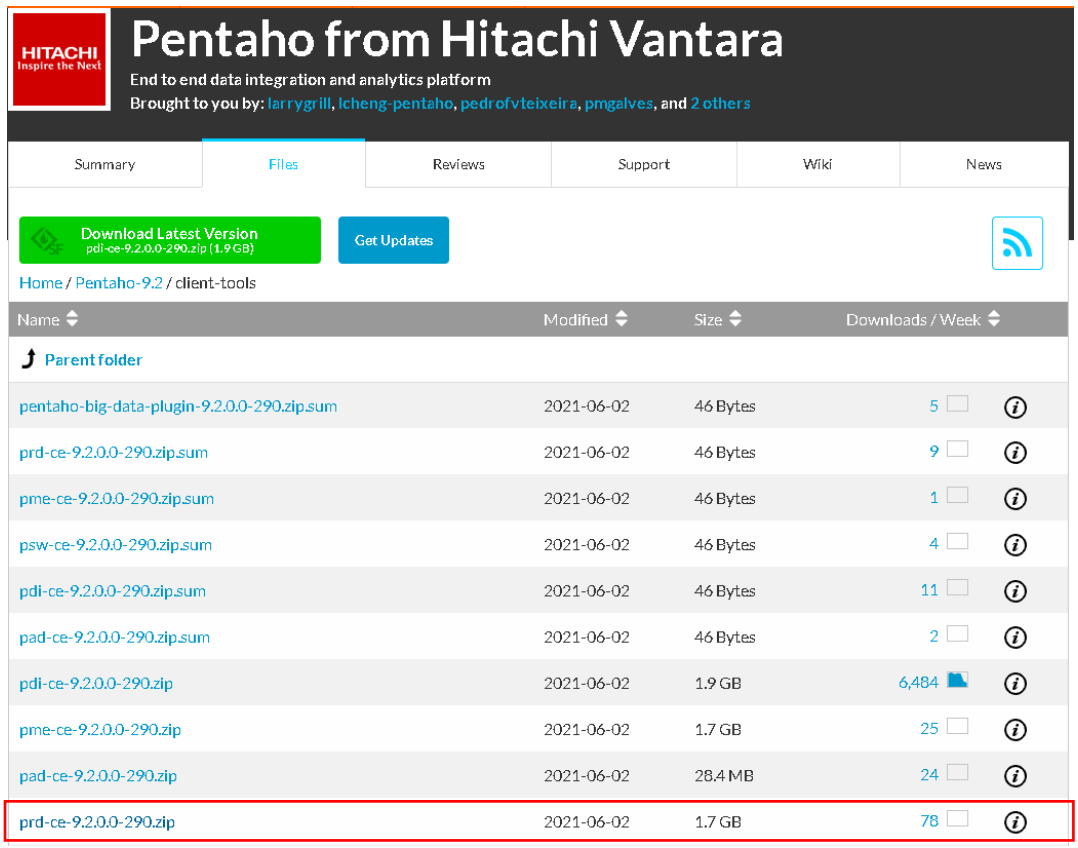
Finalmente, para iniciar la herramienta, nos dirigimos al directorio principal de Schema Workbench y ejecutaremos el archivo **workbench.bat**, el cual iniciara el cliente grafico de la herramienta y una ventana de conola de comandos de Windows, la cual no se debe cerrar mientras la herramienta se tenga abierta.



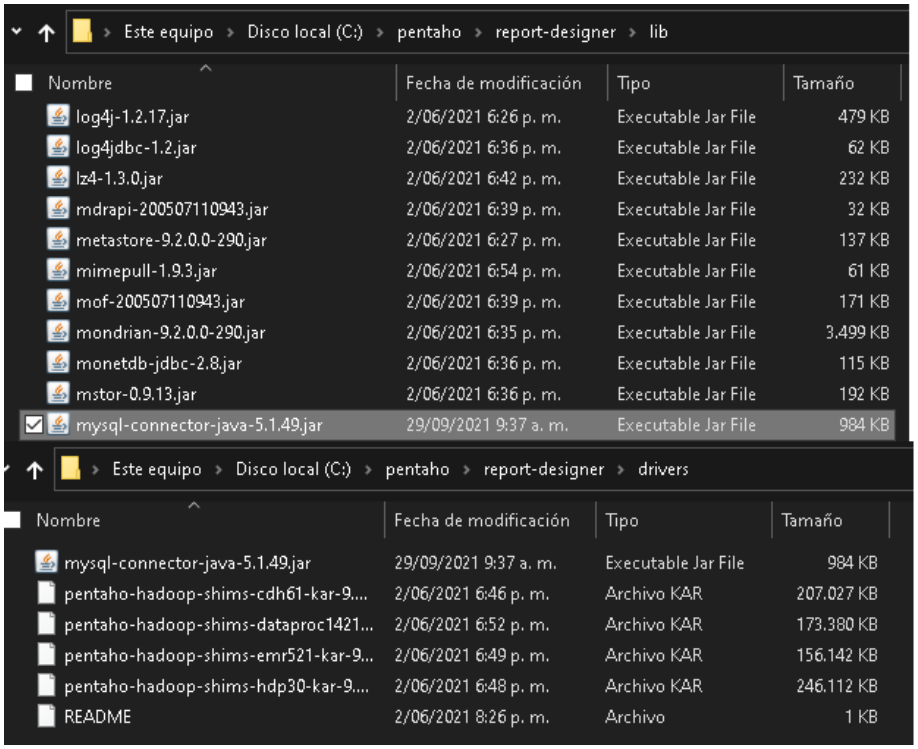
Pentaho Report Desginer

Esta herramienta nos permitirá realizar las actividades correspondientes al diseño de los reportes requeridos por la empres por medio de la creación de consultas y representaciones graficas de la información requerida. Para descargar esta herramienta, nos dirigiremos a la pagina de SourceForge de donde hemos descargado los anteriores recursos y descargaremos el archivo pdr-ce-

9.2.0.0.290.zip del siguiente enlace https://sourceforge.net/projects/pentaho/files/Pentaho-9.2/client-tools/prd-ce-9.2.0.0-290.zip/download .



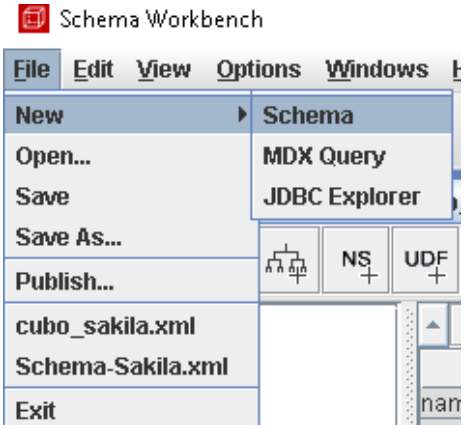
Con el archivo comprimido descargado, lo descomprimiremos dentro de la carpeta Pentaho en el directorio raíz, y como actividad final para el adecuamiento de la herramienta, se debe incluir el conector de MySQL dentro de las carpetas lib y drivers.



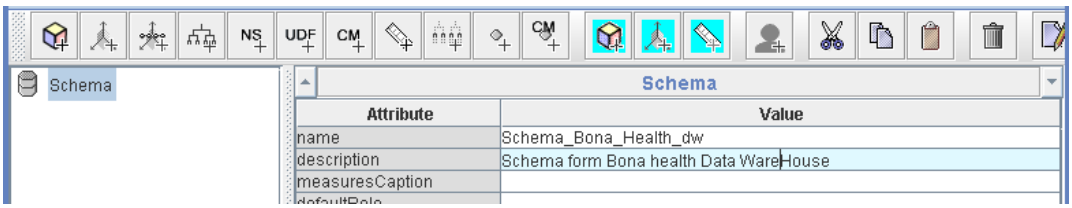
Con estas adecuaciones. La herramienta estaría lista para su uso posteriormente.

Creación del cubo

Para la creación del cubo se empleará la herramienta Schema Workbech, y se realizara con cada una de las tablas de hechos definidas, de modo tal se crearán las dimensiones asacadas a cada una de estas con la jerarquía que las compone. Primero, se creará el esquema en el cual se contendrán los cubos asociados a las tablas de hechos, para esto se dará clic en File/ New / Schema.

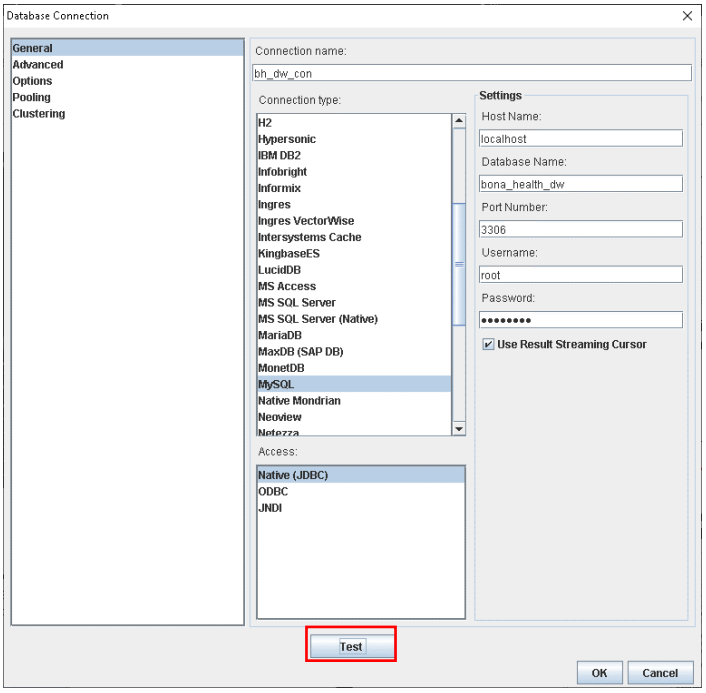


Con el esquema creado, se le dará un nombre a este, para esto se le da clic el Item **Schema** en la columna izquierda y en la ventana central, se ingresará el nombre del esquema y una descripción a este.



Luego, se requiere configurar la conexión con la base de datos con el fin de modelar cada uno de los cubos, para esto, no0s dirigiremos a Options / Connections y creamos una conexión con los siguientes datos:

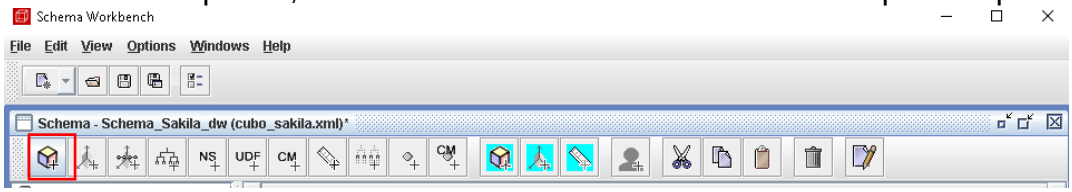
- Connection Name: bh_dw_con
- Connection Type: MySQL
- Access: Native (JDBC)
- Hostname: localhost o dirección IP del equipo con la base datos.
- Database Name: bona_health_dw
- Port Number: 3306
- Username: root
- Password: contraseña del usuario root
- Use Result Streaming Cursor: debe estar marcado.



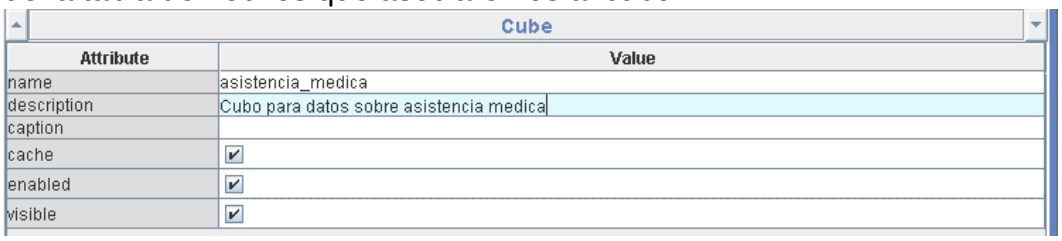
Con estos datos definidos, se realiza una prueba de la conexión a la base de datos correspondiente al Data Warehouse dando clic al botón test y debemos obtener la siguiente respuesta si ingresamos todos los datos correctamente. De lo contrario, por favor revisar los datos correspondientes a la base de datos en su equipo.

Para la creación de cad auno de los cubos se deben seguir los siguientes pasos:

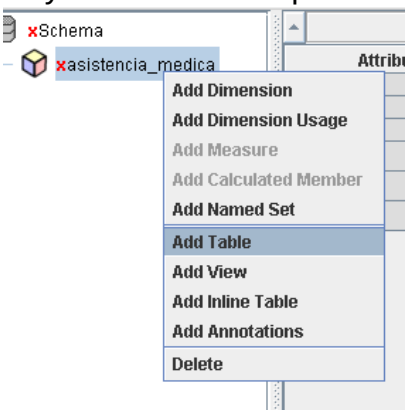
1. Dentro del esquema, dar clic en el icono de cubo ubicado en la parte superior:



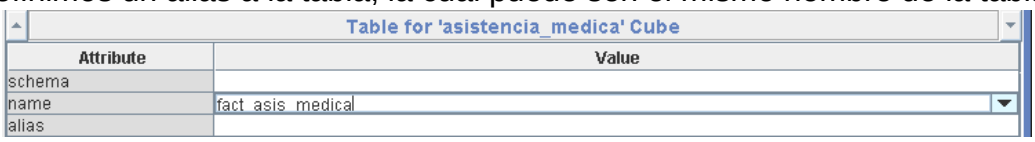
2. En la ventana desplegada, definir el nombre del cubo el cual sera el nombre de la tabla de hechos que asociaremos al cubo.



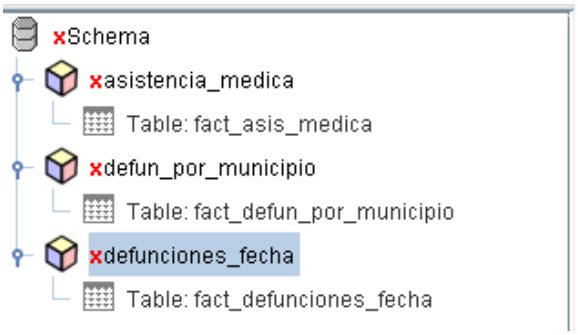
3. Para asociar al cubo recien creado a su tabla de hechos correspondiente, dar clic derecho sobre el cubo y seleccionar la opcion **Add Table**.



En la ventana lateral, se define la tabla, para esto damos clic en la lista desplegable de name y seleccionamos el nombre de la tabla de hechos y definimos un alias a la tabla, la cual puede serl el mismo nombre de la tabla.



Con estos pasos definidos, se realizo la creación de cada uno de los cubos requeridos, mostrabdo acontinaucón la evidencia de estos, teniendo en cuenta que los erores que se reportan es debido a que no existen dimenciones aun en n ingun cubo.

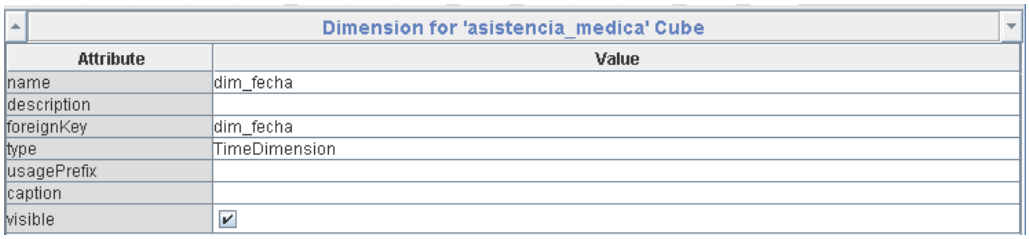


A continuación se mostrara la creación de cada una de las dimenciones, esto teniendo en cuenta que el proceso de creacion de las dimenciones es repetitivo y que solo se debe de replicar con base a las dimenciones involucradas en cada uno de los cubos.

Dim_fecha

Para la creación de esta dimensión, en cada cubo se dará clic derecho sobre el cubo y se seleccionará la opción **Add Dimension**, en la ventana que se nos genera, definiremos los siguientes datos para esta dimensión:

- **Name**: dim_fecha
- **ForeingKey**: dim_fecha
- **Type**: TimeDimesion



Con estos datos definidos, se creará la jerarquía correspondiente para esta dimensión, para esto, daremos clic derecho sobre la dimensión previamente creada y seleccionaremos la opción **Add Herarchy,** en la ventana que se nos genera definiremos los siguientes datos para la jerarquía:

- **Name**: fecha
- **PrimaryKey**: dim_fecha – fecha_completa

Hierarchy for 'dim_fecha' Dimension

Attribute	Value
name	fecha
description	
hasAll	☑
allMemberName	
allMemberCaption	
allLevelName	
defaultMember	
memberReaderClass	
primaryKeyTable	
primaryKey	fecha_completa
caption	
visible	☑

Con la jerarquía creada, se debe asociar la tabla correspondiente a esta, por lo que daremos clic derecho sobre la jerarquía y seleccionaremos la opción **Add Table**, en la ventana que se nos genera definiremos el dato name como dim_fecha.

Table for 'fecha' Hierarchy

Attribute	Value
schema	
name	dim_fecha
alias	

Nota: si Schema Workbench marca algún error en la creación de la jearquia con sus datos o no permite ingresar la Primary Key, se requiere crear la jerarquía vacía y luego añadirle la tabla, esto con el fin de poder acceder apropiadamente a los atributos de la tabla en cuestión.

A continuación, se crearán los miembros que la conforman esta jerarquía desde el mayor nivel, en este caso el año, hasta el menor nivel, que sería la fecha completa, a continuación, se mostrara la creación de los niveles de esta jerarquía:

- Año: para este nivel se definen los siguientes datos:
  - Name: anio
  - Table: dim_fecha*
  - Column: anio
  - NameColumn: anio
  - Type: Numeric
  - LevelType: TimeYears

Attribute	Value
name	anio
description	
table	dim_fecha
column	anio
nameColumn	anio
parentColumn	
nullParentValue	
ordinalColumn	
type	Numeric
internalType	
uniqueMembers	☐
levelType	TimeYears
hideMemberIf	Never
approxRowCount	
caption	
captionColumn	
formatter	
visible	☑

- Mes: para este nivel se definen los siguientes datos:
  - Name: mes
  - Table: dim_fecha*
  - Column: mes
  - NameColumn: mes
  - Type: String
  - LevelType: TimeMonths

Attribute	Value
name	mes
description	
table	dim_fecha
column	mes
nameColumn	
parentColumn	
nullParentValue	
ordinalColumn	
type	String
internalType	
uniqueMembers	☐
levelType	TimeMonths
hideMemberIf	Never
approxRowCount	
caption	
captionColumn	
formatter	
visible	☑

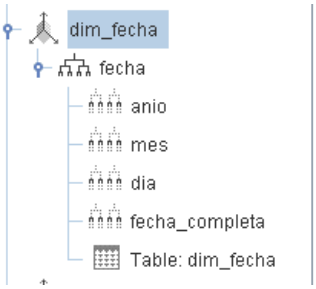
- Dia: para este nivel se definen los siguientes datos:
  - Name: día
  - Table: dim_fecha*
  - Column:dia
  - NameColumn: dia
  - Type: Numeric
  - LevelType: TimeDays

Attribute	Value
name	dia
description	
table	dim_fecha
column	dia
nameColumn	dia
parentColumn	
nullParentValue	
ordinalColumn	
type	Numeric
internalType	
uniqueMembers	☐
levelType	TimeDays
hideMemberIf	Never
approxRowCount	
caption	
captionColumn	
formatter	
visible	☑

- Fecha_completa: para este nivel se definen los siguientes datos:
    - Name: fecha_completa
    - Table: dim_fecha*
    - Column: fecha_completa
    - NameColumn: fecha_completa
    - Type: Time
    - LevelType: TimeDays

Attribute	Value
name	fecha_completa
description	
table	dim_fecha
column	fecha_completa
nameColumn	fecha_completa
parentColumn	
nullParentValue	
ordinalColumn	
type	Time
internalType	
uniqueMembers	☐
levelType	TimeDays
hideMemberIf	Never
approxRowCount	
caption	
captionColumn	
formatter	
visible	☑

Con la dimensión, su respectiva jerarquía definida y los niveles de esta declarados, sin olvidar la asociación de esta jerarquía a la correspondiente tabla dentro del Data Warehouse de la dimensión, se obtiene la siguiente estructura para la dimensión de fecha:



Dim_Municipio

Para la creación de esta dimensión, en cada cubo se dará clic derecho sobre el cubo y se seleccionará la opción **Add Dimension**, en la ventana que se nos genera, definiremos los siguientes datos para esta dimensión:

- **Name**: dim_municipio
- **ForeingKey**: dim_municipio
- **Type**: StandardDimesion

Dimension for 'asistencia_medica' Cube

Attribute	Value
name	dim_municipio
description	
foreignKey	dim_municipio
type	StandardDimension
usagePrefix	
caption	
visible	☑

Con estos datos definidos, se creará la jerarquía correspondiente para esta dimensión, para esto, daremos clic derecho sobre la dimensión previamente creada y seleccionaremos la opción **Add Herarchy,** en la ventana que se nos genera definiremos los siguientes datos para la jerarquía:

- **Name**: municipio
- **PrimaryKey**: dim_municipio – id_municipio

Attribute	Value
name	municipio
description	
hasAll	☑
allMemberName	
allMemberCaption	
allLevelName	
defaultMember	
memberReaderClass	
primaryKeyTable	
primaryKey	id_municipio
caption	
visible	☑

Con la jerarquía creada, se debe asociar la tabla correspondiente a esta, por lo que daremos clic derecho sobre la jerarquía y seleccionaremos la opción **Add Table**, en la ventana que se nos genera definiremos el dato name como dim_municipio.

Table for 'municipio' Hierarchy	
Attribute	**Value**
schema	
name	dim_municipio
alias	

A continuación, se crearán los miembros que la conforman esta jerarquía desde el mayor nivel, en este caso el departamento, hasta el menor nivel, que sería id_municipio, a continuación, se mostrara la creación de los niveles de esta jerarquía:

- Departamento: para este nivel se definen los siguientes datos:
  - Name: departamento
  - Table: dim_municipio
  - Column: departamento
  - NameColumn: departamento
  - Type: String
  - LevelType: Regular

Attribute	Value
name	departamento
description	
table	dim_municipio
column	departamento
nameColumn	departamento
parentColumn	
nullParentValue	
ordinalColumn	
type	String
internalType	
uniqueMembers	☐
levelType	Regular
hideMemberIf	Never
approxRowCount	
caption	
captionColumn	
formatter	
visible	☑

- Municipio: para este nivel se definen los siguientes datos:
  - Name: municipio
  - Table: dim_municipio
  - Column: municipio
  - NameColumn: municipio
  - Type: String
  - LevelType: Regular

Attribute	Value
name	municipio
description	
table	dim_municipio
column	municipio
nameColumn	municipio
parentColumn	
nullParentValue	
ordinalColumn	
type	String
internalType	
uniqueMembers	☐
levelType	Regular
hideMemberIf	Never
approxRowCount	
caption	
captionColumn	
formatter	
visible	☑

- Id_ municipio: para este nivel se definen los siguientes datos:
  - Name: id_ municipio
  - Table: dim_ municipio
  - Column: id_ municipio
  - NameColumn: id_ municipio
  - Type: String
  - LevelType: Regular

Attribute	Value
name	id_municipio
description	
table	dim_municipio
column	id_municipio
nameColumn	id_municipio
parentColumn	
nullParentValue	
ordinalColumn	
type	String
internalType	
uniqueMembers	☐
levelType	Regular
hideMemberIf	Never
approxRowCount	
caption	
captionColumn	
formatter	
visible	☑

Con la dimensión, su respectiva jerarquía definida y los niveles de esta declarados, sin olvidar la asociación de esta jerarquía a la correspondiente tabla dentro del Data Warehouse de la dimensión, se obtiene la siguiente estructura para la dimensión de municipio:

Dim_Enfermedad

Para la creación de esta dimensión, en cada cubo se dará clic derecho sobre el cubo y se seleccionará la opción **Add Dimension**, en la ventana que se nos genera, definiremos los siguientes datos para esta dimensión:

- **Name**: dim_enfermedad
- **ForeingKey**: dim_enfermedad
- **Type**: StandardDimesion

Attribute	Value
name	dim_enfermedad
description	
foreignKey	dim_enfermedad
type	StandardDimension
usagePrefix	
caption	
visible	☑

Con estos datos definidos, se creará la jerarquía correspondiente para esta dimensión, para esto, daremos clic derecho sobre la dimensión previamente creada y seleccionaremos la opción **Add Herarchy,** en la ventana que se nos genera definiremos los siguientes datos para la jerarquía:

- **Name**: enfermedad
- **PrimaryKey**: dim_enfermedad – nombre

Attribute	Value
name	enfermedad
description	
hasAll	☑
allMemberName	
allMemberCaption	
allLevelName	
defaultMember	
memberReaderClass	
primaryKeyTable	
primaryKey	nombre
caption	
visible	☑

Con la jerarquía creada, se debe asociar la tabla correspondiente a esta, por lo que daremos clic derecho sobre la jerarquía y seleccionaremos la opción **Add Table**, en la ventana que se nos genera definiremos el dato name como dim_enfermedad.

Table for 'enfermedad' Hierarchy	
Attribute	**Value**
schema	
name	dim_enfermedad
alias	

A continuación, se crearán los miembros que la conforman esta jerarquía desde el mayor nivel, en este caso la descripción, hasta el menor nivel, que sería el nombre de la enfermedad, a continuación, se mostrara la creación de los niveles de esta jerarquía:

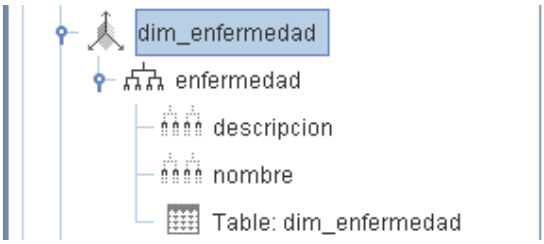
- Descripción: para este nivel se definen los siguientes datos:
  - Name: descripción
  - Table: dim_enfermedad
  - Column: descrip
  - NameColumn: descrip
  - Type: String
  - LevelType: regular

Attribute	Value
name	descripcion
description	
table	dim_enfermedad
column	descript
nameColumn	descript
parentColumn	
nullParentValue	
ordinalColumn	
type	String
internalType	
uniqueMembers	☐
levelType	Regular
hideMemberIf	Never
approxRowCount	
caption	
captionColumn	
formatter	
visible	☑

- Nombre: para este nivel se definen los siguientes datos:
  - Name: nombre
  - Table: dim_enfermedad
  - Column: nombre
  - NameColumn: nombre
  - Type: String
  - LevelType: regular

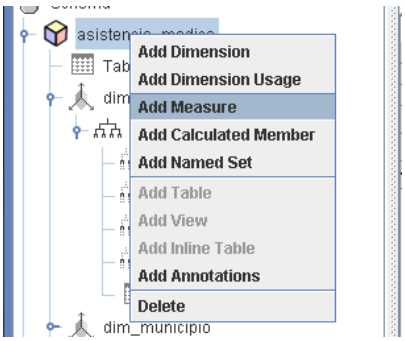
Attribute	Value
name	nombre
description	
table	dim_enfermedad
column	nombre
nameColumn	nombre
parentColumn	
nullParentValue	
ordinalColumn	
type	String
internalType	
uniqueMembers	☐
levelType	Regular
hideMemberIf	Never
approxRowCount	
caption	
captionColumn	
formatter	
visible	☑

Con la dimensión, su respectiva jerarquía definida y los niveles de esta declarados, sin olvidar la asociación de esta jerarquía a la correspondiente tabla dentro del Data Warehouse de la dimensión, se obtiene la siguiente estructura para la dimensión de enfermedad:



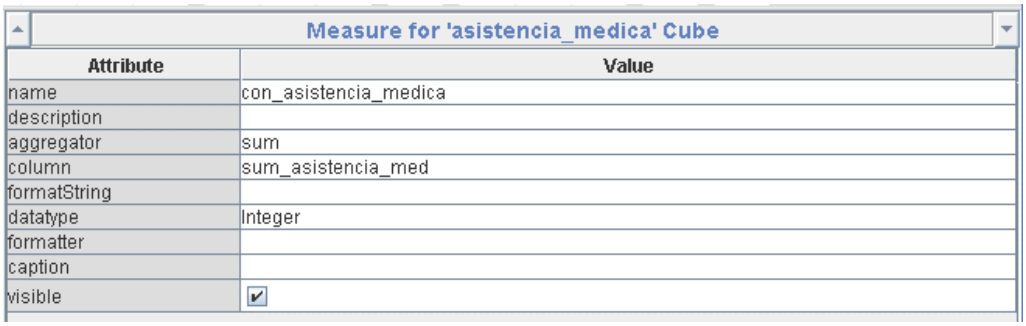
Ejemplo de creacion de una medida

Antes de continuar con la especificacion de cada uno de los cubos, se requiere tener presente la creacion de medidas para cada uno de estos, este proceso de realiza dando clic derecho sobre el cubo correspondiente y seleccionando la opcion **Add Measure.**



Al dar clic en esta, se nos generara la medida y una ventana en la cual definimos los siguientes datos, en este caso, se creara la medida de Con_asistencia_medica del cubo asistencia medica.

- Name: con_asistencia_medica
- Aggregator: sum
- Column: sum_asistencia_med
- Datatype: integer



Measure for 'asistencia_medica' Cube

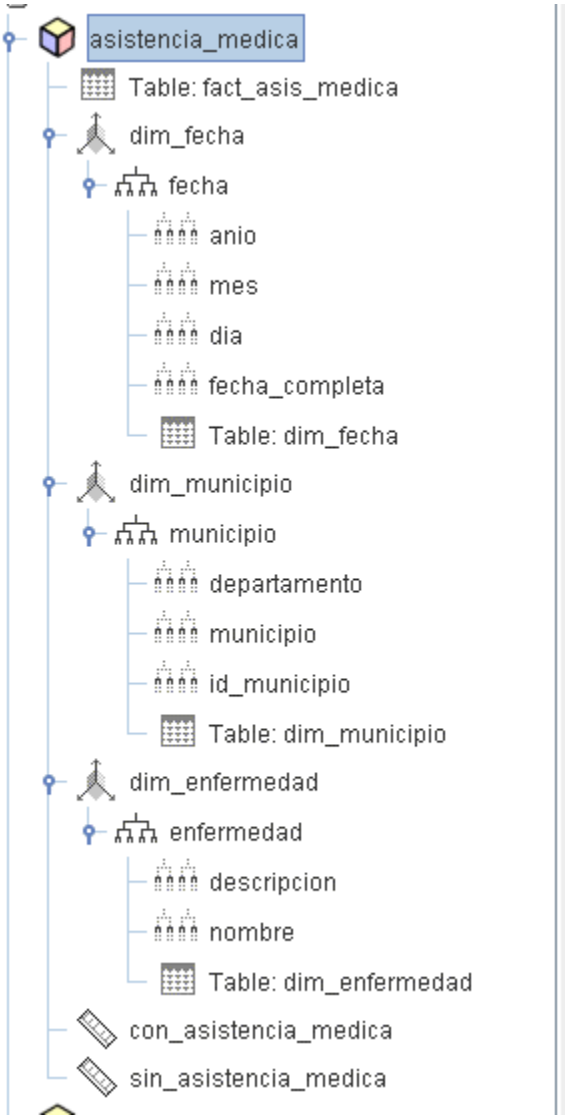
Attribute	Value
name	con_asistencia_medica
description	
aggregator	sum
column	sum_asistencia_med
formatString	
datatype	Integer
formatter	
caption	
visible	☑

Este proceso se repitira con cada una de las medidas de cada una de las tablas de hechos definidas en la base de datos.

Con las dimenciones previamente definidas, teniendo presente la forma de creacion de las medidas, se continuara a definir cada uno de los cubos resultantes, las dimenciones que lo componen y sus rectivias medidas.

Asistencia_medica

En este cubo se emplearon las dimensiones de fecha, municipio y enfermedad, empleando las medidas de fallecidos con y sin asistencia, obteniendo la siguiente estructura al finalizar el modelado del cubo.



Defun_por_municipio

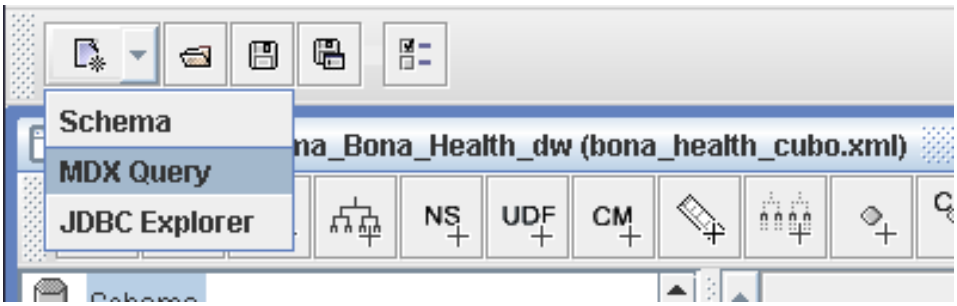
En este cubo se emplearon las dimensiones de fecha, municipio y enfermedad, empleando la medida de cantidad de defunciones registradas, obteniendo la siguiente estructura al finalizar el modelado del cubo.

Defunciones_fecha

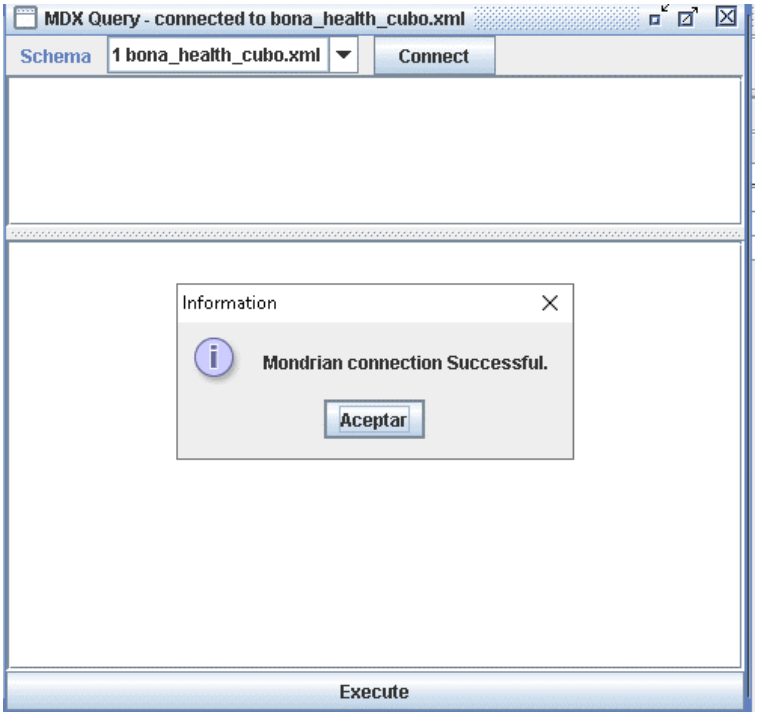
En este cubo se emplearon las dimensiones de fecha y enfermedad, empleando la medida de cantidad de defunciones, obteniendo la siguiente estructura al finalizar el modelado del cubo.

Prueba

Con todos los cubos definidos correctamente, para saber que estos no generen ningún conflicto, se realizara una prueba de ejecución de consulta MDX, para esto, nos dirigimos al icono de New y elegiremos la opción MDX Query.



Si el esquema logra sincronizarse correctamente, obtendremos el siguiente mensaje, de lo contrario, revisar la advertencia que el programa nos presente y realizar las modificaciones pertinentes.



Dentro de la ventana desplegada, se ejecutará la siguiente sentencia la cual retornará la cantidad de defunciones en el año 2010, si esta sentencia funciona correctamente, se podrá continuar con la publicación del cubo, de lo contrario, se realizarán las correcciones necesarias.

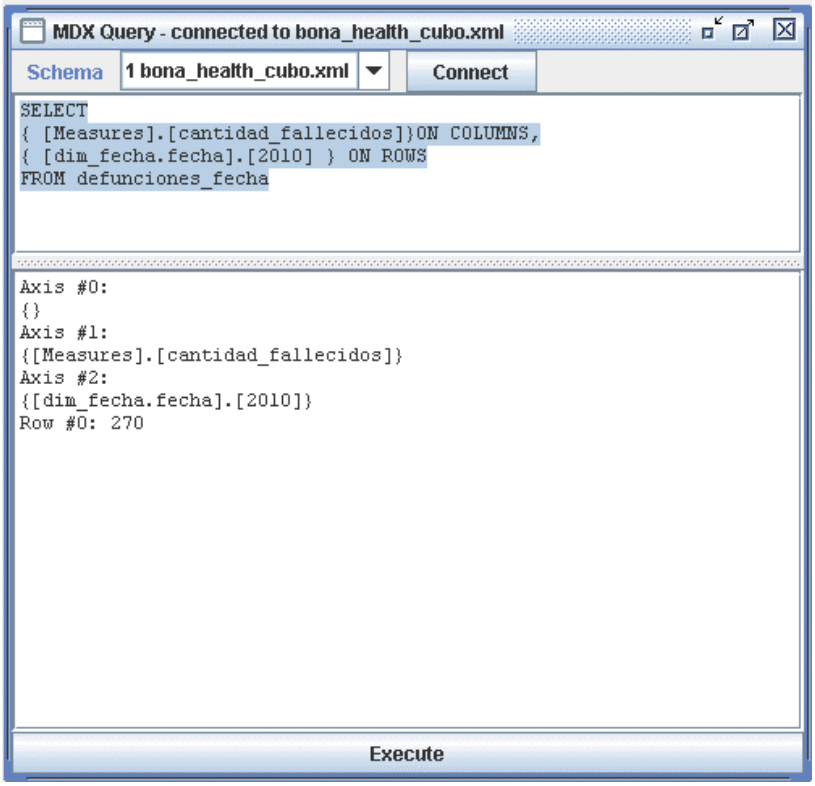
```
SELECT

{ [Measures].[cantidad_fallecidos]}ON COLUMNS,

{ [dim_fecha.fecha].[2010] } ON ROWS

FROM defunciones_fecha
```

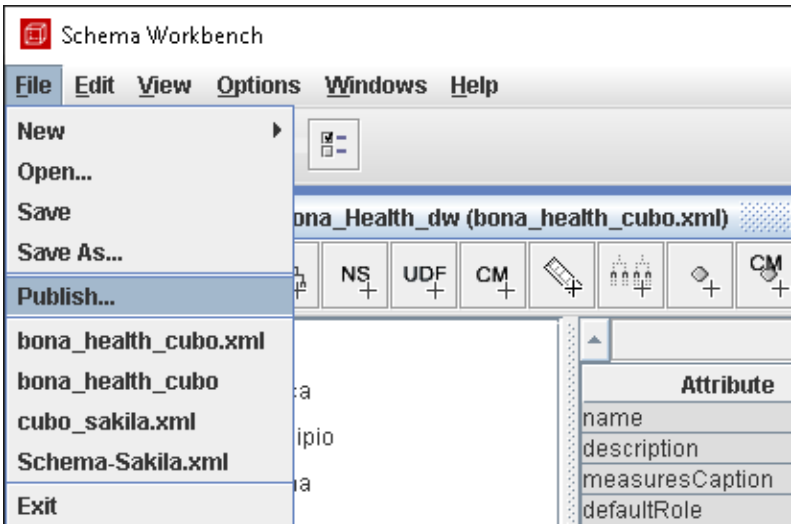
Al ejecutar esta sentencia se obtuvo una respuesta positiva al retornar los valores esperados:



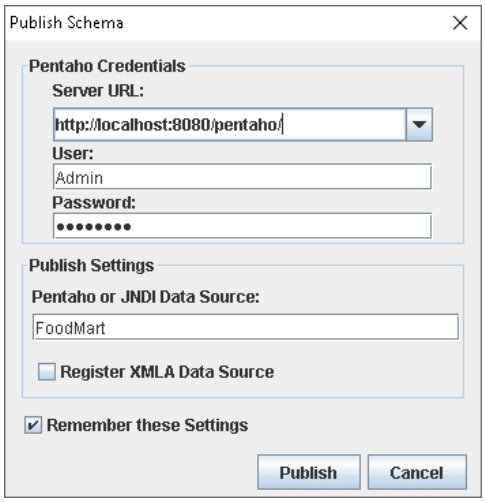
Publicación del cubo

Finalmente, para realizar la publicación del cubo, se debe tener en cuenta que Pentaho server debe estar activo, para finalmente realizar la publicación del cubo, se debe realizar el siguiente proceso:

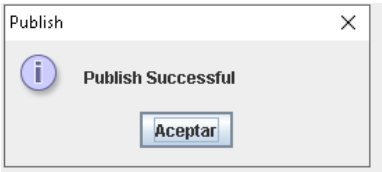
1. Clic en New / Publish



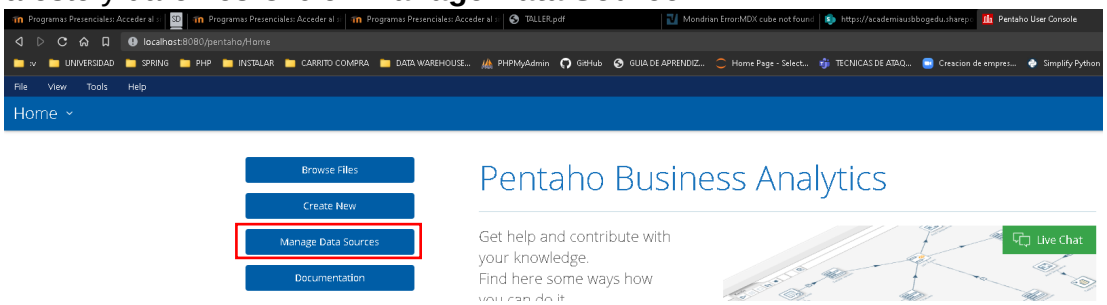
2. En la siguiente ventana verificaremos que el path de acceso a pentaho server sea el correcto, en el cual se encuentre el puerto 8080 definido, ingresaremos las credenciales de administrador (Admin – password) y daremos clic en Publish



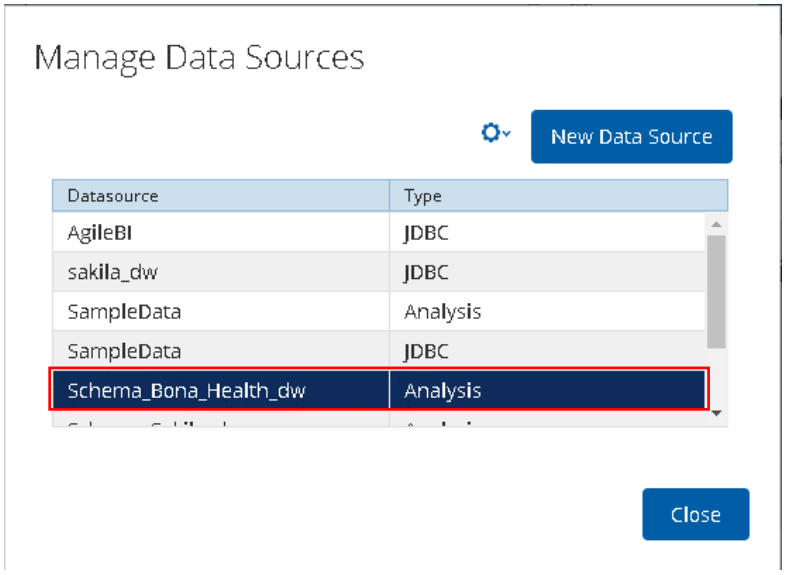
3. Si el proceso de publicación del cubo se realizó correctamente, obtendremos el siguiente mensaje.



4. Para verificar finalmente que el cubo se encuentra disponible ya en el servidor, dentro de Pentaho Server, reiniciaremos el servidor, ingresaremos a este y daremos Clic en **Manage Data Source**.



Dentro de la ventana que se nos despliegue buscaremos schema_sakila_dw, si lo encontramos significa que el proceso de publicación se realizó correctamente, de lo contrario, se recomienda reiniciar Pentaho Server y/o realizar de nuevo la publicación desde Schema Workbench.



En la siguiente parte, se realizará las adecuaciones necesarias para el despliegue de un Dashboard destinado para el cubo publicado, todo esto a través de Pentaho Server.

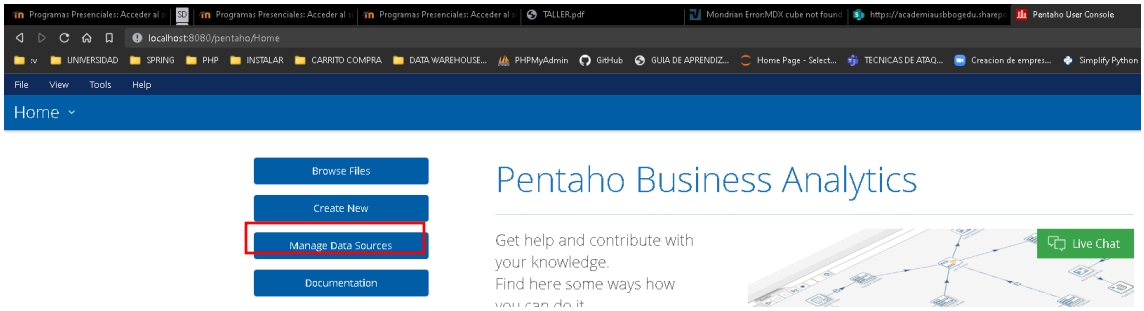
Dashboard del cubo en Pentaho Server

Para poder desplegar operaciones de consulta y manipulación del cubo recién creado, Pentaho y su servidor presenta un entorno llamado Dashboard el cual nos permitirá realizar las tareas previamente mencionadas, para esto proceso, primero debemos realizar la configuración del entorno.

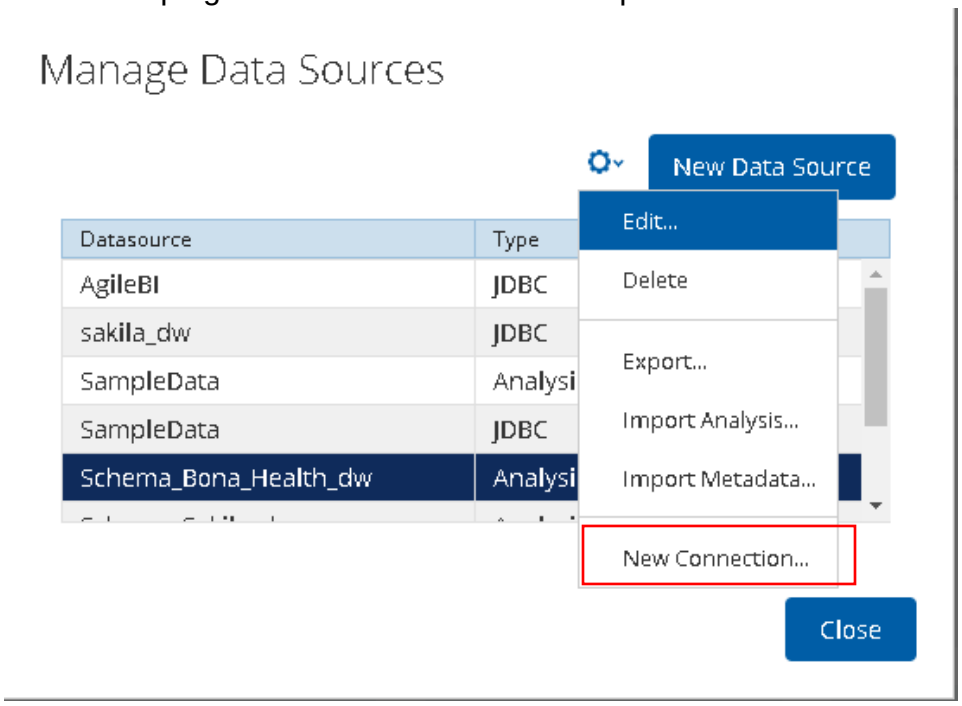
Conexión al Data Warehouse

Es importante tener presente que el cubo que nosotros publicamos es la estructuración de la base de datos correspondiente al Data Warehouse, es una estructura vacía y para llenarla o asociarla a los datos, requerimos de incluir una conexión JDBC, para esto se realizaran los siguientes pasos:

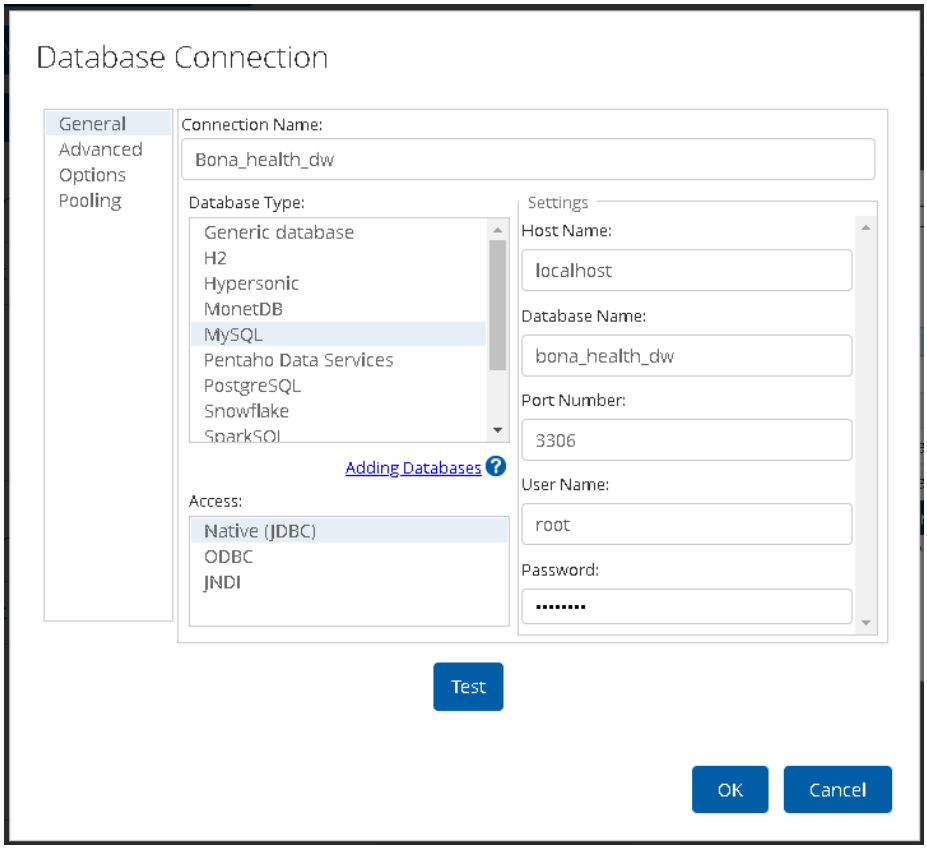
1. Dentro del Home de Pentaho server, daremos clic en la opción de Manage Data Sources



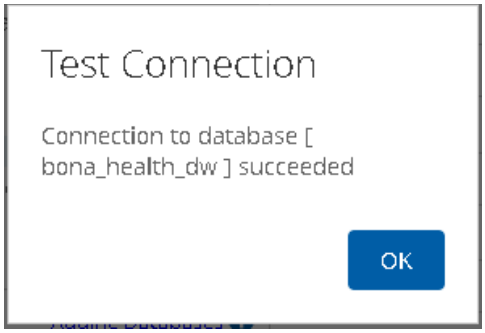
2. Dentro de la ventana desplegada, daremos clic en el icono de configuración, y en la lista desplegada vamos a dar clic en la opción New Connection



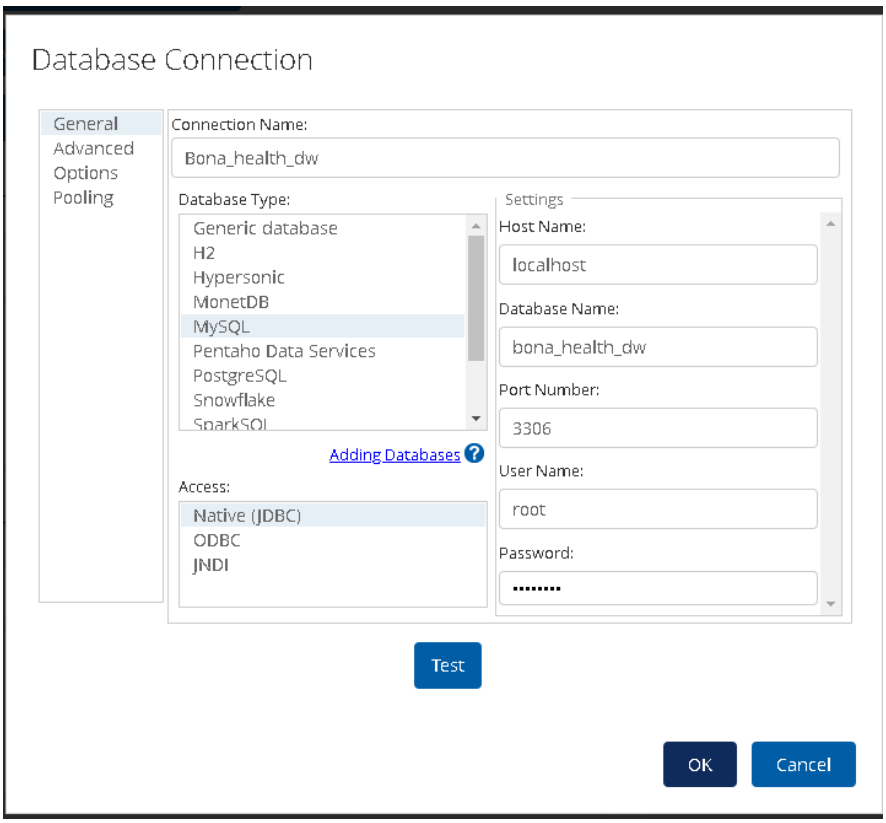
3. En la ventana desplegada, vamos a configurar la conexión a la base de datos correspondiente al Data Warehouse, para esto definimos los siguientes datos:
   - Connection name: Bona_health_dw
   - Database Type: MySQL
   - Access: Native (JDBC)
   - Host name: localhost
   - Database name. bona_heatlh_dw
   - Port number: 3306
   - Username: root
   - Password: contraseña del usuario root



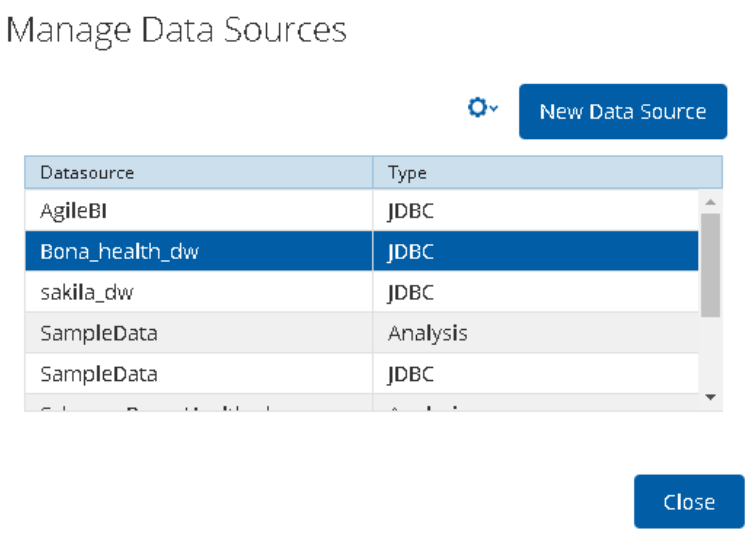
4. Para comprobar que los parámetros ingresados son correctos, daremos clic en el botón Test y deberemos obtener la siguiente respuesta que verifique que todo está correcto.



5. Finalmente se da clic en Ok para guardar la conexión

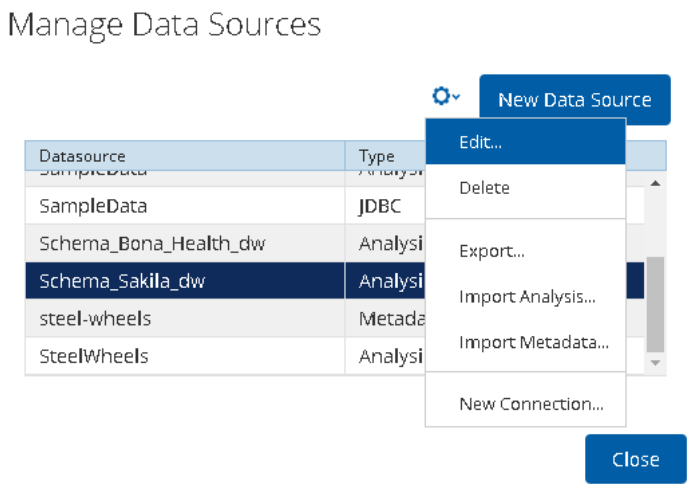


6. El listado de Data Source se actualizará y en este encontraremos la conexión creada lista para ser usada.

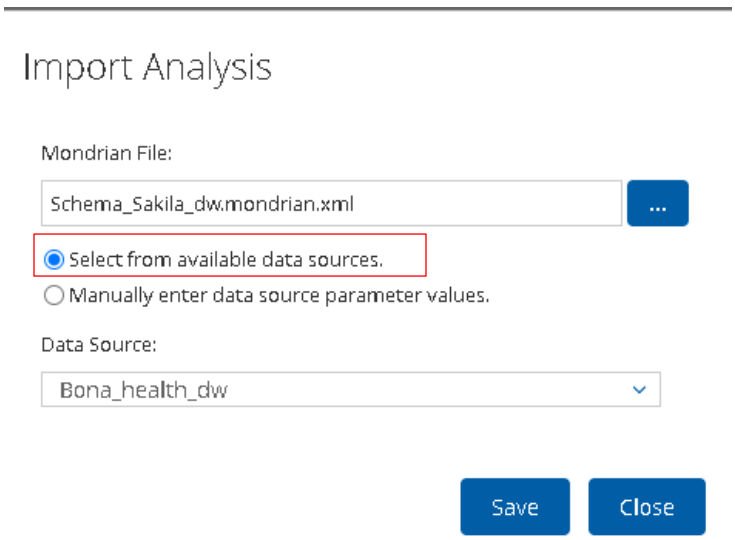


Configuración del Schema

Con la conexión empleada y teniendo en cuenta que previamente ya hayamos publicado el cubo en el servidor, vamos a realizar una simple configuración al Schema la cual posibilitara el acceso a los datos dentro del Data Warehouse. Para esto, dentro del listado de Data Sources vamos a seleccionar el Schema del cubo, daremos click en el icono de configuración y elegiremos la opción de Edit.



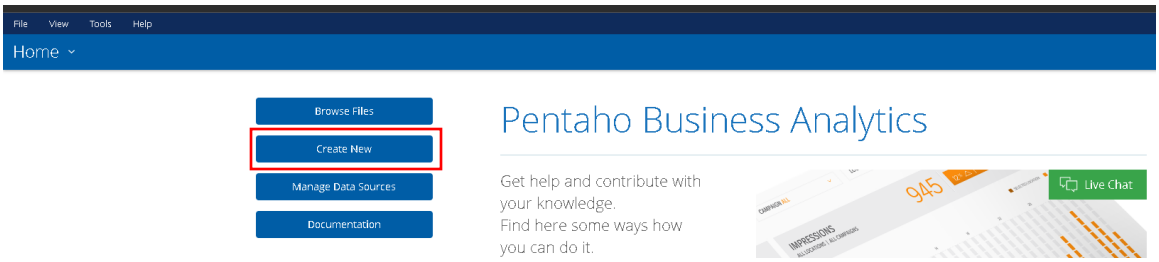
Dentro de la ventana que se nos genera, vamos a definir la fuente de datos, para esto elegiremos la opción **Select from avaible data sources** y en la lista de data sources seleccionaremos a **Bona_health_dw**



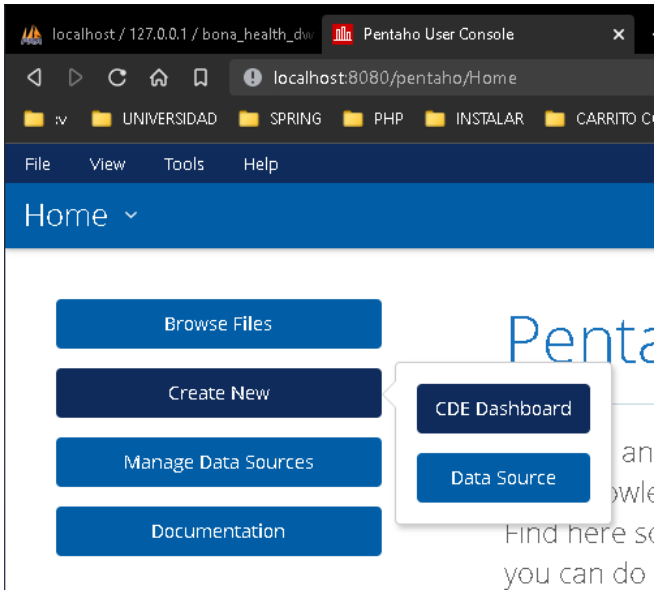
Con estos elementos definidos, daremos clic en **Save** y tendremos definida la fuente de datos correctamente asociada al Data Warehouse.

Despliegue del Dashboard

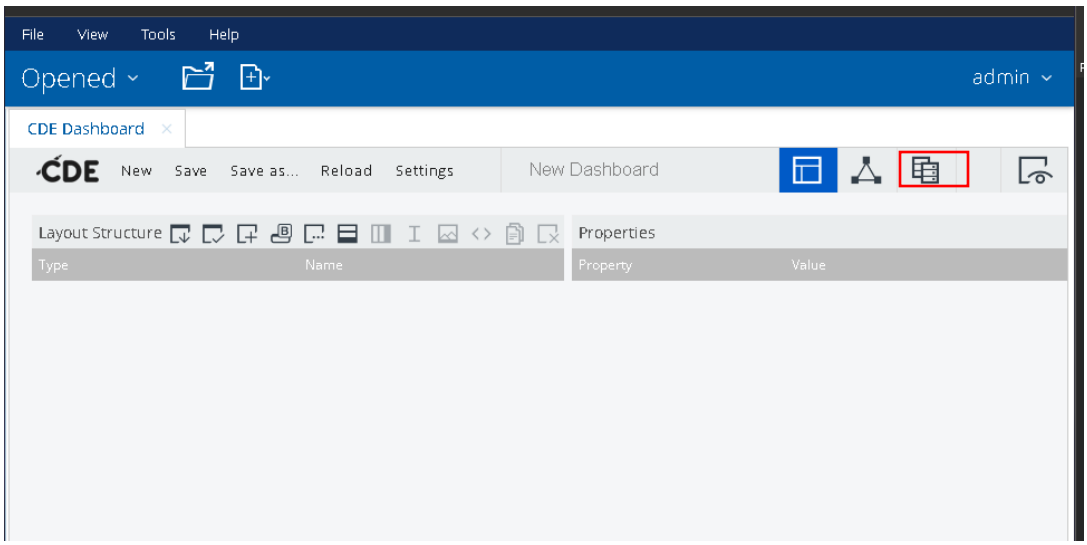
Para realizar el despliegue del Dashboard del cubo, dentro de la pagina de home, daremos clic en la opción Create New:



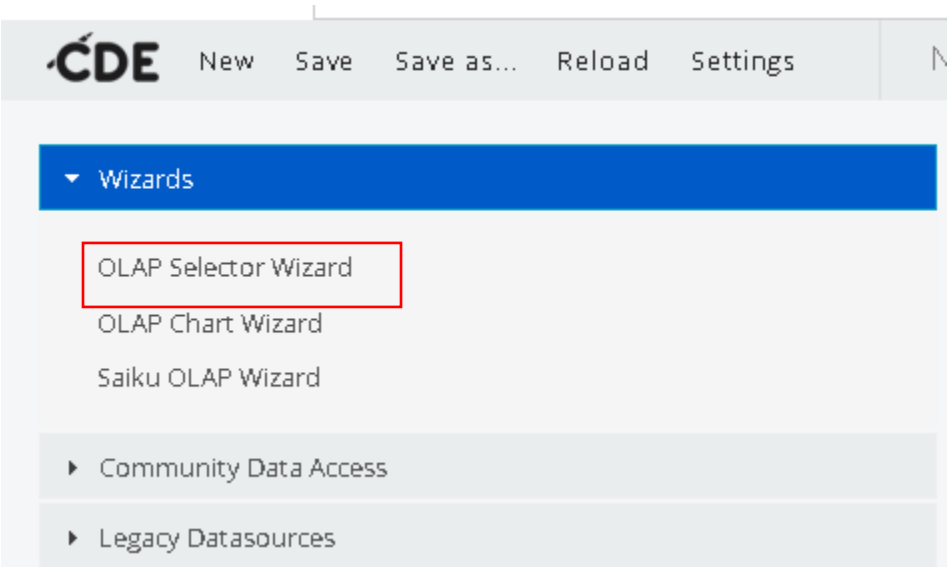
Elegiremos la opción de crear un nuevo CDE Dashboard, el cual es un entorno en el cual se nos permitirá la manipulación y creación de consultas MDX sobre el cubo creado:



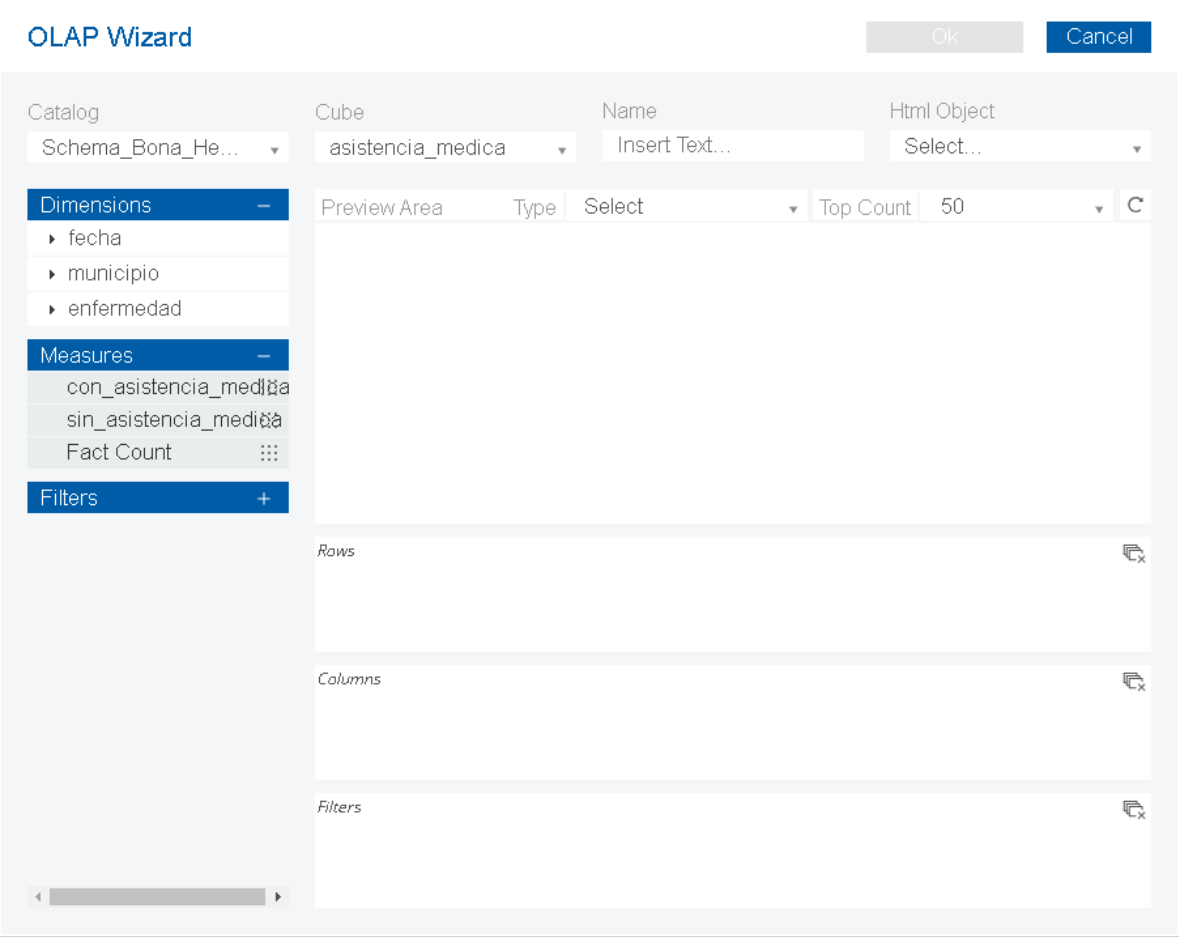
El proceso de carga del Dashboard en un poco demorado, pero al momento de que se nos despliegue la siguiente interfaz indicara que el proceso ha terminado. En esta interfaz daremos clic en el icono de Datasource Panel.



Dentro de la ventana que se nos genera nos dirigiremos a Wizards y daremos clic en la opción OLAP Selector Wizard.



Si todo el proceso de publicación y configuración del cubo se realizó correctamente, en la ventana que se nos genera, seleccionaremos del listado de Catalog el Schema de Bona Heatlh y debe desplegarnos los cubos, dimensiones y medidas asociados a este. En esta interfaz podremos realizar la manipulación grafica de consultas MDX sobre el cubo.



Con esto último terminaríamos con todo el proceso de despliegue de un cubo empleando la base de datos asociada al Data Warehouse.

CREACIÓN DE REPORTES

En esta sección se explicará el proceso de creación de reportes, los cuales permitan la visualización de los datos almacenados en el Data Warehouse teniendo en cuenta las necesidades de la organización usando cada uno de los cubos/tablas de hechos creados previamente. Para este propósito utilizaremos la herramienta de Jaspersoft Studio, que nos facilitará la creación de reportes. La instalación de esta herramienta es tan sencilla como dirigirnos al enlace, descargar el instalador y ejecutarlo. Su interfaz está basada en Eclipse, por lo que es fácil familiarizarse con ella.

Reporte 1

Teniendo en cuenta los objetivos planteados al principio del proyecto, realizaremos un reporte que contemple las asistencias médicas en los registros. Para esto

tendremos en cuenta el número de asistencias médicas que se otorgó en cada municipio con respecto a una de las tres enfermedades que son el objeto de estudio del proyecto.

En principio vamos a realizar un script en MySQL que nos permita obtener los datos necesarios para la realización de este reporte.

```
SELECT DISTINCT
       dim_municipio.municipio as Municipio,
       sum(sum_asistencia_med) as Asistencias,
       nombre as Enfermedad
FROM
       bona_health_dw.fact_asis_medica
INNER JOIN
       dim_municipio
ON
       fact_asis_medica.municipio = dim_municipio.id_municipio
GROUP BY
       municipio, nombre;
```

Este comando nos mostrará el municipio sobre el cual se realizó la asistencia médica, el número de asistencias que ocurrieron ahí y el tipo de enfermedad a la cual se realizó la asistencia.

Luego de tener esta consulta, nos dirigiremos a Jaspersoft Studio y realizaremos lo siguiente:

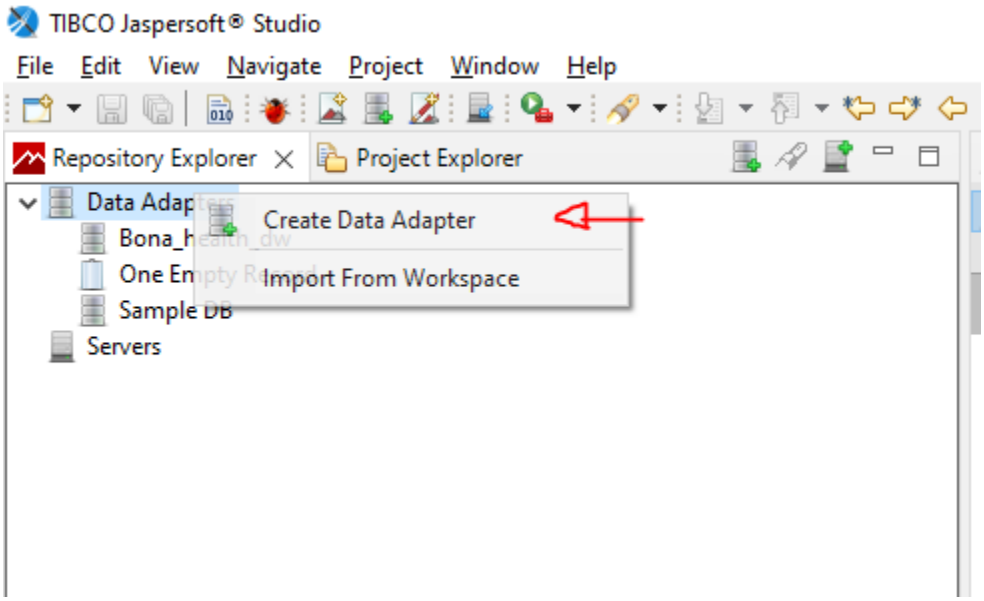


Ilustración 29. Interfaz Jaspersoft Studio

Añadimos un Data Adapter, que es la conexión a la base de datos que contiene los usuarios que vamos a utilizar.

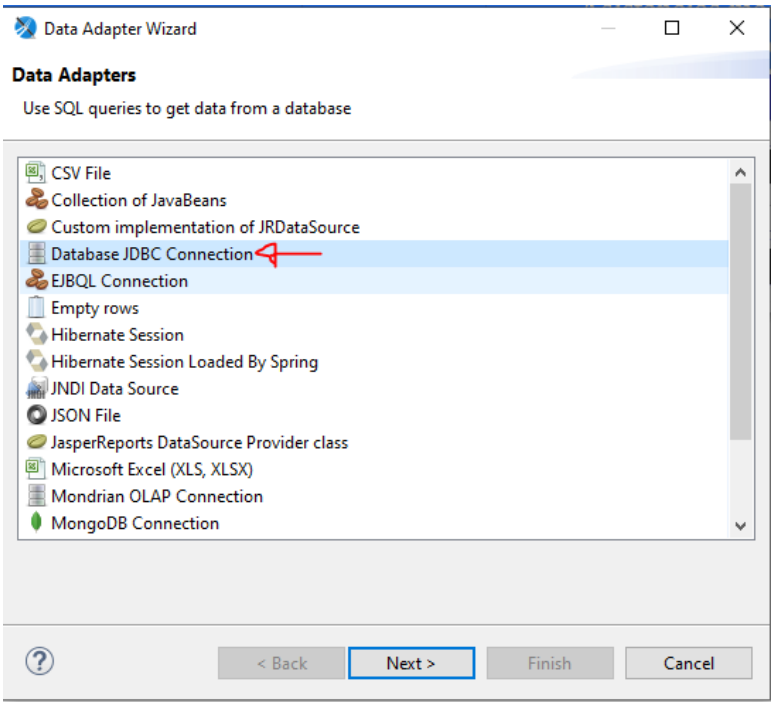


Ilustración 30. Tipo de conexión en Jaspersoft Studio

Seleccionamos el tipo de conexión que necesitamos (en este caso como es una conexión a una Base de datos MySQL, seleccionamos la opción de JDBC).

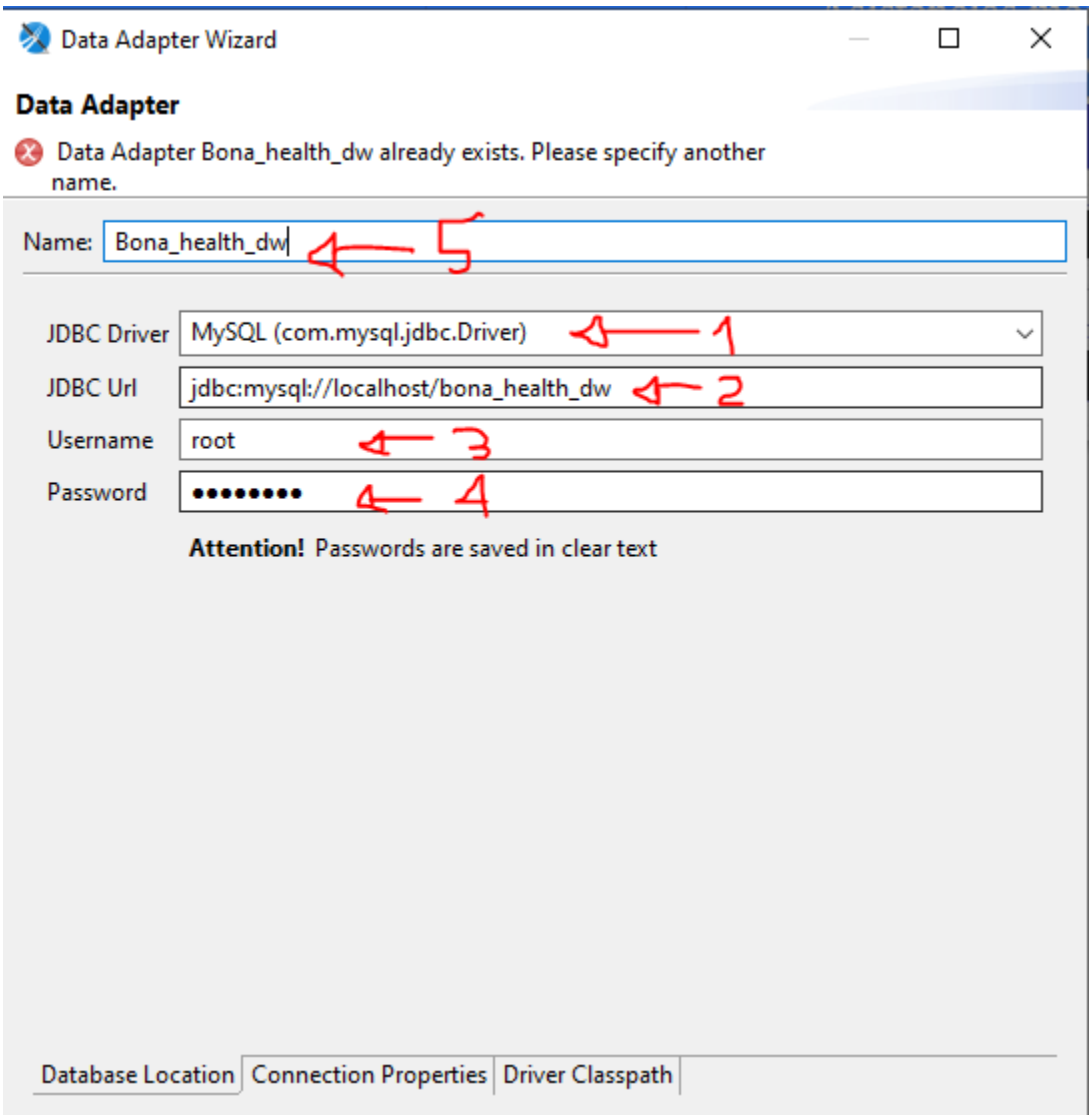


Ilustración 31. Configuración del adaptador

El programa nos solicitará los siguientes datos (al momento de realizar el ejercicio ya se había creado el adaptador, es por eso que sale el mensaje de error):

1. El driver que vamos a utilizar (seleccionamos el com.mysql).
2. La URL de la base de datos a utilizar (colocamos el nombre de la base de datos al final).
3. El usuario (por defecto es root)

4. La contraseña (por defecto es password)
5. El nombre de la conexión.

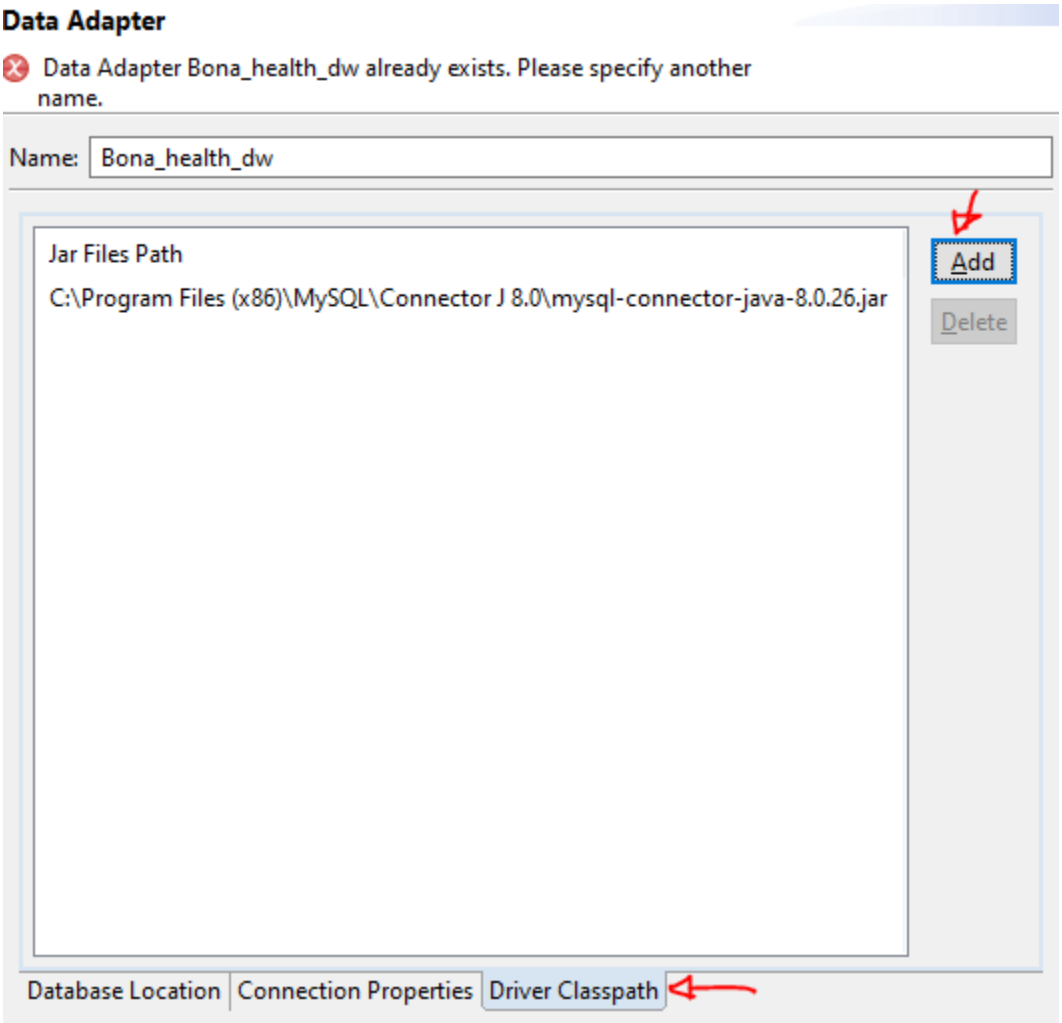


Ilustración 32. Driver Classpath

Luego, en la sección de Driver Classpath seleccionamos “Add” y especificamos la ruta del conector de mysql que estemos utilizando para realizar la conexión.

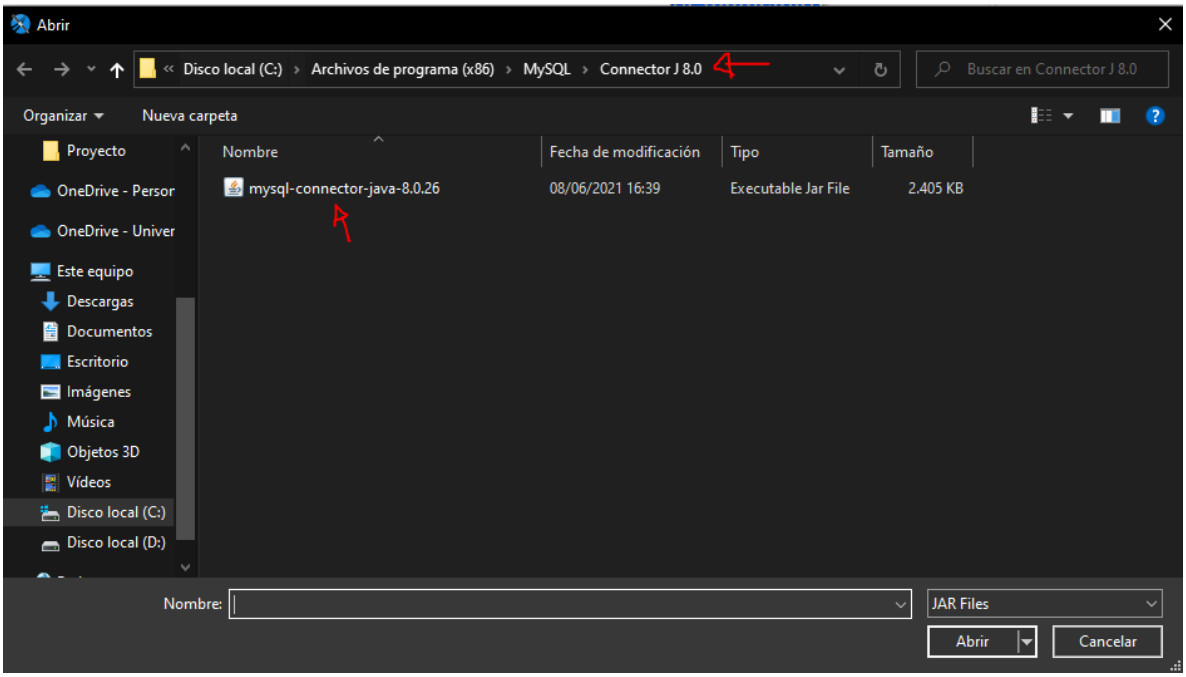


Ilustración 33. Ruta del conector de MySQL

Luego de realizar la conexión a la base de datos, crearemos nuestro reporte.

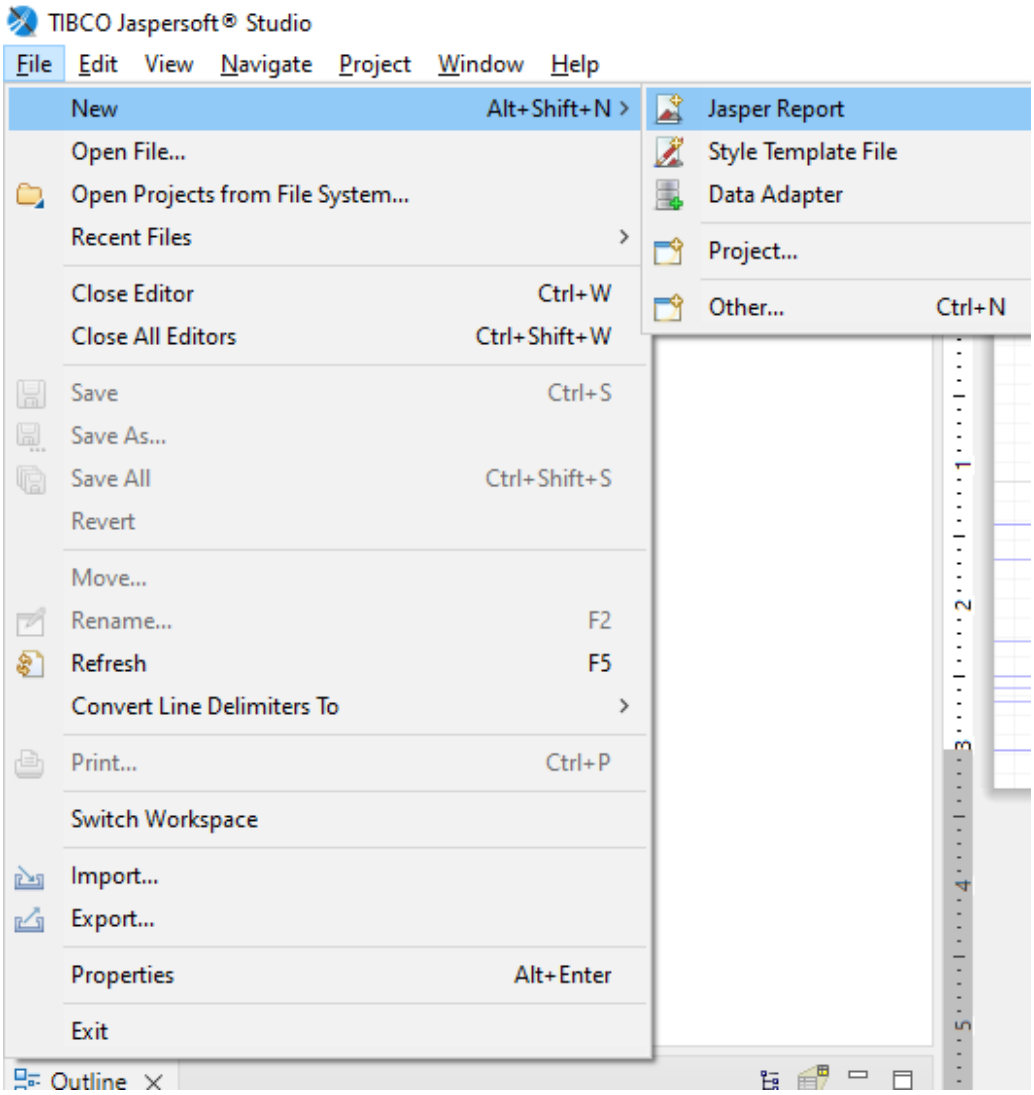


Ilustración 34. Creación de reportes en Jaspersoft Studio

Para fines prácticos, seleccionaremos una de las plantillas que nos provee la aplicación, ya que nos permite utilizar directamente los datos que necesitamos visualizar, como se muestra a continuación.

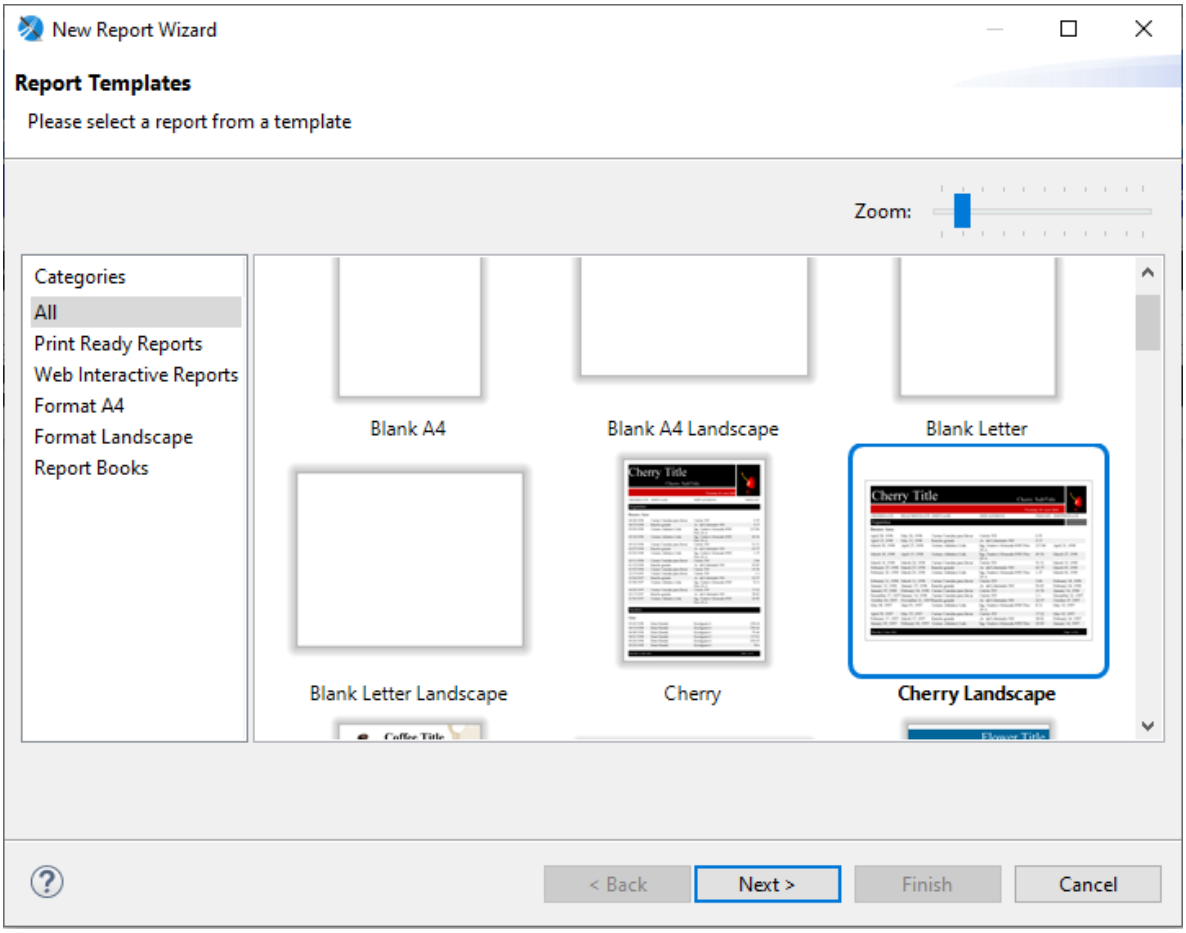


Ilustración 35. Selección de plantillas

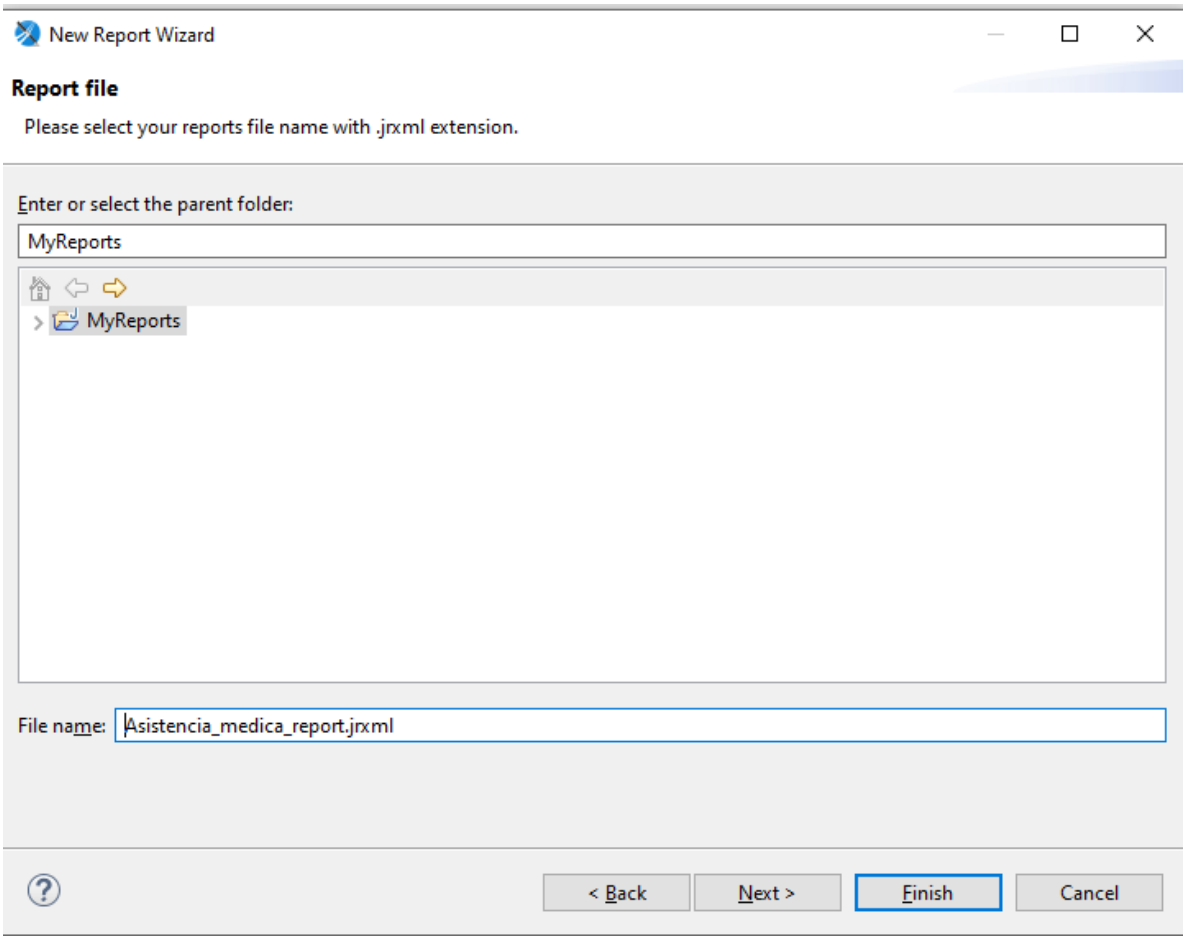


Ilustración 36. Creación del archivo

Al darle siguiente nos mostrará esta ventana, en donde seleccionaremos la carpeta por defecto que aparece ahí, y le asignaremos un nombre al reporte.

Luego de esto seleccionaremos el Data source(1) que configuramos hace un momento y colocamos el script que planteamos(2) al inicio del reporte.

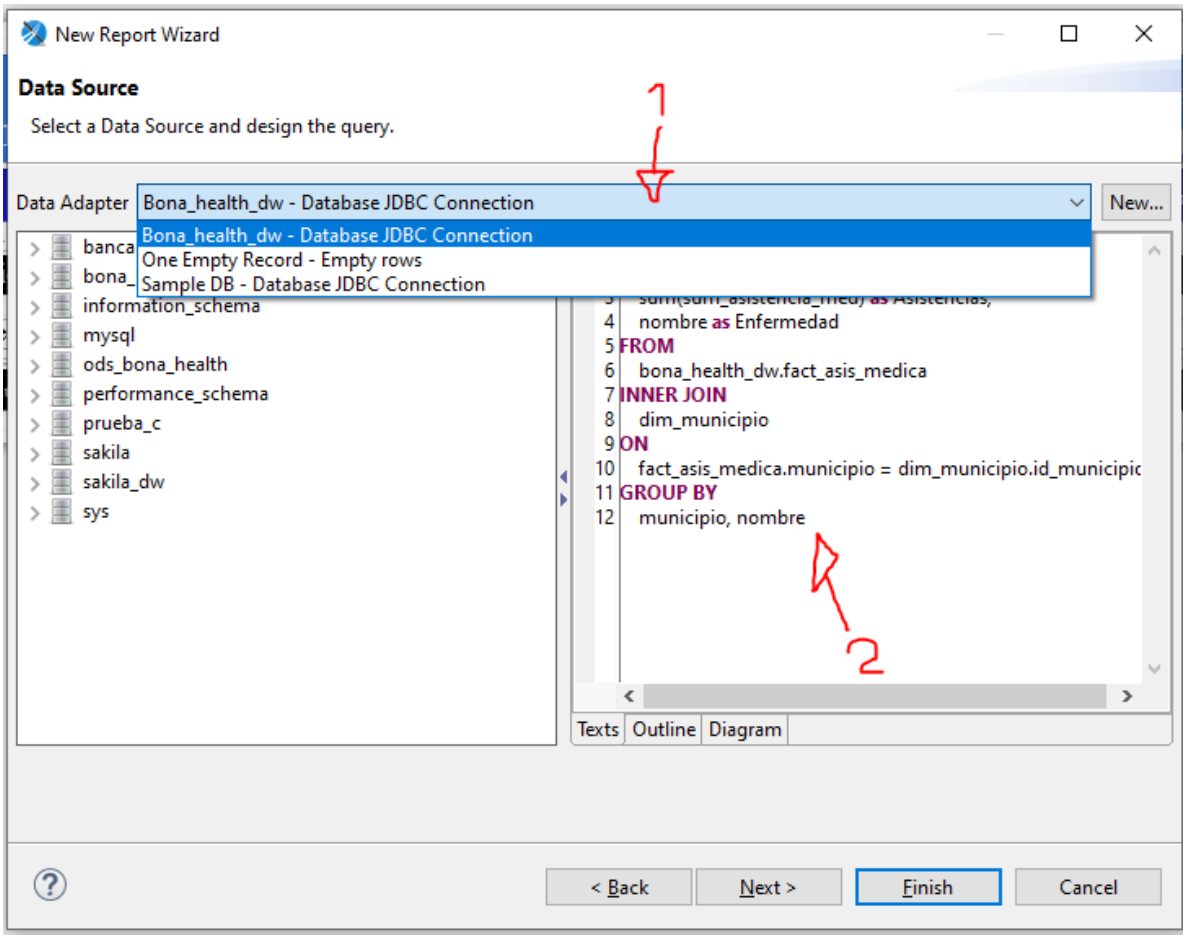


Ilustración 37. Selección y uso del Data Source

En la siguiente ventana nos aparecerán los campos que son el resultado del script (municipio, enfermedad y asistencia médica). Aquí seleccionaremos los campos que se van a utilizar en el reporte. Seleccionamos la opción de “>>”.

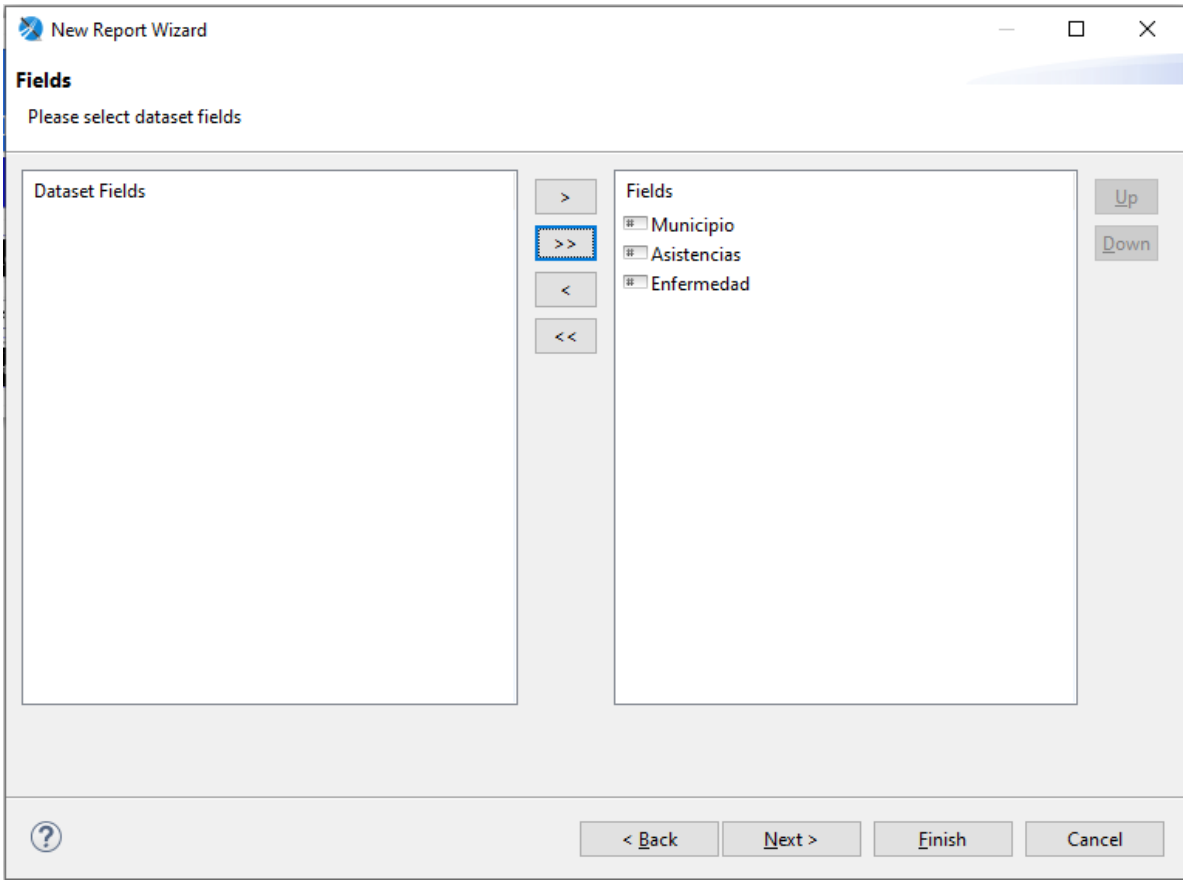


Ilustración 38. Selección de Campos de datos

Luego nos solicitará escoger los campos que se utilizaran para agrupar a todos los demás. Debido a que buscamos visualizar el número de asistencias por municipio principalmente, seleccionaremos ese campo.

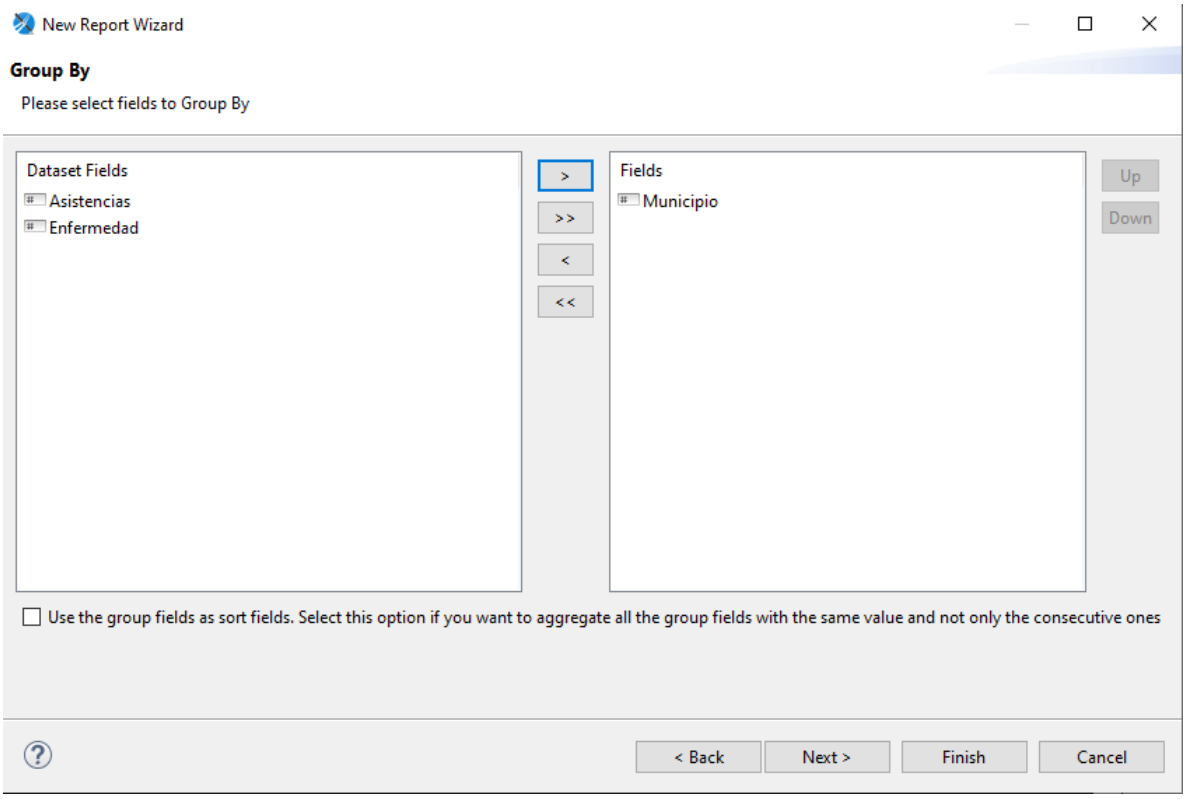


Ilustración 39. Selección de agrupaciones

Luego de esto damos finalizar y nos aparecerá un reporte parecido al siguiente.

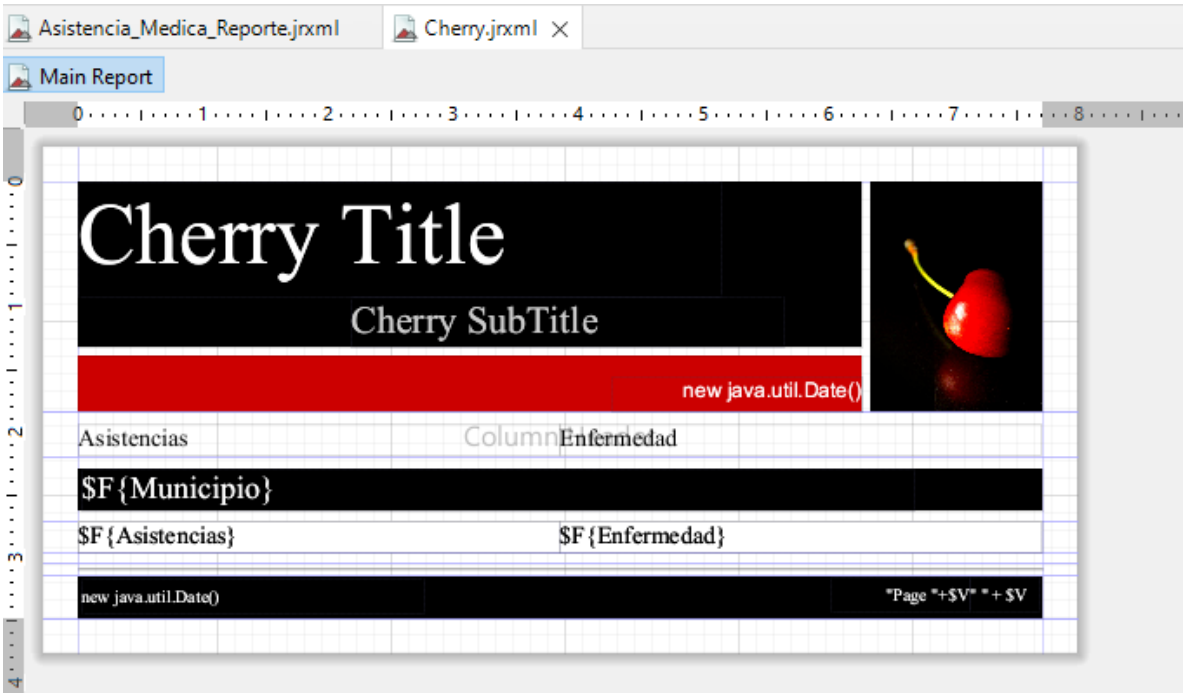


Ilustración 40. Primer reporte creado con Jaspersoft Studio

Luego de realizarle un par de modificaciones, que sean más acordes a este proyecto, tenemos lo siguiente.

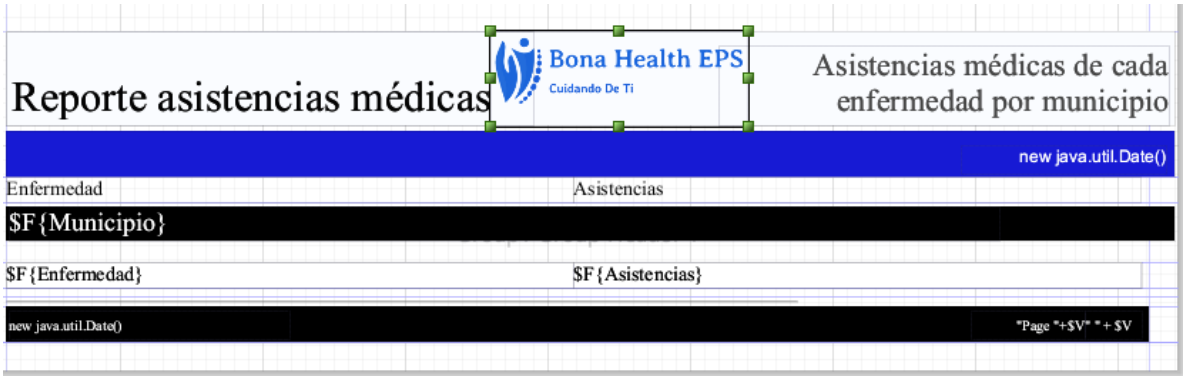


Ilustración 41. Reporte modificado

Luego realizaremos una previsualización del reporte, y lo exportamos.

Reporte asistencias médicas

Bona Health EPS
Cuidando De Ti

Asistencias médicas de cada enfermedad por municipio

martes 16 noviembre 2021

Enfermedad	Asistencias
Viterbo	
Cancer de Mama	0
Cancer de Pulmón	3
VIH	1
Villavicencio	
Cancer de Pulmón	1
Villamaría	
Cancer de Mama	7
Cancer de Pulmón	11
VIH	2
Victoria	
Cancer de Mama	1
Cancer de Pulmón	4
VIH	2

martes 16 noviembre 2021 Page 1 of 8

Ilustración 42. Reporte Nº1 Finalizado

Con esto hemos finalizado el primer reporte.

Reporte 2

Para el segundo reporte tomaremos el cubo de Defunciones por municipio como referencia. En este caso realizaremos un reporte acerca del número de muertes en total por cada enfermedad, ya que en el reporte anterior sacamos el número de defunciones específico por cada departamento. Para empezar realizaremos la consulta que nos deberá proveer estos datos para su correcta visualización.

```
SELECT distinct
      enfermedad as Enfermedad,
    sum(cant_defun) as Defunciones
FROM
      bona_health_dw.fact_defun_por_municipio
GROUP BY
```

```
enfermedad;
```

Para este reporte realizaremos las mismas acciones detalladas en las ilustraciones 34 y 35, solo que esta vez seleccionaremos una plantilla más adecuada a nuestro propósito.

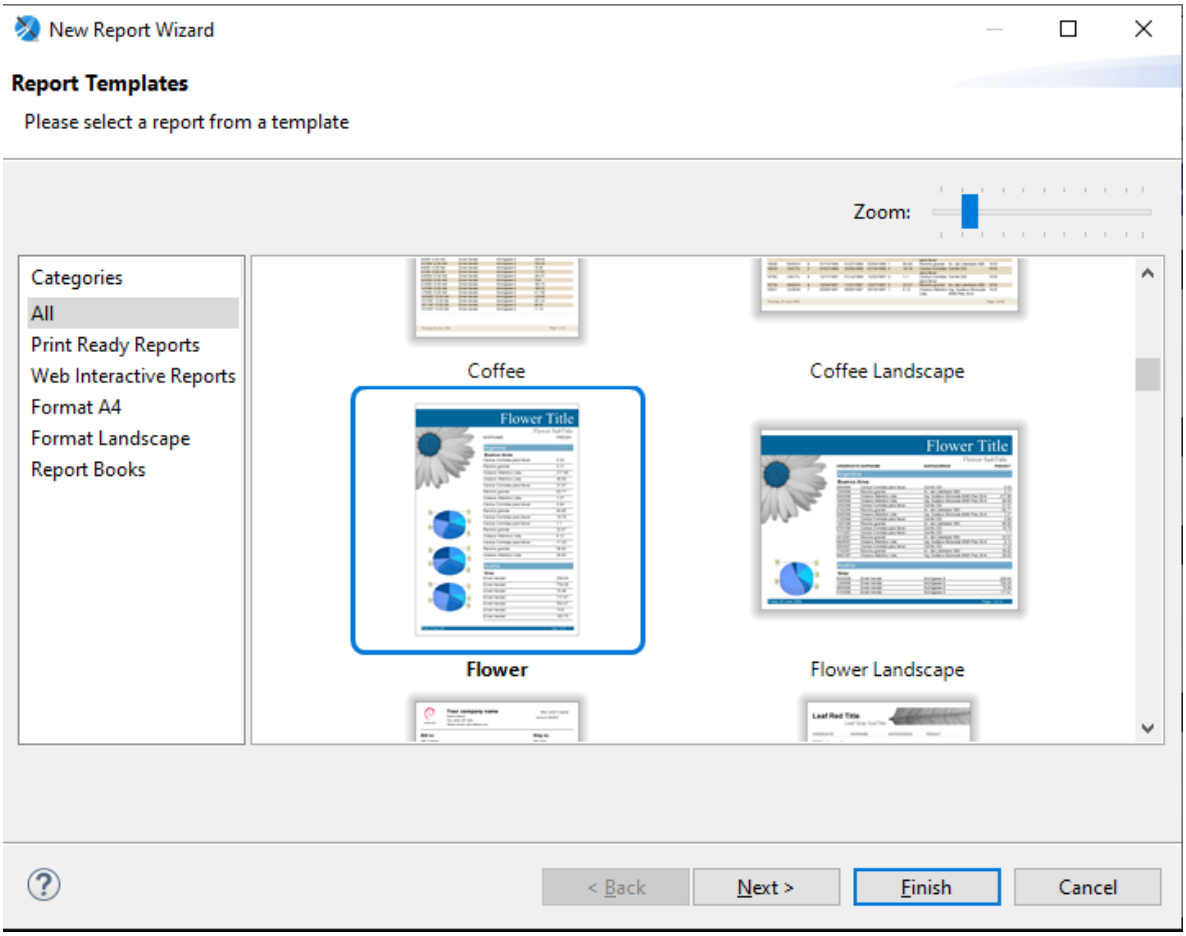
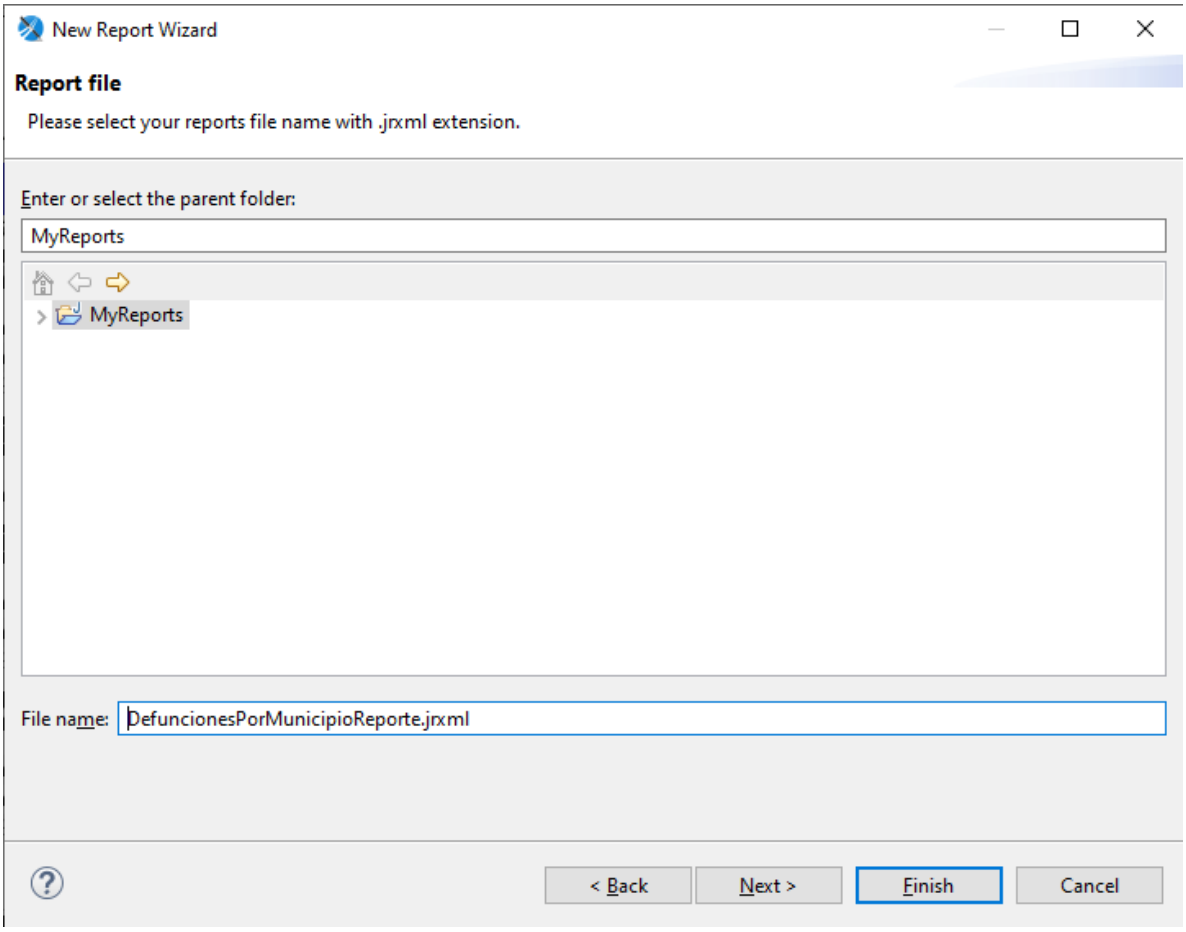


Ilustración 43. Plantilla reporte Nº2

De la misma manera le asignamos un nombre.



Realizamos el mismo procedimiento que realizamos en la ilustración 37.

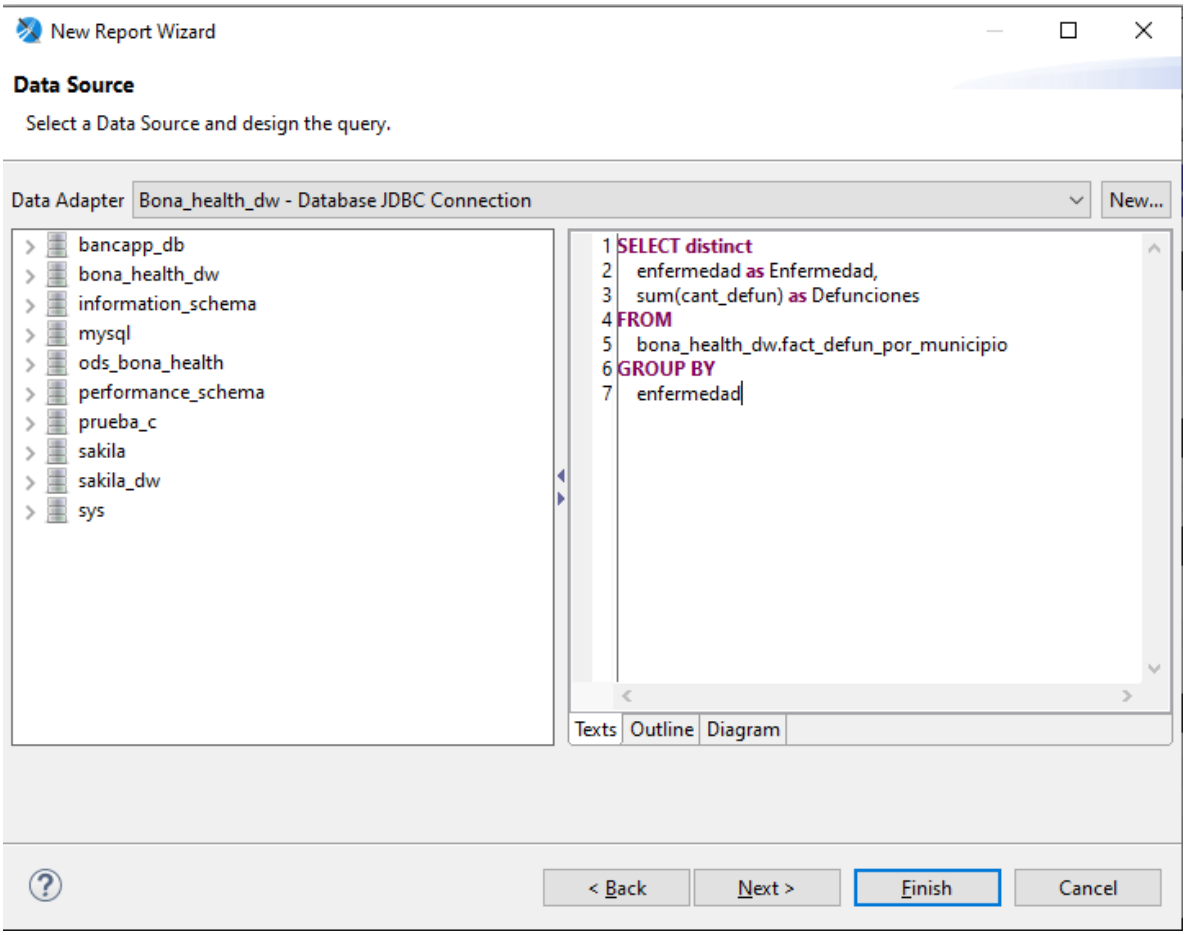


Ilustración 44. Inserción de Consulta MySQL

Repetimos el procedimiento realizado en la ilustración 38, y en el procedimiento de la ilustración 39, seleccionamos el campo de agrupación como Enfermedad.

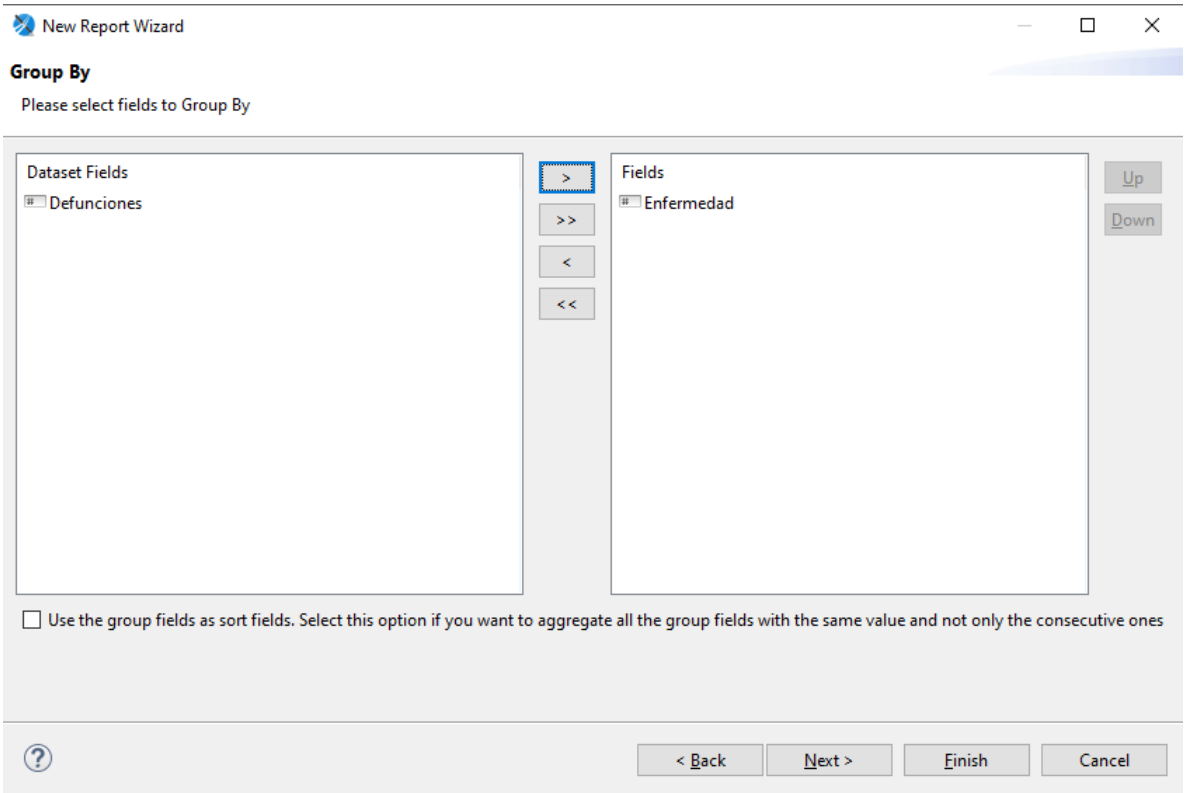


Ilustración 45. Campo de agrupación

Luego de darle finalizar, modificaremos la plantilla de manera que al exportar el reporte quede de la siguiente manera.

Reporte de defunciones por enfermedad

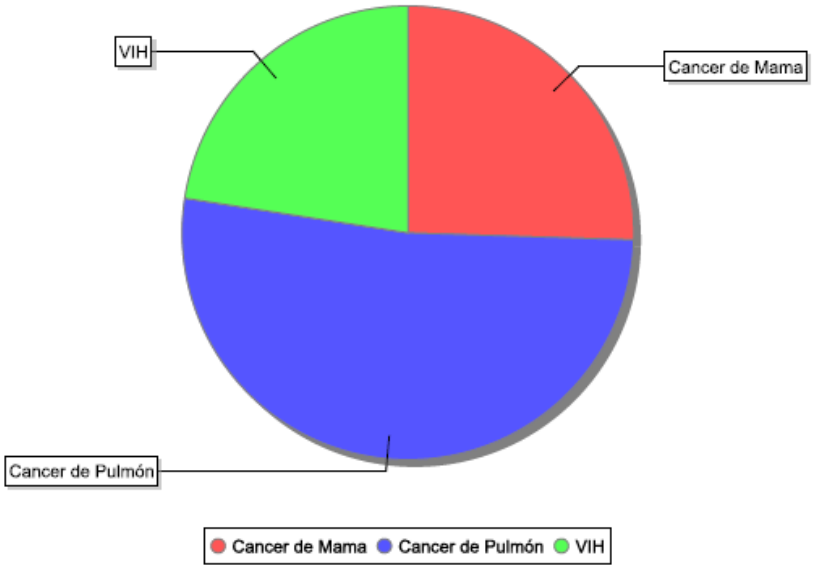
Bona Health EPS
Cuidando De Ti

Defunciones

Cancer de Mama
503

Cancer de Pulmón
1025

VIH
445



martes 16 noviembre 2021 Page 1 of 1

Ilustración 46. Reporte Nº2 Finalizado

Reporte 3

Finalmente, para el último reporte necesitamos saber el crecimiento de la tasa de mortalidad a lo largo del tiempo. Para esto realizaremos una consulta en MySQL que nos provea de la información que necesitamos visualizar.

```
SELECT DISTINCT
       anio as Año,
    mes as Mes,
    sum(cantidad_fallecidos),
    enfermedad
FROM
       bona_health_dw.fact_defunciones_fecha
INNER JOIN
       dim_fecha
ON
       fact_defunciones_fecha.fecha = dim_fecha.fecha_completa
GROUP BY
       anio, mes;
```

Realizamos el mismo procedimiento que hemos realizado para los anteriores reportes.

Luego de hacer uso de nuestra herramienta para realizar el reporte, nos queda de la siguiente manera.

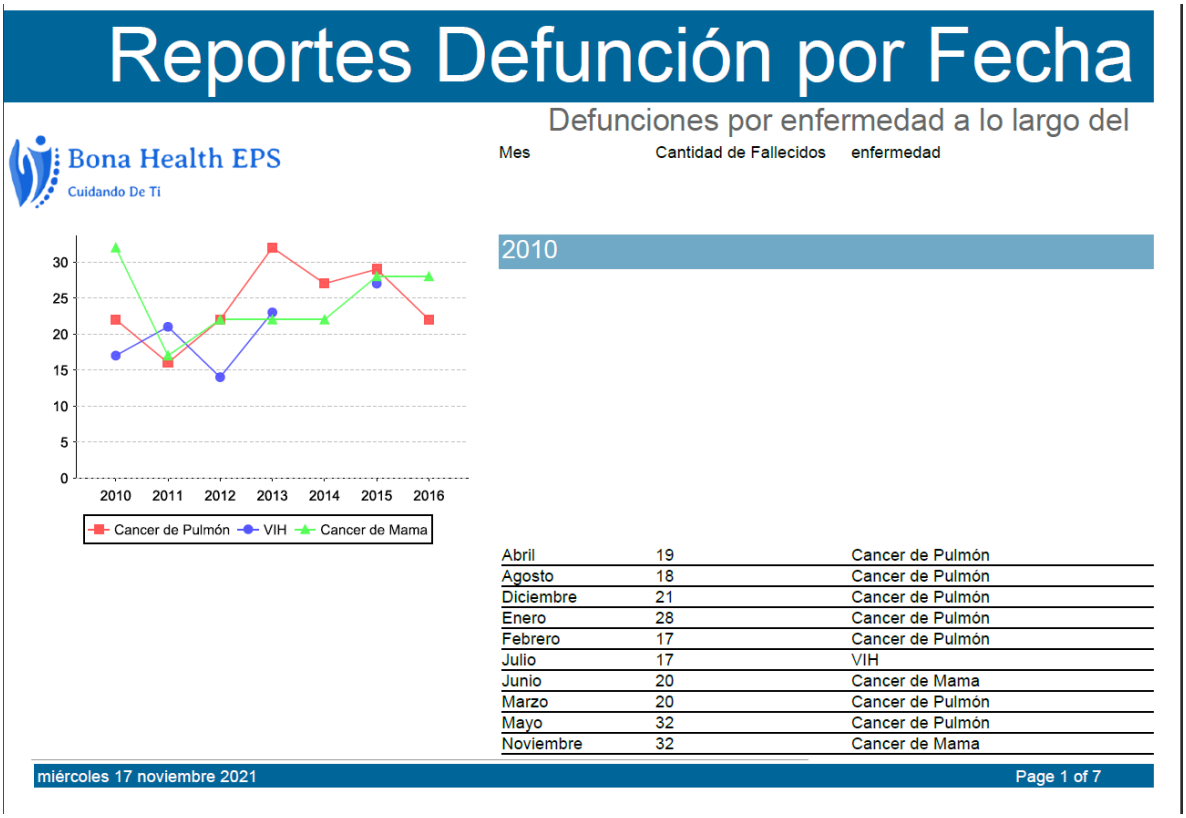


Ilustración 47. Reporte Nº3 Finalizado

Con esto finalizamos los reportes del proyecto.

TABLA DE ILUSTRACIONES

Ilustración 1 Defunciones por VIH 6
Ilustración 2 Defunciones por cáncer de mama 6
Ilustración 3 Defunciones por cáncer de pulmón..... 6
Ilustración 4 Asistencia medida prestada a los fallecidos por VIH..... 7
Ilustración 5 Atención medica prestada a los fallecidos por cáncer de mama 7
Ilustración 6 Atención medica prestada a los fallecidos por cáncer de pulmón..... 8
Ilustración 7 defunciones por VIH clasificadas por municipio 8
Ilustración 8 defunciones por cáncer de pulmón clasificadas por municipio 9
Ilustración 9 defunciones por cáncer de mama clasificadas por municipio 9
Ilustración 10 Muestra de datos del dataset de mortalidad por VIH 11
Ilustración 11 Muestra de datos del dataset de mortalidad por cáncer del pulmón 12
Ilustración 12 Muestra de datos del dataset de mortalidad por cáncer de mama.. 13
Ilustración 13 Muestra de datos del dataset de departamentos y municipios de Colombia 20

Ilustración 14 Modelo entidad relación de las fuentes de datos 23
Ilustración 15 Diseño de dimensiones y medidas..... 24
Ilustración 16 Modelo CMDM – Defunciones por fecha 25
Ilustración 17 Modelo CMDM – Asistencia médica 26
Ilustración 18 Modelo CMDM – Defunciones por municipio 26
Ilustración 19 Modelo CMDM – Muertes 27
Ilustración 20 Muertes 30
Ilustración 21 Modelo ROLAP 31
Ilustración 22 Diagrama de Clases y componentes del sistema 36
Ilustración 23 Secuencia de procesos ETL 37
Ilustración 24 Salida en consola del programa ETL 56
Ilustración 25 Muestra de datos Fact_asistencia_medica 57
Ilustración 26 Muestra de datos Fact_defunciones_fecha 57
Ilustración 27 Muestra de datos fact_defunciones_municipio 58
Ilustración 28 Muestra de datos fact_muertes..... 58
Ilustración 29. Interfaz Jaspersoft Studio 95
Ilustración 30. Tipo de conexión en Jaspersoft Studio 95
Ilustración 31. Configuración del adaptador 96
Ilustración 32. Driver Classpath..... 97
Ilustración 33. Ruta del conector de MySQL 98
Ilustración 34. Creación de reportes en Jaspersoft Studio 99
Ilustración 35. Selección de plantillas..... 100
Ilustración 36. Creación del archivo 101
Ilustración 37. Selección y uso del Data Source 102
Ilustración 38. Selección de Campos de datos..... 103
Ilustración 39. Selección de agrupaciones 104
Ilustración 40. Primer reporte creado con Jaspersoft Studio 105
Ilustración 41. Reporte modificado 105
Ilustración 42. Reporte Nº1 Finalizado 106
Ilustración 43. Plantilla reporte Nº2 107
Ilustración 44. Inserción de Consulta MySQL..... 109
Ilustración 45. Campo de agrupación..... 110
Ilustración 46. Reporte Nº2 Finalizado 111
Ilustración 47. Reporte Nº3 Finalizado 113