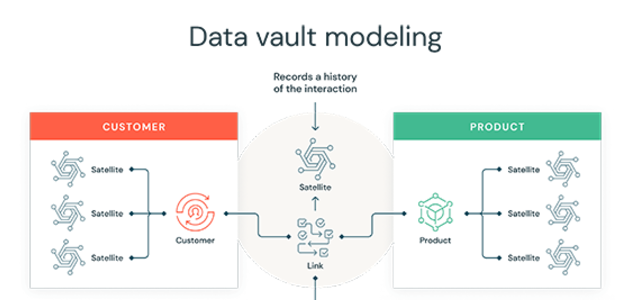
**Documento Funcional: Diseño de un Data Mart para Análisis de Ventas**

**Introducción**

El modelo Data Vault para el sistema de ventas está diseñado para proporcionar una estructura robusta y flexible que permite conectar a los clientes con los productos que compran. Este enfoque garantiza que todos los datos de ventas se gestionen de manera eficiente, facilitando tanto la integración de nuevos datos como el mantenimiento de un historial detallado para análisis y auditorías. A continuación, se detalla la estructura de las tablas Hub, Satélite y Link, cada una desempeñando un papel crucial en el sistema.

**Descripción de las Tablas**

En este esquema de Data Vault, los hubs sirven para identificar de manera única a las entidades principales: productos y clientes. El link modela las transacciones de ventas, conectando los clientes con los productos que compran. Los satélites proporcionan detalles adicionales y permiten el seguimiento de cambios históricos en los datos de productos y clientes. Este modelo ofrece una estructura flexible y escalable para gestionar datos de ventas, facilitando la integración de nuevos datos y el mantenimiento de un historial detallado para análisis y auditorías.



1. **Tablas Hub:**
   * **hubcustomer**:
     + **Descripción**: Este hub contiene las claves únicas de negocio para los clientes. Es el punto central para identificar a cada cliente.
     + **Columnas**:

* customer\_id: Clave sustituta de tipo entero que se incrementa automáticamente y actúa como clave primaria de la tabla.
* hashkeycustomer: Clave hash única de 32 caracteres utilizada para identificar de manera única a cada cliente.
* dt\_load\_date: Marca temporal que indica la fecha y hora en que el registro fue cargado en la tabla.
* recordsource: Cadena de texto de hasta 50 caracteres que especifica la fuente de donde proviene el registro.
  + **hubproduct**:
    - **Descripción**: Este hub almacena las claves de negocio únicas para los productos. Su función es identificar de manera única cada producto en el sistema.
    - **Columnas**:
* product\_id: Clave sustituta de tipo entero que se incrementa automáticamente y sirve como clave primaria de la tabla.
* hashkeyproduct: Clave hash única de 32 caracteres para identificar de manera única cada producto.
* dt\_load\_date: Marca temporal que señala la fecha y hora de carga del registro.
* recordsource: Cadena de texto de hasta 50 caracteres que identifica la fuente del registro.

1. **Tablas Satélite:** 
   * **satcustomer**:
     + **Descripción**: Este satélite contiene detalles sobre los clientes, como su nombre, dirección y otros datos de contacto. Permite mantener un historial de cambios en la información del cliente.
     + **Columnas**:
       - customer\_id: Clave foránea de 32 caracteres que referencia al hubcustomer.
       - dt\_load\_date: Marca temporal que indica la fecha y hora de carga del registro.
       - dt\_start\_date: Fecha y hora en que el registro se vuelve efectivo.
       - dt\_end\_date: Fecha y hora en que el registro deja de ser efectivo.
       - current\_record: Booleano que indica si el registro es el actual.
       - first\_name, last\_name, full\_name: Campos de texto para el nombre del cliente.
       - social\_scrty\_no: Número de seguridad social del cliente.
       - cd\_natural\_legal y ds\_natural\_legal: Códigos y descripciones para distinguir entre personas naturales y jurídicas.
       - dt\_birth: Fecha de nacimiento del cliente.
       - address: Dirección del cliente, permitiendo hasta 255 caracteres.
       - phone\_number: Número de teléfono del cliente, permitiendo hasta 255 caracteres.
       - recordsource: Fuente del registro.
       - PRIMARY KEY: Compuesta por customer\_id y dt\_start\_date.
   * **satproduct**:
     + **Descripción**: Este satélite almacena información descriptiva y de contexto sobre los productos. Captura detalles que pueden cambiar con el tiempo, como el precio o la descripción.
     + **Columnas**:
     + product\_id: Clave de negocio o compuesta que referencia al hubproduct.
     + dt\_load\_date: Marca temporal que indica la fecha y hora de carga del registro.
     + dt\_start\_date: Fecha y hora en que el registro se vuelve efectivo.
     + dt\_end\_date: Fecha y hora en que el registro deja de ser efectivo.
     + current\_record: Booleano que indica si el registro es el actual.
     + cd\_product y ds\_product: Código y descripción del producto.
     + cd\_package\_group: Código numérico del grupo de paquetes.
     + ds\_package\_group: Descripción del grupo de paquetes, permitiendo hasta 500 caracteres.
     + recordsource: Fuente del registro.
     + PRIMARY KEY: Compuesta por product\_id y dt\_start\_date.
2. **Tabla Link**
   * **linkproductocustomer**:
     + **Descripción**: Este link modela la relación entre los clientes y los productos a través de transacciones de venta. Conecta los hubs de clientes y productos, representando las ventas realizadas.
     + **Columnas**:
     + link\_key: Clave surrogada opcional que se incrementa automáticamente.
     + customer\_id: Clave foránea de 32 caracteres que referencia al hubcustomer.
     + product\_id: Clave foránea de 32 caracteres que referencia al hubproduct.
     + dt\_load\_date: Marca temporal que indica la fecha y hora de carga del registro.
     + dt\_transaction\_date: Fecha y hora de la transacción.
     + recordsource: Cadena de texto de hasta 50 caracteres que especifica la fuente del registro.
     + PRIMARY KEY: Compuesta por customer\_id, product\_id y dt\_transaction\_date.
     + FOREIGN KEY: customer\_id referencia a hubcustomer y product\_id referencia a hubproduct.

**Script Incremental**

Este script está diseñado para realizar una carga de datos incremental desde una tabla fuente a una tabla de destino en una base de datos MySQL. A continuación se ofrece una descripción funcional de cada parte del script:

1. **Configuración de Logging**:
   * Se configura el sistema de logging para registrar eventos del script en un archivo llamado data\_load.log. Los eventos registrados incluyen información sobre el momento en que ocurrieron, el nivel de severidad y un mensaje descriptivo.
2. **Funciones Auxiliares**:
   * fetch\_source\_data: Esta función recibe un cursor de base de datos y una fecha de última carga (last\_load\_date). Ejecuta una consulta SQL para obtener registros nuevos o modificados desde la tabla source\_hubcustomer que tienen una fecha de carga posterior a last\_load\_date. Los registros se devuelven como una lista de tuplas.
   * insert\_staging\_data: Inserta un único registro en la tabla de staging staging\_hubcustomer usando los datos proporcionados en la tupla data.
3. **Función Principal load\_data**:
   * **Lectura de Configuración**: Se lee la configuración de la base de datos desde un archivo db\_config.ini usando configparser. Si hay un error en la configuración, se registra y el proceso se detiene.
   * **Conexión a la Base de Datos**: Se establece una conexión con la base de datos MySQL usando los parámetros leídos. Si la conexión falla, se registra el error y el proceso se detiene.
   * **Transacción y Control de Carga**:
     + Se inicia una transacción.
     + Se asegura que la tabla de control load\_control tenga un registro para la tabla hubcustomer con su última fecha de carga.
     + Se obtiene la última fecha de carga para determinar qué registros nuevos o modificados deben ser extraídos.
   * **Extracción de Datos**: Se llama a fetch\_source\_data para obtener los registros relevantes desde la tabla fuente.
   * **Carga Incremental**:
     + Se insertan los registros extraídos en la tabla de staging staging\_hubcustomer.
     + Se realiza una carga incremental en la tabla hubcustomer, insertando solo aquellos registros que no existen ya en la tabla de destino.
   * **Actualización de la Tabla de Control**: Se actualiza la tabla load\_control con la nueva última fecha de carga basada en los datos insertados.
   * **Confirmación de la Transacción**: Si todo el proceso se completa sin errores, la transacción se confirma.
   * **Limpieza**: Se vacía la tabla de staging para prepararla para futuras cargas.
4. **Manejo de Errores**:
   * Si ocurre algún error durante la ejecución de las consultas SQL, la transacción se revierte y se registra el error.
   * Si ocurre un error inesperado, también se revierte la transacción y se registra el error.
5. **Cierre de Conexión**:
   * Finalmente, se cierra el cursor y la conexión a la base de datos, registrando el cierre en el log.

El script está diseñado para ser ejecutado directamente, iniciando el proceso de carga de datos incremental cuando se ejecuta como un programa principal.

**Script de Particionado Mensual**

Este script está diseñado para automatizar el particionado mensual de tablas en una base de datos MySQL. Utiliza particionamiento basado en rangos de fechas para optimizar el rendimiento y la gestión de grandes volúmenes de datos. A continuación, se detallan los pasos técnicos que realiza el script:

1. **Importación de Módulos Necesarios**:
   * mysql.connector: Proporciona la funcionalidad necesaria para conectarse y operar con bases de datos MySQL desde Python.
   * datetime y timedelta del módulo datetime: Se utilizan para realizar cálculos de fechas, cruciales para determinar los límites de partición.
2. **Configuración de la Conexión a la Base de Datos**:
   * Se define un diccionario database\_config que incluye los parámetros de conexión: user, password, host, y database.
   * Este diccionario es utilizado para establecer una conexión segura a la base de datos MySQL.
3. **Definición de la Función add\_monthly\_partition**:
   * **Conexión y Manejo de Errores**:
     + Se conecta a la base de datos usando mysql.connector.connect(\*\*database\_config). La conexión es gestionada mediante un bloque try-except-finally para asegurar el cierre adecuado de la conexión incluso si ocurre un error.
   * **Cálculo de Límites de Partición**:
     + Se determina el primer día del mes siguiente utilizando operaciones de fecha: (today.replace(day=1) + timedelta(days=32)).replace(day=1).
     + Se calcula el último día del mes siguiente: (first\_day\_next\_month + timedelta(days=31)).replace(day=1) - timedelta(days=1).
   * **Generación del Nombre de Partición**:
     + Se construye un nombre de partición en el formato pYYYYMM utilizando strftime para extraer el año y mes.
   * **Registro de Eventos en event\_logs**:
     + Se inserta un registro en la tabla event\_logs para documentar el inicio del proceso de creación de partición.
   * **Construcción y Ejecución de la Sentencia SQL**:
     + Se genera una sentencia SQL ALTER TABLE para añadir una partición con VALUES LESS THAN, utilizando TO\_DAYS para definir el límite superior de la partición.
     + Se ejecuta la sentencia SQL a través del cursor y se registra la ejecución en event\_logs.
   * **Confirmación de Transacciones**:
     + Se realiza un commit para confirmar los cambios en la base de datos.
   * **Cierre de la Conexión**:
     + Se cierra el cursor y la conexión utilizando cursor.close() y connection.close() en el bloque finally.
4. **Iteración sobre Tablas Específicas**:
   * Se define una lista de tablas (tables) que requieren particionamiento.
   * Se itera sobre cada tabla y se invoca add\_monthly\_partition para gestionar el particionado de manera individual.
5. **Automatización del Script**:
   * Se sugiere la implementación de un cron job o un sistema de planificación de tareas para ejecutar el script mensualmente, asegurando la actualización continua de las particiones sin intervención manual.

Este script es fundamental para bases de datos MySQL que manejan datos en crecimiento constante, permitiendo un acceso más eficiente y una administración simplificada de los datos almacenados.

Top of Form