ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UNA SOLUCIÓN PARA RECOLECCIÓN DE DATOS Y GENERACIÓN DE REPORTES WEB DE LOS SISTEMAS DE CO_2 Y REFRIGERACIÓN PARA EL ÁREA DE MANUFACTURA DE CERVECERÍA NACIONAL C.N. PLANTA QUITO

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN ELECTRÓNICA Y REDES DE INFORMACIÓN

MARÍA EUGENIA LIVE ALMEIDA

maria.live@epn.edu.ec

DIRECTOR: ING. JULIO CÉSAR CAIZA MSC.

julio.caiza@epn.edu.ec

QUITO, agosto 2016

DECLARACIÓN

Yo, María Eugenia Live Almeida, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

María Eugenia Live Almeida

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por María Eugenia Live, bajo mi supervisión.

_____

Ing. Julio César Caiza, MSc.
DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

A Dios por darme la vida, la sabiduría y la fuerza cada día.

A mi padre y madre por siempre impulsarme a salir adelante sin importar las circunstancias, por su generosidad y su apoyo incondicional que me han permitido culminar esta etapa de mi vida.

A mis hermanas que siempre supieron alegrarme aún en los días más oscuros y que de una u otra manera colaboraron conmigo en este proceso.

A mi familia que me recibe con las puertas abiertas y me apoya cuando emprendo mis proyectos.

A mi director de tesis, Ing. Julio César Caiza, Msc por su compromiso y paciencia para guiarme en el desarrollo de este trabajo de titulación

A mis amigos que con los años, se han vuelto como hermanos.

EUGENIA

DEDICATORIA

A mi familia que es el motor de mi vida.

EUGENIA

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE SEGMENTOS DE CÓDIGO

ÍNDICE DE TABLAS

RESUMEN

En el proyecto presentado se obtiene un conjunto de datos de una base de datos industrial denominada InSQL, la cual almacena la información los sensores de la maquinaria a través de los PLC (*Programmable Logic Controller*), en archivos históricos y la recupera mediante una base de datos intermedia en un servidor SQL (*Structured Query Language*) utilizando procedimientos almacenados

También se diseña una aplicación web a la que se puede acceder desde la intranet, en la cual se presenta el conjunto de datos obtenido en forma de reportes y gráficas. Se utiliza SCRUM como metodología de desarrollo, y el patrón DAO (*Data Access Object*) como intermediario entre la base de datos y las clases de la aplicación. La aplicación consta de un módulo de autenticación, un módulo de reportes fijos y variables, un módulo de gráficas, un módulo de administración y un módulo de conexión a la base de datos.

El módulo de autenticación verifica credenciales y establece variables de sesión por usuario.

El módulo de reportes permite al usuario seleccionar un reporte y un rango de fecha y hora, y usa la librería *Crystal Reports* para presentar el conjunto de datos en formato de reporte, en el caso de un reporte variable se permite al usuario seleccionar hasta cuatro variables.

El módulo de gráficas permite al usuario seleccionar un rango de fecha y hora, una escala de horas, días o meses, un tipo continuo o discreto y hasta cuatro variables. Con esta información el sistema muestra gráficas usando los datos históricos de la información almacenada en la base de datos.

El módulo de administración permite a los administradores crear, actualizar y eliminar usuarios y sistemas. Los sistemas son un conjunto de maquinaria que realiza una función determinada.

El módulo de conexión a la base de datos permite acceder a la base y recuperar los datos.

Se realizaron pruebas unitarias para comprobar segmentos de código y pruebas de carga que midieran el rendimiento para un acceso simultáneo de usuarios.

PRESENTACIÓN

En la actualidad el conjunto de datos de los procesos en la industria entrega mucha información para realizar prevención y corrección en la maquinaria. Para realizar un análisis óptimo, se requiere que los datos estén completos, debidamente almacenados, que puedan ser accedidos y presentados en una interfaz de una manera adecuada.

Se espera que con este trabajo, se comience a cambiar la forma en la que los operadores de planta recolectan datos manualmente, y se pueda pasar a una era modernizada en la que se aproveche tiempo y recursos para mejorar la calidad de operación de maquinaria.

En el capítulo 1 se desarrolla el marco teórico con los conceptos básicos para cubrir el presente trabajo.

En el capítulo 2 se realiza el análisis del estado actual del entorno donde se desarrolla el proyecto, se definen requerimientos y se propone una arquitectura para la solución.

En el capítulo 3 se desarrolla el diseño e implementación de la solución planteada en el capítulo anterior, se diseña e implementa los procedimientos almacenados, clases e interfaces de usuario.

En el capítulo 4 se detallan las pruebas realizadas a la solución y los resultados de las mismas.

En el capítulo 5 se presenta un conjunto de conclusiones y recomendaciones para culminar con el desarrollo del proyecto.

En los anexos se presenta la discriminación de variables, procedimientos almacenados, clases, archivos XML, hojas de estilo y pruebas asociados al proyecto.

CAPÍTULO 1

MARCO TEÓRICO

1.1. APLICACIONES WEB [1]

Una aplicación web es un tipo de aplicación cliente-servidor, en el que el cliente es un explorador web, el servidor es de tipo web y se comunican mediante el protocolo HTTP (*Hipertext Transfer Protocol*).

HTTP es un protocolo sin estado que utiliza el paradigma de petición/respuesta para la comunicación ente el cliente y el servidor, en el cual la aplicación cliente realiza una petición al servidor que devuelve una respuesta hacia el cliente.

1.1.1. CLIENTE WEB

El cliente web es un programa (navegador) con el que interacciona el usuario y mediante este solicita al servidor el envío de recursos e interpretarlos para ser presentados en el programa cliente.

Entre las tecnologías que se usan para programar se tiene HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*), lenguajes de script.

1.1.2. SERVIDOR WEB

Un servidor web procesa las peticiones entrantes, genera mensajes de respuesta y transmite esas respuestas a los solicitantes; el módulo de red es responsable de recibir y transmitir las solicitudes en la red. Una solicitud entrante debe pasar por el módulo de resolución de direcciones, que es responsable de analizar y realizar un proceso previo de las peticiones. En el proceso previo se realiza un *hosting* virtual en caso de que el servidor almacene múltiples dominios, un mapeo de direcciones para determinar si la petición busca contenido estático o dinámico y un proceso de autenticación si el recurso al que se desea acceder está protegido

1.1.2.1. SERVIDOR WEB IIS [1]

IIS (*Internet Information Server*) es desarrollado por Microsoft y conforma un conjunto de servicios que admiten la creación, configuración y administración de sitios web y de funciones de internet. El servidor dispone de una tecnología

conocida como ISAPI (*Internet Server Application Program Interface*) que ofrece un conjunto de funciones para crear DDL (*Dynamic Link Library*) que se ejecutan cuando el cliente solicita un archivo con determinada extensión y se procesan como un script que se ejecuta en el servidor.

El sistema ASP.NET es una DLL de este tipo, la cual es invocada para los archivos con extensión .asp.

1.1.3. ASP.NET [2]

El presente proyecto utiliza la plataforma de desarrollo de ASP.NET (*Active Server Pages*), que provee un modelo de programación, una infraestructura de software y varios servicios para desarrollar aplicaciones web.

Las aplicaciones generadas son de código compilado, utilizan componentes reusables y controles presentes en el *framework* de .NET. El código de la aplicación puede escribirse usando C#, *visual basic, Jscript, J#.*

ASP. NET cuenta con dos modelos que permiten el desarrollo de aplicaciones n el entorno de *Visual Studio*; un modelo de formularios web y un modelo de componentes.

El modelo de formularios web extiende el modelo manejado por eventos a las aplicaciones web en el cual el navegador realiza una petición al servidor web y el servidor retorna una página HTML en respuesta. El modelo permite almacenar el estado de aplicación utilizando el estado de sesión ASP.NET y la infraestructura en el lado el servidor para hacer un seguimiento de la información recolectada en la sesión.

El modelo de componente provee bloques de páginas ASP.NET. Es un modelo de objetos que describe el equivalente en el lado del servidor de casi todos los elementos y etiquetas HTML y los controles de servidor, por ejemplo los controles *Calendar* o *Gridview*, que permiten desarrollar una interfaz de usuario completa.

1.1.3.1. FRAMEWORK .NET [2]

El *framework* .NET es una tecnología que permite compilar y ejecutar aplicaciones y servicios web. El diseño de este *framework* tiene como objetivo proporcionar un

entorno de programación orientada a objetos en el cual el código se pueda almacenar y ejecutar en forma local, distribuida en la red o en forma remota.

Además provee un entorno de ejecución que permite versionar el software, reducir los conflictos del despliegue y promover una ejecución segura del código. Este entorno también previene los problemas de rendimiento en caso de que se usen scripts.

El *framework* de .NET define los componentes mencionados a continuación:

1. CLR (*Common Language Runtime*)

   El CLR realiza funciones de administración de memoria, manejo de excepciones, depuración, revisión de seguridad, ejecución de código, verificación y compilación. El código manejado directamente por el CLR se conoce como código administrado. Cuando el código administrado es compilado, el compilador convierte el código fuente en un código de lenguaje intermedio independiente (IL).

2. Librería de clases

   La librería de clases contiene una colección de tipos, clases, interfaces, estructuras y valores enumerados que son reusables.

3. Especificaciones de lenguajes comunes

   Contiene las especificaciones para los lenguajes que soportan .NET y la implementación de integración de lenguajes.

4. Sistema de tipos comunes

   Provee una guía para declarar, utilizar y administrar los tipos en tiempo de ejecución.

5. *Metadata* y *Assemblies*

   La metadata es la información en binario que describe el programa, que puede ser almacenada en un archivo portable ejecutable o en la memora. Un assembly es una unidad lógica que consiste del tipo de metadata, código IL y un conjunto de recursos como por ejemplo archivos de imagen.

6. *Windows Forms*

Los *forms* contiene la representación gráfica de cualquier ventana desplegada en la aplicación.

7. ADO.NET

Es la tecnología utilizada para trabajar con datos y bases de datos. Provee acceso a la fuente de datos y permite la conexión para recuperar, manipular y actualizar datos.

8. *LINQ*

Permite capacidades de consulta a lenguajes .NET usando una sintaxis similar al lenguaje de consulta de SQL.

1.1.3.2. ESTADO DE SESIÓN [3]

El estado de sesión en ASP.NET permite almacenar en variables de sesión la información de un usuario mientras navega por las diferentes páginas de una aplicación web. Se desea mantener estos valores debido a que el protocolo HTTP procesa cada solicitud de forma independiente sin guardar información de los valores de las variables para solicitudes anteriores.

Las variables de sesión se almacenan usando la propiedad *Session* que almacena las variables en una colección indexada por nombre, las variables pueden ser de cualquier tipo, incluso se pueden almacenar objetos en una variable de sesión. Las variables de sesión se mantienen durante el tiempo que dura la sesión que también puede ser configurado.

1.2. METODOLOGÍA DE DESARROLLO SCRUM [4]

Scrum se basa en la teoría de control de procesos empírica o empirismo. El empirismo asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo. [4]

Los actores de Scrum se definen como: *Scrum master,* que se encarga de las reuniones con el *product owner* y de guiar al equipo para cumplir con la metodología. El *product owner,* que es el cliente que solicita el software y

establece los requerimientos. El equipo de Scrum, está constituido por grupos de entre tres y nueve miembros, sin embargo es flexible para permitir el trabajo de un solo desarrollador que desee seguir la metodología. Con un solo miembro en el equipo se aumentan las iteraciones y se reduce las ganancias en productividad [3].

Los requerimientos se obtienen a partir de historias de usuario, que luego se reúne en una lista de requerimientos. Esta lista permitirá definir las tareas que conformarán el la tabla de planificación de *sprints* donde se define la iteración de cada *sprint,* el autor y el número de horas requerido. A esta lista se la conoce como *Scrum Backlog*.

Un *sprint* es un bloque temporal corto, usualmente de 2 semanas, aunque dependiendo de las necesidades puede extenderse hasta 4 semanas como límite máximo. Cada *sprint* debe proporcionar un resultado completo, un incremento del producto final que pueda presentarse al *product owner* durante las reuniones. El incremento del producto debe ser utilizable y potencialmente desplegable.

Es más conveniente si la duración de los sprints es consistente a lo largo del esfuerzo de desarrollo. Cada nuevo Sprint comienza inmediatamente después de la finalización del Sprint previo

1.3. PATRÓN DAO

"El patrón DAO (*Data Access Object*) es una arquitectura orientada a objetos bien estructurada está llena de patrones. La calidad de un sistema orientado a objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos. Los patrones conducen a arquitecturas más pequeñas, más simples y más comprensibles". [3]

Los patrones de diseño se enfocan en que el modelo de datos sea adaptable para las diferentes vistas y tenga un mecanismo de control que establezca el formato de presentación de los datos recuperados.

El patrón de diseño DAO define una relación entre la lógica de presentación y la de negocio con la capa de datos mediante clases DAO. En la Figura 1.1 se puede ver que DAO devuelven al usuario un conjunto de datos obtenidos

realizando consultas a la base, de manera que el cliente solo recibe un objeto de transferencia que almacena los resultados los cuales se devuelven al usuario a través de la aplicación.

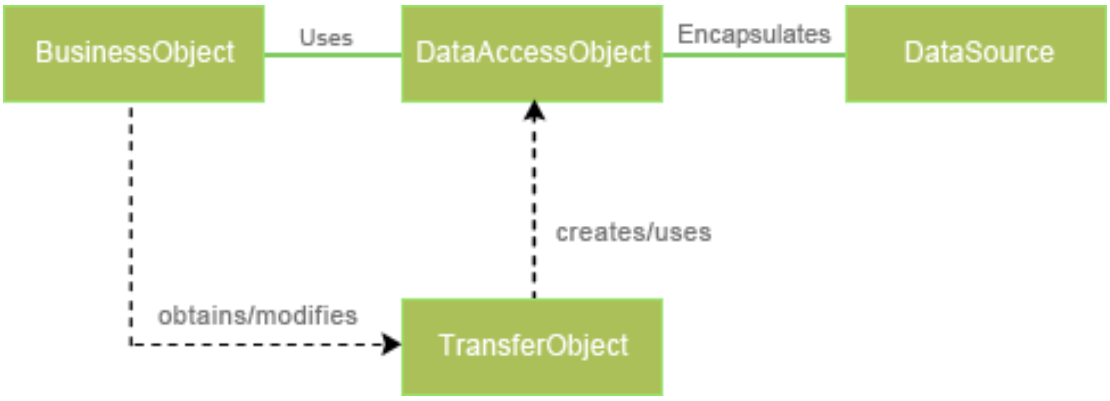

Figura 1.1 Función de los objetos DAO

1.4. BASE DE DATOS [5]

Se define a una base de datos como una colección de elementos organizados en tablas a las que se puede acceder y modificar. Para la base de datos se tiene dos tipos de modelos: el modelo entidad-relación y el modelo relacional.

El modelo entidad-relación consta de entidades y las relaciones entre ellas. Las entidades son abstracciones de un objeto de la realidad y se definen con un conjunto de atributos, las relaciones son las asociaciones entre varias entidades. Uno de los atributos de la entidad debe ser único para que funcione como una clave.

El modelo relacional se compone de un conjunto de tablas que representan los datos, cada columna de cada tabla corresponde a un atributo del objeto y cada fila de la tabla corresponde a una relación entre los componentes de la misma.

El modelo entidad-relación puede implementarse como modelo relacional y con este modelo definir el esquema de la base de datos que luego se implementará.

1.4.1. SQL

Se define a SQL (*Structured Query Language*) como una forma de comunicarse con una base de datos relacional que permite definir, ver, modificar y controlar los datos mediante instrucciones conocidas como consultas. [3]

La estructura básica de una consulta se puede ver en el Segmento de código 1.1 la cual requiere del empleo de las cláusulas como `select`, `from` y `where`

```
select A1, A2,…, An
from r1, r2,…, rm
where P
```

Segmento de código 1.1 Estructura básica de una consulta

La cláusula de `Select` corresponde a una operación de proyección de álgebra relacional y permite listar los atributos del resultado de la consulta, la cláusula `from` es una operación producto cartesiano y lista las tablas involucradas en la consulta; la cláusula `where` corresponde al predicado selección y establece las condiciones de la consulta.

Las consultas pueden ser simples o anidadas. Las consultas simples son aquellas que se realizan a una sola tabla mientras que las consultas anidadas son consultas realizadas a dos o más tablas.

En ciertas ocasiones se necesita realizar consultas a un servidor externo para lo cual se usa la instrucción `openquery`. Esta instrucción permite ejecutar una consulta al servidor vinculado y presentar los resultados en el servidor local. El formato de la sentencia se puede apreciar en el Segmento de código 1.2.

```
SELECT * FROM OPENQUERY(MyLinkedServer, 'SELECT * FROM tabla
WHERE condición ' )
```

Segmento de código 1.2 Esquema del uso de la sentencia openquery

Mediante esta consulta se puede vincular a los dos servidores de base de datos que se tienen en el proyecto presente. Para una mejor comprensión se presenta un ejemplo de una consulta, considere que se tiene un servidor vinculado conectado a otro servidor SQL como se muestra en la Figura 1.2.

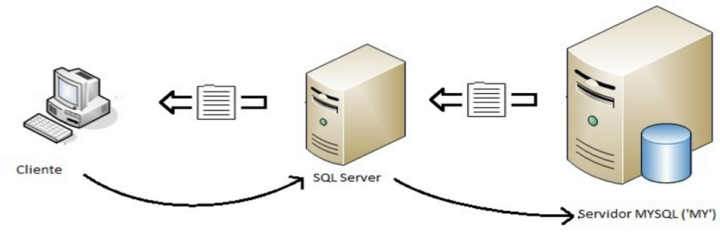

Figura 1.2 Esquema de conexión de servidor SQL a servidor vinculado

La consulta con openquery permitirá cargar una tabla intermedia con los datos de venta almacenados en el servidor vinculado. La consulta se muestra en el Segmento de código 1.3.

```
SELECT *
FROM openquery(MY, 'SELECT * FROM Ventas.CABECERAS
WHERE date_format(FECHA, ''%Y-%m-%d'') >= '''2012-12-01''')
```

Segmento de código 1.3 Ejemplo de consulta usando openquery

En esta consulta se recupera los datos de la tabla CABECERAS, que esta almacenada en el servidor vinculado, para las cuales la fecha sea superior a la definida.

1.5. INSQL [7]

El servidor *IndustrialSQL* (InSQL) se define como una base de datos relacional que adquiere y almacena datos de procesos en tiempo real y provee datos almacenados en archivos históricos.

El servidor InSQL está conformado de 6 subsistemas: configuración, adquisición de datos, almacenamiento de datos, recuperación de datos, eventos y réplicas. Estos subsistemas trabajan en conjunto para definir si los datos adquiridos son almacenados o devueltos en tiempo real. Si son datos almacenados, el servidor permite definir el intervalo de tiempo en el que los datos se guardan en los archivos históricos.

1.5.1. ALMACENAMIENTO DE INFORMACIÓN [7]

Para almacenar la información, en el servidor se configuran tags, que de acuerdo a [6] “son unidades atómicas de almacenamiento, una variable que representa a un único atributo de algún proceso físico o un dispositivo”.

Un tag tiene algunos atributos como el tipo, forma de adquirir los valores y como se almacenan los valores. El atributo tipo puede ser análogo o discreto. El primero hace referencia a una variable que mide una cantidad física continua, por ejemplo la temperatura; y el segundo es discreto cuando la variable solo tiene dos estados 1 y 0.

Los valores de las variables se almacenan en archivos conocidos como históricos, además InSQL crea una base llamada Runtime en un servidor SQL que almacena información de configuración y posee un conjunto de tablas, vistas y procedimientos almacenados. El servidor SQL usa OLE DB (*Object Linking and Embedding for Databases*) para acceder a los datos almacenados en los históricos fuera del servidor.

La base de datos Runtime dispone de dos tipos de tablas para InSQL, la primera es una tabla básica y la segunda es una tabla *wide*, por lo tanto se pueden realizar dos tipos de consulta, una usando una consulta simple cuando se trata de tablas básicas y una consulta con openquery cuando se usan las tablas *wide*.

La sintaxis de la consulta de acceso para OLEDB a una tabla básica se muestra en el Segmento de código 1.4, se puede notar que para una consulta de tipo Select la cláusula where es obligatoria.

```
SELECT select_list

FROM table_source

WHERE search_condition [ GROUP BY group_by_expression ] [
HAVING search_condition ] [ ORDER BY order_expression [ ASC
| DESC ] ]
```

Segmento de código 1.4 Sintaxis de consulta para OLEDB

La consulta de acceso a una tabla *wide* se realiza mediante la función openquery como se puede ver en el Segmento de código 1.5

```
SELECT * FROM OPENQUERY (INSQL, 'SELECT DateTime,

SysTimeSec FROM WideHistory WHERE DateTime >= "2001-09-12
12:59:00"

AND DateTime <= "2001-09-12 13:00:00" ')
```

Segmento de código 1.5 Ejemplo de la función openquery en tabla wide

1.6. CRYSTAL REPORTS [8]

Crystal Reports es la herramienta de elaboración de informes estándar para Visual Studio .NET. Permite crear contenido interactivo con calidad de presentación en la plataforma .NET [3].

Los reportes se diseñan con la herramienta de .NET y se puede configurar el origen de los datos.

1.6.1. CONEXIÓN A DATOS EN CRYSTAL REPORTS [9]

Cuando se usa la aplicación *Crystal Reports*, es necesario escoger el tipo de acceso a los datos; se tiene dos métodos de adquisición de datos: *pull* y *push*

1.6.1.1. MÉTODO *PULL*

El método *pull* se define como la extracción de la información de una fuente de datos conectada para lo cual se usa OLE DB e incluso tiene compatibilidad con una hoja de Excel o una base de datos Access. El desarrollador selecciona la fuente de datos desde el asistente de *Crystal Reports*.

La principal desventaja del uso de este método es que si la fuente de datos se modifica, el diseño del reporte debe ser nuevamente implementado.

1.6.1.2. MÉTODO *PUSH*

El método *push* usa un procedimiento diferente para acceder a los datos, definiendo un *dataset* con el esquema de la tabla de origen de datos que se requiere y usando este *dataset* para definir el reporte. Luego el *dataset* puede ser poblado de datos desde código.

Este modelo requiere más código para ser implementado, pero puede ser simplificado usando un *dataset* tipado para definir un esquema XML que luego puede ser usado para el esquema del origen de datos de reporte mediante una conexión ADO (*ActiveX Data Objects*).

1.6.2. REPORTE

Un reporte es un archivo en el cual se organiza la estructura de la información que se va a presentar al usuario. El diseño del reporte se realiza en una vista de diseño propia de *Crystal Reports* donde se pueden arrastrar y soltar los campos del reporte que se definen utilizando una tabla que se obtiene mediante el establecimiento de una conexión a una base de datos. En la Figura 1.3 se puede observar el diseñador de reportes en Visual Studio y el explorador de campos de *Crystal Report.*

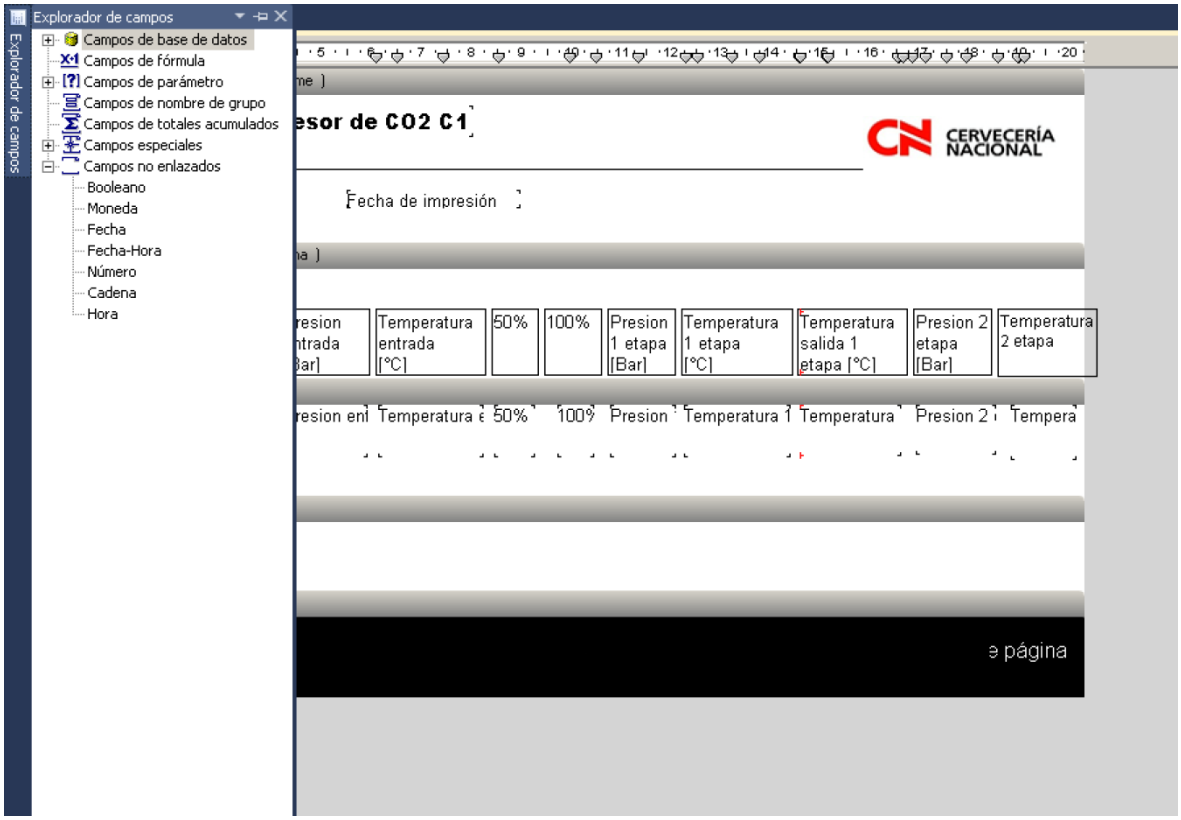

Figura 1.3 Diseñador de reportes y explorador de campos de Crystal Reports

El reporte tiene varios campos que están listados en el explorador de campos entre los que se tiene:

- Los campos de base de datos en los cuales se tiene los campos que corresponden a las columnas de la tabla del origen de datos.
- Los campos de fórmulas permiten operar con los campos de datos para entregar información adicional.
- Los campos de parámetros permiten añadir campos variables que pueden ser modificadas desde código.
- Los campos no enlazados permiten añadir cajas de texto para crear títulos en el reporte.

Mediante un clic derecho en el diseñador se puede añadir imágenes y gráficos.

Este archivo de reporte se presenta en la aplicación mediante un visor de reporte conocido como *CrystalReportViewer*. Si se requiere presentar varios reportes, se utiliza el mismo control y se cambia el reporte de origen.

CAPÍTULO 2

ANÁLISIS DEL ESTADO ACTUAL

En el presente capítulo se describe el estado actual de la empresa enfocándose en los espacios y sistemas relacionados al proyecto, además se describe los procesos que se llevan actualmente para la generación de reportes y gráficas. Se identifican los problemas en los procesos y se propone una solución general para el proyecto.

2.1. INFRAESTRUCTURA

En la empresa Cervecería Nacional, el área de mantenimiento y servicios auxiliares se encarga de la operación, supervisión y control de la maquinaria que se encuentra en la sala de fuerza. En este espacio se ubican los diferentes sistemas de acuerdo a la distribución mostrada en la Figura 2.1, las líneas entrecortadas limitan los espacios que ocupan las máquinas que componen los diferentes sistemas, mientras que las líneas continuas denotan espacios fijos.

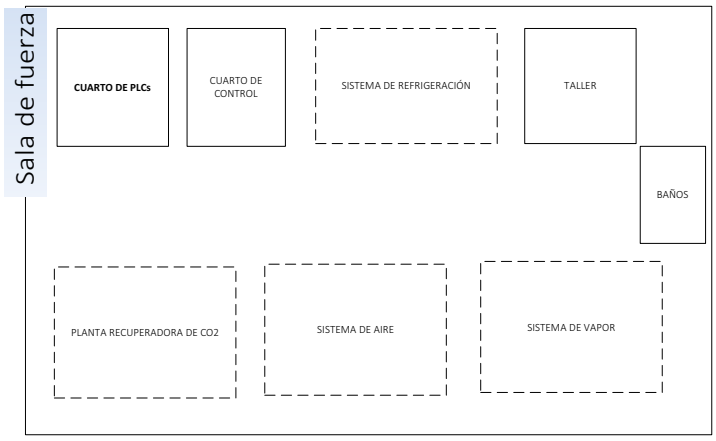

Figura 2.1 Distribución de espacios en la sala de fuerza

Cada sistema tiene un conjunto de sensores que están conectados mediante una red Ethernet a los PLC que se encuentran en el cuarto destinado. En el cuarto de control se tienen computadoras con un sistema SCADA (*Supervisory Control and Data Acquisition*) que permite la visualización de los datos de la maquinaria en tiempo real además de funciones de control, estas computadoras están conectadas a la red a través de un switch. El diagrama completo de red se puede apreciar en la Figura 2.2.

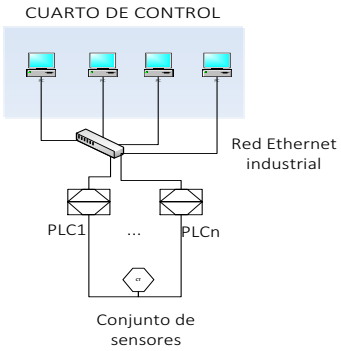

Figura 2.2 Diagrama de red

Las pantallas del sistema SCADA muestran información en tiempo real, en el caso de que se requiera enviar una señal desde control, como la apertura o cierre de una válvula, se lo hace a través del sistema de automatización el cual envía una señal al PLC y se ejecuta la acción en la maquinaria

El sistema de automatización controla todo el conjunto de maquinaria de la sala de fuerza, sin embargo para el propósito de este proyecto solo se toma en cuenta la planta recuperadora de CO2 y el sistema de refrigeración.

La planta recuperadora de CO2 posee un conjunto de maquinaria cuyo propósito es recuperar el CO2 que se obtiene del proceso de fermentación en la producción de la cerveza, para ser utilizado nuevamente a nivel de envase. La Figura 2.3 muestra una visión general del sistema de CO2 en la interfaz HMI[1] del sistema SCADA.

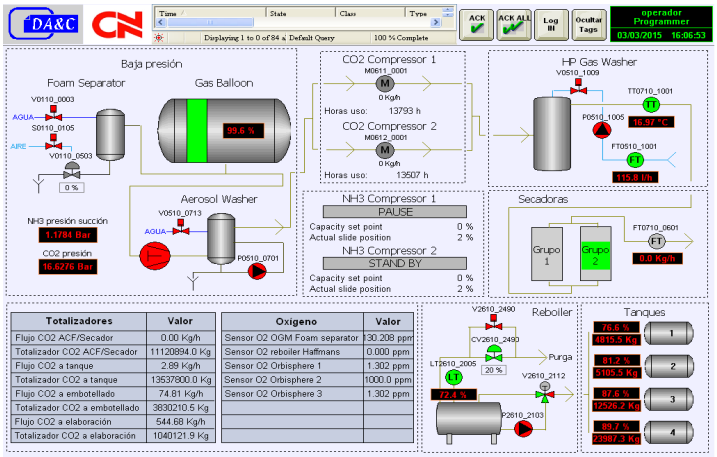

Figura 2.3 Visión general del sistema de CO2 en interfaz HMI

[1] HMI son las siglas de Human-Machine Interface. Un sistema SCADA está compuesto de uno o más HMI.

La planta de CO2 se compone de los siguientes bloques:

- El separador de espuma tiene como propósito separar la espuma de chorro gaseoso de CO2 de la fermentación de cerveza a fin de proteger el sistema de polución.
- El balón gasométrico está diseñado para regular el gas de CO2 mezclado con el aire, libre de espuma y líquidos.
- La lavadora de aerosoles está diseñada para eliminar azúcares contenidos en el gas CO2 por lavado.
- La lavadora de gases está diseñada para eliminar las purezas solubles en agua del gas de CO2.
- El compresor de CO2 se utiliza para comprimir gas de CO2 con una pureza del 90 al 100% en condiciones especificadas.
- El preenfriador está diseñado para enfriar el gas de CO2 húmedo proveniente de los compresores de CO2.
- La unidad de refrigeración NH3 tiene como función el proporcionar suficiente capacidad de condensación para licuar el CO2 gaseoso a través de un condensador de CO2.
- El tanque de almacenamiento de CO2 tiene la función de almacenar CO2 líquido de alta pureza a una presión de 18,5 bares a una temperatura de menos 24 grados centígrados.
- El evaporador de aire de CO2 está diseñado para evaporar CO2 líquido con alta pureza.

En el caso del sistema de refrigeración, este se compone de tres compresores de NH3 cuyo propósito es proveer suficiente refrigeración para la operación de la maquinaria.

2.2. GENERACIÓN DE REPORTES Y GRÁFICAS

El objetivo de tener un reporte es que en caso de presentarse una anomalía los especialistas puedan analizar el conjunto de datos o generar una gráfica para descubrir las posibles causas de las anomalías y presentar soluciones y mejoras.

Actualmente, los operadores generan un reporte que se entrega al finalizar los turnos de 8 horas. El reporte se realiza en una hoja de datos Excel en la cual se

registran manualmente la información de la maquinaria aproximadamente cada hora.

La principal desventaja de esta generación de reportes es que en el caso de una falla en la maquinaria, la prioridad es resolver la falla y la toma de datos es relegada a segundo plano. La falta de datos resulta en reportes incompletos al final de la jornada.

Otra desventaja se presenta debido a que la hoja de Excel utilizada como reporte, por su gran extensión, se vuelve complicada de actualizar, debido a que fácilmente se puede colocar valores en filas o columnas que no correspondan. De la misma manera, la búsqueda y recuperación de valores de una variable específica requiere de la apertura de uno o más archivos dependiendo del intervalo de tiempo que se quiera analizar.

El diagrama de la Figura 2.4 muestra el proceso actual de generación de reportes manuales realizado por los operadores.

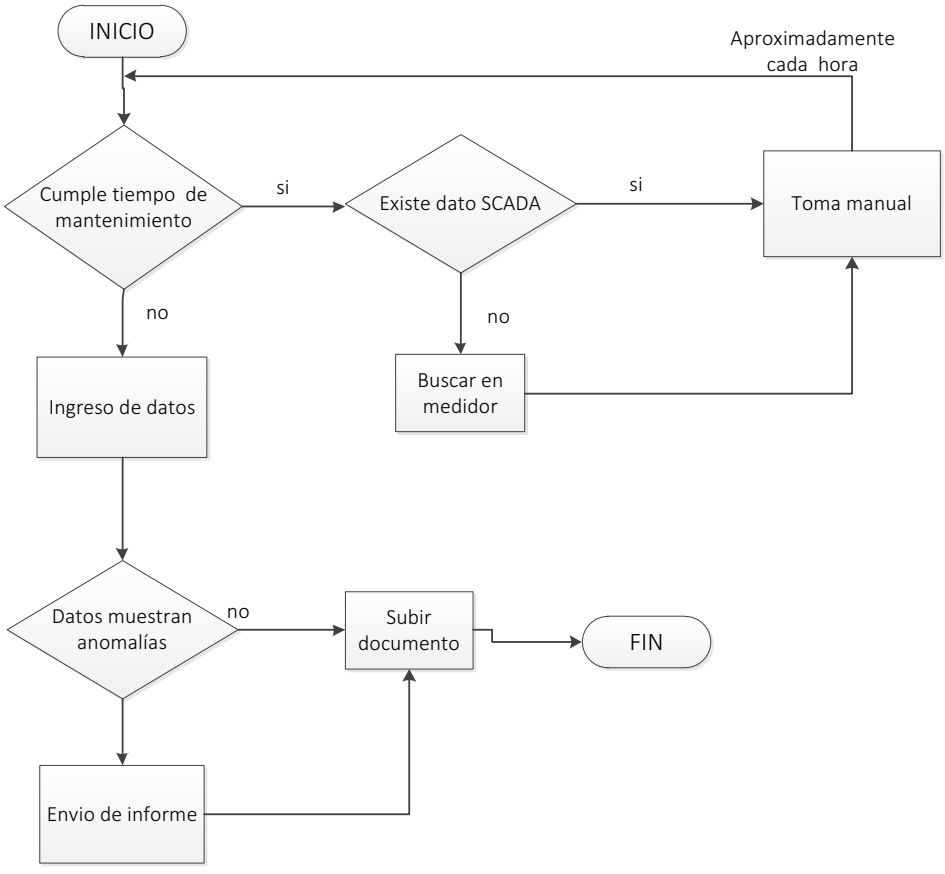

Figura 2.4 Diagrama de generación de un reporte realizado por el operador

Respecto a la generación de gráficas, el sistema SCADA muestra en una de sus pantallas un conjunto de gráficas de ocho variables que están programadas para mostrarse en tiempo real, estas gráficas se conocen como *trends*. La principal desventaja es que no se puede graficar cualquier variable sino solo aquellas ya definidas, por ejemplo la presión de entrada del sistema de CO2. Además, los históricos se borran cada 3 meses por lo cual no se puede hacer un análisis comparativo de meses y años.

La Figura 2.5 muestra el proceso actual de un especialista para recolectar los datos y generar gráficas.

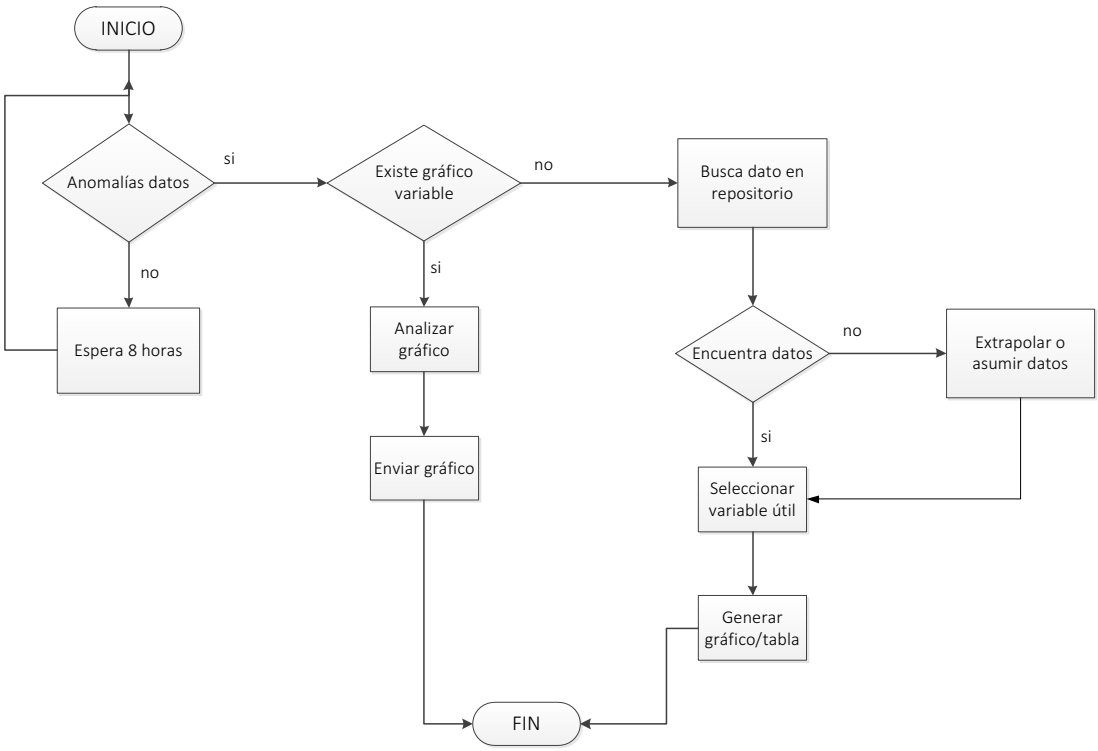

Figura 2.5 Generación de gráficas y recuperación de datos

2.3. PROBLEMAS ENCONTRADOS

Tras realizar un análisis de los procesos actuales de generación de reportes y gráficas se han identificado los siguientes problemas.

- Existen reportes que no poseen datos debido a que las prioridades de reparación y mantenimiento relevan a segundo plano la toma de datos lo que resulta en reportes con espacios sin datos.

- El archivo Excel que sirve como registro no valida el formato de ingreso de los datos, no hace diferencia entre letras; números; números fuera de rango, etc. lo que puede producir valores inservibles.
- Existe pérdida de registros, corrupción de registros y registros incompletos lo que provoca una falta de registros que pudieran ser necesitados.
- Existen registros duplicados, lo que provoca que no se tenga seguridad sobre cuáles son los datos correctos que se requieren.
- Existe posibilidad de manipulación de archivos lo que repercute en la veracidad de los datos.
- No existe almacenamiento automático de los datos que se muestran en el sistema SCADA
- El acceso a un dato en particular requiere la apertura de múltiples archivos que contengan la información.
- Las gráficas de las variables solo pueden retroceder hasta tres meses por lo que no se puede analizar tendencias en un mayor período de tiempo
- No existen gráficas de todas las variables en el sistema SCADA, para generar una gráfica nueva se requiere recopilar datos de varios archivos.
- Los especialistas dependen de la toma de datos de los operadores para poder analizar problemas o generar gráficas.
- La toma de datos manual implica un tiempo de aproximadamente 30 minutos por cada 2 horas que al final del turno representa 120 minutos.

2.4. APLICACIÓN DE SCRUM EN LA DEFINICIÓN DE ROLES Y LISTADO DE REQUERIMIENTOS

Definido el estado actual y la problemática, se define los roles y sus funciones para el desarrollo del proyecto, así como el *product backlog* que entrega como resultado un listado de requerimientos.

2.4.1. DEFINICIÓN DE ROLES

2.4.1.1. SCRUM MASTER

El rol de *SCRUM master* lo llevará a cabo el Ing. César Caiza, Msc. asumiendo la función de guiar en el cumplimiento de las reglas y procesos de la metodología.

La función de realización de reuniones, asociada a este rol, será llevada por el desarrollador quien tiene la oportunidad de fijar las reuniones con el PO con mayor facilidad y por disponibilidad de tiempo.

2.4.1.2. PRODUCT OWNER (PO)

El rol de *product owner* está conformado por el jefe eléctrico y el jefe de mantenimiento y servicios auxiliares los cuales definen los requisitos que conformarán las historias de usuario y formarán parte de las reuniones para revisar los *sprints*.

2.4.1.3. DESARROLLADOR

El rol de *team* se reemplaza por una única desarrolladora del proyecto, que tiene como funciones realizar las tareas definidas para cada *sprint* y trabajar con el *product owner* en la creación de historias y realización de reuniones.

2.4.2. PRODUCT BACKLOG

Durante la primera reunión, el *product owner* y la desarrolladora establecieron los requisitos con los cuales se trabajará durante el desarrollo del proyecto. Utilizando estos requisitos se definieron las historias de usuario presentadas a continuación.

La Tabla 2.1 corresponde a la historia de usuario Administración de usuarios, donde se define que los usuarios deben autenticarse para usar la aplicación y que se deben poder crear, editar y eliminar los usuarios de la aplicación.

Numero: 1	**Usuario:** administrador de la aplicación
Nombre de historia: Administración de usuarios	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Programador responsable: María Eugenia Live	
Descripción: Se necesita autenticar los usuarios que acceden a la aplicación y que se pueda crear, editar y eliminar los usuarios.	
Validación: El sistema debe autentificar a los diferentes tipos de usuarios que van a utilizar la aplicación. El sistema debe permitir crear, editar y eliminar los usuarios.	

Tabla 2.1 Historia de usuario 1

La Tabla 2.2 corresponde a la historia de usuario Tipos de usuario, donde se define que existen dos tipos de usuario para la aplicación, un administrador y un usuario normal.

Numero: 2	**Usuario:** PO
Nombre de historia: Tipos de usuarios	
Prioridad en negocio: Medio	**Riesgo en desarrollo:** Medio
Programador responsable: María Eugenia Live	
Descripción: Se requiere los siguientes usuarios: administrador y usuario normal.	
Validación: El sistema debe soportar dos tipos de usuario: usuario normal y administrador. Los permisos de administrador son sugeridos por el programador.	

Tabla 2.2 Historia de usuario 2

La Tabla 2.3 corresponde a la historia de usuario Almacenamiento de datos de medidores, donde se puntualiza la necesidad de un sistema de almacenamiento de los datos de los medidores, además se establece que las variables almacenadas para cada sistema serán definidas.

Numero: 3	**Usuario:** PO
Nombre de historia: Almacenamiento de datos de medidores	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Programador responsable: María Eugenia Live	
Descripción: Se necesita un sistema de almacenamiento de los datos de los medidores de la maquinaria que conforman los sistemas de CO2 y refrigeración. Las variables para cada sistema, que serán almacenados, se detallan en el ANEXO 1.	
Validación: El sistema requiere una base de datos para almacenar los datos de los medidores. Se define el tiempo de toma de datos cada treinta minutos o una hora dependiendo de las variables.	

Tabla 2.3 Historia de usuario 3

La Tabla 2.4 corresponde a la historia de usuario Acceso a los datos almacenados de medidores, donde se establece que después de implementado el sistema de almacenamiento se requiere recuperar la información previamente almacenada. Sugerido un sistema de base de datos, se requiere implementar un conjunto de métodos accesorios que se ejecuten sobre la base y permitan obtener un conjunto de datos.

Numero: 4	**Usuario:** PO
Nombre de historia: Acceso a los datos almacenados de medidores	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Programador responsable: María Eugenia Live	
Descripción: Se requiere acceder a los datos almacenados en la base de datos.	
Validación: El sistema requiere un conjunto de métodos accesores que permitan recuperar la información de la base de datos.	

Tabla 2.4 Historia de usuario 4

La Tabla 2.5 corresponde a la historia de usuario Diseño de reportes almacenados de medidores en la que se establece que los reportes que entrega la solución deben reemplazar al sistema de reportes actual conformado por una hora de Excel que es llenada por los operadores antes de terminar su turno. Esta hoja de Excel posteriormente se sube a un portal de archivos de la empresa.

Se sugiere separar la hoja de Excel y conformar un conjunto de reportes que conformen la totalidad de los datos de la hoja. Para agrupar la información se sugiere el uso de procedimientos almacenados

Numero: 5	**Usuario:** PO
Nombre de historia: Diseño de reportes	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 4	**Iteración asignada:**
Programador responsable: María Eugenia Live	
Descripción: El sistema debe reemplazar el sistema de archivos actual y agrupar los datos de forma similar a los reportes en Excel.	
Validación: Para agrupar las variables con el formato adecuado se utilizará los procedimientos almacenados.	

Tabla 2.5 Historia de usuario 5

La Tabla 2.6 corresponde a la historia de usuario Reportes combinados y establece que la solución debe permitir la generación de un reporte combinado, que permitirá que el usuario seleccione desde una hasta cuatro variables que luego serán presentadas en un reporte. Se determina que cuatro es un número adecuado para analizar las relaciones entre las variables involucradas en el caso de que se presente una anomalía en la maquinaria o se requiera un análisis del funcionamiento de la misma.

Numero: 6	**Usuario:** PO
Nombre de historia: Reportes combinados	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Bajo
Programador responsable: María Eugenia Live	
Descripción: El sistema debe permitir crear reportes que combinen hasta 4 variables seleccionadas por el usuario.	
Validación: Para agrupar las variables con el formato adecuado se utilizará los procedimientos almacenados.	

Tabla 2.6 Historia de usuario 6

La Tabla 2.7 corresponde a la historia de usuario Formatos de exportación de reporte y establece los formatos en los que se podrán exportar los reportes generados.

Numero: 7	**Usuario:** Usuario de la aplicación
Nombre de historia: Formatos de exportación de reportes	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Bajo
Programador responsable: María Eugenia Live	
Descripción: Se debe permitir exportar los reportes en formato de Excel.	
Validación: La aplicación permitirá exportar los reportes en formatos Excel, se sugiere también PDF.	

Tabla 2.7 Historia de usuario 7

La Tabla 2.8 corresponde a la historia de usuario Generación de gráficas y determina que se deben generar gráficas que combinen hasta cuatro variables, con base a la justificación para los reportes combinados. Se requiere además que se pueda escoger un rango de tiempo y una escala para obtener una gráfica de acuerdo a las necesidades del usuario.

Numero: 8	**Usuario:** Usuario de la aplicación
Nombre de historia: Generación de gráficas	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Programador responsable: María Eugenia Live	
Descripción: Se requiere graficas que se generan en función de 1 hasta 4 variables en un rango de tiempo que se pueda escoger.	
Validación: Con un procedimiento almacenado se puede obtener los datos que crean la gráfica deseada.	

Tabla 2.8 Historia de usuario 8

La Tabla 2.9 corresponde a la historia de usuario Nombre y fecha en reportes y establece que se requiere que el reporte presente el nombre del autor del reporte, es decir el usuario que generó el reporte, además de la fecha de generación.

Numero: 9	**Usuario:** Usuario de la aplicación
Nombre de historia: Nombre y fecha en reportes	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Bajo
Programador responsable: María Eugenia Live	
Descripción: Los reportes deben presentar el nombre del autor del reporte y la fecha de creación del reporte	
Validación: Con el nombre del usuario que autentica el sistema se establece el autor del reporte.	

Tabla 2.9 Historia de usuario 9

La Tabla 2.10 corresponde a la historia de usuario Tipo de aplicación y determina que la aplicación debe ser de tipo web para que pueda ser accedida desde los navegadores de la intranet.

Numero: 10	**Usuario:** PO
Nombre de historia: Tipo de aplicación	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Bajo
Programador responsable: María Eugenia Live	
Descripción: La aplicación debe ser de tipo web para que pueda ser accedida desde los exploradores web en las computadoras de la intranet.	
Validación: Se desarrolla una aplicación web.	

Tabla 2.10 Historia de usuario 10

La Tabla 2.11 corresponde a la historia de usuario Tipo de base de datos y establece el uso de la base de datos industrial InSQL

Numero: 11	**Usuario:** PO
Nombre de historia: Tipo de base de datos	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Bajo
Programador responsable: María Eugenia Live	
Descripción: La aplicación debe utilizar la base de datos INSQL.	
Validación: La instalación de InSQL requiere el sistema operativo Windows Server 2003 y SQL Server para el acceso a la base	

Tabla 2.11 Historia de usuario 11

La Tabla 2.12 corresponde a la historia de usuario Administración de sistemas y establece que la aplicación debe permitir crear, modificar o eliminar un sistema.

Numero: 12	**Usuario:** Administrador de la aplicación
Nombre de historia: Administración de sistemas	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Bajo
Programador responsable: María Eugenia Live	
Descripción: La aplicación debe permitir crear, modificar o eliminar un sistema.	
Validación: El crear nuevos sistemas debe permitir que la aplicación sea escalable en función de reportes combinados y gráficas.	

Tabla 2.12 Historia de usuario 12

La Tabla 2.13 corresponde a la historia de usuario Administración de sistemas y establece que la aplicación debe permitir administrar los permisos para los diferentes perfiles de usuario

Numero: 13	**Usuario:** Administrador de la aplicación
Nombre de historia: Administración de perfiles y permisos	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Bajo
Programador responsable: María Eugenia Live	
Descripción: La aplicación debe administrar los permisos para cada perfil de usuario	
Validación: La aplicación debe permitir asignar o eliminar permisos a los usuarios.	

Tabla 2.13 Historia de usuario 13

2.4.3. REQUISITOS FUNCIONALES

Un requisito funcional indica que es lo que debe hacer la aplicación, es decir determina las diversas funcionalidades de la aplicación.

La Tabla 2.14 presenta la lista de los requisitos funcionales de acuerdo a las historias de usuario, la tabla asigna a cada requisito un código de referencia para que cuando se requiera mencionar uno de los requisitos durante el diseño e implementación de la solución se haga solamente referencia a este código en lugar de escribir nuevamente el requisito.

Cod. HU	Cod. Req.	Descripción	Prioridad (1-5)
1	RF1	El sistema debe autenticar los usuarios que deseen acceder a la aplicación	2
2	RF2	El sistema debe soportar dos tipos de usuario: administrador y usuario normal	2
3	RF3	El sistema requiere una base de datos para almacenar los datos de los medidores. Se define el tiempo de toma de datos cada treinta minutos o una hora dependiendo de las variables.	5
3	RF4	El sistema requiere acceder a la información de la base de datos	4
3	RF5	El sistema requiere un conjunto de consultas, tablas y procedimientos almacenados que permitan acceder de manera estructurada a la aplicación	4
5	RF6	El sistema debe reemplazar el sistema de archivos actual	4
5	RF7	El sistema debe desplegar reportes dada una fecha y hora inicial y final	3
6	RF8	El sistema debe generar reportes de una hasta cuatro variables combinadas.	4
5	RF9	El sistema debe generar reportes con una estructura similar a los reportes Excel actuales	4
7	RF10	El sistema debe permitir exportar los reportes en formatos de Excel y PDF	3
8	RF11	El sistema debe generar gráficas que muestren tendencias con manejo de escalas	4
8	RF12	El sistema debe permitir gráficas con intervalos de horas, días, meses.	3
8	RF13	El sistema debe generar gráficas combinadas de 1 hasta 4 variables.	3
9	RF14	El sistema debe permitir que se muestre el nombre del autor y la fecha de creación del reporte.	1
3	RF15	El sistema debe evitar corrupción de información	5
3	RF16	El motor de base de datos debe almacenar las variables cada media hora o una hora.	5
13	RF17	El sistema debe administrar permisos para cada perfil de usuario	3

Tabla 2.14 Tabla de requisitos funcionales

2.4.4. REQUISITOS NO FUNCIONALES

La Tabla 2.15 presenta la lista de los requisitos no funcionales, la prioridad se establece en 5 debido a que se debe comenzar el desarrollo del proyecto con base a estos.

Cod. Req.	Descripción	Prioridad (1-5)
RNF1	El sistema debe ser web para ser accedida desde la intranet	5
RNF2	El sistema debe usar INSQL como motor de base de datos por licencias adquiridas	5
RNF3	El sistema debe usar WINDOWS Server 2003 como sistema operativo por compatibilidad con base de datos	5
RNF4	El sistema debe usar SQL Server 2000 por compatibilidad con base de datos	5

Tabla 2.15 Tabla de requisitos no funcionales

2.4.5. DISCRIMINACIÓN DE VARIABLES

Durante la segunda reunión con el PO se discrimina las variables que se almacenarán en la base de datos y que servirán para construir los reportes. En la Tabla 2.16 se puede ver el conjunto de variables del bloque de baja presión del sistema de CO2, con sus unidades y una breve descripción.

Se realiza una discriminación de variables debido a que el sistema de CO2 está compuesto por un extenso conjunto de maquinaria entrega una gran cantidad de datos al monitorear los sistemas y no todas las variable son útiles para realizar los estudios que se requieren para realizar tareas de mantenimiento predictivo. Además se establece que almacenar todas las variables asociadas al sistema de CO2 ocuparía, eventualmente, mucho espacio de almacenamiento en el servidor, de manera que se debería establecer un plan de migración hacia un servidor de mayor capacidad.

En el Anexo 1 se encuentran todas las tablas con las variables de cada bloque del sistema de CO2.

Para el sistema de refrigeración se ocupan todas las variables que entregan los medidores

BAJA PRESIÓN					
Sensor	**Nombre de la variable (TAG)**	**Tipo de variable**	**Unidades**	**Intervalo de tiempo [min]**	**Descripción**
XE	XE0110_0003	Evento		30	Apertura y cierre de válvulas
XE	XE0110_0103	Evento			Apertura y cierre de válvulas
QT	QT0110_0501	Análogo	Ppm		Medida de calidad de CO2
PT	PT0110_0502	Análogo	mB		Medida de presión
LT	LT0410_0001	Análogo	%		Porcentaje de llenado de globo
M	M0410_0501	Análogo	H		Contador número de horas trabajadas
M	M0510_0704	Análogo	H		Contador número de horas trabajadas
MB	P0510_0701	Análogo	H		Contador número de horas trabajadas

Tabla 2.16 Ejemplo de variables que componen el bloque de baja presión.

2.5. ARQUITECTURA PROPUESTA PARA LA SOLUCION

Con base a los requerimientos definidos en la primera reunión, se propone una solución que recuperará los datos de los históricos utilizando la base SQL como intermediario para desplegarlos en una aplicación web. La arquitectura propuesta se muestra en la Figura 2.6.

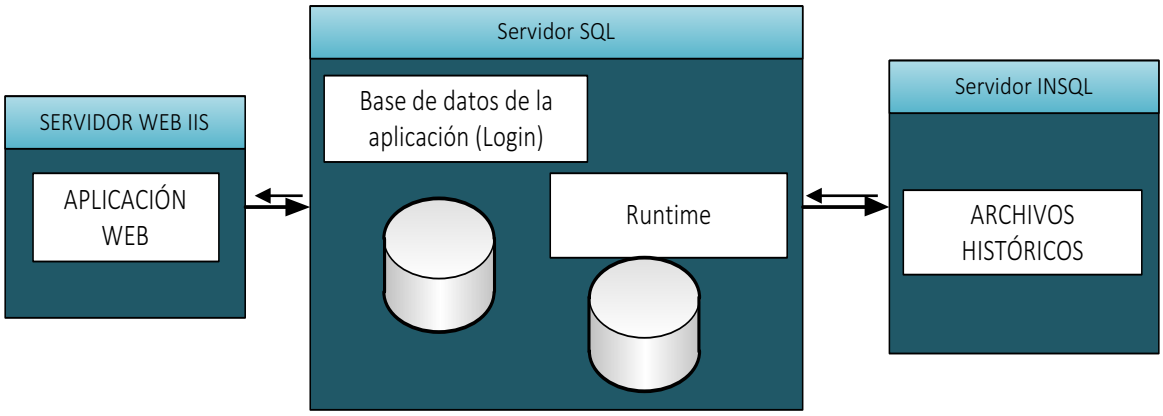

Figura 2.6 Arquitectura propuesta de la solución

La máquina que funciona como servidor almacena tres instancias principales:

- Una instancia de servidor SQL.
- Una instancia de servidor INSQL.
- Una instancia de servidor web IIS.

La máquina tendrá instalado el sistema operativo Windows Server 2003 y un servidor SQL 2000 por compatibilidad con el servidor INSQL de acuerdo a los requerimientos RNF3 y RNF4.

En el servidor SQL se genera una base de datos al momento de la instalación de INSQL. Dicha base utiliza un conjunto de tablas y procedimientos almacenados para acceder a la información de los bloques históricos[2]. Adicionalmente, el servidor SQL almacenará la base de datos que deberá manejar información de los usuarios, sistemas y permisos de usuario.

El servidor INSQL recogerá los datos de los PLC de la planta cada media hora o una hora y los almacenará en los bloques históricos de cumpliendo con el RF3.

El servidor web alojará la aplicación web en cumplimiento con el requisito RNF1.

2.5.1. APLICACIÓN WEB

La Figura 2.7 muestra los cinco módulos que conformarán la aplicación.

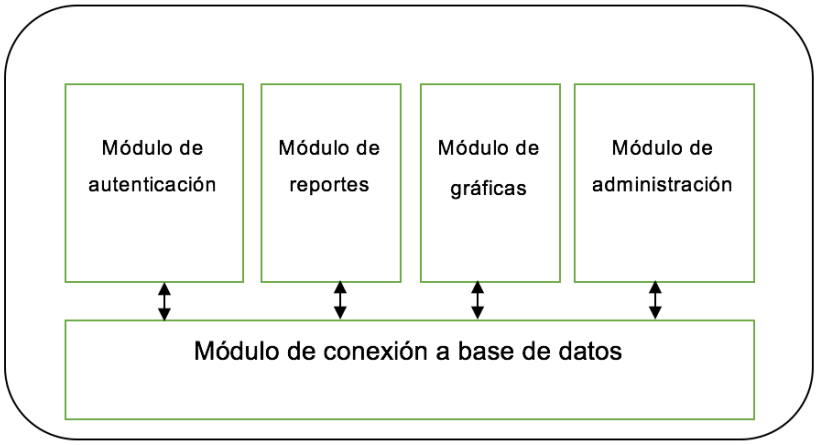

Figura 2.7 Módulos de la aplicación

[2] Los bloques históricos son archivos propios de INSQL que almacenan la información recibida de los PLC.

- El módulo de autenticación permitirá validar las credenciales de los usuarios que desean acceder al sistema de acuerdo al requerimiento RF1.
- El módulo de reportes desplegará reportes de formato fijo y reportes combinados de acuerdo a los requerimientos RF9 y RF8.
- El módulo de generación de gráficas permitirá el despliegue de gráficas de acuerdo al requerimiento RF13.
- El módulo de administración permitirá al usuario con rol de administrador realizará acciones CRUD (create, read, update, delete) para usuarios y sistemas. Este requerimiento no está especificado sin embargo es parte del desarrollo de una aplicación que pueda crecer en funcionalidad.
- El módulo de conexión a la base de datos estará compuesto de las clases que permitan el establecimiento de conexión y acceso a las bases de datos y la recuperación de los datos que se utilizarán dependiendo de sus requerimientos.

2.6. SPRINT BACKLOG

En la Tabla 2.17 se detalla la organización de los sprints los cuales tienen una duración de dos semanas.

REQUISITO	TAREA	SPRINT	TOTAL DE HORAS ESTIMADAS
Instalación de software	Instalación Windows Server 2003	1	10
	Instalación de SQL Server 200		
	Instalación de InSQL		
Acceso a base de datos	Diseño e implementación de la base Login	2	30
	Diseño e implementación de procedimientos almacenados en la base Runtime		
	Pruebas a procedimientos almacenados		
Autenticación de usuarios	Diseño de módulo de autenticación	3	15
	Implementación de módulo de autenticación		
	Pruebas en módulo de autenticación		

REQUISITO	TAREA	SPRINT	TOTAL DE HORAS ESTIMADAS
Administración de usuarios	Diseño de clases de administración de usuarios	3	15
	Implementación de clases de administración de usuarios		
	Pruebas a clases de administración de usuarios		
Administración de sistemas	Diseño de clases de administración de sistemas	4	15
	Implementación de clases de administración de sistemas		
	Pruebas a clases de administración de sistemas		
Administración de permisos	Diseño de las clases de administración de permisos	4	15
	Implementación de clases de administración de permisos		
	Pruebas a clases de administración de permisos		
Módulo de reportes	Diseñar las clases de módulos de reportes	5	30
	Implementación de módulo de reportes		
	Pruebas en módulo de reportes		
Módulo de gráficas	Diseñar las clases de módulo de gráficas	6	20
	Implementación de módulo de gráficas		
	Pruebas en módulo de gráficas		

Tabla 2.17 Tabla de planificación de iteración

CAPÍTULO III

DISEÑO E IMPLEMENTACIÓN

El diseño de la aplicación se realiza a través de diagramas UML de clases y procesos para representar la lógica del sistema y las relaciones entre los distintos componentes de la aplicación.

3.1. DIAGRAMA DE CLASES

En la figura 2.8 se puede apreciar el diagrama de clases donde se muestran los atributos, métodos y relaciones de las clases para la aplicación.

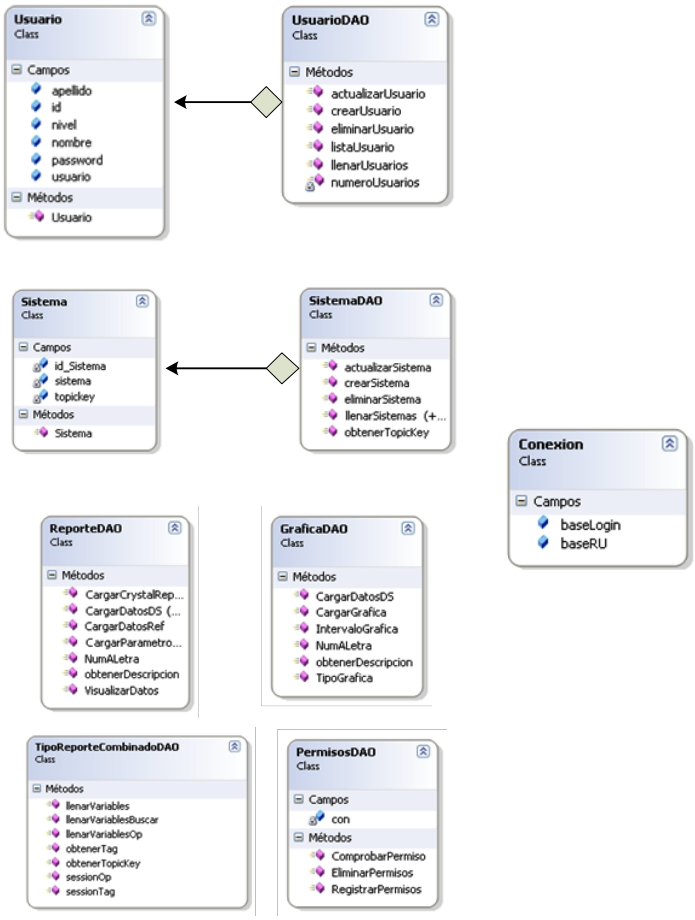

Figura 2.8 Diagrama de clases

La clase usuario se utilizará para registrar los usuarios que utilizarán la aplicación. Los atributos definidos son: nombre, apellido, usuario, contraseña y perfil. Estos datos se utilizan para registrar y autenticar los usuarios de la aplicación.

La clase Usuario tiene una relación de agregación a la clase UsuarioDAO, la cual utiliza la clase de conexión para acceder a la base de datos. La clase UsuarioDAO contiene los métodos CRUD que dan la funcionalidad de administrar los usuarios que pueden acceder a la aplicación.

La clase Sistema se utilizará para registrar nuevos sistemas en la aplicación. Los atributos del objeto sistema son: TopicKey y nombre. Esta clase tiene una relación de agregación a la clase SistemaDAO la cual tiene una funcionalidad similar a la clase UsuarioDAO.

La clase Conexión define las cadenas que contienen la información para establecer y mantener la conexión a las bases de datos Runtime y Login

La clase ReporteDAO permite realizar las consultas a la base de datos Runtime y retornar el conjunto de datos. Con los parámetros seleccionados por el usuario se devuelve un dataset que será desplegado como una tabla o un reporte dependiendo de la selección del usuario.

La clase TipoReporteCombinadoDAO tiene una funcionalidad similar a la clase ReporteDAO, devolviendo el conjunto de datos para los reportes combinados.

La clase GraficaDAO retorna el conjunto de datos que serán graficados y presentados al usuario.

La clase PermisosDAO permite registrar y eliminar los permisos de acceso para los diferentes tipos de usuario.

3.2. INSQL Y ALMACENAMIENTO DE DATOS

InSQL almacena datos en dos lugares. El primero es una base de datos nombrada Runtime que guarda información de configuración y se administra a través de SQL. El segundo es un conjunto de archivos conocidos como bloques históricos que almacenan los valores de las variables. El acceso a los datos de los bloques históricos se hace mediante consultas a las tablas que están definidas en la base `Runtime`.

Las consultas se realizan en la tabla `WideHistory` (Runtime.dbo.WideHistory), mostrada en la Figura 2.9 a) y b), la cual presenta los datos de una o más

variables en un mismo rango de tiempo. En la consulta que se realiza se debe especificar un intervalo de tiempo y las variables. Es importante resaltar que a la tabla solo se puede acceder utilizando la función openquery.

Datetime	Tag 1	Tag 2	….	Tag n

a) Esquema

Datetime	XE0110_0003	XE0110_0004	M0410_0501
2016-05-24 00:00:00	0	1	1500

b) Ejemplo de valores

Figura 2.9 Tabla WideHistory

Los valores de Tag1 corresponden a la variable que se desea consultar, estos nombres en general son una combinación de letras y números que no dan una idea de a que variable corresponden por lo que se utiliza la tabla Tag (Runtime.dbo.Tag), mostrada en la Figura 2.10, para presentar al usuario un identificador de variable que pueda comprender.

De la tabla Tag se utilizan tres columnas: la columna `TagName`, que funciona como clave primaria para la tabla, corresponde al nombre de la variable; la columna `TopicKey` es un número único que identifica a un tópico[3] y se usa para asociar la variable a un sistema; la columna `Description` corresponde a un campo editable el cual permitirá identificar a las variables de una forma comprensible para el usuario.

Tagname	TopicKey	Description	…
XE0110_0003	133	Presencia de espuma entrada	

Figura 2.10 Tabla Tag

La interacción de estas tablas es constante, pues al usuario se desplegará la lista de la descripción variables que pertenecen a un tópico haciendo una consulta a la tabla Tag. Tras la selección de las variables se hace una consulta en reversa para

[3] Un tópico en el ámbito de InSQL corresponde a un sistema, es decir el conjunto de maquinaria que cumple una función.

obtener el TagName para poder armar las consultas. Por ejemplo, tomando en cuenta la Figura 2.9 (b) como el formato inicial, el conjunto de datos devuelto al usuario tendría un formato como se muestra en la Figura 2.11

Datetime	Presencia de espuma en entrada	Presencia de espuma en la salida	Número de horas motor booster
2016-05-24 00:00:00	0	1	1500

Figura 2. 11 Ejemplo de tabla presentada al usuario

3.3. SQL

Además de la base de datos Runtime, la instancia de SQL contiene la base llamada Login que almacena la información que se requieren para el funcionamiento de la aplicación. Las tablas definidas en la base Login, sus relaciones, y la interacción con la tabla de la base Runtime se pueden apreciar en la Figura 2.12

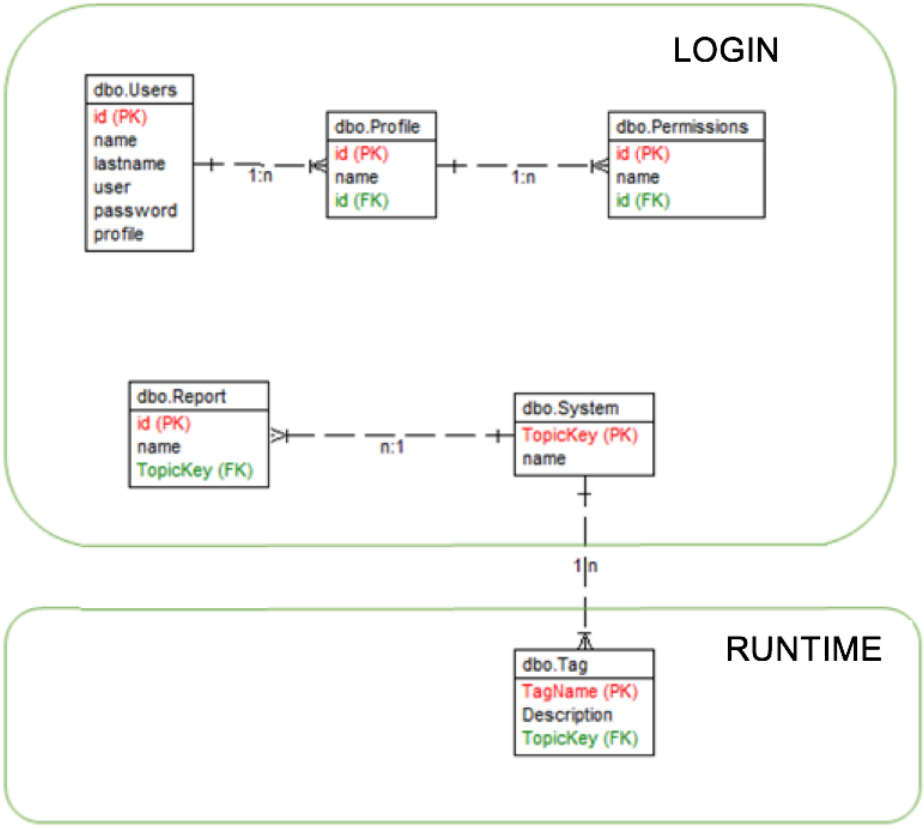

Figura 2. 12 Diagrama de base de datos relacional

La tabla dbo.Users permitirá manejar la información de los usuarios, contiene los campos de identificador, nombre, apellido, usuario, contraseña y perfil. Esta tabla se basa en los campos de clase Usuario.

La tabla dbo.System permitirá manejar la información de los sistemas y corresponde a los campos de la clase Sistema. El campo TopicKey es un campo propio que asigna el servidor InSQL cuando se añade un nuevo sistema con su grupo de variables. Este campo es importante porque permite que al cargar nuevas variables al servidor InSQL, puedan ser visibles automáticamente en la aplicación al añadir el sistema con su respectivo TopicKey.

La tabla dbo.Tag es propia del InSQL, tiene una relación de uno a muchos con la tabla dbo.System asociadas por el campo TopicKey

La tabla dbo.Report permitirá manejar la información de los títulos de reportes que están asignados a cada sistema. La relación entre la tabla dbo.System y dbo.Report es de uno a muchos.

Las tablas de perfiles y permisos contienen la información correspondiente a los perfiles de usuario y los permisos asociados a cada uno. La relación entre perfiles y permisos es de uno a muchos y un usuario puede tener uno o más perfiles.

3.3.1. DIAGRAMAS DE PROCESOS

Los actores involucrados en la solución son: el usuario de la aplicación, que interactúa con la aplicación enviando credenciales de autenticación y parámetros para la presentación de reportes y gráficas. Para el actor usuario se establecen dos roles, el administrador que tiene permisos para ver, crear, eliminar y editar los usuarios y sistemas, y un usuario normal cuyos permisos solamente permiten ver reportes y gráficas. El segundo actor es la base de datos Login y Runtime que ejecutan las consultas y procedimientos almacenados y retornan el resultado de la consulta hacia la aplicación.

Definidos los actores, se presenta los diagramas de proceso para identificar y relacionar los procesos realizados por los actores y la aplicación.

En la Figura 2.13 presenta el diagrama de procesos para la función de autenticación.

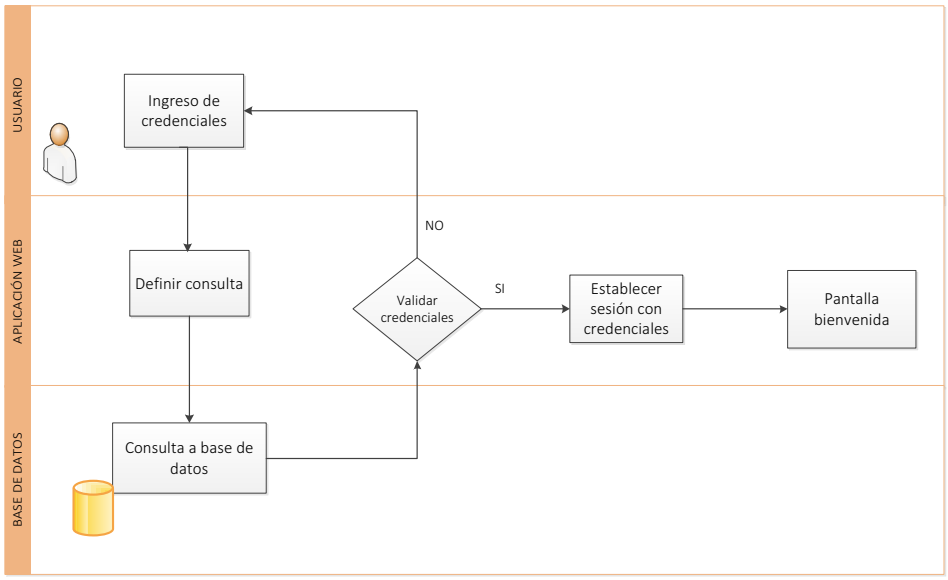

Figura 2.13 Diagrama de procesos para autenticación

La función de autenticación se encargará de validar las credenciales, nombre de usuario y contraseña, mediante una consulta a la base de datos. La consulta retornará el nombre del usuario que se usará para establecer una sesión en caso de que la validación sea exitosa, tras el establecimiento de sesión se presentará la pantalla de bienvenida en la cual se puede acceder a los módulos de generación de reportes o gráficas.

La Figura 2.14 muestra el diagrama de procesos para la visualización y generación de reportes.

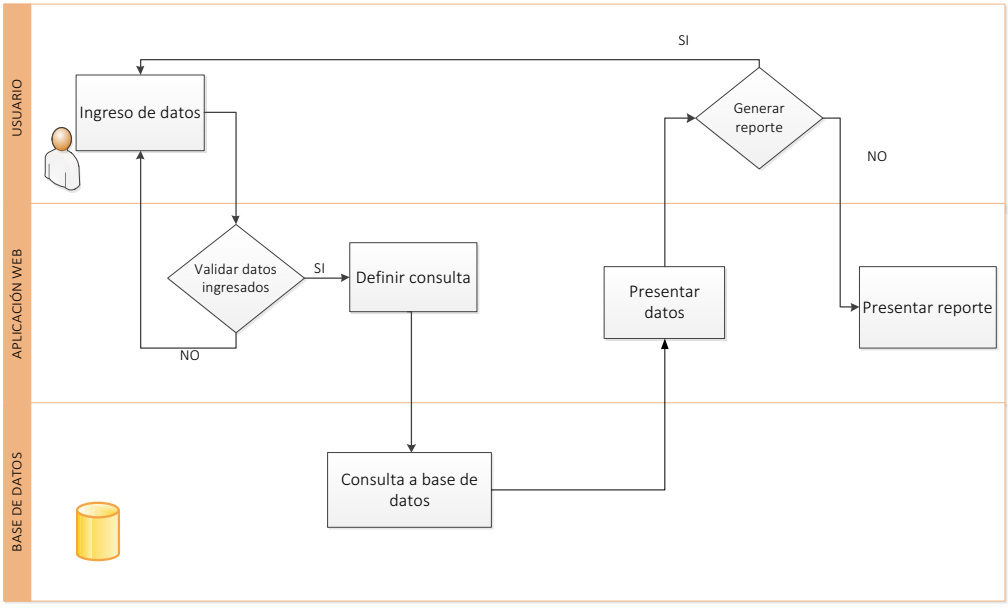

Figura 2.14 Diagrama de procesos para generación de reportes

La visualización y generación de reportes tiene el ingreso de los datos por parte del usuario los cuales son validados. Con el proceso exitoso de validación se usará los datos para definir una consulta que al ser ejecutada en la base de datos devolverá el conjunto de datos que serán presentados al usuario el cual puede escoger presentar los datos en forma de reporte o realizar una nueva consulta.

Los reportes serán de dos tipos, unos reportes que tienen estructura fija definida por los reportes actuales y unos reportes en los cuales se pueden combinar hasta cuatro variables.

La Figura 2.15 muestra el diagrama de procesos para el módulo de generación de gráficas.

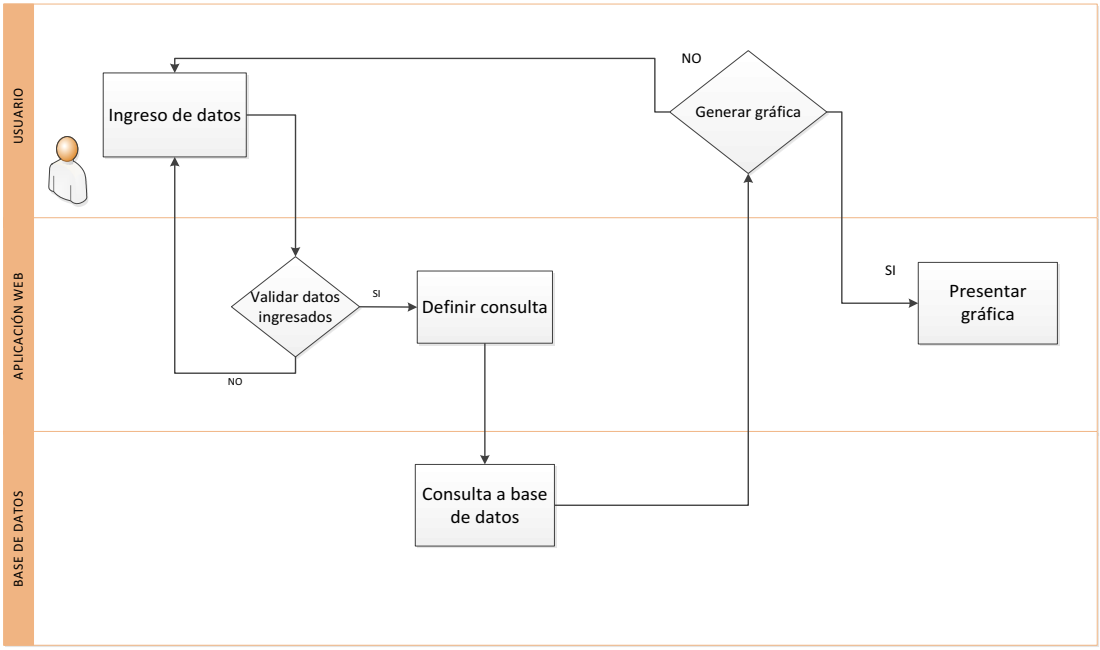

Figura 2.15 Diagrama de procesos para generación de gráficas.

Las gráficas utilizarán la misma consulta de los reportes combinados, de forma que se pueden tener gráficas de hasta cuatro variables en un mismo intervalo de tiempo.

3.3.2. DIAGRAMAS DE SECUENCIA

La Figura 2.16 presenta el diagrama de secuencia para los módulos de autenticación, generación de reportes y generación de gráficas mostrando como se realiza el intercambio de información cuando el usuario realiza las peticiones en los módulos establecidos.

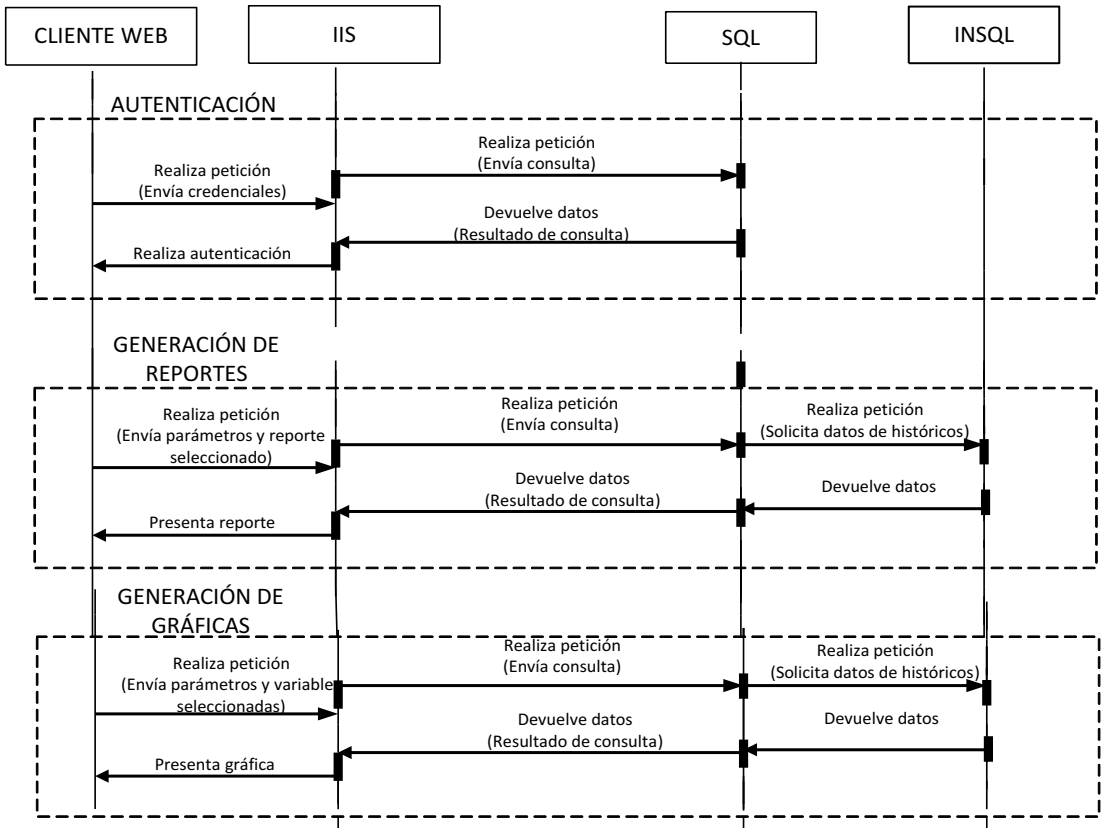

Figura 2.16 Diagramas de secuencia

3.3.3. INTERFACES DE USUARIO

En base a los módulos planteados se propone las siguientes vistas para la aplicación que ofrecen un esquema de cómo se conformará la interfaz de usuario

En la Figura 2.17 se tiene la vista de autenticación, que presentará a los usuarios un conjunto de controles para ingresar las credenciales de usuario y enviar la información.

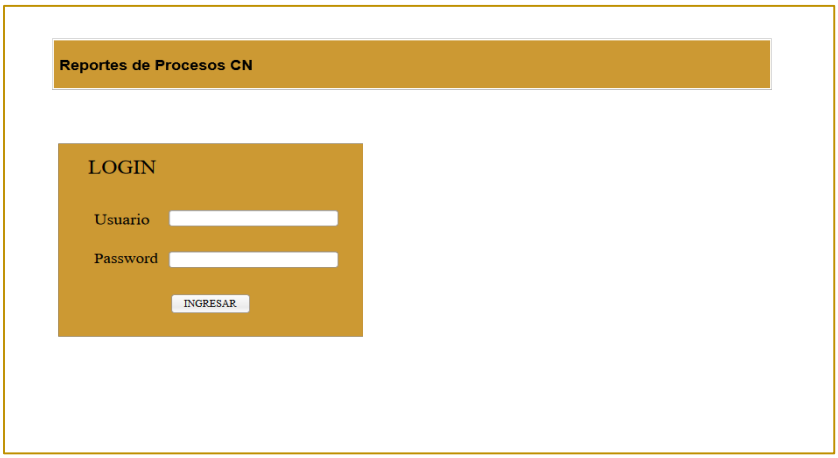

Figura 2.17 Vista de autenticación

La Figura 2.18 corresponde a la vista de bienvenida y presenta a los usuarios un menú que les permita desplazarse por los diferentes módulos de la aplicación. Además presenta imágenes de la planta y una frase motivacional.

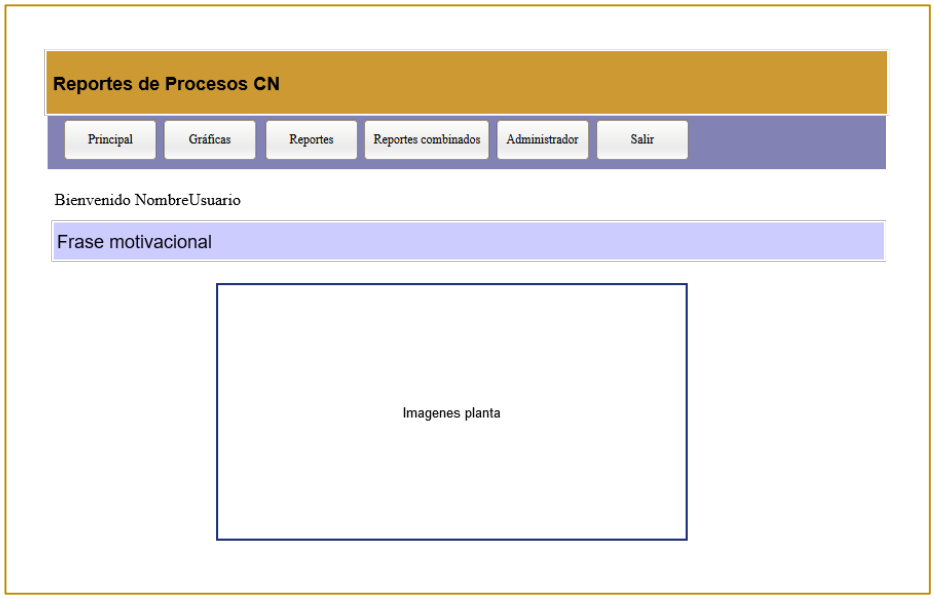

Figura 2.18 Vista de bienvenida

La Figura 2.19 corresponde a la vista de generación de reportes y muestra a los usuarios un conjunto de controles para ingresar los parámetros de fecha y hora inicial y final; seleccionar el sistema y el reporte correspondiente y enviar la información al servidor que se presentará como reporte en el control inferior.

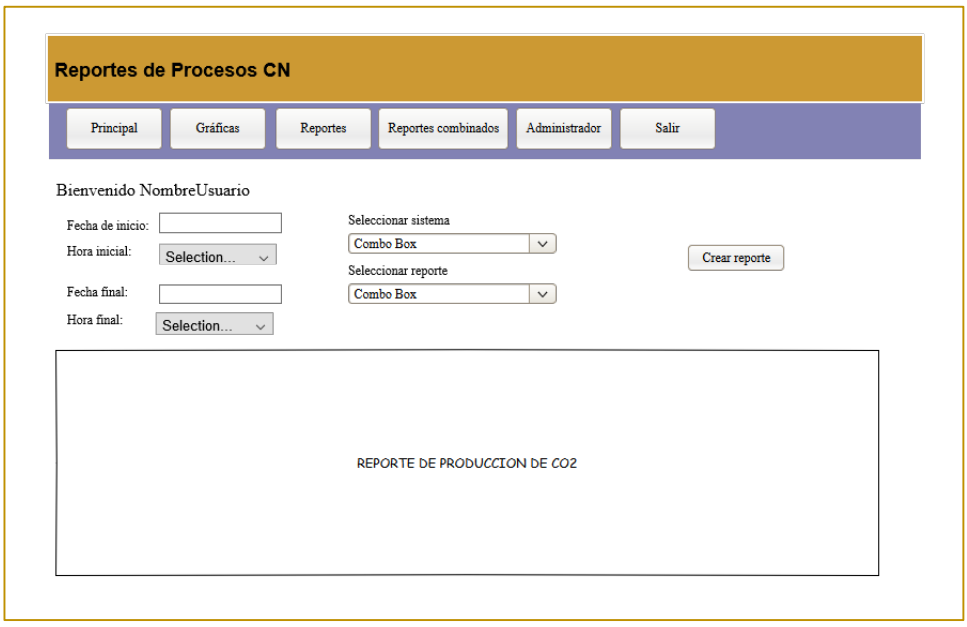

Figura 2.19 Vista de generación de reportes

La Figura 2.20 corresponde a la vista de generación de reportes combinados y presenta al usuario un conjunto de controles para ingresar información de manera similar a un reporte normal.

Figura 2. 20 Vista de generación de reportes combinados

La Figura 2.21 corresponde a la vista de generación de gráficas, nuevamente se muestra al usuario el conjunto de controles para el ingreso de información de manera similar a los reportes, sin embargo se despliega una gráfica en lugar de un reporte.

Figura 2.21 Vista de generación de gráficas

La Figura 2.22 corresponde a la vista del módulo de administración y presenta a los usuarios con permisos de administrador un conjunto de controles que permitirán enviar al servidor la información para crear, eliminar y modificar los usuarios y sistemas.

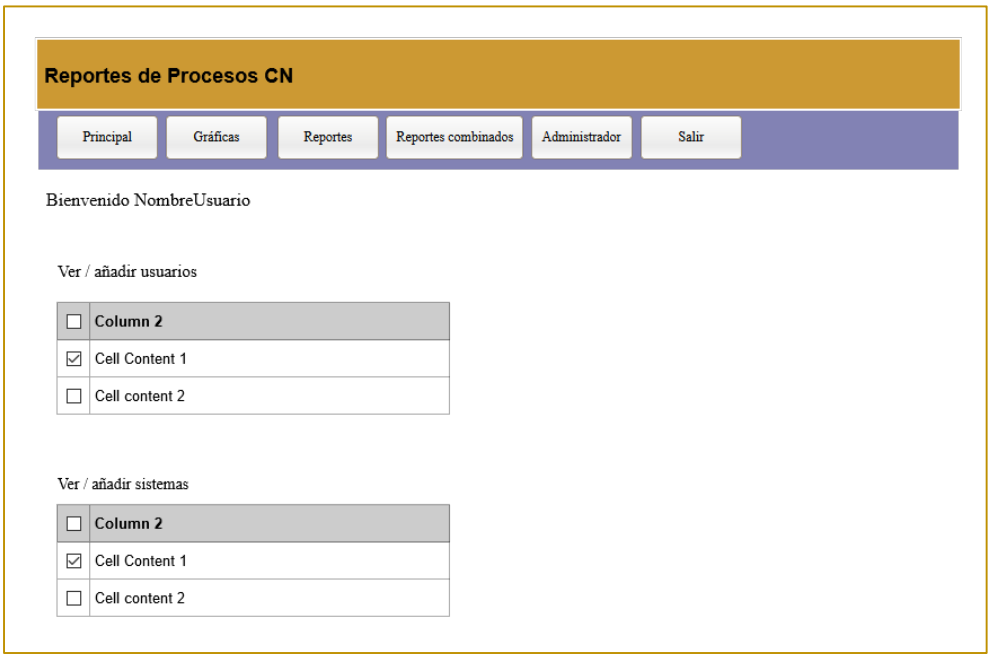

Figura 2.22 Vista de módulo de administración

3.4. IMPLEMENTACION

La implementación de la solución requiere en principio de la elección de las herramientas para cumplir con los requisitos por lo que se tiene un conjunto de cuadros comparativos. Se mostrará el esquema del procedimiento almacenado utilizado para recuperar la información y describirá las clases implementadas mencionando la funcionalidad de los métodos definidos en cada clase.

Se mencionará además el uso de las variables de sesión en la aplicación y se mostrará aspectos de la implementación de las interfaces de usuario, reportes, controles *ajax* y presentación de gráficas.

3.4.1. COMPARACIÓN DE LENGUAJES DE PROGRAMACIÓN

De acuerdo al requerimiento no funcional RNF1, se debe desarrollar una aplicación web. Se debe tomar en cuenta los RF3, RF4, RF5 y RF11, con los cuales se ha desarrollado la Tabla 3.1

	PHP	C# con ASP.NET	JAVA
Compatibilidad con IIS (Windows Server 2003)	×	✓	×
Compatibilidad con SQL	✓	✓	✓
Compatibilidad con desarrollo de aplicación web	✓	✓	✓
Presentación de reportes	✓	✓	✓
Generación de reportes	×	✓	×
Generación de gráficas	✓	✓	✓

Tabla 3.1 Tabla de comparación de lenguajes de programación

Cabe recalcar que las aplicaciones desarrolladas tanto en PHP como Java eventualmente pueden llegar a ser implementadas en el servidor web IIS, sin embargo la compatibilidad no es directa sino mediante terceras aplicaciones [10], [11].

De igual manera, para el caso de la generación de reportes, ASP.NET cuenta con una herramienta propia para generación de reportes, PHP y Java también permiten crear reportes mediante librerías externas.

Mediante la comparación se decide elegir como lenguaje de programación c# por su completa compatibilidad con los requisitos y además por tener conocimientos previos adquiridos en la carrera.

3.4.2. COMPARACION DE HERRAMIENTAS DE GENERACIÓN DE REPORTES

Los reportes son de gran prioridad en la aplicación y permiten cumplir el RNF2 y el RF11, además se añade la consideración de licencia y el trabajo con gran cantidad de datos para que soporte la carga de las consultas. La Tabla 3.2

recolecta los parámetros a ser considerados para escoger la herramienta de reportes.

	Report Viewer	CrystalReport	SQL Report
Compatibilidad con SQL 2000	✓	✓	×
Exportación en formatos PDF y Excel	✓	✓	✓
Licencia	×	✓	✓
Trabajo con gran cantidad de datos	×	✓	✓

Tabla 3.2 Tabla de comparación de herramientas de reporte

La herramienta de Crystal Report cumple con los requisitos completos, además tiene una librería compatible con ASP.NET y c#. También se puede mencionar que hay mucha información y tutoriales que explican la instalación y funcionamiento de la herramienta.

3.4.3. COMPARACION DE METODOS DE CONSULTA A LA BASE DE DATOS

Es necesario definir el método de consulta a la BDD debido a que este nos permitirá acceder a los datos almacenados. Hay que tener en cuenta que para poder conectarse y realizar las consultas, el método debe tener soporte para el uso de la función openquery.

Existen varios métodos que permiten conseguir esta funcionalidad, estos son: consultas, funciones y procedimientos almacenados. Se ha escogido el método más adecuado considerando los siguientes criterios:

- Devolver en forma de tabla el conjunto de resultados para luego usar la tabla como origen de datos en el reporte
- Aceptar parámetros de entrada para poder establecer las fechas de inicio y fin que son variables.
- Que tenga un alto rendimiento por la gran cantidad de datos que se deben procesar y retornar.

Las consideraciones se presentan en la Tabla 3.3.

	Consultas	**Procedimientos almacenados**	**Funciones**
Devolver en forma de tabla el conjunto de resultados	✓	✓	×
Aceptar parámetros de entrada	×	✓	✓
Uso de openquery	✓	✓	✓
Alto rendimiento	×	✓	✓

Tabla 3.3 Tabla de comparación de métodos de consulta

En conclusión la herramienta para generar consultas será un procedimiento almacenado, el cual cumple con todos los criterios establecidos para el proyecto.

3.4.4. ESQUEMA DEL PROCEDIMIENTO ALMACENADO

Los procedimientos almacenados se escriben en la base de datos `Runtime` en donde mediante esta base de datos intermedia, se puede obtener la información de los datos que se guardan en los históricos de InSQL El esquema general del procedimiento almacenado que se utiliza en las consultas de este proyecto puede verse en el Segmento de código 3.1

El procedimiento almacenado toma como valores de entrada la fecha y hora de inicio y fin. Dependiendo del reporte que se desea se crea una tabla temporal, cuyo número de columnas es igual al número de variables que tenga el reporte. Se define una consulta que se ejecutará en la tabla WideHistory. La consulta es del tipo anidado, donde la externa se ejecuta sobre los resultados de la interna. La consulta interna es una sentencia openquery que recibe la fecha inicial y la fecha final, ambas especificada en el formato *aaaa-mm-dd hh:mm:ss* los valores a consultar dependen de la elección del usuario.

Mientras la fecha final sea mayor que la inicial, se seguirán realizando consultas. Cada vez que se realiza una consulta, se aumentan 30 min a la fecha inicial; esto

para evitar la presencia de valores espurios[4] en los datos que retorna la consulta. Los valores de cada consulta se van almacenando como un registro en la tabla temporal. La cual al final almacenará todos los datos a usarse en el reporte final.

```
USE [Runtime]
GO
CREATEPROCEDURE [dbo].[NOMBRE_PROCEDIMIENTO]
@StartDateTime datetime,
@EndDateTime datetime

AS
createtable #nombreTabla
(
       DateTime datetime,
       valor1 varchar(40),
       valor2 varchar(40),
       …
       valor n varchar(40)
while(@StartDateTime <= @EndDateTime)
       BEGIN
       SETNOCOUNTON
       DECLARE @SQLString varchar(8000)
       SET @SQLString =’SET QUOTED_IDENTIFIER OFF ‘
       SET @SQLString = @SQLString +’SELECT DateTime, ROUND(TagName1,2) as
       “Descripcion1”, ‘
       SET @SQLString = @SQLString +’ROUND(TagName2,2) as “Descripcion2”,‘
       . . .
       SET @SQLString = @SQLString +’ ROUND(TagNameN,2) as “DescripcionN”‘
       SET @SQLString = @SQLString +’ FROM OPENQUERY(INSQL, ‘
       SET @SQLString = @SQLString ++’”SELECT DateTime = convert(nvarchar,
       DateTime, 21), ‘
       SET @SQLString = @SQLString +’TagName1, TagName2,...,TagNameN ‘

       SET @SQLString = @SQLString +’FROM WideHistory WHERE TagName1 is not null
       ‘
       SET @SQLString = @SQLString +’AND DateTime =
       ‘+CHAR(39)+CAST(@StartDateTime Asvarchar(50))+CHAR(39)+’”) ‘

       insert into #nombreTabla // …
       EXEC(@SQLString)// …
       set @StartDateTime =Dateadd(mi,30,@StartDateTime)
END
Selectdatetime,valor1 as’Descripcion1’, valor2 as’Descripcion2’,…, valorn
as’DescripcionN’from #nombreTabla
```

Segmento de código 3.1 Esquema de procedimiento almacenado

[4] Los valores espurios son valores que se introducen entre los tiempos configurados en InSQL por un cambio repentino en el estado del sensor (por ejemplo se apaga o sufre sobre voltaje), o cuando el valor supera el rango establecido de valores.

3.4.4.1. NOTAS EXTRAS SOBRE LA IMPLEMENTACIÓN DE LOS PROCEDIMIENTOS

Uno de los procedimientos almacenados requería la entrega de datos de consumo de CO2 cada hora, para lo cual se utilizó otro tipo de código en el procedimiento almacenado, el cual se presenta en el Segmento de código 3.2.

```
CREATE PROCEDURE [dbo].[ProduccionCO2]
  @StartDateTime datetime,
  @EndDateTime datetime
AS
declare  @ProdCO2 TABLE
  (
  fecha datetime,
  valor int
  )
Declare @c1 datetime
Declare @c2 varchar (44)
declare @anterior int
set @anterior = 0
declare @fecha datetime
declare @final datetime
set @fecha = @StartDateTime
set @final = Dateadd(hh,1,@EndDateTime)
WHILE @StartDateTime <= @final
BEGIN
declare cursorProd CURSOR FOR
select datetime, cast(value  as int)
from Runtime.dbo.History where tagname = 'Tot_4' and datetime
=Dateadd(hh,-1,@StartDateTime)
open cursorProd
fetch next from cursorProd
into @c1,@c2
while @@FETCH_STATUS = 0
begin
Insert @ProdCO2 values (@c1,@c2-@anterior)
set @anterior = @c2
fetch next from cursorProd
into @c1,@c2
end
close cursorProd
deallocate cursorProd
set @StartDateTime=DateAdd(hh,1,@StartDateTime)
END
select fecha as 'DateTime',valor as 'Produccion CO2' from @ProdCO2 where
fecha>=@fecha
```

Segmento de código 3.2 Procedimiento almacenado para producción de CO2

La consideración es que el primer valor de la consulta select no se toma en cuenta en el retorno final de los datos, ya que solo se utiliza para recuperar el valor de la hora anterior a la definida en la hora de inicio y así poder recuperar el consumo de esa hora. El consumo se obtiene mediante la diferencia del totalizador de la hora n+1 y el totalizador de la hora n.

De la misma manera se suma una hora a la fecha final con la misma consideración antes mencionada.

Todos los procedimientos almacenados definidos se encuentran en el Anexo 2.

3.4.5. IMPLEMENTACIÓN DE CLASES

Para la implementación de las clases se mostrará solamente el código de algunas clases que servirá como ejemplos, debido a que el código de la aplicación es muy extenso.

Sin embargo se describirá la funcionalidad de cada una de las clases implementadas, así como la funcionalidad de los métodos que contengan.

3.4.5.1. CLASE CONEXIÓN

En la clase se establece creando la cadena de conexión para la base `Login` y la base Runtime. El contenido de la clase se puede ver en el Segmento de código 3.3.

```
Public class Conexion
{
        public string baseRU = "Data Source=ECQTS005;Initial
        Catalog=Runtime;Integrated Security=True"; //Cadena de conexión para
        la base Runtime

        public string baseLogin = "Data Source=ECQTS005;Initial
        Catalog=Login;Integrated Security=True"; //Cadena de conexión parala
        base Login

}
```

Segmento de código 3.3 Esquema de clase conexión

3.4.5.2. CLASE USUARIO

La clase Usuario es una clase plana, es decir que cuenta con atributos, métodos accesores y métodos constructores; como se puede ver en el Segmento de código 3.4. Define los atributos para un usuario registrado en la aplicación.

```
   public class Usuario
{
        //Atributos
        private string id;
        private string nombre;
        ...
        private string nivel;

        //Métodos accesores
        public string Id
        {
           get { return id; }
          set { id = value; }
        }

        public string Nombre
        {
            get { return nombre; }
            set { nombre = value; }
        }
        ...
        // Constructor
        public Usuario(string id, string nombre, string apellido, string
        usuario, string password, string nivel)
       {
              this.id = id;
              this.nombre = nombre;
              this.apellido = apellido;
              this.usuario = usuario;
              this.password = password;
              this.nivel = nivel;
       }

  }
```

Segmento de código 3.4 Clase Usuario

3.4.5.3. CLASE USUARIO DAO

La clase UsuarioDao permite establecer métodos CRUD (Create, Read, Update, Delete) que se ejecutan sobre la tabla de la base de datos User. En casos en los cuales los métodos requieran de valor de retorno con los datos del usuario, se devuelven instancias de la clase Usuario.

Los métodos que define la clase son:

1. El método listaUsuario permite retornar objeto de tipo *List*, que contiene la lista de los usuarios registrados para usar la aplicación. Se devuelve una lista de usuarios con la que se llena una tabla que se presenta a los usuarios con permisos de administrador.
2. EL método numeroUsuarios permite recuperar el número de usuarios para realizar la numeración automática cuando se crea un nuevo usuario.

3. El método crearUsuario inserta un nuevo usuario en la lista de usuarios, utilizando los datos ingresados por el administrador como argumento de entrada.
4. El método actualizarUsuario modifica los datos de un usuario de la lista de usuarios; el método toma como argumento de entrada los datos del usuario y compara el campo id, si lo encuentra modifica los datos.
5. El método eliminarUsuario elimina de la lista de usuarios el usuario cuyo identificador sea igual al enviado en el argumento de entrada.

3.4.5.4. CLASE SISTEMA

La clase Sistema es una clase plana, es decir que cuenta con los atributos, métodos accesores y métodos constructores; como se puede ver en el Segmento de código 3.10.

```
public class Sistema
    {
        //Atributos
        private int id_Sistema;
        private string sistema;
        private string topickey;

        //Métodos accesores
        public int Id_Sistema
        {
            get { return id_Sistema; }
            set { id_Sistema = value; }
        }

        public string Sistema1
        {
            get { return sistema; }
            set { sistema = value; }
        }

        public string Topickey
        {
            get { return topickey; }
            set { topickey = value; }
        }

        // Constructor
          public Sistema(int id, string sistema, string topickey)
            {
                this.id_Sistema = id;
                this.sistema = sistema;
                this.topickey = topickey;
            }
        }
    }
```

Segmento de código 3.5 Clase Sistema

3.4.5.5. CLASE SISTEMA DAO

La clase SistemaDao define los métodos CRUD para la clase sistema. Los métodos listaSistema, crearSistema, actualizarSistema y eliminarSistema tienen una funcionalidad similar a los métodos definidos para la clase usuario.

El momento de crear un nuevo sistema se pide que se designe un `topickey` al sistema el cual debe coincidir con el valor que el servidor InSQL le asigne al momento de crear el tópico y añadir sus variables, de otra forma la aplicación no podrá enlazar las variables al nuevo sistema creado en la aplicación.

El método obtenerTopickey recupera el valor de `topickey` asociado a cada sistema en la base Runtime.

3.4.5.6. CLASE REPORTE DAO

La clase ReporteDao implementa una serie de métodos que permiten construir el reporte que será presentado al usuario a través de la vista de generación de reportes. Aunque no es una clase DAO propiamente se utiliza la notación para debido a que es una clase que accede a la base de datos y en lugar de retornar un objeto devuelve un *dataset* que contendrá los datos de los reportes que se presentaran al usuario.

Los métodos definidos para esta clase se detallan a continuación

1. El método CargarParametrosCR recibe como argumento de entrada un control de visualización de reporte propio de Crystal Report y una variable que corresponde al nombre del usuario. El método define un *ArrayList* que contiene todos los objetos de tipo *ParameterField* que contenga el reporte y añade los parámetros que se requieran-

   En el Segmento de código 3.6 se tiene el código del método que permite establecer la firma de creación de reportes utilizando el nombre del usuario que se autenticó en la aplicación para lo que se utiliza una variable de sesión que contiene el nombre del usuario autenticado cuyo desarrollo se mostrará más adelante

```
public void CargarParametrosCR(CrystalReportViewerrv, stringnom)
{

//Se crea un objeto de tipo campo de parámetro que se utilizará
para presentar la firma de creación de reportes

ParameterFieldparam = newParameterField();

param.ParameterFieldName = "CreadoPor";

ParameterDiscreteValuediscreteValue =
newParameterDiscreteValue();

discreteValue.Value = "Creado por:" + nom;

param.CurrentValues.Add(discreteValue);

//Se define un objeto de tipo arreglo de parámetros al que se
añade el parámetro creado

ParameterFieldsparamFiels = newParameterFields();

paramFiels.Add(param);

rv.ParameterFieldInfo = paramFiels;

}
```

Segmento de código 3.6 Esquema del método CargarParametrosCR

2. El método obtenerDescripcion permite obtener la descripción asociada a un `tagname`. Esta descripción se utilizará en los títulos de las columnas de los reportes combinados y en las etiquetas de las gráficas.
3. La sobrecarga del método CargarParametrosCR permite establecer los títulos de las columnas en los reportes combinados además de la firma de creación utilizando el *ArrayList* de parámetros.
4. El método CargarDatosDS mostrado en el Segmento de código 3.7 retorna un `dataset` con los datos que conformarán el contenido del reporte. Además el método escribe los archivos XML asociados a cada uno de los reportes que se utilizarán como origen de datos para el los esquemas correspondientes. Este archivo se guarda en una carpeta dentro del proyecto.

   Después de la primera ejecución del código el archivo XML es escrito. Se debe asegurar que hayan datos al momento de esta ejecución, de otra manera el esquema se presenta vacío,

```
Public dataSet CargarDatosDS(string inicio, string final, string dbo, string
xmlName)
 {
       //La conexión se hace a la base Runtime
     using (var con = new SqlConnection(con.runtime)
     {
       // Usando el nombre del procesimiento almacenado, se asignan los
       parámetros de entrada de fecha y hora
          using (var cmd = new SqlCommand(dbo, con))
          {
              cmd.CommandType = CommandType.StoredProcedure;

              cmd.Parameters.AddWithValue("@StartDateTime", inicio);

              cmd.Parameters.AddWithValue("@EndDateTime", final);

              // El timeout se establece en 1200 debido a que el tiempo de
              respuesta puede extenderse dependiendo de la cantidad de datos
              que se recuperen
              cmd.CommandTimeout = 1200;

              // Usando el adaptador sql se llena el dataset
              using (SqlDataAdapter da = new SqlDataAdapter(cmd))
              {
                  using (dsSistemaCO2 ds = new dsSistemaCO2())
                  {
                      da.Fill(ds);


                     // Se escribe un archivo XML con el formato del dataset.
                     Este archivo nos servirá para dar el esquema al reporte

                      ds.WriteXml(HttpContext.Current.Server.MapPath("~/xml/"
                     + xmlName));
                      return ds;
                  }
              }
          }
     }

 }
```

Segmento de código 3. 7 Método cargarDatosDs

5. La sobrecarga del método CargarDatosDS se utiliza para recuperar los datos utilizados en los reportes combinados y gráficas, las variables seleccionadas por el usuario se recuperan de las variables de sesión.

6. El método CargarCrystalReport, mostrado en el Segmento de código 3.8, recibe como argumentos de entrada la fecha y hora de inicio y fin; un objeto *ReportDocument* y un objeto *CrystalReportViewer*. Este método permite desplegar en el visor el reporte seleccionado con el conjunto de datos.

```
 Public void CargarCrystalReport(string eleccion, string fechaInicio,
string fechaFinal, ReportDocument crystalReport, CrystalReportViewer
crv)
  {
        crystalReport = new ReportDocument();

      switch (eleccion)
      {
        // Mediante el switch se hace una búsqueda para el segmento de
        código que corresponda al nombre del reporte

        case “NombreReporte”:

       // El dataset almacenará la tabla que contiene los valores del
       reporte. Para llenar la tabla se usa el método CargarDatosDs

       datasetSistema ds =
       baseDatos.CargarDatosDS(fechaInicio, fechaFinal,
       “procedimientoAlmacenado”, “dt”, “sistema.xml”);

 // Se carga en el documento el reporte correspondiente

       crystalReport.Load(HttpContext.Current.Server.MapPath(“Rpt/Nombr
       eReporte.rpt”));
       crystalReport.SetDataSource(ds);
       crv.ReportSource = crystalReport;

 // Para la firma de creación se usa el nombre de usuario que está
 guardado en la variable de sesión “Nombre”

        CargarParametrosCR(crv,HttpContext.Current.Session[“Nombre”].To
        String());

 // El reporte se guarda en una variable de sesión de manera que se
 pueda mantener su estado en la sesión hasta una nueva consulta

        HttpContext.Current.Session.Add(“reporte”, crystalReport);
        crv.DataBind();
        break;
}
```

Segmento de código 3.8 Método sobrescrito de ingreso de parámetros

3.4.5.7. CLASE GRÁFICA DAO

La clase GraficaDao implementa una serie de métodos que permiten generar la gráfica que será presentada al usuario a través de la vista de generación de gráficas.

Los métodos son:

1. El método IntervaloGrafica permite seleccionar la escala de la gráfica que puede ser en horas, días o meses.
2. El método obtenerDescripcion permite recuperar la descripción asociada a un tag para utilizarla en los títulos de las columnas del reporte.

3. El método CargarGrafica retorna un *DataTable* con los datos serán graficados.

3.4.5.8. CLASE PERMISOS DAO

Se implementa una clase que gestiona los permisos de los usuarios de la aplicación, para que se pueda distinguir entre el rol de administrador y el de usuario normal. Aunque actualmente solo se tiene dos roles de usuario y estos se diferencian solamente por el acceso al módulo de administración, se permite escalabilidad en un futuro para definir nuevos niveles con sus respectivos permisos.

El código completo de las clases implementadas en esta aplicación se encuentra en el Anexo 3.

3.4.6. VARIABLES DE SESIÓN

El proyecto utiliza variables de sesión para mantener los valores de usuario mientras se navega por las diferentes páginas. A continuación se describirán las variables que se utilizaron durante el desarrollo de la aplicación.

1. El valor de la variable `Session[“Nombre”]` se establece durante la autenticación, y se muestra al usuario en forma de saludo (Bienvenido Nombre_Usuario) mientras navega por la aplicación. Esta variable se utiliza para mostrar el nombre del usuario que genera el reporte y también para comprobar que exista una sesión iniciada, si la variable es nula se envía nuevamente a la página de registro, evitando que los usuarios accedan a la aplicación mediante una URL de alguna de las diferentes páginas que componen la aplicación.
2. Durante el proceso de autenticación también se recupera el tipo de usuario, que se almacena en la variable `Session[“Nivel”]` y permite discriminar a los usuarios normales de los administradores, restringiendo el acceso a ciertas páginas.

   En el Segmento de código 3.9 se presenta el código del método `Page_Load` de la clase `AdministrarSistema` que muestra el uso de las variables de sesión antes mencionadas.

```
protected void Page_Load(object sender, EventArgs e)
 {
     if (Session["Nombre"] == null)
     {
         Response.Redirect("Ingreso.aspx");
     }
     else
     {
         if (Session["Nivel"].ToString() == "administrador")
             lblBienvenido.Text = Session["Nombre"].ToString();
         else
             Response.Redirect("Principal.aspx");
     }
      lblBienvenido.Text = "Bienvenido " + Session["Nombre"].ToString();

     if (!IsPostBack)
     {
         LlenaGrid(GridViewSistema);
     }
 }
```

Segmento de código 3.9 Método Page_Load de la clase AdministrarSistema

3. La variable de sesión `Session["reporte"]` almacena el objeto reporte que se genera cuando el usuario envía la petición. Esta variable se utiliza para mantener el reporte en el `CrystalReportviewer` cuando se realiza un *postback*, por ejemplo cuando se desea exportar un reporte para algún formato. El código que realiza esta acción se encuentra en el Segmento de código 3.10 y corresponde al método `Page_Init` de la clase `PresentacionReporte`.

```
protected void Page_Init(object sender, EventArgs e)
 {
   if (IsPostBack)
     {
      CrystalReportViewer1.ReportSource = Session["reporte"];
     }

   }
```

Segmento de código 3.10 Método Page_Init de la clase PresentacionReporte

El método `Page_Init` permite que el reporte se mantenga en el momento de realizar el postback. El método `Page_Load` no es útil para esta funcionalidad debido a que a lo largo de la navegación se puede realizar múltiples eventos de postback y se puede mostrar reportes previos al pasar de la página de reportes a la página de reportes combinados.

4. Las variable de sesión `Session["op"]` permite almacenar el número de variables que selecciona el usuario para los reportes combinados o las gráficas, luego se utiliza para recuperar las variables seleccionadas.

3.4.7. IMPLEMENTACIÓN DE UN REPORTE

De acuerdo a lo definido en 3.3.2, la herramienta a utilizarse para generar los reportes es Crystal Reports. Se describirá brevemente las librerías usadas y luego se mostrará cómo se esquematiza el reporte para ser utilizado en la aplicación.

Para trabajar con CR requerimos de la librería Crystal Reports for .NET Framework 4.0, la librería contiene los DLL que se cargan a la aplicación.

Como se muestra en la Figura 3.1, si la instalación de la librería es correcta, al agregar un nuevo elemento de tipo reporte se tiene entra las opciones una plantilla de Crystal Reports con extensión rpt.

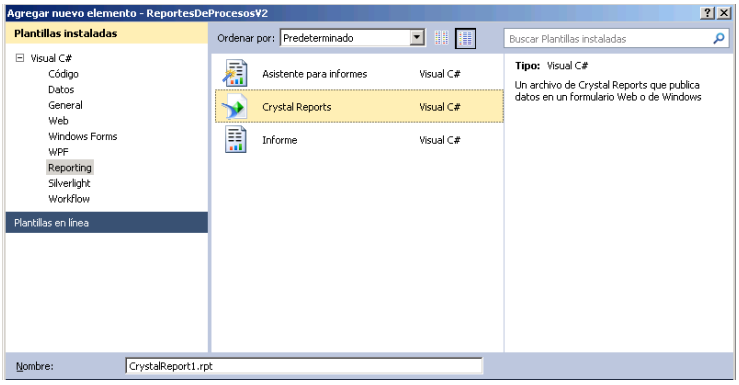

Figura 3.1Plantilla de Crystal Report en Visual Studio

Al escoger la opción se presenta una ventana que permite seleccionar el origen de los datos, sin embargo se puede evitar este paso ya que el origen de datos se hace mediante un archivo XML el cual se muestra en el Segmento de código 3.11.

Usando la interfaz de Visual Studio para editar los reportes y mediante el asistente de base de datos mostrado en la Figura 3.2, se escoge una conexión de tipo ADO.NET (XML) y se busca el archivo XML. Tras realizar este procedimiento se tiene una tabla la cual se usa para llenar los campos del reporte, este procedimiento se debe realizar para cada uno de los reportes con formato fijo y además se debe ubicar manualmente los campos en el reporte, las consideraciones de tamaño de letra y espacios quedan a criterio del desarrollador.

```
<?xml version="1.0" standalone="yes" ?>
- <dsSistemaCO2 xmlns="http://tempuri.org/dsSistemaCO2.xsd">
- <dtProduccionCO2>
<DateTime>01/12/2015 0:00:00</DateTime>
<Produccion_x0020_CO2>170</Produccion_x0020_CO2>
<Consumo_x0020_CO2_x0020_linea_x0020_1>0</Consumo_x0020_CO2_x0020
    _linea_x0020_1>
<Consumo_x0020_CO2_x0020_linea_x0020_2>0</Consumo_x0020_CO2_x0020
    _linea_x0020_2>
<Consumo_x0020_CO2_x0020_elaboracion>165</Consumo_x0020_CO2_x0020_
    elaboracion>
<Stock_x0020_CO2_x0020_tanques>46658</Stock_x0020_CO2_x0020_tanques
    >
</dtProduccionCO2>
- <dtProduccionCO2>
```

Segmento de código 3.11 Esquema XML para creación de reportes

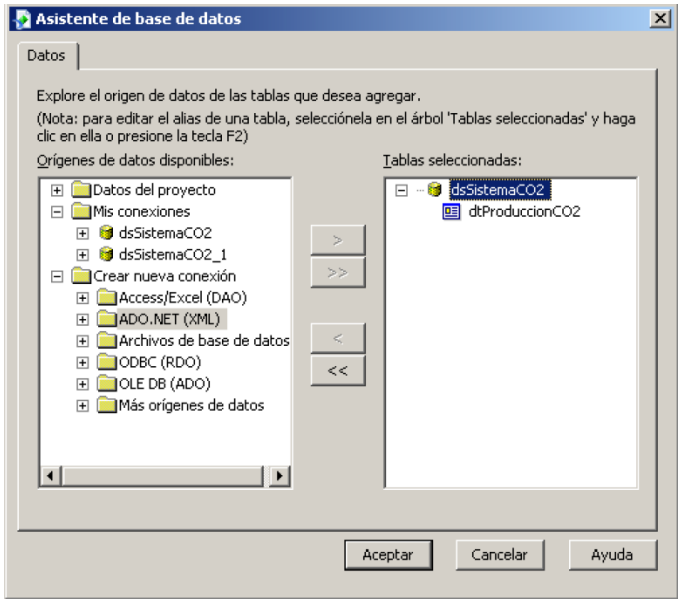

Figura 3.2 Asistente de base de datos de Crystal Reports

Es importante puntualizar que se escogió un esquema XML debido a que los procedimientos almacenados mostraban solo el campo `DateTime`, por el tipo de código que se requería.

La Figura 3.4 muestra cómo se esquematiza el reporte con los campos de la tabla que antes se han seleccionado. Los campos se arrastran hacia el espacio del reporte y se colocan en el lugar donde se desean.

Se usa este tipo de esquema para los reportes ya que una vez que se establecen los campos del reporte solo se necesita el DataSet para llenar los reportes con datos.

En la Figura 3.3, se pueden observar campos de fórmula, que se pueden añadir al reporte para aumentar funcionalidades en el reporte, para el proyecto actual se utilizó un campo de fórmula que mostraba la sumatoria de todos los datos en su columna.

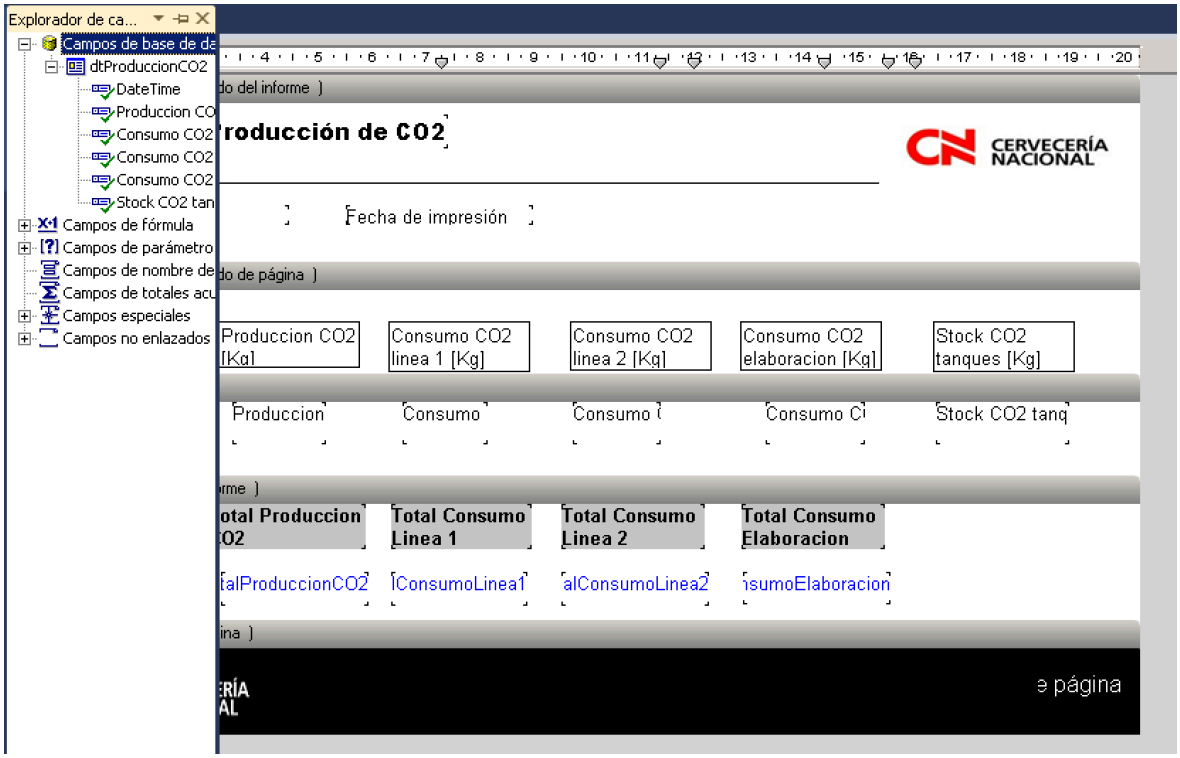

Figura 3.3 Diseño del esquema del reporte

Los XML completos que se utilizaron para crear las tablas que conforman los esquemas de los reportes utilizados en el presente proyecto se encuentran en el Anexo 4.

3.4.8. IMPLEMENTACIÓN DE UNA GRÁFICA

Se utiliza un control Chart para presentar las gráficas de las variables seleccionadas por el usuario, como se puede ver en la Figura 3.4.

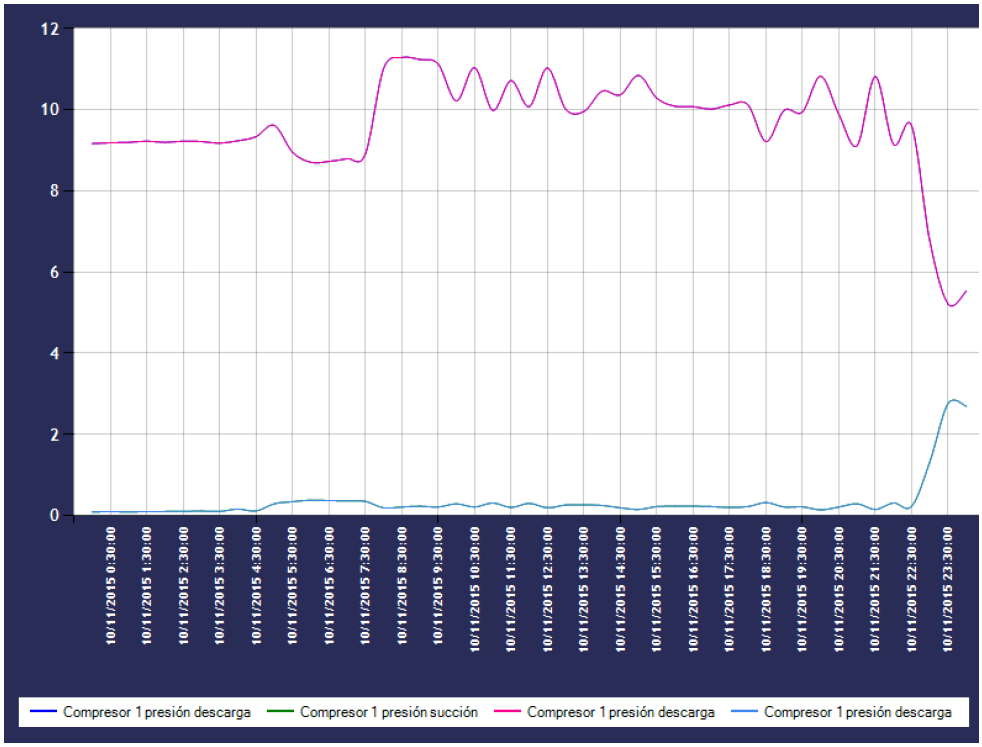

Figura 3.4 Ejemplo de una gráfica generada con la aplicación

Para que la gráfica sea aún más útil, se añade un *tooltip* que entrega el valor, la fecha y hora cuando se coloca el puntero sobre un punto de la gráfica. Esto se realizó con el Segmento de código 3.12.

```
Chart1.Series[“Series1”].LabelToolTip =
“[#DateTime[#valor]]”;

Chart1.Series[“Series1”].ToolTip = “#VALY{#0.00}  #VALX”;
```

Segmento de código 3.12 Código de tooltip

3.4.9. IMPLEMENTCIÓN DE CONTROLES AJAX

En el diseño de las vistas no se designa espacio para desplegar calendarios visibles, debido a que se desea que el diseño sea los más simple e intuitivo posible para el usuario final, por lo que se utiliza controles Ajax Calendar Extender como el mostrado en la Figura 3.5 que se despliegan al dar un clic en el cuadro de texto de fechas.

Para usar estos controles se agrega la referencia *AjaxControlToolKit*, para luego ser colocada en el código de la página como se muestra en el Segmento de código 3.13.

```
<ajaxtoolkit:CalendarExtender id="calendarInicio" runat="server"
      TargetControlID="txtFechaInicio" Format="yyyy-MM-dd"
       CssClass="calendar"></ajaxtoolkit:CalendarExtender>
```

Segmento de código 3. 13 Código correspondiente a calendar extender

Figura 3.5 Calendar Extender

También se utilizan controles *popup* para desplegar las opciones como `CheckBoxList` en la interfaz que presenta las gráficas como se puede ver en la Figura 3.6

Figura 3.6 CheckBoxList Extender

3.4.10. IMPLEMENTACIÓN DE LAS INTERFACES DE USUARIO

En la implementación de las interfaces de usuario se utilizó la funcionalidad *drag and drop* para los controles, para el manejo de posición y propiedades se utilizaron elementos `<div>` y hojas de estilo.

Para la aplicación se diseñó una página maestra que contiene los controles, imágenes y diseño que se mantienen en las páginas. La página maestra se muestra en la Figura 3.7.

Figura 3.7 Página maestra

Las páginas luego se construyen dentro de la etiqueta <body> de la página maestra. Los controles de las páginas para reportes, reportes combinados y gráficas son muy similares, un ejemplo del diseño de estas páginas se puede ver en la figura 3.7 que corresponde a la página de reportes combinados.

Figura 3.8 Diseño de la página reportes combinados

El diseño puede también modificarse mediante la vista de código, que contiene etiquetas y elementos HTML además de los controles propios de ASP.NET, en la vista de código se puede modificar las propiedades de los controles y en el caso de los controles que permiten seleccionar opciones, se puede establecer la colección de ítems.

3.4.10.1. CSS

Como se menciona antes, la posición y las propiedades fueron definidas utilizando una hoja de estilos. Un ejemplo puede verse en el Segmento de código 3.14 que pertenece a la interfaz de la página principal que da la bienvenida al usuario después del proceso de autenticación.

```
#galeria, #galeria * {box-sizing:border-box,-moz-box-sizing:border-box}
#galeria {
border:0 !important;;  /* Borde de la galería */
padding: 0px;
padding-bottom: 0;
background: rgba(159,136,82,0.8);  /* Fondo de la galería */
width: 700px;  /* Ancho de la galería */
}
#galeria_miniaturas {
display: table;
margin: 0 auto;
}
#imgGaleria {
border: 1px solid #F2F2F2;  /* Borde de la imagen */
padding: 3px;
width: x; /* Ancho de la imagen */
height: x; /* Alto de la imagen */
}
.miniatura {
width:  60px;  /* Ancho de las miniaturas */
height:  60px;  /* Alto de las miniaturas */
float: left;
cursor: pointer;
padding: 5px;
```

Segmento de código 3. 14 Ejemplo de hoja de estilos.

La galería como se muestra al momento de ejecución del programa se puede observar en la Figura 3.9, las imágenes corresponden a los sistemas de CO2, refrigeración y planta de agua. Las hojas de estilo utilizadas para la aplicación se pueden encontrar en el Anexo 5.

Figura 3.9 Galería de imágenes con CSS

CAPÍTULO IV

PRUEBAS

4.1. PRUEBAS UNITARIAS

Una prueba unitaria es un método que prueba una unidad de código. Al hablar de una unidad de código se hace referencia a un requerimiento [2]. Se conoce que una prueba unitaria tiene las siguientes características:

- Prueba solamente pequeñas cantidades de código: Solamente prueba el código del requerimiento específico.
- Se prueba solamente los métodos de tipo público: Los métodos de tipo público pueden albergar métodos privados y de esta manera se prueba todos los códigos.
- Los resultados son automatizados: Cuando se ejecuta las pruebas se puede hacer de forma individual o de forma grupal. Estas pruebas las hace el motor de prueba y los resultados de los mismos deben de ser precisos con respecto a cada prueba unitaria desarrollada
- Los resultados son repetibles y predecibles.
- Son códigos sencillos: Las pruebas unitarias deben ser simples y rápidas de desarrollar.

Para el proyecto, en el Segmento de código 4.1 se tiene un ejemplo de prueba unitaria para la obtención del nivel de un usuario

```
[HostType("ASP.NET")]
[AspNetDevelopmentServerHost(ReportesDeProcesosV3", "/")]
 [UrlToTest("http://localhost:1213/Ingreso.aspx")]
        public void obtenerNivelTest()
        {
            Ingreso target = new Ingreso();
            string usuario = "mlive";
            // Registramos el valor de nivel que deberíamos tener para ese
            usuario
            string expected = "administrador";
            string actual;
            actual = target.obtenerNivel(usuario);
            // La clase assert comprueba las condiciones registradas y
            devuelve un valor
            Assert.AreEqual(expected, actual);
        }
```

Segmento de código 4.1 Ejemplo de prueba unitaria

Al ejecutar la prueba en el IDE, se obtiene el resultado de la prueba, como se muestra en la Figura 4.1

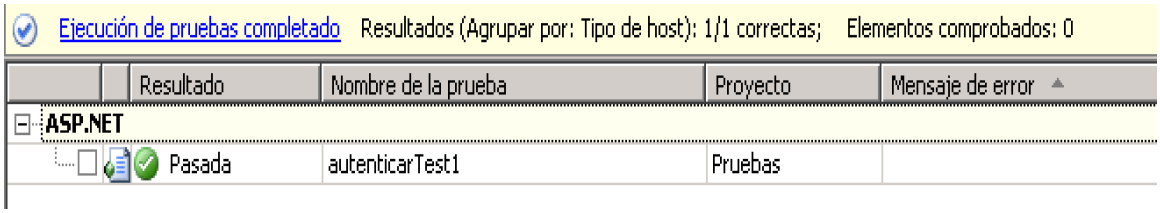

Figura 4.1 Prueba aceptada

En el Segmento de código 4.2 se puede observar el código para la prueba unitaria en la cual se prueba el código que retorna un `dataset` para generar las gráficas.

```
[TestMethod()]
  [HostType("ASP.NET")]
      [AspNetDevelopmentServerHost("ReportesDeProcesosV3", "/")]
  [UrlToTest("http://localhost:1213/Graficas.aspx")]
  public void CargarGraficaTest()
  {
      GraficaDAO target = new GraficaDAO();
      DataTable dt = new dsSistemaCO2.dtUnaVariableDataTable();
      string inicio = "2015-11-10"; // Fecha inicial
      string fin = "2015-11-11"; // Fecha final
      string variable = "CO2Pressure"; // Nombre de variable
       // Nombre de procedimiento almacenado
      string database = "dbo.EscogerTag";
      int n = 1; // número de variables seleccionadas
      DataTable expected = dt;
      DataTable actual;
      actual = target.CargarGrafica(dt, inicio, fin, variable,
      database, n);
      Assert.AreEqual(expected, actual);

  }
```

Segmento de código 4.2 Prueba unitaria del método CargarGráfica

El código de cada una de las pruebas de unitarias realizadas para la aplicación se encuentra en el Anexo 6.

4.2. PRUEBAS DE CARGA

Las pruebas de carga se realizaron utilizando la herramienta *Webserver Stress Tool*, un *freeware* que permite generar pruebas de estrés y carga.

La prueba se ejecutó con 50 usuarios simultáneos accediendo a la aplicación y realizando clics de manera aleatoria. En la Figura 4. 2 se puede observar a la aplicación generando la prueba de carga.

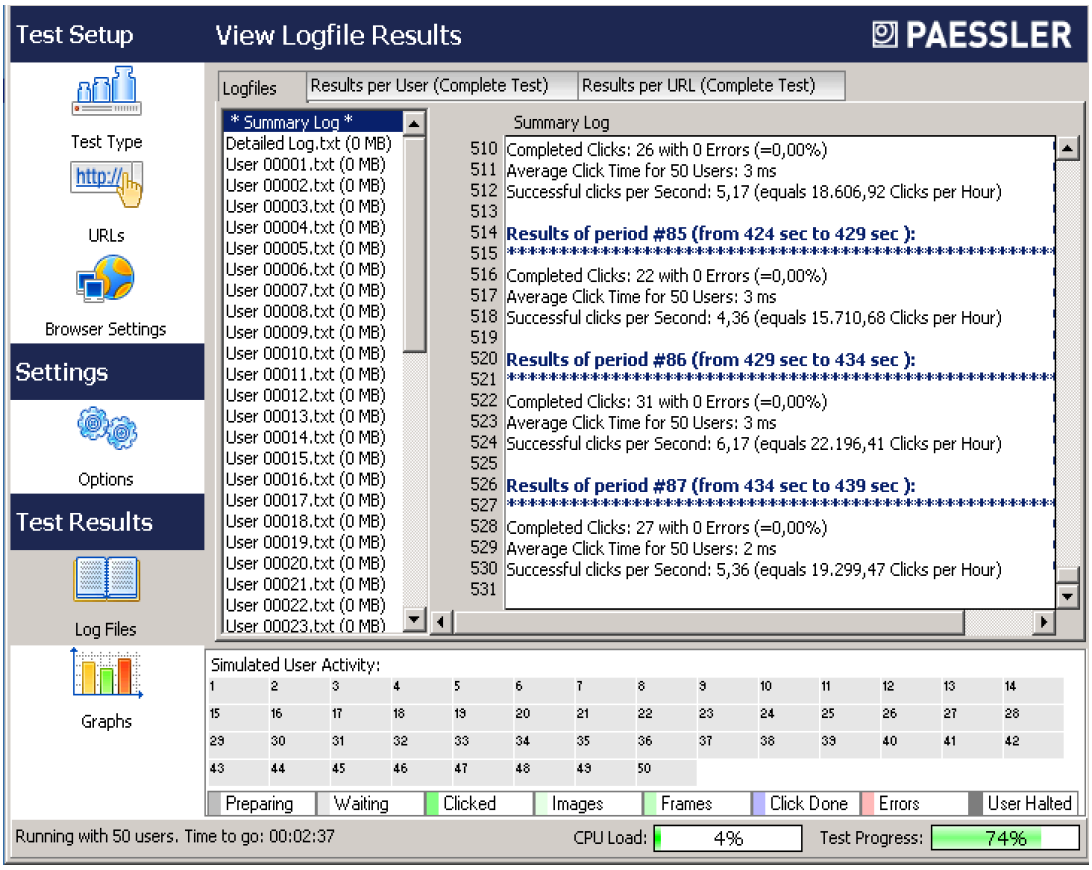

Figura 4.2 Prueba de carga en el 74%

Al final de la prueba se comprobó que no existió falla para los 50 usuarios simultáneos. El resultado de la prueba indica un tiempo promedio de clic de 4 milisegundos, se realizaron 1999 clics y no se produjo ningún error.

Posteriormente se realizó una prueba para 100 usuarios simultáneos. El resultado de la prueba indica un tiempo promedio de clic de 4 milisegundos, realizando 3999 clics y sin producirse ningún error.

Los resultados de las pruebas se pueden ver en la Figura 4.3 a) y b)

URL No.	Name	Clicks	Errors	Errors [%]	Time Spent [ms]	Avg. Click Time [ms]
1	CN	1.999	0	0,00	7.298	4

a) Resultados de la prueba de carga para 50 usuarios

URL No.	Name	Clicks	Errors	Errors [%]	Time Spent [ms]	Avg. Click Time [ms]
1	CN	3.999	0	0,00	16.328	4

Figura 4.3 b) Resultados de la prueba de carga para 100 usuarios

El número de usuarios aproximados para la aplicación es de 30. Con las pruebas de carga realizadas se puede ver que la aplicación puede dar soporte a los usuarios requeridos y abarcar más si fuera necesario.

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

La migración de los reportes con toma de datos manual a un modo automatizado se logró implementando la solución propuesta, que incluye la instalación y configuración de un servidor de base de datos industrial InSQL con la cual se automatizó la recolección de información que posteriormente se presenta en forma de reportes mediante la interfaz de la aplicación web. Con esta solución se logró reemplazar el sistema antiguo de reportes manuales en Excel.

Para obtener los conjuntos de datos que conforman los reportes se utilizó la instrucción `openquery` en conjunto con los procedimientos almacenados. El uso de procedimientos almacenados acortó los tiempos de proceso y recuperación de datos y permite que las consultas sean parametrizadas para que el usuario pueda definir fecha y hora de inicio y fin.

Para el desarrollo de la aplicación se utilizó el patrón de diseño DAO que permitió conectar la base de datos con los módulos de la aplicación. Esto hizo que los métodos de acceso a datos queden separados y modulares. Las clases DAO facilitaron las tareas de administración del usuario y del sistema. Asimismo, fueron usadas para devolver un *dataset* con los datos para cada uno de los distintos reportes.

Se reemplazó el archivo Excel que servía como fuente de datos y de reportes por un módulo de la aplicación que genera reportes los cuales están separados para el sistema de CO2 y refrigeración y son parametrizados. Si el usuario desea un reporte personalizado se utiliza el módulo de reportes combinados que permite combinar hasta cuatro variables. Para la presentación de reportes se utilizó la herramienta de *Crystal Reports* la cual mediante sus controles permitió cumplir con el requerimiento de exportación de los datos en archivos de formatos Excel y PDF.

La prueba de carga realizada determinó que para un acceso simultáneo de hasta 100 usuarios la aplicación continua ejecutándose normalmente lo que cumple con el requerimiento de soportar hasta 30 usuarios.

5.2. RECOMENDACIONES

Se recomienda estudiar adecuadamente los tipos de consulta que se deben utilizar para recuperar los datos de los archivos históricos de InSQL, puesto que los archivos no muestran la información en forma de tablas y requieren de la base de datos `Runtime` en SQL como intermediario para el acceso a los datos.

Se recomienda estudiar adecuadamente el intervalo de almacenamiento de las variables, debido a que algunas en algún caso particular pueden requerir un tiempo menor de muestreo dependiendo del tiempo en el cual puedan mostrar un cambio significativo en su valor.

Los tags en InSQL pueden importarse desde los archivos existentes de SCADA *Intouch* utilizando la herramienta de importación de `tags` propia de InSQL, esto puede ser útil si se tiene una gran cantidad de tags que de otra manera deberán ser ingresados y configurados uno a uno.

Al usar la herramienta Crystal Reports, se debe descargar la librería *crystalreportviewer13* para la compatibilidad con el control de visualización *CrystalRepotViewer* al momento de desplegar la aplicación en el servidor IIS.

Se puede establecer el acceso a los datos en modo *PULL* utilizando el asistente para origen de datos de *Crystal Reports* que presenta todas las conexiones disponibles y permite navegar en las diferentes tablas que componen las bases de datos.

Se puede realizar pruebas de aceptación de la aplicación para medir la satisfacción de uso por parte de los usuarios finales utilizando encuestas.

Al momento de escribir los procedimientos almacenados se recomienda tomar en cuenta el formato del valor que se adquiere, de manera que se pueda trabajar en el valor utilizando funciones de redondeo, multiplicación y división para coincidir con las unidades definidas en los requerimientos.

REFERENCIAS BIBLIOGRÁFICAS

[1] LUJAN, S. “Programación de aplicaciones web: Historia, principios básicos y clientes web”, España, 2014

[2] S.A. “Learn ASP.NET web aplication framework”. 2014 [fecha de consulta: 2016]. Disponible en http://www.tutorialspoint.com/asp.net/asp.net_tutorial.pdf

[3] MSDN, “Página oficial de Microsoft”, 2016 [fecha de consulta: 2016]. Disponible en https://msdn.microsoft.com/es-es/default.aspx

[4] SUTHERLAND, “La Guía de Scrum”, 2013. [Fecha de consulta: 2016]. Disponible en www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf

[5] BOOCH, G., JACOBSON, I., RUMBAUGH, J. “El Lenguaje Unificado de Modelado. Guía del usuario”, 2005

[6] SILBERSCHATZ, A. “Fundamentos de Base de datos”, España, 2002

[7] INVENSYS SYSTEMS, Inc., “IndustrialSQL Server Historian Database Reference”, USA, 2005

[8] BUSINESS OBJECTS, “Manual del usuario de Crystal Reports XI”, 2004

[9] ADVANTAGE, “Choose the right data access strategy when using Crystal Reports with .NET”, 2003 [fiche de consulta: 2016]. Disponible en http://www.techrepublic.com/article/choose-the-right-data-access-strategy-when-using-crystal-reports-with-net/

[10] NA, “Setting up PHP to work on Windows Server 2003”, 2010 [Fecha de consulta: 2016]. Disponible en http://www.visualwin.com/PHP/

[11] NA, “Configuring JSP for IIS”, 2010 [fecha de consulta: 2016]. Disponible en https://neosmart.net/blog/2006/configuring-jsp-for-iis/

ANEXOS

ANEXO 1

Discriminación de variables para ser almacenadas en la base de datos para el sistema de CO2.

SISTEMA DE CO2: BAJA PRESIÓN

BAJA PRESIÓN					
Sensor	Nombre de la variable	Tipo de TAG	Unidades	Intervalo de tiempo [min]	Descripción
XE	XE0110_0003	Evento		30	Medición nivel de espuma/Apertura y cierre de válvulas
XE	XE0110_0103	Evento		30	Medición nivel de espuma/Apertura y cierre de válvulas
QT	QT0110_0501	Análogo	ppm	30	Medida de calidad de CO2
PT	PT0110_0502	Análogo	mB	30	Medida de presión
LT	LT0410_0001	Análogo	%	30	Porcentaje de llenado de globo
M	M0410_0501	Análogo	h	30	Contador número de horas trabajadas
M	M0510_0704	Análogo	h	30	Contador número de horas trabajadas
MB	P0510_0701	Análogo	h	30	Contador número de horas trabajadas

SISTEMA DE CO2: COMPRESORES

COMPRESORES					
Compresor 1					
Sensor	Nombre de la variable	Tipo de TAG	Unidades	Intervalo de tiempo [min]	Función

PT	PT0611_0025	Análogo	Bar	30	Medida de presión de aceite a la entrada del compresor
TT	PT0611_0026	Análogo	℃	30	Medida de temperatura de aceite a la entrada del compresor
PT	PT0611_0014	Análogo	Bar	30	Medida de presión en la primera etapa
TT	TT0611_0003	Análogo	℃	30	Medida de temperatura en la primera etapa
TT	TT0611_0007	Análogo	℃	30	Medida de temperatura en la segunda etapa
PT	PT0611_0015	Análogo	Bar	30	Medida de la presión en la segunda etapa
TT	TT0611_0403	Análogo	℃	30	Medida de la temperatura a la salida
V	V0610_0703	Evento		30	Indica apertura de válvula al 50 %
V	V0610_0704	Evento		30	Indica apertura de válvula al 100%
M	M0611_0001	Análogo	h	30	Número de horas trabajadas
Compresor 2					
Sensor	Nombre de la variable	Tipo de TAG	Unidades	Intervalo de tiempo [min]	Función
PT	PT0612_0025	Análogo	Bar	30	Medida de presión de aceite a la entrada del compresor
TT	PT0612_0026	Análogo	℃	30	Medida de temperatura de

					aceite a la entrada del compresor
PT	PT0612_0014	Análogo	Bar	30	Medida de presión en la primera etapa
TT	TT0612_0003	Análogo	℃	30	Medida de temperatura en la primera etapa
TT	TT0612_0007	Análogo	℃	30	Medida de temperatura en la segunda etapa
PT	PT0612_0015	Análogo	Bar	30	Medida de la presión en la segunda etapa
TT	TT0612_0403	Análogo	℃	30	Medida de la temperatura a la salida
V	V0612_0703	Evento		30	Indica apertura de válvula al 50 %
V	V0612_0704	Evento		30	Indica apertura de válvula al 100%
M	M0612_0001	Análogo	h	30	Número de horas trabajadas

SISTEMA DE CO2: ALTA PRESIÓN

ALTA PRESIÓN					
Sensor	Nombre de la variable	Tipo de TAG	Unidades	Intervalo de tiempo [min]	Función
FT	FT0510_1001	Análogo	lt/h	30	Medida de flujo de agua
TT	TT0710_1001	Análogo	℃	30	Medida de temperatura a la salida del pre enfriador

P	CO2pressure	Análogo	Bar	30	Medida de presión de evaporación del pre enfriador
B	P0510_1005	Análogo	h	30	Número de horas de trabajo bomba de agua alta presión

SISTEMA DE CO2: SECADORAS

SECADORAS					
Sensor	Nombre de la variable	Tipo de TAG	Unidades	Intervalo de tiempo [min]	Función
FT	FT0710_0601	Análogo	Kg/h	30	Medida de flujo a las salida de las secadoras
TT	TT0710_0602	Análogo	℃	30	Medida de temperatura a la salida de las secadoras
PT	PT2610_1801	Análogo	Bar	30	Medida de presión a la salida de las secadoras
TT	TT0710_0115	Análogo	C	30	Temperatura máxima de regeneración
TT	TT0710_0215	Análogo	C	30	Temperatura del sistema

SISTEMA DE CO2: CO2 4U

CO2 4U					
Sensor	Nombre de la variable	Tipo de TAG	Unidades	Intervalo de tiempo [min]	Función
PT	PT0610_1801	Análogo	Bar	30	Medida de presión de entrada
PT	PT0610_2490	Análogo	Bar	30	Medida de presión de licuefacción

SISTEMA DE CO2: MYCOM NH3

MyCOM NH3					
Compresor 1					
Sensor	Nombre de la variable	Tipo de TAG	Unidades	Intervalo de tiempo [min]	Función
PT	C1_SucctionPressure	Análogo	Bar	30	Presión de succión
PT	C1_DischargePressure	Análogo	Bar	30	Presión de descarga
Compresor 2					
Sensor	Identificación	Tipo de TAG	Unidades	Intervalo de tiempo [min]	Función
PT	C2_SucctionPressure	Análogo	Bar	30	Presión de succión

PT	C2_DischargePressure	Análogo	Bar	30	Presión de descarga

VARIABLES ADICIONALES

Dirección de memoria	Nombre de la variable	Descripción
DB61,REAL 60	C1_OilPressure	Presión de la bomba de aceite
DB61, REAL80	C1_M1Current	Corriente del compresor
DB61,REAL68	C1_TempOil	Temperatura de bomba de aceite
DB61,REAL72	C1_SlideValvePosition	Frecuencia del compresor
DB61,REAL 44	C1_DischargeTemp	Temperatura de descarga
DB61,REAL56	C1_SuccionTemp	Temperatura de succión
DB61,REAL64	C1_PressureOilAfter	Presión de bomba de aceite después del filtro
DB61,REAL76	C1_Frecuency	Válvula de deslizamiento HS
DB61,REAL108	C1_Economizer	Temperatura de economizador
DB61,REAL132,4	C1_EconomizerOut	Válvula de salida de economizador

ANEXO 2

PROCEDIMIENTOS ALMACENADOS

FILTRO CARBON

```
ALTER PROCEDURE [dbo].[CarbonFilter]
@StartDateTime datetime,
@EndDateTime datetime
AS
create table #CF
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40),
      valor5 varchar (40),
      valor6 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN
SET NOCOUNT ON
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT a.DateTime,
Round(a.AF1T1_TEMPERATURE,2) as "Temperatura en filtro AF1", '
SET @SQLString = @SQLString + ' Round(a.AF1F1_FLOW,2) as "Flujo en filtro
AF1", '
SET @SQLString = @SQLString + ' case when a.AF1_ACTUAL<b.AF1_ACTUAL THEN
"REGENERACION" ELSE "NORMAL" END as "Estado filtro AF1", '
SET @SQLString = @SQLString + ' Round(a.A@@1,2) as "Temperatura en filtro
AF2", '
SET @SQLString = @SQLString + ' Round(a.AF2F1_FLOW,2) as "Flujo en filtro
AF2", '
SET @SQLString = @SQLString + ' case when a.AF2_ACTUAL<b.AF2_ACTUAL THEN
"REGENERACION" ELSE "NORMAL" END as "Estado filtro AF2" '

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21),AF1T1_TEMPERATURE, '
SET @SQLString = @SQLString +  'AF1F1_FLOW,AF1_ACTUAL,
A@@1,AF2F1_FLOW,AF2_ACTUAL '
SET @SQLString = @SQLString + 'FROM WideHistory  WHERE AF1F1_FLOW is not
null   '
SET @SQLString = @SQLString + 'AND DateTime = '+ CHAR(39) +
CONVERT(varchar(70),@StartDateTime,21) + CHAR(39)+ '")a, '

SET @SQLString = @SQLString + ' OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21),AF1T1_TEMPERATURE, '
SET @SQLString = @SQLString +  'AF1F1_FLOW,AF1_ACTUAL,
A@@1,AF2F1_FLOW,AF2_ACTUAL '
SET @SQLString = @SQLString + 'FROM WideHistory  WHERE AF1_ACTUAL is not
null   '
SET @SQLString = @SQLString + 'AND DateTime = '+  CHAR(39)
+Convert(varchar(70),DateAdd(hh,-1,@StartDateTime),21) + CHAR(39)+ '")b
```

```
where Datediff(hh,b.DateTime,a.DateTime)=1 group by a.DateTime,'
SET @SQLString = @SQLString + '
a.AF1T1_TEMPERATURE,a.AF1F1_FLOW,b.AF1F1_FLOW, a.AF1_ACTUAL,b.AF1_ACTUAL,
a.A@@1,a.AF2F1_FLOW,b.AF2F1_FLOW,a.AF2_ACTUAL,b.AF2_ACTUAL order by
a.DateTime asc '


insert into #CF
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime,valor1 as 'Temperatura en filtro AF1', valor2 as 'Flujo
en filtro AF1', valor3 as 'Estado filtro AF1',
valor4 as 'Temperatura en filtro AF2', valor5 as 'Flujo en filtro AF2',
valor6 as 'Estado filtro AF2'from #CF



/****** Objeto:  StoredProcedure [dbo].[CO2MyCOM1]    Fecha de la
secuencia de comandos: 11/19/2015 11:00:15 ******/
SET ANSI_NULLS ON
```

CO2 4U

```
ALTER PROCEDURE [dbo].[CO24U]
@StartDateTime datetime,
    @EndDateTime datetime
AS
create table #C4U
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40),
      valor5 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN
SET NOCOUNT ON

DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime,
ROUND(LT2610_2005_VALUE,2) as "Nivel en reboiler", '
SET @SQLString = @SQLString + ' ROUND(PT2610_2009_VALUE,2) as "Presion en
reboiler",  '
SET @SQLString = @SQLString + ' CASE WHEN P2610_2103_STATUS ="1" THEN
"Encendido" ELSE "Apagado" END as "Estado bomba CO2", '
SET @SQLString = @SQLString + ' Round(P2610_2103_hours,2) as "Horas
operacion bomba CO2",'
SET @SQLString = @SQLString + ' ROUND(LT2610_2209_VALUE,2) as "Nivel
tanque pulmon" '

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), '
SET @SQLString = @SQLString +  'LT2610_2005_VALUE, PT2610_2009_VALUE,
```

```
P2610_2103_STATUS,P2610_2103_hours,LT2610_2209_VALUE '

SET @SQLString = @SQLString + 'FROM WideHistory WHERE LT2610_2209_VALUE
is not null   '
SET @SQLString = @SQLString + 'AND DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+'") '
insert into #C4U
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END
select datetime,valor1 as 'Nivel en reboiler', valor2 as 'Presion en
reboiler', valor3 as 'Estado bomba CO2',
valor4 as 'Horas operacion bomba CO2', valor5 as 'Nivel tanque pulmon'
from #C4U
```

CO2 MyCOM 1

```
ALTER PROCEDURE [dbo].[CO2MyCOM1]
@StartDateTime datetime,
@EndDateTime datetime
AS
create table #MYCOM1
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40),
      valor5 varchar (40),
      valor6 varchar (40),
      valor7 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN
SET NOCOUNT ON
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime,
ROUND(C1_SlideValvePosition,2) as "Frecuencia MyCOM1_CO2", '
SET @SQLString = @SQLString + ' ROUND(CoolingCompressor1_Par4,2) as
"Corriente MyCOM1_CO2", '
SET @SQLString = @SQLString + ' ROUND(C1_DischargePressure,2) as "Presion
descarga MyCOM1_CO2", '
SET @SQLString = @SQLString + ' ROUND(C1_SuctionPressure,2) as "Presion
succion MyCOM1_CO2", '
SET @SQLString = @SQLString + ' ROUND(C1_SuctionTemperature,2) as
"Temperatura succion MyCOM1_CO2", '
SET @SQLString = @SQLString + ' ROUND(C1_OilPressure,2) as "Presion
aceite MyCOM1_CO2", '
SET @SQLString = @SQLString + ' ROUND(C1_OilTemp,2) as "Temperatura
aceite MyCOM1_CO2" '


SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), '
SET @SQLString = @SQLString +  'CoolingCompressor1_Par4,
C1_DischargePressure, '
SET @SQLString = @SQLString +
```

```
'C1_SuctionPressure,C1_SuctionTemperature,C1_SlideValvePosition,C1_OilPre
ssure,C1_OilTemp  '

SET @SQLString = @SQLString + 'FROM WideHistory WHERE
CoolingCompressor1_Par4 is not null   '
SET @SQLString = @SQLString + 'AND DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+ '") '

insert into #MYCOM1
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime,valor1 as 'Frecuencia MyCOM1_CO2', valor2 as 'Corriente
MyCOM1_CO2', valor3 as 'Presion descarga MyCOM1_CO2',
valor4 as 'Presion succion MyCOM1_CO2', valor5 as 'Temperatura succion
MyCOM1_CO2', valor6 as 'Presion aceite MyCOM1_CO2',
valor7 as 'Temperatura aceite MyCOM1_CO2' from #MYCOM1



/****** Objeto:  StoredProcedure [dbo].[CO2MyCOM2]    Fecha de la
secuencia de comandos: 11/19/2015 11:00:37 ******/
SET ANSI_NULLS ON
```

CO2 MyCOM 2

```
ALTER PROCEDURE [dbo].[CO2MyCOM2]
@StartDateTime datetime,
@EndDateTime datetime
AS
create table #MYCOM2
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40),
      valor5 varchar (40),
      valor6 varchar (40),
      valor7 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN
SET NOCOUNT ON
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime,
ROUND(C2_SlideValvePosition,2) as "Frecuencia MyCOM2_CO2", '
SET @SQLString = @SQLString + ' ROUND(CoolingCompressor2_Par4,2) as
"Corriente MyCOM2_CO2", '
SET @SQLString = @SQLString + ' ROUND(C2_DischargePressure,2) as "Presion
descarga MyCOM2_CO2", '
SET @SQLString = @SQLString + ' ROUND(C2_SuctionPressure,2) as "Presion
succion MyCOM2_CO2", '
SET @SQLString = @SQLString + ' ROUND(C2_SuccionTemperature,2) as
"Temperatura succion MyCOM2_CO2", '
SET @SQLString = @SQLString + ' ROUND(C2_OilPressure,2) as "Presion
aceite MyCOM2_CO2", '
```

```
SET @SQLString = @SQLString + ' ROUND(C2_OillTemp,2) as "Temperatura
aceite MyCOM2_CO2" '


SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21),CoolingCompressor2_Par4,'
SET @SQLString = @SQLString +  'C2_SlideValvePosition,
C2_DischargePressure, '
SET @SQLString = @SQLString +  'C2_SuctionPressure,C2_SuccionTemperature,
C2_OilPressure,C2_OillTemp '

SET @SQLString = @SQLString + 'FROM WideHistory WHERE C2_SuctionPressure
is not null   '
SET @SQLString = @SQLString + 'AND DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+ '") '

insert into #MYCOM2
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime,valor1 as 'Frecuencia MyCOM2_CO2', valor2 as 'Corriente
MyCOM2_CO2', valor3 as 'Presion descarga MyCOM2_CO2',
valor4 as 'Presion succion MyCOM2_CO2', valor5 as 'Temperatura succion
MyCOM2_CO2', valor6 as 'Presion aceite MyCOM2_CO2',
valor7 as 'Temperatura aceite MyCOM2_CO2' from #MYCOM2
```

PROCEDIMIENTO PARA VARIABLES COMBINADAS

```
ALTER PROCEDURE [dbo].[Comparar4Valores]
      @StartDateTime datetime,
      @EndDateTime datetime,
      @tag1 varchar(50),
      @tag2 varchar(50),
      @tag3 varchar(50),
      @tag4 varchar(50),
      @name1 varchar(100),
      @name2 varchar(100),
      @name3 varchar(100),
      @name4 varchar(100)
AS
create table #Comp4Val
(
      DateTime datetime,
      v1 varchar (40),
      v2 varchar (40),
      v3 varchar (40),
      v4 varchar (40)
)

WHILE @StartDateTime <= @EndDateTime
BEGIN
SET NOCOUNT ON
DECLARE @SQLString varchar(800)
DECLARE @SQLFinal varchar(800)
```

```
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime, ROUND('+ @tag1 +',2) as
'+ @name1 +', ROUND('+ @tag2 +',2) as '+ @name2 +',ROUND('
SET @SQLString = @SQLString + + @tag3 +',2) as '+ @name3 +', ROUND('+
@tag4 +',2) as '+ @name4 +' FROM OPENQUERY(INSQL,'
SET @SQLString = @SQLString + + '"SELECT DateTime , ' + @Tag1 + ', '+
@Tag2 + ', '+ @Tag3 + ', '+ @Tag4+' '

SET @SQLString = @SQLString + 'FROM WideHistory where '
SET @SQLString = @SQLString + 'DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+ '") '

insert into #Comp4Val
EXEC (@SQLString)
set @StartDateTime=DateAdd(mi,30,@StartDateTime)

END

set @SQLFinal = 'select Datetime, v1 as '+ @name1+',v2 as '+@name2+',v3
as '+@name3+',v4 as '+@name4+' from #Comp4Val'
exec (@SQLFinal)
```

COMPRESOR 1 CO2

```
ALTER PROCEDURE [dbo].[CompresorCO2_1]
      @StartDateTime datetime,
      @EndDateTime datetime
AS
create table #C1CO2
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40),
      valor5 varchar (40),
      valor6 varchar (40),
      valor7 varchar (40),
      valor8 varchar (40),
      valor9 varchar (40),
      valor10 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN
SET NOCOUNT ON
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime, M0611_0001_hours as
"Horas operacion", '
SET @SQLString = @SQLString + ' ROUND(PT0611_0025_VALUE,2) as "Presion
entrada", '
SET @SQLString = @SQLString + ' ROUND(PT0611_0026_VALUE,2) as
"Temperatura entrada", '
SET @SQLString = @SQLString + ' CASE WHEN V0610_0703_STATUS = "1" THEN
"50" ELSE "0" END AS "50% Capacidad", '
SET @SQLString = @SQLString + ' CASE WHEN V0610_0704_STATUS = "1" THEN
"100" ELSE "0" END AS "100% Capacidad", '
SET @SQLString = @SQLString + ' ROUND(PT0611_0014_VALUE,2) as "Presion 1
etapa", '
```

```
SET @SQLString = @SQLString + ' ROUND(TT0611_0003_VALUE,2) as
"Temperatura 1 etapa", '
SET @SQLString = @SQLString + ' ROUND(TT0611_0403_VALUE,2) as
"Temperatura salida 1 etapa", '
SET @SQLString = @SQLString + ' ROUND(PT0611_0015_VALUE,2) as "Presion 2
etapa", '
SET @SQLString = @SQLString + ' ROUND(TT0611_0007_VALUE,2) as
"Temperatura 2 etapa" '

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), '
SET @SQLString = @SQLString +  'M0611_0001_hours, PT0611_0025_VALUE,
PT0611_0026_VALUE, '
SET @SQLString = @SQLString +  'V0610_0703_STATUS,V0610_0704_STATUS,
PT0611_0014_VALUE, '
SET @SQLString = @SQLString +  'TT0611_0003_VALUE,TT0611_0403_VALUE,
PT0611_0015_VALUE, '
SET @SQLString = @SQLString +  'TT0611_0007_VALUE '

SET @SQLString = @SQLString + 'FROM WideHistory WHERE PT0611_0014_VALUE
is not null   '
SET @SQLString = @SQLString + 'AND DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+ '") '

insert into #C1CO2
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END
select datetime,valor1 as 'Horas operacion', valor2 as 'Presion entrada',
valor3 as 'Temperatura entrada', valor4 as '50% Capacidad',
valor5 as '100% Capacidad', valor6 as 'Presion 1 etapa', valor7 as
'Temperatura 1 etapa', valor8 as 'Temperatura salida 1 etapa',
valor9 as 'Presion 2 etapa', valor10 as 'Temperatura 2 etapa' from #C1CO2
```

COMPRESOR 2 CO2

```
ALTER  PROCEDURE [dbo].[CompresorCO2_2]
      @StartDateTime datetime,
      @EndDateTime datetime

AS
create table #C2CO2
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40),
      valor5 varchar (40),
      valor6 varchar (40),
      valor7 varchar (40),
      valor8 varchar (40),
      valor9 varchar (40),
      valor10 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
```

```
BEGIN
SET NOCOUNT ON
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime, M0612_0001_hours as
"Horas operacion", '
SET @SQLString = @SQLString + ' ROUND(PT0612_0025_VALUE,2) as "Presion
entrada", '
SET @SQLString = @SQLString + ' ROUND(PT0612_0026_VALUE,2) as
"Temperatura entrada", '
SET @SQLString = @SQLString + ' CASE WHEN V0612_0703_STATUS = "1" THEN
"50" ELSE "0" END AS "50% Capacidad", '
SET @SQLString = @SQLString + ' CASE WHEN V0612_0704_STATUS = "1" THEN
"100" ELSE "0" END AS "100% Capacidad", '
SET @SQLString = @SQLString + ' ROUND(PT0612_0014_VALUE,2) as "Presion 1
etapa", '
SET @SQLString = @SQLString + ' ROUND(TT0612_0003_VALUE,2) as
"Temperatura 1 etapa", '
SET @SQLString = @SQLString + ' ROUND(TT0612_0403_VALUE,2) as
"Temperatura salida 1 etapa", '
SET @SQLString = @SQLString + ' ROUND(PT0612_0015_VALUE,2) as "Presion 2
etapa", '
SET @SQLString = @SQLString + ' ROUND(TT0612_0007_VALUE,2) as
"Temperatura 2 etapa" '

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), '
SET @SQLString = @SQLString +  'M0612_0001_hours, PT0612_0025_VALUE,
PT0612_0026_VALUE, '
SET @SQLString = @SQLString +  'V0612_0703_STATUS,V0612_0704_STATUS,
PT0612_0014_VALUE, '
SET @SQLString = @SQLString +  'TT0612_0003_VALUE,TT0612_0403_VALUE,
PT0612_0015_VALUE, '
SET @SQLString = @SQLString +  'TT0612_0007_VALUE '

SET @SQLString = @SQLString + ' FROM WideHistory WHERE PT0612_0014_VALUE
is not null   '
SET @SQLString = @SQLString + 'AND DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+ '") '
insert into #C2CO2
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime,valor1 as 'Horas operacion', valor2 as 'Presion entrada',
valor3 as 'Temperatura entrada', valor4 as '50% Capacidad',
valor5 as '100% Capacidad', valor6 as 'Presion 1 etapa', valor7 as
'Temperatura 1 etapa', valor8 as 'Temperatura salida 1 etapa',
valor9 as 'Presion 2 etapa', valor10 as 'Temperatura 2 etapa' from #C2CO2
```

CONDENSADOR 8

```
ALTER PROCEDURE [dbo].[Condensador8]
@StartDateTime datetime,
 @EndDateTime datetime
AS
 declare  @EstadoCond TABLE
      (
      fecha datetime,
```

```
      valor varchar(44)
      )

 declare @EstadoVent TABLE
      (
      fecha datetime,
      valor varchar(44)
      )

 declare @VarVent TABLE
      (
      fecha datetime,
      valor varchar(44)
      )
 declare @EstadoBomb TABLE
      (
      fecha datetime,
      valor varchar(44)
      )
 declare @SP TABLE
      (
      fecha datetime,
      valor varchar(44)
      )

WHILE @StartDateTime < @EndDateTime
BEGIN


      insert into @EstadoCond(fecha,valor)
      Select datetime,case when value = '1' then 'manual' else
'automatico' end from Runtime.dbo.History where
tagname='Evaporador_8_manual' and datetime = @StartDateTime
      insert into @EstadoVent(fecha,valor)
      select datetime,case when value = '0' then 'apagado' else
'encendido'end from Runtime.dbo.History where tagname='Ventilador_8_ON'
and datetime = @StartDateTime
      insert into @VarVent(fecha,valor)
      select datetime,case when value = '0' then '50' else '100'end from
Runtime.dbo.History where tagname='Ventilador_8_ON' and datetime =
@StartDateTime
      insert into @EstadoBomb(fecha,valor)
      select datetime,case when value = '0' then 'apagada' else
'encendida'end from Runtime.dbo.History where tagname='Bomba_8_ON' and
datetime = @StartDateTime
      insert into @SP(fecha,valor)
      select datetime,Round(value,2) from Runtime.dbo.History where
tagname='Presion_Descarga_N' and datetime = @StartDateTime
      set @StartDateTime=DateAdd(mi,30,@StartDateTime)
End
      select a.fecha as 'DateTime',a.valor as 'Estado
condensador',b.valor as 'Estado ventilador',c.valor as 'Variacion
velocidad ventilador', d.valor as 'Estado bomba', e.valor as 'Set point'
from (((@EstadoCond a inner join @EstadoVent b  on a.fecha =
b.fecha)inner join @VarVent c on a.fecha = c.fecha)inner join @EstadoBomb
d on a.fecha = d.fecha)inner join @SP e on a.fecha = e.fecha
```

CONDENSADOR 9

```
ALTER PROCEDURE [dbo].[Condensador9]
```

```
@StartDateTime datetime,
 @EndDateTime datetime
AS
 declare  @EstadoCond TABLE
       (
       fecha datetime,
       valor varchar(44)
       )
 declare @EstadoVent1 TABLE
       (
       fecha datetime,
       valor varchar(44)
       )

 declare @VarVent TABLE
       (
       fecha datetime,
       valor varchar(44)
       )
 declare @EstadoVent2 TABLE
       (
       fecha datetime,
       valor varchar(44)
       )

 declare @EstadoBomb1 TABLE
       (
       fecha datetime,
       valor varchar(44)
       )
 declare @EstadoBomb2 TABLE
       (
       fecha datetime,
       valor varchar(44)
       )
 declare @SP TABLE
       (
       fecha datetime,
       valor varchar(44)
       )

WHILE @StartDateTime < @EndDateTime
BEGIN


       insert into @EstadoCond(fecha,valor)
       Select datetime,case when value = '1' then 'manual' else
'automatico' end from Runtime.dbo.History where
tagname='Evaporador_9_2_manual' and datetime = @StartDateTime
       insert into @EstadoVent1(fecha,valor)
       select datetime,case when value = '0' then 'apagado' else
'encendido'end from Runtime.dbo.History where tagname='Ventilador_9_1_On'
and datetime = @StartDateTime
       insert into @EstadoVent2(fecha,valor)
       select datetime,case when value = '0' then 'apagado' else
'encendido'end from Runtime.dbo.History where tagname='Ventilador_9_2_On'
and datetime = @StartDateTime
       insert into @VarVent(fecha,valor)
       select datetime,Round(value,1) from Runtime.dbo.History where
tagname='Lmn_Pid_Evaporadores' and datetime = @StartDateTime
```

```
	insert into @EstadoBomb1(fecha,valor)
	select datetime,case when value = '0' then 'apagada' else
'encendida'end from Runtime.dbo.History where tagname='Bomba_9_1_ON' and
datetime = @StartDateTime
	insert into @EstadoBomb2(fecha,valor)
	select datetime,case when value = '0' then 'apagada' else
'encendida'end from Runtime.dbo.History where tagname='Bomba_9_2_ON' and
datetime = @StartDateTime
	insert into @SP(fecha,valor)
	select datetime,Round(value,2) from Runtime.dbo.History where
tagname='Presion_Descarga_N' and datetime = @StartDateTime
	set @StartDateTime=DateAdd(mi,30,@StartDateTime)
End
	select a.fecha as 'DateTime',a.valor as 'Estado
condensador',b.valor as 'Estado ventilador 1',c.valor as 'Estado
ventilador 2',d.valor as 'Variacion velocidad ventilador', e.valor as
'Estado bomba 1', f.valor as 'Estado bomba 2', g.valor as 'Set point'
from (((((@EstadoCond a inner join @EstadoVent1 b  on a.fecha =
b.fecha)inner join @EstadoVent2 c on a.fecha = c.fecha)inner join
@VarVent d on a.fecha = d.fecha)inner join @EstadoBomb1 e on a.fecha =
e.fecha)inner join @EstadoBomb2 f on a.fecha = f.fecha)inner join @SP g
on a.fecha = g.fecha
```

CONSUMO EN ELABORACION

```
ALTER PROCEDURE [dbo].[ConsumoElb]

	@StartDateTime datetime,
	@EndDateTime datetime
	AS
	declare  @ConsumoElb TABLE
	(
	fecha datetime,
	valor float(44)
	)

	Declare @c1 datetime
	Declare @c2 varchar (44)

	declare @anterior int
	set @anterior = 0

	declare @fecha datetime
		declare @final datetime
	set @fecha = @StartDateTime
	set @final = Dateadd(hh,1,@EndDateTime)


WHILE @StartDateTime <= @final
BEGIN

declare cursorCe CURSOR FOR
select datetime, cast(value as int)
from Runtime.dbo.History where tagname = 'Tot_8' and datetime
=Dateadd(hh,-1,@StartDateTime)

open cursorCe
fetch next from cursorCe
into @c1,@c2
```

r

```
while @@FETCH_STATUS = 0
begin
Insert @ConsumoElb values (@c1,@c2-@anterior)
set @anterior = @c2
fetch next from cursorCe
into @c1,@c2
end
close cursorCe
deallocate cursorCe

set @StartDateTime=DateAdd(hh,1,@StartDateTime)
END


select fecha as 'DateTime',valor as 'Consumo CO2 Elaboracion' from
@ConsumoElb where fecha>=@fecha
```

CONSUMO DE CO2 DE LA LINEA 1

```
ALTER PROCEDURE [dbo].[ConsumoL1CO2]

      @StartDateTime datetime,
      @EndDateTime datetime
      AS
      declare  @ConsumoL1 TABLE
      (
      fecha datetime,
      valor float(44)
      )

      Declare @c1 datetime
      Declare @c2 varchar (44)

      declare @anterior float
      set @anterior = 0

      declare @fecha datetime
            declare @final datetime
      set @fecha = @StartDateTime
      set @final = Dateadd(hh,1,@EndDateTime)


WHILE @StartDateTime <= @final
BEGIN

declare cursorCL1 CURSOR FOR
select datetime, cast(value as float)
from Runtime.dbo.History where tagname = 'Tot_6' and datetime
=Dateadd(hh,-1,@StartDateTime)

open cursorCL1
fetch next from cursorCL1
into @c1,@c2

while @@FETCH_STATUS = 0
begin
Insert @ConsumoL1 values (@c1,@c2-@anterior)
set @anterior = @c2
fetch next from cursorCL1
```

```
into @c1,@c2
end
close cursorCL1
deallocate cursorCL1

set @StartDateTime=DateAdd(hh,1,@StartDateTime)
END


select fecha as 'DateTime',valor as 'Consumo CO2 linea 1' from @ConsumoL1
where fecha>=@fecha
```

ENERGÍA EN COMPRESORES 7 8 Y 9

```
ALTER PROCEDURE [dbo].[EnergiaCompresor7]
@StartDateTime datetime,
@EndDateTime datetime
AS
create table #EC7
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40)

)

while (@StartDateTime <= @EndDateTime)
BEGIN
SET NOCOUNT ON;
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT
DateTime,((Compresor_7_Tension_In)/10)*3.55 as "Tension de entrada", '
SET @SQLString = @SQLString + ' Corriente_compr_NH3_7 as "Corriente de
motor", '
SET @SQLString = @SQLString + ' ROUND(KWH_Comp6,2) as "KWH Diarios" '

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), Compresor_7_Tension_In, '
SET @SQLString = @SQLString +  ' Corriente_compr_NH3_7,KWH_Comp6 '
SET @SQLString = @SQLString + 'FROM WideHistory  WHERE KWH_Comp6 is not
null   '
SET @SQLString = @SQLString + 'AND DateTime = '+ CHAR(39) +
CONVERT(varchar(70),@StartDateTime,21) + CHAR(39)+ '")'

insert into #EC7
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime,valor1 as 'Tension de entrada', valor2 as 'Corriente de
motor', valor3 as 'KWH Diarios'from #EC7
```

GLOBO DE CO2 Y BAJA PRESIÓN

```
ALTER  PROCEDURE [dbo].[Globo_BajaPresion]
@StartDateTime datetime,
@EndDateTime datetime

AS
create table #GloboBP
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN

SET NOCOUNT ON
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime,
ROUND(LT0410_0001_VALUE,2) as "Nivel CO2 globo", '
SET @SQLString = @SQLString + ' M0510_0501_hours as "Motor de booster", '
SET @SQLString = @SQLString + ' M0510_0704_hours as "Motor lavadora de
aereosol", '
SET @SQLString = @SQLString + ' P0510_0701_hours as "Motor bomba divisor"
'

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), '
SET @SQLString = @SQLString +  'LT0410_0001_VALUE,
M0510_0501_hours,M0510_0704_hours, P0510_0701_hours  '

SET @SQLString = @SQLString + 'FROM WideHistory WHERE LT0410_0001_VALUE
is not null   '
SET @SQLString = @SQLString + 'AND DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+ '") '

insert into #GloboBP
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime, valor1 as 'Nivel CO2 globo', valor2 as 'Motor de
booster', valor3 as 'Motor lavadora de aereosol', valor4 as 'Motor bomba
divisor' from #GloboBP
```

LAVADORA DE ALTA PRESIÓN Y ECONOMIZADOR

```
ALTER PROCEDURE [dbo].[LavAltaPresionEconomizador]

@StartDateTime datetime,
@EndDateTime datetime
AS
create table #LAPE
(
      DateTime datetime,
```

```
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime, '
SET @SQLString = @SQLString + ' ROUND(FT0510_1001_VALUE,2) as "Flux agua
lavadora", '
SET @SQLString = @SQLString + ' ROUND(TT0710_1001_VALUE,2) as "Temp
salida preenfriador", '
SET @SQLString = @SQLString + ' ROUND(CO2pressure,2) as "Presion
evaporacion preenfriador", '
SET @SQLString = @SQLString + ' P2610_2103_hours as "Horas operacion
bomba agua" '


SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), '
SET @SQLString = @SQLString +  'FT0510_1001_VALUE, TT0710_1001_VALUE, '
SET @SQLString = @SQLString +  'CO2pressure, P2610_2103_hours '


SET @SQLString = @SQLString + 'FROM WideHistory WHERE FT0510_1001_VALUE
is not null   '
SET @SQLString = @SQLString + 'AND DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+ '") '

insert into #LAPE
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime, valor1 as 'Flux agua lavadora', valor2 as 'Temp salida
preenfriador', valor3 as 'Presion evaporacion preenfriador', valor4 as
'Horas operacion bomba agua' from #LAPE
```

OPERACIÓN DE COMPRESOR 8 9 Y 10

```
ALTER PROCEDURE [dbo].[OperacionCompresor7]
@StartDateTime datetime,
 @EndDateTime datetime
AS
 declare  @ModoOp TABLE
      (
      fecha datetime,
      Modo_Operacion varchar(44)
      )

 declare @Estado TABLE
      (
      fecha datetime,
      Estado varchar(44)
      )
 declare @SetArranque TABLE
```

```
      (
      fecha datetime,
      Seteo_Arranque varchar(44)
      )
 declare @SetFunc TABLE
      (
      fecha datetime,
      Seteo_Funcion varchar(44)
      )
 declare @SetSalida TABLE
      (
      fecha datetime,
      Seteo_Salida varchar(44)
      )

WHILE @StartDateTime < @EndDateTime
BEGIN


      insert into @ModoOp(fecha,Modo_Operacion)
      Select datetime,case when value = '1' then 'manual' else
'automatico' end from Runtime.dbo.History where
tagname='Manual_Compresor_7' and datetime = @StartDateTime
      insert into @Estado(fecha,Estado)
      select datetime,case when value = '0' then 'apagado' else
'encendido'end from Runtime.dbo.History where
tagname='NH_11_01CF_CMP_C_6' and datetime = @StartDateTime
      insert into @SetArranque(fecha,Seteo_Arranque)
      select datetime,(value*0.1) from Runtime.dbo.History where
tagname='NH_11_01CI_PT01_C_6' and datetime = @StartDateTime
      insert into @SetFunc(fecha,Seteo_Funcion)
      select datetime,(value*0.1) from Runtime.dbo.History where
tagname='NH_11_01SPA_PT01_C6' and datetime = @StartDateTime
      insert into @SetSalida(fecha,Seteo_Salida)
      select datetime,(value*0.1) from Runtime.dbo.History where
tagname='NH_11_01CO_PT01_C6' and datetime = @StartDateTime
      set @StartDateTime=DateAdd(mi,30,@StartDateTime)
End
      select a.fecha as
'DateTime',a.Modo_Operacion,b.Estado,c.Seteo_Arranque, d.Seteo_Funcion,
e.Seteo_Salida from (((@ModoOp a inner join @Estado b  on a.fecha =
b.fecha)inner join @SetArranque c on a.fecha = c.fecha)inner join
@SetFunc d on a.fecha = d.fecha)inner join @SetSalida e on a.fecha =
e.fecha
```

PRESIÓN DE COMPRESOR 7 8 Y 9

```
ALTER PROCEDURE [dbo].[PresionCompresor7]
@StartDateTime datetime,
@EndDateTime datetime
AS
create table #PC7
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40)
```

```
)

while (@StartDateTime <= @EndDateTime)
BEGIN
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime, 0.1*NH_11_01PT01_C_6 as
"Presion de succion", '
SET @SQLString = @SQLString + ' NH_11_01PT05_C_7 as "Presion de
descarga", '
SET @SQLString = @SQLString + ' NH_11_01PT06_C_6 as "Presion diferencial
de aceite", '
SET @SQLString = @SQLString + ' NH_11_01PT07_C_6 as "Presion diferencial
filtro aceite" '

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21),NH_11_01PT01_C_6, '
SET @SQLString = @SQLString +  'NH_11_01PT05_C_7,
NH_11_01PT06_C_6,NH_11_01PT07_C_6 '
SET @SQLString = @SQLString + 'FROM WideHistory  WHERE NH_11_01PT07_C_6
is not null   '
SET @SQLString = @SQLString + 'AND DateTime = '+ CHAR(39) +
CONVERT(varchar(70),@StartDateTime,21) + CHAR(39)+ '")'

insert into #PC7
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime, valor1 as 'Presion de succion', valor2 as 'Presion de
descarga', valor3 as 'Presion diferencial de aceite',
 valor4 as 'Presion diferencial filtro aceite'from #PC7
```

PROCEDIMIENTO COMPLETO DE PRODUCCIÓN DE CO2

```
ALTER PROCEDURE [dbo].[Produccion_CO2]
      @StartDateTime datetime,
      @EndDateTime   datetime

AS

BEGIN

      create table #ProdCO2
      (
      fecha datetime,
      valor int
      )
      create table #ConsumoL1
      (
      fecha datetime,
      valor int
      )

      create table #ConsumoElb
      (
      fecha datetime,
      valor int
      )
```

```
       create table #StockEnHora
      (
      fecha datetime,
      valor int
      )
      create table #Stock
      (
      fecha datetime,
      valor int
      )

      insert into #ProdCO2 exec [dbo].[ProduccionCO2] @StartDateTime =
@StartDateTime, @EndDateTime = @EndDateTime
      insert into #ConsumoL1 exec [dbo].[ConsumoL1CO2] @StartDateTime =
@StartDateTime,   @EndDateTime = @EndDateTime
      insert into #ConsumoElb exec [dbo].[ConsumoElb] @StartDateTime =
@StartDateTime,   @EndDateTime = @EndDateTime
      insert into #StockEnHora exec [dbo].[StockEnHora] @StartDateTime =
@StartDateTime,   @EndDateTime = @EndDateTime
      insert into #Stock exec [dbo].[Stock] @StartDateTime =
@StartDateTime,   @EndDateTime = @EndDateTime
      select a.fecha as 'DateTime', a.valor as 'Produccion CO2', b.valor
as 'Consumo CO2 linea 1',SUM(c.valor-c.valor)as 'Consumo CO2 linea
2',d.valor as 'Consumo CO2 elaboracion', e.valor as 'Stock CO2 tanques'
from ((((#ProdCO2 a inner join #ConsumoL1 b on a.fecha=b.fecha)inner join
#StockEnHora c on a.fecha=c.fecha)inner join #ConsumoElb d on a.fecha =
d.fecha)inner join #Stock e on a.fecha = e.fecha) group by a.fecha,
a.valor, b.valor,c.valor,d.valor, e.valor

end
```

CONSUMO DE CO2

```
ALTER PROCEDURE [dbo].[ProduccionCO2]

      @StartDateTime datetime,
      @EndDateTime datetime
AS
 declare  @ProdCO2 TABLE
      (
      fecha datetime,
      valor int
      )

Declare @c1 datetime
Declare @c2 varchar (44)


declare @anterior int
set @anterior = 0

declare @fecha datetime
declare @final datetime
set @fecha = @StartDateTime
set @final = Dateadd(hh,1,@EndDateTime)


WHILE @StartDateTime <= @final
BEGIN
```

```
declare cursorProd CURSOR FOR
select datetime, cast(value  as int)
from Runtime.dbo.History where tagname = 'Tot_4' and datetime
=Dateadd(hh,-1,@StartDateTime)

open cursorProd
fetch next from cursorProd
into @c1,@c2

while @@FETCH_STATUS = 0
begin
Insert @ProdCO2 values (@c1,@c2-@anterior)
set @anterior = @c2
fetch next from cursorProd
into @c1,@c2
end
close cursorProd
deallocate cursorProd

set @StartDateTime=DateAdd(hh,1,@StartDateTime)
END


select fecha as 'DateTime',valor as 'Produccion CO2' from @ProdCO2 where
fecha>=@fecha
```

FILTRO DE ARENA

```
ALTER PROCEDURE [dbo].[SandFilter]
@StartDateTime datetime,
@EndDateTime datetime
AS
create table #SF
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40),
      valor5 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN
SET NOCOUNT ON;
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT a.DateTime,
Round(a.BR_Volumen_Cisterna,2) as "Volumen cisterna de agua de pozos",
Round(a.KF1F1_FLOW,2) as "Flujo en filtro KF1", '
SET @SQLString = @SQLString + ' Round(a.KF2F1_FLOW,2) as "Flujo en filtro
KF2", '
SET @SQLString = @SQLString + ' case when a.KF1F1_ACT_TP<b.KF1F1_ACT_TP
THEN "REGENERACION" ELSE "NORMAL" END as "ESTADO FILTRO KF1", '
SET @SQLString = @SQLString + ' case when a.KF2F1_ACT_TP<b.KF2F1_ACT_TP
THEN "REGENERACION" ELSE "NORMAL" END as "ESTADO FILTRO KF2" '

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
```

```
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21),BR_Volumen_Cisterna, '
SET @SQLString = @SQLString +  'KF1F1_FLOW,KF2F1_FLOW,
KF1F1_ACT_TP,KF2F1_ACT_TP '
SET @SQLString = @SQLString + 'FROM WideHistory  WHERE KF2F1_ACT_TP is
not null   '
SET @SQLString = @SQLString + 'AND DateTime = '+ CHAR(39) +
CONVERT(varchar(70),@StartDateTime,21) + CHAR(39)+ '")a, '

SET @SQLString = @SQLString + ' OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21),BR_Volumen_Cisterna,'
SET @SQLString = @SQLString +
'KF1F1_FLOW,KF2F1_FLOW,KF1F1_ACT_TP,KF2F1_ACT_TP '
SET @SQLString = @SQLString + 'FROM WideHistory  WHERE KF2F1_ACT_TP is
not null   '
SET @SQLString = @SQLString + 'AND DateTime = '+  CHAR(39)
+Convert(varchar(70),DateAdd(hh,-1,@StartDateTime),21) + CHAR(39)+ '")b
where Datediff(hh,b.DateTime,a.DateTime)=1 group by a.DateTime,'

SET @SQLString = @SQLString + ' a.BR_Volumen_Cisterna,a.KF1F1_FLOW,
a.KF2F1_FLOW, a.KF1F1_ACT_TP, b.KF1F1_ACT_TP, a.KF2F1_ACT_TP,
b.KF2F1_ACT_TP order by a.DateTime asc '

insert into #SF
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime,valor1 as 'Volumen cisterna de agua de pozos', valor2 as
'Flujo en filtro KF1', valor3 as 'Flujo en filtro KF2',
valor4 as 'ESTADO FILTRO KF1', valor5 as 'ESTADO FILTRO KF2'from #SF
```

SEPARADOR DE ESPUMA

```
ALTER PROCEDURE [dbo].[SeparadorEspuma]
      @StartDateTime datetime,
    @EndDateTime datetime
AS
create table #Separador
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN
SET NOCOUNT ON

DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime,
ROUND(QT0110_0501_VALUE,2) as "Pureza ingreso CO2", '
SET @SQLString = @SQLString + ' CASE WHEN XE0110_0003_STATE=1 then "SI"
ELSE CASE  '
SET @SQLString = @SQLString + ' WHEN XE0110_0103_STATE ="1" THEN "SI"
```

```
ELSE "NO" END END as "Presencia Espuma entrada", '
SET @SQLString = @SQLString + ' Round(O2_5,2) as "Pureza de salida CO2"'

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), '
SET @SQLString = @SQLString +  'QT0110_0501_VALUE, XE0110_0003_STATE,
XE0110_0103_STATE,O2_5 '

SET @SQLString = @SQLString + 'FROM WideHistory WHERE QT0110_0501_VALUE
is not null   '
SET @SQLString = @SQLString + 'AND DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+'") '
insert into #Separador
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END
select datetime,valor1 as 'Pureza ingreso CO2', valor2 as 'Presencia
Espuma entrada', valor3 as 'Pureza de salida CO2' from #Separador
```

SISTEMA DE PRESIÓN CONSTANTE

```
ALTER PROCEDURE [dbo].[SistemaPresionConstante]
@StartDateTime datetime,
@EndDateTime datetime

AS
create table #SPC
(
      DateTime datetime,
      valor1 varchar (40),
      valor2 varchar (40),
      valor3 varchar (40),
      valor4 varchar (40),
      valor5 varchar (40),
      valor6 varchar (40),
      valor7 varchar (40),
      valor8 varchar (40),
      valor9 varchar (40),
      valor10 varchar (40),
      valor11 varchar (40),
      valor12 varchar (40),
      valor13 varchar (40),
      valor14 varchar (40),
      valor15 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN

SET NOCOUNT ON
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF '
SET @SQLString = @SQLString + 'SELECT DateTime, Round(NIVEL_TQ_2,2) as
"Nivel tanque desalcalinizada", '
SET @SQLString = @SQLString + ' Round(PRES_SAL_TQ_2,2) as "Presion tanque
desalcalinizada", '
SET @SQLString = @SQLString + ' Round(VEL_AUTO_DRIVE3_TQ2,2) as
```

```
"Porcentaje bomba 3", '
SET @SQLString = @SQLString + ' Round(VEL_AUTO_DRIVE4_TQ2,2) as
"Porcentaje bomba 4", '
SET @SQLString = @SQLString + ' CASE WHEN PRIORIDAD_MANUAL_TQ2 = "1" THEN
"Manual" ELSE "Automatico" END AS "Prioridad bombas desalcalinizador", '

SET @SQLString = @SQLString + ' Round(NIVEL_TQ_1,2) as "Nivel tanque agua
blanda", '
SET @SQLString = @SQLString + ' Round(PRES_SAL_TQ_1,2) as "Presion tanque
agua blanda", '
SET @SQLString = @SQLString + ' Round(VEL_AUTO_DRIVE3_TQ1,2) as
"Porcentaje bomba 1", '
SET @SQLString = @SQLString + ' Round(VEL_AUTO_DRIVE4_TQ1,2) as
"Porcentaje bomba 2", '
SET @SQLString = @SQLString + ' CASE WHEN PRIORIDAD_MANUAL_TQ1 = "1" THEN
"Manual" ELSE "Automatico" END AS "Prioridad bombas agua blanda", '

SET @SQLString = @SQLString + ' Round(NIVEL_TQ_3,2) as "Nivel tanque
general", '
SET @SQLString = @SQLString + ' Round(PRES_SAL_TQ_3,2) as "Presion tanque
general", '
SET @SQLString = @SQLString + ' Round(VEL_AUTO_DRIVE3_TQ3,2) as
"Porcentaje bomba 5", '
SET @SQLString = @SQLString + ' Round(VEL_AUTO_DRIVE4_TQ3,2) as
"Porcentaje bomba 6", '
SET @SQLString = @SQLString + ' CASE WHEN PRIORIDAD_MANUAL_TQ3 = "1" THEN
"Manual" ELSE "Automatico" END AS "Prioridad bombas general" '

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), '
SET @SQLString = @SQLString +  'NIVEL_TQ_2,
PRES_SAL_TQ_2,VEL_AUTO_DRIVE3_TQ2,
VEL_AUTO_DRIVE4_TQ2,PRIORIDAD_MANUAL_TQ2,  '
SET @SQLString = @SQLString +  'NIVEL_TQ_1,
PRES_SAL_TQ_1,VEL_AUTO_DRIVE3_TQ1,
VEL_AUTO_DRIVE4_TQ1,PRIORIDAD_MANUAL_TQ1,  '
SET @SQLString = @SQLString +  'NIVEL_TQ_3,
PRES_SAL_TQ_3,VEL_AUTO_DRIVE3_TQ3,
VEL_AUTO_DRIVE4_TQ3,PRIORIDAD_MANUAL_TQ3  '

SET @SQLString = @SQLString + 'FROM WideHistory WHERE NIVEL_TQ_2 is not
null   '
SET @SQLString = @SQLString + 'AND DateTime = ' + CHAR(39) +
CAST(@StartDateTime AS varchar(50)) + CHAR(39)+ '") '

insert into #SPC
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END
select datetime,valor1 as 'Nivel tanque desalcalinizada', valor2 as
'Presion tanque desalcalinizada', valor3 as 'Porcentaje bomba 3', valor4
as 'Porcentaje bomba 4', valor5 as 'Prioridad bombas desalcalinizador',
valor6 as 'Nivel tanque agua blanda', valor7 as 'Presion tanque agua
blanda', valor8 as 'Porcentaje bomba 1', valor9 as 'Porcentaje bomba 2',
valor10 as 'Prioridad bombas agua blanda', valor11 as 'Nivel tanque
general', valor12 'Presion tanque general', valor13 as 'Porcentaje bomba
5',
valor14 as 'Porcentaje bomba 6', valor15 as 'Prioridad bombas general'
 from #SPC
```

STOCK DE CO2

```
ALTER PROCEDURE [dbo].[Stock]

	@StartDateTime datetime,
	@EndDateTime datetime
	AS
	declare  @Stock TABLE
	(
	fecha datetime,
	valor int
	)

	Declare @c1 datetime
	Declare @c2 int

	declare @anterior int
	set @anterior = 0

	declare @fecha datetime
	set @fecha = @StartDateTime


WHILE @StartDateTime <= @EndDateTime
BEGIN

insert into @Stock
select datetime, Cast(Round(value,0) as int)
from Runtime.dbo.History where tagname = 'Tanks_TotalTanks' and datetime
= @StartDateTime

set @StartDateTime=DateAdd(hh,1,@StartDateTime)
END


select fecha as 'DateTime', valor  from @Stock where fecha>=@fecha
```

TEMPERATURA DE COMPRESOR 7 8 Y 9

```
ALTER PROCEDURE [dbo].[TemperaturaCompresor7]
@StartDateTime datetime,
@EndDateTime datetime
AS
create table #TC7
(
	DateTime datetime,
	valor1 varchar (40),
	valor2 varchar (40),
	valor3 varchar (40),
	valor4 varchar (40),
	valor5 varchar (40),
	valor6 varchar (40)
)

while (@StartDateTime <= @EndDateTime)
BEGIN
```

dd

```
DECLARE @SQLString varchar(8000)
SET @SQLString = 'SET QUOTED_IDENTIFIER OFF'
SET @SQLString = @SQLString + ' SELECT DateTime, 0.01*NH_11_01TT01 as
"Temperatura de succion", '
SET @SQLString = @SQLString + ' 0.1*NH_11_01TT02 as "Temperatura de
descarga", '
SET @SQLString = @SQLString + ' 0.1*NH_11_01TT03 as "Temperatura de
aceite", '
SET @SQLString = @SQLString + ' 0.1*NH_11_01TT04 as "Temperatura de
separador aceite", '
SET @SQLString = @SQLString + ' 0.1*NH_11_01TT06 as "(T)
Sobrecalentamiento succion", '
SET @SQLString = @SQLString + ' 0.1*NH_11_01TT07_C_6 as "(T)
Sobrecalentamiento descarga" '

SET @SQLString = @SQLString + ' FROM OPENQUERY(INSQL, '
SET @SQLString = @SQLString + + '"SELECT DateTime = convert(nvarchar,
DateTime, 21), NH_11_01TT01, '
SET @SQLString = @SQLString +  ' NH_11_01TT02,
NH_11_01TT03,NH_11_01TT03,NH_11_01TT04,NH_11_01TT06,NH_11_01TT07_C_6 '
SET @SQLString = @SQLString + 'FROM WideHistory  WHERE NH_11_01TT07_C_6
is not null   '
SET @SQLString = @SQLString + 'AND DateTime = '+ CHAR(39) +
CONVERT(varchar(70),@StartDateTime,21) + CHAR(39)+ '")'

insert into #TC7
EXEC (@SQLString)
set @StartDateTime = Dateadd(mi,30,@StartDateTime)
END

select datetime, valor1 as 'Temperatura de succion', valor2 as
'Temperatura de descarga', valor3 as 'Temperatura de aceite',
 valor4 as 'Temperatura de separador aceite', valor5 as '(T)
Sobrecalentamiento succion',
valor6 as '(T) Sobrecalentamiento descarga' from #TC7
```