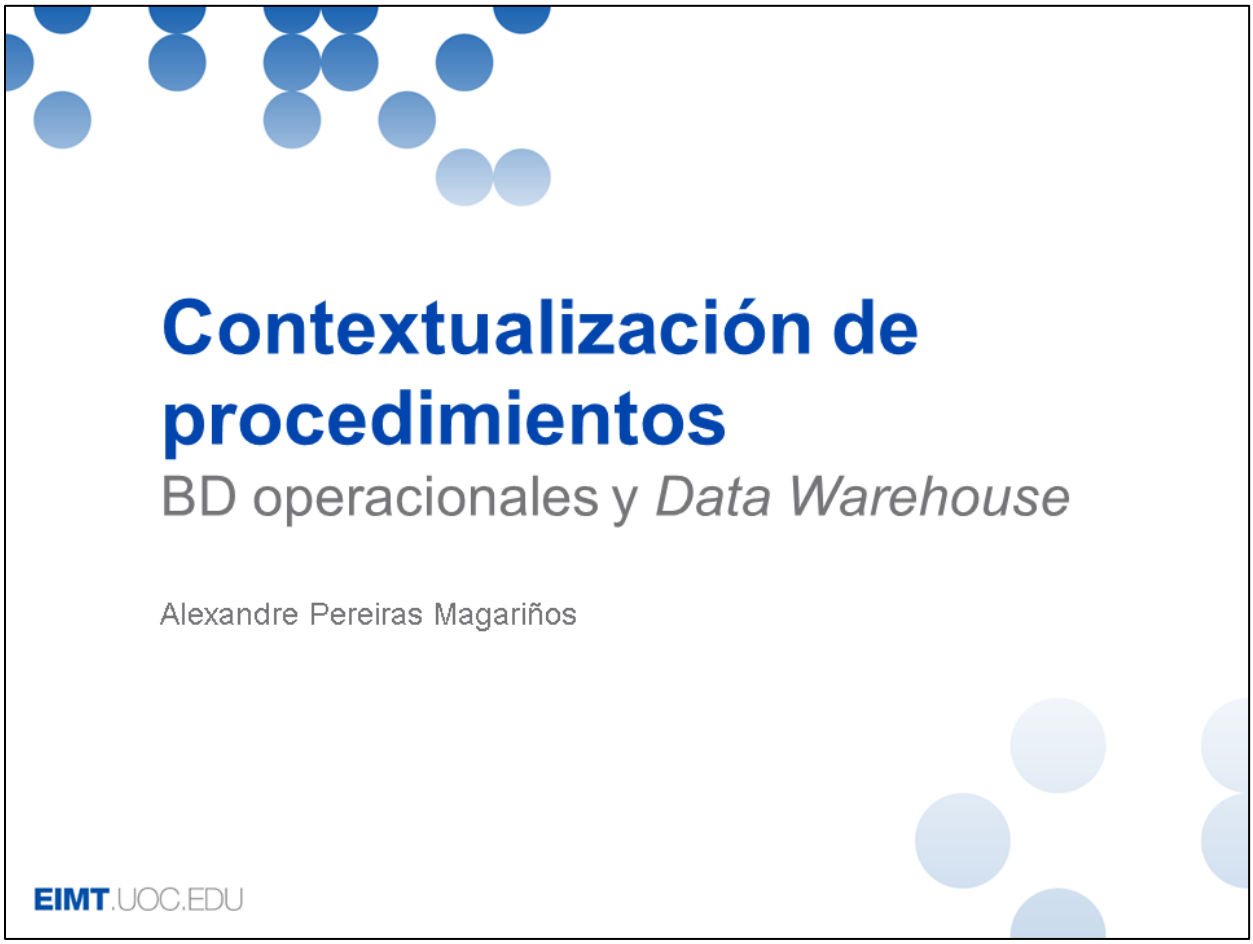
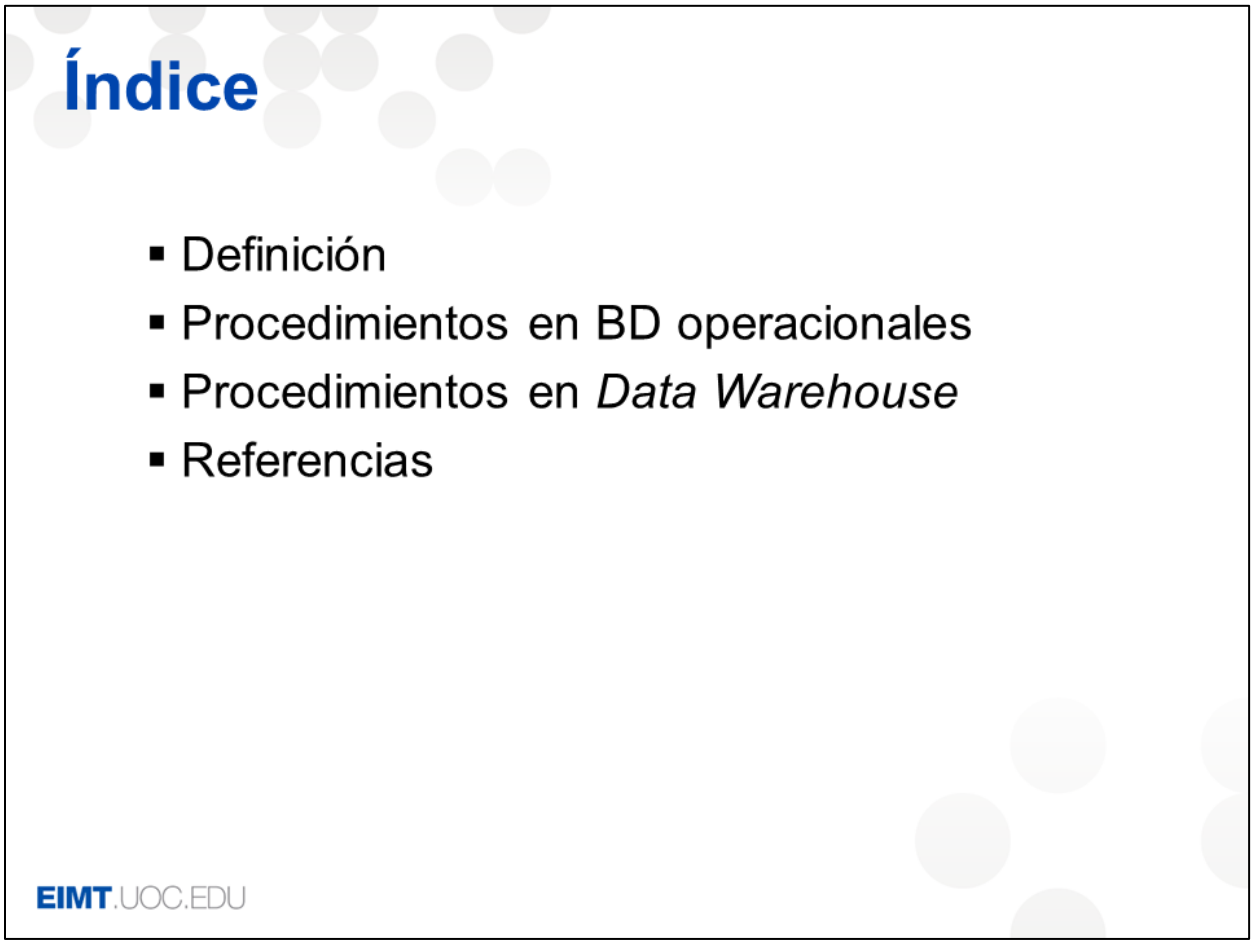
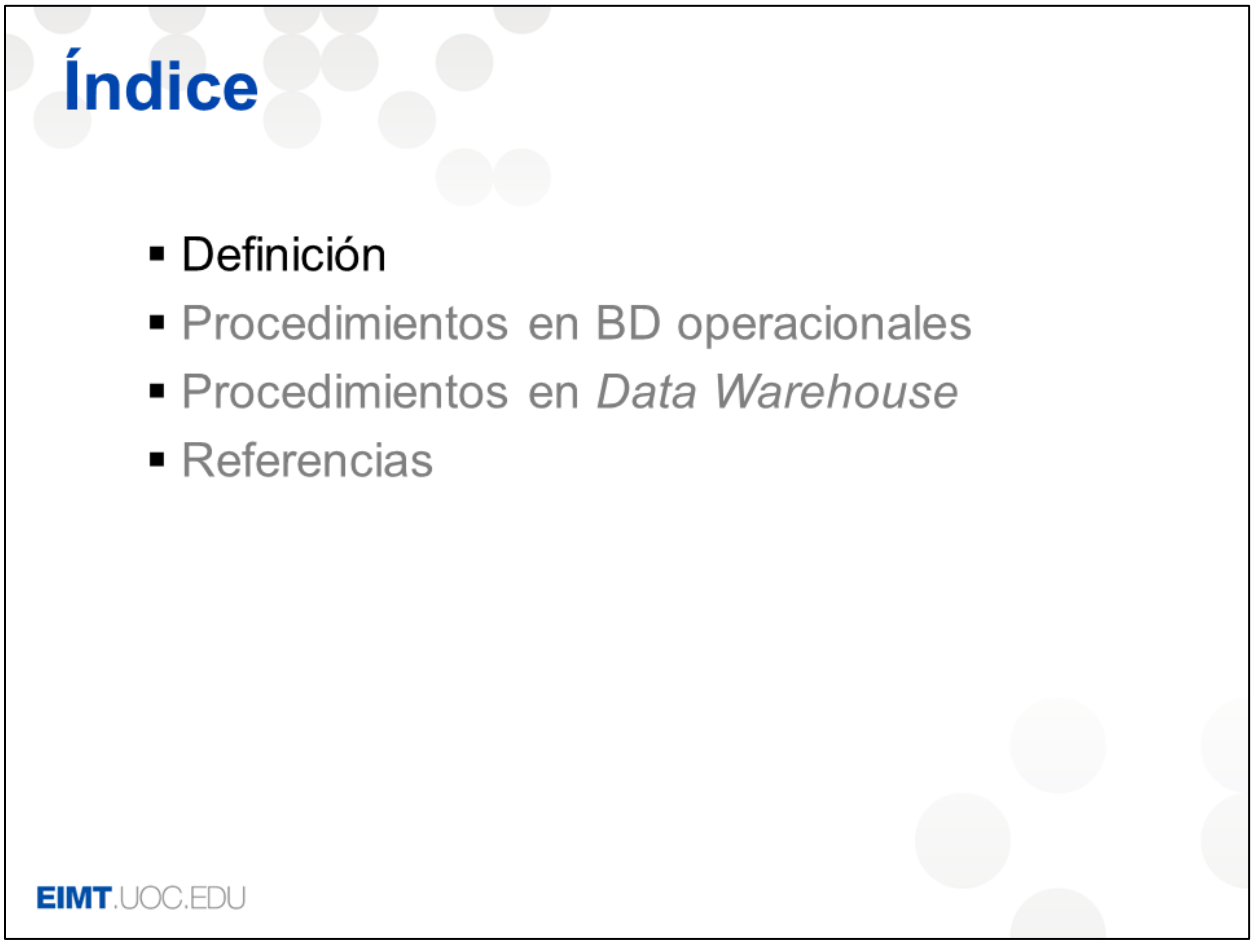


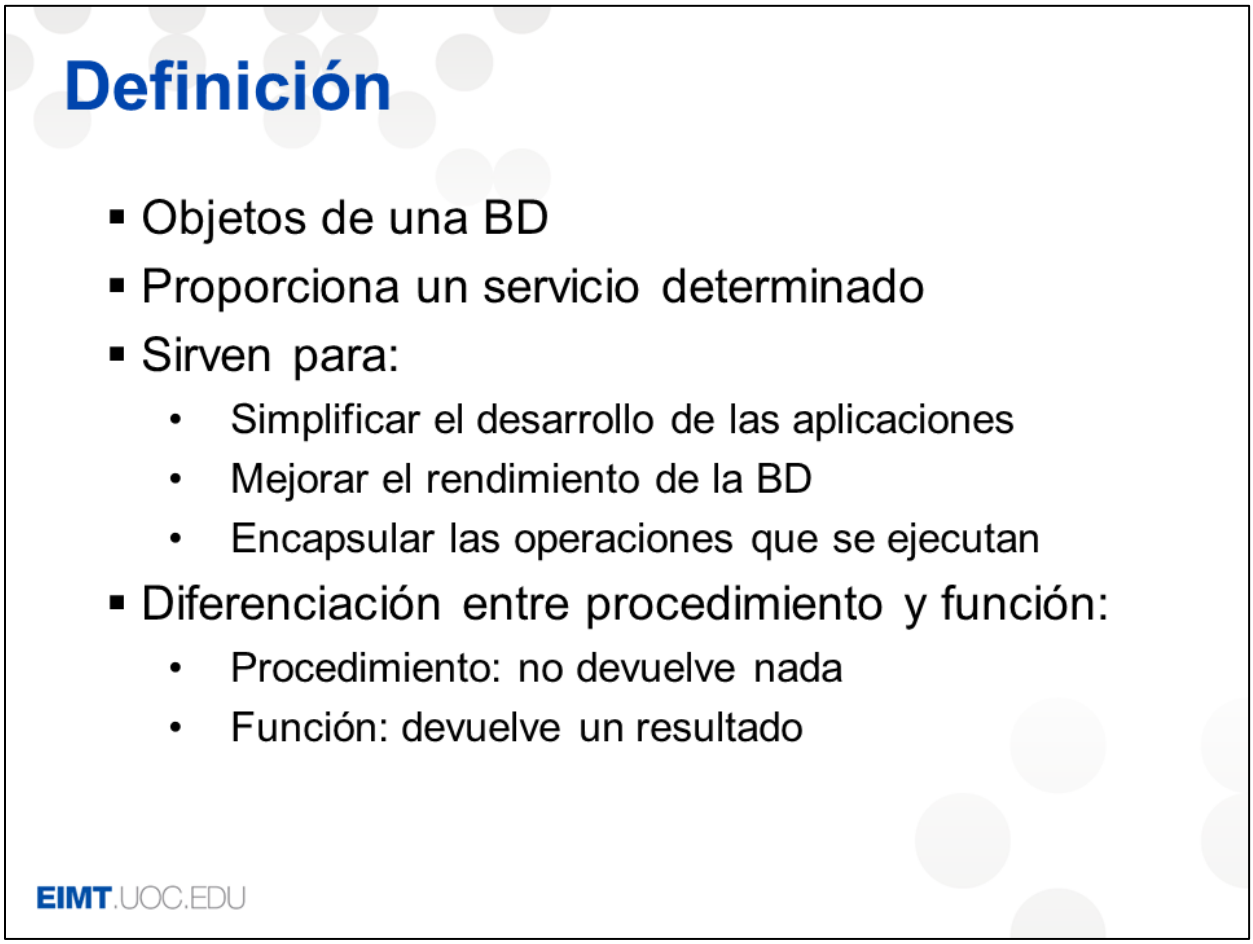

Bienvenidos a este vídeo titulado *Contextualización de procedimientos* de la asignatura de *Bases de datos para Data Warehouse*, en el que comentaremos y explicaremos qué son los procedimientos, y cómo podemos aplicarlos dentro del marco de las bases de datos (BD) operacionales y del *Data Warehouse*.




Este vídeo introducirá brevemente el concepto de procedimiento almacenado, para después proporcionar su contextualización desde un punto de vista de las BD operacionales y del *Data Warehouse*, revisando los usos que estos componentes nos pueden proporcionar en ambos tipos de entornos. Desde el punto de vista del *Data Warehouse*, veremos también una serie de ejemplos de la gestión de transacciones y errores. Por último, se proporcionarán las referencias bibliográficas que se han utilizado para elaborar esta presentación.




Vamos pues a comenzar con la definición de procedimiento almacenado y sus características.




Los procedimientos almacenados son objetos de la base de datos, definidos por los usuarios, y que encapsulan un código concreto, proporcionándoles así un servicio determinado. Estos objetos se guardan en la BD y pueden ser invocados en cualquier momento por aquellos usuarios que tengan acceso.

El uso de procedimientos almacenados tiene una serie de objetivos:

- En primer lugar, nos sirven para simplificar el desarrollo de las aplicaciones, de forma que éstas solamente tendrán que realizar una llamada al procedimiento almacenado, en lugar de ejecutar diferentes sentencias SQL una detrás de otra. Además, la lógica relativa a la BD se almacenará en el sistema gestor de bases de datos (SGBD), por lo que se favorece la reutilización del código.

- Otro de los objetivos del uso de procedimientos es que mejora el rendimiento de la BD. Hemos mencionado anteriormente el caso en el que se ejecutaban sentencias SQL una detrás de otra. La ejecución de esta secuencia provoca que la BD tenga que calcular la estrategia de ejecución de cada una de ellas justo antes de ejecutarlas, mientras que en el caso de los procedimientos almacenados, esta estrategia se calcula a la hora de crearlos en la BD.

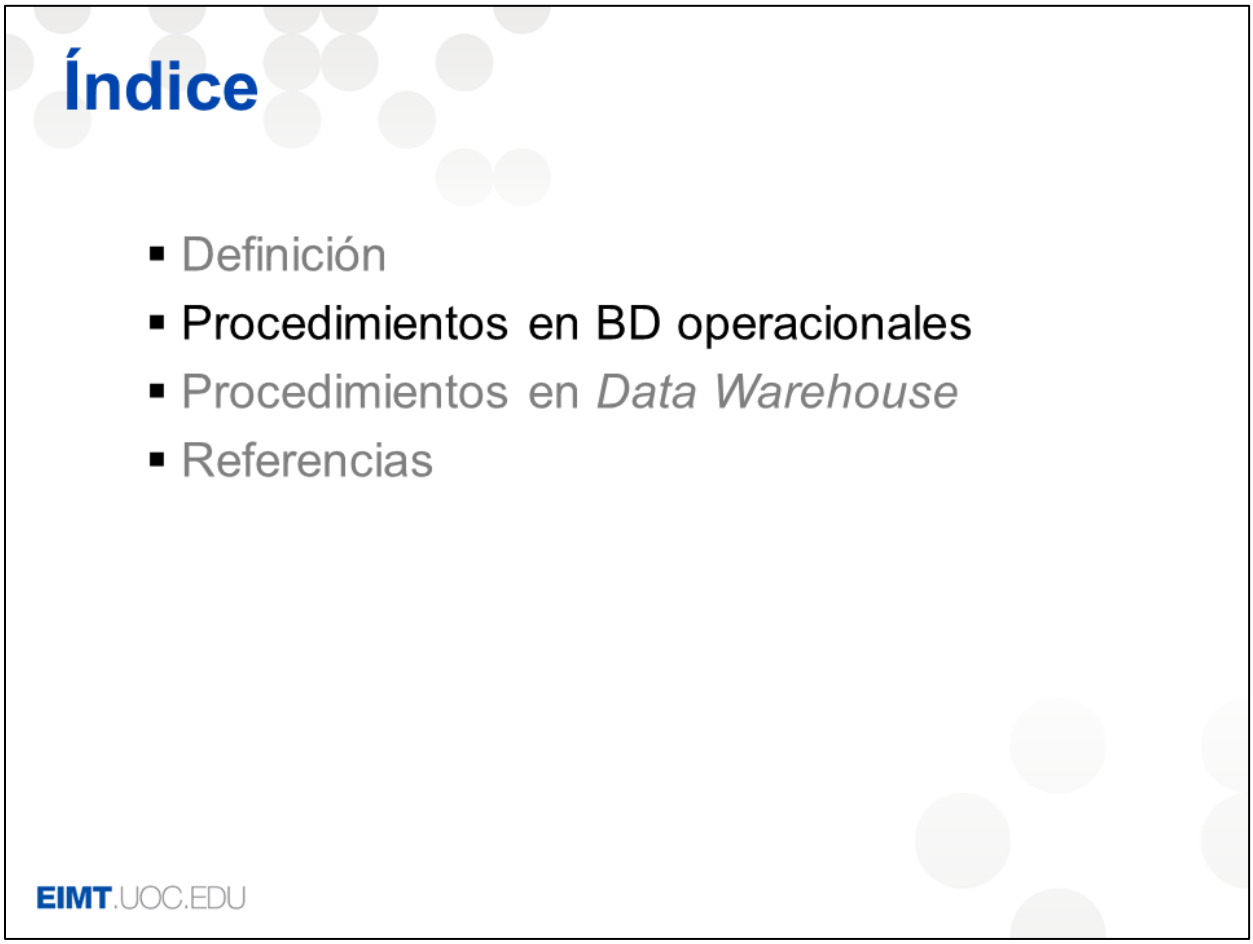
- Por último, es importante destacar que el uso de procedimientos almacenados posibilita la encapsulación de código de las operaciones que se ejecutan. De esta forma, el usuario solamente necesita conocer el nombre del procedimiento que le proporciona el servicio, evitando así que el usuario tenga que conocer las sentencias SQL que el procedimiento almacenado incorpora y los elementos de la BD que manipulan dichas sentencias.

La codificación de estos procedimientos se suele realizar mediante SQL, complementado con un lenguaje procedimental. Por ejemplo, en PostgreSQL, los dos lenguajes procedimentales más utilizados son PL/pgSQL y PL/Python.

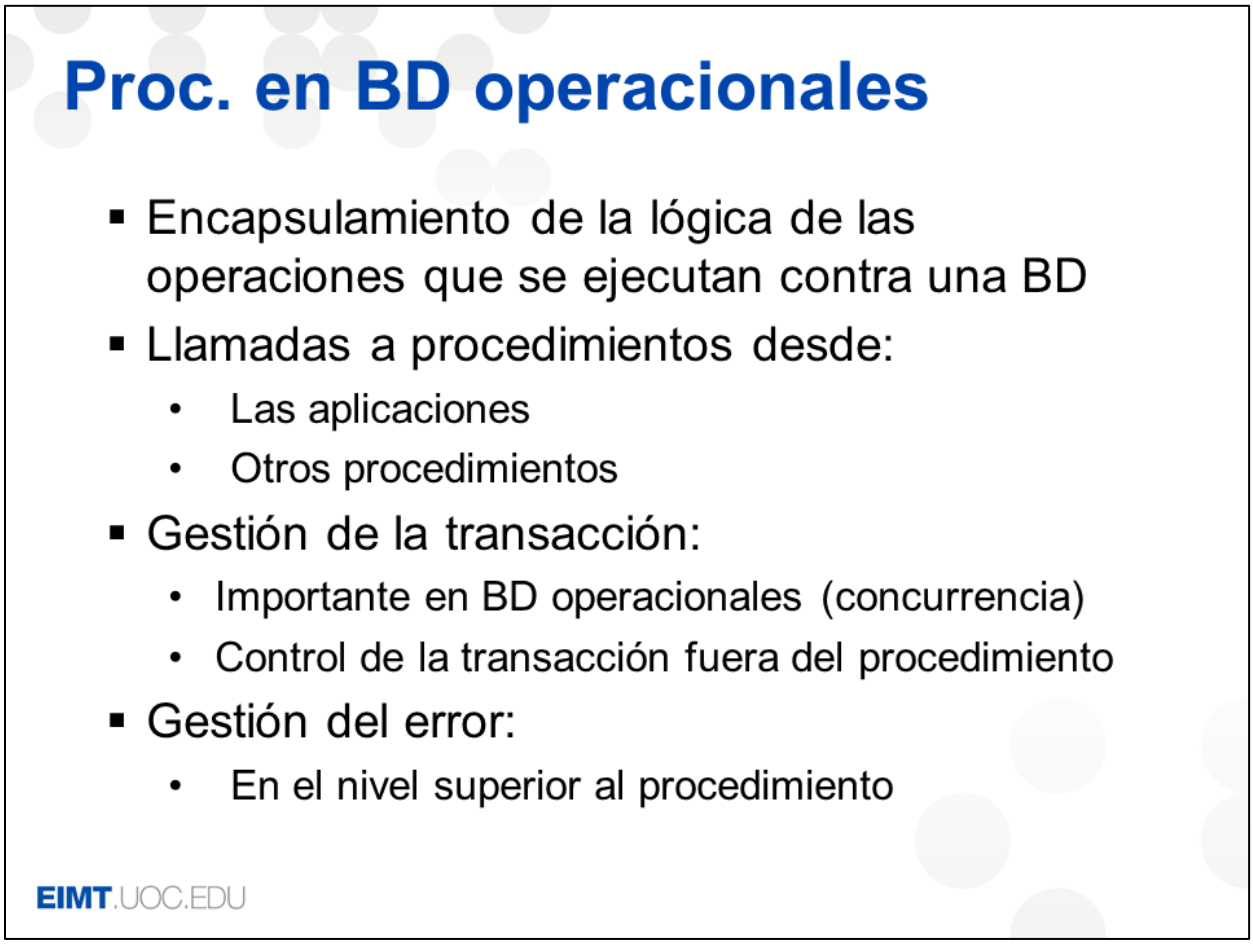
Por último, es importante mencionar que desde el punto de vista de PostgreSQL, a los procedimientos almacenados se les denomina funciones, y pueden o no devolver un resultado. Esta definición difiere con respecto a otros SGBD, como Oracle, en el que ambos tipos requieren de cláusulas SQL concretas (`CREATE PROCEDURE` o `CREATE FUNCTION`, donde el primero no devuelve ningún resultado, y el segundo sí) en oposición a PostgreSQL (`CREATE FUNCTION [...] RETURNS [...]`). Con el fin de diferenciar ambos tipos en PostgreSQL, hemos decidido denominar como **procedimiento** a aquellos procedimientos almacenados que no devuelven ningún resultado, y como **función** aquellos procedimientos almacenados que sí devuelven un resultado.


En la siguiente transparencia podemos ver un ejemplo de procedimiento almacenado en PostgreSQL creado con el lenguaje procedimental PL/pgSQL.

El procedimiento mostrado, denominado `sp_carga_prestamo()`, realiza la carga de datos de préstamos desde una tabla llamada `prestamo_tmp` a otra tabla denominada `prestamo_fact`. El procedimiento recorre fila a fila la tabla `prestamo_tmp` e intenta insertar la fila en la tabla destino. En caso de que la operación de inserción produzca un error de clave única violada (es decir, que la fila ya exista en la tabla), el procedimiento se encargará de actualizar dicha fila concatenando al nombre de libro una cadena de caracteres formada por guiones, además de actualizar el identificador y el autor del libro. Si el error producido no es de clave única violada, entonces el procedimiento no realiza nada (codificado como un bloque vacío con el comentario "*Insertar en log*") y continúa con la siguiente fila, hasta que se procese la última fila de la tabla, momento en el cual el procedimiento finaliza.




A continuación, vamos a ver en qué supuestos podemos usar los procedimientos desde el punto de vista de las BD operacionales.




Dentro del contexto de las BD operacionales, los procedimientos almacenados, como hemos mencionado anteriormente, nos resultan de mucha utilidad para encapsular la lógica de las operaciones que se ejecutan contra una BD. Esta encapsulación permite disminuir el tráfico en la red, ya que se evita que cada sentencia SQL se envíe de una manera individual por la red, y que los resultados de la ejecución de cada una de las sentencias sean recogidos por la aplicación. Con el uso de procedimientos almacenados, toda la lógica se ejecutará en el SGBD sin necesidad de utilizar la red de forma innecesaria, devolviendo los resultados una única vez a la aplicación. De esta forma, los usuarios solamente necesitan saber el nombre del procedimiento que han de ejecutar con el fin de obtener el servicio esperado.

La llamada a un procedimiento puede realizarse de las siguientes formas, además de la opción de llamarlo desde la consola o línea de comandos que se conecta a una BD:

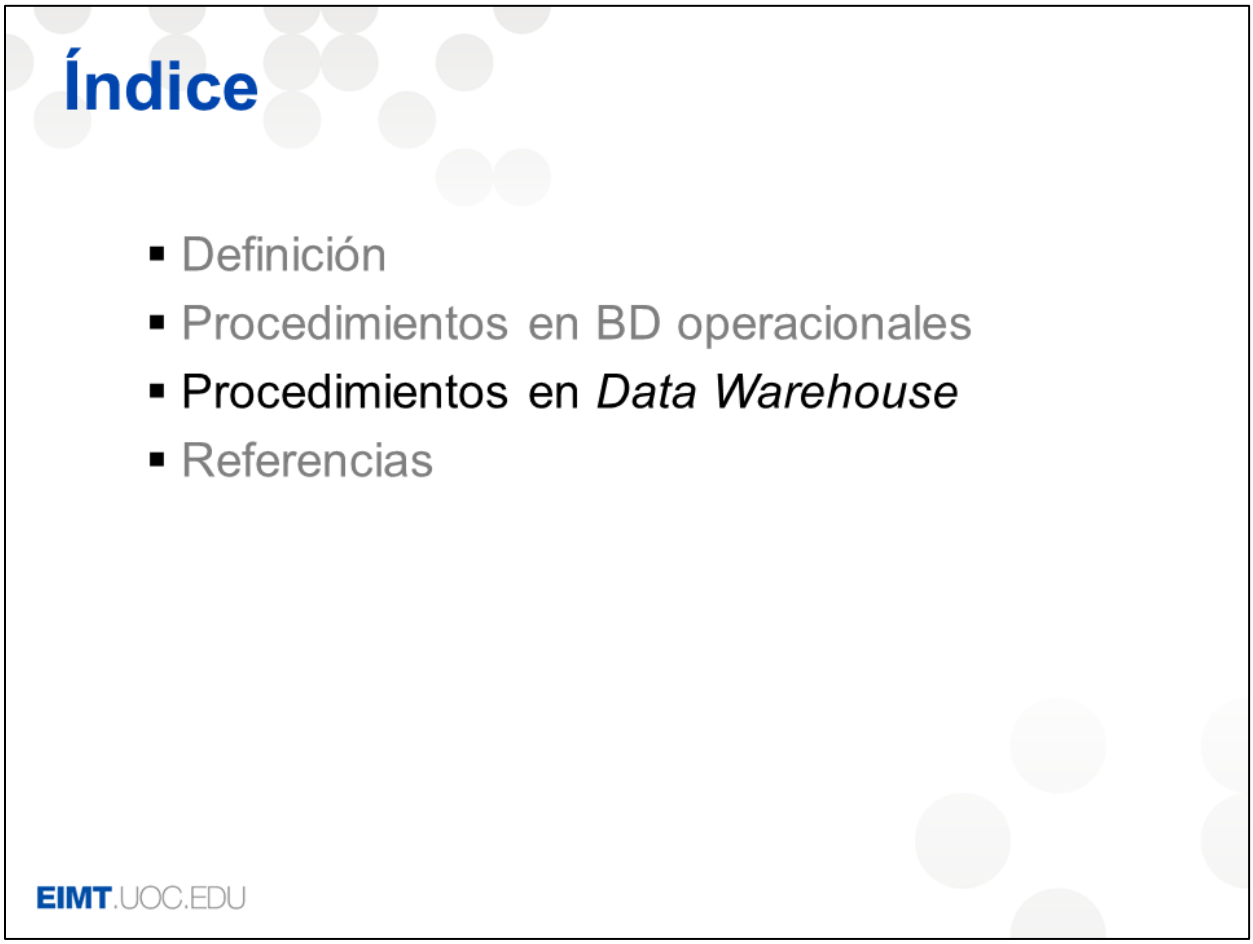
- Llamadas desde las aplicaciones: en estos casos, el código de las aplicaciones ha de establecer una conexión con la base de datos y realizar la llamada al procedimiento almacenado.

- Otros procedimientos: esta opción es común cuando queremos implementar un servicio determinado que sea utilizado por múltiples procedimientos. Un ejemplo de llamada a procedimiento dentro de otro procedimiento sería en el caso de generar información de auditoría

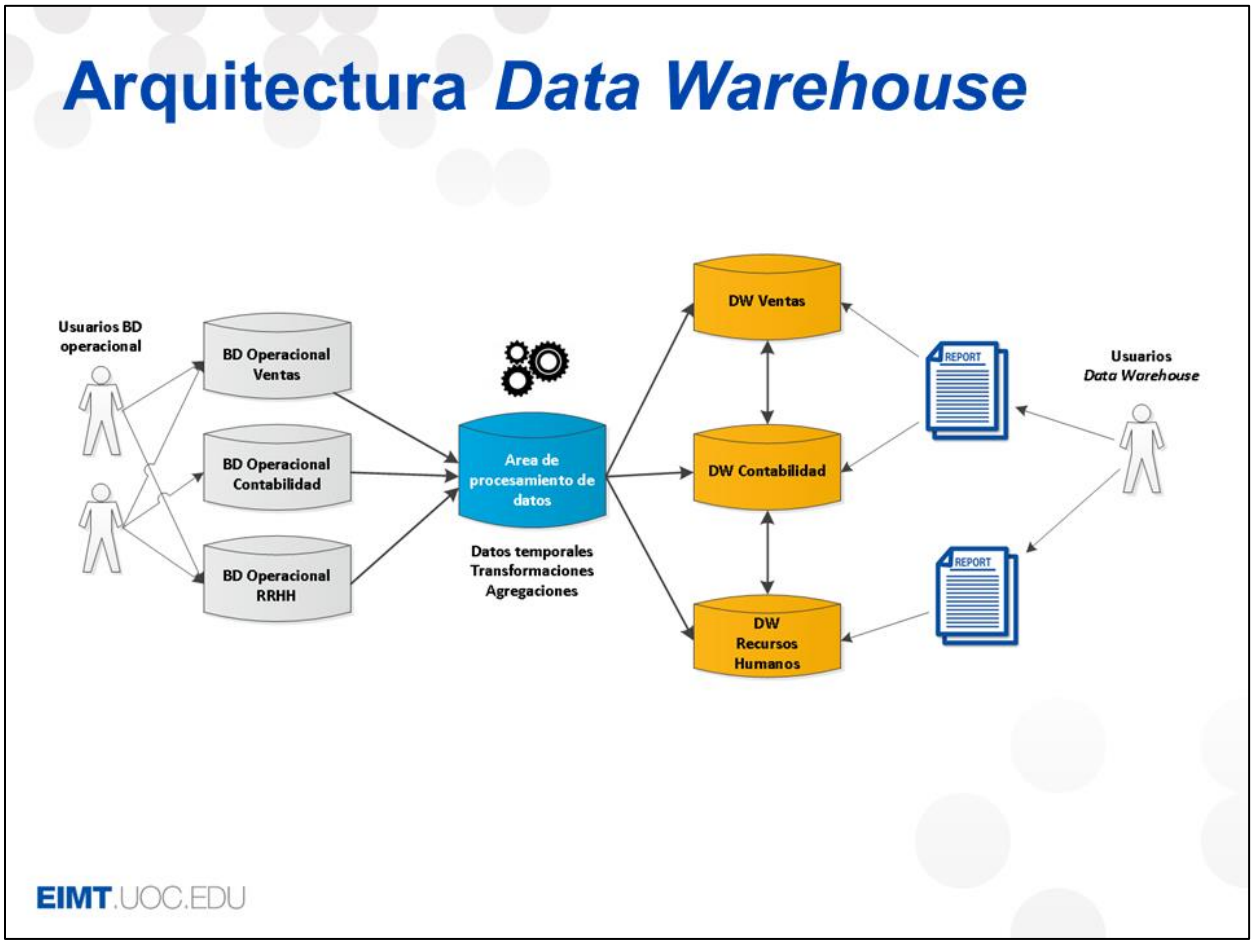
(el proceso de generación y actualización de información de auditoría suele implementarse de forma separada mediante un conjunto de procedimientos, que a su vez son llamados por aquellos procedimientos que se encargan de actualizar nuestra BD).

Un punto muy importante a tener en cuenta en el uso de procedimientos almacenados en BD operacionales es la gestión de las transacciones. En este tipo de BD es muy importante la concurrencia entre usuarios para poder mantener los datos consistentes mientras éstos lanzan transacciones a la vez que pueden leer y actualizar los mismos datos al mismo tiempo. En este tipo de entornos, es importante que la transacción se genere siempre fuera del procedimiento, y que ésta esté manejada por las aplicaciones. Es decir, es responsabilidad del código de la aplicación el iniciar y finalizar la transacción, bien mediante la confirmación de los datos, o deshaciéndolos hasta volver al estado inicial.

Un último punto a considerar en el uso de procedimientos almacenados en BD operacionales es la gestión del error. ¿Qué pasa si, por alguna razón, se produce un error durante la ejecución de un procedimiento? ¿Qué hacemos cuando nos encontramos con un error? En estos casos, el error debemos de gestionarlo siempre en el nivel superior, independientemente de si es un procedimiento o la aplicación en sí. El nivel superior ha de saber si la ejecución del procedimiento almacenado finaliza o no en una situación de error, porque de lo contrario, la BD puede quedar en un estado inconsistente. En caso de que se produzca una situación de error, en general, el nivel superior tendrá que cancelar la transacción que invoca la ejecución del procedimiento almacenado.




Una vez vistos contextualizados los procedimientos almacenados en las BD operacionales, vamos a ver la contextualización de estos componentes desde el punto de vista del *Data Warehouse*.



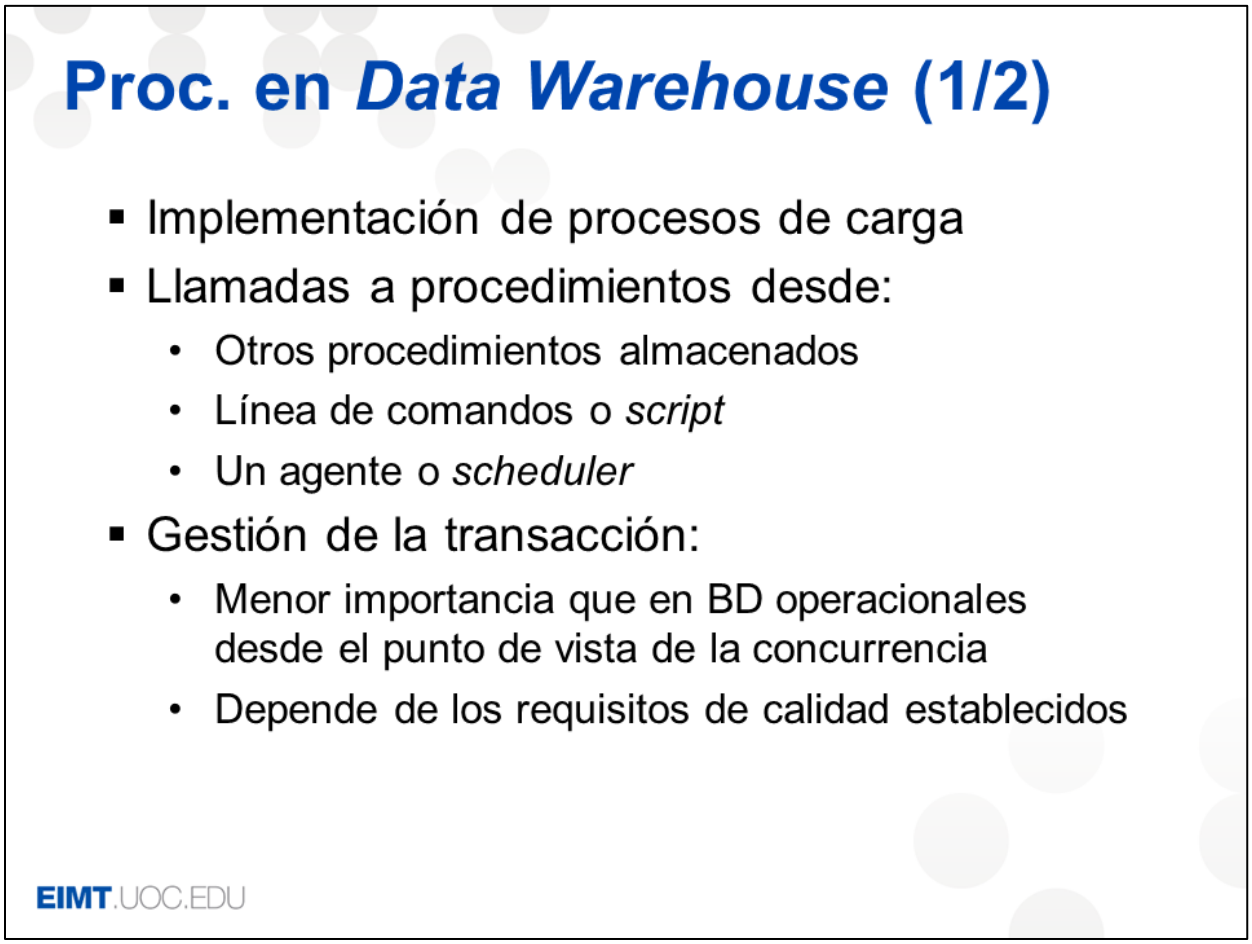

Antes de realizar la contextualización de procedimientos, vamos a ver un ejemplo típico de arquitectura de *Data Warehouse*, ejemplo que se presenta a alto nivel sin llegar a entrar en detalles. Los elementos típicos de una arquitectura *Data Warehouse* son los siguientes:

- En primer lugar, tenemos las BD operacionales. Estas BD son accedidas por muchos usuarios que realizan miles de transacciones de lectura y actualización de datos, generalmente afectando decenas o cientos de filas. Estas BD que suelen estar optimizadas para permitir un alto grado de concurrencia (miles de usuarios accediendo) y actúan como orígenes de los datos para nuestro *Data Warehouse*, ya que contienen la información sobre los procesos de negocio de nuestra empresa. En algunos casos, en lugar de BD operacionales también podemos tener como origen ficheros de texto, XML o cualquier otro formato.

- Como segundo componente, tenemos lo que denominamos área de procesamiento de datos. Se trata de una zona de trabajo intermedia, que se encarga de leer los datos de las BD operacionales y almacenarlos de forma temporal. Estas tareas son llevadas a cabo por los procesos de carga o ETL (*Extract, Transform and Load*). En esta zona temporal, los datos son transformados, mejorados según los procesos de calidad establecidos, e incluso pueden ser agregados. Ningún usuario debería de tener acceso a esta área, ya que ésta está dedicada

solamente para ser utilizada por los procesos de carga y, puntualmente, por los operadores del *Data Warehouse*.

- El tercer componente es ya el *Data Warehouse*, en el que se almacenan los datos ya transformados y agregados, preparados para ser consumidos por los usuarios finales. El número de usuarios en un *Data Warehouse* suele ser mucho menor comparado con el número de usuarios de las BD operacionales (del orden de lo cientos, generalmente), y éstos solamente realizan consultas de datos (lecturas) sobre una gran cantidad de datos. Un usuario de *Data Warehouse* nunca podrá realizar actualizaciones, ya que son los procesos de carga o ETL los encargados de realizar estas tareas a partir de las BD operacionales, tal y como hemos visto anteriormente.




A la hora de contextualizar los procedimientos en un entorno *Data Warehouse,* podemos decir que desde este punto de vista, los procedimientos almacenados nos resultan muy útiles para desarrollar e implementar procesos de carga si no disponemos de una herramienta ETL apropiada. Es muy común codificar procedimientos almacenados para la carga de tablas de dimensiones y hechos, además de gestionar las cargas de datos temporales, entre las BD origen (o BD operacionales) y destino (*Data Warehouse*).

La llamada a procedimientos almacenados en este tipo de entornos se suele realizar de las siguientes maneras:

- Llamada desde otros procedimientos: al igual que hemos comentado anteriormente, es posible que necesitemos de un servicio concreto en diferentes puntos de un proceso de carga, como es el de almacenar datos de auditoría. Por lo tanto, la llamada a procedimientos desde otros procedimientos es un mecanismo bastante común en desarrollo de procesos de carga.

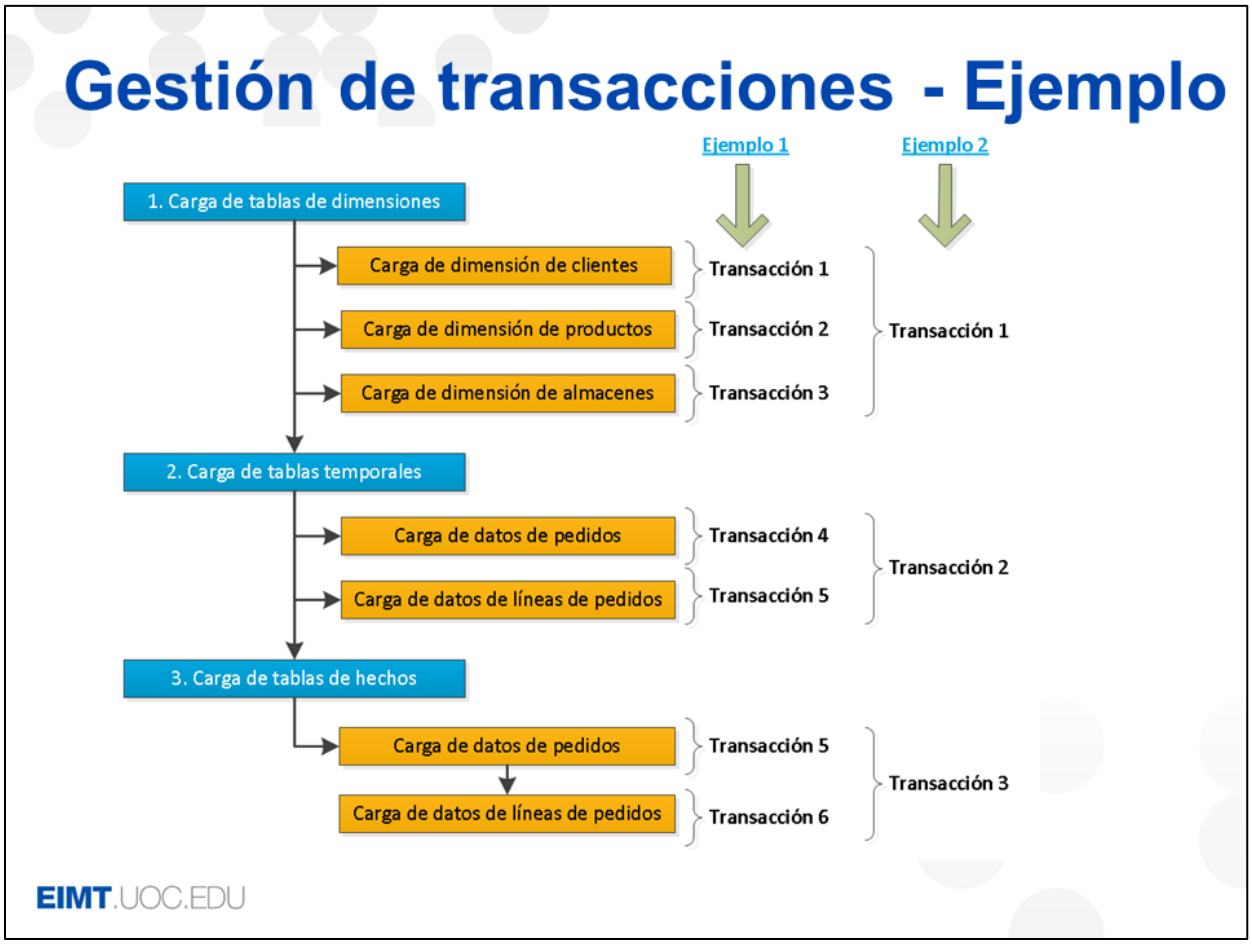
- Llamada desde la línea de comandos o desde un *script*: la llamada desde la consola suele ser común cuando necesitamos corregir cargas de datos, es decir, necesitamos realizar una tarea específica que es parte del proceso general. También pueden hacerse llamadas desde un *script* SQL, como parte de una serie de sentencias SQL.

- Llamada desde un agente o *scheduler*: los procesos de carga de datos en *Data Warehouse* suelen ser procesos que están programados para ser lanzados a una determinada hora del día, bien por la noche cuando no hay usuarios o incluso durante el día a intervalos constantes de tiempo. En estos casos, la programación del proceso se suele realizar mediante un agente o *scheduler*, que generalmente nos permite ejecutar sentencias SQL y realizar así llamadas a procedimientos almacenados.

A diferencia de lo mencionado anteriormente en BD operacionales, la gestión de las transacciones en procesos de carga es menos importante desde un punto de vista de la concurrencia. Las transacciones, si bien son necesarias para confirmar los datos en el *Data Warehouse*, pueden ser implementadas de forma que estas tengan una duración más larga, o incluso con un nivel de aislamiento inferior al que podría requerirse en una BD operacional. Suelen ser también utilizadas como puntos de control específicos para la recuperación de procesos de carga. De esta forma, llegados a un punto concreto, sabemos que podemos relanzar la carga sin problemas sin tener que manipular los datos de forma manual.

Recordemos que, por naturaleza y como hemos mencionado anteriormente, una BD operacional requiere de accesos de lectura y escritura por parte de miles de usuarios, mientras que en *Data Warehouse*, el acceso a los datos es de lectura por parte de unos cientos de usuarios como mucho. Aquí radica la diferencia, y por qué la gestión de transacciones en un *Data Warehouse* podría relajarse en comparación con una BD operacional.

De todas formas, la gestión de las transacciones en un proceso de carga codificado mediante procedimientos almacenados depende en gran medida de los requisitos de calidad establecidos en el proyecto en el que estemos trabajando.




Veamos un ejemplo de gestión de transacciones en un *Data Warehouse*.

Supongamos que tenemos un proceso de carga en un *Data Warehouse* que ejecuta cada noche los siguientes procesos de forma secuencial:

1. Carga de tablas de dimensiones, que tiene como tareas la carga de clientes, productos y almacenes.

2. Carga de tablas temporales, cuyo objetivo es el de obtener los pedidos y líneas de pedidos que han sido creados o actualizados recientemente (por ejemplo, en los últimos 3 días). Como tareas tiene la carga de pedidos y la carga de líneas de pedidos, tareas que se realizan dentro del área de procesamiento de datos.

3. Carga de tablas de hechos, que son las dos tareas que podemos ver en la parte inferior de la imagen.

Desde el punto de vista de la concurrencia, podemos suponer que no tendremos usuarios conectados mientras el proceso de carga se está llevando a cabo, por lo que podemos establecer transacciones más largas si así lo deseamos (sería el ejemplo de un proceso de carga en modo *batch* llevado a cabo

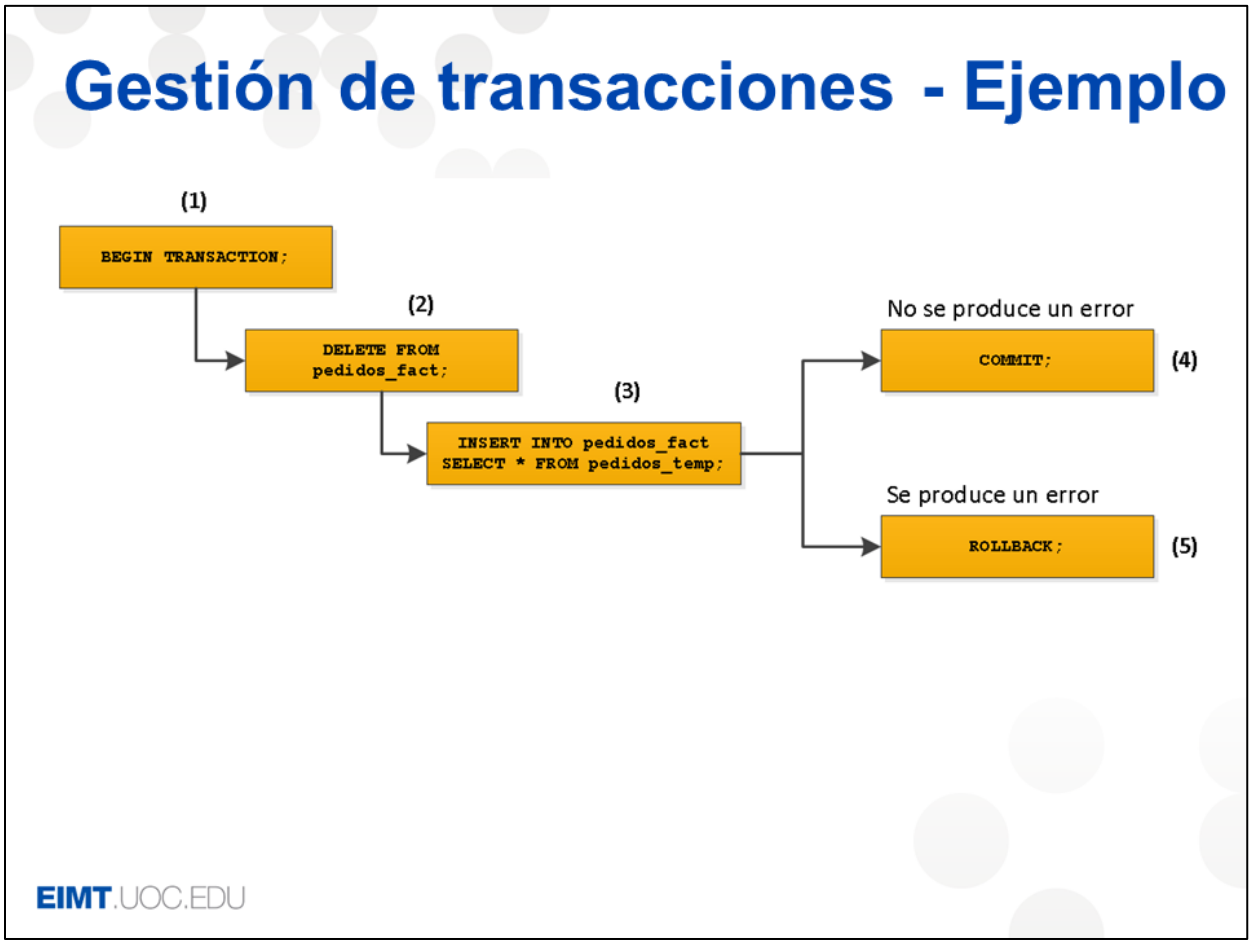
durante la noche, cuando no tenemos usuarios conectados). Si lo que necesitamos es implementar procesos de carga en tiempo real (mientras los usuarios están conectados), podría ser necesario realizar transacciones más cortas para no competir en recursos con las lecturas de datos masivas por parte de los usuarios.

Por lo tanto, en base al ejemplo presentado, podríamos pensar en establecer una transacción para cada una de las tareas que se ejecutan en los procesos anteriormente citados, como se ve en el ejemplo número 1. De esta manera, garantizamos que los datos estén disponibles tras la finalización de cada tarea. En este caso, la transacción tiende a ser corta en términos de tareas y tiempo, aunque depende en gran medida de la cantidad de datos a actualizar en cada una de las tareas implementadas.

En el ejemplo número 2, vemos que la transacción se establece para todas las tareas de cada uno de los procesos. Por lo tanto, la transacción es más larga y los datos solamente estarán disponibles al finalizar el proceso en cuestión, y no cuando la tarea finaliza.

En cualquiera de los casos propuestos, es necesario establecer un equilibrio entre la longitud de la transacción, el rendimiento de los procesos de carga, y los requisitos de calidad establecidos por los usuarios o por el proyecto. Es posible que el tiempo del proceso dure menos si establecemos las transacciones por tareas (como en el ejemplo número 1) que si estableciésemos una transacción por proceso (como en el ejemplo número 2), ya que en el primer caso, el SGBD dispondrá de más recursos libres (como por ejemplo, espacio temporal para consultas) y las tablas no permanecerán bloqueadas por un espacio de tiempo más largo, como en el caso del segundo ejemplo.

En la vida real, suele implementarse el ejemplo número 1, incluso considerando una división de la tarea en múltiples transacciones, dependiendo de lo complicado que fuese el proceso de carga implementado.



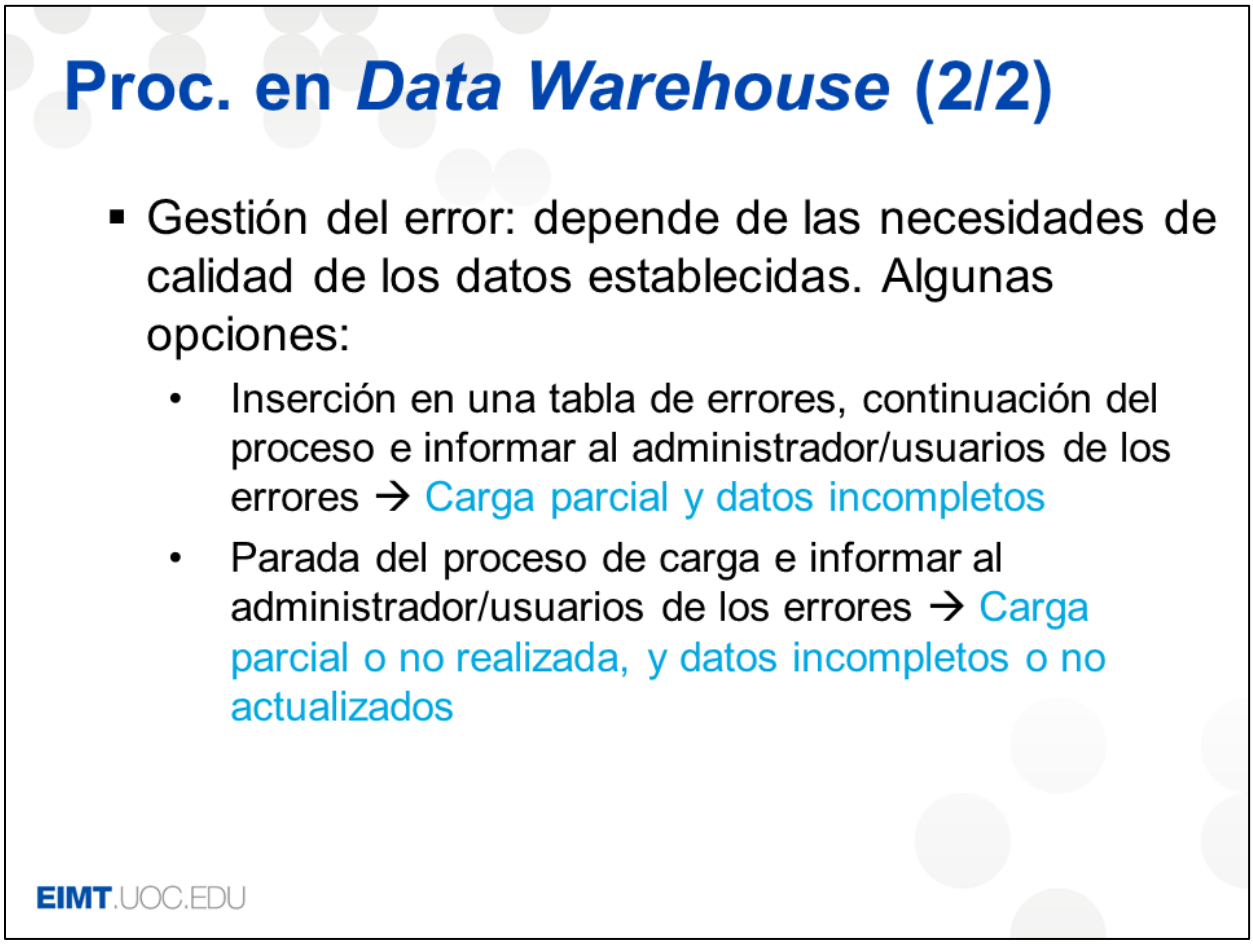

Un escenario especial relativo a las transacciones se da cuando queremos reemplazar los contenidos de una tabla por otros mediante la realización de un borrado y una carga completa (es decir, eliminación completa de los datos y recarga completa con datos nuevos justo a continuación). Este escenario se puede ver en la imagen mostrada.

Muchas veces en entornos *Data Warehouse* se requiere que la tabla de hechos siempre se encuentre cargada con datos para que los usuarios sigan accediendo a ellos, bien como requisito de la aplicación o simplemente para disponer de información para la realización de informes. El uso de las transacciones es muy necesario cuando nos encontramos en esta situación.

Para gestionar este escenario de manera eficiente, la transacción se suele iniciar (punto 1) justo antes de eliminar los datos de la tabla de hechos (punto 2) y se confirma (punto 4) justo después de recargar los datos en la misma tabla a partir de una tabla temporal (punto 3). En caso de que exista un error en la carga, la transacción desharía los cambios (punto 5) y volvería al estado inicial, garantizando que mientras la transacción se está llevando a cabo, la tabla de hechos seguiría disponiendo de los datos antiguos para que los usuarios sigan haciendo uso de éstos, por lo que la tabla de hechos nunca se encontraría vacía.

Recordad que en *Data Warehouse*, el acceso a los datos es de lectura por parte de los usuarios y nunca de escritura, por lo que la transacción de recarga de datos en la tabla de hechos nunca competirá con otras transacciones que modifiquen los datos en la misma tabla.

A este tipo de carga de datos se le suele denominar **todo o nada** (es decir, o se carga todo el conjunto de datos o no se carga nada, volviendo al estado inicial en caso de que no se pueda realizar una carga completa).

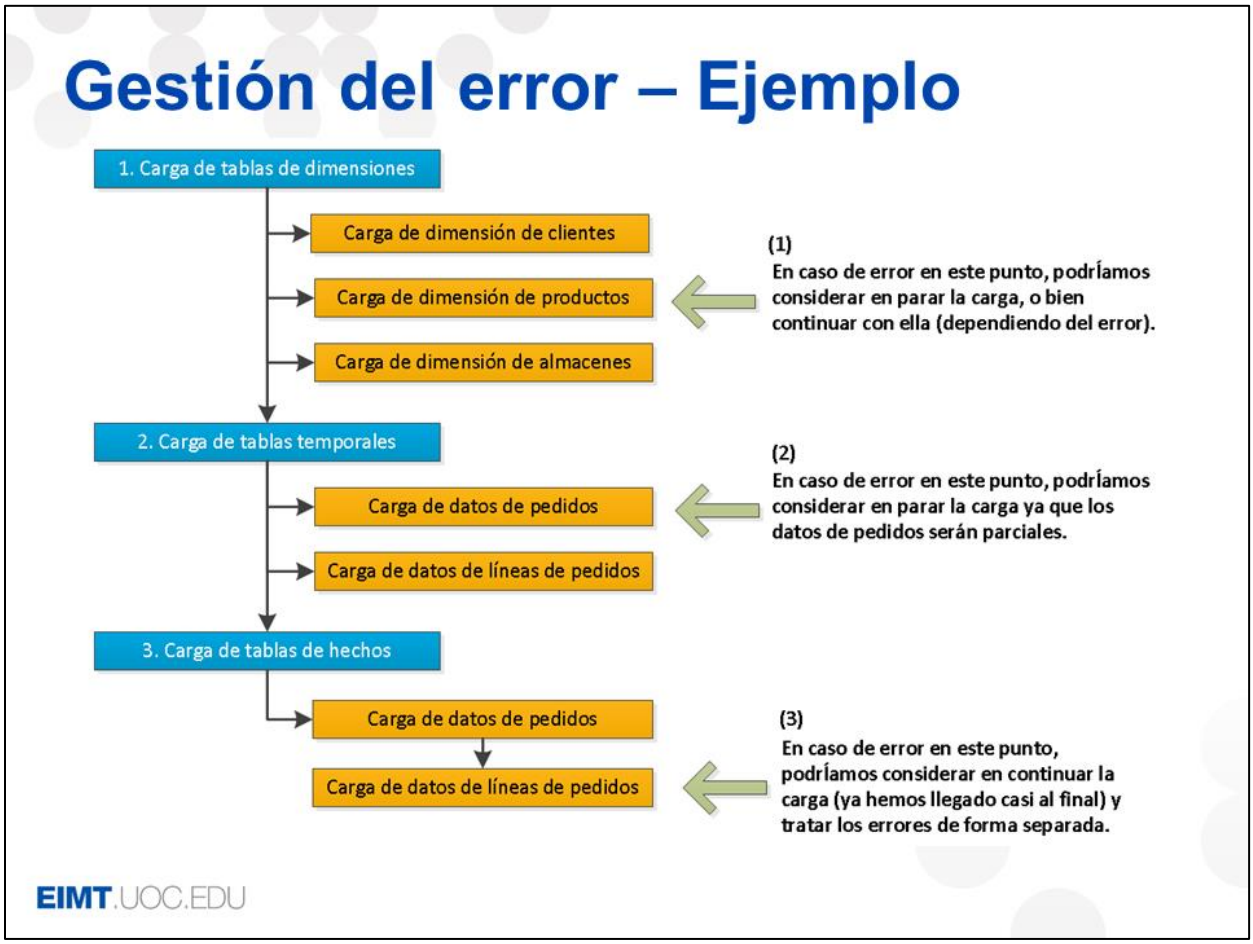



Continuando con la contextualización de procedimientos en *Data Warehouse*, pasamos a tratar la gestión del error. Este punto es muy importante en los procesos de carga, ya que depende en gran medida de los requisitos de calidad de datos impuestos como parte del proyecto. Como ejemplos de tratamiento de datos (que no son todos), proponemos los siguientes:

- Capturar el error en una tabla de errores y continuación de la carga de datos. En estos casos, se informa al administrador y a los usuarios, y se gestionan los errores generalmente de forma manual por parte de un operador. Estos errores pueden producirse por muchos motivos: falta de datos, problemas con nulos, duplicados, etc., aunque puede darse el caso de tratarse de errores con la conectividad de la red, o incluso fallos en el *hardware*. El hecho de continuar con la carga de datos (si es posible) produce que la carga se considere parcial, y que los datos estén incompletos (a la espera de la corrección de los datos).

- Parar el proceso de carga. En estos casos, nos arriesgamos a no realizar ninguna actualización de datos, especialmente si el error se produce al inicio de la carga. Esto, aunque parezca un problema, puede ser una ventaja sobre las cargas parciales, ya que los datos no están a medias sino que simplemente no han sido cargados todavía. En muchas ocasiones, basta con corregir el

error y relanzar de nuevo el proceso de carga. Si el error se produce hacia el final, entonces nos encontraremos con la problemática de la carga parcial y de presentar datos incompletos.




Veamos, utilizando el mismo ejemplo de procesos y tareas visto anteriormente, qué pasaría en los siguientes supuestos en base a lo que hemos comentado sobre la gestión de los errores:

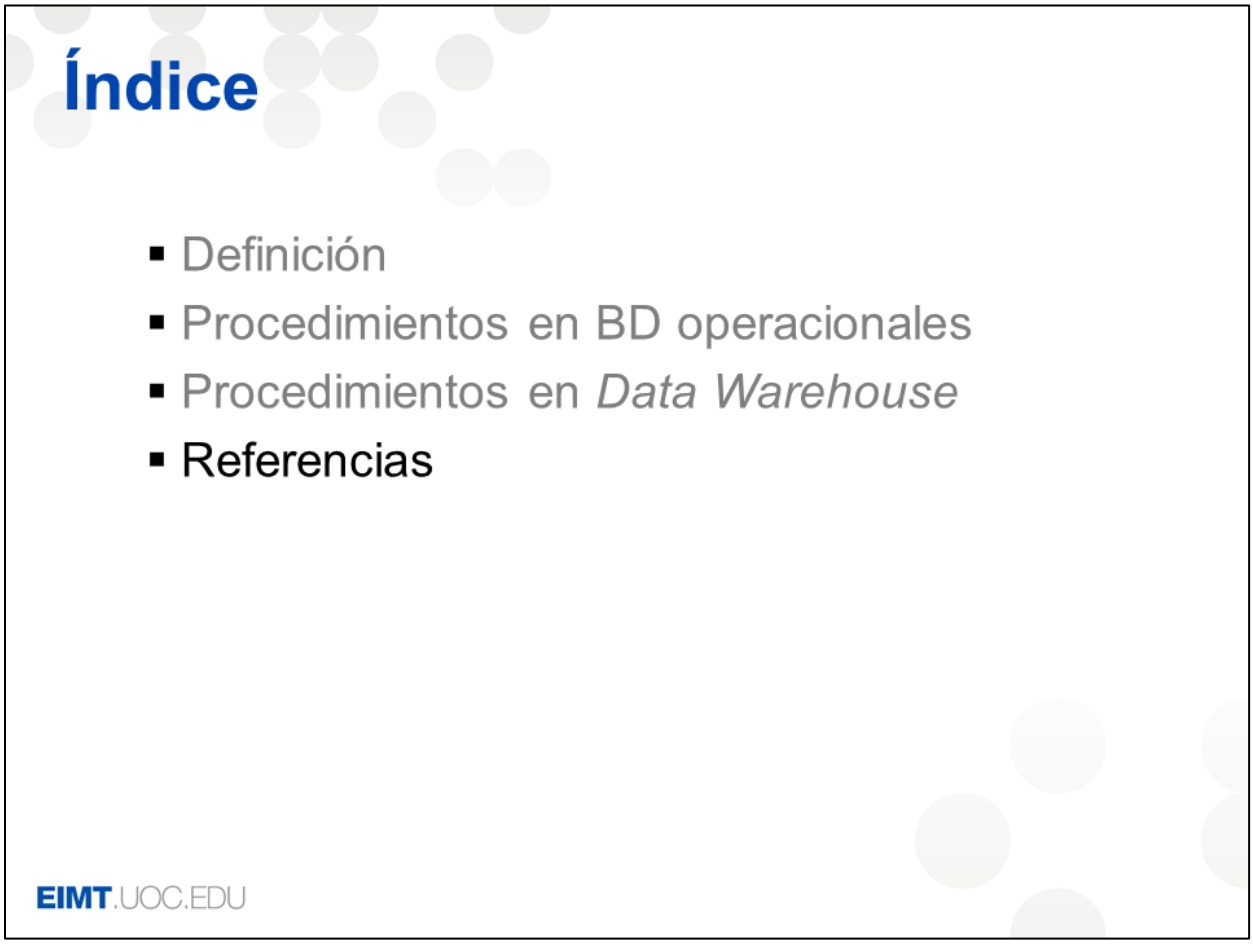
1. Se produce un error en la carga de productos: en estos casos, podemos realizar lo siguiente:
    a. Parar la carga, y notificar al administrador y a los usuarios. Se suele seguir esta opción si, como requisito, es necesario que los productos existan para cargar las líneas de pedido (es decir, que la carga de líneas de pedido genere un error en caso de que un producto no se encuentre en la dimensión de productos). En este caso, la carga de datos no está realizada, por lo que los nuevos datos de pedidos no estarán disponibles para el usuario.
    b. Inserción en una tabla de errores y continuación del proceso. Los errores serán tratados una vez notificados, pero el proceso de carga continúa. Este mecanismo sería suficiente si tenemos una gestión del error similar en la carga de líneas de pedidos cuando los productos no existen (es decir, el proceso de carga no se para si no tenemos en la dimensión el producto asociado a la línea de pedido). En este caso, dispondremos de una carga parcial, por lo que los datos estarán incompletos para los usuarios.

2. Se produce un error en la carga de datos de pedidos modificados: en estos casos, podemos realizar lo siguiente:
    a. Parar la carga, y notificar al administrador y a los usuarios. Se recomienda esta acción si la tabla de hechos de líneas de pedido tiene una clave foránea a la tabla de hechos de pedidos. También se recomienda esta opción si se prefiere no disponer de datos incompletos de pedidos.
    b. Al igual que lo explicado anteriormente, podemos insertar los datos que en una tabla de errores y procesarlos de forma diferente una vez terminada la carga, dejando los datos de pedidos como incompletos.

3. Por último, se produce un error en la carga de datos de líneas de pedidos (hechos): en estos casos, podemos realizar lo siguiente:
    a. De nuevo, parar la carga, y notificar al administrador y a los usuarios. Se recomienda esta acción si deseamos no tener datos parciales o incompletos. Se podría requerir desde un punto de vista del usuario que se volviese al estado inicial (lo que supondría guardar, de alguna manera, el estado anterior de la tabla o de los registros modificados).
    b. Insertar los datos en una tabla de errores y procesarlos de forma separada, continuando con la carga. De nuevo, el usuario puede requerir los datos en su estado inicial en caso de error.

    En el caso de cargas de tablas de hechos, es común que, si el error capturado está relacionado con la falta de datos en las dimensiones (por ejemplo, que no exista un producto en concreto), se inserte la fila en la tabla con un identificador especial (generalmente, con valor -1 y descripción DESCONOCIDO), que significa que no es posible identificar en la dimensión el valor que se ha intentado buscar. Esta técnica es muy utilizada en *Data Warehouse* para asegurar que la falta de datos en origen no afecte a los modelos dimensionales. De esta forma, las cargas de datos finalizarán correctamente y los usuarios podrán ver los datos completos, eso sí, con una calidad de datos menor que si los datos estuvieran correctamente mapeados.

Por último, es importante destacar que, generalmente y como parte de la documentación entregada en un proyecto *Data Warehouse*, se suele incluir un manual de operador, que indica cómo se pueden relanzar los procesos de carga dependiendo de dónde ha fallado, y qué acciones deben de realizarse para recuperar la consistencia de los datos.




Y hasta aquí hemos llegado con este vídeo sobre la contextualización de procedimientos almacenados. Esperamos que os haya gustado la presentación y que ésta os haya servido de mucha ayuda.

Presentaremos ahora un conjunto de enlaces y referencias de interés acerca de este tema.

# Referencias

Casany Guerrero, M. J.; Urpí Tubella, T.; Rodríguez González, M. Elena; *El Lenguaje SQL II*. Fundación UOC. *PID_00171670*

PostgreSQL:
http://www.postgresql.org/docs/9.3/static/

Oracle:
https://docs.oracle.com/cd/E11882_01/nav/portal_4.htm

EIMT.UOC.EDU

¡Que tengáis un buen día!