

Minería de Datos II

Clase 2 - ETL

- Gracias a los procesos ETL es posible que cualquier organización:

- Mueva datos desde una o múltiples fuentes.
- Reformatee esos datos y los limpie, cuando sea necesario.
- Los cargue en otro lugar como una base de datos unificada.
- Una vez alojados en un destino, esos datos se analicen.
- O, cuando ya están cargados en su ubicación definitiva, se empleen en otro sistema operacional, para apoyar un proceso de negocio.

- Fase de Extracción:

- Extraer los datos desde los sistemas de origen.
- Analizar los datos extraídos obteniendo un chequeo.
- Interpretar este chequeo para verificar que los datos extraídos cumplen la pauta o estructura que se esperaba. Si no fuese así, los datos deberían ser rechazados.
- Convertir los datos a un formato preparado para iniciar el proceso de transformación.

- Fase de Transformación:

La fase de transformación de un proceso ETL aplica una serie de reglas de negocio o funciones, sobre los datos extraídos para convertirlos en datos que serán cargados. Estas directrices pueden ser declarativas, pueden basarse en excepciones o restricciones pero, para potenciar su pragmatismo y eficacia, hay que asegurarse de que sean:

- Declarativas.
- Independientes.
- Claras.
- Inteligibles.
- Con una finalidad útil para el negocio.

- Optimización de Consultas

La optimización de consultas es de gran importancia para el rendimiento de una base de datos relacional, especialmente para la ejecución de sentencias SQL complejas. Un optimizador de consultas decide los mejores métodos para implementar cada consulta.

El optimizador de consultas selecciona, por ejemplo, si desea o no usar índices para una consulta determinada y qué métodos de unión usar al unir varias tablas. Estas decisiones tienen un tremendo efecto en el rendimiento de SQL, y la optimización de consultas es una tecnología clave para cada aplicación, desde sistemas operativos hasta sistemas de almacenamiento de datos y sistemas analíticos hasta sistemas de gestión de contenido.

Las principales consideraciones para optimizar las consultas son:

- Para que una consulta lenta sea más rápida, lo primero a tener en cuenta es que las tablas posean índices. Configurar índices en las columnas utilizadas en la cláusula, acelera la evaluación, el filtrado y la recuperación final de los resultados.
- Minimizar el número de análisis de tablas completas en sus consultas, especialmente para tablas grandes. Consultar cuando sea posible a partir de criterios específicos.
- La investigación de la documentación del SGBD para poder utilizar funciones y procedimientos que performen de mejor que otras opciones. Por lo general `LEFT(Nombre) = 'Rod'` es menos performante que `LIKE 'Rod%'`

- Estadísticas de consultas:

La ficha Resultados del editor SQL posee Estadísticas de consulta que utiliza datos del esquema de rendimiento para recopilar métricas clave para la consulta ejecutada, como la sincronización, las tablas temporales, los índices, las combinaciones y mucho más. Las estadísticas en MySQL se pueden consultar en el margen derecho de la pantalla de resultados, mediante la opción "Query Stats".

Query Statistics	
Timing (as measured at client side): Execution time: 0:00:0.01500000	Joins per Type: Full table scans (Select_scan): 1 Joins using table scans (Select_full_join): 0 Joins using range search (Select_full_range_join): 0 Joins with range checks (Select_range_check): 0 Joins using range (Select_range): 0
Timing (as measured by the server): Execution time: 0:00:0.01277070 Table lock wait time: 0:00:0.00002200	Sorting: Sorted rows (Sort_rows): 0 Sort merge passes (Sort_merge_passes): 0 Sorts with ranges (Sort_range): 0 Sorts with table scans (Sort_scan): 0
Errors: Had Errors: NO Warnings: 0	Index Usage: No Index used
Rows Processed: Rows affected: 0 Rows sent to client: 5 Rows examined: 8	Other Info: Event Id: 67 Thread Id: 69
Temporary Tables: Temporary disk tables created: 0 Temporary tables created: 0	

- Plan de explicación visual:

La característica de explicación visual genera y muestra una representación visual de la instrucción MySQL EXPLAIN mediante el uso de información extendida disponible en el formato JSON extendido. MySQL Workbench proporciona todos los formatos para las consultas ejecutadas, incluido el JSON extendido sin procesar, el formato tradicional y el plan de consulta visual.

Para ver un plan de ejecución visual explicado, ejecute la consulta desde el editor de SQL y, a continuación, seleccione Plan de ejecución en la ficha Resultados de la consulta. El plan de ejecución tiene como valor predeterminado, pero también incluye una vista similar a la que se ve al ejecutar EXPLAIN en el cliente MySQL. Para obtener información acerca de cómo MySQL ejecuta instrucciones, consulte Optimización de consultas con EXPLAIN.

El orden de ejecución en un diagrama de explicación visual es de abajo a arriba y de izquierda a derecha. Los ejemplos de diagramas que siguen proporcionan una visión

general de las convenciones gráficas, textuales e informativas utilizadas para representar aspectos de los planes de explicación visual. Para obtener información específica, consulte:

- Convenciones gráficas.
- Convenciones textuales e informativas.

El diagrama de explicación visual de la primera figura muestra una representación visual de la siguiente consulta.

La ejecución de consultas y sentencias en un plan de ejecución de consultas gráfico es mostrada por íconos. Cada ícono tiene un color específico y representa una acción específica. La presentación gráfica provee un entendimiento rápido de las características y estructura básicas del plan, por lo tanto, es útil para el análisis del desempeño. También provee suficiente información para un análisis más profundo sobre el proceso de ejecución.

- SQL – Índices

Un índice SQL es una tabla de búsqueda rápida para poder encontrar los registros que los usuarios necesitan buscar con mayor frecuencia. Ya que un índice es pequeño, rápido y optimizado para búsquedas rápidas. Además, son muy útiles para conectar las tablas relacionales y la búsqueda de tablas grandes.

Los índices de SQL son la principal herramienta de rendimiento, por lo que generalmente se aplican si una base de datos se incrementa. El motor SQL reconoce varios tipos de índices, pero uno de los más comunes es el índice agrupado. Esta clase de índice se crea automáticamente con una clave principal.

En el momento en el que se ejecuta la consulta, el motor SQL creará automáticamente un índice agrupado en la columna especificada.

Podemos ver al comprobar mediante Workbench que la tabla "carrera" se encuentra indexada.

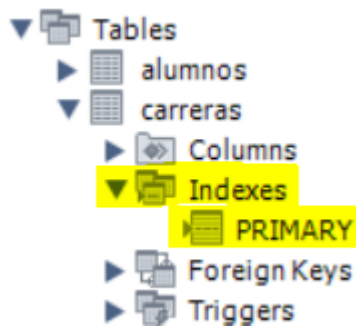
Los índices de las tablas ayudan a indexar el contenido de diversas columnas para facilitar la búsquedas de contenido de cuando se ejecutan consultas sobre esas tablas.

De ahí que la creación de índices optimiza el rendimiento de las consultas y a su vez el de la BBDD, pueden agregarse índices en caso de tablas puentes donde no se ha solucionado el problema de indexación aplicando claves concatenadas.

En MySQL puede utilizarse CREATE INDEX para crear o añadir índices en las tablas de una base de datos.

```
```sql
CREATE TABLE carrera (
 idCarrera INT NOT NULL AUTO_INCREMENT,
 nombre VARCHAR (20) NOT NULL,
 PRIMARY KEY (idCarrera) --Aquí al crear una PK, SQL además crea un índice
agrupado.
```
```

Con el código superior CREATE INDEX estaríamos creando uno o varios índices ordinarios en una tabla existente.

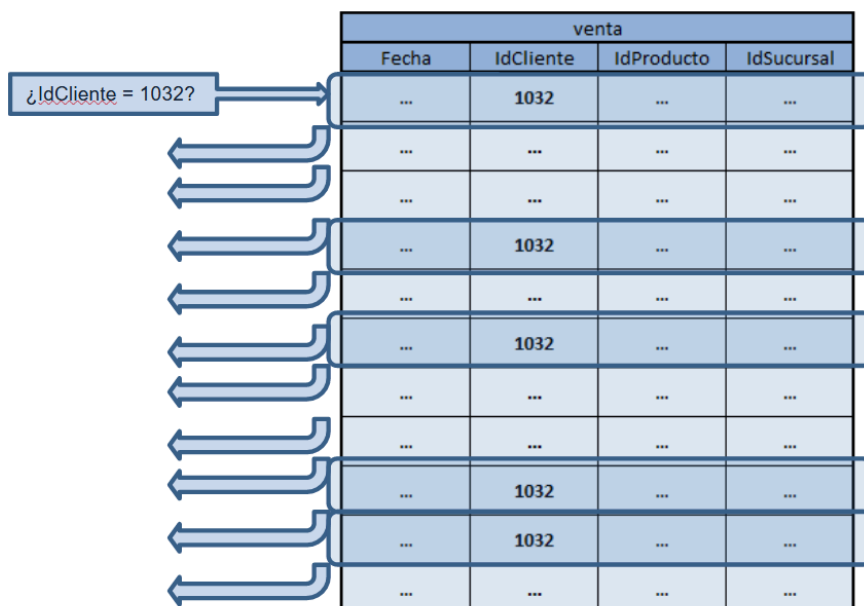


Cuando hacemos una consulta sobre un campo sin índices, el motor busca la condición de forma exhaustiva registro a registro para quedarse con aquellos que la cumplan. Sí por el contrario, contamos con una tabla de índices, esa búsqueda se de forma más óptima, ya que se tienen conocimiento de la ubicación de cada registro.

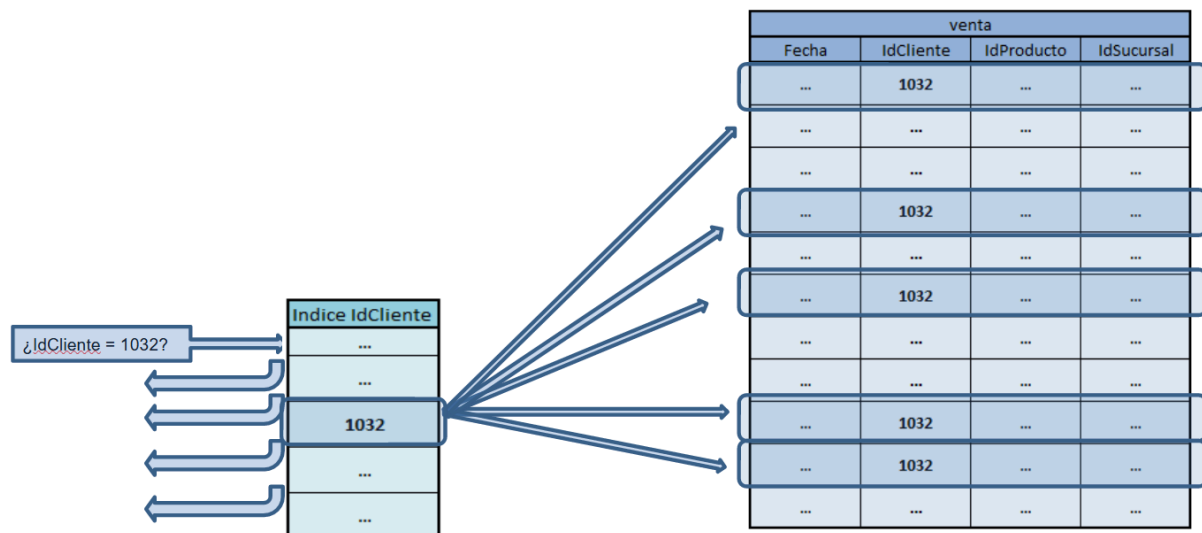
```
1 CREATE INDEX nombre_indice ON nombrede_tabla(columna [columna2...]);
```

Cuando hacemos una consulta sobre un campo sin índices, el motor busca la condición de forma exhaustiva registro a registro para quedarse con aquellos que la cumplan

```
```sql
SELECT * FROM venta WHERE IdCliente = 1032;
```
```



Sí por el contrario, contamos con una tabla de índices, esa búsqueda se realiza de forma más óptima, ya que se tienen conocimiento de la ubicación de cada registro.



Podemos tener los siguientes tipos de índices en una tabla de MySQL:

- Únicos.
- Primarios.
- Ordinarios.
- De texto completo.
- Parte de campos o columnas.

También se pueden eliminar índices mediante la sentencia DROP INDEX, siempre teniendo presente que la utilización de la sentencia DROP puede llevar a consecuencias indeseadas.

```
1 DROP INDEX 'segundo_apellido' ON clientes
2 DROP INDEX 'PRIMARY' ON clientes;
```

- Funciones envueltas alrededor la cláusula WHERE:

Un tema en la optimización es un enfoque constante en la cláusula WHERE. ¡Cuanto más rápido podamos dividir nuestro conjunto de datos a solo las filas que necesitamos, más eficiente será la ejecución de la consulta!

Al evaluar una cláusula WHERE, cualquier expresión involucrada debe resolverse antes de devolver nuestros datos. Si una columna contiene funciones a su alrededor, como DATEPART, SUBSTRING o CONVERT, estas funciones también deberán resolverse. Si la función debe evaluarse antes de la ejecución para determinar un conjunto de resultados, deberá analizarse la totalidad del conjunto de datos para completar esa evaluación.

```
```sql
```

```
#Utilizamos IN para crear un conjunto en la segmentación
```

```
SELECT *
```

```
FROM instructores
```

```
WHERE idInstructor IN (1,2,3,4,5)
```

```
#Utilizamos OR para crear el mismo conjuntos
SELECT *
FROM instructores
WHERE idInstructor = 1 OR idInstructor = 2 OR idInstructor = 3
OR idInstructor = 4 OR idInstructor = 5
``
```

En este caso, es recomendable la opción con el IN.

#### - Conclusión

Es una buena práctica indexar nuestras tablas, tanto mediante PK como de otros tipos de índices. Cuando la utilización de PK no resulte aplicable siempre se debe recurrir a otra alternativa de indexación.

La optimización del rendimiento de una base de datos no solo se limita a crear consultas con buen desempeño, este proceso comienza con la creación de una estructura eficiente que represente adecuadamente el modelo de negocios. La normalización excesiva o inadecuada puede repercutir en este punto. Por ejemplo:

- ¿Es necesario que nuestra BD tenga tablas de provincias o localidades?. ¿Es relevante para el negocio?.

Cuando el volumen de datos se vuelve considerable y las organizaciones se ven obligadas a implementar repositorios analíticos centralizados, el modelo de datos diseñado es fundamental para un buen rendimiento. Los modelos de datos analíticos presentados tienen como objetivo una rápida recuperación de un gran volumen de datos, sacrificando normalización.

Si todo lo anterior se aborda de manera adecuada, será más sencillo para quien realice las consultas encontrar la manera de escribirlas que permita el mejor desempeño.

Material complementario:

\* Optimización:

<https://www.adictosaltrabajo.com/2016/10/24/optimizacion-de-consultas-en-mysql/>

\* Índices IBM:

<https://www.ibm.com/docs/es/mam/7.6.0.8?topic=databases-database-indexing>

\* Índices MySQL: <https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html>

\* Modelos de Datos: <https://www.ibm.com/docs/es/ida/9.1.2?topic=schemas-snowflake>

#### - Carga de Datos

Ya comenzamos a descubrir como generar un proceso de ETL sencillo, una parte importante de este proceso se produce luego de que se encuentra el producción. Este proceso posterior se basa en la carga de datos la cual puede ser Full o Delta, en el primer caso se vuelcan todos los datos y en el segundo solamente los datos no almacenados en el Mart previamente.

Como abordaje preliminar podrías preguntarte, ¿es eficiente cargar todos los registros históricos en cada ejecución del proceso?. Objetivamente es mejor una carga de datos Delta, es decir que solo vuelque las diferencias con respecto a la última carga, sin embargo depende de diferentes factores.

- Orígenes de datos: Hay veces que la naturaleza del origen de datos imposibilita determinar cuáles han sido las últimas modificaciones con respecto a la última carga.
- Volumen de datos: Si el volumen de datos es pequeño es posible que no se precise de un Mart, por consiguiente la carga sería Full. Si el volumen es muy grande el tiempo y costo de procesamiento podría ser muy elevado para una carga Full.
- Velocidad de respuesta: En entornos donde se requiera un cierto tiempo de respuesta una carga Full puede no ser viable, sin embargo, en entornos donde la velocidad sea irrelevante, por ejemplo un Dashboard mensual una carga Full sería válida y más fácil de desarrollar.
- Niveles de Servicio: En algunos entornos en la nube, se paga por lo que se consume (transacciones, uso de memoria, procesador, etc) con lo cual una gran carga podría comprometer el proceso si existen time outs, es por eso que una carga Full puede requerir incrementar el costo.

#### - Extracciones basadas en fechas

Una extracción basada en fechas, normalmente selecciona todas las filas donde se crearon o modificaron los campos de fecha, lo que significa muchas veces "todos los registros de ayer". Cargar registros basados puramente en el tiempo es un error común cometido por Desarrolladores ETL sin experiencia. Este proceso es terriblemente poco fiable. La selección puede cargar filas duplicadas y requerir intervención manual y limpieza de datos si el proceso falla por cualquier razón.

Mientras tanto, si el proceso de carga nocturna no se ejecuta y se salta un día, hay un riesgo de que los datos perdidos nunca lleguen al almacén de datos.

Para utilizar este proceso se debe tener un campo fecha en la tabla de origen y de hechos, mediante una consulta SQL obtenemos esa fecha y la establecemos como un filtro:

```
```sql
--¿Cómo obtendrías la última fecha de carga de la tabla fact_inicial?
INSERT INTO fact_inicial (IdFecha, Fecha, IdSucursal, IdProducto, IdProductoFecha,
IdSucursalFecha, IdProductoSucursalFecha)
SELECT      Campos a cargar
FROM venta v JOIN calendario c
            ON (v.Fecha = c.fecha)
WHERE v.Fecha > ( Ultima fecha de carga de la tabla fact_inicial)
--WHERE v.Fecha BETWEEN ( Ultima fecha de carga de la tabla fact_inicial) AND (Fecha
más reciente)
```
```

## - Comparación de diferencias completas

Una comparación de diferencias completa, mantiene una instantánea completa de los datos de ayer y los compara, registro por registro, contra los datos de hoy para encontrar qué cambió. La buena noticia es que esta técnica es minuciosa: tiene la garantía de encontrar todos los cambios. La mala noticia obvia es que, en muchos casos, esta técnica requiere muchos recursos. Si se compara una diferencia completa es necesario, intente hacer la comparación en origen, para que no tenga que transferir toda la tabla o base de datos al entorno ETL.

## - Scrapping de registro de base de datos

El scrapping de registros, toma una instantánea de los registros de la base de datos en un momento determinado. Este es un punto en el tiempo (normalmente medianoche) y luego busca transacciones que afecten a las tablas de interés para la carga ETL. Esta es probablemente la más complicada de todas las técnicas. No es raro que los registros de transacciones se llenen y eviten nuevas transacciones del procesamiento. Cuando esto sucede en un entorno de transacciones de producción, la reacción instintiva del DBA responsable puede ser vaciar el registro. para que las operaciones comerciales puedan reanudarse, pero cuando se vacía un registro, todas las transacciones dentro de ellos se pierden.

En las cargas de datos pueden darse distintas alternativas de conservación, en donde depende de las decisiones del equipo cómo va operar cada cambio al ser capturado.

### TIPO 1

| Key | AltKey | Name | Phone   | City     |
|-----|--------|------|---------|----------|
| 101 | C123   | Mary | 5551234 | New York |



| Key | AltKey | Name | Phone   | City     |
|-----|--------|------|---------|----------|
| 101 | C123   | Mary | 5554321 | New York |

### TIPO 2


| Key | AltKey | Name | Phone   | City     | Current |
|-----|--------|------|---------|----------|---------|
| 101 | C123   | Mary | 5551234 | New York | True    |



| Key | AltKey | Name | Phone   | City     | Current |
|-----|--------|------|---------|----------|---------|
| 101 | C123   | Mary | 5551234 | New York | False   |
| 102 | C123   | Mary | 5551234 | Seattle  | True    |

### TIPO 3

| Key | AltKey | Name | Phone   | OriginalCity | CurrentCity | EffectiveDate |
|-----|--------|------|---------|--------------|-------------|---------------|
| 101 | C123   | Mary | 5551234 | New York     | New York    | 1/1/00        |



| Key | AltKey | Name | Phone   | OriginalCity | CurrentCity | EffectiveDate |
|-----|--------|------|---------|--------------|-------------|---------------|
| 101 | C123   | Mary | 5551234 | New York     | Seattle     | 6/7/11        |

## - Data Profiling

El estado de sus datos depende de qué tan bien se perfilen.



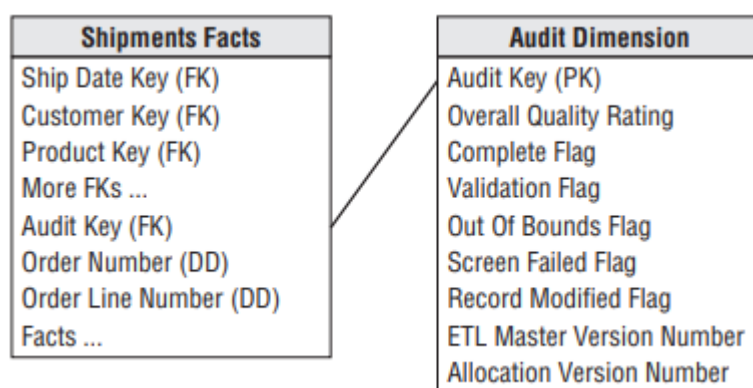
La elaboración de perfiles de datos es el proceso de examinar, analizar y crear resúmenes útiles de datos. El proceso produce una visión general de alto nivel que ayuda en el descubrimiento de problemas de calidad de datos, riesgos y tendencias generales. Más específicamente, el perfil de datos examina los datos para determinar su legitimidad y calidad. Los algoritmos analíticos detectan las características del conjunto de datos, como la media, el mínimo, el máximo, el percentil y la frecuencia, para examinar los datos en detalle minucioso. A continuación, realiza análisis para descubrir metadatos, incluidas distribuciones de frecuencia, relaciones clave, candidatos de clave externa y dependencias funcionales. Finalmente, utiliza toda esta información para exponer cómo esos factores se alinean con los estándares y objetivos del negocio.

La creación de perfiles de datos puede eliminar errores costosos que son comunes en las bases de datos de clientes. Estos errores incluyen valores nulos (valores desconocidos o faltantes), valores que no deben incluirse, valores con frecuencia inusualmente alta o baja, valores que no siguen los patrones esperados y valores fuera del rango normal.

#### - Tabla de auditoría

La tabla de auditoría es una dimensión especial que se ensambla en el sistema ETL para cada tabla de hechos. La dimensión de auditoría contiene el contexto de metadatos en el momento en que se crea una fila específica de la tabla de hechos. Se podría decir se elevan metadatos a datos reales! Para visualizar cómo se crean las filas de dimensión de auditoría, imagine esta tabla de hechos de envíos se actualiza una vez al día a partir de un archivo por lotes. Supongamos que hoy tiene una carga perfecta sin errores marcados. En este caso, generaría solo una fila de dimensión de auditoría, y se adjuntaría a cada fila de hechos cargada hoy.

Todas las categorías, puntuaciones y números de versión serían los mismos.



#### - Tablas de errores

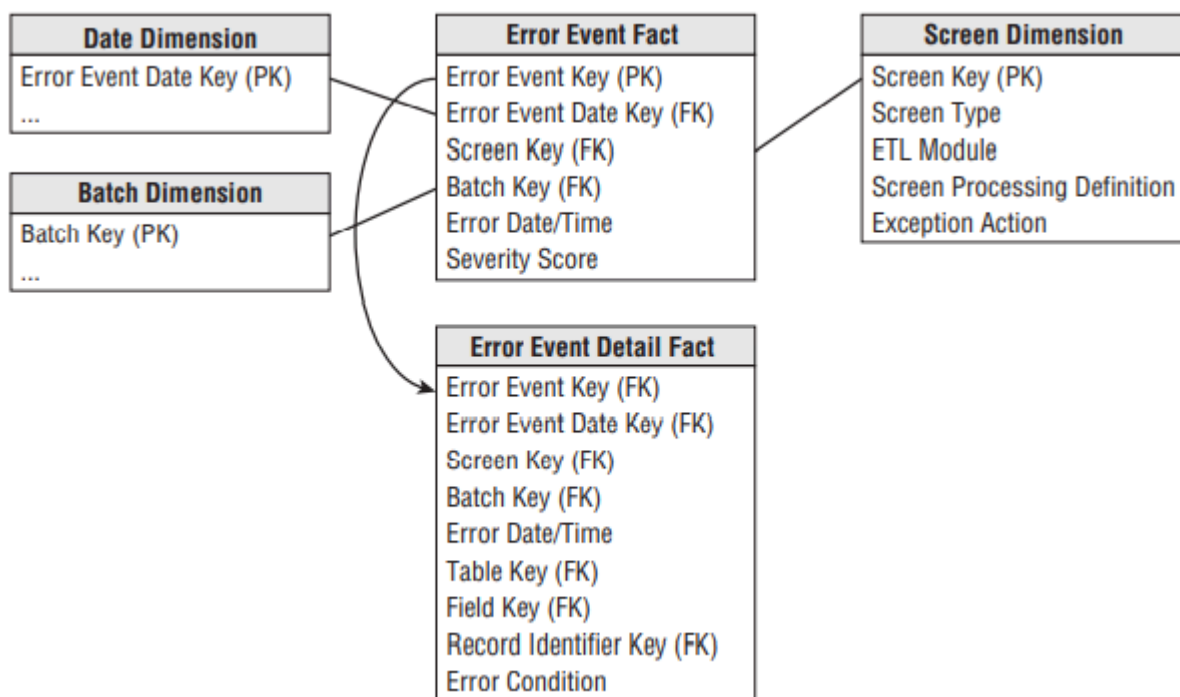
La tabla de errores es un esquema dimensional centralizado cuyo propósito es registrar cada evento de error lanzado por una pantalla de en cualquier lugar de la canalización de ETL. Aunque nos enfocamos en el procesamiento ETL, este enfoque se puede usar en

aplicaciones genéricas de integración de datos (DI) donde los datos se transfieren entre aplicaciones.

La tabla principal es la tabla de hechos de eventos de error. Sus registros se componen de cada error arrojado (producido) por una pantalla en cualquier parte del sistema ETL. Así cada error de pantalla produce exactamente una fila en esta tabla, y cada fila en la tabla corresponde a un error observado.

La tabla de hechos de eventos de error también tiene una clave principal de una sola columna, que se muestra como la clave de evento de error. Esta clave sustituta, como las claves primarias de la tabla de dimensiones, es un simple entero asignado secuencialmente a medida que se agregan filas a la tabla. Esta columna clave es necesaria en aquellas situaciones en las que se añade una enorme cantidad de filas de error a la tabla.

¡¡Esperemos que esto no suceda!!.



#### - Triggers

Un disparador es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta un evento en particular.

Un disparador queda asociado a una la tabla, la cual debe ser permanente, no puede ser una tabla TEMPORARY ni una vista. Otro punto importante es el momento en que el disparador entra en acción. Puede ser BEFORE (antes) o AFTER (después), para indicar que el disparador se ejecute antes o después que la sentencia que lo activa. Por último, se debe establecer la clase de sentencia que activa al disparador. Puede ser INSERT, UPDATE, o DELETE. Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores a insertar.

No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia. Por ejemplo, no se pueden tener dos disparadores BEFORE

UPDATE. Pero sí es posible tener los disparadores BEFORE UPDATE y BEFORE INSERT o BEFORE UPDATE y AFTER UPDATE.

Algunos usos para los disparadores es verificar valores a ser insertados o llevar a cabo cálculos sobre valores involucrados en una actualización. Por ejemplo, se puede tener un disparador que se active antes de que un registro sea borrado, o después de que sea actualizado.

La sentencia CREATE TRIGGER crea un disparador que se asocia con la tabla. También se incluyen cláusulas que especifican el momento de activación, el evento activador, y qué hacer luego de la activación:

- La palabra clave BEFORE indica el momento de acción del disparador. En este caso, el disparador debería activarse antes de que cada registro se inserte en la tabla. La otra palabra clave posible aquí es AFTER.
- La palabra clave INSERT indica el evento que activará al disparador. En el ejemplo, la sentencia INSERT causará la activación. También pueden crearse disparadores para sentencias DELETE y UPDATE.
- La sentencia siguiente, FOR EACH ROW, define lo que se ejecutará cada vez que el disparador se active, lo cual ocurre una vez por cada fila afectada por la sentencia activadora.
- Las columnas de la tabla asociada con el disparador pueden referenciarse empleando los alias OLD y NEW. OLD.nombre\_col hace referencia a una columna de una fila existente, antes de ser actualizada o borrada. NEW.nombre\_col hace referencia a una columna en una nueva fila a punto de ser insertada, o en una fila existente luego de que fue actualizada.

```
```sql
```

```
CREATE TABLE alumno (  
cedulaidentidad INT NOT NULL AUTO_INCREMENT,  
nombre VARCHAR(20),  
apellido VARCHAR(20),  
fechalnicio DATE,  
PRIMARY KEY (cedulaidentidad)  
);
```

```
CREATE TABLE alumno_auditoria (  
id_auditoria INT NOT NULL AUTO_INCREMENT,  
cedulaidentidad_auditoria INT,  
nombre_auditoria VARCHAR(20),  
apellido_auditoria VARCHAR(20),  
fechalnicio_auditoria DATE,  
usuario VARCHAR (20),  
fecha DATE,  
PRIMARY KEY (id_auditoria));
```

```
CREATE TRIGGER auditoria AFTER INSERT ON alumno  
FOR EACH ROW
```

```

INSERT INTO alumnos_auditoria (cedulaidentidad_auditoria,
                                nombre_auditoria,
                                apellido_auditoria,
                                fechalnicio_auditoria,
                                usuario,
                                fecha)
VALUES (NEW.cedulaidentidad, NEW.nombre, NEW.apellido, NEW.fechalnicio,
CURRENT_USER, NOW());
...

```

Material Complementario:

* Triggers:

<https://www.digitalocean.com/community/tutorials/how-to-manage-and-use-mysql-database-triggers-on-ubuntu-18-04-es>

* ETL Glue: <https://www.youtube.com/watch?v=yxw6kVaczyw>

* Cargas incrementales en SQL Server: <https://www.youtube.com/watch?v=NkJYEx7vFbk>

* Cargas Incrementales en Azure Data Factory:

<https://docs.microsoft.com/en-us/azure/data-factory/tutorial-incremental-copy-overview>

* Administración de ETL AWS – Glue: https://www.youtube.com/watch?v=mw6nu7-_4PI

Proyecto Integrador:

Parte 2

Normalización

1) Es necesario contar con una tabla de localidades del país con el fin de evaluar la apertura de una nueva sucursal y mejorar nuestros datos.

A partir de los datos en las tablas cliente, sucursal y proveedor hay que generar una tabla definitiva de Localidades y Provincias.

Utilizando la nueva tabla de Localidades controlar y corregir (Normalizar) los campos Localidad y Provincia de las tablas cliente, sucursal y proveedor.

2) Es necesario discretizar el campo edad en la tabla cliente.

ETL

1) Es necesario armar un proceso, mediante el cual podamos integrar todas las fuentes, aplicar las transformaciones o reglas de negocio necesarias a los datos y generar el modelo final que va a ser consumido desde los reportes.

Este proceso debe ser claro y autodocumentado.

¿Se puede armar un esquema, donde sea posible detectar con mayor facilidad futuros errores en los datos?

Optimización de Consultas

1. Generar 5 consultas simples con alguna función de agregación y filtrado sobre las tablas. Anote los resultados de la ficha de estadísticas.

2. A partir del conjunto de datos elaborado en clases anteriores, generar las PK de cada una de las tablas a partir del campo que cumpla con los requisitos correspondientes.

Optimización de Consultas

3. Generar la indexación de los campos que representen fechas o ID en las tablas:

- venta.
- cana_venta.
- producto.
- tipo_producto.
- sucursal.
- empleado.
- localidad.
- proveedor.
- gasto.
- cliente.
- compra.

4. Ahora que las tablas están indexadas, volver a ejecutar las consultas del punto 1 y evaluar las estadísticas. ¿Se nota alguna diferencia?.

5. Generar una nueva tabla que lleve el nombre fact_venta (modelo estrella) que pueda agrupar los hechos relevantes de la tabla venta, los campos a considerar deben ser los siguientes:

- IdVenta
- Fecha
- Fecha_Entrega
- IdCanal
- IdCliente
- IdEmpleado
- IdProducto
- Precio
- Cantidad

6. A partir de la tabla creada en el punto anterior, deberá poblarse con los datos de la tabla ventas.

Carga Incremental

En este punto, ya contamos con toda la información de los datasets originales cargada en el DataWarehouse diseñado. Sin embargo, la gerencia, requiere la posibilidad de evaluar agregar a esa información la operatoria diaria comenzando por la información de Ventas. Para lo cual, en la carpeta "Carga_Incremental" se disponibilizaron los archivos:

* Ventas_Actualizado.csv: Tiene una estructura idéntica al original, pero con los registros del día 01/01/2021.

* Clientes_Actualizado.csv: Tiene una estructura idéntica al original, pero actualizado al día 01/01/2021.

Es necesario diseñar un proceso que permita ingestar al DataWarehouse las novedades diarias, tanto de la tabla de ventas como de la tabla de clientes.

Se debe tener en cuenta, que dichas fuentes actualizadas, contienen la información original más las novedades, por lo tanto, en la tabla de "ventas" es necesario identificar qué registros son nuevos y cuales ya estaban cargados anteriormente, y en la tabla de clientes tambien, con el agregado de que en ésta última, pueden haber además registros modificados, con lo que hay que hacer uso de los campos de auditoría de esa tabla, por ejemplo, Fecha_Modificación.