

UNIVERSIDAD NACIONAL DE LA PATAGONIA SAN JUAN BOSCO



BASE DE DATOS II

PROFESOR ADJUNTO:

LIC. GABRIEL INGRAVALLO.

JEFE DE TRABAJOS PRÁCTICOS:

LIC. CRISTIAN PARISE.

LABORATORIO N° 5 Data Warehousing & Data Mining

GRUPO 5 - ALUMNOS:

LUCAS FOLETO.

DAVID MONJELAT.



Índice

Enunciado	3
Pautas adicionales	4
Consideraciones Agregadas	5
Desarrollo	6
Punto 1	6
Punto 2	10
Punto 3	15
Punto 4	25
Punto 5	28
Observaciones finales	44

Enunciado

- 1) Implementar la bd operacional de la cooperativa PatSur con sus dos sistemas de facturación.
- 2) Hacer inserciones masivas a todas las tablas mediante scripts similares al implementado en Lab 1.
- 3) Implementar un DW (en otra BD u otro esquema de la misma BD en postgres) para la cooperativa PatSur, basado en la BD operacional creada y utilizando el esquema de Copo de Nieve presentado en la teoría (Tabla de hecho Ventas, medidas unidad y precio, dimensiones: sucursal, región, provincia, ciudad, cliente, tipo cliente, producto, categoría, subcategoría, y la dimensión Tiempo existente en todo DW). Además implementar los scripts para hacer la ETL correspondiente desde la BD operacional.
- 4) Implementar consultas para las vistas que los analistas requirieron Venta vista por mes o por año, por sucursal, por región, por cliente y demás combinaciones entre las perspectivas. El mínimo nivel de detalle que se quiere tener disponible para el análisis de las ventas (\$ vendidos y unidades vendidas) es el de la facturas. Es necesario conocer también de qué manera influye, en las ventas de productos, la zona geográfica en la que están ubicados los locales. De cada cliente se desea conocer cuales son los que generan mayores ingresos a la cooperativa. Se necesitará hacer análisis diarios, mensuales, trimestrales y anuales. Para las mismas se deben hacer los SELECT y comentarlos con las funciones GROUPING SETS, ROLLUP y CUBE brindadas por las extensiones del lenguaje SQL:1999 para facilitar la agrupación de los datos.
- 5) Se deberá hacer una implementación en una plataforma Open Source de Business Intelligence, tomando los datos del Data Warehouse generado y armando un cubo con las medidas y dimensiones señaladas, luego utilizar la herramienta OLAP para generar las vistas indicadas en el punto 4. Basándonos en el siguiente artículo que menciona diferentes suites open source para reporting o business intelligence:

<https://opensource.com/business/16/6/top-business-intelligence-reporting-tools>

Cada grupo va a tener que investigar/implementar una de ellas para explotar el Dw según el siguiente detalle:



- Grupo 1: BIRT
- Grupo 2: JasperReport
- Grupo 3: Pentaho
- Grupo 4: SpagoBI
- Grupo 5: KNIME
- Grupo 6: ReportServer

En el mismo artículo se dan los links a los sitios principales y de descarga de cada plataforma.

Pautas adicionales

El presente laboratorio tiene las siguientes pautas adicionales:

- El Lab 5 Oficia también de Trabajo Práctico final
- Puede hacerse individualmente o de a dos como máximo
- En el informe debe constar
 - Como hicieron los scripts de carga de la base de datos operacional y su llenado en general.
 - Como hicieron el script de carga del Data Warehouse para cada sistema de facturación.
 - Las ejecuciones que hacen del script para llenar el DW.
 - Respuesta a las consultas del punto 4 (si no pueden hacer todas , algunas) con SQL con las extensiones de agrupamiento, etc.
 - El desarrollo de una pequeña reseña de la herramienta asignada de reporting o business intelligence y de cómo la pudieron implementar (o si no pudieron los problemas con los que se encontraron) y el o los reportes del punto 4 que pudieron implementar con dicha herramienta.
- Además de la entrega del informe y los scripts y archivos de trabajo, deberá exponerse la solución ante la cátedra y compañeros de cursada en fecha a designar.



Consideraciones Agregadas

- ❑ Las tablas para las cuales hay que hacer tablas de equivalencia en el DW son Productos y Clientes.
- ❑ La tabla “tiempo” en el sistema de facturación II , no corresponde. No implementarla en el sistema de facturación
- ❑ Que no exista tabla de subcategoría, simplemente en las categorías de productos la clave es compuesta cod_categoria, cod_subcategoria pero no existe una tabla extra para las subcategorías.
- ❑ Que en el DW: en la tabla de hechos, además de la fecha de la venta que está como un dato más de la venta, debe agregarse un idTiempo que va a ser una clave foránea a Tiempo, y se cargará con el id que se le otorga al periodo que se está incorporando.
- ❑ En la tabla de la dimensión tiempo por ende, no debería ir el campo día (ya que habrá una tupla por mes/año).
- ❑ En el DW , en la tabla clientes se debe eliminar el campo Apellido, dado que de los dos sistemas viene el nombre y apellido del cliente en el campo Nombre.

Recordemos que en el informe debe constar

- ❑ Como hicieron los scripts de carga de la base de datos operacional y su llenado en general.
- ❑ Como hicieron el script de carga del Data Warehouse para cada sistema de facturación.
- ❑ Las ejecuciones que hacen del script para llenar el DW.
- ❑ Respuesta a las consultas del punto 4 (si no pueden hacer todas , algunas) con SQL con las extensiones de agrupamiento, etc.
- ❑ El desarrollo de una pequeña reseña de la herramienta asignada de reporting o business intelligence y de cómo la pudieron implementar (o si no pudieron los problemas con los que se encontraron) y el o los reportes del punto 4 que pudieron implementar con dicha herramienta.

Desarrollo

Punto 1

En base al ejemplo de la teoría:

La cooperativa frutihortícola “PatSur” lleva operando 5 años en la región patagónica. Ha ido creciendo y abriendo sucursales en otras ciudades y participa en la provisión de mercaderías a fábricas, otras cooperativas, pymes, cadenas de supermercados. Ha medida que fue ampliando su ámbito de compra y de venta ha ido modificando sus sistemas para manejar nuevos datos como producciones, vendedores y compradores, lugares de compra y venta, transportes, empleados, etc. De allí que cada Gerente de área posea distintos datos en diferentes formas. Por lo tanto la primer idea es tener un información de las ventas. El estado actual de los datos que alimentarán al DW según el origen son:

- El sistema de administración de Clientes.
 - Clientes(Id_Cliente, Nombre, tipo, dirección)
- El sistema de inventario de productos.
 - Producto(Id_Producto, Nombre, id_categoria, precio)
 - Categoria(Id_categoria, descripción)
- Dos (2) sistemas distintos de facturación de productos (a partir del año 2010 hay un sistema que se contrató a una consultora, antes del 2010 había un sistema desarrollado por el sobrino de uno de los socios de la cooperativa).

Sistema Facturación I:

- Clientes (nro_Cliente, Nombre, tipo, dirección)
- Producto (nro_Producto, Nombre, nro_categ, precio_actual)
- Categoria (nro_categ, descripción)
- Venta (Fecha_Vta, nro_Factura, nro_Cliente, Nombre, forma_pago)
- Detalle_Venta (nro_factura, nro_producto, descripción, unidad, precio)

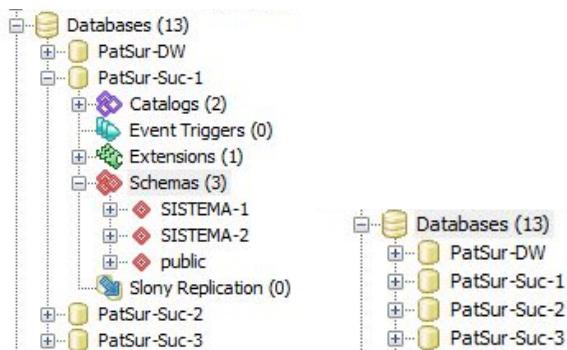
Sistema Facturación II:

- Clientes (cod_Cliente, Nombre, cod_tipo, dirección)
- Tipo_Cliente (cod_tipo, descripción)
- Producto (cod_Producto, Nombre, cod_categoria, cod_subcategoria, precio_actual)
- Categoria (cod_categoria, cod_subcategoria, descripción)
- Venta (Fecha_Vta, Id_Factura, cod_Cliente, Nombre, cod_medio_pago)
- Detalle_Venta (Id_factura, cod_producto, descripción, unidad, precio)
- Medio_Pago (cod_Medio_Pago, descripción, valor, unidad, tipo_operación)

- Tiempo (Id_fecha, día, mes, trimestre, año)

En base a esto y a las consideraciones antes planteadas y para comenzar a realizar este punto del laboratorio lo primero que planteamos es cómo vamos a hacer las bases de datos de las sucursales, y lo que decidimos es que vamos a tener una base por cada sucursal, cada una de las cuales van a ser nombradas como PatSur-Suc-N (donde N es el número de sucursal) y van a contener dos esquemas, uno para el sistema de liquidación viejo (SISTEMA-1) y otro para el sistema de liquidación nuevo (SISTEMA-2).

Por ejemplo para la sucursal 1 tendríamos lo siguiente:



En la imagen anterior podemos ver que las bases las estan creadas dentro del mismo servidor , pero también las hemos implementado sobre máquinas virtuales, y simulando que se encuentran en distintos sitios. Ahora en base a esto, para crear los esquemas con las tablas que se necesitan en cada sistema usamos:

```

3 ----- Creación Sistema-1 -----
4
5 CREATE SCHEMA "SISTEMA-1";
6
7 -- Clientes (nro_Cliente, Nombre, tipo, dirección)
8 CREATE TABLE "SISTEMA-1".CLIENTES(
9     nro_cliente int NOT NULL,
10    nombre varchar(30) NULL,
11    tipo varchar(30) NULL,
12    dirección varchar(30) NULL,
13    CONSTRAINT PK_NRO_CLIENTE PRIMARY KEY (nro_cliente)
14 );
15
16 -- Producto (nro_Producto, Nombre, nro_categ, precio_actual)
17 CREATE TABLE "SISTEMA-1".PRODUCTO(
18     nro_producto int NOT NULL,
19     nombre varchar(30) NULL,
20     nro_categ int NOT NULL,
21     precio_actual float NULL,
22     CONSTRAINT PK_NRO_PRODUCTO PRIMARY KEY (nro_producto)
23 );
24
25 -- Categoría (nro_categ, descripción)
26 CREATE TABLE "SISTEMA-1".CATEGORIA(
27     nro_categ int NOT NULL,
28     descripción varchar(30) NULL,
29     CONSTRAINT PK_NRO_CATEGORIA PRIMARY KEY (nro_categ)
30 );
31

```



```
32      -- Venta (Fecha_Vta, nro_Factura, nro_Cliente, Nombre, forma_pago)
33  □ CREATE TABLE "SISTEMA-1".VENTA(
34      fecha_vta timestamp DEFAULT current_timestamp,
35      nro_factura int NOT NULL,
36      nro_cliente int NOT NULL,
37      nombre varchar(30) NULL,
38      forma_pago char(30) NULL,
39      CONSTRAINT PK_NRO_FACTURA PRIMARY KEY (nro_factura)
40  );
41
42      -- Detalle_Venta(nro_factura, nro_producto, descripción, unidad, precio)
43  □ CREATE TABLE "SISTEMA-1".DETALLE_VENTA(
44      nro_factura int NOT NULL,
45      nro_producto int NOT NULL,
46      descripción varchar(30) NULL,
47      unidad int NULL,
48      precio int NULL
49  );
50
51      ALTER TABLE "SISTEMA-1".PRODUCTO
52      ADD CONSTRAINT FK_PRODUCTO_CATEGORIA_NRO_CATEG FOREIGN KEY(nro_categ)
53      REFERENCES "SISTEMA-1".CATEGORIA (nro_categ)
54      on delete restrict on update restrict;
55
56      ALTER TABLE "SISTEMA-1".VENTA
57      ADD CONSTRAINT FK_VENTA_CLIENTE_NRO_CLIENTE FOREIGN KEY(nro_cliente)
58      REFERENCES "SISTEMA-1".CLIENTES (nro_cliente)
59      on delete restrict on update restrict;
60
61      ALTER TABLE "SISTEMA-1".DETALLE_VENTA
62      ADD CONSTRAINT FK_DETALLE_VENTA_NRO_FACTURA FOREIGN KEY(nro_factura)
63      REFERENCES "SISTEMA-1".VENTA (nro_factura)
64      on delete restrict on update restrict;
65
66      ALTER TABLE "SISTEMA-1".DETALLE_VENTA
67      ADD CONSTRAINT FK_DETALLE_PRODUCTO_NRO_PRODUCTO FOREIGN KEY(nro_producto)
68      REFERENCES "SISTEMA-1".PRODUCTO (nro_producto)
69      on delete restrict on update restrict;
70
71
```

Donde primero creamos el esquema del sistema de liquidación viejo, luego las tablas del esquema, y por último las FOREING KEY de cada una de las tablas.

Y de la misma forma creamos el esquema del sistema de liquidación nuevo:

```
71      -----
72      ----- Creación Sistema-2 -----
73
74
75      CREATE SCHEMA "SISTEMA-2";
76
77
78      -- Clientes (cod_Cliente, Nombre, cod_tipo, dirección)
79  □ CREATE TABLE "SISTEMA-2".CLIENTES(
80      cod_cliente text NOT NULL,
81      nombre varchar(30) NULL,
82      cod_tipo int NOT NULL,
83      dirección varchar(30) NULL,
84      CONSTRAINT PK_COD_CLIENTE PRIMARY KEY (cod_cliente)
85  );
86
```

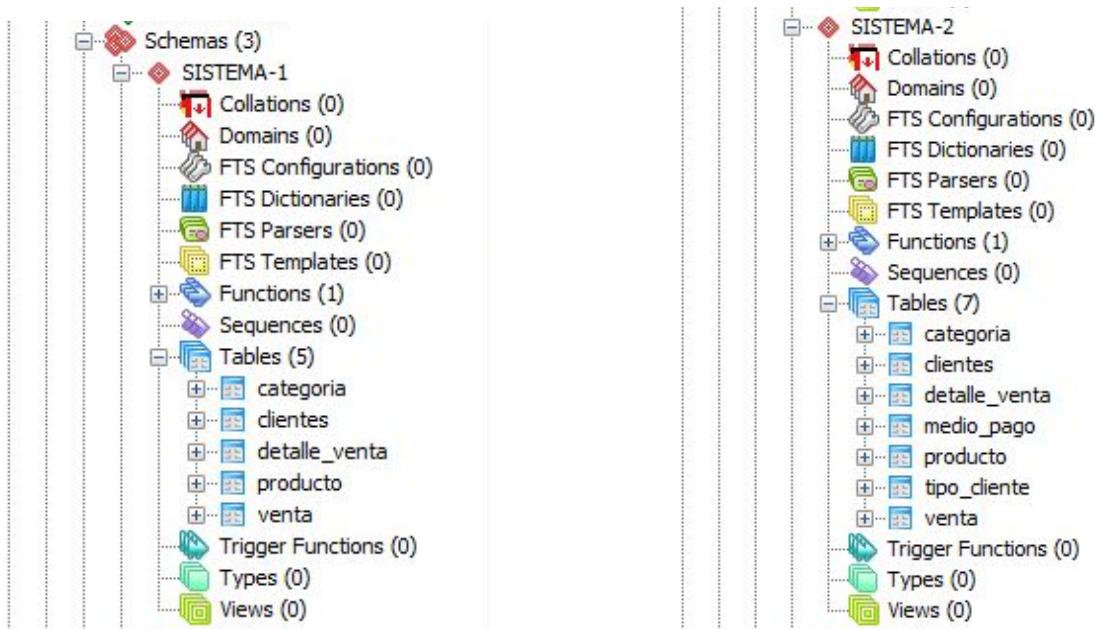


```
87      -- Tipo_Cliente (cod_tipo, descripción)
88  □ CREATE TABLE "SISTEMA-2".TIPO_CLIENTE(
89    cod_tipo int NOT NULL,
90    descripcion varchar(30) NULL,
91    CONSTRAINT PK_COD_TIPO PRIMARY KEY (cod_tipo)
92  );
93
94      -- Producto (cod_Producto, Nombre, cod_categoria, cod_subcategoria, precio_actual)
95  □ CREATE TABLE "SISTEMA-2".PRODUCTO(
96    cod_producto text NOT NULL,
97    nombre varchar(30) NULL,
98    cod_categoria text NOT NULL,
99    cod_subcategoria text NOT NULL,
100   precio_actual float NULL,
101   CONSTRAINT PK_COD_PRODUCTO PRIMARY KEY (cod_producto)
102  );
103
104     -- Categoria (cod_categoria, cod_subcategoria, descripción)
105  □ CREATE TABLE "SISTEMA-2".CATEGORIA(
106    cod_categoria text NOT NULL,
107    cod_subcategoria text NOT NULL,
108    descripcion varchar(30) NULL,
109    CONSTRAINT PK_COD_CATEGORIA PRIMARY KEY (cod_categoria, cod_subcategoria)
110  );
111
112     -- Venta (Fecha_Vta, Id_Factura, cod_Cliente, Nombre, cod_medio_pago)
113  □ CREATE TABLE "SISTEMA-2".VENTA(
114    fecha_vta timestamp DEFAULT current_timestamp,
115    id_factura int NOT NULL,
116    cod_cliente text NOT NULL,
117    nombre varchar(30) NULL,
118    cod_medio_pago int NOT NULL,
119    CONSTRAINT PK_ID_FACTURA PRIMARY KEY (id_factura)
120  );
121
122     -- Detalle_Venta(Id_factura, cod_producto, descripción, unidad, precio)
123  □ CREATE TABLE "SISTEMA-2".DETALLE_VENTA(
124    id_factura int NOT NULL,
125    cod_producto text NOT NULL,
126    descripción varchar(30) NULL,
127    unidad int NULL,
128    precio int NULL
129  );
130
131     -- Medio_Pago( cod_Medio_Pago, descripción, valor, unidad, tipo_operación)
132  □ CREATE TABLE "SISTEMA-2".MEDIO_PAGO(
133    cod_medio_pago int NOT NULL,
134    descripción varchar(30) NULL,
135    valor int NOT NULL,
136    unidad int NULL,
137    tipo_operacion int NULL,
138    CONSTRAINT PK_COD_MEDIO_PAGO PRIMARY KEY (cod_medio_pago)
139  );
140
141 ALTER TABLE "SISTEMA-2".CLIENTES
142 ADD CONSTRAINT FK_CLIENTES_TIPO_COD_TIPO FOREIGN KEY(cod_tipo)
143 REFERENCES "SISTEMA-2".TIPO_CLIENTE (cod_tipo)
144 on delete restrict on update restrict;
145
146 ALTER TABLE "SISTEMA-2".PRODUCTO
147 ADD CONSTRAINT FK_PRODUCTO_SUBCATEGORIA_COD_CATEGORIA_SUBCATEGORIA FOREIGN KEY(cod_categoria, cod_subcategoria)
148 REFERENCES "SISTEMA-2".CATEGORIA (cod_categoria, cod_subcategoria)
149 on delete restrict on update restrict;
150
151 ALTER TABLE "SISTEMA-2".VENTA
152 ADD CONSTRAINT FK_VENTA_CLIENTE_NRO_CLIENTE FOREIGN KEY(cod_cliente)
153 REFERENCES "SISTEMA-2".CLIENTES (cod_cliente)
154 on delete restrict on update restrict;
155
156 ALTER TABLE "SISTEMA-2".VENTA
157 ADD CONSTRAINT FK_VENTA_MEDIO_COD_MEDIO_PAGO FOREIGN KEY(cod_medio_pago)
158 REFERENCES "SISTEMA-2".MEDIO_PAGO (cod_medio_pago)
159 on delete restrict on update restrict;
160
```

```

161 ALTER TABLE "SISTEMA-2".DETALLE_VENTA
162 ADD CONSTRAINT FK_DETALLE_VENTA_ID_FACTURA FOREIGN KEY(id_factura)
163 REFERENCES "SISTEMA-2".VENTA (id_factura)
164 on delete restrict on update restrict;
165
166 ALTER TABLE "SISTEMA-2".DETALLE_VENTA
167 ADD CONSTRAINT FK_DETALLE_PRODUCTO_COD_PRODUCTO FOREIGN KEY(cod_producto)
168 REFERENCES "SISTEMA-2".PRODUCTO (cod_producto)
169 on delete restrict on update restrict;
170
171
    
```

Quedándonos los dos esquemas, con sus respectivas tablas, como sigue:



Punto 2

Para poder realizar las inserciones masivas en las tablas realizamos dos funciones, una para el sistema de liquidación viejo y otro para el nuevo.

La función llenarSistema-1 es para el sistema viejo:

```

173
174 ----- Inserciones Sistema-1 -----
175
176 CREATE OR REPLACE FUNCTION "SISTEMA-1"."llenarSistema-1"(cantidad int) RETURNS VOID AS $$
177 DECLARE
178     "baseCantidadClientes" integer;
179     "limiteCantidadClientes" integer;
180     "nombreC" varchar(30);
181     "apellidoC" varchar(30);
182     "tipoC" varchar(30);
183     "calleC" varchar(30);
    
```



```
184      "numMaxC" integer := 10000;
185      "numeroC" integer;
186      "baseCantidadProductos" integer;
187      "limiteCantidadProductos" integer;
188      "nroCategoriaP" integer;
189      "baseCantidadVentas" integer;
190      "limiteCantidadVentas" integer;
191      "forma_pagoV" varchar(30);
192      "diaMaxV" integer := 365;
193      "diasV" integer;
194      "nroClienteV" int;
195      "nombreClienteV" varchar(30);
196      "nroProductoDV" integer;
197      "nombreProductoDV" varchar(30);
198      "cantidadDetalleVentas" integer;
199      "cantMaxDV" integer := 15;
200      "unidadDV" integer;
201      "unidadMaxDV" integer := 100;
202      "precioDV" integer;
203      "precioMaxDV" integer := 1500;
204      "precioMinDV" integer := 500;
205      "cantidadCategorias" integer;
206      minimo integer := 1;
207      categorias varchar(30) [];

208
209      BEGIN
210          -- carga clientes
211          SELECT MAX(nro_cliente) FROM "SISTEMA-1".clientes INTO "baseCantidadClientes";
212          IF "baseCantidadClientes" IS NULL THEN
213              "baseCantidadClientes" := 0;
214          END IF;
215          "baseCantidadClientes" := "baseCantidadClientes" + minimo;
216          "limiteCantidadClientes" := "baseCantidadClientes" + cantidad - minimo;
217          FOR r IN "baseCantidadClientes" .. "limiteCantidadClientes" LOOP
218              SELECT d.n FROM (SELECT n FROM unnest(ARRAY['Alicia','Marcela','Lidia','Estela','Nora','Nor
219              SELECT d.n FROM (SELECT n FROM unnest(ARRAY['Gonzalez','Rodriguez','Gomez','Fernandez','Log
220              SELECT d.n FROM (SELECT n FROM unnest(ARRAY['publico objetivo','cliente potencial','cliente
221              SELECT d.n FROM (SELECT n FROM unnest(ARRAY['San Martin','Av. Colon','9 de Julio','Cacique
222              "numeroC" := trunc(random() * "numMaxC" + minimo);
223              INSERT INTO "SISTEMA-1".clientes(nro_cliente, nombre, tipo, "dirección") VALUES (r, "apelli
224          END LOOP;
225          -- carga categorias
226          categorias := ARRAY['almacen','panaderia','lacteos','carne vacuna','menudencias', 'carne porcina',
227          SELECT count(nro_categ) FROM "SISTEMA-1".categoria INTO "cantidadCategorias";
228          IF "cantidadCategorias" <> 11 THEN
229              FOR r IN minimo .. 11 LOOP
230                  INSERT INTO "SISTEMA-1".categoria(nro_categ, descripcion) VALUES (r, categorias[r]);
231              END LOOP;
232          END IF;
233          -- carga productos
234          SELECT MAX(nro_producto) FROM "SISTEMA-1".producto INTO "baseCantidadProductos";
235          IF "baseCantidadProductos" IS NULL THEN
236              "baseCantidadProductos" := 0;
237          END IF;
238          "baseCantidadProductos" := "baseCantidadProductos" + minimo;
239          "limiteCantidadProductos" := "baseCantidadProductos" + cantidad - minimo;
240          FOR r IN "baseCantidadProductos" .. "limiteCantidadProductos" LOOP
241              "precioDV" := trunc(random() * "precioMaxDV" + "precioMinDV");
242              SELECT nro_categ FROM "SISTEMA-1".categoria ORDER BY random() LIMIT minimo INTO "nroCategor
243              INSERT INTO "SISTEMA-1".producto(nro_producto, nombre, nro_categ, precio_actual) VALUES (r,
244          END LOOP;
245          -- carga ventas
246          SELECT MAX(nro_factura) FROM "SISTEMA-1".venta INTO "baseCantidadVentas";
247          IF "baseCantidadVentas" IS NULL THEN
248              "baseCantidadVentas" := 0;
249          END IF;
250          "baseCantidadVentas" := "baseCantidadVentas" + minimo;
251          "limiteCantidadVentas" := "baseCantidadVentas" + cantidad - minimo;
252          FOR r IN "baseCantidadVentas" .. "limiteCantidadVentas" LOOP
253              SELECT d.n FROM (SELECT n FROM unnest(ARRAY['contado','debito','credito','transferencia']))
254              "diasV" := trunc(random() * "diaMaxV" + minimo);
255              SELECT nro_cliente FROM "SISTEMA-1".clientes ORDER BY random() LIMIT minimo INTO "nroClient
256              SELECT nombre FROM "SISTEMA-1".clientes WHERE nro_cliente = "nroClienteV" INTO "nombreClien
```



```
257      INSERT INTO "SISTEMA-1".venta(fecha_vta, nro_factura, nro_cliente, nombre, forma_pago) VALUES  
258      -- carga detalles venta  
259      "cantidadDetalleVentas" := trunc(random() * "cantMaxDV" + minimo);  
260      FOR t IN minimo .. "cantidadDetalleVentas" LOOP  
261          SELECT nro_producto FROM "SISTEMA-1".producto ORDER BY random() LIMIT minimo INTO "  
262          SELECT nombre FROM "SISTEMA-1".producto WHERE nro_producto = "nroProductoDV" INTO  
263          "unidadDV" := trunc(random() * "unidadMaxDV" + minimo);  
264          SELECT precio_actual FROM "SISTEMA-1".producto WHERE nro_producto = "nroProductoDV"  
265          INSERT INTO "SISTEMA-1".detalle_venta(nro_factura, nro_producto, "descripción", uni  
266          END LOOP;  
267      END LOOP;  
268      $$ LANGUAGE plpgsql;
```

En los que se llenan con valores aleatorios la mayoría de las tablas, excepto la tabla de categorías, la función recibe como parámetro la cantidad de valores que se desean insertar, y tiene en consideración los valores ya insertados en cada tabla, para poder utilizar la función las veces que sea necesario. Se tiene en consideración que la tabla categoría se carga una sola vez (solo las primer vez que se ejecuta el script), en este caso no se utilizan valores aleatorios y finalmente se establece que el sistema viejo solo se utilizó hasta la fecha 31-12-2015 (Y vencido este plazo se comenzó a utilizar el nuevo sistema).

La función llenarSistema-2 es para el sistema nuevo, es similar a llenarSistema-1, se llenan la mayoría de las tablas con valores aleatorios , excepto las tablas categorías, tipo_cliente y medio_pago, se completan con una cantidad fija de valores, y solo se cargan la primer vez que se ejecuta el script, al igual que la anterior esta función tiene en cuenta las mismas consideraciones y el mismo parámetro de entrada.

```
287      ----- Inserciones Sistema-2 -----  
288  
289  
290  
291      CREATE OR REPLACE FUNCTION "SISTEMA-2"."llenarSistema-2"(cantidad int) RETURNS VOID AS $$  
292      DECLARE  
293          "baseCantidadClientes" integer;  
294          "limiteCantidadClientes" integer;  
295          "nombreC" varchar(30);  
296          "apellidoC" varchar(30);  
297          "tipoClienteIC" varchar(30);  
298          "calleC" varchar(30);  
299          "numMaxC" integer := 10000;  
300          "numeroC" integer;  
301          "descripcionCat" varchar(30);  
302          "baseCantidadProductos" integer;  
303          "limiteCantidadProductos" integer;  
304          "nroCategoriaP" text;  
305          "baseCantidadVentas" integer;  
306          "limiteCantidadVentas" integer;  
307          "forma_pagoMP" varchar(30);  
308          "diaMaxV" integer := 365;  
309          "diasV" integer;  
310          "ultimoID" integer;
```



```
310      "nroClienteV" text;
311      "nombreClienteV" varchar(30);
312      "nroProductoDV" text;
313      "nombreProductoDV" varchar(30);
314      "cantidadDetalleVentas" integer;
315      "cantMaxDV" integer := 15;
316      "unidadDV" integer;
317      "unidadMaxDV" integer := 100;
318      "precioDV" integer;
319      "precioMaxDV" integer := 1500;
320      "precioMinDV" integer := 100;
321      "cantidadCategorias" integer;
322      "codMedioPagoV" integer;
323      "codTipoC" integer;
324      "cantidadTipoClientes" integer;
325      "cantidadMediosPago" integer;
326      "cantSubcategoria" integer;
327      "subCategoriaMax" integer := 15;
328      "nroSubCategoriaP" text;
329      minimo integer := 1;
330      tipoclientes varchar(30) [];
331      categorias varchar(30) [];
332      mediospago varchar(30) [];
333
334      BEGIN
335          -- carga tipo clientes
336          tipoclientes := ARRAY['publico objetivo','cliente potencial','cliente eventual','interno','externo'];
337          SELECT count(cod_tipo) FROM "SISTEMA-2".tipo_cliente INTO "cantidadTipoClientes";
338          IF "cantidadTipoClientes" <> 5 THEN
339              FOR r IN minimo .. 5 LOOP
340                  INSERT INTO "SISTEMA-2".tipo_cliente(cod_tipo, descripcion)VALUES (r, tipoclientes[END LOOP;
341          END IF;
342          -- carga clientes
343          SELECT MAX(hex_to_int(cod_cliente)) FROM "SISTEMA-2".clientes INTO "baseCantidadClientes";
344          IF "baseCantidadClientes" IS NULL THEN
345              "baseCantidadClientes" := 0;
346          END IF;
347          "baseCantidadClientes" := "baseCantidadClientes" + minimo;
348          "limiteCantidadClientes":= "baseCantidadClientes" + cantidad - minimo;
349          FOR r IN "baseCantidadClientes" .. "limiteCantidadClientes" LOOP
350              SELECT d.n FROM (SELECT n FROM unnest(ARRAY['Alicia','Marcela','Lidia','Estela','Nora','Nor
351              SELECT d.n FROM (SELECT n FROM unnest(ARRAY['Gonzalez','Rodriguez','Gomez','Fernandez','Lop
352              SELECT d.n FROM (SELECT n FROM unnest(ARRAY['San Martin','Av. Colon','9 de Julio','Cacique
353              SELECT cod_tipo FROM "SISTEMA-2".tipo_cliente ORDER BY random() LIMIT minimo INTO "codTipoc
354              "numeroC" := trunc(random() * "numMaxC" + minimo);
355              INSERT INTO "SISTEMA-2".clientes(cod_cliente, nombre, cod_tipo, "dirección") VALUES (to_hex
356              END LOOP;
357              -- carga categorias
358              categorias := ARRAY['almacen','panaderia','lacteos','carne vacuna','menudencias','carne porcina',
359              SELECT count(cod_categoria) FROM "SISTEMA-2".categoria INTO "cantidadCategorias";
360              IF "cantidadCategorias" < 11 THEN
361                  FOR r IN minimo .. 11 LOOP
362                      "cantSubcategoria" := trunc(random() * "subCategoriaMax" + minimo);
363                      FOR t IN minimo .. "cantSubcategoria" LOOP
364                          INSERT INTO "SISTEMA-2".categoria(cod_categoria, cod_subcategoria, descripcion)
365                          END LOOP;
366                  END LOOP;
367              END IF;
368              -- carga productos
369              SELECT MAX(hex_to_int(cod_producto)) FROM "SISTEMA-2".producto INTO "baseCantidadProductos";
370              IF "baseCantidadProductos" IS NULL THEN
371                  "baseCantidadProductos" := 0;
372              END IF;
373              "baseCantidadProductos" := "baseCantidadProductos" + minimo;
374              "limiteCantidadProductos" := "baseCantidadProductos" + cantidad - minimo;
375              FOR r IN "baseCantidadProductos" .. "limiteCantidadProductos" LOOP
376                  "precioDV" := trunc(random() * "precioMaxDV" + "precioMinDV");
377                  SELECT cod_categoria FROM "SISTEMA-2".categoria ORDER BY random() LIMIT minimo INTO "nroCat
378                  SELECT cod_subcategoria FROM "SISTEMA-2".categoria WHERE cod_categoria = "nroCategoriaP" OF
379                  INSERT INTO "SISTEMA-2".producto(cod_producto, nombre, cod_categoria, cod_subcategoria, pre
380                  END LOOP;
```



```
382      -- carga medios de pago
383      mediospago := ARRAY['contado','debito','credito','transferencia'];
384      SELECT count(cod_medio_pago) FROM "SISTEMA-2".medio_pago INTO "cantidadMediosPago";
385      IF "cantidadMediosPago" <> 4 THEN
386          FOR r IN minimo .. 4 LOOP
387              INSERT INTO "SISTEMA-2".medio_pago(cod_medio_pago, "descripción", valor, unidad, ti
388          END LOOP;
389      END IF;
390      -- carga ventas
391      SELECT MAX(id_factura) FROM "SISTEMA-2".venta INTO "baseCantidadVentas";
392      IF "baseCantidadVentas" IS NULL THEN
393          "baseCantidadVentas" := 0;
394      END IF;
395      "baseCantidadVentas" := "baseCantidadVentas" + minimo;
396      "limiteCantidadVentas" := "baseCantidadVentas" + cantidad - minimo;
397      FOR r IN "baseCantidadVentas" .. "limiteCantidadVentas" LOOP
398          SELECT cod_medio_pago FROM "SISTEMA-2".medio_pago ORDER BY random() LIMIT minimo INTO "codM
399          "diasV" := trunc(random() * "diaMaxV" + minimo);
400          SELECT cod_cliente FROM "SISTEMA-2".clientes ORDER BY random() LIMIT minimo INTO "nroClient
401          SELECT nombre FROM "SISTEMA-2".clientes WHERE cod_cliente = "nroClienteV" INTO "nombreClien
402          INSERT INTO "SISTEMA-2".venta(fecha_vta, id_factura, cod_cliente, nombre, cod_medio_pago) V
403          -- carga detalles venta
404          "cantidadDetalleVentas" := trunc(random() * "cantMaxDV" + minimo);
405          FOR t IN minimo .. "cantidadDetalleVentas" LOOP
406              SELECT cod_producto FROM "SISTEMA-2".producto ORDER BY random() LIMIT minimo INTO "codP
407              SELECT nombre FROM "SISTEMA-2".producto WHERE cod_producto = "nroProductoDV" INTO "nroProd
408              "unidadDV" := trunc(random() * "unidadMaxDV" + minimo);
409              SELECT precio_actual FROM "SISTEMA-2".producto WHERE cod_producto = "nroProductoDV" V
410              INSERT INTO "SISTEMA-2".detalle_venta(id_factura, cod_producto, "descripción", unid
411          END LOOP;
412      END LOOP;
413  END;
414  $$ LANGUAGE plpgsql;
```

Como en el sistema nuevo se manejan códigos, en la mayoría de los códigos utilizamos números hexadecimales por lo necesitamos una función de convierta de hexadecimal a decimal.

```
272
273
274      ----- funcion hex_to_int -----
275
276
277      CREATE OR REPLACE FUNCTION hex_to_int(hexval varchar) RETURNS integer AS $$
278      DECLARE
279          result int;
280      BEGIN
281          EXECUTE 'SELECT x''' || hexval || '''::int' INTO result;
282          RETURN result;
283      END;
284  $$;
285  LANGUAGE 'plpgsql' IMMUTABLE STRICT;
```

Para poder usar estas funciones, hacemos:

```

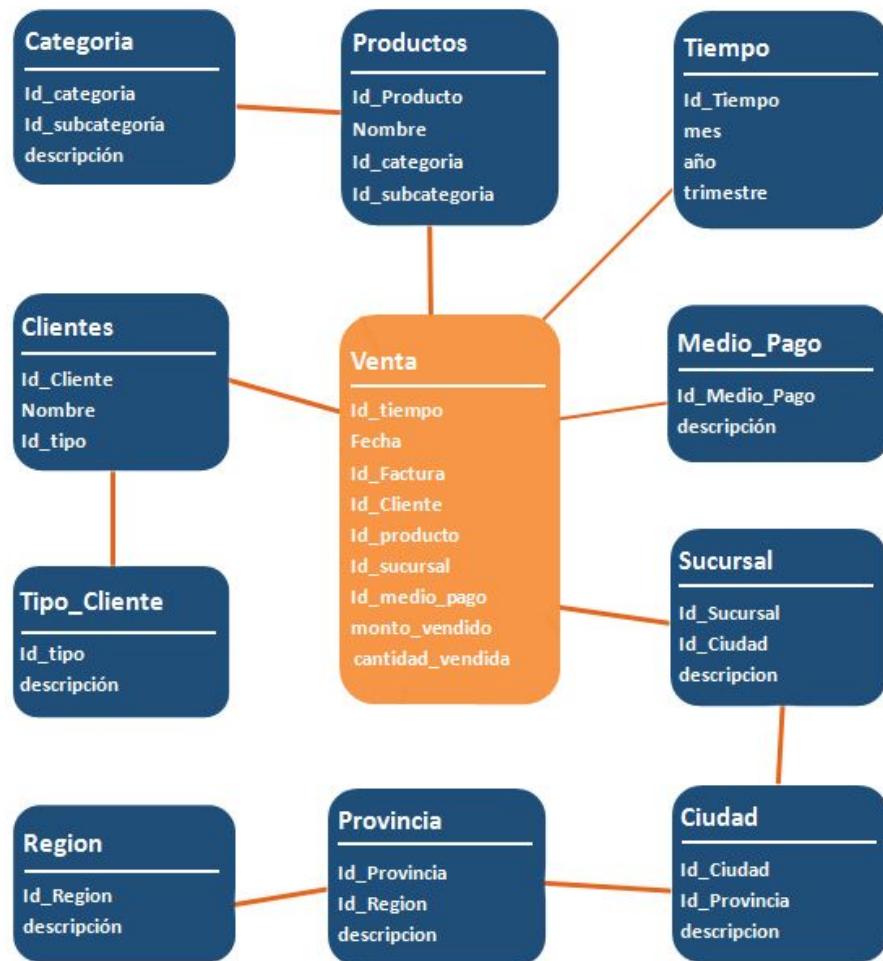
270   SELECT "SISTEMA-1"."llenarSistema-1"(1000);
271
272
415
416   SELECT "SISTEMA-2"."llenarSistema-2"(1000);
417

```

Y las ejecutamos un par de veces por sistema para poder completar las tablas. En la implementación final decidimos ejecutar las funciones con menos valores (300).

Punto 3

En este punto decidimos implementar el Data Warehouse (DW) en otra base de datos, de acuerdo al siguiente esquema:





Donde:

- ❑ Tabla de hechos:
 - ❑ Venta (Id_tiempo, Fecha, Id_Factura, Id_Cliente, Id_Producto, Id_Sucursal, Id_medio_pago, monto_vendido, cantidad_vendida)
- ❑ Tablas de dimensión:
 - ❑ Clientes (Id_Cliente, Nombre, Id_tipo)
 - ❑ Tipo_Cliente (Id_tipo, descripción)
 - ❑ Productos (Id_Producto, Nombre, Id_categoria, Id_subcategoria)
 - ❑ Categoria (Id_categoria, Id_subcategoria, descripción)
 - ❑ Medio_Pago (Id_Medio_Pago, descripción)
 - ❑ Región (Id_region, descripción)
 - ❑ Provincia (Id_provincia, descripción, Id_Región)
 - ❑ Ciudad (Id_Ciudad, descripción, Id_Provincia)
 - ❑ Sucursal (Id_Sucursal, descripcion, Id_Ciudad)
 - ❑ Tiempo (Id_Tiempo, mes, año, trimestre)

```
85
86  CREATE TABLE MEDIO_PAGO(
87      Id_MedioPago int NOT NULL,
88      descripción varchar(30) NULL,
89      CONSTRAINT PK_COD_MEDIO_PAGO PRIMARY KEY (Id_MedioPago)
90  );
91
92  INSERT INTO MEDIO_PAGO (Id_MedioPago,descripción)
93      SELECT cod_medio_pago,descripción FROM dblink('conect_suci','SELECT cod_medio_pago, descripción FRC
94
95  -- Categoría (cod_categoria, cod_subcategoria, descripción)
96  CREATE TABLE CATEGORIA(
97      Id_Categoría text NOT NULL,
98      Id_subcategoria text NOT NULL,
99      descripción varchar(30) NULL,
100     CONSTRAINT PK_ID_CATEGORIA PRIMARY KEY (Id_Categoría, Id_subcategoria)
101  );
102
103  INSERT INTO CATEGORIA (Id_Categoría, Id_subcategoria, descripción)
104      SELECT cod_categoria, cod_subcategoria, descripción FROM dblink('conect_suci','SELECT cod_categoria
105
106  -- Tipo_Cliente (Id_Tipo, descripción)
107  CREATE TABLE TIPO_CLIENTE(
108      Id_Tipo int NOT NULL,
109      descripción varchar(30) NULL,
110      CONSTRAINT PK_ID_TIPO PRIMARY KEY (Id_Tipo)
111  );
112
113  INSERT INTO TIPO_CLIENTE (Id_Tipo, descripción)
114      SELECT cod_tipo, descripción FROM dblink('conect_suci','SELECT cod_tipo, descripción FROM "SISTEMA-
```



```
116    --TABLA SUCURSAL
117    □ CREATE TABLE SUCURSAL (
118        Id_Sucursal int NOT NULL,
119        descripcion varchar(30) NULL,
120        Id_Ciudad int NOT NULL,
121        CONSTRAINT PK_SUCURSAL PRIMARY KEY (Id_Sucursal)
122    );
123
124    □ CREATE TABLE CIUDAD (
125        Id_Ciudad int NOT NULL,
126        descripcion varchar(30) NULL,
127        Id_Provincia int NOT NULL,
128        CONSTRAINT PK_CIUDAD PRIMARY KEY (Id_Ciudad)
129    );
130
131    ALTER TABLE SUCURSAL
132    ADD CONSTRAINT FK_CIUDAD FOREIGN KEY(Id_Ciudad)
133    REFERENCES CIUDAD (Id_Ciudad)
134    on delete restrict on update restrict;
135
136    □ CREATE TABLE PROVINCIA (
137        Id_Provincia int NOT NULL,
138        descripcion varchar(30) NULL,
139        Id_Region int NOT NULL,
140        CONSTRAINT PK_PROVINCIA PRIMARY KEY (Id_Provincia)
141    );
142
143    ALTER TABLE CIUDAD
144    ADD CONSTRAINT FK_PROVINCIA FOREIGN KEY(Id_Provincia)
145    REFERENCES PROVINCIA (Id_Provincia)
146    on delete restrict on update restrict;
147
148    □ CREATE TABLE REGION (
149        Id_Region int NOT NULL,
150        descripcion varchar(30) NULL,
151        CONSTRAINT PK_REGION PRIMARY KEY (Id_Region)
152    );
153
154    ALTER TABLE PROVINCIA
155    ADD CONSTRAINT FK_REGION FOREIGN KEY(Id_Region)
156    REFERENCES REGION (Id_Region)
157    on delete restrict on update restrict;
158
159    □ CREATE TABLE TIEMPO (
160        Id_Tiempo serial,
161        mes int NOT NULL,
162        año int NOT NULL,
163        trimestres int NOT NULL,
164        CONSTRAINT PK_TIEMPO PRIMARY KEY (Id_Tiempo)
165    );
166
167    □ CREATE TABLE PRODUCTOS (
168        Id_Producto int NOT NULL,
169        Id_Categoría text NOT NULL,
170        Id_subcategoria text NOT NULL,
171        nombre varchar(30) NOT NULL,
172        CONSTRAINT PK_PRODUCTOS PRIMARY KEY (Id_Producto)
173    );
174
175    ALTER TABLE PRODUCTOS
176    ADD CONSTRAINT FK_CATEGORIA FOREIGN KEY (Id_Categoría,Id_subcategoria)
177    REFERENCES CATEGORIA(Id_Categoría,Id_subcategoria);
178
179    □ CREATE TABLE CLIENTES (
180        Id_Cliente int NOT NULL,
181        nombre text NOT NULL,
182        Id_Tipo int NOT NULL,
183        CONSTRAINT PK_CLIENTES PRIMARY KEY (Id_Cliente)
184    );
185
186    ALTER TABLE CLIENTES
187    ADD CONSTRAINT FK_TIPO_CLIENTE FOREIGN KEY (Id_Tipo) REFERENCES TIPO_CLIENTE (Id_Tipo);
188
```



```
190
191 ┌─ CREATE TABLE VENTAS (
192   └─ Id_Tiempo int,
193     fecha timestamp,
194     Id_Factura int,
195     Id_Cliente int,
196     Id_Producto int,
197     Id_Sucursal int,
198     Id_MedioPago int,
199     monto_vendido real,
200     cantidad_vendida int
201   );
202
```

En lo que se puede observar que las tablas Medio_Pago, Categoría, y Tipo_Cliente las traemos directamente del sistema nuevo y tenemos especial cuidado en el momento de traer las categorías del sistema nuevo de todas las sucursales (porque al tener definida una cantidad de subcategorías aleatorias, el tamaño de las tablas categoría es distinto en todas la sucursales).

El resto las tablas las completamos usando una tabla temporal a la que llamamos tmpVentas y las tablas de equivalencia de clientes y productos (Proceso ETL). Conectándonos a las bases de datos de las sucursales mediante dblink.

```
1
2 ┌─ CREATE EXTENSION dblink;
3
4 ┌─ SELECT dblink_connect('conect_suci', 'hostaddr=192.168.1.112 port=5432 dbname=PatSur-Suci user=postgres pas
5
6
```

La tabla temporal tmpVentas las implementamos como sigue:

```
226
227 ┌─ CREATE TABLE tmpVentas (
228   └─ Id_Tiempo int,
229     fecha_vta timestamp,
230     Id_Factura int,
231     Id_Cliente text,
232     Id_Producto text,
233     Id_Sucursal int,
234     Id_medio_pago int,
235     monto_vendido real,
236     cantidad_vendida real,
237     nombre_producto varchar(30),
238     Id_categoria text,
239     Id_subcategoria text ,
240     nombre_cliente varchar(30),
241     tipo_cliente int
242   );
243
```

Y para cargarla, tenemos dos funciones, una para cada sistema:



- ❑ CargaTmpVentas(pSuc integer, pMes integer, pAño integer), para el sistema viejo.
- ❑ CargaTmpVentasSN(pSuc integer, pMes integer, pAño integer), para el sistema nuevo.

Las dos reciben como parámetros el n° de sucursal, junto con el mes y año de la fecha de importación.

```
245  |
246  |--Script ETL - extraccion de datos de ventas desde el sistema de facturacion viejo
247  CREATE OR REPLACE FUNCTION CargaTmpVentas(pSuc integer, pMes integer, pAño integer) RETURNS VOID AS
248  $$
249  DECLARE
250
251  BEGIN
252      INSERT INTO tmpVentas(fecha_vta, Id_Factura, Id_Cliente, Id_Producto, Id_Sucursal, Id_medio_pago, n
253          Id_categoria, nombre_cliente, tipo_cliente)
254      SELECT fecha_vta, Id_Factura, Id_Cliente, Id_Producto, Id_Sucursal, Id_medio_pago, monto_vendido, c
255          Id_categoria, nombre_cliente, tipo_cliente FROM
256          (SELECT fecha_vta, Id_Factura, Id_Cliente, Id_Producto, Id_Sucursal, Id_medio_pago, monto_vendido,
257              cantidad_vendida, nombre_producto, C.id_categoria as Id_categoria, nombre_cliente, tipo_cliente FRC
258          (SELECT fecha_vta, Id_Factura, Id_Cliente, Id_Producto, Id_Sucursal, Id_medio_pago,
259              monto_vendido, cantidad_vendida, nombre_producto, descripc_categ, nombre_cliente, TC.id_tipo AS tipo
260          FROM (SELECT * FROM dblink ('conect_suc1', 'SELECT fecha_vta, v.nro_factura, CAST (c.nro_cliente as
261              CASE v.forma_pago WHEN ''contado'' THEN 1 WHEN ''debito'' THEN 2 WHEN ''credito'' THEN 3 WHEN ''tra
262                  unidad * precio as monto_vendido, unidad as cantidad_vendida, p.nombre, cat.descripcion as descripc_
263                  FROM "SISTEMA-1".venta v, "SISTEMA-1".detalle_venta dv, "SISTEMA-1".clientes c, "SISTEMA-1".product
264                  WHERE v.nro_Factura = dv.nro_Factura and v.nro_cliente = c.nro_cliente and dv.nro_producto = p.nro_
265                  and date_part (''year'', fecha_vta) = ' || CAST(pAño AS text))
266          AS tmpvent (fecha_vta timestamp, Id_Factura int, Id_Cliente text, Id_Producto text, Id_Sucursal int
267              descripc_categ text, nombre_cliente varchar(30), tipo_cliente varchar(30)))
268          AS I INNER JOIN tipo_cliente AS TC ON (I.tipo_cliente = TC.descripcion)
269          AS TCI INNER JOIN (select descripcion, id_categoria from categoria group by descripcion, id_categoria
270          UPDATE tmpventas SET Id_Tiempo = InsertarTiempo(pMes, pAño) WHERE Id_Tiempo IS NULL;
271
272      $$ LANGUAGE plpgsql;
273
```

Donde la diferencia entre una y otra son los CAST y JOIN que se deben realizar en uno u otro caso, y que para el sistema viejo es necesario cargar las subcategorías (que no tiene) y unificar las categorías con las categorías del sistema nuevo.

```
275  |
276  |--Script ETL - extraccion de datos de ventas desde el sistema de facturacion nuevo
277  CREATE OR REPLACE FUNCTION CargaTmpVentasSN(pSuc integer, pMes integer, pAño integer) RETURNS VOID AS
278  $$
279  DECLARE
280
281  BEGIN
282      INSERT INTO tmpventas(fecha_vta, Id_Factura, Id_Cliente, Id_producto, Id_Sucursal, Id_medio_pago, n
283          nombre_producto, Id_categoria, Id_subcategoria, nombre_cliente, tipo_cliente)
284      SELECT * FROM dblink ('conect_suc1', 'SELECT fecha_vta, v.id_factura, c.cod_cliente, p.cod_producto
285          v.cod_medio_pago, unidad * precio as monto_vendido, unidad as cantidad_vendida, p.nombre, p.
286          FROM "SISTEMA-2".venta v, "SISTEMA-2".detalle_venta dv, "SISTEMA-2".clientes c, "SISTEMA-2"
287          WHERE v.id_factura = dv.id_factura and v.cod_cliente = c.cod_cliente and dv.cod_producto =
288              date_part (''month'', fecha_vta) = ' || CAST(pMes AS text) || ' and date_part (''year'', fe
289          AS tmpvent (fecha_vta timestamp, Id_Factura int, Id_Cliente text, Id_Producto text, Id_Sucu
290              cantidad_vendida real, nombre_producto varchar(30), Id_categoria text, Id_subcategoria text
291          UPDATE tmpventas SET Id_Tiempo = InsertarTiempo(pMes, pAño) WHERE Id_Tiempo IS NULL;
292
293      $$ LANGUAGE plpgsql;
```



En el caso de del sistema nuevo , el script resulta ser mucho más sencillo.

Junto con estas importaciones, realizamos la actualización de la tabla Tiempo, para poder tomar el Id_Tiempo que necesitamos en la tabla tmpVentas, para hacer esto usamos una función:

- InsertarTiempo(mesI integer, añoI integer) RETURNS integer

Que de acuerdo al mes y año que recibe retorna el Id_Tiempo correspondiente. La forma en que lo realizamos es: si el mes y año ya están cargados, retorna el Id_Tiempo, y si no están cargados, hace el INSERT del mes , año y trimestre (calculando el trimestre) para finalmente retornar el Id_Tiempo.

```
203  CREATE OR REPLACE FUNCTION InsertarTiempo(mesI integer, añoI integer) RETURNS integer AS
204  $$
205  DECLARE
206      id integer;
207  BEGIN
208      SELECT id_tiempo FROM public.tiempo WHERE mes=mesI and "año"= añoI into id;
209      IF id IS NULL THEN
210          CASE
211              WHEN (mesI >= 1 AND mesI <= 3) THEN
212                  INSERT INTO TIEMPO (mes,año,trimestres) VALUES (mesI,añoI,1);
213              WHEN (mesI >= 4 AND mesI <= 6) THEN
214                  INSERT INTO TIEMPO (mes,año,trimestres) VALUES (mesI,añoI,2);
215
216              WHEN (mesI >= 7 AND mesI <= 9) THEN
217                  INSERT INTO TIEMPO (mes,año,trimestres) VALUES (mesI,añoI,3);
218              ELSE
219                  INSERT INTO TIEMPO (mes,año,trimestres) VALUES (mesI,añoI,4);
220          END CASE;
221          SELECT id_tiempo FROM public.tiempo WHERE mes=mesI and "año"= añoI into id;
222      END IF;
223      RETURN id;
224  END;
225  $$ LANGUAGE plpgsql;
```

Por lo que, al ejecutar funciones de carga de tmpVentas podemos obtener, por ejemplo, algo así:



	id_tempo integer	fecha_vta timestamp w integer	id_factura integer	id_cliente text	id_producto text	id_sucursal integer	id_medio_pa integer	monto_venta real	cantidad_ver real	nombre_prod character vari	id_categoria text	id_subcatego text	nombre_cien character vari	tipo_cliente integer
537	1	2018-12-26	157	927	712	1	4	38960	20	Producto 71aae			Sanchez, Fa 5	
538	1	2018-12-26	157	927	935	1	4	26280	40	Producto 93ab4			Sanchez, Fa 5	
539	1	2018-12-26	157	927	606	1	4	52800	32	Producto 60ab0			Sanchez, Fa 5	
540	1	2018-12-26	157	927	31	1	4	52218	27	Producto 31aac			Sanchez, Fa 5	
541	1	2018-12-25	25	522	912	1	3	59826	39	Producto 91aaf			Perez, Fati 5	
542	1	2018-12-25	25	522	192	1	3	168542	94	Producto 19aad			Perez, Fati 5	
543	1	2018-12-25	25	522	575	1	3	6492	12	Producto 57ab0			Perez, Fati 5	
544	1	2018-12-25	25	522	315	1	3	78078	78	Producto 31ab4			Perez, Fati 5	
545	1	2018-12-20	11	554	570	1	3	42930	45	Producto 57aac			Gomez, Silv 5	
546	1	2018-12-20	11	554	251	1	3	30912	16	Producto 25aaf			Gomez, Silv 5	
547	1	2018-12-20	11	554	966	1	3	161210	94	Producto 96aaf			Gomez, Silv 5	
548	1	2018-12-20	11	554	518	1	3	95832	66	Producto 51aad			Gomez, Silv 5	
549	1	2018-12-05	1272	bfa	cc5	1	2	3026	17	Producto 53aac	aac		Romero, Noe 3	
550	1	2018-12-27	1809	bac	ac4	1	1	122229	81	Producto 26ab5	ab6		Gomez, Noe 5	
551	1	2018-12-31	2	b2f	d31	1	4	77058	54	Producto 64ab3	aee		Perez, Noem 5	
552	1	2018-12-31	2	b2f	b6f	1	4	97193	83	Producto 19aaf	aab		Perez, Noem 5	
553	1	2018-12-31	2	b2f	b50	1	4	53922	43	Producto 16aee	ab6		Perez, Noem 5	
554	1	2018-12-31	2	b2f	c3a	1	4	4224	22	Producto 40ab4	aab		Perez, Noem 5	
555	1	2018-12-31	2	b2f	ca5	1	4	22272	58	Producto 50ab5	aaf		Perez, Noem 5	
556	1	2018-12-31	2	b2f	e15	1	4	57798	39	Producto 27aac	ab6		Perez, Noem 5	
557	1	2018-12-31	2	b2f	c91	1	4	31614	66	Producto 48aee	ab8		Perez, Noem 5	
558	1	2018-12-31	2	b2f	b56	1	4	7887	11	Producto 17aaf	aaf		Perez, Noem 5	
559	1	2018-12-31	2	b2f	e52	1	4	55275	67	Producto 93ab0	aaf		Perez, Noem 5	
560	1	2018-12-31	2	b2f	d63	1	4	148990	94	Producto 69ab2	aad		Perez, Noem 5	

A las tablas de equivalencia de cliente TECliente y productos TEProductos, las implementamos como sigue:

```
8
9  CREATE TABLE TECliente (
10    cdw serial,
11    cvs integer DEFAULT NULL,
12    cns text DEFAULT NULL,
13    CONSTRAINT pk_tecliente PRIMARY KEY (cdw)
14  );
15
16
17  CREATE TABLE TEProductos (
18    pdw serial,
19    pvs integer DEFAULT NULL,
20    pns text DEFAULT NULL,
21    CONSTRAINT pk_teproductos PRIMARY KEY (pdw)
22  );
23
24
```

Y en este caso, llenamos las tablas de equivalencia con las funciones:

- CargaTEClientes(porcentajeEquivalentes int, suc_db text)
- CargaTEProductos(porcentajeEquivalentes int, suc_db text)

Donde cargamos en las tablas de equivalencia los id del sistema viejo, luego con los códigos del sistema nuevo, y en forma aleatoria en base a un porcentaje que le pasamos a la función, realizamos los UPDATE y DELETE para que en la tabla de equivalencia una cierta cantidad de



clientes o productos resulten equivalentes (En cuanto al id del sistema viejo y el código del sistema nuevo). Para hacer esto usamos las funciones:

```
204 -- Script ETL - Carga tabla de equivalencia de clientes
205 CREATE OR REPLACE FUNCTION CargaTEClientes(porcentajeEquivalentes int, suc_db text) RETURNS VOID AS
206 $$
207 DECLARE
208     clienteSN text;
209     clienteSV integer;
210     IDclienteSN integer;
211     IDclienteSV integer;
212     totalClientes integer;
213     cantidadEquivalentes integer;
214     res_conect text;
215     cdw_ini integer;
216
217     BEGIN
218         SELECT dblink_connect('conect_suc', 'port=5434 dbname=' || suc_db || ' user=postgres password=davi
219         cdw_ini := (SELECT max(cdw) FROM TECliente);
220         IF cdw_ini IS NULL THEN
221             cdw_ini := 0;
222         END IF;
223         INSERT INTO TECliente (cvs) SELECT * FROM dblink ('conect_suc', 'SELECT nro_cliente FROM "SISTEMA-
224         INSERT INTO TECliente (cns) SELECT * FROM dblink ('conect_suc', 'SELECT cod_cliente FROM "SISTEMA-
225         totalClientes := (SELECT max(cdw) FROM TECliente) - cdw_ini;
226         cantidadEquivalentes := (totalClientes * porcentajeEquivalentes)/100;
227         FOR r IN 1 .. cantidadEquivalentes LOOP
228             SELECT cdw FROM tecliente WHERE (cdw>cdw_ini and cns IS NOT NULL and cvs IS NULL) ORDER BY
229                 SELECT cns FROM tecliente WHERE cdw = IDclienteSN INTO clienteSN;
230                 DELETE FROM tecliente WHERE cdw = IDclienteSN;
231                 SELECT cdw FROM tecliente WHERE (cdw>cdw_ini and cns IS NULL and cvs IS NOT NULL) ORDER BY
232                     UPDATE tecliente SET cns=clienteSN WHERE cdw=IDclienteSV;
233             END LOOP;
234             SELECT dblink_disconnect('conect_suc') into res_conect;
235         END;
236         $$ LANGUAGE plpgsql;
237
238
```

Donde tenemos la precaución de que solo se inserten los clientes que no estén cargados (La primer vez se cargan todos, pero a la siguiente vez solo carga los clientes que son nuevos).

```
242 -- Script ETL - Carga tabla de equivalencia de productos
243 CREATE OR REPLACE FUNCTION CargaTEProductos(porcentajeEquivalentes int, suc_db text) RETURNS VOID AS
244 $$
245 DECLARE
246     productoSN text;
247     productoSV integer;
248     IDproductoSN integer;
249     IDproductoSV integer;
250     totalProductos integer;
251     cantidadEquivalentes integer;
252     res_conect text;
253     pdw_ini integer;
254
255     BEGIN
256         SELECT dblink_connect('conect_suc', 'port=5434 dbname=' || suc_db || ' user=postgres password=davi
257         pdw_ini := (SELECT max(pdw) FROM TEPuestos);
258         IF pdw_ini IS NULL THEN
259             pdw_ini := 0;
260         END IF;
261         INSERT INTO tepuestos (pvs) SELECT * FROM dblink ('conect_suc', 'SELECT nro_producto FROM "SISTE
262         INSERT INTO tepuestos (pns) SELECT * FROM dblink ('conect_suc', 'SELECT cod_producto FROM "SISTE
263         totalProductos := (SELECT max(pdw) FROM TEPuestos) - pdw_ini;
264         cantidadEquivalentes := (totalProductos * porcentajeEquivalentes)/100;
265         FOR r IN 1 .. cantidadEquivalentes LOOP
266             
```

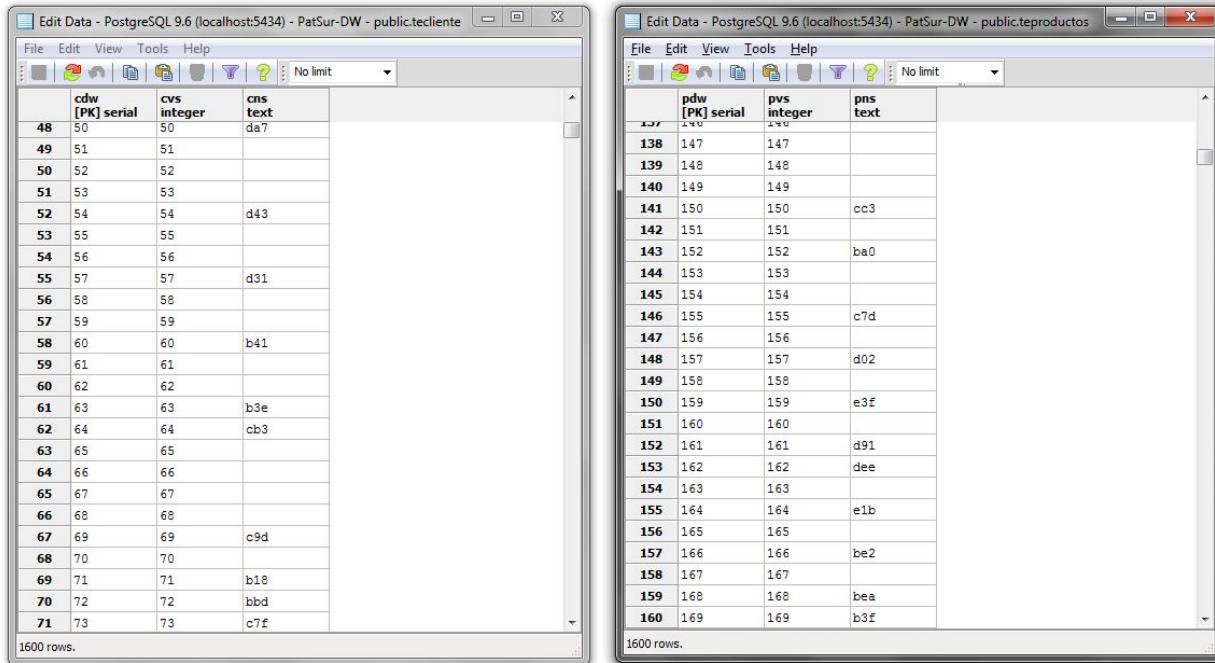
```

267      SELECT pdw FROM teproductos WHERE (pdw>pdw_ini and pns IS NOT NULL and pvs IS NULL) ORDER
268      SELECT pns FROM teproductos WHERE pdw = IDproductoSN INTO productoSN;
269      DELETE FROM teproductos WHERE pdw = IDproductoSN;
270      SELECT pdw FROM teproductos WHERE (pdw>pdw_ini and pns IS NULL and pvs IS NOT NULL) ORDER
271      UPDATE teproductos SET pns=productoSN WHERE pdw=IDproductoSV;
272      END LOOP;
273      SELECT dblink_disconnect('connect_suc') into res_conect;
274  END;
275  $$ LANGUAGE plpgsql;
276

```

Donde al igual que en la tabla de equivalencia de clientes, sólo se cargan los productos nuevos.

Y como resultado al ejecutarlas podemos ver, por ejemplo, lo siguiente:



The image shows two side-by-side PostgreSQL data editors. The left window is titled 'Edit Data - PostgreSQL 9.6 (localhost:5434) - PatSur-DW - public.tecliente'. It contains a table with three columns: 'cdw [PK] serial', 'cvs integer', and 'cns text'. The right window is titled 'Edit Data - PostgreSQL 9.6 (localhost:5434) - PatSur-DW - public.teproductos'. It contains a table with three columns: 'pdw [PK] serial', 'pvs integer', and 'pns text'. Both windows show several rows of data, with the last row in each being highlighted in red.

cdw [PK] serial	cvs integer	cns text
48	50	da7
49	51	51
50	52	52
51	53	53
52	54	d43
53	55	55
54	56	56
55	57	57
56	58	d31
57	59	59
58	60	b41
59	61	61
60	62	62
61	63	b3e
62	64	cb3
63	65	65
64	66	66
65	67	67
66	68	
67	69	c9d
68	70	
69	71	b18
70	72	bbd
71	73	c7f

pdw [PK] serial	pvs integer	pns text
138	147	147
139	148	148
140	149	149
141	150	cc3
142	151	151
143	152	ba0
144	153	153
145	154	154
146	155	c7d
147	156	156
148	157	d02
149	158	158
150	159	e3f
151	160	160
152	161	d91
153	162	dee
154	163	163
155	164	e1b
156	165	165
157	166	be2
158	167	167
159	168	bea
160	169	b3f

Una vez que ya tenemos las tablas de equivalencia de clientes y productos, y tmpVentas completas realizamos los INSERT en el resto de las tablas del DW (clientes, productos y ventas). Para esto tenemos la función:

- ❑ Carga_CPV()



```
337  -- Script ETL - Carga las tablas del DW
338  CREATE OR REPLACE FUNCTION Carga_CPV() RETURNS VOID AS
339  $$
340  DECLARE
341
342  BEGIN
343      --ingreso de clientes del viejo sistema desde tmpVentas
344      INSERT INTO Clientes
345          SELECT DISTINCT cdw, cortarApellidos(nombre_cliente), tipo_cliente
346          FROM tmpVentas tmpv, TECliente tec
347          WHERE tmpv.id_cliente = CAST (tec.csv as text) AND tec.cdw not in (SELECT id_cliente from
348          --ingreso de clientes del nuevo sistema desde tmpVentas
349      INSERT INTO Clientes
350          SELECT DISTINCT cdw, cortarApellidos(nombre_cliente), tipo_cliente
351          FROM tmpVentas tmpv, TECliente tec
352          WHERE tmpv.id_cliente = tec.cns AND tec.cdw not in (SELECT id_cliente from Clientes);
353      --ingreso de productos del viejo sistema desde tmpVentas
354      INSERT INTO Productos
355          SELECT DISTINCT pdw, id_categoria, id_subcategoria, nombre_producto
356          FROM tmpVentas tmpv, TEPProdutos tep
357          WHERE tmpv.id_producto = CAST (tep.pvs as text) and tep.pdw not in (SELECT id_producto fro
358      --ingreso de productos del nuevo sistema desde tmpVentas
359      INSERT INTO Productos
360          SELECT DISTINCT pdw, id_categoria, id_subcategoria, nombre_producto
361          FROM tmpVentas tmpv, TEPProdutos tep
362          WHERE tmpv.id_producto = tep.pns and tep.pdw not in (SELECT id_producto from Productos);
363      --ingreso de ventas del viejo sistema desde tmpVentas
364      INSERT INTO Ventas
365          SELECT id_tiempo, fecha_vta, id_factura, cdw, pdw, id_sucursal, id_medio_pago, monto_vendi
366          FROM tmpVentas tmpv, TECliente tec, TEPProdutos tep
367          WHERE tmpv.id_cliente = CAST (tec.csv as text) AND tmpv.id_producto = CAST (tep.pvs as tex
368      --ingreso de ventas del nuevo sistema desde tmpVentas
369      INSERT INTO Ventas
370          SELECT id_tiempo, fecha_vta, id_factura, cdw, pdw, id_sucursal, id_medio_pago, monto_vendi
371          FROM tmpVentas tmpv, TECliente tec, TEPProdutos tep
372          WHERE tmpv.id_cliente = tec.cns AND tmpv.id_producto = tep.pns;
373
374  END;
375  $$ LANGUAGE plpgsql;
```

En donde para realizar las inserciones en la tabla clientes debemos quedarnos sólo con el nombre (eliminando el apellido), y para hacer eso, usamos la siguiente función:

```
290
297  -- elimina apellidos de nombre_cliente
298  CREATE OR REPLACE FUNCTION cortarApellidos(nombre text) RETURNS text AS $$
299  DECLARE
300      result text;
301
302  BEGIN
303      SELECT REGEXP_REPLACE(nombre, '^((\w+\s)*\w+, \s)', '') INTO result;
304      RETURN result;
305
306  END;
307  $$ LANGUAGE plpgsql;
```

Donde utilizando expresiones regulares eliminamos los apellidos de los nombre_cliente con formato 'apellidos, nombres'.

Para facilitar el trabajo creamos dos funciones:

- Carga_Particular(pSuc integer, pMes integer, pAño integer)



□ Carga_General(ainicial integer, afinal integer)

```
377
378      -- Script ETL - Extraccion de datos de ventas DE UNA FECHA y de UNA sucursal
379      CREATE OR REPLACE FUNCTION Carga_Particular(pSuc integer, pMes integer, pAño integer) RETURNS VOID AS
380      $$$
381      DECLARE
382          a record;
383      BEGIN
384          SELECT CargaTmpVentas(pSuc, pMes, pAño) into a;
385          SELECT CargaTmpVentasSN(pSuc, pMes, pAño) into a;
386          SELECT Carga_CPV() into a;
387          DELETE FROM tmpventas;
388      END;
389      $$ LANGUAGE plpgsql;
390
391      -- Script ETL - Extraccion de datos de ventas ENTRE DOS FECHA de TODAS las sucursales
392      CREATE OR REPLACE FUNCTION Carga_General(ainicial integer, afinal integer) RETURNS VOID AS
393      $$$
394      DECLARE
395          i integer; j integer; a record;
396      BEGIN
397          FOR i IN 1 .. 3 LOOP
398              SELECT * INTO a FROM (SELECT CargaTEClientes(20,i)) as s;
399              SELECT * INTO a FROM (SELECT CargaTEProductos(20,i)) as s;
400          END LOOP;
401          FOR k IN 1 .. 3 LOOP
402              FOR i IN ainicial .. afinal LOOP
403                  FOR j IN 1 .. 12 LOOP
404                      SELECT * INTO a FROM (SELECT Carga_Particular(k,j,i)) as s;
405                  END LOOP;
406              END LOOP;
407          END LOOP;
408      END;
409      $$ LANGUAGE plpgsql;
410
411
```

Donde Carga_Particular se utiliza para realizar las cargas semanales, cargando primero la tabla tmpVentas, cargando luego las tablas del DW, y finaliza vaciando la tabla tmpVentas.

Y Carga_General usa Carga_Particular y se utiliza para hacer una carga inicial del DW recibiendo por parámetro el periodo de años que se va a importar.

Punto 4

Este punto lo resolvimos de la siguiente forma:

Para mostrar las ventas vistas por mes o por año, por sucursal, por región, por cliente y demás combinaciones entre las perspectivas usamos CUBE:



```
3      -- Venta vista por mes o por año, por sucursal, por región, por cliente y demás combinaciones entre las perspe
4      SELECT T.mes, T.año, S.descripcion, C.descripcion, CL.nombre, sum(monto_vendido), sum(V.cantidad_vendida)
5      FROM ventas V, tiempo T, sucursal S, ciudad C, clientes CL
6      WHERE V.id_tiempo = T.id_tiempo and V.id_sucursal = S.id_sucursal
7          and S.id_ciudad = C.id_ciudad and V.id_cliente = CL.id_cliente
8      GROUP BY CUBE(T.mes, T.año,
9                  (S.id_sucursal, S.descripcion),
10                 (C.id_ciudad,C.descripcion),
11                 (CL.id_cliente,CL.nombre))
12
13
```

Output pane

	mes	año	sucursal	ciudad	nombre_cliente	monto_total_vendido	cant_total_vendida
	integer	integer	character varying(30)	character varying(30)	text	real	bigint
9376		2017	Sucursal 3		Marcela	73013	85
9377			Sucursal 3		Marcela	73013	85
9378		2014	Sucursal 3	La Plata	Estela	312185	310
9379		2014	Sucursal 3		Estela	312185	310
9380			Sucursal 3		Estela	312185	310
9381		2015	Sucursal 3	La Plata	Norma	297737	238
9382		2015	Sucursal 3		Norma	297737	238
9383			Sucursal 3		Norma	297737	238
9384		2015	Sucursal 3	La Plata	Estela	202234	246
9385			Sucursal 3		Estela	202234	246
9386			Sucursal 3		Estela	202234	246
9387		2015	Sucursal 3	La Plata	Susana	25584	48
9388		2015	Sucursal 3		Susana	25584	48
9389			Sucursal 3		Susana	25584	48
9390		2013	Sucursal 3	La Plata	Susana	408925	406
9391			Sucursal 3		Susana	408925	406

OK. DOS Ln 14, Col 2, Ch 743 510 chars 24865 rows. 1451 ms

Para mostrar la manera en que influye, en la venta de productos, la zona geográfica en la que están ubicados los locales, usamos ROLLUP:

```
14      -- Es necesario conocer también de que manera influye, en las ventas de productos, la zona geográfica en la qu
15      SELECT R.descripcion, P.descripcion, C.descripcion, sum(V.monto_vendido), sum(V.cantidad_vendida)
16      FROM ventas V, sucursal S, ciudad C, provincia P, region R
17      WHERE V.id_sucursal = S.id_sucursal and S.id_ciudad = C.id_ciudad
18          and C.id_provincia = P.id_provincia and P.id_region = R.id_region
19      GROUP BY ROLLUP (
20          (R.id_region,R.descripcion),
21          (P.id_provincia, P.descripcion),
22          (C.id_ciudad, C.descripcion)
23
24
```

Output pane

	descripcion	descripcion	descripcion	monto_total_vendido	cant_total_vendida
	character varying(30)	character varying(30)	character varying(30)	real	bigint
1	Patagonica	Chubut	Trelew	9.89089e+007	91483
2	Patagonica	Chubut	Rawson	9.3579e+007	91241
3	Patagonica	Chubut		1.92488e+008	182724
4	Patagonica			1.92488e+008	182724
5	Centro-Este	Buenos Aires	La Plata	9.32025e+007	88431
6	Centro-Este	Buenos Aires		9.32025e+007	88431
7	Centro-Este			9.32025e+007	88431
8				2.8569e+008	271155

OK. DOS Ln 17, Col 1, Ch 892 458 chars 8 rows. 31 ms

Para mostrar cuales son los clientes que generan mayores ingresos usamos RANK():



```
25      -- De cada cliente se desea conocer cuales son los que generan mayores ingresos a la cooperativa.
26      SELECT C.nombre, sum(cantidad_vendida), sum(V.monto_vendido) as total_ingreso,
27          rank() over (order by (sum(V.monto_vendido)) desc) as pos
28      FROM Clientes C, Ventas V
29      WHERE C.id_cliente = V.id_cliente
30      GROUP BY C.id_cliente
31      ORDER BY pos
32
```

Output pane

Data Output Explain Messages History

	nombre	sum	total_ingreso	pos
	text	bigint	real	bigint
1	Iris	1112	1.41225e+006	1
2	Iris	933	1.37519e+006	2
3	Lidia	1118	1.33227e+006	3
4	Marcela	1221	1.2864e+006	4
5	Lidia	853	1.22599e+006	5
6	Noemi	986	1.20382e+006	6
7	Noemi	714	1.2037e+006	7
8	Fatima	997	1.19699e+006	8
9	Noemi	865	1.17375e+006	9
10	Norma	705	1.09348e+006	10
11	Iris	888	1.0746e+006	11
12	Ines	769	1.05202e+006	12
13	Iris	829	1.02768e+006	13

OK. DOS Ln 28, Col 1, Ch 1451 235 chars 996 rows. 63 ms

(En este caso da la impresión de que está rankeando dos clientes iguales en forma diferente, por ejemplo el primero que aparece 'Iris', pero eso es solo por que en la carga de las bases se repiten los nombres)

Y por último, para mostrar un análisis diario, mensual trimestral y anual usamos ROLLUP:

```
32      -- Se necesitará hacer análisis diarios, mensuales, trimestrales y anuales.
33      SELECT V.fecha, T.mes, T.año, T.trimetros, sum(cantidad_vendida) as cantidad_total,
34          sum(V.monto_vendido) as total_ingreso
35      FROM Ventas V, Tiempo T
36      WHERE V.id_tiempo = T.id_tiempo
37      GROUP BY ROLLUP(T.año, T.trimetros, T.mes)
38
39
```

Output pane

Data Output Explain Messages History

	año	trimetros	mes	fecha	timestamp without time zone	cantidad_total	total_ingreso
	integer	integer	integer			bigint	real
1268	2018	2	6	2018-06-30 00:00:00		187	126227
1269	2018	2	6			2969	2.29367e+00
1270	2018	2				10329	8.51145e+00
1271	2018	3	7	2018-07-01 00:00:00		224	154488
1272	2018	3	7	2018-07-02 00:00:00		302	228152
1273	2018	3	7	2018-07-03 00:00:00		694	596829
1274	2018	3	7	2018-07-04 00:00:00		33	15204
1275	2018	3	7	2018-07-05 00:00:00		146	150063
1276	2018	3	7			1399	1.14474e+00
1277	2018	3				1399	1.14474e+00
1278	2018					25556	2.10162e+00
1279						271155	2.8569e+008

OK. DOS Ln 35, Col 1, Ch 1769 234 chars 1279 rows. 109 ms

Punto 5

KNIME es un entorno totalmente gratuito para el desarrollo y ejecución de técnicas de minería de datos. KNIME fue desarrollado originalmente en el departamento de bioinformática y minería de datos de la Universidad de Constanza, Alemania, bajo la supervisión del profesor Michael Berthold. Que ofrece, siguiendo un concepto de fraccionamiento de datos (data pipelining) modular y procedimientos de análisis propios, posibilidades de integración de diversos algoritmos de minería de datos y de aprendizaje automático

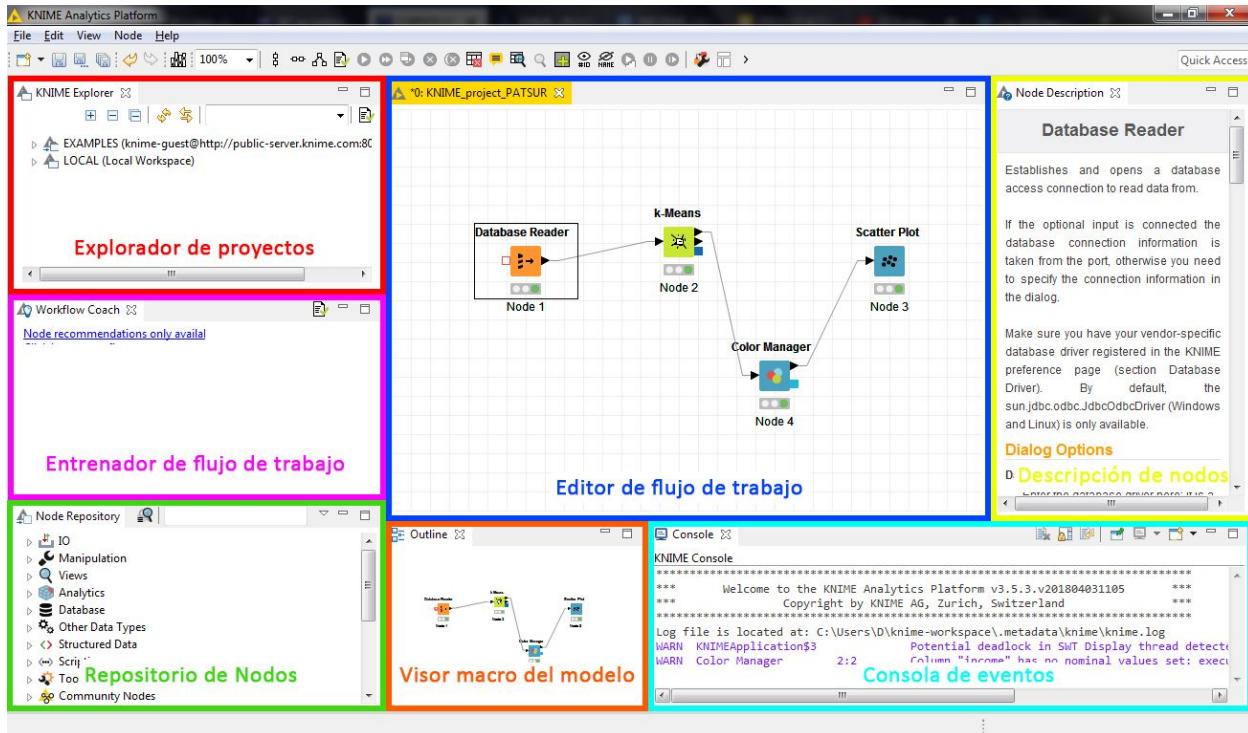
Está desarrollado sobre la plataforma Eclipse y programado, esencialmente, en Java. Es de código abierto y multiplataforma

Se diseñó en base a tres principios: modularidad, extensibilidad y ambiente de trabajo interactivo. A continuación, se describen estos principios:

- Modularidad: Plantea que no debe existir dependencias entre las unidades contenedoras de datos o de procesamiento. Además, se pueden implementar algoritmos de forma independiente. De igual forma, al no tener tipos de datos predefinidos, se pueden definir nuevos tipos de datos con sus características y especificaciones propias. Estos pueden declararse compatibles con otros existentes.
- Extensibilidad de forma sencilla: Permite adicionar nuevas unidades de procesamiento, visualización y tratamiento de datos teniendo en cuenta que esto debe ser una tarea fácil de realizar.
- Ambiente de trabajo visual e interactivo: Los flujos de trabajo deben ser fáciles e interactivos para el usuario. Por tal motivo se harán arrastrando los elementos al área de trabajo.

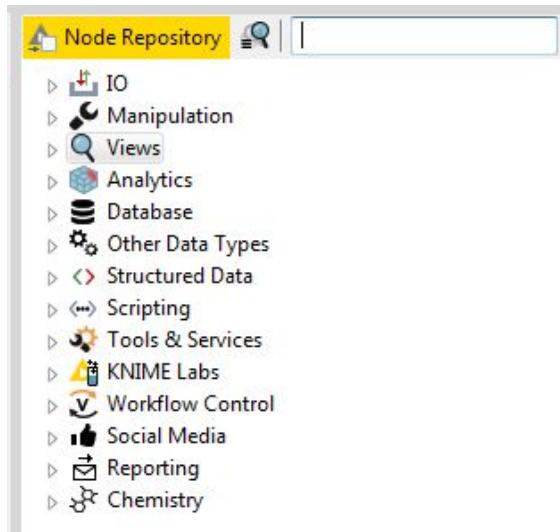
Su uso se basa en el diseño de un flujo de trabajo que plasma las distintas etapas de un proyecto de minería de datos.

Cuando abrimos el KNIME, nos encontramos con la siguiente ventana, donde se pueden observar distintas áreas de trabajo.



Siendo algunas de estas:

- ❑ Explorador de proyectos: donde se muestran los proyectos existentes en el workspace.
- ❑ Repositorio de nodos: donde se encuentran todos los nodos que se pueden utilizar en el flujo de trabajo, agrupados en fichas o categorías, como por ejemplo:
 - ❑ Entrada de datos [IO > Read].
 - ❑ Salida de datos [IO > Write].
 - ❑ Preprocesamiento [Data Manipulation], para filtrar, discretizar, normalizar, filtrar, seleccionar variables...
 - ❑ Minería de datos [Mining], para construir modelos (reglas de asociación, clustering, clasificación, MDS, PCA...).
 - ❑ Salida de resultados [Data Views] para mostrar resultados en pantalla (ya sea de forma textual o gráfica).

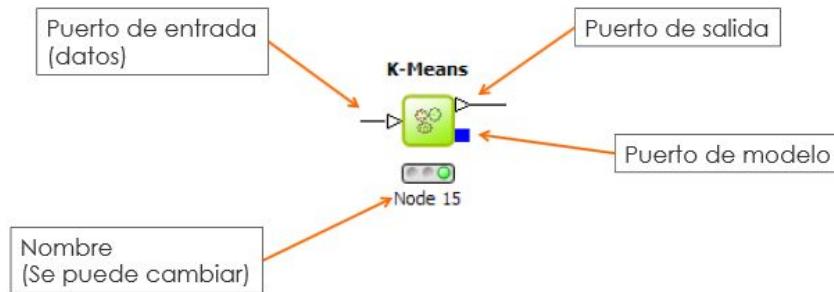


- ❑ Editor de flujo de trabajo: es la ventana donde se construyen los diagramas de análisis de datos (flujos de trabajo), arrastrando y soltando nodos dentro del editor, para luego conectarlos entre ellos (Para crear un flujo de trabajo, las salidas de unos nodos se utilizan como entradas de otros).
- ❑ Consola de eventos: Muestra los mensajes de warning y error. Estos mensajes también se almacenan en un archivo llamado knime.log (stack trace) ubicado en una carpeta .metadata del proyecto.
- ❑ Descripción de nodos: Provee información sobre cualquier nodo seleccionado. O sobre una categoría de nodos seleccionada. Si no hay nada seleccionado, se muestra vacía.

Los nodos implementan distintos tipos de acciones que pueden ejecutarse sobre una tabla de datos:

- ❑ Manipulación de filas, columnas, muestreos, transformaciones, agrupaciones, etc.
- ❑ Visualización (histogramas, etc.).
- ❑ Creación de modelos estadísticos y de minería de datos, como árboles de decisión, máquinas de vector soporte, regresiones, etc.
- ❑ Validación de modelos, como curvas ROC, etc.
- ❑ Scoring o aplicación de dichos modelos sobre conjuntos nuevos de datos.
- ❑ Creación de informes a medida gracias a su integración con BIRT.

Los nodos son básicamente las unidades de procesamiento del flujo de trabajo. Y a partir de estos se construyen los flujos de trabajo. Las partes de un nodo son:

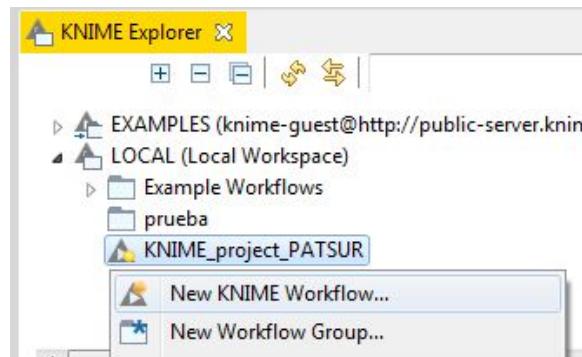


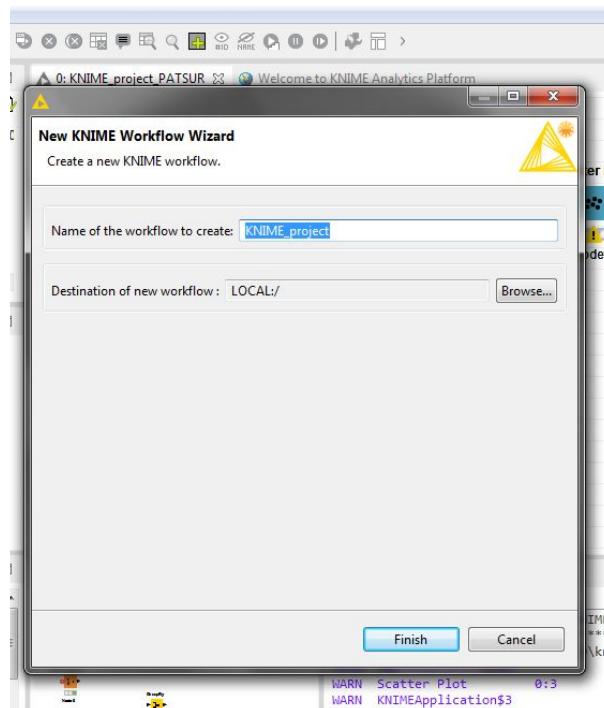
Los datos en el flujo de trabajo se transportan entre los nodos a través de los puertos de entrada/salida. Y es necesario conectar las entradas de un nodo con la salida de otro. Solo se pueden conectar puertos del mismo tipo.

Los tipos de puerto de un nodo son:

- Datos (triángulo blanco): transfieren tablas de datos entre nodos
- Bases de datos (cuadrado marrón)
- PMML: transfieren modelos ya aprendidos
- Otros puerto.

Para crear un proyecto clicleamos con el botón derecho sobre el explorador de proyectos, elegimos la opción New KNIME workflow y después le ponemos nombre al proyecto, y apretamos finalizar.





Para construir un flujo de trabajo arrastramos y soltamos los nodos desde el repositorio de nodos al editor de flujo de trabajo, conectándose entre ellos (Es necesario, que una vez colocados los nodos en el editor, se conecta la salida de cada nodo con el predecesor).

Una vez que el flujo de trabajo está totalmente conectado, si existen nodos que tiene su estado en color rojo, es necesario configurarlos. Y si su estado presenta un color amarillo, significa que la configuración de los mismos se realizó utilizando los valores por defecto.

Los estados de un nodo pueden ser:



Para configurar un nodo clicleamos con el botón derecho sobre el nodo, elegimos la opción configure y después, de acuerdo al tipo de nodo realizamos la configuración.

Cuando todos los nodos del flujo de trabajo poseen color amarillo, entonces el mismo puede ser ejecutado. Los nodos se ejecutan de izquierda a derecha, es decir, que un nodo sólo puede ejecutarse así todos los nodos predecesores han terminado su ejecución.

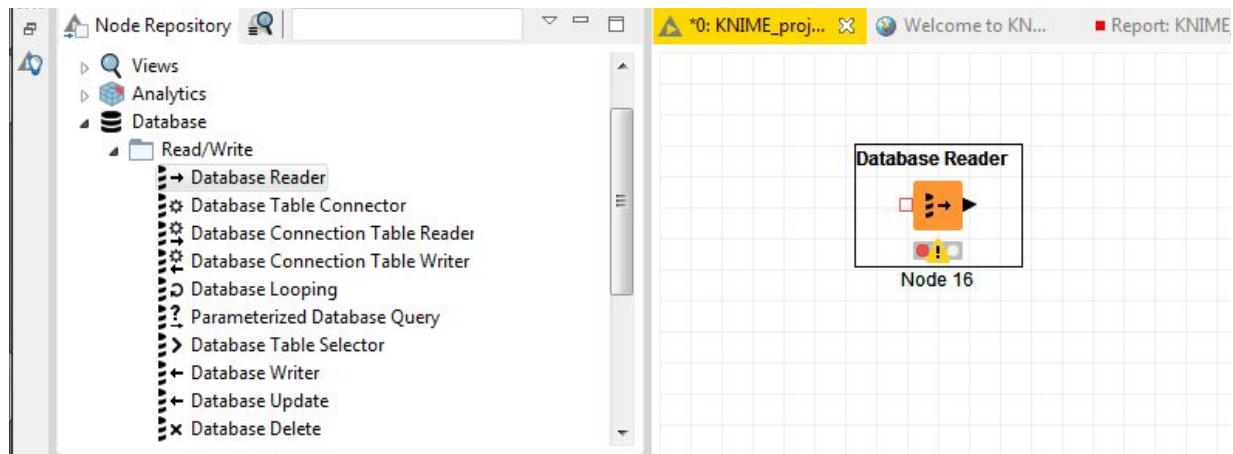
Formas de ejecución:

- Por nodo (con la opción Execute)
- Ejecutando el último nodo del flujo (KNIME ejecuta los predecesores)
- Seleccionando varios nodos y disparando la ejecución (KNIME determina el orden y ejecuta nodos en paralelo, si es posible).

Un flujo básico suele ser de la forma:

- Nodo de lectura de datos.
- Nodo de preprocessamiento.
- Nodo de modelado (por ejemplo, modelo de clasificación)
- Nodo de salida de resultados.

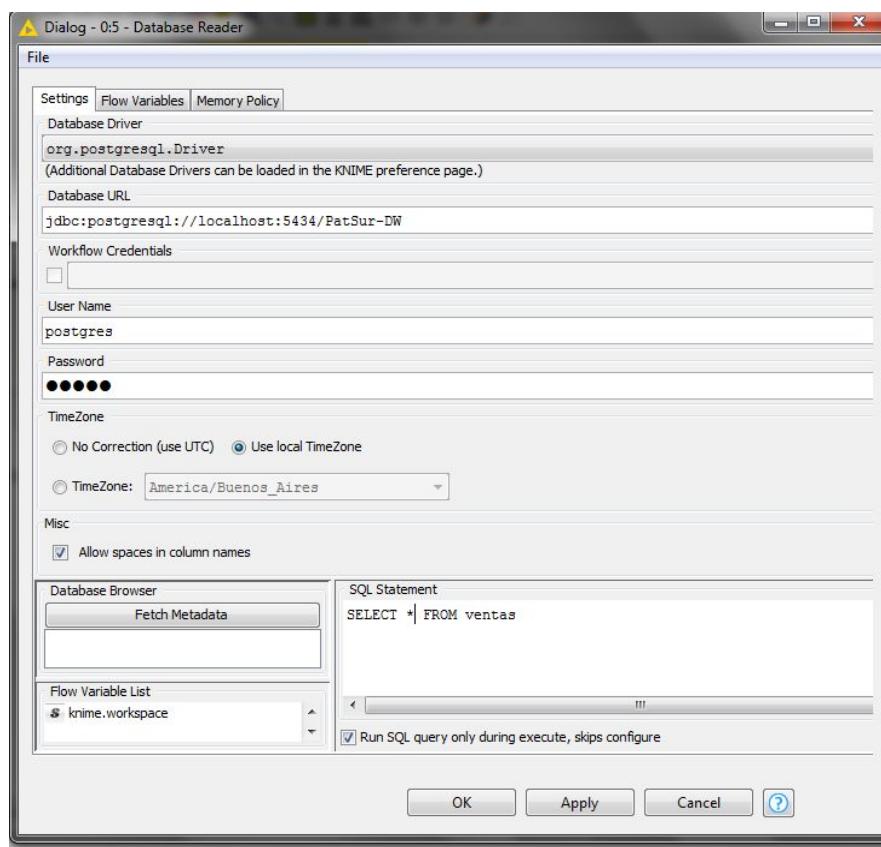
Donde, por ejemplo, como nodo de lectura de datos utilizamos el nodo Database Reader para poder conectar la base de datos del DW.





Para configurarlo completamos los siguientes campos:

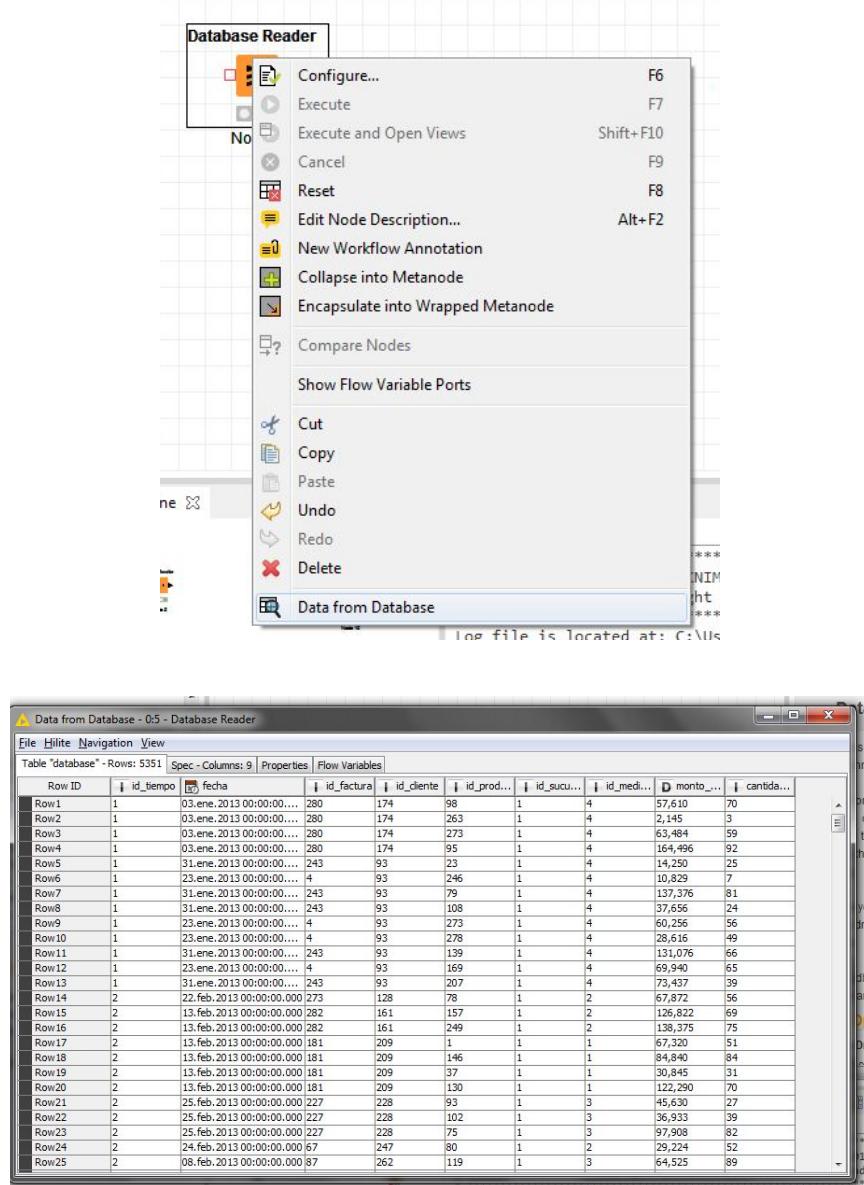
- database driver: driver de postgresql (org.postgresql.Driver).
- database url: host, puerto y nombre de la base de datos que queremos conectar.
- user name: usuario de la base de datos que queremos conectar.
- password: contraseña del usuario de la base de datos que queremos conectar.
- sql statement: consulta con los datos que queremos traer de la base de datos.



En el caso del sql statement lo que hacemos es incluir una consulta sql (esta consulta podría ser inclusive las consultas del punto 4). En este ejemplo consultamos la tabla ventas. Luego lo ejecutamos.



Podemos consultar los datos del nodo (no en todos los nodos se puede hacer esto):



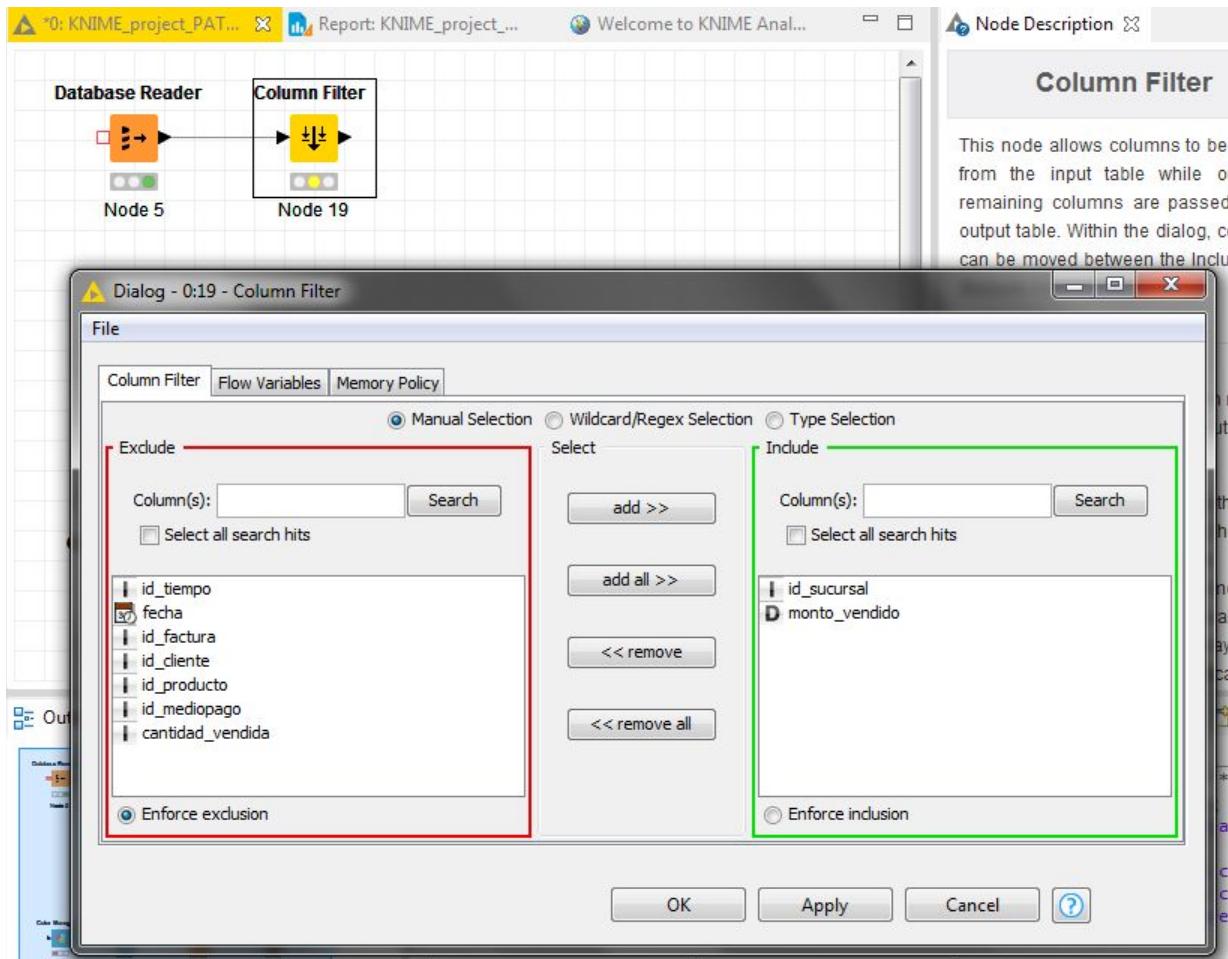
The screenshot shows the KNIME interface. A 'Database Reader' node is selected, and its context menu is open. The menu includes options like 'Configure...', 'Execute', 'Execute and Open Views', 'Cancel', 'Reset', 'Edit Node Description...', 'New Workflow Annotation', 'Collapse into Metanode', 'Encapsulate into Wrapped Metanode', 'Compare Nodes', 'Show Flow Variable Ports', 'Cut', 'Copy', 'Paste', 'Undo', 'Redo', and 'Delete'. Below the menu, a sub-menu for 'Data from Database' is visible. At the bottom of the interface, a log file path 'C:\Users\...\\log file is located at: C:\Users...' is shown.

Data from Database - 0:5 - Database Reader

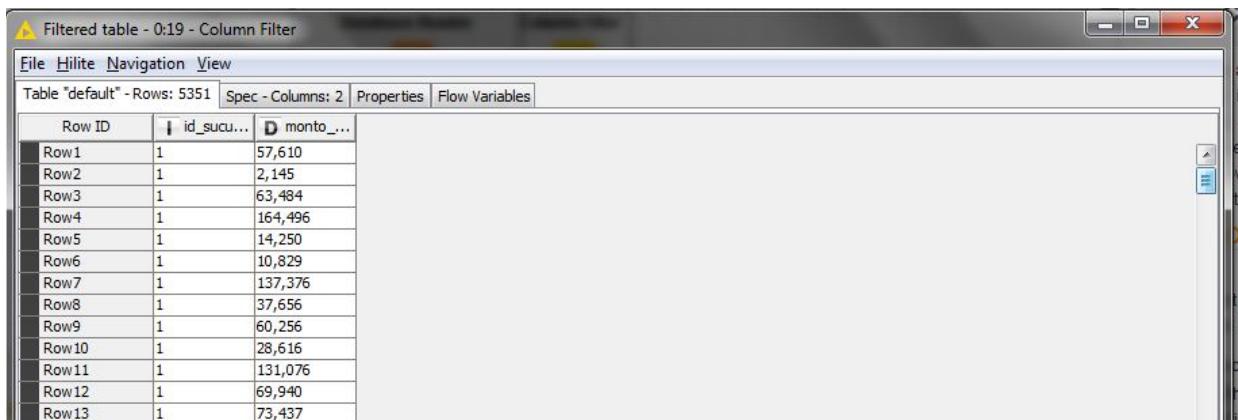
Table "database" - Rows: 5351 Spec - Columns: 9 Properties Flow Variables

Row ID	id_tempo	fecha	id_factura	id_cliente	id_producto	id_sucursal	id_medio	monto...	cantida...
Row 1	1	03.ene.2013 00:00:00....	280	174	98	1	4	57,610	70
Row 2	1	03.ene.2013 00:00:00....	280	174	263	1	4	2,145	3
Row 3	1	03.ene.2013 00:00:00....	280	174	273	1	4	63,484	59
Row 4	1	03.ene.2013 00:00:00....	280	174	95	1	4	164,496	92
Row 5	1	31.ene.2013 00:00:00....	243	93	23	1	4	14,250	25
Row 6	1	23.ene.2013 00:00:00....	4	93	246	1	4	10,829	7
Row 7	1	31.ene.2013 00:00:00....	243	93	79	1	4	137,376	81
Row 8	1	31.ene.2013 00:00:00....	243	93	108	1	4	37,656	24
Row 9	1	23.ene.2013 00:00:00....	4	93	273	1	4	60,256	56
Row 10	1	23.ene.2013 00:00:00....	4	93	278	1	4	28,616	49
Row 11	1	31.ene.2013 00:00:00....	243	93	139	1	4	131,076	66
Row 12	1	23.ene.2013 00:00:00....	1	93	169	1	4	69,940	55
Row 13	1	31.ene.2013 00:00:00....	243	93	207	1	4	73,437	39
Row 14	2	22.feb.2013 00:00:00....	273	128	78	1	2	67,872	56
Row 15	2	13.feb.2013 00:00:00....	282	161	157	1	2	126,822	69
Row 16	2	13.feb.2013 00:00:00....	282	161	249	1	2	138,375	75
Row 17	2	13.feb.2013 00:00:00....	181	209	1	1	1	67,320	51
Row 18	2	13.feb.2013 00:00:00....	181	209	146	1	1	84,840	84
Row 19	2	13.feb.2013 00:00:00....	181	209	37	1	1	30,845	31
Row 20	2	13.feb.2013 00:00:00....	181	209	130	1	1	122,290	70
Row 21	2	25.feb.2013 00:00:00....	227	228	93	1	3	45,630	27
Row 22	2	25.feb.2013 00:00:00....	227	228	102	1	3	36,933	39
Row 23	2	25.feb.2013 00:00:00....	227	228	75	1	3	97,908	82
Row 24	2	24.feb.2013 00:00:00....	67	247	80	1	2	29,224	52
Row 25	2	08.feb.2013 00:00:00....	87	262	119	1	3	64,525	89

Como la consulta inicial contiene todas los campos de la tabla ventas podemos filtrar las columnas con las que nos queremos quedar incluyendo nodos de preprocesamiento. En este ejemplo usamos el nodo column filter, que permite que solo las columnas seleccionadas pasen a la tabla de salida.



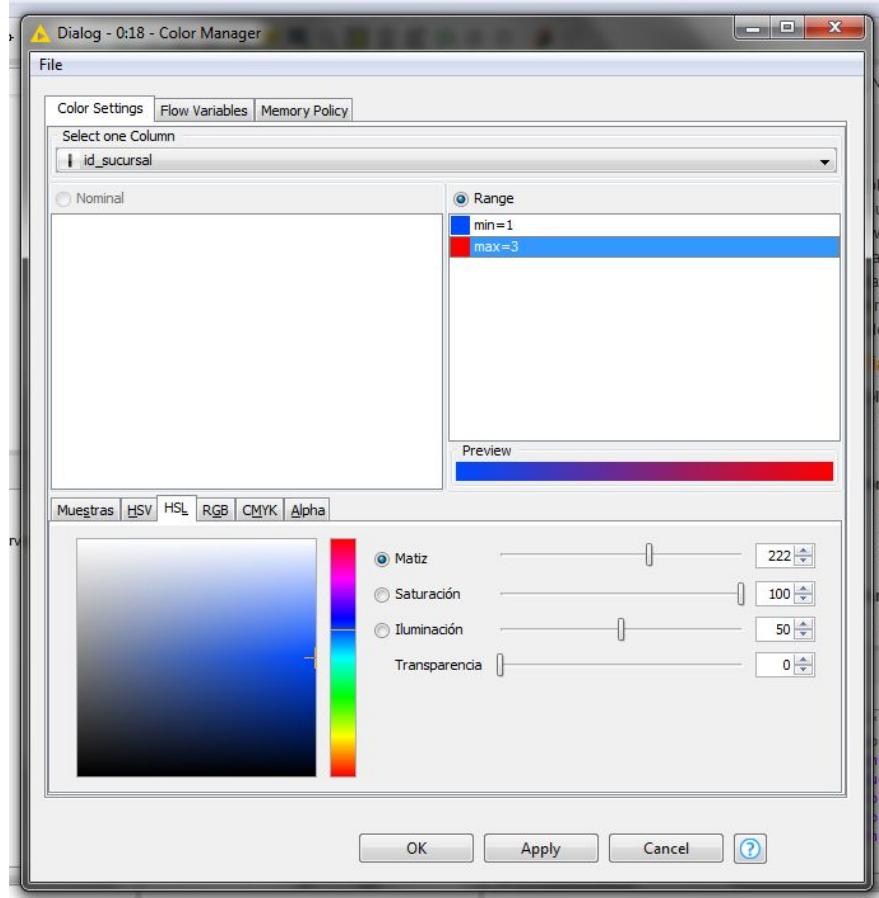
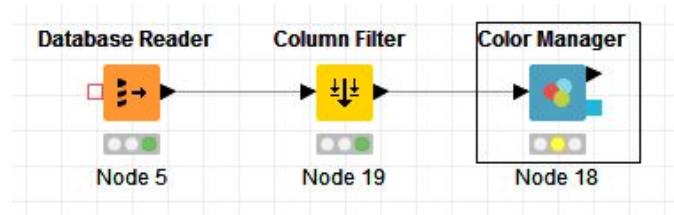
Dentro del diálogo, las columnas se pueden mover entre la lista de inclusión y exclusión. Despues de ejecutarlo podemos ver la columnas filtradas:



Row ID	monto_vendido
Row1	57,610
Row2	2,145
Row3	63,484
Row4	164,496
Row5	14,250
Row6	10,829
Row7	137,376
Row8	37,656
Row9	60,256
Row10	28,616
Row11	131,076
Row12	69,940
Row13	73,437

En este ejemplo no usamos nodos de clasificación, pero si decidimos hacerlo deberíamos agregarlos en este momento.

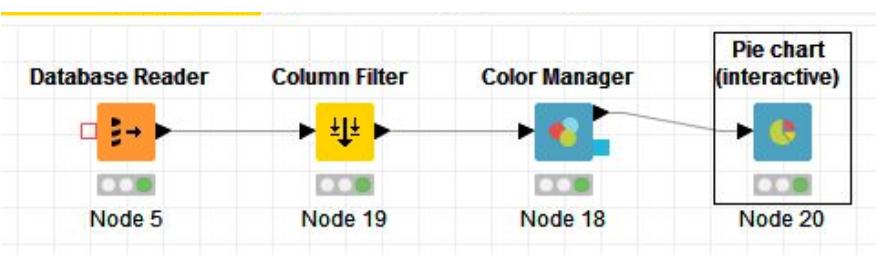
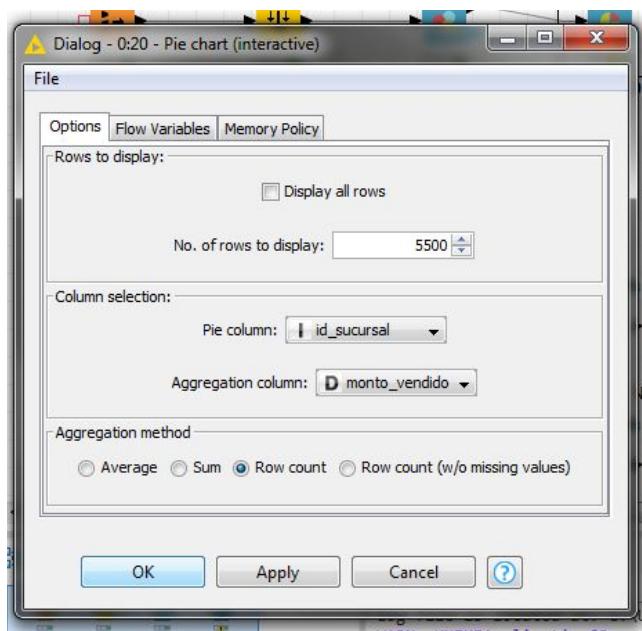
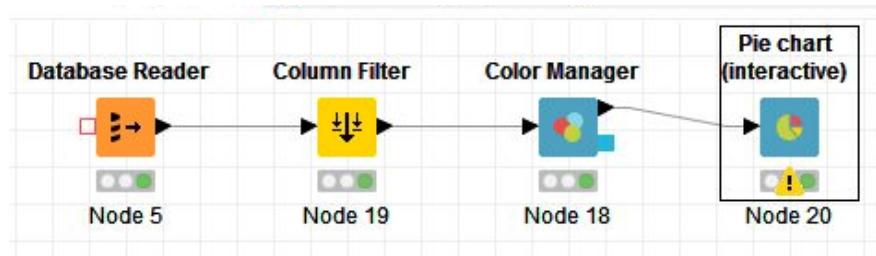
Ahora pasamos a la salida de los datos, KNIME incluye el nodo color manager que permite asignar colores a las columnas nominales o numéricas (con límites inferior y superior). Los valores de color se calculan luego durante la ejecución.



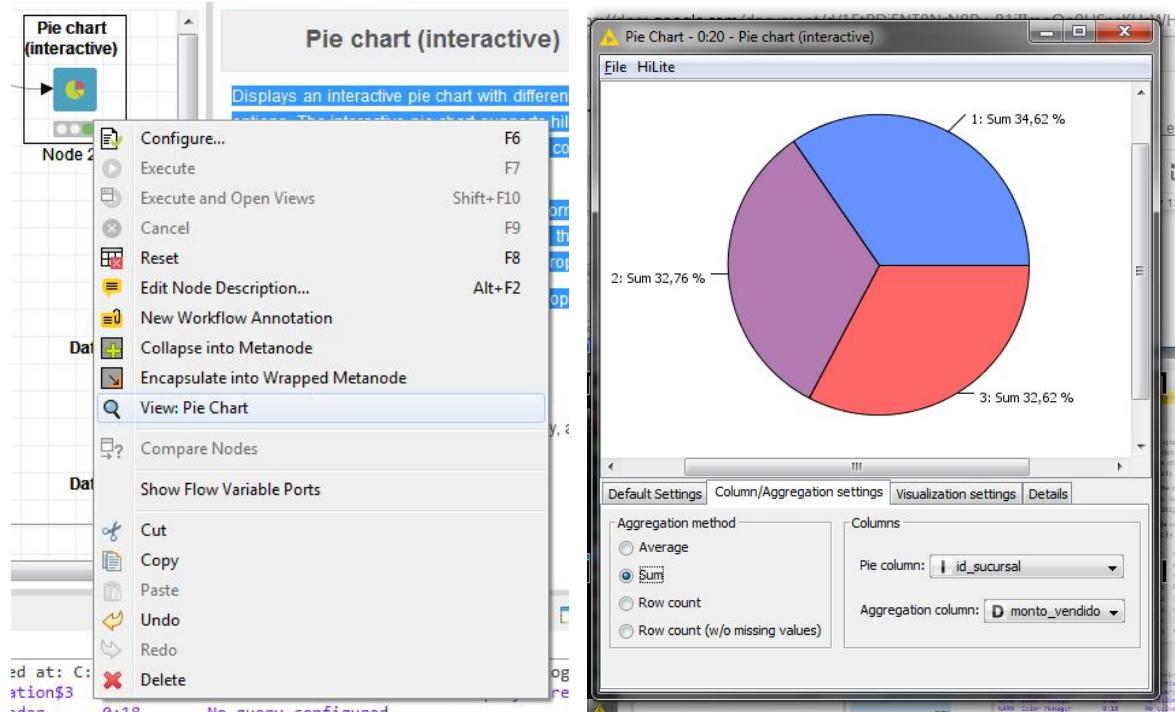
Si se selecciona un atributo de columna, el color se puede cambiar con el selector de color.

Decidimos mostrar la salida de datos con en nodo pie chart, que muestra un gráfico circular interactivo con diferentes opciones de visualización, si el gráfico circular aparece en gris, es porque no se estableció ningún color para la columna circular seleccionada en el nodo color manager.

En el momento de ejecutarlo, nos aparece una advertencia debido a que la cantidad de filas que tiene configuradas por defecto no alcanza para la cantidad de valores que queremos mostrar. Lo configuramos. y ejecutamos nuevamente.

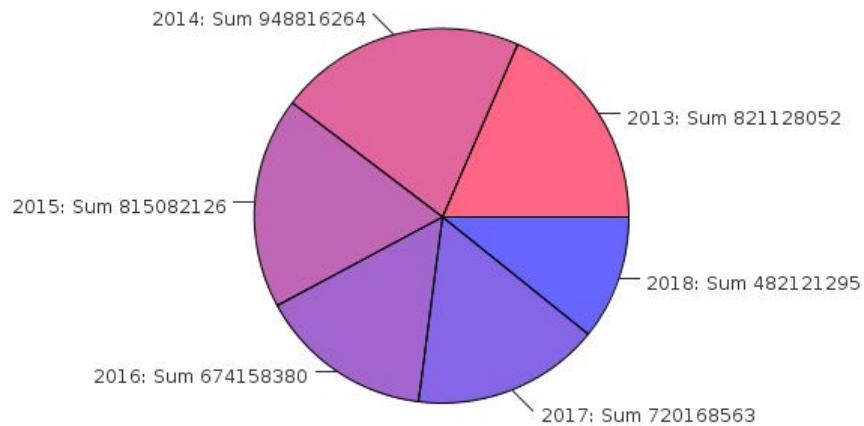
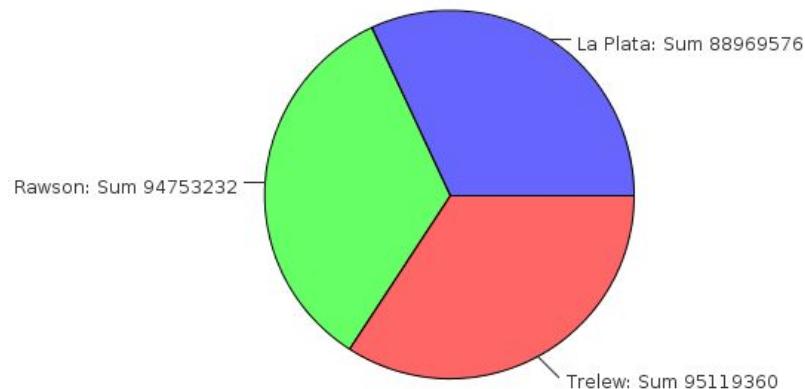
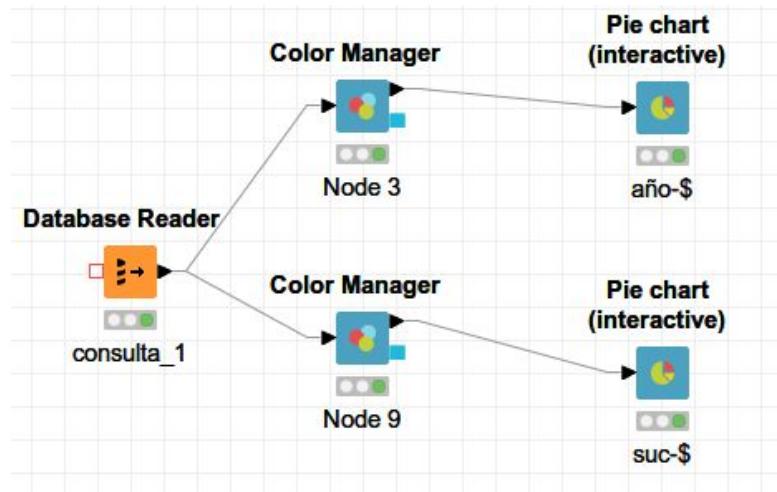


Vemos el gráfico de salida



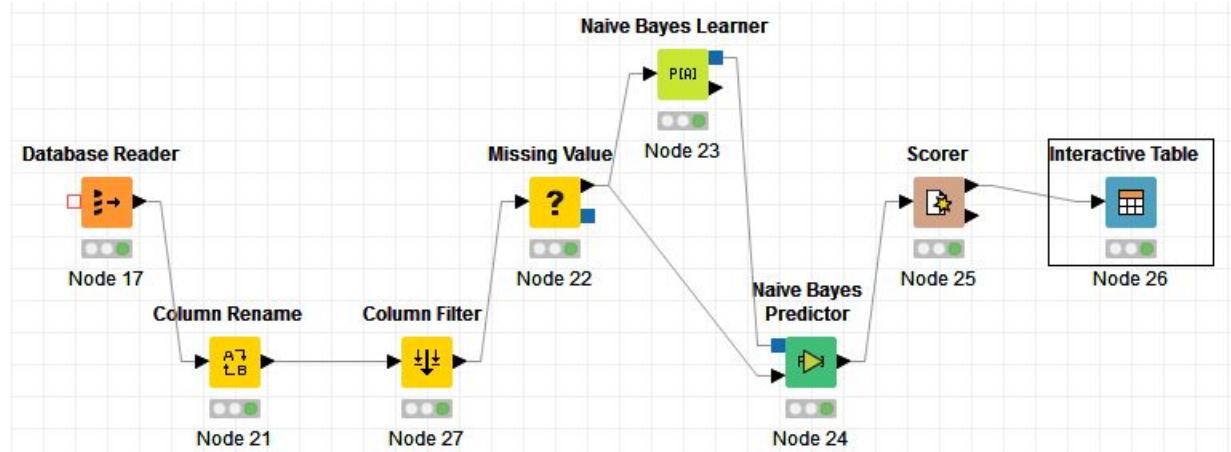
Donde nos permite cambiar sobre la marcha por ejemplo el método de agregación y en este caso elegimos suma.

De la misma forma que obtuvimos este gráfico hemos podido obtener otros similares. En el ejemplo usamos un filtro solo para mostrar cómo se usa, pero este tipo de gráfico nos permite filtrar en forma interactiva las columnas que queremos mostrar.

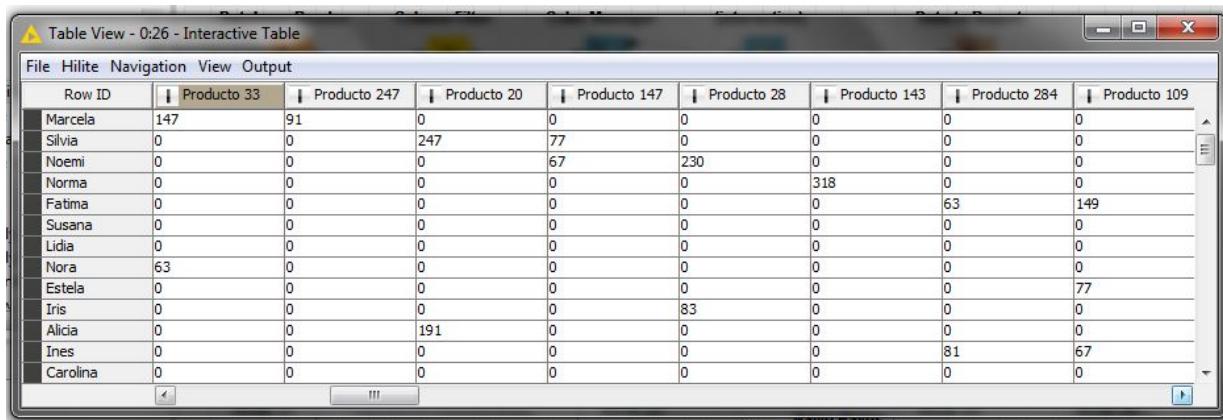


Otro forma de usar KNIME podría ser para predecir el producto comprado según el nombre del cliente (como la promoción de una bebida gaseosa con el nombre en la etiqueta), o según la fecha de compra o mes, cosa que seria mas util para generar algunas promociones u ofertas.

Para el caso de nombre producto - nombre cliente lo hacemos:



Y obtenemos:



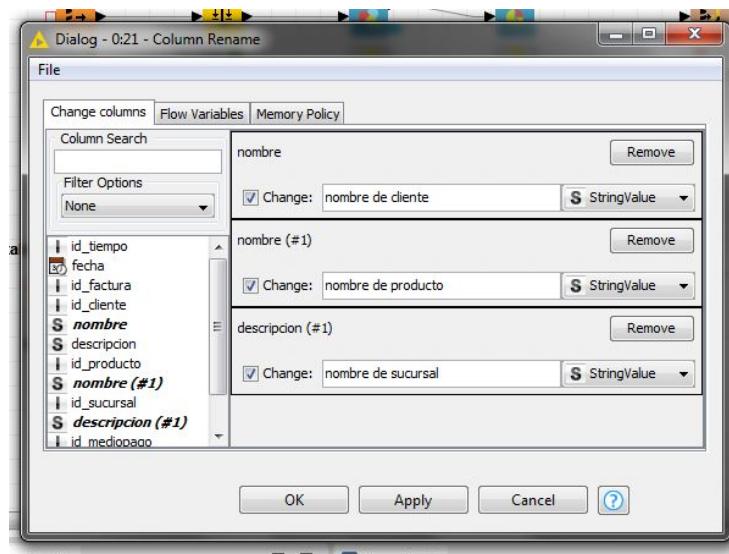
The screenshot shows the 'Table View' window from KNIME. The title bar reads 'Table View - 0:26 - Interactive Table'. The table has columns for Row ID and ten different product codes (Producto 33, Producto 247, Producto 20, Producto 147, Producto 28, Producto 143, Producto 284, Producto 109). The data rows list 15 different customer names (Marcela, Silvia, Noemi, Norma, Fatima, Susana, Lidia, Nora, Estela, Iris, Alicia, Ines, Carolina) and their corresponding predicted purchase counts for each product. For example, Marcela has purchases of 147 for Producto 33 and 91 for Producto 247.

Row ID	Producto 33	Producto 247	Producto 20	Producto 147	Producto 28	Producto 143	Producto 284	Producto 109
Marcela	147	91	0	0	0	0	0	0
Silvia	0	0	247	77	0	0	0	0
Noemi	0	0	0	67	230	0	0	0
Norma	0	0	0	0	0	318	0	0
Fatima	0	0	0	0	0	0	63	149
Susana	0	0	0	0	0	0	0	0
Lidia	0	0	0	0	0	0	0	0
Nora	63	0	0	0	0	0	0	0
Estela	0	0	0	0	0	0	0	77
Iris	0	0	0	0	83	0	0	0
Alicia	0	0	191	0	0	0	0	0
Ines	0	0	0	0	0	0	81	67
Carolina	0	0	0	0	0	0	0	0

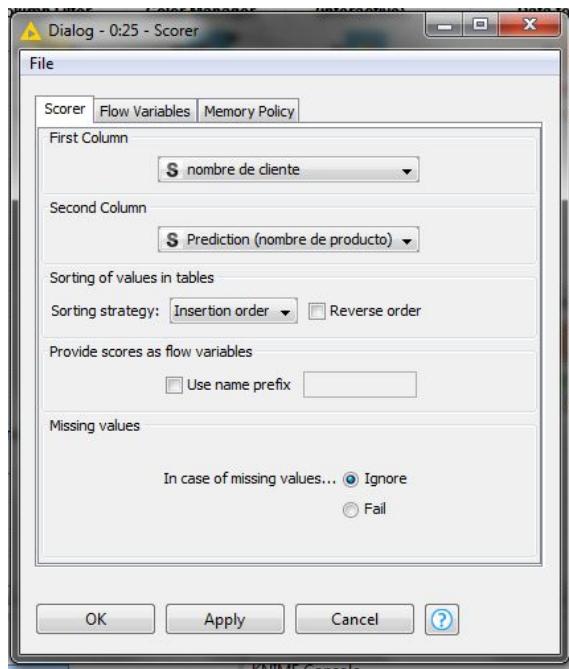
Acá es necesario hacer la observación de que los datos no son muy reales debido a que las tablas de las bases de las sucursales fueron cargadas con nombres de los cliente son aleatorios y se repiten, pero para dar un ejemplo alcanza.

En este ejemplo aparecen algunos nodos que no mostramos antes, algunos de ellos son:

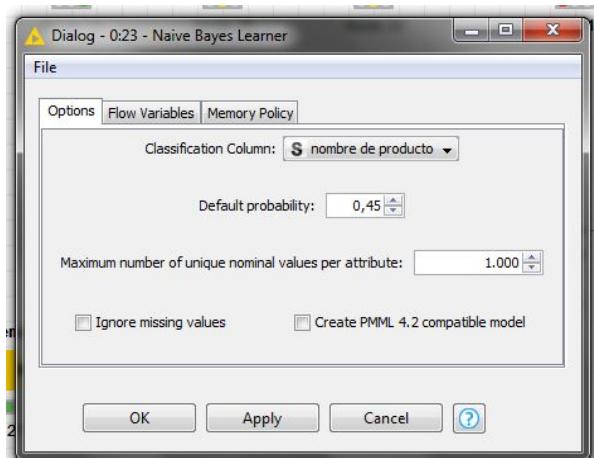
Column rename : Cambia el nombre de las columnas o cambia sus tipos.



Scorer: Compara dos columnas por sus pares de valores de atributos y muestra la matriz, es decir cuántas filas de qué atributo y su clasificación coinciden. El diálogo le permite seleccionar dos columnas para comparar. La salida del nodo es la matriz con el número de coincidencias en cada celda.



Naive Bayes Learner: Este nodo crea un modelo bayesiano a partir de los datos de entrenamiento proporcionados. Calcula el número de filas por valor de atributo por clase para atributos nominales y la distribución gaussiana para atributos numéricos.



Naive Bayes Predictor: Predice la clase por fila en función del modelo aprendido. La probabilidad de clase es el producto de la probabilidad por atributo y la probabilidad del atributo de clase en sí. La probabilidad de valores nominales es el número de ocurrencias del valor dado por el valor de clase. La probabilidad de valores numéricos se calcula suponiendo una distribución normal por atributo.

En este nodo usamos la configuración por defecto.

Para el caso de nombre producto - mes solo cambiamos algunas configuraciones y obtenemos el siguiente resultado :

Row ID	Producto 33	Producto 247	Producto 20	Producto 147	Producto 28	Producto 143	Producto 284	Producto 25
1	15	14	54	21	5	19	3	10
2	17	4	23	5	43	24	22	19
3	31	14	36	12	31	50	35	19
4	11	7	30	19	31	26	14	29
5	7	4	41	11	30	26	8	11
6	20	7	19	16	19	26	13	10
7	4	5	27	16	6	30	12	10
8	0	0	18	15	0	23	2	2
9	2	1	5	1	0	13	2	3
10	0	5	3	0	0	9	0	6
11	8	4	12	9	4	34	4	13
12	11	0	17	8	11	13	7	17

Y para terminar otra forma de usar KNIME es para realizar reportes. Para esto es necesario tener instalada la extensión de KNIME Report Designer, esta proporciona una plantilla de informe para cada flujo de trabajo. Donde por ejemplo se puede realizar un reporte como el siguiente:

BASE DE DATOS

INFORME LABORATORIO N°5



Monto vendido por sucursal



Monto y cantidad vendida por mes

mes	monto_vendido	cantidad_vendida
1	23682748	22243
2	23056665	22512
3	29895974	29767
4	26239346	23813
5	23962702	24122
6	23300555	22480
7	25765793	24401
8	19776981	17308
9	22842602	21352
10	25557238	23588
11	19879369	19061
12	21730409	20508

Monto y cantidad vendida por mes

id_sucursal	monto_vendido	cantidad_vendida
1	98909008	91483
2	93579043	91241
3	93202331	88431



La forma de realizar los reportes es, primero seleccionar, transformar, preprocesar, filtrar y preparar los datos que desea usar en el informe. Esto, como ya hemos mostrado, se lleva a cabo en un flujo de trabajo. Luego con los nodos 'Data to Report' habilitamos las tablas de datos que vamos a usar dentro del reporte. Y dentro del diseñador de reporte diseñamos, aplicamos estilos y editamos la presentación de los datos con el editor de plantillas de informes.

En resumen, con el editor de flujo de trabajo de KNIME se realiza todo el preproceso en los datos de una manera completa, mientras que el editor de plantillas de informes se presentarán los datos del reporte.

Los reportes realizados se pueden exportar a distintos formatos, el siguiente reporte, por ejemplo, fue exportado en un archivo de word.

BASE DE DATOS

INFORME LABORATORIO N°5

Monto y cantidad vendida por sucursal por tipo de cliente

tipo cliente	cliente eventual		cliente potencial		externo		interno		publico objetivo	
sucursal	monto	cantidad	monto	cantidad	monto	cantidad	monto	cantidad	monto	cantidad
Sucursal 1	17469233	16045	22336072	20183	19920917	18115	20150916	18998	19031870	18142
Sucursal 2	16572195	15528	20234128	20723	20945142	20072	17751884	17439	18075694	17479
Sucursal 3	17159786	16802	15636158	14936	20115186	18812	20062503	19300	20096986	18485
Grand Total	51201214	48375	58206358	55842	60981245	56999	57965303	55737	57204550	54106

Created with KNIME Analytics Platform

www.knime.com



Para finalizar este punto (punto 5) mencionamos algunas características a destacar de la plataforma KNIME:

- Tiene numerosas herramientas de manipulación de datos altamente intuitivas y fáciles de implementar.



- Cada paso que queremos implementar en nuestro flujo viene muy bien documentado.
- Es una aplicación suficientemente optimizada como para tratar grandes volúmenes de datos cómodamente.
- Se tiene acceso a un repositorio público dentro de la aplicación que permite visualizar cientos de ejemplos con datasets incluidos.
- Cuenta con una potente herramienta de manipulación de datos que establece la base para su posterior análisis.
- No solo dispone de manipulación de datos sino que además tiene análisis de datos.

Observaciones finales

Se adjuntan al informe:

- Los flujos de trabajo de Knime
- Los Scripts SQL
- La presentación en PDF

El trabajo se puede encontrar en el repositorio de github <https://github.com/88lucas88/BDII>