**Instituto Tecnológico de Costa Rica**
**Escuela de Ingeniería en Computación**
**Programa de Maestría en Computación**


# *Desarrollo de una notación y herramienta de visualización de datos para mejorar la capacidad de análisis de procedimientos almacenados SQL*

Informe final de tesis para optar por el grado de Magíster Scientiae en Computación, con énfasis en Ciencias de la Computación

Estudiante: Erick Carvajal González

Tutor: Carlos González Alvarado, PhD

Junio 2017

# Tabla de Contenido

# Índice de tablas

# Tabla de gráficos y figuras

## Resumen

La era digital en la que vivimos está enmarcada por un desarrollo vertiginoso de la computación tanto en hardware como en software, que hace que prácticamente en todas las empresas los procesos productivos estén controlados por sistemas computacionales que cada día soportan más operaciones y por ende se vuelven más complejos, aunado a esto, los avances en la capacidad de almacenamiento de las computadoras permite que las empresas puedan almacenar más datos, los cuales son utilizados para la toma de decisiones. Estos datos requieren tecnologías que permitan el procesamiento de información de manera eficiente y eficaz, lo cual ha hecho que las tecnologías de bases de datos tomen auge y con ello también el lenguaje SQL (Lenguaje estructurado de consultas por sus siglas en inglés) de consulta y manipulación de datos. Hoy en día es común encontrar gran parte de la lógica de negocios de las empresas escrita en lenguaje SQL (e.g procedimientos almacenados). El mantenimiento y evolución del código SQL es una tarea compleja y costosa para las empresas porque, entre otros factores requiere tiempo para que una persona pueda entender código ya escrito y que pueda modificarlo sin que esto tenga repercusiones en otros procedimientos almacenados. Es por ello que el objetivo de este trabajo ha sido incrementar la capacidad de análisis de código SQL y a su vez, disminuir el tiempo requerido para entender instrucciones SQL tomando ventaja de la capacidad que tiene el cerebro humano de interpretar información visual en forma paralela, algo que no es posible con el texto el cual es procesado secuencialmente.

Para lograr el objetivo anterior la presente investigación se enfocó en: a) crear una notación visual capaz de representar gráficamente instrucciones de código SQL, b) desarrollar una herramienta (software) que utilice esta notación para visualizar procedimientos almacenados SQL y c) validar que el uso de la notación visual ahorra tiempo con respecto a entender código en texto. La notación visual fue diseñada tomando en cuenta principios que maximizan la efectividad cognitiva (i.e velocidad, facilidad y precisión con que la información puede ser extraída), a saber: claridad semiótica, capacidad de discriminación perceptiva, inmediatez perceptual, expresividad visual y parsimonia gráfica entre otros.

## Abstract

The digital age in which we live is framed by a vertiginous development of both hardware and software, which means that in most of the companies the production processes are controlled by computer systems that every day support more operations and therefore become more complex, at the same time, advances in the storage capacity of computers allows companies to store more data, which are used for decision making. These data require technologies that allow efficient processing of information, which has made database technologies very important technologies for companies and also the SQL (Structured Query Language) language for query and data manipulation. Nowadays it is common to find much of the business logic of companies written in SQL language (e.g. stored procedures). The maintenance and evolution of the SQL code is a complex and expensive task for the companies because, among other factors, it takes time for a person to understand code already written and to be able to modify it without this having repercussions in other stored procedures. Therefore, the objective of this work has been to increase the ability to analyze SQL code and also reduce the time required to understand SQL statements taking advantage of the human brain's ability to interpret visual information in parallel, something that is not possible with the text which is processed sequentially.

To achieve the above objective the present research focused on: a) creating a visual notation capable of graphically representing SQL code instructions, b) developing a tool that uses this notation to visualize SQL stored procedures, and c) validate that the use of visual notation saves time with regards to understanding code in text. Visual notation was designed considering principles that maximize cognitive effectiveness (i.e. speed, ease and precision with which information can be extracted), namely: semiotic clarity, perceptive discrimination, perceptual immediacy, visual expressiveness and graphic parsimony between others.

## APROBACIÓN DE LA TESIS

# 'Desarrollo de una notación y herramienta de visualización de datos para mejorar la capacidad de análisis de procedimientos almacenados SQL"

**TRIBUNAL EXAMINADOR**

Doctor Carlos González Alvarado
Profesor Asesor

Doctor José Enrique Araya Monge
Profesor Lector

Máster Mario Granados Solís
Profesional Externo

Dr. Roberto Cortés Morales
Coordinador/Programa De Maestría

Agosto, 2017

# Agradecimiento

A mi profesor asesor Carlos González por todos los consejos que me brindó durante este proceso de presentación del anteproyecto y la escritura de la tesis.

A todas las personas: profesores, compañeros de la maestría y compañeros del trabajo que me ayudaron para poder finalizar esta tesis.

# Dedicatoria

A Dios por darme todo lo que soy y haberme permitido culminar esta tesis.

A mi esposa Karla y mis hijas Emily y Elisa por estar a mi lado de manera incondicional en todo momento, por sacrificar tiempo de familia sin reclamarlo para que yo pudiera trabajar en la tesis y darme el apoyo que siempre necesité para culminar esta tesis.

A mis padres Gerardo y Miriam por brindarme educación y consejos para convertirme en la persona que soy hoy.

# 1 Introducción

Desde sus orígenes la computación ha estado enfocada entre otras cosas, en simplificar tareas que para el ser humano resultan difíciles de realizar y/o consumen mucho tiempo como por ejemplo el procesamiento de grandes volúmenes de información. Esta capacidad de las computadoras de poder procesar grandes volúmenes de datos ha fomentado el desarrollo de tecnologías específicas para el almacenamiento y manipulación de información (i.e tecnologías de bases de datos) las cuales en su mayoría (ver Gráfico 1) han adoptado el modelo relacional como paradigma de funcionamiento.

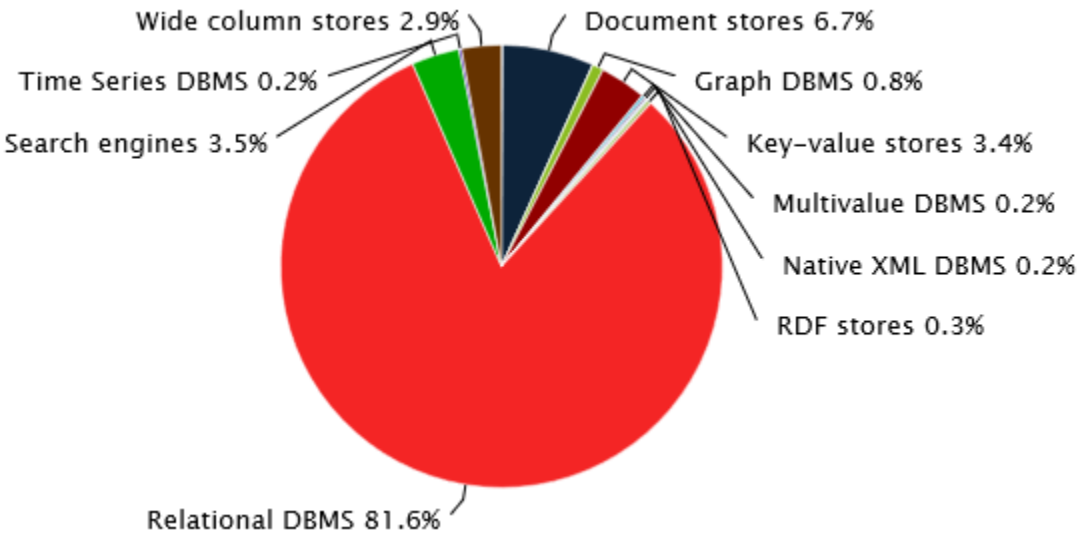



*Gráfico 1: Porcentaje de bases de datos por modelo (DB-Engines.com, 2016)*

El modelo relacional utiliza el SQL como lenguaje principal de consulta, manipulación de datos y programación lo que ha hecho que actualmente en las empresas la lógica del negocio esté en muchos casos contenida en las mismas bases de datos mediante procedimientos almacenados, los cuales están programados en lenguaje SQL.

Como se puede entender, las empresas invierten muchos recursos (e.g programadores, tiempo, dinero) manteniendo y desarrollando procedimientos almacenados y consultas SQL (ver Gráfico 2) lo cual hace deseable encontrar alternativas para simplificar el proceso de comprensión y modificación de código SQL.

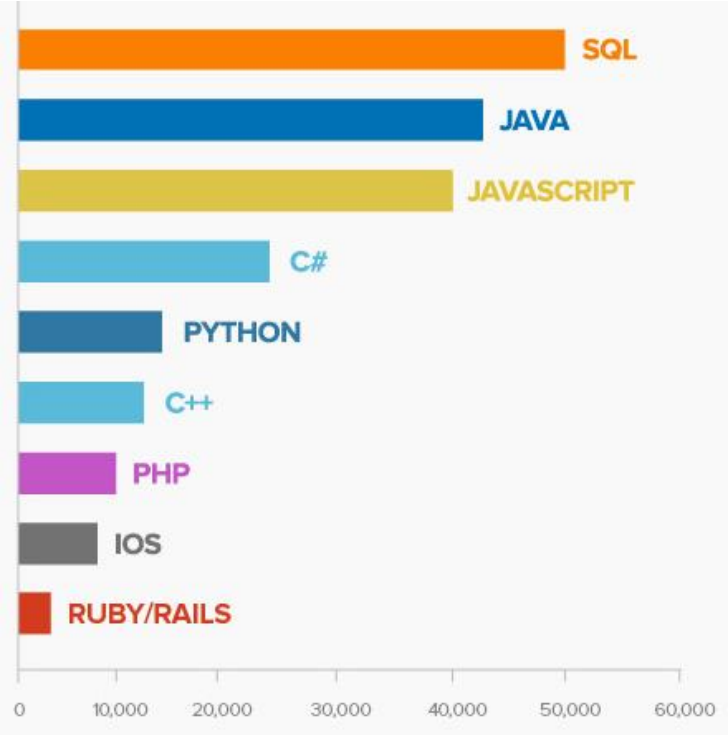



*Gráfico 2: Cantidad de trabajos por lenguaje de programación (Indeed.com, 2016)*

El entender un programa es una tarea compleja pero vital para poder evolucionar el código, lo cual implica entender dos tipos de información: en primer lugar, hay información sobre el conjunto básico de objetos, incluyendo sus propiedades, el conjunto de operaciones que se pueden realizar en estos objetos, y el orden en que estas operaciones se llevan a cabo. En segundo lugar, hay información sobre la relación entre los objetos y los operadores en un dominio y los que están en un dominio cercano. (Brooks, 1978).

Es por ello que este trabajo facilita el proceso de entendimiento y análisis de procedimientos almacenados SQL mediante una herramienta de visualización de datos que representa gráficamente el código del procedimiento SQL sin perder las características y atributos principales del programa original y a su vez permite abstraer la lógica de forma tal que su representación gráfica se pueda entender rápidamente.

Normalmente en el área de ingeniería de software las representaciones gráficas dejan de lado los aspectos visuales y se enfocan en la parte semántica lo cual hace que no se logre maximizar la efectividad cognitiva (que es la velocidad, facilidad y precisión con que la información puede ser

extraída de un diagrama), y esto es primordial pues las representaciones visuales son creadas para los humanos y tienen poco valor o no tienen del todo para las computadoras (Moody & van Hillegersberg, 2009).

Por lo tanto, el diseño de la notación visual que se desarrolló como resultado de esta investigación se rigió por la teoría de la física de las notaciones (Moody D. L., 2009) y sus principios, los cuales ayudan a incrementar la efectividad cognitiva de la notación, a saber:

1. Principio de *claridad semiótica* (ver 2.3.1)
2. Principio de *capacidad de discriminación perceptiva* (ver 2.3.2)
3. Principio de *transparencia semántica* (ver 2.3.3)
4. Principio de *administración de la complejidad* (ver 2.3.4)
5. Principio de *integración cognitiva* (ver 2.3.5)
6. Principio de *expresividad visual* (ver 2.3.6)
7. Principio de *codificación dual* (ver 2.3.7)
8. Principio de *economía gráfica* (ver 2.3.8)
9. Principio de *ajuste cognitivo* (ver 2.3.9)

## 1.1 El problema y su importancia

Los sistemas de cómputo actuales son cada vez más complejos y dependen en gran medida de datos almacenados en bases de datos y procedimientos almacenados para procesar esos datos, los cuales en muchos casos superan los miles de líneas de código SQL, esto hace que el proceso de entender el código sea una tarea laboriosa para el ser humano, por lo cual es muy importante contar con herramientas que faciliten la comprensión del código. Este proceso (de entender un programa) depende de manera directa de la complejidad del programa que se está tratando de entender.

La complejidad de un programa determina lo difícil que es para los programadores trabajar con el código fuente y más en detalle denota la capacidad de prueba, el mantenimiento, la facilidad de lectura y / o la inteligibilidad de un programa. De manera particular la complejidad de un programa es importante en la fase de desarrollo (influye en el esfuerzo necesario para desarrollar, depurar y probar programas) y en la fase de mantenimiento (determina la dificultad para localizar y corregir los errores de implementación no detectados, y la cantidad de esfuerzo que se necesitará para modificar los módulos del programa para incorporar los cambios de especificación). Siendo esta última la parte más costosa del ciclo de vida del software. (Schneider & Sedlmeyer, 1981)

Una persona entiende un programa cuando es capaz de explicar el programa, su estructura, su comportamiento, sus efectos, y sus relaciones con su dominio de aplicación en los términos que son diferentes de los símbolos utilizados para construir el código fuente del programa (Biggerstaff, 1993).

Para entender un programa, un programador realiza un análisis del código fuente del mismo. En este análisis, un programa tal como se entiende por un programador, consta de un cuerpo considerablemente mayor de información que la que utiliza el computador durante su compilación. Esta información se describe en términos de conceptos psicológicos de dominios de conocimiento. Un *dominio de conocimiento* consiste de un conjunto de objetos primitivos, las propiedades de los objetos, las relaciones entre los objetos y los operadores que manipulan estas propiedades o relaciones. (Brooks, 1978).

El proceso de usar esta información para reconstruir el dominio del conocimiento está basado en el refinamiento sucesivo de hipótesis sobre el funcionamiento del programa. Estas hipótesis se generan inicialmente a partir de los conocimientos del programador y de la programación. El papel de las fuentes de información es ayudar al programador en la refinación y la elaboración de sus hipótesis al confirmar o refutar partes de ellos y sugerir alternativas. El proceso de refinamiento se concluye cuando el programador siente que sabe lo suficiente sobre los dominios de conocimiento para comenzar su tarea. (Brooks, 1978)

Este problema es una tarea muy compleja, pero a su vez es un factor crítico en el ciclo de vida del software. En la fase de desarrollo la complejidad influye fuertemente en el esfuerzo necesario para depurar y probar módulos, programas y subsistemas. En la fase de mantenimiento la complejidad determina la dificultad para localizar y corregir los errores de implementación no detectados, y también la cantidad de esfuerzo que se necesitará para modificar los módulos del programa al incorporar cambios en la especificación. (Schneider, Sedlmeyer, & Kearney, 1981).

El programa se considera cada vez más como un objeto para el consumo en el cual el valor del mismo se basa en parte en lo fácil que es entender, así los programadores lo leen, lo entienden, y lo modifican. Un objetivo parcial de muchos de los nuevos métodos de construcción de software es producir programas que sean más comprensibles. (Brooks, 1978)

De acuerdo con (Gatterbauer, 2011), y con lo que se pudo investigar todavía no existen sistemas o herramientas que permitieran a los usuarios visualizar consultas SQL para facilitar el proceso de comprender un programa de una manera sencilla y rápida. Por lo que es deseable contar con herramientas que faciliten entender y analizar código fuente.

Como se puede observar en el Gráfico 1 las bases de datos relacionales constituyen el segmento mayor del mercado en tecnologías de bases de datos y por otra parte en el caso de código SQL, dada la gran cantidad de empresas que utilizan bases de datos relacionales, es tanta la cantidad de código que se está desarrollando y/o manteniendo que actualmente SQL es el lenguaje con mayor demanda del mercado (ver Gráfico 2). Esto presupone que son muchas personas que se enfrentan día con día con el problema de entender código escrito en lenguaje SQL. Dado lo complejo que puede ser un programa, algoritmo, etc., se requiere de un tiempo considerable para poder entender ese código, principalmente si está escrito por otro programador, por lo que se

hace necesario contar con una alternativa visual que cumpla con los objetivos que se presentan en la sección 1.2)

## 1.2 Objetivos

El objetivo de esta investigación fue plantear una nueva notación visual para representar gráficamente instrucciones SQL y adicionalmente desarrollar una herramienta capaz de convertir procedimientos almacenados a su equivalente utilizando esta notación visual. Esta representación visual permite al ser humano procesar y entender código SQL de manera paralela lo cual el cerebro humano realiza considerablemente más rápido que texto esto por cuanto el texto es procesado de manera secuencial, Además de ahorrar tiempo, también la visualización ayudará en factores como: producir menos errores y dar más confianza durante el proceso de comprensión de código.

**Objetivo General**

Desarrollar una notación y herramienta de visualización de datos que mejore la capacidad de análisis de procedimientos almacenados SQL.

**Objetivos Secundarios**

1. Diseñar una notación visual capaz de abstraer, representar, y analizar código SQL.
2. Analizar y escoger la técnica de visualización de datos que permita representar código SQL mediante el mapeo de instrucciones con sus respectivas representaciones gráficas.
3. Tipificar y caracterizar las instrucciones para ser analizadas por la herramienta.
4. Reducir el tiempo necesario para leer y entender código SQL.
5. Validar la eficacia del modelo propuesto.

# 2 Marco Teórico

## 2.1 Antecedentes

En 1974 Kernighan y Plauger hicieron la siguiente declaración en su documento:

> *La mejor documentación de un programa de cómputo es una estructura limpia. También ayuda si el código está bien formateado, con buenos identificadores mnemotécnicos, etiquetas, y un puñado de comentarios esclarecedores. Diagramas de flujo y descripciones de los programas son de importancia secundaria; la única documentación fiable de un programa de ordenador es el propio código. La razón es simple - cuando hay múltiples representaciones de un programa, existe la posibilidad de discrepancia. Si el código es un error, diagramas de flujos artísticos y comentarios detallados son en vano. (Brooks, 1978)*

En la actualidad esa afirmación sigue siendo válida en tanto haya múltiples representaciones de un programa ciertamente se puede presentar discrepancias entre las representaciones. De ahí la importancia de que la representación visual que se pueda obtener del código sea instantánea (el usuario la pueda obtener directamente del código fuente) de manera que siempre represente una versión fiel del código original.

La importancia de las representaciones visuales en la comprensión de los programas de computador no es nueva, Goldstein y von Neumann (1947) demostraron la utilidad de los diagramas de flujo, mientras que Haibt (1959) desarrolló un sistema que podría abstraerlos automáticamente de programas en lenguaje ensamblador o Fortran. Knuth (1963) desarrolló un sistema similar que integra la documentación con el código fuente y podría generar automáticamente diagramas de flujo. (Price & Baecker , 1994)

Autores como Shneiderman y otros realizaron experimentos en los que no lograron demostrar la utilidad de los diagramas de flujo en la comprensión, la depuración, o modificación del programa, y señalaron que es necesario seguir trabajando para replicar los resultados, y explorar otras áreas en las que los diagramas de flujo pueden ser útiles, para estudiar otras formas de diagramas de flujo y contribuir a una comprensión más profunda de las habilidades cognitivas necesarias en la programación de computadoras. (Shneiderman, Mayer, McKay, & Heller, 1977).

Por su parte (Scanlan, 1989) demostró que los diagramas de flujo son preferidos por estudiantes para la comprensión de los algoritmos con respecto al pseudocódigo. Sus resultados indican claramente que los diagramas de flujo estructurados ayudan a la comprensión de algoritmos: (1) llevan menos tiempo para comprender, (2) producen menos errores en la comprensión, (3) dan a los estudiantes mayor confianza en su comprensión de un algoritmo, (4) reducen el tiempo dedicado a responder a las preguntas acerca de un algoritmo, y (5) reducen el número de veces que los estudiantes necesitan mirar a un algoritmo.

Si bien es cierto que los experimentos de Shneiderman y otros no son alentadores en cuanto al uso de notaciones visuales para mejorar la comprensión de programas, consideramos que estos experimentos dejaron por fuera la variable más importante que es el tiempo requerido para entender un programa, algoritmo, etc. Esto por cuanto cualquier representación ya sea visual o textual puede ser entendida en un tiempo determinado, pero lo importante es (al igual como lo demostró Scanlan con sus experimentos) que el utilizar representaciones visuales (e.g diagramas de flujo como ayuda en la comprensión de programas) se ahorra tiempo en el proceso de comprensión de código.

En 2011 Gatterbauer propone una nueva forma de visualizar consultas SQL (ver Figura 3) y aunque consideramos que la visualización propuesta por (Gatterbauer, 2011) adolece de conceptos importantes como son la agregación (group by), ordenamiento (order by) y filtrado (condiciones where) si presenta un concepto interesante para representar la relación de entidades (Joins):

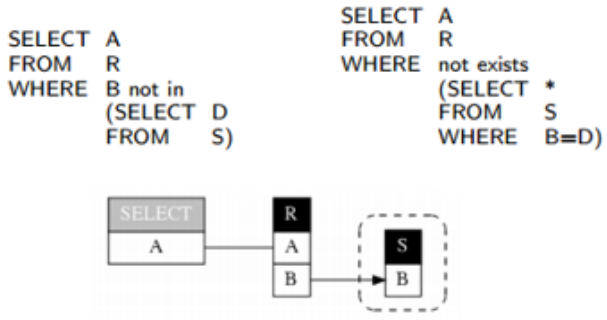



*Figura 3: Visualización de consultas SQL propuesta por (Gatterbauer, 2011)*

Este trabajo se ha enfocado en mejorar la comprensión de código SQL mediante la visualización de estatutos SQL, los cuales al ser representados gráficamente pueden ser procesados por el

sistema de percepción humana de forma paralela en lugar de tener que ser procesados de la manera tradicional (como texto) de forma secuencial.

El problema de mejorar la capacidad de análisis de código SQL, según (Gatterbauer, 2011) se puede resolver de cuatro formas:

1. **Manipulación visual del texto**: Permite a los usuarios interpretar más fácilmente el código con la ayuda de colores, indentación (alineación de bloques de texto), resaltado de construcciones sintácticas, etc. Aunque es útil, no es suficiente para ayudar a los usuarios a entender la intención de una consulta rápida, por ejemplo:

```
SELECT [columna_1]
   FROM [tabla_a]
   WHERE [columna_2] > 100;
```

2. **Traducción a lenguaje natural**: Consiste en explicar las consultas en lenguaje natural. Lo cual tiene el inconveniente que por la naturaleza del lenguaje natural hace que sea muy verboso el resultado, incrementando así la cantidad de información que se requiere procesar para entender el código.
3. **Ilustración con casos de ejemplo**: Consiste en la generación de datos de ejemplo para los flujos de datos del programa lo cual permite ilustrar rápidamente la semántica de los operadores.
4. **Visualización de la consulta**: La visualización de consultas crea una representación esquemática de una consulta SQL.

Dado que el objetivo principal de esta investigación fue disminuir el tiempo necesario para entender código SQL la misma se enfocó en la opción número cuatro (visualización de consultas) principalmente porque le permite al usuario entender el código en forma paralela (ver Figura 4) y por ende permite disminuir más el tiempo requerido para entender las consultas:

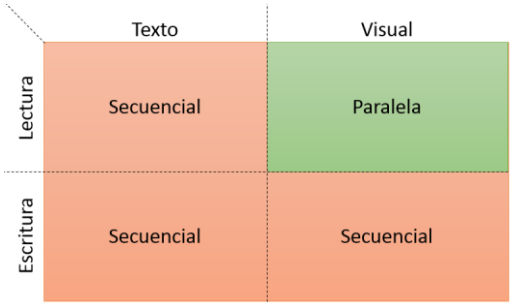



*Figura 4: Interpretación de información por el cerebro humano (Gatterbauer, 2011)*

Esta investigación se enmarcó en las áreas de: visualización de información (específicamente en lo que se denomina como visualización de software), entendimiento de software y reingeniería de software.

Específicamente en el área de visualización de software, el problema se ubicó dentro de las dimensiones de visualización de software (Marcus, Feng, & Maletic, 2003), las cuales se describen en la Tabla 1.

*Tabla 1: Dimensiones de visualización de software*

| Dimensión | **Aplicación** |
|---|---|
| Tareas<br>**¿Por qué es necesaria la visualización?** | La representación es necesaria para comprender código SQL en menos tiempo del que tomaría comprender la representación textual. |
| Público<br>**¿Quién va a utilizar la visualización?** | La visualización puede ser utilizada por cualquier persona que conozca lenguaje SQL y que necesite comprender, depurar, modificar código SQL. |
| Meta<br>**¿Cuál es la fuente de datos para representar?** | Como fuente de datos se utiliza el mismo código SQL de los procedimientos almacenados los cuales son procesados (parser) de forma automática y convertidos en una representación visual del código |
| Representación<br>**Forma de representarla** | Para representar el código cada sentencia SQL (las instrucciones incluidas en este trabajo se pueden ver en la sección 3.1.1) cuenta con una equivalencia visual que permite identificar de manera única el tipo de sentencia. |

| | |
|---|---|
| Medio<br><br>**¿Dónde representar la visualización?** | La representación se lleva a cabo en una herramienta de visualización que se desarrolló para este fin como parte del proyecto de investigación. |

Como alcance de la investigación se desarrolló una representación gráfica que mejora la capacidad de análisis de código SQL con respecto a su equivalencia textual. Dicha representación gráfica soporta las necesidades de los usuarios (Shneiderman B. , 1996) descritas en la Tabla 2)

*Tabla 2: Necesidades de visualizaciones gráficas*

| Necesidad | Descripción | Ejemplo |
|---|---|---|
| Contenido | Permite obtener una visión general de toda la colección de datos que se representa. | Todo el código SQL se representa en un mismo diagrama el cual dependiendo de la extensión del código puede contener múltiples páginas |
| Zoom | Incluye métodos para profundizar en los niveles más bajos de abstracción. | El diagrama cuenta con zoom el cual permite agrandar o disminuir el área que se está desplegando en pantalla. |
| Filtro | Filtrar los elementos sin interés. Puede ser mediante la eliminación de partes de la visualización que no están relacionadas con un objeto específico. | La herramienta cuenta con capacidades de filtrado a nivel de tabla lo cual permite desplegar únicamente las instrucciones que incluyen la tabla que se quiere filtrar. |
| Detalles bajo demanda | Seleccionar un elemento o grupo y obtener detalles cuando sea necesario. | |
| Relacionar | Ver relaciones entre los elementos. Esta es una de las características más importantes de muchos sistemas de visualización software. Los sistemas de software se basan en muchos componentes interrelacionados, que trabajan juntos para resolver problemas | |
| Historia | Mantener un historial de acciones para apoyar a | |

| | | |
|---|---|---|
| | deshacer, reproducir, y el refinamiento progresivo incluyendo una ruta de visitas. | |
| Extracto | Permitir la extracción de subcolecciones y de los parámetros de consulta. Esto se relaciona únicamente con la aplicación y el conjunto de datos subyacente. ¿Cómo se visualiza los datos? no afecta a esta. | |

## 2.2 Lenguaje SQL

En junio de 1970 el Dr. E. F. Codd publicó un artículo titulado "Un modelo relacional de datos para grandes bancos de datos compartidos" en la revista comunicaciones de la ACM (Association of Computer Machinery). Este modelo de Codd ha sido aceptado como el modelo estándar para sistemas de administración de bases de datos relacionales (RDBMS). El modelo relacional utiliza un lenguaje de consulta que originalmente fue llamado Lenguaje Estructurado de Consultas en inglés (SEQUEL) y fue desarrollado por Donald D. Chamberlin y Raymond F. Boyce de IBM Corporation, Inc. a principios de 1970, para usar el modelo de Codd. SEQUEL más tarde se convirtió en SQL (todavía se pronuncia "sequel") porque el acrónimo SEQUEL era una marca registrada de la compañía de aviones Hawker Siddeley. En 1979, Relational Software, Inc. (ahora Oracle) presentó la primera implementación de SQL disponible comercialmente (Oracle Version2 para computadoras VAX) (Chamberlin & Boyce, 1974).

La versión, inicial de SEQUEL fue diseñada para manipular y recuperar datos almacenados en el sistema original de gestión de bases de datos cuasi-relacional de IBM, System R, que un grupo del Laboratorio de Investigación IBM San José había desarrollado durante los años 70 (Chamberlin & Boyce, 1974).

Después de probar SQL en sitios de prueba de clientes para determinar la utilidad y practicidad del sistema, IBM comenzó a desarrollar productos comerciales basados en su prototipo System R, incluyendo System / 38, SQL / DS y DB2.

En la actualidad SQL es un lenguaje estándar que incluye un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de control de datos. El ámbito de SQL incluye inserción de datos, consulta, actualización y eliminación, creación y modificación de esquemas y control de acceso a datos.

SQL se convirtió en un estándar del Instituto Nacional de Estándares Americanos (ANSI) en 1986 y de la Organización Internacional de Normalización (ISO) en 1987. Desde entonces ha sido revisado para incluir un conjunto más amplio de características.

## 2.3 Diseño de notaciones visuales

Uno de las etapas claves de este trabajo fue la creación de una notación para representar código SQL en forma visual. En el diseño de esta notación se utilizó como base la teoría de la "*Física de Notaciones*" (Moody D. L., 2009) la cual brinda una base científica para poder diseñar notaciones visuales que estén optimizadas para ser procesadas por la mente humana (Teixeira, 2014). Como se verá más adelante en la sección de desarrollo, la notación visual tomó muy en cuenta los nueve principios que establece la teoría de (Moody D. L., 2009), a saber:

### 2.3.1 Principio de claridad semiótica

Para que una notación pueda satisfacer los requisitos de un sistema de notación, tiene que haber una correspondencia de uno a uno entre los símbolos y sus conceptos referentes (Moody D. L., 2009). Los sistemas de notaciones tienen como requisito el limitar las expresiones permitidas para maximizar la precisión, la expresividad, y parsimonia. Cuando no hay una correspondencia uno a uno entre los conceptos y las construcciones, pueden ocurrir alguna de las anomalías que se presentan en la Figura 5:

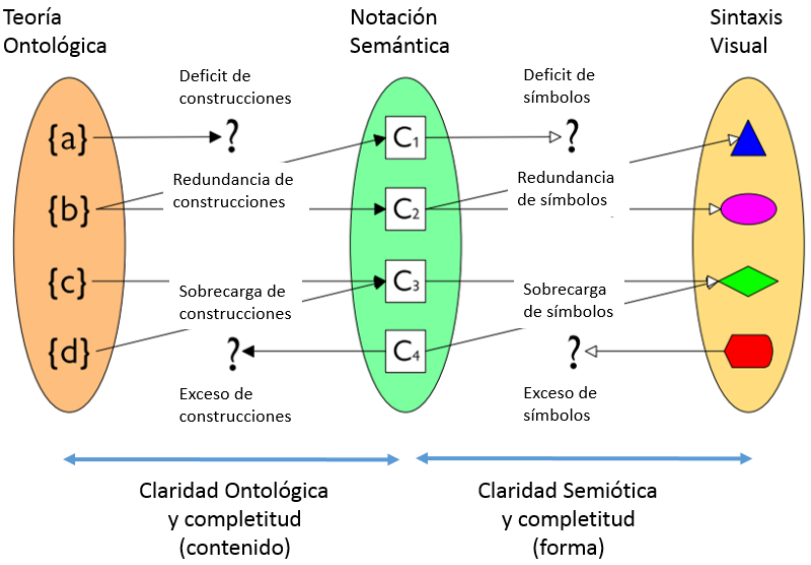



*Figura 5: Claridad ontológica y claridad semiótica (Moody D. L., 2009)*

- **Déficit de construcciones:** es cuando no existe construcción en la notación correspondiente a un concepto ontológico particular.

- **Redundancia de construcciones**: es cuando múltiples construcciones en la notación se pueden utilizar para representar un solo concepto ontológico.
- **Sobrecarga de construcciones**: es cuando una única construcción en la notación puede representar varios conceptos ontológicos.
- **Exceso de construcciones**: es cuando una construcción de notación no corresponde a ningún concepto ontológico.

Si existe un déficit de construcciones, la notación se dice que es *ontológicamente incompleta*; si existe alguna de las otras tres anomalías, no es clara ontológicamente.

Cuando no hay una correspondencia uno a uno entre las construcciones y símbolos, pueden ocurrir alguna de las siguientes anomalías:

- **Redundancia de símbolos**: se produce cuando múltiples símbolos gráficos se pueden utilizar para representar la misma construcción semántica.
- **Sobrecarga de símbolos**: se produce cuando dos construcciones diferentes pueden ser representados por el mismo símbolo gráfico. Este es el peor tipo de anomalía, ya que conduce a la ambigüedad y la posibilidad de una mala interpretación.
- **Exceso de símbolos**: se produce cuando hay símbolos gráficos no corresponden a ninguna construcción semántica. El exceso de símbolo incrementa la complejidad de diagramación y la complejidad gráfica, que afecta negativamente la comprensión.
- **Déficit de símbolos**: se produce cuando hay construcciones semánticas que no están representados por ningún símbolo gráfico.

### 2.3.2 Principio de capacidad de discriminación perceptiva

De acuerdo con (Moody D. L., 2009) discriminación perceptiva es la facilidad y precisión con la que los símbolos gráficos pueden ser diferenciados unos de otros, es decir si los diferentes símbolos pueden ser claramente distinguibles entre sí.

#### *2.3.2.1 Distancia Visual*

La Discriminación perceptiva está determinada principalmente por la distancia visual entre símbolos. Esto se mide por el número de variables visuales en las que difieren y el tamaño de estas diferencias lo cual es medido por el número de pasos perceptibles. Cada variable visual

tiene un número infinito de variaciones físicas, pero sólo un número finito de pasos perceptibles o valores que son confiablemente distinguible por la mente humana. En general, cuanto mayor es la distancia visual entre los símbolos, más rápido y con mayor precisión será reconocidos (Moody D. L., 2009). La distancia visual entre los símbolos se puede aumentar mediante el uso de múltiples variables visuales para distinguir entre ellos por ejemplo forma y color, según se muestra en la Figura 6.

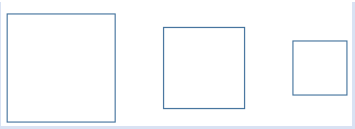
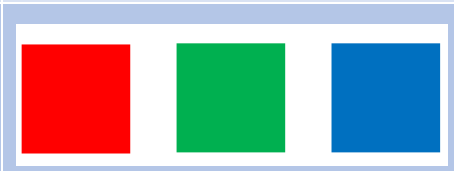
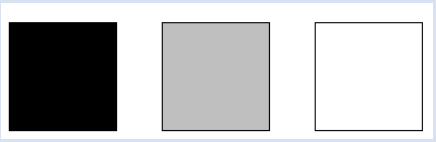
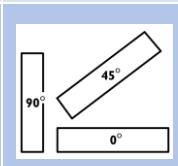
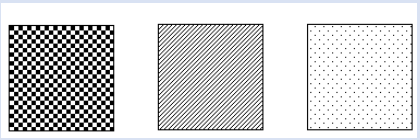

*Figura 6: Aumento de la distancia visual utilizando forma y color*

### *2.3.2.2 Variables Visuales*

Bertin (Moody D. L., 2009) identificó ocho variables visuales que se pueden utilizar para codificar gráficamente información. Estas se dividen en variables planas (las dos dimensiones espaciales) y variables de la retina, según se muestra en la Tabla 3.

*Tabla 3: Variables visuales*

| Variable | Tipo | Valor |
|---|---|---|
| **Horizontal Position** | Plana | |
| **Vertical Position** | Plana | |
| **Forma** | De retina |  |

| | | |
|---|---|---|
| **Tamaño** | De retina |  |
| **Color** | De retina |  |
| **Brillo** | De retina |  |
| **Orientación** | De retina |  |
| **Textura** | De retina |  |

### *2.3.2.3 La primacía de la forma*

De todas las variables visuales, la forma juega un papel especial en la discriminación entre símbolos, ya que representa la principal base para identificar los objetos en el mundo real. Por esta razón, la forma debe ser utilizada como la variable visual primaria para distinguir entre diferentes construcciones semánticas.

### *2.3.2.4 Codificación redundante*

La distancia visual entre símbolos puede incrementarse mediante la codificación redundante: utilizando múltiples variables visuales para distinguir entre ellas. Por ejemplo, agregarle color a un diagrama para mejorar la discriminabilidad entre figuras.

#### *2.3.2.5 Resaltamiento perceptual*

De acuerdo con la teoría de la integración de características, los elementos visuales con valores únicos para al menos una variable visual pueden detectarse sin mucha atención y en paralelo a través del campo visual. Estos elementos parecen resaltar en una pantalla sin esfuerzo consciente. Por otro lado, los elementos visuales que se diferencian por combinaciones únicas de valores (conjunciones) requieren una búsqueda en serie, que es mucho más lenta y propensa a errores. Esto implica que en el diseño de una notación visual cada símbolo gráfico debe tener un valor único en al menos una variable visual.

### 2.3.3 Principio de transparencia semántica

La transparencia semántica se define como la medida en que el significado de un símbolo puede inferirse desde su aparición (Moody D. L., 2009). Mientras que la discriminación perceptual simplemente requiere que los símbolos deben ser diferentes entre sí, este principio requiere que proporcionen señales de su significado (la forma implica contenido). El concepto de transparencia semántica formaliza las nociones informales de "naturalidad" o "intuición" que se usan a menudo cuando se habla de notaciones visuales, ya que puede evaluarse experimentalmente. Las representaciones semánticamente transparentes reducen la carga cognitiva porque tienen mnemotécnicas incorporadas: ser o no ser percibido directamente o fácilmente aprendido. La transparencia semántica no es un estado binario sino un continuo (ver Figura 7)

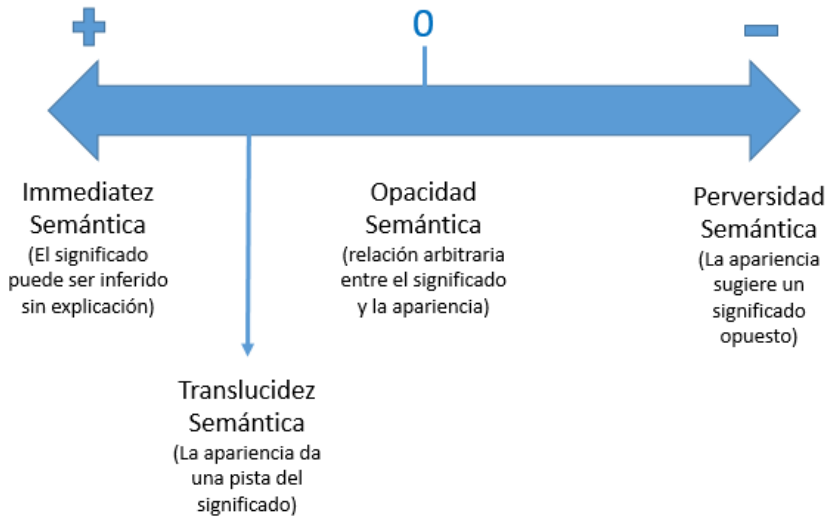



*Figura 7: Grados de transparencia semántica (Moody D. L., 2009)*

- Un símbolo es *semánticamente inmediato* si un lector novato es capaz de inferir su significado desde su aparición solo (por ejemplo, un triángulo sobre un cuadrado para representar una casa)
- Un símbolo es *semánticamente opaco* (o convencional) si existe una relación puramente arbitraria entre su apariencia y su significado (por ejemplo, círculos en diagramas de flujo de datos): Esto representa el punto cero en la escala
- Un símbolo es *semánticamente perverso* si un lector pudiera inferir un significado diferente u opuesto desde su aparición: Esto representa un valor negativo en la escala.
- Un símbolo es *semánticamente translucido* si está entre semánticamente inmediato y semánticamente opaco es decir si puede dar una pista al lector sobre su significado.

#### *2.3.3.1 Semejanza perceptual*

Los iconos se parecen perceptualmente a los conceptos que representan. Esto refleja una distinción básica en la semiótica entre signos simbólicos e icónicos. Las representaciones icónicas aceleran el reconocimiento y la memoria, hacen los diagramas más atractivos (las personas prefieren objetos reales en lugar de formas abstractas) y mejoran la inteligibilidad de los diagramas para los usuarios ingenuos y principiantes.

Los iconos se utilizan mucho en interfaces gráficas de usuario, pero no tanto en notaciones de ingeniería de software las cuales utilizan normalmente formas geométricas abstractas para representar construcciones.

#### *2.3.3.2 Relaciones semánticamente transparentes*

La transparencia semántica también se aplica a la representación de las relaciones. El posicionamiento de los elementos visuales de cierta manera predispone a la persona hacia una interpretación particular de la relación incluso antes de que se conozca el significado de los elementos. La mayoría de las notificaciones visuales en ingeniería de software hacen uso limitado de tales relaciones y se basan principalmente en diferentes tipos de líneas de conexión para representar relaciones. Esto transmite la relación entre las entidades de una manera más semánticamente transparente que el uso de flechas, por lo que es más probable que se interprete correctamente y se recuerde más fácilmente.

### 2.3.4 Principio de manejo de la complejidad

Este principio se refiere (Moody D. L., 2009) a como una notación visual es capaz de manejar la complejidad es decir es capaz de representar información sin sobrecargar la mente humana. En ingeniería de software los niveles de complejidad exceden a los de cualquier otra disciplina por lo que hace que esto sea uno de los problemas más difíciles en el diseño de notaciones visuales. Este principio define los requisitos generales para administra la complejidad en las anotaciones visuales. En este contexto, la "complejidad" se refiere a la complejidad del diagrama, que se mide por el número de elementos (símbolos o tokens) en un diagrama. Aunque esto es aparentemente un problema de nivel de diagrama, la notación requiere de ciertas características para resolverlo. La complejidad tiene un efecto importante sobre la efectividad cognitiva ya que la cantidad de información que puede ser transmitida de manera efectiva por un solo diagrama es limitada, por las capacidades perceptivas y cognitivas humanas:

**Límites perceptuales**: La capacidad de discriminar entre los elementos del diagrama aumenta con el tamaño del diagrama.

**Límites cognitivos**: El número de elementos del diagrama que pueden ser comprendidos a la vez está limitado por la capacidad de la memoria de trabajo. Cuando se supera, se produce un estado de sobrecarga cognitiva y la comprensión se degrada rápidamente.

El manejo eficaz de la complejidad es especialmente importante cuando se trata de principiantes, que están menos preparados para hacer frente a la complejidad. La complejidad excesiva es una de las principales barreras para la comprensión por el usuario final de los diagramas.

Sorprendentemente, algunas de las principales notaciones visuales de ingeniería de software carecen de mecanismos para administrar la complejidad. En ausencia de éstos, los problemas deben ser representados como diagramas monolíticos únicos, no importa cuán complejos sean. Por ejemplo, el modelo entidad-relación ha estado en uso durante más de tres décadas y todavía carece de tales mecanismos y en la práctica, abruma a los usuarios finales.

Algunas formas de manejar la complejidad son:

- **Modularización**: las notaciones deben proporcionar la capacidad de dividir grandes diagramas en "pedazos" más pequeños que sean perceptualmente y cognitivamente manejables.
- **Jerarquía (niveles de abstracción)**: La jerarquía es una de las maneras más eficaces de organizar la complejidad para la comprensión humana, ya que permite que los sistemas se representen en diferentes niveles de detalle, con complejidad manejable en cada nivel.

### 2.3.5 Principio de integración cognitiva

Según (Moody D. L., 2009) la integración cognitiva aplica cuando se utilizan múltiples diagramas para representar un sistema. Este es un problema crítico en ingeniería de software, donde los problemas son típicamente representados por sistemas de diagramas en lugar de diagramas individuales. Este principio está estrechamente relacionado con el manejo de la complejidad, que conduce a múltiples diagramas como resultado de la modularización, pero se aplica incluso cuando la modularidad no se utiliza (debido a la integración heterogénea).

Representar sistemas usando múltiples diagramas agregar retos cognitivos adicionales en el lector para integrar mentalmente información de diferentes diagramas y mantener pista de donde están. Para que las representaciones multidiagrama sean cognitivamente efectivas, debe incluir mecanismos explícitos para apoyar:

- **Integración conceptual**: Mecanismos para ayudar al lector a reunir información a partir de diagramas separados en una representación mental coherente del sistema.
- **Integración perceptual**: señales perceptuales para simplificar la navegación y las transiciones entre diagramas

**Integración conceptual:** en la integración conceptual se puede utilizar las siguientes técnicas:

- un **diagrama general**, que proporciona una visión global del sistema. Esto actúa como un mapa cognitivo en el cual la información de los diagramas individuales puede ser unificada. En la integración homogénea, este diagrama es un resultado natural de la estructuración jerárquica (es la "raíz" de la jerarquía), sin embargo, en la integración heterogénea, es un nuevo diagrama.
- **Contextualización** es una técnica utilizada en la visualización de la información donde la parte de un sistema de interés actual (enfoque) se muestra en el contexto del sistema como un todo. En un contexto de diagramación, esto significa incluir información contextual en cada diagrama mostrando sus relaciones con elementos de otros diagramas. La manera más simple y efectiva de hacerlo es incluir todos los elementos directamente relacionados de otros diagramas (su "vecindad inmediata") como elementos extraños. Incluir la superposición entre diagramas de esta manera permite que cada elemento del sistema de diagramas se entienda en términos de sus relaciones con todos los demás elementos, lo que apoya la integración conceptual.

**Integración perceptual:** se basa en el diseño de espacios físicos (urbanismo), espacios virtuales (HCI) y espacios gráficos (cartografía y visualización de la información) siguiendo estas cuatro etapas:

1. Orientación: ¿Dónde estoy?
2. Selección de ruta: ¿A dónde puedo ir?
3. Seguimiento de ruta: ¿Estoy en el camino correcto?
4. Reconocimiento de destinos: ¿Todavía estoy allí?

El etiquetado claro de diagramas (identificación) apoya el reconocimiento de orientación y destino. La numeración de niveles (como se utiliza para mostrar la estructura de los documentos) es compatible con la orientación mostrando al usuario dónde están en el sistema de diagramas.

La inclusión de las señales de navegación en los diagramas (señalización) apoya la elección de la ruta. Un mapa de navegación, que muestra todos los diagramas y las rutas de navegación entre ellos, soporta la orientación, la supervisión de ruta y la elección de ruta.

### 2.3.6 Principio de la expresividad visual

De acuerdo con (Moody D. L., 2009) la expresividad visual se refiere al número de diferentes variables visuales utilizados en una notación visual. Hay 8 variables visuales primarias que se pueden utilizar para codificar gráficamente la información (ver Tabla 3). Estas se clasifican en variables planas (las dos dimensiones espaciales) y variables de la retina (características de la imagen de la retina). Esto maximiza la descarga computacional y es compatible con la plena utilización del espacio de diseño gráfico. Diferentes variables visuales tienen propiedades que los hacen adecuados para la codificación de algunos tipos de información, pero no otros. El conocimiento de estas propiedades es necesario tomar decisiones eficaces.

**Uso del color**

El color es una de las variables visuales más efectivas cognitivamente: el sistema visual humano es altamente sensible a las variaciones de color y puede distinguirlas rápidamente y con precisión. Las diferencias de color se detectan tres veces más rápido que la forma y también se recuerdan más fácilmente. Sin embargo, el color nunca debe usarse como la única base para distinguir entre símbolos ya que es sensible a variaciones en la percepción visual (por ejemplo, ceguera de color) y características de pantalla / impresora (por ejemplo, impresoras en blanco y negro). Para evitar la pérdida de información (diseño robusto), el color sólo debe utilizarse para la codificación redundante.

**Elección de las variables visuales**

La elección de las variables visuales no debe ser arbitraria, sino que debe basarse en la naturaleza de la información a transmitir. Diferentes variables visuales tienen propiedades que las hacen adecuadas para codificar diferentes tipos de información. Por ejemplo, el color sólo se puede utilizar para datos nominales, ya que no es psicológicamente ordenado. Además, diferentes variables visuales tienen diferentes capacidades (número de pasos perceptibles). Las propiedades de cada variable visual han sido establecidas por la investigación en sicofísica (ver Tabla 4). El

objetivo es hacer coincidir las propiedades de las variables visuales con las propiedades de la información a representar:

| Variable | Información | Capacidad |
|---|---|---|
| **Posición Horizontal** | Intervalo | 10-15 |
| **Posición Vertical** | Intervalo | 10-15 |
| **Tamaño** | Intervalo | 20 |
| **Color** | Nominal | 7-10 |
| **Textura** | Nominal | 3-5 |
| **Forma** | Nominal | Ilimitada |
| **Orientación** | Nominal | 4 |

*Tabla 4: Capacidades de las variables visuales*

### 2.3.7 Principio de codificación dual

De acuerdo con la teoría de codificación dual (Paivio, 1990), el uso de texto y gráficos en conjunto para transmitir información es más eficaz que utilizarlos por separado. Cuando la información se presenta verbal y visualmente, las representaciones de esa información se codifican en sistemas separados en la memoria de trabajo y las conexiones referenciales entre los dos se fortalecen. Esto sugiere que la codificación textual es más eficaz cuando se utiliza en un rol de apoyo: para complementar en lugar de sustituir a los gráficos. En particular, el texto nunca debe utilizarse como base única para distinguir entre símbolos (como se discute en la discriminabilidad perceptual), pero puede utilizarse de manera útil como una forma de codificación redundante, para reforzar y aclarar el significado. Otro argumento para la codificación dual es que las personas difieren ampliamente en sus habilidades de procesamiento espacial y verbal. La inclusión de gráficos y texto es probable que mejore la comprensión por parte de las personas en todo el espectro de habilidades espaciales y verbales.

### 2.3.8 Principio de economía gráfica

La complejidad gráfica se define por el número de símbolos gráficos en una notación (el tamaño del vocabulario visual). Esto difiere de la complejidad diagramática (Manejo de la complejidad),

ya que se relaciona con la complejidad en el nivel de tipo (lenguaje) en lugar del nivel de símbolo (oración). Un diseñador de notaciones puede crear un número ilimitado de símbolos combinando variables visuales de diferentes maneras. Sin embargo, en comparación con los lenguajes textuales, esta estrategia sólo es eficaz hasta cierto punto, ya que hay límites cognitivos en el número de categorías visuales que pueden ser reconocidos de manera efectiva. Más allá de esto, cada nuevo símbolo introducido reduce la eficacia cognitiva.

La complejidad gráfica afecta a los principiantes mucho más que a los expertos, ya que necesitan mantener conscientemente los significados de los símbolos en la memoria de trabajo. Si los símbolos no son mnemotécnicos, el usuario debe recordar lo que significan los símbolos o bien una leyenda debe ser suministrada y frecuentemente referenciada, lo cual se suma al esfuerzo de procesar los diagramas. La capacidad humana de discriminar entre alternativas perceptualmente distintas (rango de juicio absoluto) es alrededor de seis categorías. Esto define un límite superior para la complejidad gráfica. Existen tres estrategias principales para tratar con la complejidad gráfica:

- Reducir la complejidad semántica.
- Introducir el déficit de símbolos.
- Aumentar la expresividad visual

### 2.3.9 Principio de adecuación cognitiva

El principio de adecuación cognitiva utiliza diferentes dialectos visuales para diferentes tareas y audiencias. La teoría del ajuste cognitivo es una teoría ampliamente aceptada en el campo de los sistemas de información (IS) que ha sido validada en una amplia gama de dominios.
Para programar el mantenimiento. La teoría establece que diferentes representaciones de la información son adecuadas para diferentes tareas y diferentes audiencias. El rendimiento de la resolución de problemas (que corresponde aproximadamente a la efectividad cognitiva) se determina mediante un ajuste de tres vías entre la representación del problema, las características de la tarea y las habilidades para resolver problemas. En ingeniería de software normalmente se usa una sola representación visual para todos los propósitos. Sin embargo, la teoría del ajuste cognitivo sugiere que esta suposición de "talla única" puede ser inapropiada y diferentes

dialectos visuales pueden ser requeridos para diferentes tareas y / o audiencias. Hay al menos dos razones para crear múltiples dialectos visuales en un contexto de ingeniería de software:

- Nivel de experiencia (principiante – experto)
- Características de la tarea

# 3 Desarrollo

Como se mencionó anteriormente para poder satisfacer el objetivo general de este trabajo el cual era ahorrar tiempo al analizar código SQL, se desarrolló una solución que incluyó el diseño de una notación visual para representar instrucciones SQL y una herramienta de software que convierte instrucciones SQL a su equivalente de la notación visual. Por restricciones de tiempo la solución se implementó para un conjunto reducido pero importante de instrucciones SQL.

## 3.1 Tipificación de instrucciones

Una de las etapas iniciales de este trabajo fue seleccionar las instrucciones a incluir en la notación y a graficar por las herramientas lo cual se realizó mediante un análisis de 5282 archivos SQL los cuales contenían 832918 de líneas de código SQL. En este análisis se utilizó el analizador sintáctico (parser) de SQL Server® de la versión 2014 lo que permitió categorizar las instrucciones SQL con precisión y poder determinar la utilización e importancia de varias instrucciones. La Tabla 5 muestra los resultados de la aparición de instrucciones SQL en la muestra de código analizada:

| Instrucción | Líneas | Porcentaje |
|---|---|---|
| SELECTs | 8465 | 7% |
| INSERTs | 22698 | 19% |
| UPDATEs | 2799 | 2% |
| DELETEs | 906 | 1% |
| DECLARES | 18587 | 16% |
| Exec | 4426 | 4% |
| If | 15064 | 13% |
| Set | 8558 | 7% |
| Alter_Table | 2531 | 2% |
| Create_Table | 3658 | 3% |
| Predicate set | 11317 | 10% |
| Otras | 18144 | 15% |

| | | |
|---|---|---|
| **Total** | **117153** | |

*Tabla 5: Porcentaje de aparición de instrucciones en la muestra de código SQL analizado*

**Otras Instrucciones**:

El siguiente es el detalle de las instrucciones secundarias que aparecen con mayor frecuencia en el código analizado y que se cuantificaron en la categoría de "Otras" en la Tabla 5

| Instrucción | Cantidad |
|---|---|
| BeginEndBlockStatement | 2081 |
| BeginTransactionStatement | 1044 |
| CommitTransactionStatement | 771 |
| CreateIndexStatement | 1303 |
| CreateProcedureStatement | 2011 |
| GoToStatement | 782 |
| GrantStatement | 1663 |
| ReturnStatement | 1409 |
| TryCatchStatement | 2470 |
| **Total** | **18392** |

*Tabla 6: Instrucciones SQL secundarias con más apariciones en la muestra*

### 3.1.1 Instrucciones incluidas

Con base en los resultados del análisis de código SQL, para la implementación de la notación y la herramienta se decidió incluir las instrucciones que representan aproximadamente 80% del total de instrucciones (ver Tabla 7) y que permiten tener una visión general del funcionamiento de un bloque de código SQL:

1. Las instrucciones básicas de manipulación de datos (DML) que permiten las capacidades Creación, Recuperación/Consulta, Actualización y Borrado (CRUD por sus siglas en inglés.
2. Instrucción básica de declaración de variables (DECLARE)
3. Instrucción básica de asignación de variables (SET)
4. Instrucción básica de flujo de control (IF)
5. Instrucción básica de ejecución (EXECUTE)

Tabla 7: Instrucciones incluidas en la herramienta de visualización

| Instrucción | Líneas | Porcentaje |
|---|---|---|
| SELECTs | 8465 | 7% |
| INSERTs | 22698 | 19% |
| UPDATEs | 2799 | 2% |
| DELETEs | 906 | 1% |
| DECLARES | 18587 | 16% |
| EXECUTE | 4426 | 4% |
| If | 15064 | 13% |
| Set | 8558 | 7% |
| Predicate set | 11317 | 10% |
| Total | **117153** | **79%** |

*Tabla 7: Instrucciones incluidas en la herramienta de visualización*

La instrucción “Select” permite obtener registros desde la base de datos y habilita la selección de uno más registros de una o más tablas. Si bien la sintaxis de la instrucción en SQL server® es compleja las principales cláusulas se detallan a continuación:

### 3.1.2 Consulta (Select)

**Sintaxis**

```
[ WITH { [ XMLNAMESPACES ,] [ <tabla común> ] } ]
SELECT lista de columnas
[ INTO nueva tabla]
[ FROM tabla origen]
[ WHERE lista de condiciones]
[ GROUP BY lista de columnas de agrupamiento]
[ HAVING lista de condiciones]
[ ORDER BY lista de columnas de ordenamiento [ ASC | DESC ] ]
```

Los operadores UNION, INTERSECT y EXCEPT pueden ser usados para combinar o comparar resultados de consultas en un solo resultado.

*Tabla 8: Sintaxis de instrucción Select*

Nota: La herramienta de visualización está utilizando el analizador sintáctico de SQL Server 2014 por lo cual soporta todas las cláusulas incluidas en esta sintaxis.

### 3.1.3 Inserción (Insert)

La instrucción Insert permite agregar uno o más registros en una tabla o vista. Esta instrucción puede insertar valores individuales o resultados de otras consultas.

**Sintaxis**

La sintaxis de la instrucción Insert en Microsoft® SQL Server® se puede resumir como:

```
[ WITH <common_table_expression> [ ,...n ] ]
INSERT
{
        [ TOP ( expression ) [ PERCENT ] ]
        [ INTO ]
        { <object> | rowset_function_limited
          [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
        }
    {
        [ ( column_list ) ]
        [ <OUTPUT Clause> ]
        { VALUES ( { DEFAULT | NULL | expression } [ ,...n ] ) [ ,...n     ]
        | derived_table
        | execute_statement
        | <dml_table_source>
        | DEFAULT VALUES
        }
    }
}
[;]

<object> ::=
{
    [ server_name . database_name . schema_name .
      | database_name .[ schema_name ] .
      | schema_name .
    ]
  table_or_view_name
}

<dml_table_source> ::=
    SELECT <select_list>
    FROM ( <dml_statement_with_output_clause> )
      [AS] table_alias [ ( column_alias [ ,...n ] ) ]
    [ WHERE <search_condition> ]
        [ OPTION ( <query_hint> [ ,...n ] ) ]
```

*Tabla 9: Sintaxis de la instrucción Insert*

Nota: La herramienta de visualización está utilizando el analizador sintáctico de SQL Server® 2014 por lo cual soporta todas las cláusulas mostradas en esta sintaxis.

### 3.1.4 Actualización (Update)

La instrucción Update permite modificar uno o más registros en una tabla o vista. En cada registro permite modificar una o más columnas. Esta instrucción puede modificar las columnas con valores literales, valores de otras columnas o de otras tablas.

**Sintaxis**

La sintaxis de la instrucción Update en Microsoft® SQL Server® se puede resumir como:

```
[ WITH <common_table_expression> [...n] ]
UPDATE
    [ TOP ( expression ) [ PERCENT ] ]
    { { table_alias | <object> | rowset_function_limited
         [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]       }
      | @table_variable
    }
    SET
        { column_name = { expression | DEFAULT | NULL }
          | { udt_column_name.{ { property_name = expression
                                | field_name = expression }
                                | method_name ( argument [ ,...n ] )
                              }
          }
          | column_name { .WRITE ( expression , @Offset , @Length ) }
          | @variable = expression
          | @variable = column = expression
          | column_name { += | -= | *= | /= | %= | &= | ^= | |= } expression
          | @variable { += | -= | *= | /= | %= | &= | ^= | |= } expression
          | @variable = column { += | -= | *= | /= | %= | &= | ^= | |= }
expression
        } [ ,...n ]

    [ <OUTPUT Clause> ]
    [ FROM{ <table_source> } [ ,...n ] ]
    [ WHERE { <search_condition>
            | { [ CURRENT OF { { [ GLOBAL ] cursor_name }
                              | cursor_variable_name}
                ]
              }
            }
    ]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
[ ; ]

<object> ::=
{
    [ server_name . database_name . schema_name .
    | database_name .[ schema_name ] .
    | schema_name .
    ]
    table_or_view_name}
```

*Tabla 10: Sintaxis de la instrucción Update*

### 3.1.5 Borrado (Delete)

La instrucción Delete permite borrar uno o más registros en una tabla o vista.

**Sintaxis**

La sintaxis de la instrucción Delete en Microsoft® SQL Server® se puede resumir como:

```
[ WITH <common_table_expression> [ ,...n ] ]
DELETE
    [ TOP ( expression ) [ PERCENT ] ]
    [ FROM ]
    { { table_alias
      | <object>
      | rowset_function_limited
      [ WITH ( table_hint_limited [ ...n ] ) ] }
      | @table_variable
    }
    [ <OUTPUT Clause> ]
    [ FROM table_source [ ,...n ] ]
    [ WHERE { <search_condition>
            | { [ CURRENT OF
                    { { [ GLOBAL ] cursor_name }
                        | cursor_variable_name
                    }
                ]
              }
            }
    ]
    [ OPTION ( <Query Hint> [ ,...n ] ) ]
[; ]

<object> ::=
{
    [ server_name.database_name.schema_name.
      | database_name. [ schema_name ] .
      | schema_name.
    ]
    table_or_view_name
}
```

*Tabla 11: Sintaxis de la instrucción Delete*

Nota: La herramienta de visualización está utilizando el analizador sintáctico de SQL Server® 2014 por lo cual soporta todas las cláusulas mostradas en esta sintaxis.

### 3.1.6 Declaración de variables (DECLARE)

La instrucción Declare permite declarar variable en el cuerpo de un bloque de ejecución o un procedimiento almacenado.

**Sintaxis**

```
DECLARE
{
    { @local_variable [AS] data_type  | [ = value ] }
  | { @cursor_variable_name CURSOR }
} [,...n]
| { @table_variable_name [AS] <table_type_definition> }

<table_type_definition> ::=
     TABLE ( { <column_definition> | <table_constraint> } [ ,... ] )

<column_definition> ::=
     column_name { scalar_data_type | AS computed_column_expression }
     [ COLLATE collation_name ]
     [ [ DEFAULT constant_expression ] | IDENTITY [ (seed ,increment ) ]
]
     [ ROWGUIDCOL ]
     [ <column_constraint> ]

<column_constraint> ::=
     { [ NULL | NOT NULL ]
     | [ PRIMARY KEY | UNIQUE ]
     | CHECK ( logical_expression )
     | WITH ( <index_option > )
     }

<table_constraint> ::=
     { { PRIMARY KEY | UNIQUE } ( column_name [ ,... ] )
     | CHECK ( search_condition )
     }
```

*Tabla 12: Sintaxis de la instrucción Declare*

Nota: La herramienta de visualización está utilizando el analizador sintáctico de SQL Server® 2014 por lo cual soporta todas las cláusulas incluidas en esta sintaxis.

### 3.1.7 Ejecución de instrucciones SQL (EXECUTE)

La instrucción EXECUTE permite ejecutar instrucciones de código SQL las cuales pueden ser procedimientos almacenados o funciones.

**Sintaxis**

```
[ { EXEC | EXECUTE } ]
    {
      [ @return_status = ]
      { module_name [ ;number ] | @module_name_var }
        [ [ @parameter = ] { value
                           | @variable [ OUTPUT ]
                           | [ DEFAULT ]
                           }
        ]
      [ ,...n ]
      [ WITH <execute_option> [ ,...n ] ]
    }
[;]

Execute a character string
{ EXEC | EXECUTE }
    ( { @string_variable | [ N ]'tsql_string' } [ + ...n ] )
    [ AS { LOGIN | USER } = ' name ' ]
[;]

Execute a pass-through command against a linked server
{ EXEC | EXECUTE }
    ( { @string_variable | [ N ] 'command_string [ ? ]' } [ + ...n ]
        [ { , { value | @variable [ OUTPUT ] } } [ ...n ] ]
    )
    [ AS { LOGIN | USER } = ' name ' ]
    [ AT linked_server_name ]
[;]

<execute_option>::=
{
        RECOMPILE
    | { RESULT SETS UNDEFINED }
    | { RESULT SETS NONE }
    | { RESULT SETS ( <result_sets_definition> [,...n ] ) }
}
```

*Tabla 13: Sintaxis de la instrucción Execute*

### 3.1.8 Ejecución Condicional (IF ELSE)

La instrucción IF ELSE permite ejecutar bloques de código SQL con base en el resultado de la evaluación de una condición. Si el resultado es verdadero se ejecuta el bloque del IF y si el resultado es falso se ejecuta el bloque ELSE.

**Sintaxis**

```
IF Boolean_expression
     { sql_statement | statement_block }
[ ELSE
     { sql_statement | statement_block } ]
```

*Tabla 14: Sintaxis de la instrucción If Else*

### 3.1.9 Asignación de variables (SET)

La instrucción Set permite asignar un valor a una variable.

**Sintaxis**

```
SET
{ @local_variable
    [ . { property_name | field_name } ] = { expression | udt_name {
. | :: } method_name }
}
|
{ @SQLCLR_local_variable.mutator_method
}
|
{ @local_variable
    {+= | -= | *= | /= | %= | &= | ^= | |= } expression
}
|
  { @cursor_variable =
    { @cursor_variable | cursor_name
    | { CURSOR [ FORWARD_ONLY | SCROLL ]
        [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
        [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
        [ TYPE_WARNING ]
    FOR select_statement
        [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
      }
    }
}
```

*Tabla 15: Sintaxis de la instrucción Set*

Nota: La herramienta de visualización está utilizando el analizador sintáctico de SQL Server® 2014 por lo cual soporta todas las cláusulas incluidas en esta sintaxis.

## 3.2 Conceptos

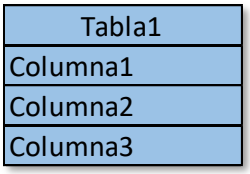
A continuación, se presentan los diferentes conceptos del lenguaje SQL que se incluyeron en el diseño de la notación y que se implementaron en la herramienta, así como las opciones que se analizaron para cada concepto y las consideraciones que tomaron en cuenta para decidir cuál opción era la más adecuada para representar cada concepto.

### 3.2.1 Tablas

Las tablas en el modelo relacional son las entidades que almacenan los datos. Están compuestas por columnas y dado que de manera natural una tabla se puede representar como una rejilla, se utilizó esa representación para graficarla. Las siguientes son las opciones analizaron para representar las tablas:

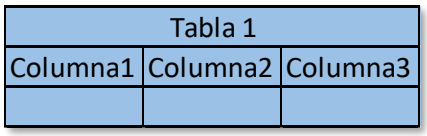
**Verticales**

En este caso se despliegan las columnas como una lista de valores.




**Horizontales**

Esta opción se asemeja a la forma en que los datos son representados en la base de datos, sin embargo, tiene el inconveniente de que para visualizar una tabla con muchas columnas se genera un diagrama sumamente extenso




**Horizontales con valores**

Esta opción es la que más se asemeja a la realidad, pero para efectos de visualizar y entender código SQL incluye mucha información irrelevante (i.e los valores) lo cual hace que la visualización se vea muy saturada y por el principio de manejo de la complejidad se desea evitar información que no es relevante.

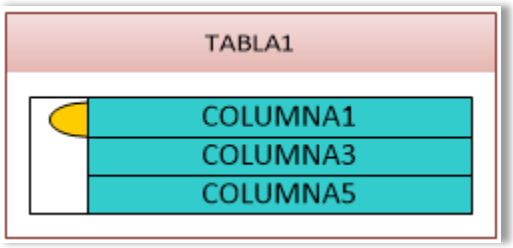
| Tabla 1 | | |
|---|---|---|
| Columna1 | Columna2 | Columna3 |
| 1 | A | 2/2/2017 |

**Implementación:**

La opción que se implemento fue la vertical porque:

1. Las columnas se representan horizontalmente hacia abajo para que el ancho total de la tabla no sea muy grande
2. Dado que los valores para efectos de la notación no agregan valor para comprender el funcionamiento del código SQL se eliminaron de la opción final.

El siguiente es un ejemplo como se ve una tabla en la herramienta de visualización:




### 3.2.2 Producto (Join)

Los operadores producto permiten devolver los registros que se encuentran asociados entre dos tablas. Para representar gráficamente el producto entre dos tablas se utilizaron líneas de conexión con flechas que indican el tipo de join. Para lograr una mayor expresividad visual (ver 2.3.6) se utilizaron diferentes colores lo cuales incrementan la distancia visual.

**Producto Interior (Inner Join)**

El operador producto interior permite devolver los registros que tienen valores idénticos en las columnas que se comparan entre dos tablas. La siguiente figura muestra mediante teoría de conjuntos la porción (en azul) de los datos que devuelve este operador:

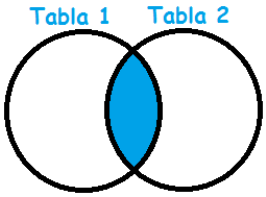



*Figura 8: Producto Interior (Inner Join)*

Ejemplo de la Implementación

Para la implementación se utilizaron flechas en ambos extremos de la conexión, como se muestra en el siguiente ejemplo:

```
SELECT * FROM Tabla1
INNER JOIN Tabla2 ON Tabla1.Columna1 = Tabla2.Columna1 AND Tabla1.Columna2 =
Tabla2.Columna2 AND Tabla1.Columna3 = Tabla2.Columna3
```

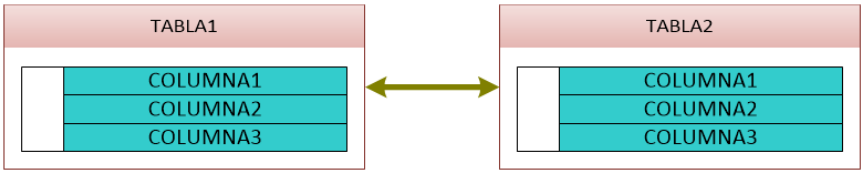



*Figura 9: Ejemplo de producto interior*

## Producto Completo (Full Join)

El operador producto completo permite devolver todos los registros que tienen dos tablas sin importar los valores de las columnas. La siguiente figura muestra mediante teoría de conjuntos la porción (en azul) de los datos que se devuelve este operador:

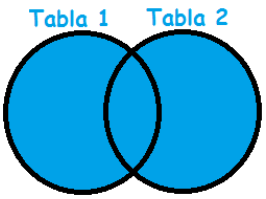



*Figura 10: Producto Completo (Full Join)*

Ejemplo de la Implementación

Para la implementación se utilizaron flechas dobles en ambos extremos de la conexión, como se muestra en el siguiente ejemplo:

```
SELECT * FROM Tabla1
FULL JOIN Tabla2 ON Tabla1.Columna1 = Tabla2.Columna1 AND Tabla1.Columna2 =
Tabla2.Columna2 AND Tabla1.Columna3 = Tabla2.Columna3
```

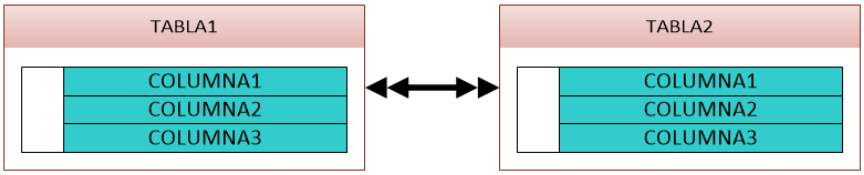



*Figura 11: Ejemplo de producto completo*

## Producto hacia la derecha (Right Join)

El operador producto hacia la derecha permite devolver todos los registros que tiene la tabla 2 sin importar si no hay registros con valores idénticos en la tabla 1. La siguiente figura muestra mediante teoría de conjuntos la porción (en azul) de los datos que se devuelve este operador:

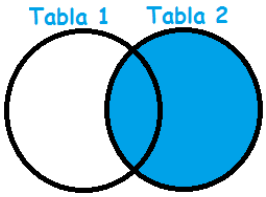



*Figura 12: Producto hacia la derecha (Right Join)*

### Ejemplo de la Implementación

Para la implementación se utilizó una flecha en el extremo derecho de la conexión, como se muestra en el siguiente ejemplo:

```
SELECT *
FROM Tabla1
RIGHT JOIN Tabla2 ON Tabla1.Columna1 = Tabla2.Columna1
       AND Tabla1.Columna2 = Tabla2.Columna2
       AND Tabla1.Columna3 = Tabla2.Columna3
```

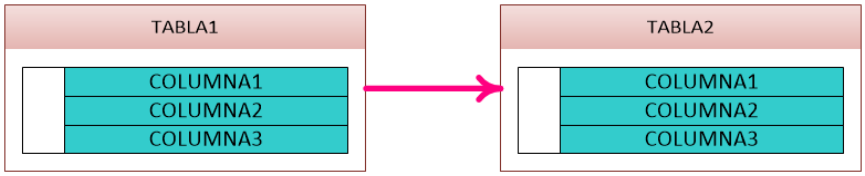



*Figura 13: Ejemplo de producto hacia la derecha*

## Producto hacia la Izquierda (Left Join)

El operador producto hacia la izquierda permite devolver todos los registros que tiene la tabla 1 sin importar si no hay registros con valores idénticos en la tabla 2. La siguiente figura muestra mediante teoría de conjuntos la porción (en azul) de los datos que se devuelve este operador:

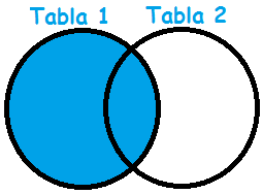



*Figura 14: Producto hacia la izquierda (Left Join)*

### Ejemplo de la Implementación

Para la implementación se utilizó una flecha en el extremo izquierdo de la conexión, como se muestra en el siguiente ejemplo:

```
SELECT *
FROM Tabla1
LEFT JOIN Tabla2 ON Tabla1.Columna1 = Tabla2.Columna1
       AND Tabla1.Columna2 = Tabla2.Columna2
       AND Tabla1.Columna3 = Tabla2.Columna3
```

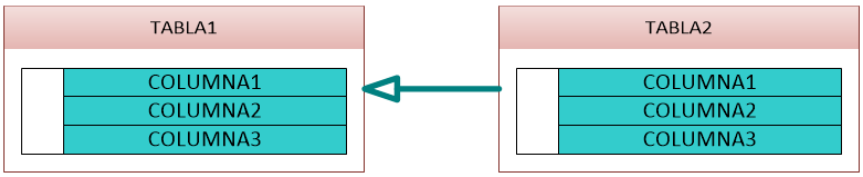
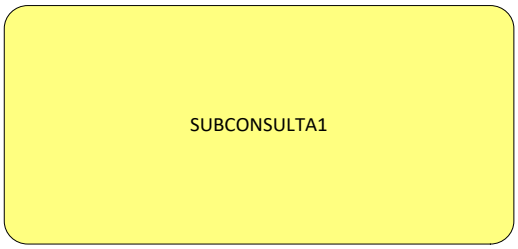



*Figura 15: Ejemplo de producto hacia la izquierda*

### 3.2.3 Sub Consultas

Una subconsulta es una consulta que está anidada dentro de otra instrucción SQL como lo puede ser un Update o Insert por ejemplo y que permite unificar varias tablas, vistas o inclusive otras subconsultas en una sola entidad de datos la cual puede ser utilizada como si fuera una vista.

Para representar gráficamente subconsultas se evaluaron varios tipos de figuras que pudieran funcionar como contenedores, por ejemplo:




Ejemplo de la Implementación

Para la implementación de las subconsultas se utilizó un contenedor de esquinas redondas que identifica y separa las tablas que pertenecen a la subconsulta y permite ver la integración de la consulta con otras entidades, como se muestra en el siguiente ejemplo:

```
SELECT Tabla1.Columna1, Subconsulta1.Columna3
FROM Tabla1
INNER JOIN
   (
          SELECT Columna3, Columna4
          FROM Tabla1
   ) Subconsulta1 ON Tabla1.Columna1 = Subconsulta1.Columna3;
```

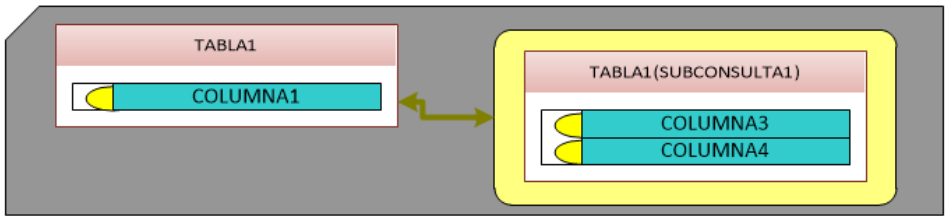



*Figura 16: Ejemplo de una subconsulta*

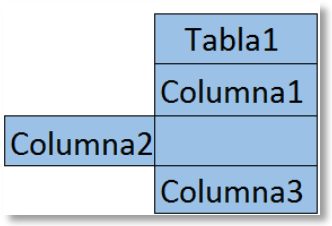
### 3.2.4 Selección (Select)

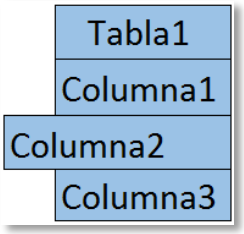
Para representar las columnas que forman parte de la cláusula "*Select*" se consideraron las siguientes opciones:

**Externo a la tabla**

La idea de esta opción es que se pueda visualizar las columnas como si estuvieran fuera de la tabla ya sea totalmente o parcialmente y así representar lo que sucede con las columnas del select que "extraen" la información de la consulta para hacerla disponible a otras instrucciones SQL o como resultados finales de la consulta, por ejemplo:

```
SELECT Columna1
FROM Tabla1
```

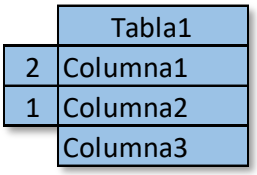



**Con orden**

En este caso se intenta representar el orden que tienen las columnas en la cláusula "*Select*", por ejemplo, en la siguiente consulta la columna2 se despliega primero que la columna1:

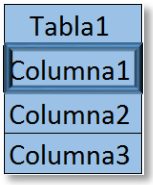
```
SELECT Columna2, Columna1
FROM Tabla1
```




**Encendido y apagado**

La idea con esta opción es representar las columnas en las tablas como botones que están encendidos o apagados de tal forma que las columnas que están encendidas son las que están incluidas en la selección, por ejemplo:

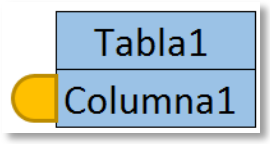
```
SELECT Columna1
FROM Tabla1
```




**Con figuras externas**

En este caso se agrega una figura externa a la columna para indicar que sale de la tabla, por ejemplo.

```
SELECT Columna1
FROM Tabla1
```




**Implementación:**

La opción seleccionada fue la externa con figuras, por cuanto:

1. Cumple con el principio de claridad semiótica (ver 2.3.1) al utilizar una figura única para representar columnas seleccionadas

2. Aumenta la distancia visual (ver 2.3.2.1) al utilizar la variable de color como diferenciador.

Ejemplo de la implementación:

Como se muestra en la Figura 17 las columnas C2 y C6 son parte de la cláusula "*Select*"

```
SELECT Columna1,Columna6
FROM Tabla2
WHERE Columna4 = 2
```

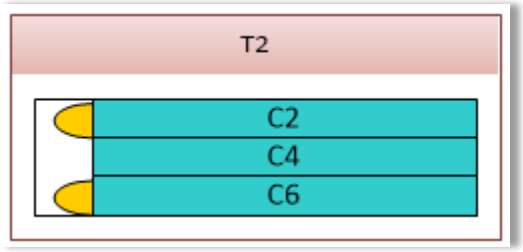


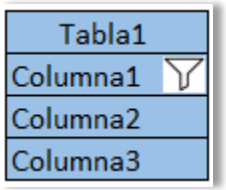

*Figura 17: Ejemplo de cláusula "Select"*

### 3.2.5 Calificación (Where)

Para representar las columnas que forman parte de la cláusula "*Where*" y que se encargan de filtrar los datos se consideraron las siguientes opciones:

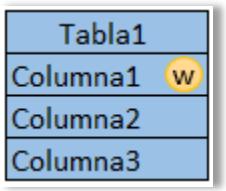
**Iconos**

La idea con esta opción era agregar un icono a las columnas que son parte de la cláusula "W*here*"




**Figuras con Texto**

Con esta opción se quiso agregar una figura que permitiera también desplegar texto para ayudar con la mnemotecnia de los elementos gráficos.




**Implementación:**

La opción seleccionada fue la de figuras con texto, por cuanto:

1. Cumple con el principio de codificación dual (ver 2.3.7) que hace más eficiente recordar una figura.
2. Aumenta la distancia visual (ver 2.3.2.1) al utilizar la variable de color, forma y posición horizontal como diferenciadores.

Ejemplo de la implementación:

Como se muestra en la siguiente figura la Columna3 es parte de la condición "*Where*":

```
SELECT Columna1, Column2
FROM Tabla1
WHERE Column3=1
```

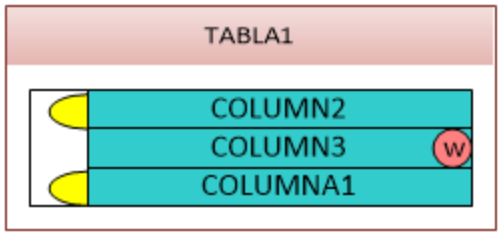


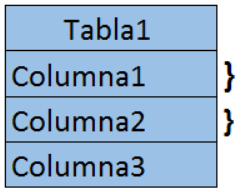

*Figura 18: Ejemplo de cláusula "Where"*

### 3.2.6 Agrupamiento (Group by)

Para representar las columnas que forman parte de la cláusula "*Group by*" se consideraron las siguientes opciones:

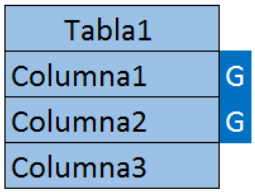
**Iconos**

Los iconos permiten diferenciar las columnas que están agrupando los datos




**Figuras con Texto**

Las figuras con texto incrementan la mnemotecnia de los elementos gráficos.




**Implementación:**

La opción seleccionada fue la de figuras con texto, por cuanto:

1. Cumple con el principio de codificación dual (ver 2.3.7) que hace más eficiente recordar una figura.
2. Aumenta la distancia visual (ver 2.3.2.1) al utilizar la variable de color, forma y posición horizontal como diferenciadores.

Ejemplo de la implementación:

Como se muestra en la siguiente figura la Columna1 y la Columna5 son parte del agrupamiento:

```
SELECT Columna1,Columna5,sum(*)
FROM Tabla1
GROUP BY Columna1, Columna5
```

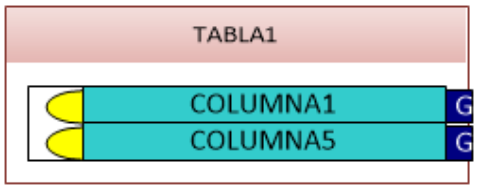
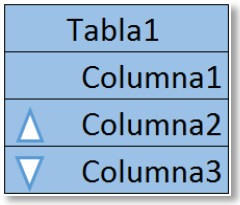



*Figura 19: Ejemplo de cláusula "Group by"*

### 3.2.7 Ordenamiento (Order by)

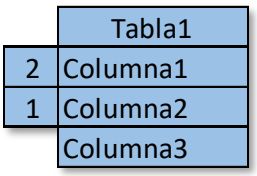
Para representar las columnas que forman parte de la cláusula "*Order by*" y que se encargan del ordenamiento de los datos se consideraron las siguientes opciones:

**Con Iconos**




**Con Números**

Los números permiten representar el orden de las columnas en la cláusula de ordenamiento




**Implementación**:

La opción que se implementó fue la de iconos porque:

1. Permite identificar si el orden es ascendente o descendente de manera visual
2. Por el principio transparencia semántica (ver 2.3.3) ayuda a ver fácilmente que se trata de ordenamiento
3. Lo opción de números va en contra del principio de manejo de la complejidad (ver 2.3.4)

Ejemplo de la implementación:

Como se muestra en la Figura 20: Ejemplo de Cláusula "*Order by*" las columnas Columna2 y Columna3 son parte de la cláusula de ordenamiento:

```
SELECT Columna1
FROM Tabla1
ORDER BY Columna2 Asc, Columna3 Desc
```

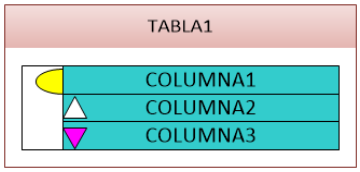



*Figura 20: Ejemplo de Cláusula "Order by"*

### 3.2.8 Uniones (Union)

El operador unión permite combinar y devolver los registros de dos o más consultas como si fuera un solo conjunto de datos. Para poder utilizar este operador se requiere que las consultas a unir tengan la misma cantidad de columnas y que el tipo de datos de una columna sea igual a su columna equivalente en la otra consulta. La siguiente figura muestra mediante teoría de conjuntos la porción (en azul) de los datos que se devuelve este operador:

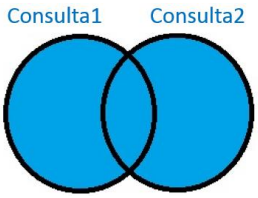
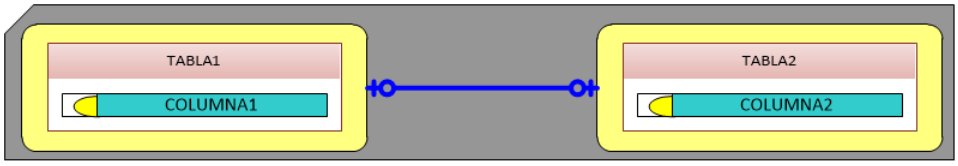



*Figura 21: Unión (union)*

Ejemplo de la Implementación

Para la implementación se utilizó una línea de conexión con símbolos “+” en los extremos para incrementar la transparencia semántica (ver 2.3.3) y representar la unión de los datos, como se muestra en el siguiente ejemplo:

```
SELECT Columna1
FROM Tabla1
UNION
SELECT Columna2
FROM Tabla2;
```



### 3.2.9 Intersecciones (intersec)

El operador intersección permite combinar y devolver los registros que coinciden en dos o más consultas como si fuera un solo conjunto de datos. Para poder utilizar este operador se requiere que las consultas a unir tengan la misma cantidad de columnas y que el tipo de datos de una columna sea igual a su columna equivalente en la otra consulta. La siguiente figura muestra mediante teoría de conjuntos la porción (en azul) de los datos que se devuelve este operador:

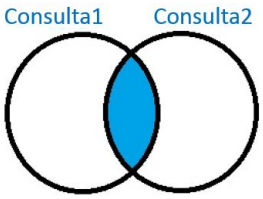
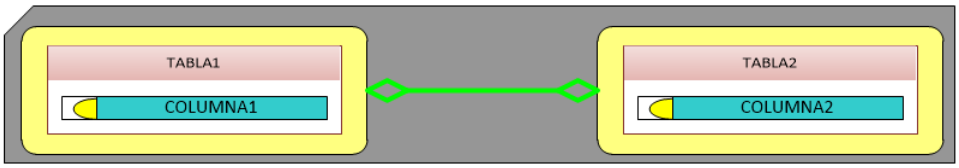



*Figura 22: Intersección (intersec)*

Ejemplo de la Implementación

Para la implementación se utilizó una línea de conexión con símbolos "◊" en los extremos para aumentar la distancia visual con otros operadores, como se muestra en el siguiente ejemplo:

```
SELECT Columna1
FROM Tabla1
INTERSECT
SELECT Columna2
FROM Tabla2;
```




### 3.2.10 Diferencia (Except)

El operador diferencia permite combinar los registros de la primera consulta que no coinciden con los registros de la segunda y devuelve el resultado en un solo conjunto de datos. Para poder utilizar este operador se requiere que las consultas a unir tengan la misma cantidad de columnas y que el tipo de datos de una columna sea equivalente a la columna que ocupa la misma posición

en la otra consulta. La siguiente figura muestra mediante teoría de conjuntos la porción (en azul) de los datos que se devuelve este operador:

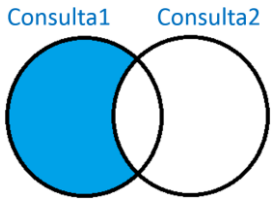
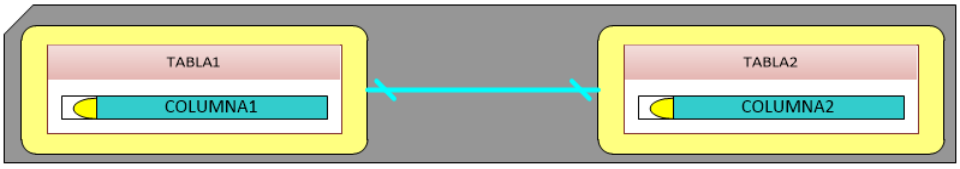



*Figura 23: Diferencia (except)*

Ejemplo de la Implementación

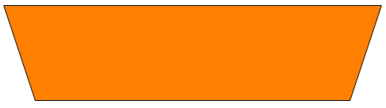
Para la implementación se utilizó una línea de conexión con símbolos "\" en los extremos para incrementar la transparencia semántica (ver 2.3.3) y representar eliminación de datos, como se muestra en el siguiente ejemplo:

```
SELECT Columna1
FROM Tabla1
EXCEPT
SELECT Columna2
FROM Tabla2;
```



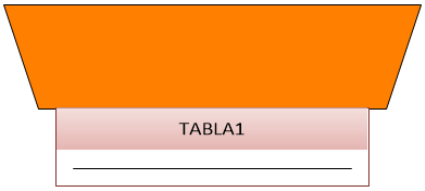

### 3.2.11 Creación (Insert)

La instrucción Insert permite agregar uno o más registros en una tabla o vista. Esta instrucción puede insertar valores individuales o resultados de otras consultas (ver 3.1.3). Para graficar la instrucción "Insert" se utilizó la figura de un trapecio invertido para representar un embudo:



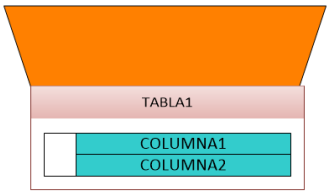
y en su parte inferior se colocó la tabla en la que se van a insertar los datos, por ejemplo:

```
INSERT INTO Tabla1
VALUES ('A', '20080414');
```



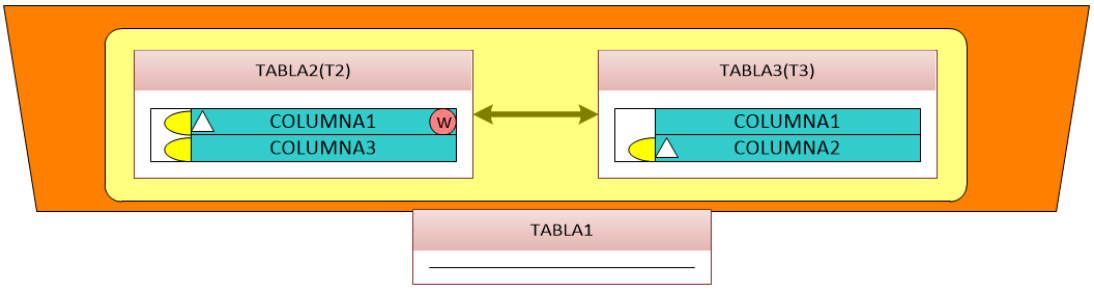

O inclusive con nombres de columnas:

```
INSERT INTO Tabla1 (Columna1, Columna2)
VALUES ('A', '20080414');
```




con lo cual se pretende representar que todo lo que se muestra dentro del trapecio (“embudo”) se va a procesar e insertar en la tabla destino y así soportar el principio de transparencia semántica (ver 2.3.3), por ejemplo:

```
INSERT INTO Tabla1
    SELECT 'A', T2.Columna1, T3.Columna2, T2.Columna3
    FROM Tabla2 AS T2
    INNER JOIN Tabla3 AS T3
        ON T2.Columna1 = T3.Columna1
    WHERE T2.Columna1 LIKE '2%'
    ORDER BY T2.Columna1, T3.Columna2;
```

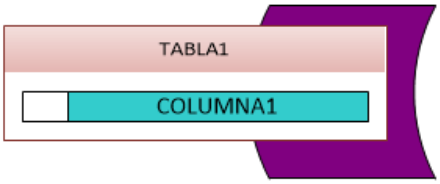


### 3.2.12 Actualización (Update)

La instrucción “Update” permite modificar uno o más registros en una tabla o vista (ver 3.1.4). En cada registro permite modificar una o más columnas. Esta instrucción puede modificar las columnas con valores literales, valores de otras columnas o de otras tablas. Para graficar la instrucción “Update” se utilizó la figura de datos externos de los diagramas de flujo para representar un túnel:

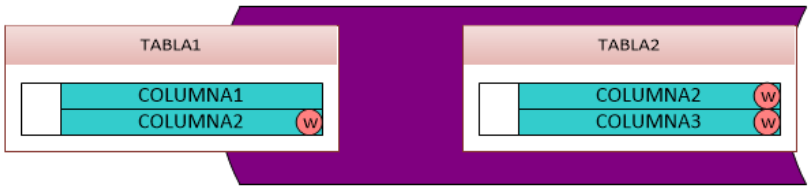
y en su parte izquierda se colocó la tabla en la que se van a actualizar los datos, por ejemplo:

```
UPDATE Tabla1 SET Columna1 = 25
```



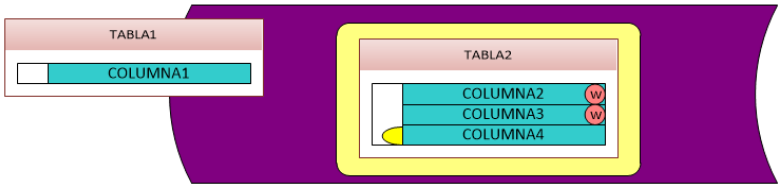

con lo cual se pretende representar que todo lo que se muestra dentro del túnel se va a procesar y actualizar la tabla destino y de esta forma soportar el principio de transparencia semántica (ver 2.3.3), por ejemplo:

```
UPDATE Tabla1
   SET Columna1 = Tabla2.Columna4 * .90
  FROM Tabla2
 WHERE Tabla2.Columna2 = Tabla1.Columna2
   AND Tabla2.Columna3 = 'A'
```




O con subconsultas:

```
UPDATE Tabla1
   SET Columna1 =
       (SELECT (Columna4 * .90)
          FROM Tabla2
         WHERE Tabla2.Columna2 = Tabla1.Columna2
           AND Tabla2.Columna3 = 'B')
```

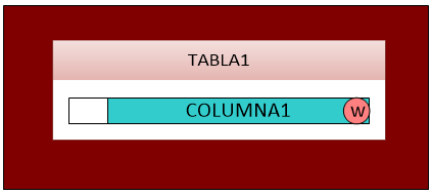



### 3.2.13 Borrado (Delete)

La instrucción “Delete” permite borrar uno o más registros en una tabla o vista. Para graficar la instrucción “Delete” se utilizó una figura de un rectángulo de color rojo para representar por medio del color rojo la instrucción la cual es riesgosa porque elimina datos:

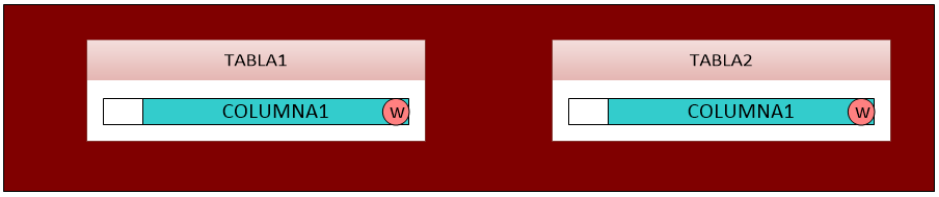
Dentro del rectángulo se colocó la tabla que contiene los datos a borrar, por ejemplo:

```
DELETE FROM Tabla1
WHERE Columna1 = 'Dato1'
```




Si la instrucción contiene otras tablas, estas también se representan dentro del rectángulo:

```
DELETE Tabla1 FROM Tabla2
WHERE Tabla1.Columna1 = Tabla2.Columna1
  AND Tabla1.Columna1 = 'A'
```



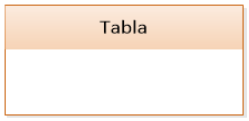
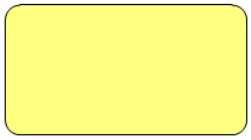
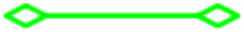
### 3.2.14 Notación Visual

La notación desarrollada en esta investigación incorpora todos los conceptos descritos en las secciones anteriores de modo que la integración de los mismos permite visualizar instrucciones SQL en diagramas de Microsoft Visio®

Las tablas: Tabla 16, Tabla 17 y Tabla 18 muestran cada elemento de la notación con los valores de las variables visuales que fueron asignados a ese elemento de manera que se pueda comparar la distancia visual y los pasos perceptibles entre los elementos:

**Conceptos Generales**

Esta sección describe los conceptos del lenguaje SQL implementados en la notación y los valores asignados para cada variable visual:

| Concepto | Figura | Color | Textura | Tamaño | Orientación | Brillo |
|---|---|---|---|---|---|---|
| Tabla |   | RGB(248,222,200) | Sólida | Mediano | Vertical y Horizontal | Medio |
| Subconsulta |  | RGB(255,255,128) | Sólida | Grande | Vertical y Horizontal | Medio |
| Unión |  | RGB(0,0,255) | Sólida | Mediano | Horizontal | Medio |
| Diferencia (Except) |  | RGB(0,255,255) | Sólida | Mediano | Horizontal | Alto |
| Intersección (Intersec) |  | RGB(0,255,0) | Sólida | Mediano | Horizontal | Alto |
| Producto |  | RGB(0,0,0) | Sólida | Mediano | Horizontal | Bajo |

| | | | | | | |
|---|---|---|---|---|---|---|
| (Full Join) | | | | | | |
| Producto (Right Join) |  | RGB(255,0,128) | Sólida | Mediano | Horizontal | Alto |
| Producto (Left Join) |  | RGB(0,128,128) | Sólida | Mediano | Horizontal | Medio |
| Producto (Inner Join) |  | RGB(128,128,0) | Sólida | Mediano | Horizontal | Medio |

*Tabla 16: Notación Visual - Conceptos Generales*

## Instrucciones SQL

Esta sección describe las instrucciones del lenguaje SQL implementados en la notación y los valores asignados para cada variable visual:

| Concepto | Figura | Color | Textura | Tamaño | Orientación | Brillo |
|---|---|---|---|---|---|---|
| Consulta |  | RGB(128,128,128) | Sólida | Grande | Vertical y Horizontal | Bajo |
| Creación (Insert) |  | RGB(255,128,0) | Sólida | Grande | Vertical y Horizontal | Medio |
| Actualización (Update) |  | RGB(128,0,128) | Sólida | Grande | Vertical y Horizontal | Medio |
| Borrado (Delete) |  | RGB(128,0,0) | Sólida | Grande | Vertical y Horizontal | Medio |

*Tabla 17: Notación Visual - Instrucciones SQL*

**Atributos de columnas**

Esta sección describe los conceptos del lenguaje SQL implementados en la notación y los valores asignados para cada variable visual:

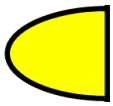
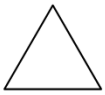
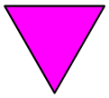
| Concepto | Figura | Color | Textura | Tamaño | Orientación | Brillo |
|---|---|---|---|---|---|---|
| Selección (Select) |  | RGB(255,255,0) | Sólida | Pequeño | Horizontal | Alto |
| Calificación (Where) |   | RGB(255,128,128) | Sólida | Pequeño | Horizontal | Medio |
| Agrupamiento (Group by) |   | RGB(0,0,128) | Sólida | Pequeño | Horizontal | Medio |
| Ordenamiento (order by ascending) |  | RGB(255,255,255) | Sólida | Pequeño | Horizontal | Alto |
| Ordenamiento (order by descending) |  | RGB(255,0,255) | Sólida | Pequeño | Horizontal | Alto |

*Tabla 18: Notación Visual - Atributos de columna*

## 3.3 Visualización de código SQL

Para visualizar el código SQL se requieren diferentes niveles de abstracción que permitan entenderlo más fácilmente. En algunas ocasiones un diagrama de alto nivel es la mejor alternativa para entender el funcionamiento de un procedimiento almacenado y en otras ocasiones se requiere mayor nivel de detalle para entender aspectos específicos del código. Los siguientes niveles de abstracción permiten llenar esas necesidades.

### 3.3.1 Nivel 01: Interacción de procedimientos almacenados y entidades de datos

Este nivel de visualización muestra la interacción entre procedimientos almacenados y entidades de datos (e.g tablas), incluyendo las operaciones (e.g Insert, Select, etc.) que realiza cada procedimiento sobre cada tabla.

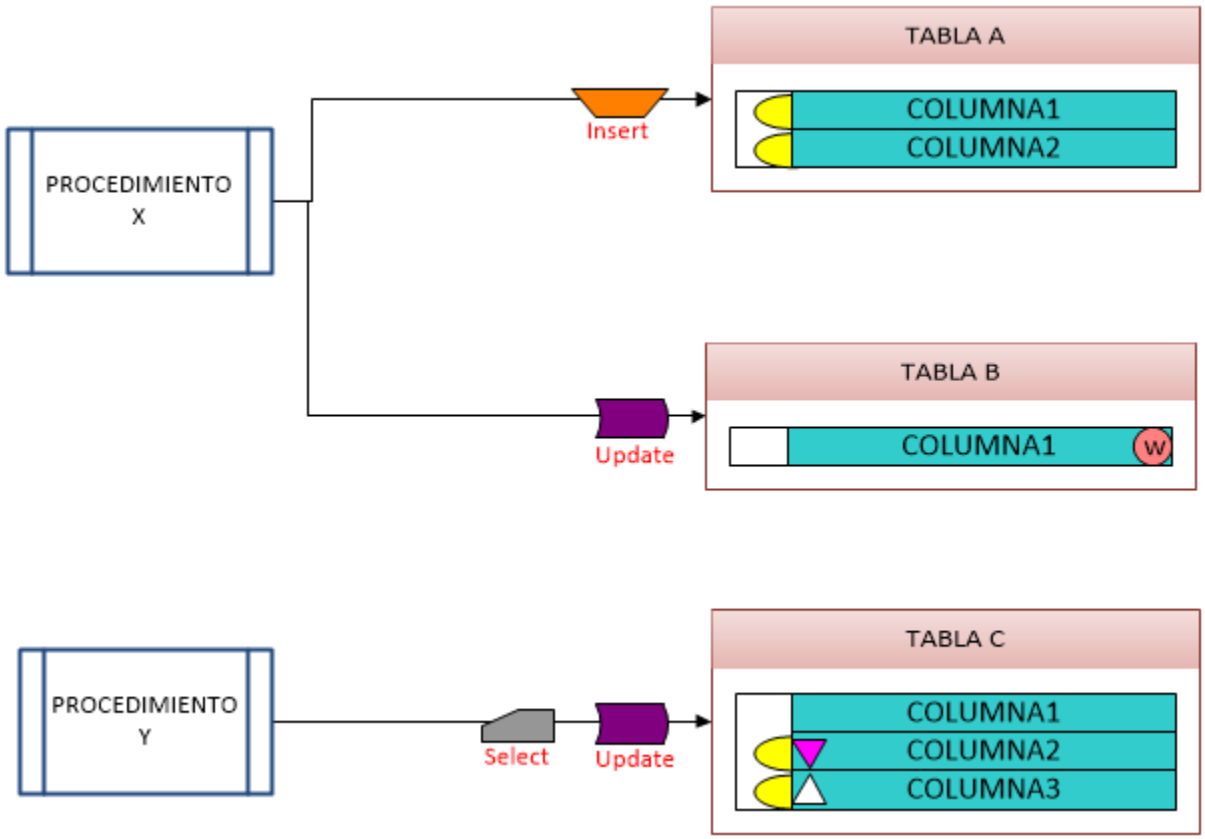



*Figura 24: Interacción de procedimientos y entidades de datos*

### 3.3.2 Nivel 02: Expansión de procedimientos almacenados:

Este nivel muestra las instrucciones que componen un procedimiento almacenado, así como las entidades de datos (e.g tablas) con sus columnas y atributos. La Figura 25 presenta una parte del diagrama generado por la herramienta de visualización para un procedimiento almacenado de ejemplo, en el cual se puede observar que hay 4 instrucciones “Select”, 2 “Insert”, 2 “Update” y 2 “Delete”. En la herramienta de visualización se puede utilizar la capacidad de “Zoom” para ver los detalles de las instrucciones (tablas, columnas, atributos, “Joins”, etc.).

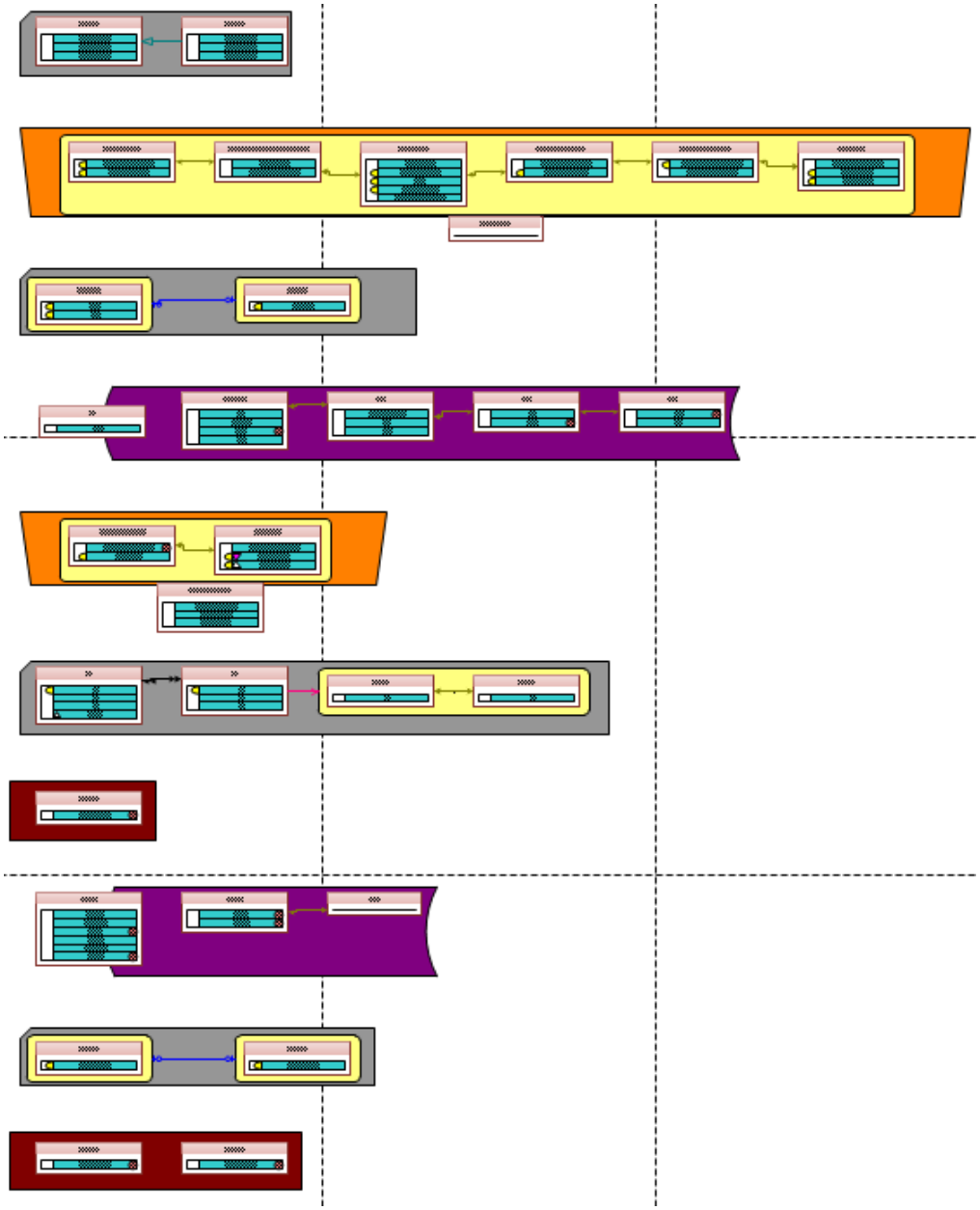
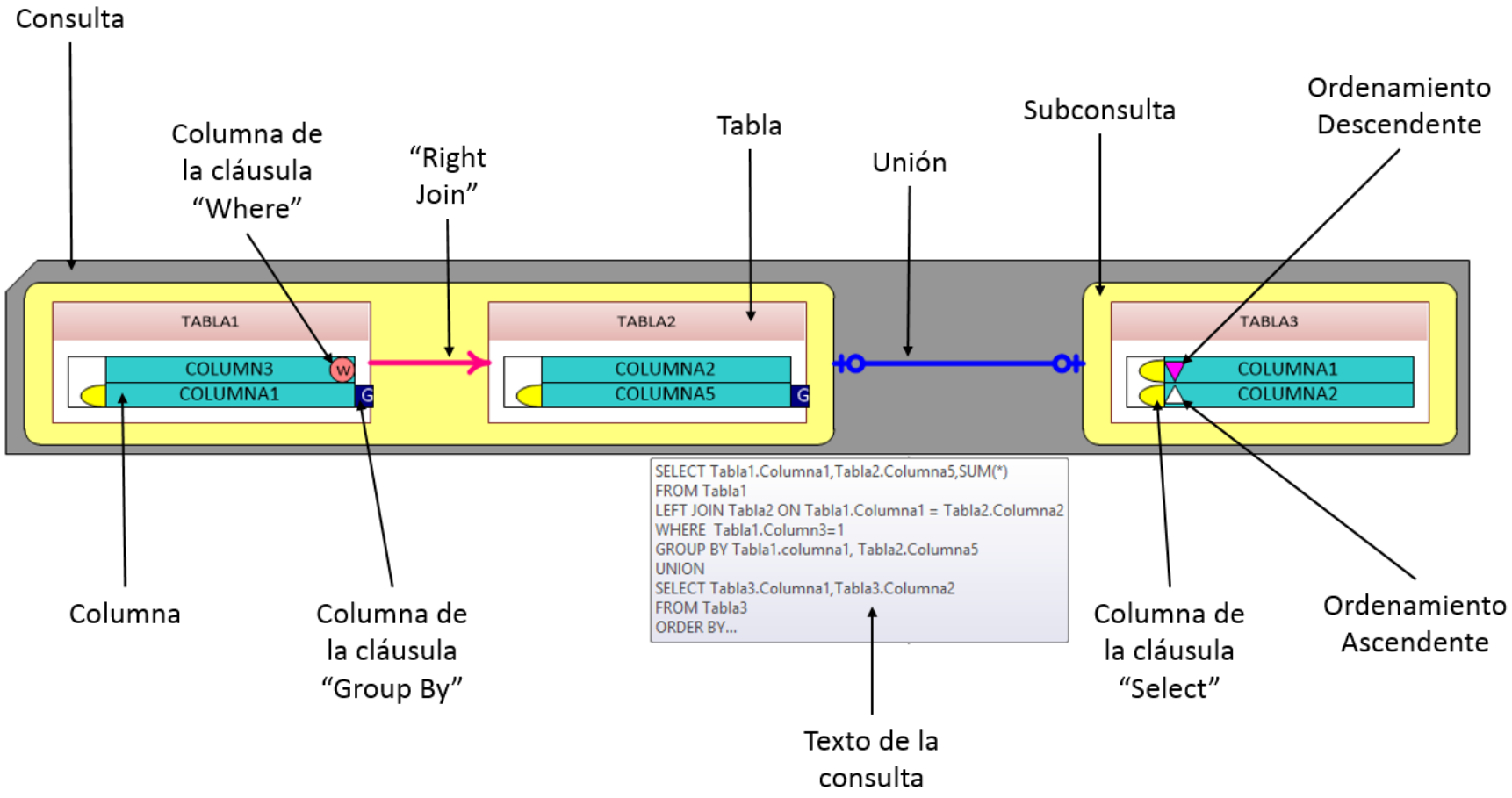


*Figura 25: Visualización de procedimientos almacenados*

### 3.3.3 Visualización de consultas SQL

Este nivel permite visualizar cada consulta con todos los detalles que se soporta la notación visual, por ejemplo, esta consulta:

```
SELECT Tabla1.Columna1,Tabla2.Columna5,SUM(*) FROM Tabla1 LEFT JOIN Tabla2 ON Tabla1.Columna1 = Tabla2.Columna2
 WHERE Tabla1.Column3 = 1
 GROUP BY Tabla1.columna1, Tabla2.Columna5
UNION
SELECT Tabla3.Columna1,Tabla3.Columna2 FROM Tabla3
ORDER BY Tabla3.columna1 DESC, Tabla3.columna2
```



## 3.4 La herramienta de visualización

Como resultado final de esta investigación se desarrolló una herramienta para visualizar código SQL la cual permite graficar los conceptos desarrollados en la notación (ver 3.2). Esta herramienta se desarrolló en el lenguaje de programación C# dentro del ambiente de programación Visual Studio 2015.

La herramienta tiene tres componentes principales:

1. Analizador sintáctico de código SQL
2. Plataforma de visualización
3. Representación intermedia para integrar los resultados del analizador y poder visualizarlos en la plataforma de visualización.

Estos componentes funcionan de la siguiente forma:

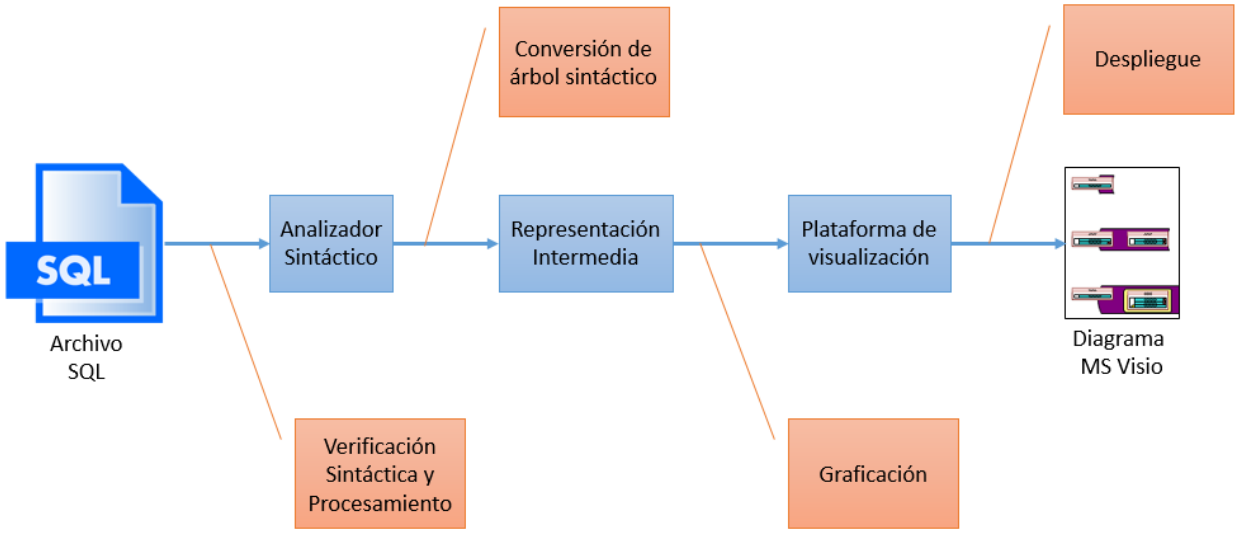



*Figura 26: Componentes de la herramienta de visualización*

La Figura 27 muestra una imagen real de la herramienta de visualización:

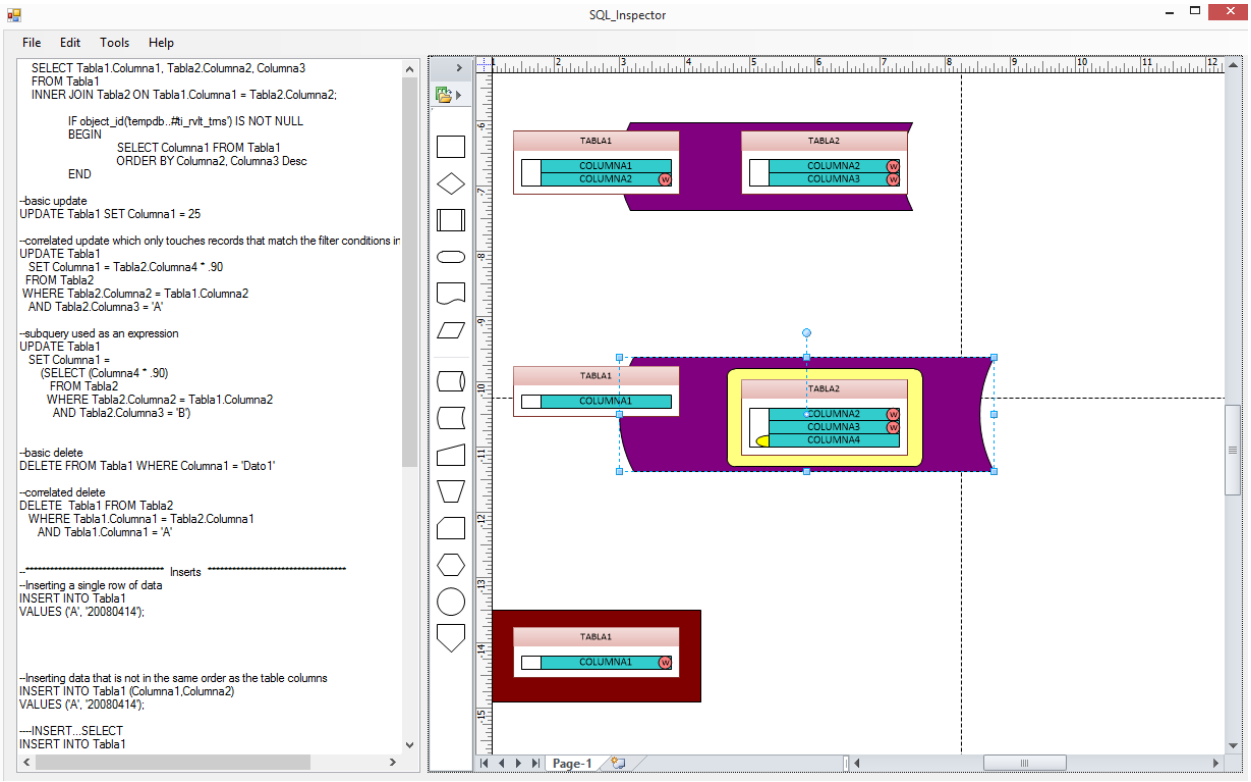



*Figura 27: Herramienta de visualización*

### 3.4.1 El analizador sintáctico

Para analizar y procesar el código SQL se utilizó el analizador sintáctico de SQL Server® el cual permitió verificar la sintaxis del código SQL antes de visualizarlo, de esta forma solo se grafican instrucciones que son correctas sintácticamente. Este analizador es el mismo que utilizan las herramientas de SQL Server® como “SQL Server Management Studio” para verificar la sintaxis del código SQL que escriben los programadores. Una vez que el código ha sido verificado sintácticamente el analizador genera un árbol sintáctico. Este árbol sintáctico contiene la información de todas las instrucciones presentes en el código, así como las cláusulas y atributos de cada instrucción. La Figura 28, Figura 29 y Figura 30 muestran algunas partes del árbol sintáctico de una consulta:

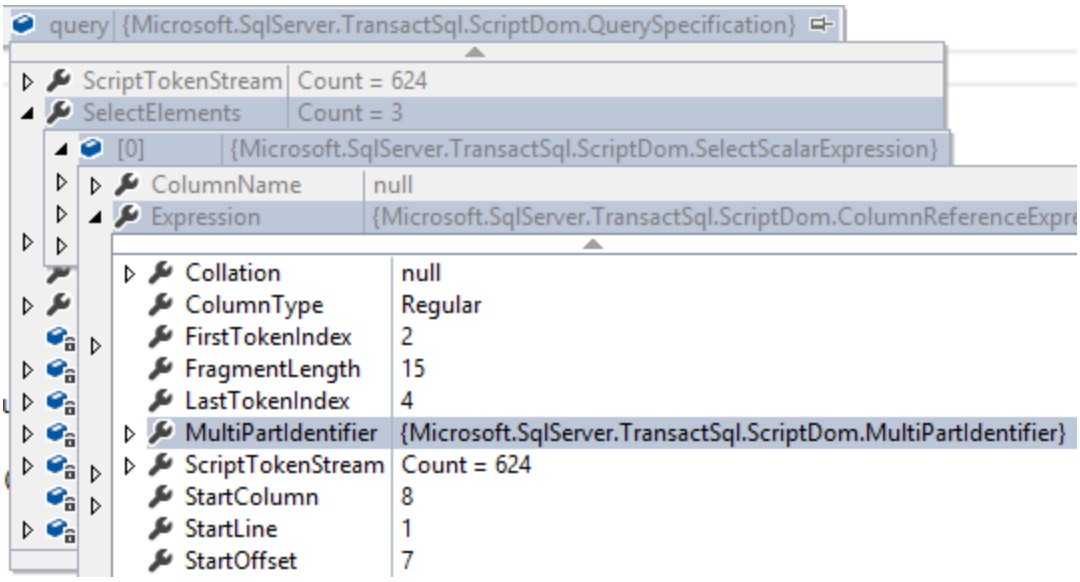



*Figura 28: Ejemplo árbol sintáctico - cláusula "Select"*

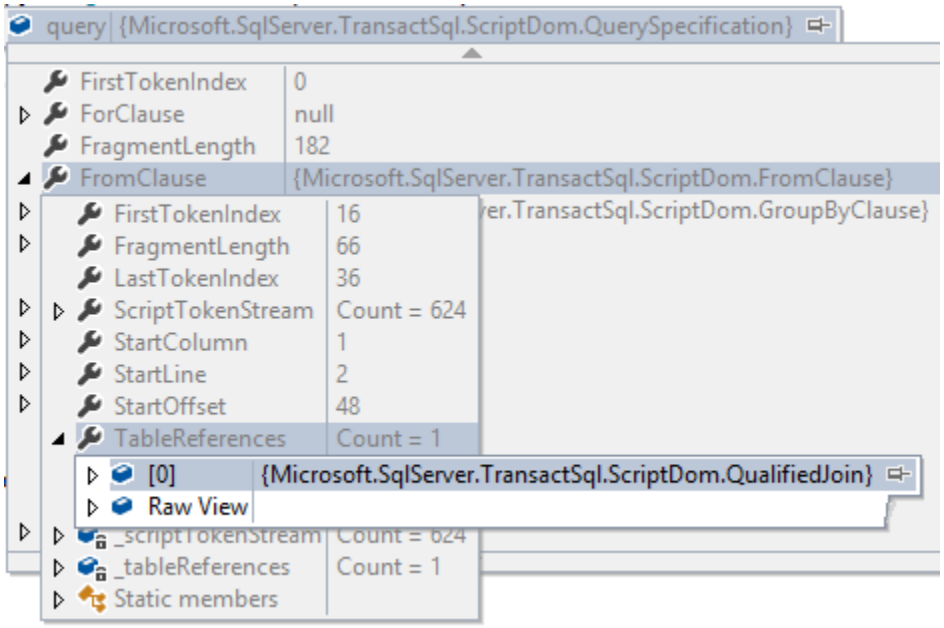



*Figura 29: Ejemplo árbol sintáctico - cláusula "From"*

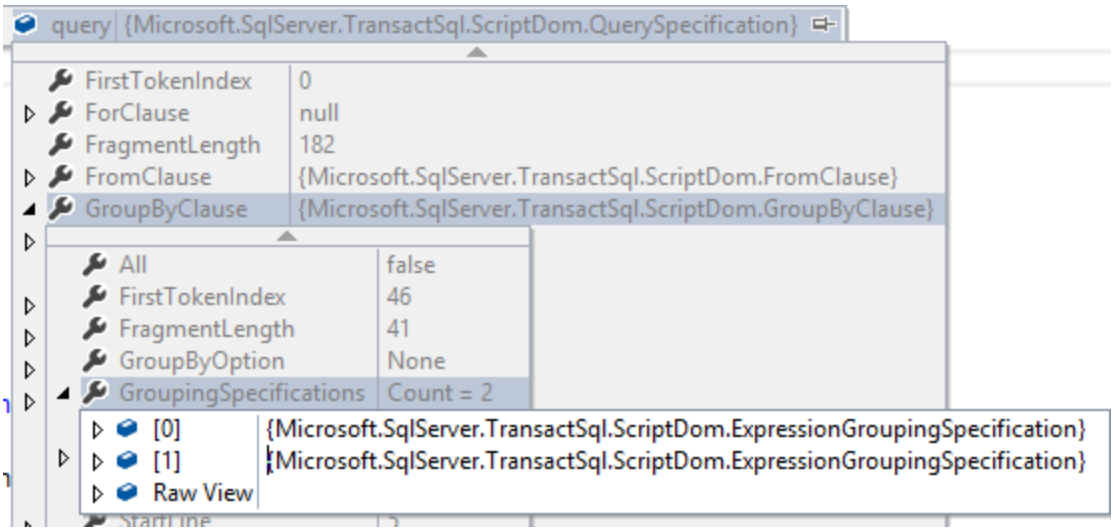
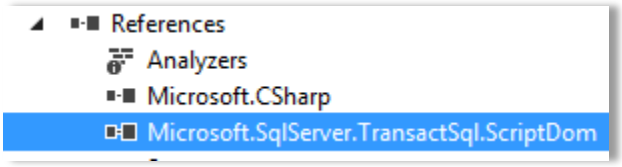



*Figura 30: Ejemplo árbol sintáctico - cláusula "Group by"*

La herramienta de visualización recorre el árbol sintáctico mediante una clase visitante (ver el patrón de diseño visitante 0). En este proyecto se utilizó la versión 2014 del analizador de SQL Server® desde un programa escrito en C#.

Para poder utilizar el analizador sintáctico desde el programa de C# se requirió:

1. incluir una referencia a la biblioteca ScriptDom:




2. Adicionalmente se requiere incluir una instrucción (“using”) para hacer referencia al espacio de nombres (“namespace”) que contiene todas las clases del analizador:

```
using Microsoft.SqlServer.TransactSql.ScriptDom;
```

3. Una vez que se tiene acceso a la funcionalidad del analizador se requiere declarar e inicializar una variable de tipo `TSql120Parser:`

```
TSql120Parser SQLParser = new TSql120Parser(true);
```

4. Se puede analizar el código SQL mediante el método “parse”, el cual requiere como parámetro de entrada una secuencia de caracteres (“streamreader”) para poder leer el código desde un archivo:

```
try
{   // Open the text file using a stream reader.
    using (StreamReader sr = new StreamReader(this.fileName))
    {
        sqlFragment = SQLParser.Parse(sr, out parseErrors);
        if (parseErrors.Count > 0)
        {
            System.Console.WriteLine("Error: the parser failed, please check syntax errors");
        }
    }
}
```

5. Recorrer el árbol sintáctico generado para el fragmento (“`sqlFragment`”) de código que retorna el analizador (“parser”), el cual puede ser recorrido mediante una clase (“visitor”) que implementa el patrón de diseño visitante (ver 0)
6. Más detalles sobre la implementación del analizador sintáctico se pueden ver en el anexo 8.1

### 3.4.2 Representación intermedia

Para poder integrar los resultados del analizador sintáctico y la plataforma de graficación sin tener que depender de las implementaciones específicas de cada plataforma se desarrolló un conjunto de clases que funcionan como una interfaz entre ambas implementaciones (analizador sintáctico y la plataforma de visualización). Estas clases abstraen la funcionalidad requerida tanto por el analizador como por el graficador. Así por ejemplo, la clase join permite representar uniones entre tablas (left, right, inner, full joins) o entre consultas (union, intersect, except).

Las clases que forman parte de la representación intermedia son:

- Una clase base (abstracta) para representar instrucciones SQL
- Clases derivadas para cada tipo de instrucción SQL implementada en esta investigación.
- Clases para representar los conceptos de SQL como: Tablas, Columnas, Joins

Más detalles sobre la implementación de estas clases se pueden ver en el anexo 8.3

### 3.4.3 La plataforma de visualización

En el desarrollo de este proyecto la capacidad de visualización era un aspecto sumamente importante por lo cual se analizaron dos plataformas de visualización para seleccionar la plataforma que presentaba las mayores capacidades y facilidades para graficar. Las plataformas que se analizaron fueron:

- NShape: https://www.dataweb.de/en/products/diagramming.html
- Microsoft Visio® https://products.office.com/en-us/visio/flowchart-software?tab=tabs-1

**NShape**

El desarrollo del proyecto se inició con la plataforma NShape. La cual es una plataforma que permite:

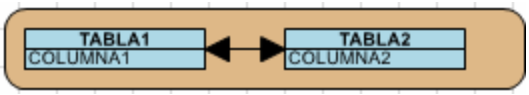
- Visualización de diagramas con muchos, inclusive miles de elementos
- Desarrollo de nuevas figuras personalizadas
- Soporte para colores, estilos, letras, etc.

Sin embargo, después de realizar la visualización de consultas básicas se determinó que la plataforma tenía las siguientes limitaciones:

1. La creación de figuras se hacía con base en instancias de meta figuras que hacía difícil la inclusión de nuevas figuras.
2. La plataforma no soportaba el concepto de contenedores el cual era muy importante para la graficación de tablas las cuales tienen muchos elementos pequeños que requieren se unificados dentro del contenedor.
3. Existía poca documentación de ejemplos sobre aspectos específicos de graficación como cambio de colores, cambio de tamaño de figuras, etc.

Los siguientes son ejemplos de consultas visualizadas con la plataforma NShape:

```
SELECT Tabla1.Columna1, Tabla2.Columna2, Columna3
FROM Tabla1
INNER JOIN Tabla2;
```




```
SELECT Tabla1.Columna1, Subconsulta1.Columna3
FROM Tabla1
INNER JOIN
(
	SELECT Columna3 , Columna4
	FROM Tabla1
) Subconsulta1 ON Tabla1.Columna1 = Subconsulta1.Columna3;
```

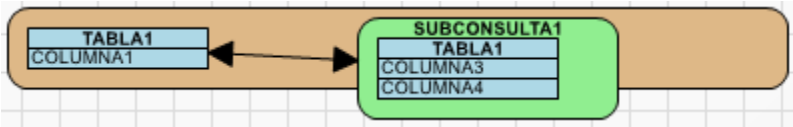



*Figura 31: Ejemplos de consultas visualizadas con NShape*

## Microsoft® Visio®

La plataforma de visualización Microsoft® Visio® permitió graficar todos los elementos especificados en la notación por lo cual fue la plataforma que se escogió para el proyecto. Uno de los aspectos que pesó más en la decisión fue la documentación que se puede encontrar en internet sobre aspectos y dudas específicas de graficación los cuales son críticos en el proceso de programación.

Algunas de las características que se utilizaron con esta plataforma son:

### Figuras

Para poder representar las instrucciones SQL se utilizaron diferentes figuras (“Shapes”) de Microsoft® Visio®. Para poder utilizar una figura se requiere agregarlas a la página actual de Visio lo cual se puede realizar de dos formas:

- Dibujando figuras geométricas como rectángulos, por ejemplo

```
Shape shape1 = currentPage.DrawRectangle(table.X1, table.Y1, table.X2, table.Y2);
```

- Utilizando figuras predefinidas en Visio, por ejemplo

```
updateTemplate = flowChartStencil.Masters["External Data"];
```

```
shape1 = currentPage.Drop(updateTemplate, x, y);
```

**Colores**

Para cambiar colores en Visio se pueden utilizar diferentes métodos como RGB, CMYK, colores del sistema. Dado la flexibilidad que permite el sistema RGB (los colores se pueden definir numéricamente) se escogió este sistema para implementar la herramienta.

Lo primero que hay escoger es el código del color:

```
private const string UpdateColor = "RGB(255,204,153)";
```

Y posteriormente asignarlo a la figura mediante el método "**get_CellsSRC**", y utilizar como parámetro el valor **visFillForegnd** para indicar que el color se va a utilizar para el relleno de la figura.

```
//set the shape back color
shape1.get_CellsSRC((short)VisSectionIndices.visSectionObject, (short)VisRowIndices.visRowFill,
    (short)VisCellIndices.visFillForegnd).FormulaU = UpdateColor;
```

Para que la distancia visual entre los distintos elementos que se diseñaron en la notación visual fuera mayor (mayor número de pasos perceptibles) se utilizaron tres valores de la escala de RGB (Red, Green, Blue) para cada color:

- 0 = inicio de la escala
- 128 = mitad de la escala
- 255 = final de la escala

Con cada uno de estos valores se creó la siguiente tabla de combinaciones, y se asignó a cada elemento un color de la tabla:

| R | G | B | Color | |
|---|---|---|---|---|
| 0 | 0 | 0 | | Full Join |
| 128 | 0 | 0 | | Delete |

| R | G | B | Color | |
|---|---|---|---|---|
| 0 | 0 | 128 | | Group by |
| 128 | 0 | 128 | | Update |

| R | G | B | Color | |
|---|---|---|---|---|
| 0 | 0 | 255 | | Union |
| 128 | 0 | 255 | | Set |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 255 | 0 | 0 | | Else | 255 | 0 | 128 | | Right Join | 255 | 0 | 255 | | Descending |
| 0 | 128 | 0 | | IF | 0 | 128 | 128 | | Left Join | 0 | 128 | 255 | | |
| 128 | 128 | 0 | | Inner Join | 128 | 128 | 128 | | Query | 128 | 128 | 255 | | Exec |
| 255 | 128 | 0 | | Insert | 255 | 128 | 128 | | Where | 255 | 128 | 255 | | |
| 0 | 255 | 0 | | Intersect | 0 | 255 | 128 | | Declare | 0 | 255 | 255 | | Except |
| 128 | 255 | 0 | | | 128 | 255 | 128 | | | 128 | 255 | 255 | | |
| 255 | 255 | 0 | | Select | 255 | 255 | 128 | | Sub Query | 255 | 255 | 255 | | Ascending |

*Tabla 19: Colores utilizados en la visualización*

### Tamaños

Para cambiar el tamaño de una figura se requiere utilizar el método **"resize"** de la figura e indicar la dirección y la cantidad de unidades que se va a ampliar

```
shape1.Resize(VisResizeDirection.visResizeDirS, Update.Y1 - Update.Y2, VisUnitCodes.visInches);
shape1.Resize(VisResizeDirection.visResizeDirE, Update.X2 - Update.X1, VisUnitCodes.visInches);
```

### Unir formas

Para poder conectar formas se requiere crear un conector el cual típicamente es una línea que está unida en sus extremos a dos figuras, en la herramienta de visualización la conexión se realizó mediante el método `VisioUtils.ConnectWithDynamicGlueAndConnector`:

```
Microsoft.Office.Interop.Visio.Shape connector =
    VisioUtils.ConnectWithDynamicGlueAndConnector(
        (Microsoft.Office.Interop.Visio.Shape)shapesArray[join.First.Key],  //Source Shape
        (Microsoft.Office.Interop.Visio.Shape)shapesArray[join.Second.Key]); //Target Shape
```

### Contenedores

Algunos conceptos como las tablas requieren de otras figuras pequeñas para representar columnas y sus atributos. Para este tipo de construcciones es deseable que todos estos elementos pequeños se puedan agrupar como una sola figura, lo cual en Visio se puede realizar por medio de los contenedores que son elementos que permiten incluir otras figuras dentro de sí.

El siguiente código muestra como agregar una figura a un contenedor:

```
containerShape.ContainerProperties.AddMember(shape1,
    VisMemberAddOptions.visMemberAddExpandContainer);
```

# 4 Resultados y Validación

Para validar que con el trabajo realizado en esta investigación se confirma la hipótesis planteada al inicio, la cual es:

> ***"Utilizando una notación visual para representar sentencias SQL es posible mejorar la capacidad de análisis de procedimientos almacenados, reduciendo el tiempo necesario para comprender el código y produciendo menos errores de comprensión".***

Se realizaron dos tipos de análisis:

1. Análisis cuantitativo
2. Análisis estadístico

## 4.1 Análisis cuantitativo

Este análisis consistió en aplicar una prueba a ingenieros en computación, en la que se mide la cantidad de tiempo que duran los ingenieros en responder preguntas con respecto a un código SQL y a su correspondiente versión visual, así como también la cantidad de respuestas correctas que obtienen los ingenieros en ambas versiones.

### 4.1.1 Prueba para validar la visualización

Para validar que el código SQL (textual) es más difícil de entender y por ende requiere más tiempo de comprensión que su equivalente en forma visual, se diseñó una prueba en la que código SQL (el cual se puede ver en el anexo 8.5) fue convertido con la herramienta de visualización (ver 3.4) que se desarrolló en esta investigación, a su representación equivalente en la notación visual (la cual se puede ver en el anexo 8.6). Para cada una de las representaciones (texto y visual) se diseñaron diez preguntas que permitieron medir los tiempos que tomaba responder preguntas y analizar la precisión de las respuestas (cantidad de respuestas correctas y erróneas en cada versión).

La Tabla 20 muestra la lista de preguntas que se aplicaron en la versión de texto:

| Pregunta |
|---|
| **¿Cuántas instrucciones 'Insert' tiene este código?** |
| **¿A cuál tabla pertenece la columna SALESYTD?** |
| **¿Cuántas subconsultas('Subqueries') tiene este código?** |
| **¿En cuántas instrucciones se está utilizando la tabla RTT?** |
| **¿Qué operaciones (borrado, inserción, actualización, consulta) se están realizando en la tabla TABLA3?** |
| **¿En cuántas instrucciones están haciendo join con la tabla STATEPROVINCE?** |
| **¿La columna CPN de la tabla LT en que instrucciones se está utilizando?** |
| **¿Qué tablas están proporcionando la información que se está insertando en la EMPLOYEESALES?** |
| **¿Qué tablas están haciendo un FULL JOIN?** |
| **¿Hay alguna instrucción repetida en este código?** |

*Tabla 20: Preguntas aplicadas al participante (SQL textual)*

La Tabla 21 muestra la lista de preguntas que se aplicaron en la versión de visual:

| Pregunta |
|---|
| **¿Cuántas instrucciones 'UPDATE' tiene este código?** |
| **¿A cuál tabla pertenece la columna TSC?** |
| **¿Cuántas subconsultas('Subqueries') tiene este código?** |
| **¿En cuántas instrucciones se está utilizando la tabla LOCATION?** |
| **¿Qué operaciones (borrado, inserción, actualización, consulta) se están realizando en la tabla MFR?** |
| **¿En cuántas instrucciones están haciendo join con la tabla SALESPERSON?** |
| **¿La columna GROUPNAME de la tabla DEPARTMENT en que instrucciones se está utilizando?** |
| **¿Qué tabla(s) está(n) proporcionando la información que se está insertando en la tabla @MYTABLEVAR?** |
| **¿Qué tablas están haciendo un LEFT JOIN?** |
| **¿Hay alguna instrucción repetida en este código?** |

*Tabla 21: Preguntas aplicadas al participante (SQL visual)*

### 4.1.2 Aplicación de la prueba

Para validar la hipótesis se realizaron experimentos para medir el tiempo necesario que toma comprender un algoritmo escrito en SQL, para esto los experimentos contaron con:

a. **Participantes**: Estudiantes de maestría en computación del instituto tecnológico e ingenieros en computación.
b. **Prueba**: al igual que (Scanlan, 1989) se utilizaron instrucciones de diferente complejidad: sencillo, mediano y complejo. Cada instrucción tendrá 2 versiones una en código SQL y otra con la notación visual. Estas instrucciones son equivalentes semánticamente, pero con diferente orden de tal forma que los participantes no puedan asociar uno con otro y de esta forma permitan comparar los tiempos de respuesta en las dos versiones (texto y visual).
c. **Variable dependiente**: tiempo requerido para responder la prueba (preguntas sobre el código SQL) el cual es medido en cantidad de segundos. Adicionalmente se midió otra variable que es la cantidad de errores (respuestas incorrectas) al contestar la prueba.

Para aplicar las preguntas a cada participante se desarrolló un software que permite:

- Mostrar cada pregunta
- Medir el tiempo que el participante duraba en contestar
- Llevar una bitácora de los resultados

Este software se desarrolló en .net utilizando el lenguaje de programación C#:

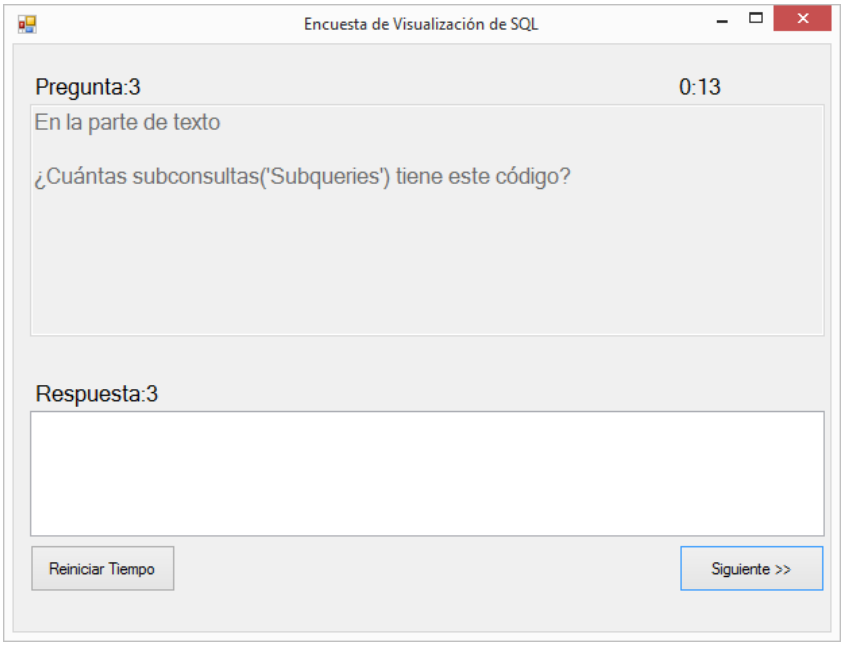



*Figura 32: Software para aplicar la prueba*

La siguiente Figura 33muestra la bitácora de respuestas de uno de los participantes:

| | Num Preg | Minutos | Seconds | Respuesta | Pregunta | Tiempo |
|---|---|---|---|---|---|---|
| 2 | 1 | 2 | 7 | 5 | En la parte de texto¿Cuántas intrucciones 'Insert' tiene este código? | 127 |
| 3 | 2 | 2 | 8 | sales.salespers( | En la parte de texto¿A cuál tabla pertenece la columna SALESYTD? | 128 |
| 4 | 3 | 2 | 25 | 7 | En la parte de texto¿Cuántas subconsultas('Subqueries') tiene este código? | 145 |
| 5 | 4 | 4 | 39 | 1 | En la parte de texto¿En cuantas instrucciones se está utilizando la tabla RTT? | 279 |
| 6 | 5 | 2 | 19 | 1 | En la parte de texto¿Qué operaciones(borrado, inserción, actualización, consulta) se están realizando en | 139 |
| 7 | 6 | 1 | 21 | 2 | En la parte de texto¿En cuántas instrucciones están haciendo join con la tabla STATEPROVINCE? | 81 |
| 8 | 7 | 2 | 10 | update | En la parte de texto¿La columna CPN de la tabla LT en que instrucciones se está utilizando? | 130 |
| 9 | 8 | 3 | 38 | sales.salespers( | En la parte de texto¿Qué tablas están proporcionando la información que se está insertando en la EMPLC | 218 |
| 10 | 9 | 1 | 45 | t1 - t2 | En la parte de texto¿Qué tablas están haciendo un FULL JOIN? | 105 |
| 11 | 10 | 0 | 43 | si | En la parte de texto¿Hay alguna intrucción repetida en este código? | 43 |
| 12 | 11 | | | | Por favor cambie a la parte gráfica. | 0 |
| 13 | 12 | 0 | 25 | 3 | En la parte de gráfica¿Cuántas intrucciones 'UPDATE' tiene este código? | 25 |
| 14 | 13 | 0 | 50 | ct1(src) | En la parte de gráfica¿A cuál tabla pertenece la columna TSC? | 50 |
| 15 | 14 | 1 | 5 | 12 | En la parte de gráfica¿Cuántas subconsultas('Subqueries') tiene este código? | 65 |
| 16 | 15 | 1 | 14 | 1 | En la parte de gráfica¿En cuantas instrucciones se está utilizando la tabla LOCATION? | 74 |
| 17 | 16 | 0 | 54 | update | En la parte de gráfica¿Qué operaciones(borrado, inserción, actualización, consulta) se están realizando e | 54 |
| 18 | 17 | 1 | 14 | 1 | En la parte de gráfica¿En cuántas instrucciones están haciendo join con la tabla SALESPERSON? | 74 |
| 19 | 18 | 0 | 56 | insert | En la parte de gráfica¿La columna GROUPNAME de la tabla DEPARTMENT en que instrucciones se está util | 56 |
| 20 | 19 | 1 | 36 | location | En la parte de gráfica¿Qué tabla(s) está(n) proporcionando la información que se está insertando en la ta | 96 |
| 21 | 20 | 0 | 36 | tabla 2 | En la parte de texto¿Qué tablas están haciendo un LEFT JOIN? | 36 |
| 22 | 21 | 0 | 30 | si | | 30 |

*Figura 33: Bitácora de los resultados de un participante*

### 4.1.3 Resultados de la aplicación de la prueba

En esta sección se presentan los resultados de la aplicación de la prueba, la cual se aplicó a 13 ingenieros (ver el cálculo del tamaño de la muestra en la sección 4.2.1) que trabajan con SQL como parte de su trabajo diario.

#### *4.1.3.1 Resultados Totales*

El Gráfico 34 muestra los resultados del tiempo promedio de respuesta de los participantes en la parte de texto versus la parte visual. Como se puede apreciar las respuestas en la parte visual les tomaron a los participantes en promedio solo un 40% del promedio de tiempo que requirieron los participantes para responder las preguntas de la parte de texto.

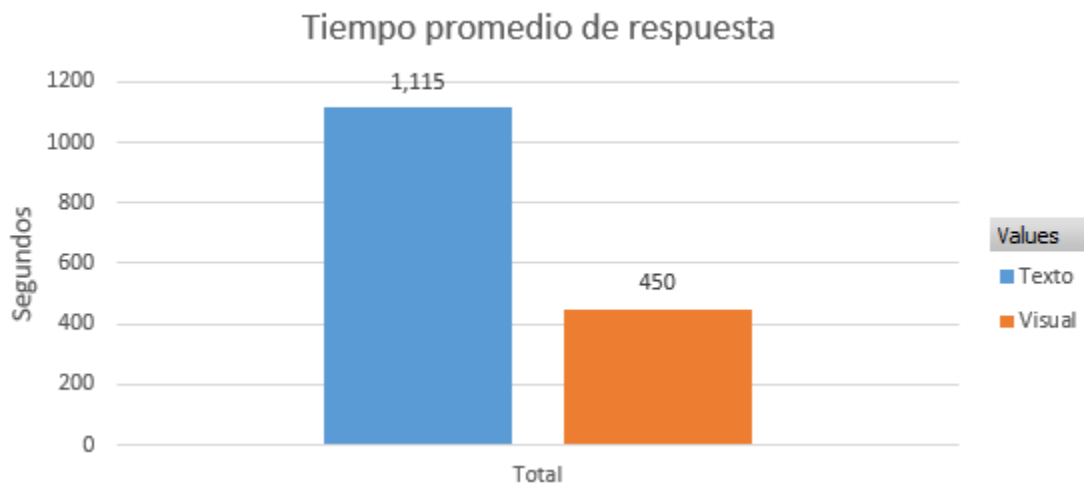



*Gráfico 34: Tiempo promedio de respuesta*

El Gráfico 35 muestra los resultados del promedio de respuestas incorrectas de los participantes en la parte de texto versus la parte visual. Como se puede apreciar las respuestas en la parte visual son en promedio solo un 52% del promedio de respuestas incorrectas que tuvieron los participantes al responder las preguntas de la parte de texto.

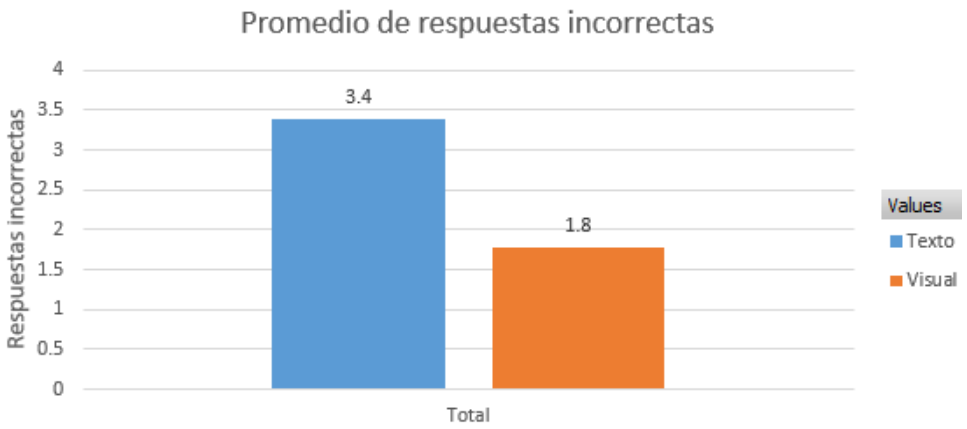



*Gráfico 35: Promedio de respuestas incorrectas*

La Tabla 22 muestra el detalle del tiempo promedio de respuesta de los participantes, así como la cantidad de respuestas incorrectas

| Participante | Tiempo Promedio respuesta (Texto) | Tiempo Promedio respuesta (Visual) | Proporción tiempo texto vrs visual | Cantidad Respuestas Incorrectas (Texto) | Cantidad Respuestas Incorrectas (Visual) | Proporción respuestas incorrectas texto vrs visual |
|---|---|---|---|---|---|---|
| P3 | 1285 | 337 | 26% | 3 | 1 | 33% |
| P1 | 1395 | 560 | 40% | 4 | 0 | 0% |
| P2 | 1473 | 522 | 35% | 3 | 1 | 33% |
| P4 | 1330 | 443 | 33% | 2 | 2 | 100% |
| P5 | 904 | 331 | 37% | 5 | 3 | 60% |
| P6 | 1166 | 462 | 40% | 3 | 1 | 33% |
| P7 | 1362 | 391 | 29% | 5 | 3 | 60% |
| P8 | 1162 | 485 | 42% | 5 | 3 | 60% |
| P9 | 693 | 354 | 51% | 2 | 2 | 100% |
| P10 | 847 | 456 | 54% | 3 | 2 | 67% |
| P11 | 964 | 524 | 54% | 2 | 1 | 50% |
| P12 | 942 | 474 | 50% | 4 | 3 | 75% |
| P13 | 971 | 511 | 53% | 3 | 1 | 33% |

*Tabla 22: Detalle de resultados por participante*

#### *4.1.3.2 Resultados categorizados*

Para segregar los resultados de la aplicación de la prueba se utilizaron las siguientes categorías:

- Carrera que estudió
- Cursos de SQL que llevó
- Años de experiencia

Para cada categoría se incluyen gráficos que presentan los resultados del tiempo que tomó a los participantes responder las preguntas con la versión de texto y con la versión visual, así como la cantidad de errores (respuestas incorrectas) en la parte de texto y la parte visual.

Como se puede observar en los gráficos siguientes la muestra es muy diversa:

- 5 carreras
- desde personas sin cursos formales de SQL hasta personas con 5 cursos de SQL
- desde personas junior (menos de 2 años de experiencia) hasta personas senior (más de 10 años de experiencia)

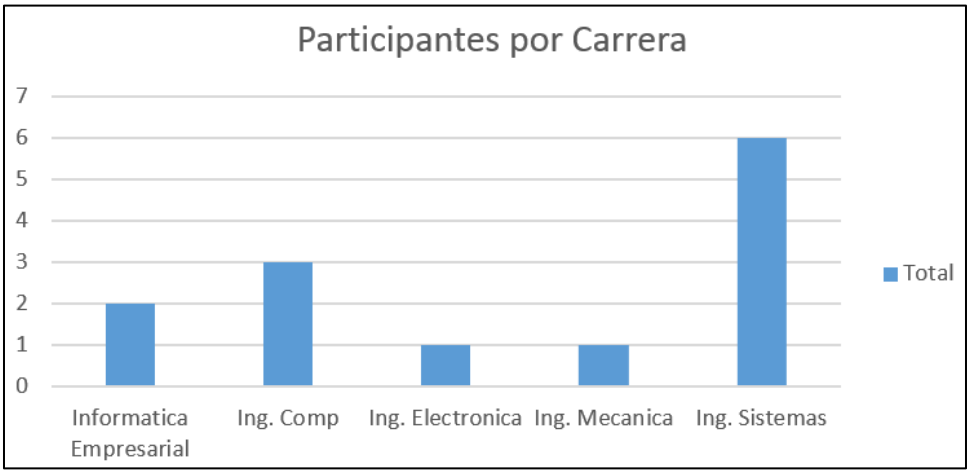
La siguiente tabla muestra el detalle de los datos de los participantes en las diferentes categorías:

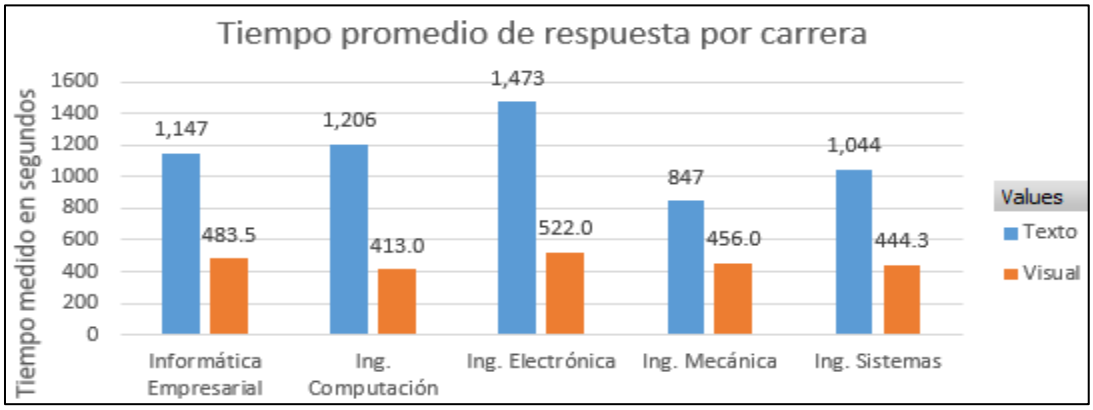
| Participante | Carrera | Años Experiencia | Cursos de Bases de datos |
|---|---|---|---|
| P1 | Ing. Sistemas | Entre 2 y 10 | 2 |
| P2 | Ing. Electrónica | Mas de 10 | 0 |
| P3 | Ing. Computación | Menos de 2 | 2 |
| P4 | Informática Empresarial | Mas de 10 | 0 |
| P5 | Ing. Sistemas | Menos de 2 | 2 |
| P6 | Ing. Sistemas | Menos de 2 | 2 |
| P7 | Ing. Computación | Menos de 2 | 2 |
| P8 | Ing. Sistemas | Mas de 10 | 2 |
| P9 | Ing. Sistemas | Mas de 10 | 2 |
| P10 | Ing. Mecánica | Menos de 2 | 0 |
| P11 | Informática Empresarial | Mas de 10 | 3 |
| P12 | Ing. Sistemas | Entre 2 y 10 | 5 |
| P13 | Ing. Computación | Mas de 10 | 2 |

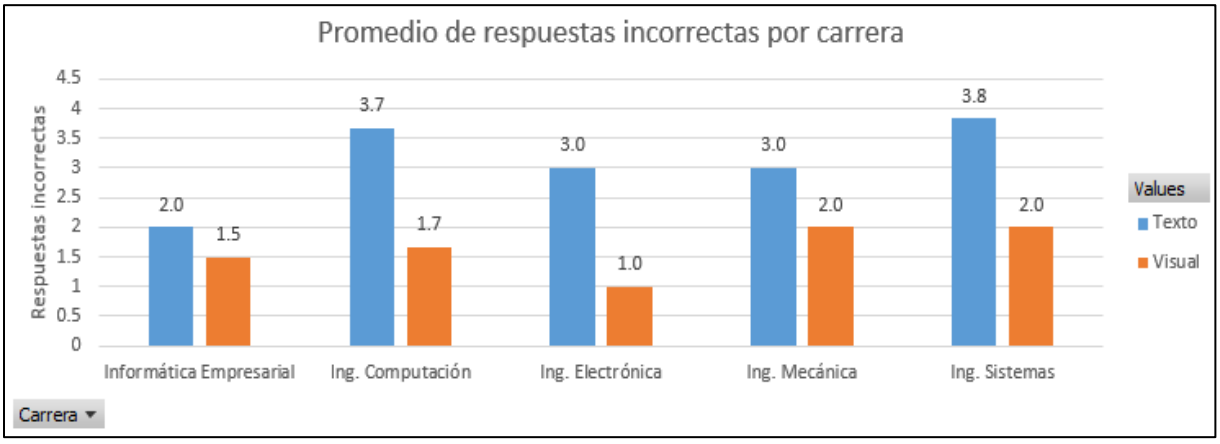
*Tabla 23: Datos de los participantes*

**Carrera**

Los siguientes gráficos muestran los datos categorizados con respecto a la carrera que estudiaron los participantes, como se puede observar no existe una correlación entre las carreras estudiadas por los participantes y los tiempos promedios de respuesta o la cantidad de respuestas incorrectas, pero si se observa que en todos los casos la notación visual mejora las respuestas:

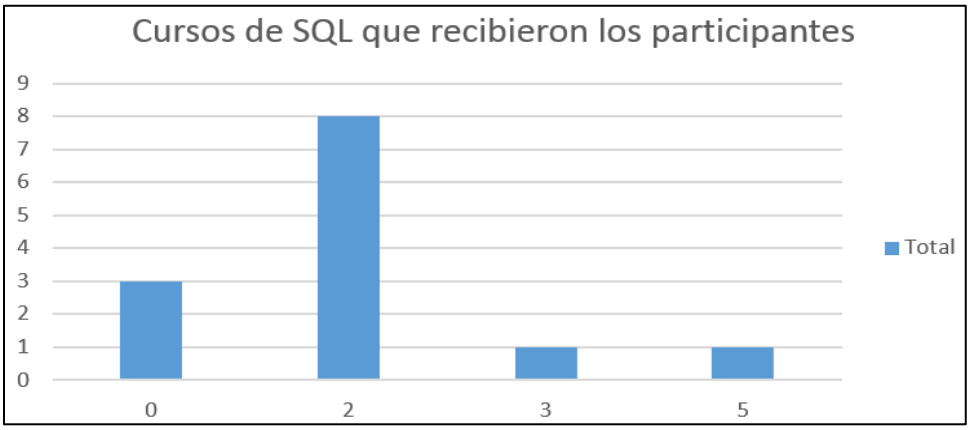


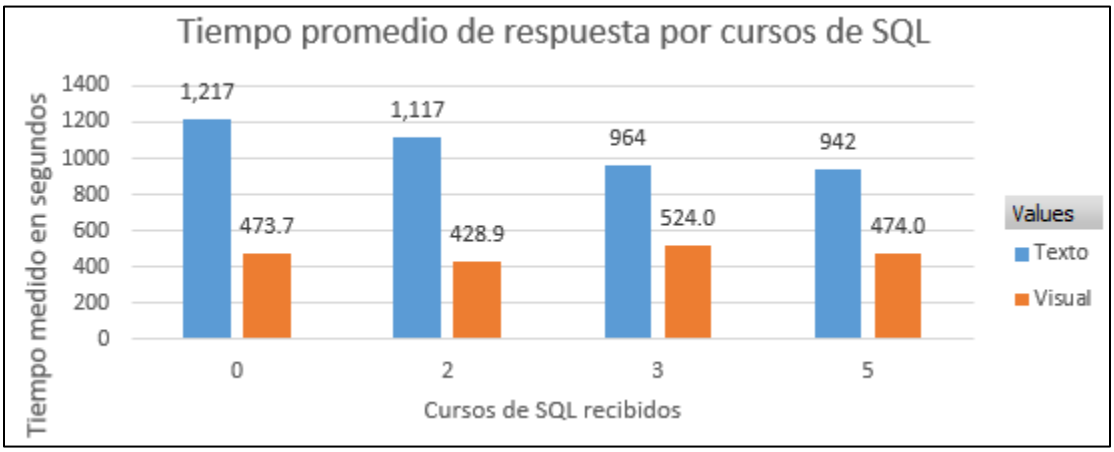



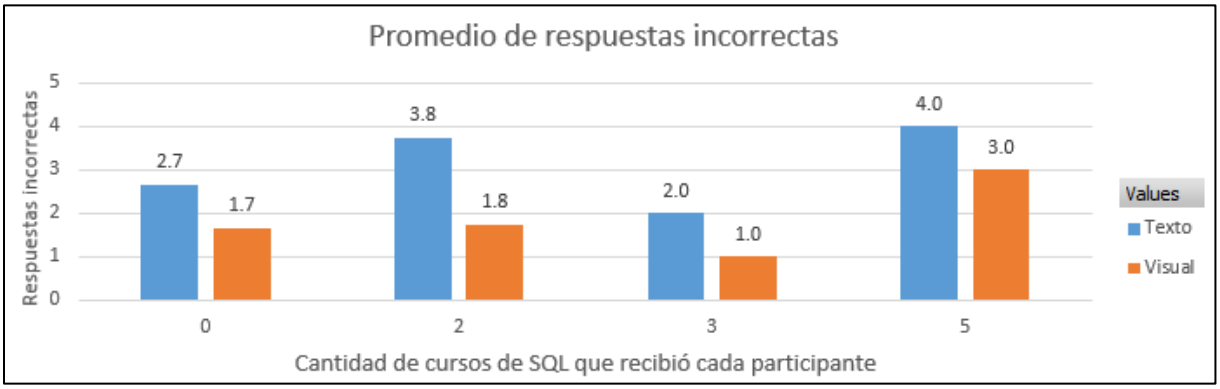



**Cursos de SQL**

Los siguientes gráficos muestran los datos categorizados con respecto a la cantidad de cursos de SQL que recibieron los participantes, como se puede observar no existe una correlación entre los cursos de SQL recibidos y la cantidad de respuestas incorrectas, pero si se observa una correlación entre los tiempos promedios de respuesta y los cursos recibidos, sin embargo, en todos los casos la notación visual mejora las respuestas:

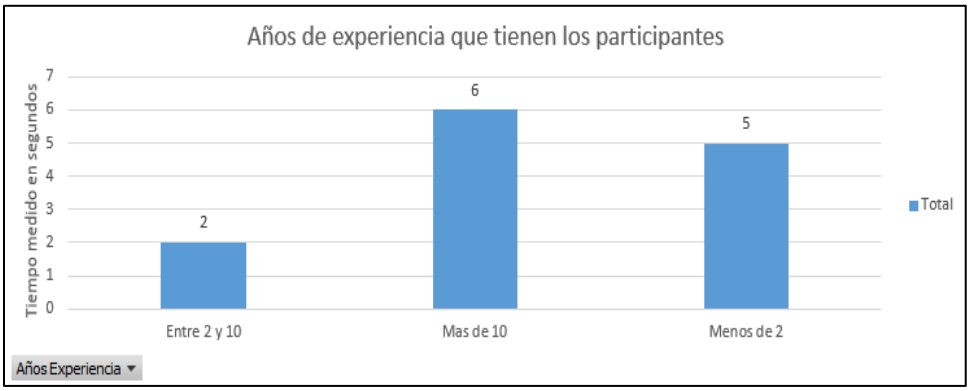


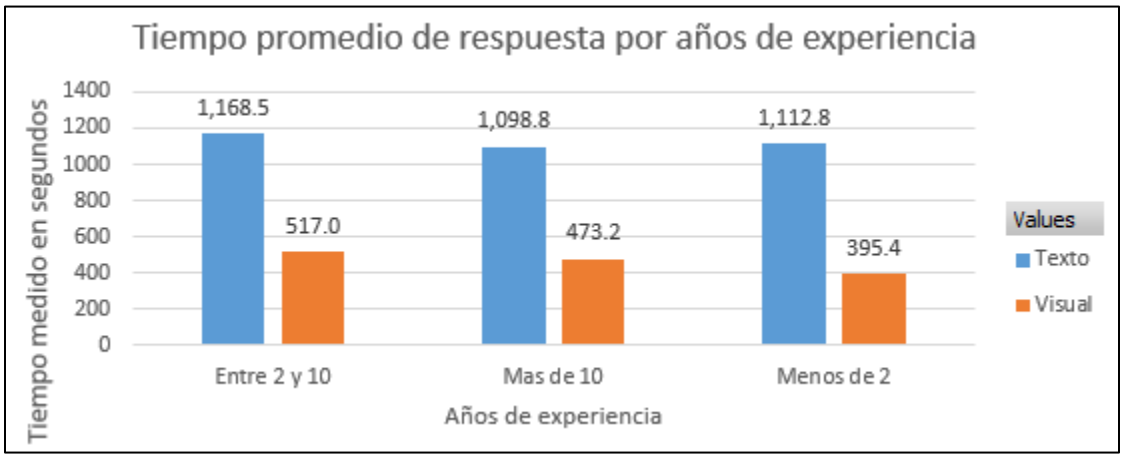



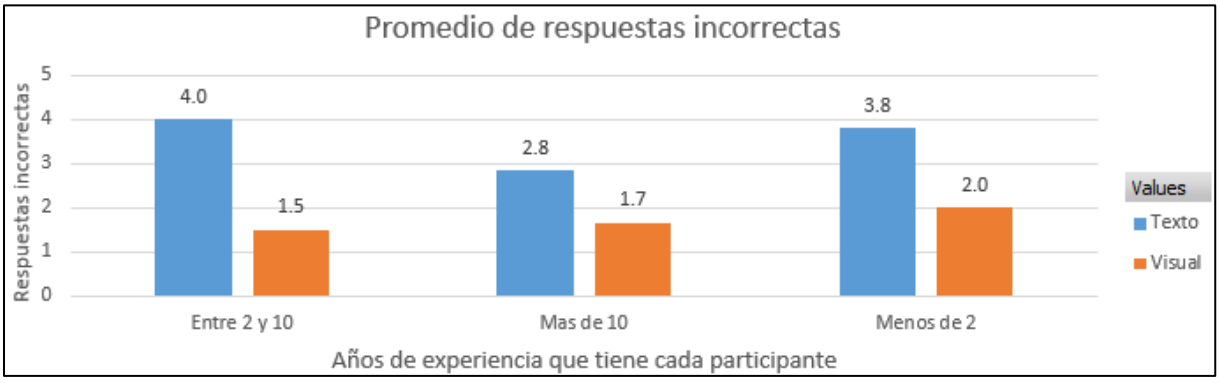



**Años de experiencia**

Los siguientes gráficos muestran los datos categorizados con respecto a los años de experiencia que tienen los participantes trabajando con SQL, como se puede observar no existe una correlación entre los años de experiencia y los tiempos promedios de respuesta o la cantidad de respuestas incorrectas, pero si se observa que en todos los casos la notación visual mejora las respuestas:



## 4.2 Análisis estadístico

Este análisis consiste en realizar una serie de pruebas estadísticas sobre los resultados recolectados en la prueba de validación para confirmar la hipótesis de investigación. Entre las pruebas realizadas están pruebas de normalidad para verificar si la muestra proviene de una población con distribución normal, y una vez confirmada la normalidad de los datos se realiza un análisis de varianza (ANOVA) para confirmar si hay evidencia estadística de que los métodos comparados (de texto y utilizando la notación visual) son diferentes.

### 4.2.1 Tamaño de la muestra para aplicación de la prueba

Para calcular el tamaño de la muestra necesaria (cantidad mínima de encuestas que se requerían de aplicar) para poder extrapolar los resultados a la población general se utilizó la siguiente fórmula

$$n = \frac{N * {Z_a}^2 * p * q}{d^2 * (N-1) + {Z_a}^2 * p * q}$$

Donde:

**N** = Tamaño de la población (para extrapolar lo datos a una población desconocida infinita se acepta un valor de 20000)

$\boldsymbol{Z_a}$= Nivel de confianza (para un nivel de confianza del 85% utilizando una distribución normal y con una prueba bilateral utiliza el valor 1.44)

***p*** = probabilidad de éxito o proporción esperada (si se desconoce el valor de 0.5 (50%) maximiza el tamaño muestral).

***q*** = (1-p) = probabilidad de fracaso = 0.5

***d*** = precisión = porcentaje de error = 20%

$$12.95 = \frac{20000 * 1.44^2 * 0.5 * 0.5}{0.2^2 * (20000-1) + 1.44^2 * 0.5 * 0.5}$$

Por lo tanto, la prueba se aplicó a 13 participantes.

### 4.2.2 Prueba de normalidad

Para validar que los datos provienen de una población con una distribución normal se aplicaron varias pruebas estadísticas a los resultados, utilizando el software R® con la herramienta RStudio®. La Tabla 24 muestra los resultados de las pruebas de normalidad aplicadas a los datos, donde se puede ver que en todas las pruebas el valor p es > 0.05 (nivel de significancia de 5%) lo que permite aceptar la hipótesis nula de que los valores están distribuidos normalmente.

| Prueba de normalidad | Valor P |
|---|---|
| **Shapiro-Francia** | 0.634 |
| **Shapiro-Wilk** | 0.541 |
| **Anderson-Darling** | 0.4329 |
| **Cramer-von Mises** | 0.3488 |
| **Lilliefors (Kolmogorov-Smirnov)** | 0.2682 |
| **Pearson chi-square** | 0.2533 |

*Tabla 24: Pruebas de Normalidad*

Adicionalmente si analizamos los siguientes gráficos (generados para los resultados de las pruebas de validación), podemos ver que:

1. El gráfico de cuantiles (“Q-Q plot”) muestra que los valores se mantienen bastante cerca de la línea diagonal (distribución normal teórica).
2. El gráfico de probabilidad (“P-P plot”) que permite comparar la distribución empírica de una muestra de datos, con la distribución normal muestra que los valores se mantienen bastante cerca de la línea diagonal (distribución normal teórica).
3. El histograma a pesar de tener pocos valores si tiene una forma simétrica lo que refuerza la premisa de normalidad de los datos.

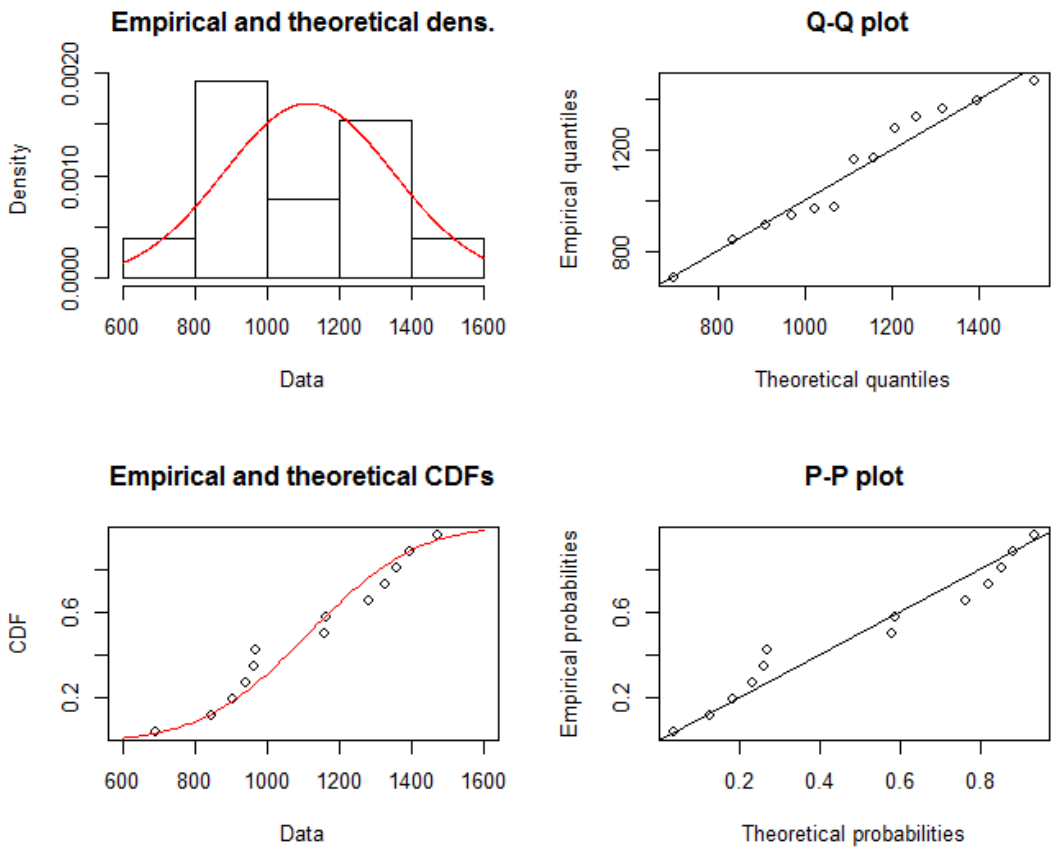



*Gráfico 36: Gráficos estadísticos de los resultados*

### 4.2.3 Análisis de varianza (ANOVA)

Realizando un análisis de varianza sobre la variable dependiente (tiempo) para los dos métodos (texto y visual) obtenemos un valor p de 1.69e-09 (ver Figura 37) lo cual es mucho menor al nivel de significancia de 0.05 y por ende rechazamos la hipótesis nula (el método de texto y el método visual son iguales) y aceptamos la hipótesis alternativa de que contamos con evidencia estadística suficiente para determinar que el método de la notación visual es considerablemente diferente al método de texto.

```
> summary(aov(tiempo ~ metodo))
            Df  Sum Sq Mean Sq F value   Pr(>F)
metodo       1 2873798 2873798   88.03 1.68e-09 ***
Residuals   24  783541   32648
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

*Figura 37: Resultados del análisis de varianza en R*

Dado que la variable categórica "método" solo tiene dos niveles (texto y visual) no requerimos de una prueba de pares para determinar cuáles de los dos grupos son diferentes entre sí.

# 5 Análisis retrospectivo

El lenguaje SQL es sumamente extenso y para poder desarrollar una notación y herramienta que cubra el cien por ciento de las instrucciones, se requiere de un esfuerzo considerablemente mayor al que se puede abarcar dentro de esta investigación por lo tanto la presente investigación no pretende cubrir la totalidad del lenguaje sino más bien servir como una prueba de factibilidad de las ideas propuestas desde el inicio de la investigación, para que en futuros trabajos se pueda completar las instrucciones no implementadas.

Como parte de este trabajo se utilizaron herramientas que aceleraron el proceso de desarrollo y que permitieron llevar la herramienta de visualización a una etapa totalmente funcional, como por ejemplo:

1. **El parser de SQL,** ahorró mucho tiempo de desarrollo dado que crear un "parser" desde el inicio es por si solo un proyecto que requiere mucho tiempo. Este analizador tenía la ventaja de que ya era un producto estable y al ser el mismo que utiliza SQL Server® soporta todas las instrucciones de transact SQL (la versión SQL de Microsoft®).
2. **La plataforma de graficación Visio** permitió obtener beneficios adicionales como impresión, salvar diagramas, "Zoom", etc. lo cual no requirió de programación adicional para que estuvieran disponibles en la herramienta.

Sin embargo, aunque se logró alcanzar los objetivos de la investigación, existen problemas que a este punto no están resueltos, que se pueden solucionar en un trabajo posterior y que se describen en la sección 5.1:

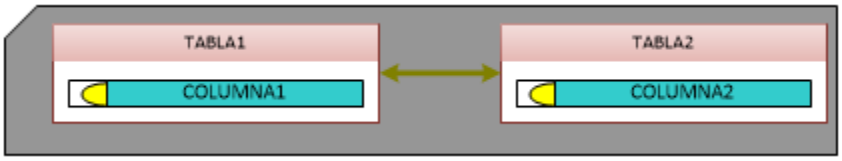
## 5.1 Trabajo a futuro

Durante el diseño de la notación y el desarrollo de la herramienta se encontraron algunos problemas que no fue posible resolver dentro del alcance de esta investigación, por ejemplo:

1. **Columnas no identificables:** Cuando una consulta de SQL incluye más de una tabla y tiene alguna columna no calificada (i.e Tabla.Columna), el analizador sintáctico no puede determinar a cuál tabla pertenece la columna por lo que esa columna no se despliega, por ejemplo, en la siguiente consulta para la Columna3 no es posible determinar si pertenece a la tabla1 o a la tabla2 por lo tanto no se despliega:

```
SELECT Tabla1.Columna1, Tabla2.Columna2, Columna3
```

```
FROM Tabla1
INNER JOIN Tabla2
```




Este problema es posible resolverlo incluyendo un diccionario de datos como parte de la herramienta o que la herramienta de visualización tenga acceso a la base de datos a que pertenecen las tablas para poder verificar a que tabla pertenece la columna.

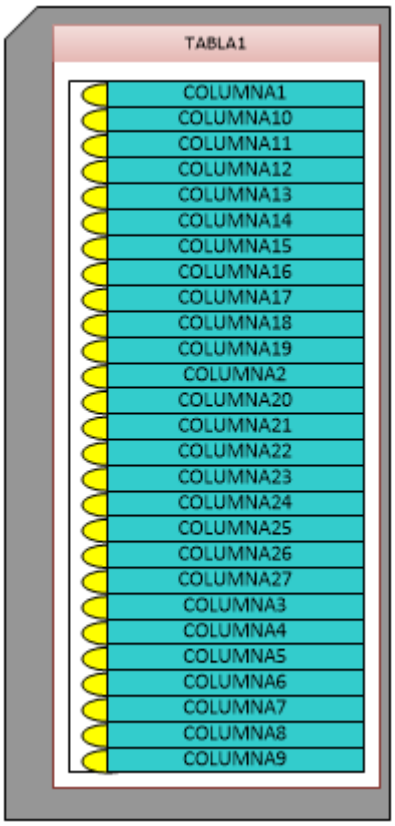
1. **Expresiones con mucho texto**: dado que el lenguaje SQL es muy extenso y tiene muchas funcionalidades, fácilmente se pueden encontrar expresiones que tienen muchos caracteres, por ejemplo, expresiones que utilizan funciones:

   ```
   REPLACE(REPLACE([miss_sls_val],',',''),'$','')
   ```

   En este caso se decidió omitir el texto de la expresión para no sobrecargar visualmente las instrucciones.

2. **Tablas con muchas columnas:** SQL Server permite definir tablas con hasta 1024 columnas lo cual hace que un diagrama pierda su efectividad y se vea saturado de información, actualmente la herramienta está graficando todas las columnas que intervienen en cada instrucción, por ejemplo:

```
SELECT Columna1
,Columna2
,Columna3
,Columna4
,Columna5
,Columna6
,Columna7
,Columna8
,Columna9
,Columna10
,Columna11
,Columna12
,Columna13
,Columna14
,Columna15
,Columna16
,Columna17
,Columna18
,Columna19
,Columna20
,Columna21
,Columna22
,Columna23
,Columna24
,Columna25
,Columna26
,Columna27
FROM Tabla1;
```

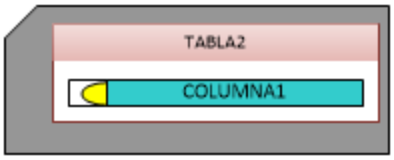



Este problema se puede mejorar graficando únicamente las columnas que son más relevantes para la consulta (las que son parte de las cláusulas "*Where", "Order by", "Group by")*

3. **Subconsultas utilizadas en la cláusula "*Select*"**: SQL server permite incluir subconsultas ("subqueries") en la cláusula "*Select*" para definir el valor de una columna, por ejemplo:

```
SELECT Columna1 = ( SELECT ( Columna4 * .90 )
                    FROM   Tabla2
                    WHERE  Tabla2.Columna2 = Tabla1.Columna2
                           AND Tabla2.Columna3 = 'B'
                  )
From Tabla2;
```

Esta funcionalidad en este momento no está soportada por la notación y en consecuencia no se está graficando en la herramienta. La consulta anterior actualmente es visualizada como:




4. **Instrucciones no incluidas en la notación:** Como se puede ver en la sección 3.1.1, en este trabajo solo se incluyeron algunas instrucciones de SQL (las más importantes deacuerdo a su frecuencia de uso en el código analizado).

5. **Cantidad de atributos y cláusulas por instrucción:** la mayoría de instrucciones en SQL tienen una sintaxis muy extensa como el "Update" (ver 3.1.4), las cual en muchos casos puede incluir opciones para aspectos tan específicos forzar o modificar los planes de ejecución que establece el optimizador de SQL server como por ejemplo los "hints" o el soporte para XML de la instrucción "Select":

```
<SELECT statement> ::=
    [ WITH { [ XMLNAMESPACES ,] [ <common_table_expression> [,...n] ] } ]
    <query_expression>
    [ ORDER BY { order_by_expression | column_position [ ASC | DESC ] }
  [ ,...n ] ]
    [ <FOR Clause>]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
```

   Estas opciones particulares y muy específicas de cada instrucción no se implementaron ni en la notación ni en la herramienta.

6. Si bien es cierto, en la notación se lograron plasmar conceptos importantes de las instrucciones seleccionadas también cabe destacar que se requiere incorporar más conceptos de SQL para poder tener una equivalencia cognitiva entre el código SQL y el diagrama generado.

# 6 Conclusiones y Recomendaciones

## 6.1 Conclusiones

1. Se logró cumplir con los objetivos propuestos al inicio de esta investigación al desarrollar de forma exitosa una notación visual y herramienta de visualización que permite convertir código SQL en su equivalente con la notación visual.
2. Con base en los resultados del análisis cuantitativo se pudo determinar que la utilización de la notación visual y más específicamente de los diagramas generados por la herramienta de visualización ahorran aproximadamente 60% del tiempo que requieren los ingenieros para analizar y comprender instrucciones SQL en texto.
3. Los resultados de la aplicación de la prueba de validación confirmaron que los participantes tuvieron mayor precisión (menos respuestas incorrectas) para responder las preguntas de la parte visual, lo que muestra que la notación visual ayuda en el proceso de análisis de código SQL.
4. La aplicación de la prueba de validación permitió confirmar que la notación visual requiere poco tiempo de entrenamiento para entender los conceptos y los grafemas asociados a esos conceptos. En la aplicación de la prueba los participantes requirieron entre 5 y 15 minutos de explicación para poder responder las preguntas de la parte visual.
5. La notación visual tiene un beneficio adicional a los señalados en los puntos anteriores y es el de separar y categorizar la información de tal forma que fácilmente se puede determinar por ejemplo, la ubicación de los nombres de tablas, nombres de columnas, atributos, etc.
6. El análisis estadístico permitió confirmar que los resultados de la muestra provenían de poblaciones con distribución normal y el análisis de varianza permitió comprobar que los dos métodos analizados (texto y con la notación visual) son significativamente diferentes.
7. La representación intermedia ayudó a desligar el analizador sintáctico del visualizador lo cual permite que se en el futuro se puedan cambiar esos componentes para soportar código SQL de otros motores de bases de datos como Oracle®, Teradata®, etc.

## 6.2 Recomendaciones

1. El trabajo iniciado en esta investigación se puede continuar para agregar más instrucciones a la notación visual y a la herramienta de visualización como lo son las instrucciones de flujo de control (“loops”, “ifs”, etc.) para lograr mayor equivalencia entre el texto y los diagramas.
2. En la herramienta de graficación se puede incrementar la velocidad de despliegue almacenando en archivos la información cargada en los objetos de la representación intermedia (tablas, joins, etc.) de tal forma que el análisis sintáctico solo sea necesario si hay alguna actualización en el código de un procedimiento almacenado.
3. Se recomienda agregar la funcionalidad de filtrar información a la herramienta de visualización para poder desplegar solo las tablas y columnas que interesan en la consulta o grupo de consultas que se están analizando.
4. A la herramienta de visualización se le puede agregar la funcionalidad de conexión con bases de datos para poder obtener meta datos de las tablas que se utilizan en el código SQL, de esa forma se pueden resolver problemas de ambigüedad de columnas e incrementar la equivalencia entre el código textual y la representación visual.
5. Es recomendable agregar un diagrama para ver la interacción de tablas con procedimientos almacenados el cual incrementaría la capacidad de análisis de procedimientos almacenados.
6. Se pueden agregar la capacidad de representar y graficar Join de múltiples columnas a la herramienta de graficación para incrementar la equivalencia entre la parte textual y la visual.
7. En la herramienta de visualización se puede sacar mayor provecho de la capacidad de despliegue de información mediante “Tool tips” principalmente para desplegar expresiones asociadas a columnas, por ejemplo expresiones de columnas que pertenecen a la cláusula “*Where*” de una instrucción

# 7 Glosario

**Analizador sintáctico:** Un analizador sintáctico (o parser) es una de las partes de un compilador que transforma su entrada en un árbol de derivación.

El análisis sintáctico convierte el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada. Un analizador léxico crea tokens de una secuencia de caracteres de entrada y son estos tokens los que son procesados por el analizador sintáctico para construir la estructura de datos, por ejemplo un árbol de análisis o árboles de sintaxis abstracta.

**Cognición**: La cognición (del latín cognoscere, 'conocer') se define como la facultad de un ser vivo para procesar información a partir de la percepción, el conocimiento adquirido (experiencia) y características subjetivas que permiten valorar la información. Consiste en procesos tales como el aprendizaje, el razonamiento, la atención, la memoria, la resolución de problemas, la toma de decisiones, los sentimientos.

**Grafema**: Unidad mínima e indivisible de un sistema de representación gráfica de la lengua.

**Percepción**: La percepción es la manera en la que el cerebro de un organismo interpreta los estímulos sensoriales que recibe a través de los sentidos para formar una impresión consciente de la realidad física de su entorno.

**Semántica**: Semántica se refiere a los aspectos del significado, sentido o interpretación de signos lingüísticos como símbolos, palabras, expresiones o representaciones formales. En principio las expresiones del lenguaje formal o de una lengua natural admiten algún tipo de correspondencia con situaciones o conjuntos de cosas que se encuentran en el mundo físico o abstracto que puede ser descrito por dicho medio de expresión.

**Semiótica**: La semiología o semiótica (del griego "simiotikos") es la ciencia que trata de los sistemas de comunicación dentro de las sociedades humanas, estudiando las propiedades generales de los sistemas de signos, como base para la comprensión de toda actividad humana. Aquí, se entiende por signo un objeto o evento presente que está en lugar.

# 8 Anexos

## 8.1 El analizador sintáctico de Transact SQL

Este proyecto permite analizar código SQL de Microsoft® SQL Server®, para tal efecto SQL Server® de Microsoft® provee un analizador sintáctico el cual se referencia y accede mediante el archivo ("Parser.cs") de la herramienta de visualización:

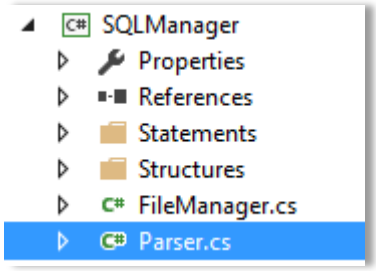



*Figura 38: Programa del analizador sintáctico*

Algunos fragmentos importantes del código del analizador son:

1. Inicialización del parser

```
private string fileName;
internal System.Collections.ArrayList arrayStatements;

public Parser(string fileName)
{
    this.fileName = fileName;
    arrayStatements = new System.Collections.ArrayList();
}
```

2. Método principal de invocación del analizador sintáctico

```
public System.Collections.ArrayList Parse()
{
    TSql120Parser SQLParser = new TSql120Parser(true);
    TSqlFragment sqlFragment = null;
    IList<ParseError> parseErrors;

    try
    {   // Open the text file using a stream reader.
        using (StreamReader sr = new StreamReader(this.fileName))
        {
            sqlFragment = SQLParser.Parse(sr, out parseErrors);
            if (parseErrors.Count > 0)
            {
                System.Console.WriteLine("Error: the parser failed, please check
                                                                    syntax errors");
            }
        }
```

```
        }
        catch (Exception e)
        {
            Console.WriteLine("The file could not be read: " + e.Message);
        }
        SQLVisitor myVisitor = new SQLVisitor(ref this.arrayStatements);
        sqlFragment.Accept(myVisitor);
        myVisitor.DumpStatistics();
        return arrayStatements;

    }
}
```

3. Método para recorrer un nodo del árbol, correspondiente a una instrucción “Select”

```
public override void ExplicitVisit(SelectStatement Node)
 {
     var BinaryQueryExpressionType = typeof(BinaryQueryExpression);
     Console.WriteLine("statement");

     QueryExpression QE = Node.QueryExpression;
     Statements.Select select = new Statements.Select();
     if (parserInfo.Instance.latestStatement != null) select.Parent =
                                    parserInfo.Instance.latestStatement;

     select.Text = GetNodeTokenText(Node);
     select.Alias = "";

     if (BinaryQueryExpressionType.IsInstanceOfType(Node.QueryExpression))
     {
         SQLManager.Structures.Join join = new Structures.Join
          (
          ProcessQueryExpression("",
                     ((BinaryQueryExpression)QE).FirstQueryExpression, select),
          ProcessQueryExpression("",
                     ((BinaryQueryExpression)QE).SecondQueryExpression, select),

       getJoinType(((BinaryQueryExpression)QE).BinaryQueryExpressionType.ToString(
       ))
           );
         select.Joins.Add(join);
         if (((BinaryQueryExpression)QE).OrderByClause!=null)
         {
             ProcessOrderBy(((BinaryQueryExpression)QE).OrderByClause,
                    ((SQLManager.Structures.Table)select.Tables[0]).Select);
             ProcessOrderBy(((BinaryQueryExpression)QE).OrderByClause,
                    ((SQLManager.Structures.Table)select.Tables[1]).Select);
         }
     }
     else
     {
         ProcessQuery((QuerySpecification)QE, select);
     }
     arrayStatements.Add(select);
     SELECTcount++;
 }
```

## 8.2 El patrón de diseño visitante (“Visitor”)

### Definición

El patrón de diseño de visitante es una forma de separar una operación de una estructura de objetos en la que opera. En esencia, este patrón permite agregar nuevas funciones virtuales a una familia de clases sin modificar las clases mismas;

El patrón de visitantes consta de dos partes:

- Un método llamado Visit () que es implementado por el visitante y es llamado para cada elemento en la estructura de datos
- Clases visitables que proporcionan métodos Accept () que aceptan a un visitante

En el caso del analizador sintáctico de SQL, todo fragmento de SQL implementa el método (“accept”):

```
sqlFragment.Accept(myVisitor);
```

El cual requiere de una clase visitante (“visitor”) que pueda recorrer el árbol sintáctico construido por el analizador:

```
SQLVisitor myVisitor = new SQLVisitor(ref this.arrayStatements);
```

y la implementación de métodos “visit” para cada instrucción que se quiere recorrer, por ejemplo para el “Insert” este es el método que lo recorre:

```
public override void ExplicitVisit(InsertStatement node)
{
    SQLManager.Statements.Insert insert = new Statements.Insert();
    if (parserInfo.Instance.latestStatement != null) insert.Parent =
                                    parserInfo.Instance.latestStatement;

    insert.Text = GetNodeTokenText(node);
    ProcessInsert(node, insert);
    arrayStatements.Add(insert);
    INSERTcount++;
}
```

## 8.3 La representación intermedia que integra el analizador sintáctico y el visualizador.

Para poder integrar el analizador sintáctico con el visualizador y evitar dependencias entre sí, se desarrolló una representación intermedia que es independiente de cualquier analizador sintáctico y a su vez independiente de cualquier plataforma de visualización.

La representación intermedia se compone de:

- Una clase base (abstracta) para representar instrucciones SQL
- Clases derivadas para cada tipo de instrucción SQL implementada en esta investigación.
- Clases para representar los conceptos de SQL como: Tablas, Columnas, Joins

**Arreglo de instrucciones:**

Este arreglo permite almacenar todas las instrucciones procesadas por el analizador sintáctico.

```
internal System.Collections.ArrayList arrayStatements;
```

**La Clase "Statement"**

Esta clase contiene los atributos generales que se requieren para transferir la información necesaria entre el analizador sintáctico y la plataforma de visualización.

```
public abstract class SQLStatement: ISQLStatement
  {
      ///Atributos de esta clase
      private StatementType type;
      private string text;
      private string alias;
      private ArrayList tables;
      private ArrayList joins;
      private int maxTables;
      private int maxColumns;
      private SQLManager.ISQLStatement parent;
      private double x1;
      private double y1;
      private double x2;
      private double y2;
      private int level;

      public SQLStatement()
      {
          this.Type = StatementType.Select;
```

```
            this.Tables = new ArrayList();
            this.Joins = new ArrayList();
            this.MaxColumns = 0;
            this.MaxTables = 0;
            this.Alias = "";
        }
        /// Esta clase cuenta con métodos para acceder cada atributo (Get y Set)
    }
```

### La Clase “Table”

Esta clase contiene los atributos que se requieren para almacenar la información más importante de una tabla como lo son: el nombre de la tabla, el alias, las columnas, etc.

```
    public class Table
    {
        ///Atributos de esta clase
        private string tableName;
        private string alias;
        private bool isMainTable;
        private SortedList columns;
        private string qServer;
        private string qDatabase;
        private string qSchema;
        private string key;
        private SQLManager.Statements.Select select;
        private double x1;
        private double y1;
        private double x2;
        private double y2;
        private SQLManager.Statements.SQLStatement parent;

        public Table(string tableName, SQLStatement Parent)
        {
            this.TableName = tableName;
            this.Columns = new SortedList();
            this.IsMainTable = false;
            this.Alias = "";
            this.Server = "";
            this.Database = "";
            this.Schema = "";
            this.Select = null;
            this.Parent = Parent;
        }

        /// Esta clase cuenta con métodos para acceder cada atributo (Get y Set)
    }
```

### La Clase “Column”

Esta clase contiene los atributos que se requieren para almacenar la información más importante de una columna como lo son: el nombre de la columna, y si es parte de alguna cláusula como: “*Select”, “Where”, “Group by”, “Order by*”.

```
public class Column
{
    ///Atributos de esta clase
    string columnName;
    private double x1;
    private double y1;
    private double x2;
    private double y2;
    private Order order;
    private bool selected;
    private bool grouped;
    private bool filtered;

    public Column(string ColumnName, bool isSelected)
    {
        columnName = ColumnName;
        Order= Order.None;
        Selected = isSelected;
    }
    /// Esta clase cuenta con métodos para acceder cada atributo (Get y Set)
}
```

### La Clase “Join”

La clase join permite representar uniones entre tablas (left, right, inner, full joins) o entre consultas (union, intersect, except)

```
public enum JoinType
{
    /// Represents an inner join.
    Inner,
    /// Represents a left join.
    Left,
    /// Represents a right join.
    Right,
    /// Represents a full join.
    Full,
    /// Represents an Union.
    Union,
    /// Represents an Intersec.
    Intersec,
    /// Represents an Except.
    Except
}
```

```
public class Join
{
     ///Atributos de esta clase
     private SQLManager.Structures.Table first, second;
     JoinType type;

     public Join(SQLManager.Structures.Table First,
          SQLManager.Structures.Table Second,
          SQLManager.Structures.JoinType JoinType)
     {
         this.Type = JoinType;
         this.First = First;
         this.Second = Second;
     }
     /// Esta clase cuenta con métodos para acceder cada atributo (Get y Set)
}
```

## 8.4 La interfaz de programación de Microsoft® Visio®

La interfaz de programación de Microsoft Visio® permite dibujar la información almacenada en la representación intermedia en diagramas, los cuales despliegan las instrucciones SQL en forma gráfica (utilizando la notación visual).

Los siguientes, son los fragmentos más importantes de la clase encargada de la visualización:

1. **Constantes utilizadas en la visualización**: Estas constantes definen los colores utilizados por los diferentes elementos gráficos, así como márgenes, longitudes, separaciones, etc. requeridas por el graficador.

```
class VisioDrawer
{
    //constants
    private const double TableWidth = 2;
    private const double TableHeight = 1;
    private const double TableMarginX = 0.2;
    private const double TableMarginY = 0.2;
    private const double JoinWidth = 1.5;
    private const double ColumnHeight = 0.2;
    private const double StatementsSeparation = 3;
    private const double InsertFlipAdjustment = 0;

    //colors
    private const string ColumnColor = "RGB(51,204,204)";
    private const string ColumnSelectedColor = "RGB(255,255,0)";
    private const string ColumnFilteredColor = "RGB(255,128,128)";
    private const string ColumnGroupedColor = "RGB(0,0,128)";
    private const string ColumnAscendingColor = "RGB(255,255,255)";
    private const string ColumnDescendingColor = "RGB(255,0,255)";

    private const string TableColor = "RGB(255,255,255)";
    private const string SelectColor = "RGB(150,150,150)";
    private const string UpdateColor = "RGB(128,0,128)";
    private const string InsertColor = "RGB(255,128,0)";
    private const string DeleteColor = "RGB(128,0,0)";
    private const string SubQueryColor = "RGB(255,255,128)";

    private const string InnerJoinColor = "RGB(128,128,0)";
    private const string RightJoinColor = "RGB(255,0,128)";
    private const string LeftJoinColor = "RGB(0,128,128)";
    private const string FullJoinColor = "RGB(0,0,0)";
    private const string ExceptColor = "RGB(0,255,255)";
    private const string IntersectColor = "RGB(0,255,0)";
    private const string UnionColor = "RGB(0,0,255)";

    private const string LineWeight = "3 pt";

    const string DYNAMIC_CONNECTOR_MASTER = "Dynamic Connector";
    const string MESSAGE_NOT_SAME_PAGE = "Both the shapes are not on the same page.";
```

2. Método para dibujar una Consulta

```
        private void DrawSelect(double x, double y, SQLManager.Statements.Select Select,
string Key)
        {

            Shape shape1;
            Select.X1 = x;
            Select.Y1 = y;
            Select.X2 = x;
            Select.Y2 = y;
            bool isSubQuery = !Key.Equals("");

            if (isSubQuery)
            {
                Select.X2 += TableMarginX*2;

                // subQuery
                shape1 = DrawSubQuery( Select.X2,  Select.Y1, Key);
            }
            else
            {
                //Normal select
                shape1 = currentPage.Drop(selectTemplate, Select.X1 , Select.Y1);
                //set the shape back color
                shape1.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                      (short)VisRowIndices.visRowFill,
                      (short)VisCellIndices.visFillForegnd).FormulaU = SelectColor;
            }

            shape1.Text = Select.Alias;
            shape1.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                      (short)VisRowIndices.visRowMisc,
                     (short)VisCellIndices.visComment).FormulaU =
                     "\"" + Select.Text + "\"";

            //tables drawing
            DrawNTables(x, y, Select, 1, true);

            AddJoinConnections(Select, shapesArray);
            if (isSubQuery)
            {
                shape1.Resize(VisResizeDirection.visResizeDirE, Select.X2 –
                                   Select.X1 - 0.5, VisUnitCodes.visInches);
                shape1.Resize(VisResizeDirection.visResizeDirS, (Select.Y1 - Select.Y2)
                                   -(TableMarginY*1.5), VisUnitCodes.visInches);
                Select.X2 += JoinWidth;
            }
            else
            {
                shape1.Resize(VisResizeDirection.visResizeDirE, Select.X2
                                   - Select.X1 , VisUnitCodes.visInches);
                shape1.Resize(VisResizeDirection.visResizeDirS, (Select.Y1
                                   - Select.Y2), VisUnitCodes.visInches);
            }

        }
```

3. Método para dibujar una tabla: Este método permite dibujar una tabla la cual es a su vez un contenedor para las columnas y atributos.

```
private void DrawTable(double x, double y, SQLManager.Structures.Table table,
                                                                    bool updateParent)
{
    table.X1 = x; // table.Parent.X2 ;
    table.Y1 = y; // table.Parent.Y1 -TableMarginY;
    table.X2 = table.X1 + TableWidth;
    table.Y2 = table.Y1 - (ColumnHeight * table.Columns.Count);

    //updates parent's height and width
    if (updateParent)
    {
        if (table.X2 > table.Parent.X2) table.Parent.X2 = table.X2;
        if (table.Y2 < table.Parent.Y2) table.Parent.Y2 = table.Y2;
    }

    Shape shape1 = currentPage.DrawRectangle(table.X1, table.Y1, table.X2,
                                                                    table.Y2);

    //create selection
    Microsoft.Office.Interop.Visio.Selection selection =
      currentPage.CreateSelection(
      Microsoft.Office.Interop.Visio.VisSelectionTypes.visSelTypeEmpty,
      Microsoft.Office.Interop.Visio.VisSelectMode.visSelModeSkipSuper,  null);
    selection.Select(shape1,
      (short)Microsoft.Office.Interop.Visio.VisSelectArgs.visSelect);

    //create container
    Microsoft.Office.Interop.Visio.Shape container = null;
    container = currentPage.DropContainer(
      builtInStencil.Masters.get_ItemU("Container 5"),selection);
    container.BringToFront();
    shape1.BringToFront();

    shapesArray.Add(table.Key, container);

    //set the shape back color
    shape1.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                    (short)VisRowIndices.visRowFill,
                    (short)VisCellIndices.visFillForegnd).FormulaU = TableColor;

    if (table.Alias.Length > 0)
    {
        container.Text = table.TableName + "(" + table.Alias + ")";
    }
    else
    {
        container.Text = table.TableName;
    }
    DrawColumns(table.Y1, table.X1, table,container);
    container.ContainerProperties.LockMembership = true;
}
```

4. Método para dibujar columnas: Este método permite dibujar las columnas y sus atributos dentro de la tabla.

```
private void DrawColumns(double y, double x,  SQLManager.Structures.Table table,
                         Microsoft.Office.Interop.Visio.Shape containerShape)
{
    x += 0.3;
    //addedDocument.Application.ScreenUpdating = Convert.ToInt16(false);
    foreach (SQLManager.Structures.Column column in table.Columns.Values)
    {
        column.X1 = x;
        column.Y1 = y;
        column.X2 = x + TableWidth;
        column.Y2 = y - ColumnHeight;

        //draw an indicator that the column is part of the select columns
        if (column.Selected)
        {
            Shape shape2 = null;
            shape2 = currentPage.DrawOval(column.X1 - 0.2, column.Y1-0.02,
                                          column.X1+0.2, column.Y2);
            shape2.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                        (short)VisRowIndices.visRowFill,
                        (short)VisCellIndices.visFillForegnd).FormulaU =
                        ColumnSelectedColor;
            shape2.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                        (short)VisRowIndices.visRowMisc,
                        (short)VisCellIndices.visComment).FormulaU=
                        "\"Select: "+column.ColumnName+"\"";
            shape2.BringToFront();
        }

        //group by
        if (column.Grouped)
        {
            Shape shape2 = null;
            shape2 = currentPage.DrawRectangle(column.X2, column.Y1 - 0.02,
                                               column.X2 + 0.15, column.Y2);
            shape2.Text = "G";
            shape2.get_CellsSRC((short)VisSectionIndices.visSectionCharacter,
                        (short)VisRowIndices.visRowCharacter,
                        (short)VisCellIndices.visCharacterColor).FormulaForceU
                        = TableColor;
            shape2.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                        (short)VisRowIndices.visRowFill,
                        (short)VisCellIndices.visFillForegnd).FormulaU =
                        ColumnGroupedColor;
            shape2.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                        (short)VisRowIndices.visRowMisc,
                        (short)VisCellIndices.visComment).FormulaU=
                        "\"Group by: "+column.ColumnName+"\"";
            shape2.BringToFront();
        }
```

```
Shape shape1 = null;
shape1 = currentPage.DrawRectangle(column.X1, column.Y1,
                           column.X2, column.Y2);

shape1.Text = column.ColumnName;

//Shape's color
shape1.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                  (short)VisRowIndices.visRowFill,
                  (short)VisCellIndices.visFillForegnd).FormulaU =
                  ColumnColor;
containerShape.ContainerProperties.AddMember(shape1,
    VisMemberAddOptions.visMemberAddExpandContainer);
shape1.BringToFront();

//where condition
if (column.Filtered)
{
    Shape shape2 = null;
    shape2 = currentPage.DrawOval(column.X2 - 0.2, column.Y1 ,
                                        column.X2, column.Y2);
    shape2.Text = "w";
    shape2.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                  (short)VisRowIndices.visRowFill,
                  (short)VisCellIndices.visFillForegnd).FormulaU =
                  ColumnFilteredColor;
    shape2.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                  (short)VisRowIndices.visRowMisc,
                  (short)VisCellIndices.visComment).FormulaU=
                  "\"Where: "+column.ColumnName+"\"";
    shape2.BringToFront();
}

//order by
if (column.Order == SQLManager.Order.Ascending)
{
    Shape shape3 = currentPage.Drop(ascendingTemplate,
                           column.X1+0.76, column.Y2 -0.82);
    shape3.Resize(VisResizeDirection.visResizeDirS,
                           -1.35, VisUnitCodes.visInches);
    shape3.Resize(VisResizeDirection.visResizeDirE,
                           -1.35, VisUnitCodes.visInches);
    shape3.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                  (short)VisRowIndices.visRowFill,
                  (short)VisCellIndices.visFillForegnd).FormulaU =
                  ColumnAscendingColor;
    shape3.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                  (short)VisRowIndices.visRowMisc,
                  (short)VisCellIndices.visComment).FormulaU=
                  "\"Order: Ascending: "+column.ColumnName+"\"";
    containerShape.ContainerProperties.AddMember(shape3,
                  VisMemberAddOptions.visMemberAddDoNotExpand);
    shape3.BringToFront();
}
else if (column.Order == SQLManager.Order.Descending)
{
    Shape shape3 = currentPage.Drop(ascendingTemplate,
```

```
                                column.X1 + 0.76, column.Y2 - 0.79);
            shape3.Resize(VisResizeDirection.visResizeDirS,
                                 -1.35, VisUnitCodes.visInches);
            shape3.Resize(VisResizeDirection.visResizeDirE,
                                -1.35, VisUnitCodes.visInches);
            shape3.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                          (short)VisRowIndices.visRowMisc,
                          (short)VisCellIndices.visComment).FormulaU= "\"Order:
                          Descending: "+column.ColumnName+"\"";
            shape3.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                          (short)VisRowIndices.visRowFill,
                          (short)VisCellIndices.visFillForegnd).FormulaU =
                          ColumnDescendingColor;

            //flip vertical
            Microsoft.Office.Interop.Visio.Selection selection =
             currentPage.CreateSelection(
             Microsoft.Office.Interop.Visio.VisSelectionTypes.visSelTypeEmpty,
             Microsoft.Office.Interop.Visio.VisSelectMode.visSelModeSkipSuper,
             null);
            selection.Select(shape3,
             (short)Microsoft.Office.Interop.Visio.VisSelectArgs.visSelect);
            selection.FlipVertical();

            containerShape.ContainerProperties.AddMember(shape3,
             VisMemberAddOptions.visMemberAddDoNotExpand);
            shape3.BringToFront();
        }

        y -= ColumnHeight;
    }
}
```

5. Método para dibujar un “join”: Este método permite dibujar líneas para relacionar dos tablas o dos consultas.

```
private void AddJoinConnections(SQLManager.ISQLStatement Select,
                    System.Collections.Hashtable shapesArray)
{
    foreach (SQLManager.Structures.Join join in Select.Joins)
    {
        Microsoft.Office.Interop.Visio.Shape connector =
            VisioUtils.ConnectWithDynamicGlueAndConnector(
(Microsoft.Office.Interop.Visio.Shape)shapesArray[join.First.Key], //Source Shape
(Microsoft.Office.Interop.Visio.Shape)shapesArray[join.Second.Key]); //Target

        switch (join.Type)
        {
            case SQLManager.Structures.JoinType.Inner:
                connector.CellsU["EndArrow"].FormulaU = "5";
                connector.CellsU["BeginArrow"].FormulaU = "5";
                connector.CellsU["LineColor"].FormulaU = InnerJoinColor;
                connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                    (short)VisRowIndices.visRowMisc,
```

```
                (short)VisCellIndices.visComment).FormulaU=
                 "\"Inner Join: \"";
            connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                (short)VisRowIndices.visRowLine,
                (short)VisCellIndices.visLineWeight).FormulaForce =
                LineWeight;
            break;
        case SQLManager.Structures.JoinType.Right:
            connector.CellsU["EndArrow"].FormulaU = "7";
            connector.CellsU["LineColor"].FormulaU = RightJoinColor;
            connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                (short)VisRowIndices.visRowMisc,
                (short)VisCellIndices.visComment).FormulaU=
                 "\"Right Join: \"";
            connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                (short)VisRowIndices.visRowLine,
                (short)VisCellIndices.visLineWeight).FormulaForce =
                 LineWeight;
            break;
        case SQLManager.Structures.JoinType.Left:
            connector.CellsU["BeginArrow"].FormulaU = "14";
            connector.CellsU["LineColor"].FormulaU = LeftJoinColor;
            connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                (short)VisRowIndices.visRowMisc,
                (short)VisCellIndices.visComment).FormulaU=
                "\"Left Join: \"";
            connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                (short)VisRowIndices.visRowLine,
                (short)VisCellIndices.visLineWeight).FormulaForce =
                LineWeight;
            break;
        case SQLManager.Structures.JoinType.Full:
            connector.CellsU["EndArrow"].FormulaU = "39";
            connector.CellsU["BeginArrow"].FormulaU = "39";
            connector.CellsU["LineColor"].FormulaU = FullJoinColor;
            connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                (short)VisRowIndices.visRowMisc,
                (short)VisCellIndices.visComment).FormulaForce =
                "\"Full Join: \"";
            connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                (short)VisRowIndices.visRowLine,
                (short)VisCellIndices.visLineWeight).FormulaU= LineWeight;
            break;
        case SQLManager.Structures.JoinType.Union:
            connector.CellsU["EndArrow"].FormulaU = "30";
            connector.CellsU["BeginArrow"].FormulaU = "30";
            connector.CellsU["LineColor"].FormulaU = UnionColor;
            connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                (short)VisRowIndices.visRowMisc,
                (short)VisCellIndices.visComment).FormulaU= "\"Union: \"";
            connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                (short)VisRowIndices.visRowLine,
                (short)VisCellIndices.visLineWeight).FormulaU= LineWeight;
            break;
        case SQLManager.Structures.JoinType.Intersec:
            connector.CellsU["EndArrow"].FormulaU = "22";
            connector.CellsU["BeginArrow"].FormulaU = "22";
            connector.CellsU["LineColor"].FormulaU = IntersectColor;
```

```
                    connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                        (short)VisRowIndices.visRowMisc,
                        (short)VisCellIndices.visComment).FormulaU= "\"Intersec: \"";
                    connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                        (short)VisRowIndices.visRowLine,
                        (short)VisCellIndices.visLineWeight).FormulaU= LineWeight;
                    break;
                case SQLManager.Structures.JoinType.Except:
                    connector.CellsU["EndArrow"].FormulaU = "23";
                    connector.CellsU["BeginArrow"].FormulaU = "23";
                    connector.CellsU["LineColor"].FormulaU = ExceptColor;
                    connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                        (short)VisRowIndices.visRowMisc,
                        (short)VisCellIndices.visComment).FormulaU= "\"Except: \"";
                    connector.get_CellsSRC((short)VisSectionIndices.visSectionObject,
                        (short)VisRowIndices.visRowLine,
                        (short)VisCellIndices.visLineWeight).FormulaU= LineWeight;
                    break;
                default:
                    break;
            }

        }

    }
```

## 8.5 Código SQL utilizado en la prueba de validación de resultados

Este anexo muestra las instrucciones de SQL utilizadas en la prueba:

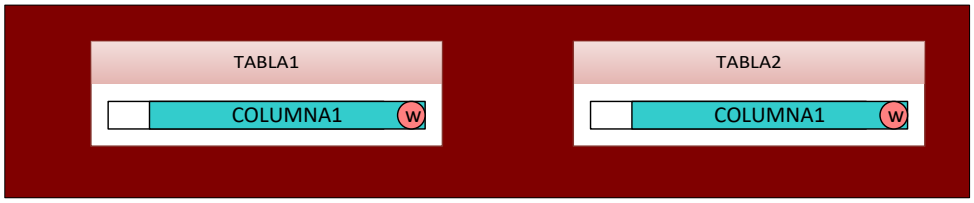
```
DELETE  Tabla1
FROM    Tabla2
WHERE   Tabla1.Columna1 = Tabla2.Columna1
AND Tabla1.Columna1 = 'A';

INSERT  INTO Tabla1
SELECT  'A' ,
T2.Columna1 ,
T3.Columna2 ,
T2.Columna3
FROM    Tabla2 AS T2
INNER JOIN Tabla3 AS T3 ON T2.Columna1 = T3.Columna1
WHERE   T2.Columna1 LIKE '2%'
ORDER BY T2.Columna1 ,
T3.Columna2;

WITH    EmployeeTemp ( EmpID, LastName, FirstName, Phone, Address, City, StateProvince,
PostalCode, CurrentFlag )
AS ( SELECT   e.BusinessEntityID ,
c.LastName ,
c.FirstName ,
pp.PhoneNumber ,
a.AddressLine1 ,
a.City ,
sp.StateProvinceCode ,
a.PostalCode ,
e.CurrentFlag
FROM     HumanResources.Employee e
INNER JOIN Person.BusinessEntityAddress AS bea ON e.BusinessEntityID = bea.BusinessEntityID
INNER JOIN Person.Address AS a ON bea.AddressID = a.AddressID
INNER JOIN Person.PersonPhone AS pp ON e.BusinessEntityID = pp.BusinessEntityID
INNER JOIN Person.StateProvince AS sp ON a.StateProvinceID = sp.StateProvinceID
INNER JOIN Person.Person AS c ON e.BusinessEntityID = c.BusinessEntityID
)
INSERT  INTO HumanResources.Employee
SELECT  EmpID ,
LastName ,
FirstName ,
Phone ,
Address ,
City ,
StateProvince ,
PostalCode ,
CurrentFlag
FROM    EmployeeTemp;

SELECT  *
FROM    Tabla1
LEFT JOIN Tabla2 ON Tabla1.Columna1 = Tabla2.Columna1
AND Tabla1.Columna2 = Tabla2.Columna2
AND Tabla1.Columna3 = Tabla2.Columna3;

INSERT TOP ( 10 )
INTO    HumanResources.Employee2
SELECT  e.BusinessEntityID ,c.LastName ,c.FirstName ,pp.PhoneNumber ,
a.AddressLine1 ,
```

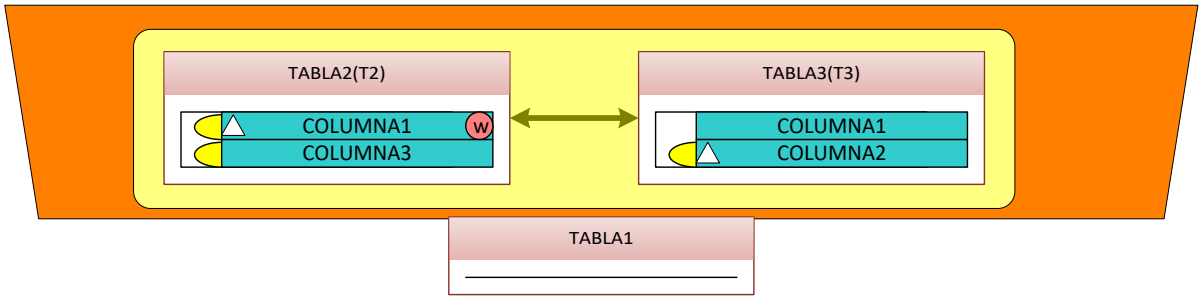
```
a.City ,
sp.StateProvinceCode ,
a.PostalCode ,
e.CurrentFlag
FROM    HumanResources.Employee1 e
INNER JOIN Person.BusinessEntityAddress AS bea ON e.BusinessEntityID = bea.BusinessEntityID
INNER JOIN Person.Address AS a ON bea.AddressID = a.AddressID
INNER JOIN Person.PersonPhone AS pp ON e.BusinessEntityID = pp.BusinessEntityID
INNER JOIN Person.StateProvince AS sp ON a.StateProvinceID = sp.StateProvinceID
INNER JOIN Person.Person AS c ON e.BusinessEntityID = c.BusinessEntityID;

SELECT  msc ,
dbo.ucmsc(msc) AS fmsc
INTO    #mscf
FROM    ( SELECT    tsc AS msc
FROM      id.ct1
UNION
SELECT    site_cd
FROM      id.ct
) src;

UPDATE  lt
SET     cpn = 'S' + RIGHT(pn, LEN(pn) - 1)
FROM    lt trns WITH ( NOLOCK )
INNER JOIN  mfr ON trns.ssc = mfr.ssc
AND trns.rt_nm = mfr.mfg_rt_nm
INNER JOIN rtt ON mfr.rti = rtt.rti
INNER JOIN sfp ON trns.pn = sfp.ild
WHERE   trns.rud >= @vActionDT
AND rtt.rtn = 'abc'
AND sfp.icn = 'cde';

INSERT  INTO dbo.EmployeeSales
( LastName ,
FirstName ,
CurrentSales
)
OUTPUT  INSERTED.LastName ,
INSERTED.FirstName ,
INSERTED.CurrentSales
INTO @MyTableVar
SELECT  c.LastName ,
c.FirstName ,
sp.SalesYTD
FROM    Sales.SalesPerson AS sp
INNER JOIN Person.Person AS c ON sp.BusinessEntityID = c.BusinessEntityID
WHERE   sp.BusinessEntityID LIKE '2%'
ORDER BY c.LastName ,
c.FirstName DESC;

SELECT  t1.c1 ,
t2.c2
FROM    t1
FULL JOIN t2 ON t1.c3 = t2.c4
AND t1.c5 = t2.c6
RIGHT JOIN ( SELECT c7
FROM   T3
JOIN t5 ON t3.c1 = t5.c2
) t4 ON t4.c7 = t2.c6
ORDER BY t1.otra;
```

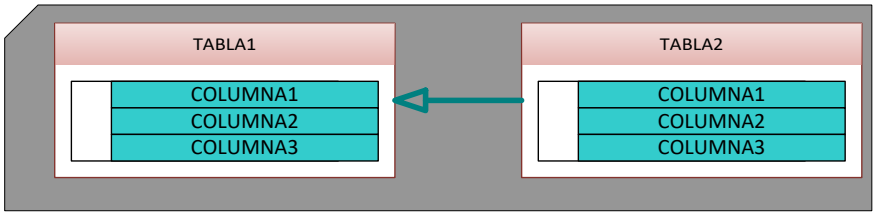
```
DELETE  FROM Tabla1
WHERE   Columna1 = 'Dato1';

--Update t_asd table
UPDATE  t_asd
SET     t_asd.r_t_i = 'Y' ,
t_asd.r_t_gm = rvlt.r_t_gm ,
t_asd.s_t_g = rvlt.s_t_g ,
t_asd.e_t_g = rvlt.e_t_g ,
t_asd.s_t_l_d = rvlt.s_t_l_d
FROM    t_asd
JOIN rvlt ON ( rvlt.id = t_asd.fr_id
AND rvlt.l_n = t_asd.l_n
)
WHERE   rvlt.r_t_gm IS NOT NULL
AND t_asd.s_v_c = 'abc'
AND t_asd.r_t_i = 'N';

SELECT  Columna1
FROM    Tabla1
UNION
SELECT  Columna2
FROM    Tabla2;

--correlated delete
DELETE  Tabla1
FROM    Tabla2
WHERE   Tabla1.Columna1 = Tabla2.Columna1
AND Tabla1.Columna1 = 'A';

SELECT  *
FROM    Tabla1
LEFT JOIN Tabla2 ON Tabla1.Columna1 = Tabla2.Columna1
AND Tabla1.Columna2 = Tabla2.Columna2
AND Tabla1.Columna3 = Tabla2.Columna3;

UPDATE  Tabla1
SET     Columna1 = ( SELECT ( Columna4 * .90 )
FROM   Tabla2
WHERE  Tabla2.Columna2 = Tabla1.Columna2
AND Tabla2.Columna3 = 'B');

-- Insert values into the table variable.
INSERT  INTO @MyTableVar
( LocationID ,
CostRate ,
ModifiedDate
)
SELECT  LocationID ,
CostRate ,
GETDATE()
FROM    Production.Location
WHERE   CostRate > 0;

-- Specify the remote data source in the FROM clause using a four-part name
-- in the form linked_server.catalog.schema.object.
INSERT  INTO MyLinkServer.AdventureWorks2008R2.HumanResources.Department
( Name ,
GroupName
)
VALUES  ( N'Public Relations' ,N'Executive General and Administration');
```

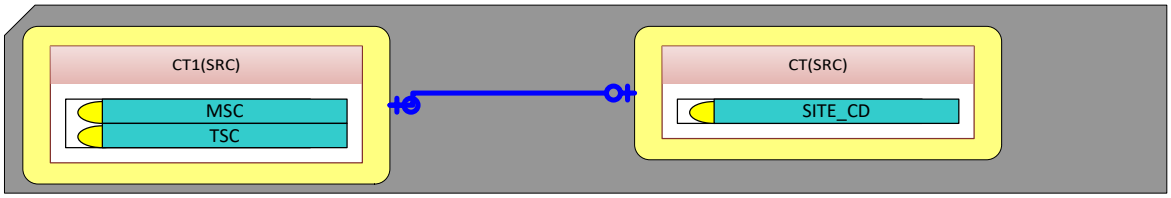
## 8.6 Visualización utilizada en la prueba de validación de resultados

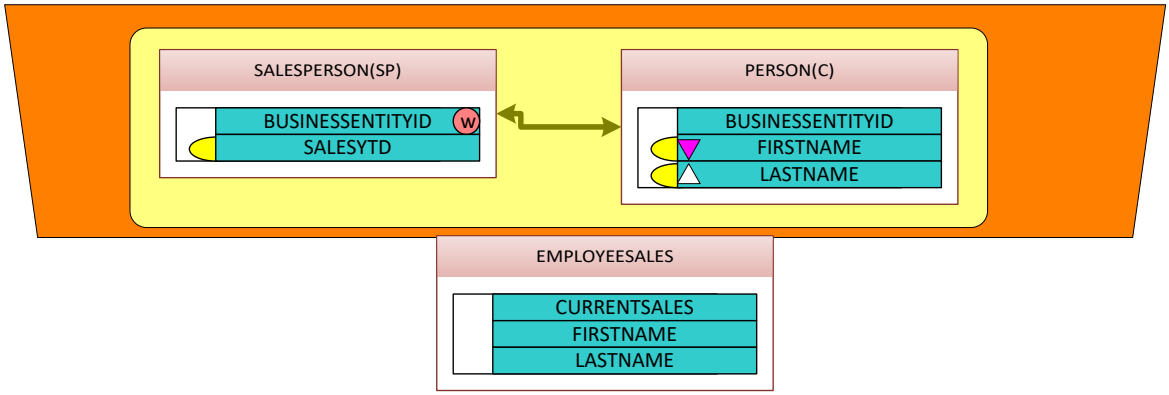
Este anexo muestra los diagramas generados por la herramienta de visualización para las instrucciones de SQL (ver anexo 8.5) utilizadas en la prueba.



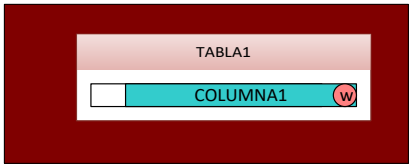
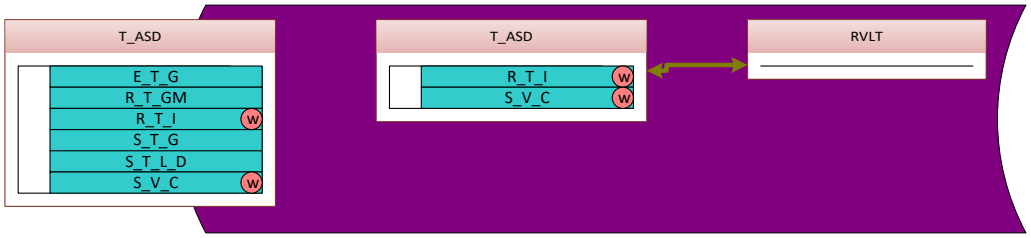


TABLA1
COLUMNA1
W


T_ASD
E_T_G
R_T_GM
R_T_I
S_T_G
S_T_L_D
S_V_C
W
T_ASD
R_T_I
S_V_C
W
W
RVLT

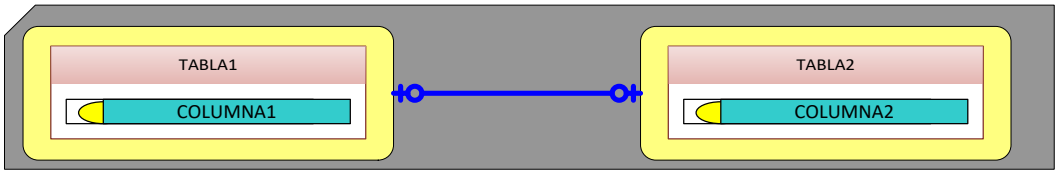


TABLA1
COLUMNA1
TABLA2
COLUMNA2

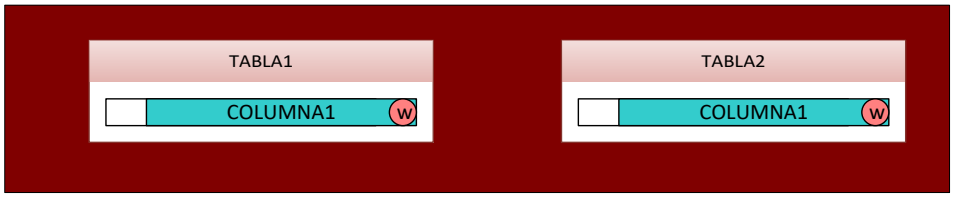


TABLA1
COLUMNA1
W
TABLA2
COLUMNA1
W

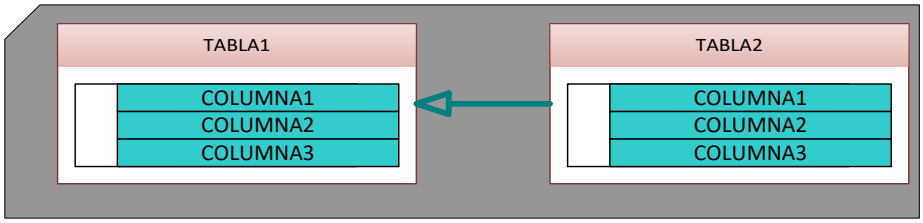


TABLA1
COLUMNA1
COLUMNA2
COLUMNA3
TABLA2
COLUMNA1
COLUMNA2
COLUMNA3

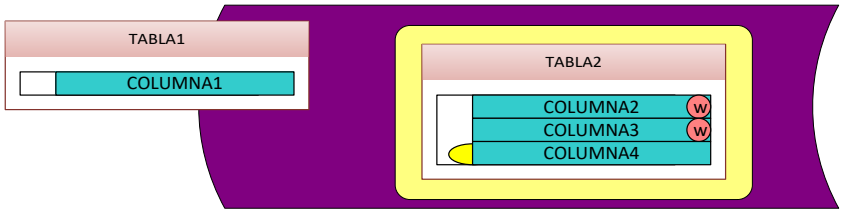
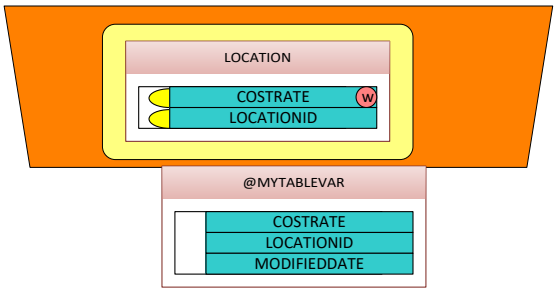
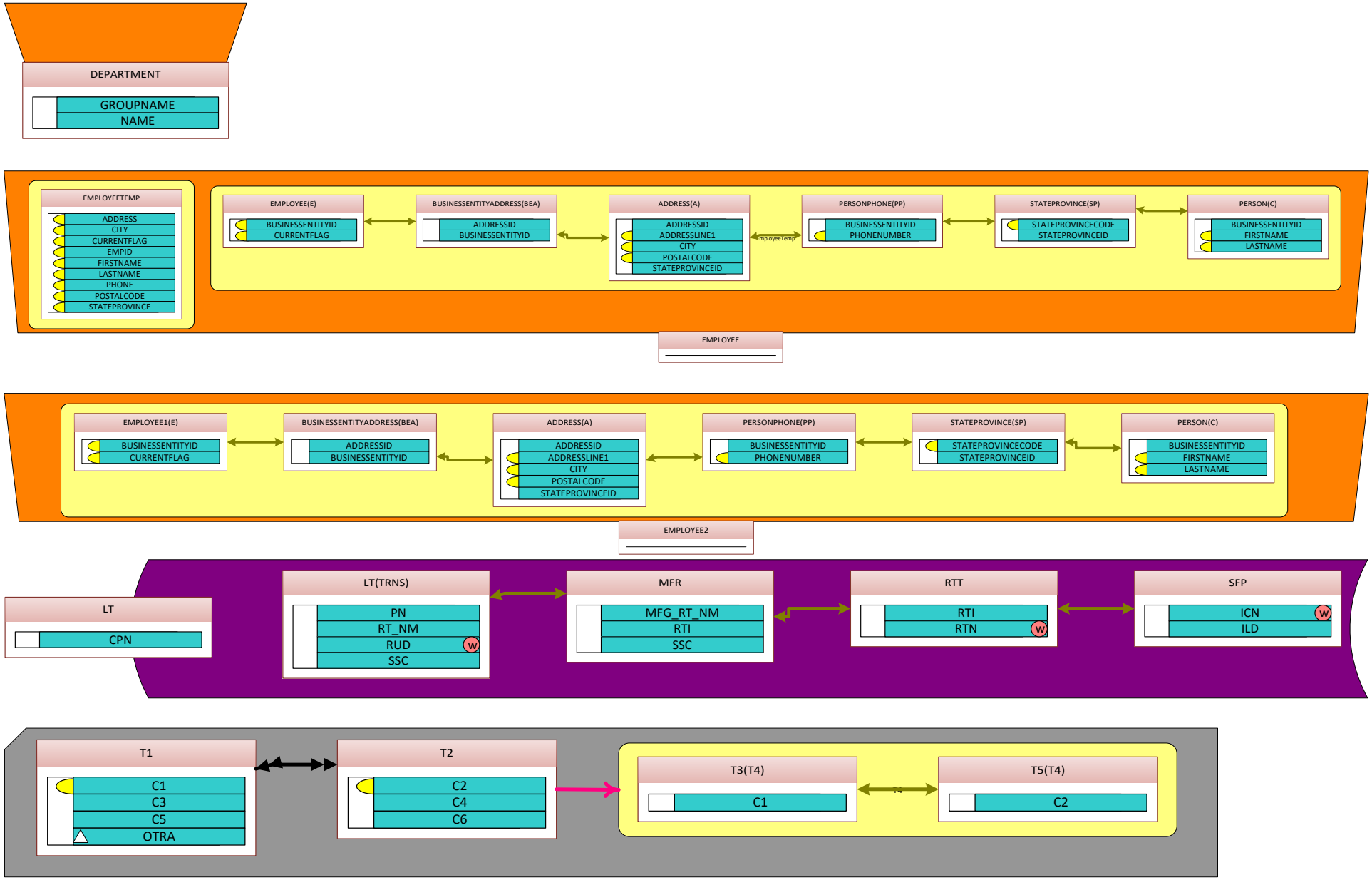


TABLA1
COLUMNA1
TABLA2
COLUMNA2
COLUMNA3
COLUMNA4
W
W


LOCATION
COSTRATE
LOCATIONID
W
@MYTABLEVAR
COSTRATE
LOCATIONID
MODIFIEDDATE


DEPARTMENT
GROUPNAME
NAME
EMPLOYEETEMP
ADDRESS
CITY
CURRENTFLAG
EMPID
FIRSTNAME
LASTNAME
PHONE
POSTALCODE
STATEPROVINCE
EMPLOYEE(E)
BUSINESSENTITYID
CURRENTFLAG
BUSINESSENTITYADDRESS(BEA)
ADDRESSID
BUSINESSENTITYID
ADDRESS(A)
ADDRESSID
ADDRESSLINE1
CITY
POSTALCODE
STATEPROVINCEID
employee1 emp
PERSONPHONE(PP)
BUSINESSENTITYID
PHONENUMBER
STATEPROVINCE(SP)
STATEPROVINCECODE
STATEPROVINCEID
PERSON(C)
BUSINESSENTITYID
FIRSTNAME
LASTNAME
EMPLOYEE
EMPLOYEE1(E)
BUSINESSENTITYID
CURRENTFLAG
BUSINESSENTITYADDRESS(BEA)
ADDRESSID
BUSINESSENTITYID
ADDRESS(A)
ADDRESSID
ADDRESSLINE1
CITY
POSTALCODE
STATEPROVINCEID
PERSONPHONE(PP)
BUSINESSENTITYID
PHONENUMBER
STATEPROVINCE(SP)
STATEPROVINCECODE
STATEPROVINCEID
PERSON(C)
BUSINESSENTITYID
FIRSTNAME
LASTNAME
EMPLOYEE2
LT
CPN
LT(TRNS)
PN
RT_NM
RUD
W
SSC
MFR
MFG_RT_NM
RTI
SSC
RTT
RTI
RTN
W
SFP
ICN
W
ILD
T1
C1
C3
C5
OTRA
T2
C2
C4
C6
T3(T4)
C1
T4
T5(T4)
C2

# 9 Referencias


Biggerstaff, T. J. (1993). The concept assignment problem in program understanding. Microsoft Research.

Brooks, R. (1978). Using A Behavioral Theory of Program Comprehension in Software Engineering. *Proceedings of the 3rd international conference on Software engineering* (págs. 196-201). NJ, USA: IEEE Press.

DB-Engines.com. (July de 2016). *DBMS popularity broken down by database model*. Obtenido de DB-Engines.com: http://db-engines.com/en/ranking_categories

Gatterbauer, W. (2011). Databases will Visualize Queries too. *International Conference on Very Large Data Bases.*

Indeed.com. (2016). *The 9 Most In-Demand Programming Languages of 2016*. Obtenido de Coding Dojo: http://www.codingdojo.com/blog/9-most-in-demand-programming-languages-of-2016/

Marcus, A., Feng, L., & Maletic, J. (2003). 3D Representations for Software Visualization. *Proceedings of the 2003 ACM symposium on Software visualization.* San Diego, California.

Moody, D. L. (2009). The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 756-779.

Moody, D., & van Hillegersberg, J. (2009). *Evaluating the Visual Syntax of UML: An Analysis of the Cognitive Effectiveness of the UML Family of Diagrams.* Enschede, Netherlands: Department of Information Systems & Change Management,University of Twente.

Paivio, A. (1990). *Mental Representations: A Dual Coding Approach.* OXford University Press. Obtenido de plato.stanford.edu: https://plato.stanford.edu/entries/mental-imagery/theories-memory.html

Price, B. A., & Baecker , R. (1994). A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing* , (págs. 211-266).

Scanlan, D. A. (1989). Structured Flowcharts Outperform Pseudocode: An Experimental Comparison. *IEEE Software*, 28-36.

Schneider, G., Sedlmeyer, R. L., & Kearney, J. (1981). On the complexity of measuring software complexity. *Proceedings of the May 4-7* . Chicago, Illinois: National Computer Conference.

Schneider, M., & Sedlmeyer, R. L. (1981). On the complexity of measuring software complexity. 317-322.

Shneiderman, B. (1996). The Eyes Have It A Task by Data Type Taxonomy for Information Visualizations. *Proceedings of IEEE Visual Languages*, (págs. 336-343).

Shneiderman, B., Mayer, R., McKay, D., & Heller, P. (1977). Experimental investigation of the utility of detailed flowcharts in programming. *Association for Computing Machinery*, 373-381.

Teixeira, M. d. (2014). PoN-S: A Systematic Approach for Applying the Physics of Notation (PoN). *Brazilian Research Funding Agency*.