1. Introducción

El objetivo de este seminario es por una parte presentar la tecnología de ***Bases de Datos Activas*** y por otra parte verificar el comportamiento de la integridad de los datos, mediante un caso práctico, implantado de acuerdo a las definiciones de Díaz y Paton (1997).

Una ***Base de Datos Activa*** (en adelante ***BDA*** si hablamos del concepto o ***SGBDA*** si hablamos del gestor) consiste en una extensión de un ***SGBD***, compuesta de trozos de programas coordinados de tal manera que permitan almacenar la semántica de los datos además de los propios datos, para convertirla en un ***SGBDA***.

En este sentido, no basta con aplicar las facilidades que proporcionan los ***SGBD*** modernos tales como ORACLE, SYBASE, INFORMIX etc., si no, que hace falta una infraestructura que sea capaz de soportar una adecuada interfaz con el usuario, para su configuración y control. Esta infraestructura puede ser proporcionada por el proveedor del gestor o bien desarrollarse en base a él, lo que requiere un profundo conocimiento tanto del paradigma de ***BDA*** como del gestor a utilizar, además de adoptar un "aproach" o integrar varios de ellos para su implementación, por parte de los desarrolladores.

Se puede pensar que las ***BDA*** transforma la Aplicación de Bases de Datos moviendo la conducta reactiva de la aplicación hacia el sistema de gestión de bases de datos. Con esto se obtiene de inmediato tres mejoras relacionadas con el rendimiento y los costos de mantención, globales de la aplicación:

- Una aplicación más liviana en el cliente
- El usuario administrador de la aplicación tiene acceso al control de la Base de Datos.
- El Modelo de Conocimiento y de Ejecución Queda Disponible para el usuario y no encapsulado para el programador.

Tema abierto: Hacer una interfaz inteligente de ***BDA***. La interfaz a construir en este trabajo, si bien cumple con las definiciones de ***BDA*** y hace disponible el modelo de conocimiento y de ejecución, aún es compleja ya que aún hay mucho código que manejar.

Precisaremos algunos conceptos vertidos más arriba:
La medida a utilizar será la cantidad de registros inconsistentes con las reglas por unidad de tiempo, el tiempo considerado tiene que ver con el "delay", que se produce en las situaciones de conflicto, decisiones de conflictos, determinaciones de prioridades y chequeos generales, principalmente en casos de grandes cantidades de reglas de conocimientos y parámetros de la ***BDA***.

La integridad de los datos a medir va ligada con las reglas del negocio, las cuáles reflejan las restricciones que existen en el negocio dado, de modo que nunca sea posible llevar a cabo acciones no válidas, dejando por sentado que la integridad primaria (PK), referencial (FK) y de dominio, no es el ámbito prioritario de un ***SGBDA***, aunque los incluye.

La definición técnica de una BDA activa incluye evento-condición-acción, llamados reglas ECA la que constituye el modelo de conocimiento de una BDA. La parte del evento de una regla describe un acontecimiento en que la regla puede ser capaz de responder. La parte de la condición de la regla examina el contexto en que el evento ha tenido lugar. La acción describe donde la tarea se lleva a cabo por la regla, si el evento pertinente ha tenido lugar y si la condición ha evaluado las reglas. Esta regla se ejecutan a través de un modelo de ejecución.

Las características de un SGBDA Según un Manifiesto ACTNET 1996 se dividen en tres apartados:

A) <u>Características de los SGBDA</u>, como un SGBDA es un SGBD, debe soportar facilidades para el modelado, lenguaje de consultas, accesos multiusuarios, recuperaciones, además proporciona un modelo de reglas (ECA), por lo que extiende el lenguaje de definición de datos, en el que debe proporcionar medios para la definición de tipos de eventos, condiciones y acciones; soporta la gestión de reglas, que son parte de la información de la base de datos y a la evolución de la base de reglas, las que no deben caballeares en el

SGBD, para odre modificar las definiciones del evento, condición y acción, soportando además la activación y desactivación de las reglas.

B) Características de Ejecución de Reglas ECA, debe ser capaz de detectar ocurrencias de eventos en forma automática, evaluar las condiciones y ejecutar acciones mediante distintos modelos de acoplamiento, además de gestionar la historia de los eventos para ser extendido a otras transacciones similares e implementar resolución de conflictos que se producen cuando varias transacciones pueden ejecutarse en un momento dado.

C) Características de aplicación y usabilidad del SGBDA, debe soportar un entorno de programación y ser ajustable, es decir que no sufra una degradación del rendimiento.

Los ***triggers*** son trozos de Programas que realizan la actividad de la BDA, a través de secuencias de operaciones de manipulación de la base de datos (INSERT, DELETE y UPDATE), estos también corresponden a las acciones de una regla ECA implementada en los triggers asociados a las tablas en las cuáles se realiza la manipulación de la Base de Datos.

El ***Modelo de Conocimiento*** es el que permite mantener algún mecanismo para que los usuarios describan la conducta reactiva de la base de datos, a través de las ***reglas ECA***.

El ***Modelo de ejecución*** es el que supervisa y reacciona frente a las circunstancias pertinentes mediante una conducta proactiva, señalando qué pasos se ejecutan desde que se recibe la señal del evento hasta la ejecución de la acción.

Para esto desarrollaremos un Caso Práctico sobre el tratamiento de las Bases De Datos Activas, utilizando la metodología Case Method para el desarrollo del análisis y construcción del sistema, en donde propondremos una arquitectura simple, utilizando la versión Oracle disponible en la universidad.

2. Definición de un SGBDA

El ***SGBDA*** es aquel que cuando se producen ciertas condiciones, ejecuta de forma automática, es decir, sin la intervención del usuario, las acciones especificadas de antemano en la fase de definición de la base de datos. Un SGBDA debe ser capaz, por tanto, de monitorizar y reaccionar ante eventos de manera oportuna y eficiente.

Una manera eficaz de supervisar las circunstancias es controlando que ninguna regla de integridad se viole. Una restricción de integridad puede especificar la legalidad de los datos y por tanto prevenir los errores, esta prevención consiste en rechazar las acciones que violen esta integridad, para ello existen técnicas en la prevención de inconsistencias que se puedan producir, o el descubrimiento que implica deshacer transacciones incorrectas.

Por lo tanto es importante que un Sistema de Base de Datos activa (SGBDA) pueda detectar y corregir, en lo posible las operaciones incorrectas.

Características de los SGBDA

- Un ***SGBDA*** es un ***SGBD.***
- Un ***SGBDA*** tiene un modelo de reglas ***ECA***.
- Un ***SGBDA*** debe soportar la gestión de reglas y la evolución de la base de reglas.

2.1 Justificaciones

No se puede hablar de que el sistema gestor de bases de datos Oracle, u otros similares poseen una arquitectura activa, lo que realmente poseen es un comportamiento activo mediante la utilización de Triggers, pero es necesaria la construcción de un nivel superior que pueda almacenar los triggers, las reglas y los datos, ya que existen limitaciones en los Sistemas de SQL, principalmente una falta de flexibilidad**,** hay posibilidades en que el compromiso de la activación de un Trigger depende del compromiso de la transacción del Trigger que lo generó, posibilidades como éstas no son soportadas usualmente por los modelos de ejecución de Triggers y requieren el desarrollo de un nivel de regla activo sobre la base de datos designada.

Cuando se comparan los cambios de una aplicación con y sin triggers, se observa que las versiones basadas en triggers corren significativamente más lento, las conclusiones que se pueden derivar de este comportamiento en las bases de datos activas es la inmadurez de las implementaciones mediante triggers, la carencia de experiencia en los desarrolladores en los programas de triggers, sin embargo los triggers son usados para toda clase de tareas, como código de integridad, seguridad, reglas de negocio, restricciones de tiempo, es decir con implementaciones de esta clase es que actualmente planea explotar las bases de datos activas, ya que el mejor camino para hacer los chequeos a tiempo es usar los triggers que pueden ser accionados una vez ocurrido un evento.

2. 2 Modelo de Conocimiento

El Modelo de Conocimiento es el que describe la situación y la conducta correspondiente. Es un sistema de la base de datos activa e indica lo que puede decirse sobre las reglas activas en este sistema. Esto está en el contraste con el modelo de ejecución que determina cómo se comportan las reglas en tiempo de ejecución.

Se considera que el modelo de conocimiento de una regla activa tiene tres componentes principales que son: el evento, la condición, y la acción ya que Cuando ocurre un determinado **evento** se evalúa la **condición** y se está se satisface se ejecuta la **acción**.

Avanzando ya sobre los mecanismos de expresión de las respuestas de estos sistemas activos, se consigna que un conjunto de reglas, habitualmente denominadas triggers, consisten de un evento, que provoca la evaluación de una condición, que si es satisfecha, dispara un procedimiento, llamado la acción de la regla.

Entonces se define a una regla como un evento que provoca la evaluación de una condición; si ésta se satisface, causa la ejecución de una acción predefinida. Las condiciones provocan consultas a la base de datos y finalmente las acciones ejecutan cambios sobre la base de datos. Las reglas son denominadas generalmente TRIGGERs o ECAs y son poderosas herramientas para expresar restricciones de integridad, reglas del negocio, eventos temporales entre otros.

Las reglas ECA son del tipo Evento-Condición-Acción y expresan cuales son las acciones que se deben disparar ante un determinado evento, habiéndose evaluado satisfactoriamente la condición.

2.3 Regla de Comportamiento Evento-Condición-Acción

- **Evento**

El evento especifica el suceso a cuya ocurrencia debe responder el sistema. Los eventos pueden ser clasificados en:

Eventos primitivos: aquellos que son predefinidos en el sistema. Para cada uno de ellos debe proveerse un mecanismo eficiente de detección, probablemente embebido en el sistema. A su vez, estos pueden clasificarse en:

Eventos de base de datos están relacionados con las operaciones, en el modelo relacional: selección, inserción, actualización y borrado.

Eventos temporales son los relacionados con el tiempo y son a su vez de dos tipos: absolutos y relativos. Los absolutos mapean a puntos discretos a lo largo de la línea de tiempo, mientras que los relativos son definidos con respecto a un punto de referencia explícito.

Eventos explícitos o externos son aquellos eventos que son detectados y generados junto con sus parámetros por programas de aplicación y son solo manejados por el sistema. Previo a su uso, los eventos explícitos y sus parámetros formales necesitan ser registrados con el sistema. Cada evento tiene un conjunto bien definido de parámetros que son instanciados para cada ocurrencia del evento.

Eventos complejos o compuestos: son los que se forman por la aplicación de un conjunto de operadores a otros eventos primitivos o compuestos.

Tipos de Eventos

- Operaciones de actualización de la base de datos: **Insert, Delect, Update**.
- Tiempo: un instante de tiempo, un periodo de tiempo (cada día a las 12:00)
- Definidos por las aplicaciones.

Composición de Eventos

- Composición lógica (**And, Nor, Or**)
- Secuencia de eventos
- Composición temporal

- **Condición**

Se trata de una expresión que debe ser satisfecha para que se pueda proceder al disparo de la acción.

En algunas bases de datos comerciales, no existe manera de especificar la activación de la acción. Ambas forman un mismo bloque, que es responsabilidad del programador de aplicaciones.

En algunos enfoques de reglas basadas en eventos, la condición directamente no existe, por lo que es siempre satisfecha.

En otros más sofisticados, en los que las reglas son disparadas por modificaciones en los datos, se permite referenciar tanto en la condición como en la acción a los valores previos y posteriores a la modificación. Estos mecanismos se denominan condiciones de transición.

La problemática que existe en este punto es la de seleccionar o crear un lenguaje que permita su especificación. En algunas implementaciones la condición se expresa como un query (consulta), considerándose verdadera cuando la cantidad de filas seleccionadas es mayor que cero.

Tipos de Condiciones

- Expresiones lógicas del lenguaje SQL: cláusula Where.
- Consultas a la base de datos (Si el resultado de la consulta es vacía, la condición no se cumple y viceversa)
- Procedimiento de tipo lógico en un lenguaje de programación.

Instanciación de la Condición

En la condición se pueden hacer referencia a los parámetros del evento instanciándose de esta forma la condición cundo se **activa** la regla.

Parametrización de la Condición

- La condición puede parametrizarse con referencias a datos relacionados con la condición (datos consultados)
- Los parámetros de la condición se inicializan con valores obtenidos durante la evaluación de la condición cuando la regla es seleccionada por el sistema de base de datos (Ej:

cuando la condición es una consulta los parámetros se pueden inicializar con los datos resultantes de la consulta).

- **Acción**

Especifica las acciones que deben ser ejecutadas por el sistema como respuesta a la ocurrencia del evento cuando la condición es cierta.

Existe una acción implícita en la evaluación de las condiciones, que es la de aceptar o rechazar una operación. Pero se debe describir explícitamente el mecanismo que hace a la propagación de los efectos de la actualización en las relaciones necesarias, en función del modo definido para la operación (restricto, cascada y nulificación o anulado).

Tipos de Acciones

- Operaciones de actualización de la base de datos: **Insert, Delete, Update.**
- Operaciones de consulta de la base de datos: **Select.**
- Operaciones de control de transacciones: Rollback, Commit.
- Llamadas a procedimientos en un lenguaje de programación.

Instanciación de la Acción

En la acción se puede hacer referencia a los parámetros del evento (respuesta de la condición) instanciandose de esta forma la acción cuando la regla se activa (respuesta es evaluada por el sistema de base de datos).

Composición de Acciones

- Secuencia de acciones.

- **Lenguajes de Reglas**

En algunos sistemas de Base de Datos los lenguajes de reglas están limitados a monitorear solo un evento. En general una regla puede monitorear un conjunto de eventos y ésta es disparada si cualquiera de ellos ocurrió.

A veces la condición puede testear que evento fue el que disparó la regla como por ejemplo los predicados condicionales en Oracle.

- **Acoplamiento entre Evento Condición y Acción**

El modo de acoplamiento o ensamble hace referencia al momento en que es evaluada la condición y ejecutada la acción con respecto a la transacción. Las alternativas son:

- **Inmediato**: ejecución inmediata de la condición después del evento, o ejecución de la acción inmediatamente después del evento.
- **Diferido**: ejecución al final de la transacción, tanto cuando se trata de condición o de la acción.
- **Desacoplado**: ejecución en una transacción separada de la condición, acción o de ambas.

- **Selección de Reglas**

En muchos sistemas varias reglas pueden ser disparadas a la vez, decimos entonces que estas reglas forman el conjunto conflicto, y el algoritmo que procesa las reglas debe tener una política para seleccionar las reglas de este conjunto.

En las bases de datos activas, la selección de la siguiente regla a ser procesada típicamente ocurre antes de la consideración de la regla y posible ejecución. O sea puedo seleccionar todas las reglas que debo procesar, armar una lista y luego ir tomando de a una hasta que se vacía.

Normalmente los sistemas tienen una manera de definir en que orden se procesan las reglas. En estos casos las ejecuciones son repetibles, es decir dos ejecuciones de la misma transacción sobre el mismo estado de la base de datos con el mismo conjunto de triggers da la misma secuencia de ejecución.

- **Características de Ejecución de reglas ECA**
  - Un ***SGBDA*** tiene un modelo de ejecución

    Un ***SGBDA*** debe ofrecer diferentes modelos de acoplamiento

2.4 Modelo De Ejecución

El modelo de ejecución realiza un seguimiento de la situación y gestiona el comportamiento activo. Especifica cómo un conjunto de reglas es tratado en tiempo de ejecución, se encarga de realizar el seguimiento de la situación, gestionando el comportamiento activo, comportamiento que es ilustrado en la Figura 2.1:

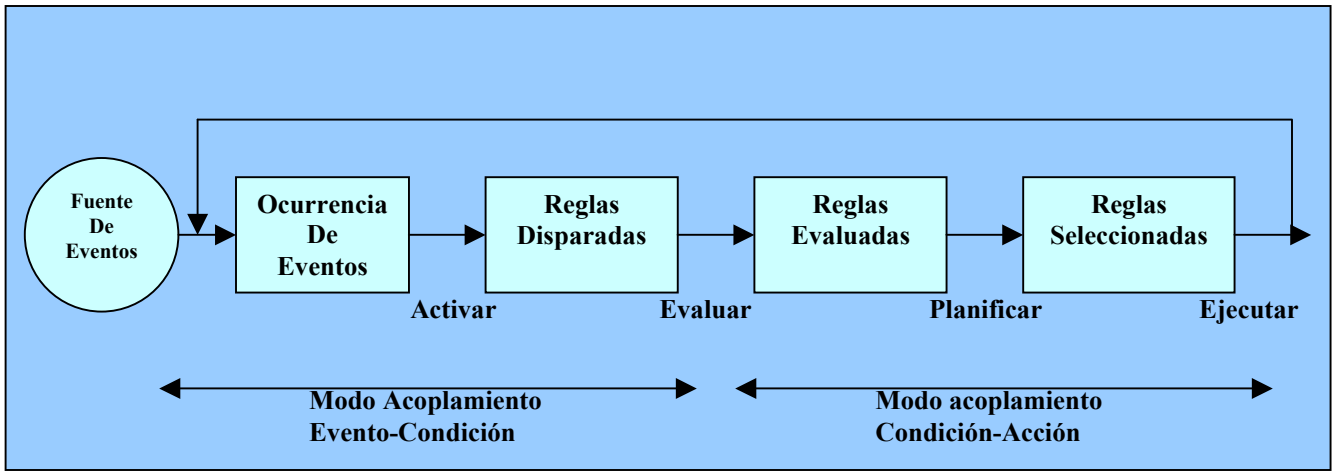


Figura 2.1: Modelo De Ejecución de una BDA

- **Señalización:** En la señalización ocurre la aparición de las ocurrencias en el evento.
- **Activación:** en la activación se toman los eventos producidos y se dispara las reglas correspondientes.
- **Evaluación:** Se selecciona una regla activada y se evalúa su condición.
- **Planificación:** en la planificación se indica cómo se procesa el conjunto de reglas.
- **Ejecución:** la ejecución es la que lleva a cabo las acciones de las reglas escogidas.

Estrategia de ejecución

Para animar el sistema especificado, se define una estrategia de ejecución e interacción. Esta estrategia es cercana a las técnicas de realidad virtual, en el sentido de que un objeto activo se introduce en la sociedad de objetos como miembro de ella e interactúa con los demás enviando y recibiendo mensajes. Para iniciar una sesión de ejecución, los pasos a seguir son:

Identificación del usuario (control de acceso)

Consiste en la conexión del usuario al sistema. Una vez conectado se le proporciona una visión clara de la sociedad de objetos (ofreciéndole qué clases de objetos puede ver, los servicios que puede activar y los atributos que puede consultar).

Activación de servicios

El usuario podrá activar cualquier servicio (evento o transacción) disponible en su visión de la sociedad. Además, podrá realizar observaciones del sistema (object queries).

Las clases que implementan las tareas de control de acceso y construcción de la vista del sistema (clases y servicios visibles) se implementarán en el nivel de interfaz. La información necesaria para configurar la vista del sistema está incluida en la especificación del sistema (relaciones de agente) obtenida en la fase de modelado conceptual.

Cualquier activación de un servicio tiene dos partes: la construcción del mensaje y la ejecución (sí es posible).

Identificación del objeto servidor: Si el objeto existe el nivel de persistencia se encargará de recuperar el objeto servidor de la base de datos y si es un evento de creación reservará espacio para su almacenamiento.

Introducir los argumentos necesarios para la ejecución del evento: el nivel de interfaz preguntará por los argumentos del evento que va a activarse (sí es necesario).

Una vez el mensaje se ha enviado, se identifica el objeto servidor (la existencia del objeto servidor es una condición implícita para ejecutar cualquier evento, excepto si se trata del evento creación) y se procede a seguir una secuencia de acciones sobre dicho objeto:

Transición válida de estado: se verifica en el diagrama de transición de estados que exista una transición válida (desde el estado actual a un nuevo estado) para el servicio seleccionado.

Satisfacción de precondición: se comprueba que se cumpla la precondición asociada al servicio en ejecución (sí existe).

Si no se cumplen las dos anteriores se elevará una excepción informando que el servicio no puede ejecutarse.

Evaluaciones: se modifican los valores de los atributos afectados por la ocurrencia del servicio (como fuera especificado en el modelo funcional).

Comprobación de las restricciones de integridad: las evaluaciones del servicio deben dejar al objeto en un estado válido. Se comprueba que no se violan las restricciones de integridad (estáticas y dinámicas). Si alguna de ellas se viola, se generará una excepción y el cambio de estado producido se ignorará.

Comprobación de las relaciones de disparo: después de un cambio de estado válido, y como acción final, se debe verificar el conjunto de reglas condición-acción que representa la actividad interna del sistema. Si alguna de ellas se cumple, se activará el servicio correspondiente.

Una vez finalizadas con éxito las acciones precedentes, los componentes de la capa de persistencia se encargan de actualizar (UPDATE) la BD correspondiente.
Los pasos anteriores guiarán la implementación de cualquier aplicación para asegurar la equivalencia funcional entre la descripción del sistema, recogida en el modelo conceptual, y su reificación en un entorno de programación de acuerdo con el modelo de ejecución.
Las fases del modelo de ejecución no se ejecutan necesariamente contiguamente, sino que dependen de los modos de acoplamiento del evento, de la condición y de la acción.

2.4.1 Tipos de Acoplamiento

Acoplamiento Evento-Condición
Determina cuando se evalúa la condición (evaluación) con respecto a la ocurrencia del evento (Activación).

- **Inmediato:** La condición se evalúa inmediatamente después de producirse el evento que activa la regla.
- **Diferido:** la condición se evalúa en algún instante posterior a la ocurrencia del evento que activa la regla.

Acoplamiento Condición-Acción

Determina cuando se ejecuta la acción (Ejecución) con respecto a la evaluación de la condición (evaluación).

- **Inmediato:** la acción se ejecuta inmediatamente después de evaluarse la condición de la regla.
- **Diferido:** La acción se ejecuta en algún instante posterior a la evaluación de la condición de la regla.

Las opciones para los modos de acoplamiento utilizados con más frecuencia son:

Modo de acoplamiento Inmediato: La figura 2.2 nos muestra en forma sencilla este caso, donde la condición (acción) es evaluada (ejecutada) inmediatamente después del evento(condición). Los modos inmediatos del acoplador se pueden utilizar, por ejemplo, para hacer cumplir apremios de la seguridad o propagos de actualizaciones.

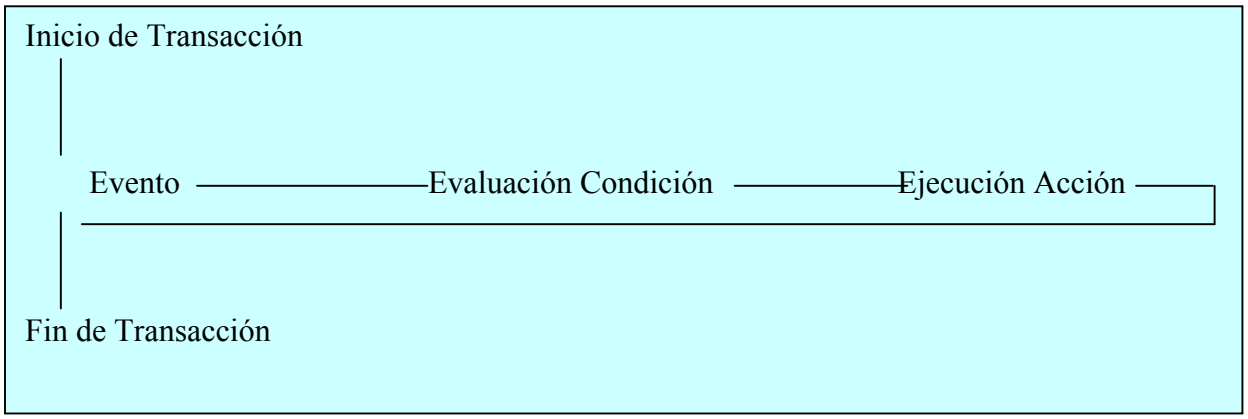


Figura 2.2: Acoplamiento Inmediato

En el siguiente algoritmo se asume un acoplamiento Condición-acción de tipo inmediato:

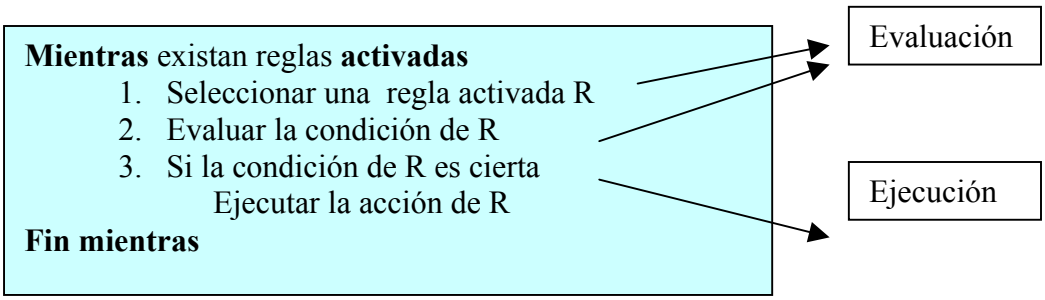


Figura 2.3: Algoritmo condición-acción de tipo Inmediato

Modo de acoplamiento Diferido: en este caso la condición (acción) se evalúa (ejecuta) dentro de la misma transacción que el evento (condición) de la regla, pero no necesariamente en la primera oportunidad. Normalmente, la transformación posterior se deja hasta el final de la transacción, o hasta una demanda del programa de la regla que procesa en lugar de la toma de un punto de aserción de la regla. La Figura 2.4 muestra el orden de accionar que caracteriza a este modo de acoplamiento.

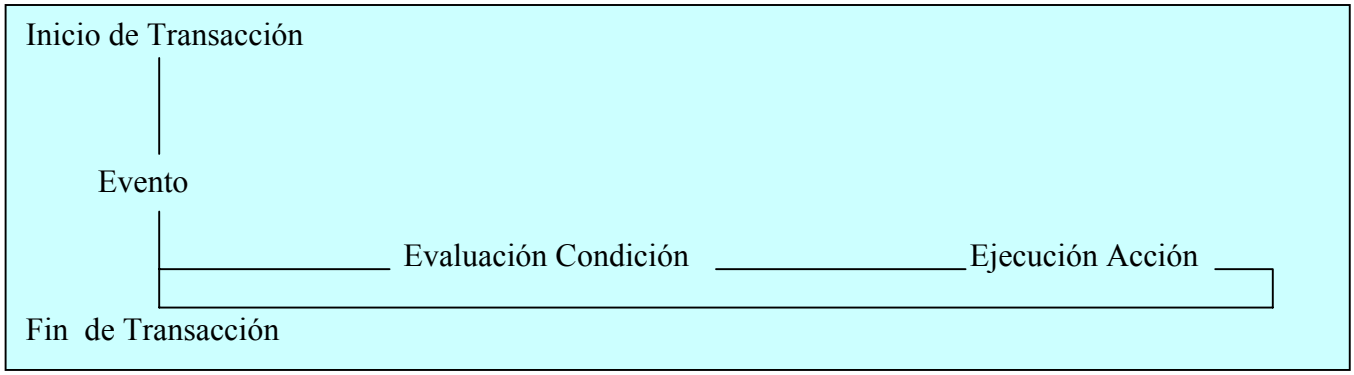


Figura 2.4: Acoplamiento Diferido

Modo de acoplamiento Separado: en este caso la condición (acción) se evalúa (ejecuta) dentro de una transacción diversa del evento (condición). La ejecución de la acción puede ser dependiente o independiente de confiar en la transacción en la cual el acontecimiento ocurrió o donde la condición fue evaluada. Los modos separados del acoplador se asocian normalmente al control de las actividades externas de las base de datos. Este modo e acoplamiento se divide en independiente y dependiente.

Independiente: este se produce cuando la condición (Acción), se evalúa (Ejecuta), en una transacción diferente en la que se evalúa el evento (Condición). La figura 2.5 detalla como se produce la evaluación del Evento y su posterior ejecución.

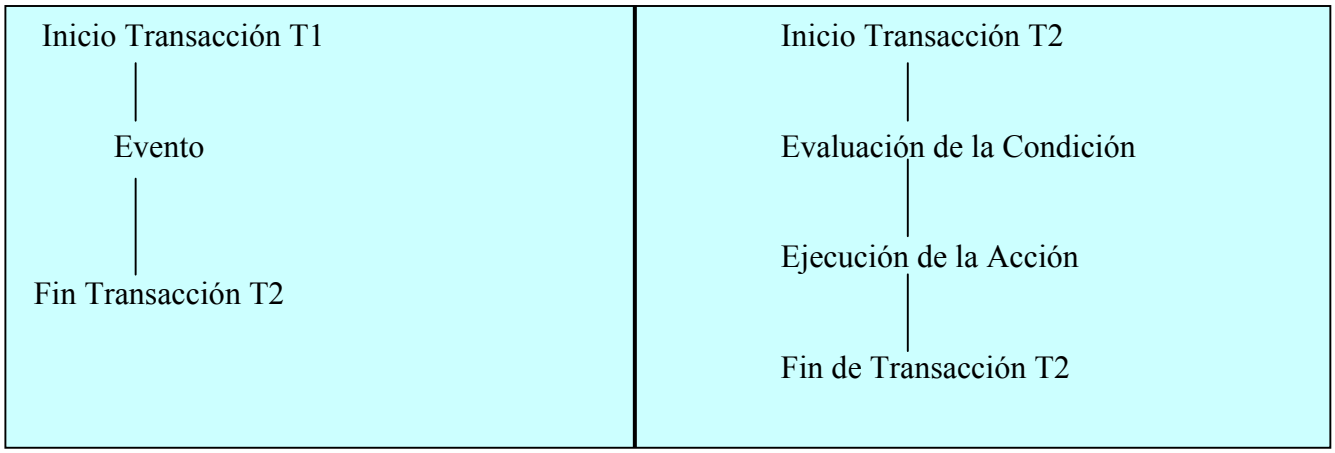


Figura 2.5: Acoplamiento Separado Independiente

Dependiente: se produce cuando la condición(Acción), se evalúa(Ejecuta), en una transacción diferente en la que se evalúa el evento (Condición); pero en este caso la ejecución es dependiente de la grabación(Commit) de la transacción en la que el evento tiene lugar o en la que se evalúa la condición, comportamiento que detalla la Figura 2.6:

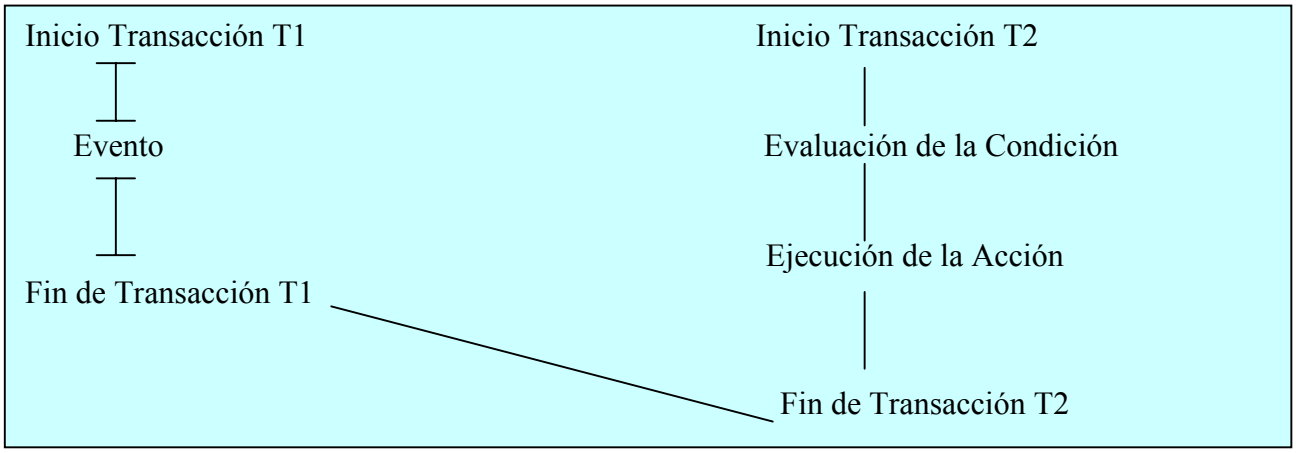


Figura 2.6: Acoplamiento Separado Dependiente

2.5 Arquitectura Lógica de una Base de Datos Activa

La Arquitectura abstracta comprende los objetos de datos y los procedimientos que contienen todos los pasos necesarios para la ejecución de una regla, ver figura 2.7. Los óvalos representan la información principal y los rectángulos a los procesos utilizados. Ahora definiremos cada uno de los componentes de la arquitectura.

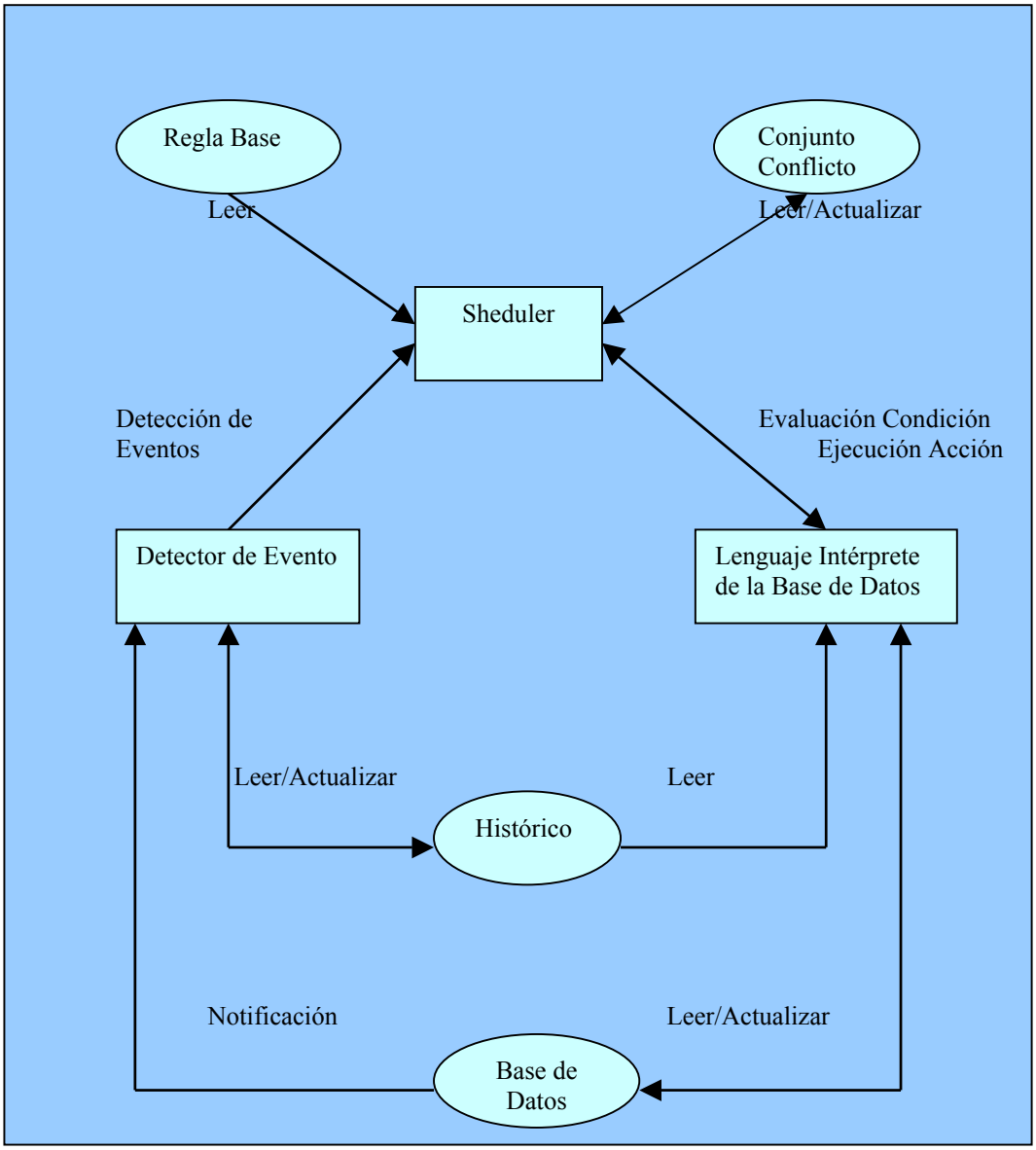


Figura 2.7: Arquitectura Lógica de una BDA

- **Detector de Eventos:** el DBMS notifica al detector de eventos las ocurrencias dentro de la base de datos en las cuáles el detector de evento ha registrado algún interés. El detector de evento puede leer entonces o puede poner al día la Historia sobre una cantidad de información que es de relevancia para el descubridor de evento para saber sobre lo que ha pasado en la base de datos.

- **Scheduler** pide la información sobre los eventos descubiertos por el descubridor de evento, y lee de la Regla las descripciones Bajas de reglas que van asociadas con los eventos que han tenido lugar. También pide información sobre reglas que se han activado, pero que el scheduler puede desear procesar después, esta se guarda en el conjunto de Conflicto.

- **Intérprete de la base de datos** es llamado por el scheduler siempre que una condición de la regla se evalúe o la acción se haya ejecutado. Este proceso puede requerir información leída o escrita desde la base de datos, y puede ser posible para el idioma intérprete reparar ciertas estructuras conforme a lo que ha pasado, mientras se ha usando la información histórica. Estos accesos que ponen al día la base de datos puede llevar a su vez el descubrimiento de eventos extensos, mientras la ejecución de la regla ha causado un proceso para ser repetido.

2.6 Arquitectura Física de una Base de Datos Activa

La arquitectura Física de una base de datos activa esta compuesta por un conjunto de tablas que mantienen la configuración de la Base de Datos Activa y las funciones, procedimientos y triggers almacenados que pertenecen a un usuario administrador cuyo esquema es compartido con los usuarios finales del sistema para su utilización.

Puesto que un sistema de base de datos activa por definición debe proporcionar además de capacidades activas, características completas del sistema gestor de base de datos, puede ser visto como extensión de una base de datos pasivo. Hay varias características del sistema gestor de base de datos ya existente y de la estrategia arquitectónica usada para poner en ejecución las extensiones activas que tendrán un impacto en las funciones y en el

funcionamiento de la base de datos activa. Las dimensiones más importantes que necesitan ser consideradas son:

- El grado de integración entre el sistema de base de datos ya existente y las capacidades activas,
- La configuración del sistema del sistema de base de datos ya existente, y
- El modelo de datos del sistema de base de datos y del lenguaje de programación usado para las extensiones activas.

Para explicar con mas detalle, en la Figura 2.8, se observa que en la Arquitectura Física de la BDA, los Datos y las reglas las almacenan en el mismo lugar físico, dando lugar a todas las ventajas mencionadas anteriormente.

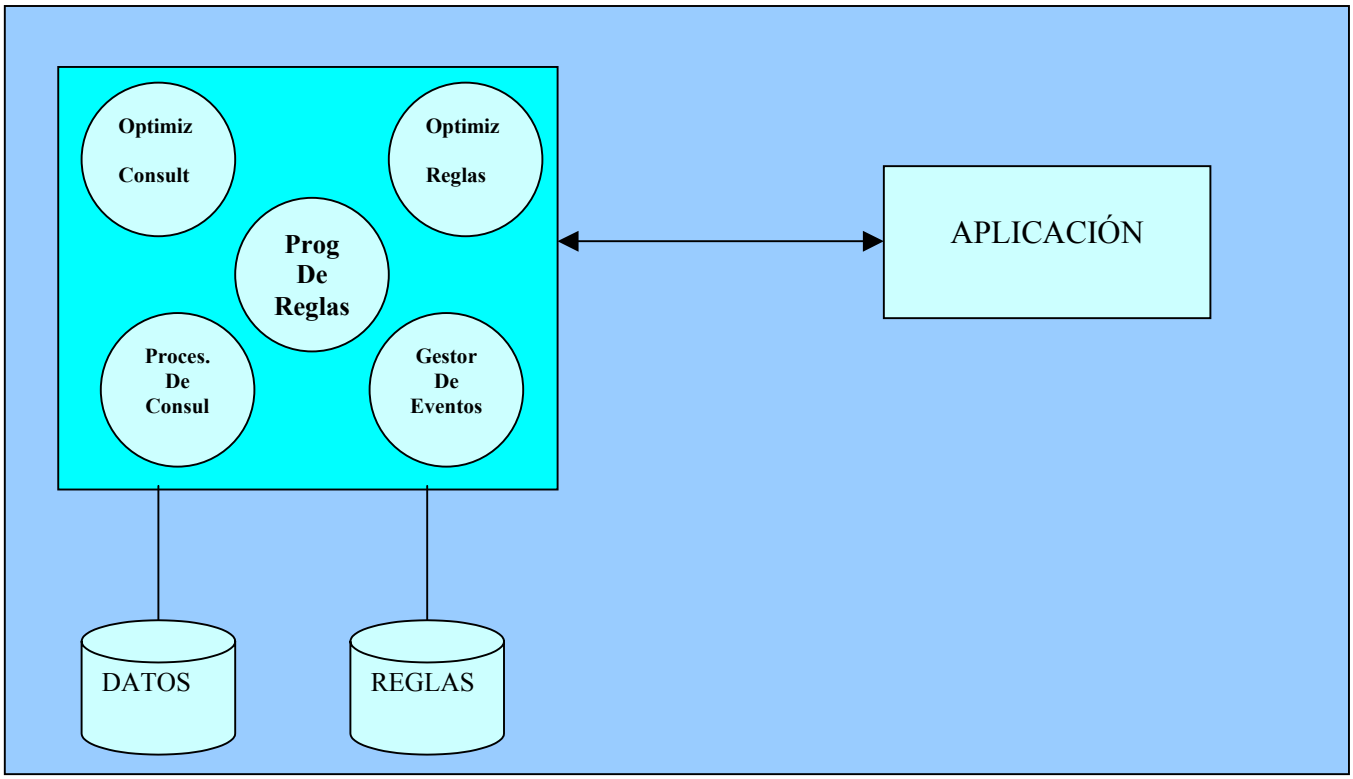


Figura 2.8: Arquitectura Física de una BDA

En la figura 2.8, se muestran las partes esenciales de un sistema de administración de bases de datos activa. En la parte inferior vemos una representación del lugar donde se guardan los datos y las reglas almacenadas.

- **Aplicación:** es un sistema de base de datos relacionadas con las necesidades particulares del negocio, el cuál se complementa con la BDA.
- **Optimizador de Reglas:** se encarga de que todas las operaciones se realicen de la manera apropiada. En concreto, la ejecución apropiada requiere que las propiedades ACID, abreviatura de los cuatro requisitos principales de la ejecución:
- **Atomicidad:** realizar la transacción completa o nada de ella.
- **Consistencia:** que la información cumple con nuestras expectativas.
- **Aislamiento:** cuando dos o más transacciones son ejecutadas al mismo tiempo, es preciso aislar sus efectos.
- **Durabilidad:** el efecto no debe perderse en caso de una falla del sistema.
- **Procesador de consulta:** se encarga de convertir una consulta o manipulación de la base de datos, que puede estar expresada en un nivel muy alto (ejemplo, como consulta en SQL), en un serie de peticiones de datos almacenados.

No solo maneja consultas sino también las peticiones de modificaciones de datos. Su función consiste en encontrar la mejor manera de llevar a cabo una operación solicitada y emitir comandos al administrador de almacenamiento que los ejecutará.

- **Optimizador de Consultas:** es el encargado de seleccionar un buen plan de consulta o sea una serie de peticiones al sistema de almacenamiento que las atenderá. Las consultas son realizadas por el usuario a la interfaz de la aplicación.
- **Programación de Reglas:** se encarga de la programación de las reglas de la base de datos. Estas reglas son programadas en la base de datos para mantener la consistencia y la integridad de los datos.
- **Gestor de Eventos:** es un programa incorporado por el fabricante a la base de datos que chequea las operaciones Insert, Update, Delete.

2.7 Arquitectura Lógica y Física de una BD Pasiva.

El objetivo del diseño lógico es convertir el esquema conceptual de datos en un esquema lógico ajustado específicamente al ***SGBD*** que se tenga a disposición. Obteniendo una representación que use de manera más eficiente los recursos para estruturar datos y modelar restricciones disponibles en el modelo lógico.

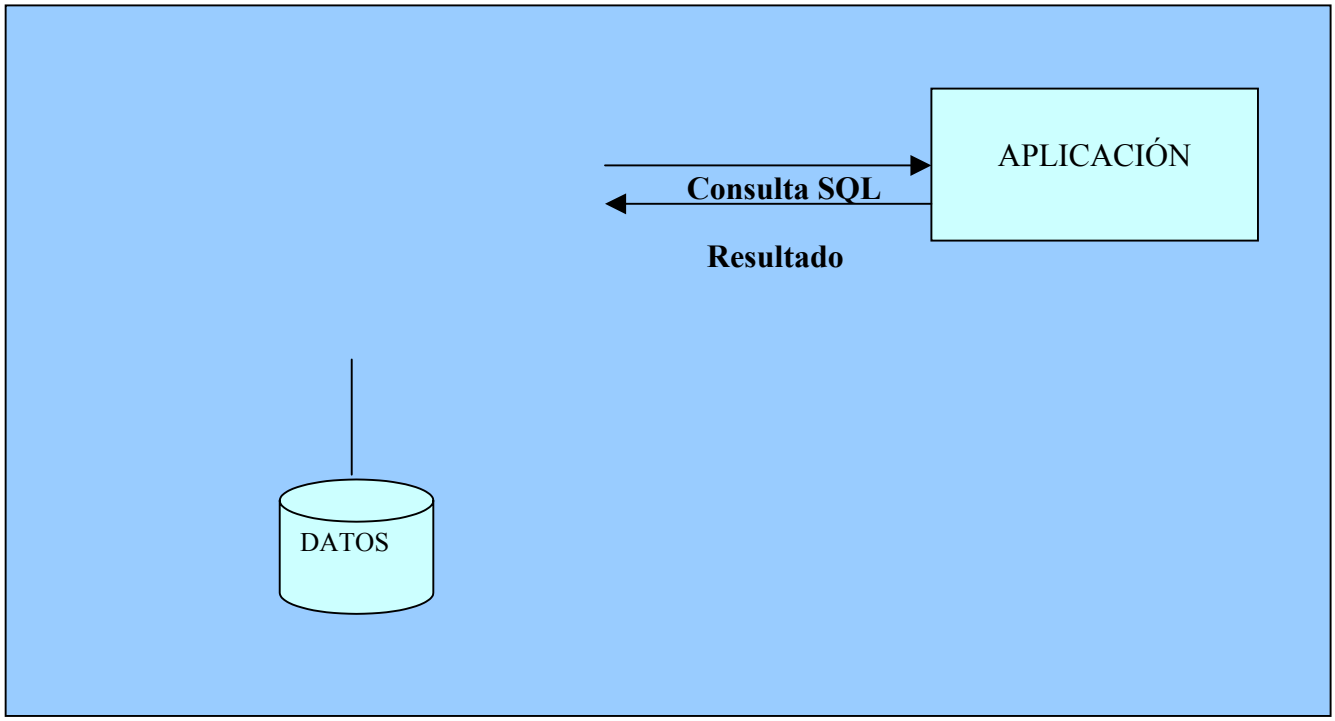


Figura 2.9: Arquitectura Física de una BD Pasiva

En la figura 2.9, se muestran las partes esenciales de un sistema de administración de bases de datos. En la parte inferior vemos una representación del lugar donde se guardan los datos, este componente no solo contiene datos sino que también metadatos, o sea información referente a la estructura de los datos.

En la misma Figura vemos también la **aplicación**, cuya función es obtener del almacenamiento de datos la información solicitada y modificarla allí cuando se lo pidan los demás niveles del sistema. También observamos un **Procesador de Consultas**, no solo maneja consultas, sino también las peticiones de modificaciones de datos o los metadatos. Su función consiste en encontrar la mejor manera de llevar a cabo una operación solicitada y emitir comandos a la aplicación que los ejecutará.

La aplicación es el componente encargado de conservar la integridad del sistema. Debe asegurarse de que las consultas que se ejecutan al mismo tiempo no interfieran entre ellas y de que el sistema no pierda información cuando sobrevenga alguna falla. Interactúa con el procesador de consultas, pues debe saber sobre que datos están operando las consultas actuales (con el fin de impedir acciones conflictivas) y posiblemente necesite posponer ciertas consultas u operaciones para evitar esos conflictos. Interactúa con la aplicación porque los esquemas de protección de información suelen requerir guardar en bitácora de los cambios efectuados en ella. Al organizar bien las operaciones, la bitácora contendrá un registro de los cambios, de modo que tras una falla del sistema puedan reejecutar incluso aquellos cambios que nunca llegaron al disco.

A diferencia de una base de datos activa, en una base de datos pasiva las reglas se encuentran almacenadas dentro de la aplicación.

2.8 Comparaciones entre una BDA y una BD Pasiva.

1.- Una base de datos Activa es reactiva, ya que es capaz de reaccionar ante eventos externos; mientras que una base de datos pasiva es pasiva porque debe ejecutar órdenes de actualización provenientes del usuario.

2.- El Subsistema de Integridad de un sistema de base de datos debe detectar y corregir en lo posible las Transacciones incorrectas, este es un punto muy débil, ya que casi toda la verificación de la integridad se realiza mediante código de procedimiento escrito por los usuarios; en cambio en las bases de datos activas los eventos son chequeados si existe alguna condición con respecto al evento producido, existen tablas almacenadas dentro de la base de datos que revisan al Evento en su tabla respectiva, si este está sujeto a alguna condición, se registra la tabla respectiva, para posteriormente efectuar la acción relativa al evento producido condicional.

3.- En cuanto a la implantación, un sistema de base de datos pasiva si es un sistema monousuario, no requiere equipos más sofisticados que los que normalmente soporta; la base de datos activa es costosa en equipos, tanto físico como lógico.

4.- La puesta en marcha de la base de datos pasiva no supera los limites acostumbrados; en cambio en las bases de datos activas su puesta en marcha es larga y costosa, debido a que es una nueva tecnología, aún poco utilizada.

5.- En las bases de datos pasivas, el número de personal necesario para administrar el Sistema es mayor; las bases de datos activas necesitan menor número de personal, pero más especializado.

6.- Los sistemas de bases de datos pasivas son independiente de los datos con respecto a los tratamientos; las bases de datos activas no sólo son independientes de los datos, sino también de los eventos.

7.- La base de datos pasiva es coherente, pero generalmente sufre trastornos, ya que la verificación de la integridad se realiza mediante códigos de procedimientos escritos por los usuarios; en cambio las bases de datos activas tienen mayor coherencia, debido a la preocupación por mantener la integridad de la Base de Datos en cada actualización.

8.- La flexibilidad del funcionamiento del sistema de base de datos pasivo está sujeto a la detección o petición del usuario, quien escribe los eventos mediante el código utilizado por el Sistema; las bases de datos activas debido a su arquitectura activa ofrece mayor flexibilidad para atender a demandas cambiantes.

2.9 Ventajas de una BDA

Centralización de la Información:

Esto permite un mejor mantenimiento, ya que las reglas son almacenadas dentro de la Base de Datos, si es necesaria alguna modificación se hace sólo una vez en el Diccionario de Datos, en lugar de hacerlo en cada Programa.

También permite una mayor Productividad, ya que los programas se simplifican al cargarlos de características que corresponden a la semántica de los datos.

Proporcionan mayor independencia de los datos ya que permiten realizar funciones que en los Sistemas Pasivos deberían ser programadas en el código de las aplicaciones.

Encapsulamiento de Procedimientos

Esto permite una mayor productividad ya que se pueden normalizar los procesos, sacando factor común de cierta lógica de los programas que se almacena una sola vez de forma centralizada.

Permite la Reutilización del código, ya que no es necesario escribirlo cada vez que se quiera realizar una determinada acción, sino que está almacenado en la Base de Datos, disponible cada vez que se necesite.

Rendimiento del Software

Reduce el Tráfico de mensaje, ya que al almacenar parte de los procedimientos en los servidores, se limita la cantidad de información que éstos deben devolver.

Además de la aplicación clásica de las Bases de Datos Pasivas(Construcción de Sistemas de Información), aparecen aplicaciones a otro tipo de problemas (Sistemas expertos con grandes volúmenes de datos).

Posibilidad de Optimización Semántica, ya que permite la posibilidad de tener acceso a la Base de Datos y cambiar lo que ya existe, con el fin de optimizarlo.

Facilitar el acceso a los usuarios finales, ya que al almacenar las reglas de actualización en el propio Sistema, éste podrá preservar la integridad de los datos, independientemente de cual sea el método de acceso empleado, lo que permite a los usuarios finales accesar a la Base de Datos sin peligro de dañarla.

2.10 Limitaciones de una BDA

Estructura Compleja: Es compleja en cuanto a su arquitectura interna, ya que consta de dos modelos que deben estar estrechamente entrelazados, el Modelos de Ejecución y Conocimiento.

Necesita herramientas de administración y diseño para su construcción: los modelos de Conocimiento y Ejecución deben ser diseñados con una visión integrada de lo que son los Triggers y las operaciones, de modo que los Modelos reaccionen a los Eventos de la manera esperada, sin sacrificar la consistencia de los datos en la Base de Datos.

Se necesita un gran Número de reglas para validar las operaciones de los Triggers, una regla simple podría generar la definición de muchos Triggers, debido a que se necesita uno para cada modificación de datos, esta proliferación de reglas puede llegar a complicar la consistencia de los datos ya que podría ocasionar contradicciones en las restricciones, es decir una acción de un trigger puede violar la condición de otro Trigger y viceversa, hasta continuar indefinidamente.

Necesita personal de alto nivel para supervisar la base de datos.

Para implementarla es necesario la utilización de equipos costosos: que permitan una implementación eficiente de la arquitectura física de la BDA.

Requiere de una interfaz inteligente capas de administrar lar Reglas ECA en forma Eficiente.

2.11 Aplicaciones de una BDA

- ***Comprobación de la integridad:*** La figura 2.10 muestra un modelo relacional y su correspondiente forma de guardar la integridad de los datos:

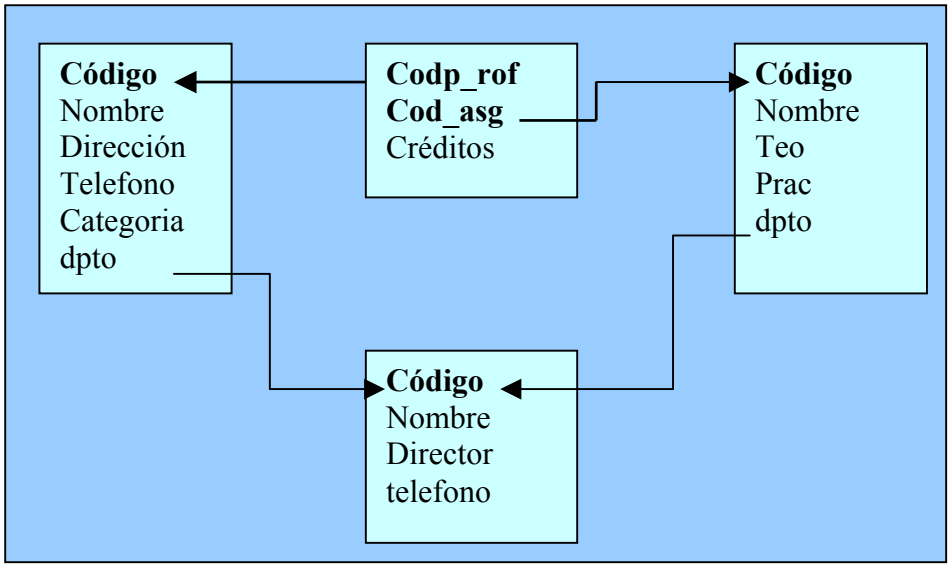


Figura 2.10: Modelo relacional de comprobación de integridad

" Los profesores que imparten la asignatura de Código Est1 (Estadística 1) deben ser del departamento de estadística (Est)". Ver Figura 2.11:

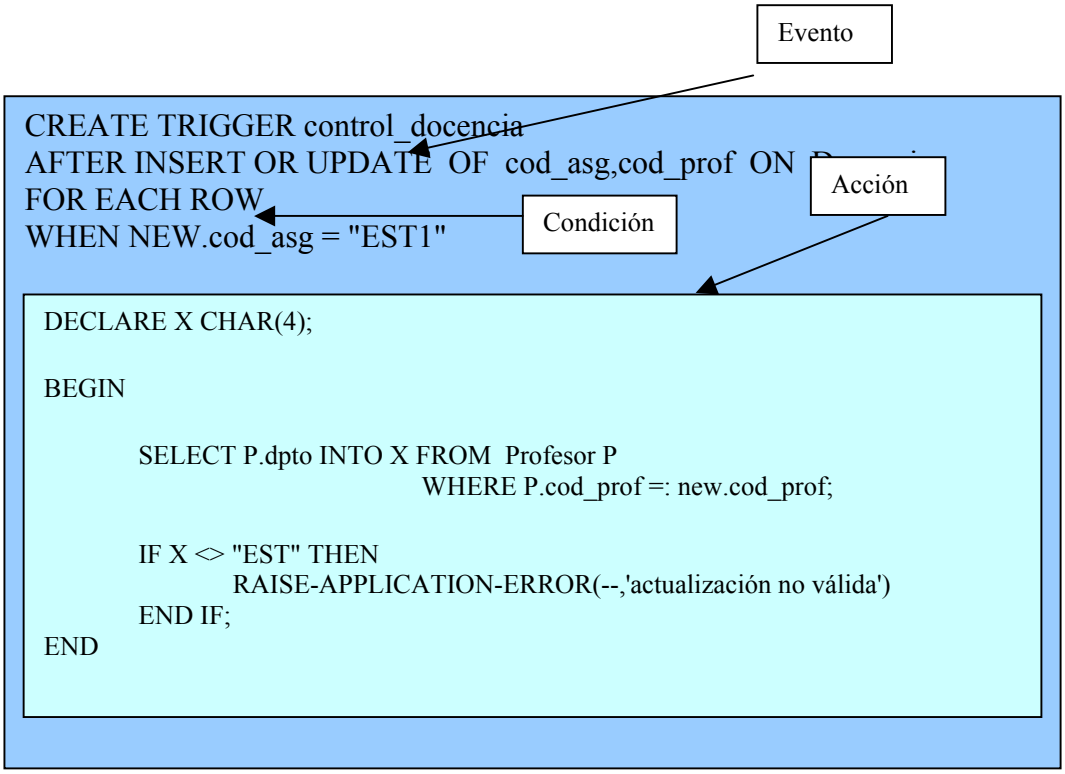


Figura 2.11: Trigger de comprobación de Integridad

- ***Restauración de la consistencia***
- ***Generación de datos derivados (materialización de vistas)***
- ***Control de la seguridad (accesos permitidos***
- ***Definición de las reglas de funcionamiento interno de la organización***.

El ejemplo que muestra la Figura 2.12, detalla el Comportamiento de las Reglas:

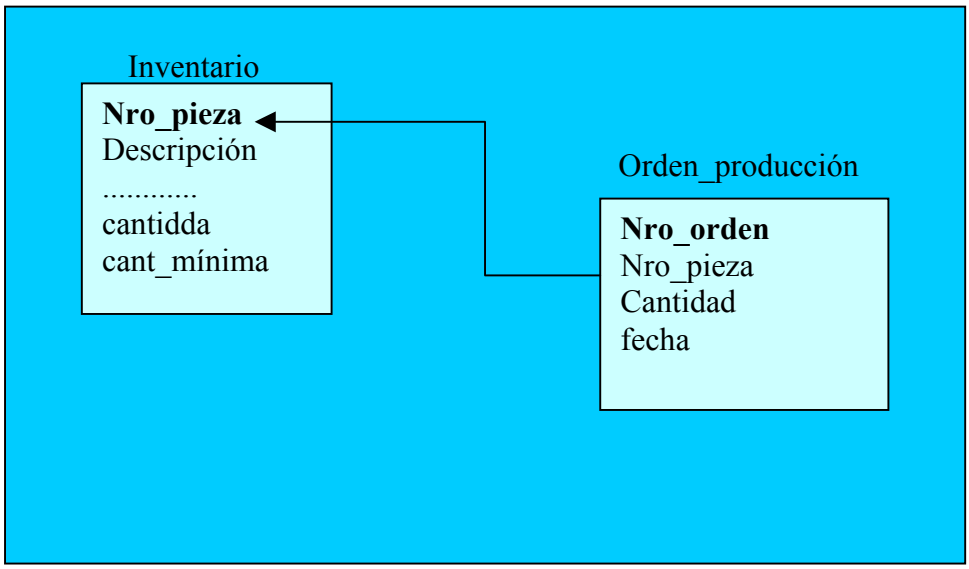


Figura 2.12: Modelo relacional reglas de funcionamiento de la organización

" Cuando la cantidad en almacén de una pieza esté por debajo de la cantidad mínima establecida, se debe lanzar un orden de producción para asegurar la cantidad mínima de la pieza en el almacén", ver Figura 2.13:

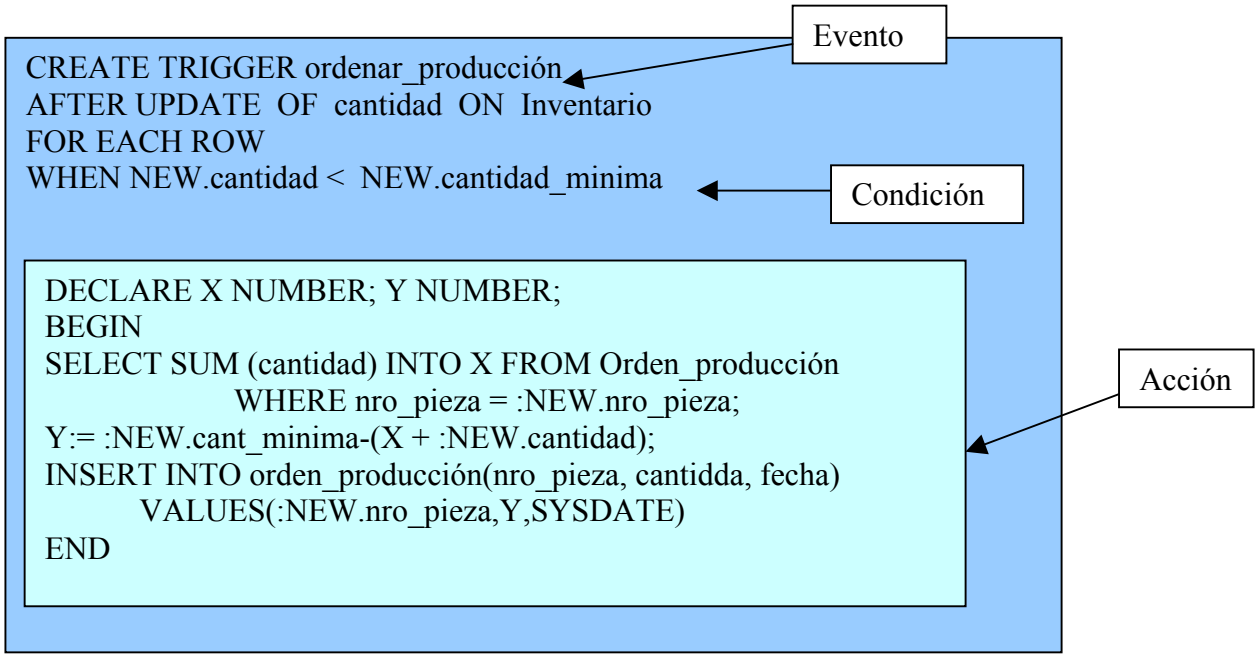


Figura 2.13: Trigger de Reglas de Funcionamiento de la Organización

3. Reglas de Integridad

Una regla de integridad esta compuesta por tres componentes, los cuáles son:
La **restricción** propiamente tal, que establece la condición que deben cumplir los datos.
La **respuesta a la transgresión**, que especifica las acciones a tomar, como rechazar las operaciones, informar al usuario, corregir el error con acciones complementarias, etc.
Condición de disparo, que especifica cuando debe desencadenarse la acción especificada en la restricción de integridad: antes, después o durante cierto evento.
Las reglas de integridad deben almacenarse en el diccionario de datos, como parte integrante de los datos (como control centralizado de la semántica), de modo que no han de incluirse en los programas.
Por lo tanto como el término de **Integridad de Datos** se refiere a la corrección y completitud de los datos en una base de datos, es decir, que los datos sean correctos, es necesario implementar por medio de **Restricciones en SQL**.
Existen **3 tipos de Integridad**:

- **Restricción de Integridad de Entidad**

La restricción de integridad de entidad establece que ningún valor de identificador primario puede ser nulo ni repetirse. Esto es porque ellas identifican tuplas de la relación en forma única. Se Implementa a través de la Llave Primaria.
Existe un caso especial de integridad que es el caso de clave única que no necesariamente es no nula como la primary key, que puede servir como clave alternativa. Se implementa a través de una clave única.

- **Restricción de Integridad de Dominio**

Para un atributo dado se restringen los valores permitidos para ese dominio. Se pueden restringir los valores de un atributo declarando un dominio con una restricción similar y luego declarándolo como el tipo de datos del atributo.
Se implementan en SQL (checks, TRIGGERS, procedimientos almacenados).

- **Restricción de Integridad Referencial**

La restricción de integridad referencial se especifica entre dos relaciones y se usa para mantener la consistencia entre tuplas de las dos relaciones. Informalmente, una tupla en una relación que hace referencia a otra relación debe referirse a una tupla existente en esa relación. Para 2 tuplas que están relacionadas, se debe asegurar que las tuplas en la Tabla Referencial (Débil) corresponden a Tuplas Existentes en la Tabla Referenciada (Fuerte).
Se implementan en SQL (foreign keys, TRIGGERS).

Verificación en 3 momentos:

Insertar en T. Débil: (Verifica si el valor de la llave foránea existe en tabla fuerte)
Eliminar en T. Fuerte: (Restringe (no borra), Cascada (fuerte, débil, nulo - fuerte))
Modificar en T. Fuerte: (Al modificar llave en t. fuerte, es modificada en tabla débil)

3.1 Reglas del Negocio

La introducción de la actividad lleva consigo más trabajo a la hora de diseñar bases de datos, debiéndose examinar cuidadosamente las reglas del negocio, que se codifican mediante disparadores y se implementan mediante restricciones de forma declarativa.
Toda aplicación trata de reflejar parte del funcionamiento del mundo real, para automatizar tareas que de otro modo serían llevadas a cabo de modo más ineficiente, o bien no podrían realizarse. Para ello, es necesario que cada aplicación refleje las restricciones que existen en el negocio dado, de modo que nunca sea posible llevar a cabo acciones no válidas. A las reglas que debe seguir la aplicación para garantizar esto se las llama **reglas de negocio**.

En realidad, la información puede ser manipulada por muchos programas distintos, el hecho de que la información sea manipulada por diversos programas hace más difícil garantizar que todos respetan las reglas, especialmente si las aplicaciones corren en diversas máquinas, bajo distintos sistemas operativos, y están desarrolladas con distintos lenguajes y herramientas.

Este tipo de Reglas proporcionan Integridad, el cuál es el objetivo central de una BDA.

Tipos de reglas de negocio

Reglas del Modelo de Datos: El primer grupo de reglas de negocio engloba todas aquellas reglas que se encargan de controlar que la información básica almacenada para cada atributo o propiedad de una entidad u objeto es válida: no hay precios de artículos negativos, el sexo de una persona solo puede ser masculino o femenino, una fecha siempre debe ser una fecha válida (no existe el 30 de Febrero, ¿cierto?), etc.

Reglas de Relación: Otro grupo importante de reglas incluye todas aquellas reglas que controlan las relaciones entre los datos. Estas reglas especifican, por ejemplo, que todo pedido debe ser realizado por un cliente, y que el mismo debe estar dado de alta en nuestro sistema: además, una vez que un cliente haya hecho algún pedido, se deberá garantizar que no es posible eliminarlo, a menos que previamente se eliminen todos sus pedidos.

Reglas de Derivación: Es frecuente que a partir de cierta información se pueda derivar otra: por ejemplo, el total de un pedido se puede calcular a partir de las distintas líneas que lo componen, mientras que el total de cada línea se puede calcular a partir del número de unidades vendidas y el precio por unidad. Al conjunto de reglas que especifican y controlan la obtención de información que se puede calcular a partir de la ya existente se las llama reglas de derivación.

Reglas de Restricción: Son las que restringen los datos que el sistema puede contener. Nótese que este grupo de reglas se solapa en cierto modo con las reglas del modelo de datos, dado que aquellas también impiden la introducción de datos erróneos, como se vio anteriormente. La diferencia estriba en que las reglas de restricción restringen el valor de los atributos o propiedades de una entidad más allá de las restricciones básicas que sobre las mismas existen: por ejemplo, para un saldo existe una regla básica (regla del modelo de datos) que indica que éste debe ser un número (¡no por obvia es menos regla!), pero, además puede haber una regla que indique que el saldo nunca puede ser menor que cierta cantidad tope establecida para cierto tipo de clientes. Esta sería lo que aquí denominamos una regla de restricción, y la diferencia fundamental estriba en el hecho de que este tipo de reglas requiere para su verificación del acceso a otros fragmentos de información, algo que no sucede con las reglas del modelo de datos. Esto tiene ciertas consecuencias que se verán más adelante.

Reglas de Flujo: El último grupo de reglas de negocio incluye aquellas reglas que determinan y limitan cómo fluye la información a través de un sistema. Por ejemplo, un cliente puede hacer una petición de análisis a un laboratorio, que anota un encargado: hecho esto, se genera un parte para uno o más analistas, estos realizan las mediciones correspondientes y devuelven los partes con la información pertinente, a partir de la cuál se genera un informe de análisis, que será un análisis válido solo cuando sea firmado por los responsables de garantizar su corrección. A las reglas que indican qué camino recorre la información y obligan a que se sigan solo los caminos válidos se las llama reglas de flujo.

4. Metodología Utilizada en el Proyecto

Utilizamos dos tipos de metodologías para abordar el desarrollo del Seminario, una metodología de investigación en la cuál adquirimos conocimientos acerca del tema y una de desarrollo, basada en CASE METHOD de Oracle , que nos sirvió de apoyo en el diseño y la implementación.

4.1 Etapa de Investigación.

Investigamos bibliografías de diferentes autores para adquirir antecedentes teóricos acerca del tema. También realizamos clases expositivas, tanto de nuestro profesor guía a nosotras, como de nosotras a él a medida que se adelantaba en el desarrollo del seminario, lo cuál nos ayudo al desarrollo del seminario. Los libros estudiados fueron:

- Introducción a los Sistemas de Bases de Datos de Jeffrey Ullman, en el que adquirimos conocimiento acerca de la integridad de los datos y triggers.
- Diseño de Base de Datos Relacionales de Miguel Castaño, en el que adquirimos información acerca del comportamiento de una base de datos pasiva.
- Manual de Referencia de George Koch, este libro nos aporto información para introducirnos a Oracle.
- Bases De Donnees Object & relational de georges Gandanin, este libro fue un gran aporte sobretodo en el tema de la integridad de los datos, el cuál es uno de los temas más importantes en una BDA.
- Aplique SQL de James Groff, al momento de construir las tablas en la creación de la base de datos, este libro nos fue de gran ayuda en todas las dudas que se nos presentaron en el camino.
- Sistemas de Bases de Datos, Conceptos Fundamentales de Elmasri y navathe,
- Manual de Oracle de Paul Dorsey, nos ayudo a comprender conceptos de Oracle que eran desconocidos.
- Manual de Oracle Developer/2000 de Robert Muller, este libro nos ayudo a introducirnos en el Developer, el cual es el que aporta todas las herramientas necesarias para la creación de Pantallas y Forms, es decir, todo lo físico de la BDA.
- Manual de Oracle Designer/2000 de Paul Dorsey, este libro nos ayudo a construir el diseño de la BDA, pasando por cada una de las etapas de la herramienta CASE.

- Diseño conceptual de bases de datos de Batini Ceri, nos entrego información acerca de la arquitectura lógica de una BD pasiva.
- Active Rules in Database Systems de Norman Paton, nos ayudo a comprender mejor el comportamiento de las bases de datos activas.

También revisamos lecturas complementarias encontradas en Internet como:

- Bases de Datos Avanzadas de Matilde Celma, relacionada con la página:
  http://www.dsic.upv.es/ asignaturas/facultad/bdv/documentos/ teoria/temaII_1.pdf.
  Con este material pudimos adquirir un conocimiento más profundo del como se comporta una BDA.
- Bases de Datos Activas de Mario Piattini, relacionada con la página:
  http://alarcos.inf- cr.uclm.es /doc/ bbddavanzadas/ doc99activas.pdf.
  Este nos ayudo a comprender las fases de una BDA.

4.2 Etapa de Desarrollo

Fases del Desarrollo de Sistemas Según **Metodología ORACLE - CASE -Method**.

La herramienta CASE es una metodología para el diseño de sistemas, ya que automatizan la mayor parte del trabajo manual, repetitivo y propenso a errores necesarios en el desarrollo de sistemas, pueden, si se utilizan correctamente, incrementar grandemente la productividad de aquellos que producen los sistemas, así como la exactitud del diseño y la robustez de la implementación.

Las fases de una Metodología CASE en Designer 2000 se ilustran en la Figura 4.1:

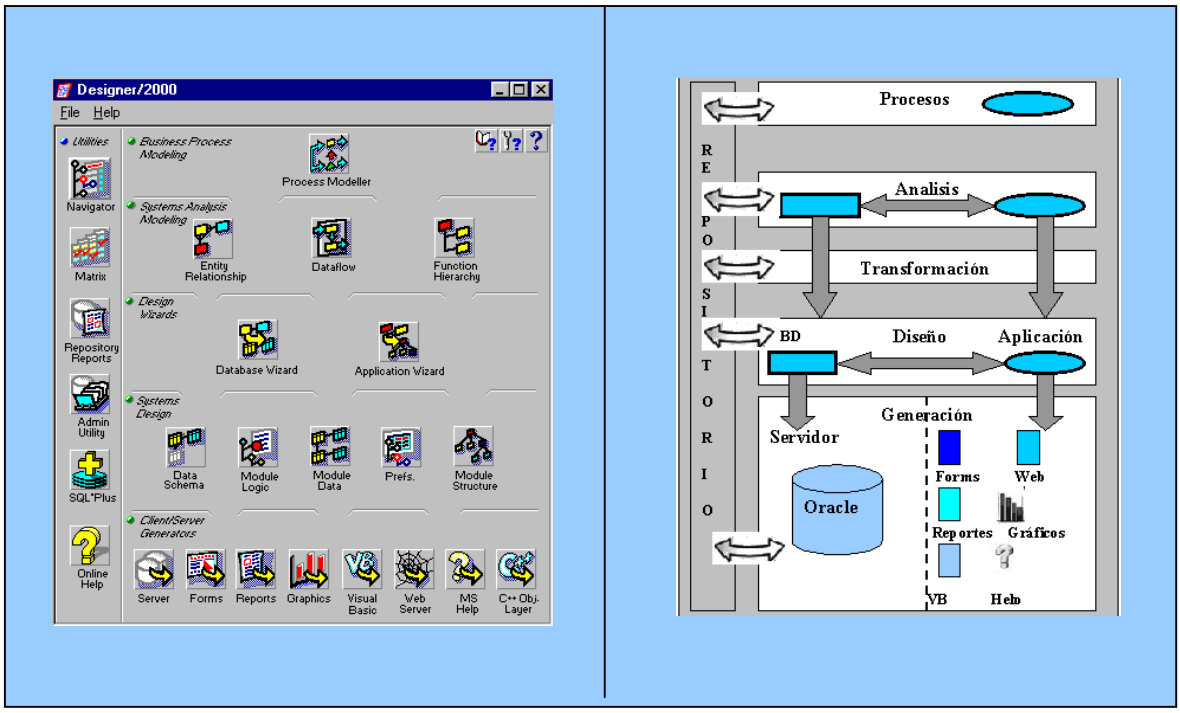


Figura 4.1: Etapas de desarrollo Designer 2000

Fases Case Method.

1. ***Estrategia***: estudio de procesos mediante el desarrollo de un diagrama de procesos.
2. ***Análisis***:
    a. Modelo entidad Relación nivel lógico
    b. Modelo DFD que se obtiene del diagrama de procesos(1)
    c. Diagrama Funcional que se obtiene de l DFD (c)
3. ***Diseño***
    a. Diseño Físico de Tablas Se obtiene de 2.a
    b. Diseño de módulos y programas: se especifica la funcionalidad de 2.c y se dibujan pantallas y reportes
    c. Se especifica los procedimientos almacenados
4. ***Construcción***
    a. Creación y ejecución del Script de tablas junto con el script de restricciones de chequeo, FK y PK
    b. Creación de los módulos Pantallas y Reportes
    c. Creación y ejecución de los scripts de procedimientos almacenados
5. **Documentación**
    a. Informes de Análisis, Diseño y Construcción

5. Implantación del sistema

La Figura 5.1, detalla todos los elementos utilizados para poder llevar a acabo la arte practica de este estudio, para lo cual fue necesario crear una arquitectura básica de una BDA usando algunos productos de Oracle.

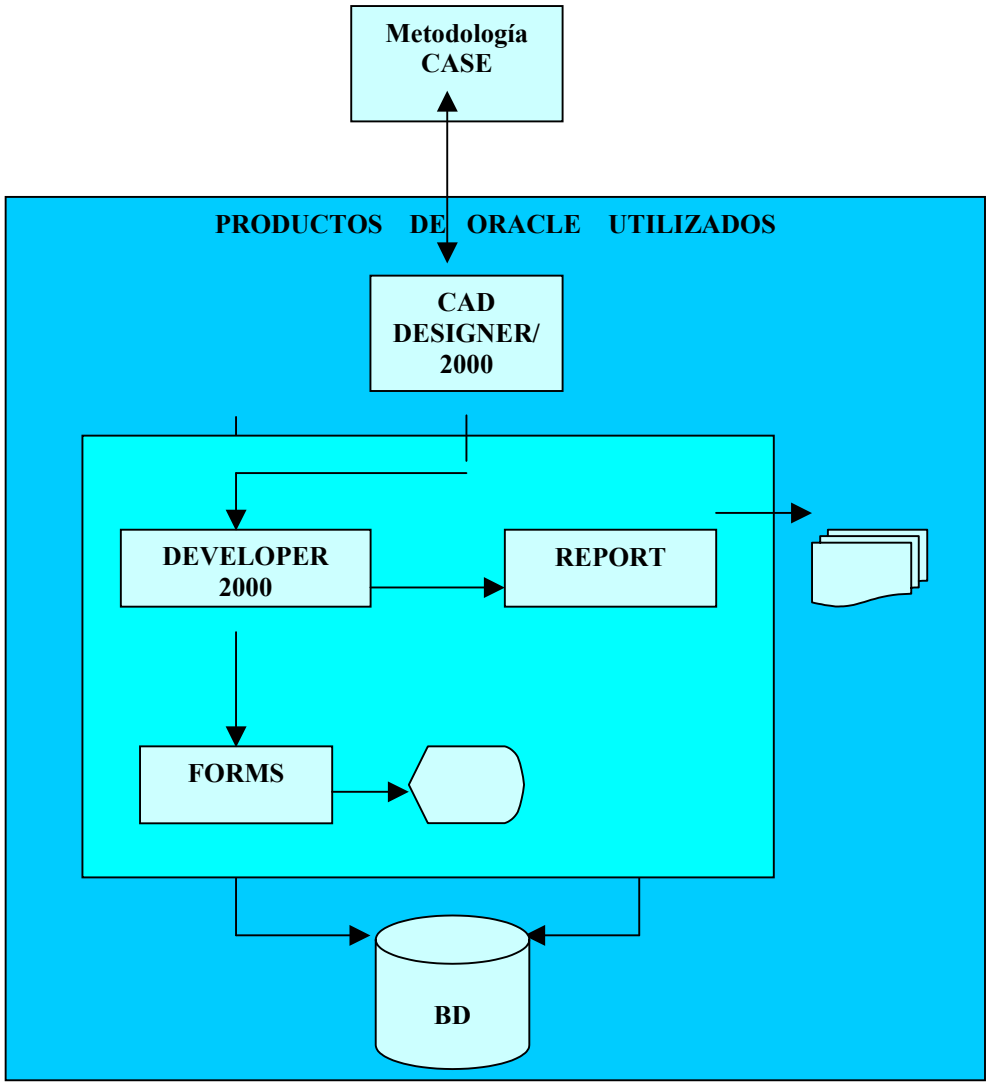


Figura 5.1: Metodología de Desarrollo

5.1. Fase de Estrategia

La creación del diagrama de procesos con la funcionalidad de mantención de los parámetros de las bases de datos activas: Variables, Eventos, Condiciones, Acciones, Procesos, Privilegios, y su relación con los almacenamientos etc.

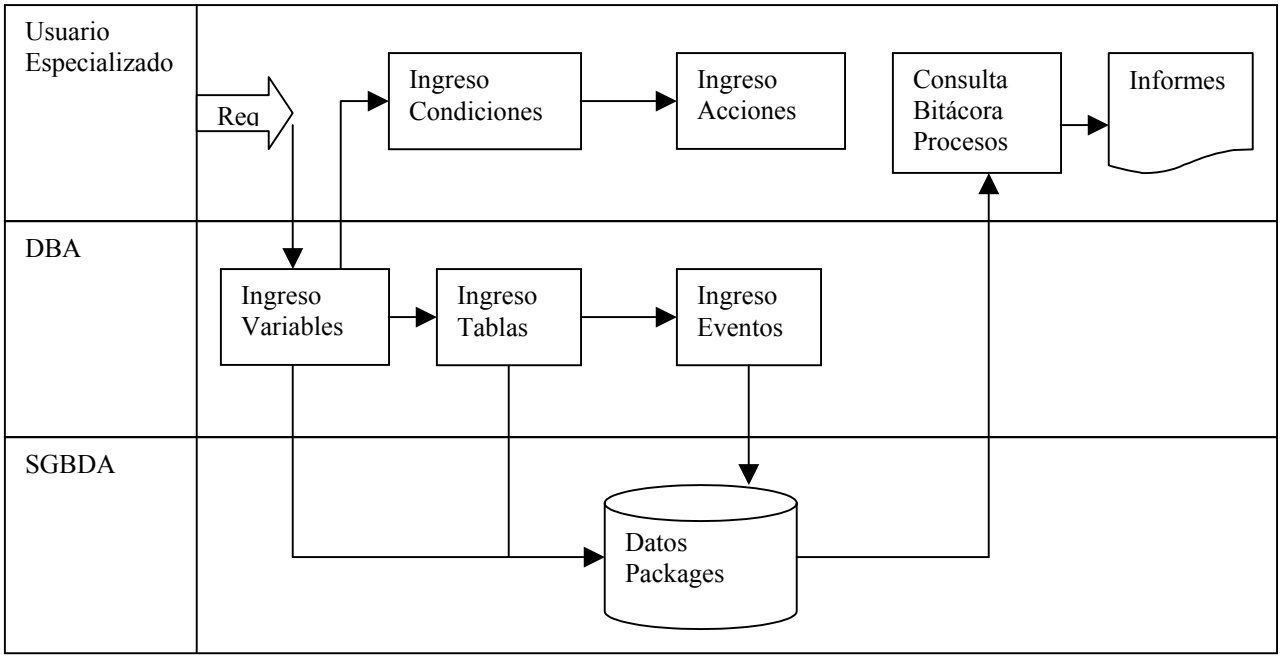


Figura 5.2: Diagrama de Procesos

5.2 Análisis

5.2.1 Análisis de datos:

MER

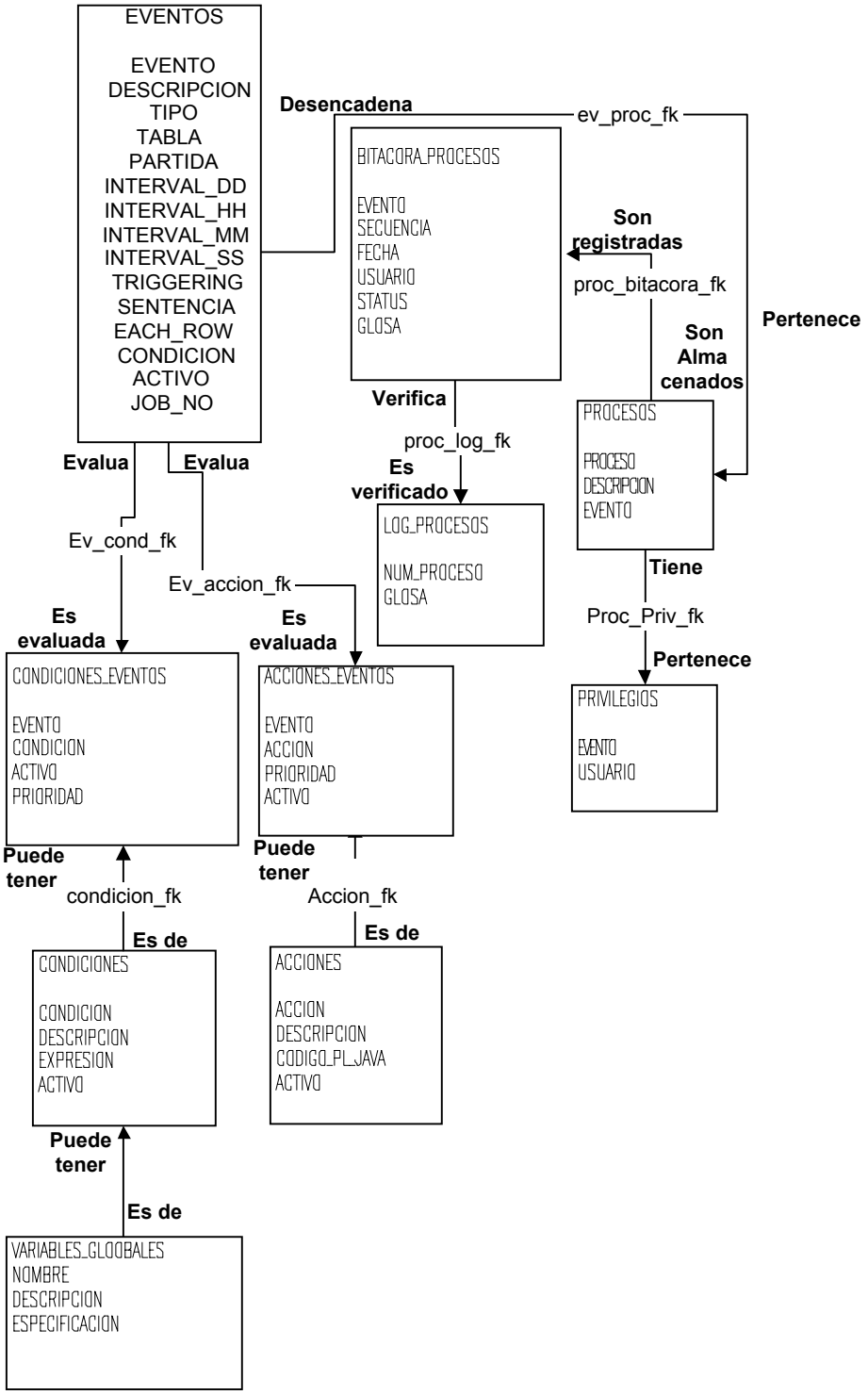


Figura 5.3: Modelo Entidad Relación

5.2.2 Análisis de Flujo de los datos

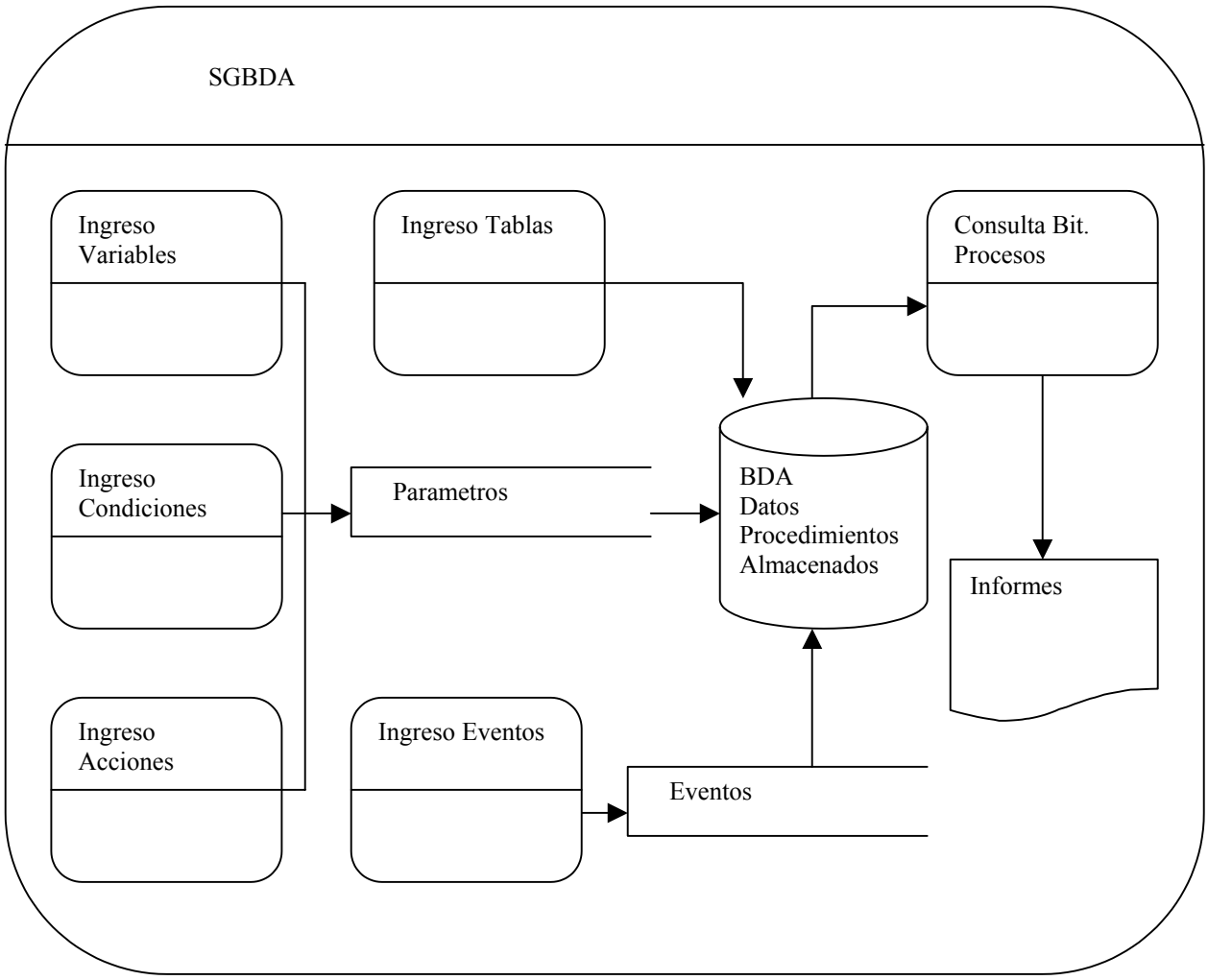


Figura 5.4: Diagrama de Flujo de Datos

5.2.3 Análisis funcional

A partir de los Parámetros se evalúan las condiciones y se ejecutan las acciones, y los cambios se efectúan según el estado de las variables y las tablas asociadas con el codigo de Programa de la aplicación, los Eventos son evaluados por el manejador de Eventos, y finalmente las consultas se almacenan en la bitácora de procesos, con esta se realizan informes de las acciones y condiciones ejecutadas.

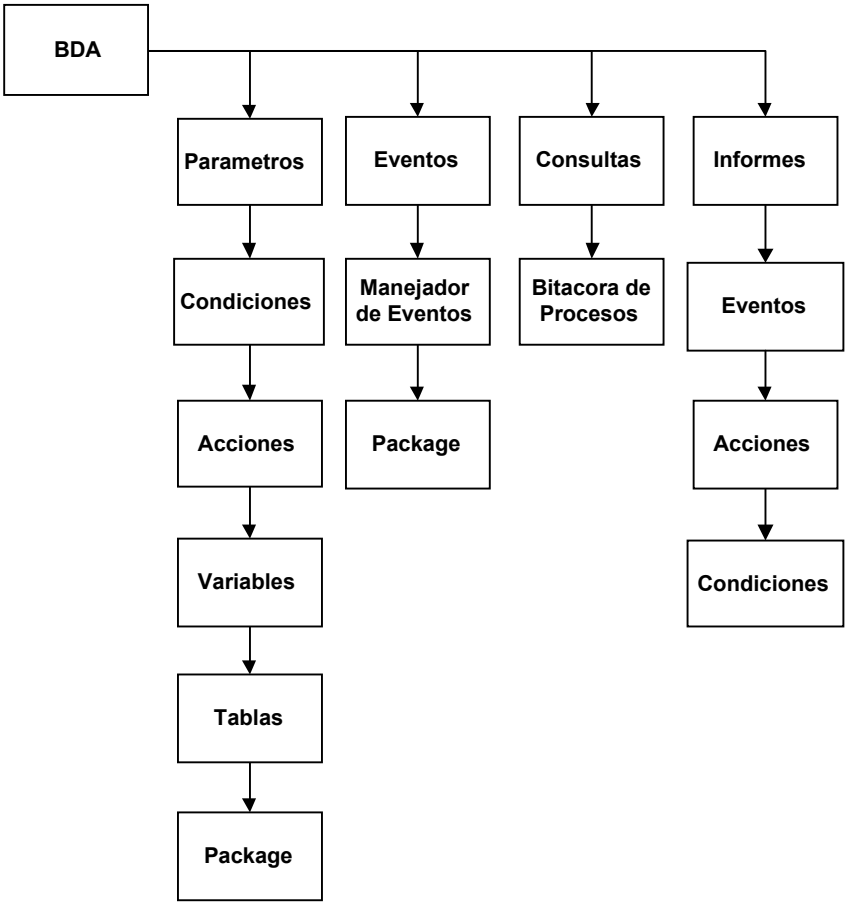


Figura 5.5: Diagrama Funcional de la BDA

5.3 Fase de Diseño:

En nuestra aplicación, la tabla de Eventos es la que almacena todos los Eventos, estos pueden ser de 2 tipos:

- **Tabla:** están asociados a un trigger, se ejecuta antes o despues de una inserción, actualización o eliminación.
- **Temporal:** están asociados a un Job, se ejecutan a una determinada hora definida por el usuario.

Cada evento tiene asociado una Condición y una acción, es a través de la Tabla condiciones_eventos y acciones_eventos que el evento se relaciona con las condiciones y acciones. Se evalúan las condiciones del evento y si todas son verdaderas ejecuta las acciones asociadas al evento, siempre y cuando el usuario tenga privilegios.

Esto desencadena finalmente un proceso, el cual tiene los privilegios del usuario para poder ejecutar la Base de Datos y almacena en la Bitácora de procesos los datos del evento, como la fecha en que fue ejecutado, el estado en que se encuentra y que usuario lo utilizo, finalmente en la tabla log_Procesos queda verificado el numero del Proceso según el orden de prioridad el cual es verificado por la misma Bitácora de Procesos.

5.3.1 Diseño Físico de los Datos

Primary key

evento_pk : clave primaria de la tabla eventos
bitacora_pk : clave primaria de la tabla bitacora_procesos
condiciones_pk : clave primaria de la tabla condiciones
acciones_pk : clave primaria de la tabla acciones
privilegios_pk : clave primaria de la tabla privilegios

Foreing Key.

fk_evento_condiciones: Esta clave foránea relaciona la tabla de eventos y condiciones_eventos.

fk_condicion_condiciones : Esta clave foránea relaciona la tabla condiciones_eventos y condiciones.

fk_evento_acciones : Esta clave foránea relaciona la tabla acciones_eventos y eventos.

fk_accion_acciones : Esta clave foránea relaciona la tabla acciones_eventos y acciones.

fk_evento_bitacora : Esta clave foránea relaciona la tabla bitacora_procesos y eventos.

fk_evento_privilegios: Esta clave foránea relaciona la tabla privilegios y eventos.

Diagrama de Tablas:

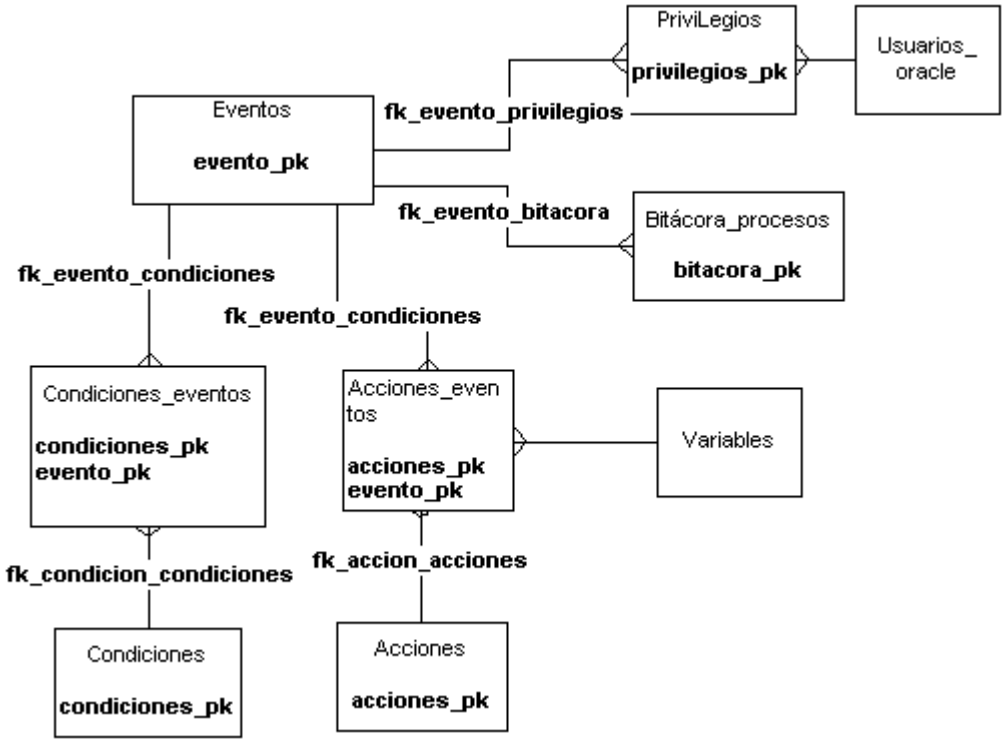


Figura 5.6 : Diagrama de Tablas

5.3.2 Diseño Físico de la Aplicación

Las Funciones del Package son:

- **Función submit_job**

  Parámetros de la Función:

  p_proc, p_fecha, p_interval

  Esta Función ejecuta un proceso oracle p_proc, para su ejecución en la fecha p_fecha, con un intervalo de ejecución p_intervalo medido en segundos.

- **Procedimiento execute_eca**

  Parámetro: p_evento

  Procedimiento que verifica las condiciones del evento p_evento y si todas son verdaderas ejecuta las acciones asociadas al evento p_evento, pero si el usuario no tiene privilegios, no se ejecuta la acción.

- **Procedimiento create_tab**

  Parámetro: p_table

  Procedimiento que crea una tabla correspondiente a la tabla p_table

- **Procedimiento create_pack_tab**

  Parámetro: p_table

  Procedimiento que crea un package asociado a la tabla p_table con variables correspondientes a las columnas de la tabla.

- **Función get_new_old_str**

  Parámetro:p_table

  Función que retorna un string con las asignaciones de las variables new y old correspondientes a las columnas de la tabla

- **Procedimiento execute_acc**

  Parámetro:p_acc

  Procedimiento que ejecuta una acción p_acc.

- **Procedimiento execute_str**

  Parámetro: p_str

  Procedimiento que ejecuta una instrucción ddl p_ddl

- **Función get_cond**

  Parámetro: p_cond

  Función que retorna el resultado de la evaluación de la condición p_cond
- **Función existe_obj**

  Parámetro: p_obj

  Función que retorna TRUE si el objeto existe en la BD y FALSE en caso contrario
- **Función get_status**

  Parámetro: p_obj, p_type

  función que retorna el STATUS del objeto
- **Función get_next_date**

  Parámetro :p_job_no

  función que retorna la próxima ejecución del job
- **Procedimiento create_obj**

  Parámetros: p_obj, p_cuerpo, p_obj_type, p_return_type

  Procedimiento que crea el objeto p_obj de tipo p_type
- **Procedimiento drop_obj**

  Parámetro: p_obj, p_type

  Procedimiento que borra el objeto p_obj de tipo p_type
- **Procedimiento drop_job**

  Parámetro: p_job_no

  Procedimiento que borra el job p_job_no
- **Procedimiento enable_trigger**

  Parámetro: p_trigger

  Función que habilita el Trigger **p_trigger** asociado al Evento
- **Procedimiento disable_trigger**

  Parámetro: p_trigger

  Función que deshabilita el Trigger **p_trigger** asociado al Evento.
- **Procedimiento active_condicion**

  Parámetro: p_condicion

  Activa la condición, p_condicion, actualizando la tabla condiciones_eventos, para relacionar la condición con el Evento y la Acción asociada.

- **Procedimiento active_accion**

  Parámetro: p_accion

  Activa la Acción, p_accion, actualizando la tabla acciones_eventos.

- **Procedimiento desactive_condicion**

  Parámetro: p_condicion

  Desactiva la condicion, p_condicion, actualizando la tabla condiciones_eventos.

- **Procedimiento desactive_accion**

  Parámetro: p_accion

  Desactiva la Acción, p_accion, actualizando la tabla acciones_eventos.

- **Procedimiento ins_bitacora**

  Parámetros: p_evento, p_status, p_glosa

  Inserta en la Bitacora de Procesos el Evento(p_evento), el estado del Evento, p_status, (Valido o Invalido) y la descripción del Evento.

  - **Función get_privilegio**

  Parámetros: p_evento, p_user

  Función que retorna TRUE si el evento y el usuario se encuentran en la tabla de privilegios FALSE en caso contrario

5.4. Construcción

5.4.1 Construcción de la Base de datos

Scripts de creación:

```
drop table eventos
/
create table eventos (
evento varchar2(30) not null,
descripcion varchar2(100) not null,
tipo varchar2(15) not null,          -- tabla / temporal
tabla varchar2(30),                  -- nombre de tabla
partida date,                        -- fecha y hora de la primera ejecucion
interval_dd number,                  -- periodicidad de ejecucion en dias
interval_hh number,                  -- periodicidad de ejecucion en horas
interval_mm number,                  -- periodicidad de ejecucion en minutos
interval_ss number,                  -- periodicidad de ejecucion en segundos
triggering varchar2(10),             -- before /after
sentencia varchar2(10),              -- insert, update, delete
each_row varchar2(1),                -- 'Y' or 'N'
condicion varchar2(200),             -- condicion activacion
activo varchar2(1) not null,         -- registro habilitado (S/N)
job_no number                        -- nro de job
)
/
drop table condiciones
/
create table condiciones (
condicion varchar2(30) not null,
descripcion varchar2(100) not null,
expresion varchar2(2000) not null,
activo varchar2(1) not null          -- registro activo (S/N)
)
/
drop table acciones
/
create table acciones (
accion varchar2(30) not null,
descripcion varchar2(100) not null,
codigo_pl_java varchar2(2000) not null,
activo varchar2(1) not null          -- registro habilitado (S/N)
)
/
drop table condiciones_eventos
/
create table condiciones_eventos (
evento varchar2(30) not null,
condicion varchar2(30) not null,
```

```
activo varchar2(1) not null,          -- registro activo (S/N)
prioridad number not null
)
/
drop table acciones_eventos
/
create table acciones_eventos (
evento varchar2(30) not null,
accion varchar2(30) not null,
prioridad number not null,
activo varchar2(1) not null          -- registro activo (S/N)
)
/
drop table variables
/
create table variables (
variable varchar2(30) not null,
descripcion varchar2(100) not null,
tipo_variable varchar2(30) not null,
consulta_sql varchar2(2000) not null
)
/
drop table variables_globales
/
create table variables_globales (
nombre varchar2(30) not null,
descripcion varchar2(80) not null,
especificacion varchar2(2000) not null
)
/
drop table bitacora_procesos
/
create table bitacora_procesos (
evento varchar2(30) not null,
secuencia number not null,
fecha date not null,
usuario varchar2(30) not null,
status varchar2(30) not null,
glosa varchar2(2000) not null
)
/
drop table privilegios
/
create table privilegios (
evento varchar2(30) not null,
usuario varchar2(30) not null
)
/
```

```
drop table tablas
/
create table tablas (
tabla varchar2(30),
descripcion varchar2(80)
)
/
drop table columnas_tablas
/
create table columnas_tablas (
tabla varchar2(30) not null,
orden number not null,
columna varchar2(30) not null,
descripcion varchar2(80) not null,
tipo varchar2(30) not null,
tamano number,
nula varchar2(1) not null
)
/
--
-- primary keys
--
alter table eventos add constraint evento_pk primary key (evento) using index
/
alter table bitacora_procesos add constraint bitacora_pk primary key (secuencia) using index
/
alter table condiciones add constraint condiciones_pk primary key (condicion) using index
/
alter table acciones add constraint acciones_pk primary key (accion) using index
/
alter table privilegios add constraint privilegios_pk primary key (evento,usuario) using index
/
--
-- foreign keys
--
alter table condiciones_eventos add constraint fk_evento_condiciones foreign key (evento)
references eventos(evento)
/
alter table condiciones_eventos add constraint fk_condicion_condiciones foreign key
(condicion) references condiciones(condicion)
/
alter table acciones_eventos add constraint fk_evento_acciones foreign key (evento)
references eventos(evento)
/
alter table acciones_eventos add constraint fk_accion_acciones foreign key (accion) references
acciones(accion)
/
```

```
alter table bitacora_procesos add constraint fk_evento_bitacora foreign key (evento)
references eventos(evento)
/
alter table privilegios add constraint fk_evento_privilegios foreign key (evento) references
eventos(evento)
/
```

5.4.2 Construcción de la aplicación:

Creación de la Interfaz(pantallas) de la Aplicación Nivel Cliente: Para la Construcción se utilizó Developer/2000

Eventos

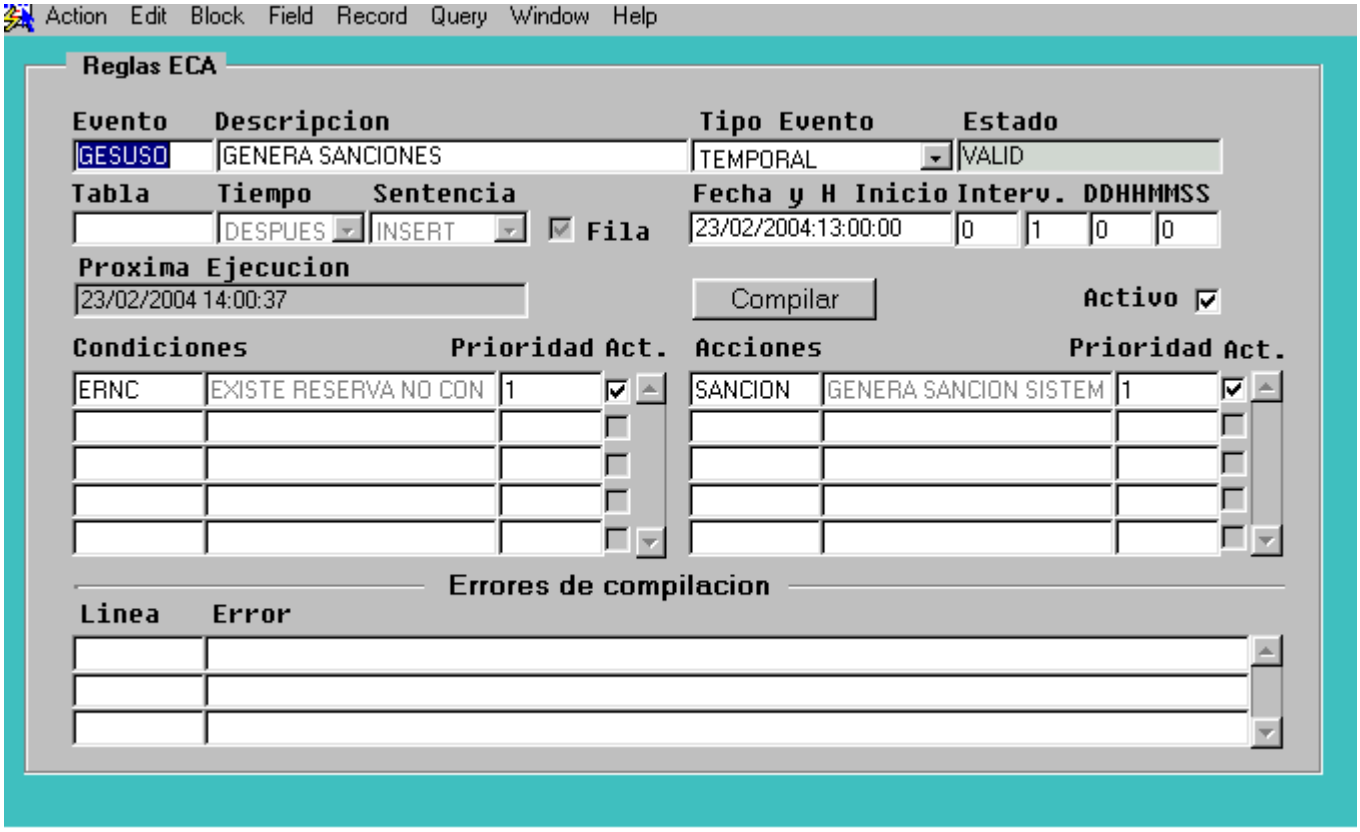


Figura 5.7: Pantalla de Eventos

La pantalla Eventos de la Figura 5.5, es una pantalla importante, en donde esta el evento a ejecutarse, las condiciones y las acciones asociadas a ese evento que ejecuta la acción una vez que la condición sea verdadera, podemos observar diferentes campos, los cuáles son:

- **Evento:** muestra el evento
- **Descripción:** describe el evento
- **Tipo Evento:** ***puede ser de 2 tipos: de tipo temporal cuando esta asociado a un JOB o de tipo tabla cuando esta asociada al TRIGGER.***

- **Estado:** ***es valido o invalido, una vez que el evento a sido compilado exitosamente se encuentra valido.***
- **Condiciones:** aquí es donde se almacenan todas las condiciones que están asociadas al evento, con su descripción.
- **Acción:** también incluye varios campos y al igual que en la condición son ingresadas todas las acciones que se pueden ejecutar asociadas al evento, con su descripción.
- **Fecha y hora de inicio:** se ingresa la hora en que se desea que comience a ejecutarse el evento, también se debe ingresar el intervalo en días horas, minutos y segundos en el que se desea que se siga ejecutando el evento, por ej. Cada una hora, cada 10 minutos, (diario o Mensuales para las aplicaciones comerciales)etc.
- **Activo:** si este campo esta seleccionado, entonces el evento esta activo.
- **Errores de compilación :** Este campo se encuentra en todos los Forms y es el que al momento de presionar el botón compilar arroja los errores de compilación si es que los hubiera.

Código del Form Eventos

```
Name                          ON-CLEAR-DETAILS
    Class                         <Null>
    Trigger Text
      -- Begin default relation program section
      BEGIN
        Clear_All_Master_Details;
      END;
      -- End default relation program section
Name                          ON-CHECK-DELETE-MASTER
      Class                         <Null>
      Trigger Text
        -- Begin default relation declare section
        DECLARE
          Dummy_Define CHAR(1);
          -- Begin CONDICIONES_EVENTOS detail declare section
          CURSOR CONDICIONES_EVENTOS_cur IS
            SELECT 1 FROM CONDICIONES_EVENTOS
            WHERE EVENTO = :EVENTOS.EVENTO;
          -- End CONDICIONES_EVENTOS detail declare section
          -- Begin ACCIONES_EVENTOS detail declare section
          CURSOR ACCIONES_EVENTOS_cur IS
```

```
            SELECT 1 FROM ACCIONES_EVENTOS
            WHERE EVENTO = :EVENTOS.EVENTO;
          -- End ACCIONES_EVENTOS detail declare section
        -- End default relation declare section
        -- Begin default relation program section
BEGIN
          -- Begin CONDICIONES_EVENTOS detail program section
          OPEN CONDICIONES_EVENTOS_cur;
          FETCH CONDICIONES_EVENTOS_cur INTO Dummy_Define;
          IF ( CONDICIONES_EVENTOS_cur%found ) THEN
            Message('Cannot delete master record when matching detail records exist.');
            CLOSE CONDICIONES_EVENTOS_cur;
            RAISE Form_Trigger_Failure;
          END IF;
          CLOSE CONDICIONES_EVENTOS_cur;
          -- End CONDICIONES_EVENTOS detail program section
          -- Begin ACCIONES_EVENTOS detail program section
          OPEN ACCIONES_EVENTOS_cur;
          FETCH ACCIONES_EVENTOS_cur INTO Dummy_Define;
          IF ( ACCIONES_EVENTOS_cur%found ) THEN
            Message('Cannot delete master record when matching detail records exist.');
            CLOSE ACCIONES_EVENTOS_cur;
            RAISE Form_Trigger_Failure;
          END IF;
          CLOSE ACCIONES_EVENTOS_cur;
          -- End ACCIONES_EVENTOS detail program section
        END;
        -- End default relation program section
Name                          ON-POPULATE-DETAILS
      Class                          <Null>
      Trigger Text
        -- Begin default relation declare section
        DECLARE
          recstat     CHAR(20) := :System.record_status;
          startitm    CHAR(61) := :System.cursor_item;
          rel_id      Relation;
        -- End default relation declare section
        -- Begin default relation program section
        BEGIN
          IF ( recstat = 'NEW' or recstat = 'INSERT' ) THEN
            RETURN;
          END IF;
          -- Begin CONDICIONES_EVENTOS detail program section
          IF ( (:EVENTOS.EVENTO is not null) ) THEN
            rel_id := Find_Relation('EVENTOS.EVENTO_CONDICION');
            Query_Master_Details(rel_id, 'CONDICIONES_EVENTOS');
          END IF;
          -- End CONDICIONES_EVENTOS detail program section
```

```
-- Begin ACCIONES_EVENTOS detail program section
IF ( (:EVENTOS.EVENTO is not null) ) THEN
  rel_id := Find_Relation('EVENTOS.EVENTO_ACCION');
  Query_Master_Details(rel_id, 'ACCIONES_EVENTOS');
END IF;
-- End ACCIONES_EVENTOS detail program section
-- Begin USER_ERRORS detail program section
IF ( (:EVENTOS.EVENTO is not null) ) THEN
  rel_id := Find_Relation('EVENTOS.EVENTOS_ERRORES');
  Query_Master_Details(rel_id, 'USER_ERRORS');
END IF;
-- End USER_ERRORS detail program section
IF ( :System.cursor_item <> startitm ) THEN
   Go_Item(startitm);
   Check_Package_Failure;
END IF;
END;
End default relation program section

Name                    POST-CHANGE
Class                   <Null>
Trigger Text
  if :system.record_status in ('NEW','INSERT') THEN
     if existe_obj THEN
        msg1('Ya existe objeto '||:eventos.evento);
        raise form_trigger_failure;
     end if;
  end if;
  :eventos.status := get_status;
  if :eventos.status = 'VALID' THEN
     if :eventos.tipo = 'TEMPORAL' THEN
        :eventos.prox_ejec :=
to_char(pkg_sys.get_next_date(:eventos.job_no),'DD/MM/YYYY HH24:MI:SS');
     else
        :eventos.prox_ejec := null;
     end if;
  else
     :eventos.prox_ejec := null;
     :eventos.activo := 'N';
  end if;

Name                    WHEN-VALIDATE-ITEM
Class                   <Null>
Trigger Text
  if :eventos.tipo = 'TEMPORAL' THEN
     if :eventos.partida <= sysdate THEN
        msg1('La hora del evento debe ser en el futuro');
        raise form_trigger_failure;
```

```
            end if;
          end if;
Triggers
        Name                        WHEN-CHECKBOX-CHANGED
        Class                       <Null>
        Trigger Text
          if :eventos.status <> 'INVALID' THEN
            if :eventos.activo = 'S' THEN
                enable_evento;
            else
                disable_evento;
            end if;
          else
            :eventos.activo := 'N';
          end if;

      Triggers
        Name                        POST-CHANGE
        Class                       <Null>
        Trigger Text
          select descripcion
          into :condiciones_eventos.desc_cond
          from condiciones
          where condicion = :condiciones_eventos.condicion;
          EXCEPTION
          WHEN others THEN
              msg1('Ha ocurrido un error, buscando condicion P-CH, condicion');
              raise form_trigger_failure;

      Triggers
        Name                        POST-CHANGE
        Class                       <Null>
        Trigger Text
          select descripcion
          into :acciones_eventos.desc_accion
          from acciones
          where accion = :acciones_eventos.accion;
          EXCEPTION
          WHEN others THEN
              msg1('Ha ocurrido un error, buscando condicion P-CH, accion');
              raise form_trigger_failure;

Program Units
    CHECK_PACKAGE_FAILURE (Procedure Body)
      Procedure Check_Package_Failure IS
      BEGIN
        IF NOT ( Form_Success ) THEN
          RAISE Form_Trigger_Failure;
```

```
    END IF;
  END;
```

CLEAR_ALL_MASTER_DETAILS (Procedure Body)

```
  PROCEDURE Clear_All_Master_Details IS
    mastblk  CHAR(30);  -- Initial Master Block Cusing Coord
    coordop  CHAR(30);  -- Operation Causing the Coord
    trigblk  CHAR(30);  -- Cur Block On-Clear-Details Fires On
    startitm CHAR(61);  -- Item in which cursor started
    frmstat  CHAR(15);  -- Form Status
    curblk   CHAR(30);  -- Current Block
    currel   CHAR(30);  -- Current Relation
    curdtl   CHAR(30);  -- Current Detail Block

    FUNCTION First_Changed_Block_Below(Master CHAR)
    RETURN CHAR IS
      curblk CHAR(30);  -- Current Block
      currel CHAR(30);  -- Current Relation
      retblk CHAR(30);  -- Return Block
    BEGIN
      -- Initialize Local Vars
      curblk := Master;
      currel := Get_Block_Property(curblk,  FIRST_MASTER_RELATION);
      -- While there exists another relation for this block
      WHILE currel IS NOT NULL LOOP
        -- Get the name of the detail block
        curblk := Get_Relation_Property(currel, DETAIL_NAME);
        -- If this block has changes, return its name
        IF ( Get_Block_Property(curblk, STATUS) IN('CHANGED','INSERT') ) THEN
          RETURN curblk;
        ELSE
          -- No changes, recursively look for changed blocks below
          retblk := First_Changed_Block_Below(curblk);
          -- If some block below is changed, return its name
          IF retblk IS NOT NULL THEN
            RETURN retblk;
          ELSE
            -- Consider the next relation
            currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
          END IF;
        END IF;
      END LOOP;
      -- No changed blocks were found
      RETURN NULL;
    END First_Changed_Block_Below;

  BEGIN
    -- Init Local Vars
```

```
mastblk  := :System.Master_Block;
coordop  := :System.Coordination_Operation;
trigblk  := :System.Trigger_Block;
startitm := :System.Trigger_Item;
frmstat  := :System.Form_Status;
-- If the coordination operation is anything but CLEAR_RECORD or
-- SYNCHRONIZE_BLOCKS, then continue checking.
IF coordop NOT IN ('CLEAR_RECORD', 'SYNCHRONIZE_BLOCKS') THEN
  -- If we're processing the driving master block...
  IF mastblk = trigblk THEN
    -- If something in the form is changed, find the
    -- first changed block below the master
    IF frmstat = 'CHANGED' THEN
      curblk := First_Changed_Block_Below(mastblk);
      -- If we find a changed block below, go there
      -- and Ask to commit the changes.
      IF curblk IS NOT NULL THEN
        Go_Block(curblk);
        Check_Package_Failure;
        Clear_Block(ASK_COMMIT);
        -- If user cancels commit dialog, raise error
        IF NOT ( :System.Form_Status = 'QUERY'
                OR :System.Block_Status = 'NEW' ) THEN
          RAISE Form_Trigger_Failure;
        END IF;
      END IF;
    END IF;
  END IF;
END IF;
-- Clear all the detail blocks for this master without
-- any further asking to commit.
currel := Get_Block_Property(trigblk, FIRST_MASTER_RELATION);
WHILE currel IS NOT NULL LOOP
  curdtl := Get_Relation_Property(currel, DETAIL_NAME);
  IF Get_Block_Property(curdtl, STATUS) <> 'NEW'  THEN
    Go_Block(curdtl);
    Check_Package_Failure;
    Clear_Block(NO_VALIDATE);
    IF :System.Block_Status <> 'NEW' THEN
      RAISE Form_Trigger_Failure;
    END IF;
  END IF;
  currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
END LOOP;
-- Put cursor back where it started
IF :System.Cursor_Item <> startitm THEN
  Go_Item(startitm);
  Check_Package_Failure;
```

```
    END IF;

  EXCEPTION
    WHEN Form_Trigger_Failure THEN
      IF :System.Cursor_Item <> startitm THEN
        Go_Item(startitm);
      END IF;
      RAISE;

  END Clear_All_Master_Details;
```

COMMIT_MUDO (Procedure Body)

```
  PROCEDURE commit_mudo IS
  rlevel varchar2(30);
  BEGIN
    rlevel := :system.message_level;
    :system.message_level := '25';
     commit;
    :system.message_level := rlevel;
  END;
```

CREATE_EVENTO (Procedure Body)

```
  PROCEDURE create_evento IS
  BEGIN
    if :eventos.tipo = 'TABLA' THEN
        create_trigger;
    elsif :eventos.tipo = 'TEMPORAL' THEN
        set_item_property('EVENTOS.TABLA',REQUIRED,PROPERTY_FALSE);
        create_job;
    end if;
  END;
```

CREATE_JOB (Procedure Body)

```
  PROCEDURE create_job IS
  -- procedimiento que crea el job asociado al evento de tipo temporal
  v_intervalo number;
  BEGIN
    if :eventos.job_no is null THEN
        v_intervalo := :interval_dd + (:interval_hh/24) + (:interval_mm/(24*60)) +
(:interval_ss/(24*60*60));
       :eventos.job_no :=
pkg_sys.submit_job('pkg_sys.execute_eca('''||:eventos.evento||''');',:eventos.partida,'sysdate +
'||to_char(v_intervalo));
         set_item_property('EVENTOS.TABLA',REQUIRED,PROPERTY_FALSE);
        commit_mudo;
    else
       msg1('Job ya esta en proceso, verifique estado');
    end if;
```

```
    :eventos.activo := 'S';
    EXCEPTION
    WHEN others THEN
        msg1('Error al cerear job: '||:eventos.evento||': '||SQLERRM);
        raise form_trigger_failure;
  END;
```

CREATE_TRIGGER (Procedure Body)

```
  PROCEDURE create_trigger IS
  -- funcion que crea el trigger asociado al evento de tipo tabla
    v_cursor number;
    v_ret number;
    v_tiempo varchar2(20);
    v_eachrow varchar2(30) := null;
    v_new_old varchar2(2000) := null;
    v_trigger varchar2(2000) := null;
  BEGIN
  -- cada fila
    if :eventos.each_row = 'S' THEN
       v_eachrow := 'FOR EACH ROW';
       v_new_old := pkg_sys.get_new_old_str(:eventos.tabla);
    end if;
  -- cuerpo del trigger
    v_trigger := :eventos.triggering||' '||:eventos.sentencia||' ON '||:eventos.tabla||' '||
              v_eachrow||' '||chr(10)||
              'BEGIN '||chr(10)||
                 v_new_old||chr(10)||
               'pkg_sys.execute_eca('''||:eventos.evento||'''); '||chr(10)||
              'END;';
    pkg_sys.create_obj(:eventos.evento,v_trigger,'TRIGGER',null);
    :eventos.activo := 'S';
    EXCEPTION
    WHEN others THEN
        msg1('Error al crear trigger '||:eventos.evento||': '||SQLERRM);
        raise form_trigger_failure;
  END;
```

DELREC (Procedure Body)

```
  PROCEDURE delrec IS
  begin
    IF get_permiso_borrar THEN
      drop_evento;
      begin
        delete_record;
        commit_mudo;
        EXCEPTION
        WHEN others THEN
            msg1('Error al borrar geistro de evento: '||SQLERRM);
```

```
            raise form_trigger_failure;
        end;
        execute_query;
      END IF;
    end;
```

DISABLE_EVENTO (Procedure Body)

```
    PROCEDURE disable_evento IS
    BEGIN
      if :eventos.tipo = 'TABLA' THEN
        pkg_sys.disable_trigger(:eventos.evento);
        msg1('Proceso deshabilitado');
      else
        drop_job;
        msg1('Job deshabilitado');
      end if;
      e_query;
      EXCEPTION
      WHEN others THEN
          msg1('Error al deshabilitar evento: '||SQLERRM);
    END;
```

DROP_EVENTO (Procedure Body)

```
    PROCEDURE drop_evento IS
    BEGIN
      if :eventos.tipo = 'TABLA' then
         pkg_sys.drop_obj(:eventos.evento,'TRIGGER');
      elsif :eventos.tipo = 'TEMPORAL' then
         drop_job;
      end if;
      EXCEPTION
      WHEN others THEN
          msg1('Error al borrar evento: '||SQLERRM);
          raise form_trigger_failure;
    END;
```

DROP_JOB (Procedure Body)

```
    PROCEDURE drop_job IS
    -- procedimiento que borra el job asociado al evento de tipo temporal
    BEGIN
      if :eventos.job_no is not null THEN
        pkg_sys.drop_job(:eventos.job_no);
        :eventos.job_no := null;
        commit_mudo;
      else
        msg1('Error '||:eventos.evento||' no tiene asociado Job');
      end if;
      EXCEPTION
```

```
    WHEN others THEN
        msg1('Error al borrar job: '||:eventos.evento||': '||SQLERRM);
        raise form_trigger_failure;
  END;
```

ENABLE_EVENTO (Procedure Body)

```
  PROCEDURE enable_evento IS
  BEGIN
    if :eventos.tipo = 'TABLA' THEN
      pkg_sys.enable_trigger(:eventos.evento);
    else
      create_job;
    end if;
    EXCEPTION
    WHEN others THEN
        msg1('Error al habilitar evento: '||SQLERRM);
  END;
```

EXISTE_JOB (Function Body)

```
  function existe_job return boolean IS
  v_dummy varchar2(1);
  begin
    if :system.record_status in ('NEW','INSERT') THEN
      begin
        select 'x'
        into v_dummy
        from eventos
        where evento = :eventos.evento;
        return(TRUE);
        EXCEPTION
        WHEN no_data_found THEN
            return(FALSE);
        WHEN others THEN
            msg1('Error al buscar evento: '||:eventos.evento||': '||SQLERRM);
            raise form_trigger_failure;
      end;
    end if;
  end;
```

EXISTE_OBJ (Function Body)

```
  function existe_obj return boolean IS
  BEGIN
    if :eventos.tipo = 'TABLA' THEN
       return(pkg_sys.existe_obj(:eventos.evento));
    elsif :eventos.tipo = 'TEMPORAL' THEN
       return(existe_job);
    else
       msg1('Error: Tipo de Evento inexistente: '||:eventos.tipo);
```

```
      end if;
    END;
```

E_QUERY (Procedure Body)

```
    PROCEDURE e_query IS
    BEGIN
      SET_BLOCK_PROPERTY('EVENTOS', DEFAULT_WHERE, 'evento =
'''||:eventos.evento||''');
      execute_query;
      SET_BLOCK_PROPERTY('EVENTOS', DEFAULT_WHERE, '');
    END;
```

GET_PERMISO_BORRAR (Function Body)

```
    FUNCTION get_permiso_borrar RETURN boolean IS
    ret number;
    BEGIN
      if :eventos.evento is null THEN
        return(FALSE);
      end if;
      ret := show_alert('ALERTA_BORRADO');
      if ret = ALERT_BUTTON1 THEN
        return(TRUE);
      else
        return(FALSE);
      end if;
    END;
```

GET_STATUS (Function Body)

```
    FUNCTION get_status return varchar2 IS
    BEGIN
      if :eventos.tipo = 'TABLA' THEN
         return(nvl(pkg_sys.get_status(:eventos.evento,'TRIGGER'),'INVALID'));
      elsif :eventos.tipo = 'TEMPORAL' THEN
         if pkg_sys.get_next_date(:eventos.job_no) > sysdate THEN
           return('VALID');
         else
           return('INVALID');
         end if;
      else
         msg1('No existe tipo de Evento: '||:eventos.tipo);
         raise form_trigger_failure;
      end if;
    END;
```

K_COMMIT (Procedure Body)

```
    PROCEDURE k_commit IS
    BEGIN
      if :eventos.evento is not null THEN
```

```
        go_block('EVENTOS');
        commit;
        drop_evento;
        create_evento;
        e_query;
        :eventos.status := get_status;
        if nvl(:eventos.status,'INVALID') = 'INVALID' THEN
          :eventos.activo := 'N';
 --         commit_mudo;
        end if;
        msg1('El evento ha sido compilado');
    end if;
  END;
```

MSG (Procedure Body)

```
  PROCEDURE MSG(p_msg in varchar2) IS
  BEGIN
    message(p_msg,NO_ACKNOWLEDGE);
  END;
```

MSG1 (Procedure Body)

```
  PROCEDURE msg1(p_msg in varchar2) IS
  v_resp number;
  BEGIN
    Set_Alert_Property('UNA_VIA',alert_message_text,p_msg);
    v_resp := show_alert('UNA_VIA');
  END;
```

MSG2 (Function Body)

```
  FUNCTION msg2(p_msg in varchar2) return number IS
  v_resp number;
  BEGIN
    Set_Alert_Property('DOS_VIAS',alert_message_text,p_msg);
    v_resp := show_alert('DOS_VIAS');
    if v_resp = ALERT_BUTTON1 THEN
      return(1);
    else
      return(2);
    end if;
  END;
```

QUERY_MASTER_DETAILS (Procedure Body)

```
  PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
   oldmsg CHAR(2);  -- Old Message Level Setting
   reldef CHAR(5);  -- Relation Deferred Setting
  BEGIN
   --
   -- Initialize Local Variable(s)
```

```
      reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
      oldmsg := :System.Message_Level;
      -- If NOT Deferred, Goto detail and execute the query.
      IF reldef = 'FALSE' THEN
        Go_Block(detail);
        Check_Package_Failure;
        :System.Message_Level := '10';
        Execute_Query;
        :System.Message_Level := oldmsg;
      ELSE
        -- Relation is deferred, mark the detail block as un-coordinated
        Set_Block_Property(detail, COORDINATION_STATUS, NON_COORDINATED);
      END IF;

    EXCEPTION
       WHEN Form_Trigger_Failure THEN
         :System.Message_Level := oldmsg;
         RAISE;
    END Query_Master_Details;

   VER_CAMPOS (Procedure Body)
    PROCEDURE ver_campos IS
    BEGIN
      IF :eventos.tipo = 'TABLA' THEN
        set_item_property('EVENTOS.TABLA',ENABLED,PROPERTY_TRUE);
        set_item_property('EVENTOS.TABLA',UPDATEABLE,PROPERTY_TRUE);
        set_item_property('EVENTOS.TABLA',REQUIRED,PROPERTY_TRUE);
        set_item_property('EVENTOS.TRIGGERING',ENABLED,PROPERTY_TRUE);

set_item_property('EVENTOS.TRIGGERING',UPDATEABLE,PROPERTY_TRUE);
        set_item_property('EVENTOS.TRIGGERING',REQUIRED,PROPERTY_TRUE);

        set_item_property('EVENTOS.SENTENCIA',ENABLED,PROPERTY_TRUE);
        set_item_property('EVENTOS.SENTENCIA',UPDATEABLE,PROPERTY_TRUE);
        set_item_property('EVENTOS.SENTENCIA',REQUIRED,PROPERTY_TRUE);
        set_item_property('EVENTOS.EACH_ROW',ENABLED,PROPERTY_TRUE);
        set_item_property('EVENTOS.EACH_ROW',UPDATEABLE,PROPERTY_TRUE);

        set_item_property('EVENTOS.PARTIDA',ENABLED,PROPERTY_FALSE);
        set_item_property('EVENTOS.PARTIDA',REQUIRED,PROPERTY_FALSE);
        set_item_property('EVENTOS.PARTIDA',UPDATEABLE,PROPERTY_FALSE);

        set_item_property('EVENTOS.INTERVAL_DD',ENABLED,PROPERTY_FALSE);
        set_item_property('EVENTOS.INTERVAL_DD',REQUIRED,PROPERTY_FALSE);

set_item_property('EVENTOS.INTERVAL_DD',UPDATEABLE,PROPERTY_FALSE);

        set_item_property('EVENTOS.INTERVAL_HH',ENABLED,PROPERTY_FALSE);
```

```
        set_item_property('EVENTOS.INTERVAL_HH',REQUIRED,PROPERTY_FALSE);

set_item_property('EVENTOS.INTERVAL_HH',UPDATEABLE,PROPERTY_FALSE);

        set_item_property('EVENTOS.INTERVAL_MM',ENABLED,PROPERTY_FALSE);

set_item_property('EVENTOS.INTERVAL_MM',REQUIRED,PROPERTY_FALSE);

set_item_property('EVENTOS.INTERVAL_MM',UPDATEABLE,PROPERTY_FALSE);

        set_item_property('EVENTOS.INTERVAL_SS',ENABLED,PROPERTY_FALSE);
        set_item_property('EVENTOS.INTERVAL_SS',REQUIRED,PROPERTY_FALSE);

set_item_property('EVENTOS.INTERVAL_SS',UPDATEABLE,PROPERTY_FALSE);
      ELSE
        set_item_property('EVENTOS.TABLA',ENABLED,PROPERTY_FALSE);
        set_item_property('EVENTOS.TABLA',REQUIRED,PROPERTY_FALSE);
        set_item_property('EVENTOS.TABLA',UPDATEABLE,PROPERTY_FALSE);

        set_item_property('EVENTOS.TRIGGERING',ENABLED,PROPERTY_FALSE);
        set_item_property('EVENTOS.TRIGGERING',REQUIRED,PROPERTY_FALSE);

set_item_property('EVENTOS.TRIGGERING',UPDATEABLE,PROPERTY_FALSE);

        set_item_property('EVENTOS.SENTENCIA',ENABLED,PROPERTY_FALSE);
        set_item_property('EVENTOS.SENTENCIA',REQUIRED,PROPERTY_FALSE);
        set_item_property('EVENTOS.SENTENCIA',UPDATEABLE,PROPERTY_FALSE);

        set_item_property('EVENTOS.EACH_ROW',ENABLED,PROPERTY_FALSE);
        set_item_property('EVENTOS.EACH_ROW',UPDATEABLE,PROPERTY_FALSE);

        set_item_property('EVENTOS.PARTIDA',ENABLED,PROPERTY_TRUE);
        set_item_property('EVENTOS.PARTIDA',UPDATEABLE,PROPERTY_TRUE);
        set_item_property('EVENTOS.PARTIDA',REQUIRED,PROPERTY_TRUE);

        set_item_property('EVENTOS.INTERVAL_DD',ENABLED,PROPERTY_TRUE);

set_item_property('EVENTOS.INTERVAL_DD',UPDATEABLE,PROPERTY_TRUE);
        set_item_property('EVENTOS.INTERVAL_DD',REQUIRED,PROPERTY_TRUE);

        set_item_property('EVENTOS.INTERVAL_HH',ENABLED,PROPERTY_TRUE);

set_item_property('EVENTOS.INTERVAL_HH',UPDATEABLE,PROPERTY_TRUE);
        set_item_property('EVENTOS.INTERVAL_HH',REQUIRED,PROPERTY_TRUE);
        set_item_property('EVENTOS.INTERVAL_MM',ENABLED,PROPERTY_TRUE);

set_item_property('EVENTOS.INTERVAL_MM',UPDATEABLE,PROPERTY_TRUE);
        set_item_property('EVENTOS.INTERVAL_MM',REQUIRED,PROPERTY_TRUE);
```

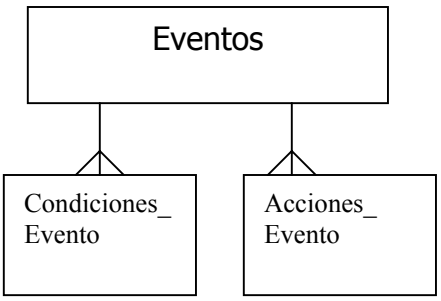
```
        set_item_property('EVENTOS.INTERVAL_SS',ENABLED,PROPERTY_TRUE);

set_item_property('EVENTOS.INTERVAL_SS',UPDATEABLE,PROPERTY_TRUE);
        set_item_property('EVENTOS.INTERVAL_SS',REQUIRED,PROPERTY_TRUE);
      END IF;
      END;
```

El resto de las pantallas y código de forms se encuentran en el Anexo B de Pantallas.

5.4.3 Reportes:

Reporte de Eventos

En base a la relación de los Eventos con las tablas Condiciones_ Evento y Acciones_ Evento, se basó la construcción del Form, para accesar y mantener un buen funcionamiento de los datos en la Base de datos, además la interfaz muestra claramente cuales son las condiciones y acciones asociadas al estado del Evento que también se muestra en Pantalla.

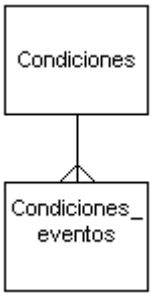


EVENTO	**DESCRIPCION**	**TIPO EVENTO**	**CONDICION**	**ACCION**
GESUSO	Genera Sanciones	Temporal	ERNC	SANCION
EBORRASA	Borrado Sanciones	Temporal	ESAN	BORRASAN
RECHRESE	Rechaza Reserva	Tabla	ESSANCIO	RECHSANC

Figura 5.8: Reporte de Eventos

Reporte de Condiciones

En las Condiciones, la relación con la tabla condiciones_eventos, almacena, esta ultima tabla, todas las condiciones posibles asociada a cada Evento, para esto se implementó una Interfaz, con todas las condiciones existentes y un checkbox, al lado del texto que contiene la Condición, con la finalidad que el administrador del Sistema habilite o deshabilite la Condición cuando se requiera.

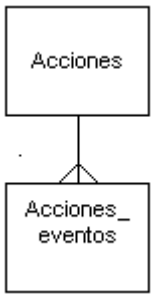


CONDICIONES	DESCRIPCION
ERNC	Existe reserva no Confirmada
ESAN	Existe Sanción
ESSANCIO	Usuario esta Sancionado

Figura 5.9: Reporte de Condiciones

Reporte de Acciones

En las acciones, la relación con la tabla acciones_eventos, almacena, esta ultima tabla, todas las acciones posibles asociada a cada Evento, para esto se implementó una Interfaz, con todas las acciones existentes y un checkbox, al lado del texto que contiene la accion, con la finalidad que el administrador del Sistema habilite o deshabilite la accion cuando se requiera.

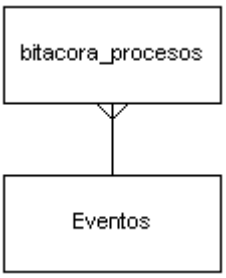


ACCION	DESCRIPCION
SANCIÓN	Genera Sanción
BORRASAN	Borra Sanciones
RECHASANC	Rechazar Sanción

Figura 5.10: Reporte de Acciones

Reporte de Bitácora de Procesos

La bitácora del Evento esta relacionada con el Evento, ya que almacena todo el historial del comportamiento de cada Evento que se ha programado en el Sistema, en este caso se construyo una Interfaz que básicamente contiene los atributos mas importantes de la Tabla Eventos, para que el administrador del Sistema pueda ver que ha sucedido o está sucediendo con la Base de datos en cada momento.



EVENTO	DESCRIPCIÓN	TIPO	USUARIO
GESUSO	Genera Sanciones	Temporal	Seminario
EBORRASA	Borrado Sanciones	Temporal	Seminario
RECHRESE	Rechaza Reserva	Tabla	Seminario

Figura 5.11: Reporte de Bitácora de Procesos

5.4.4 Package, módulo de programa almacenado en la base de datos: nivel Servidor coya funcioón es de proveer el motor de la BDA utilizando tecnología BD Relacional y actividad nativa ORACLE):

```
create or replace package pkg_sys as
g_valor_cond boolean;
-----------------------------------------------------------------------------------------
function submit_job(p_proc in varchar2, p_fecha in date, p_interval in varchar2) return
number;
-----------------------------------------------------------------------------------------
procedure execute_eca(p_evento in varchar2);
-----------------------------------------------------------------------------------------
procedure create_tab(p_table in varchar2);
-----------------------------------------------------------------------------------------
procedure create_pack_tab(p_table in varchar2);
-----------------------------------------------------------------------------------------
function get_new_old_str(p_table in varchar2) return varchar2;
-----------------------------------------------------------------------------------------
procedure execute_acc(p_acc in varchar2);
-----------------------------------------------------------------------------------------
procedure execute_str(p_str in varchar2);
-----------------------------------------------------------------------------------------
function get_cond(p_cond in varchar2) return boolean;
-----------------------------------------------------------------------------------------
function existe_obj(p_obj in varchar2) return boolean;
-----------------------------------------------------------------------------------------
function get_status(p_obj in varchar2, p_type in varchar2) return varchar2;
-----------------------------------------------------------------------------------------
function get_next_date(p_job_no in number) return date;
-----------------------------------------------------------------------------------------
procedure create_obj(p_obj in varchar2, p_cuerpo in varchar2,
                     p_obj_type in varchar2, p_return_type in varchar2);
-----------------------------------------------------------------------------------------
procedure drop_obj(p_obj in varchar2, p_type in varchar2);
-----------------------------------------------------------------------------------------
procedure drop_job(p_job_no in number);
-----------------------------------------------------------------------------------------
procedure enable_trigger(p_trigger in varchar2);
-----------------------------------------------------------------------------------------
procedure disable_trigger(p_trigger in varchar2);
-------------------------------------------------------------------------------
procedure active_condicion(p_condicion in varchar2);
-----------------------------------------------------------------------------------------
procedure active_accion(p_accion in varchar2);
```

```
-----------------------------------------------------------------------------------------
procedure desactive_condicion(p_condicion in varchar2);
-----------------------------------------------------------------------------------------
procedure desactive_accion(p_accion in varchar2);
-----------------------------------------------------------------------------------------
procedure ins_bitacora(p_evento in varchar2, p_status in varchar2, p_glosa in varchar2);
-----------------------------------------------------------------------------------------
function get_privilegio(p_evento in varchar2, p_user in varchar2) return boolean;
-----------------------------------------------------------------------------------------
end pkg_sys;
/
create or replace package body pkg_sys as
-----------------------------------------------------------------------------function submit_job(p_proc in
varchar2, p_fecha in date, p_interval in varchar2) return number is
--
-- procedimiento que submite un proceso oracle p_proc, para su ejecucion en la fecha p_fecha
-- con un intervalo de ejecucion p_intervalo medido en segundos
--
  v_job number;
begin
  dbms_job.submit(v_job,p_proc,p_fecha,p_interval);
  return(v_job);
  exception
  when others then
      raise_application_error(-20200,' pkg_sys.submit_job: '||SQLERRM);
end submit_job;
-------------------------------------------------------------------------------
function get_tipo_evento(p_evento in varchar2) return varchar2 is
--
-- funcion que retorna el tipo de evento (temporal o tabla)
--
salida varchar2(15);
BEGIN
  select tipo
  into salida
  from eventos
  where evento = p_evento;
  return(salida);
  EXCEPTION
  WHEN no_data_found THEN
      return(null);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.get_tipo_evento : evento: '||p_evento||':
'||SQLERRM);
--
END get_tipo_evento;
-------------------------------------------------------------------------------
procedure execute_eca(p_evento in varchar2) is
```

```
--
-- procedimiento que verifica las condiciones del evento p_evento y si todas son verdaderas
-- ejecuta las acciones asociadas al evento p_evento, solo si el usuario tiene privilegios
--
  cursor c_cond is
        select condicion
        from condiciones_eventos
        where evento = p_evento
        and activo = 'S'
        order by prioridad;
--
  cursor c_acc is
        select accion
        from acciones_eventos
        where evento = p_evento
        and activo = 'S'
        order by prioridad;
--
  v_cond boolean;
  v_num_condiciones number;
  v_num_condiciones_err number;
  v_num_no_cumple number;
  v_num_acciones number;
  v_num_acciones_err number;
  v_tipo_evento varchar2(15);
--
begin
--
  ins_bitacora(p_evento,'INICIO','Inicio ejecucion regla ECA');
--
-- verifica privilegios
--
  if get_privilegio(p_evento,user) then
--
--    tipo de evento
--
     v_tipo_evento := get_tipo_evento(p_evento);
--
--    condiciones
--
     v_num_condiciones := 0;
     v_num_no_cumple := 0;
     v_num_condiciones_err := 0;
--
     for i in c_cond loop
        begin
           v_cond := get_cond(i.condicion);
           if v_cond THEN
```

```
            v_num_condiciones := v_num_condiciones + 1;
          else
            v_num_no_cumple := v_num_no_cumple + 1;
            ins_bitacora(p_evento,'NO CUMPLE CONDICION',' Condicion: '||i.condicion);
          end if;
          EXCEPTION
          WHEN others THEN
              v_num_condiciones_err := v_num_no_cumple + 1;
              ins_bitacora(p_evento,'ERROR','Error en condicion: '||i.condicion||': '||SQLERRM);
        end;
    end loop;
--
    if v_num_no_cumple = 0 and v_num_condiciones_err = 0 THEN
      if v_num_condiciones = 0 THEN
        ins_bitacora(p_evento,'EN PROCESO','Evento sin condiciones');
      else
        ins_bitacora(p_evento,'EN PROCESO','Cumple: '||v_num_condiciones||', todas las
condiciones');
      end if;
    else
      if v_num_condiciones_err = 0 THEN
        if v_tipo_evento = 'TABLA' THEN
          ins_bitacora(p_evento,'FIN DE PROCESO','No cumple: '||v_num_no_cumple||'
condiciones');
          raise_application_error(-20200,'No cumple: '||v_num_no_cumple||' condiciones');
        else
          ins_bitacora(p_evento,'FIN DE PROCESO','No cumple: '||v_num_no_cumple||'
condiciones');
          return;
        end if;
      else
        if v_num_no_cumple = 0 THEN
          if v_tipo_evento = 'TABLA' THEN
            ins_bitacora(p_evento,'FIN CON ERROR',v_num_condiciones_err||' condiciones
(todas) erroneas');
            raise_application_error(-20200,'ERROR: '||v_num_condiciones_err||' condiciones
(todas) erroneas');
          else
            ins_bitacora(p_evento,'FIN CON ERROR',v_num_condiciones_err||' condiciones
(todas) erroneas');
            return;
          end if;
        else
          if v_tipo_evento = 'TABLA' THEN
            ins_bitacora(p_evento,'FIN CON ERROR','No se cumplen '||v_num_no_cumple||
                        'condiciones y '||v_num_condiciones_err||' erroneas');
            raise_application_error(-20200,'ERROR: No se cumplen '||v_num_no_cumple||
                        'condiciones y '||v_num_condiciones_err||' erroneas');
```

```
            else
              ins_bitacora(p_evento,'FIN CON ERROR','No se cumplen '||v_num_no_cumple||
                           'condiciones y '||v_num_condiciones_err||' erroneas');
              return;
            end if;
          end if;
        end if;
      end if;
--
--    acciones
--
      v_num_acciones := 0;
      v_num_acciones_err := 0;
      for i in c_acc loop
         begin
           execute_acc(i.accion);
           v_num_acciones := v_num_acciones + 1;
           Exception
           WHEN others THEN
               v_num_acciones_err := v_num_acciones_err + 1;
               ins_bitacora(p_evento,'ERROR','Error en accion: '||i.accion||': '||SQLERRM);
         end;
      end loop;
      if v_num_acciones_err = 0 THEN
        if v_num_acciones = 0 THEN
          ins_bitacora(p_evento,'FIN DE PROCESO','Evento sin acciones');
        else
          ins_bitacora(p_evento,'FIN DE PROCESO','Se procesaron: '||v_num_acciones||', todas
las acciones');
          return;
        end if;
      else
        ins_bitacora(p_evento,'FIN CON ERROR',v_num_acciones_err||' acciones con error');
        return;
      end if;
    else
      ins_bitacora(p_evento,'ERROR','Usuario: '||user||' no tiene privilegios para ejecutar evento:
'||p_evento);
    end if;
    EXCEPTION
    WHEN others THEN
        if v_tipo_evento = 'TABLA' THEN
          raise_application_error(-20200,'ERROR:'||substr(SQLERRM,1,2000));
        else
          ins_bitacora(p_evento,'ERROR',substr(SQLERRM,1,2000));
        end if;
--
end execute_eca;
```

```
-----------------------------------------------------------------------------------------
procedure create_tab(p_table in varchar2) is
--
-- procedimiento que crea una tabla correspondiente a la tabla p_table
--
  v_cuerpo varchar2(8000) := null;
begin
  for i in (select orden, columna, tipo, tamano, nula
            from columnas_tablas
            where tabla = p_table
            order by orden) loop
--
      begin
        select v_cuerpo||decode(i.orden,1,null,',')||i.columna||' '||i.tipo||
               decode(i.tipo,'VARCHAR2','('||i.tamano||') ',' ')||
               decode(i.nula,'N','NOT NULL',null)||chr(10)
        into v_cuerpo
        from dual;
        Exception
        WHEN others THEN
            raise_application_error(-20201,' pkg_sys.create_table: '||SQLERRM);
      end;
  end loop;
  create_obj(p_table,v_cuerpo,'TABLE',null);
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20202,' pkg_sys.create_tab: '||SQLERRM);
--
end create_tab;
-----------------------------------------------------------------------------------------
procedure create_pack_tab(p_table in varchar2) is
--
-- procedimiento que crea un package asociado a la tabla p_table con variables
-- correspondientes a las columnas de la tabla
--
  v_pack varchar2(8000) := null;
begin
  for i in (select column_name, data_type, data_length
            from user_tab_columns
            where table_name = p_table) loop
--
      begin
        select v_pack||'new_'||i.column_name||' '||i.data_type||
               decode(i.data_type,'VARCHAR2','('||i.data_length||'); ','; ')||chr(10)||
               'old_'||i.column_name||' '||i.data_type||
               decode(i.data_type,'VARCHAR2','('||i.data_length||'); ','; ')||chr(10)
        into v_pack
        from dual;
```

```
        Exception
        WHEN others THEN
            raise_application_error(-20201,' pkg_sys.create_pack_tab: '||SQLERRM);
      end;
   end loop;
   create_obj('g_'||p_table,v_pack,'PACKAGE',null);
   EXCEPTION
   WHEN others THEN
       raise_application_error(-20202,' pkg_sys.create_pack_tab: '||SQLERRM);
--
end create_pack_tab;
-----------------------------------------------------------------------------------------
function get_new_old_str(p_table in varchar2) return varchar2 is
--
-- funcion que retorna un string con las asignaciones de las variables new y old
-- correspondientes a las columnas de la tabla
--
   v_str varchar2(8000) := null;
begin
   for i in (select column_name, data_type, data_length
            from user_tab_columns
            where table_name = p_table) loop
--
      begin
        select v_str||'g_'||p_table||'.'||'new_'||i.column_name||' := :new.'||i.column_name||'; '||
                    'g_'||p_table||'.'||'old_'||i.column_name||' := :old.'||i.column_name||'; '
        into v_str
        from dual;
        Exception
        WHEN others THEN
            raise_application_error(-20201,' pkg_sys.get_new_old_str: '||SQLERRM);
      end;
   end loop;
   return(v_str);
--
end get_new_old_str;
-----------------------------------------------------------------------------------------
procedure execute_acc(p_acc in varchar2) is
--
-- procedimiento que ejecuta una accion p_acc.
--
   v_tex varchar2(200);
BEGIN
   v_tex := 'begin '||
              p_acc||'; '||
           'end;';
--
   execute_str(v_tex);
```

```
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.execute_acc: p_acc: '||p_acc||': '||SQLERRM);
--
END execute_acc;
-----------------------------------------------------------------------------------------
procedure execute_str(p_str in varchar2) is
--
-- ejecuta una instruccion ddl p_ddl
--
  v_cursor number;
  v_ret number;
--
BEGIN
--
  v_cursor := dbms_sql.open_cursor;
  DBMS_SQL.PARSE(v_cursor,p_str,2);
  v_ret := DBMS_SQL.EXECUTE(v_cursor);
  DBMS_SQL.CLOSE_CURSOR(v_cursor);
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.execute_str: '||SQLERRM);
--
END execute_str;
-----------------------------------------------------------------------------------------
function get_cond(p_cond in varchar2) return boolean is
--
-- funcion que retorna el resultado de la evaluacion de la condicion p_cond
--
  v_cursor number;
  v_tex varchar2(200);
BEGIN
  v_tex := 'begin '||
             'pkg_sys.g_valor_cond := '||p_cond||'; '||
         'end;';
--
  execute_str(v_tex);
  return(g_valor_cond);
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.get_cond: p_cond: '||p_cond||': '||SQLERRM);
--
END get_cond;
-----------------------------------------------------------------------------------------
function existe_obj(p_obj in varchar2) return boolean is
--
-- funcion que retorna TRUE si el objeto existe en la BD y FALSE en caso contrario
--
```

```
v_dummy varchar2(1);
BEGIN
  select 'x'
  into v_dummy
  from user_objects
  where object_name = p_obj
  and rownum = 1;
  return(TRUE);
  EXCEPTION
  WHEN no_data_found THEN
      return(FALSE);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.existe_obj: objeto: '||p_obj||': '||SQLERRM);
--
END existe_obj;
-----------------------------------------------------------------------------------------
function get_status(p_obj in varchar2, p_type in varchar2) return varchar2 is
--
-- funcion que retorna el STATUS del objeto
--
salida varchar2(30);
BEGIN
  select status
  into salida
  from user_objects
  where object_name = p_obj
  and object_type = p_type;
  return(salida);
  EXCEPTION
  WHEN no_data_found THEN
      return(null);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.get_status: objeto: '||p_obj||', tipo: '||p_type||':
'||SQLERRM);
--
END get_status;
-----------------------------------------------------------------------------------------
function get_next_date(p_job_no in number) return date is
--
-- funcion que retorna la proxima ejecucion del job
--
salida date;
BEGIN
  select next_date
  into salida
  from user_jobs
  where job = p_job_no;
  return(salida);
```

```
  EXCEPTION
  WHEN no_data_found THEN
      return(null);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.get_next_date: job: '||p_job_no||': '||SQLERRM);
--
END get_next_date;
-----------------------------------------------------------------------------------------
procedure create_obj(p_obj in varchar2, p_cuerpo in varchar2,
                 p_obj_type in varchar2, p_return_type in varchar2) is
v_return_type varchar2(15);
BEGIN
  if p_obj_type = 'PROCEDURE' THEN
    execute_str('create or replace procedure '||p_obj||' as '||
                 'BEGIN '||
                   p_cuerpo||' '||
                   'EXCEPTION '||
                   'WHEN others THEN '||
                       'raise_application_error(-20200," Error en procedure: '||p_obj||':
"||SQLERRM); '||
                 'END;');
  elsif p_obj_type = 'FUNCTION' THEN
    begin
      select decode(p_return_type,'VARCHAR2','varchar2(200)',p_return_type)
      into v_return_type
      from dual;
      execute_str('create or replace function '||p_obj||' return '||p_return_type||' as '||chr(10)||
                p_obj||' '||v_return_type||'; '||chr(10)||
                'BEGIN '||
                   p_cuerpo||' '||
                  'return('||p_obj||'); '||chr(10)||
                  'EXCEPTION '||
                  'WHEN others THEN '||
                      'raise_application_error(-20200," Error en function: '||p_obj||':
"||SQLERRM); '||
                'END;');
   end;
  elsif p_obj_type = 'PACKAGE' THEN
    execute_str('create or replace package '||p_obj||' as '||chr(10)||p_cuerpo||' END;');
  elsif p_obj_type = 'TABLE' THEN
    drop_obj(p_obj,'TABLE');
    execute_str('create table '||p_obj||' ('||chr(10)||p_cuerpo||')');
  elsif p_obj_type = 'TRIGGER' THEN
    execute_str('CREATE OR REPLACE TRIGGER '||p_obj||' '||chr(10)||p_cuerpo);
  else
    raise_application_error(-20200,' No existe tipo de objeto: '||p_obj_type);
  end if;
  EXCEPTION
```

```
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.create_obj: '||p_obj_type||': '||p_obj||':
'||SQLERRM);
end create_obj;
-----------------------------------------------------------------------------------------
procedure drop_obj(p_obj in varchar2, p_type in varchar2) is
--
-- procedimiento que borra el objeto p_obj de tipo p_type
--
begin
   if existe_obj(p_obj) THEN
      execute_str('drop '||p_type||' '||p_obj);
   end if;
   EXCEPTION
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.drop_obj: '||p_obj||': '||SQLERRM);
--
end drop_obj;
-----------------------------------------------------------------------------------------
function existe_job(p_job_no in number) return boolean is
--
-- funcion que retorna TRUE si el job existe en la BD y FALSE en caso contrario
--
v_dummy varchar2(1);
BEGIN
   select 'x'
   into v_dummy
   from user_jobs
   where job = p_job_no;
   return(TRUE);
   EXCEPTION
   WHEN no_data_found THEN
       return(FALSE);
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.existe_job: job: '||to_char(p_job_no)||':
'||SQLERRM);
--
END existe_job;
-----------------------------------------------------------------------------------------
procedure drop_job(p_job_no in number) is
--
-- procedimiento que borra el job p_job_no
--
begin
   if existe_job(p_job_no) THEN
      dbms_job.remove(p_job_no);
   end if;
   EXCEPTION
```

```
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.drop_obj: '||to_char(p_job_no)||': '||SQLERRM);
--
end drop_job;
-----------------------------------------------------------------------------------------
procedure enable_trigger(p_trigger in varchar2) is
BEGIN
   if get_status(p_trigger,'TRIGGER') = 'VALID' THEN
      execute_str('alter trigger '||p_trigger||' enable');
   end if;
   EXCEPTION
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.enable_trigger: '||p_trigger||': '||SQLERRM);
end enable_trigger;
-----------------------------------------------------------------------------------------
procedure disable_trigger(p_trigger in varchar2) is
BEGIN
   if get_status(p_trigger,'TRIGGER') = 'VALID' THEN
      execute_str('alter trigger '||p_trigger||' disable');
   end if;
   EXCEPTION
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.disable_trigger: '||p_trigger||': '||SQLERRM);
end disable_trigger;
-----------------------------------------------------------------------------------------
procedure active_condicion(p_condicion in varchar2) is
BEGIN
   update condiciones_eventos
   set activo = 'S'
   where condicion = p_condicion;
   EXCEPTION
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.active_condicion: '||p_condicion||':
'||SQLERRM);
end active_condicion;
-----------------------------------------------------------------------------------------
procedure active_accion(p_accion in varchar2) is
BEGIN
   update acciones_eventos
   set activo = 'S'
   where accion = p_accion;
   EXCEPTION
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.active_accion: '||p_accion||': '||SQLERRM);
end active_accion;
-----------------------------------------------------------------------------------------
procedure desactive_condicion(p_condicion in varchar2) is
BEGIN
```

```
   update condiciones_eventos
   set activo = 'N'
   where condicion = p_condicion;
   EXCEPTION
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.desactive_condicion: '||p_condicion||':
'||SQLERRM);
end desactive_condicion;
-----------------------------------------------------------------------------------------
procedure desactive_accion(p_accion in varchar2) is
BEGIN
   update acciones_eventos
   set activo = 'N'
   where accion = p_accion;
   EXCEPTION
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.desactive_accion: '||p_accion||': '||SQLERRM);
end desactive_accion;
-----------------------------------------------------------------------------------------
procedure ins_bitacora(p_evento in varchar2,p_status in varchar2,p_glosa in varchar2) is
begin
   insert into bitacora_procesos
values(p_evento,sec_proc.nextval,sysdate,user,p_status,p_glosa);
end ins_bitacora;
-----------------------------------------------------------------------------------------
function get_privilegio(p_evento in varchar2, p_user in varchar2) return boolean is
--
-- funcion que retorna TRUE si el evento y el usuario se encuentra en la tabla de privilegios
-- y FALSE en caso contrario
--
v_dummy varchar2(1);
BEGIN
   select 'x'
   into v_dummy
   from privilegios
   where evento = p_evento
   and usuario = p_user;
   return(TRUE);
   EXCEPTION
   WHEN no_data_found THEN
       return(FALSE);
   WHEN others THEN
       raise_application_error(-20200,' pkg_sys.get_privilegio: evento: '||p_evento||' usuario
'||p_user||SQLERRM);
--
end get_privilegio;
-----------------------------------------------------------------------------------------
end pkg_sys;
```

6. Pruebas

6.1 Implementacion de las Pruebas

Para realizar las pruebas, es necesario crear 2 tablas: Reservas y Sanciones, tal como se muestra en la Figura 6.1:

Tabla: Reservas

Campos:
Pista number
Fecha date
Rut number
Confirmada varchar 2(1)

Tabla: Sanciones

Campos:
rut number
Fecha date

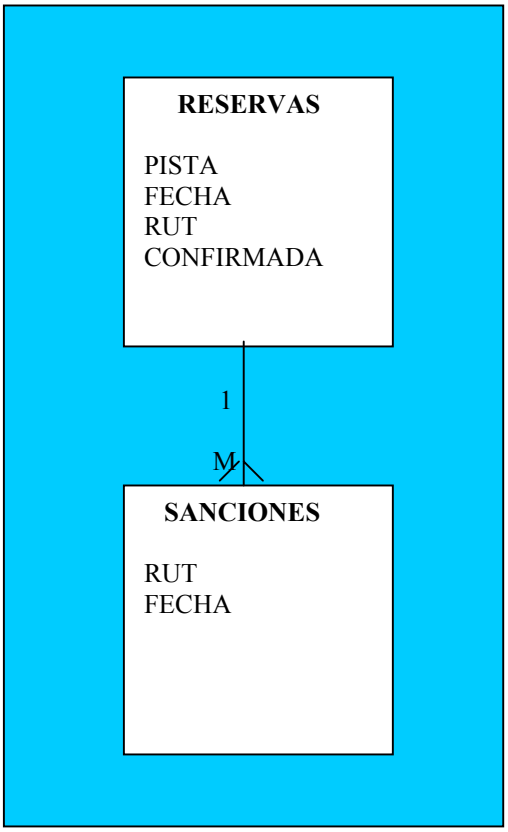


Figura 6.1 Tablas de Pruebas

Reglas del Negocio, expresadas como condiciones ECA: (Evento, Condición, Acción)

1. Cada día a las 24 hrs. Si existe reserva no confirmada generar una sanción:

E: Cada día a las 24 hrs.

C: Existe reserva no confirmada

A: Generar una sanción

Implementación:

E: GESUSO Genera Sanciones Temporal

C: Condición: ERNC

Descripción: EXISTE RESERVA NO CONFIRMADA

Expresión: RNC = 1

A: Acción: SANCION

Descripción: GENERA SANCION SISTEMA DE RESERVAS

Expresión:

```
FOR I IN (SELECT RUT, FECHA
                FROM RESERVAS
                WHERE NVL(CONFIRMADA,'N') = 'N'
                      AND TRUNC(FECHA)=TRUNC(SYSDATE))
                 LOOP
                INSERT INTO SANCIONES VALUES(I.RUT,I.FECHA+1);
PKG_SYS.INS_BITACORA('GESUSO','GESUSO','INSERTANDO SANCION
RUT='||I.RUT);
END LOOP;
```

La Figura 6.2 nos muestra a continuación la implementacion del ejemplo dado anteriormente:

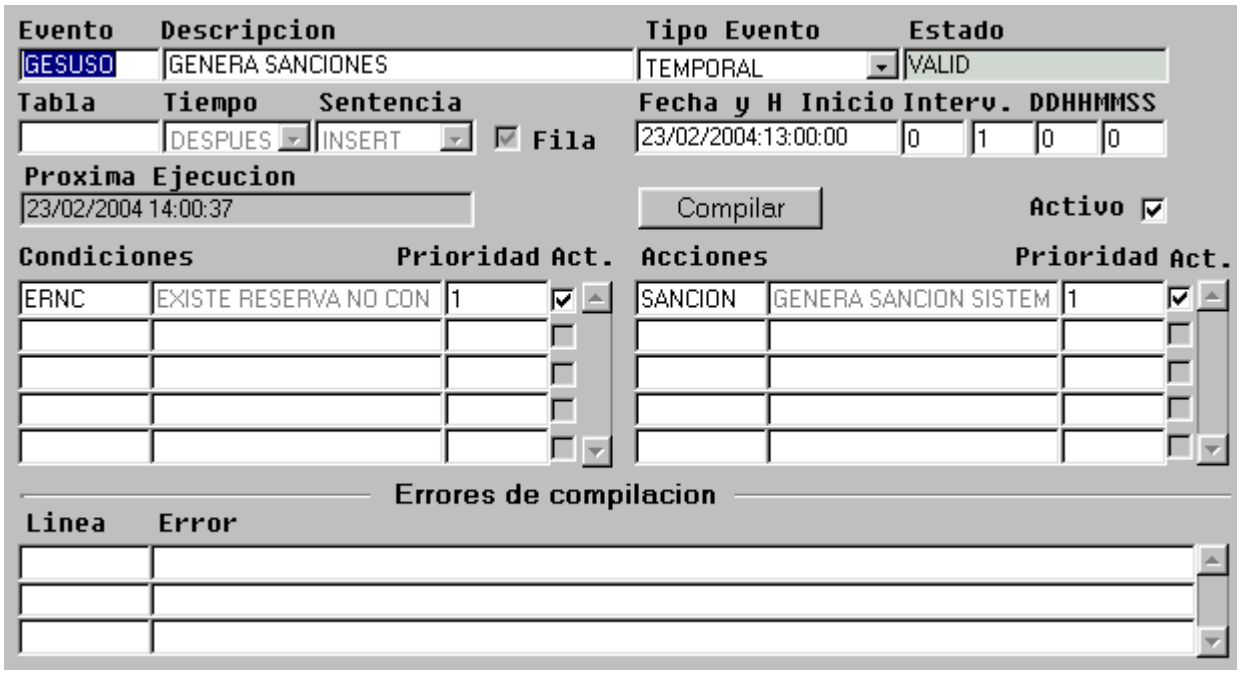


Figura 6.2: Pantalla Evento Genera Sanciones

2. Cada día a las 24 hrs. Si existe una sanción que termina en la fecha se debe borrar:

E: Cada día a las 24 hrs.

C: Existe sanción que termina en la fecha

A: Eliminar sanción

Implementación:

E: BORRASANC Borrado de Sanciones Temporal

C: Condición: ESAN

Descripción: EXISTE SANCION

Expresión: ESANCION = 1

A: Acción: BORRASANC

Descripción: BORRA SANCIONES

Expresión:

FOR I IN (SELECT RUT, ROWID RWD FROM SANCIONES WHERE TRUNC(FECHA)=TRUNC(SYSDATE))
LOOP

Evento: EBORRASA
Descripcion: BORRADO SANCIONES
Tipo Evento: TEMPORAL
Estado: VALID
Tabla:
Tiempo: DESPUES
Sentencia: INSERT
Fila
Fecha y H Inicio: 20/02/2004:19:40:00
Interv.: 0
DDHHMMSS: 1 0 0
Proxima Ejecucion: 24/02/2004 18:47:37
Compilar
Activo

Condiciones		Prioridad	Act.
ESAN	EXISTE SANCION	1	✓

Acciones		Prioridad	Act.
BORRASAN	BORRA SANCIONES	1	✓

Errores de compilacion

Linea	Error

Figura 6.3: Pantalla Evento Borrado de Sanciones

3. Al realizar una reserva, si el usuario esta sancionado, rechazar la reserva:

E: Al realizar la reserva

C: Usuario sancionado

A: Rechazar la reserva

Implementación:

E: RECHARESERV Rechazo de Reservas Tabla (Reservas)

C: Condición: ESSANCION

Descripción: USUARIO ESTA SANCIONADO

Expresión: SANCIONADO = 1

A: Acción: RECHSANC

Descripción: RECHAZAR SANCION

Expresión:

```
BEGIN
        RAISE_APLICCATION_ERROR(-20200,'USUARIO TIENE SANCION
PENDIENTE');
END;
```

A continuación se muestra la Figura 6.4, la que detalla como es ingresado en pantalla el Evento Rechaza Reservas:

Evento: RECHRESE
Descripcion: RECHAZA RESERVAS
Tipo Evento: TABLA
Estado: VALID
Tabla: RESERVAS
Tiempo: ANTES
Sentencia: INSERT
Fila: ☑
Fecha y H Inicio: 11/02/2004:00:00:00
Interv. DDHHMMSS: 0 0 0 0
Proxima Ejecucion:
Compilar
Activo: ☑

Condiciones		Prioridad	Act.
ESSANCIO	USUARIO ESTA SANCIONA	1	☑
			☐
			☐
			☐
			☐

Acciones		Prioridad	Act.
RECHSANC	RECHAZAR SANCION	1	☑
			☐
			☐
			☐
			☐

Errores de compilacion

Linea	Error

Figura 6.4: Pantalla Evento Rechaza Reservas

Form de VARIABLES: se ingresan las variables utilizadas en los otros Forms.

Variable: **RNC**

Descripción. RETORNA 0 SI EXISTE RESERVA NO CONFIRMADA

Tipo: NUMBER

Expresión:

```
BEGIN
SELECT 1
INTO RNC
FROM DUAL
WHERE EXISTS ( SELECT 'X'
                FROM RESERVAS
                WHERE NVL(CONFIRMADA,'N' )='N'
                AND TRUNC(FECHA)=TRUNC(SYSDATE));

                EXCEPTION
                        WHEN NO_DATA_FOUND THEN
                        RETURN 0;
END;
```

Variable: **ESANCION**

Descripción: RETORNA 1 SI EXISTE SANCION ACTUAL

Tipo: NUMBER

Expresión:

```
BEGIN
SELECT 1
INTO ESANCION
FROM DUAL
WHERE EXISTS (SELECT 'X'
                FROM SANCIONES
                WHERE TRUNC(FECHA)=TRUNC(SYSDATE));
                EXCEPTION
                        WHEN NO_DATA_FOUND THEN
                        RETURN 0;
END;
```

Variable: **SANCIONADO**

Descripción: USUARIO SANCIONADO

Tipo: NUMBER

Expresión:
```
SELECT 1
INTO SANCIONADO
FROM DUAL
WHERE  EXISTS (
SELECT 'X'
FROM SANCIONES
WHERE RUT = G_SANCIONES.NEW_RUT)
UNION
SELECT 0
FROM DUAL
WHERE NOT EXISTS(
SELECT 'X'
FROM SANCIONES
WHERE  RUT = G_SANCIONES.NEW_RUT);
```

Form de PRUEBAS: a través de este se ingresan los datos para realizar las pruebas, como se ilustra en la Figura 6.5:

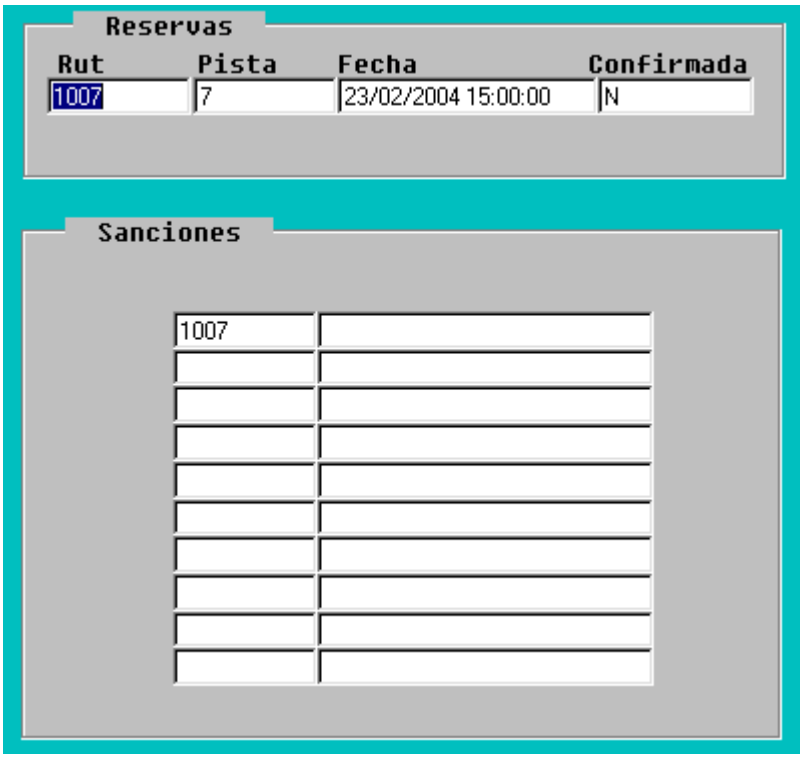


Figura 6.5: Pantalla de Pruebas

Al no ser confirmada la reserva, la **Base de Datos Activa** esta programada para ejecutarse a las 12:11 P.M en el evento genera sanciones y así el usuario estará sancionado e inhabilitado para realizar otra reserva, como se ilustra en la Figura 6.6:

Figura 6.6: Pantalla Evento Genera Sanciones

Y en la bitácora, la **Base de Datos Activa** ha dejado al usuario del RUT **1007** sancionado, como se ilustra en la Figura 6.7:

Bitacora de Procesos de Eventos

Evento	Descripción	Tipo	Estado
GESUSO	GENERA SANCIONES	TEMPORAL	VALID

Estado proceso	Fecha	Glosa	Usuario
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1010	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1009	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1008	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1007	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1006	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1005	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1004	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1003	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1002	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1001	SEMINARIO

Figura 6.7: Pantalla Bitácora de Procesos

Al tratar entonces el mismo de usuario de pedir una nueva reserva. La **Base de Datos Activa** no lo dejara ya que el usuario esta con sanción pendiente, enviando un mensaje de error, como se ilustra en la Figura 6.8:

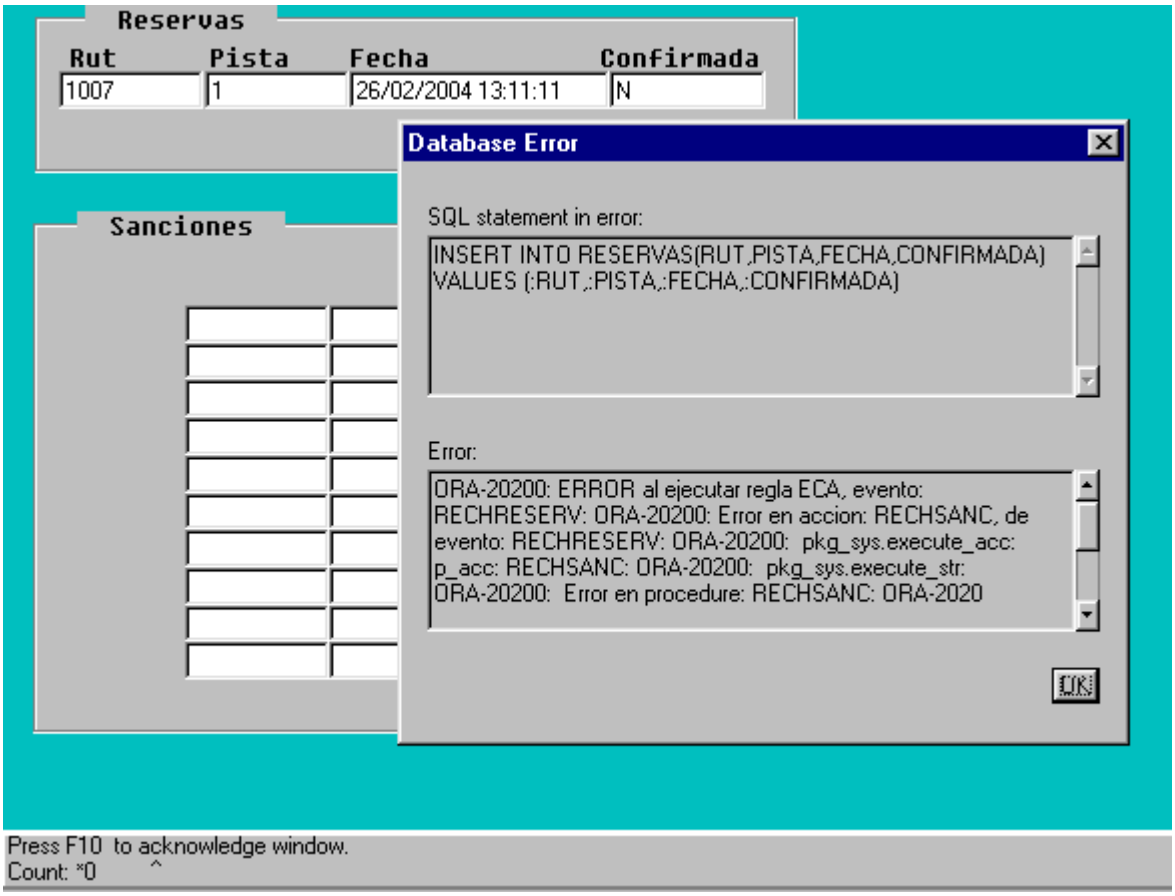


Figura 6.8: Mensaje de error para los usuarios con sanción

La Base de Datos Activa se ha programado ahora para borrar la sanción a las 12:20 P.M, como se ilustra en la Figura 6.9:

Reglas ECA

Evento: EBORRASA
Descripcion: BORRADO SANCIONES
Tipo Evento: TEMPORAL
Estado: VALID
Tabla:
Tiempo: DESPUES
Sentencia: INSERT
Fila
Fecha y H Inicio: 26/02/2004:12:20:00
Interv. DDHHMMSS: 0 1 0 0
Proxima Ejecucion: 26/02/2004 13:20:33
Compilar
Activo

Condiciones		Prioridad	Act.	Acciones		Prioridad	Act.
ESAN	EXISTE SANCION	1	☑	BORRASAN	BORRA SANCIONES	1	☑

Errores de compilacion

Linea	Error

Figura 6.9: Formato del Evento Borrado de Sanciones

Quedando ahora en la bitácora el usuario sin sanción y habilitado para poder realizar otra reserva, la Figura 6.10 muestra que el Evento Borrado de Sanciones se ha ejecutado satisfactoriamente.

Bitacora de Procesos de Eventos

Evento	Descripción	Tipo	Estado
EBORRASANC	BORRADO SANCIONES	TEMPORAL	VALID

Estado proceso	Fecha	Glosa	Usuario
EBORRASANC	24/02/2004 10:41:56	BORRANDO SANCION RUT=1011	SEMINARIO
EBORRASANC	24/02/2004 10:41:56	BORRANDO SANCION RUT=1010	SEMINARIO
EBORRASANC	24/02/2004 10:41:56	BORRANDO SANCION RUT=1009	SEMINARIO
EBORRASANC	24/02/2004 10:41:55	BORRANDO SANCION RUT=1008	SEMINARIO
EBORRASANC	24/02/2004 10:41:55	BORRANDO SANCION RUT=1007	SEMINARIO
EBORRASANC	24/02/2004 10:41:55	BORRANDO SANCION RUT=1006	SEMINARIO
EBORRASANC	24/02/2004 10:41:55	BORRANDO SANCION RUT=1005	SEMINARIO
EBORRASANC	24/02/2004 10:41:55	BORRANDO SANCION RUT=1004	SEMINARIO
EBORRASANC	24/02/2004 10:41:55	BORRANDO SANCION RUT=1003	SEMINARIO
EBORRASANC	24/02/2004 10:41:55	BORRANDO SANCION RUT=1002	SEMINARIO

Figura 6.10: Ejecución del Evento Borrado de Sanciones

6.2. - ANALISIS DEL COMPORTAMIENTO DE LA INTEGRIDAD DE LOS DATOS DE LA BDA

6.2.1. - ANALISIS DE LA INTEGRIDAD

Como la **Integridad de los Datos** se refiere a la corrección y validez de los datos, verificaremos que los datos sean los correctos y mediremos el tiempo en que la Base de Datos Activa se demora en garantizar la integridad de los datos.

Definamos la medida de la inconsistencia en el tiempo de un registro de la BDA, como la diferencia o cantidad de tiempo en que se encuentra definido un evento y el tiempo en que realmente tiene efecto este evento sobre los registros de la base de datos y quedan consistentes:

Se pueden definir por ejemplo el tiempo en que se define un evento (i) y Rj la hora en que el registro (j) se torna consistente debido a la actividad de la base de datos.

Entonces la cantidad de tiempo en que se encuentra inconsistente un registro j respecto del evento i es:

Tij = Rj – Ei Con Ri > Ej (puesto que cada registro se torna consistente en un tiempo posterior al que se define el evento) y Ei es fijo para cada evento (Esto para los eventos de tipo temporal en el caso de los eventos de tipo tabla, el tiempo de respuesta es casi instantáneo).

Entonces el tiempo promedio (definido por Tip) en que se encuentra inconsistente la base de datos respecto del evento i es:

$$\mathbf{Tip} = \frac{\text{Suma}(Tij)}{N}$$

Donde N es el número de registros que se encuentran inconsistentes (que están en el alcance o ámbito del evento) y deben cambiar su estado debido a la actividad de la base de datos.

Con las pruebas realizadas se generaron 1000 registros para los cuales se midió el tiempo que se demoró en ser afectado por el evento, la moda del tiempo inconsistente es 41 segundos para 757 registros.

El promedio es Tp = Suma (ti) 261,146 (seg)

---------- = ---------- = 0,261 Segundos por registro

N 1000

Entonces con esto el promedio Tip = 0,261

El error total es de +- 0,1 seg.

Luego el Promedio es 0,261 +- 0,1 seg.

Con las pruebas realizadas se generaron 1000 registros para los cuales se midió el tiempo que se demoró en ser afectado por el evento, la moda del tiempo inconsistente es 41 segundos para 757 registros.

El promedio es Tp = Suma (ti) 261,146 (seg)

---------- = ---------- = 0,261 Segundos por registro

N 1000

Entonces con esto el promedio Tip = 0,261

El error total es de +- 1 seg.

Luego el Promedio es 0,261 + - 1 seg.

6.2.2. - ANALISIS DE LOS DATOS OBTENIDOS:

A) GENERACIÓN DE SANCIONES

Calculo Matematico En El Que La Base De Datos Se Demora En Garantizar La Integridad De Los Datos Para La Generación De Sanciones.

MUESTRA:

RUT
1001
.
.
.
.
1196

DATOS REALES OBTENIDOS:

14:00 Hrs.

41 Seg.: genero sanción desde el Rut 1001 al 1158

42 Seg. : genero sanción desde el Rut 1159 al 1196

15:00 Hrs.

43 Seg.: volvió a ejecutarse y genero sanción nuevamente desde el Rut 1001 al 1196 Ya que la Base de Datos Activa estaba Programada para ejecutarse cada 1 Hr.

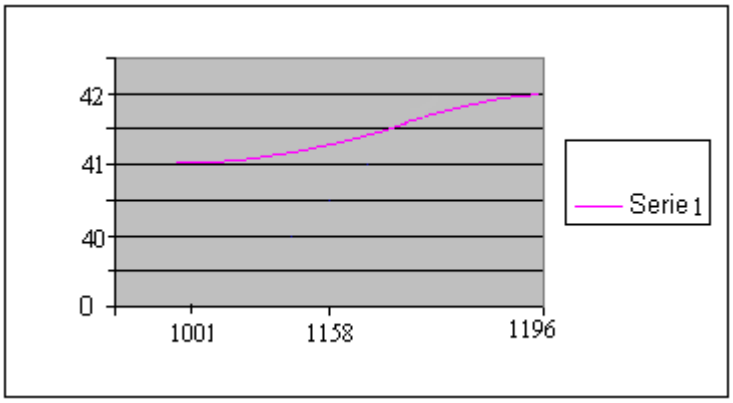


Figura 6.11: Gráfica de Resultados del Evento Generar Sanciones

El Error de Propagación, según la gráfica de la Figura 5.18, el Error aumenta linealmente, luego la función es:

F(x) = ax + b

Con las pruebas realizadas se generaron 196 registros para los cuales se midió el tiempo que se demoró en ser afectado por el evento, la moda del tiempo inconsistente es 41 segundos para 157 registros.

El promedio es Tp = Suma (ti) 83(seg)

---------- = ---------- = 0,423 Segundos por registro

N 196

Entonces con esto el promedio Tip = 0,423

El error es de los 41 a los 42 segundos en tardar de efectuar la operación el sistema, luego el error es de + - 1 seg.

El Promedio es 0,423 + - 1 seg.

B) BORRADO DE SANCIONES

Calculo Matemático En El Que La Base De Datos Se Demora En Garantizar La Integridad De Los Datos Para El Borrado De Sanciones.

MUESTRA:

RUT
1001
.
.
.
.
1196

DATOS REALES OBTENIDOS:

41 Seg.: borro sanción desde el Rut 1001 al 1060

42 Seg.: borro sanción desde el Rut 1061 al 1119

43 Seg.: borro sanción desde el Rut 1120 al 1154

45 Seg.: borro sanción desde el Rut 1155 al 1196

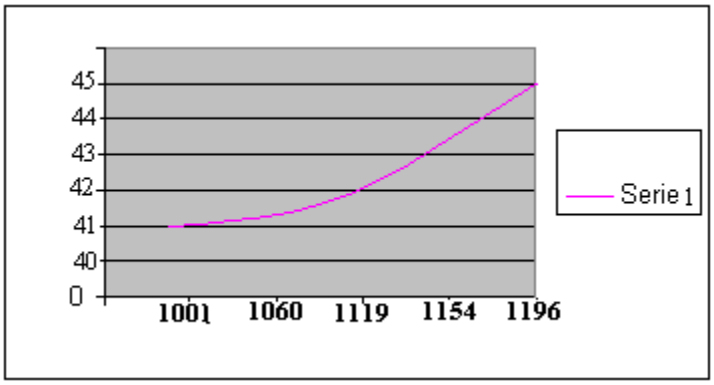


Figura 6.12: Gráfica de Resultados del Evento Borrar Sanciones

El Error de Propagación, según la gráfica de la Figura 5.19, el Error aumenta linealmente, luego la función es:

$F(x) = ax + b$

Con las pruebas realizadas se generaron 196 registros para los cuales se midió el tiempo que se demoró en ser afectado por el evento Borrado de Sanciones, la moda del tiempo inconsistente es 41 segundos para 196 registros.

El promedio es Tp = Suma (ti) 171(seg)

$$\frac{\text{Suma (ti)}}{N} = \frac{171(\text{seg})}{196} = 0{,}872 \text{ Segundos por registro}$$

Entonces con esto el promedio Tip = 0,872

El error total es de + - 0,1 seg.

El Promedio de Error es de los 41 a los 45 segundos, luego el margen de error promedio es de 4 seg.

Luego el Promedio es 0,872 + - 4 seg.

Si Interpolamos los datos para analizar la propagación de Error, el cual es el tiempo en que la Base de Datos Activa tarda en efectuar una operación, podemos concluir que en mil ejecuciones para el Evento Insertar Sanción, tardara un tiempo de 1041 segundos en efectuar la operación, tiempo en que la Base de Datos se encontrara Inconsistente alrededor de 17 minutos.

En el caso del Evento Borrado de Sanciones el tiempo que demora la BDA en efectuar la operación es mayor, por lo que tarda en mil ejecuciones programadas, alrededor de 5456 segundos, lo que equivale a una hora y treinta minutos aproximadamente.

Como conclusión a los dos casos antes expuestos, podemos decir que esta aplicación no es conveniente para una base de datos que contenga un gran volumen de información, o cuando esta Base de Datos sea utilizada por varios usuarios interactivamente, aun cuando la solución para este caso es muy simple, las actualizaciones se pueden programar para aquellas horas que no hay tráfico de información, como por ejemplo aquellas empresas en las cuales los datos son actualizados durante la noche como Financieras o Cooperativas de Ahorro (Ahorrocoop Ltda), en algunos casos es totalmente satisfactorio, todo depende del uso final, en otros casos la información es relevante que durante el día se realice alguna actualización. Todo depende de la Aplicación.

7. Conclusiones

El comportamiento de las reglas de integridad se puede analizar desde dos puntos de vista:

Comportamiento "Reactivo"

Una base de datos con comportamiento reactivo es aquella en la que cada vez que se produce un evento se ejecuta la acción.

Comportamiento "Proactivo"

La base de datos es la que ejecuta las reglas de manera proactiva se cumplan o no se cumplan las condiciones. Si estás se cumplen ve a que evento pertenece y en ese momento ejecuta la acción.

En el caso de nuestra Base de Datos Activa normalmente es de ambos tipos. En cambio una base de datos pasiva puede tener a lo sumo un comportamiento reactivo y con pocas posibilidades de modificación de las Reglas, pero si tiene la integridad fuertemente asegurada pero en un dominio mucho más reducido. En el caso de la base de datos activa esta seguridad de dominio es más amplia pero más débil.

Pero si, se gana en flexibilidad por parte del usuario. No se garantiza en intervalos de tiempo pequeño, pero si en intervalos de tiempo más grande, depende de la estabilidad de la base de datos. La estabilidad de la base de datos activa puede ser menor, ya que necesita de más procesos que la estén chequeando constantemente.

Esto puede ser despreciable con el advenimiento de procesadores más rápidos y sistemas de gestión de bases de datos más estables.

Como conclusión del tema de la integridad es claro que las Bases de Datos Activas son de un dominio mucho más amplio y flexibles que las bases de datos pasiva, sin embargo, esta última es más segura pero mucho más limitada.

8. Anexos

Anexo A : Oracle

Introducción

Oracle es la base de datos más extensamente usada en todo el mundo, es potente y altamente eficiente. Corre virtualmente sobre cualquier tipo de computador. Funciona sobre cualquier máquina, así que cuando se aprende a usar en una de ellas, puede usarse en cualquier máquina. Oracle dispone de un motor principal para la gestión de Bases de Datos basado en Lenguaje SQL estándar, éste permite la creación de perfiles de usuarios y esquemas físicos de Bases de datos de cualquier envergadura o tamaño. En él es posible almacenar Tablas y procedimientos escritos en un lenguaje de programación llamado PL/SQL que comparte con otros productos de la familia Oracle permitiendo gestionar los datos almacenados.

Oracle dispone de varios otros productos para apoyar la construcción de software orientado a la gestión de Bases de Datos tales como: Designer/2000, un producto para dar soporte a todas las fases de los métodos tradicionales de desarrollo de sistemas. Developer/2000 es un producto para la construcción de software de aplicaciones tales como interfaz de usuario, diseño de reportes impresos, gráficos y administración de proyectos.

Objetos de Oracle

Oracle soporta todas las funciones que se esperan, tales como: un lenguaje de diseño de bases de datos muy completo (PL/SQL) que permite implementar diseños "activos", con triggers y procedimientos almacenados, con una integridad referencial integrada por la estructura física de las tablas.

PL/SQL

PL/SQL (Lenguaje Procedural/SQL) es una extensión de SQL, que incorpora muchas de las características de diseño propias de los lenguajes de programación modernos. Permite manipular datos e incluir sentencias de consulta SQL dentro de unidades de código procedural estructuradas en bloques, esto hace de PL/SQL un lenguaje poderoso para el procesamiento de transacciones.

Procedimientos Almacenados

Un procedimiento es un bloque de código PL/SQL, que se almacena en el diccionario de datos y que es llamado por las aplicaciones. Se pueden utilizar para implementar seguridad, no dando acceso directamente a determinadas tablas sino es a través de procedimientos que acceden a esas tablas. Cuando se ejecuta un procedimiento se ejecuta con los privilegios del propietario del procedimiento. Pueden ser llamados usando el nombre que se le haya asignado.

Funciones

Una función es un conjunto de instrucciones en PL/SQL, que pueden ser llamados usando el nombre con que se le haya creado. Se diferencian de los procedimientos, en que las funciones retornan un valor al ambiente desde donde fueron llamadas.

Triggers

Un trigger es un bloque PL/SQL asociado a una tabla, que se ejecuta cuando se produce un determinado evento en la BD. **Se pueden utilizar para mejorar y reforzar la integridad y la seguridad de la BD.**

Esta asociado con una tabla o vista y que automáticamente realiza una acción cuando se ejecuta una operación de INSERT, UPDATE o DELETE sobre la tabla que lo tiene definido.

Un trigger nunca es llamado directamente. sino cuando una aplicación o usuario intenta insertar, actualizar o eliminar filas de una tabla, cualquier trigger asociado con la tabla y la operación es automáticamente ejecutado o "disparado" se dice entonces que hay una activación implícita.

Un trigger puede hacer uso de excepciones para el manejo de errores. Cuando se produce una excepción en un trigger, retorna un mensaje de error, termina y deshace cualquier cambio realizado a menos que la excepción este manejada con una sentencia when en el trigger.

Los trigger pueden utilizarse para:

Forzar automáticamente a las restricciones de datos.

Ventajas del uso de triggers

Verificación automática de restricciones e integridad complejas.

Centralizar la lógica de las restricciones.

Vuelco automático de cambios en las tablas. En otras tablas llamadas log.

Desventajas del uso de triggers

Los lenguajes de bajo nivel 4GL requieren que el usuario especifique completamente todas las condiciones asociadas con las reglas.

Dificultad en el mantenimiento y administración de las reglas

Dificultad para tener una visión global de cuáles tareas están siendo ejecutadas por un conjunto de triggers.

Semántica de los Triggers en Oracle

- No es posible dar prioridades a los triggers asociados al mismo evento y modo, el orden es controlado por el sistema.
- Cada actualización en la acción puede activar otro trigger entonces se suspende la ejecución del actual y se pasa a considerar el otro.
- El máximo numero de triggers en cascada es 32.
- Cuando ocurre una excepción hace un rollback de los cambios hechos por la sentencia original y la acción.

Eventos Básicos de Oracle

Los eventos básicos de Oracle son las sentencias: **Insert, Update y Delete** que son las que entregan un nivel básico de actividad.

Oracle Designer/2000 - Herramienta CASE

Designer/2000 ofrece la posibilidad de hacer desarrollos de sistemas de aplicación más precisos, ya que proporciona un medio para capturar y gestionar una cantidad voluminosa de datos asociados a un sistema.

La herramienta CASE apoya la aplicación de la Metodología Case Method en Forma Gráfica y produciendo las entregas normales de las etapas de desarrollo de un sistema de información, generando el código necesario para la puesta en producción.

Además soporta cualquier metodología de desarrollo que quiera utilizarse para el desarrollo del mismo incluyendo: Buisness Process Reengineering (BPR), Ingeniería en Reverso, Bases de datos distribuidas, etc.

Cuenta con Diagramadores y Herramientas para cada etapa del desarrollo completamente integradas entre ellas, las cuales obtienen la información de un repositorio de información común, que permite acceso a múltiples usuarios en forma simultánea sobre la misma aplicación; esto facilita la creación de aplicaciones corporativas trabajando en equipos de trabajo.

Fase de Estrategia

La meta de la fase de estrategia es adquirir una idea de conjunto de qué es lo que debería hacer el sistema que se va a diseñar y definir el alcance del proyecto.

El propósito de la fase de estrategia es formular una descripción básica del alcance global del proyecto y de cómo transcurrirá el proyecto.

Fase de Análisis

Análisis es el proceso de recoger los requisitos del sistema. El análisis consta de dos partes: recogida de información y análisis de requisitos.

Se crea el Modelo E/R con sus respectivos Atributos y relaciones. Este es el diagrama entidad relación final antes de la construcción. Ahora es cuando se debe realizar cualquier desnormalización necesaria. Hay que considerar los detalles de bajo nivel; y los diseños de las tablas deben ser rigurosos, probados con datos de ejemplo antes de su implementación.

Fase de Diseño:

Del diseño de la base de datos forman parte el diseño de las tablas y columnas junto con la especificación detallada de los dominios y verificación de las restricciones de las columnas. El diseño de la base de datos incluye también la desnormalización de la base de datos para mejorar el rendimiento junto con los disparadores asociados que soporten la desnormalización.

Diseño Físico de los datos

Se crean las Tablas respectivas, en lo posible Debe corresponden a las entidades del Diagrama E/R. También se diseñan las Primary key y Foreing Key.

Fase Construcción

La fase de construcción comprende dos áreas: la base de datos y las aplicaciones. Si todas las fases anteriores han sido cuidadosamente desarrolladas, esta fase se desarrollara sin problemas. La construcción de la base de datos se genera mediante designer/2000. Todos los disparadores y estructuras de datos pueden mantenerse con el modelo físico del designer/2000.

Documentación

La documentación debiera ser un proceso continuo a lo largo de todo el proceso de desarrollo del sistema. Debe acompañar al primer prototipo que el usuario vea. La documentación no debe ser simplemente un paso separado al final del proceso.

Así como las aplicaciones se prueban, la documentación también debe pasar un proceso de pruebas. Probar el sistema en sí es imposible antes de escribir la documentación del sistema, puesto que no hay nada contar lo que comprobar.

Pruebas

Las pruebas son una de las más importantes pero normalmente peor realizadas fases en el proceso de diseño de sistemas. La clave para probar correctamente es utilizar pruebas múltiples. Ninguna prueba única, no importa cuán cuidadosamente efectuada, encontrará todos los errores del sistema.

Cuando se satisface la fase de pruebas del proceso de desarrollo no se necesita revisar el diseño lógico o físico de la base de datos. Esto ya se realizó en las fases de análisis y diseño.

Implantación

En algún momento, el sistema finalizado tiene que ser traspasado a los usuarios y puesto en marcha (después de que halla tenido lugar, por supuesto, la formación del usuario sobre el nuevo sistema).

Mantenimiento

Incluso cuando un sistema está finalizado y puesto en marcha, está cambiando continuamente. Habrá, por supuesto, algunos problemas con cualquier nuevo sistema junto con la necesidad de aumentar los usuarios, peticiones de cambio del funcionamiento del sistema, la necesidad de nuevos informes, etc.

El objetivo principal de la fase de mantenimiento es proporcionar un proceso que pase por un filtro, clasifique y después gestione estos problemas y cambios del sistema. Es necesario reconocer estos cambios, que no sólo afectan a la base de datos y a las aplicaciones, sino que también deben ser reflejados en la documentación del usuario y del sistema, y en la formación. Al personal involucrado en las funciones de ayuda del usuario se le debe mantener informado de cualquier cambio.

Developer/2000

Developer/2000 es e Producto de Oracle que hace fácil el desarrollo de aplicaciones de base de datos.

Una aplicación es un programa informático que realiza una tarea. Una aplicación de base de datos es un programa que utiliza los datos de un sistema de gestión de base de datos como Oracle7. La mayoría de las aplicaciones de base de datos presentan datos de forma útil o introducen y actualizan datos en la base de datos.

Forms

El componente Forms de Developer/2000 es la parte del entorno de desarrollo en la que se construyen los módulos de formularios. También proporciona el entorno de trabajo para desarrollar menús y módulos de biblioteca PL/SQL.

Una aplicación de formularios es una aplicación que presenta datos en un formato interactivo, consiste en un conjunto de campos colocados en una o más ventanas.

Jerarquía de Elementos dentro de un Forms

Módulo de Formularios

El módulo de formularios es el componente principal de las aplicaciones interactivas. También es el módulo más complejo en términos de la estructura interna, ya que contiene muchas clases distintas de elementos.

Triggers (Disparadores)

Un disparador es un bloque de código PL/SQL que se asocia a otro elemento: un formulario, un bloque de datos o un elemento de un bloque de datos. El disparador se lanza o se ejecuta, cuando se producen ciertos eventos: el evento lanza el código.

Bloques de Datos

El bloque de datos es la unidad de construcción intermedia de los formularios. Un bloque de datos se puede ver de dos formas, como una colección de elementos o como una colección de registros, cada uno de los cuáles tiene la misma estructura.

Canvas y Ventanas

Un canvas es la base sobre la que se sitúa el texto plano y los elementos. Cada elemento hace referencia a un único canvas en su hoja de propiedades. Los elementos de un bloque de datos se pueden dividir entre diferentes canvas. Ver Figura 8.1:

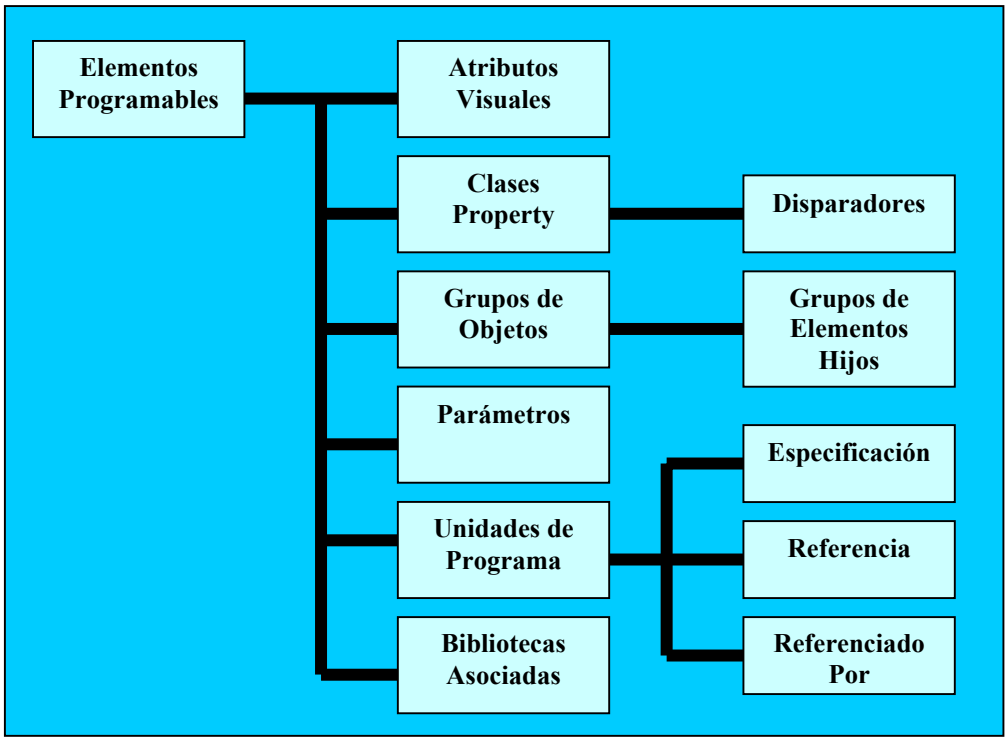


Figura 8.1: Elementos Programables

Un **Atributo Visual con nombre** es una colección de propiedades visuales a las cuáles se les puede hacer referencia desde otro elemento. Si se hiciera eso, las propiedades del atributo visual sobreescribiría las propiedades del elemento. Si se cambiara una propiedad del atributo visual, ésta se cambiaría en todas las propiedades que la heredan del atributo visual.

Una **Clase Property** es un elemento que pertenece a un módulo que contiene un conjunto de propiedades, cualquier propiedad. De la misma forma que con los atributos visuales, cuando un elemento se basa en una propiedad visual, se obtienen todas las propiedades de dicha clase que tienen sentido para el tipo e elemento que se está definiendo.

El **Grupo de elementos** permite agrupar elementos reutilizables para copiarlos o hacer referencia a ellos posteriormente. Recoge un conjunto de elementos en un módulo debajo de un a única cabecera. Copiando o haciendo referencia al grupo de elementos, se obtienen todos los elementos que contiene.

Parámetros

Un parámetro es un formulario, un menú, un informe o un elemento de datos visualizable que se define al nivel del módulo. En el ámbito del módulo, el parámetro se puede utilizar como variable en cualquier unidad de programa PL/SQL.

Unidades de Programa (Program Unit)

Las unidades de programa de un módulo son los paquetes PL/SQL, los procedimientos y las funciones que se definen en el ámbito de dicho módulo. Todos los módulo Developer/2000 permiten definir unidades de programa como parte del módulo. El módulo bibliotecas es algo especial; agrupa las unidades de programa como un módulo y las hace disponibles a otros módulos mediante el elemento **Biblioteca Asociada**.

Report

El componente Report de developer/2000 es la parte del entorno del desarrollo con la que se realizan los módulos de informes. En este entorno se puede hacer referencia a elementos de consultas externas, y se pueden configurar y almacenar elementos de depuración.

Los componentes básicos de un informe son su modelo de datos, su formulario de parámetros, sus disparadores de informes y su composición.

El modelo de datos dl informe es la estructura de datos y sus diferentes representaciones en el informe.

Anexo B: Pantallas

Condiciones Posibles

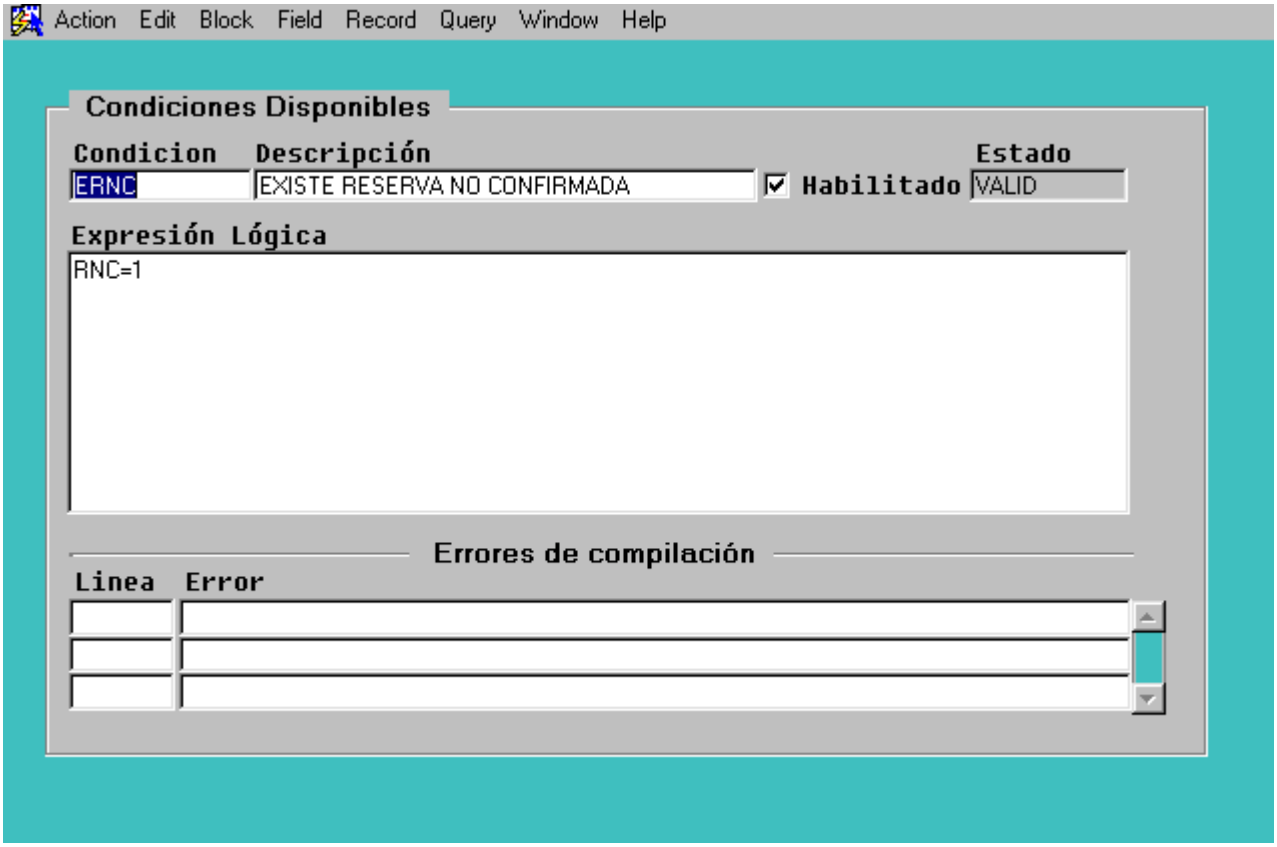


Figura 8.2: Pantalla de Condiciones Posibles

La pantalla Condiciones Posibles de la Figura 8.2, es la encargada de almacenar las condiciones que están asociadas al evento. Los campos son:

- **Condición:** es ingresada la condición, este valor es numérico.
- **Descripción:** es descrita la función de la condición.
- **Expresión Lógica:** en este campo es ingresado un conjunto de términos conectados con operadores lógicos de manera que el resultado tenga sentido, es decir, si este es booleano, se cumple o no se cumple la condición y si este resultado es coherente (tiene sentido lógico).
- **Habilitado:** ***al estar seleccionado el form esta habilitado (activo).***
- **Estado:** valido o invalido, esta valido cuando a sido compilado exitosamente.

Las condiciones van a estar formadas por valores que tienen el resultado de una consulta, cuyo resultado va a ser booleano.

Acciones

Action Edit Block Field Record Query Window Help

Acciones Disponibles

Acción	Descripción		Estado
SANCION	GENERA SANCION SISTEMA DE RESERVAS	☑ Habilitado	VALID

Codigo Pl/SQL Java

```
FOR I IN (SELECT RUT, FECHA
                 FROM RESERVAS
                 WHERE NVL(CONFIRMADA,'N') = 'N'
                  AND TRUNC(FECHA)=TRUNC(SYSDATE))
                  LOOP
                 INSERT INTO SANCIONES VALUES(I.RUT,I.FECHA+1);
PKG_SYS.INS_BITACORA('GESUSO','GESUSO','INSERTANDO SANCION RUT='||I.RUT);
END LOOP;
```

Errores de compilación

Linea	Error

Figura 8.3: Pantalla de Acciones

Ejecuta las acciones asociadas al evento cuando la condición se cumple. (Figura 8.3).

- **Acción:** se ingresa la acción correspondiente.
- **Descripción:** es descrita la acción
- **Código PL Java:** es ingresado por el usuario el código de programa PL/SQL que será capaz de llamar a cualquier servicio del sistema.

Al igual que las pantallas anteriores se encontrara habilitado y en estado valido cuando el form sea compilado sin errores de compilación.

Las acciones tendrán una prioridad de ejecución.

Bitácora de Procesos

Bitacora de Procesos de Eventos

Evento	Descripción	Tipo	Estado
GESUSO	GENERA SANCIONES	TEMPORAL	VALID

Estado proceso	Fecha	Glosa	Usuario
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1010	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1009	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1008	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1007	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1006	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1005	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1004	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1003	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1002	SEMINARIO
GESUSO	23/02/2004 13:00:38	INSERTANDO SANCION RUT=1001	SEMINARIO

Figura 8.4: Bitácora de Procesos - Log de Procesos

En la Figura 8.4, la Bitácora de Procesos con la tabla Log de procesos están relacionadas como Master-Detail (maestro-detallado), es decir una tabla es dependiente de la otra. Los campos son los siguientes:

Bitácora de Procesos

- ***Num Proceso:*** se ingresa el número del proceso que se esta ejecutando por la BD.
- ***Proceso:*** se ingresa el proceso que se esta ejecutando.
- ***Estado:*** indica el estado en el que se encuentra el proceso.
- ***Fecha:*** indica la fecha de ejecución del proceso.
- ***Usuario:*** identifica al usuario esta utilizando la base de datos.

Log de Procesos

- ***Num Proceso:*** indica el proceso en el cual se produjo un error.
- ***Glosa:*** describe el error del proceso.

Como en la bitácora de procesos están almacenados todos los procesos asociados al evento, indicando el estado en el que se encuentra, al producirse un error va inmediatamente va al log de procesos para identificar el número del proceso en el que fue producido el error.

Procesos - Privilegios

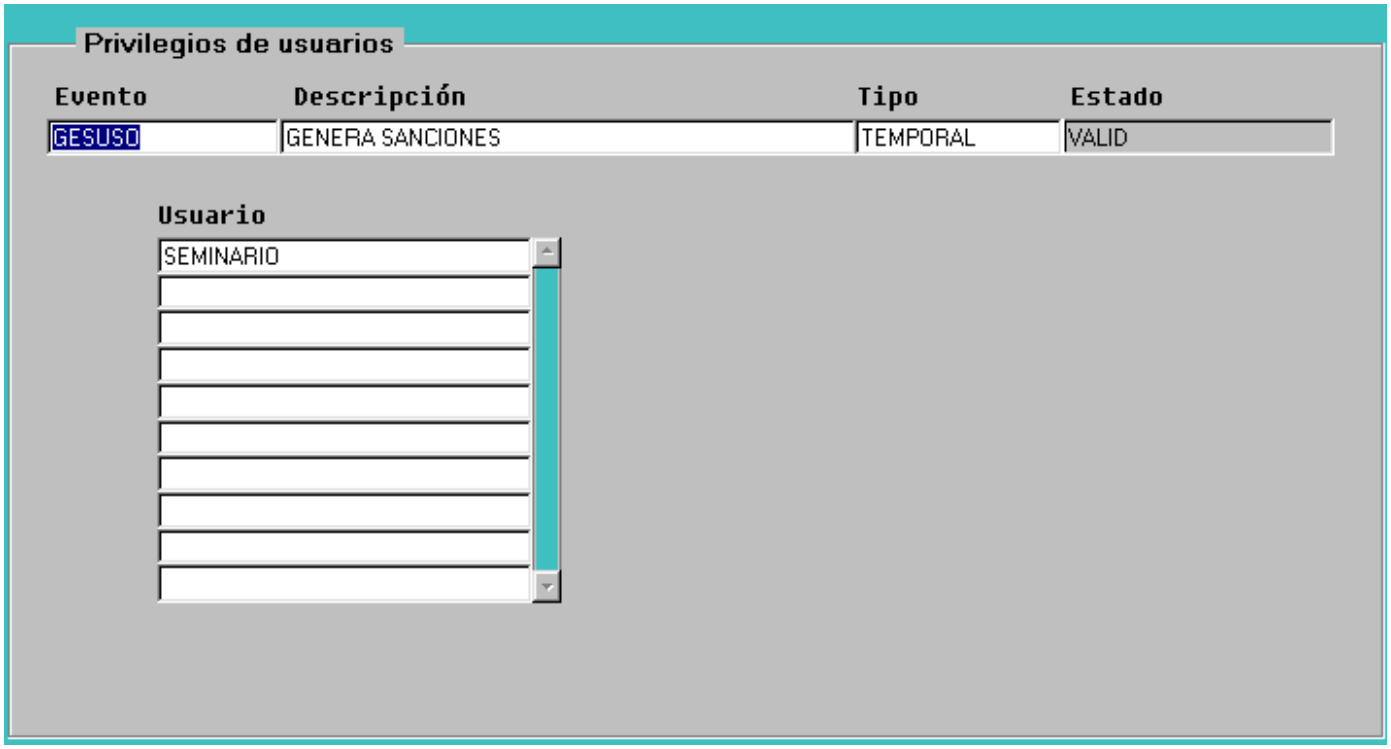


Figura 8.5: Pantalla de Privilegios del Sistema

En la Figura 8.5, las tablas de Procesos y Privilegios están relacionadas como Master-Detail (maestro-detallado), es decir una tabla es dependiente de la otra. Los campos son los siguientes:

Procesos

- ***Proceso:*** es ingresado el proceso que se esta ejecutando.
- ***Descripción:*** describe al proceso en ejecución.
- ***Evento:*** indica que evento esta asociado al proceso que se está ejecutando.

Privilegios

- ***Proceso:*** indica que proceso esta usando el usuario.
- ***Usuario:*** identifica al usuario que esta utilizando la base de datos.

Variables Globales

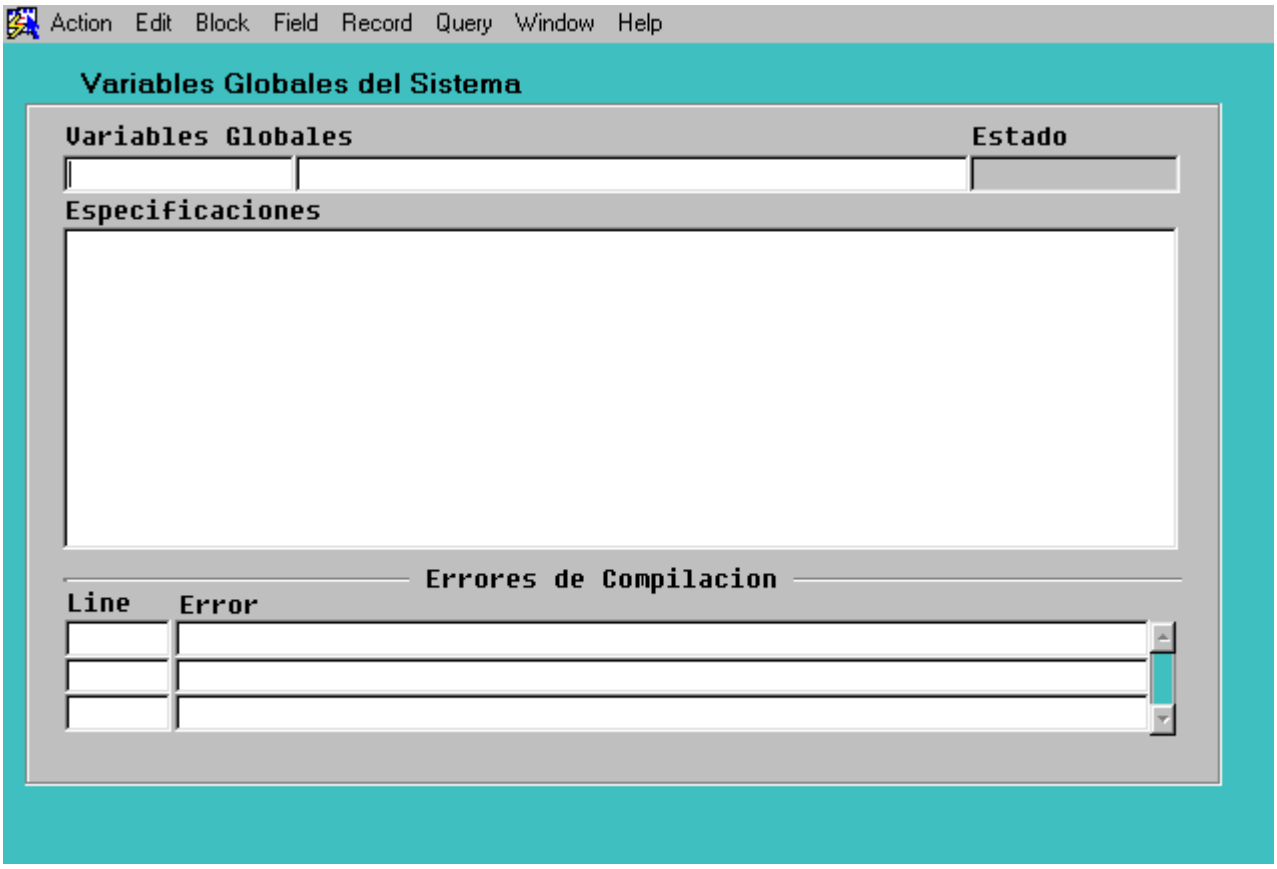


Figura 8.6: Pantalla de las Variables Globales del Sistema

Se ha implementado la Tabla de Variables Globales del Sistema, para que las variables sean accesibles desde cualquier Form de la aplicación, declarándolas como públicas desde un módulo de código, una variable es un valor que cambia en el tiempo pero que en un tiempo "t" determinado y en la evaluación de la expresión tiene un único valor, de lo contrario seria inconsistente, esta pantalla, como lo muestra la Figura 7.4, contiene los siguientes campos:

- ***Variables Globales:*** nombre de la variable con su descripción.
- ***Especificaciones:*** son definidas las variables globales del sistema a utilizar en el cuerpo.
- ***Cuerpo:*** se definen las variables que nos son globales y van a pertenecer solamente al código de programa ingresado en el cuerpo.
- ***Errores de Compilación:*** muestra los errores en tiempo de ejecución.

Variables de Condiciones y Programas

En la Figura 8.7, están contenidas las Variables utilizadas en la evaluación de la Condición, los campos son:

- ***Variable:*** es ingresada la variable.
- ***Descripción:*** es descrita la variable.
- ***Tipo variable:*** las variables tienen asociadas un tipo de dato, el que puede ser de tipo String, Date o Numérico, que se especifica en forma manual por parte del usuario.
- ***Consulta SQL:*** es ingresada por el usuario cualquier consulta que se le desee hacer a la base de datos.
- ***Errores de compilación:*** son enumerados por orden los posibles errores de ejecución.

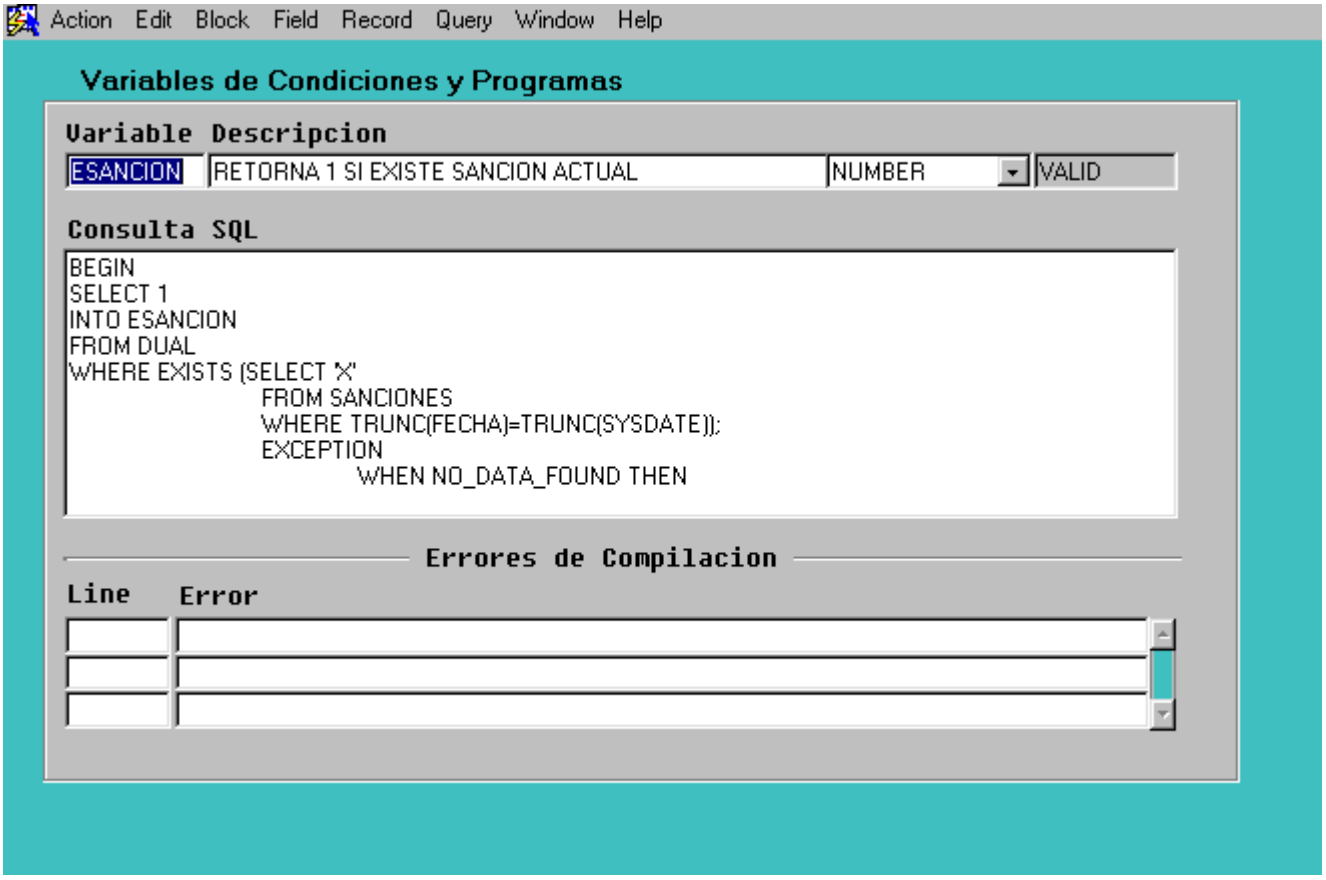


Figura 8.7: Pantalla de las Variables de las Condiciones y Programas

Código de Forms.

Form Condiciones

```
Triggers
        Name                          ON-CHECK-DELETE-MASTER
        Class                        <Null>
        Trigger Text
          -- Begin default relation declare section
          DECLARE
            Dummy_Define CHAR(1);
            -- Begin USER_ERRORS detail declare section
            CURSOR USER_ERRORS_cur IS
              SELECT 1 FROM USER_ERRORS
              WHERE NAME = :condiciones.condicion;
            -- End USER_ERRORS detail declare section
          -- End default relation declare section
          -- Begin default relation program section
          BEGIN
            -- Begin USER_ERRORS detail program section
            OPEN USER_ERRORS_cur;
            FETCH USER_ERRORS_cur INTO Dummy_Define;
            IF ( USER_ERRORS_cur%found ) THEN
              Message('Cannot delete master record when matching detail records exist.');
              CLOSE USER_ERRORS_cur;
              RAISE Form_Trigger_Failure;
            END IF;
            CLOSE USER_ERRORS_cur;
            -- End USER_ERRORS detail program section
          END;
          -- End default relation program section

        Name                          ON-POPULATE-DETAILS
        Class                        <Null>
        Trigger Text
          -- Begin default relation declare section
          DECLARE
            recstat     CHAR(20) := :System.record_status;
            startitm    CHAR(61) := :System.cursor_item;
            rel_id      Relation;
          -- End default relation declare section
          -- Begin default relation program section
          --
          BEGIN
            IF ( recstat = 'NEW' or recstat = 'INSERT' ) THEN
              RETURN;
            END IF;
```

```
-- Begin USER_ERRORS detail program section
IF ( (:condiciones.condicion is not null) ) THEN
  rel_id := Find_Relation('CONDICIONES.RELATION5');
  Query_Master_Details(rel_id, 'USER_ERRORS');
END IF;
-- End USER_ERRORS detail program section

IF ( :System.cursor_item <> startitm ) THEN
   Go_Item(startitm);
   Check_Package_Failure;
END IF;
END;
-- End default relation program section
Triggers
Name                          POST-CHANGE
Class                         <Null>
Trigger Text
  if :system.record_status in ('NEW','INSERT') THEN
     if pkg_sys.existe_obj(:condiciones.condicion) THEN
        msg1('Ya existe objeto: '||:condiciones.condicion);
        raise form_trigger_failure;
     end if;
  end if;
  :condiciones.status := pkg_sys.get_status(:condiciones.condicion,'FUNCTION');
Triggers
Name                          WHEN-CHECKBOX-CHANGED
Class                         <Null>
Trigger Text
  IF :condiciones.condicion is not null and
     :condiciones.expresion is not null THEN
     IF :condiciones.activo = 'S' THEN
        k_commit;
        if :condiciones.status = 'VALID' THEN
           pkg_sys.active_condicion(:condiciones.condicion);
        else
           pkg_sys.desactive_condicion(:condiciones.condicion);
        end if;
     else
        pkg_sys.desactive_condicion(:condiciones.condicion);
     end if;
  END IF; -- :condiciones.condicion is not null

Program Units
  CHECK_PACKAGE_FAILURE (Procedure Body)
    Procedure Check_Package_Failure IS
    BEGIN
      IF NOT ( Form_Success ) THEN
        RAISE Form_Trigger_Failure;
```

```
    END IF;
  END;

CLEAR_ALL_MASTER_DETAILS (Procedure Body)
  PROCEDURE Clear_All_Master_Details IS
    mastblk  CHAR(30);  -- Initial Master Block Cusing Coord
    coordop  CHAR(30);  -- Operation Causing the Coord
    trigblk  CHAR(30);  -- Cur Block On-Clear-Details Fires On
    startitm CHAR(61);  -- Item in which cursor started
    frmstat  CHAR(15);  -- Form Status
    curblk   CHAR(30);  -- Current Block
    currel   CHAR(30);  -- Current Relation
    curdtl   CHAR(30);  -- Current Detail Block

    FUNCTION First_Changed_Block_Below(Master CHAR)
    RETURN CHAR IS
      curblk CHAR(30);  -- Current Block
      currel CHAR(30);  -- Current Relation
      retblk CHAR(30);  -- Return Block
    BEGIN
      --
      -- Initialize Local Vars
      curblk := Master;
      currel := Get_Block_Property(curblk,  FIRST_MASTER_RELATION);
      -- While there exists another relation for this block
      WHILE currel IS NOT NULL LOOP
        -- Get the name of the detail block
        curblk := Get_Relation_Property(currel, DETAIL_NAME);
         -- If this block has changes, return its name
        IF ( Get_Block_Property(curblk, STATUS) IN('CHANGED','INSERT') ) THEN
          RETURN curblk;
        ELSE
          -- No changes, recursively look for changed blocks below
          retblk := First_Changed_Block_Below(curblk);
          --
          -- If some block below is changed, return its name
          IF retblk IS NOT NULL THEN
            RETURN retblk;
          ELSE
            -- Consider the next relation
            currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
          END IF;
        END IF;
      END LOOP;
      -- No changed blocks were found
      RETURN NULL;
    END First_Changed_Block_Below;
```

```
BEGIN
  -- Init Local Vars
  mastblk  := :System.Master_Block;
  coordop  := :System.Coordination_Operation;
  trigblk  := :System.Trigger_Block;
  startitm := :System.Trigger_Item;
  frmstat  := :System.Form_Status;

  -- If the coordination operation is anything but CLEAR_RECORD or
  -- SYNCHRONIZE_BLOCKS, then continue checking.
  IF coordop NOT IN ('CLEAR_RECORD', 'SYNCHRONIZE_BLOCKS') THEN
    -- If we're processing the driving master block...
    IF mastblk = trigblk THEN
      -- If something in the form is changed, find the
      -- first changed block below the master
      IF frmstat = 'CHANGED' THEN
        curblk := First_Changed_Block_Below(mastblk);
        -- If we find a changed block below, go there
        -- and Ask to commit the changes.
        IF curblk IS NOT NULL THEN
          Go_Block(curblk);
          Check_Package_Failure;
          Clear_Block(ASK_COMMIT);
          -- If user cancels commit dialog, raise error
          IF NOT ( :System.Form_Status = 'QUERY'
                  OR :System.Block_Status = 'NEW' ) THEN
            RAISE Form_Trigger_Failure;
          END IF;
        END IF;
      END IF;
    END IF;
  END IF;
  -- Clear all the detail blocks for this master without
  -- any further asking to commit.
  currel := Get_Block_Property(trigblk, FIRST_MASTER_RELATION);
  WHILE currel IS NOT NULL LOOP
    curdtl := Get_Relation_Property(currel, DETAIL_NAME);
    IF Get_Block_Property(curdtl, STATUS) <> 'NEW'  THEN
      Go_Block(curdtl);
      Check_Package_Failure;
      Clear_Block(NO_VALIDATE);
      IF :System.Block_Status <> 'NEW' THEN
        RAISE Form_Trigger_Failure;
      END IF;
    END IF;
    currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
  END LOOP;
  -- Put cursor back where it started
```

```
      IF :System.Cursor_Item <> startitm THEN
        Go_Item(startitm);
        Check_Package_Failure;
      END IF;

    EXCEPTION
      WHEN Form_Trigger_Failure THEN
        IF :System.Cursor_Item <> startitm THEN
          Go_Item(startitm);
        END IF;
        RAISE;

    END Clear_All_Master_Details;

  COMMIT_MUDO (Procedure Body)
    PROCEDURE commit_mudo IS
    rlevel varchar2(30);
    BEGIN
      rlevel := :system.message_level;
      :system.message_level := '25';
      commit;
      :system.message_level := rlevel;
    END;

  CREATE_CONDICION (Procedure Body)
    PROCEDURE create_condicion IS
    BEGIN
      pkg_sys.create_obj(:condiciones.condicion,'return('||:CONDICIONES.EXPRESION||');
','FUNCTION','boolean');
      EXCEPTION
      WHEN others THEN
          msg1('Error al crear condicion: '||:condiciones.condicion||': '||SQLERRM);
          raise form_trigger_failure;
    END;

  DELREC (Procedure Body)
    PROCEDURE delrec IS
    ret number;
    begin
      IF get_permiso_borrar THEN
        go_block('CONDICIONES');
        drop_condicion;
        begin
          delete_record;
          commit_mudo;
          EXCEPTION
          WHEN others THEN
              msg1('Error al borrar geistro de condicion: '||SQLERRM);
```

```
            raise form_trigger_failure;
        end;
        execute_query;
      END IF;
    end;
```

DROP_CONDICION (Procedure Body)

```
  PROCEDURE drop_condicion IS
  -- procedimiento que borra la accion
  BEGIN
    pkg_sys.drop_obj(:condiciones.condicion,'FUNCTION');
    EXCEPTION
    WHEN others THEN
        msg1('Error al borrar condicion: '||:condiciones.condicion||': '||SQLERRM);
        raise form_trigger_failure;
  END;
```

GET_PERMISO_BORRAR (Function Body)

```
  FUNCTION get_permiso_borrar RETURN boolean IS
  ret number;
  BEGIN
    if :condiciones.condicion is null THEN
      return(FALSE);
    end if;
    ret := msg2('Está Seguro(a) de borrar este registro?');
    if ret = 1 THEN
      return(TRUE);
    else
      return(FALSE);
    end if;
  END;
```

K_COMMIT (Procedure Body)

```
  PROCEDURE k_commit IS
  BEGIN
    IF :condiciones.condicion is not null THEN
        go_block('CONDICIONES');
        commit;
        create_condicion;
        commit_mudo;
        go_block('USER_ERRORS');
        execute_query;
        go_block('CONDICIONES');
        :condiciones.status := pkg_sys.get_status(:condiciones.condicion,'FUNCTION');
        if nvl(:condiciones.status,'INVALID') = 'INVALID' THEN
          :condiciones.activo := 'N';
          commit_mudo;
        end if;
```

```
    END IF;
  END;
```

MSG (Procedure Body)

```
  PROCEDURE MSG(p_msg in varchar2) IS
  BEGIN
    message(p_msg,NO_ACKNOWLEDGE);
  END;
```

MSG1 (Procedure Body)

```
  PROCEDURE msg1(p_msg in varchar2) IS
  v_resp number;
  BEGIN
    Set_Alert_Property('UNA_VIA',alert_message_text,p_msg);
    v_resp := show_alert('UNA_VIA');
  END;
```

MSG2 (Function Body)

```
  FUNCTION msg2(p_msg in varchar2) return number IS
  v_resp number;
  BEGIN
    Set_Alert_Property('DOS_VIAS',alert_message_text,p_msg);
    v_resp := show_alert('DOS_VIAS');
    if v_resp = ALERT_BUTTON1 THEN
      return(1);
    else
      return(2);
    end if;
  END;
```

QUERY_MASTER_DETAILS (Procedure Body)

```
  PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
   oldmsg CHAR(2);  -- Old Message Level Setting
   reldef CHAR(5);  -- Relation Deferred Setting
  BEGIN
   -- Initialize Local Variable(s)
   reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
   oldmsg := :System.Message_Level;
   -- If NOT Deferred, Goto detail and execute the query.
   IF reldef = 'FALSE' THEN
     Go_Block(detail);
     Check_Package_Failure;
     :System.Message_Level := '10';
     Execute_Query;
     :System.Message_Level := oldmsg;
   ELSE
     -- Relation is deferred, mark the detail block as un-coordinated
     Set_Block_Property(detail, COORDINATION_STATUS, NON_COORDINATED);
```

```
  END IF;

EXCEPTION
  WHEN Form_Trigger_Failure THEN
    :System.Message_Level := oldmsg;
    RAISE;
END Query_Master_Details;
```

Form Acciones

```
Triggers
    Name                    ON-CHECK-DELETE-MASTER
    Class                   <Null>
    Trigger Text
      -- Begin default relation declare section
      DECLARE
        Dummy_Define CHAR(1);
        -- Begin USER_ERRORS detail declare section
        CURSOR USER_ERRORS_cur IS
          SELECT 1 FROM USER_ERRORS
          WHERE NAME = :ACCIONES.ACCION;
        -- End USER_ERRORS detail declare section
        -- End default relation declare section
      -- Begin default relation program section
      BEGIN
        -- Begin USER_ERRORS detail program section
        OPEN USER_ERRORS_cur;
        FETCH USER_ERRORS_cur INTO Dummy_Define;
        IF ( USER_ERRORS_cur%found ) THEN
          Message('Cannot delete master record when matching detail records exist.');
          CLOSE USER_ERRORS_cur;
          RAISE Form_Trigger_Failure;
        END IF;
        CLOSE USER_ERRORS_cur;
        -- End USER_ERRORS detail program section
      END;
            End default relation program section

Name                    ON-POPULATE-DETAILS
    Class                   <Null>
    Trigger Text
      -- Begin default relation declare section
      DECLARE
        recstat     CHAR(20) := :System.record_status;
        startitm    CHAR(61) := :System.cursor_item;
        rel_id      Relation;
      -- End default relation declare section
```

```
-- Begin default relation program section
BEGIN
  IF ( recstat = 'NEW' or recstat = 'INSERT' ) THEN
    RETURN;
  END IF;              --
  -- Begin USER_ERRORS detail program section
  IF ( (:ACCIONES.ACCION is not null) ) THEN
    rel_id := Find_Relation('ACCIONES.RELATION5');
    Query_Master_Details(rel_id, 'USER_ERRORS');
  END IF;
  -- End USER_ERRORS detail program section
  IF ( :System.cursor_item <> startitm ) THEN
     Go_Item(startitm);
     Check_Package_Failure;
  END IF;
END;
-- End default relation program section

Triggers
  Name                          POST-CHANGE
  Class                        <Null>
  Trigger Text
    if :system.record_status in ('NEW','INSERT') THEN
      if pkg_sys.existe_obj(:acciones.accion) THEN
        message('Ya existe objeto '||:acciones.accion);
        message('Ya existe objeto '||:acciones.accion);
        raise form_trigger_failure;
      end if;
    end if;
    :acciones.status := pkg_sys.get_status(:acciones.accion,'PROCEDURE');
Triggers
  Name                          WHEN-CHECKBOX-CHANGED
  Class                        <Null>
  Trigger Text
    IF :acciones.accion is not null and
      :acciones.codigo_pl_java is not null THEN
      IF :acciones.activo = 'S' THEN
         k_commit;
      END IF; -- :acciones.habilitado = 'S'
    END IF; -- :acciones.accion is not null

Program Units
  CHECK_PACKAGE_FAILURE (Procedure Body)
    Procedure Check_Package_Failure IS
    BEGIN
      IF NOT ( Form_Success ) THEN
        RAISE Form_Trigger_Failure;
      END IF;
```

```
  END;

CLEAR_ALL_MASTER_DETAILS (Procedure Body)
  PROCEDURE Clear_All_Master_Details IS
    mastblk  CHAR(30);  -- Initial Master Block Cusing Coord
    coordop  CHAR(30);  -- Operation Causing the Coord
    trigblk  CHAR(30);  -- Cur Block On-Clear-Details Fires On
    startitm CHAR(61);  -- Item in which cursor started
    frmstat  CHAR(15);  -- Form Status
    curblk   CHAR(30);  -- Current Block
    currel   CHAR(30);  -- Current Relation
    curdtl   CHAR(30);  -- Current Detail Block

    FUNCTION First_Changed_Block_Below(Master CHAR)
    RETURN CHAR IS
      curblk CHAR(30);  -- Current Block
      currel CHAR(30);  -- Current Relation
      retblk CHAR(30);  -- Return Block
    BEGIN
      -- Initialize Local Vars
      curblk := Master;
      currel := Get_Block_Property(curblk,  FIRST_MASTER_RELATION);
      -- While there exists another relation for this block
      WHILE currel IS NOT NULL LOOP
        -- Get the name of the detail block
        curblk := Get_Relation_Property(currel, DETAIL_NAME);
        -- If this block has changes, return its name
        IF ( Get_Block_Property(curblk, STATUS) IN('CHANGED','INSERT') ) THEN
          RETURN curblk;
        ELSE
          -- No changes, recursively look for changed blocks below
          retblk := First_Changed_Block_Below(curblk);
          -- If some block below is changed, return its name
          IF retblk IS NOT NULL THEN
            RETURN retblk;
          ELSE
            -- Consider the next relation
            currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
          END IF;
        END IF;
      END LOOP;
      -- No changed blocks were found
      RETURN NULL;
    END First_Changed_Block_Below;

  BEGIN
    -- Init Local Vars
    mastblk  := :System.Master_Block;
```

```
coordop  := :System.Coordination_Operation;
trigblk  := :System.Trigger_Block;
startitm := :System.Trigger_Item;
frmstat  := :System.Form_Status;
-- If the coordination operation is anything but CLEAR_RECORD or
-- SYNCHRONIZE_BLOCKS, then continue checking.
IF coordop NOT IN ('CLEAR_RECORD', 'SYNCHRONIZE_BLOCKS') THEN
  -- If we're processing the driving master block...
  IF mastblk = trigblk THEN
    -- If something in the form is changed, find the
    -- first changed block below the master
    IF frmstat = 'CHANGED' THEN
      curblk := First_Changed_Block_Below(mastblk);
      -- If we find a changed block below, go there
      -- and Ask to commit the changes.
      IF curblk IS NOT NULL THEN
        Go_Block(curblk);
        Check_Package_Failure;
        Clear_Block(ASK_COMMIT);
        -- If user cancels commit dialog, raise error
        IF NOT ( :System.Form_Status = 'QUERY'
                OR :System.Block_Status = 'NEW' ) THEN
          RAISE Form_Trigger_Failure;
        END IF;
      END IF;
    END IF;
  END IF;
END IF;
-- Clear all the detail blocks for this master without
-- any further asking to commit.
currel := Get_Block_Property(trigblk, FIRST_MASTER_RELATION);
WHILE currel IS NOT NULL LOOP
  curdtl := Get_Relation_Property(currel, DETAIL_NAME);
  IF Get_Block_Property(curdtl, STATUS) <> 'NEW'  THEN
    Go_Block(curdtl);
    Check_Package_Failure;
    Clear_Block(NO_VALIDATE);
    IF :System.Block_Status <> 'NEW' THEN
      RAISE Form_Trigger_Failure;
    END IF;
  END IF;
  currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
END LOOP;
-- Put cursor back where it started
IF :System.Cursor_Item <> startitm THEN
  Go_Item(startitm);
  Check_Package_Failure;
END IF;
```

```
  EXCEPTION
    WHEN Form_Trigger_Failure THEN
      IF :System.Cursor_Item <> startitm THEN
        Go_Item(startitm);
      END IF;
      RAISE;
  END Clear_All_Master_Details;
COMMIT_MUDO (Procedure Body)
  PROCEDURE commit_mudo IS
  rlevel varchar2(30);
  BEGIN
    rlevel := :system.message_level;
    :system.message_level := '25';
    commit;
    :system.message_level := rlevel;
  END;
CREATE_ACCION (Procedure Body)
  PROCEDURE create_accion IS
  BEGIN

pkg_sys.create_obj(:acciones.accion,:ACCIONES.CODIGO_PL_JAVA,'PROCEDURE',null);
    EXCEPTION
    WHEN others THEN
        message('Error al crear accion: '||:acciones.accion||': '||SQLERRM);
        message('Error al crear accion: '||:acciones.accion||': '||SQLERRM);
        raise form_trigger_failure;
  END;

DELREC (Procedure Body)
  PROCEDURE delrec IS
  begin
    IF get_permiso_borrar THEN
      go_block('ACCIONES');
      drop_accion;
      begin
        delete_record;
        commit_mudo;
        EXCEPTION
        WHEN others THEN
            message('Error al borrar geistro de accion: '||SQLERRM);
            message('Error al borrar geistro de accion: '||SQLERRM);
            raise form_trigger_failure;
      end;
      execute_query;
    END IF;
  end;

DROP_ACCION (Procedure Body)
```

```
  PROCEDURE drop_accion IS
  -- procedimiento que borra la accion
  BEGIN
    pkg_sys.drop_obj(:acciones.accion,'PROCEDURE');
    EXCEPTION
    WHEN others THEN
        message('Error al borrar accion: '||SQLERRM);
        message('Error al borrar accion: '||SQLERRM);
        raise form_trigger_failure;
  END;
```

GET_PERMISO_BORRAR (Function Body)

```
  FUNCTION get_permiso_borrar RETURN boolean IS
  ret number;
  BEGIN
    if :acciones.accion is null THEN
      return(FALSE);
    end if;
    ret := show_alert('ALERTA_BORRADO');
    if ret = ALERT_BUTTON1 THEN
      return(TRUE);
    else
      return(FALSE);
    end if;
  END;
```

K_COMMIT (Procedure Body)

```
  PROCEDURE k_commit IS
  BEGIN
    IF :acciones.accion is not null THEN
        go_block('ACCIONES');
        commit;
        create_accion;
        commit_mudo;
        go_block('USER_ERRORS');
        execute_query;
        go_block('ACCIONES');
        :acciones.status := pkg_sys.get_status(:acciones.accion,'PROCEDURE');
        if nvl(:acciones.status,'INVALID') = 'INVALID' THEN
          :acciones.activo := 'N';
          commit_mudo;
        end if;
    END IF;
  END;
```

QUERY_MASTER_DETAILS (Procedure Body)

```
  PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
   oldmsg CHAR(2);  -- Old Message Level Setting
```

```
  reldef CHAR(5);  -- Relation Deferred Setting
BEGIN
  -- Initialize Local Variable(s)
  reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
  oldmsg := :System.Message_Level;
  --
  -- If NOT Deferred, Goto detail and execute the query.
  IF reldef = 'FALSE' THEN
    Go_Block(detail);
    Check_Package_Failure;
    :System.Message_Level := '10';
    Execute_Query;
    :System.Message_Level := oldmsg;
  ELSE
    -- Relation is deferred, mark the detail block as un-coordinated
    Set_Block_Property(detail, COORDINATION_STATUS, NON_COORDINATED);
  END IF;
EXCEPTION
    WHEN Form_Trigger_Failure THEN
      :System.Message_Level := oldmsg;
      RAISE;
END Query_Master_Details;
```

Form Bitácora Procesos

```
Triggers
  Name                          ON-CLEAR-DETAILS
  Class                         <Null>
  Trigger Text
    -- Begin default relation program section
    BEGIN
      Clear_All_Master_Details;
    END;
    -- End default relation program section
    Triggers
      Name                      ON-CHECK-DELETE-MASTER
      Class                     <Null>
      Trigger Text
        -- Begin default relation declare section
        DECLARE
          Dummy_Define CHAR(1);
          -- Begin BITACORA_PROCESOS detail declare section
          CURSOR BITACORA_PROCESOS_cur IS
            SELECT 1 FROM BITACORA_PROCESOS
            WHERE EVENTO = :EVENTOS.EVENTO;
          -- End BITACORA_PROCESOS detail declare section
        -- End default relation declare section
        -- Begin default relation program section
```

```
BEGIN
  -- Begin BITACORA_PROCESOS detail program section
  OPEN BITACORA_PROCESOS_cur;
  FETCH BITACORA_PROCESOS_cur INTO Dummy_Define;
  IF ( BITACORA_PROCESOS_cur%found ) THEN
    Message('Cannot delete master record when matching detail records exist.');
    CLOSE BITACORA_PROCESOS_cur;
    RAISE Form_Trigger_Failure;
  END IF;
  CLOSE BITACORA_PROCESOS_cur;
  -- End BITACORA_PROCESOS detail program section
END;
-- End default relation program section

Name                          ON-POPULATE-DETAILS
Class                         <Null>
Trigger Text
  --
  -- Begin default relation declare section
  DECLARE
    recstat     CHAR(20) := :System.record_status;
    startitm    CHAR(61) := :System.cursor_item;
    rel_id      Relation;
  -- End default relation declare section
  -- Begin default relation program section
  BEGIN
    IF ( recstat = 'NEW' or recstat = 'INSERT' ) THEN
      RETURN;
    END IF;
    --Begin BITACORA_PROCESOS detail program section
    IF ( (:EVENTOS.EVENTO is not null) ) THEN
      rel_id := Find_Relation('EVENTOS.EV_BIT');
      Query_Master_Details(rel_id, 'BITACORA_PROCESOS');
    END IF;
    -- End BITACORA_PROCESOS detail program section

    IF ( :System.cursor_item <> startitm ) THEN
       Go_Item(startitm);
       Check_Package_Failure;
    END IF;
  END;
  -- End default relation program section

Program Units
 CHECK_PACKAGE_FAILURE (Procedure Body)
  Procedure Check_Package_Failure IS
  BEGIN
    IF NOT ( Form_Success ) THEN
```

```
    RAISE Form_Trigger_Failure;
  END IF;
END;
```

CLEAR_ALL_MASTER_DETAILS (Procedure Body)

```
PROCEDURE Clear_All_Master_Details IS
  mastblk  CHAR(30);  -- Initial Master Block Cusing Coord
  coordop  CHAR(30);  -- Operation Causing the Coord
  trigblk  CHAR(30);  -- Cur Block On-Clear-Details Fires On
  startitm CHAR(61);  -- Item in which cursor started
  frmstat  CHAR(15);  -- Form Status
  curblk   CHAR(30);  -- Current Block
  currel   CHAR(30);  -- Current Relation
  curdtl   CHAR(30);  -- Current Detail Block

  FUNCTION First_Changed_Block_Below(Master CHAR)
  RETURN CHAR IS
    curblk CHAR(30);  -- Current Block
    currel CHAR(30);  -- Current Relation
    retblk CHAR(30);  -- Return Block
  BEGIN
    -- Initialize Local Vars
    curblk := Master;
    currel := Get_Block_Property(curblk,  FIRST_MASTER_RELATION);
    -- While there exists another relation for this block
    WHILE currel IS NOT NULL LOOP
      -- Get the name of the detail block
      curblk := Get_Relation_Property(currel, DETAIL_NAME);
      -- If this block has changes, return its name
      IF ( Get_Block_Property(curblk, STATUS) IN('CHANGED','INSERT') ) THEN
        RETURN curblk;
      ELSE
        -- No changes, recursively look for changed blocks below
        retblk := First_Changed_Block_Below(curblk);
        -- If some block below is changed, return its name
        IF retblk IS NOT NULL THEN
          RETURN retblk;
        ELSE
          -- Consider the next relation
          currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
        END IF;
      END IF;
    END LOOP;
    -- No changed blocks were found
    RETURN NULL;
  END First_Changed_Block_Below;

BEGIN
```

```
-- Init Local Vars
mastblk  := :System.Master_Block;
coordop  := :System.Coordination_Operation;
trigblk  := :System.Trigger_Block;
startitm := :System.Trigger_Item;
frmstat  := :System.Form_Status;
-- If the coordination operation is anything but CLEAR_RECORD or
-- SYNCHRONIZE_BLOCKS, then continue checking.
IF coordop NOT IN ('CLEAR_RECORD', 'SYNCHRONIZE_BLOCKS') THEN
  -- If we're processing the driving master block...
  IF mastblk = trigblk THEN
    -- If something in the form is changed, find the
    -- first changed block below the master
    IF frmstat = 'CHANGED' THEN
      curblk := First_Changed_Block_Below(mastblk);
      -- If we find a changed block below, go there
      -- and Ask to commit the changes.
      IF curblk IS NOT NULL THEN
        Go_Block(curblk);
        Check_Package_Failure;
        Clear_Block(ASK_COMMIT);
        -- If user cancels commit dialog, raise error
        IF NOT ( :System.Form_Status = 'QUERY'
                OR :System.Block_Status = 'NEW' ) THEN
          RAISE Form_Trigger_Failure;
        END IF;
      END IF;
    END IF;
  END IF;
END IF;
-- Clear all the detail blocks for this master without
-- any further asking to commit.
currel := Get_Block_Property(trigblk, FIRST_MASTER_RELATION);
WHILE currel IS NOT NULL LOOP
  curdtl := Get_Relation_Property(currel, DETAIL_NAME);
  IF Get_Block_Property(curdtl, STATUS) <> 'NEW'  THEN
    Go_Block(curdtl);
    Check_Package_Failure;
    Clear_Block(NO_VALIDATE);
    IF :System.Block_Status <> 'NEW' THEN
      RAISE Form_Trigger_Failure;
    END IF;
  END IF;
  currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
END LOOP;
-- Put cursor back where it started
IF :System.Cursor_Item <> startitm THEN
  Go_Item(startitm);
```

```
    Check_Package_Failure;
  END IF;

EXCEPTION
  WHEN Form_Trigger_Failure THEN
    IF :System.Cursor_Item <> startitm THEN
      Go_Item(startitm);
    END IF;
    RAISE;

END Clear_All_Master_Details;
```

GET_STATUS (Function Body)

```
FUNCTION get_status return varchar2 IS
BEGIN
  if :eventos.tipo = 'TABLA' THEN
     return(nvl(pkg_sys.get_status(:eventos.evento,'TRIGGER'),'INVALID'));
  elsif :eventos.tipo = 'TEMPORAL' THEN
     if pkg_sys.get_next_date(:eventos.job_no) > sysdate THEN
       return('VALID');
     else
       return('INVALID');
     end if;
  else
     msg1('No existe tipo de Evento: '||:eventos.tipo);
     raise form_trigger_failure;
  end if;
END;
```

MSG1 (Procedure Body)

```
PROCEDURE msg1(p_msg in varchar2) IS
v_resp number;
BEGIN
  Set_Alert_Property('UNA_VIA',alert_message_text,p_msg);
  v_resp := show_alert('UNA_VIA');
END;
```

QUERY_MASTER_DETAILS (Procedure Body)

```
PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
  oldmsg CHAR(2);  -- Old Message Level Setting
  reldef CHAR(5);  -- Relation Deferred Setting
BEGIN
  -- Initialize Local Variable(s)
  reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
  oldmsg := :System.Message_Level;
  -- If NOT Deferred, Goto detail and execute the query.
  IF reldef = 'FALSE' THEN
    Go_Block(detail);
```

```
      Check_Package_Failure;
      :System.Message_Level := '10';
      Execute_Query;
      :System.Message_Level := oldmsg;
    ELSE
      -- Relation is deferred, mark the detail block as un-coordinated
      Set_Block_Property(detail, COORDINATION_STATUS, NON_COORDINATED);
    END IF;

  EXCEPTION
      WHEN Form_Trigger_Failure THEN
        :System.Message_Level := oldmsg;
        RAISE;
  END Query_Master_Details;
```

Form Provilegios

```
Triggers
    Name                          ON-CLEAR-DETAILS
    Class                         <Null>
    Trigger Text
      -- Begin default relation program section
      BEGIN
        Clear_All_Master_Details;
      END;
      -- End default relation program section
Name                          ON-POPULATE-DETAILS
        Class                         <Null>
        Trigger Text
          -- Begin default relation declare section
          DECLARE
            recstat     CHAR(20) := :System.record_status;
            startitm    CHAR(61) := :System.cursor_item;
            rel_id      Relation;
          -- End default relation declare section
          -- Begin default relation program section
          BEGIN
            IF ( recstat = 'NEW' or recstat = 'INSERT' ) THEN
              RETURN;
            END IF;
            -- Begin PRIVILEGIOS detail program section
            IF ( (:EVENTOS.EVENTO is not null) ) THEN
              rel_id := Find_Relation('EVENTOS.EVENTOS_PRIVILEGIOS');
              Query_Master_Details(rel_id, 'PRIVILEGIOS');
            END IF;
            -- End PRIVILEGIOS detail program section

            IF ( :System.cursor_item <> startitm ) THEN
```

```
            Go_Item(startitm);
            Check_Package_Failure;
          END IF;
        END;
        End default relation program section

Program Units
    CHECK_PACKAGE_FAILURE (Procedure Body)
      Procedure Check_Package_Failure IS
      BEGIN
        IF NOT ( Form_Success ) THEN
          RAISE Form_Trigger_Failure;
        END IF;
      END;

    CLEAR_ALL_MASTER_DETAILS (Procedure Body)
      PROCEDURE Clear_All_Master_Details IS
        mastblk  CHAR(30);  -- Initial Master Block Cusing Coord
        coordop  CHAR(30);  -- Operation Causing the Coord
        trigblk  CHAR(30);  -- Cur Block On-Clear-Details Fires On
        startitm CHAR(61);  -- Item in which cursor started
        frmstat  CHAR(15);  -- Form Status
        curblk   CHAR(30);  -- Current Block
        currel   CHAR(30);  -- Current Relation
        curdtl   CHAR(30);  -- Current Detail Block

        FUNCTION First_Changed_Block_Below(Master CHAR)
        RETURN CHAR IS
          curblk CHAR(30);  -- Current Block
          currel CHAR(30);  -- Current Relation
          retblk CHAR(30);  -- Return Block
        BEGIN
          -- Initialize Local Vars
          curblk := Master;
          currel := Get_Block_Property(curblk,  FIRST_MASTER_RELATION);
          -- While there exists another relation for this block
          WHILE currel IS NOT NULL LOOP
            -- Get the name of the detail block
            curblk := Get_Relation_Property(currel, DETAIL_NAME);
            -- If this block has changes, return its name
            IF ( Get_Block_Property(curblk, STATUS) IN('CHANGED','INSERT') ) THEN
              RETURN curblk;
            ELSE
              -- No changes, recursively look for changed blocks below
              retblk := First_Changed_Block_Below(curblk);
              -- If some block below is changed, return its name
              IF retblk IS NOT NULL THEN
                RETURN retblk;
```

```
      ELSE
        -- Consider the next relation
         currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
      END IF;
    END IF;
  END LOOP;
  -- No changed blocks were found
  RETURN NULL;
END First_Changed_Block_Below;

BEGIN
  -- Init Local Vars
  mastblk  := :System.Master_Block;
  coordop  := :System.Coordination_Operation;
  trigblk  := :System.Trigger_Block;
  startitm := :System.Trigger_Item;
  frmstat  := :System.Form_Status;
  -- If the coordination operation is anything but CLEAR_RECORD or
  -- SYNCHRONIZE_BLOCKS, then continue checking.
  IF coordop NOT IN ('CLEAR_RECORD', 'SYNCHRONIZE_BLOCKS') THEN
    -- If we're processing the driving master block...
    IF mastblk = trigblk THEN
      -- If something in the form is changed, find the
      -- first changed block below the master
      --
      IF frmstat = 'CHANGED' THEN
        curblk := First_Changed_Block_Below(mastblk);
        -- If we find a changed block below, go there
        -- and Ask to commit the changes.
        IF curblk IS NOT NULL THEN
          Go_Block(curblk);
          Check_Package_Failure;
          Clear_Block(ASK_COMMIT);
          -- If user cancels commit dialog, raise error
          IF NOT ( :System.Form_Status = 'QUERY'
                 OR :System.Block_Status = 'NEW' ) THEN
            RAISE Form_Trigger_Failure;
          END IF;
        END IF;
      END IF;
    END IF;
  END IF;

  -- Clear all the detail blocks for this master without
  -- any further asking to commit.
  currel := Get_Block_Property(trigblk, FIRST_MASTER_RELATION);
  WHILE currel IS NOT NULL LOOP
    curdtl := Get_Relation_Property(currel, DETAIL_NAME);
```

```
      IF Get_Block_Property(curdtl, STATUS) <> 'NEW'  THEN
        Go_Block(curdtl);
        Check_Package_Failure;
        Clear_Block(NO_VALIDATE);
        IF :System.Block_Status <> 'NEW' THEN
          RAISE Form_Trigger_Failure;
        END IF;
      END IF;
      currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
    END LOOP;
    -- Put cursor back where it started
    IF :System.Cursor_Item <> startitm THEN
      Go_Item(startitm);
      Check_Package_Failure;
    END IF;

  EXCEPTION
    WHEN Form_Trigger_Failure THEN
      IF :System.Cursor_Item <> startitm THEN
        Go_Item(startitm);
      END IF;
      RAISE;

  END Clear_All_Master_Details;
```

GET_STATUS (Function Body)

```
  FUNCTION get_status return varchar2 IS
  BEGIN
    if :eventos.tipo = 'TABLA' THEN
        return(nvl(pkg_sys.get_status(:eventos.evento,'TRIGGER'),'INVALID'));
    elsif :eventos.tipo = 'TEMPORAL' THEN
        if pkg_sys.get_next_date(:eventos.job_no) > sysdate THEN
          return('VALID');
        else
          return('INVALID');
        end if;
    else
        msg1('No existe tipo de Evento: '||:eventos.tipo);
        raise form_trigger_failure;
    end if;
  END;
```

MSG1 (Procedure Body)

```
  PROCEDURE msg1(p_msg in varchar2) IS
  v_resp number;
  BEGIN
    Set_Alert_Property('UNA_VIA',alert_message_text,p_msg);
    v_resp := show_alert('UNA_VIA');
```

```
END;

QUERY_MASTER_DETAILS (Procedure Body)
PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
  oldmsg CHAR(2);  -- Old Message Level Setting
  reldef CHAR(5);  -- Relation Deferred Setting
BEGIN
  -- Initialize Local Variable(s)
  reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
  oldmsg := :System.Message_Level;
  -- If NOT Deferred, Goto detail and execute the query.
  IF reldef = 'FALSE' THEN
    Go_Block(detail);
    Check_Package_Failure;
    :System.Message_Level := '10';
    Execute_Query;
    :System.Message_Level := oldmsg;
  ELSE
    -- Relation is deferred, mark the detail block as un-coordinated
    Set_Block_Property(detail, COORDINATION_STATUS, NON_COORDINATED);
  END IF;

EXCEPTION
    WHEN Form_Trigger_Failure THEN
      :System.Message_Level := oldmsg;
      RAISE;
END Query_Master_Details;
```

Form Variables Globales

```
Triggers
      Name                          ON-CHECK-DELETE-MASTER
      Class                         <Null>
      Trigger Text
        -- Begin default relation declare section
        DECLARE
          Dummy_Define CHAR(1);
          -- Begin USER_ERRORS detail declare section
          CURSOR USER_ERRORS_cur IS
            SELECT 1 FROM USER_ERRORS
            WHERE NAME = :VARIABLES_GLOBALES.NOMBRE;
          -- End USER_ERRORS detail declare section
        -- End default relation declare section
        BEGIN
          -- Begin USER_ERRORS detail program section
          OPEN USER_ERRORS_cur;
```

```
        FETCH USER_ERRORS_cur INTO Dummy_Define;
        IF ( USER_ERRORS_cur%found ) THEN
          Message('Cannot delete master record when matching detail records exist.');
          CLOSE USER_ERRORS_cur;
          RAISE Form_Trigger_Failure;
        END IF;
        CLOSE USER_ERRORS_cur;
        -- End USER_ERRORS detail program section
      END;
      -- End default relation program section
Name                          ON-POPULATE-DETAILS
      Class                          <Null>
      Trigger Text
      -- Begin default relation declare section
      DECLARE
        recstat     CHAR(20) := :System.record_status;
        startitm    CHAR(61) := :System.cursor_item;
        rel_id      Relation;
      -- End default relation declare section
      -- Begin default relation program section
      BEGIN
        IF ( recstat = 'NEW' or recstat = 'INSERT' ) THEN
          RETURN;
        END IF;
        -- Begin USER_ERRORS detail program section
        IF ( (:VARIABLES_GLOBALES.NOMBRE is not null) ) THEN
          rel_id := Find_Relation('VARIABLES_GLOBALES.RELATION12');
          Query_Master_Details(rel_id, 'USER_ERRORS');
        END IF;
        -- End USER_ERRORS detail program section
        IF ( :System.cursor_item <> startitm ) THEN
           Go_Item(startitm);
           Check_Package_Failure;
        END IF;
      END;
      -- End default relation program section
Triggers
        Name                          POST-CHANGE
        Class                          <Null>
        Trigger Text
          if :system.record_status in ('NEW','INSERT') THEN
             if pkg_sys.existe_obj(:variables_globales.nombre) THEN
                message('Ya existe objeto: '||:variables_globales.nombre);
                message('Ya existe objeto: '||:variables_globales.nombre);
                raise form_trigger_failure;
             end if;
          end if;
```

```
            :variables_globales.status :=
pkg_sys.get_status(:variables_globales.nombre,'PACKAGE');
Program Units
    CHECK_PACKAGE_FAILURE (Procedure Body)
      Procedure Check_Package_Failure IS
      BEGIN
        IF NOT ( Form_Success ) THEN
          RAISE Form_Trigger_Failure;
        END IF;
      END;

    CLEAR_ALL_MASTER_DETAILS (Procedure Body)
      PROCEDURE Clear_All_Master_Details IS
        mastblk  CHAR(30);  -- Initial Master Block Cusing Coord
        coordop  CHAR(30);  -- Operation Causing the Coord
        trigblk  CHAR(30);  -- Cur Block On-Clear-Details Fires On
        startitm CHAR(61);  -- Item in which cursor started
        frmstat  CHAR(15);  -- Form Status
        curblk   CHAR(30);  -- Current Block
        currel   CHAR(30);  -- Current Relation
        curdtl   CHAR(30);  -- Current Detail Block

        FUNCTION First_Changed_Block_Below(Master CHAR)
        RETURN CHAR IS
          curblk CHAR(30);  -- Current Block
          currel CHAR(30);  -- Current Relation
          retblk CHAR(30);  -- Return Block
        BEGIN
          -- Initialize Local Vars
          curblk := Master;
          currel := Get_Block_Property(curblk,  FIRST_MASTER_RELATION);
          -- While there exists another relation for this block
          WHILE currel IS NOT NULL LOOP
            -- Get the name of the detail block
            curblk := Get_Relation_Property(currel, DETAIL_NAME);
            -- If this block has changes, return its name
            IF ( Get_Block_Property(curblk, STATUS) IN('CHANGED','INSERT') ) THEN
              RETURN curblk;
            ELSE
              -- No changes, recursively look for changed blocks below
              retblk := First_Changed_Block_Below(curblk);
              -- If some block below is changed, return its name
              IF retblk IS NOT NULL THEN
                RETURN retblk;
              ELSE
                -- Consider the next relation
                currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
              END IF;
```

```
      END IF;
    END LOOP;
    -- No changed blocks were found
    RETURN NULL;
  END First_Changed_Block_Below;

BEGIN
  -- Init Local Vars
  mastblk  := :System.Master_Block;
  coordop  := :System.Coordination_Operation;
  trigblk  := :System.Trigger_Block;
  startitm := :System.Trigger_Item;
  frmstat  := :System.Form_Status;
  -- If the coordination operation is anything but CLEAR_RECORD or
  -- SYNCHRONIZE_BLOCKS, then continue checking.
  IF coordop NOT IN ('CLEAR_RECORD', 'SYNCHRONIZE_BLOCKS') THEN
    -- If we're processing the driving master block...
    IF mastblk = trigblk THEN
      -- If something in the form is changed, find the
      -- first changed block below the master
      IF frmstat = 'CHANGED' THEN
        curblk := First_Changed_Block_Below(mastblk);
        -- If we find a changed block below, go there
        -- and Ask to commit the changes.
        IF curblk IS NOT NULL THEN
          Go_Block(curblk);
          Check_Package_Failure;
          Clear_Block(ASK_COMMIT);
          --
          -- If user cancels commit dialog, raise error
          --
          IF NOT ( :System.Form_Status = 'QUERY'
                 OR :System.Block_Status = 'NEW' ) THEN
            RAISE Form_Trigger_Failure;
          END IF;
        END IF;
      END IF;
    END IF;
  END IF;
  -- Clear all the detail blocks for this master without
  -- any further asking to commit.
  currel := Get_Block_Property(trigblk, FIRST_MASTER_RELATION);
  WHILE currel IS NOT NULL LOOP
    curdtl := Get_Relation_Property(currel, DETAIL_NAME);
    IF Get_Block_Property(curdtl, STATUS) <> 'NEW'  THEN
      Go_Block(curdtl);
      Check_Package_Failure;
      Clear_Block(NO_VALIDATE);
```

```
          IF :System.Block_Status <> 'NEW' THEN
            RAISE Form_Trigger_Failure;
          END IF;
        END IF;
        currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
      END LOOP;
      -- Put cursor back where it started
      IF :System.Cursor_Item <> startitm THEN
        Go_Item(startitm);
        Check_Package_Failure;
      END IF;

    EXCEPTION
      WHEN Form_Trigger_Failure THEN
        IF :System.Cursor_Item <> startitm THEN
          Go_Item(startitm);
        END IF;
        RAISE;

    END Clear_All_Master_Details;

  COMMIT_MUDO (Procedure Body)
    PROCEDURE commit_mudo IS
    rlevel varchar2(30);
    BEGIN
      rlevel := :system.message_level;
      :system.message_level := '25';
      commit;
      :system.message_level := rlevel;
    END;

  CREATE_PACK* (Procedure Body)
    PROCEDURE create_pack IS
    BEGIN

pkg_sys.create_obj(:variables_globales.nombre,:variables_globales.especificacion,'PACKAG
E',null);
      EXCEPTION
      WHEN others THEN
          msg1('Error al crear pack de variables globales: '||:variables_globales.nombre||':
'||SQLERRM);
          raise form_trigger_failure;
    END;

  DELREC* (Procedure Body)
    PROCEDURE delrec IS
    ret number;
    begin
```

```
    IF get_permiso_borrar THEN
      go_block('VARIABLES_GLOBALES');
      drop_pack;
      begin
        delete_record;
        commit_mudo;
        EXCEPTION
        WHEN others THEN
            message('Error al borrar registro de variables globales: '||SQLERRM);
            message('Error al borrar registro de variables globales: '||SQLERRM);
            raise form_trigger_failure;
      end;
      execute_query;
    END IF;
  end;
```

DROP_PACK* (Procedure Body)

```
  PROCEDURE drop_pack IS
  -- procedimiento que borra pack de variables globales
  BEGIN
    pkg_sys.drop_obj(:variables_globales.nombre,'PACKAGE');
    EXCEPTION
    WHEN others THEN
        message('Error al borrar accion: '||SQLERRM);
        message('Error al borrar accion: '||SQLERRM);
        raise form_trigger_failure;
  END;
```

GET_PERMISO_BORRAR (Function Body)

```
  FUNCTION get_permiso_borrar RETURN boolean IS
  ret number;
  BEGIN
    if :variables_globales.nombre is null THEN
      return(FALSE);
    end if;
    ret := msg2('Está Seguro(a) de borrar este registro?');
    if ret = 1 THEN
      return(TRUE);
    else
      return(FALSE);
    end if;
  END;
```

K_COMMIT* (Procedure Body)

```
  PROCEDURE k_commit IS
  BEGIN
    IF :variables_globales.nombre is not null THEN
       go_block('VARIABLES_GLOBALES');
```

```
            create_pack;
            commit;
            go_block('USER_ERRORS');
            execute_query;
            go_block('VARIABLES_GLOBALES');
            :variables_globales.status :=
pkg_sys.get_status(:variables_globales.nombre,'PACKAGE');
          END IF;
        END;
```

MSG1 (Procedure Body)

```
        PROCEDURE msg1(p_msg in varchar2) IS
        v_resp number;
        BEGIN
          Set_Alert_Property('UNA_VIA',alert_message_text,p_msg);
          v_resp := show_alert('UNA_VIA');
        END;
```

MSG2 (Function Body)

```
        FUNCTION msg2(p_msg in varchar2) return number IS
        v_resp number;
        BEGIN
          Set_Alert_Property('DOS_VIAS',alert_message_text,p_msg);
          v_resp := show_alert('DOS_VIAS');
          if v_resp = ALERT_BUTTON1 THEN
            return(1);
          else
            return(2);
          end if;
        END;
```

QUERY_MASTER_DETAILS (Procedure Body)

```
        PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
          oldmsg CHAR(2);  -- Old Message Level Setting
          reldef CHAR(5);  -- Relation Deferred Setting
        BEGIN
          -- Initialize Local Variable(s)
          reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
          oldmsg := :System.Message_Level;
          -- If NOT Deferred, Goto detail and execute the query.
          IF reldef = 'FALSE' THEN
            Go_Block(detail);
            Check_Package_Failure;
            :System.Message_Level := '10';
            Execute_Query;
            :System.Message_Level := oldmsg;
          ELSE
            -- Relation is deferred, mark the detail block as un-coordinated
```

```
    Set_Block_Property(detail, COORDINATION_STATUS, NON_COORDINATED);
  END IF;

EXCEPTION
  WHEN Form_Trigger_Failure THEN
    :System.Message_Level := oldmsg;
    RAISE;
END Query_Master_Details;
```

Form Pruebas

```
Triggers
    Name                        ON-CHECK-DELETE-MASTER
    Class                       <Null>
    Trigger Text
      -- Begin default relation declare section
      DECLARE
        Dummy_Define CHAR(1);
        -- Begin SANCIONES detail declare section
        CURSOR SANCIONES_cur IS
          SELECT 1 FROM sanciones
          WHERE RUT = :RESERVAS.RUT;
        -- End SANCIONES detail declare section
      -- End default relation declare section
     -- Begin default relation program section

      BEGIN
        -- Begin SANCIONES detail program section
        OPEN SANCIONES_cur;
        FETCH SANCIONES_cur INTO Dummy_Define;
        IF ( SANCIONES_cur%found ) THEN
          Message('Cannot delete master record when matching detail records exist.');
          CLOSE SANCIONES_cur;
          RAISE Form_Trigger_Failure;
        END IF;
        CLOSE SANCIONES_cur;
        -- End SANCIONES detail program section
      END;
      End default relation program section

Name                        ON-POPULATE-DETAILS
    Class                       <Null>
    Trigger Text
      -- Begin default relation declare section
      DECLARE
        recstat     CHAR(20) := :System.record_status;
        startitm    CHAR(61) := :System.cursor_item;
```

```
      rel_id      Relation;
    -- End default relation declare section
    -- Begin default relation program section
    BEGIN
      IF ( recstat = 'NEW' or recstat = 'INSERT' ) THEN
        RETURN;
      END IF;
      -- Begin SANCIONES detail program section
      IF ( (:RESERVAS.RUT is not null) ) THEN
        rel_id := Find_Relation('RESERVAS.RES_SANC');
        Query_Master_Details(rel_id, 'SANCIONES');
      END IF;
      -- End SANCIONES detail program section

      IF ( :System.cursor_item <> startitm ) THEN
         Go_Item(startitm);
         Check_Package_Failure;
      END IF;
    END;
    End default relation program section

Program Units
    CHECK_PACKAGE_FAILURE (Procedure Body)
      Procedure Check_Package_Failure IS
      BEGIN
        IF NOT ( Form_Success ) THEN
          RAISE Form_Trigger_Failure;
        END IF;
      END;

    CLEAR_ALL_MASTER_DETAILS (Procedure Body)
      PROCEDURE Clear_All_Master_Details IS
        mastblk  CHAR(30);  -- Initial Master Block Cusing Coord
        coordop  CHAR(30);  -- Operation Causing the Coord
        trigblk  CHAR(30);  -- Cur Block On-Clear-Details Fires On
        startitm CHAR(61);  -- Item in which cursor started
        frmstat  CHAR(15);  -- Form Status
        curblk   CHAR(30);  -- Current Block
        currel   CHAR(30);  -- Current Relation
        curdtl   CHAR(30);  -- Current Detail Block

        FUNCTION First_Changed_Block_Below(Master CHAR)
        RETURN CHAR IS
          curblk CHAR(30);  -- Current Block
          currel CHAR(30);  -- Current Relation
          retblk CHAR(30);  -- Return Block
        BEGIN
          -- Initialize Local Vars
```

```
    curblk := Master;
    currel := Get_Block_Property(curblk,  FIRST_MASTER_RELATION);
     -- While there exists another relation for this block
    WHILE currel IS NOT NULL LOOP
      -- Get the name of the detail block
      curblk := Get_Relation_Property(currel, DETAIL_NAME);
      -- If this block has changes, return its name
      IF ( Get_Block_Property(curblk, STATUS) IN('CHANGED','INSERT') ) THEN
        RETURN curblk;
      ELSE
        -- No changes, recursively look for changed blocks below
        retblk := First_Changed_Block_Below(curblk);
        -- If some block below is changed, return its name
        IF retblk IS NOT NULL THEN
          RETURN retblk;
        ELSE
          -- Consider the next relation
          currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
        END IF;
      END IF;
    END LOOP;
    -- No changed blocks were found

    RETURN NULL;
  END First_Changed_Block_Below;

BEGIN
  -- Init Local Vars
  mastblk  := :System.Master_Block;
  coordop  := :System.Coordination_Operation;
  trigblk  := :System.Trigger_Block;
  startitm := :System.Trigger_Item;
  frmstat  := :System.Form_Status;
  -- If the coordination operation is anything but CLEAR_RECORD or
  -- SYNCHRONIZE_BLOCKS, then continue checking.
  IF coordop NOT IN ('CLEAR_RECORD', 'SYNCHRONIZE_BLOCKS') THEN
    -- If we're processing the driving master block...
    IF mastblk = trigblk THEN
      -- If something in the form is changed, find the
      -- first changed block below the master
      IF frmstat = 'CHANGED' THEN
        curblk := First_Changed_Block_Below(mastblk);
        -- If we find a changed block below, go there
        -- and Ask to commit the changes.
        IF curblk IS NOT NULL THEN
          Go_Block(curblk);
          Check_Package_Failure;
          Clear_Block(ASK_COMMIT);
```

```
          -- If user cancels commit dialog, raise error
          IF NOT ( :System.Form_Status = 'QUERY'
                 OR :System.Block_Status = 'NEW' ) THEN
            RAISE Form_Trigger_Failure;
          END IF;
        END IF;
      END IF;
    END IF;
  END IF;
  -- Clear all the detail blocks for this master without
  -- any further asking to commit.
  currel := Get_Block_Property(trigblk, FIRST_MASTER_RELATION);
  WHILE currel IS NOT NULL LOOP
    curdtl := Get_Relation_Property(currel, DETAIL_NAME);
    IF Get_Block_Property(curdtl, STATUS) <> 'NEW'  THEN
      Go_Block(curdtl);
      Check_Package_Failure;
      Clear_Block(NO_VALIDATE);
      IF :System.Block_Status <> 'NEW' THEN
        RAISE Form_Trigger_Failure;
      END IF;
    END IF;
    currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
  END LOOP;
  -- Put cursor back where it started
  IF :System.Cursor_Item <> startitm THEN
    Go_Item(startitm);
    Check_Package_Failure;
  END IF;

EXCEPTION
  WHEN Form_Trigger_Failure THEN
    IF :System.Cursor_Item <> startitm THEN
      Go_Item(startitm);
    END IF;
    RAISE;

END Clear_All_Master_Details;

QUERY_MASTER_DETAILS (Procedure Body)
PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
  oldmsg CHAR(2);  -- Old Message Level Setting
  reldef CHAR(5);  -- Relation Deferred Setting
BEGIN
  --
  -- Initialize Local Variable(s)
  --
  reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
```

```
    oldmsg := :System.Message_Level;
    --
    -- If NOT Deferred, Goto detail and execute the query.
    --
    IF reldef = 'FALSE' THEN
      Go_Block(detail);
      Check_Package_Failure;
      :System.Message_Level := '10';
      Execute_Query;
      :System.Message_Level := oldmsg;
    ELSE
      --
      -- Relation is deferred, mark the detail block as un-coordinated
      --
      Set_Block_Property(detail, COORDINATION_STATUS, NON_COORDINATED);
    END IF;

  EXCEPTION
    WHEN Form_Trigger_Failure THEN
      :System.Message_Level := oldmsg;
      RAISE;
  END Query_Master_Details;
```

FORM TABLAS

```
Name                          ON-POPULATE-DETAILS
    Class                           <Null>
    Trigger Text
      -- Begin default relation declare section
      DECLARE
        recstat     CHAR(20) := :System.record_status;
        startitm    CHAR(61) := :System.cursor_item;
        rel_id      Relation;
      -- End default relation declare section
      -- Begin default relation program section
      BEGIN
        IF ( recstat = 'NEW' or recstat = 'INSERT' ) THEN
          RETURN;
        END IF;
        -- Begin COLUMNAS_TABLAS detail program section
        IF ( (:TABLAS.TABLA is not null) ) THEN
          rel_id := Find_Relation('TABLAS.TABLAS_COLUMNAS_TABLAS');
          Query_Master_Details(rel_id, 'COLUMNAS_TABLAS');
        END IF;
        -- End COLUMNAS_TABLAS detail program section

        IF ( :System.cursor_item <> startitm ) THEN
           Go_Item(startitm);
```

```
            Check_Package_Failure;
          END IF;
         END;
         -- End default relation program section
Triggers
          Name                        POST-CHANGE
          Class                      <Null>
          Trigger Text
            if :system.record_status in ('NEW','INSERT') THEN
              if pkg_sys.existe_obj(:tablas.tabla) THEN
                message('Ya existe objeto '||:tablas.tabla);
                message('Ya existe objeto '||:tablas.tabla);
                raise form_trigger_failure;
              end if;
            end if;
            :tablas.tab_status := nvl(pkg_sys.get_status(:tablas.tabla,'TABLE'),'INVALID');
            :tablas.pack_status :=
nvl(pkg_sys.get_status('G_'||:tablas.tabla,'PACKAGE'),'INVALID');
Triggers
          Name                        WHEN-LIST-CHANGED
          Class                      <Null>
          Trigger Text
            if :columnas_tablas.tipo = 'VARCHAR2' THEN
               set_item_property('columnas_tablas.tamano',ENABLED,PROPERTY_TRUE);
            else
               set_item_property('columnas_tablas.tamano',ENABLED,PROPERTY_FALSE);
            end if;

Program Units
     CHECK_PACKAGE_FAILURE (Procedure Body)
       Procedure Check_Package_Failure IS
       BEGIN
         IF NOT ( Form_Success ) THEN
           RAISE Form_Trigger_Failure;
         END IF;
       END;

     CLEAR_ALL_MASTER_DETAILS (Procedure Body)
       PROCEDURE Clear_All_Master_Details IS
         mastblk  CHAR(30);  -- Initial Master Block Cusing Coord
         coordop  CHAR(30);  -- Operation Causing the Coord
         trigblk  CHAR(30);  -- Cur Block On-Clear-Details Fires On
         startitm CHAR(61);  -- Item in which cursor started
         frmstat  CHAR(15);  -- Form Status
         curblk   CHAR(30);  -- Current Block
         currel   CHAR(30);  -- Current Relation
         curdtl   CHAR(30);  -- Current Detail Block
```

```
  FUNCTION First_Changed_Block_Below(Master CHAR)
  RETURN CHAR IS
    curblk CHAR(30);  -- Current Block
    currel CHAR(30);  -- Current Relation
    retblk CHAR(30);  -- Return Block
  BEGIN
    -- Initialize Local Vars
    curblk := Master;
    currel := Get_Block_Property(curblk,  FIRST_MASTER_RELATION);
    -- While there exists another relation for this block
   WHILE currel IS NOT NULL LOOP
      -- Get the name of the detail block
      curblk := Get_Relation_Property(currel, DETAIL_NAME);
      -- If this block has changes, return its name
      IF ( Get_Block_Property(curblk, STATUS) IN('CHANGED','INSERT') ) THEN
        RETURN curblk;
      ELSE
        -- No changes, recursively look for changed blocks below
        retblk := First_Changed_Block_Below(curblk);
        -- If some block below is changed, return its name
        IF retblk IS NOT NULL THEN
          RETURN retblk;
        ELSE
          -- Consider the next relation
          --
          currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
        END IF;
      END IF;
    END LOOP;
    -- No changed blocks were found
    RETURN NULL;
  END First_Changed_Block_Below;

BEGIN
  -- Init Local Vars
  mastblk  := :System.Master_Block;
  coordop  := :System.Coordination_Operation;
  trigblk  := :System.Trigger_Block;
  startitm := :System.Trigger_Item;
  frmstat  := :System.Form_Status;
  -- If the coordination operation is anything but CLEAR_RECORD or
  -- SYNCHRONIZE_BLOCKS, then continue checking.
  IF coordop NOT IN ('CLEAR_RECORD', 'SYNCHRONIZE_BLOCKS') THEN
    -- If we're processing the driving master block...
    IF mastblk = trigblk THEN
      -- If something in the form is changed, find the
      -- first changed block below the master
      IF frmstat = 'CHANGED' THEN
```

```
      curblk := First_Changed_Block_Below(mastblk);
      -- If we find a changed block below, go there
      -- and Ask to commit the changes.
      IF curblk IS NOT NULL THEN
        Go_Block(curblk);
        Check_Package_Failure;
        Clear_Block(ASK_COMMIT);
        -- If user cancels commit dialog, raise error
        IF NOT ( :System.Form_Status = 'QUERY'
                OR :System.Block_Status = 'NEW' ) THEN
          RAISE Form_Trigger_Failure;
        END IF;
      END IF;
    END IF;
  END IF;
END IF;

--
-- Clear all the detail blocks for this master without
-- any further asking to commit.
--
currel := Get_Block_Property(trigblk, FIRST_MASTER_RELATION);
WHILE currel IS NOT NULL LOOP
  curdtl := Get_Relation_Property(currel, DETAIL_NAME);
  IF Get_Block_Property(curdtl, STATUS) <> 'NEW'  THEN
    Go_Block(curdtl);
    Check_Package_Failure;
    Clear_Block(NO_VALIDATE);
    IF :System.Block_Status <> 'NEW' THEN
      RAISE Form_Trigger_Failure;
    END IF;
  END IF;
  currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
END LOOP;

-- Put cursor back where it started
IF :System.Cursor_Item <> startitm THEN
  Go_Item(startitm);
  Check_Package_Failure;
END IF;

EXCEPTION
  WHEN Form_Trigger_Failure THEN
    IF :System.Cursor_Item <> startitm THEN
      Go_Item(startitm);
    END IF;
    RAISE;
```

```
END Clear_All_Master_Details;

COMMIT_MUDO (Procedure Body)
  PROCEDURE commit_mudo IS
  r_level varchar2(5);
  BEGIN
    r_level := :system.message_level;
    :system.message_level := '25';
    commit;
    :system.message_level := r_level;
  END;

CREATE_TABLE (Procedure Body)
  PROCEDURE create_table IS
  BEGIN
    if get_permiso_crear THEN
      commit_mudo;
      pkg_sys.create_tab(:tablas.tabla);
      pkg_sys.create_pack_tab(:tablas.tabla);
      :tablas.tab_status := nvl(pkg_sys.get_status(:tablas.tabla,'TABLE'),'INVALID');
      :tablas.pack_status :=
nvl(pkg_sys.get_status('G_'||:tablas.tabla,'PACKAGE'),'INVALID');
    end if;
    EXCEPTION
    WHEN others THEN
        message('Error al crear tabla: '||SQLERRM);
        message('Error al crear tabla: '||SQLERRM);
        raise form_trigger_failure;
  END;

DELREC (Procedure Body)
  PROCEDURE delrec IS
  ret number;
  begin
    IF get_permiso_borrar THEN
      drop_table;
      begin
        delete_record;
        commit_mudo;
        EXCEPTION
        WHEN others THEN
            message('Error al borrar geistro de tabla: '||SQLERRM);
            message('Error al borrar geistro de tabla: '||SQLERRM);
            raise form_trigger_failure;
      end;
      execute_query;
    END IF;
  end;
```

DROP_TABLE (Procedure Body)

```
PROCEDURE drop_table IS
BEGIN
  pkg_sys.drop_obj(:tablas.tabla,'TABLE');
  pkg_sys.drop_obj('G_'||:tablas.tabla,'PACKAGE');
  EXCEPTION
  WHEN others THEN
      message('Error al borrar tabla: '||SQLERRM);
      message('Error al borrar tabla: '||SQLERRM);
      raise form_trigger_failure;
END;
```

GET_PERMISO_BORRAR (Function Body)

```
FUNCTION get_permiso_borrar RETURN boolean IS
ret number;
BEGIN
  if :tablas.tabla is null THEN
    return(FALSE);
  end if;
  ret := msg2('Está Seguro(a) de Borrar la tabla?');
  if ret = 1 THEN
    return(TRUE);
  else
    return(FALSE);
  end if;
END;
```

GET_PERMISO_CREAR (Function Body)

```
FUNCTION get_permiso_crear RETURN boolean IS
ret number;
BEGIN
  if :tablas.tabla is null THEN
    return(FALSE);
  end if;
  ret := msg2('Está seguro(a) de Crear la tabla?');
  if ret = 1 THEN
    return(TRUE);
  else
    return(FALSE);
  end if;
END;
```

K_COMMIT (Procedure Body)

```
PROCEDURE k_commit IS
BEGIN
  commit;
  :tablas.tab_status := nvl(pkg_sys.get_status(:tablas.tabla,'TABLE'),'INVALID');
```

```
        :tablas.pack_status :=
nvl(pkg_sys.get_status('G_'||:tablas.tabla,'PACKAGE'),'INVALID');
      END;

    MSG1 (Procedure Body)
      PROCEDURE msg1(p_msg in varchar2) IS
      v_resp number;
      BEGIN
        Set_Alert_Property('UNA_VIA',alert_message_text,p_msg);
        v_resp := show_alert('UNA_VIA');
      END;

    MSG2 (Function Body)
      FUNCTION msg2(p_msg in varchar2) return number IS
      v_resp number;
      BEGIN
        Set_Alert_Property('DOS_VIAS',alert_message_text,p_msg);
        v_resp := show_alert('DOS_VIAS');
        if v_resp = ALERT_BUTTON1 THEN
          return(1);
        else
          return(2);
        end if;
      END;

    QUERY_MASTER_DETAILS (Procedure Body)
      PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
        oldmsg CHAR(2);  -- Old Message Level Setting
        reldef CHAR(5);  -- Relation Deferred Setting
      BEGIN
        -- Initialize Local Variable(s)
        reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
        oldmsg := :System.Message_Level;
        -- If NOT Deferred, Goto detail and execute the query.
        IF reldef = 'FALSE' THEN
          Go_Block(detail);
          Check_Package_Failure;
          :System.Message_Level := '10';
          Execute_Query;
          :System.Message_Level := oldmsg;
        ELSE
          -- Relation is deferred, mark the detail block as un-coordinated
          Set_Block_Property(detail, COORDINATION_STATUS, NON_COORDINATED);
        END IF;

      EXCEPTION
          WHEN Form_Trigger_Failure THEN
            :System.Message_Level := oldmsg;
```

```
    RAISE;
  END Query_Master_Details;
```

FORM VARIABLES

```
Triggers
    Name                    ON-CHECK-DELETE-MASTER
    Class                   <Null>
    Trigger Text
      -- Begin default relation declare section
      DECLARE
        Dummy_Define CHAR(1);
        -- Begin USER_ERRORS detail declare section
        CURSOR USER_ERRORS_cur IS
          SELECT 1 FROM user_errors
          WHERE NAME = :VARIABLES.VARIABLE;
        -- End USER_ERRORS detail declare section
      -- End default relation declare section
      -- Begin default relation program section
      BEGIN
        -- Begin USER_ERRORS detail program section
        OPEN USER_ERRORS_cur;
        FETCH USER_ERRORS_cur INTO Dummy_Define;
        IF ( USER_ERRORS_cur%found ) THEN
          Message('Cannot delete master record when matching detail records exist.');
          CLOSE USER_ERRORS_cur;
          RAISE Form_Trigger_Failure;
        END IF;
        CLOSE USER_ERRORS_cur;
        -- End USER_ERRORS detail program section
      END;
      -- End default relation program section
Name                    ON-POPULATE-DETAILS
    Class                   <Null>
    Trigger Text
      -- Begin default relation declare section
      DECLARE
        recstat     CHAR(20) := :System.record_status;
        startitm    CHAR(61) := :System.cursor_item;
        rel_id      Relation;
      -- End default relation declare section
      -- Begin default relation program section
      BEGIN
```

```
        IF ( recstat = 'NEW' or recstat = 'INSERT' ) THEN
          RETURN;
        END IF;
        -- Begin USER_ERRORS detail program section
        IF ( (:VARIABLES.VARIABLE is not null) ) THEN
          rel_id := Find_Relation('VARIABLES.RELATION10');
          Query_Master_Details(rel_id, 'USER_ERRORS');
        END IF;
        -- End USER_ERRORS detail program section
        IF ( :System.cursor_item <> startitm ) THEN
           Go_Item(startitm);
           Check_Package_Failure;
        END IF;
      END;
      End default relation program section

Triggers
        Name                          POST-CHANGE
        Class                        <Null>
        Trigger Text
          if :system.record_status in ('NEW','INSERT') THEN
            if pkg_sys.existe_obj(:variables.variable) THEN
              message('Ya existe objeto '||:variables.variable);
              message('Ya existe objeto '||:variables.variable);
              raise form_trigger_failure;
            end if;
          end if;
          :variables.status := pkg_sys.get_status(:variables.variable,'FUNCTION');

Program Units
    CHECK_PACKAGE_FAILURE (Procedure Body)
      Procedure Check_Package_Failure IS
      BEGIN
        IF NOT ( Form_Success ) THEN
          RAISE Form_Trigger_Failure;
        END IF;
      END;

    CLEAR_ALL_MASTER_DETAILS (Procedure Body)
      PROCEDURE Clear_All_Master_Details IS
        mastblk  CHAR(30);  -- Initial Master Block Cusing Coord
        coordop  CHAR(30);  -- Operation Causing the Coord
        trigblk  CHAR(30);  -- Cur Block On-Clear-Details Fires On
        startitm CHAR(61);  -- Item in which cursor started
        frmstat  CHAR(15);  -- Form Status
        curblk   CHAR(30);  -- Current Block
        currel   CHAR(30);  -- Current Relation
        curdtl   CHAR(30);  -- Current Detail Block
```

```
  FUNCTION First_Changed_Block_Below(Master CHAR)
  RETURN CHAR IS
    curblk CHAR(30);  -- Current Block
    currel CHAR(30);  -- Current Relation
    retblk CHAR(30);  -- Return Block
  BEGIN
    -- Initialize Local Vars
    curblk := Master;
    currel := Get_Block_Property(curblk,  FIRST_MASTER_RELATION);
    -- While there exists another relation for this block
    WHILE currel IS NOT NULL LOOP
      -- Get the name of the detail block
      curblk := Get_Relation_Property(currel, DETAIL_NAME);
      -- If this block has changes, return its name
      IF ( Get_Block_Property(curblk, STATUS) IN('CHANGED','INSERT') ) THEN
        RETURN curblk;
      ELSE
        -- No changes, recursively look for changed blocks below
        retblk := First_Changed_Block_Below(curblk);
        -- If some block below is changed, return its name
        IF retblk IS NOT NULL THEN
          RETURN retblk;
        ELSE
          -- Consider the next relation
          currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
        END IF;
      END IF;
    END LOOP;
    -- No changed blocks were found
    RETURN NULL;
  END First_Changed_Block_Below;

BEGIN
  -- Init Local Vars
  mastblk  := :System.Master_Block;
  coordop  := :System.Coordination_Operation;
  trigblk  := :System.Trigger_Block;
  startitm := :System.Trigger_Item;
  frmstat  := :System.Form_Status;

  -- If the coordination operation is anything but CLEAR_RECORD or
  -- SYNCHRONIZE_BLOCKS, then continue checking.
  IF coordop NOT IN ('CLEAR_RECORD', 'SYNCHRONIZE_BLOCKS') THEN
    -- If we're processing the driving master block...
    IF mastblk = trigblk THEN
      -- If something in the form is changed, find the
      -- first changed block below the master
```

```
      IF frmstat = 'CHANGED' THEN
        curblk := First_Changed_Block_Below(mastblk);
        -- If we find a changed block below, go there
        -- and Ask to commit the changes.
        IF curblk IS NOT NULL THEN
          Go_Block(curblk);
          Check_Package_Failure;
          Clear_Block(ASK_COMMIT);
          -- If user cancels commit dialog, raise error
          IF NOT ( :System.Form_Status = 'QUERY'
                 OR :System.Block_Status = 'NEW' ) THEN
            RAISE Form_Trigger_Failure;
          END IF;
        END IF;
      END IF;
    END IF;
  END IF;

  -- Clear all the detail blocks for this master without
  -- any further asking to commit.
  currel := Get_Block_Property(trigblk, FIRST_MASTER_RELATION);
  WHILE currel IS NOT NULL LOOP
    curdtl := Get_Relation_Property(currel, DETAIL_NAME);
    IF Get_Block_Property(curdtl, STATUS) <> 'NEW'  THEN
      Go_Block(curdtl);
      Check_Package_Failure;
      Clear_Block(NO_VALIDATE);
      IF :System.Block_Status <> 'NEW' THEN
        RAISE Form_Trigger_Failure;
      END IF;
    END IF;
    currel := Get_Relation_Property(currel, NEXT_MASTER_RELATION);
  END LOOP;

  -- Put cursor back where it started
  IF :System.Cursor_Item <> startitm THEN
    Go_Item(startitm);
    Check_Package_Failure;
  END IF;

EXCEPTION
  WHEN Form_Trigger_Failure THEN
    IF :System.Cursor_Item <> startitm THEN
      Go_Item(startitm);
    END IF;
    RAISE;

END Clear_All_Master_Details;
```

COMMIT_MUDO (Procedure Body)

```
PROCEDURE commit_mudo IS
rlevel varchar2(30);
BEGIN
  rlevel := :system.message_level;
  :system.message_level := '25';
  commit;
  :system.message_level := rlevel;
END;
```

CREATE_VARIABLE (Procedure Body)

```
PROCEDURE create_variable IS
BEGIN

pkg_sys.create_obj(:variables.variable,:VARIABLES.CONSULTA_SQL,'FUNCTION',:varia
bles.tipo_variable);
  EXCEPTION
  WHEN others THEN
     message('Error al crear variable: '||:variables.variable||': '||SQLERRM);
     message('Error al crear variable: '||:variables.variable||': '||SQLERRM);
     raise form_trigger_failure;
END;
```

DELREC (Procedure Body)

```
PROCEDURE delrec IS
ret number;
begin
  IF get_permiso_borrar THEN
    go_block('VARIABLES');
    drop_variable;
    begin
      delete_record;
      commit_mudo;
      EXCEPTION
      WHEN others THEN
          message('Error al borrar registro de variable: '||SQLERRM);
          message('Error al borrar registro de variable: '||SQLERRM);
          raise form_trigger_failure;
    end;
    execute_query;
  END IF;
end;
```

DROP_VARIABLE (Procedure Body)

```
PROCEDURE drop_variable IS
-- procedimiento que borra la variable
BEGIN
```

```
    pkg_sys.drop_obj(:variables.variable,'FUNCTION');
    EXCEPTION
    WHEN others THEN
        message('Error al borrar accion: '||SQLERRM);
        message('Error al borrar accion: '||SQLERRM);
        raise form_trigger_failure;
  END;
```

GET_PERMISO_BORRAR (Function Body)

```
  FUNCTION get_permiso_borrar RETURN boolean IS
  ret number;
  BEGIN
    if :variables.variable is null THEN
      return(FALSE);
    end if;
    ret := show_alert('Está Seguro(a) de borrar este registro?');
    if ret = 1 THEN
      return(TRUE);
    else
      return(FALSE);
    end if;
  END;
```

K_COMMIT (Procedure Body)

```
  PROCEDURE k_commit IS
  BEGIN
    IF :variables.variable is not null THEN
        go_block('VARIABLES');
        commit;
        create_variable;
        commit_mudo;
        go_block('USER_ERRORS');
        execute_query;
        go_block('VARIABLES');
        :variables.status := pkg_sys.get_status(:variables.variable,'FUNCTION');
    END IF;
  END;
```

MSG (Procedure Body)

```
  PROCEDURE MSG(p_msg in varchar2) IS
  BEGIN
    message(p_msg,NO_ACKNOWLEDGE);
  END;
```

MSG1 (Procedure Body)

```
  PROCEDURE msg1(p_msg in varchar2) IS
  v_resp number;
  BEGIN
```

```
    Set_Alert_Property('UNA_VIA',alert_message_text,p_msg);
    v_resp := show_alert('UNA_VIA');
  END;
```

MSG2 (Function Body)

```
  FUNCTION msg2(p_msg in varchar2) return number IS
  v_resp number;
  BEGIN
    Set_Alert_Property('DOS_VIAS',alert_message_text,p_msg);
    v_resp := show_alert('DOS_VIAS');
    if v_resp = ALERT_BUTTON1 THEN
      return(1);
    else
      return(2);
    end if;
  END;
```

QUERY_MASTER_DETAILS (Procedure Body)

```
  PROCEDURE Query_Master_Details(rel_id Relation,detail CHAR) IS
   oldmsg CHAR(2);  -- Old Message Level Setting
   reldef CHAR(5);  -- Relation Deferred Setting
  BEGIN
   -- Initialize Local Variable(s)
   reldef := Get_Relation_Property(rel_id, DEFERRED_COORDINATION);
   oldmsg := :System.Message_Level;
   -- If NOT Deferred, Goto detail and execute the query.
   IF reldef = 'FALSE' THEN
     Go_Block(detail);
     Check_Package_Failure;
     :System.Message_Level := '10';
     Execute_Query;
     :System.Message_Level := oldmsg;
   ELSE
     -- Relation is deferred, mark the detail block as un-coordinated
     Set_Block_Property(detail, COORDINATION_STATUS, NON_COORDINATED);
   END IF;

  EXCEPTION
     WHEN Form_Trigger_Failure THEN
       :System.Message_Level := oldmsg;
       RAISE;
  END Query_Master_Details;
```

Anexo C: Código de Programas

```
clear screen
--
-- script de creación de objetos Seminario "Base de datos Activas"
-- Autoras: Sandra Macaya, Julia Cáceres
-- Prof Guía: Francisco Venegas
-- 05/2001
--
-- Eliminación de claves foráneas
--
alter table condiciones_eventos drop constraint fk_evento_condiciones
/
alter table condiciones_eventos drop constraint fk_condicion_condiciones
/
alter table acciones_eventos drop constraint fk_evento_acciones
/
alter table acciones_eventos drop constraint fk_accion_acciones
/
alter table bitacora_procesos drop constraint fk_evento_bitacora
/
alter table privilegios drop constraint fk_evento_privilegios
/
--
-- Tablas
--
drop table eventos
/
create table eventos (
evento varchar2(30) not null,
descripcion varchar2(100) not null,
tipo varchar2(15) not null,          -- tabla / temporal
tabla varchar2(30),                  -- nombre de tabla
partida date,                        -- fecha y hora de la primera ejecucion
interval_dd number,                  -- periodicidad de ejecucion en dias
interval_hh number,                  -- periodicidad de ejecucion en horas
interval_mm number,                  -- periodicidad de ejecucion en minutos
interval_ss number,                  -- periodicidad de ejecucion en segundos
triggering varchar2(10),             -- before /after
sentencia varchar2(10),              -- insert, update, delete
each_row varchar2(1),                -- 'Y' or 'N'
condicion varchar2(200),             -- condicion activacion
activo varchar2(1) not null,         -- registro habilitado (S/N)
job_no number                        -- nro de job
)
/
drop table condiciones
```

```
/
create table condiciones (
condicion varchar2(30) not null,
descripcion varchar2(100) not null,
expresion varchar2(2000) not null,
activo varchar2(1) not null                -- registro activo (S/N)
)
/
drop table acciones
/
create table acciones (
accion varchar2(30) not null,
descripcion varchar2(100) not null,
codigo_pl_java varchar2(2000) not null,
activo varchar2(1) not null              -- registro habilitado (S/N)
)
/
drop table condiciones_eventos
/
create table condiciones_eventos (
evento varchar2(30) not null,
condicion varchar2(30) not null,
activo varchar2(1) not null,         -- registro activo (S/N)
prioridad number not null
)
/
drop table acciones_eventos
/
create table acciones_eventos (
evento varchar2(30) not null,
accion varchar2(30) not null,
prioridad number not null,
activo varchar2(1) not null         -- registro activo (S/N)
)
/
drop table variables
/
create table variables (
variable varchar2(30) not null,
descripcion varchar2(100) not null,
tipo_variable varchar2(30) not null,
consulta_sql varchar2(2000) not null
)
/
drop table variables_globales
/
create table variables_globales (
nombre varchar2(30) not null,
```

```
descripcion varchar2(80) not null,
especificacion varchar2(2000) not null
)
/
drop table bitacora_procesos
/
create table bitacora_procesos (
evento varchar2(30) not null,
secuencia number not null,
fecha date not null,
usuario varchar2(30) not null,
status varchar2(30) not null,
glosa varchar2(2000) not null
)
/
drop table privilegios
/
create table privilegios (
evento varchar2(30) not null,
usuario varchar2(30) not null
)
/
drop table tablas
/
create table tablas (
tabla varchar2(30),
descripcion varchar2(80)
)
/
drop table columnas_tablas
/
create table columnas_tablas (
tabla varchar2(30) not null,
orden number not null,
columna varchar2(30) not null,
descripcion varchar2(80) not null,
tipo varchar2(30) not null,
tamano number,
nula varchar2(1) not null
)
/
--
-- primary keys
--
alter table eventos add constraint evento_pk primary key (evento) using index
/
alter table bitacora_procesos add constraint bitacora_pk primary key (secuencia) using
index
```

```
/
alter table condiciones add constraint condiciones_pk primary key (condicion) using
index
/
alter table acciones add constraint acciones_pk primary key (accion) using index
/
alter table privilegios add constraint privilegios_pk primary key (evento,usuario) using
index
/
--
-- foreign keys
--
alter table condiciones_eventos add constraint fk_evento_condiciones foreign key
(evento) references eventos(evento)
/
alter table condiciones_eventos add constraint fk_condicion_condiciones foreign key
(condicion) references condiciones(condicion)
/
alter table acciones_eventos add constraint fk_evento_acciones foreign key (evento)
references eventos(evento)
/
alter table acciones_eventos add constraint fk_accion_acciones foreign key (accion)
references acciones(accion)
/
alter table bitacora_procesos add constraint fk_evento_bitacora foreign key (evento)
references eventos(evento)
/
alter table privilegios add constraint fk_evento_privilegios foreign key (evento)
references eventos(evento)
/
--
-- secuencias
drop sequence sec_proc
/
create sequence sec_proc
/
--
create or replace package pkg_sys as
g_valor_cond boolean;
-----------------------------------------------------------------------------------------
function submit_job(p_proc in varchar2, p_fecha in date, p_interval in varchar2) return
number;
-----------------------------------------------------------------------------------------
procedure execute_eca(p_evento in varchar2);
-----------------------------------------------------------------------------------------
procedure create_tab(p_table in varchar2);
-----------------------------------------------------------------------------------------
procedure create_pack_tab(p_table in varchar2);
```

```
-----------------------------------------------------------------------------------------
function get_new_old_str(p_table in varchar2) return varchar2;
-----------------------------------------------------------------------------------------
procedure execute_acc(p_acc in varchar2);
-----------------------------------------------------------------------------------------
procedure execute_str(p_str in varchar2);
-----------------------------------------------------------------------------------------
function get_cond(p_cond in varchar2) return boolean;
-----------------------------------------------------------------------------------------
function existe_obj(p_obj in varchar2) return boolean;
-----------------------------------------------------------------------------------------
function get_status(p_obj in varchar2, p_type in varchar2) return varchar2;
-----------------------------------------------------------------------------------------
function get_next_date(p_job_no in number) return date;
-----------------------------------------------------------------------------------------
procedure create_obj(p_obj in varchar2, p_cuerpo in varchar2,
                     p_obj_type in varchar2, p_return_type in varchar2);
-----------------------------------------------------------------------------------------
procedure drop_obj(p_obj in varchar2, p_type in varchar2);
-----------------------------------------------------------------------------------------
procedure drop_job(p_job_no in number);
-----------------------------------------------------------------------------------------
procedure enable_trigger(p_trigger in varchar2);
-----------------------------------------------------------------------------------------
procedure disable_trigger(p_trigger in varchar2);
-----------------------------------------------------------------------------------------
procedure active_condicion(p_condicion in varchar2);
-----------------------------------------------------------------------------------------
procedure active_accion(p_accion in varchar2);
-----------------------------------------------------------------------------------------
procedure desactive_condicion(p_condicion in varchar2);
-----------------------------------------------------------------------------------------
procedure desactive_accion(p_accion in varchar2);
-----------------------------------------------------------------------------------------
procedure ins_bitacora(p_evento in varchar2, p_status in varchar2, p_glosa in
varchar2);
-----------------------------------------------------------------------------------------
function get_privilegio(p_evento in varchar2, p_user in varchar2) return boolean;
-----------------------------------------------------------------------------------------
end pkg_sys;
/
create or replace package body pkg_sys as
-----------------------------------------------------------------------------------------
function submit_job(p_proc in varchar2, p_fecha in date, p_interval in varchar2) return
number is
--
-- procedimiento que submite un proceso oracle p_proc, para su ejecucion en la fecha
p_fecha
```

```
-- con un intervalo de ejecucion p_intervalo medido en segundos
--
  v_job number;
begin
  dbms_job.submit(v_job,p_proc,p_fecha,p_interval);
  return(v_job);
  exception
  when others then
      raise_application_error(-20200,' pkg_sys.submit_job: '||SQLERRM);
end submit_job;
-----------------------------------------------------------------------------------------
function get_tipo_evento(p_evento in varchar2) return varchar2 is
--
-- funcion que retorna el tipo de evento (temporal o tabla)
--
salida varchar2(15);
BEGIN
  select tipo
  into salida
  from eventos
  where evento = p_evento;
  return(salida);
  EXCEPTION
  WHEN no_data_found THEN
      return(null);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.get_tipo_evento : evento: '||p_evento||':
'||SQLERRM);
--
END get_tipo_evento;
-----------------------------------------------------------------------------------------
procedure execute_eca(p_evento in varchar2) is
--
-- procedimiento que verifica las condiciones del evento p_evento y si todas son
verdaderas
-- ejecuta las acciones asociadas al evento p_evento, solo si el usuario tiene privilegios
--
  cursor c_cond is
        select condicion
        from condiciones_eventos
        where evento = p_evento
        and activo = 'S'
        order by prioridad;
--
  cursor c_acc is
        select accion
        from acciones_eventos
        where evento = p_evento
```

```
        and activo = 'S'
        order by prioridad;
--
  v_cond boolean;
  v_num_condiciones number;
  v_num_condiciones_err number;
  v_num_no_cumple number;
  v_num_acciones number;
  v_num_acciones_err number;
  v_tipo_evento varchar2(15);
--
begin
--
  ins_bitacora(p_evento,'INICIO','Inicio ejecucion regla ECA');
--
-- verifica privilegios
--
  if get_privilegio(p_evento,user) then
--
--   tipo de evento
--
     v_tipo_evento := get_tipo_evento(p_evento);
--
--   condiciones
--
     v_num_condiciones := 0;
     v_num_no_cumple := 0;
     v_num_condiciones_err := 0;
--
     for i in c_cond loop
        begin
          v_cond := get_cond(i.condicion);
          if v_cond THEN
            v_num_condiciones := v_num_condiciones + 1;
          else
            v_num_no_cumple := v_num_no_cumple + 1;
            ins_bitacora(p_evento,'NO CUMPLE CONDICION',' Condicion:
'||i.condicion);
          end if;
          EXCEPTION
          WHEN others THEN
              v_num_condiciones_err := v_num_no_cumple + 1;
              ins_bitacora(p_evento,'ERROR','Error en condicion: '||i.condicion||':
'||SQLERRM);
        end;
     end loop;
--
     if v_num_no_cumple = 0 and v_num_condiciones_err = 0 THEN
```

```
      if v_num_condiciones = 0 THEN
        ins_bitacora(p_evento,'EN PROCESO','Evento sin condiciones');
      else
        ins_bitacora(p_evento,'EN PROCESO','Cumple: '||v_num_condiciones||', todas
las condiciones');
      end if;
    else
      if v_num_condiciones_err = 0 THEN
        if v_tipo_evento = 'TABLA' THEN
          ins_bitacora(p_evento,'FIN DE PROCESO','No cumple: '||v_num_no_cumple||'
condiciones');
          raise_application_error(-20200,'No cumple: '||v_num_no_cumple||'
condiciones');
        else
          ins_bitacora(p_evento,'FIN DE PROCESO','No cumple: '||v_num_no_cumple||'
condiciones');
          return;
        end if;
      else
        if v_num_no_cumple = 0 THEN
          if v_tipo_evento = 'TABLA' THEN
            ins_bitacora(p_evento,'FIN CON ERROR',v_num_condiciones_err||'
condiciones (todas) erroneas');
            raise_application_error(-20200,'ERROR: '||v_num_condiciones_err||'
condiciones (todas) erroneas');
          else
            ins_bitacora(p_evento,'FIN CON ERROR',v_num_condiciones_err||'
condiciones (todas) erroneas');
            return;
          end if;
        else
          if v_tipo_evento = 'TABLA' THEN
            ins_bitacora(p_evento,'FIN CON ERROR','No se cumplen
'||v_num_no_cumple||
                         'condiciones y '||v_num_condiciones_err||' erroneas');
            raise_application_error(-20200,'ERROR: No se cumplen
'||v_num_no_cumple||
                          'condiciones y '||v_num_condiciones_err||' erroneas');
          else
            ins_bitacora(p_evento,'FIN CON ERROR','No se cumplen
'||v_num_no_cumple||
                         'condiciones y '||v_num_condiciones_err||' erroneas');
            return;
          end if;
        end if;
      end if;
    end if;
--
```

```
--    acciones
--
     v_num_acciones := 0;
     v_num_acciones_err := 0;
     for i in c_acc loop
        begin
          execute_acc(i.accion);
          v_num_acciones := v_num_acciones + 1;
          Exception
          WHEN others THEN
              v_num_acciones_err := v_num_acciones_err + 1;
              ins_bitacora(p_evento,'ERROR','Error en accion: '||i.accion||': '||SQLERRM);
        end;
     end loop;
     if v_num_acciones_err = 0 THEN
       if v_num_acciones = 0 THEN
         ins_bitacora(p_evento,'FIN DE PROCESO','Evento sin acciones');
       else
         ins_bitacora(p_evento,'FIN DE PROCESO','Se procesaron: '||v_num_acciones||',
todas las acciones');
         return;
       end if;
     else
       ins_bitacora(p_evento,'FIN CON ERROR',v_num_acciones_err||' acciones con
error');
       return;
     end if;
   else
     ins_bitacora(p_evento,'ERROR','Usuario: '||user||' no tiene privilegios para ejecutar
evento: '||p_evento);
   end if;
   EXCEPTION
   WHEN others THEN
       if v_tipo_evento = 'TABLA' THEN
         raise_application_error(-20200,'ERROR:'||substr(SQLERRM,1,2000));
       else
         ins_bitacora(p_evento,'ERROR',substr(SQLERRM,1,2000));
       end if;
--
end execute_eca;
-----------------------------------------------------------------------------------------
```

```
procedure create_tab(p_table in varchar2) is
--
-- procedimiento que crea una tabla correspondiente a la tabla p_table
--
  v_cuerpo varchar2(8000) := null;
begin
  for i in (select orden, columna, tipo, tamano, nula
          from columnas_tablas
          where tabla = p_table
          order by orden) loop
--
     begin
       select v_cuerpo||decode(i.orden,1,null,',')||i.columna||' '||i.tipo||
            decode(i.tipo,'VARCHAR2','('||i.tamano||') ',' ')||
            decode(i.nula,'N','NOT NULL',null)||chr(10)
       into v_cuerpo
       from dual;
       Exception
       WHEN others THEN
           raise_application_error(-20201,' pkg_sys.create_table: '||SQLERRM);
     end;
  end loop;
  create_obj(p_table,v_cuerpo,'TABLE',null);
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20202,' pkg_sys.create_tab: '||SQLERRM);
--
end create_tab;
-----------------------------------------------------------------------------------------
procedure create_pack_tab(p_table in varchar2) is
--
-- procedimiento que crea un package asociado a la tabla p_table con variables
-- correspondientes a las columnas de la tabla
--
  v_pack varchar2(8000) := null;
begin
  for i in (select column_name, data_type, data_length
          from user_tab_columns
          where table_name = p_table) loop
--
     begin
       select v_pack||'new_'||i.column_name||' '||i.data_type||
            decode(i.data_type,'VARCHAR2','('||i.data_length||'); ','; ')||chr(10)||
            'old_'||i.column_name||' '||i.data_type||
            decode(i.data_type,'VARCHAR2','('||i.data_length||'); ','; ')||chr(10)
       into v_pack
       from dual;
```

```
        Exception
        WHEN others THEN
            raise_application_error(-20201,' pkg_sys.create_pack_tab: '||SQLERRM);
      end;
   end loop;
   create_obj('g_'||p_table,v_pack,'PACKAGE',null);
   EXCEPTION
   WHEN others THEN
       raise_application_error(-20202,' pkg_sys.create_pack_tab: '||SQLERRM);
--
end create_pack_tab;
-----------------------------------------------------------------------------------------
function get_new_old_str(p_table in varchar2) return varchar2 is
--
-- funcion que retorna un string con las asignaciones de las variables new y old
-- correspondientes a las columnas de la tabla
--
   v_str varchar2(8000) := null;
begin
   for i in (select column_name, data_type, data_length
            from user_tab_columns
            where table_name = p_table) loop
--
      begin
        select v_str||'g_'||p_table||'.'||'new_'||i.column_name||' := :new.'||i.column_name||'; '||
                  'g_'||p_table||'.'||'old_'||i.column_name||' := :old.'||i.column_name||'; '
        into v_str
        from dual;
        Exception
        WHEN others THEN
            raise_application_error(-20201,' pkg_sys.get_new_old_str: '||SQLERRM);
      end;
   end loop;
   return(v_str);
--
end get_new_old_str;
-----------------------------------------------------------------------------------------
```

```
procedure execute_acc(p_acc in varchar2) is
--
-- procedimiento que ejecuta una accion p_acc.
--
  v_tex varchar2(200);
BEGIN
  v_tex := 'begin '||
             p_acc||'; '||
         'end;';
--
  execute_str(v_tex);
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.execute_acc: p_acc: '||p_acc||':
'||SQLERRM);
--
END execute_acc;
-----------------------------------------------------------------------------------------
procedure execute_str(p_str in varchar2) is
--
-- ejecuta una instruccion ddl p_ddl
--
  v_cursor number;
  v_ret number;
--
BEGIN
--
  v_cursor := dbms_sql.open_cursor;
  DBMS_SQL.PARSE(v_cursor,p_str,2);
  v_ret := DBMS_SQL.EXECUTE(v_cursor);
  DBMS_SQL.CLOSE_CURSOR(v_cursor);
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.execute_str: '||SQLERRM);
--
END execute_str;
-----------------------------------------------------------------------------------------
```

```
function get_cond(p_cond in varchar2) return boolean is
--
-- funcion que retorna el resultado de la evaluacion de la condicion p_cond
--
  v_cursor number;
  v_tex varchar2(200);
BEGIN
  v_tex := 'begin '||
             'pkg_sys.g_valor_cond := '||p_cond||'; '||
          'end;';
--
  execute_str(v_tex);
  return(g_valor_cond);
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.get_cond: p_cond: '||p_cond||':
'||SQLERRM);
--
END get_cond;
-----------------------------------------------------------------------------------------
function existe_obj(p_obj in varchar2) return boolean is
--
-- funcion que retorna TRUE si el objeto existe en la BD y FALSE en caso contrario
--
v_dummy varchar2(1);
BEGIN
  select 'x'
  into v_dummy
  from user_objects
  where object_name = p_obj
  and rownum = 1;
  return(TRUE);
  EXCEPTION
  WHEN no_data_found THEN
      return(FALSE);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.existe_obj: objeto: '||p_obj||':
'||SQLERRM);
--
END existe_obj;
-----------------------------------------------------------------------------------------
```

```
function get_status(p_obj in varchar2, p_type in varchar2) return varchar2 is
--
-- funcion que retorna el STATUS del objeto
--
salida varchar2(30);
BEGIN
  select status
  into salida
  from user_objects
  where object_name = p_obj
  and object_type = p_type;
  return(salida);
  EXCEPTION
  WHEN no_data_found THEN
      return(null);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.get_status: objeto: '||p_obj||', tipo:
'||p_type||': '||SQLERRM);
--
END get_status;
-----------------------------------------------------------------------------------------
function get_next_date(p_job_no in number) return date is
--
-- funcion que retorna la proxima ejecucion del job
--
salida date;
BEGIN
  select next_date
  into salida
  from user_jobs
  where job = p_job_no;
  return(salida);
  EXCEPTION
  WHEN no_data_found THEN
      return(null);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.get_next_date: job: '||p_job_no||':
'||SQLERRM);
--
END get_next_date;
-----------------------------------------------------------------------------------------
```

```
procedure create_obj(p_obj in varchar2, p_cuerpo in varchar2,
                 p_obj_type in varchar2, p_return_type in varchar2) is
v_return_type varchar2(15);
BEGIN
  if p_obj_type = 'PROCEDURE' THEN
    execute_str('create or replace procedure '||p_obj||' as '||
                 'BEGIN '||
                   p_cuerpo||' '||
                   'EXCEPTION '||
                   'WHEN others THEN '||
                       'raise_application_error(-20200,'' Error en procedure: '||p_obj||':
''||SQLERRM); '||
                 'END;');
  elsif p_obj_type = 'FUNCTION' THEN
    begin
      select decode(p_return_type,'VARCHAR2','varchar2(200)',p_return_type)
      into v_return_type
      from dual;
      execute_str('create or replace function '||p_obj||' return '||p_return_type||' as
'||chr(10)||
                p_obj||' '||v_return_type||'; '||chr(10)||
                'BEGIN '||
                  p_cuerpo||' '||
                 'return('||p_obj||'); '||chr(10)||
                 'EXCEPTION '||
                 'WHEN others THEN '||
                     'raise_application_error(-20200,'' Error en function: '||p_obj||':
''||SQLERRM); '||
                'END;');
   end;
  elsif p_obj_type = 'PACKAGE' THEN
    execute_str('create or replace package '||p_obj||' as '||chr(10)||p_cuerpo||' END;');
  elsif p_obj_type = 'TABLE' THEN
    drop_obj(p_obj,'TABLE');
    execute_str('create table '||p_obj||' ('||chr(10)||p_cuerpo||')');
  elsif p_obj_type = 'TRIGGER' THEN
    execute_str('CREATE OR REPLACE TRIGGER '||p_obj||' '||chr(10)||p_cuerpo);
  else
    raise_application_error(-20200,' No existe tipo de objeto: '||p_obj_type);
  end if;
  EXCEPTION
  WHEN others THEN
     raise_application_error(-20200,' pkg_sys.create_obj: '||p_obj_type||': '||p_obj||':
'||SQLERRM);
end create_obj;
-----------------------------------------------------------------------------------------
```

```
procedure drop_obj(p_obj in varchar2, p_type in varchar2) is
--
-- procedimiento que borra el objeto p_obj de tipo p_type
--
begin
  if existe_obj(p_obj) THEN
    execute_str('drop '||p_type||' '||p_obj);
  end if;
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.drop_obj: '||p_obj||': '||SQLERRM);
--
end drop_obj;
-----------------------------------------------------------------------------------------
function existe_job(p_job_no in number) return boolean is
--
-- funcion que retorna TRUE si el job existe en la BD y FALSE en caso contrario
--
v_dummy varchar2(1);
BEGIN
  select 'x'
  into v_dummy
  from user_jobs
  where job = p_job_no;
  return(TRUE);
  EXCEPTION
  WHEN no_data_found THEN
      return(FALSE);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.existe_job: job: '||to_char(p_job_no)||':
'||SQLERRM);
--
END existe_job;
-----------------------------------------------------------------------------------------
```

```
procedure drop_job(p_job_no in number) is
--
-- procedimiento que borra el job p_job_no
--
begin
  if existe_job(p_job_no) THEN
    dbms_job.remove(p_job_no);
  end if;
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.drop_obj: '||to_char(p_job_no)||':
'||SQLERRM);
--
end drop_job;
-----------------------------------------------------------------------------------------
procedure enable_trigger(p_trigger in varchar2) is
BEGIN
  if get_status(p_trigger,'TRIGGER') = 'VALID' THEN
    execute_str('alter trigger '||p_trigger||' enable');
  end if;
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.enable_trigger: '||p_trigger||':
'||SQLERRM);
end enable_trigger;
-----------------------------------------------------------------------------------------
procedure disable_trigger(p_trigger in varchar2) is
BEGIN
  if get_status(p_trigger,'TRIGGER') = 'VALID' THEN
    execute_str('alter trigger '||p_trigger||' disable');
  end if;
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.disable_trigger: '||p_trigger||':
'||SQLERRM);
end disable_trigger;
-----------------------------------------------------------------------------------------
procedure active_condicion(p_condicion in varchar2) is
BEGIN
  update condiciones_eventos
  set activo = 'S'
  where condicion = p_condicion;
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.active_condicion: '||p_condicion||':
'||SQLERRM);
```

```
end active_condicion;
-----------------------------------------------------------------------------------------
procedure active_accion(p_accion in varchar2) is
BEGIN
  update acciones_eventos
  set activo = 'S'
  where accion = p_accion;
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.active_accion: '||p_accion||':
'||SQLERRM);
end active_accion;
-----------------------------------------------------------------------------------------
procedure desactive_condicion(p_condicion in varchar2) is
BEGIN
  update condiciones_eventos
  set activo = 'N'
  where condicion = p_condicion;
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.desactive_condicion: '||p_condicion||':
'||SQLERRM);
end desactive_condicion;
-----------------------------------------------------------------------------------------
procedure desactive_accion(p_accion in varchar2) is
BEGIN
  update acciones_eventos
  set activo = 'N'
  where accion = p_accion;
  EXCEPTION
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.desactive_accion: '||p_accion||':
'||SQLERRM);
end desactive_accion;
-----------------------------------------------------------------------------------------
procedure ins_bitacora(p_evento in varchar2,p_status in varchar2,p_glosa in varchar2)
is
begin
  insert into bitacora_procesos
values(p_evento,sec_proc.nextval,sysdate,user,p_status,p_glosa);
end ins_bitacora;
-----------------------------------------------------------------------------------------
```

```
function get_privilegio(p_evento in varchar2, p_user in varchar2) return boolean is
--
-- funcion que retorna TRUE si el evento y el usuario se encuentra en la tabla de
privilegios
-- y FALSE en caso contrario
--
v_dummy varchar2(1);
BEGIN
  select 'x'
  into v_dummy
  from privilegios
  where evento = p_evento
  and usuario = p_user;
  return(TRUE);
  EXCEPTION
  WHEN no_data_found THEN
      return(FALSE);
  WHEN others THEN
      raise_application_error(-20200,' pkg_sys.get_privilegio: evento: '||p_evento||'
usuario '||p_user||SQLERRM);
--
end get_privilegio;
-----------------------------------------------------------------------------------------
end pkg_sys;
/
-- todo
-- ok cuando se desactiva una condicion o accion esta debe desactivar en todas partes de
los eventos
-- al activar una condicion o accion se debe chequear que esta este activa
-- ok agregar prioridad a las condiciones y a las acciones de eventos
-- ok al desactivar el evento debe desactivarse trigger o borrarse el job
-- ok tecnica de control de errores: el error se propaga hacia arriba en el flujo de
llamadas y en la llamada superior se hace rollback
-- ok y se almacena el error
-- trigger en bitacora de procesos para ver si hay privilegios
-- listas de valores de condiciones y acciones
-- ok verificar al crear funciones o procedimientos, que estos no existan
--
```

```
-- convenciones usadas
-- -------------------
-- pr_ = procedure
-- fn_ o get_ = function
-- pkg_ = pakage
-- trg_ = trigger
-- t_ = tipo definidos por el usuario
-- p_ = parametro
-- fk_ foreign key
-- pk_ primary_key
-- ind_ indice
-- v_ variable local
-- g_ variable global
-- tab_ tabla pl_sql
-- c_ cursores
```

9. Bibliografía

[Ull1999] Ullman, Jefrey, “Introducción a los Sistemas de Bases de Datos”,1 Ed de., Prentice-Hall, 1999.

[Cas1999] Castaño, Miguel, “Diseño de Base de Datos Relacionales”, 1 Ed de.,Ra-Ma, 1999.

[Koc1992] Koch, George, “Manual de Referencia”, 1 Ed de., Mcgraw-Hill, 1992.

[Gar1992] Georges Gardanin, “Bases De Donnees Object & relational”, 1 Ed de., Eyrolles, 1992.

[Gro1990] Groff, James, “Aplique SQL”, Falta edicion., Mcgraw-Hill, 1990.

[Cer1994] Ceri, Batini, “Diseño conceptual de bases de datos”, 1 Ed de, Addison-Wesley, 1994.

[Elm1992]Elmasri / Navathe,”Sistemas de Bases de Datos, Conceptos Fundamentales”, 2 Ed de., Addison Wesley, 1992

[Dor1997] Dorsey, Paul, “ Manual de Oracle”, 1 Ed de., Mcgraw-Hill, 1997

[Mul1997] Muller, Robert, “Manual de Oracle Developer/2000”, 1 Ed de., Mcgraw-Hill, 1997

[Dor1997] Dorsey, Paul, “Manual de Oracle Designer/2000”, 1 Ed de., Mcgraw-Hill, 1997

[Pat1998] Paton, Norman, “Active Rules in Database Systems”, 1 Ed de., Springer-Verlag, 1998

Lecturas Complementarias

[Cel2000] Celma,Matilde. http://www.dsic.upv.es/ asignaturas/facultad/bdv/documentos/ teoria/temaII_1.pdf.

Universidad Politécnica de Valencia, Departamento de Sistemas Informáticos y Computación.

[Pia1999]Piattini, Mariohttp://alarcos.inf- cr.uclm.es /doc/ bbddavanzadas/ doc99 activas.pdf.

Universidad Castilla-La Mancha, Departamento de Informática.

[Cal2000] Calero, Coral http:// alarcos.inf-cr.uclm.es/per/ccalero/

Universidad Castilla-La Mancha, Departamento de Informática.

10. Glosario

SGBD **:**Sistema gestor de bases de datos.

SGBDA **:**Sistema gestor de bases de datos activas.

BDA **:** Bases de datos activas.

Aproach :Planteamiento que considera la integración de unidades con el fin de construir una estructura completa y mas compleja.

ECA **:** Evento Condición Acción.

Metadatos **:** Guarda los datos y la estructura de los datos.

PK **:** Clave Primaria

FK **:** Clave Foránea

Commit **:** Grabar, (Actualizar), los últimos cambios realizados

Delay **: Retraso** que se produce en las situaciones de conflicto o chequeos generales.

Interfaz **:** La interfaz son las Pantallas de un Sistema, que relacionan al Usuario con la Base de Datos en forma amigable**.**

PL/SQL : En un lenguaje de programación que comparte con otros productos de la familia Oracle permitiendo gestionar los datos almacenados

SQL **:** Lenguaje que constituye el motor principal para la gestión de Bases de Datos de cualquier envergadura.

Rollback **:**Actualización de los datos.

BPR **:**Proceso de Reingenieria

E/R : Se refiere al Modelo Entidad Relación, que representa como los Datos están ordenados y relacionados en la Base de Datos.

Forms : Es la parte del entorno de desarrollo en la que se construyen los módulos de formularios.

LDD **:** Lenguaje de Definición de Datos, define el comportamiento de las acciones que se ejecutan sobre los datos.

Observaciones a la aplicación, según las definiciones de SGBDA.

1. **Nuestra SGBDA es un SGBD, ya que soporta lenguajes para el modelado, lenguajes de consultas**, Nuestra base de datos activa tiene como base un modelo relacional, el que está modelado para cumplir con lo que se quiere de la aplicación y soporta lenguaje de consultas, ya que la interfaz de nuestra aplicación le proporciona al usuario la posibilidad de consultar la base de datos.

2. **Un SGBDA tiene un modelo de reglas ECA, por lo que se debe extender el LDD (lenguaje de definición de datos) del SGBD para definir reglas.**
   Las reglas ECA están compuestas por el evento, la condición y la acción, las cuáles están presentes en nuestra base de datos activa, y también extiende el lenguaje de definición de datos de una base de datos, ya que no es necesario solamente las sentencias que se utilizan en una base de datos relacional, (Ej: create table, dop table, etc) sino que necesita de otra sentencia como lo es el ***Trigger*** para poder definir las reglas de la base de datos activa.

3. **Un SGBDA debe soportar la gestión de reglas y la evolución de la base de reglas**, para la administración de las reglas son utilizados los triggers, los cuáles son los que se encargarán de resguardar la consistencia de la base de datos una vez que sea manipulada.

4. **Un SGBDA tiene un modelo de ejecución**, nuestra base de datos activa tiene un modelo de ejecución, este modelo es esencial ya que en él se especifica como se comportan las reglas en
   tiempo de ejecución, desde el momento en que ocurre el evento hasta que se ejecuta la ación.

5. **Un SGBDA debe ofrecer diferentes modelos de acoplamiento**, nuestra base de datos activa ofrece solo un modelo de acoplamiento que es el Inmediato, si bien las condiciones se encuentran almacenadas en nuestra base de datos activa, una vez seleccionada la condición que se desea, esta se evalúa y si es verdadera se ejecuta la acción.

6. **Un SGBDA debería soportar un entorno de programación**, el SGBDA debe ser **usable,** en cambio SQL en una base de datos pasiva no es directamente usable ya que para que cualquier persona pueda usarlo debe entenderlo primero, en cambio la base de datos activa le proporciona esa interface al usuario para que pueda entender el SQL. También debe proporcionar un vizualizador de reglas, el cuál SQL no tiene.

7. **Un SGBDA debería ser ajustable, para que no sufra una degradación del rendimiento en comparación con soluciones realizadas sobre los SGBD pasivos**, esto se refiere a que como una base de datos activa necesita controlar muchas líneas de código para poder controlar el manejo de la base de datos y la funcionalidad de las reglas, puede verse afectado su rendimiento, en nuestra BDA el código esta almacenado en la base de datos, lo que produce una mayor interacción con la base de datos, sin que sea necesario tener por intermediaria a la aplicación para generar los resultados.

   También se requiere un diseño físico para los SGBDA, al momento de poner en práctica el desarrollo de una aplicación de base de datos activa esta necesita un diseño físico de cuáles elementos se va a componer la base de datos y como se van a comportar, en nuestro caso diseñamos el modelo relacional, para luego proceder a la creación de las tablas que componen la BDA.