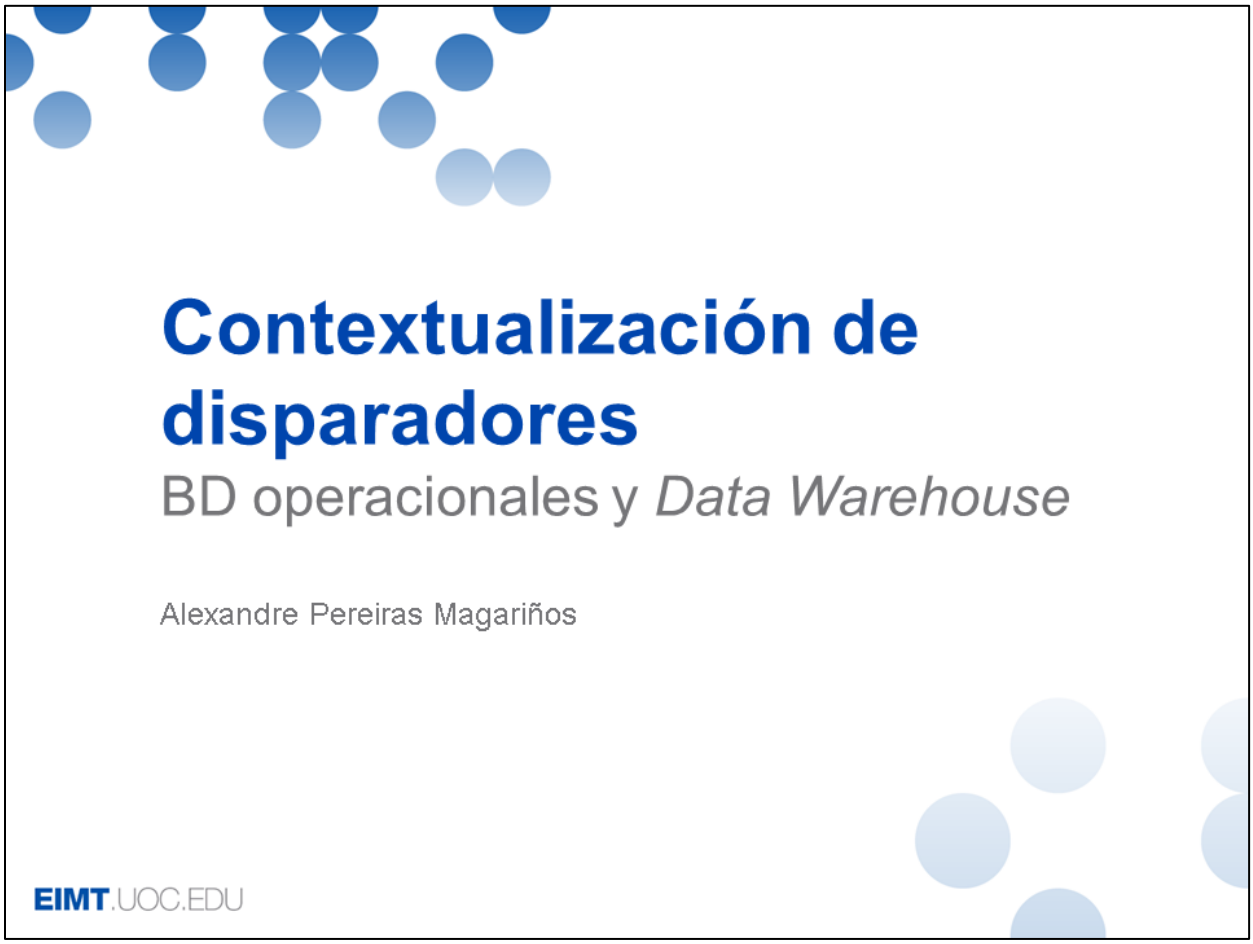
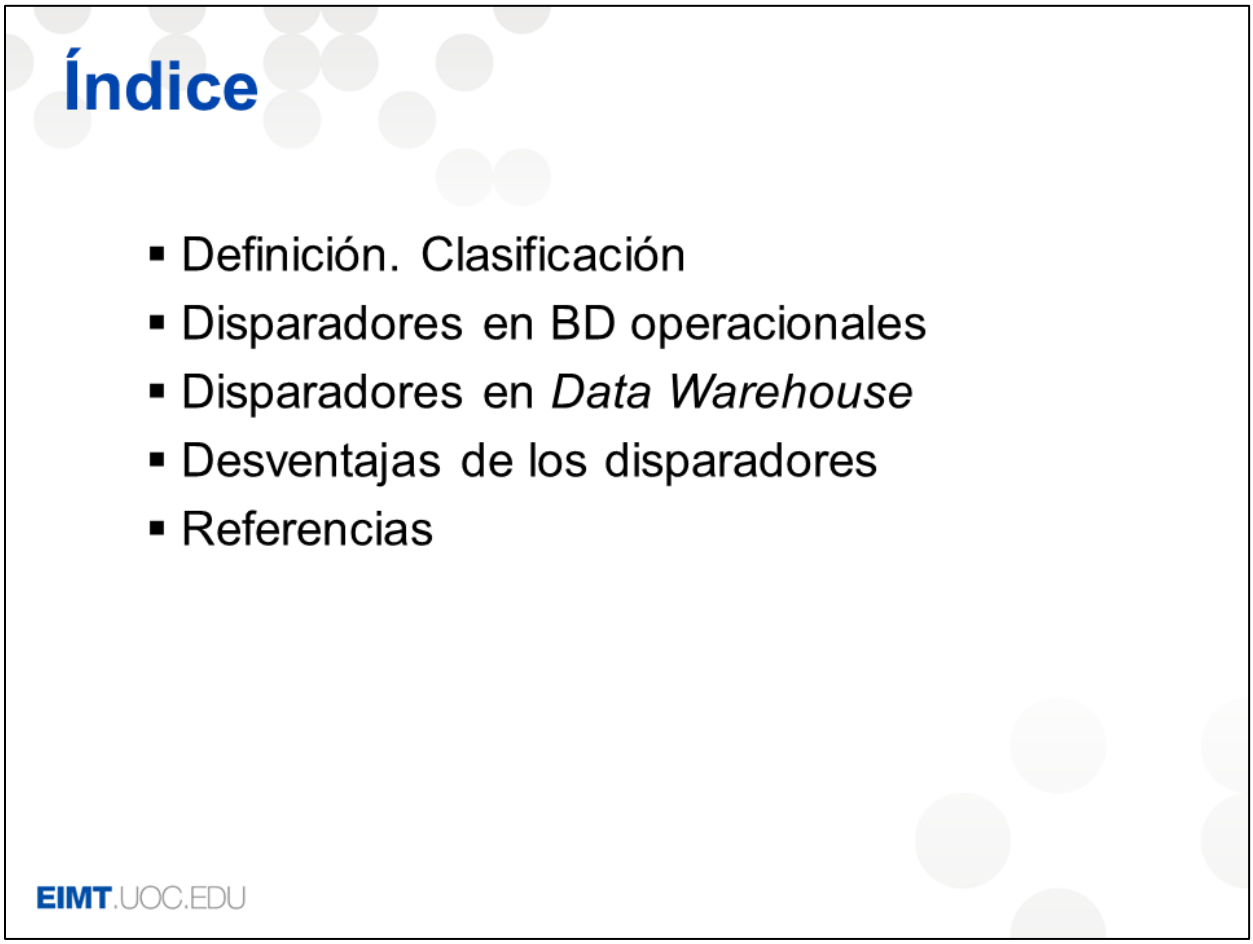
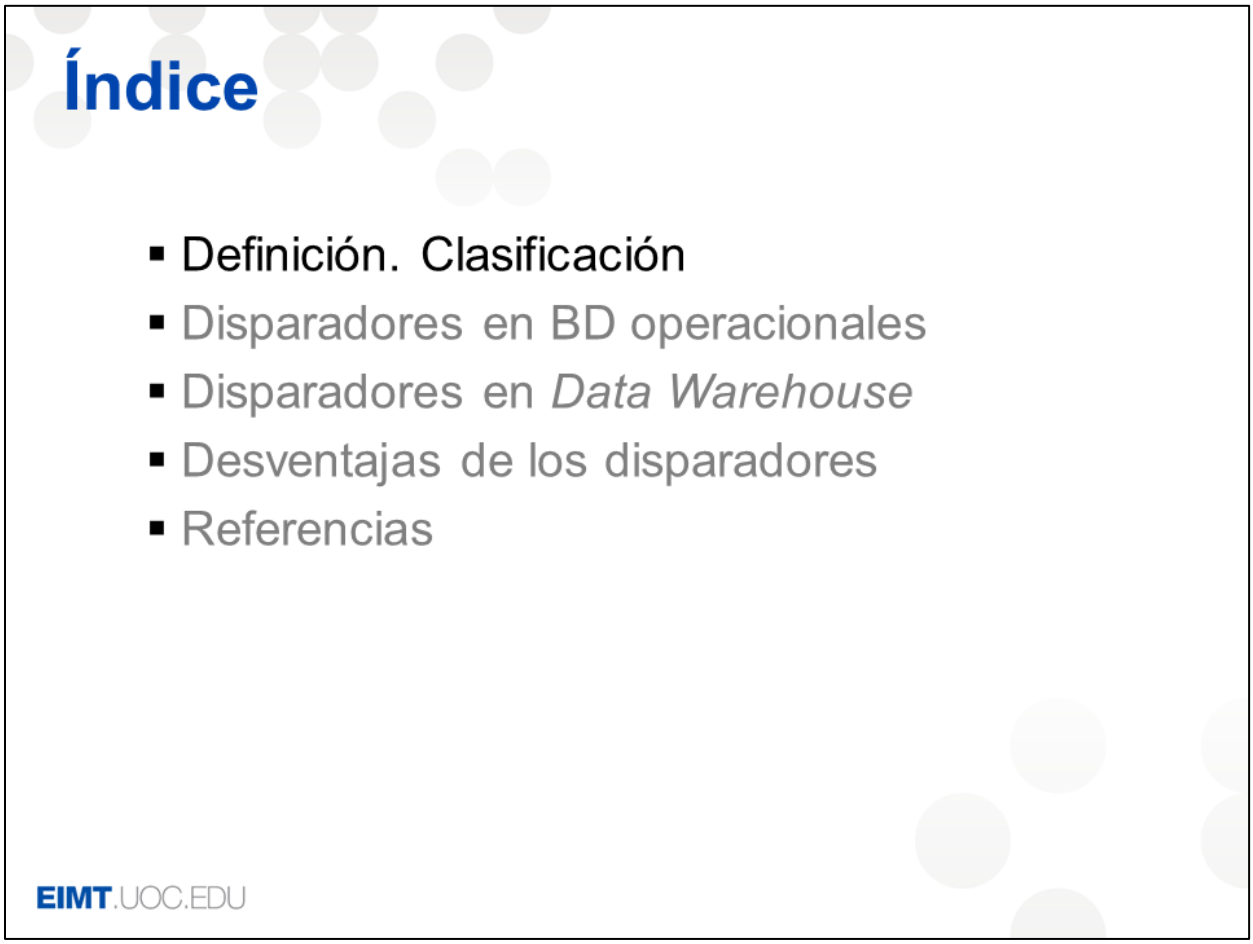


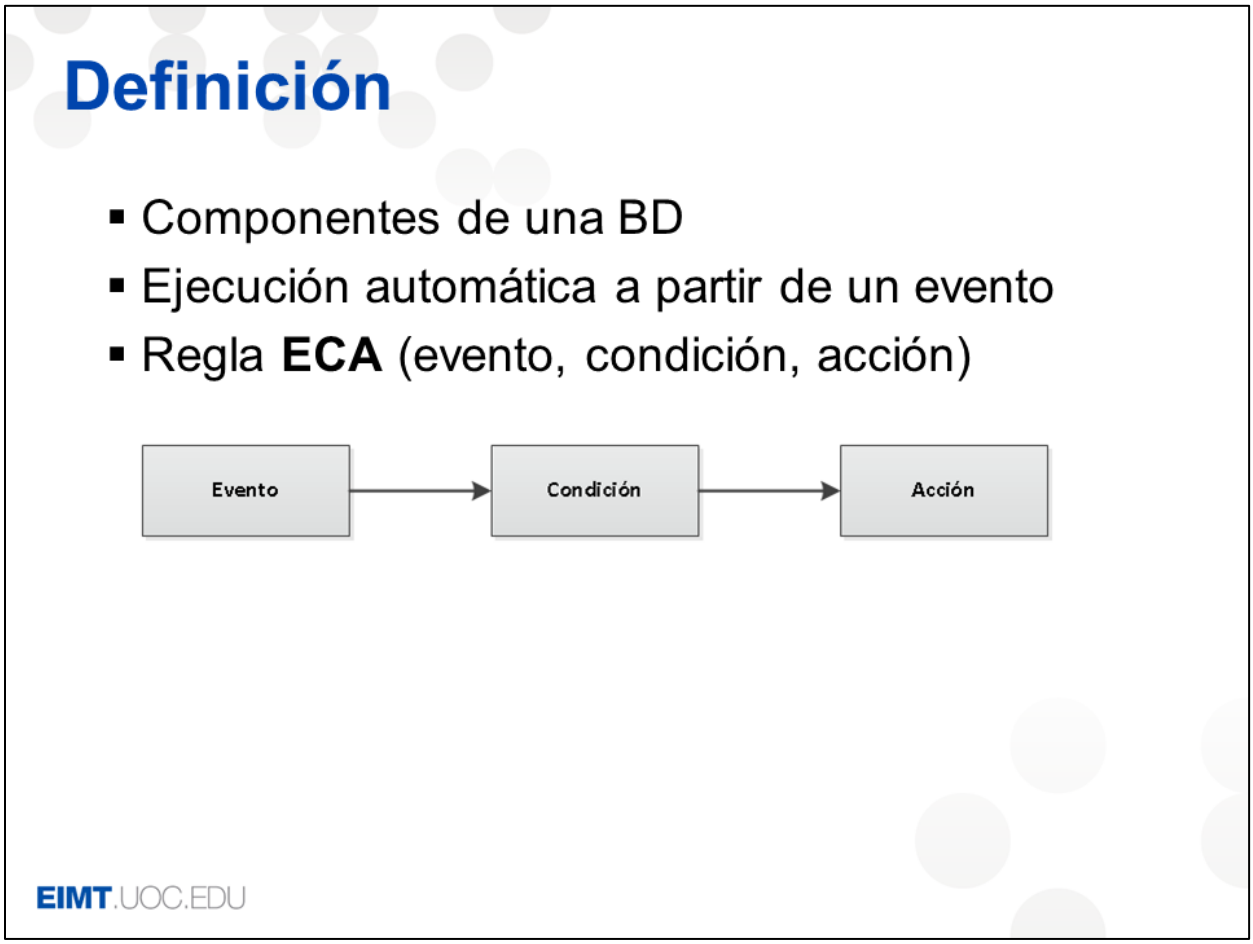
Bienvenidos a este vídeo titulado *Contextualización de disparadores* de la asignatura de *Bases de datos para Data Warehouse*, en el que comentaremos y explicaremos qué son los disparadores, y cómo podemos aplicarlos dentro del marco de las bases de datos (BD) operacionales y de *Data Warehouse*.



Este vídeo se centrará primero en introducir brevemente los disparadores y a continuación, se contextualizará los disparadores desde un punto de vista de BD operacionales y de *Data Warehouse*, revisando los usos que estos componentes nos pueden proporcionar en este tipo de entornos. Veremos también una serie de desventajas que acarrea el uso de disparadores, y por último, se proporcionarán las referencias bibliográficas utilizadas que se han utilizado para elaborar esta presentación.

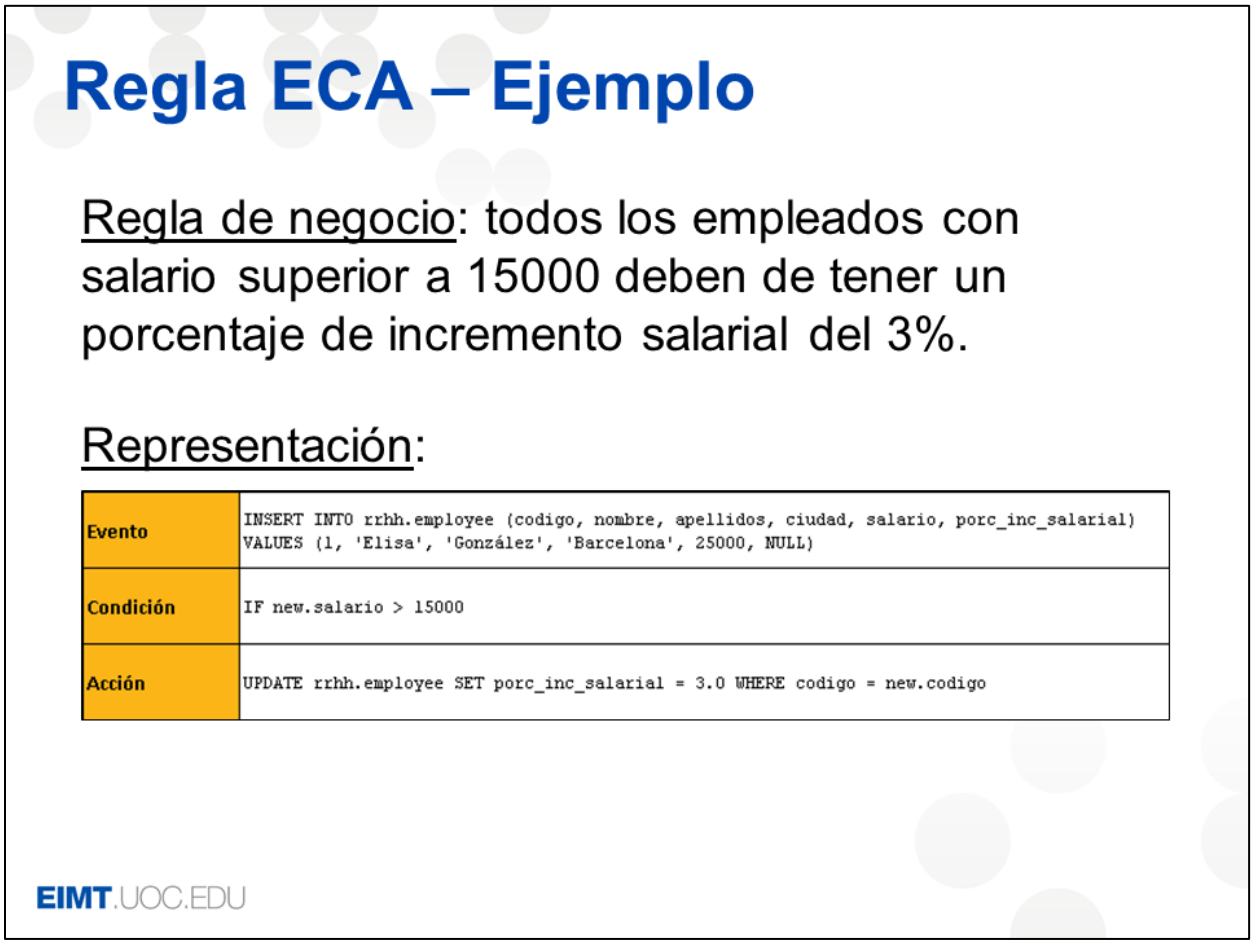


Vamos pues a comenzar con la definición de disparador y las clasificaciones de éstos.



Los disparadores son componentes específicos de una base de datos (BD) que se asocian a una tabla o a una vista, y que tienen un comportamiento particular: estos objetos se ejecutan de forma automática, reaccionando ante un evento sobre la tabla o vista definida para realizar así una acción concreta. Los diferentes eventos a los que un disparador reacciona son las operaciones de `INSERT`, `UPDATE` o `DELETE`.

Este tipo de mecanismos evento–acción suele conocerse, de forma genérica, como mecanismos o **reglas ECA**, donde la E representa Evento, la C representa Condición y la A representa Acción. Las reglas ECA son, por lo tanto, reglas que reaccionan ante un evento, evalúan una condición, y dependiendo de dicha evaluación, ejecutan una acción concreta.

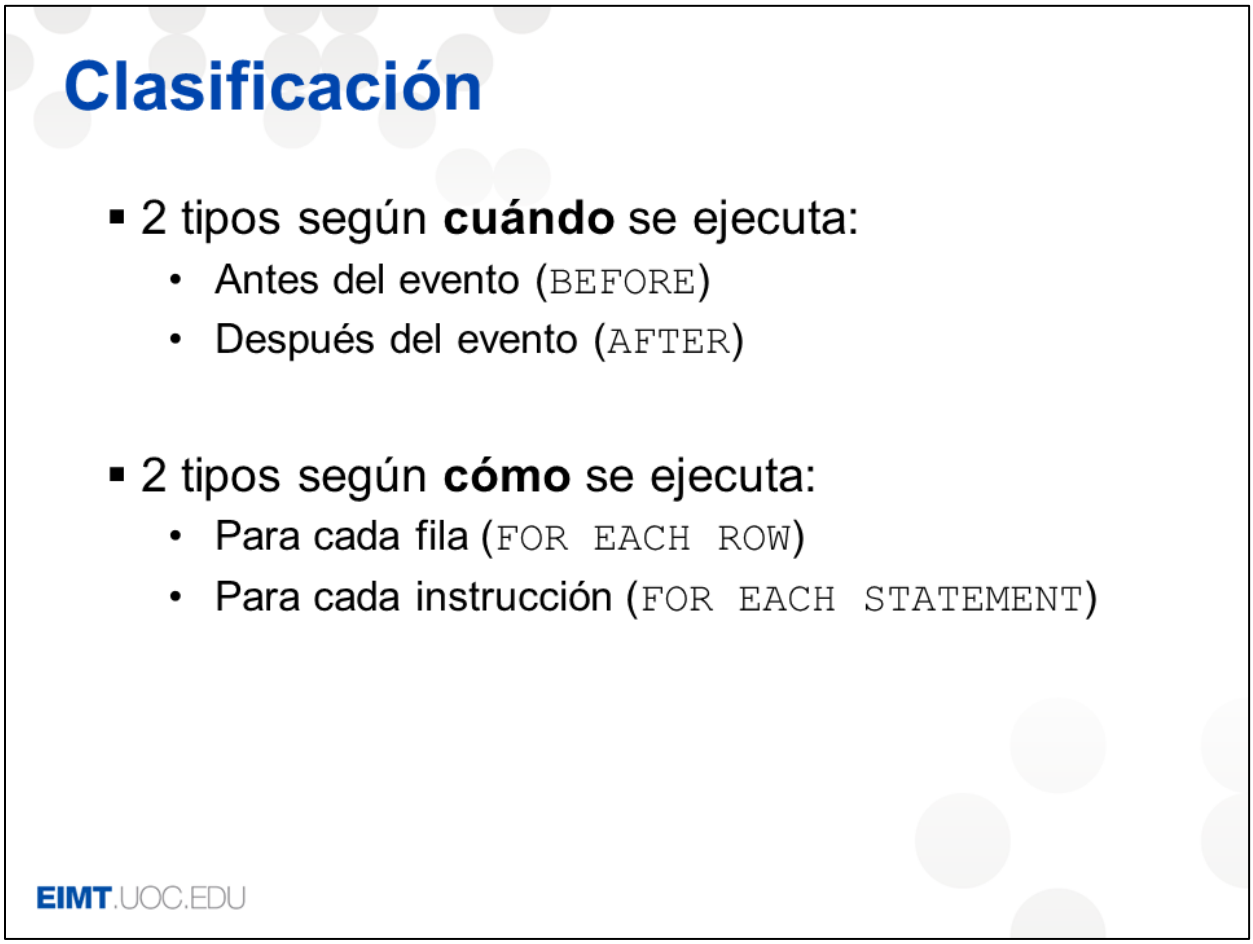


Veamos un ejemplo de regla ECA implementada en un sistema gestor de bases de datos (SGBD).

Imaginad que disponemos de una BD de Recursos Humanos con una tabla de empleados (*employee*) en la que se guarda la siguiente información de los empleados de una empresa: código de empleado, nombre, apellidos, ciudad, salario y porcentaje de incremento salarial. Queremos implementar una regla de negocio utilizando una regla ECA, de forma que cada vez que se inserta un nuevo empleado en la tabla, se debe de comprobar que si el salario es superior a 15000 unidades, entonces el porcentaje de incremento salarial asignado debe de ser del 3% (3.0).

Esta regla se puede representar de la siguiente manera, según el gráfico mostrado en la parte inferior de esta diapositiva: vemos que se detalla el evento como la inserción de los datos del empleado Elisa González, con un salario de 25000 unidades. Una vez que se ha capturado el evento de inserción, la regla evalúa la condición (`IF new.salario > 15000`). Si la evaluación de dicha condición es positiva, entonces se lleva a cabo la acción, que es la de actualizar el porcentaje de incremento salarial de dicho empleado al 3%.

Ved que en el ejemplo propuesto se ha utilizado `new` para poder referenciar aquellos valores de la fila que se ha insertado como parte del evento.



De forma genérica, los disparadores pueden definirse de diferentes maneras atendiendo a dos criterios ortogonales entre sí: **cuándo** se ejecuta y **cómo** se ejecuta.

Desde un punto de vista temporal, se pueden definir disparadores que se ejecuten **antes** del evento (la evaluación de la condición y la ejecución de la acción se llevarán a cabo antes de la sentencia SQL asociada al evento) o bien **después** del evento (caso en el que la evaluación de la condición y la ejecución de la acción se llevarán a cabo después de la sentencia SQL asociada al evento). Para poder definir estos dos tipos de disparadores se utilizan las cláusulas SQL `BEFORE` y `AFTER` respectivamente.

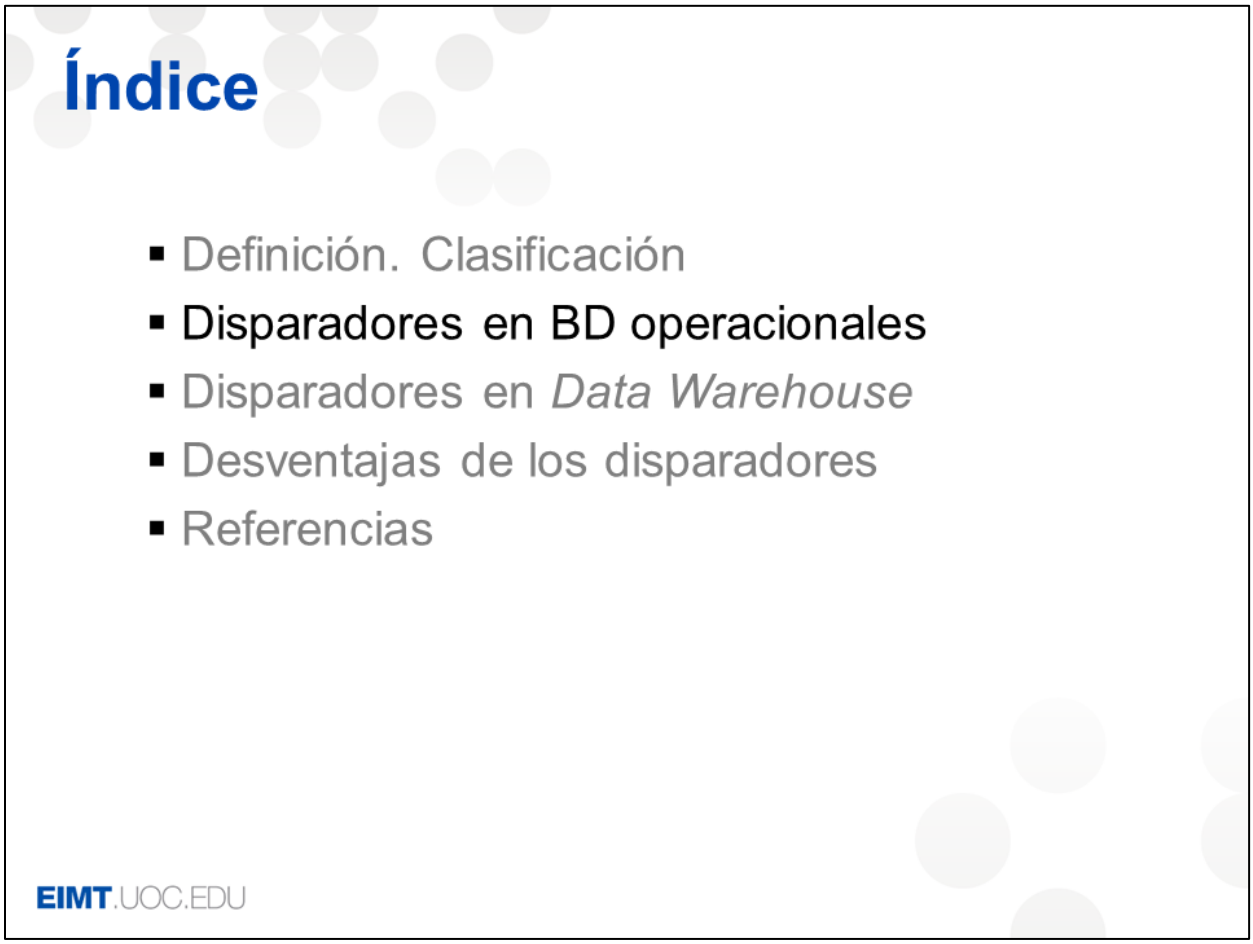
Desde un punto de vista funcional, se pueden definir disparadores que se ejecuten **para cada fila** (la evaluación de la condición y la ejecución de la acción se llevarán a cabo de forma individual para cada fila modificada por la sentencia SQL del evento) o bien **para cada instrucción** (caso en el que la evaluación de la condición y la ejecución de la acción se llevarán a cabo una única vez para la instrucción especificada). Para poder definir estos dos tipos de disparadores se utilizan las cláusulas SQL `FOR EACH ROW` y `FOR EACH STATEMENT` respectivamente.

Por tanto, y como podéis imaginar, podemos combinar estos dos criterios a la hora de definir disparadores. Por ejemplo, podemos definir un disparador `BEFORE / FOR EACH ROW` que se ejecutará

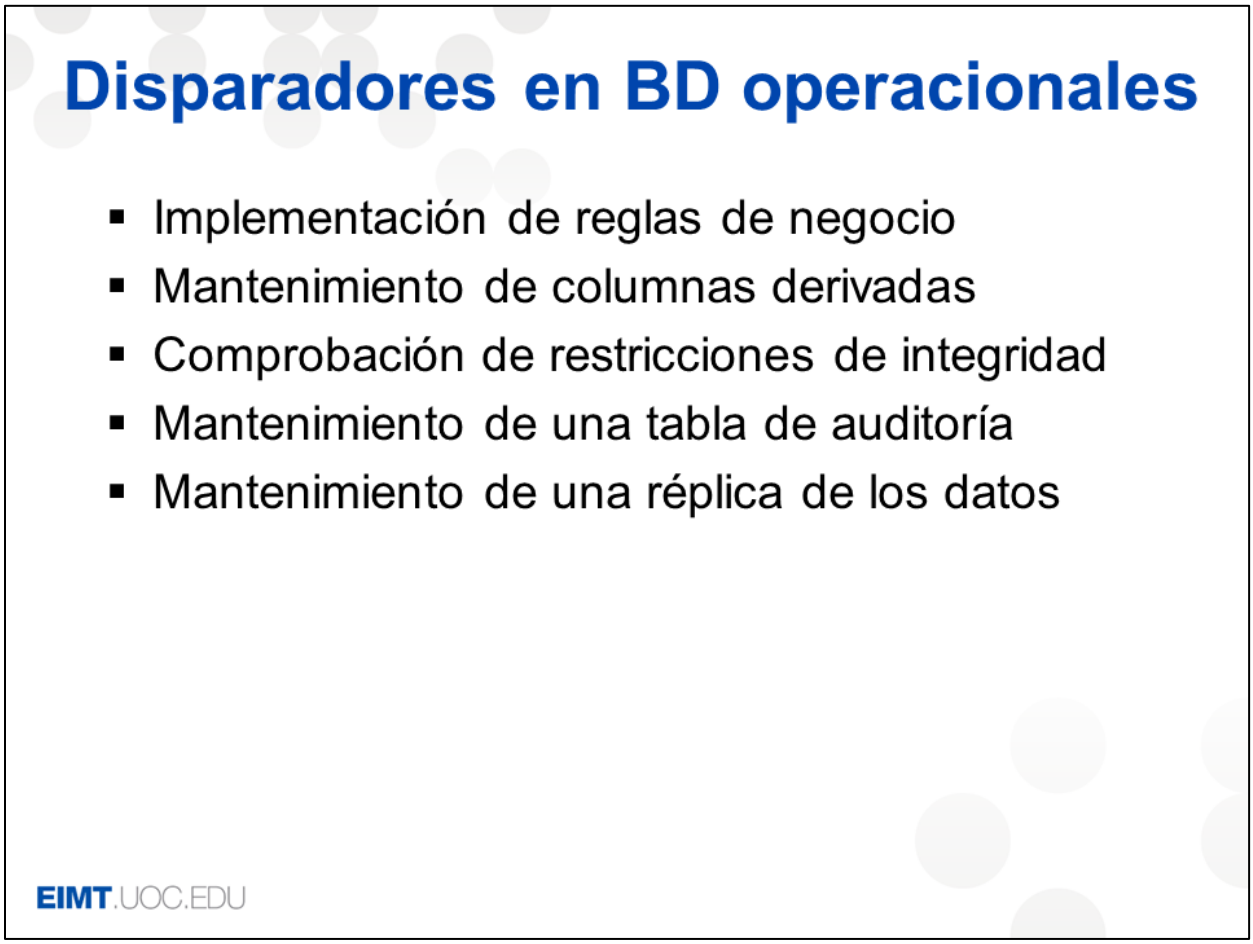
antes del evento y para cada fila, o bien `AFTER/FOR EACH STATEMENT` que se ejecutará después del evento y una única vez para toda la instrucción.

Según el SGBD, pueden existir extensiones a las opciones vistas o incluso características especiales, por lo que recomendamos que se revisen los manuales técnicos proporcionados por cada SGBD. En el caso de PostgreSQL, recomendamos revisar la siguiente referencia bibliográfica, además de los manuales en línea de PostgreSQL 9.3 (http://www.postgresql.org/docs/9.3/static/), ambas proporcionadas al final de este vídeo:

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



A continuación, vamos a ver en qué supuestos podemos usar los disparadores en BD operacionales.



Los disparadores, en el ámbito de las BD operacionales, pueden ser utilizados para diferentes propósitos. En esta transparencia mencionamos algunos de ellos:

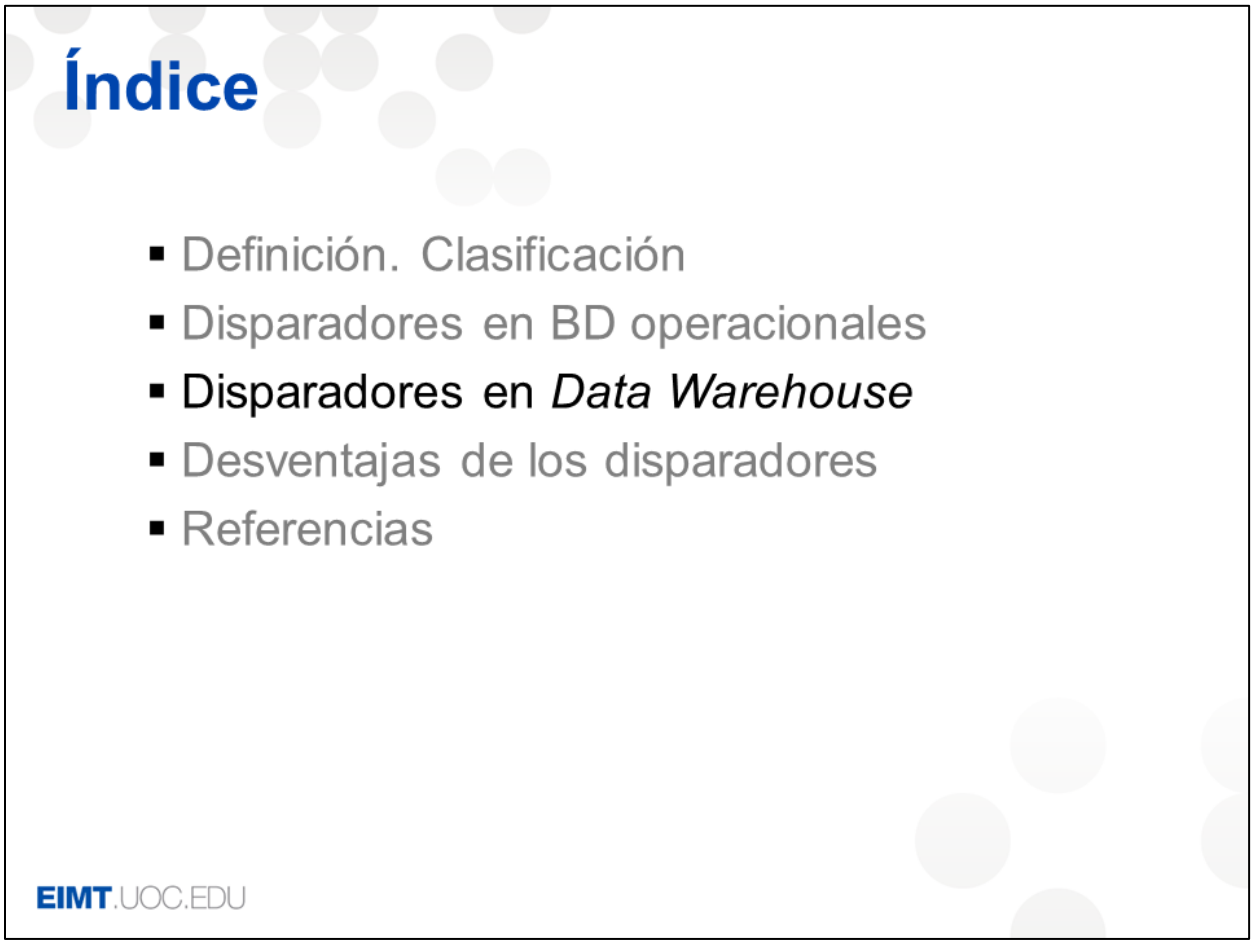
- **Implementación de reglas de negocio**: como por ejemplo, la regla de negocio propuesta en este mismo vídeo para definir las reglas ECA.

- **Mantenimiento de columnas derivadas**: el valor de una columna derivada en una tabla se calcula a partir del valor de otras columnas dentro de la misma tabla, e incluso de otras tablas. Por lo tanto, cuando se modifican las columnas base, hay que recalcular de una manera automática el valor de la columna derivada para asegurar que los datos sean consistentes.

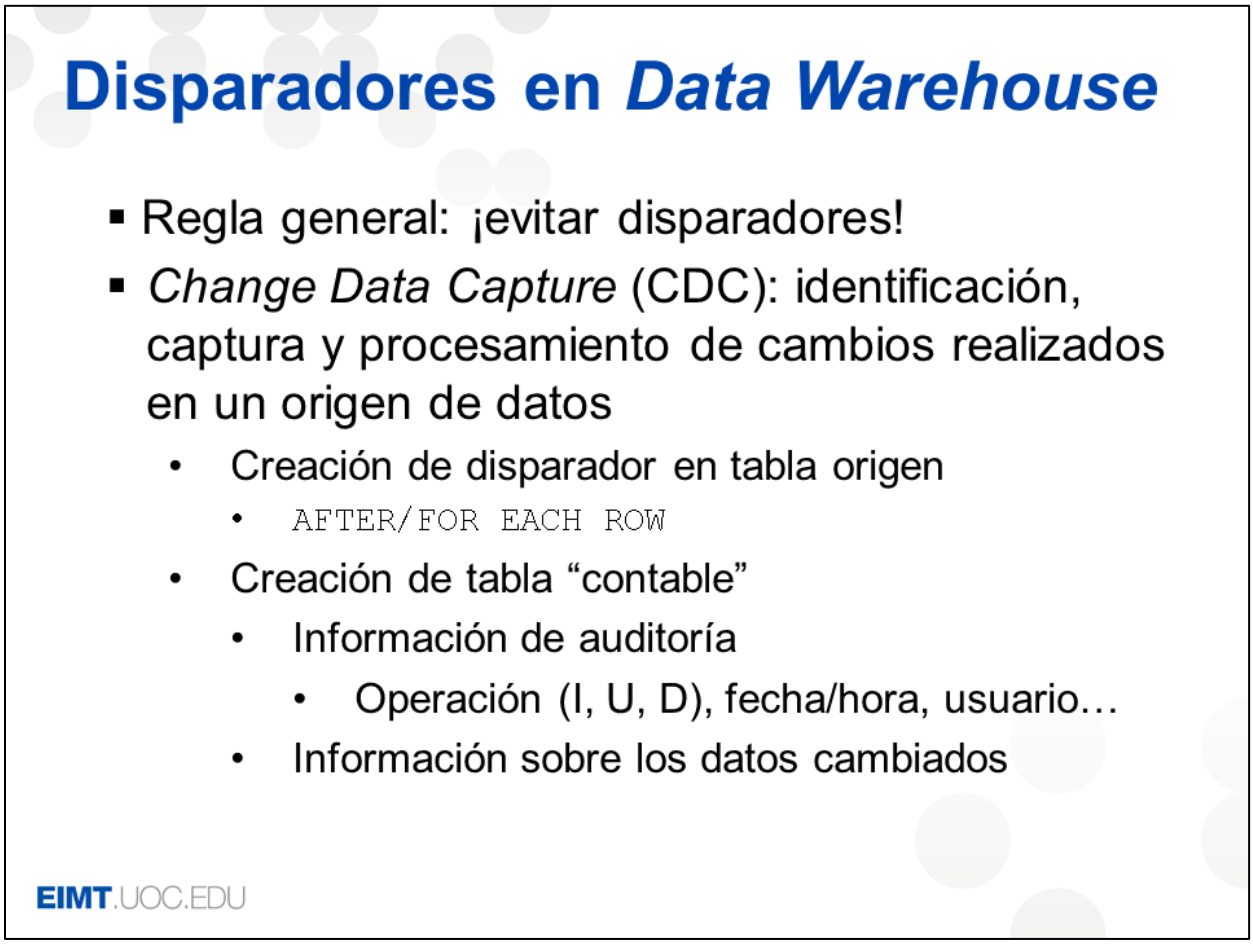
- **Comprobación de restricciones de integridad referencial**: como es el caso de las reglas dinámicas, que pueden necesitar referenciar al estado anterior y posterior de una modificación. Por ejemplo, la implementación de la regla *la nota de examen de un estudiante que solicita revisión no puede experimentar una bajada*. Esta regla no podría ser implementadas mediante una restricción de tabla o columna `CHECK`, por lo que la implementación de ésta mediante un disparador suele ser una buena solución.

- **Mantenimiento automático de una tabla de auditoría de la actividad en la BD**: el propósito de esta tabla de auditoría es la de registrar de una manera automática los cambios que se hacen en una tabla o tablas determinadas. Los datos recopilados en esta tabla de auditoría podrían ser utilizados para realizar estadísticas o incluso como origen de datos para un *Data Warehouse*. Este último punto lo comentaremos en la siguiente sección.

- **Mantenimiento de una réplica de datos**: pueden existir casos en los que es necesario mantener una réplica de los datos de una tabla de forma automática. Por ejemplo, si queremos almacenar una copia de una tabla de clientes, con la misma estructura que la original, y que tenga los datos sincronizados en tiempo real. Para ello, el uso de disparadores podría ser una buena solución.



Una vez vistos algunos de los diferentes usos de los disparadores en BD operacionales, vamos a ver el uso de estos componentes en el entorno *Data Warehouse*.



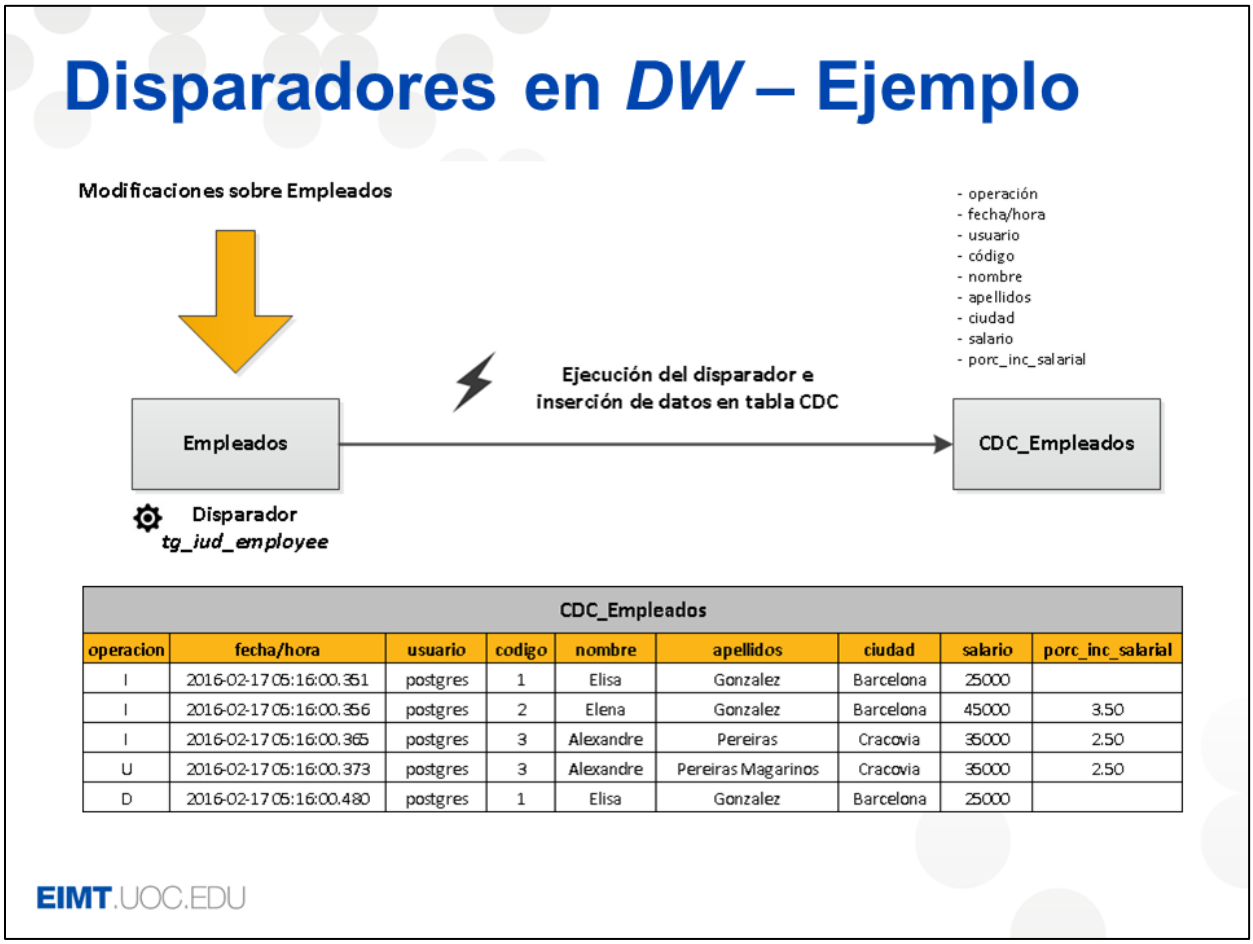
Como regla general, ¡en un *Data Warehouse* debemos de evitar los disparadores! La razón principal se debe a la naturaleza de este tipo de entornos: todas las operaciones de cálculo y reglas implementadas se realizan a través de procesos de carga o ETL (en inglés, *Extract, Transform and Load*), procesos que son lanzados con una frecuencia diaria y que, normalmente, se encargan de leer datos en los orígenes, realizar transformaciones y cálculos, y cargarlos en el entorno *Data Warehouse*. En general, podríamos decir que el uso dado a los disparadores en una BD operacional no debe ser dado en un entorno *Data Warehouse*.

Existe, en cambio, una excepción a la norma comentada y para ello es necesario presentar el concepto de *Change Data Capture* (CDC). *Change Data Capture* es un mecanismo de identificación, captura y procesamiento de datos muy utilizado en entornos *Data Warehouse* para identificar, capturar y procesar cambios realizados en los orígenes de los datos, de forma que los datos que cambian en origen puedan ser actualizados en nuestro *Data Warehouse*.

Existen diferentes formas de implementar CDC. Entre las dos más comunes, están el uso de una columna de tipo fecha/hora en cada tabla que registre el momento en que una fila se modifique, y el uso de una tabla *contable*. Ambas alternativas propuestas se basan en el uso de disparadores.

En el primer caso, la columna de tipo fecha/hora se podría actualizar mediante la creación de un disparador que asegure que por cada fila modificada, se actualice el campo con la fecha y hora actuales. Este mecanismo haría uso de un disparador de tipo `AFTER/FOR EACH ROW`, y tiene la desventaja de que no permite registrar borrados de filas, ya que una vez eliminada la fila, deja de existir en la tabla y no la podemos actualizar.

El uso de la tabla *contable*, el segundo caso mencionado, soluciona esta desventaja. Mediante el uso de un disparador similar al descrito en el caso anterior (`AFTER/FOR EACH ROW`), podemos capturar en una tabla de auditoría o tabla *contable* los cambios realizados en las tablas origen. En esta tabla *contable*, se registraría la operación realizada (`INSERT`, `UPDATE` o `DELETE`), el usuario y la fecha/hora en que se ha realizado la operación (entre otros), además de la información actual (e incluso anterior, si así se desea) de la fila afectada. De esta forma, podemos utilizar esta información para identificar y capturar los cambios realizados, sea cual sea la operación, y procesarlos según las reglas de nuestro *Data Warehouse*.



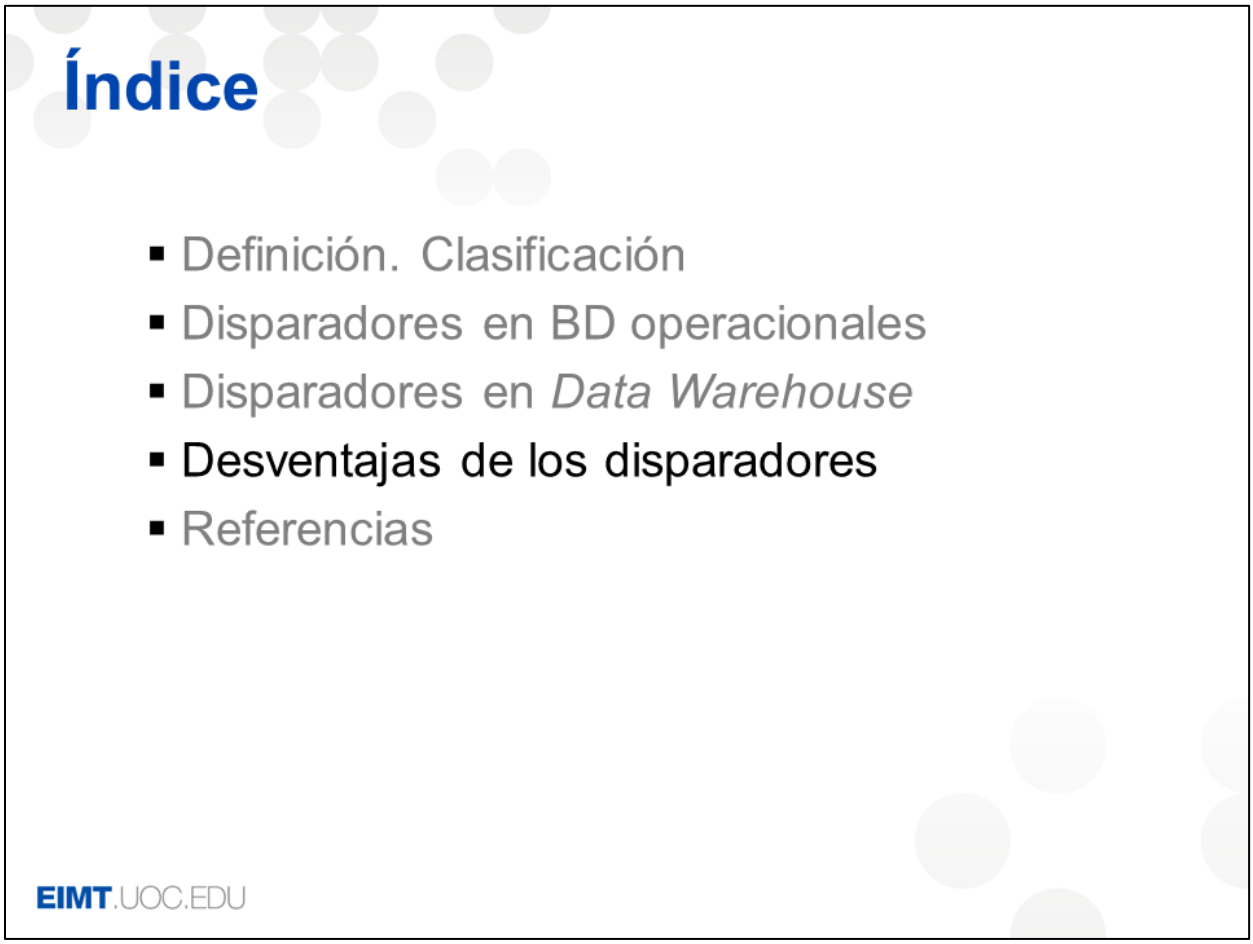
CDC_Empleados								
operacion	fecha/hora	usuario	codigo	nombre	apellidos	ciudad	salario	porc_inc_salarial
I	2016-02-17 05:16:00.351	postgres	1	Elisa	Gonzalez	Barcelona	25000	
I	2016-02-17 05:16:00.356	postgres	2	Elena	Gonzalez	Barcelona	45000	3.50
I	2016-02-17 05:16:00.365	postgres	3	Alexandre	Pereiras	Cracovia	35000	2.50
U	2016-02-17 05:16:00.373	postgres	3	Alexandre	Pereiras Magarinos	Cracovia	35000	2.50
D	2016-02-17 05:16:00.480	postgres	1	Elisa	Gonzalez	Barcelona	25000	

Veamos un ejemplo de implementación de CDC mediante disparador y tabla *contable*, ejemplo que podemos ver en la imagen superior.

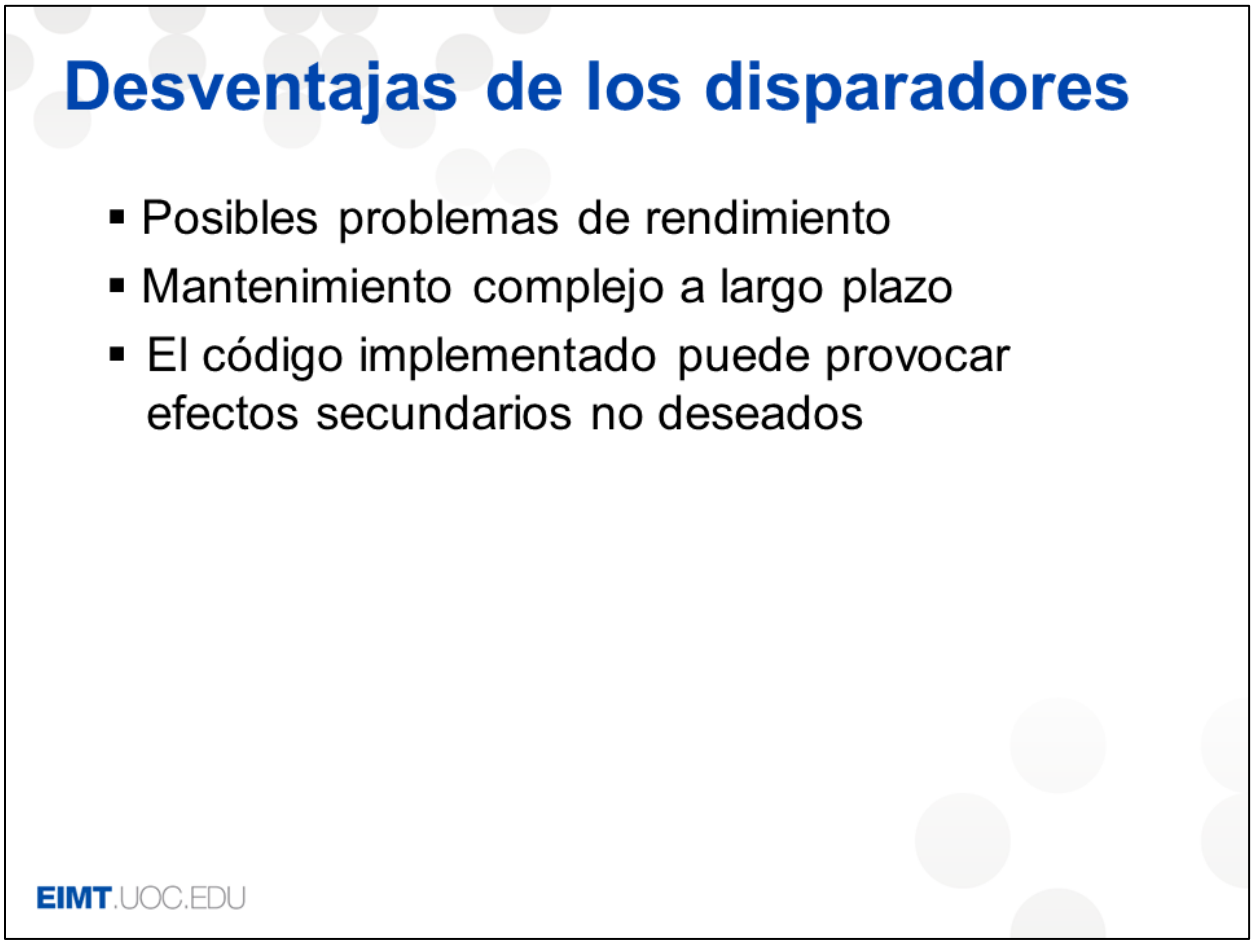
Supongamos una tabla de empleados con un disparador de tipo `AFTER/FOR EACH ROW` llamado *tg_iud_employee* que captura todas las operaciones sobre la tabla de empleados. Para cada fila modificada en la tabla de empleados, se ejecutará el disparador y se almacenará la información de los cambios (en el caso de `INSERT` y `UPDATE`, la nueva información tras la operación, y en el caso de `DELETE`, la información actual borrada) en una tabla *contable* denominada *CDC_Empleados*. En esta tabla contable, se almacena información de la operación (mediante un carácter, `I`, `U` o `D`), la fecha y hora del cambio, el usuario de la BD que ha realizado el cambio, y todos los datos de las columnas de la tabla empleados.

En la imagen inferior, podemos ver el resultado de realizar 5 operaciones consecutivas sobre la tabla de empleados utilizando la arquitectura propuesta, asumiendo que la tabla estaba inicialmente vacía:

- Primero, se añaden 3 empleados (Elisa, Elena y Alexandre) mediante 3 operaciones de `INSERT`.
- A continuación, se actualizan los apellidos de Alexandre, mediante una operación de `UPDATE`.
- Por último, se elimina el usuario Elisa, mediante una operación de `DELETE`.



Para finalizar, presentamos una serie de desventajas del uso de disparadores tanto en BD operacionales como en entornos *Data Warehouse*.

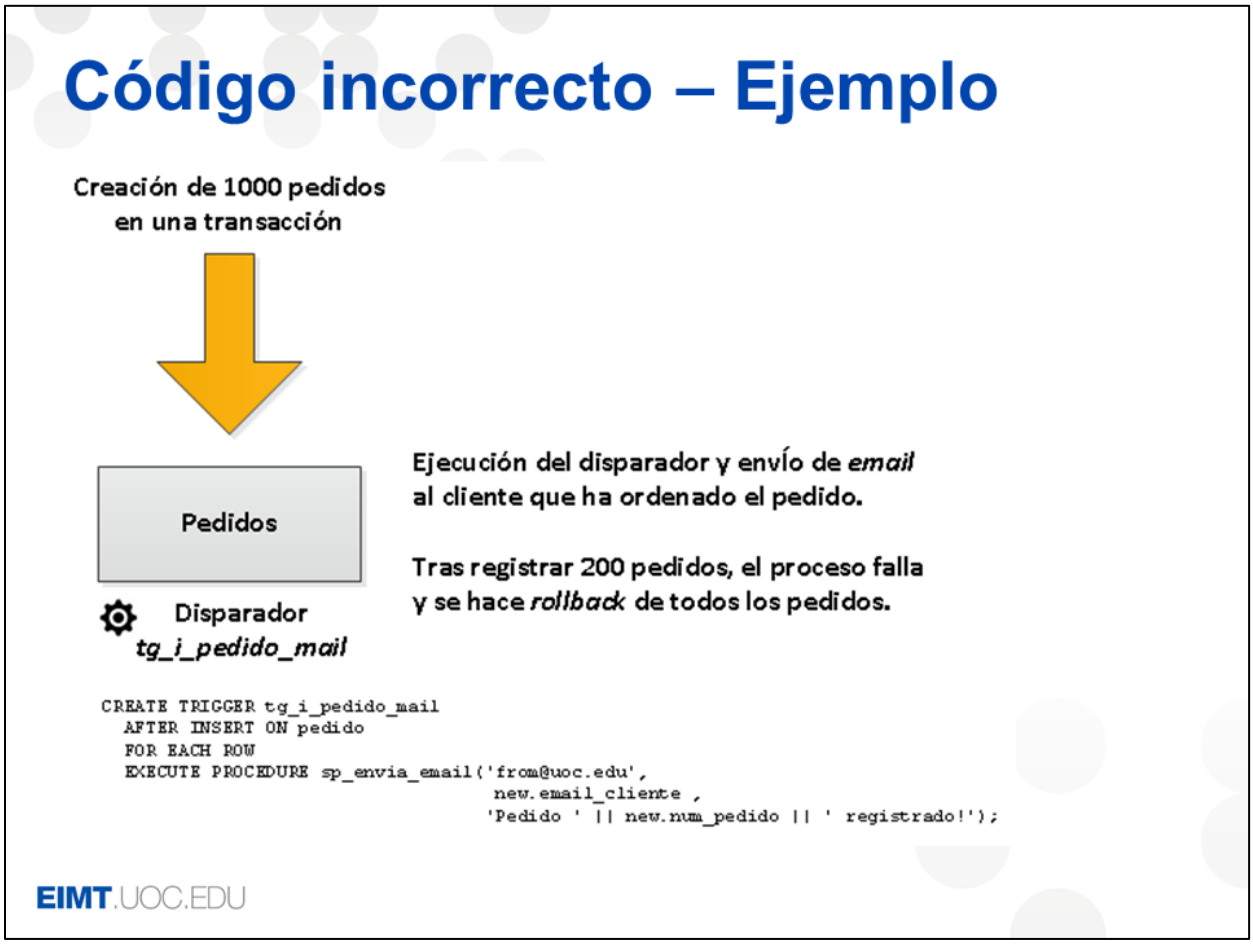


Una primera desventaja que podemos mencionar es que los disparadores pueden causar problemas de rendimiento si las operaciones realizadas sobre la tabla donde está definido el disparador afectan a una gran cantidad de filas. Este escenario se puede dar en entornos *Data Warehouse* cuando implementamos mecanismos de CDC, como hemos visto anteriormente. En el caso de que las operaciones en la BD origen impliquen actualizaciones masivas de datos, la implementación de disparadores para capturar datos cambiados puede no ser la mejor opción, ya que pueden causar problemas de rendimiento que afecten a las operaciones sobre la BD origen, perjudicando a los usuarios y a la actividad diaria de la empresa.

Otra desventaja del uso de disparadores es que el mantenimiento a largo plazo es muy complejo. Imaginad que heredáis el mantenimiento de una BD operacional plagada de disparadores que gestionan reglas de negocio. Este escenario puede convertirse en una auténtica pesadilla, ya que cualquier operación en la base de datos puede desencadenar una multitud de actualizaciones automáticas e incontrolables, en muchos casos sin que el usuario o administrador sepa de ellas. Si uno no está al tanto de la existencia de estos componentes o de la lógica implementada por éstos, puede resultar en un problema grave de la gestión de los datos.

Por último, es importante mencionar que el código implementado en disparadores puede provocar efectos secundarios no deseados si el código no está correctamente escrito desde un punto de vista de negocio y no sintáctico, o simplemente, si no se han anticipado las consecuencias de dicho código

dentro del contexto de las reglas de negocio. Estos casos suelen suceder cuando en el código implementado en los disparadores invocamos funciones auxiliares proporcionadas por el SGBD, funciones para las cuales no podemos deshacer los cambios mediante `ROLLBACK`.

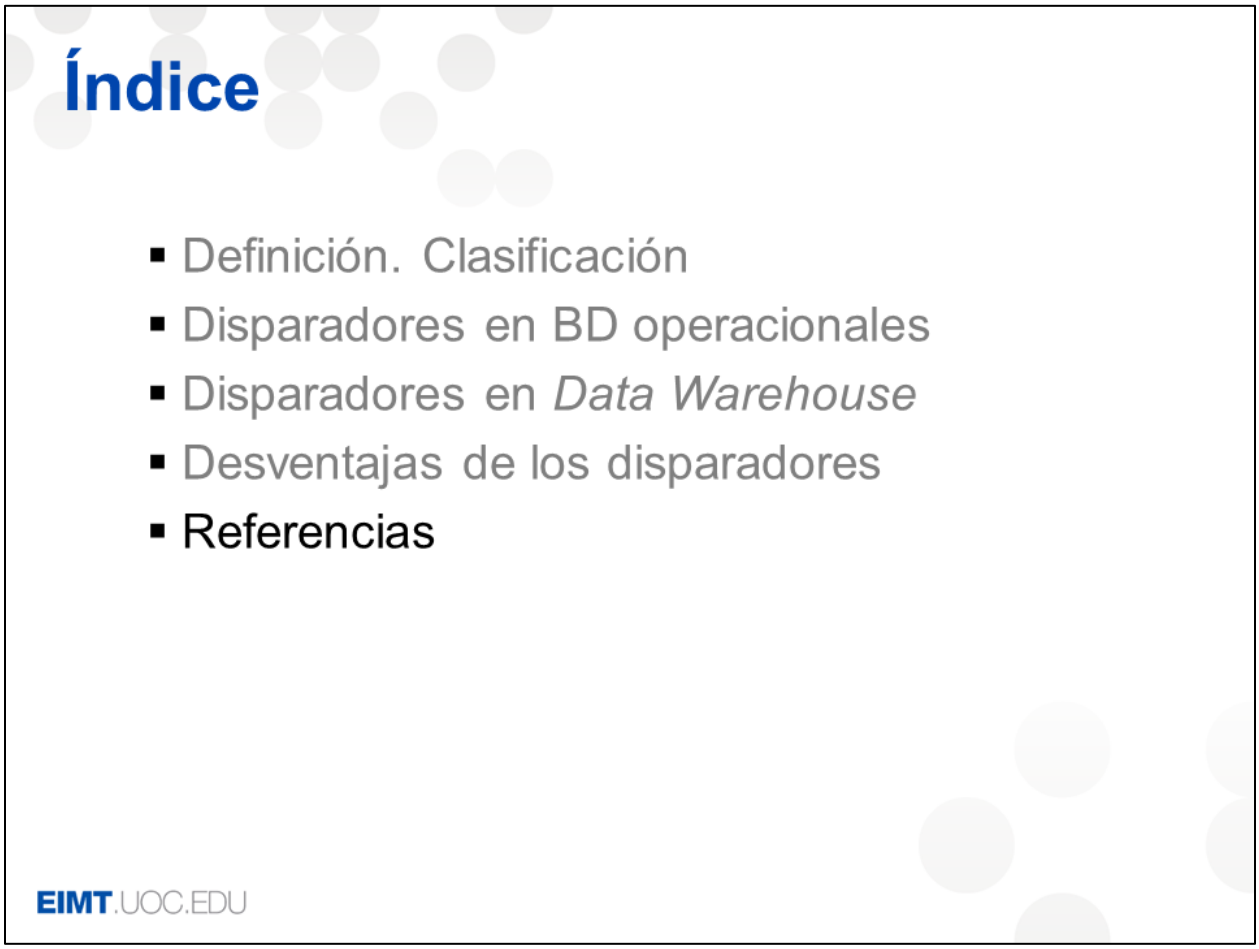


Veamos un ejemplo de efectos secundarios provocados por código implementado dentro de un disparador.

Supongamos que un proceso debe crear 1000 pedidos en la tabla de Pedidos, donde tenemos un disparador creado que ejecuta un procedimiento almacenado que envía un *email* al cliente una vez que el pedido está registrado en la tabla. La creación de estos pedidos se realiza dentro de una misma transacción, es decir, o se insertan los 1000 pedidos correctamente, o bien no se inserta ninguno.

Cuando ya se han insertado 200 pedidos, y por tanto enviado sus correspondientes *emails*, el proceso falla y realiza un *rollback* de los pedidos generados, por lo que la tabla de Pedidos volverá al estado inicial. ¡El problema reside ahora en que ya hemos enviado emails a muchos clientes de que sus pedidos han sido registrados!

Por lo tanto, como podéis ver, tenemos que ser muy cuidadosos con el código que se implementa dentro de un disparador, analizando minuciosamente los posibles efectos secundarios que éste puede tener al trabajar con transacciones en nuestra base de datos.



Y hasta aquí hemos llegado con este vídeo sobre la contextualización de disparadores. 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

Kyte, T. (2008). *The trouble with triggers*. Oracle Magazine.
http://www.oracle.com/technetwork/testcontent/o58asktom-101055.html

EIMT.UOC.EDU

¡Que tengáis un buen día!