



By HENRY HENRY

Variables:

En los scripts SQL, se pueden utilizar variables para almacenar valores durante la ejecución de una secuencia de comandos y utilizarlos luego.

MySQL reconoce diferentes tipos de variables. El primer tipo son las variables definidas por el usuario, identificadas por un símbolo @.

Para inicializar una variable definida por el usuario, necesitas usar una declaración **SET**. Se puede inicializar muchas variables a la vez, separando cada declaración de asignación con una coma. Una vez que asignas un valor a una variable, tendrá un tipo de acuerdo al valor dado.

```
SET @carrera1 = 'Data Science',@carrera2 = 'Full Stack'
```

```
SELECT @carrera1, @carrera2
```

Resulta útil muchas veces guardar el valor de una consulta **SELECT** en una variable. El valor de la variable, puede ser utilizado luego para realizar otras consultas u operaciones.

La sintaxis para asignar una variable dentro de una sentencia **SELECT** es:

```
SELECT @cod := codigo FROM cohortes WHERE idCohorte = 1235;  
SELECT @cod  
SELECT fechaInicio  
FROM cohortes  
WHERE codigo = @cod
```

En segundo lugar están las variables locales. Este tipo de variables no necesitan el prefijo @ en sus nombres, se declararan antes de que puedan ser usadas. Se pueden utilizar variables locales de dos maneras:

- Utilizando la declaración **DECLARE**.
- Como un parámetro dentro de una declaración **STORED PROCEDURE**. Se utilizan variables en los procedimientos almacenados para obtener resultados inmediatos. Estas variables son locales dentro del procedimiento.

Cuando se declara una variable local, se suele asignar un valor por defecto. La variable se inicializa con un valor **NULL** si no asignas ningún valor.

Cada variable existe en el espacio delimitado por el bloque **BEGIN - END**.

```
-- Más abajo daremos más detalles sobre los procedimientos almacenados.
DELIMITER $$

CREATE PROCEDURE GetTotalAlumnos()
BEGIN
    DECLARE totalAlumnos INT DEFAULT 0;
    SELECT COUNT(*)
    INTO totalAlumnos
    FROM alumnos;
    SELECT totalAlumnos;
END$$
DELIMITER ;

CALL GetTotalAlumnos()
```

Existen también las denominadas variables del sistema, las cuales se establecen normalmente al inicio del servidor.

Las variables del sistema se identifican con un doble signo **@** o utilizando las palabras **GLOBAL** o **SESSION** en la sentencia **SET**. Indican la configuración por defecto y pueden ser modificadas en caso de ser necesario.

Para ver las variables de sistema en uso dentro de una sesión o en el servidor, se utiliza la sentencia **SHOW VARIABLES**.

Es necesario a veces conocer el valor de estas variables, como el caso de la versión que se está utilizando, o conocer y cambiar un timeout. Son muy útiles para quienes tratan de mejorar el rendimiento de su sesión.

```
SHOW VARIABLES -- Muestra todas las variables.
```

```
SHOW SESSION VARIABLES
```

```
SHOW LOCAL VARIABLES
```

```
SHOW VARIABLES
```

-- Se puede utilizar el operador LIKE '%Variable%' para acceder a una variable en particular.

-- Ejemplo:

```
SHOW SESSION VARIABLES LIKE 'version'
```

```
SHOW SESSION VARIABLES LIKE 'version_comment'
```

Funciones:

Las funciones nos permiten procesar y manipular datos de forma procedural de un modo muy eficiente. Existen funciones integradas dentro de SQL, algunas de las que utilizamos son **AVG**, **SUM**, **CONCATENATE**, etc.

Pueden ser utilizadas en las sentencias SQL independientemente del lenguaje de programación del servidor sobre el que se ejecuten las consultas. Las funciones también se pueden crear dentro de SQL y esto permite personalizar ciertas operaciones propias del proyecto. Para poder crear funciones se deben tener los permisos **INSERT** y **DELETE**.

La sintaxis de una función almacenada es la siguiente:

```
-- EJEMPLO 1:

-- Esta función recibe una fecha de ingreso y calcula la antigüedad en meses del
alumno.

DELIMITER $$
CREATE FUNCTION antigüedadMeses(fechaIngreso DATE) RETURNS INT -- Asignamos un
nombre, parámetros de la función y tipo de dato a retornar.

-- La función se define entre BEGIN - END.
BEGIN
    DECLARE meses INT; -- Declaramos las variables que van a operar en la
función
    SET meses = TIMESTAMPDIF(MONTH, fechaIngreso, DATE(NOW())); --
Definimos el script.
    RETURN meses; -- Retornamos el valor de salida que debe coincidir con el tipo
declarado en CREATE
END$$

DELIMITER ;

SELECT * , antigüedadMeses(fechaIngreso)

FROM alumnos

-- EJEMPLO 2:

-- Esta función recibe el id de un alumno y devuelve su antigüedad en meses.

DELIMITER $$
CREATE FUNCTION antigüedadMeses2(id INT) RETURNS INT
BEGIN
    DECLARE meses INT;
    SELECT TIMESTAMPDIF(MONTH, fechaIngreso, DATE(NOW()))
    INTO meses
    FROM alumnos
    WHERE idAlumno = id;
    RETURN meses;
END$$

DELIMITER ;
```

Procedimientos Almacenados:

Es un objeto que se crea con la sentencia **CREATE PROCEDURE** y se invoca con la sentencia **CALL**. Un procedimiento puede tener cero o muchos parámetros de entrada y cero o muchos parámetros de salida.

Para definir un procedimiento almacenado es necesario modificar temporalmente el carácter separador que se utiliza para delimitar las sentencias SQL.

El carácter separador que se utiliza por defecto en SQL es el punto y coma (;). Para crear procedimientos almacenados vamos a utilizar los caracteres \$\$ para delimitar las instrucciones SQL.

DELIMITER \$\$ DELIMITER;

Resulta útil cuando se repite la misma tarea repetidas veces, siendo un buen método para encapsular el código. Al igual que ocurre con las funciones, también puede aceptar datos como parámetros, de modo que actúa en base a éstos.

Los parámetros de los procedimientos almacenados de MySQL pueden ser de tres tipos:

- **IN:** Es el tipo de parámetro que se usa por defecto. La aplicación o código que invoque al procedimiento tendrá que pasar un argumento para este parámetro. El procedimiento trabajará con una copia de su valor, teniendo el parámetro su valor original al terminar la ejecución del procedimiento.
- **OUT:** El valor de este parámetro puede ser cambiado en el procedimiento, y además su valor modificado será enviado de vuelta al código o programa que invoca el procedimiento.
- **INOUT:** Es una mezcla de los dos conceptos anteriores. La aplicación o código que invoca al procedimiento puede pasarle un valor a éste, devolviendo el valor modificado al terminar la ejecución. En caso de resultarte confuso, echa un ojo al ejemplo que verás más adelante.

Join:

La instrucción **Join**, se usa para acceder los datos de dos tablas relacionadas a través de algún campo en común, este acceso se da gracias a las “foreign key” que permiten generar las relaciones entre ellas.

Join permite tener un alto grado de normalización en las tablas y aún así poder accederlas de forma sencilla.

Teoría de conjuntos

El álgebra de conjuntos engloba las relaciones que se pueden establecer entre ellos.

- **Unión de conjuntos:** La unión de dos o más conjuntos contiene cada elemento que está contenido, al menos, en alguno de ellos.
- **Intersección de conjuntos:** La intersección de dos o más conjuntos incluye todos los elementos que estos conjuntos comparten o que tienen en común.
- **Diferencia de conjuntos:** La diferencia de un conjunto respecto a otro es a igual a los elementos del primer conjunto menos los elementos del segundo.
- **Conjuntos complementarios:** El complemento de un conjunto incluye todos los elementos que no están contenidos en dicho conjunto (pero que sí pertenecen a otro conjunto de referencia).
- **Diferencia simétrica:** La diferencia simétrica de dos conjuntos incluye todos elementos que están en uno o en otro, pero no en ambos al mismo tiempo.
- **Producto cartesiano:** Es una operación que da como resultado un nuevo conjunto. Contiene como elementos los pares ordenados o las tuplas (series ordenadas) de los elementos que pertenecen a dos o más conjuntos. Son pares ordenados si se trata de dos conjuntos y tuplas si son más de dos conjuntos.

Diagrama de Venn

Como vimos en módulos anteriores los diagramas de Venn son esquemas usados en la teoría de conjuntos. Estos diagramas muestran colecciones (conjuntos) de cosas (elementos) por medio de líneas cerradas. La línea cerrada exterior abarca a todos los elementos bajo consideración, el conjunto universal U. Usa círculos que se superponen u otras figuras para ilustrar las relaciones lógicas entre dos o más conjuntos de elementos.

- **Conjunto:** Una colección de cosas. Dada la versatilidad de los diagramas de Venn, las cosas pueden ser realmente lo que quieras. Pueden ser elementos, objetos, miembros o términos similares.
- **Unión:** Todos los elementos de los conjuntos.
- **Intersección:** Los elementos que se superponen en los conjuntos. A veces se denominan "subconjuntos".
- **Diferencia simétrica entre dos conjuntos:** Todo, excepto la intersección.
- **Complemento relativo:** En un conjunto pero no en el otro..

Tipos de JOIN en SQL

- **Inner Join:** Devuelve sólo aquellas filas donde haya un "match", es decir, las filas donde el valor del campo de la tabla A que se utiliza para hacer el Join coincida con el valor del campo correspondiente en la tabla B. Ejemplo: devolver todos los productos para los que haya como mínimo un pedido en los últimos días (el inner join enlazará el campo producto en la tabla Pedido con la clave primaria de ese producto en la tabla Producto).
nSe aplica por defecto que se aplica cuando no indicamos otra cosa al hacer la consulta.

- **Left Join:** Cuando quieres todas las filas para las que haya match pero también aquellas de la Tabla A que no hagan match. Siguiendo el ejemplo anterior, si quieres listar todos los productos con datos de sus pedidos pero mostrando también aquellos productos para lo que no tengas todavía un pedido, la solución sería hacer una Left Outer join entre Producto y Pedido.
- **Right Join:** Exactamente lo mismo pero a la inversa, cuando quieres listar las filas de la tabla B aunque no estén relacionadas con ninguna fila de la tabla A. Es un operador un poco redundante ya que se podría cambiar simplemente el orden de las tablas en el Join y utilizar un left para conseguir el mismo efecto. No obstante, y como parte de Joins múltiples, es útil tener los dos para un mejor comprensión de la consulta
- **Full outer join:** Es como la suma de las dos anteriores. Queremos tanto las filas de la A como las de B, tanto si hay match como si no (evidentemente cuando haya match la consulta devolverá todos los campos de A y B que hayamos indicado, cuando no, la consulta devolverá sólo los campos de A o B).

Subconsultas

Una consulta secundaria es una consulta de selección que está contenida dentro de otra consulta. La consulta de selección interna generalmente se usa para determinar los resultados de la consulta de selección externa. Consiste en utilizar los resultados de una consulta dentro de otra, que se considera la principal.

Es una sentencia **SELECT** anidada dentro de una instrucción **SELECT**, **SELECT...INTO**, **INSERT...INTO**, **DELETE**, o **UPDATE** o dentro de otra subconsulta.

Una subconsulta tiene la misma sintaxis que una sentencia **SELECT** exceptuando que aparece encerrada entre paréntesis, no puede contener la cláusula **ORDER BY**, ni puede ser la **UNION** de varias sentencias **SELECT**.

Las consultas que utilizan subconsultas suelen ser más fáciles de interpretar por el usuario. Recordemos que dentro del motor SQL, los datos solo son accesibles a través de consultas. Es por ello que si queremos por ejemplo obtener la fecha donde ingreso el primer estudiantes dentro del moodelo desarrollado en el modulo anterior, la instrucción sería la siguiente:

```
SELECT MIN(fechaIngreso) AS fecha
```

```
FROM alumnos
```

Si tuviéramos la intención de utilizar este valor para obtener un listado de todos los estudiantes que ingresaron en esa fecha lo intuitivo sería lo siguiente:

```
SELECT idAlumno, fechaIngreso
```

FROM alumnos

WHERE fechaIngreso = MIN(fechaIngreso)

Lo anterior no resulta posible, debido a que como mencionamos más arriba para poder acceder a ese valor MIN(fechaIngreso), se debe hacer una consulta. Es por ello que aquí radica la utilidad y el fundamento en la utilización de subconsultas.

SELECT idAlumno, fechaIngreso

FROM alumnos

WHERE fechaIngreso = (SELECT MIN(fechaIngreso) AS fecha FROM alumnos)

Resulta siempre muy importante entender que valor esta devolviendo una consulta específica, ya que de haber presentado esta última instrucción de forma inicial quizás la comprensión hubiera sido más difícil. Las subconsultas pueden devolver al igual que cualquier otra instrucción tablas o valores. Y estos resultados a su vez pueden utilizarse para otro tipo de operaciones.

Las subconsultas son un proceso de selección interno, y se pueden utilizar en cualquier sentencia que permita una expresión.

- **SELECT:** Para calcular y crear un nuevo campo virtual a la consulta principal.
- **FROM:** Para devolver una tabla secundaria calculada o un campo calculado con un contexto diferente.
- **WHERE:** Para definir filtros compuestos calculados. Si se conociera el valor a calcular con la subconsulta, su utilizaría ese valor. Por ej:

La fecha más reciente "MAX(date)". El operador utilizado para comparar, depende del resultado de la subconsulta:

Listas --> IN , NOT IN.

Valor único --> = , >=, <=, etc.

Las tablas de las subconsultas, solo "existen" en ellas. Es decir, por fuera del parentesis que abarca la subconsulta, no se pueden agregar campos que refieran a esa tabla. Salvo que se incluya en el FROM de la principal.

En resumen:

- Cuando necesites agregar un filtro calculado, agregarás una subconsulta en **WHERE** o **HAVING**.
- Cuando necesites anexar un campo calculado de otra tabla y este NO requiera agregación, la utilizaras en **SELECT**.
- Cuando necesites anexar un campo calculado de otra tabla y este SI requiera agregación, la utilizaras en **FROM**. Para romper el contexto de filas.

¡EXISTEN MÁS CASOS Y POSIBILIDADES!

En cuestiones de rendimiento, no se puede asegurar al 100% que las subconsultas sean más efectivas que otros métodos como JOIN, UNION o variables, ya que en realidad esto depende de diversos factores como el manejo de los índices, el tamaño de las tablas, el motor de la base de datos, entre otros. No obstante, proveen una forma bastante interesante de extraer datos.

Vistas

Es un mecanismo que permite almacenar de forma permanente el resultado de una consulta en SQL. A su vez este resultado almacenado en la vista se puede acceder como si fuera una tabla, denominándose a la vista como una tabla virtual. Las vistas se componen de campos y filas provenientes del resultado de la consulta, las cuales pueden venir de varias tablas. Al igual que con los otros objetos que forman parte de la base de datos se crean mediante la sentencia **CREATE** y se eliminan mediante **DROP**.

-- Crear una vista.

```
CREATE VIEW primerosAlumnos AS SELECT idAlumno, fechaIngreso FROM alumnos  
WHERE fechaIngreso = ( SELECT MIN(fechaIngreso) AS fecha FROM alumnos)
```

-- Obtener los resultados de una vista.

```
SELECT *  
FROM primerosAlumnos
```

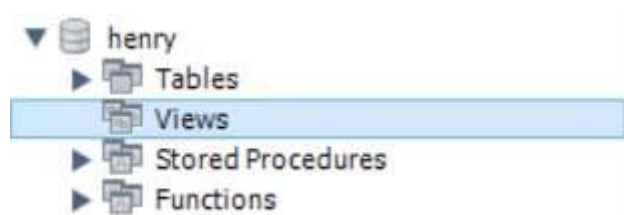
-- Modificar una vista.

```
ALTER VIEW primerosAlumnos AS SELECT idAlumno, CONCAT(apellido," ",nombre), fechaIngreso  
FROM alumnos WHERE fechaIngreso = ( SELECT MIN(fechaIngreso) AS fecha FROM alumnos)
```

-- Eliminar una vista

```
DROP VIEW primerosAlumnos
```

Al crear una vista, esta queda alojada en la base de datos correspondiente y se pueden ver en la interfaz del gestor de base de datos.



Una vista actúa como filtro de las tablas subyacentes a las que se hace referencia en ella. La consulta que define la vista puede provenir de una o de varias tablas, o bien de otras vistas de la base de datos actual u otras bases de datos. Asimismo, es posible utilizar las consultas distribuidas para definir vistas que utilicen datos de orígenes heterogéneos.

Esto puede resultar de utilidad, por ejemplo, si desea combinar datos de estructura similar que proceden de distintos servidores, cada uno de los cuales almacena los datos para una región distinta de la organización.

Ventajas...

- **1 Permite centrar, simplificar y personalizar la forma de mostrar la información a cada usuario.**
- **2 Se usa como mecanismo de seguridad, el cual permite a los usuarios obtener acceso a la información proveniente de la vista sin acceder a otras opciones.**
- **3 Vistas en SQL Server.**
- **4 Vistas en MySQL.**

FUNCIONES VENTANA

MySQL ha admitido funciones de ventana desde la versión 8.0. Esta función se admite desde hace mucho tiempo en la mayoría de las bases de datos comerciales y algunas bases de datos de código abierto, y algunas también se denominan funciones analíticas.

El concepto de ventana es muy importante, puede entenderse como un conjunto de registros y una función que se ejecuta sobre un conjunto de registros que cumple determinadas condiciones. Para cada registro se debe ejecutar una función en esta ventana. Es decir que se trabaja en la relación de una fila con el resto, dentro de la ventana definida, o bien, selección de datos definida.

Las funciones ventana evita los **JOIN** de una tabla consigo misma, por ejemplo, si en una consulta de ventas por fecha, quisiéramos además tener el valor del promedio de venta, también por fecha, necesitaríamos por un lado, realizar una agregación de ventas por fecha, obteniendo el promedio y por el otro lado, realizar la consulta de todas las ventas para cada fecha, comparando así ambos valores:

Ahora bien ¿cómo podría una función ventana ayudar en este caso? En principio, vamos a concluir que nuestra "ventana" serán los registros pertenecientes a una misma fecha, sobre los cuales, vamos a aplicar una función de agregación para obtener el promedio. Notar la mejora en el tiempo de respuesta, **la función ventana viene con una mejora en la performance**. La consulta final queda:

La función ventana se puede descomponer en las siguientes partes:

- **OVER:** Define una ventana o conjunto de filas que debe utilizar una función ventana, incluyendo cualquier orden. No está restringido por sí mismo, e incluye todas las filas. Se pueden usar múltiples cláusulas OVER en una sola consulta, cada una con su propio particionamiento y ordenación si es necesario.
- **Cláusula de partición (PARTITION BY):** Las ventanas se agrupan según esos campos y las funciones de ventana se ejecutan en diferentes grupos.
- **Orden por cláusula (ORDER BY):** Según los campos a ordenar, la función de ventana numerará los registros según el orden de clasificación. Se puede utilizar junto con la cláusula de partición o solo.
- **Cláusula de marco:** El marco es un subconjunto de la partición actual. La cláusula se usa para definir las reglas del subconjunto y generalmente se usa como una ventana deslizante.
 - UNBOUNDED significa ir todo el camino al límite en la dirección especificada por PRECEDING o FOLLOWING (comienzo o final)
 - CURRENT ROW indica inicio o final en la fila actual en la partición
 - ROWS BETWEEN permite definir un rango de filas entre dos puntos

Repasemos algunos ejemplos de función ventana, tomando de base el modelo de datos con el que se viene trabajando hasta el momento.

Se requiere visualizar una tabla con el acumulado de la venta por fecha, en este caso, se hace uso del marco "ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW" para tomar dentro de la partición, las filas desde el inicio de la misma hasta la fila actual:

a función RANK() tiene la particularidad de que utiliza un salto o gap entre los registros, notar lo que pasa en la consulta anterior, con el puesto 4 del ranking. El cuarto aparece dos veces, debido

a que son ventas con el mismo valor, y luego el siguiente valor de ranking es el 4. Con el mismo 4 y con el 7, vuelve a pasar lo mismo:

Ranking_Venta	Fecha	IdCliente	Precio	Cantidad	Venta
25	2015-01-10	838	457.820	2	915.640
26	2015-01-10	540	543.180	1	543.180
26	2015-01-10	540	543.180	1	543.180
1	2015-01-12	2765	1254.880	3	3764.640
2	2015-01-12	328	1237.500	3	3712.500
2	2015-01-12	328	1237.500	3	3712.500
4	2015-01-12	2765	1254.880	2	2509.760
4	2015-01-12	2765	1254.880	2	2509.760
6	2015-01-12	328	1237.500	2	2475.000
7	2015-01-12	1476	512.000	2	1024.000
7	2015-01-12	1476	512.000	2	1024.000
9	2015-01-12	783	353.320	2	706.640
1	2015-01-13	1370	2802.800	3	8408.400
2	2015-01-13	548	4141.000	2	8282.000

Quizas querramos ver esto de otra manera, para lo cual, deberíamos usar la función DENSE_RANK():

Qué pasa si ahora, se quieren mostrar solamente las 3 ventas más altas por cada fecha. Para esto, sí vamos a acudir a una subconsulta, ya que no es posible utilizar la cláusula WHERE con las funciones ventana, debido a que la función ventana, ejecuta en el último paso, justo antes del ORDER BY, esto es parte de lo que le da su buena performance.

Al listado anterior, agregamos el requerimiento de que sea para la sucursal con id = 12 y además que se muestre el porcentaje acumulado, haciendo uso de la función PERCENT_RANK():

Se requiere ver el listado de clientes numerado, particionando por Localidad. Para esto, es posible hacer uso de la función ROW_NUMBER()

Agreguemos ahora, el primer y el último nombre de los clientes, haciendo uso de las funciones

A la consulta anterior, agregamos la necesidad de ver el enésimo nombre de los clientes, haciendo uso de la función NTH_VALUE(<posición>):

Se requiere ver un listado con el detalle de las ventas para cada cliente, que contenga la cantidad de días transcurridos entre operación y operación de venta, para lo cual, es útil la función LEAD() que trae el valor que contiene ese campo en el registro anterior, según la partición y el orden que se le de. La función LAG() obtiene el valor que contiene el registro siguiente:

Con el listado anterior, ahora es necesario obtener el promedio de días que transcurren, por cliente, entre operación y operación de venta:

También, es posible definir las ventanas con un alias:

Convertir los Datos en Conocimiento

Los datos son un conjunto de hechos almacenados.

La información es el conjunto de datos procesados en tiempo y en forma, que constituyen un mensaje relevante y reduce la incertidumbre.

El conocimiento se adquiere con la práctica y la experiencia. Dota a las personas con la capacidad de tomar decisiones.

Inteligencia de Negocios (BI)

Con la experiencia nos hacemos expertos...

- Adquirimos el conocimiento sobre el dominio del negocio.
- Somos capaces de describir su comportamiento y comprender sus aspectos y variables más importantes.
- ¿Está esa experiencia e información, de donde se obtuvo el conocimiento, plasmada en los datos?
- ¿Cuánto valor se le puede dar entonces a los datos que el negocio genera?
- ¿Cómo podemos hacer para generar información a partir de los datos?
- Ese es el objetivo principal de la Inteligencia de Negocios, convertir los datos en información oportuna y relevante por medio de diferentes técnicas de transformación, análisis y visualización.

Soporte a la Decisión...

- En una organización se toman decisiones
- Es importante mitigar la incertidumbre
- Lograr respaldo y seguridad

- La Inteligencia de Negocios, enfocada en la calidad del dato, procura procesarlo, desde su origen para su análisis
- Soporte a la toma de decisiones

En una **organización**, normalmente se trabaja con diversos sistemas transaccionales, es decir, sistemas que se utilizan para la operatoria diaria, que son el punto de ingreso de los datos, a través de clientes, operadores, administrativos, usuarios en general, pero también se generan datos a partir de dispositivos de medición, como sensores o mecanismos de log, por ejemplo un servidor web dedicado a un portal de venta de productos, que en cada click deja un log con los datos del producto y usuario que lo visitó.

Esto trae en consecuencia que en una primera instancia **los datos no están en un único repositorio**, ni tienen necesariamente una estructura apta para ser almacenados en una base de datos relacional (RDBMS), Por tanto, al momento de querer extraer datos para su posterior análisis, nos encontramos con el problema de que debemos acceder a varias fuentes, unificarlo en un repositorio común, y luego realizar las consultas requeridas.

En este proceso hay una situación muy importante, y tiene que ver con que el hecho de tener que **unificar datos**, de distintas fuentes y darle una estructura adecuada, deja visibles inconvenientes de incongruencia, incompletitud y falencias varias en los datos, producto de que desde un principio, no se diseñó de manera integral el camino que transcurre el dato desde que se origina hasta que es utilizado y se transforma en información y conocimiento consecuentemente, este camino del dato es conocido como “Ciclo de Vida del Dato”, y es un concepto al que recurriremos frecuentemente.

El punto a cuidar aquí, tiene que ver entonces con el grado de calidad que podemos garantizar que la información que se disponibiliza va a tener, por lo tanto, es importante poder tener una apreciación clara acerca de la forma de cuantificar el grado de calidad, para lo cual, en primer lugar, es necesario tener en cuenta que los datos de la realidad pueden ser impuros debido a incompletitud, ruido o inconsistencias, sin embargo, debemos asegurarnos la fiabilidad del dato, teniendo el camino del dato, es decir, la trazabilidad.

Lo que presentamos es una tarea muy importante debido a que la calidad de datos confiabiliza en análisis. Si utilizamos datos no confiables podemos llegar a conclusiones erróneas.

Este proceso nos ayuda a **recuperar información perdido o incompleta y resolver conflictos en los datos**. Así mismo, las fuentes son críticas en el proceso de selección de datos. Por último, es necesario armar métodos de control, auditoría, corrección y confiabilización de datos.

Calidad del Dato

¿Para qué buscamos la calidad de los datos?

- La calidad de datos confiabiliza en análisis.

- Si utilizamos datos no confiables podemos llegar a conclusiones erróneas.
- Las fuentes son críticas en el proceso de selección de datos.
- Existen métodos de control, auditoría, corrección y confiabilización de datos.
- Recuperar información perdida o incompleta.
- Resolver conflictos en los datos.

Definición

- Implica transformar los datos y disponibilizarlos.
- Los datos de la realidad pueden ser impuros debido a incompletitud, ruido o inconsistencias.
- Debemos asegurarnos la fiabilidad del dato, teniendo el camino del dato, es decir, la trazabilidad.

Causas de la mala calidad de los datos

- Carga de datos en forma manual o Data Entry.
- Carga de datos externos sin los recaudos correctos para su adecuación.
- Problemas de carga originados en los sistemas transaccionales utilizados como fuente de datos.
- Implementación de nuevas aplicaciones en la organización, implica nuevos orígenes de datos, que necesitan ser congruentes con los datos ya existentes.
- Cambios en las aplicaciones existentes o migraciones de sus bases de datos.

ACTUALIZACION:

- Los datos deben estar actualizados. Debe existir referencias de la fecha de confección o de la fecha de última actualización.
- Por ejemplo: Información de deuda sin una referencia en cuanto a la fecha de actualización.

COMPLETITUD:

- Los datos deben estar completos. Puede parecer obvio pero es una de las situaciones más habituales.
- Por Ejemplo: tablas con datos filiatorios y de contacto con campos vacíos aleatoriamente.

FIABILIDAD:

- La procedencia y la trazabilidad del dato son características que hacen a la fiabilidad.
- Si trabajamos con una tabla de la que no conocemos su procedencia o bien que bajamos de una pagina poco confiable, los datos no serán fiables y nos pueden llevar a análisis erróneos.
- De igual manera, si no podemos reconstruir el camino completo del dato desde su captura hasta la actualidad, el conjunto de datos no tiene trazabilidad.

ACCECIBILIDAD:

- Los datos deben ser accesibles con bajo nivel de esfuerzo.
- Deben estar en lugares previsibles y ser fácilmente ubicables y elegibles.
- Ejemplo: Una tabla con nombres de campos numerados: Campo1, Campo2, etc...
- Ejemplo 2: Un reporte que se aloja en una ubicación poco habitual.

CONSISTENCIA:

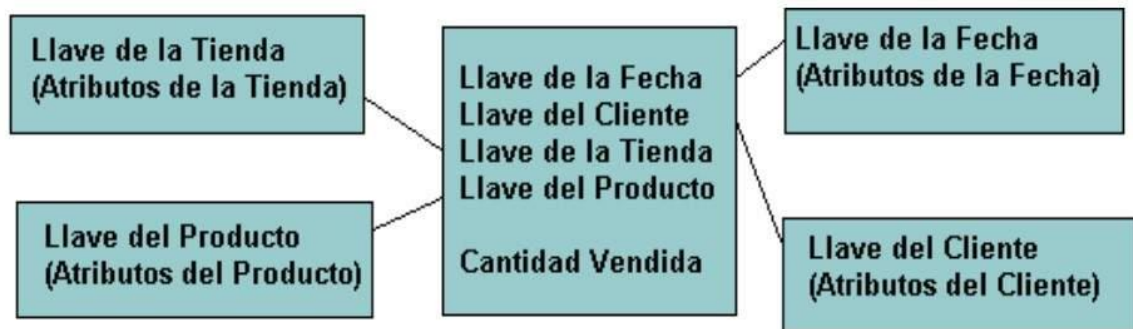
- Interna: Calidad de caracteres y de lo que se guarda en los campos.
- Externa: Calidad de interdependencia y racionalidad de los campos.
- Ejemplo: Las claves primarias y las claves foráneas deben ser consistentes y permitir la relación entre tablas.

Hechos y Dimensiones

Son **tablas dimensionales o maestros**, aquellas que contienen información respecto de las entidades que están incluidas en el modelo. Por ejemplo una tabla de clientes, donde se tenga una clave primaria para identificar a cada uno de manera unívoca, y además los datos filiatorios, como ser nombre, apellido, fecha de nacimiento ó nacionalidad.

Existe un y sólo un registro por cada **cliente**. Son tablas de hechos aquellas que guardan eventos donde se relacionan las entidades incluidas en el modelo y también las métricas asociadas. Por ejemplo una tabla de ventas, donde se guarda la fecha de la venta, cliente y producto implicados, la sucursal actuante y el monto de la venta realizada. Existe un registro por evento, al cuál se asocian las entidades relacionadas y también una métrica.

Consecuentemente, una tabla de hechos puede también reflejar una entidad en sí misma, como el caso de la venta ó una generación de factura. Concluyendo entonces que ambos, hechos y dimensiones, son también entidades.



Preparación de los Datos

En la preparación de los datos se pueden generar conjuntos de datos mas pequeños que el original. En este sentido es importante la agrupación en «temas» o «familias» de datos. El objetivo es generar «información de calidad» desestimando la información accesoria o redundante.

- La etapa de preparación de datos llega a insumir el 90% del trabajo de una tarea de análisis de datos.
- Sin la preparación de datos no es posible realizar análisis veraces.

Metodología para la preparación y análisis

En la etapa de la preparación de datos se procede a la integración, limpieza, transformación, modelado y reportes.

Las técnicas de preparación son obligatorias para la incorporación de nuevos datos a nuestras bases, las mismas son:

- Integración
- Limpieza del ruido
- Transformación:
- Imputación de valores faltantes.

Integración de Datos



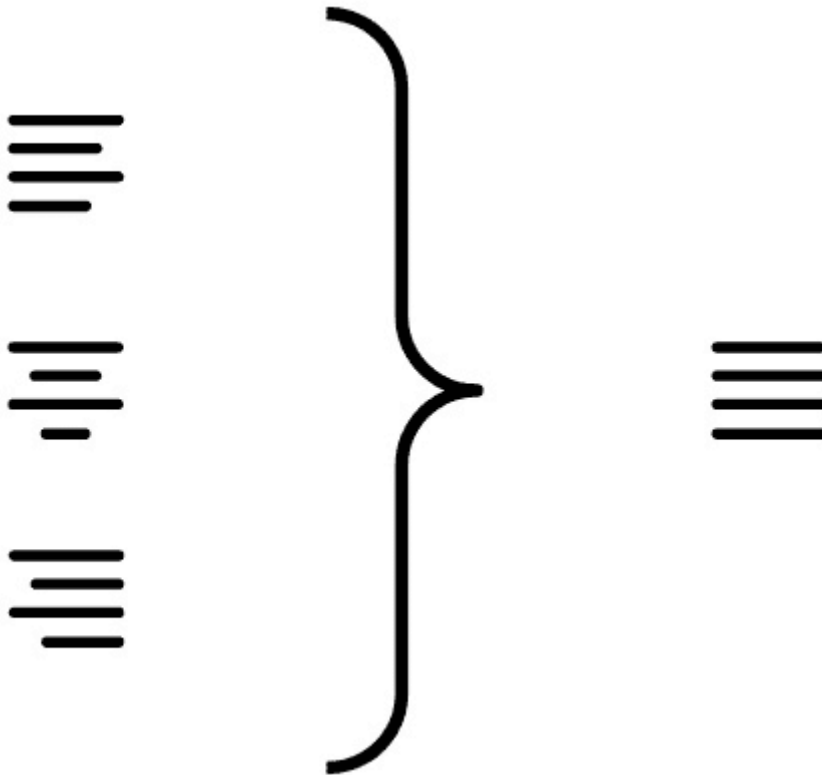
Es el primer paso e implica integrar datos de diferentes fuentes e incorporar los datos a formatos tabulares si no lo estuvieran.

Limpieza de datos



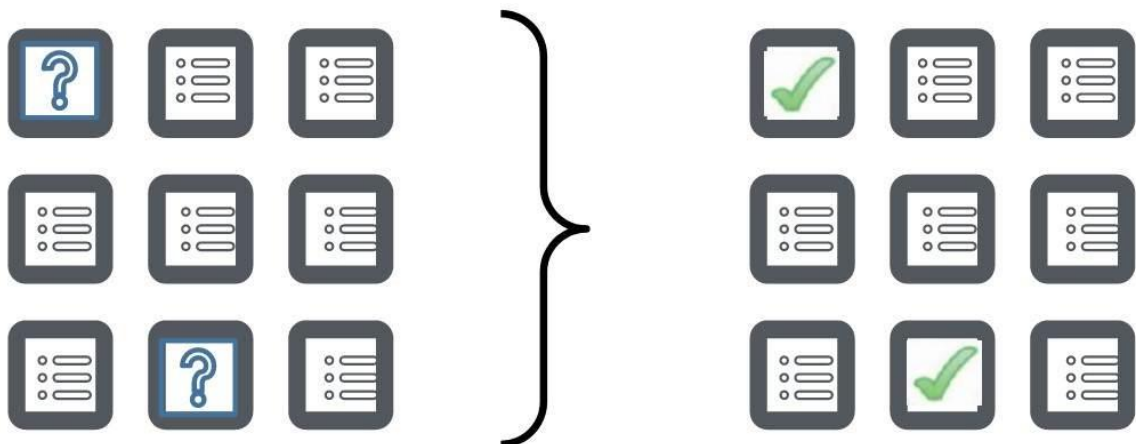
Proceso por el cual se discriminan los datos importantes de los accesorios y a la vez los veraces de los erróneos. Luego se procede a desestimar o borrar aquellos datos que no serán utilizados y a validar los que se conservarán.

Normalización de Datos



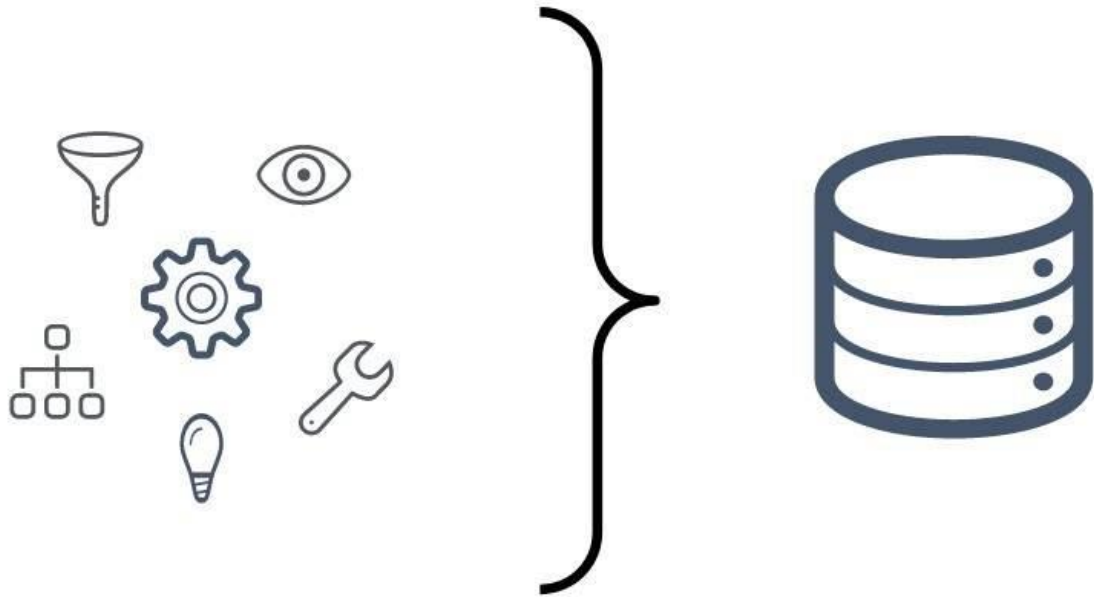
En este paso se identifican aquellos valores iguales con notaciones diferentes y se los reescribe de una manera uniforme. Ejemplo: Calle S Martín, Calle Gral. San Martín, Calle José de San Martín.... = Calle Gral. José de San Martín.

Imputación de Valores Faltantes



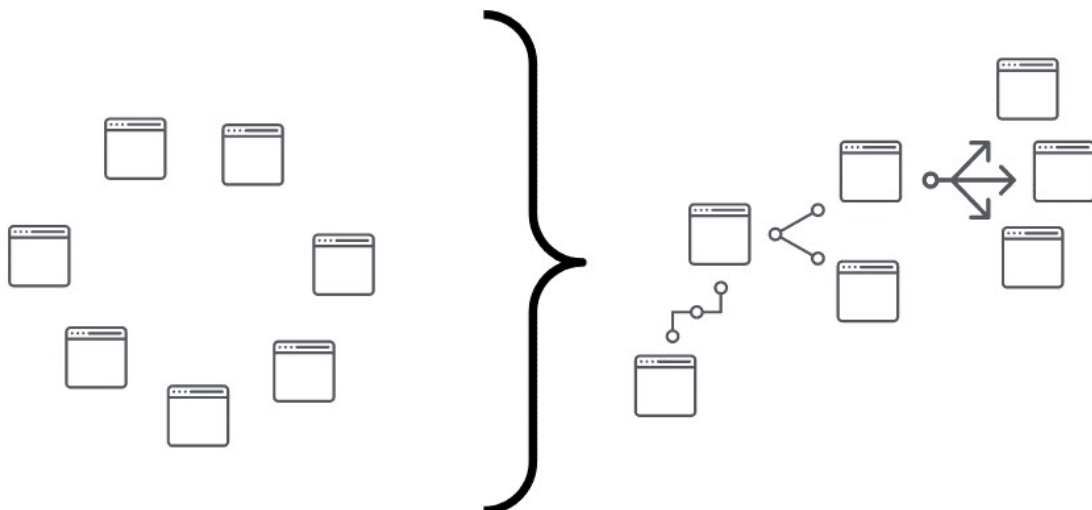
La imputación de valores faltantes esta relacionada con la identificación de los casos en donde exista perdida o ausencia de datos. En estos casos será importante realizar acciones de reconstrucción, aunque no siempre es posible resolverlas.

Transformación de Datos



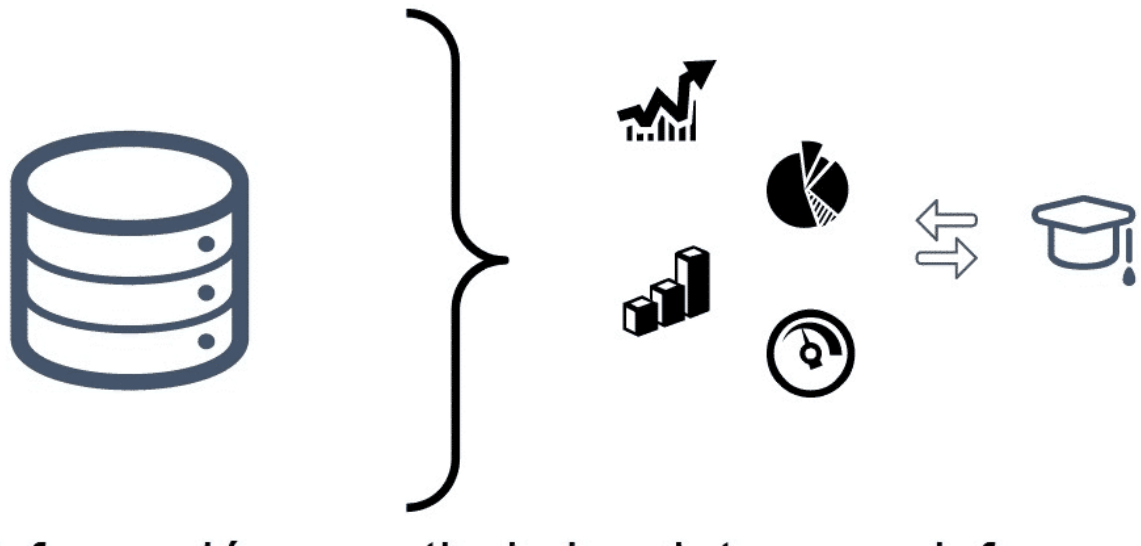
Este es el momento en que la información nueva será convertida a un formato compatible con base de datos. En este punto es crítico reducir el efecto distorsivo de la transformación.

Modelado de Datos



Se establecen las relaciones entre los datos, se determinan que entidades contienen datos maestros y cuáles contienen hechos, dando lugar a las dimensiones y las métricas.

Reportes y Visualización



Se genera Información a partir de los datos, esa información queda disponible para su análisis, que da lugar al Conocimiento.

Recomendaciones Generales:

- Lo simple es mejor.
- Evitemos detalles superfluos.
- Creemos variables adicionales antes de destruir las originales.
- Analicemos los detalles antes de confiar en los resúmenes.
- Verificar la precisión de las variables derivadas y las recodificadas.

Claves subrogadas: una clave subrogada es un identificador único que se asigna a cada registro de una tabla. Puede obtenerse a partir de la conjunción de columnas ya preexistentes.

Extracción, Transformación y Carga - 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

Para llevar a cabo el proceso de extracción, primera parte del ETL, hay que seguir los siguientes pasos:

- 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.

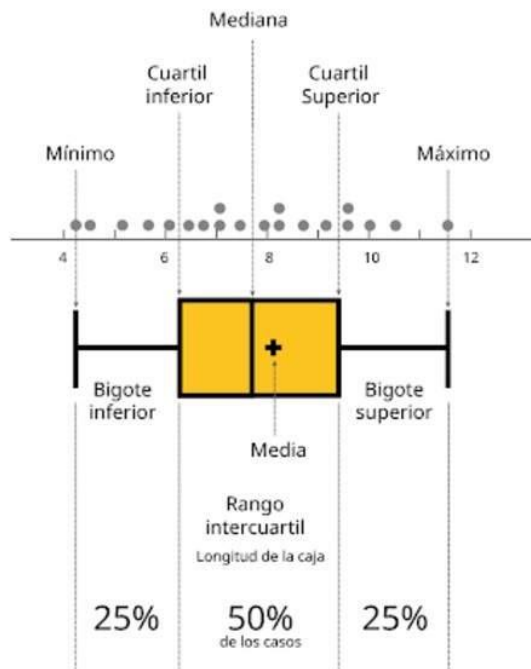
Outliers

Un elemento fundamental a descubrir dentro de las tareas de identificación del ruido son los valores atípicos o outliers...

Los **Outliers** son elementos que por su comportamiento se apartan notoriamente del comportamiento general. Esto se puede deber a un error en los datos o a un dato correcto que representa anomalías en la realidad.

Diagrama de Caja y Regla de las tres sigmas

Diagrama de Caja



El Diagrama de Caja permite observar la distribución completa de los datos al mismo tiempo que su mediana y sus cuartiles. También, muestra los elementos que se escapan del universo, los outliers.

Rango intercuartílico o IQR:

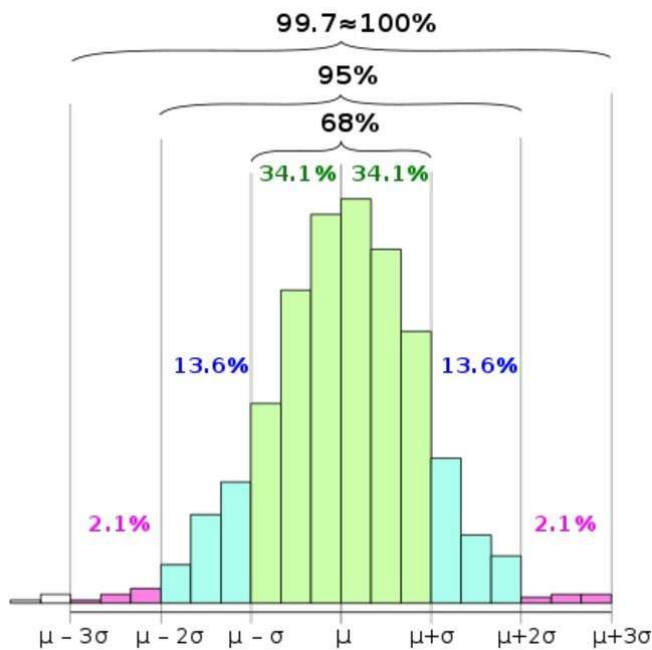
- **mínimo** = $Q1 - 1.5 \times IQR$
- **máximo** = $Q3 + 1.5 \times IQR$

Regla de las tres sigmas:

La Regla de las Tres Sigmas se basa en el valor promedio y la desviación estándar para obtener el rango, fuera del cual, podemos asumir que un valor es atípico.

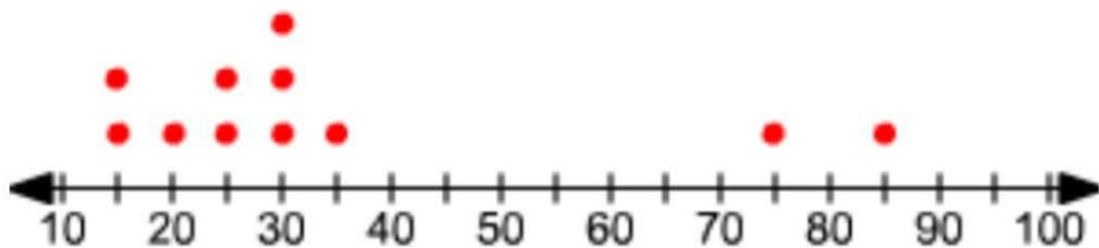
mínimo = Promedio – 3 * Desviación Estándar

máximo = Promedio + 3 * Desviación Estándar



Consideraciones

A veces, es la variable la que nos lo indica. Por ejemplo, la asistencia a un curso no puede ser menor que cero o mayor al número de alumnos que tiene el curso.



Query Optimization

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.

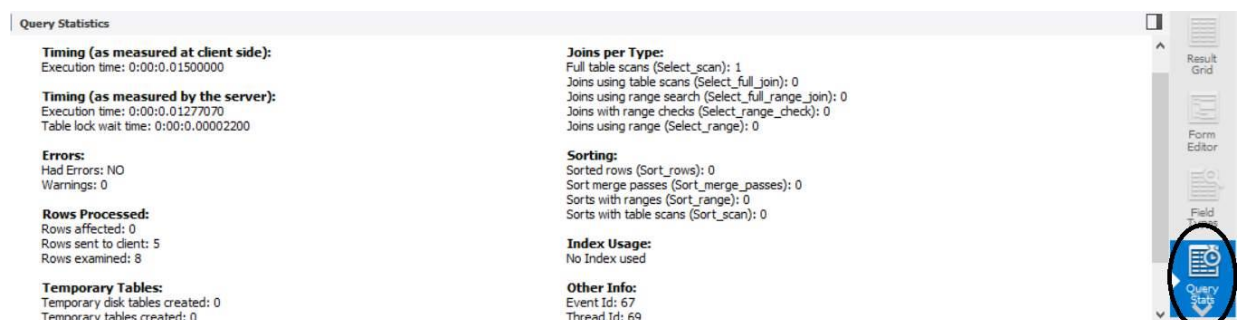
Las principales consideraciones para optimizar las consultas son:

- 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%'` (profundizaremos en estas funciones en las clases siguientes).

En las próximas clases avanzaremos sobre la utilización de funciones y consultas complejas, el abordaje inicial de estos temas nos permite entender los escenarios de su utilización y la mejor combinación al momentos de utilizarlas.

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".



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, que 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.

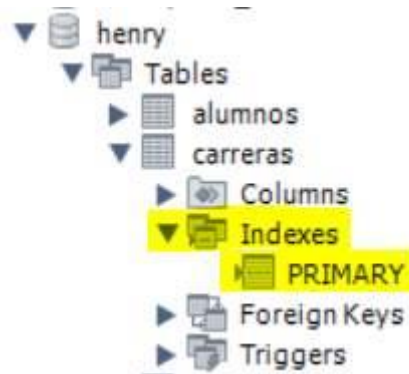
```
CREATE TABLE carrera (idCarrera INT NOT NULL AUTO_INCREMENT,nombre VARCHAR (20) NOT NULL, PRIMARY KEY (idCarrera) --Aquí al crear una PK, SQL además crea un índice agrupado.);
```

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 aplicado claves concatenadas.

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

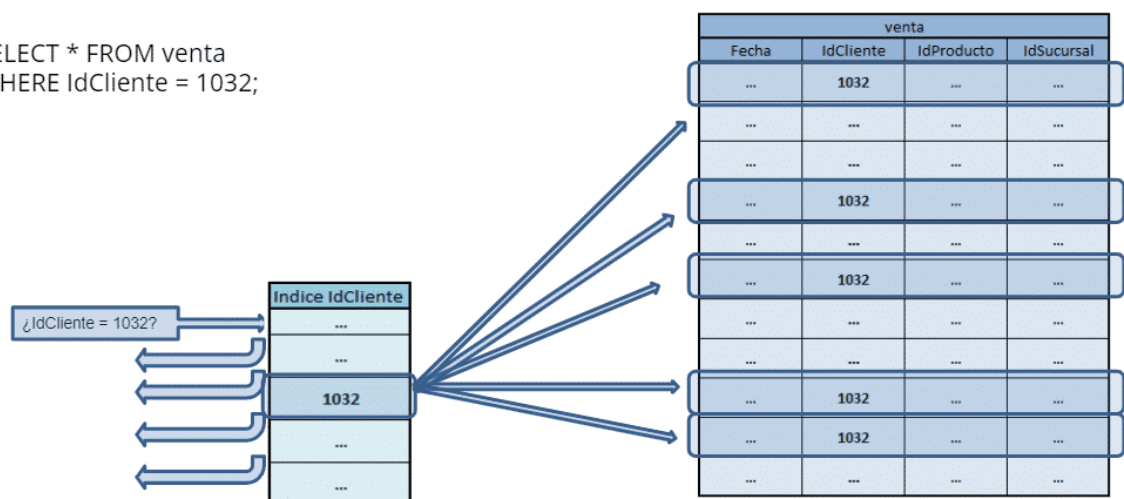


Con el código superior **CREATE INDEX** estaríamos creando uno o varios índices ordinarios en una tabla existente.

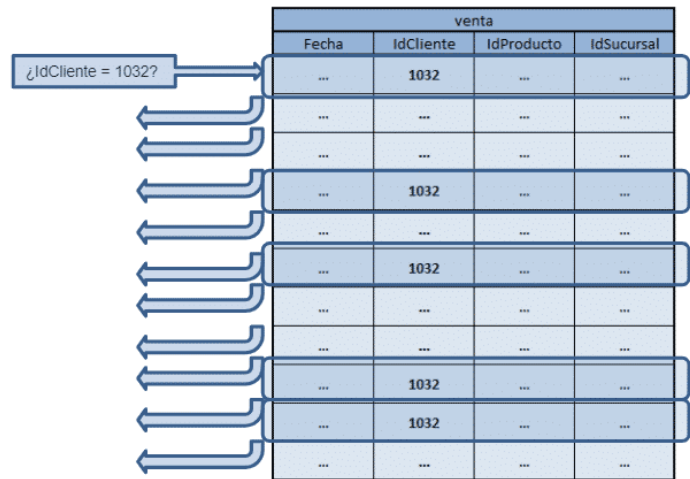
```
1 CREATE INDEX nombre_indice ON nombrede_tabla(columna [columna2...]);
```

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. 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.

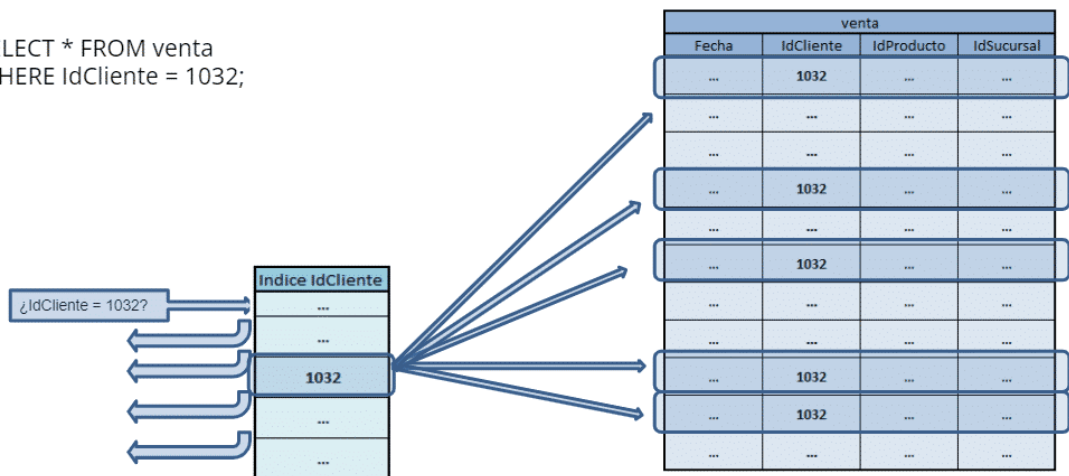
```
SELECT * FROM venta  
WHERE IdCliente = 1032;
```



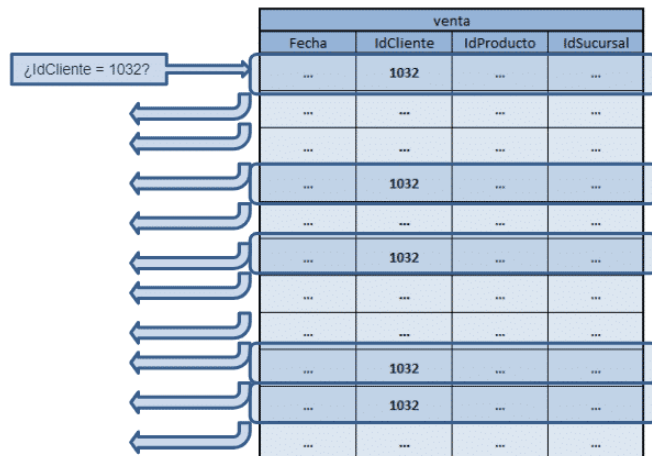
SELECT * FROM venta
WHERE IdCliente = 1032;



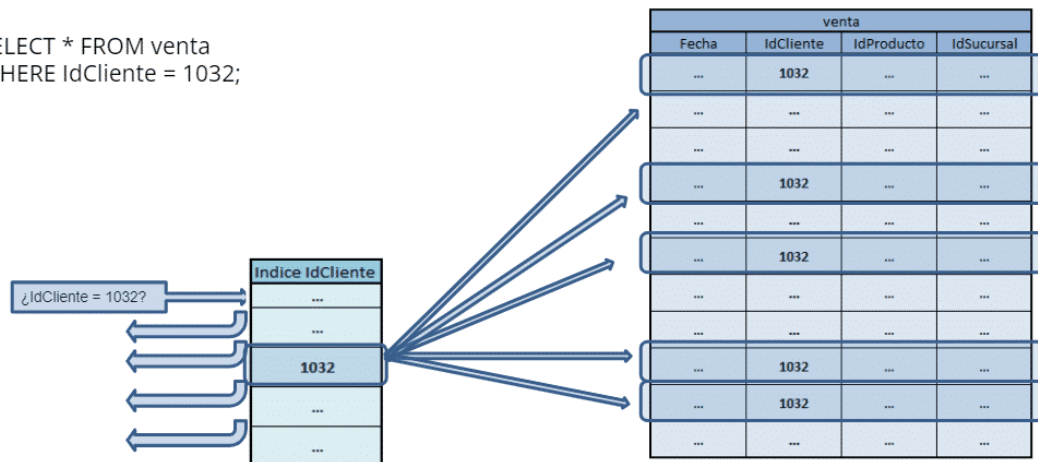
SELECT * FROM venta
WHERE IdCliente = 1032;



SELECT * FROM venta
WHERE IdCliente = 1032;



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



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

```
S DROP INDEX ,bb1w4k4, ON cji6uf62?
J DROP INDEX ,2ednuqo~abefj1qo, ON cji6uf62
```

En 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.

Funciones envueltas alrededor la cláusula WHERE

Un tema en la optimización es un enfoque constante en la cláusula WHERE. ¡Cuánto 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.

#Utilizamos IN para crear un conjunto en la segmentación

```
SELECT *FROM instructoresWHERE 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
```

The screenshot shows the 'Query Statistics' window for a query using the IN operator. The statistics are as follows:

- Timing (as measured at client side):**
Execution time: 0:00:0.000000000
- Timing (as measured by the server):**
Execution time: 0:00:0.00042100
Table lock wait time: 0:00:0.00000400
- Errors:**
Had Errors: NO
Warnings: 0
- Rows Processed:**
Rows affected: 0
Rows sent to client: 5
Rows examined: 5
- Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0
- Joins per Type:**
Full table scans (Select_scan): 0
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): 1
- 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
- Index Usage:**
At least one Index was used
- Other Info:**
Event Id: 98
Thread Id: 69

The right sidebar shows the 'Query Stats' tab is selected, with other options like 'Result Grid', 'Form Editor', 'Field Types', and 'Execution Plan'.

IN

The screenshot shows the 'Query Statistics' window for a query using the OR operator. The statistics are as follows:

- Timing (as measured at client side):**
Execution time: 0:00:0.000000000
- Timing (as measured by the server):**
Execution time: 0:00:0.00031930
Table lock wait time: 0:00:0.00000300
- Errors:**
Had Errors: NO
Warnings: 0
- Rows Processed:**
Rows affected: 0
Rows sent to client: 5
Rows examined: 5
- Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0
- Joins per Type:**
Full table scans (Select_scan): 0
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): 1
- 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
- Index Usage:**
At least one Index was used
- Other Info:**
Event Id: 101
Thread Id: 69

The right sidebar shows the 'Query Stats' tab is selected, with other options like 'Result Grid', 'Form Editor', 'Field Types', and 'Execution Plan'.

OR

Formas Normales

Las **formas normales** son conjuntos de criterios que utilizamos para diseñar la estructura de las bases de datos. Para mejorar el desempeño de una base de datos, así como evitar redundancia en la información que contiene y, en consecuencia, generar condiciones para un mejor diseño, se deben conocer las formas de normalización y condiciones en las que la desnormalización es recomendable.

Primera Forma Normal (1FN)

Una relación se encuentra en 1FN sólo si cada uno de sus atributos contiene un único valor para un registro determinado.

Supongamos que guardamos las sucursales donde compraron los clientes considerando el diseño de la imagen.

Podemos observar, que en el mismo registro se guardan todas las sucursales asociadas al cliente, motivo por el cual no se cumple la 1FN.

Codigo	Nombre	Apellido	Sucursal
3467	Daniel Arnoldo	Abib	Avellaneda, Caseros y Córdoba Quiroz
3243	Hugo Briz	Abilio	Quilmes y Velez
3124	Jaime Gaston	Acevedo Salinas	Caballito, Caseros y Quilmes

Para corregirlo, se crean dos tablas, donde vamos a poder ver que cada registro guarda un solo valor. De esta manera, el esquema cumple la 1FN.

Codigo	Nombre	Apellido
3467	Daniel Arnoldo	Abib
3243	Hugo Briz	Abilio
3124	Jaime Gaston	Acevedo Salinas

Codigo	Sucursal
3467	Avellaneda
3467	Caseros
3467	Córdoba Quiroz
3243	Quilmes
3243	Velez
3124	Caballito
3124	Cabildo
3124	Mendoza
3124	Moron

Segunda Forma Normal (2FN)

Una relación se encuentra en 2FN sólo si se cumple 1FN y todos sus atributos no clave dependen en forma completa de la clave.

Supongamos que tenemos una tabla donde guardamos cuántas ventas se hizo a cada cliente en cada sucursal, y contamos con un esquema como el de la imagen.

La clave de esta tabla, está formada por los campos Codigo_Cliente y Codigo_Sucursal y la relación se encuentra en 1FN, pero:

- Apellido_Nombre sólo depende de Codigo_Cliente.
- Sucursal sólo depende de Codigo_Sucursal.
- Ventas depende de la clave completa Codigo_Cliente + Codigo_Sucursal.

Por lo que ocurre en 1 y 2 no se cumple con la 2FN.

Codigo_Cliente	Codigo_Sucursal	Apellido_Nombre	Sucursal	Ventas
3467	101	Abib, Daniel Arnoldo	Avellaneda	3
3467	103	Abib, Daniel Arnoldo	Caseros	2
3467	106	Abib, Daniel Arnoldo	Córdoba Quiroz	5
3243	107	Abilio, Hugo Briz	Quilmes	6
3243	110	Abilio, Hugo Briz	Velez	2
3124	109	Acevedo Salinas, Jaime Gaston	Caballito	3
3124	103	Acevedo Salinas, Jaime Gaston	Caseros	1
3124	107	Acevedo Salinas, Jaime Gaston	Quilmes	2

Para corregirlo, debemos llegar a un esquema de 3 tablas como el que se puede observar. De esta manera sí se cumple la 2FN para el esquema.

Codigo_Cliente	Apellido_Nombre
3467	Abib, Daniel Arnoldo
3243	Abilio, Hugo Briz
3124	Acevedo Salinas, Jaime Gaston

Codigo_Cliente	Codigo_Sucursal	Ventas
3467	101	3
3467	103	2
3467	106	5
3243	107	6
3243	110	2
3124	109	3
3124	103	1
3124	107	2

Codigo_Sucursal	Sucursal
101	Avellaneda
103	Caseros
106	Córdoba Quiroz
107	Quilmes
110	Velez
109	Caballito
103	Caseros
107	Quilmes

Tercera Forma Normal (3FN)

Una relación se encuentra en 3FN sólo si se cumple 2FN y los campos no clave dependen únicamente de la clave o los campos no clave no dependen unos de otros.

Supongamos que tenemos una tabla donde guardamos datos filiatorios de clientes que tienen que ver con la Localidad y Provincia en que viven y tenemos la estructura de la imagen.

Como se puede observar, surgen las siguientes dependencias:

- Codigo_Cliente Apellido_Nombre.
- Codigo_Cliente Localidad.
- Codigo_Cliente Provincia.

Aunque cumple con la 2FN, la Provincia está también ligada a la Localidad, con lo que no se cumple la 3FN.

Codigo_Cliente	Apellido_Nombre	Localidad	Provincia
3467	Abib, Daniel Arnoldo	Cordoba	Cordoba
3243	Abilio, Hugo Briz	Quilmes	Buenos Aires
3124	Acevedo Salinas, Jaime Gaston	CABA	Buenos Aires

Para corregirlo, debemos llegar a un esquema de 2 tablas como el que se puede observar. De esta manera se cumple la 3FN para el esquema.

Codigo_Cliente	Apellido_Nombre	Localidad
3467	Abib, Daniel Arnoldo	Cordoba
3243	Abilio, Hugo Briz	Quilmes
3124	Acevedo Salinas, Jaime Gaston	CABA

Localidad	Provincia
Cordoba	Cordoba
Quilmes	Buenos Aires
CABA	Buenos Aires

Cuarta Forma Normal (4FN)

Una relación se encuentra en 4FN sólo si se cumple 3FN y no posee dependencias multivaluadas no triviales.

Supongamos que tenemos un esquema como el de la imagen, donde guardamos que una sucursal vende un determinado producto mediante un canal de venta.

Notemos que debido a que la tabla tiene una clave única y ningún atributo no clave, no viola ninguna forma normal hasta la 3FN. Pero debido a que los canales de venta de una sucursal son independientes de los productos que vende, hay redundancia en la tabla: por ejemplo vemos tres veces que Caballito vende OnLine. Esto se describe como que Canal de Venta está teniendo una dependencia multivalor en Sucursal e impide que se cumpla con la 4FN en la relación.

Sucursal	Canal de Venta	Producto
Avellaneda	OnLine	Parlante Jbl Go Blue Bluetooth
Avellaneda	Presencial	Parlante Jbl Go Blue Bluetooth
Caseros	Presencial	Parlante Kingta So-101 Bluetooth/ Waterproof Red
Caseros	Presencial	Parlante Jbl Go Blue Bluetooth
Caballito	OnLine	Parlante Jbl Go Blue Bluetooth
Caballito	OnLine	Parlante Kingta So-101 Bluetooth/ Waterproof Red
Caballito	OnLine	Parlante Jbl Flip 4 Gray Bluetooth

Para corregirlo, debemos poner los hechos sobre los Canales de Ventas por los que se vende en una tabla diferente a los hechos sobre los productos que se vende. De esta manera se cumple la 4FN para el esquema.

Sucursal	Canal de Venta
Avellaneda	OnLine
Avellaneda	Presencial
Caseros	Presencial
Caballito	OnLine

Sucursal	Producto
Avellaneda	Parlante Jbl Go Blue Bluetooth
Caseros	Parlante Kingta So-101 Bluetooth/ Waterproof Red
Caseros	Parlante Jbl Go Blue Bluetooth
Caballito	Parlante Jbl Go Blue Bluetooth
Caballito	Parlante Kingta So-101 Bluetooth/ Waterproof Red
Caballito	Parlante Jbl Flip 4 Gray Bluetooth

Quinta Forma Normal (5FN)

Una relación se encuentra en 5FN sólo si se cumple 4FN y cada dependencia de unión en ella es implicada por las claves candidatas.

Siguiendo con el ejemplo anterior, la relación también será válida para la 5FN si existe una regla en el escenario real que limite una relación de una Canal de Venta con un Producto al no ser vendido ese Producto por la Sucursal.

El hecho de que no haya forma de limitar las combinaciones inválidas del mundo real, limita el cumplimiento de la 5FN.

Sucursal	Canal de Venta	Producto
Avellaneda	OnLine	Parlante Jbl Go Blue Bluetooth
Avellaneda	Presencial	Parlante Jbl Go Blue Bluetooth
Caseros	Presencial	Parlante Kingta So-101 Bluetooth/ Waterproof Red
Caseros	Presencial	Parlante Jbl Go Blue Bluetooth
Caballito	OnLine	Parlante Jbl Go Blue Bluetooth
Caballito	OnLine	Parlante Kingta So-101 Bluetooth/ Waterproof Red
Caballito	OnLine	Parlante Jbl Flip 4 Gray Bluetooth

Para corregirlo, además de poner en tablas diferentes, la relaciones posibles Sucursal-Canal de Venta y Sucursal-Producto, debemos hacer lo propio con la relación Canal de Venta-Producto. De esta manera se cumple la 5FN para el esquema.

Sucursal	Producto
Avellaneda	Parlante Jbl Go Blue Bluetooth
Caseros	Parlante Kingta So-101 Bluetooth/ Waterproof Red
Caseros	Parlante Jbl Go Blue Bluetooth
Caballito	Parlante Jbl Go Blue Bluetooth
Caballito	Parlante Kingta So-101 Bluetooth/ Waterproof Red
Caballito	Parlante Jbl Flip 4 Gray Bluetooth

Sucursal	Canal de Venta
Avellaneda	OnLine
Avellaneda	Presencial
Caseros	Presencial
Caballito	OnLine

Canal de Venta	Producto
OnLine	Parlante Jbl Go Blue Bluetooth
Presencial	Parlante Jbl Go Blue Bluetooth
Presencial	Parlante Kingta So-101 Bluetooth/ Waterproof Red
OnLine	Parlante Kingta So-101 Bluetooth/ Waterproof Red
OnLine	Parlante Jbl Flip 4 Gray Bluetooth

Modelos de Datos

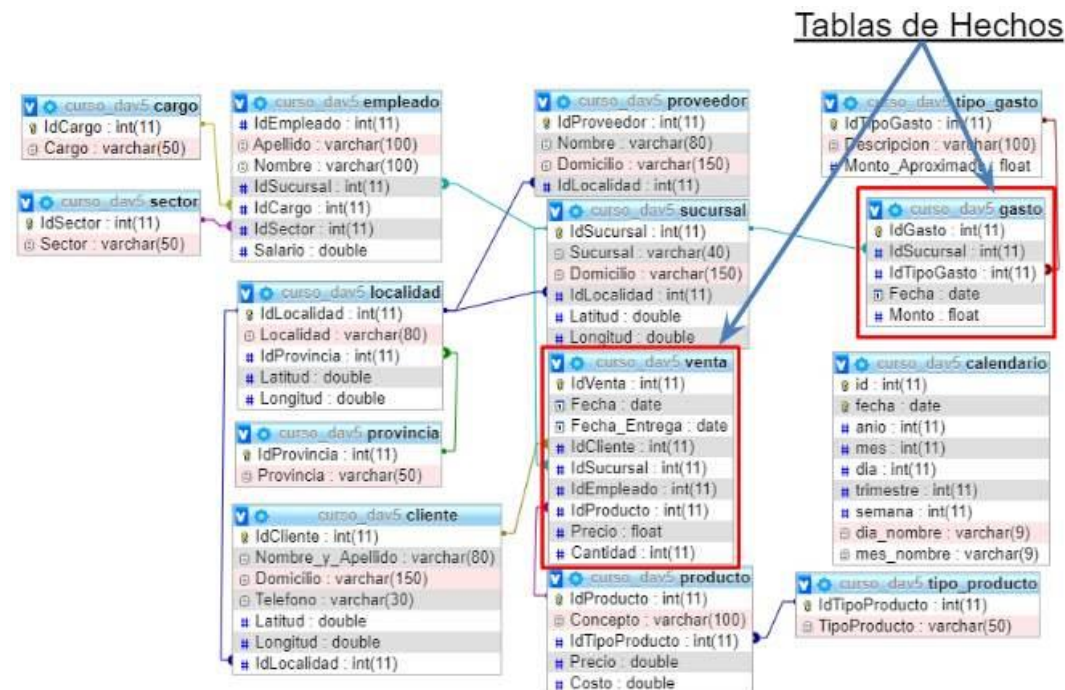
Los modelos de datos buscan representar un realidad que es posible representar mediante las entidades que la conforman. Esas entidades quedan representadas en tablas, y pueden ser de dos tipos:

Tablas de hechos

—

Registran las operaciones ocurridas, todo tipo de transacciones donde intervienen las diferentes entidades del modelo. Por lo general están confeccionadas por campos que refieren a un

momento en el tiempo, campos que refieren a las entidades correspondientes y campos donde se guardan valores numéricos, que luego, pasan a ser métricas. Un ejemplo pueden ser las tablas de ventas, donde queda reflejado día y hora de la transacción, cliente, producto, sucursal como entidades, representadas por claves foráneas, y por último, los valores de precio y cantidad de unidades vendidas.

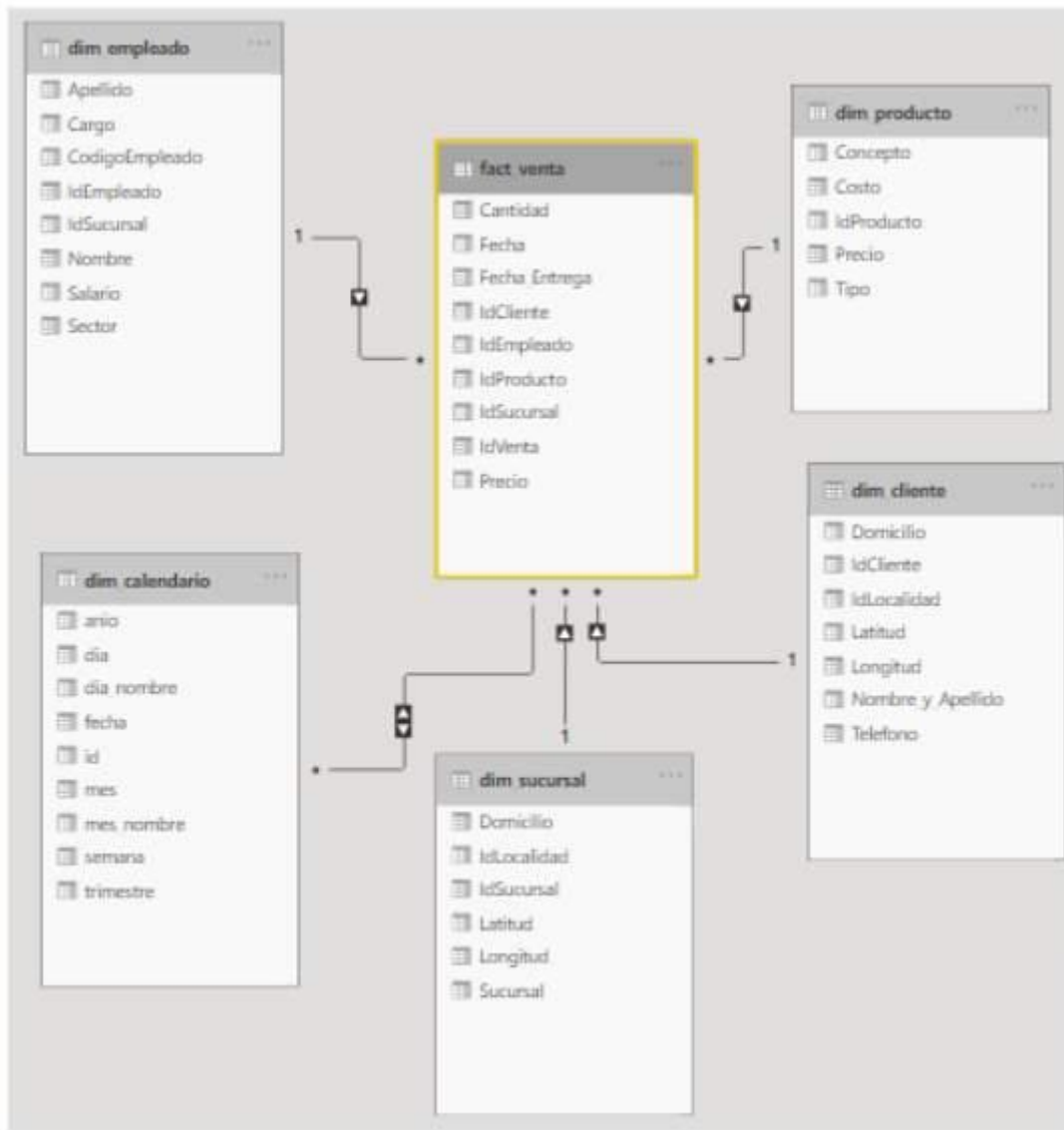


Tablas maestros

Registran los atributos particulares de las entidades, representando cada registro a una única instancia, por medio de una clave primaria. Un ejemplo puede ser la tabla de maestro de clientes, donde cada registro posee un identificador único y representa los datos respectivos de un único cliente.

Modelos de Estrella

Es muy común encontrar en herramientas de BI, modelos de este tipo, donde las tablas de hecho son centrales, mientras que las tablas de maestros, que también son denominadas de dimensiones, están alrededor.



La nomenclatura típica, es anteponer “Fact_” a las tablas de hechos, y “Dim_” a las tablas de dimensiones o maestros.

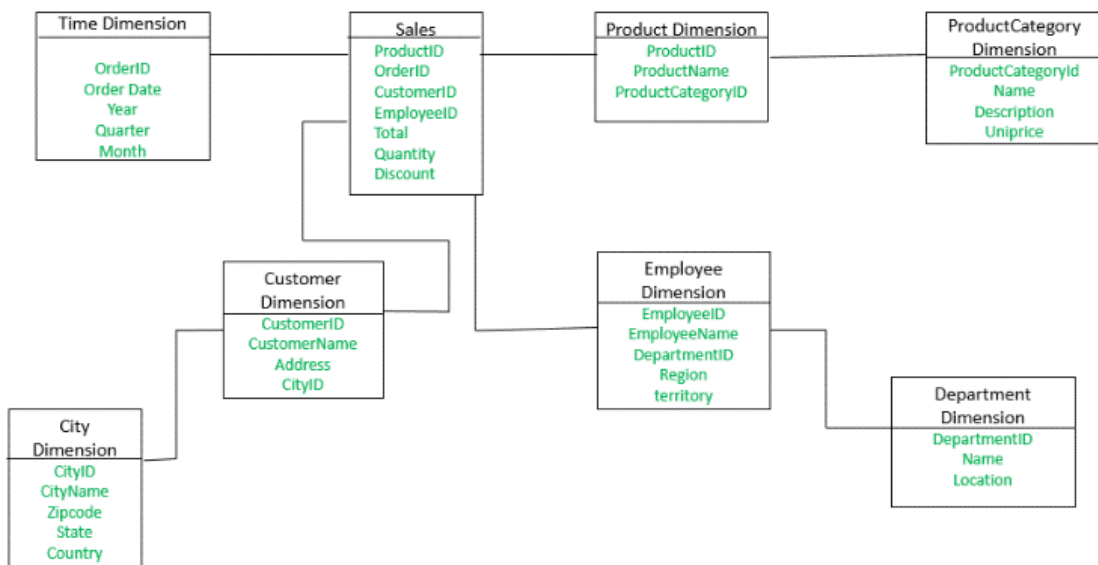
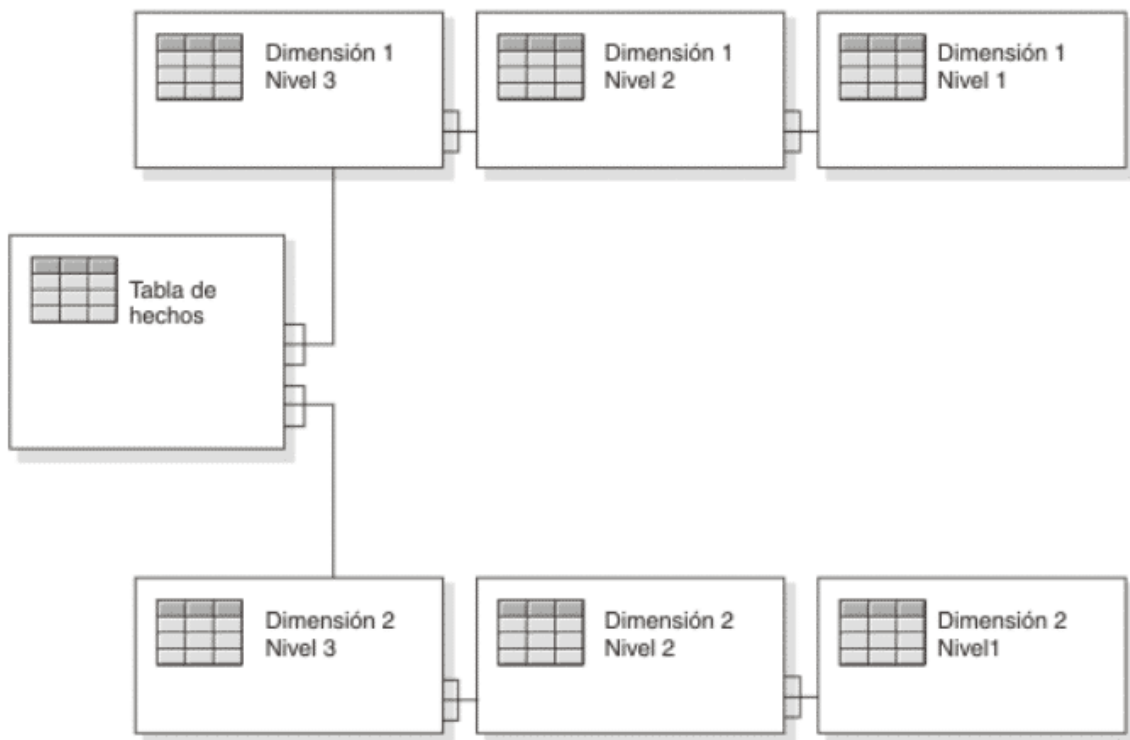
Por su forma de trabajar filtrando la información en base a selecciones, en las herramientas de BI no se utilizan referencias circulares, ya que, al filtrar por una dimensión, el filtro se aplica en cascada sobre la tabla de hechos y las dimensiones asociadas a esos registros.

Modelos de Copo de Nieve

El esquema de copo de nieve consta de una tabla de hechos que está conectada a muchas tablas de dimensiones, que pueden estar conectadas a otras tablas de dimensiones a través de una relación de muchos a uno. Las tablas de un esquema de copo de nieve generalmente se normalizan en la tercera forma normal. Cada tabla de dimensiones representa exactamente un nivel en una jerarquía.

En la siguiente figura se muestra un esquema de copo de nieve con dos dimensiones, cada una con tres niveles. Un esquema de copo de nieve puede tener varias dimensiones y cada dimensión puede tener varios niveles.

Esquema de copo de nieve



- La tabla de dimensiones de Empleado contiene los atributos: EmployeeID, EmployeeName, DepartmentID, Region, Territory.
- El atributo DepartmentID se vincula con la tabla Empleado con la tabla de dimensiones Departamento.
- La dimensión Departamento se utiliza para proporcionar detalles sobre cada departamento, como el Nombre y la Ubicación del departamento.

- La tabla de dimensiones del cliente ahora contiene los atributos: CustomerID, CustomerName, Address, CityID. Los atributos CityID vinculan la tabla de dimensiones del cliente con la tabla de dimensiones de la ciudad.
- La tabla de dimensiones de la ciudad tiene detalles sobre cada ciudad, como el nombre de la ciudad, el código postal, el estado y el país.

La principal **diferencia** entre el esquema de estrella y el esquema de copo de nieve es que la tabla de dimensiones del esquema de copo de nieve se mantiene en forma normalizada para reducir la redundancia.

La **ventaja** aquí es que tales tablas (normalizadas) son fáciles de mantener y ahorran espacio de almacenamiento. Sin embargo, también significa que se necesitarán más combinaciones para ejecutar la consulta. Esto afectará negativamente al rendimiento del sistema.

Conclusión

La optimización del rendimiento de una base de datos no solo se limita a crear consultas performantes, 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 más performante de escribirlas.

Extracción, Transformación y Carga - ETL

Durante las clases anteriores 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 una 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 ultima carga, sin embargo depende de diferentes factores.

1. Reverso de la tarjeta

Orígenes de datos

Hay veces que la naturaleza del origen de datos imposibilita determinar cuales han sido las últimas modificaciones con respecto a la última carga.

Haz clic para voltear

2. Reverso de la tarjeta

Volumen de datos

Si el volumen de datos es pequeño es posible que no se precise de un Mart, por consiguiente la carga seria Full. Si el volumen es muy grande el tiempo y costo de procesamiento podría ser muy elevado para una carga Full.

3. Reverso de la tarjeta

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 seria válida y mas fácil de desarrollar.

4. Reverso de la tarjeta

Niveles de Servicio

En algunos entornos en la nube, se paga por lo 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:

--¿Como 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 como 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

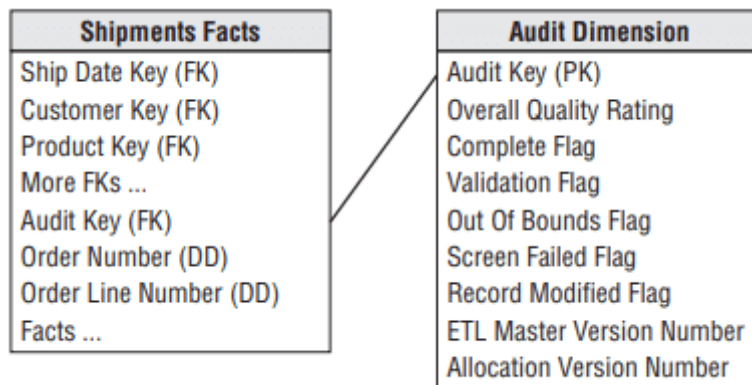
El estado de sus datos depende de qué tan bien los profile. Hemos abordado este concepto mediante los outliers, ahora retomaremos nuevamente este concepto. La **elaboración de perfiles de datos** es el proceso de examinar, analizar y crear resúmenes útiles de datos. El proceso produce una visión general 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 de su 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 auditoria



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!

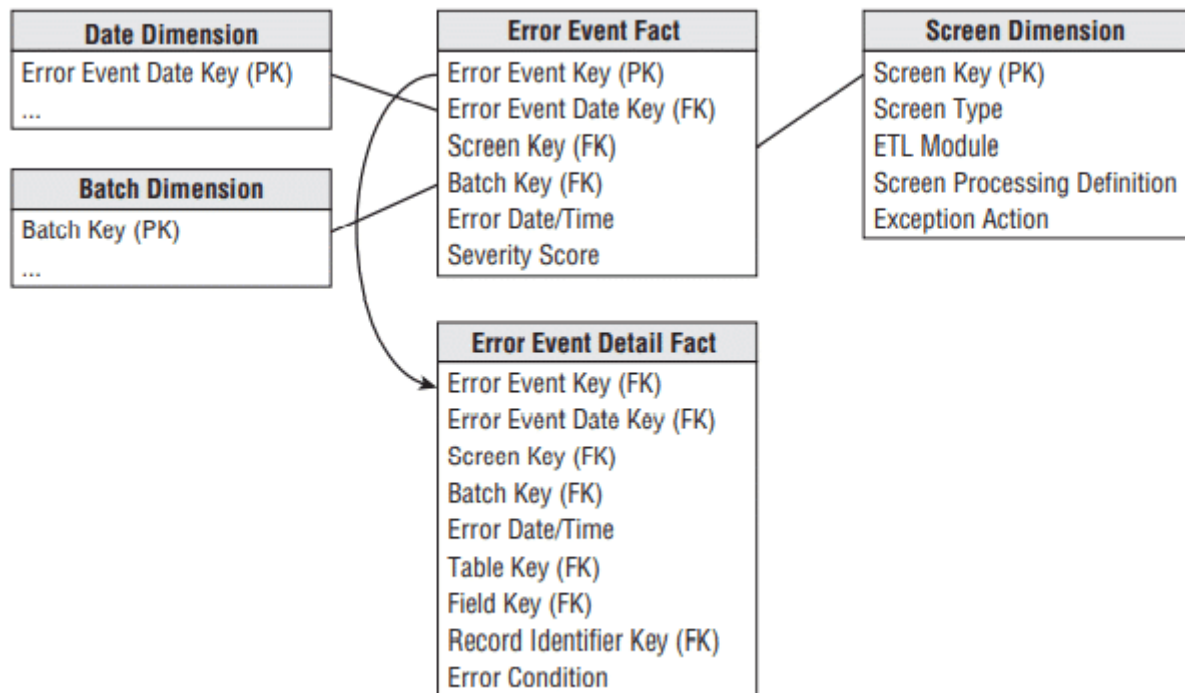
Para visualizar cómo se crean las filas de dimensión de auditoría, imagine esta tabla de hechos de envíos se actualiza una vez al día a partir de un archivo por lotes. Supongamos que hoy Tiene una carga perfecta sin errores marcados.

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.

Filtrado de las filas de un DataFrame

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.

```
CREATE TABLE alumno (cedulaIdentidad INT NOT NULL AUTO_INCREMENT,nombre
VARCHAR(20),apellido VARCHAR(20),fechaInicio 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),fechaInicio_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,
```

```
fechaInicio_auditoria,
```

```
usuario,
```

fecha)

VALUES (NEW.cedulaIdentidad,

NEW.nombre,

NEW.apellido,

NEW.fechaInicio,

CURRENT_USER,

NOW());