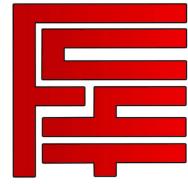




UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
CARRERA DE INGENIERÍA DE SISTEMAS



ALGORÍTMOS Y TÉCNICAS DE GENERACIÓN DE DATOS DE PRUEBA PARA BASE DE DATOS BASADOS EN IDIOMS

PROYECTO DE GRADO PRESENTADO PARA OPTAR
AL DIPLOMA ACADÉMICO DE LICENCIATURA
EN INGENIERÍA DE SISTEMAS.

PRESENTADO POR: MILVER FELIPE FLORES ACEVEDO
TUTOR: LIC. JUAN MARCELO FLORES SOLÍZ

COCHABAMBA - BOLIVIA
AGOSTO - 2016

Dedictoria

Este trabajo se la dedico al creador de todas las cosas quién supo guiarme por el buen camino, darme fuerzas para seguir adelante y no desmayar en los problemas que se presentaban, enseñándome a encarar las adversidades sin perder nunca la dignidad ni desfallecer en el intento.

A mi familia quienes por ellos soy lo que soy. Para mis padres por su apoyo, consejos, comprensión, amor, ayuda en los momentos difíciles, y por ayudarme con los recursos necesarios para estudiar. Me han dado todo lo que soy como persona, mis valores, mis principios, mi carácter, mi empeño, mi perseverancia, mi coraje para conseguir mis objetivos. A mis hermanas por estar siempre presentes, acompañándome para poderme realizar.

A mi tutor por la orientación y ayuda que me brindó para la realización de este trabajo, por su apoyo y amistad que me permitieron aprender mucho más que lo estudiado en el proyecto.

Índice general

1. Introducción	1
1.1. Aplicaciones generadores de datos de prueba para base de datos	2
2. Algoritmos de ordenamiento y mecanismos de manejo referencial	5
2.1. Técnicas de ordenamiento	6
2.2. Algoritmo de ordenamiento	6
2.3. Aplicación del algoritmo	7
2.4. Algoritmo de ordenación <i>primeros en ser llenado</i>	9
2.5. Aplicación del algoritmo de <i>primeros en ser llenado</i>	9
2.6. Mecanismo de <i>manejo de referencial</i> de una base de datos	10
2.6.1. Llaves primarias compuestas (composite keys)	10
2.6.2. Mejor uso de Join	12
2.7. Mecanismo de referenciación	12
2.7.1. Referencia simple	12
2.7.2. Referencia compuesta	13
3. Algoritmos de generación de datos de prueba	15
3.1. Algoritmos de generación de datos	16
3.2. Tipos numéricos	16
3.3. Tipos monetarios	17
3.4. Tipos de caracteres	17
3.4.1. Generación de nombres	17
3.4.1.1. Generación de nombres a partir de vocales y consonantes . .	17
3.4.1.2. Generación de nombres a partir de una lista de nombres y apellidos	19
3.5. Tipos de datos binarios(Binary Data Types)	19
3.5.1. Bytea formato hexadecimal	20
3.5.2. Bytea escapar formato	20

3.6.	Tipos fecha/hora	21
3.6.1.	Generación de fechas	21
3.6.1.1.	Algoritmo de generación de fechas	21
3.6.2.	Generación de dato tipo Time	22
3.7.	Tipos de direcciones de red	23
3.7.1.	Estructura de una dirección IPv4	24
3.7.1.1.	ID de red	24
3.7.1.2.	ID de host	24
4.	Ordenando metadatos y aplicando algoritmos propuestos	27
4.1.	Metadatos en PostgreSQL	28
4.1.1.	Almacenamiento y organización de datos	29
4.1.1.1.	Los índices	29
4.1.2.	Como se procesa un consulta(Query)	30
4.1.2.1.	Métodos para relacionar tablas	31
4.2.	Obtener la estructura de una base de datos	31
4.2.1.	Obtener el detalle de una tabla	32
4.2.1.1.	Usando OIDs	32
4.2.1.2.	Usando Information schema	34
4.2.2.	Obteniendo las relaciones entre las tablas	35
4.2.3.	Obteniendo las tablas independientes	37
4.3.	Ordenando los metadatos	38
4.3.1.	Ordenando las tablas	40
4.3.2.	Uniendo <i>foreign keys</i>	42
5.	Crear proyecto de configuración	49
5.1.	JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) . . .	49
5.2.	Persistencia de la información de metadatos	52
5.2.1.	Creando la estructura de un proyecto	53
5.3.	Configuración de columnas	57
5.3.1.	Configuración para llaves foráneas	57
5.3.1.1.	Problemas	59
5.3.1.2.	Soluciones	60
6.	Poblando datos en la base de datos y probando el comportamiento	63
6.1.	Poblado de datos a la base de datos	63
6.1.1.	Tipos de datos tratados como texto	63

6.1.2. Tipos de datos tratados como números	64
6.1.3. Tipo de dato bytea	64
6.1.4. Cantidad de columnas por tabla	64
7. Uso del prototipo	65
8. Conclusiones	75
Bibliografía	77

Índice de figuras

2.1. Ejemplo modelo ER	8
2.2. Modelo Ordenado	8
2.3. Orden de llenado	10
2.4. Llaves compuestas	11
2.5. Modelo E-R con llaves compuestas	12
3.1. Consonantes y vocales	18
3.2. Apartir de una lista de nombres	19
3.3. Generación de fechas	21
3.4. Generación de Time	23
3.5. Combinación IPv4	24
4.1. PostgreSQL System Concept Architecture [9]	28
4.2. Almacenamiento y Organización de datos [9]	29
4.3. Como se procesa un query [9]	30
4.4. Detalle tabla OIDs	33
4.5. Information schema	35
4.6. Referencias entre tablas	36
4.7. Tablas independientes	38
4.8. Modelo ER compuesto	38
4.9. Tabla venta	39
4.10. Tabla detalle inserción correcta	39
4.11. Tabla detalle inserción incorrecta	39
4.12. Detalle de relaciones entre tablas	40
4.13. Secuencia	41
4.14. Orden correcto	42
4.15. Detalle de la tabla <i>detalle</i>	43
4.16. Detalle de referencias de la tabla detalle	44

5.1. Object JSON	50
5.2. Array JSON	50
5.3. Value JSON	51
5.4. String JSON	51
5.5. Number JSON	52
5.6. Estructura	53
5.7. Foraneas line	58
5.8. Problema llaves foraneas	59
5.9. Ejemplo foraneas unidas	59
5.10. primer intento	60
5.11. segundo intento	60
5.12. tercer intento	61
5.13. cuarto intento	61
7.1. Base de datos prueba	65
7.2. lista de tablas	66
7.3. Lista de proyectos	66
7.4. Formulario para crear un nuevo proyecto	67
7.5. Conexion exitosa	68
7.6. Boton crear	68
7.7. Proyecto creado	69
7.8. Lista de tablas obtenida	69
7.9. Tipo de dato	70
7.10. Llenar campo para la tabla	70
7.11. Llenado exitosa	71
7.12. Tabla <code>usuario_rol</code>	71
7.13. Estado de configuracion	72
7.14. Llenando la base de datos	72
7.15. Tabla docente	73
7.16. Tabla rol	73
7.17. Sql generado	74

Índice de cuadros

3.1. Tipos de datos	16
3.2. Tipos de datos de red	23
4.1. tablas que referencian a otra	41
4.2. tabla de referencias para la tabla detalle	43
4.3. tabla referencias formateada	44
4.4. tabla con columnas aumentadas para la tabla <i>detalle</i>	46
4.5. columnas de la tabla detalle que referencian a venta	46
4.6. columna de la tabla detalle que referencia a producto	46
4.7. tabla de muestra de atributos foráneas	47
4.8. tabla sin los atributos foráneas	47

Resumen

En el desarrollo de software que implica el uso de base de datos, es bastante común realizar pruebas para conocer el comportamiento del software conjuntamente con la base de datos, para lo cual es necesario tener la base de datos de prueba similar a una base de datos en un entorno de producción.

Si bien se tiene claro que es necesario tener la base de datos con datos de prueba, no resulta ser tan sencillo poblar con datos de prueba, ya que como resulta ser tedioso, debido a que no siempre se tiene en mente lo que se quiere, tener en mente nombres de las tablas y quien referencia a quien, por lo cual normalmente se acostumbra a insertar pocos datos de prueba, como resultado deja la incertidumbre de como vaya a ser el comportamiento de las consultas que se realiza. Este trabajo presenta propuestas de algoritmos y técnicas que ayudan en el llenado de datos de prueba en una base de datos, tomando en cuenta las referencias compuestas comunes en modelos basados en ER-Idioms, ademas se presenta algoritmos para obtener el orden correcto en que deben ser llenado con datos de prueba las tablas de una base de datos para lo cual se tiene algoritmos que generan datos según el tipo dato. Para el caso de las llaves foraneas se ofrece mecanismos que autogeneran a partir de la tabla referenciada para evitar la inconsistencia de datos. Otro caso importante es el tipo de dato binario, en este proyecto se opta por trabajar con Postgresql que ofrece su tipo de dato `bytea` para almacenar archivos en la base de datos como demostración de todo lo desarrollado se presenta un prototipo que genera datos directamente a la base de datos con la opción de crear un archivo con extensión sql siempre que la base de datos no tenga tipo de datos binarios.

Capítulo 1

Introducción

Cuando se está desarrollando software que hacen uso de una bases de datos, que está en gran parte de las aplicaciones, es común tener la necesidad de realizar pruebas sobre la base de datos, para verificar si los datos se están gestionando de la manera correcta o verificar la eficiencia de las consultas, para esto se insertan datos de prueba haciendo uso de comandos del Lenguaje de definición de datos (DDL) [3]. Otra manera es hacerlo por una interfaz gráfica dependiendo del sistema gestor de base de datos, entre las mas usadas se tiene a:

- **Postgresql:** PgAdmin.
- **MySQL:** PhpMyAdmin, MySQL Workbench.
- **SQLServer:** SQLServer managment studio.

Los que trabajan con base de datos llenaron datos de prueba de esta manera en algún momento o siguen con los mismos métodos.

Insertar datos de prueba en una base de datos de forma manual sea por interfaz gráfica o por una interfaz de texto como ser la consola, no suele ser agradable porque: conlleva mucho tiempo de trabajo, no siempre se tiene en mente lo que se quiere. Por lo que, normalmente se insertan pocos datos de prueba para ver el comportamiento de las consultas que se realizan en el software.

Una aplicación en un entorno de producción de seguro no lleva pocos datos, al contrario llegan a almacenar gigabytes de información.

El problema surge ahí, porque no es lo mismo hacer pruebas con cien datos que con cien mil datos o más, si bien una consulta funciona de manera correcta con los pocos datos, es diferente el comportamiento con una población de datos mas grande, puede que las consultas no funcionen de forma correcta. Un error común en principiantes es realizar un `SELECT * FROM nombre_tabla` que de seguro no tendría problemas con los pocos datos. Sin embargo, en un entorno de producción es catastrófico.

Sin duda es un problema a considerarse en el desarrollo de software, sobre todo cuando no se realizan pruebas a una base de datos, dejándonos incertidumbres sobre su comportamiento con cantidades de datos que podría tener cuando el software ya esté en un entorno de producción.

1.1. Aplicaciones generadores de datos de prueba para base de datos

Para los que son nuevos en el mundo de base de datos es normal desconocer sobre herramientas que facilitan la tarea, y optan en llenar de forma manual una base de datos, que resulta ser tedioso a la vez frustrante llenar con datos de prueba una base de datos, si a principios de la aparición de la base de datos no se contaba con herramientas de apoyo en esta área, hoy en día existen herramientas que automatizan procesos como el llenado de datos de prueba en cantidades grandes sobre una base de datos ya existente.

Entre las herramientas para la generación de datos de prueba para base de datos, se puede mencionar a: Datanamic Data Generator MultiDB perteneciente a Datanamic [5], Generatedata para mysql de código abierto con licencia GNU [7], EMS Data Generator for MySQL de la línea de EMS y EMS Data Generator for PostgreSQL también de la línea de EMS [11] y MyDataGen [4]. Son algunas que se puede mencionar lo cual no significa que sean las únicas, con estas herramientas es posible realizar el poblado de datos para su posterior realización de pruebas.

De las herramientas la mas destacada es Datanamic, por la forma en como permite configurar. En la página oficial [5] están disponibles para MySql, PostgreSQL, Oracle y otros. En el caso de Datanamic para PostgreSQL es una de las más destacables entre las mencionadas ,donde se puede ver opciones de generar datos lo primero es escoger una base de datos existente en otra base de datos y lo que hace el software es reconocer características de la base de datos, con sus respectivos tipo de datos y por defecto presenta la opción de generar cien registros por tabla, con la libertad de configurar a gusto además da la opción de elegir la fuente de datos que se hará uso, a partir de una lista, otro es obtener datos como nombres por ejemplo a partir de un archivo, y poder seleccionar y escoger si será aleatoriamente o una forma secuencial, si será único o si se repetirá esto dependiendo de cómo se quiere llenar datos.

De las otras herramientas mencionadas tienen una similitud en el manejo y en cómo se realiza el llenado de los datos, a diferencia de Generatedata que es una herramienta libre y de código abierto escrita en JavaScript, PHP y MySQL, permite generar de una forma rápida grandes volúmenes de datos personalizados en una variedad de formatos, para su uso en pruebas de software, llenar bases de datos, etc. Los desarrolladores pueden escribir sus

propios tipos de datos para generar nuevos tipos de datos aleatorios e incluso personalizar los tipos de exportación. Para las personas interesadas en la generación de datos de localización geográfica, se pueden añadir nuevos complementos para proporcionar nombres de regiones (estados, provincias, territorios, etc.), nombres de ciudades y formatos de códigos postales para su país, todo esto porque es libre de código abierto. Las críticas que podría tener esta herramienta es porque no hace el llenado a una base de datos existente lo cual quita puntos a favor además que solo funciona para MySQL entre sus punto a favor es que es libre de código abierto. A diferencia Datanamic viene para distintos motores de base de datos pero si hay uno que es multifunciona(Datanamic DataDiff MultiDB).

Haciendo el uso de cualquiera de las herramientas mencionadas, el tiempo de llenando datos de prueba en una base de datos es considerablemente inferior a lo que llevaría hacerlo manualmente, cuanto más datos mayor es el tiempo en llenarlo en la forma manual. Sin embargo, haciendo uso de alguna de estas herramientas que ayudan en realizar esta tarea por nosotros, el tiempo es muy similar que llenar pocos registros así que es muy conveniente llenar la mayor cantidad de datos en la base de datos siempre que se disponga una fuente por ejemplo una lista de nombres, si se quiere llenar nombres en una entidad.

Sería mucho mejor encontrar información sobre cómo llenar ya que hay poca información sobre estas herramientas con una documentación no muy clara de algunas como MyDataGen, PgDataGen que sin embargo Datanamic si cuenta con una documentación mas clara [6] de cómo se realiza el llenado.

Una de las deficiencias que se puede encontrar y que le quita puntos a su favor, es cuando se tiene relaciones compuestas no las reconoce como tal y esto llega a ser un problema.

Las herramientas comerciales como es el caso de Datanamic, MyDataGen y PgDataGen, no provee el acceso al código fuente y es un problema saber cómo es la lógica de la generación de datos, pero es observable mediante la interfaz gráfica el como elige el generador de datos adecuado para cada columna, basado en las características de la columna, con más de cuarenta generadores de datos incorporados (específicos para países e idiomas), genera datos realistas con el uso inteligente del generador (para, por ejemplo, códigos postales) y una gran colección de listas con nombres, direcciones, ciudades, calles, etc. Obtiene datos al azar de una fuente externa y obtiene datos de fuentes existentes como las otras tablas, opción para deshabilitar los desencadenantes, vista previa en tiempo real de los datos que se van a generar, genera datos de prueba para una base de datos completa o para una selección de tablas, incluye una utilidad de línea de comandos para automatizar aún más el proceso de generación de datos, inserta datos directamente en la base de datos o genera un secuencia de comandos SQL con instrucciones de inserción, guarda tu plan de generación de datos de prueba a un archivo de proyecto, validación extensa de configuración del generador de datos, detección automática de los cambios en el esquema de base de datos, opción de generar valores únicos.

De todas estas características es interesante conocer acerca de cómo realiza la generación de datos, al tratar de ver como lo realiza, no se cuenta con información suficiente de parte de la aplicación, solo se puede visualizar. Sin embargo el objetivo es entender cómo hace la generación de datos según lo requerido, al observar las herramientas listadas, los datos que generan van dependiendo siempre del tipo de dato, y poder tomar como parámetros la cantidad de datos si serán únicos entre otras, todas estas características da una idea de como hacer un generador de datos.

Al hacer uso de cualquiera de estas herramientas se puede encontrar con un problema que se consideraría que es una deficiencia, ninguna de las mencionadas trata de ayudar al usuario en el orden del llenado de las tablas, si bien internamente lo hace la diferencia entre las tablas que se deben configurar primero y cuales las siguientes así sucesivamente.

Capítulo 2

Algoritmos de ordenamiento y mecanismos de manejo referencial

Una base de datos relacional esta fuertemente ligada al concepto Entidad Relación [1](ER “Entity Relationship”). Una entidad que representa gráficamente a un concepto del mundo real o abstracta, que da lugar a una tabla en la base de datos. Una relación entre dos o más entidades describe alguna interacción entre las mismas, el tipo de relación dará lugar a un comportamiento entre las entidades involucradas [8].

En un modelo Entidad Relación (E-R) para base de datos basados en ER Idioms[10], se tiene patrones de diseño más definidos, las relaciones que llegan a tener entre entidades y la forma en que se hacen da lugar prácticamente a los siguientes siete patrones de diseño para un modelo ER.

1. Una entidad que no hace referencia a otra pero si es referenciada es una entidad de tipo *catalogo*. Actúa como un tipificador y generalmente almacenan pequeñas cantidades de datos, los datos que se almacenan se conocen a priori y la cantidades de datos son predecibles lo cual no significa que sean estables pueden llegar ha incrementarse, la entidad que hace referencia al *catalogo* llega a ser una entidad de tipo *catalogado*.
2. En algunos casos se tiene entidades que se encuentran sueltas por lo tanto no hacen referencia ni son referenciadas por ninguna otra, la cual es una entidad de tipo *simple*.
3. En un modelo entidad relación donde una entidad hace referencia a más de una y que su existencia depende de las mismas, a todo este conjunto se le denomina *composición*.
4. Cuando una entidad es dependiente de la existencia de otra al que detalla es de tipo *detalle* de la alguna entidad maestra, donde el detalle obedece a la maestra, a diferencia de un catalogador en un maestro no se puede determinar los posibles datos a priori ni mucho menos estimar la cantidad aproximada de datos que pueda tener y que generalmente almacena cantidades grandes de datos.

5. Hay veces que una entidad referencia a otra del mismo tipo llegando a ser una relación recursiva la cual llega a ser una entidad de tipo *reflexivo simple*, cuando se implementa este tipo de relación es importante tener en cuenta que la relación no debe ser de obligatoriedad.
6. En ocasiones es necesario relacionar entidades del mismo tipo y guardar un historial de ellas, la forma de representar este concepto es que una entidad representa la forma que se relacionan la entidad que hace una doble referencia, lo cual lleva a ser una entidad tipo *reflexivo compuesto*.
7. Cuando se quiere hacer una especialización a una entidad en particular esta llega a ser la entidad hija de la entidad generalizada, este tipo de relación es conocida como *is a* en idioms.

2.1. Técnicas de ordenamiento

En una base de datos las tablas tienen una secuencia de prioridades en el llenado de datos, una manera de obtener esta secuencia es buscar todas las tablas que no tengan ningún **foreign key**, posteriormente las tablas que tienen como **foreign key** que refieren a las anteriores y así sucesivamente de manera secuencial, hasta acabar con el último al que no referencia ninguna otra tabla, esto llega a ser confuso sobre todo si son muchas tablas al momento de hacer el llenado, para lo cual es necesario alguna técnica para obtener el orden correcto del llenado en la base de datos.

La propuesta en este proyecto sobre la técnica para obtener el orden según la prioridad en que deben ser llenado las tablas con datos de prueba, es identificar primero todas aquellas que son una entidad tipo (**entity type**) catalogador, simple y aquella que no tenga dependencia de ninguna otra, puede que en algunos casos estén los de tipo maestra o las que son padre de una generalización, los siguientes a identificar son los catalogados, las que hacen referencia a las identificadas anteriormente pero que estas no deben depender de otras que aún no se llenó para evitar tener problemas de inconsistencia. Un caso especial y para tener cuidado es la entidad tipo reflexivo simple, una entidad que hace referencia a uno de su mismo tipo con características ya mencionadas anteriormente, este tipo de entidad va en la primera siempre que no haga referencia a una distinta de si misma.

2.2. Algoritmo de ordenamiento

Para obtener la lista de tablas según el orden en que debe ser llenado una base de datos, es necesario tener una lista capaz de almacenar elementos de diferentes contenido, esto debido

a que el primer elemento puede ser un conjunto diferente al segundo para que es necesario aplicar el siguiente algoritmo:

1. Crear una lista de conjuntos.
2. Seleccionar las entidades de tipo catalogo, simple, maestra que no dependan de otra entidad y entidades que sean de tipo *reflexivo simple*, que no hagan referencia a otra distinta de el, para luego almacenar este conjunto en la lista.
3. De la lista de conjuntos obtener el ultimo conjunto de entidades almacenada y por cada elemento buscar todas las entidades que le hagan referencia y que estas llegan a formar parte de otro conjunto de entidades, una vez recorrido todos los elementos agregar como un nuevo conjunto en la lista creada anteriormente.
4. Verificar que se ha recorrido todas las entidades, en caso de que se recorrió todo pasar al paso cinco, en caso contrario repetir el paso tres.
5. Si se llego hasta aquí es porque se tiene la lista de conjuntos según a las dependencias, es decir un elemento de un conjunto referencia a otro elemento del conjunto anterior por lo tanto esta aun no es la lista.

Ya que se da el caso de que el nombre de una entidad se repita en más de una ocasión dentro de un mismo conjunto o puede darse el caso que se repita el nombre en un conjunto anterior o posterior, por lo tanto es necesario aplicar el algoritmo de ordenación; primeros en ser llenados a la matriz de conjuntos. Antes de pasar al dicho algoritmo vease en un ejemplo en la siguiente sección.

2.3. Aplicación del algoritmo

En la Figura 2.1, se tiene casos de tipos de entidades (`entity type`) según sea el caso se observa PFK `primary foreign key` llave foranea que a la vez es parte de la llave primaria, PK `primary key` llave primaria comunmente usadas en modelos ER Idioms.

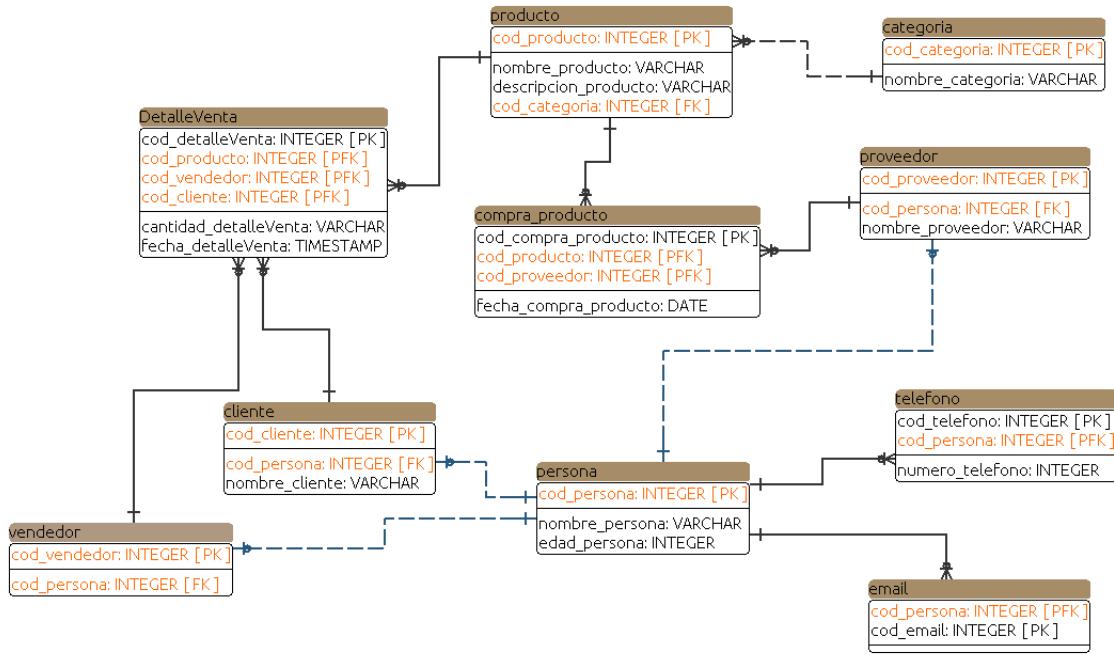


Figura 2.1: Ejemplo modelo ER

Aplicando el paso dos del algoritmo se identifica una entidad catalogador *categoria* y otra entidad que generaliza *persona*, a partir ellas se busca los que le hacen referencia, *categoria* y *persona* llega a ser la raíz del árbol a generar, las entidades que hacen referencia serían (*producto*, *cliente*, *vendedor*, *proveedor*, *telefono*, *email*) y un nivel mas abajo (*detalleVenta* y *compra_producto*) aplicando el algoritmo se llega a la siguiente Figura 2.2.

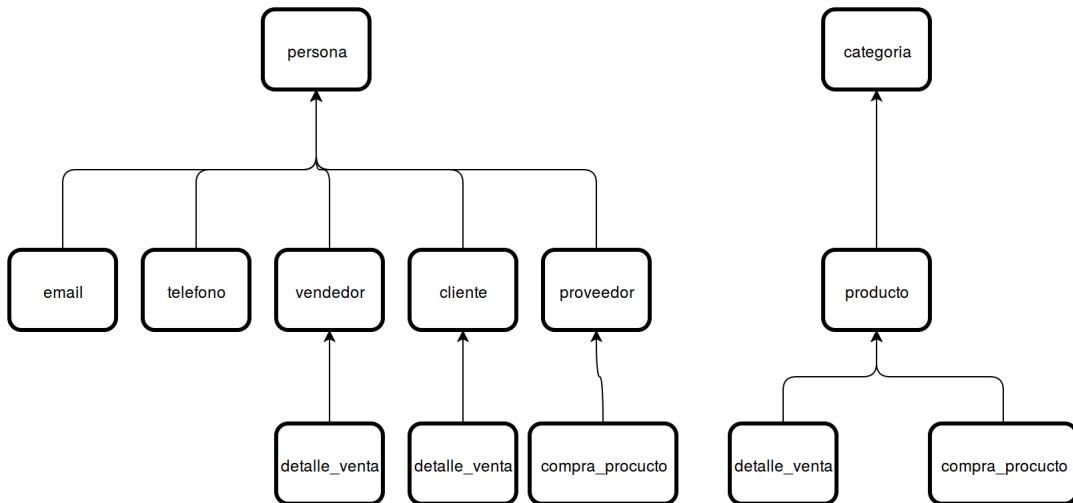


Figura 2.2: Modelo Ordenado

2.4. Algoritmo de ordenación *primeros en ser llenado*

Para obtener una lista de acuerdo al orden que se debe realizar el llenado, se inicia el recorrido de los conjuntos que trae la lista obtenida del algoritmo anterior, porque la lista se ordenó según a que una entidad no dependiera de otra entidad, al recorrer se puede encontrar que una entidad se repita en distintos conjuntos, al inicio de un conjunto, puede también estar listada en medio o al final incluso se puede dar el caso que en un conjunto se repite más de una ocasión el nombre de una entidad. En la Figura 2.2 la entidad *detalleVenta* se repite más de una vez en la misma fila.

Entonces cual tomar como válido?. Para lo cual el recorrido se comienza con el último conjunto que tiene la lista, una vez encontrada las demás que puedan encontrarse en posteriores no llegará a ser válida porque si bien hizo una referencia a un principio será valida el último. Esto se justifica porque si se encuentra al último es porque aún hace referencia a algún elemento de la matriz anterior al que pertenece, por lo tanto se debió esperar que se llegue hasta ese punto. para el caso de la Figura 2.2 listar el último evitando que se repita. Para obtener una lista ordenada según el orden en que deben ser llenados es necesario seguir la siguiente secuencia de pasos:

1. Crear un matriz de tamaño de la cantidad de entidades de la base de datos seleccionada.
2. Obtener el ultimo conjunto de entidades de la matriz.
3. Recorrer la matriz y verificar por cada elemento no exista en la matriz del paso uno, en caso de que no exista añadir, de lo contrario no añadir y pasar a la siguiente elemento.
4. Eliminar el ultimo conjunto de entidades de la matriz.
5. Si existen mas conjuntos de entidades en la matriz bidimensional volver a repetir el paso dos, caso contrario pasar al siguiente paso.
6. Invertir el orden de la matriz del paso uno.
7. Retornar la matriz creado en el paso uno que llegaría a ser el orden que se requiere.

2.5. Aplicación del algoritmo de *primeros en ser llenado*

Con el algoritmo se llega a tener el siguiente orden en que debe ser llenado para el ejemplo dado en la figura 2.1, el orden a seguir para este ejemplo es iniciar por el color verde terminando en el color rojo, el orden en cada fila no influye en el resultado permitiendo así la

flexibilidad de tomar cualquier elemento para el inicio de cada fila o el conjunto de elementos del mismo color.



Figura 2.3: Orden de llenado

2.6. Mecanismo de *manejo de referencial* de una base de datos

Cuando un modelo entidad relación se lleva a un sistema gestor de base de datos, donde por cada entidad se crea una tabla y las referencias son representadas mediante las llaves primarias y llaves foráneas, En caso de un modelo entidad relación basado en ER Idioms tiene ciertas características que se explican en las siguientes subsecciones.

2.6.1. Llaves primarias compuestas (composite keys)

Cuando se tiene más de un **primary key**, entre ellas las que son propias de la tabla y otras pertenecientes a las que referencia que vienen como primarias, estas llegan a formar parte del **primary key** de la tabla, formando así **composite keys** para la misma. En conceptos de entidad relación en la figura 2.4 se puede observar las entidades que hacen referencia a otra. Donde se puede ver que la entidad *usuario_rol* hace referencia a las entidades *usuario* y *rol*, las tres entidades llegarían a formar una composición (*usuario_rol* compone de *usuario* y *rol*) donde la existencia de *usuario_rol* es dependiente de las que compone

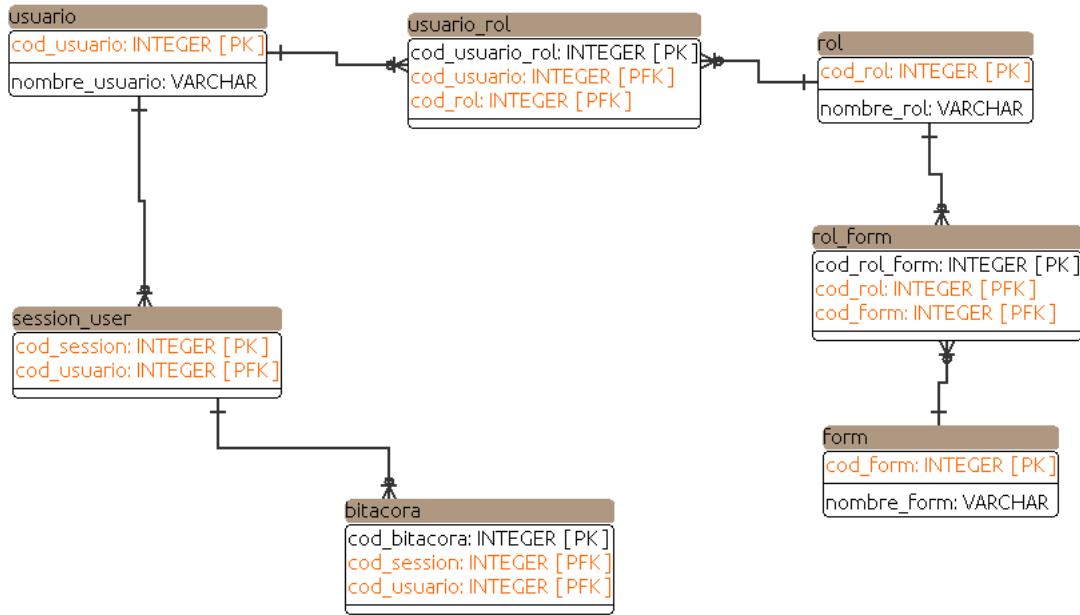


Figura 2.4: Llaves compuestas

Si este modelo se lleva a un gestor de base de datos un DBMS llega a tener como en el código 2.1.

```

1 CREATE TABLE usuario_rol(
2   cod_usuario_rol serial NOT NULL ,
3   cod_usuario INTEGER NOT NULL ,
4   cod_rol INTEGER NOT NULL ,
5   CONSTRAINT cod_usuario_rol PRIMARY KEY (cod_usuario_rol,cod_usuario,
6   cod_rol),
7   CONSTRAINT rol_usuario_rol_fk FOREIGN KEY (cod_rol)
8     REFERENCES rol (cod_rol) MATCH SIMPLE
9     ON UPDATE NOT ACTION ON DELETE NOT ACTION ,
10  CONSTRAINT usuario_usuario_rol_fk FOREIGN KEY (cod_usuario)
11    REFERENCES usuario (cod_usuario) MATCH SIMPLE
12    ON UPDATE NOT ACTION ON DELETE NOT ACTION
13 )

```

Código 2.1: SQL tabla usuario_rol

El primary key propia es independiente en cambio *cod_usuario* es perteneciente a la tabla *usuario* pero viene como primary key lo mismo sucede con el campo *cod_rol* que pertenece a la tabla *rol*, como ambas foreign key vienen como primary key la tabla *usuario_rol* llegaría a tener un primary key compuesta de tres *cod_usuario_rol*, *cod_usuario*, *cod_rol*. Cuando se quiere hacer una inserción de un registro a la tabla *usuario_rol* sin antes haber realizado una inserción a las tablas de *usuario* y *rol* cualquier DBMS no lo realiza la inserción por razones de primero debe existir datos en las tablas.

2.6.2. Mejor uso de Join

El hacer uso de llaves compuestas(**composite keys**) hace que un identificador llegue más allá de lo que normalmente se acostumbra vease en la figura 2.5

Donde en la entidad bitácora se tiene tres **primary key** llega a ser una llave compuesta(**composite key**) y una de ellas es *cod_usuario* que si bien viene de *session_user* realmente su origen es en *usuario*, por lo tanto se puede hacer un **join** entre *bitacora* y *usuario* evitando hacerlo con *session_user* obteniendo así una consulta mas eficiente.

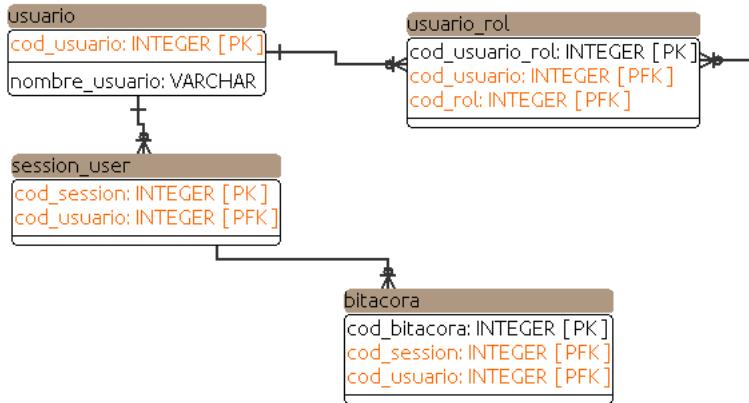


Figura 2.5: Modelo E-R con llaves compuestas

2.7. Mecanismo de referenciación

Cuando se genera datos de prueba para una base de datos es importante tener en cuenta el manejo de las llaves compuestas, no se puede generar al azar los **foreign key** porque se llegaría a tener problemas de inconsistencia.

Una técnica para evitar los problemas de inconsistencia es tener como fuente la tabla al que se referencia y los atributos de la tabla referenciada, tomarlo como base para la generación de *n* datos.

2.7.1. Referencia simple

Las referencias simples son cuando una tabla recibe solo un **primary key** o **foreign key** por parte de otra, en el código 2.2 se puede observar el SQL (por sus siglas en inglés Structured Query Language) de la tabla *usuario_rol* de la figura 2.4. La tabla mencionada esta conformada de dos llaves que no son propias de la misma, *cod_usuario* forma parte de la tabla *usuario* y *cod_rol* apunta a la tabla *rol*, ambos son campos que apuntan a su respectiva tabla de manera única, por lo tanto, se deja entender que se hace una referencia simple de llaves.

```

1 CREATE TABLE usuario_rol(
2   cod_usuario_rol serial NOT NULL ,
3   cod_usuario INTEGER NOT NULL ,
4   cod_rol INTEGER NOT NULL ,
5   CONSTRAINT cod_usuario_rol PRIMARY KEY (cod_usuario_rol,cod_usuario ,
6   cod_rol),
7   CONSTRAINT rol_usuario_rol_fk FOREIGN KEY (cod_rol)
8     REFERENCES rol (cod_rol) MATCH SIMPLE
9     ON UPDATE NOT ACTION ON DELETE NOT ACTION ,
10  CONSTRAINT usuario_usuario_rol_fk FOREIGN KEY (cod_usuario)
11    REFERENCES usuario (cod_usuario) MATCH SIMPLE
12    ON UPDATE NOT ACTION ON DELETE NOT ACTION
13 )

```

Codigo 2.2: Referencia simple

2.7.2. Referencia compuesta

La referencia compuesta ocurren cuando una tabla recibe más de un `primary key` o `foreign key` por parte de otra. Es importante tomar los atributos que apuntan a otra como un conjunto de atributos para manejar como base para la generación de los datos requeridos.

```

1 CREATE TABLE bitacora(
2   cod_bitacora serial NOT NULL ,
3   cod_session INTEGER NOT NULL ,
4   cod_usuario INTEGER NOT NULL ,
5   CONSTRAINT bitacora_pk PRIMARY KEY (cod_bitacora,cod_session,cod_usuario
6   ),
7   CONSTRAINT session_user_bitacora_fk FOREIGN KEY (cod_session,cod_usuario
8   )
9     REFERENCES "session_user" (cod_session,cod_usuario) MATCH SIMPLE
10    ON UPDATE NOT ACTION ON DELETE NOT ACTION
11 )

```

Codigo 2.3: Referencia compuesta

En el código 2.3 el campo `cod_session` y `cod_usuario` de la tabla `bitacora` hacen referencia a `session_user` a los campos `cod_session` y `cod_usuario`, son dos campos que referencia a la misma tabla, este caso `session_user` por lo tanto llega a ser una referencia compuesta.

Capítulo 3

Algoritmos de generación de datos de prueba

En el desarrollo de un software que hace uso de una base de datos es muy importante para los desarrolladores sobre todo para los que son encargados de la base de datos, trabajar con una base de datos con datos de prueba que se asemejen más a las reales tanto en cantidad como en tipo de datos, para lo cual es necesario tener una cantidad grande de datos de prueba cuanto más mejor, esto lleva a un enfoque de generar datos de prueba para base de datos.

Poseer con una cantidad suficiente de datos de prueba no es tan sencillo por lo que un desarrollador normalmente acostumbra hacer la inserción de forma manual lo cual toma un tiempo excesivo. Ademas cuando se quiere hacer la inserción es necesario tener en cuenta los diferentes tipos de datos que la gran mayoría de los SGBD (**Sistemas gestor de base de datos**) posee.

Los tipos de datos que manejan los SGBD mediante SQL permiten definir el formato de la información a usar en columnas, variables y expresiones. Cabe aclarar que dependiendo del SGBD a usarse los tipos de datos pueden cambiar.

Al igual que otros lenguajes de programación, para SQL se puede encontrar números enteros, flotantes, cadenas, booleanos y demás. entre los mas utilizados se tiene:

- Tipos de datos flotantes para representar numero de punto flotante (**float, real**).
- Tipos de datos temporales es seguro que en algún momento es necesario guardar registros que contengan información sobre fechas de cumpleaños, tiempo de llegada. Para este caso existen tipos de datos(**date, datetime, timestamp, time**).
- Cadenas de caracteres, guardar nombres, apellidos, direcciones y otros tipos de datos denominativos requiere que el uso de cadenas de caracteres para lo cual se tiene tipos de datos (**char, varchar, text**).

Cuadro 3.1: Tipos de datos

Tipo de dato	Características
VARCHAR(tamaño)	Almacena cadenas de caracteres de una longitud variable. La longitud máxima son 4000 caracteres
CHAR(tamaño)	Almacena caracteres con una longitud fija. Siendo 2000 caracteres el máximo
NUMBER(precisión, escala)	Almacena datos numéricos, tanto enteros como decimales, con o sin signo. Precisión, indica el número máximo de dígitos que va a tener el dato. Escala, indica el número de dígitos que puede haber a la derecha del punto decimal.
LONG	Almacena cadenas de caracteres de longitud variable. Puede almacenar hasta 2 gigas de información
LONG RAW	Almacena datos binarios. Se emplea para el almacenamiento de gráficos, sonidos, etc. Su tamaño máximo es de 2 gigas
DATE	Almacena información de fechas y horas. De forma predeterminada almacena un dato con el siguiente formato: siglo/año/mes/día/hora/minutos/segundos. Este formato se puede cambiar con otros parámetros.
RAW(tamaño)	Almacena datos binarios. Puede almacenar como mucho 2000 bytes.
ROWID	Se trata de un campo que representa una cadena hexadecimal que indica la dirección de una fila en su tabla
NVARCHAR2(tamaño)	Es similar al varchar2 pero el tamaño de un carácter depende de la elección del juego de caracteres. El tamaño máximo es 2000 bytes.
NCHAR(tamaño)	Similar al CHAR y con las mismas características que el nvarchar2
CLOB	Similar al LONG y se usa para objetos carácter
BLOB	Similar al LONG RAW. Este se usa para objetos binarios.

3.1. Algoritmos de generación de datos

La generación de datos de prueba para base de datos no llega a ser tan sencilla por los diferentes tipos de datos y el límite en el tamaño, pero llega a ser una solución para pruebas que se quiera realizar a una base de datos determinada. Para realizar la generación de datos es necesario tener algoritmos generadores por cada tipo de datos que tome en cuenta las características de la misma y sean capaces de generar cantidades grandes tomando como base una pequeña cantidad de datos o a lo mejor sin contar ninguna base.

3.2. Tipos numéricos

Existen varios tipos de datos para manejar números enteros dentro de SQL. Todas esas opciones se diferencian en el tamaño de memoria asignado para un rango de valores enteros.

Los números enteros es uno de los más sencillos a generar, con solo incrementar en una unidad al número inicial que se pasa como parámetro se llega en algún momento al límite que también se pasa como parámetro.

3.3. Tipos monetarios

Los tipos de datos monetarios se almacenan como un numero cualquiera y tiene una similitud con las decimales, el SGBD es quien se encarga de dar el formato necesario.

3.4. Tipos de caracteres

3.4.1. Generación de nombres

El generar nombres se puede hacer de varias formas en este proyecto se muestra dos maneras: a partir de una lista de nombres y apellidos o también se puede generar haciendo combinaciones de las vocales y las consonantes, a continuación se hará una descripción de cómo generar de las dos formas.

3.4.1.1. Generación de nombres a partir de vocales y consonantes

Un nombre se compone generalmente por más de tres caracteres puede que tenga menos y a lo mucho llega a tener 10 a excepción de algunos. Y mucho depende del idioma o el país. Para generar un nombre es importante tener claro lo mencionado anteriormente, para ello se aplica el siguiente algoritmo para generar nombres haciendo uso de las vocales y consonantes de la Figura 3.1.

1. Crear una matriz e insertar las cinco vocales.
2. Crear una matriz e insertar las consonantes se puede omitir las que no se desea usar.
3. Generar un número aleatorio entre 1 a 10 que representa la cantidad de caracteres del nombre y crear una variable al que se le asignará el nombre.
4. Declarar una variable bandera que indicará el turno a quien corresponde sea una vocal o consonante.
5. Preguntar si la variable bandera indica si es verdadero pasar al siguiente paso 6 caso contrario saltar al paso 7.
6. Generar un número aleatorio entre 0 a un máximo de la cantidad de elementos del arreglo de consonantes menos uno, este llega a ser el índice del elemento a obtener de las consonantes para luego realizar la concatenación a la variable que maneja el nombre, contradecir la variable bandera.
7. Generar un número aleatorio entre 0 a un máximo de 4 por la cantidad de vocales, esta llega a ser la posición del elemento a obtener del arreglo de vocales y luego concatenar a la variable que maneja el nombre, contradecir la variable bandera.

8. Preguntar si el tamaño del nombre es igual al número generado en el paso 3, si es verdadero pasar al paso 9 y si es falso volver al paso 5.

9. Finalizar y retornar el nombre generado.

Algoritmo 1 Algoritmo de generación de palabras

Entrada: mínimo , máximo de caracteres.

```

1: cantidad ← random(minimo,maximo)
2: numero ← random(0,1)
3: cadena ← ""
4: mientras n < cantidad hacer
5:   si numero == 0 entonces
6:     cadena ← cadena + obtenerVocal
7:   si no
8:     cadena ← cadena + obtenerConsonante
9:   fin si
10:  numero ← random(0,1)
11: fin mientras
12: devolver cadena

```

El Algoritmo 1 hace uso de vocales y consonantes para generar palabras.

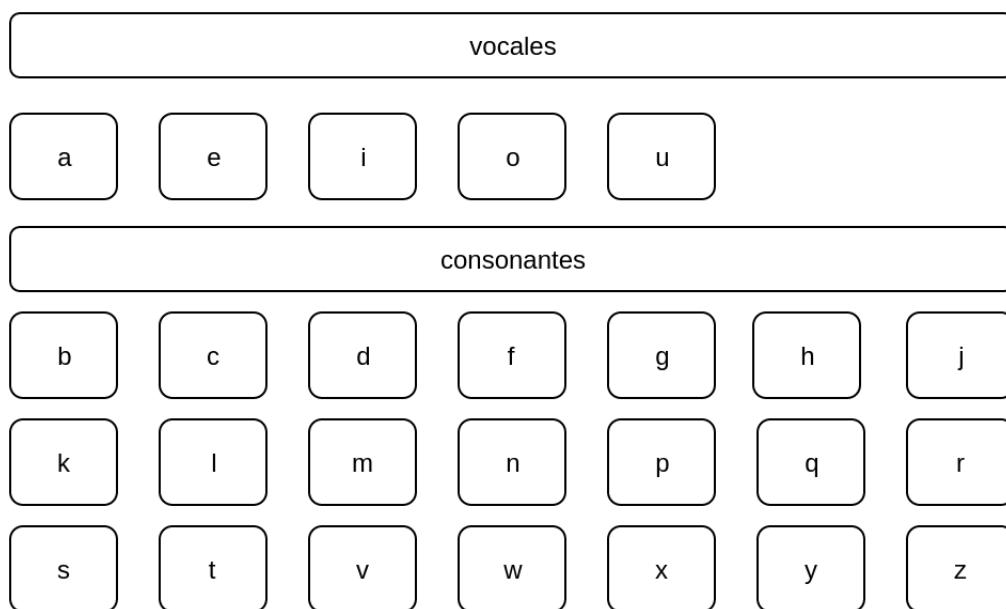


Figura 3.1: Consonantes y vocales

3.4.1.2. Generación de nombres a partir de una lista de nombres y apellidos

La generación de nombre más apellido requiere de una lista de nombres y otro de apellidos, para generar se aplica el algoritmo 2 basado en la combinación del producto cartesiano como se observa en la Figura 3.2.

Algoritmo 2 Algoritmo de generacion de nombresLista

Entrada: nombres [], apellidosPaternos [],apellidosMaternos []

```
1: nombresCombinadas [ ]
2: ind
3: para iniNomb ← 0; iniNomb < tamanio(nombres);iniNomb++ hacer
4:   para iniApePat ← 0; iniApePat<tamanio(apellidosPaternos);iniApePat++ hacer
5:     para iniApePat ← 0; iniApePat<tamanio(apellidosPaternos);iniApePat++ hacer
6:       nombresCombinados[ind] ← nombres [ iniNomb ] + apellidosPaternos [ iniApePat
          ] + apellidosMaternos [ iniApePat ]
7:     fin para
8:   fin para
9: fin para
10: devolver cierto
```

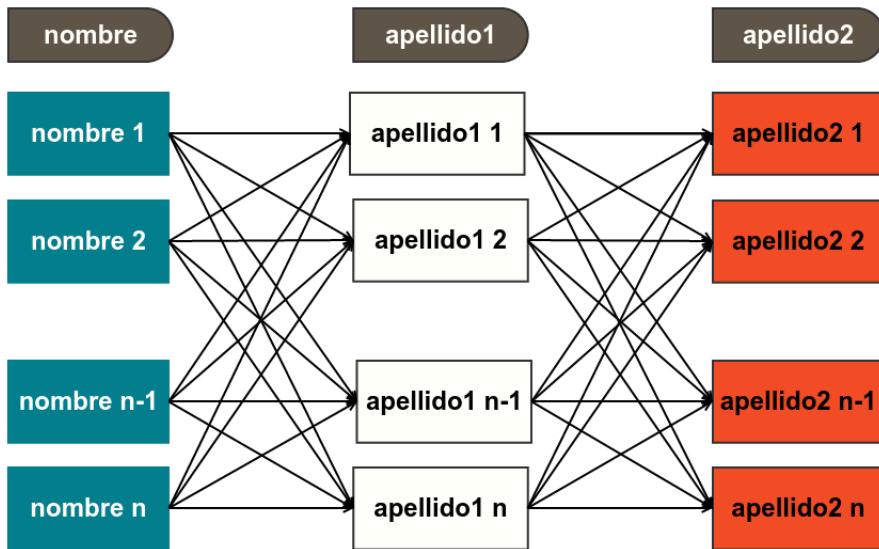


Figura 3.2: Apartir de una lista de nombres

3.5. Tipos de datos binarios(Binary Data Types)

Los sistemas gestores de base de datos(SGBD) permiten almacenar archivos en formatos como **bytea**, **blob** entre otros, variando estos segun el SGBD utilizado.

En este proyecto se pondra a consideracion trabajar con postgresql el cual permite almacenar archivos en formato `bytea`. El tipo de dato `bytea` permite almacenar objetos de gran tamaño, postgreSQL no conoce nada sobre el tipo de información que se almacena `bytea`, simplemente lo considera como cadena binaria, Una cadena binaria es una secuencia de octetos (o bytes).

Las cadenas binarias se distinguen de las cadenas de caracteres de dos maneras. En primer lugar, las cadenas binarias permiten específicamente el almacenamiento de octetos de valor cero y otros “no imprimibles” octetos (por lo general, octetos fuera comprendida entre 32 a 126). Las cadenas de caracteres no permite cero octetos, y también no permite ningún otro valor de octeto y secuencias de valores de octeto que no son válidos según la codificación de caracteres seleccionado de la base de datos. En segundo lugar, las operaciones en cadenas binarias procesan los bytes reales, mientras que el procesamiento de cadenas de caracteres depende de la configuración regional. En resumen, las cadenas binarias son apropiadas para el almacenamiento de datos que el programador piensa como “bytes en bruto”, mientras que las cadenas de caracteres son apropiados para almacenar texto.

El tipo `bytea` soporta dos formatos externos de entrada y salida de formato “escape” y “hex”. Ambos de estos siempre son aceptados en la entrada. El formato de salida depende del parámetro de configuración, el valor predeterminado es hexagonal. (Tenga en cuenta que el formato hexadecimal se introdujo en PostgreSQL 9.0. Versiones anteriores y algunas herramientas no lo entienden).

El SQL estándar define un tipo de cadena binaria diferente, llamado BLOB o de objeto binario grande . El formato de entrada es diferente de `bytea` , pero las funciones y operadores proporcionados son en su mayora de la misma.

3.5.1. Bytea formato hexadecimal

El formato “hex” de datos binarios codifica como 2 dígitos hexadecimales por byte. Toda la cadena es precedido por la secuencia del signo barra contrario (para distinguirlo del formato de escape). En algunos contextos, puede ser necesario escapado con la duplicación de ella.

3.5.2. Bytea escapar formato

El formato “escape” es el tradicional formato de PostgreSQL para el tipo `bytea`.

A momento de generar datos de prueba para base de datos, el tipo de dato `bytea` llega a ser un caso especial, debido a que no es posible crearlo como cualquier otro dato, como ser un numero de teléfono que llega a ser combinaciones de números bajo ciertas condiciones o el caso de un nombre que son combinaciones de vocales y consonantes, sin embargo `bytea` es posible generar haciendo uso de archivos existentes teniendo solo la ruta del archivo es

suficiente para poder insertar en la base datos.

3.6. Tipos fecha/hora

3.6.1. Generación de fechas

Una fecha esta compuesta por tres partes:

- Año la parte del año para nuestros días comprende de cuatro dígitos desde 1000 hasta el año 9999 el rango no estrictamente establecido.
- Mes. la parte del mes se representa en número de dos dígitos que comprende en un rango establecido, con un inicio de 01 hasta 12 representando los doce meses del año.
- Día. la cantidad de días en un mes es variable mucho depende a que mes se refiera, el rango comprende desde el día uno y con un final variable desde 28 a 31 días.

Para generar una cantidad de fechas es necesario tener una fecha inicial y final. a partir de ello se hace la combinacion cartesiana como se observa en la Figura 3.3.

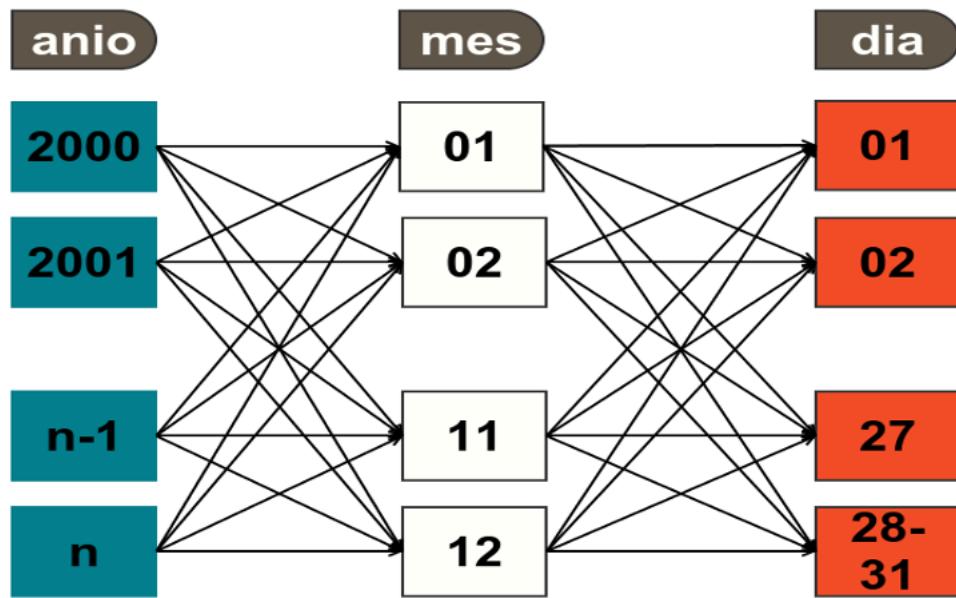


Figura 3.3: Generación de fechas

3.6.1.1. Algoritmo de generación de fechas

Para realizar la aplicación del algoritmo de generación de fechas es necesario tener dos datos una fecha inicial y otra fecha limite final donde es importante que la fecha inicial debe ser una fecha anterior a la final, la cantidad de fechas a obtener dependería del tamaño de rango que existe entre la inicial y la final.

A continuación se presenta el algoritmo 3 de generación de fechas en formato dd/mm/aa, tomando en cuenta que la cantidad de días es variable por cada mes además tomando en cuenta años bisiestos.

Algoritmo 3 Algoritmo de generación de fechas

Entrada: fechaInicio fechaFinal

```
1: fechas[ ]
2: contador ← 0
3: si fechaInicio < fechaFinal entonces
4:   mientras fechaInicio < fechaFinal hacer
5:     fechas [ contador ] ← fechaInicio+ 1 dia
6:     contador ← contador+1
7:   fin mientras
8: si no
9:   devolver error
10: fin si
11: devolver fechas
```

3.6.2. Generación de dato tipo Time

La estructura del dato tipo tiempo es muy similar a las fechas con la diferencia de que estas tienen un rango ya establecidos sin ninguna variación, esta compuesta por:

1. Hora. las hora se representa en un número de dos dígitos comenzando desde las 00 horas hasta las 23.
2. Minuto. los minutos también se representa en un número de dos dígitos con un inicio en 00 hasta 59 minutos.
3. Segundo. el rango es idéntica al de los minutos.

Para generar el tipo de dato time se hace combinaciones de las tres partes que tiene este tipo de dato, realizando combinaciones se obtiene la cantidad de datos que requerida.

En la Figura 3.4 se hace la combinación en modo cartesiano para la generación para este tipo de dato:

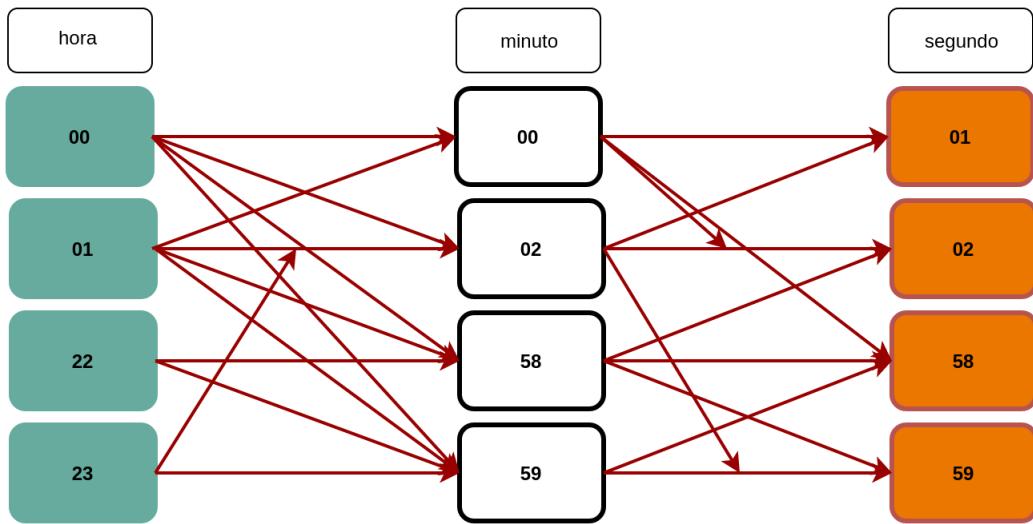


Figura 3.4: Generación de Time

Algoritmo 4 Algoritmo de generación de fecha hora

Entrada: fechaHoraInicio fechaHoraFinal

```

1: fechasHoras[ ]
2: contador ← 0
3: si fechaHoraInicio < fechaHoraFinal entonces
4:   mientras fechaHoraInicio < fechaHoraFinal hacer
5:     fechasHoras [ contador ] ← fechaHoraInicio+ 1 minuto
6:     contador ← contador+1
7:   fin mientras
8: si no
9:   devolver error
10: fin si
11: devolver fechasHoras

```

3.7. Tipos de direcciones de red

Las direcciones de red que almacena una base de datos son la IPv4, IPv6 y dirección mac como se observa con mayor detalle en el cuadro 3.2.

Cuadro 3.2: Tipos de datos de red

nombre	tamaño	descripción
cidr	12 ó 24 bytes	Redes IPv4 o IPv6
inet	12 ó 24 bytes	Hosts y redes IPv4 o IPv6
macaddr	6 bytes	Dirección MAC

3.7.1. Estructura de una dirección IPv4

Al igual que la dirección de una casa que tiene dos partes (una calle y un código postal), una dirección IP también está formada por dos partes: el ID de host y el ID de red.

3.7.1.1. ID de red

La primera parte de una dirección IP es el ID de red, que identifica el segmento de red en el que está ubicado el equipo. Todos los equipos del mismo segmento deben tener el mismo ID de red, al igual que las casas de una zona determinada tienen el mismo código postal.

3.7.1.2. ID de host

La segunda parte de una dirección IP es el ID de host, que identifica un equipo, un router u otro dispositivo de un segmento. El ID de cada host debe ser exclusivo en el ID de red, al igual que la dirección de una casa es exclusiva dentro de la zona del código postal. Es importante observar que al igual que dos zonas de código postal distintas pueden tener direcciones iguales, dos equipos con diferentes IDs de red pueden tener el mismo ID de host. Sin embargo, la combinación del ID de red y el ID de host debe ser exclusivo para todos los equipos que se comuniquen entre sí. Las clases de direcciones se utilizan para asignar IDs de red a organizaciones para que los equipos de sus redes puedan comunicarse en Internet.

Para generar un dato de este tipo se puede hacer una combinación cartesiana entre un rango de números entre 1 hasta 253 como se observa en la Figura 3.5.

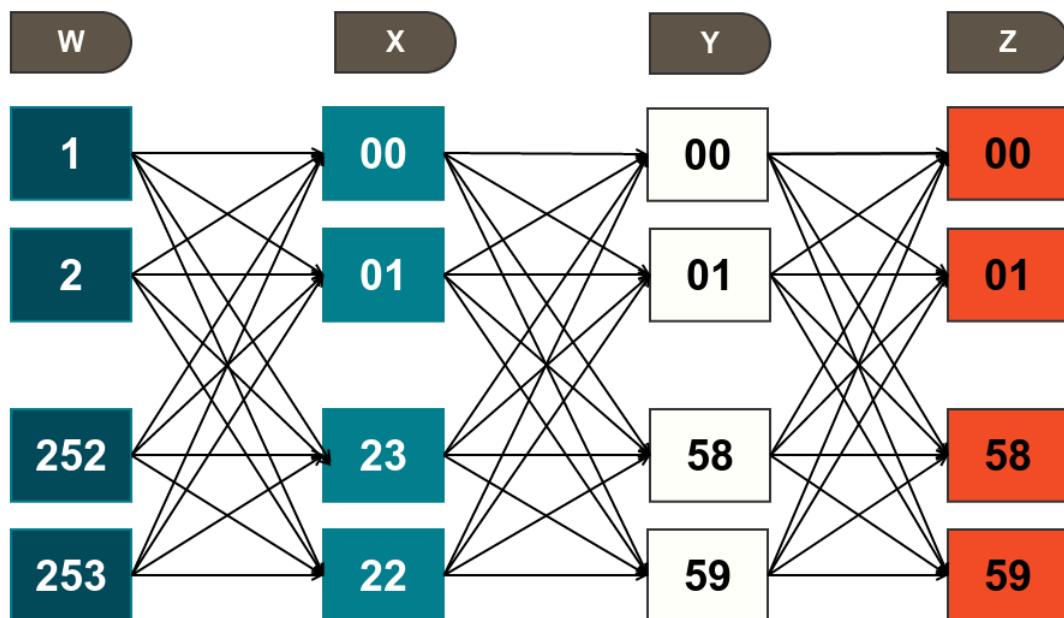


Figura 3.5: Combinación IPv4

A continuación se presenta el algoritmo 5 para generar direcciones IPv4 para lo cual es necesario un dato de entrada de inicio y otra final.

Algoritmo 5 Algoritmo de generación de IPv4

Entrada: inicio final

```
1: direcciones[ ]
2: contador ← 0
3: si inicio < fiinal entonces
4:   mientras inicio < final hacer
5:     direcciones [ contador ] ← inicio+ 1
6:     contador ← contador+1
7:   fin mientras
8: si no
9:   devolver error
10: fin si
11: devolver direcciones
```

Capítulo 4

Ordenando metadatos y aplicando algoritmos propuestos

En este capítulo se hará muestra de las técnicas necesarias para obtener la estructura de una base de datos, una base de datos existente por lo general lleva tablas que de alguna manera se relacionan entre ellas. El detallar la estructura de una base de datos comprende:

- Listar todas las tablas de una base de datos.
- Listar las tablas que hacen referencia y a que tablas.
- Listar las llaves primarias de una tabla.
- Listar las llaves foráneas de una tabla.
- Detallar los atributos de una tabla como ser el tipo de dato, tamaño, si puede ser nulo.

Es muy importante realizar lo listado anteriormente para obtener la estructura de una base de datos, y usar estos resultados para construir una interfaz gráfica de configuración de generación de datos de acuerdo a las características de la columna de una tabla. Obtener la estructura de una base de datos puede variar dependiendo del DBMS(Database Management System) con la que se trabaja, entre los DBMS se tiene a varios, las mas usadas son MySQL, PostgreSQL, Oracle, SQLServer y otras. En este proyecto se hace la elección de trabajar con PostgreSQL las razones para esta elección son las siguientes.

- Soporta llaves primarias compuestas(lo cual permite aplicar patrones de diseño ER Idioms).
- Es un DBMS de licencia BSD libre.

Esto no significa que en las otras no se pueda aplicar este proyecto de lo contrario son aplicables a DBMS relacionales claro que existen diferencias de como manejan los datos

internamente cada una de ellas, se puede mencionar que PostgreSQL almacena en metadatos todas las tablas que se crea, por lo tanto se deduce que para obtener la estructura de una base de datos es necesario trabajar con metadatos de PostgreSQL.

4.1. Metadatos en PostgreSQL

PostgreSQL tiene una arquitectura que involucra muchos estilos, en su nivel mas alto es un esquema clásico cliente-servidor, mientras que el acceso a metadatos es un esquema en capas como se observa en la Figura 4.1 donde:

- Libpq es el responsable de manipular las comunicaciones entre la aplicación cliente y el postmaster(Servicio del PostgreSQL en el servidor).
- El servidor esta compuesto por dos grandes subsistemas, “Postmaster” que es el responsable de aceptar las comunicaciones con el cliente, autenticar y dar acceso. “Postgre” se encarga de la administración de los consultas(querys) y comandos enviados por el cliente. PostgreSQL trabaja bajo el concepto de “process per user”, eso significa un solo proceso cliente por conexión. Tanto el Postmaster como el “Postgre” deben estar junto en el mismo servidor siempre.
- El gestor de almacenamiento (Storage Manager) es responsable de la administración general del almacenamiento de los datos, controla todos los trabajos del back end incluido la administración del buffer, archivos, bloqueos y control de consistencia de la información.

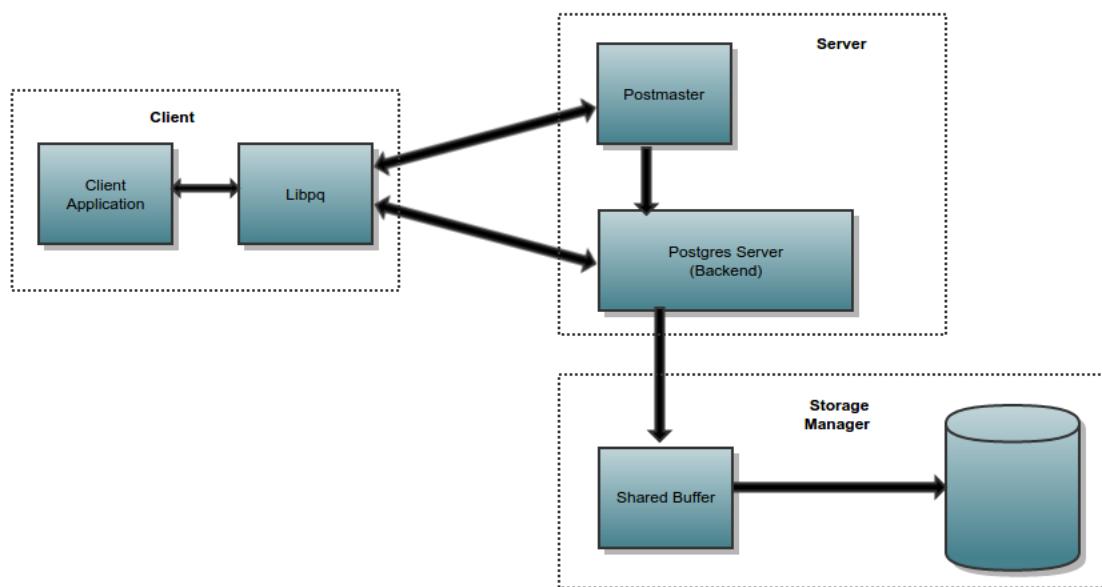


Figura 4.1: PostgreSQL System Concept Architecture [9]

4.1.1. Almacenamiento y organización de datos

Los datos siempre se guardan en el “disco” (esto puede no ser literalmente un Hard Drive). Esto genera un intenso trabajo de I/O(entrada y salida), cuando se lee datos se saca del “disco” para pasarlo a la memoria RAM, cuando se escribe se baja de la RAM al “disco” como se observa en la Figura 4.2.

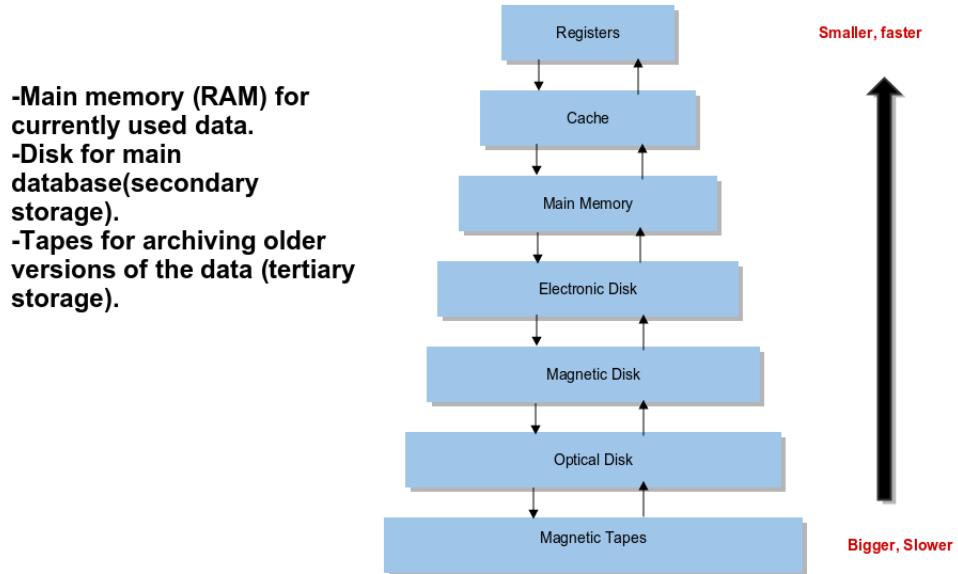


Figura 4.2: Almacenamiento y Organización de datos [9]

PostgreSQL posee un gestor de almacenamiento (Storage Manager) que esta compuesto por varios módulos que proveen administración de las transacciones y acceso a los objetos de la base de datos.

Los módulos se programaron bajo tres lineamientos bien claros:

- Manejar transacciones sin necesidad de escribir código complejo de recuperación en caso de caídas.
- Mantener versiones históricas de los datos bajo el concepto de “graba una vez, lee muchas veces” .
- Tomar las ventajas que ofrece el hardware especializado como multiprocesadores, memoria no volátil.

4.1.1.1. Los índices

Cada tipo de búsqueda tienen un tipo de índice adecuado para trabajarla, básicamente un índice es un “archivo” donde está parte de un dato y la estructura de una tabla con “search key” de búsqueda.

4.1.2. Como se procesa un consulta(Query)

A continuación se presenta la Figura 4.3 donde se muestra como se procesa una consulta.

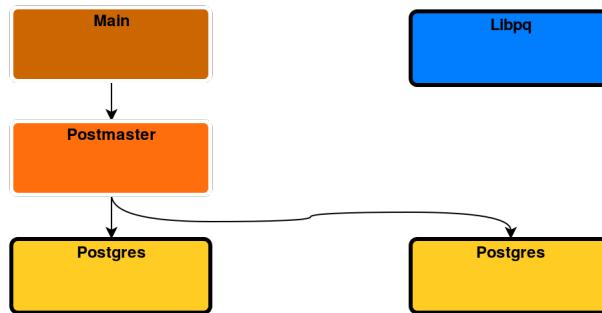


Figura 4.3: Como se procesa un query [9]

Un identificador de reglas de que lo escrito sea sintácticamente entendible, que los dígitos y los números sean reconocibles, luego se descompone “palabra” a “palabra” el query para pasar a la estructura que le corresponde según el query, en este caso a la estructura de un SELECT, esto se ve así:

```
1 simple_select: SELECT opt_distinct target_list
2     into_clause from_clause where_clause
3     group_clause having_clause
4     [
5         SelectStmt *n = makeNode(SelectStmt);
6         n->distinctClause = $2;
7         n->targetList = $3;
8         n->isTemp = (bool)((Value *) lfirst($4))->val.ival;
9         n->into = (char *) lnext($4);
10        n->fromClause = $5;
11        n->whereClause = $6;
12        n->groupClause = $7;
13        n->havingClause = $8;
14        $$ = (Node)n;
15    ]
```

```
1 typedef struct SelectStmt
2 {
3     NodeTag type;
4     List *distinctClause;
5
6     char *into;
7     bool isTemp;
8     List *targetList;
9     List *fromClause;
10    Node *whereClause;
11    List *groupClause;
12    Node *havingClause;
13
14    List *sortClause;
```

```

15 char *portalname;
16 bool binary;
17 Node *limitOffset;
18 Node *limitCount;
19 List *forUpdate;
20
21 SetOperartion op;
22 bool all;
23 struct SelectStmt *larg;
24 struct SelectStmt *rarg;
25 } SelectStmt;

```

4.1.2.1. Métodos para relacionar tablas

Nested loop join Consume mas recursos de memoria pero la cantidad de búsquedas a realizar es menor.

Merge join Requiere que los datos este ordenada para ubicar las relaciones, el costo esta justamente en mantener la data ordenada.

Hash join Aparentemente sería la forma mas rápida de acceder a un dato gracias a la creación de tablas indexadas, pero limitada a una búsqueda de igualdad. Las combinaciones hash requerirá una combinación de igualdad predicado (un predicado comparación de los valores de una tabla con los valores de la otra tabla utilizando el operador igual ‘=’), las combinaciones hash también puede ser evaluado por un predicado anti-join (un predicado seleccionar valores de una tabla cuando no hay valores relacionados se encuentran en el otro). Dependiendo de los tamaños de las tablas, diferentes algoritmos se pueden aplicar.

El “Executor” toma el plan de ejecución que el “planer” le entrega e inicia el procesamiento, ejecuta un “plan tree”. Este “plan tree” tiene varios nodos de ejecución que se van ejecutando uno a uno y de cada uno de ellos se obtiene un set de datos(tuplas). Los nodos tienen subnodos y otros a su vez otros subnodos, tantos como sea necesario.

4.2. Obtener la estructura de una base de datos

En este caso se requiere obtener una estructura de información con detalles por cada columna de una tabla semejante a esta:

```

column_name => cedula,
datatype => integer,
key => PRI,
is_nullable => NO,
max_length => 8,

```

```
column_default =>
```

Se puede observar el detalle de un atributo de una tabla por lo tanto es posible realizar para cada una donde:

datatype: es un tipo de dato interno de postgresql.

key: UNI = **unique**, el campo es un índice único.

key: PRI = **primary key**, el campo es un índice primario,

key: FK = **foreign key**, el campo es un índice de una llave foránea.

max_length: Si el campo es integer, muestra la precisión del entero (2,4,8), si es un varchar, la longitud en caracteres (ej. 75).

column_default: muestra el tipo de valor por defecto si la tabla es serial, se puede ver la llamada al nextval de la secuencia: nexval(personas_cliente_id_seq'::regclass). Lo que permite determinar que campo de nuestra tabla es serial (auto-incremental).

4.2.1. Obtener el detalle de una tabla

Para entender cada tabla del **pg_catalog**, debe ser interrogada con el **oid** de la tabla, que se puede obtener de **pg_class**. Los campos y sus atributos se puede sacar de la tabla **pg_attribute**. El tipo de datos se puede sacar de la tabla **pg_type** los constraints de la tabla se obtiene de la tabla **pg_constraint** y el valor por defecto sacar de la tabla **pg_attrdef**.

4.2.1.1. Usando OIDs

Los OIDs (Identificador de Objeto) los utiliza PostgreSQL internamente como clave primaria para varias tablas del sistema. También se puede utilizar como identificadores únicos en nuestras tablas, aunque es una opción desaconsejable que, a partir de la versión 8.1 viene deshabilitada por defecto en PostgreSQL, por lo tanto no se agregan a las tablas creadas por el usuario, a menos que se especifique **WITH OIDS** cuando se crea la tabla, o la variable de configuración **default_with_oids** está habilitada. A continuación se presenta el código SQL 4.1 haciendo uso de OIDs.

```
1 SELECT pgca.attname AS column_name ,
2       t.typname AS data_type ,
3 CASE
4     WHEN cc.contype='p' THEN 'PRI'
5     WHEN cc.contype='u' THEN 'UNI'
6     WHEN cc.contype='f' THEN 'FK'
7     ELSE ''
8 END AS key ,
9 CASE
10    WHEN pgca.attnotnull=false THEN 'YES'
11 ELSE      'NO'
12 END AS is_nullable ,
```

```

13 CASE
14     WHEN pgca.attlen=-1 THEN(pgca.atttypmod-4)
15     ELSE pgca.attlen
16 END as max_length,
17 d.adsrc as column_default
18 FROM pg_catalog.pg_attribute pgca
19     LEFT JOIN pg_catalog.pg_type t ON
20         t.oid=pgca.atttypid
21     LEFT JOIN pg_catalog.pg_class c ON
22         c.oid=pgca.attrelid
23     LEFT JOIN pg_catalog.pg_constraint cc ON
24         cc.conrelid=c.oid AND
25         cc.conkey[1]=pgca.attnum
26     LEFT JOIN pg_catalog.pg_attrdef d ON
27         d.adrelid=c.oid AND
28         pgca.attnum=d.adnum
29 WHERE c.relname='nombretabla' AND
30     pgca.attnum>0 AND
31     t.oid = pgca.atttypid.

```

Código 4.1: Query para obtener detalle tabla con OIDs

nombretabla representa el nombre de la tabla al que se quiere interrogar para obtener su metadato. Si el modelo de la Figura 2.1 se lleva al gestor de base de datos en este caso PostgreSQL y se hace uso del código de Figura 4.1 se obtiene algo similar como se observa en la Figura 4.4 donde:

- *column_name* muestra el nombre de la columna de la tabla.
- *data_type* indica que tipo almacena esta columna.
- *key* indica si es una llave, sea primaria, foránea o sea único.
- *is_nullable* indica si este campo puede ser nulo.
- *max_length* indica el tamaño de memoria de información máxima.
- *column_default* este campo indica si es autoincremental en PostgreSQL (serial, bigserial y smallserial) que normalmente se suele usar en llaves primarias las cuales no son obligatorias insertar ya que el DBMS se encarga de realizarlo por nosotros.

	Data Output	Explain	Messages	History		
	column_name name	data_type name	key text	is_nullable text	max_length integer	column_default text
1	cod_detalleventa	int4	PRI	NO	4	nextval('detalleventa_cod_detalleventa_seq'::regclass)
2	cod_cliente	int4	FK	NO	4	
3	cantidad_detalleventa	varchar		NO	-5	
4	cod_vendedor	int4	FK	NO	4	
5	cod_producto	int4	FK	NO	4	
6	fecha_detalleventa	date		NO	4	

Figura 4.4: Detalle tabla OIDs

Son algunos campos disponibles en el metadato, de las muchas que se puede obtener y depende de lo que sea necesario, las listadas son básicas sobre la información detallada de una determina tabla de una base de datos.

4.2.1.2. Usando Information schema

El código 4.1 para obtener los metadatos de una tabla no es la única forma existe otras formas de realizar. PostgreSQL a partir de la versión 8.0 introdujo el INFORMATION_SCHEMA. Las vistas definidas en el INFORMATION_SCHEMA le dan acceso a la información almacenada en las tablas del sistema PostgreSQL. El INFORMATION_SCHEMA se define como parte del estandar SQL y encontrarás un INFORMATION_SCHEMA en sistemas de bases de datos más comerciales (y algunos de código abierto). Por ejemplo, para ver una lista de las tablas definidas en la base de datos actual, puede ejecutar el código 4.2:

```

1 SELECT tc.column_name,
2       data_type,
3       character_maximum_length,
4       numeric_precision,
5       is_nullable,
6       tcs.constraint_type,
7       column_default,
8       check_clause
9 FROM information_schema.columns AS tc
10 LEFT OUTER JOIN
11      information_schema.constraint_column_usage AS cc
12      ON tc.table_name = cc.table_name AND
13      tc.column_name = cc.column_name
14 LEFT OUTER JOIN
15      information_schema.table_constraints AS tcs
16      ON tcs.constraint_name = cc.constraint_name
17 LEFT OUTER JOIN
18      information_schema.check_constraints AS cccs
19      ON cccs.constraint_name = tcs.constraint_name
20 WHERE tc.table_name = 'NOMBRE DE LA TABLA' AND
21   tc.table_schema = 'public' AND
22   (tcs.constraint_type='PRIMARY KEY' OR
23    tcs.constraint_type='CHECK' OR
24    tcs.constraint_type ISNULL)
25 ORDER BY ordinal_position;
```

Código 4.2: Query para detalle obtener el detalle de una tabla information scheme

Si se hace una comparación con la anterior se puede ver las diferencias, en esta ya no se hace uso de los OIDS las tablas que son parte del metadato, las tablas necesarias para esta consulta son:

```

1 SELECT * FROM information_schema.columns
2 SELECT * FROM information_schema.constraint_column_usage
3 SELECT * FROM information_schema.table_constraints
```

```
4 SELECT * FROM information_schema.check_constraints
```

Si se ejecuta el código 4.2 se obtiene como resultado lo que se ve en la Figura 4.5.

Data Output Explain Messages History								
	column_name character varying	data_type character varying	character_max_length integer	numeric_precision integer	is_nullable character varying(3)	constraint_type character varying	column_default character varying	check_clause character varying
1	cod_detalleventa	integer		32	NO	PRIMARY KEY	nextval('detallevent')	
2	cod_producto	integer		32	NO	PRIMARY KEY		
3	cod_vendedor	integer		32	NO	PRIMARY KEY		
4	cod_cliente	integer		32	NO	PRIMARY KEY		
5	cantidad_detalleventa	character varying			NO			
6	fecha_detalleventa	date			NO			

Figura 4.5: Information schema

Donde se puede hacer una descripción detallada:

- *column_name* muestra el nombre de la columna de la tabla.
- *data_type* indica el tipo de datos que está permitido insertar, si se compara con los resultados de la figura 4.4 aquí devuelve el tipo de dato (integer en lugar de int4) como se definió en el modelo de la Figura 2.1 Modelo ER.
- *constraint_type* indica si es una llave sea primaria, foránea o sea única.
- *is_nullable* indica si este campo puede ser nulo.
- *character_max_length* indica el tamaño de memoria de información máxima.
- *column_default* en este campo indica si es autoincrementable en PostgreSQL(serial, bigserial y smallserial) que normalmente se suele usar en llaves primarias las cuales no son obligatorias insertar ya que el DBMS se encarga de realizarlo.
- *check_clause* en esta columna se muestra si el campo tiene restricciones al realizar la inserción de datos por ejemplo: se puede decidir si se quiere registrar edades entre 18 a 60 años.
- *numeric precision* indica el tamaño del tipo, aparte de pertenecer a un cierto tipo en los DBMS suelen tener tipos de datos más precisos.

4.2.2. Obteniendo las relaciones entre las tablas

La estructura de las relaciones entre tablas en una base de datos es el resultado de su modelo ER, donde las relaciones en PostgreSQL son representadas por *constraints* en un sistema gestor de base de datos. De alguna manera se necesita saber mediante un script las relaciones entre tablas de una base de datos, que tablas se relacionan con otra y exactamente qué columnas están involucradas, al decir las columnas involucradas se menciona exactamente

a las columnas de una determinada tabla que son las llaves foráneas y que estas existen en la tabla que es referenciada, si se ejecuta el código 4.3.

```

1 SELECT (SELECT relname
2      FROM pg_catalog.pg_class c
3        LEFT JOIN pg_catalog.pg_namespace n ON
4          n.oid = c.relnamespace
5      WHERE c.oid=r.conrelid) as tablas,
6      conname,
7      pg_catalog.pg_get_constraintdef(oid, true) as ref
8  FROM pg_catalog.pg_constraint r
9 WHERE r.conrelid
10    IN(SELECT c.oid
11      FROM pg_catalog.pg_class c LEFT JOIN
12          pg_catalog.pg_namespace n ON
13          n.oid = c.relnamespace
14      WHERE c.relname !~ 'pg_' AND
15          c.relkind = 'r' AND
16          pg_catalog.pg_table_is_visible(c.oid)) AND
17 r.contype = 'f'

```

Codigo 4.3: Query para obtener el detalle de referencias

Se tiene el resultado como se observa en la Figura 4.6.

Data Output Explain Messages History			
	tablas name	conname name	referencias text
1	detalleventa	producto detalleventa fk	FOREIGN KEY (cod_producto) REFERENCES producto(cod_producto)
2	detalleventa	cliente detalleventa fk	FOREIGN KEY (cod_cliente) REFERENCES cliente(cod_cliente)
3	detalleventa	vendedor detalleventa fk	FOREIGN KEY (cod_vendedor) REFERENCES vendedor(cod_vendedor)
4	vendedor	persona vendedor fk	FOREIGN KEY (cod_persona) REFERENCES persona(cod_persona)
5	producto	categoría producto fk	FOREIGN KEY (cod_categoria) REFERENCES categoria(cod_categoria)
6	cliente	persona cliente fk	FOREIGN KEY (cod_persona) REFERENCES persona(cod_persona) ON UPDATE RESTRICT
7	proveedor	persona proveedor fk	FOREIGN KEY (cod_persona) REFERENCES persona(cod_persona)
8	telefono	persona telefono fk	FOREIGN KEY (cod_persona) REFERENCES persona(cod_persona)
9	compra producto	producto compra producto fk	FOREIGN KEY (cod_producto) REFERENCES producto(cod_producto)
10	compra producto	proveedor compra producto fk	FOREIGN KEY (cod_proveedor) REFERENCES proveedor(cod_proveedor)
11	email	persona email fk	FOREIGN KEY (cod_persona) REFERENCES persona(cod_persona)

Figura 4.6: Referencias entre tablas

A continuación se puede ver un detalle:

- **tablas** Muestra la lista de tablas que hacen referencia, puede encontrarse que el nombre de una tabla llegue a repetirse en mas de una ocasión no es un problema, se hara un detalle después de ver la explicación de la tercera columna.
- **conname** Muestra el CONSTRAINT de la relación, que seria algo como el nombre de la relación entre las tablas involucradas.
- **referencias** En esta columna de la Figura 4.6 trae toda la información necesaria para ser usado. se puede ver la cadena de texto de una de ellas:

```
1 "FOREIGN KEY (cod_producto) REFERENCES producto(cod_producto)"
```

La cadena de texto posee información relevante donde (*cod_producto*) es el campo que hace referencia como indica REFERENCES a la tabla *producto* y al campo (*cod_producto*). Aunque el resultado esta en modo texto existen formas de solucionar para obtener la información separada, una manera es realizar un parseo al texto que las distintos lenguajes de programación ya tienen funciones implementadas para estas tareas.

En la columna *tablas* llegan a repetirse el nombre de una tabla en mas de una vez, esto es debido a que lista por cada relación que llegue a tener una tabla con otras

4.2.3. Obteniendo las tablas independientes

Para obtener las tablas que son independientes de otras es necesario usar el script anterior la cual da como resultado un conjunto solo de las tablas que se relacionan de alguna manera entre ellas y tener otro conjunto de todas las tablas de la base de datos, como se tiene estos dos conjuntos de tablas realizar una operación de resta entre los conjuntos. La lista de todas las tablas menos la lista de las tablas que se relacionan, como resultado se tiene las tablas que son independientes y que llegarían a ser los primeros en ser llenados. El código SQL para obtener esa información es la que se tiene en el código 4.4.

```

1 SELECT tablename
2 FROM pg_tables
3 WHERE schemaname = 'public' AND
4       tablename NOT IN
5       (SELECT (SELECT relname
6                 FROM pg_catalog.pg_class c LEFT JOIN
7                     pg_catalog.pg_namespace n ON
8                         n.oid = c.relnamespace
9                 WHERE
10                     c.oid=r.conrelid) as nombre
11            FROM pg_catalog.pg_constraint r
12            WHERE r.conrelid IN
13            (SELECT c.oid
14                  FROM pg_catalog.pg_class c LEFT JOIN
15                      pg_catalog.pg_namespace n ON
16                          n.oid = c.relnamespace
17                  WHERE c.relname !~ 'pg_' AND
18                      c.relkind = 'r' AND
19                      pg_catalog.pg_table_is_visible(c.oid))
20            AND r.contype = 'f')

```

Código 4.4: Query para obtener tablas independientes

Si se ejecuta esta consulta da como resultado como se observa en la Figura 4.7.

Al analizar la Figura 2.1 Modelo ER se puede observar que las entidades que son independientes que no hacen referencia a otra son las mismas que da como resultado en la Figura 4.7.

Data Output		Explain	Messages	History
	tablename			
1	persona			
2	categoria			

Figura 4.7: Tablas independientes

4.3. Ordenando los metadatos

Si ya se cuenta con la información de los metadatos de una base de datos es necesario por una parte tener claro en el detalle de una tabla, que las llaves foráneas pueden ser un conjunto de columnas que hagan referencia a una determinada tabla.

Al momento de hacer las inserciones de datos se debe tener cuidado con este caso si se inserta valores a columnas que hacen referencia estas deben existir en la tabla referenciada.

Como se puede observar en la Figura 4.8 la entidad *venta* es una composición de *vendedor* y *cliente* y que esta a la vez llega ser maestra de la entidad *detalle*, por lo tanto la entidad *detalle* tiene una llave compuesta.

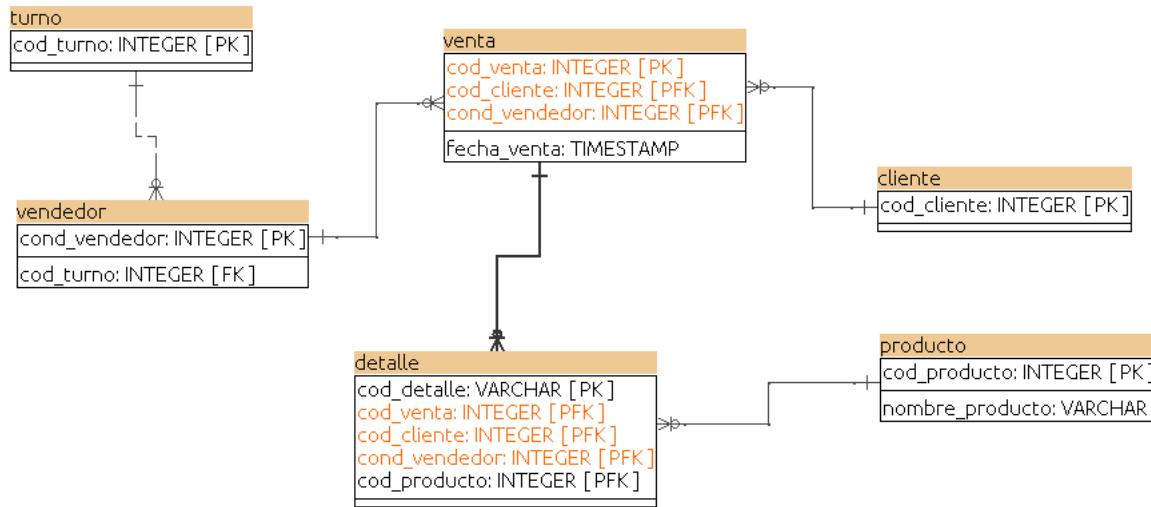


Figura 4.8: Modelo ER compuesto

Si el modelo se lleva a un sistema gestor de base de datos en este caso PostgreSQL y se llena con datos de prueba como se ve en la Figura 4.9 se tiene llaves compuestas.

	cod_venta [PK] serial	cod_cliente [PK] integer	cond_vendedor [PK] integer	fecha_venta timestamp without time zone
1	1	1	1	2014-09-09 00:00:00
2	2	1	2	2014-01-25 00:00:00
3	3	2	1	2014-11-11 00:00:00
4	4	2	3	2014-12-22 00:00:00
5	5	3	3	2014-11-21 00:00:00
*				

Figura 4.9: Tabla venta

Al realizar la inserción en *detalle* se debe tener cuidado en no cometer el error de la ultima inserción que se quiere hacer en la Figura 4.11, esta llega a ser incorrecta debido a que no existe una fila de (*cod_venta,cod_cliente,cod_vendedor*) en la tabla *venta* con valores de (2,1,1), llegando a no cumplir la integridad referencial además son datos inconsistentes.

La ultima inserción de la Figura 4.10 es correcta porque en la Figura 4.9 se puede encontrar una fila también conocida como tupla que (*cod_venta,cod_cliente,cod_vendedor*) tengan los valores (2,1,2) cumpliendo así la integridad referencial y consistencia de datos.

	cod_detalle [PK] character	cod_venta [PK] integer	cod_cliente [PK] integer	cond_vendedor [PK] integer	cod_producto [PK] integer
1	1	1	1	1	1
2	2	1	1	1	2
3	3	1	1	1	3
4	4	2	1	2	1
*					

Figura 4.10: Tabla detalle inserción correcta

	cod_detalle [PK] character	cod_venta [PK] integer	cod_cliente [PK] integer	cond_vendedor [PK] integer	cod_producto [PK] integer
1	1	1	1	1	1
2	2	1	1	1	2
3	3	1	1	1	3
4	4	2	1	2	1
*		2	1	1	1

Figura 4.11: Tabla detalle inserción incorrecta

La otra parte es ordenar la lista de tablas según la prioridad que deben ser llenados, esta claro que las tablas independientes son los primeros de ahí en adelante aun no esta claro, por lo tanto es necesario desarrollar mecanismos para obtener una lista de tablas según el orden en que se requiere.

4.3.1. Ordenando las tablas

El código 4.4 da como resultado las tablas que deben ser llenados primero, las siguientes son algunas de la lista que entrega el código 4.3, que prácticamente están desordenados.

Si el modelo entidad relación de la Figura 4.8 se lleva a PostgreSQL y se aplica el código 4.3 se obtiene el resultado de la Figura 4.12

Output pane		
Data Output Explain Messages History		
tablas name	connname name	referencias text
1 detalle	venta detalle fk	FOREIGN KEY (cod_cliente, cod_venta, cond_vendedor) REFERENCES venta(cod_cliente, cod_venta, cond_vendedor)
2 detalle	producto detalle fk	FOREIGN KEY (cod_producto) REFERENCES producto(cod_producto)
3 vendedor	fecha factura vendedor fk	FOREIGN KEY (cod_turno) REFERENCES turno(cod_turno)
4 venta	cliente factura fk	FOREIGN KEY (cod_cliente) REFERENCES cliente(cod_cliente) ON UPDATE CASCADE ON DELETE CASCADE
5 venta	vendedor factura fk	FOREIGN KEY (cond_vendedor) REFERENCES vendedor(cond_vendedor)

Figura 4.12: Detalle de relaciones entre tablas

En la Figura 4.12 se tiene la primera columna el nombre de la tabla que hace referencia a una o mas tablas, pero en la tercera columna no se tiene separada el nombre de la tabla que es referenciada por que es necesario separarlos de alguna manera, el formato de texto que da como resultado para la primera linea perteneciente a *detalle*:

```
1 "FOREIGN KEY (cod_cliente, cod_venta, cond_vendedor) REFERENCES venta(  
cod_cliente, cod_venta, cond_vendedor)"
```

Lo que hace es separar en cinco partes la cadena de texto:

1. FOREIGN KEY
2. cod_cliente, cod_venta, cond_vendedor
3. REFERENCES venta
4. cod_cliente, cod_venta, cond_vendedor
5. ...

Las manera de implementar puede variar de acuerdo a la tecnología sea java, php, python y otros, sin embargo muchas de estas tecnologías ya vienen implementadas estas funciones para hacer estas tareas, por ejemplo en java se puede llevar la cadena de texto a un arreglo de textos simplemente se define delimitadores que este caso serian “(,)” obteniendo así un resultado similar a la que se listo. A partir de esa lista se escoge el de la posición 2 iniciando a contar desde 0 que llega ser *REFERENCES venta* en este caso en particular, esta cadena se vuelve a separar en dos:

1. REFERENCES

2. venta

Como se tiene el nombre de la tabla se retorna este valor como la tabla que es referenciada para *detalle*. Se realiza esto para cada una de la lista de la Figura 4.12.

Cabe aclarar que en la segunda columna en la segunda separación de texto que se hizo puede que en algunos casos sobre todo cuando se hace uso de scheme(esquemas) en PostgreSQL venga concatenada el nombre del scheme antes del nombre de la tabla concatenada con un punto seguido con el nombre de la tabla.

public.venta.

Lo cual no debería causar problemas por el simple hecho de que ayuda a identificar en que scheme se encuentra la tabla. Como resultado de las operaciones que se hizo se obtiene el resultado que se observa en el cuadro 4.1.

Cuadro 4.1: tablas que referencian a otra

tablas que refieren	tablas referenciadas
detalle	venta
detalle	producto
vendedor	turno
venta	cliente
venta	vendedor

En la primera columna se tiene las tablas que hacen referencia y en la segunda columna las que son referenciados. Para hacer uso del algoritmo de ordenación 2.2 del Capítulo 2, ya se cuenta con datos hasta el paso dos por lo tanto pasar al paso tres donde se realiza la búsqueda para todas aquellas entidades que le hacen referencia a los que son independientes, que para el modelo ER de la Figura 4.8 llegaría a ser la figura 4.13.

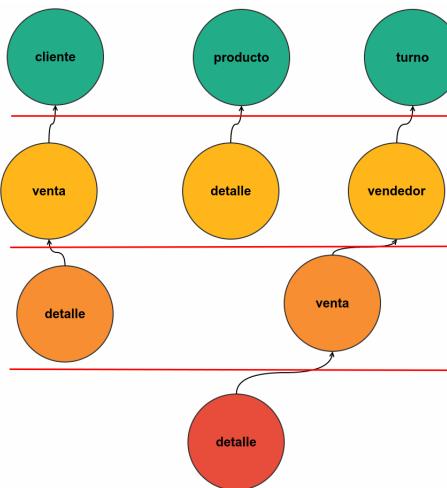


Figura 4.13: Secuencia

Donde se puede observar que el nombre de una tabla se llega a repetir en varios lugares para lo cual se aplica el 2.4 de Capítulo 2 , como resultado se llega a tener la figura 4.14 la cual tiene el orden correcto.

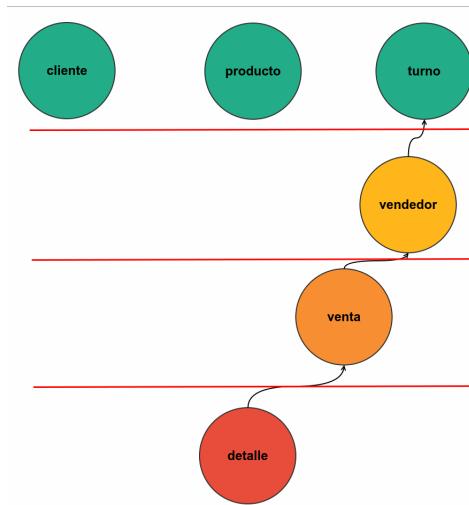


Figura 4.14: Orden correcto

4.3.2. Uniendo foreign keys

Una vez que ya se tiene la información detallada por cada una de las tablas que hacen referencia y de las cuales las columnas que estén involucradas llegan a ser llaves extranjeras, que si bien pueden ser parte de la llave primaria compuesta de la tabla o simplemente ser una llave foránea, al momento de insertar puede llegar a dar lo mismo, por lo tanto no se va a centrar en ese detalle.

La información que provee el query de la Figura 4.2 da una detallada información sobre una tabla en particular pero para lo que se necesita generar datos de prueba es importante tener mecanismos que eviten cometer errores en las llaves que no son propias de una tabla. En el query de la Figura 4.12 se tiene una información valiosa en la tercera columna:

```

1 "detalle";"venta_detalle_fk";"FOREIGN KEY (cod_cliente, cod_venta,
    cond_vendedor) REFERENCES venta(cod_cliente, cod_venta, cond_vendedor)"
2 "detalle";"producto_detalle_fk";"FOREIGN KEY (cod_producto) REFERENCES
    producto(cod_producto)"
```

La tabla *detalle* con las columnas (*cod_cliente*, *cod_venta*, *cond_vendedor*) hace referencia a la tabla *venta* a las columnas (*cod_cliente*, *cod_venta*, *cond_vendedor*), además la columna (*cod_producto*) hace referencia a la tabla *producto* a la columna (*cod_producto*), son las llaves que no son propias de la tabla *detalle*.

En el detalle que da como información la Figura 4.15, las llaves que no son propias no están agrupadas de acuerdo a la tabla al que referencia como sucede en la Figura 4.12 donde si lo agrupa pero solo provee esos campos que son llaves que apuntan a otra tabla, a diferencia de la Figura 4.15 si da la información de todas las columnas.

Output pane

	column_name	data_type	character_m	numeric_precision	is_nullable	constraint_type	column_default	check_clause
	character varying	character varying	integer	integer	character varying(3)	character varying	character varying	character varying
1	cod_detalle	character varying			NO	PRIMARY KEY	nextval('detalle_coc')	
2	cod_venta	integer		32	NO	PRIMARY KEY		
3	cod_cliente	integer			32	NO	PRIMARY KEY	
4	cod_vendedor	integer			32	NO	PRIMARY KEY	
5	cod_producto	integer			32	NO	PRIMARY KEY	

Figura 4.15: Detalle de la tabla *detalle*

El cuadro 4.2 es resultado al que se quiere llegar uniendo la información de la Figura 4.15 y la Figura 4.15.

Cuadro 4.2: tabla de referencias para la tabla detalle

nombre columna	tipo de dato	es primaria?	serial	tabla a la que referencia	columnas a la que referencia
cod_detalle	INTEGER	PRIMARY KEY	si	NULL	null
cod_producto		FORANEA		producto	cod_producto
cod_venta, cod_cliente, cod_vendedor		FORANEA		venta	cod_venta, cod_cliente, cod_vendedor

Si se recuerda el query de la Figura 4.3 da como resultado una lista de tablas que referencian, en este caso solo es necesario consultar específicamente para una tabla, para lo cual se hace alguna modificación a la consulta del código 4.3 quedando como resultado el código 4.5.

```

1 SELECT (SELECT relname
2         FROM pg_catalog.pg_class c
3         LEFT JOIN
4             pg_catalog.pg_namespace n ON
5                 n.oid=c.relnamespace
6         WHERE
7             c.oid=r.conrelid) AS nombre,
8         conname,
9         pg_catalog.pg_get_constraintdef(oid,true) AS ref
10    FROM
11        pg_catalog.pg_constraint r
12   WHERE r.conrelid IN
13       (SELECT
14           c.oid
15           FROM pg_catalog.pg_class c
16           LEFT JOIN
17               pg_catalog.pg_namespace n ON
18                   n.oid = c.relnamespace
19           WHERE
20               c.relname !~ 'pg_' AND
21               c.relkind='r' AND
22               pg_catalog.pg_table_is_visible(c.oid)) AND
23       r.contype = 'f' AND
24       (SELECT relname

```

```

25      FROM pg_catalog.pg_class c
26      LEFT JOIN
27          pg_catalog.pg_namespace n ON
28      n.oid = c.relnamespace
29 WHERE
30      c.oid=r.conrelid)='nombreTabla';";

```

Codigo 4.5: Query para detalle referencias para una tabla

A diferencia de la consulta del código 4.3 en esta consulta se especifica exactamente para que tabla se quiere saber a quienes referencia agregando al final las siguientes líneas de código SQL 4.6.

```

1      AND
2      (SELECT relname
3      FROM pg_catalog.pg_class c
4      LEFT JOIN
5          pg_catalog.pg_namespace n ON
6      n.oid = c.relnamespace
7 WHERE
8      c.oid=r.conrelid)='nombreTabla';";

```

Codigo 4.6: Parte que determina para una tabla

Con la adición del código extra se logra que filtre solo para la tabla que se requiere, bastara con solo cambiar el *nombreTabla*. El resultado de esta consulta daría solo los registros donde una determinada tabla hace referencia, ver para el caso de la tabla *detalle* en la Figura 4.16.

The screenshot shows the PostgreSQL output pane with the 'Data Output' tab selected. The results are displayed in a table with the following columns: 'nombre name', 'conname name', and 'referencias text'. There are two rows of data:

	nombre name	conname name	referencias text
1	detalle	producto detalle fk	FOREIGN KEY (cod producto) REFERENCES producto(cod producto)
2	detalle	venta detalle fk	FOREIGN KEY (cod cliente, cod venta, cond vendedor) REFERENCES venta(cod cliente, cod venta, cond vendedor)

Figura 4.16: Detalle de referencias de la tabla detalle

Los resultados no llegan a ser tan buenos debido a que devuelve en texto todo los datos de las columnas que referencian y la tabla que es referenciada con sus respectivos columnas. Se hará uso de las mismas técnicas que se aplicó al momento de realizar el ordenamiento de las tablas según el orden que deben ser llenados que al final se necesita tener un resultado similar al cuadro 4.3.

Cuadro 4.3: tabla referencias formateada

tabla que referencia	columnas que referencian	tabla referenciada	columnas referenciadas
detalle	cod_producto	producto	cod_producto
detalle	cod_cliente, cod_venta, cond_vendedor	venta	cod_cliente, cod_venta, cond_vendedor

De la Figura 4.16 se toma la columna 3 y la fila 2 como ejemplo escoger la fila 2 por razones didácticas, es donde se encuentra la información en modo texto:

```
1 "FOREIGN KEY (cod_cliente, cod_venta, cond_vendedor) REFERENCES venta(
    cod_cliente, cod_venta, cond_vendedor)"
```

La cadena de texto se separa en 5 partes:

Lista en 5 partes

1. FOREIGN KEY
2. cod_cliente, cod_venta, cond_vendedor
3. REFERENCES venta
4. REFERENCES venta
5. ...

Para obtener esta lista separada se lleva a un arreglo la cadena de texto teniendo como separadores a “,”. La mayoría de los lenguajes de programación proveen funciones que realizan esta tarea de llevar una cadena de texto a un arreglo con solo indicar los caracteres separadores.

En la mayoría de los lenguajes de programación el conteo de las posiciones se inicia en 0 por tanto es necesario basarse en esa regla, del arreglo solo se necesita el de la posición 1 es donde se encuentra las columnas que hacen referencia en conjunto a la tabla de la posición 2 de arreglo sin antes aclarar que este elemento de la posición debe ser separado en dos partes:

1. REFERENCES
2. venta.

De esta lista solo es útil el de la posición 1 es donde se encuentra en nombre de la tabla al que se hace referencia.

Si se vuelve a la lista separada en 5 partes el otro elemento útil es de la posición 3, es donde se encontró las columnas referenciadas.

Se realiza este procedimiento por cada relación que haga la tabla obteniendo así un resultado similar a la tabla del cuadro 4.3. Con los resultados obtenidos de la Figura 4.15 y

el de la Figura tabla formateada del cuadro 4.3 realizar la union de estos dos resultados para tener una tabla como se ve en el cuadro 4.2.

Para obtener un resultado del cuadro 4.2 es necesario agregar campos al resultado que provee la Figura 4.15, agregar cuatro campos adicionales:

- *es_foranea* En esta columna se puede agregar si es foránea o no para luego ser evaluado como tal.
- *referencian* En esta columna agregar los nombres de las columnas que hacen referencia a otra tabla.
- *tabla* En esta columna agregar el nombre de la tabla al que referencia.
- *referenciados* En esta columna agregar los nombres de las columnas que son referenciados.

Quedando como resultado el cuadro 4.4.

Cuadro 4.4: tabla con columnas aumentadas para la tabla *detalle*

column_name	data_type	constraint_type	es_foranea	columnas referencian	tabla referenciada	columnas referenciadas
cod_detalle	character varying	PRIMARY_KEY				
cod_venta	INTEGER	PRIMARY_KEY					
cod_cliente	INTEGER	PRIMARY_KEY					
cod_vendedor	INTEGER	PRIMARY_KEY					
cod_producto	INTEGER	PRIMARY_KEY					

Si se recuerda la cadena de texto:

```

1 "FOREIGN KEY (cod_cliente, cod_venta, cond_vendedor) REFERENCES venta(
    cod_cliente, cod_venta, cond_vendedor)
2 FOREIGN KEY (cod_producto) REFERENCES producto(cod_producto)"

```

Se lleva a un arreglo de 5 elementos, el elemento de la posición 1 que llega a ser lo siguiente:

```
1 "cod_cliente, cod_venta, cond_vendedor"
```

Es donde se encuentra las columnas que hacen referencia por lo tanto a esta cadena es necesario también llevar a un arreglo lineal donde el carácter separador llega a ser el “,” quedando como resultado como se observa en el cuadro 4.5.

Cuadro 4.5: columnas de la tabla detalle que referencian a venta

nombre columna	cod_cliente	cod_venta	cond_vendedor
posición	0	1	2

Cuadro 4.6: columna de la tabla detalle que referencia a producto

nombre columna	cod_producto
posición	0

Los datos del cuadro 4.5 y 4.6 son las que hacen referencia a otra tabla para unir las llaves foráneas y llegar a un resultado como en el cuadro 4.2 para lo cual se realiza el siguiente procedimiento.

1. Realizar la unión en un arreglo único los elementos del cuadro 4.5 y 4.6 llamemosle *referencian* y se declara dos variables denominemosle *pos* y *posTabla* declarada con valor inicial de 0 para controlar la posición del nuevo arreglo creado.
2. Obtener el elemento de la la posición *pos* y comparar con el elemento de la posición *posTabla* del cuadro de 4.4 y comparar si son iguales.
3. Si llegan a ser iguales es porque este campo del cuadro 4.4 es un campo que hace referencia a otra entidad por lo tanto agregar el valor de *true* en su columna *es_foranea* e incrementar el valor de pos en una unidad y volver al paso 2 asignar un valor de 0 a la variable *posTabla*, de lo contrario pasar al siguiente paso.
4. Es caso de que no sean iguales esta claro que este atributo no hace referencia a ninguna otra tabla y agregar con un valor de *false* y volver al paso 2 e incrementar el valor de la variable *posTabla*.

Llegado a un resultado como en el siguiente cuadro:

Cuadro 4.7: tabla de muestra de atributos foráneas	
nombre columna	cod_producto
posición	0

En el cuadro 4.7 ya se tiene claro que atributos que no son propias y que dependen de la existencia de registros en la tabla a la que referencia, así como esta no es como se quiere el siguiente procedimiento a realizar es eliminar estos atributos y reemplazarlos por los que se tiene en el cuadro 4.3, para realizar este reemplazo se necesita tener una tabla con los mismo atributos (*tabla_name , data_type, check_clause ... ,referencian,tabla,referenciados*). Eliminando los atribustos que tengan el valor de *true* en la columna *es_foranea* se llega a tener como resultado como el siguiente cuadro

Cuadro 4.8: tabla sin los atributos foráneas	
nombre columna	cod_producto
posición	0

Para obtener la otra tabla con solo de llaves foráneas se agrega los campos que llega a ser similar a la tabla del cuadro 4.8 realizar las siguientes operaciones:

1. Crear un arreglo llamemosle *clon* con las mismas dimensiones que del cuadro 4.4 y declarar una variable llamemosle *indice* que hara el control de la pocisión

2. Obtener la fila de la posición *indice* del cuadro 4.7 y verificar el valor de la columna *es_foranea* en caso que sea *true* pasar a 3 y si no hacer un salto al 4.
3. A esta fila no lo hacer la copia en *clon* porque este atributo no es propia de la tabla e incrementar en una unidad a *índice*.
4. Realizar la copia en *clon* e incrementar en una unidad a *índice*.
5. Si no hay mas elementos que comparar pasar al siguiente de lo contrario volver a 2.
6. Al realizar los pasos anteriores el resultado obtenido será todas los atributos que son propias de la tabla, al lo cual se debe completar con las restantes que son dependientes de otras tablas pero en una forma diferente, para lo cual tomar el de la posición 1 2 y 3 del arreglo separado en 5 partes que como resultado final se tiene en el cuadro 4.8.

Capítulo 5

Crear proyecto de configuración

Cuando se realiza el llenado de datos de prueba sobre una base de datos, hay un inicio y un final donde no siempre se inicia y acaba sin alguna interrupción, por muchas razones fallas eléctricas, cansancio entre otras, para lo cual es importante que toda la información obtenida en el capítulo anterior sea persistente y que sea posible tener esa información sin volver a ejecutar los algoritmos además sin la necesidad de volver a conectar a la base de datos. Para lo cual se almacena en archivos de texto plano con algún formato, entre los formatos mas conocidos se tiene a XML (Extensible Markup Language o Lenguaje de Marcas Extensibles) y JSON (JavaScript Object Notation - Notación de Objetos de JavaScript).

5.1. JSON (JavaScript Object Notation - Notación de Objetos de JavaScript)

Es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos. JSON est constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

En JSON, se presentan de estas formas:

Un objeto es un conjunto desordenado de pares nombre/valor como se ve en la Figura 5.1. Un objeto comienza con { (llave de apertura) y termina con } (llave de cierre). Cada nombre es seguido por : (dos puntos) y los pares nombre/valor estn separados por , (coma).

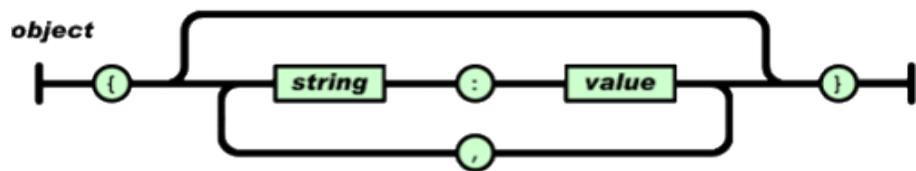


Figura 5.1: Object JSON

Un arreglo es una colección de valores. Un arreglo comienza con [(corchete izquierdo) y termina con] (corchete derecho) como se ve en la Figura 5.2. Los valores se separan por , (coma).

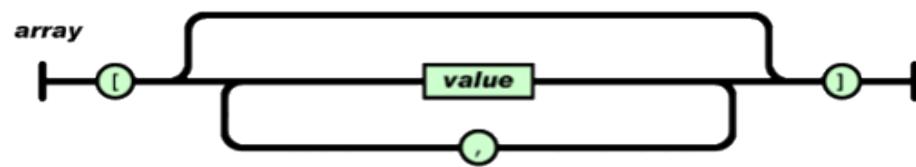


Figura 5.2: Array JSON

Un valor puede ser una cadena de caracteres con comillas dobles, o un número, o true o false o null, o un objeto o un arreglo como se observa en la Figura 5.3. Estas estructuras pueden anidar

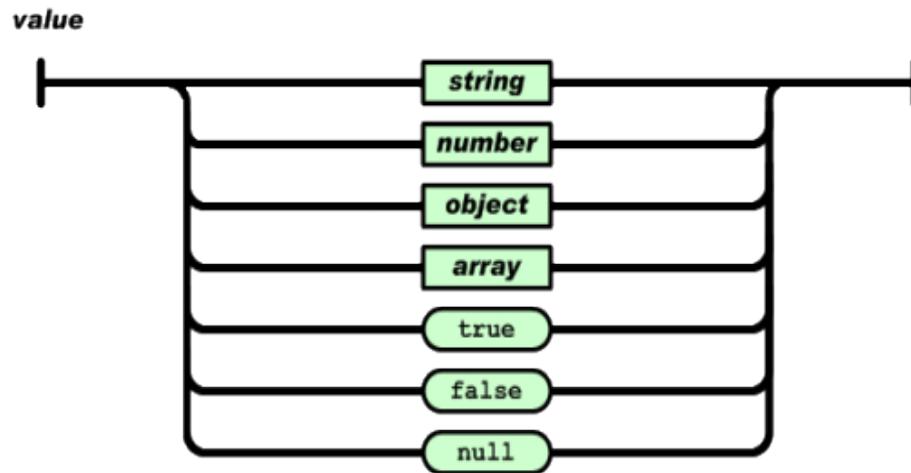


Figura 5.3: Value JSON

Una cadena de caracteres es una colección de cero o más caracteres Unicode, encerrados entre comillas dobles, usando barras divisorias invertidas como escape. Un carácter está representado por una cadena de caracteres de un único carácter ver Figura 5.4. Una cadena de caracteres es parecida a una cadena de caracteres C o Java.

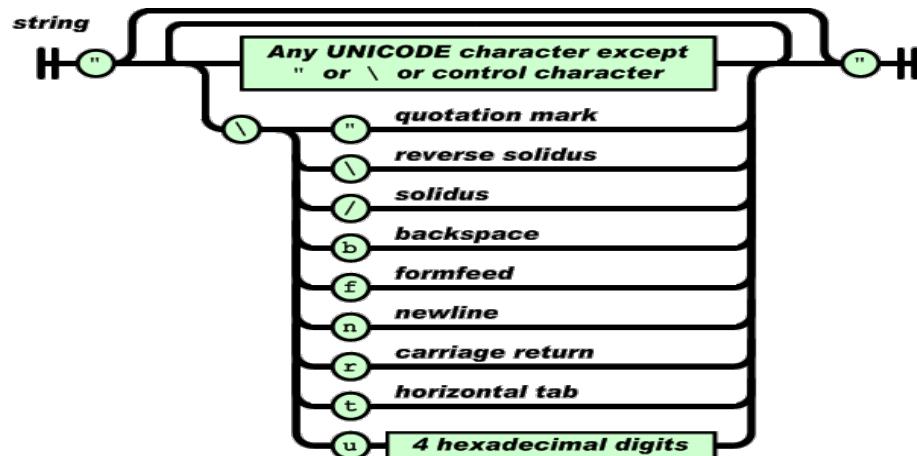


Figura 5.4: String JSON

Un número es similar a un número C o Java, excepto que no se usan los formatos octales y hexadecimales ver la Figura 5.5.

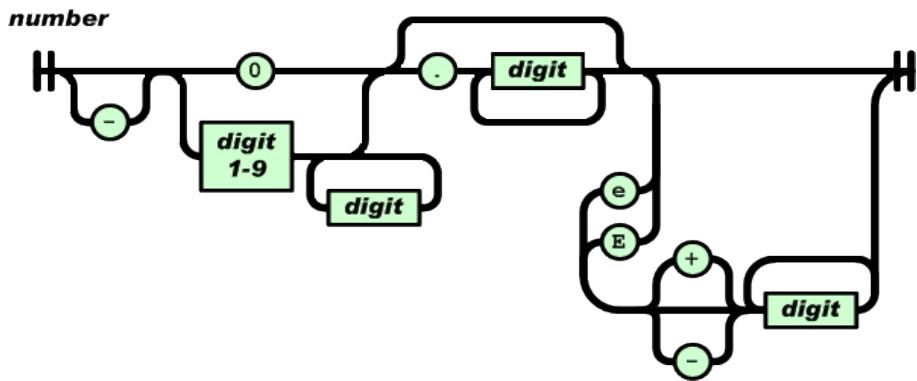


Figura 5.5: Number JSON

Los espacios en blanco pueden insertarse entre cualquier par de símbolos.

5.2. Persistencia de la información de metadatos

La información obtenida en el anterior capítulo es necesario que sean persistentes para la reanudación en el proceso de configuración para el objetivo, la información necesaria a persistir son las siguientes:

- Datos de la conexión a la base de datos como ser el nombre de la base de datos, usuario, contraseña, puerto y el host.
- La lista de las tablas de la base de datos elegida según al orden en que estos deben ser llenados que se obtuvo en el capítulo anterior.
- El detalle por cada una de las tablas (nombre de la columna, el tipo de dato, si acepta que sea nulo, si es una llave etc...).

Una alternativa para dar solución a este requisito es almacenar toda esta información en archivos sea en el formato JSON o XML, en este proyecto se optará por Json las razones son las siguientes:

- Soporta dos tipos de estructuras, una de ellas son objetos que contienen una colección de pares llave-valor y el otro tipo se trata de arrays de valores. Esto proporciona una gran sencillez en las estructuras.
- No tiene espacios de nombres, cada objeto es un conjunto de claves independientes de cualquier otro objeto.
- JSON no necesita ser extensible por que es flexible por sí solo. Puede representar cualquier estructura de datos pudiendo añadir nuevos campos con total facilidad.

- Es mucho mas simple que XML, el cual proporciona pesadas tecnologías que le avalan (Scheme, XSLT, XPath).
- Si se compara el tamaño de un archivo JSON con uno XML y que contenga la misma información el primero llega a ser mucho mas pequeño.

5.2.1. Creando la estructura de un proyecto

Cuando sea crea un proyecto java ,php , python u otro, normalmente se tiene un estructura de directorios y archivos, donde ciertos archivos guardan configuraciones sobre recursos que se hacen uso, la version del proyecto entre otros. Para este proyecto se realiza algo similar para lo cual se va tener como base la estructura de la Figura 5.6 de directorios y archivos.

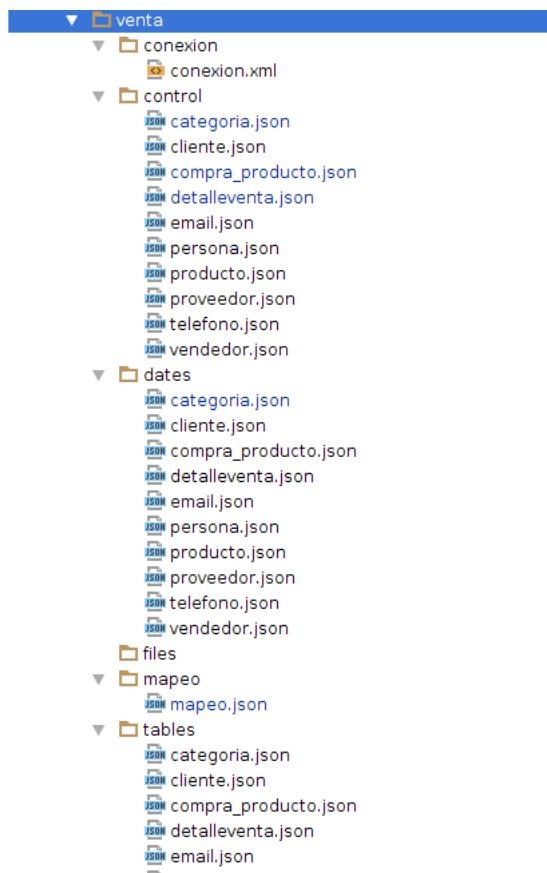


Figura 5.6: Estructura

Donde se guarda la información de los archivos de configuración, veamos en detalle cada directorio y su contenido:

- **conexión** En este directorio se tiene un archivo *conexion.xml* la cual contiene la información de los datos de conexión. Va separada en un directorio por razones de que puede existir un nombre de una tabla igual al archivo por lo que es necesario que no se confunda.

- **control** Si se observa el contenido de los directorios *control,dates,tables* son similares la diferencia esta en el contenido de los archivos.

En este directorio se almacena archivos de control de las columnas por cada tabla y que tienen el mismo nombre que en la base de datos ver el contenido para la tabla *compra_producto* de la Figura 2.1 se obtiene informacion como se ve en el código 5.1.

```

1  [
2    {
3      "column_name": "cod_producto",
4      "is_nullable": "NO",
5      "rellenado": false
6    },
7    {
8      "column_name": "cod_proveedor",
9      "is_nullable": "NO",
10     "rellenado": false
11   },
12   {
13     "column_name": "cod_compra_producto",
14     "is_nullable": "NO",
15     "rellenado": false
16   },
17   {
18     "column_name": "fecha_compra_producto",
19     "is_nullable": "NO",
20     "rellenado": false
21   }
22 ]

```

Codigo 5.1: Ejemplo archivo control

Se encuentra información en formato JSON clave - valor y donde *column_name* indica el nombre de la columna, *is_nullable* indica si este campo puede ser nulo y por ultimo *rellenado* llega a ser la mas importante porque es aquí donde controlar si ya fue configurada esta columna.

- **dates** los archivos de este directorio almacenan información generada por cada columna, a excepción de tipos de datos como bytea o blob, para este tipo de datos es recomendable almacenar el nombre del archivo. El formato del archivo que almacena la información generada es el codigo 5.2:

```

1  [
2    {
3      "nombre_categoria": "bebida",
4      "cod_categoria": "1"
5    },
6    {

```

```

7      "nombre_categoria": "comidrapida",
8      "cod_categoria": 2
9    },
10   {
11     "nombre_categoria": "enlatados",
12     "cod_categoria": 3
13   },
14   {
15     "nombre_categoria": "especial",
16     "cod_categoria": 4
17   },
18   {
19     "nombre_categoria": "ensaladas",
20     "cod_categoria": 5
21   }
22 ]

```

Codigo 5.2: Ejemplo archivo control

Si se observa la Figura 2.1 la tabla *categoria* tiene dos atributos *nombre_categoria* y *cod_categoria*, al ver el contenido del archivo *categoria.json* de directorio *dates* existen varios registros con los valores asignados, la cantidad puede variar dependiendo de la cantidad de datos que se quiere generar.

- **files** En este directorio se almacenan los archivos de tipo bytea y que al momento de hacer la insercion usar por su nombre.
- **mapeo** Solo existe un archivo con un contenido de la lista de las tablas según el orden en que estas deben ser configurados además tiene dos atributos mas *nivel* la cual indica cual es el orden en que le corresponde y por ultimo la *cantidad* si se encuentra un valor igual a cero es porque esta tabla no tiene columna alguna configurada esto deja entender que si se da un valor, es para la tabla en general. Ver para la el caso de la Figura 2.1 se tiene la estructura de control como se ve en el codigo 5.3.

```

1 [
2   {
3     "tablename": "categoria",
4     "nivel": 0,
5     "cantidad": "5"
6   },
7   {
8     "tablename": "persona",
9     "nivel": 0,
10    "cantidad": 0
11  },
12  {
13    "tablename": "producto",
14    "nivel": 1,
15    "cantidad": 0
16  },
17  .

```

```

18 .
19 .
20 {
21   "tablename": "compra_producto",
22   "nivel": 2,
23   "cantidad": 0
24 },
25 {
26   "tablename": "detalleventa",
27   "nivel": 2,
28   "cantidad": 0
29 }
30 ]

```

Codigo 5.3: Ejemplo archivo control

- **tables** En el directorio tables es donde se almacenó la información detallada por cada una de las tablas, además cada archivo representa a una tabla de la base de datos y que llevan el mismo nombre. Se puede ver el contenido del archivo *categoria.json* en el código 5.4 que representa a la tabla *categoria*:

```

1 [
2 {
3   "column_name": "cod_categoria",
4   "data_type": "integer",
5   "character_maximum_length": null,
6   "es_foranea": "false",
7   "referencian": null,
8   "tabla": null,
9   "referenciados": null,
10  "numeric_precision": "32",
11  "is_nullable": "NO",
12  "constraint_type": "PRIMARY KEY",
13  "column_default": "nextval('categoria_cod_categoria_seq'::regclass)",
14  },
15  {
16    "column_name": "nombre_categoria",
17    "data_type": "character varying",
18    "character_maximum_length": null,
19    "es_foranea": "false",
20    "referencian": null,
21    "tabla": null,
22    "referenciados": null,
23    "numeric_precision": null,
24    "is_nullable": "NO",
25    "constraint_type": null,
26    "column_default": null,
27    "check_clause": null
28  }
29 ]
30 ]

```

Codigo 5.4: Ejemplo archivo control

Es una estructura de directorios que no necesariamente se tienen que llamar así, sin embargo el nombre de los archivos es aconsejable que lleven el mismo nombre que en la base de datos para que resulte mas amigable.

5.3. Configuración de columnas

Una vez que se tiene el proyecto creado, la configuración que se hace es por cada columna de la tabla para lo cual es necesario saber que tipo de dato acepta cada columna o si hace referencia a otra tabla. Las columnas de una tabla puede ser de diferente tipo de dato(*integer, varchar, boolean y otros*). Independientemente del tipo de dato de una columna se puede agruparlos tomando en cuenta ciertas características como ser:

- Que el tipo de dato sea texto, fechas hora, direcciones de red al momento de insertar a la base de datos estos son tomados como si fuesen un texto('valor').
- El tipo de dato sea un numero en las que estan (*integer, serial, smallserial, bigserial, bigint ...smallint*) están son insertados como un numero (*valor*).
- Que sea una llave primaria sin importar el tipo de dato están deben ser únicas al momento de generarlos.
- Cuando sea una llave foránea no se debe generar por que estas deben existir en la tabla que hace referencia para lo cual se toma los valores generados en la tabla referenciada.
- El tipo de dato sea bytea es un caso especial que no se trata como una cadena ni como un numero.

5.3.1. Configuración para llaves foráneas

Las llaves foráneas o primarias que no son propias de la tabla no se necesitan generarlos con un algoritmo, lo que se hace es trabajar con los datos generados en la columna de la tabla al que se hace referencia, En este proyecto la técnica empleada para el manejo de estas fue agruparlos todas las que hacen referencia en conjunto a alguna tabla como si fuese una sola columna a continuación se presenta un ejemplo en la Figura 5.7.

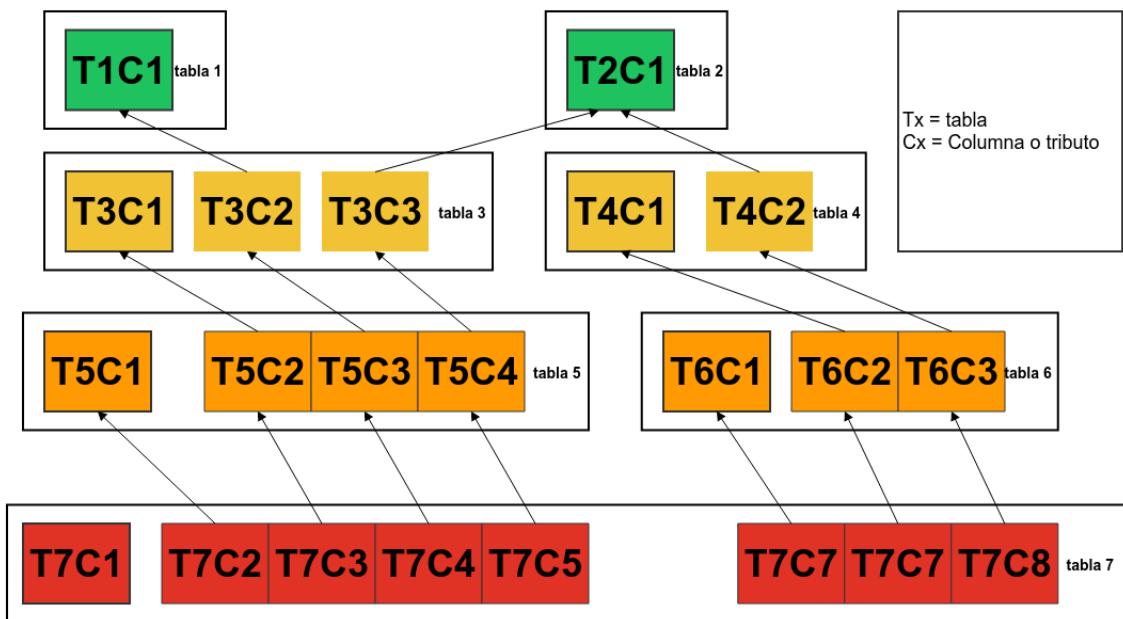


Figura 5.7: Foraneas line

Donde la *tabla1* cuenta con una llave primaria (*T1C1*) al igual que la *tabla2*, y si se baja un nivel mas abajo a la *tabla3* esta hace referencia a *tabla1* y *tabla2* y que las columnas de las tablas referenciadas mandan como llaves primarias, formando asi una llave compuesta para *tabla3*. Por otro lado la *tabla4* hace referencia a *tabla2* y que tambie tiene una llave compuesta. Si se baja a la *tabla5* esta compuesta de 4 columnas de las cuales 3 no son propias, si se observa vienen juntadas como si fuera una columna es lo que se hizo al momento de guardar el detalle de una tabla, En la *tabla7* se hace referencia a la *tabla5* y *tabla6* y las columnas que no son propias son agrupadas de acuerdo a que tabla se haga referencia es el caso de *T7C2, T7C3, T7C4, T7C5* que en conjunto hacen referencia a la *tabla5* y *T7C6, T7C7, T7C8* hacen referencia a *tabla6*.

Al momento de hacer la configuración se llega a tener un problema, de la *tabla5* el campo *T5C2, T5C3, T5C4* no es encontrada si se lo busca como una columna en la *tabla3* como es posible si se tiene esas columnas? Es cierto que existen pero están separadas como se observa en la Figura 5.8.

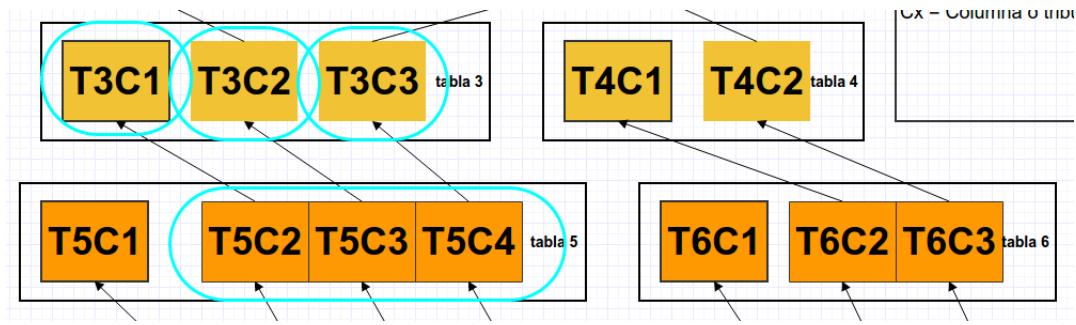


Figura 5.8: Problema llaves foraneas

Se da el mismo problema para la *tabla7* la columna *T7C2, T7C3, T7C4, T7C5* no se encuentra en una solo columna en la *tabla5* pero hay algo interesante que se puede deducir como se observa en la Figura 5.9.

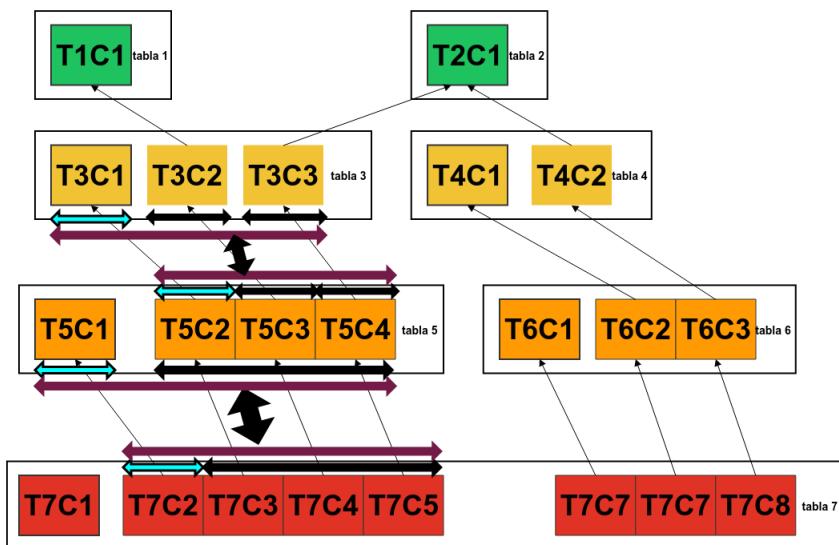


Figura 5.9: Ejemplo foraneas unidas

En la *tabla5* las columnas *T5C2, T5C3, T5C4* en conjunto hacen referencia como una sola a la *tabla3* donde *T5C2* apunta a *T3C1* y que esta es una llave primaria propia de la *tabla3*, en la *tabla7* la columna *T7C2* hace referencia a una propia de la *tabla5*, se puede determinar que el comportamiento de las llaves primarias que no son propias siguen este modelo.

5.3.1.1. Problemas

El problema surge al momento de realizar la validación que por ejemplo cuando por alguna razón se trata de configurar un atributo que hace referencia y que previamente no se configuró los atributos referenciados en la tabla referenciada no se llega a encontrar como se ve en la figura 5.9. Las columnas de la *tabla3* se van encontrar todas al igual que en la *tabla4*, pero en la *tabla5* no se llega a encontrar la columna *T5C2T5C3T5C4* lo mismo sucede

con $T6C2T6C3$. Una vez hecha la validación el siguiente paso es configurar, para lo cual se necesita los datos de la tabla referenciada, al fijarse en la Figura 5.9 se encuentra en el mismo problema de la validación de columnas no encontradas.

5.3.1.2. Soluciones

Una solución obvia al problema de la validación presentado es verificar que todas las columnas de la tabla referenciada se encuentren configuradas para no tener problemas al obtener los datos pero no es tan cierto se puede configurar con solo tener configuradas las columnas referenciadas se puede optar por cualquiera son cuestiones de validación.

En cuanto al problema de la configuración una vez pasada la anterior la solución no llega a ser tan sencilla se necesita aplicar algun mecanismo(os) de obtener los datos, La solución que se da no estrictamente así depende del modelo y del manejo de iniciar con la Figura 5.10.

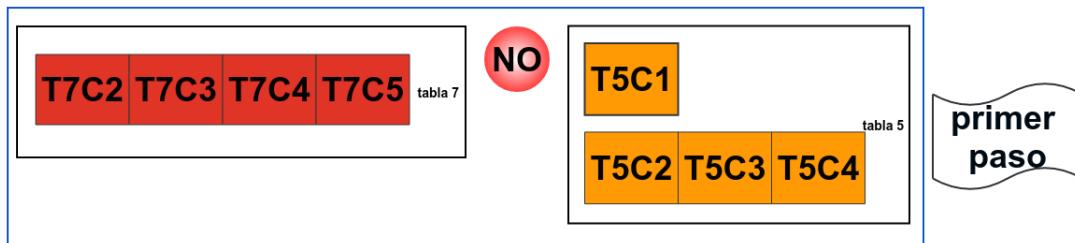


Figura 5.10: primer intento

En el primer intento no se llega a la solución ya que no se encuentra la columna en la *tabla5*, pasar a observar a la Figura 5.11.

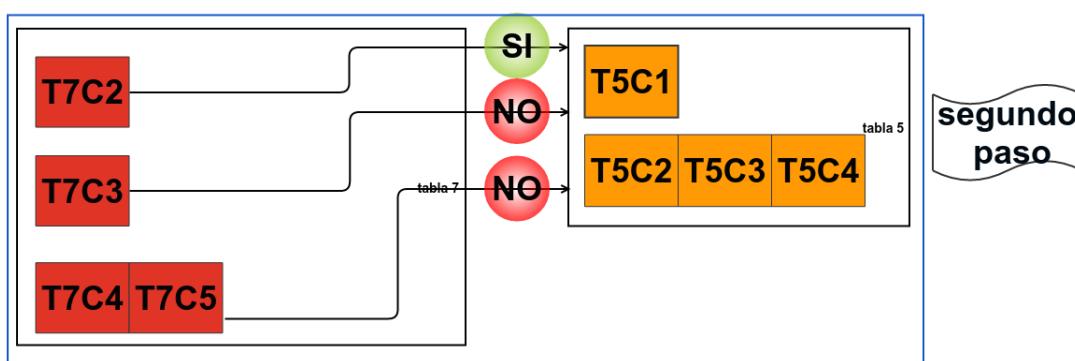


Figura 5.11: segundo intento

Iniciar con el primer elemento buscando en la tabla referenciada en caso de que exista pasar al siguiente elemento caso contrario listar en una lista de los elementos no encontrados, pasar a ver la Figura 5.12.

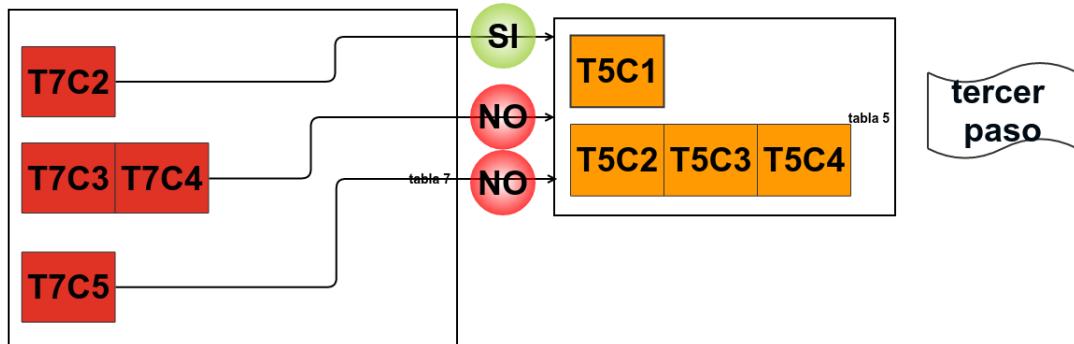


Figura 5.12: tercer intento

Pasar con el siguiente elemento uniendo con las no encontradas y se vuelve a buscar en la tabla referenciada como aun no se encontro pasar al siguiente como se observa en la Figura 5.13.

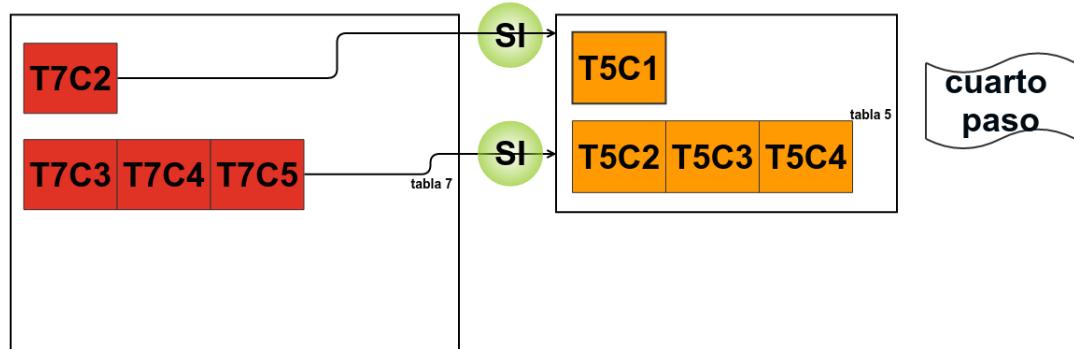


Figura 5.13: cuarto intento

En este paso se vuelve a juntar las no encontradas y el elemento a obtener y buscar en la tabla referenciada y como se encontro y no hay mas elementos que buscar se llega a la solucion.

Capítulo 6

Poblando datos en la base de datos y probando el comportamiento

Una vez que se tiene configurada en forma completa un proyecto, el siguiente proceso a realizar es el poblado de datos y posteriormente realizar algunas consultas de prueba para ver el comportamiento con una población de datos mayor. Para llevar adelante este objetivo es necesario tener la siguiente información.

- Recuperar datos de conexión para la base de datos y realizar la conexión.
- Obtener los datos de archivo *mapeo.json* donde se encuentra el orden de las tablas.
- Por cada tabla crear la estructura SQL de inserción `INSERT INTO tabla (col1,col2...coln)VALUES (val1,val2...valn)`.

Existe casos muy importantes al crear la estructura de datos, la cantidad de columnas que tiene una tabla, el tipo de dato de cada columna, al momento de hacer la inserción es importante tomar en cuenta estos casos , la insercion varia dependiendo el tipo de dato.

6.1. Poblado de datos a la base de datos

Los distintos tipos de datos que provee los DBMS, algunos con una cantidad mayor de tipos de dato y otras con una cantidad mas reducida, es importante analizar como se hara la inserción según al tipo de dato, además se debe tomar encuenta la cantidad de columnas que tiene una tabla.

6.1.1. Tipos de datos tratados como texto

Los tipos de datos tratados como cadenas de texto son:

- Las fechas y horas (DATE, DATETIME, TIME).
- Las cadenas de texto (VARCHAR, CHARACTER VARYIN, TEXT).
- Las direcciones de red (MACADDRESS, INET).

Estos tipos de datos van entre comillas simples(`INTO tabla(col)VALUES('col')`).

6.1.2. Tipos de datos tratados como números

Los tipos de datos son tratado como un numero entero sea decimal flotante son los tipos de dato como:

- Los tipos enteros (INTEGER, BIGINT, SMALLINT, SERIAL, BIGSERIAL).
- Los tipos decimales (FLOAT, DECIMAL MONEY).

Los tipos de datos numéricos a diferencia del anterior no van entre comillas(`INTO tabla(col)VALUES(val)`). Existe otro tipo de dato mas que se puede incorporar es el tipo BOOLEANO si bien no es un número esta no necesita ir dentro las comillas.

6.1.3. Tipo de dato bytea

El tipo de dato bytea es un tipo especial ya que nececita una conversion previa a la inserción, las distintas tecnologias ya se php, java , python , ruby etc... proveen metodos para realizar esta conversión, por lo cual no es un tema de preocupación. Si se recuerda al momento de generar los datos el tipo de dato bytea no lo se lo guarda en el archivo generado solo el nombre del archivo, con la que se forma un codigo SQL de insercion de la siguiente manera (`INTO tabla(col)VALUES(conversionprevia(nombre archivo))`).

6.1.4. Cantidad de columnas por tabla

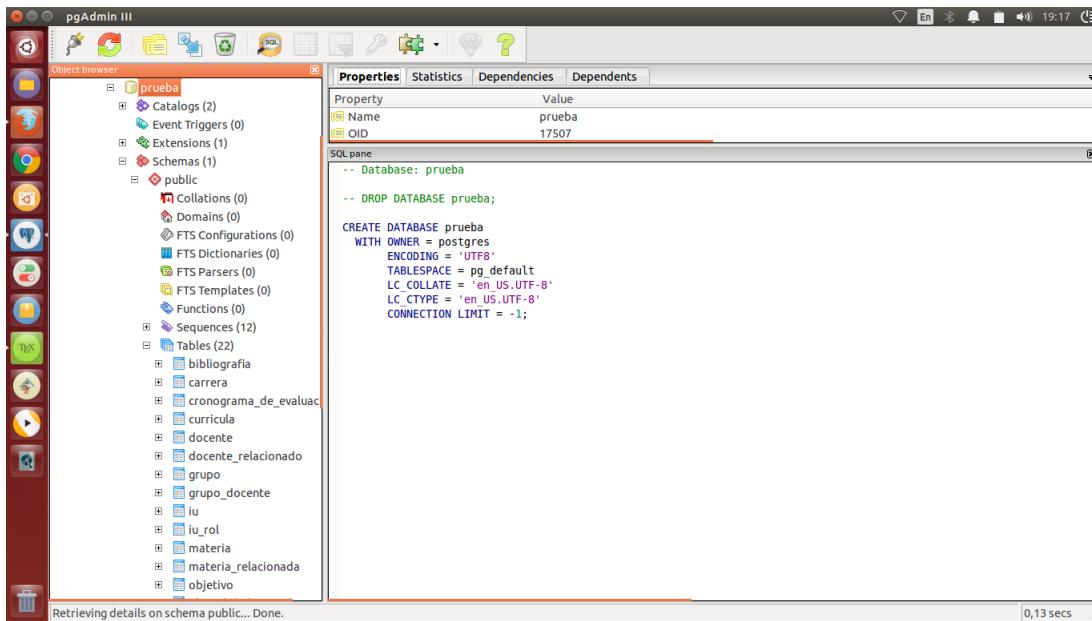
La cantidad de columnas de una tabla es variable, pudiendo tener una o mas columnas para formar la estructura del comando `INSERT` es necesario obtener la información del directorio *tables*. Donde cada tabla es representada por un archivo con el mismo nombre de la tabla y que además esta contiene toda la información detallada de la tabla, entre ellas esta los nombre de las columnas. Con esta informacion se forma la parte necesaria del comando `INSERT INTO tabla(col1, col2,...coln)VALUES()`. En la parte de los valores obtener la información del directorio *dates del proyecto* y referencia [2].

Capítulo 7

Uso del prototipo

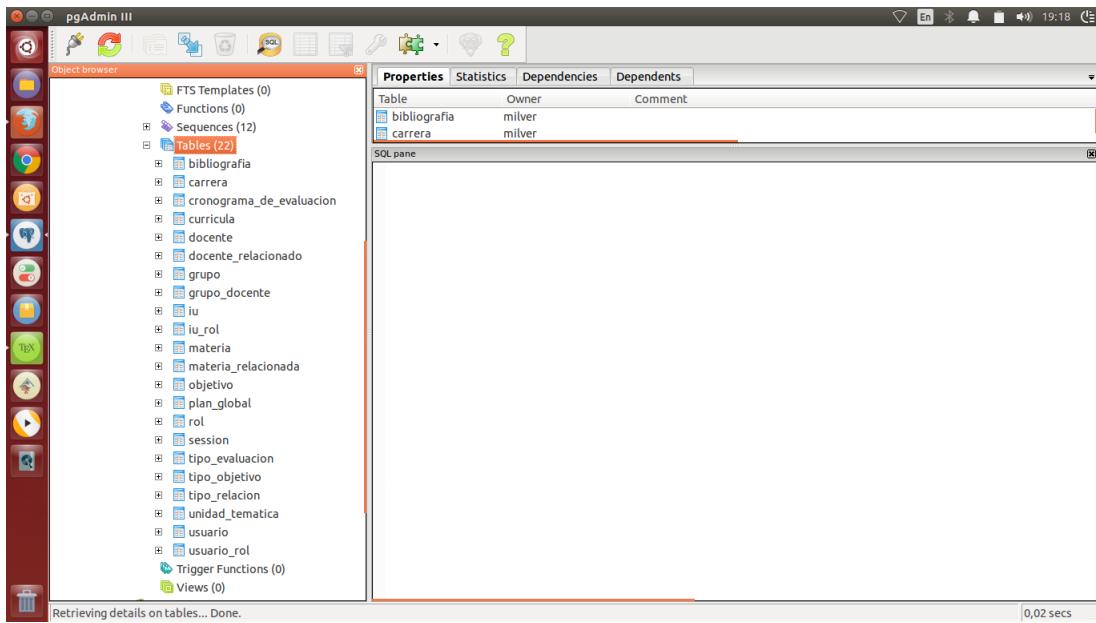
En este capítulo se hace uso del prototipo desarrollado para el poblado de datos de prueba para una base de datos, para lo cual es necesario que se tenga cantidad de tablas. Como ejemplo se toma una base de datos denominada prueba como observa en la Figura 7.1

Figura 7.1: Base de datos prueba



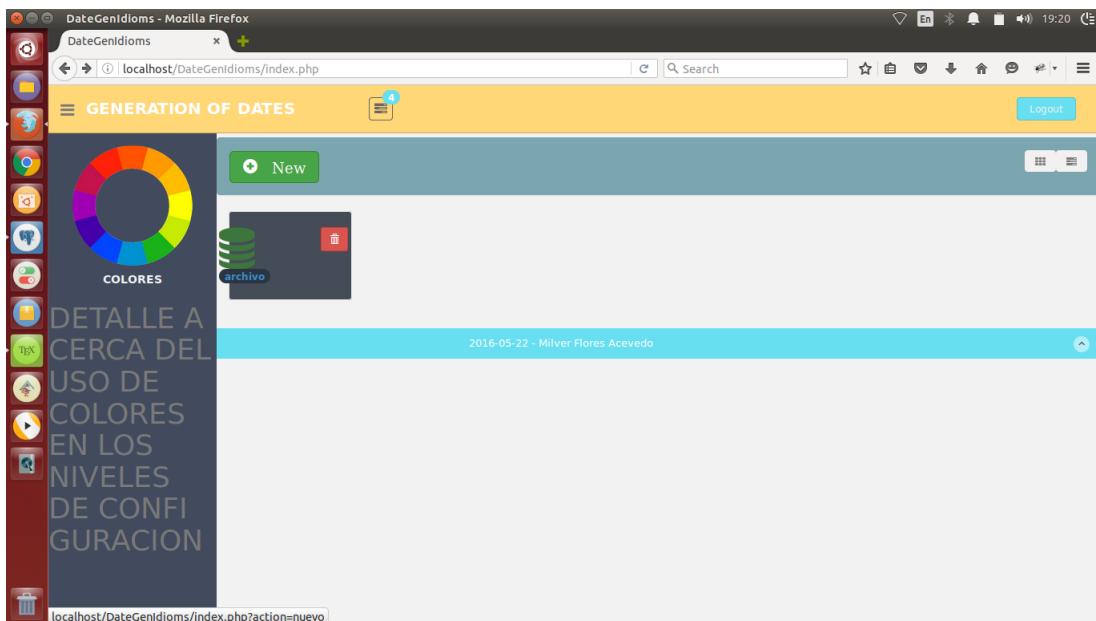
Esta base de datos se tiene veinte y dos tablas como se observa en la Figura 7.2.

Figura 7.2: lista de tablas



En el prototipo se tiene la opción de crear como un proyecto en el botón nuevo y la lista de proyectos como se ve en la Figura 7.3.

Figura 7.3: Lista de proyectos



Al hacer clic llevará a un formulario como se ve en la Figura 7.4

Figura 7.4: Formulario para crear un nuevo proyecto

The screenshot shows a Mozilla Firefox window with the title 'DateGenidioms - Mozilla Firefox'. The address bar displays 'localhost/DateGenidioms/index.php?action=nuevo'. The main content area has a yellow header bar with the text 'GENERATION OF DATES'. Below this, there is a sidebar on the left with a color wheel icon labeled 'COLORES' and text 'DETALLE A CERCA DEL USO DE COLORES EN LOS NIVELES DE CONFIGURACION'. On the right, there is a 'new project' form with the following fields:
Nombre: llenar
SGBD: PostgreSQL
base de datos: llenar
host: localhost
puerto: 5432
usuario: postgres
password: postgres
At the bottom of the form are two buttons: 'test connection' (in green) and 'crear' (in white).

Para crear un proyecto es necesario llenar los datos en el formulario que son:

Nombre este campo es a elección con la restricción que no se puede tener dos proyectos con el mismo nombre.

sgbd el sistema gestor de base de datos que en este trabajo se elige trabajar con PostgreSQL.

base de datos en este campo es necesario el nombre exacto de la base de datos por que sera de la cual se obtendra su estructura.

host la url donde se encuentra alojada la base de datos. En nuestro caso localhost.

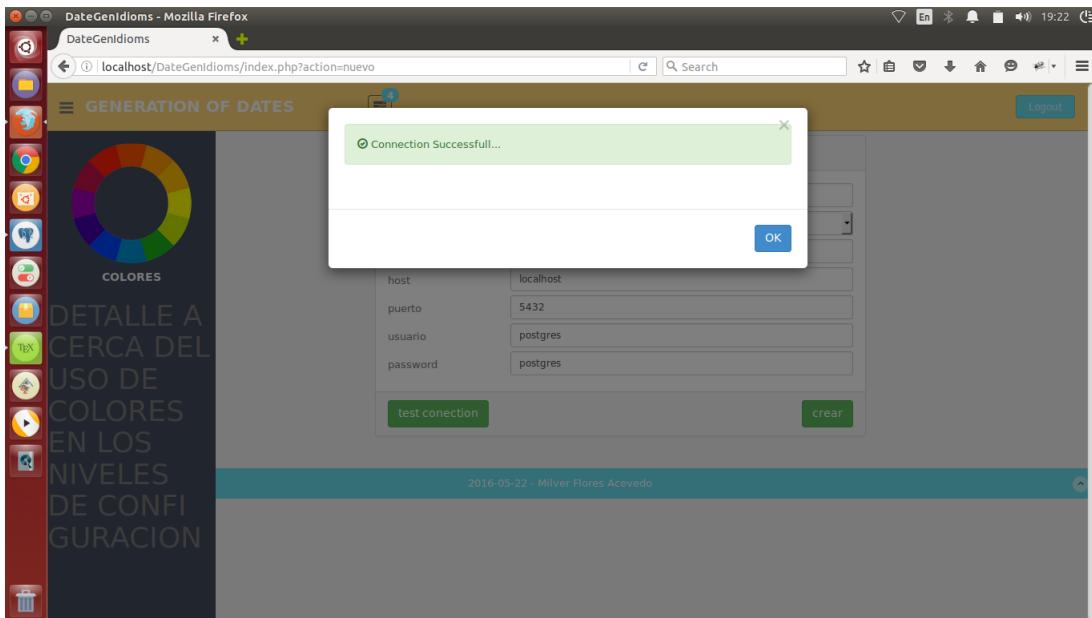
puerto normalmente el puerto que usa PostgreSQL es 5432.

usuario con el usuario que se conectara con privilegio de acceso a metadatos.

password la contraseña del usuario

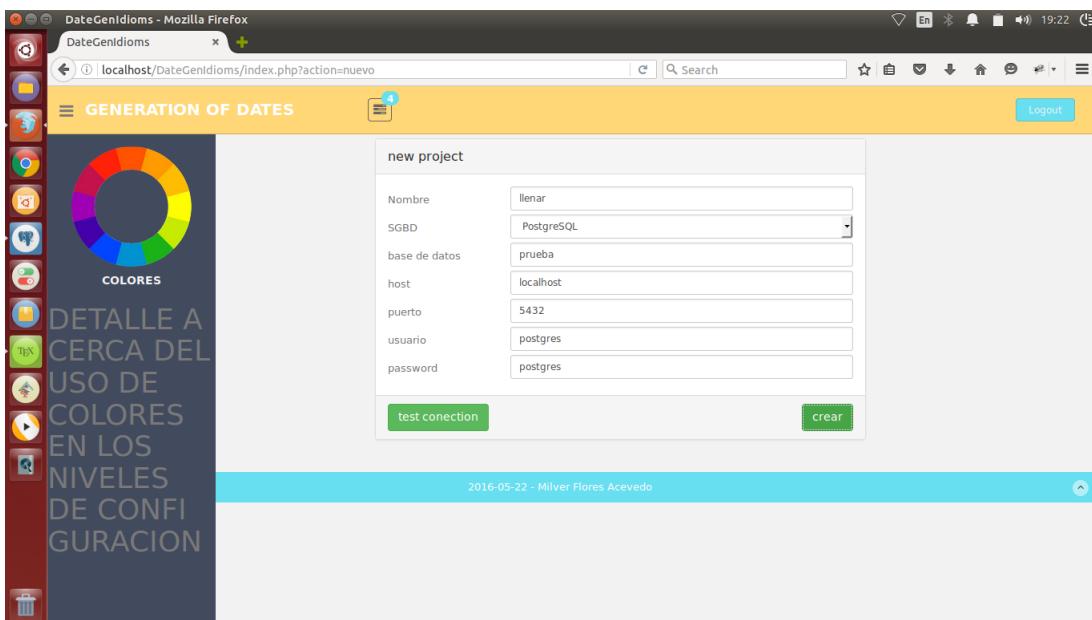
Una vez completada el formulario de creación, lo siguiente es probar la conexión dando clic en el botón de test connection, si es exitosa muestra el mensaje de una conexión exitosa como se observa en la Figura 7.5.

Figura 7.5: Conexion exitosa



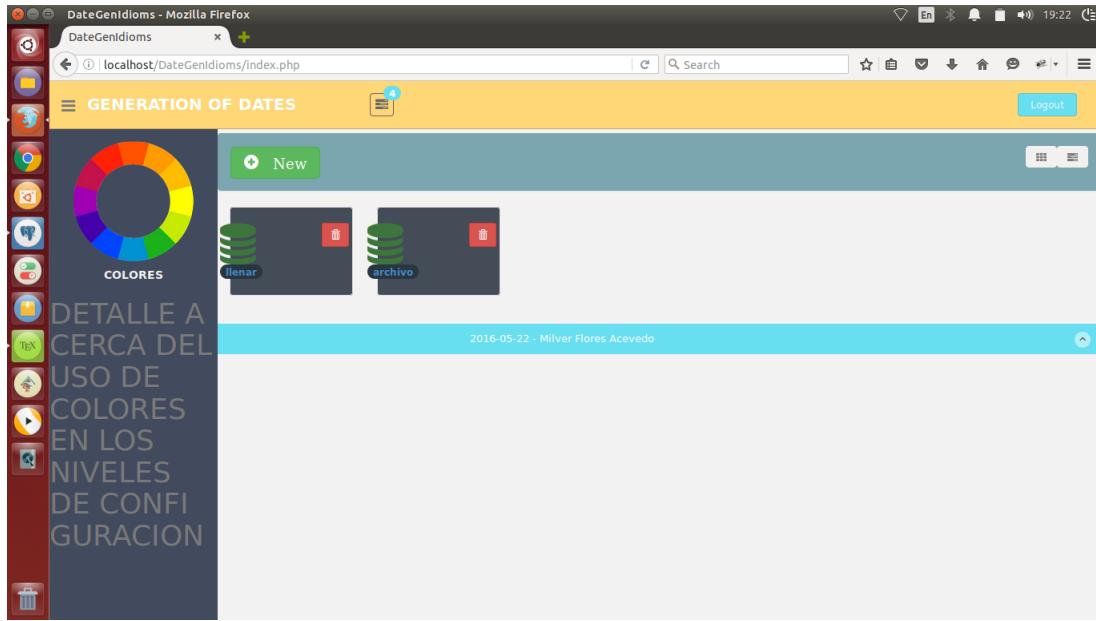
Si los datos del formulario estan de manera correcta dar clic en el boton crear como se observa en la Figura 7.6.

Figura 7.6: Boton crear



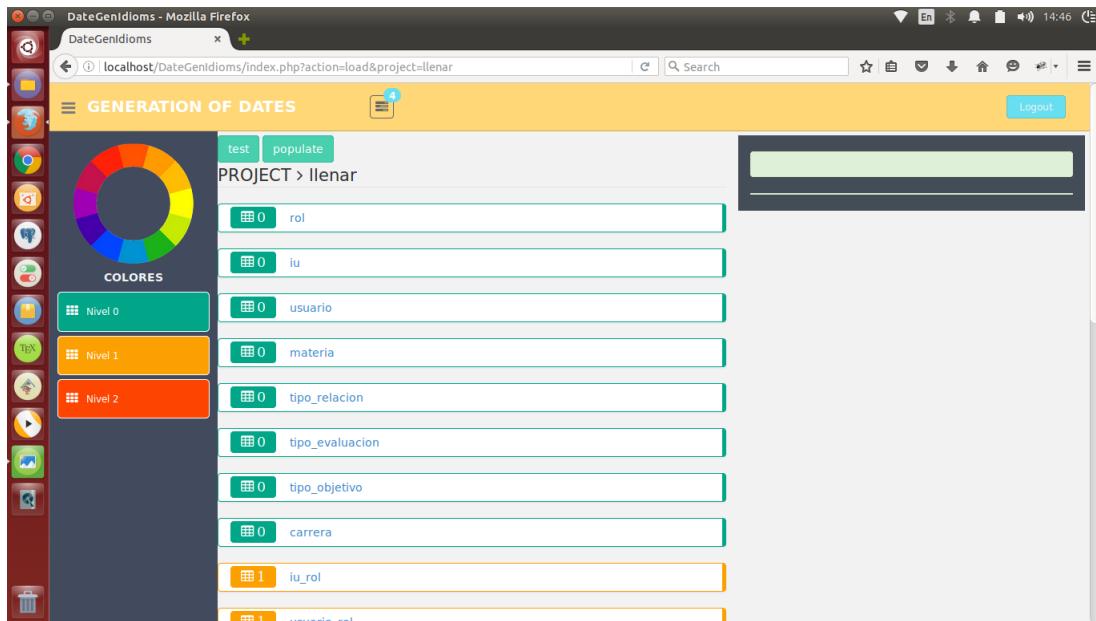
A continuacióin redirigir   a la lista de proyectos en la cual aparecera el proyecto que se creo como se observa en la Figura 7.7.

Figura 7.7: Proyecto creado



A continuacion dar clic en el nombre del proyecto lo cual muestra toda la estructura del proyecto creado como se observa en la Figura 7.8 ahí se tiene listo con el orden en que se debe llenar la base de datos iniciando primero todos los que son de color verde terminando con los rojos.

Figura 7.8: Lista de tablas obtenida



Si se observa en la Figura 7.9 se tiene `id_rol`, `nombre`, donde `id_rol` es la llave primaria por lo cual este tipo de datos se genera en un cierto rango de número.

Figura 7.9: Tipo de dato

Según el tipo de dato el formulario tiene una variación y las opciones que ofrece como queremos generarla, para el caso de `id_rol` al ser una llave primaria se tiene la opción de generar en un cierto rango de numeros como se observa en la Figura 7.10.

Figura 7.10: Llenar campo para la tabla

Una vez llenada el formulario dar clic en guardar.

Figura 7.11: Llenado exitosa

The screenshot shows the 'GENERATION OF DATES' interface in Mozilla Firefox. On the left, there's a sidebar with a color wheel icon labeled 'COLORES' and three levels: 'Nivel 0' (green), 'Nivel 1' (orange), and 'Nivel 2' (red). The main area displays a table structure for the 'rol' table with columns: name, key, null, type, and estado. The 'name' column has entries: 'id_rol' (key, NO, integer) and 'nombre' (YES, character varying). To the right, a modal window titled 'rol' shows a success message: 'SUCCEFULL... cantidad de datos a generar para la tabla rol'. It includes fields for 'ROL.id_rol' (set to 3), 'rango' (set to 1), 'Limite inferior' (set to 1), 'Limite superior' (set to 5), and a 'guardar' button.

Ahora veamos a una tabla que haga referencia `usuario_rol` donde no es necesario llenar formularios como se observa en la Figura 7.12.

Figura 7.12: Tabla `usuario_rol`

This screenshot shows the same application interface. The main table structure for the 'usuario_rol' table is displayed, featuring columns: 'rol_id_rol' (key, NO, FOREIGN), 'usuario_id_usuario' (key, NO, FOREIGN), 'activo' (YES, boolean), 'fecha_inicio' (YES, date), 'fecha_fin' (YES, date), and 'session' (NO, timestamp). To the right, a modal window titled 'usuario_rol' contains a message: 'cantidad de datos a generar para la tabla usuario_rol'. It lists 'USUARIO_ROL.rol.id_rol' and provides details: 'columnas : rol.id_rol', 'REFERENCIA a : rol', and 'a las columnas : id_rol'. A 'guardar' button is also present.

Una vez que se completa la configuración de todas las tablas dar clic en el botón test y como se puede observar en la Figura 7.13

Figura 7.13: Estado de configuracion

The screenshot shows a web-based application titled "GENERATION OF DATES". On the left, there's a sidebar with a color wheel icon labeled "COLORES" and three levels of hierarchy: "Nivel 0", "Nivel 1", and "Nivel 2". The main area has two tabs: "test" and "populate", with "populate" being active. A large button labeled "PROJECT > llenar" is present. To the right of the button is a table titled "en la tabla" with a column "FALTA CONFIGURAR" containing a list of database tables. The table includes: rol, iu, usuario, materia, tipo_relacion, tipo_evaluacion, tipo_objetivo, carrera, iu_rol, usuario_rol, session, docente, plan_global, grupo, curricula, docente_relacionado, and materia_relacionada. Most entries have a green checkmark in the "FALTA CONFIGURAR" column.

Si se tiene toda la configuración ya se puede llenar la base de datos dando clic en el botón populate como se observa en la Figura 7.14, esperar que haga el llenado para posteriormente ver como se ven reflejado en la base de datos.

Figura 7.14: Llenando la base de datos

This screenshot shows the same application interface as Figure 7.13, but the "populate" button is now active. A progress bar at the top indicates the process is in progress. The table on the right shows the status of the population for each table: rol, iu, usuario, materia, tipo_relacion, tipo_evaluacion, tipo_objetivo, carrera, iu_rol, and usuario_rol. The status for these tables is now "listo" (ready), indicated by a green checkmark in the "FALTA CONFIGURAR" column. The "curricula" and "docente_relacionado" tables remain in the "FALTA CONFIGURAR" state.

En la base de datos la tabla `docente` ya se tiene datos de prueba como se observa en la Figura 7.15

Figura 7.15: Tabla docente

1	1	Adria	4452259	pedro.alfaro@casado.com	Ruela Angel, 904, 33º E, 38720, L Escribano
2	2	Malak	4452260	carlos9@yahoo.com	Plaza Villarreal, 71, Atico 4º, 66672, Polanco
3	3	Laura	4452261	manuel.arevalo@gmail.com	Plaza Leo, 882, Bajos, 25273, La Berrios del V
4	4	Beatriz	4452262	rosa.miguelangel@yahoo.com	Paseo Murillo, 7, Atico 4º, 36976, L Barreto
5	5	Adriana	4452263	ismael95@hispanista.com	Camino Lorente, 293, 99º F, 91730, Saavedra de
6	6		4452264	farid@outlook.es	Prado Malasaña, 48, 1º B, 36739, L Jiménez
7	7	Jesus	4452265	eduardo88@hotmail.com	Avenida Alfonso 253, 72, 28040, La Madera
8	8	Adriana	4452266	langule@latinmail.com	Calle Arevalo, 3, 24º 2º, 04660, Ybarra del P
9	9	Hugo	4452267	watencio@arcoquin.org	Travesia Alfonso, 5, 78º A, 71310, O Carranza
10	10	Ines	4452268	sroig@orange.es	Plaza Pol, 2, Bajos, 56466, Villa Márquez del
11	11	Saul	4452269	saul95@laby.com	Camino Jan, 1, 1º B, 07699, Las Villegas del
12	12	Aaron	4452270	julia07@live.com	Ronda Coronado, 1, 76º C, 66635, La Jasse
13	13	Gabriel	4452271	blanca0@roque.org	Calle Benavides, 866, 4º D, 39771, A Gastelur
14	14	Lidia	4452272	orozco.marina@hispanista.com	Passeig Iria, 71, 9º A, 43893, Las Vega
15	15	Nicolas	4452273	cnunez@herrera.com	Paseo Sedillo, 2, Atico 0º, 14400, Los Acevedo
16	16	Oma	4452274	kluna@gmail.com	Ronda Marcos, 70, 23º C, 36393, Sáenz de San F
17	17	Ariadna	4452275	hherniquez@digia.com	Travesia Emma, 9, Entresuelo 8º, 54765, Frias
18	18		4452276	rodriguezduque@gmail.com	Avenida Alfonso, 5, 5º B, 16001, Oviedo de S
19	19	Ane	4452277	juan.rojas@hispanista.com	Camino Lola, 3, 5º F, 46650, San Carvajal
20	20	Aleix	4452278	vela.carla@aleman.net	Rúa Negrete, 75, 47º C, 69080, Fariñas del Pozo
21	21	Pablo	4452279	arellano.jaime@odriguez.com.es	Avenida Alnara, 475, 6º B, 91524, L Aragón de
22	22	Yago	4452280	clara.polanco@gmail.com	Calle Samuel, 59, 9º, 90239, L Holguín
23	23	Adria	4452281	alexia.gonzales@becerra.com	Avenida Delvalle, 87, 73º E, 03134, A Ortiz
24	24	Malak	4452282	nora.arenas@terera.com	Rúa Alcántar, 8, 7º, 55917, Os Alanis
25	25	Antonio	4452283	marti.pardo@gmail.com	Camino Leyre, 06, 63º F, 28021, Fonseca del B
26	26	Aaron	4452284	dfigueroa@anton.com	Plaza Delao, 51, 0º A, 06688, Las Dávila
27	27	Alexandra	4452285	alberto7@hotmail.com	Calle Román, 36, 87º B, 35130, La Franco
28	28		4452286	gutierrez.erika@gmail.com	Rúa Adelante, 5, 73º B, 34020, As Mariñas de Ari
29	29	Leire	4452287	jordi.llorente@yahoo.es	Rúa Bruno, 3, 31º, 04095, La Matanza
30	30	Diego	4452288	lebron.emma@hispanista.com	Travesera Ismael, 5, 0º 7º, 41268, El Pozo
31	31	Mireia	4452289	guillen.curiel@yanex.net	Rúa Olivo, 145, 76º F, 69749, A Redondo
32	32	Guillem	4452290	marina.chavez@mota.org	Rúa Carla, 4, 44º F, 05702, Valladolid
33	33	Iridia	4452291	hun17@hercules.com	Rúa Martínez de la Torre, 1, 30004, Murcia

Pasa lo mismo con la tabla `rol` se puede observar los tres datos introducidos.

Figura 7.16: Tabla rol

1	1	administrador
2	2	docente
3	3	estudiante

Si se tiene la misma base de datos en distintas máquinas se puede usar el SQL generado por el generador.

Figura 7.17: Sql generado

```
INSERT INTO rol (id_rol, nombre) VALUES (1, 'administrador');
INSERT INTO rol (id_rol, nombre) VALUES (2, 'docente');
INSERT INTO rol (id_rol, nombre) VALUES (3, 'estudiante');

INSERT INTO iu (id_iu, activo) VALUES (1, TRUE);
INSERT INTO iu (id_iu, activo) VALUES (2, TRUE);
INSERT INTO iu (id_iu, activo) VALUES (3, TRUE);
INSERT INTO iu (id_iu, activo) VALUES (4, TRUE);
INSERT INTO iu (id_iu, activo) VALUES (5, TRUE);
INSERT INTO iu (id_iu, activo) VALUES (6, TRUE);
INSERT INTO iu (id_iu, activo) VALUES (7, TRUE);
INSERT INTO iu (id_iu, activo) VALUES (8, TRUE);
INSERT INTO iu (id_iu, activo) VALUES (9, TRUE);
INSERT INTO iu (id_iu, activo) VALUES (10, TRUE);

INSERT INTO usuario (id_usuario, password, login, activo) VALUES (1, 'Raul', 'Marco', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (2, 'Vega', 'Pol', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (3, 'Manuel', 'Ainara', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (4, 'Zoe', 'Dario', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (5, 'Jan', 'Cesar', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (6, 'Nahia', 'Ruben', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (7, 'Izan', 'Manuel', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (8, 'Marina', 'Nahia', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (9, 'Rafael', 'Julia', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (10, 'Ariadna', 'Gerard', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (11, 'Claudia', 'Miguel', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (12, 'Diego', 'Ignacio', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (13, 'Paola', 'Elena', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (14, 'Blanca', 'Bruno', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (15, 'Diana', 'Alexandra', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (16, 'Jimena', 'Jimena', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (17, 'Jose Manuel', 'Nicolas', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (18, 'Diana', 'Sofia', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (19, 'Erik', 'Arnau', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (20, 'Angela', 'Yago', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (21, 'Gabriela', 'Pablo', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (22, 'Ivan', 'Martina', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (23, 'Yaiza', 'Malak', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (24, 'Jimena', 'Jan', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (25, 'Daniel', 'Paula', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (26, 'Manuel', 'Nadia', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (27, 'Victor', 'Ismael', TRUE);
INSERT INTO usuario (id_usuario, password, login, activo) VALUES (28, 'Aaron', 'Biel', TRUE);
```

Capítulo 8

Conclusiones

En este trabajo se presenta un diseño e implementación de un conjunto de algoritmos para generar datos de prueba para base de datos y técnicas para obtener el orden en que se debe llenarlo, con el objetivo de automatizar el proceso del llenado de una base de datos, entre las técnicas y algoritmos mas importantes se puede mencionar:

1. La implementación de algoritmos para obtener el orden correcto apartir de una lista de tablas pertenecientes a una base de datos, para lo cual se tomó encuenta diferentes casos que podria darse en el diseño de una base datos. Como resultado se tiene una lista de tablas ordenadas segun el orden correcto en que deben ser llenados, en el resultado se podria tener un lista de conjuntos de tablas donde las tablas que estén en el mismo conjunto tienen el mismo nivel de prioridad.
2. La implemetación de técnicas para el manejo referencial de las llaves primarias y foraneas tomando encuenta que una base de datos esté basada en el concepto E-R Idioms. Obteniendo como resultado el correcto manejo de las llaves foraneas evitando asi la inconsistencia de datos.
3. La implementación de algoritmos para generar datos con caracteres validos que se asemejen mas a datos reales.
4. La implementación de un prototipo como parte de la demostración de los algoritmos y tecnicas propuestos para lo cual se eligio usar como DBMS PostgreSQL, la razon principal es porque soporta llaves compuestas necesarias para una base de datos basados en E-R Idioms.
5. La implementación de técnicas para insertar un tipo de dato `bytea` en una base de datos, para lo cual es necesario la conversión previa, como en el proyecto se puede generar datos en el caso `bytea` es necesario almacenar los archivos en un directorio para luego convertir `pg_escape_bytea(archivo)` e insertar.

Es importante aclarar que al momento de crear un proyecto de configuración el usuario debe tener privilegios de acceso a metadatos del DBMS.

Bibliografía

- [1] Henry F. Korth Abraham Silberschatz. *Fundamentos de base de datos.* McGraw-hill/interamericana de españa, S.A.U, 2002.
- [2] Lorenzo Alberton. Extracting meta information from postgresql. 2006. URL http://www.alberton.info/postgresql_meta_info.html#.VUKMcif_6ko. Online; accessed 30-Abril-2015.
- [3] aulaClic S.L. Ddl, lenguaje de definición de datos. 2010. URL http://www.aulaclic.es/sqlserver/t_8_1.htm. Online; accessed 30-Abril-2015.
- [4] Solvusoft Corporation. What is mydatagen. 2011. URL <http://www.solvusoft.com/en/files/error-virus-removal/exe/windows/ems-database-management-solutions-inc/sql-manager-net-ems-database-management-solutions/mydatagen-exe/>. Online; accessed 30-Abril-2015.
- [5] Datanamic. Datanamic. 2014. URL <http://www.datanamic.com/>. Online; accessed 30-Abril-2015.
- [6] Datanamic. Generating test data with default settings. 2014. URL <http://www.datanamic.com/support/vd-ddg001.html>. Online; accessed 02-Abril-2015.
- [7] Generatedata. Generatedata. 2014. URL <http://www.generatedata.com/#t3>. Online; accessed 30-Abril-2015.
- [8] Addison Wesley Longman. *Introducción a los sistemas de bases de datos.* Design and Production Services, 7 ed^{ón}., 2001.
- [9] Ernesto Quiñones Azcárate. Postgresql como funciona una base de datos por dentro. URL https://wiki.postgresql.org/images/4/43/Postgresql_como_funciona_una_dbms_por_dentro.pdf. Online; accessed 30-Abril-2015.
- [10] Marcelo Flores Soliz. Er idioms. 2006. URL <https://marcelofloress.wordpress.com/er-idioms/>. Online; accessed 30-Abril-2015.

- [11] EMS Database Management Solutions. Ems datagenetor for postgresql. 1999. URL <http://www.sqlmanager.net/en/products/postgresql/datagenerator>. Online; accessed 30-Abril-2015.