Unidad VIII.- Diseño y desarrollo de aplicaciones cliente-servidor

Si Andalucía *es una empresa de software cada día más consolidada, y no para de recibir encargos de sus clientes, además de captar otros nuevos.* ***José*** *confía plenamente en la profesionalidad de su equipo, y por eso ha decidido poner a* ***Carmen*** *al frente del equipo de desarrollo de un Sistema Global de Planificación de Recursos (Enterprise Resource Planning o ERP) que ha encargado uno de sus nuevos clientes, para lo que contará con la ayuda de* ***Víctor****. El departamento de análisis, con* ***María*** *a la cabeza, les ha dado un informe detallado de los requisitos del cliente.*

Por este análisis saben que la empresa tiene dos sedes, bastante separadas entre sí, por lo que algunos módulos de la aplicación deberán ser accesibles por Internet, como los catálogos de artículos o la información de los pedidos de los clientes, necesaria para varios departamentos de la empresa, y que debe estar centralizada en una misma base de datos, que contendrá, de hecho, toda la información de la empresa.

Además ***José*** *tiene planeado que una vez desarrollada la solución, quiere poder comercializarla para distintas empresas, que obviamente usarán distintos SGBD. Además decide que aunque el ERP contendrá las reglas de negocio lo más globales posible, debe de existir la posibilidad de modificarlas o añadir nuevas a petición de los clientes, con el menor coste de desarrollo posible. Obviamente, también poder añadir nuevas funcionalidades con el mismo requisito.*

Para ***Carmen*** *parece claro que la mejor solución será optar por una arquitectura cliente-servidor para esta aplicación, y* ***Víctor*** *no sabe todavía demasiado sobre aplicaciones cliente-servidor, y no lo tiene tan claro, pero sí sabe que todavía quedan muchos más detalles por concretar, como saber si será una arquitectura n-capas, si se usaría cliente pesado o ligero, ...*

En las unidades precedentes has estudiado y practicado las diferentes **técnicas** utilizadas para analizar una situación del mundo real, y posteriormente llegar a obtener una representación modelizada de la realidad que permita su tratamiento informático por un sistema gestor de **bases datos.** Has aprendido a realizar estas tareas mediante herramientas CASE y también has aprendido el lenguaje SQL. Aunque todavía no tengas conocimiento de ello, como aprenderás en esta unidad, hasta ahora lo que has aprendido es **a diseñar y desarrollar la capa de datos de una aplicación cliente-servidor**. Pues bien, ha llegado el momento de aplicar todos estos conocimientos para aprender a diseñar y desarrollar **aplicaciones cliente-servidor**.

En esta unidad vas a aprender a diseñar y desarrollar la lógica de negocio de una aplicación en el Back-End. Para ello, aprenderás primero

- qué es y en que consiste la arquitectura cliente-servidor de n-capas,
- qué son las capas y niveles y a implementar procedimientos almacenados, funciones y triggers o disparadores.

Para ayudarnos en esta tarea usaremos las herramientas que nos suministra el entorno del **Oracle Application Express**, algunas de las cuales ya las has manejado en unidades anteriores y otras las conocerás en esta unidad.

CASO.

***Carmen** le explica a **Víctor** que si necesario que la aplicación sea fácilmente adaptable para otros clientes, permitiendo ajustarse a las reglas de negocio de cada uno de ellos, y que pueda ser extendida, incluyéndole nuevas funcionalidades, es necesario que sea **versátil**, **modular, flexible, escalable** y con una máxima interoperabilidad que le proporcione independencia respecto al SGBD. Todas ésas son las características que proporciona la arquitectura cliente-servidor. **Víctor** lo ve más claro, y está seguro de que mientras más conozca sobre esa arquitectura, más claro tendrá para proporcionar todas esas características tan deseables para la aplicación que quieren desarrollar.*

Como hemos visto en la introducción, vamos a profundizar en **la arquitectura cliente-servidor** hasta llegar a ser capaces de desarrollar aplicaciones basadas en este modelo. Pero para alcanzar este objetivo hemos de conocer cuáles son las características que distinguen a la arquitectura cliente-servidor de otros entornos.

- El término **cliente-servidor** apareció a principios de los **años 80** haciendo referencia a los ordenadores personales (PC´s) en una red. El modelo cliente-servidor comenzó a ganar aceptación a finales de los 80 hasta llegar a ser hoy en día el modelo de arquitectura más extendido como fruto del descenso de los costes de hardware, así como por los significativos avances tecnológicos alcanzados, tanto en el desarrollo de las máquinas servidor, como sobretodo, la aparición y las mejoras importantes en la capacidad de procesamiento de los PC´s y en el desarrollo continuado de las redes.

- La **arquitectura cliente-servidor,** llamada modelo cliente-servidor o servidor-cliente, es una forma de dividir y especializar programas y equipos de cómputo a fin de que la tarea que cada uno de ellos realiza se efectúe con la mayor eficiencia y permita simplificarla.

Desde el punto de vista del software, que es el que nos interesa en este módulo, la arquitectura del software del cliente-servidor es una infraestructura versátil, y modular que se desarrolla para mejorar la utilidad, flexibilidad, interoperabilidad, y ser escalable con respecto a la arquitectura centralizada o la arquitectura de servidor de archivos, las cuales estudiaremos a continuación. Pero tanto a nivel hardware como software, **en la arquitectura cliente-servidor, el objetivo que se pretende alcanzar es que la capacidad de proceso esté repartida entre el servidor y los clientes**.

- El **cliente** es quien **recibe los servicios que ofrece un servidor**. El término se usó inicialmente para dispositivos que no eran capaces de ejecutar programas por sí mismos, pero podían interactuar con ordenadores remotos por red. Estos terminales tontos eran clientes de los ordenadores centrales o mainframes de tiempo compartido.
- El **servidor** es una **aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes**. Algunos servicios habituales son los servicios de archivos, que permiten a los usuarios almacenar y acceder a los archivos de un ordenador y los servicios de aplicaciones, que realizan tareas en beneficio directo del usuario final. Éste es el significado original del término. Es posible que un ordenador cumpla simultáneamente las funciones de cliente y de servidor.

Aunque esta unidad va dedicada al diseño y desarrollo de aplicaciones en entornos cliente servidor, como podrás imaginar, no es esta la única forma de hacer las cosas. En otros módulos, de hecho, podrás ver otras formas de hacer las cosas, es decir, otras "arquitecturas" para el funcionamiento de las aplicaciones.

¿Cuáles son, por tanto, esas **arquitecturas** típicas para el software desarrollado?

Vamos a conocer los distintos tipos de arquitectura para así comprender mejor la arquitectura cliente-servidor:

- Arquitectura centralizada.
- Arquitectura de servidor de archivos.
- Arquitectura cliente-servidor.

Arquitectura centralizada

Este modelo de arquitectura se basa en la existencia de una máquina servidora que almacena los datos y las aplicaciones que los procesan.

- En este modelo de arquitectura, los clientes se comportan como terminales y sólo sirven para introducir datos desde teclado.

Ventajas:

- Gran nivel de seguridad.
- Facilidad de administración.

Desventajas:

- Alto coste.
- La máquina servidora está frecuentemente sobrecargada.

Arquitectura de servidor de archivos

Este modelo de arquitectura se basa en una o más **máquinas servidoras** que **almacenan** datosy **estaciones de trabajo** que **ejecutan** aplicaciones.

- Los clientes son **activos**, es decir, no son terminales o usuarios tontos.

Ventajas:

- Bajo coste.
- Escalable.

Desventajas:

- Alta dependencia de las comunicaciones.
- Alto tráfico en la red.

Cuando el equipo **n** lanza una consulta al fichero **A** (por ejemplo: **select * from empleados where departamento='contabilidad'**) el servidor de ficheros no devuelve el resultado de la consulta, sino el

fichero **A** completo, y es en el equipo **n** donde se ejecuta la consulta sobre el fichero **A**.

Obviamente esto produce un tráfico de datos innecesario que además sobrecarga la red.

Arquitectura cliente-servidor

CASO.

La aplicación ERP que quieren desarrollar maneja bastante información referente a pedidos y productos, clientes, proveedores y los propios comerciales que hacen las ventas. Lo que más peso tiene es la consulta de información, y las actualizaciones sobre la base de datos, no existiendo cálculos complicados sobre esa información. La mayoría de las veces se muestra tal y como se registra, sin ningún cambio. Por tanto, no parece que el peso de procesamiento de la lógica de negocio sea demasiado grande. Eso ha hecho a ***Carmen*** *decidirse por una arquitectura cliente-servidor con cliente ligero, que se limita a mostrar la* ***capa de presentación,*** *sin asumir ninguno de los procesos de cálculo asociados a la lógica de negocio, y que pueden ser atendidos por el servidor sin riesgo de sobrecargarlo.*

Víctor *le pide que le aclare a qué se refiere cuando habla de "capa de presentación", y* ***Carmen*** *le pone un ejemplo:*

¿Qué es lo que aparece en la pantalla del empleado de la oficina de un banco, cuando le pides que te haga un reintegro, por ejemplo? Verá una ventana, en la que hay una serie de menús que le permiten seleccionar la operación que va a realizar (Reintegro), que le pedirá que introduzca los datos del usuario, como por ejemplo tu DNI a través de un campo de texto, le permitirá elegir de una lista desplegable la cuenta concreta de entre las muchas que tengas abiertas en ese banco para hacer la operación, mostrando en un cuadro de texto no editable el saldo de tu cuenta, y en otro campo de texto la cantidad a reintegrar. Todo ello, claro está, con los oportunos botones de Aceptar, Cancelar, etc., e incluso mostrará un cuadro de diálogo indicando que la operación se ha hecho correctametne, o que ha habido algún problema que lo ha impedido,,,,, Es la forma en la que la aplicación presenta los datos que hay almacenados en su base de datos, y la forma en la que la apliacación pide los datos que necesita del usuario, es decir, el interface con el usuario. Eso es la capa de presentación, que estará hecha en un lenguaje de programación.

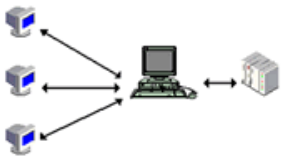
Detrás estarán los programas que recogerán toda esa información, y harán los cálculos, como comprobar que hay saldo suficiente para hacer la operación, y restarle la cantidad reintegrada al saldo de tu cuenta. Esas operaciones son la lógica o capa de negocio, es en lo que consiste el trabajo, realmente, y esta capa puede estar hecha en un lenguaje diferente. Finlamente, tanto los datos que hay que mostrar como los resultados de los cálculos están guardados en una base de datos a la que se accede usando algún lenguaje de manipulación de datos. Esto es la ***capa de acceso a datos****., y*

todo en conjunto conforma lo que se llama una arquitectura en tres capas.

*A **Víctor** le ha quedado mucho más claro lo que es la capa de presentación, desde luego,,, pero ahora hay que desarrollarla para la aplicación que traen entre manos....*

- Este modelo de arquitectura se basa en la existencia de **dos tipos de aplicaciones ejecutándose independientemente**.
- Una de las aplicaciones actúa como **servidora** y otra como **cliente**.
- El trabajo se reparte entre dos ordenadores. De acuerdo con la distribución de la lógica de la aplicación hay dos posibilidades:
  1. Cliente ligero (o cliente fino): si el cliente sólo se hace cargo de la presentación (capa de presentación).
  2. Cliente pesado (o cliente grueso): si el cliente asume también la lógica del negocio.

Ventajas:



- El servidor no necesita tanta potencia de procesamiento, parte del proceso se reparte con los clientes.
- Se reduce el tráfico de red considerablemente. Idealmente, el cliente se conecta al servidor cuando es estrictamente necesario, obtiene los datos que necesita y cierra la conexión dejando la red libre para otra conexión.

Desventajas:

- Alta dependencia de las comunicaciones.
- Instalación de nuevas aplicaciones. (Una nueva aplicación debe ser instalada en cada uno de los clientes que deseemos que la ejecuten)

Cuando el cliente **n** lanza una consulta sobre la base de datos (por ejemplo: **select * from empleados where departamento='contabilidad'**) el SGBD devuelve únicamente el resultado de la consulta siendo recibido el mismo por el cliente.

Para que comprendamos mejor la diferencia entre el modelo servidor de archivos con el cliente-servidor vamos a pensar en un **ejemplo**.

Supongamos que tenemos un fichero de 100.000 clientes. Supongamos ahora que de esos 100.000, sólo 10 habitan en Aguadulce. En un momento dado lanzamos una consulta para ver sólo los clientes de Aguadulce.

- Si estamos trabajando en una arquitectura de servidor de archivos, a nuestro equipo llega el fichero con los 100.000 registros, y es en nuestro equipo donde se ejecuta la consulta. Es decir, por la red ha sido transmitido un fichero que contiene 100.00 registros.
- Si por el contrario trabajamos sobre una aplicación cliente-servidor, lanzaremos la consulta, el SGBD la recibe, la ejecuta y devuelve solamente el resultado de la misma. Es decir, por la red han sido transmitidos tan solo 10 registros.

Autoevaluación

De las siguientes afirmaciones referidas a los distintos modelos de arquitectura, señala la que consideres correcta.

a) Una de las muchas ventajas del uso de una arquitectura

centralizada es el bajo coste de su implantación.

b) Tanto en la arquitectura centralizada como en la de servidor de archivos, los clientes son terminales o usuarios tontos.

c) El uso de una arquitectura de servidor de archivos no es aconsejable debido a su alto coste.

d) La arquitectura de servidor de archivos se basa en una o más máquinas servidoras que almacenan datos y estaciones de trabajo que ejecutan aplicaciones.

Comprobar

Arquitectura n-capas

CASO.

*Para el diseño de la aplicación ERP, **Carmen** opta por una arquitectura n-capas, concretamente de 3 capas (presentación, lógica de negocio, acceso a datos), ya que queremos que exista un reparto claro de funciones, y al mismo tiempo dotar a la aplicación de la mayor independencia para cada uno de esos bloques. De esta manera, se favorece el desarrollo en paralelo. Mientras **Víctor** se puede encargar de la capa de Acceso a datos, que ya domina, y de la capa de presentación, que también domina, **Carmen**, más experimentada, podrá desarrollar la capa de lógica de negocio, que es siempre bastante más delicada, ya que es la que se encarga de procesar la información vital de la empresa.*

Al mismo tiempo, si la capa de datos es independiente de las demás, apenas si habrá que hacer cambios en las otras dos capas cuando queramos adaptar la aplicación a una nueva empresa con un SGBD distinto, siempre y cuando la lógica de negocio sea similar.

***Víctor** le plantea la duda de cuántos **niveles** va a tener la aplicación. ¿Van a ser ordenadores distintos los que se ocupen de cada capa, o varias capas van a residir en un mismo ordenador?*

***Carmen** le cuenta que para esta empresa en concreto, con sólo dos sedes, y con una lógica de negocio simple que no genera procesos demasiado pesados, pensar en alojar cada capa en un ordenador diferente parece excesivo. Lo más adecuado, a su modo de ver es alojar la capa de datos y la de lógica de negocio en un mismo ordenador, que actuaría de servidor, y que estaría físicamente en una de las dos sedes, y alojar la capa de presentación en los clientes ligeros que se usarán de forma remota accediendo al servidor a través de Internet, y que estarán repartidos por todos los puestos de trabajo de las dos sedes de la empresa.*

Vamos a estudiar la **arquitectura n-capas o n-tier**. Este término surge como evolución de la arquitectura cliente-servidor. La arquitectura n-capas podemos afirmar que es una especialización de la arquitectura cliente-servidor.

¿Cuál es la diferencia entre arquitecturas cliente-servidor y n-capas?

En realidad nos referimos al mismo modelo, la diferencia es que en la

arquitecturacliente-servidor el software reparte su carga de cómputo en dos partes independientes pero **sin reparto claro de funciones** mientras que hablaremos de **arquitectura n-capas** cuando **exista un reparto claro de funciones**. Por ejemplo, una capa para la presentación, otra para el cálculo y otra para el almacenamiento (arquitectura de 3 capas). El modelo n-capas está especialmente indicado para el desarrollo en paralelo.

Pero antes de profundizar en este modelo hemos de entender qué son las capas y qué son los niveles.

El término capa hace referencia a la forma como una solución es segmentada desde el punto de vista lógico:

- Presentación/ Lógica de Negocio/ Datos.

En cambio, el término nivel**,** corresponde a la forma en que las capas lógicas se encuentran distribuidas físicamente. Por ejemplo:

- Una solución de tres capas (presentación, lógica, datos) que residen en un solo ordenador (Presentación+lógica+datos). Se dice, que la arquitectura de la solución es de tres capas y un nivel.
- Una solución de tres capas (presentación, lógica, datos) que residen en dos ordenadores (presentación+lógica, lógica+datos). Se dice que la arquitectura de la solución es de tres capas y dos niveles.
- Una solución de tres capas (presentación, lógica, datos) que residen en tres ordenadores (presentación, lógica, datos). La arquitectura que la define es: solución de tres capas y tres niveles.

Vamos a estudiar los tipos de capa y en qué consisten.

- Capa de presentación**:** es la que ve el usuario**, el interface de usuario**, presenta el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). **Esta capase comunica** únicamente con la **capa de negocio**.
- Capa de negocio o lógica de negocio**:** es donde residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues **es aquí donde se establecen todas las reglas que deben cumplirse**. **Esta capa se comunica con la capa de presentación**, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos el almacenamiento o recuperación de datos de él. Es la capa responsable de tener acceso a la capa de datos **para recuperar, para modificar y para suprimir datosen y desde** la capa de los datos y para enviar los resultados a la capa de la presentación. Esta capa es también responsable de procesar los datos recuperados y enviados a la capa de presentación

- Capa de datos**:** es donde **residen los datos**. Está formada por uno o más gestor de bases de datos que realizan todo el almacenamiento de datos, y reciben solicitudes de almacenamiento o recuperación de información desde **la capa de negocio**.

Autoevaluación

De las siguientes afirmaciones referidas a los modelos de arquitectura n-capas, señala la que consideres correcta.

a) El término capa hace referencia a la forma en que las capas lógicas se encuentran distribuidas de forma física.
b) Podemos afirmar que la arquitectura cliente-servidor es una especialización de la arquitectura n-capas.
c) Una de las desventajas del uso de la arquitectura n-capas es que no existe un reparto claro de funciones.
d) El término capa hace referencia a la forma como una solución es segmentada desde el punto de vista lógico.

Comprobar

Modelos de arquitecturas n-capas

Ya sabemos en qué consiste la arquitectura n-capas. ¿Te has preguntado el porqué del uso del término matemático n para denominar este modelo? Como sabes, en general se usa el término **n** cuando queremos indicar que se pueden tomar varios valores numéricos. De esta manera, al referirnos a la arquitectura n-capas, indicamos que n puede tener varios valores, es decir, que existen los modelos de una capa, de dos capas...etc.

Nosotros vamos a centrarnos en los modelos más extendidos. El modelo 2-capas y el modelo 3-capas.

Arquitectura de 2 Capas.

En una estructura 2-capas. **La lógica está partida en dos capas**, hechas generalmente partiendo de la lógica del acceso de los datos. Esto da lugar a la estructura mostrada en el siguiente cuadro:

Obviamente, la arquitectura de 2-capas también permite la estructura en que la primera capa contenga sólo la lógica de presentación y la segunda capa contenga la lógica de negocio y la lógica de acceso a datos (clientes ligeros). Es decir, con la arquitectura 2-capas hemos de seleccionar según los requisitos, si usaremos clientes ligeros o pesados.

La estructura de la figura (estructura 2-capas cliente pesado) presenta la **ventaja de que evita todas las complejidades de comunicarse con la base de datos a una capa separada**. Por lo tanto debe ser posible cambiar a un nuevo sistema de bases de datos (SGBD) sin realizar apenas cambios en el contenido de la capa de datos, ya que sólo habría que modificar la conexión con el SGBD. Si las operaciones y componentes de la primera capa siguen siendo constantes no debe haber necesidad de modificar ningún componente en la primera capa.

Por **ejemplo**: Hemos realizado una aplicación de dos capas cliente pesado en la que la capa de datos corre en el SGBD SQLServer. Si quisiéramos cambiar el SGBD, por ejemplo a Oracle, sólo habría que modificar en la capa de datos el código de conexión con la bases de datos (conectores de bases de datos), quedando la primera capa igual. Es decir, el coste de migración es mínimo (referido a modificaciones en la programación).

Arquitectura de 3 Capas.

Este modelo **divide cada una de las tres áreas lógicas en su propia capa**. Para que esta estructura trabaje con eficiencia deben desarrollarse interfaces bien definidos entre cada una de las capas. Esto debe permitir que se puedan modificar componentes de una capa sin requerir cambios a cualquiera de los componentes de otras capas. Un **ejemplo**, como hemos visto anteriormente, sería cambiar de un SGBD a otro, o cambiar todo o partes del interface de nuestra aplicación.

La **ventaja principal** de esta estructura sobre el modelo 2-capas es que **toda la lógica del negocio está contenida** en su **propia capa** y es **compartida** por muchos componentes en la capa de presentación. Cualquier cambio a las reglas de negocio puede por lo tanto ser realizado en un lugar y estar inmediatamente disponible a través del uso de todas las capas.

Autoevaluación

De las siguientes afirmaciones referidas al modelo de arquitectura n-capas, señala la correcta.

a) En una estructura de 2 capas cliente ligero, la primera capa contiene la lógica de presentación y la segunda la lógica de negocio y la de datos.
b) La ventaja principal del modelo 3-capas sobre el modelo 2-capas es que el modelo de 3-capas es mucho más fácil de implementar.
c) La estructura 2-capas cliente pesado, presenta la desventaja de que comunicarse con la base de datos mediante una capa separada es muy complejo.
d) En una estructura de 2 capas cliente pesado, la primera capa contiene la lógica de presentación y la segunda la lógica de negocio y la de datos.

Comprobar

Para saber más

En el siguiente enlace encontrarás una muy buena referencia de los patrones arquitectónicos de software.

Patrones Arquitectónicos

http://www.lsi.us.es/docencia/get.php?id=1130 [versión en cache]

El siguiente enlace contiene una serie de reflexiones sobre la conveniencia de la utilización de la arquitectura de 3-capas frente a la de 2-capas.

Arquitectura de aplicaciones y servicios

http://msmvps.com/blogs/pmackay/archive/2004/10/04/14900.aspx [versión en cache]

CASO:

Ya que una parte de los procesos van a ser lanzados a través de Internet por los usuarios desde sus clientes ligeros, ***Carmen*** *tiene claro que debe construir un sitio web usando el modelo de servidor de aplicaciones, que contendrá por lo menos tres "subcapas" en el back-end (capa del servidor web, capa del servidor de aplicaciones y capa de datos). De esta manera, en los ordenadores clientes de los usuarios se mostrará la interfaz gráfica que contendrá los resultados y capturará los datos del usuario y sus peticiones de procesamiento, además de hacer comprobaciones de formato sobre los datos (la fecha es correcta, los valores numéricos son válidos, etc.). Ese interfaz gráfico será suministrado por el servidor web de la aplicación. Cualquier operación que solicite el usuario, será enviada al servidor de aplicaciones, que tras consultar los datos en la base de datos y procesarlos correctamente, generará una página web para el usuario que le será enviada por el servidor web.*

Víctor *le pregunta si no se podría haber construido la aplicación cliente-servidor usando una red P2P (peer to peer).* ***Carmen*** *le explica que*

aunque se trata de una arquitectura cliente servidor, su uso es adecuado para compartir información en redes colaborativas, intercambiando información con el resto de usuarios de la red, de forma que cada usuario tiene un equipo que hace de cliente y de servidor al mismo tiempo. No obstante, no es apropiada para el negocio de la empresa que tienen por cliente, donde es necesario que toda la información esté centralizada en una sola base de datos que la mantenga actualizada para todos los usuarios.

Vamos a realizar un breve estudio de las **tendencias actuales** a las distintas arquitecturas, ya que las novedades son muchas y sería interminable comentarlas todas, por lo que nos ceñiremos a las más usadas actualmente.

En concreto conoceremos brevemente:

Los servidores Web, ya que hoy en día es imprescindible conocer todo lo relacionado con Internet.

Las redes Peer-to-peer como nuevas tendencias en los modelos de redes y porque seguro que las habéis utilizado alguna vez.

Servidor Web (WEB Server) y Servidor de Aplicaciones (Application Server)

Seguro que habrás oído infinidad de veces usar el término servidor web. La mayoría de las veces, como verás a continuación, el uso del término no es correcto, ya que se asocia erróneamente, entre otros, a la máquina o computador que nos permite la navegación por internet, o bien a la máquina o computador donde está alojada una página web. Como aprenderás a continuación, tanto un servidor web como un servidor de aplicaciones son el software que nos proporcionan una serie de servicios y que están íntimamente ligados a la programación de aplicaciones cliente-servidor.

Comenzaremos viendo el SERVIDOR WEB:

Un **web server o servidor web es un programa** que, usando el modelo del cliente-servidor y el protocolo de transferencia de hipertexto (http) del World Wide Web, sirve los archivos que forman páginas web a los usuarios de la Web. Cada computador en Internet que contiene un sitio web debe tener un programa web server.

Ejemplos de servidores web:

- Servidor de información de Internet de Microsoft (IIS), que viene con el servidor de Windows NT;
- Servidores de Nestcape FastTrack y de la empresa;
- Apache, un web server para Windows y UNIX .

Los servidores web integran a menudo servidor de correo y servidor FTP.

Veamos también el SERVIDOR DE APLICACIONES:

Los servidores de aplicaciones son software que nos ayudan a construir la separación lógica (a veces también física) entre la capa de datos y la lógica o capa de negocio.

Desarrollar un servidor de aplicaciones implica el ocuparse de muchas tareas complicadas:

- Gerencia de la conexión de red
- Diseñar una consola de la gerencia
- Balancear la carga
- Funcionamiento del marco del desarrollo

- etc.

Un servidor de aplicaciones puede simplificar el proceso de desarrollo de una aplicación de tres capas.

Los sitios Web construidos usando el modelo del servidor de aplicaciones contienen por lo menos tres "subcapas" en el back-end. Éstas son

1. Capa del servidor web.
2. Capa del servidor de aplicaciones.
3. Capa de datos.

En este modelo, la mayoría o toda la lógica de negocio está en los servidores de aplicaciones manejando todas las operaciones de creación y manipulación de datos.

Esquema de funcionamiento de una petición de un cliente Web a un sitio Web con servidor de aplicaciones

1. Un **usuario** conecta con su navegador mediante el protocolo http con el servidor de aplicaciones del sitio Web.
2. El **servidor de aplicaciones** establece una sesión de usuario y verifica la información de la conexión en forma cruzada contra una base de datos en una máquina separada.
3. Una página **Web** es creada por el servidor de aplicaciones permitiendo al usuario lanzar una consulta a la base de datos.
4. Las **consultas** se envían al servidor de aplicaciones, que establece un acoplamiento a la máquina y recupera los datos solicitados.
5. El **servidor de aplicaciones** recoge los datos y construye una nueva página Web para el usuario.
6. Los archivos pedidos en html son vueltos a dirigir por el servidor de aplicaciones a una tercera máquina, que utiliza un servidor web separado para producirlos.

Autoevaluación

En tu opinión, ¿qué son los servidores de aplicaciones?

a) Los servidores de aplicaciones son software que nos ayuda a construir la separación lógica entre la capa de datos y la lógica o capa de negocio.
b) Los servidores de aplicaciones son software que, usando el modelo del cliente-servidor y el protocolo de transferencia de hipertexto del World Wide Web, sirve los archivos que forman páginas web a los usuarios de la Web.
c) Los servidores de aplicaciones son software que integran a menudo servidor de correo y servidor FTP.
d) El que se encarga de suministrar al usuario servicios de correo.

Para saber más

Ésta es la página oficial de Apache Tomcat, desde aquí podrás descargarte este servidor web.

Apache Tomcat

http://tomcat.apache.org/ **[versión en cache]**

Aquí aprenderás cómo configurar un servidor de aplicaciones en el Windows 2003 Server.

Función de servidor de aplicaciones: Configurar un servidor de aplicaciones

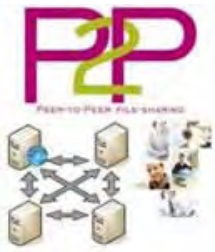
http://www.microsoft.com/technet/prodtechnol/windowsserver2003/es/library/ServerHelp/21dfa1f1-4dff-4a75-a797-42247b6c278a.mspx?mfr=true **[versión en cache]**

Este enlace te muestra, de forma resumida, los factores a tener en cuenta a la hora de seleccionar un servidor de aplicaciones.

Elección de un servidor de aplicaciones

http://livedocs.macromedia.com/dreamweaver/8_es/using/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=25_weba5.htm **[versión en cache]**

Peer-to-Peer

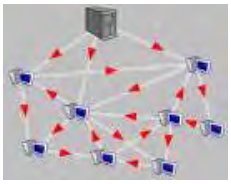


A grandes rasgos, una red informática **entre iguales** (en inglés *peer-to-peer* - que se traduciría de par a par- o de punto a punto, y más conocida como **P2P**) se refiere a una red que no tiene clientes y servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red. Este **modelo de red** contrasta con el modelo cliente-servidor, que se rige por una arquitectura monolítica donde no hay distribución de tareas entre sí, sólo una simple comunicación entre un usuario y un terminal en donde el cliente y el servidor no pueden cambiar de roles.

En la **comunicación P2P** cualquier nodo puede iniciar, detener o completar una transacción compatible. La eficacia de los nodos en el enlace y transmisión de datos puede variar según su configuración local, velocidad de proceso, disponibilidad de ancho de banda de su conexión a la red y capacidad de almacenamiento en disco.

Filosofía de las redes p2p (Peer-to-Peer)

El **P2P** se basa principalmente en la filosofía e ideales de que todos los usuarios deben compartir. Conocida como **filosofía P2P**, es aplicada en algunas redes en forma de un sistema enteramente meritocrático en donde "el que más comparta, más privilegios tiene y más acceso dispone de manera más rápida a más contenido". Con este sistema se pretende asegurar la disponibilidad del contenido compartido, ya que de lo contrario no sería posible la subsistencia de la red.



Aquellos usuarios que no comparten contenido en el sistema y con ello no siguen la filosofía propia de esta red, se les denomina "leechers". Estos muchas veces representan una amenaza para la disponibilidad de recursos en una red P2P debido a que únicamente consumen recursos sin reponer lo que consumen, por ende podrían agotar los recursos compartidos y atentar contra la estabilidad de la misma.

Funcionamiento

Debido a que la mayoría de los ordenadores domésticos no tienen una **IP** fija, sino que le es asignada por el proveedor (ISP) en el momento de conectarse a Internet, no pueden conectarse entre sí porque no saben las direcciones que han de usar de antemano.

La solución habitual es realizar una conexión a un servidor (o servidores) con dirección conocida (normalmente **IP** fija), que se encarga de mantener la relación de direcciones **IP** de los clientes de la red, de los demás servidores y habitualmente información adicional, como un índice de la información de que disponen los clientes. Tras esto, los clientes ya tienen información sobre el resto de la red, y pueden intercambiar información entre sí, ya sin intervención de los servidores.

> ***Para saber más***
>
> ***Ésta es la página de Emule, una de las aplicaciones más conocida del P2P. Desde este enlace podrás descargártelo.***
>
> ***Emule***
>
> ***http://www.emule-project.net/home/perl/general.cgi?l=17&rm=download*** **[versión en cache]**

> *CASO.*
>
> *A la hora de decidir el sistema a usar para generar la capa de negocio del back-end,* ***Carmen*** *no ha tenido que pensar mucho, ya que este tipo de trabajos en la empresa los hacen siempre usando Oracle Application Express. Para ello, un primer paso es crear bloques PL/SQL, y usar los procedimientos, funciones y triggers, controlando el manejo de excepciones.*
>
> 
>
> 
>
> ***Víctor*** *no tiene claro la diferencia entre el SQL que él ha venido usando, y el PL/SQL, pero con los ejemplos que le ha visto a* ***Carmen****, ve claro que sólo se trata de añadir sentencias de control de flujo al SQL, de forma que las distintas sentencias SQL puedan insertarse en esas estructuras para ser ejecutadas secuencial o de forma condicional, etc.*
>
> *Esto tiene grandes ventajas, algunas de las cuales son evidentes para* ***Víctor****, pero otras se las tiene que explicar* ***Carmen****. Pero en cualquier caso, rápidamente ha aprendido la sintaxis de creación de bloques PL/SQL, incluidos los bloques anónimos.*

Ha llegado el momento de que apliquemos lo visto hasta ahora en esta unidad de una forma **práctica**. Vas a aprender a generar la capa de negocio en el Back-End. Para ello utilizaremos el Oracle Application Express, aplicación que ya has conocido en unidades anteriores.

ORACLE DATABASE EXPRESS EDITION

Aprenderemos a crear bloques PL/SQL para poder utilizarlos posteriormente en nuestras aplicaciones, también conoceremos los procedimientos, funciones y triggers o disparadores. Es decir, vamos a aprender a implementar la capa de lógica de negocio en el back-end mediante el uso de procedimientos, funciones y triggers mediante las herramientas que nos suministra el entorno de Oracle Application Express.

Añadir funcionalidad PL/SQL a una aplicación

Vamos a aprender a usar código PL/SQL para de esta manera poder ampliar funcionalidades en las aplicaciones que aprenderás a desarrollar en las unidades posteriores.

Seguramente te estarás preguntando **¿Qué es PL/SQL?**

Bien, PL/SQL es la extensión o ampliación del lenguaje procedural del SQL. EL lenguaje ofrece un entorno de programación robusto que te posibilitará programar de forma procedural y/o mediante técnicas de programación orientada a objetos como la encapsulación, la ocultación de la información, y la sobrecarga de procedimientos o funciones. Con PL/SQL puedes implementar programación de alto nivel para el Oracle Database Server y el set de herramientas del mismo.

Diseño y desarrollo de aplicaciones cliente-servidor

¿Porqué usar PL/SQL en una aplicación cliente-servidor?

El principal motivo de usar PL/SQL en las aplicaciones es que ofrece construcciones procedurales como variables, constantes y tipos. También hay que tener en cuenta que el lenguaje PL/SQL ofrece construcciones selectivas e iterativas a SQL.

Las mayores ventajas de la utilización de PL/SQL son :

- **Integración de construcciones procedurales con SQL.** Es decir, el PL/SQL integra declaraciones de control y condicionales con SQL. Esto te ofrece mejor control sobre declaraciones SQL y su ejecución.
- **Reducción de congestión de red.** PL/SQL te permite combinar declaraciones SQL, de forma lógica como una unidad. La aplicación puede enviar el bloque completo a la base de datos en vez de enviar las declaraciones SQL una por una. Esto reduce el tráfico por la red.
- **Desarrollo modularizado o modular.** PL/SQL te permite agrupar declaraciones lógicamente relacionadas, en bloques. Puedes anidar bloques dentro de otros bloques más grandes para construir programas potentes. También puedes dividir tu programa en módulos más pequeños.
- **Integración con herramientas.** La máquina PL/SQL está integrada en herramientas de Oracle como HTML DB, Oracle Databese XE, Oracle Forms, Oracle Reports, etc. Al utilizar estas herramientas, la máquina PL/SQL disponible localmente procesa las declaraciones procedurales y solamente las declaraciones SQL se pasan a la base de datos.
- **Portabilidad.** Los programas PL/SQL pueden ejecutarse en cualquier sitio. Oracle Server corre independientemente del sistema operativo y de la plataforma. No necesitas adaptarlos a cada nuevo entorno.
- **Gestión de excepciones.** Una excepción es un error en PL/SQL que surge durante la ejecución de un bloque. PL/SQL te permite gestionar excepciones eficientemente. Puedes definir bloques independientes que se encarguen del manejo de excepciones.

Puedes considerar necesario utilizar aplicaciones de bases de datos que incluyan lógica de programación, declaraciones secuenciales y declaraciones SQL. Utilizando PL/SQL puedes construir aplicaciones resistentes al cambio a lo largo del tiempo, enfocados a mayor número de usuarios.

Crear un bloque PL/SQL

La construcción básica en PL/SQL es un bloque. Un bloque se compone de un conjunto de declaraciones SQL y/o PL/SQL, combinados y pasados a la máquina Oracle, todo en un movimiento.

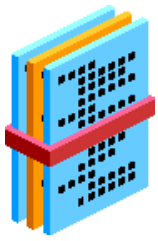
Un bloque PL/SQL se compone de tres secciones:

- **Declarativo** (opcional). Esta sección empieza por la palabra clave **DECLARE** y termina al empezar tu sección ejecutable.
- **Ejecutable** (obligatorio) Esta sección empieza por la palabra clave **BEGIN** y termina por **END**. La palabra clave END debería terminar con un punto y coma.
- **Gestión de excepciones** (opcional). La sección excepción está anidada en la sección ejecutable. Esta sección empieza por la palabra clave **EXCEPTION**.

Sección	Descripción
Declarativo (**DECLARE**)	Contiene declaraciones de todas las variables, constantes, cursores y excepciones definidas por usuario, referenciados en las secciones "ejecutable" y "excepción"
Ejecutable (**BEGIN...END**)	Contiene declaraciones SQL para rescatar datos de la base de datos y declaraciones PL/SQL para manipular datos en el bloque.
Excepción (**EXCEPTION**)	Especifica las acciones a realizar al surgir errores y condiciones anormales en la sección ejecutable.

Un programa PL/SQL se compone de uno o más bloques. Estos bloques **pueden estar** totalmente **separados o anidados dentro de otro bloque.**

Tipos de bloques



Existen tres tipos de bloques que componen un programa PL/SQL:

- **Bloques anónimos:** Estos son bloques PL/SQL sin nombre, encuadrados en una aplicación o emitidos interactivamente.
- **Procedimientos:** Estos son los llamados bloques PL/SQL. Estos bloques aceptan parámetros de entrada pero no devolverán valores explícitos.
- **Funciones:** Estos son los llamados bloques PL/SQL. Estos bloques aceptan parámetros de entrada y siempre devolverán un valor.

La diferencia entre un procedimiento y una función es que la función siempre debe devolver un valor al programa que le está llamando.

Autoevaluación

De las siguientes afirmaciones referidas a las secciones que componen un bloque PL/SQL, señala la que consideres correcta.

a) Gestión de excepciones (obligatoria). La sección excepción está anidada en la sección ejecutable.
b) Ejecutable (obligatorio). Esta sección empieza por la palabra clave declare y termina por end.
c) Declarativo (opcional). Esta sección empieza por la palabra clave declare y termina al empezar la sección ejecutable.
d) Declarativo (opcional). Esta sección empieza por la palabra clave declare y termina al empezar la sección excepción.

Comprobar

Crear un bloque anónimo

Los bloques anónimos son bloques sin nombre. Están declarados en línea en el punto de una aplicación donde deberán ser ejecutados y son compilados cada vez que la aplicación es ejecutada.

Características de los Bloques Anónimos

- Los bloques anónimos no se almacenan en la base de datos y se pasan a la máquina PL/SQL para su ejecución en el momento de la ejecución.
- No podrás invocar o llamar el bloque que escribiste anteriormente porque los bloques son anónimos y no existen tras haber sido ejecutados.
- Los bloques anónimos pueden utilizarse como bloques dentro de procedimientos, funciones y otros bloques anónimos.

La **sintaxis** para la creación de un bloque anónimo en PL/SQL es la siguiente:

```
[DECLARE
variables, constantes, excepciones de
usuario...]
BEGIN
<órdenes SQL>
<órdenes PL/SQL>
[EXCEPTION
acciones a realizar al ocurrir un error]
END;
/
```

Vamos a realizar un **ejemplo** de uso de un bloque anónimo.

Para ello vamos a suponer que estamos desarrollando una aplicación en la cual necesitamos una funcionalidad que nos indique el nombre de usuario que tiene abierta una sesión en ese momento y deseamos también la fecha y hora en que se genera el informe. Desarrollaremos un **bloque anónimo** que recupere el nombre de usuario y lo guarde en una variable llamada v_user. La fecha de informe del usuario debe ser mostrada en el formato DD MM YY.

Para crear el bloque anónimo en Oracle Express, deberemos introducir las sentencias SQL que se detallan a continuación, para ello el Oracle nos suministra la herramienta **Introducir Comando**.

El código sql para crear el bloque descrito es el siguiente:

```
DECLARE
v_user varchar2 (30);
 BEGIN
 SELECT SYS_CONTEXT('USERENV', 'current_user') INTO
v_user from dual;
 dbms_output.put_line('Informado en: ' ||
TO_CHAR(SYSDATE, 'DD-Mon-YY HH:MIPM'));
 dbms_output.put_line('User = ' ||v_user);
END;
```

Usamos la función SYS_CONTEXT, esta función devuelve los atributos de sesión.

Sintaxis de la función sys_context:

sys_context (nombre de espacio, parámetro)

usereven nombre de espacio que describe la sesión en curso

current_user es el parámetro válido para el nombre de espacio userenv y devuelve el nombre del usuario de la sesión en curso.

Vamos a ver de forma práctica como crear y ejecutar el bloque descrito.

Crear un bloque anónimo

CASO.

***Carmen** está muy atareada con la programación de la capa de negocio. **José** le comenta que debe usar Triggers o disparadores para garantizar la seguridad en determinadas operaciones que debe realizar la aplicación. **Víctor** no sabe lo que son los Triggers, y **José** le indica que no son más que bloques PL/SQL asociados a una tabla o vista y que se ejecutan siempre que se produce un determinado evento. Cuando **José** se va, **Víctor** le pide un ejemplo a **Carmen**, y ésta le muestra uno que tiene programado: Se trata de impedir que los usuarios hagan modificaciones sobre la base de datos, tales como cambiar los datos de un pedido, fuera del horario de trabajo. Cada vez que se va a hacer una inserción, modificación o borrado sobre la tabla Pedidos, se lanza el trigger, que comprueba la fecha y hora, y si no es dentro del horario de trabajo, impide la modificación de los datos, mandando un mensaje de error.*

*Otro trigger que ha construido **Carmen** es para evitar que ningún usuario no autorizado pueda cambiar los datos personales de otro usuario. Así, por ejemplo, otros usuarios podrán consultar las direcciones o teléfonos de los demás, pero sólo el propio usuario puede actualizarlos. Cada vez que se va a ejecutar una actualización de los datos de un usuario, se lanza un trigger que comprueba la identidad del usuario que se quiere modificar, y si es el mismo que hace la modificación, la permite, pero en caso contrario no, y además muestra un mensaje de error.*

De la misma forma, le comenta que tiene que hacer otro trigger para que automáticamente se cree un registro de las operaciones realizadas sobre la tabla Pedidos, de forma que se sepa qué usuario, desde qué puesto, y en qué día y hora ha hecho cualquier modificación sobre esa tabla, una especie de auditoría sobre la tabla.

Cuando trabajamos con una aplicación de bases de datos, puedes querer añadir funcionalidades propias de la programación que se ejecutan cuando operaciones específicas tienen lugar en la base de

datos. Por **ejemplo**, pueden darse las siguientes situaciones.

- Ejemplo 1: Puedes querer insertar datos en una tabla. Te encuentras con que los datos nuevos son inconsistentes con los datos existentes en la tabla. Podrías querer que el sistema dé un error que deshaga (Roll back) la transacción.
- Ejemplo 2: Puedes querer que el sistema guarde información como la hora y los detalles de una modificación de datos en una tabla por parte del usuario.

Los triggers o disparadores, también llamados a veces desencadenadores, ayudan al usuario a solucionar éstas y otras situaciones complicadas. Un trigger o disparador es un bloque PL/SQL o un procedimiento PL/SQL asociados a una tabla, vista, esquema o base de datos. Los triggers se utilizan para asegurar la integridad de datos, revisando datos de forma consistente. Un trigger se ejecuta implícitamente siempre que un evento específico tiene lugar.

¿Cuándo usar triggers?

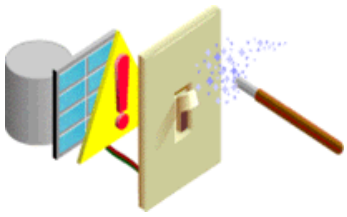
Las situaciones que aconsejen el uso de un disparador son muy diversas y sería muy extenso enumerarlas todas. A continuación se describen las principales funcionalidades de los triggers aplicables a la capa de negocio para que las tengas en cuenta a la hora de decidir la conveniencia o no del uso de disparadores en tu aplicación.

- Los triggers ofrecen **chequeos de seguridad** ampliados y complejos.
- Los triggers refuerzan la **integridad** dinámica de datos y restricciones de integridad referencial compleja.
- Los triggers **aseguran** que las operaciones relacionadas son ejecutadas juntas en su totalidad.

Tipos de Triggers o disparadores

Antes de profundizar en cómo crear un trigger, hemos de conocer qué tipos de disparadores existen y sus características.

Los triggers pueden ser los siguientes:



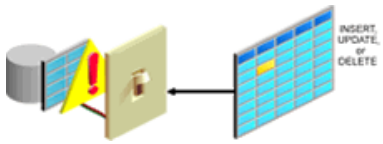
- **Triggers de aplicación**: Este trigger se dispara siempre que un evento tiene lugar con una aplicación en particular.
- **Trigger de base de datos**: Este trigger se dispara siempre que tiene lugar un evento de datos (como DML) o un evento de sistema (como un logon o shutdown) sobre un esquema o base de datos.

Es también necesario que tengamos en cuenta las siguientes **pautas para el diseño de los triggers**:

- Puedes diseñar triggers para llevar a cabo acciones relacionadas y centralizar operaciones globales.
- No debes diseñar triggers donde la funcionalidad ya está incluida en el SGBD.
- Evita diseñar triggers que dupliquen otros triggers.
- Puedes crear procedimientos almacenados e invocarlos en un trigger si el código PL/SQL es muy largo.
- Debes de tener en cuenta que un uso excesivo de triggers que se ejecutan en tablas que presenten relaciones de interdependencia complejas, pueden llegar a ser bastante difíciles de mantener en aplicaciones grandes.

Generación de un Trigger o disparador DML

Vamos a aprender la sintaxis para la creación de un trigger DML. Hemos de tener en cuenta que en esta unidad estamos usando Oracle Express, es decir, usaremos sentencias en PL/SQL.

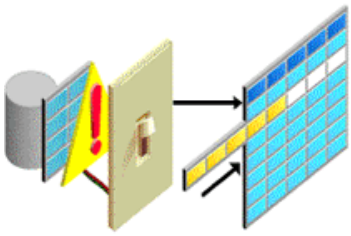


La sintaxis para crear un trigger DML en PL/SQL es la siguiente:

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
event1 [OR event2 OR event3]
ON object_name
[ [REFERENCING OLD AS old/NEW AS new]
FOR EACH ROW
[WHEN (condition)] ]
trigger_body
```

Analicemos los componentes de la sintaxis:

- **Nombre del trigger**. Trigger_name, identifica el trigger de forma única.
- **Momento**. Timing. Momento indica cuándo el trigger se dispara en relación al evento disparador, es decir, si es lanzado antes o después de realizar la operación que lo lanza. Los valores son BEFORE **o** AFTER(antes o después).
- **Evento**. Evento identifica la operación DML que causa el disparo del trigger. Es decir, si es lanzado al insertar, al actualizar o al borrar de una tabla. Los valores son INSERT**,** UPDATEyDELETE**.**
- **Nombre de objeto**. Object_name, indica la tabla o la vista asociada al trigger.
- **Referenciar**. La cláusula REFERENCING se utiliza para elegir un nombre de correlación para referenciar los valores antiguos y nuevos de la línea actual. REFERENCING nos permite asignar un alias a los valores NEWo/yOLD de las filas afectadas por la operación. Si el disparador es lanzado al insertar, el valor antiguo no tendrá sentido y el valor nuevo será la fila que estamos insertando. Para un disparador lanzado al actualizar, el valor antiguo contendrá la fila antes de actualizar y el valor nuevo contendrá la fila que vamos a actualizar. Para un disparador lanzado al borrar sólo tendrá sentido el valor antiguo. Sólo podrá ser utilizados con disparadores para filas.
- **Cuando**. La cláusula WHEN se utiliza para aplicar un predicado condicional, entre paréntesis, que es evaluado en cada línea para determinar si ejecutar el cuerpo del trigger. WHEN nos permite indicar al disparador que sólo sea lanzado cuando sea TRUE una cierta condición evaluada para cada fila afectada.
- **Cuerpo del trigger**. El trigger_body es la acción llevada a cabo por el trigger.



Estamos analizando la sintaxis para crear un trigger DML, por lo tanto hemos de conocer qué tipos de triggers DML existen. Son los siguientes:

- **Trigger de declaración** . Un trigger de declaración es el tipo de trigger por defecto y se ejecuta una vez por el evento disparador. Un trigger de declaración se dispara incluso si no afecta a ningún registro.
- **Trigger de fila o registro**. Un trigger de fila se ejecuta una vez por cada fila afectada por el evento disparador. Un trigger de fila no se ejecuta si el evento disparador no afecta a ninguna fila.
  Para indicar un trigger de fila se especifica FOR EACH ROW.

Autoevaluación

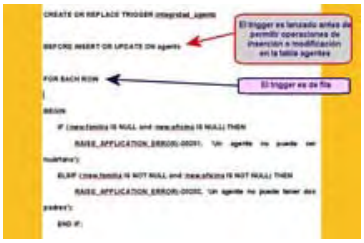
De las siguientes afirmaciones referidas a los triggers, señala la que consideres correcta.

a) Un trigger de aplicación se dispara siempre que tiene lugar un evento de sistema.
b) Un trigger de base de datos se dispara siempre que tenga lugar un evento de datos y un evento de sistema.
c) Un trigger de fila no se ejecuta si el evento disparador no afecta a ninguna fila.
d) Un trigger de fila se ejecuta una vez para todas las fila afectadas por el evento disparador.

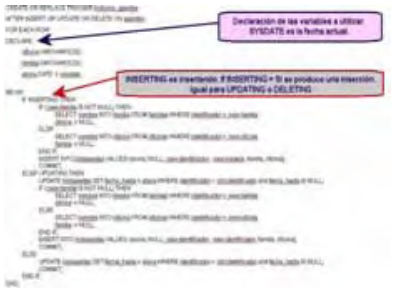
Comprobar

Ejemplos de Triggers. Uso e implementación

Ejemplo 1: Supongamos que tenemos una tabla agentes. **Restricción**: un agente debe pertenecer a una familia o una oficina pero no puede pertenecer a una familia y a una oficina a la vez. Para poder cumplir esta restricción deberemos implementar un disparador que la lleve a cabo ya que Oracle no nos permite definirla.



Ejemplo 2: Supongamos que tenemos una tabla de históricos para agentes que nos permita auditar las familias y oficinas por la que ha ido pasando un agente. La tabla tiene la fecha de inicio y la fecha de finalización del agente en esa familia u oficina, el identificador del agente, el nombre del agente, el nombre de la familia y el nombre de la oficina. Queremos hacer un disparador que inserte en esa tabla.



Ejemplo 3: Queremos realizar un disparador que no nos permita llevar a cabo operaciones con la tabla familias si no estamos en la jornada laboral.

Crear un Trigger DML en ORACLE Application Express

Vamos a crear el trigger del ejemplo 3 usando el Oracle Application Express. Modificaremos el **código** de dicho ejemplo de manera que **no** permitiremos operaciones de inserción ni modificación ni borrado en la tabla empleados si dichas operaciones son realizadas fuera del horario laboral (suponemos que el horario laboral será de lunes a viernes de las 08:00 a las 18:00 horas).

Para la creación y manipulación de triggers, Oracle Application Express nos suministra la herramienta **CREAR DISPARADOR** del **EXPLORADOR DE OBJETOS**.

Ya solamente nos queda ver de forma práctica cómo implementar el trigger en Oracle Application Express. Es algo que podrás ver en la siguiente presentación.

Cómo implementar el trigger en Oracle Application Express

Y ahora puedes comprobar el funcionamiento del trigger que hemos creado. Ejecuta la animación que hay a continuación para verlo en funcionamiento.

Comprobar el funcionamiento del trigger

La gestión de los Triggers

Un trigger tiene dos estados o modos: **ENABLED** y **DISABLED**. En el momento de crear un trigger, éste estará habilitado (enabled) por defecto. El Oracle Server revisa las restricciones de integridad en

busca de triggers habilitados y garantiza que los triggers no puedan comprometer las operaciones a realizar.

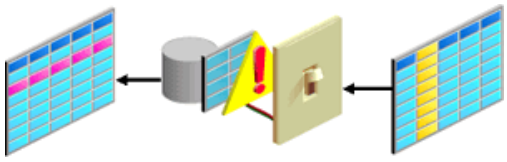
En nuestra aplicación, si por algún motivo deseamos asegurarnos que no exista ningún trigger que nos restrinja las operaciones a realizar sobre alguna tabla, debemos deshabilitar los posibles triggers que afecten a dicha tabla, es decir, pasarlos al estado DISABLED.

Hemos de obrar del mismo modo si deseamos deshabilitar o "desactivar" todos los triggers de una base de datos.

Para ello hemos de utilizar la siguiente sintaxis:

- **ALTER TRIGGER** trigger_name **DISABLE**. Deshabilita un trigger de base de datos
- **ALTER TABLE** table_name **DISABLE ALL TRIGGERS**. Deshabilita todos los triggers para una tabla
- **ALTER TRIGGER** trigger_name **COMPILE**. Recompila un trigger para una tabla

Funciones de auditoría usando Triggers



Es posible que desees hacer un **seguimiento** de las siguientes operaciones que se realicen en una aplicación de gestión:

- Inserción de nuevos datos
- Eliminación de datos existentes
- Modificación de datos existentes

La **auditoria de aplicación** se utiliza cuando deseamos tener un registro de las operaciones realizadas en alguna tabla.

Los triggers están especialmente indicados para realizar esta función, aunque también podría realizarse con procedimientos almacenados.

Vamos a implementar a continuación una **función de auditoría** mediante Oracle Express, mediante la cual guardaremos el valor de un campo antes de ser modificado, valor después y fecha y hora de la operación.

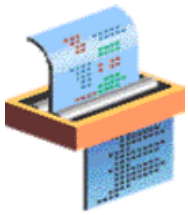
Para crear un sistema de auditoría debe existir una tabla auditora donde guardar la información deseada. Esta tabla se utiliza para el seguimiento de los cambios en los datos. Al dispararse el trigger de la base de datos, se insertan los "expedientes" en la tabla de auditoría. Puedes verlo de forma práctica en la siguiente animación.

Una función de auditoría mediante Oracle Express

CASO.

***Víctor** está acostumbrado a usar procedimientos a los que se les pasa una lista de parámetros, y se pregunta si PL/SQL permite hacer algo parecido. **Carmen** le dice que sí, que eso son, justamente, los procedimientos almacenados: bloques de código a los que se les da nombre y que agrupan una serie de declaraciones, tanto SQL como PL/SQL.*

***Carmen** le muestra uno de los procedimientos almacenados que ha creado. En él ha incluido una sentencia de SQL para actualizar el sueldo de un trabajador con un porcentaje. Este procedimiento admite como parámetros tanto el código del empleado al que se le va a actualizar, como el porcentaje de incremento. De esta forma, podremos usar el mismo procedimiento almacenado con todos los empleados que sea necesario sin más que indicar el código del trabajador y el porcentaje de incremento a aplicar. Además, ese procedimiento ha sido compilado y se mantiene en la base de datos, de forma que las siguientes veces que se invoque su ejecución será más rápido. Por supuesto, podrá usarse para definir otros procedimientos almacenados. Por ejemplo, podría haber alguna vez un procedimiento para cambiar a un empleado de puesto de trabajo. Si en el nuevo puesto el empleado cobra más, el procedimiento de incrementarle el sueldo podría ser invocado, sin necesidad de escribirlo ni definirlo entero de nuevo.*



Los procedimientos almacenados son bloques nombrados de código que posibilitan la agrupación y organización de una serie de declaraciones SQL y PL/SQL. Tanto el código fuente como el código ejecutable se almacenan en la base de datos. Al almacenarlos en la base de datos, el código se encuentra en una ubicación centralizada y accesible y eso hace que **la invocación al código almacenado sea muy eficiente.**

Los procedimientos almacenados pueden ser llamados de forma **interactiva** desde aplicaciones cliente, desde otros procedimientos almacenados, y también desde los disparadores o desencadenadores. Los procedimientos almacenados **pueden aceptar y devolver parámetros, también diversos conjuntos de resultados, y un código de estado**.

Los procedimientos almacenados son un buen lugar donde programar las **reglas de negocio de las aplicaciones**, y generalmente son más eficientes que programar tales reglas en los programas ejecutables en la parte "cliente".

Entre las **principales características** de un procedimiento almacenado se pueden mencionar:

- **Facilitan la reutilización y mantenimiento**. Al ser validados, pueden utilizarse en cualquier número de aplicaciones. Si los requerimientos cambian, solamente debe actualizarse el código.
- **Aceptar parámetros de entrada y devolver** varios valores en forma de **parámetros de salida** al lote o al procedimiento que realiza la llamada.
- **Contener instrucciones de programación** que realicen operaciones en la base de datos, incluidas las llamadas a otros procedimientos.
- **Devolver un valor de estado** que indica si la operación se ha realizado correctamente o ha habido un error (y el motivo del mismo).
- **Permiten una ejecución más rápida**, ya que los procedimientos son analizados y optimizados en el momento de su creación, y es posible utilizar una versión del procedimiento que se encuentra en la memoria después de que se ejecute por primera vez.
- Pueden **reducir el tráfico de red**.
- **Pueden utilizarse como mecanismo de seguridad**, ya que se puede conceder permisos a los usuarios para ejecutar un procedimiento almacenado, incluso si no cuentan con permiso para ejecutar directamente las instrucciones del procedimiento.

Crear procedimientos almacenados

Los procedimientos almacenados se crean con la declaración CREATE PROCEDURE, que puede declarar una lista de parámetros y debe definir las acciones a realizar por el bloque estándar PL/SQL. La cláusula CREATE te posibilita crear procedimientos independientes que se guarden en la base de datos Oracle.

La sintaxis de un procedimiento almacenado en PL/SQL es la siguiente:

```
CREATE [OR REPLACE] PROCEDURE nombre_procedimiento (
param [IN | OUT | INOUT] <tipo> [, param [IN|OUT|INOUT]
<tipo>,...]
IS|AS
[DECLARE]
       [<nombre de variables locales>]
BEGIN
       <código del procedimiento>
[EXCEPTION]
END [nombre];
```

- Los bloques PL/SQL **empiezan** por BEGIN, opcionalmente precedido por la declaración de variables locales. Los bloques PL/SQL **terminan** mediante ENDo END nombre_procedimiento.
- La opción REPLACE indica que si el procedimiento existe, éste es descartado y reemplazado por la nueva versión creada por la declaración.
- La opción DECLARE se utiliza si deseamos usar variables locales en el procedimiento.
- La opción EXCEPTION se usa para manejar los posibles errores que se generen en la ejecución.
- Cuando se crea un procedimiento, éste se compila en primer lugar y queda almacenado en la base de datos de forma compilada. El código compilado puede ser posteriormente utilizado por cualquier bloque PL/SQL.
- Para modificar un procedimiento creado debemos reemplazarlo por el nuevo volviendo a compilarlo añadiendo las palabras clave **OR REPLACE.**
- Podemos eliminar un procedimiento mediante la orden **DROP PROCEDURE nombre.**

Uso de parámetros

Los parámetros se utilizan para transferir valores de datos entre el entorno invocador y el procedimiento (o subprograma). Los parámetros también se conocen como argumentos. Se declaran en el encabezado del procedimiento o subprograma, después del nombre y antes de la sección de declaración para variables locales. Los parámetros están sujetos a uno de los tres modos de paso de parámetros: **IN, OUT,** e **IN OUT.**

- **Un parámetro IN** pasa un valor constante desde el entorno de llamada al procedimiento.
- **Un parámetro OUT** pasa un valor del procedimiento al entorno de llamada.
- **Un parámetro IN OUT** pasa un valor del entorno de llamada al procedimiento y un valor posiblemente diferente del procedimiento de vuelta al entorno de llamada utilizando el mismo parámetro.

Ejemplo práctico de creación y uso de procedimientos

Vamos a realizar un procedimiento almacenado tomando como referencia el código siguiente, el cual tiene dos parámetros de entrada: **p_id** y **p_percent**, que son respectivamente el código de empleado y el % de subida que deseamos aplicar al salario.

El procedimiento modifica el salario con un tanto por ciento que introduce el usuario sólo al empleado cuyo código de empleado es igual al introducido también por el usuario (el código es único por empleado).

Después se ejecuta el procedimiento (**EXECUTE nom_procedimiento([parámetros])**). Incrementar un 10% el empelado con **employee_id=176.**

```
CREATE OR REPLACE PROCEDURE raise_salary
  (p_id      IN employees.employee_id%TYPE,
   p_percent IN NUMBER)
IS
BEGIN
    UPDATE employees
    SET salary = salary * (1+p_percent/100)
    WHERE employee_id = p_id;
END raise_salary;
/
```

```
EXECUTE raise_salary(176,10)
```

Vamos a crear el procedimiento mediante la herramienta **Explorador de Objetos**, la cual nos suministra el entorno de Oracle Application Express, para luego comprobar su funcionamiento.

Ya solamente nos queda ver de forma práctica cómo implementar con Oracle Application Express el procedure descrito.

Cómo implementar con Oracle Application Express el procedure descrito

---

Diseño y desarrollo de aplicaciones cliente-servidor

CASO.

Víctor *pregunta si las funciones PL/SQL no son lo mismo que los procedimientos almacenados, y* ***Carmen*** *le comenta que la diferencia es que las funciones pueden devolver un valor tras los cálculos. Comparando con el procedimiento almacenado anterior, que calculaba un incremento para el sueldo de un trabajador, la función podría hacer lo mismo, y además devolver en una variable el nuevo valor asignado a sueldo, para que pueda ser usado en otras sentencias o cálculos.*

Ya puestos, ***Carmen*** *le muestra también cómo se realiza la gestión de excepciones, cosa que no le resulta nada extaña a* ***Víctor****, que está acostumbrado a usarlas con Java, de una forma bastante similar. En este caso, se ha definido una excepción que se capturará y lanzará si se intenta asignar al incremento un valor negativo o nulo, o bien demasiado alto (superior al 5%):Cuando se lanza la excepción, en este caso, el efecto es que se muestra un mensaje de error adecuado: "Incremento de sueldo erróneo: No puede ser menor o igual que 0 ni mayor o igual a 5"*

Una función **es un bloque PL/SQL nombrado que puede aceptar parámetros, ser llamado y devolver un valor**. Las funciones son iguales que los procedimientos, pero además devuelven un valor, por lo que la llamada a una función debe realizarse dentro de una expresión. En general, se

utiliza una función para computar un valor.

- Una nueva función se crea con la declaración **CREATE FUNCTION**.
- Las funciones y los procedimientos se estructuran de la misma forma. Una función debe devolver un valor al entorno de llamada, mientras que un procedimiento devuelve cero o más valores a su entorno de llamada mediante parámetros **OUT**.
- Al igual que un procedure, una función tiene un encabezado, una sección declarativa, una sección ejecutable y una sección opcional de gestión de excepciones.
- Una función debe tener una cláusula **RETURN** en el encabezado y al menos una declaración **RETURN** en la sección ejecutable. La orden **RETURN** dentro de una función recibe el valor que la función debe devolver, el cual se convierte al tipo especificado en la cabecera de la función.

Sintaxis de una función PL/SQL:

```
CREATE [OR REPLACE] FUNCTION nombre_función
    [(parameter1 [mode1] datatype1,
      parameter2 [mode] datatype2, ...)]
RETURN datatype
IS|AS
[DECLARE]
    [local_variable_declarations; …]
BEGIN
    <código del procedimiento>
RETURN <valor>
[EXCEPTION]
END;
```


Ejemplo de una función PL/SQL

Vamos a generar una función para compararla con el procedimiento almacenado visto en el apartado 8.3.

```
CREATE OR REPLACE FUNCTION get_sal
(p_id employees.employee_id%TYPE)
RETURN NUMBER
IS
   v_sal employees.salary%TYPE := 0;
BEGIN
    SELECT salary
      INTO v_sal
      FROM employees
      WHERE employee_id = p_id;
    RETURN v_sal;
END get_sal;
/
```

```
EXECUTE dbms_output.put_line(get_sal(100))
```

- En este ejemplo la función **get_sal** se crea con un único parámetro input y devuelve el salario como número.
- La función **get_sal** devuelve un valor de retorno a una variable y utiliza una sola declaración **RETURN** en la sección ejecutable del código para devolver el valor guardado en la variable local.
- Si una función contiene una sección de excepción, también pude contener una declaración **RETURN**.
- El segundo cuadro de código utiliza el comando **EXECUTE** para llamar el procedimiento

BMS_OUTPUT.PUT_LINE, del cual su argumento es el valor de devolución de la función **get_sal**. En este caso, **get_sal** se invoca en primer lugar para calcular el salario del empleado con **ID** 100. El valor de salario devuelto se suministra como valor del **DBMS_OUTPUT.PUT_LINE** parámetro que muestra el resultado.

Para implementar una función en Oracle Application Express es igual que el proceso que hemos aprendido para los procedimientos almacenados ("8.3. EJEMPLO PRÁCTICO DE CREACIÓN Y USO DE PROCEDIMIENTOS"). La única diferencia es que debemos seleccionar FUNCIÓN en la opción CREAR del EXPLORADOR DE OBJETOS. Debes recordar también que la llamada a una función debe realizarse dentro de una expresión

Para saber más

Este interesante enlace es un breve manual (12 páginas) en el que profundizarás en la programación de funciones.

Funciones en PL/SQL

http://foobar.cl/~chepito/software/PL_pgSQL.pdf [versión en cache]

Excepciones en PL/SQL

- En PL/SQL una advertencia o condición de error es llamada una **excepción**.
- Hasta ahora hemos escrito los bloques PL/SQL con una sección declarativa (empezando con la palabra clave DECLARE) y una sección ejecutable (empezando con la palabra clave BEGIN y terminando con END). Para la gestión de excepción se puede en PL/SQL incluir una sección opcional llamada la sección de excepción. Esta sección empieza con la palabra clave **EXCEPTION**, y si la declaramos, debe ser la última sección del bloque.
- Una excepción es un error que surge durante la ejecución de un bloque. Un bloque siempre termina cuando se produce un error en la ejecución, pero gracias al uso de excepciones podemos especificar un gestor de excepción para ejecutar acciones antes que finalice el bloque.
- Cuando surge la excepción el control (la ejecución del programa) salta a la sección de excepción y son ejecutadas todas las declaraciones de la sección de excepción. Así conseguimos que se ejecute el bloque de PL/SQL sin que se aborte la ejecución del mismo.
- El manejo de excepciones nos posibilita también el poder mostrar mensajes de error personalizados al usuario.

La estructura de bloque de una excepción se muestra a continuación.

```
DECLARE
 --Declaraciones

BEGIN
 --Ejecución

EXCEPTION
 --Excepción

END;
```

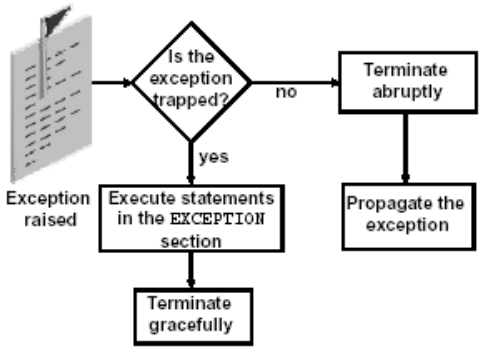
Cuando ocurre un error, se ejecuta la porción del programa marcada por el bloque **EXCEPTION**, transfiriéndose el control a ese bloque de sentencias.

El siguiente ejemplo muestra un bloque de excepciones que captura las excepciones **NO_DATA_FOUND** y **ZERO_DIVIDE.** Cualquier otra excepcion será capturada en el bloque **WHEN... OTHERS...THEN**.

```
DECLARE
  -- Declaraciones
BEGIN
  -- Ejecucion
EXCEPTION
WHEN NO_DATA_FOUND THEN
  -- Se ejecuta cuando ocurre una excepción de tipo
NO_DATA_FOUND
WHEN ZERO_DIVIDE THEN
  -- Se ejecuta cuando ocurre una excepción de tipo
ZERO_DIVIDE
WHEN OTHERS THEN
  -- Se ejecuta cuando ocurre una excepción de un tipo
     no tratado en los bloques anteriores

END;
```

- Como ya hemos visto, cuando ocurre un error, se ejecuta el bloque **EXCEPTION**, transfiriéndose el control a las sentencias del bloque. Una vez finalizada la ejecución del bloque de **EXCEPTION** no se continúa ejecutando el bloque anterior.
- Si existe un bloque de excepción apropiado para el tipo de excepción se ejecuta dicho bloque. Si no existe un bloque de control de excepciones adecuado al tipo de excepción se ejecutará el bloque de excepción **WHEN OTHERS THEN** (si existe!). **WHEN OTHERS** debe ser el último manejador de excepciones.



- Las excepciones pueden ser definidas en forma interna o explícitamente por el usuario. Ejemplos de excepciones definidas en forma interna son la división por cero y la falta de memoria en tiempo de ejecución. Estas mismas condiciones excepcionales tienen sus propios tipos y pueden ser referenciadas por ellos: **ZERO_DIVIDE** y **STORAGE_ERROR**.
- Las excepciones definidas por el usuario deben ser alcanzadas explícitamente utilizando la sentencia **RAISE**.
- Con las excepciones se pueden manejar los errores cómodamente sin necesidad de mantener múltiples chequeos por cada sentencia escrita. También provee claridad en el código ya que

permite mantener las rutinas correspondientes al tratamiento de los errores de forma separada de la lógica del negocio.

Excepciones predefinidas

PL/SQL proporciona un gran número de excepciones predefinidas que permiten controlar las condiciones de error más habituales. Las excepciones predefinidas no necesitan ser declaradas. Simplemente se utilizan cuando éstas son lanzadas por algún error determinado.

La siguiente es la lista de las excepciones predeterminadas por PL/SQL y una breve descripción de cuándo son lanzadas:

Excepción	Se ejecuta ...	SQL CODE
ACCESS_INTO_NULL	El programa intentó asignar valores a los atributos de un objeto no inicializado	-6530
COLLECTION_IS_NULL	El programa intentó asignar valores a una tabla anidada aún no inicializada	-6531
CURSOR_ALREADY_OPEN	El programa intentó abrir un cursor que ya se encontraba abierto. Recuerde que un cursor de ciclo FOR automáticamente lo abre y ello no se debe especificar con la sentencia OPEN	-6511
DUP_VAL_ON_INDEX	El programa intentó almacenar valores duplicados en una columna que se mantiene con restricción de integridad de un índice único (unique index)	-1
INVALID_CURSOR	El programa intentó efectuar una operación no válida sobre un cursor	-1001
INVALID_NUMBER	En una sentencia SQL, la conversión de una cadena de caracteres hacia un número falla cuando esa cadena no representa un número válido	-1722
LOGIN_DENIED	El programa intentó conectarse a Oracle con un nombre de usuario o password inválido	-1017
NO_DATA_FOUND	Una sentencia SELECT INTO no devolvió valores o el programa referenció un elemento no inicializado en una tabla indexada	100
NOT_LOGGED_ON	El programa efectuó una llamada a Oracle sin estar conectado	-1012
PROGRAM_ERROR	PL/SQL tiene un problema interno	-6501
ROWTYPE_MISMATCH	Los elementos de una asignación (el valor a asignar y la variable que lo contendrá) tienen tipos incompatibles. También se presenta este error cuando un parámetro pasado a un subprograma no es del tipo esperado	-6504
SELF_IS_NULL	El parámetro SELF (el primero que es pasado a un método MEMBER) es nulo	-30625
STORAGE_ERROR	La memoria se terminó o está corrupta	-6500
SUBSCRIPT_BEYOND_COUNT	El programa está tratando de referenciar un elemento de un array indexado que se encuentra en una posición más grande que el número real de elementos de la colección	-6533
	El programa está referenciando un elemento de	

SUBSCRIPT_OUTSIDE_LIMIT	un array utilizando un número fuera del rango permitido (por ejemplo, el elemento "-1")	-6532
SYS_INVALID_ROWID	La conversión de una cadena de caracteres hacia un tipo rowid falló porque la cadena no representa un número	-1410
TIMEOUT_ON_RESOURCE	Se excedió el tiempo máximo de espera por un recurso en Oracle	-51
TOO_MANY_ROWS	Una sentencia SELECT INTO devuelve más de una fila	-1422
VALUE_ERROR	Ocurrió un error aritmético, de conversión o truncamiento. Por ejemplo, sucede cuando se intenta calzar un valor muy grande dentro de una variable más pequeña	-6502
ZERO_DIVIDE	El programa intentó efectuar una división por cero	-1476

Excepciones definidas por el usuario

PL/SQL permite al usuario definir sus propias excepciones, las que deberán ser declaradas y lanzadas explícitamente utilizando la sentencia **RAISE**.

Las excepciones deben ser declaradas en el segmento **DECLARE** de un bloque, subprograma o paquete. Se declara una excepción como cualquier otra variable, asignándole el tipo **EXCEPTION**. Las mismas reglas de alcance se aplican tanto sobre variables como sobre las excepciones.

```
DECLARE
 -- Declaraciones
    MyExcepcion EXCEPTION;
BEGIN
  -- Ejecución
EXCEPTION
  -- Excepción
END;
```

Esas **reglas de alcancede la excepción** son las siguientes:

- Una excepción es válida dentro de su ámbito de alcance, es decir, el bloque o programa donde ha sido declarada. Las excepciones predefinidas son siempre válidas.
- Como las variables, una excepción declarada en un bloque es local a ese bloque y global a todos los sub-bloques que comprende.

La sentencia RAISE

La sentencia **RAISE** permite lanzar una excepción en forma explícita. Es posible utilizar esta sentencia en cualquier lugar que se encuentre dentro del alcance de la excepción.

```
DECLARE
 -- Declaramos una excepcion identificada por
VALOR_NEGATIVO
    VALOR_NEGATIVO EXCEPTION;
    valor NUMBER;
BEGIN
  -- Ejecución
valor := -1;
  IF valor < 0 THEN
        RAISE VALOR_NEGATIVO;
  END IF;

EXCEPTION
  -- Excepción
WHEN VALOR_NEGATIVO THEN
  dbms_output.put_line('El valor no puede ser
negativo');
END;
```

Con la sentencia **RAISE** podemos lanzar una excepción definida por el usuario o predefinida, siendo el comportamiento habitual lanzar excepciones definidas por el usuario.

Recordar la existencia de la excepción **OTHERS**, que simboliza cualquier condición de excepción que no ha sido declarada. Se utiliza comúnmente para controlar cualquier tipo de error que no ha sido previsto. En ese caso, es común observar alguna de las funciones **SQLCODE** - **SQLERRM**, que se detallan en el próximo punto.

Funciones SQLCODE y SQLERRM

- Al manejar una excepción es posible usar las funciones predefinidas **SQLCODE** y **SQLERRM** para aclarar al usuario la situación de error acontecida.
- **SQLCODE** devuelve el número del error de Oracle y un 0 (cero) en caso de éxito al ejecutarse una sentencia SQL.
- Por otra parte, **SQLERRM** devuelve el correspondiente mensaje de error.
- Estas funciones son muy útiles cuando se utilizan en el bloque de excepciones, para aclarar el significado de la excepción **OTHERS**.
- Estas funciones no pueden ser utilizadas directamente en una sentencia SQL, pero sí se puede asignar su valor a alguna variable de programa y luego usar esta última en alguna sentencia.

```
DECLARE
  err_num NUMBER;
  err_msg VARCHAR2(255);
  result  NUMBER;
BEGIN
  SELECT 1/0 INTO result
  FROM DUAL;
EXCEPTION
WHEN OTHERS THEN
  err_num := SQLCODE;
  err_msg := SQLERRM;

DBMS_OUTPUT.put_line('Error:'||TO_CHAR(err_num))
;
  DBMS_OUTPUT.put_line(err_msg);
END;
```

- También es posible entregarle a la función **SQLERRM** un número negativo que represente un error de Oracle y ésta devolverá el mensaje asociado.

```
DECLARE
  msg VARCHAR2(255);
BEGIN
  msg := SQLERRM(-1403);
  DBMS_OUTPUT.put_line(MSG);
END;
```


Propagación de excepciones en PL/SQL

Una de las características más interesantes de las excepciones es la propagación de excepciones.

- Cuando se lanza una excepción, el control se transfiere hasta la sección **EXCEPTION** del bloque donde se ha producido la excepción. Entonces se busca un manejador válido de la excepción (**WHEN***excepcion***THEN**, **WHEN OTHERS THEN**) dentro del bloque actual.
- En el caso de que no se encuentre ningún manejador válida el control del programa se desplaza hasta el bloque **EXCEPTION** del bloque que ha realizado la llamada PL/SQL.

Observa el siguiente bloque de PL/SQL (Nótese que se ha añadido una cláusula **WHERE 1=2** para provocar una excepción **NO_DATA_FOUND**).

```
DECLARE
fecha DATE;
       FUNCTION fn_fecha RETURN DATE
       IS
         fecha DATE;
       BEGIN
            SELECT SYSDATE INTO fecha
            FROM DUAL
            WHERE 1=2;
            RETURN fecha;
       EXCEPTION
       WHEN ZERO_DIVIDE THEN
       dbms_output.put_line('EXCEPCION ZERO_DIVIDE
CAPTURADA
                              EN
fn_fecha');
       END;
BEGIN
   fecha := fn_fecha();
   dbms_output.put_line('La fecha es '||TO_CHAR(fecha,
'DD/MM/YYYY'));
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    dbms_output.put_line('EXCEPCIÓN NO_DATA_FOUND
                  CAPTURADA EN EL BLOQUE PRINCIPAL');
END;
```

La excepción **NO_DATA_FOUND** se produce durante la ejecución de la función **fn_fecha**, pero como no existe ningún manejador de la excepción en dicha función, la excepción se propaga hasta el bloque que ha realizado la llamada. En ese momento se captura la excepción.

Vamos a aprender cómo implementar con Oracle Application Express una excepción.

Para ello vamos a modificar el procedimiento que hemos creado en el apartado "EJEMPLO PRÁCTICO DE CREACIÓN Y USO DE PROCEDIMIENTOS".

Vamos a controlar mediante excepciones que el incremento de sueldo no pueda ser menor que 0% ni

superior a un 2.5%.

En otros casos se visualizará un mensaje de error de ejecución del procedimiento.

Propagación de excepciones en PL/SQL