# Trabajo Final de Carrera

UNIVERSISTAT OBERTA DE CATALUNYA

INGENIERÍA TÉCNICA INFORMÁTICA DE SISTEMAS
Junio 2014

Autor: Bernardo Pons Bibiloni
Consultor: Manel Rella Ruiz

# [ DISEÑO E IMPLEMENTACIÓN DE LA BASE DE DATOS DE UN SISTEMA CENTRALIZADO DE CONTROL DEL GASTO PÚBLICO DE LOS PARLAMENTOS EUROPEOS. ]

# Contenido

# Tabla de figuras.

# 1. Objetivos.

El objetivo principal del presente trabajo final de carrera es realizar un ejercicio práctico que sintetice los conocimientos adquiridos en diversas asignaturas a lo largo de la carrera y, en especial, demostrar el dominio adquirido en el campo de las bases de datos relacionales.

El trabajo a realizar consiste en llevar a cabo un proyecto que incluye las fases de diseño, implementación, pruebas y documentación de una base de datos relacional conforme al análisis de requerimientos suministrado como parte del enunciado.

La base de datos cuyo diseño e implementación se solicitan tiene por objeto permitir el control del gasto público realizado por los distintos parlamentos existentes en la Unión Europea.

El enunciado proporciona una serie de requerimientos funcionales que deberá cumplir la base de datos implementada. Por ello, además de realizar el diseño e implementación de la base de datos que almacenará la información, será preciso escribir código ejecutable que se almacenará en forma de procedimientos *SQL* dentro de la misma base de datos.

Quedan fuera del alcance de este trabajo tanto la implementación de cualquier aplicación de gestión utilizando lenguajes de alto nivel como el diseño de interfaces de usuario.

Los procedimientos almacenados deberán cumplir una serie de condiciones, a saber: tendrán un parámetro de salida para indicar si su ejecución ha finalizado con éxito o con un error, dispondrán de tratamiento de excepciones y se almacenará en una tabla un registro de todas las invocaciones realizadas a estos procedimientos, donde constarán los parámetros de entrada y sus valores, los parámetros de salida y sus valores, y entre ellos el resultado de la ejecución del procedimiento.

Los procedimientos almacenados deberán documentarse interna y externamente para permitir tanto una rápida comprensión de su funcionalidad como para facilitar su mantenimiento.

Como parte de los trabajos de programación deberán escribirse *scripts* de carga de datos en la base de datos así como *scripts* de pruebas que sirvan para comprobar el correcto funcionamiento de las funcionalidades implementadas y el tratamiento de excepciones.

La implementación de la base de datos se realizará mediante el sistema de gestión de base de datos *Oracle* que goza en la actualidad de una situación de liderazgo dentro del panorama de los sistemas de gestión de bases de datos relacionales en el mundo empresarial. Uno de los objetivos del trabajo es la adquisición de nuevos conocimientos al utilizar herramientas de gestión de bases de datos relacionales con las cuales no se ha trabajado previamente.

## 2. Metodología.

La metodología que se utilizará para la ejecución del trabajo es la metodología de desarrollo en cascada o *waterfal methodology*. La siguiente figura ilustra cada una de las etapas de esta metodología.

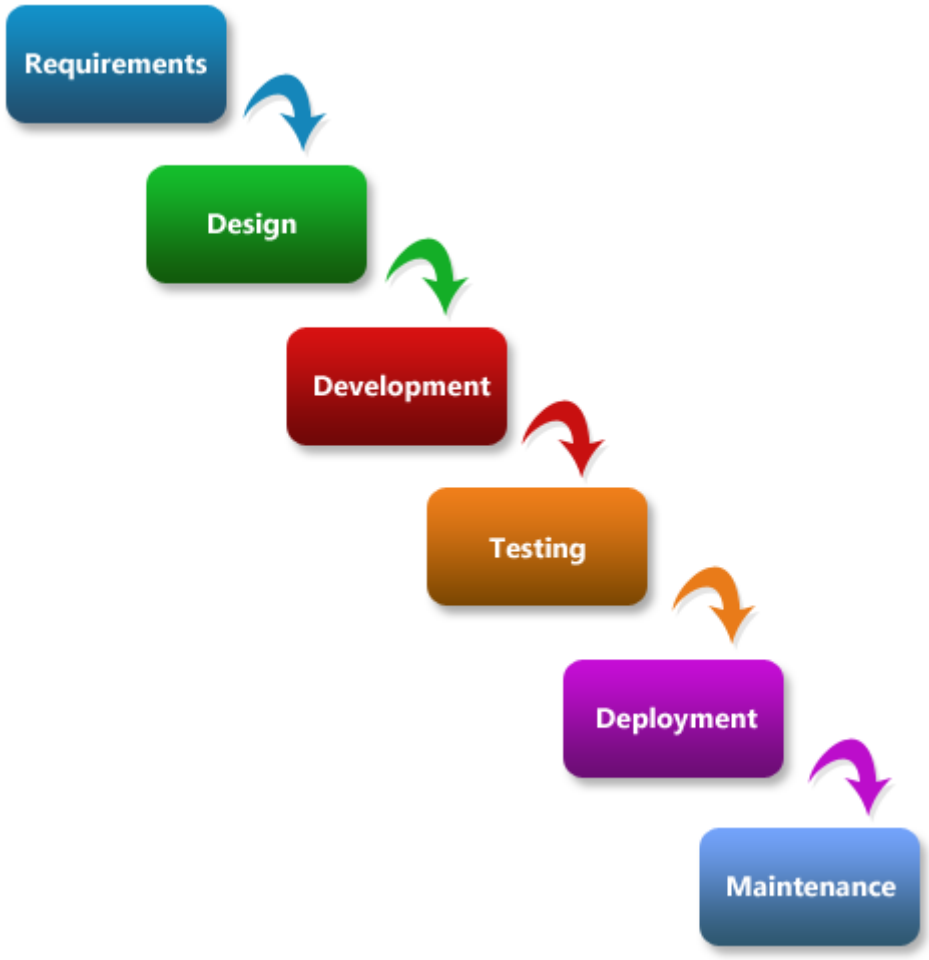


**Figura 1. Metodología de desarrollo en cascada.**

La principal razón por las que elegimos esta metodología es su sencillez. Por una parte, el reducido tamaño del proyecto también permite la adopción de esta metodología sin problemas. Por otra parte, su naturaleza secuencial se ajusta al hecho de que el número de recursos humanos disponibles para el desarrollo del trabajo es únicamente uno.

La fase de análisis de requerimientos ya viene realizada por el enunciado proporcionado. De esta fase tan sólo se realizarán tareas de eliminación de dudas o ambigüedades que pudieran surgir del enunciado.

Las fases que propiamente se desarrollarán durante este trabajo son las correspondientes al diseño, implementación, pruebas y documentación.

La implantación en un sistema real en producción no se llevará a cabo ni tampoco el mantenimiento posterior.

# 3. Planificación

## 3.1. Tareas y estimación de tiempos.

En la figura siguiente se puede observar la identificación de las principales tareas a realizar durante el desarrollo del proyecto. No obstante, debe tenerse en cuenta que el tiempo previsto de ejecución que aparece junto a cada una de ellas es tan solo una estimación.

La planificación realizada no significa que se vayan a dedicar jornadas laborables completas a la ejecución del proyecto sino que cada día de asignación al proyecto implica una sesión de trabajo ajustada al tiempo disponible en ese día concreto, que se prevee que será mayor en días festivos y fines de semana que en días laborables.

El tiempo real, medido en horas de dedicación al proyecto durante los días que aparecen en la planificación puede estimarse de forma teórica basándonos en la carga lectiva propia de la asignatura, que es de 7,5 créditos *ECTS*. Puesto que cada crédito *ECTS* equivale a una carga lectiva de 25 horas obtendríamos un tiempo de dedicación teórico de 187,5 horas.

No obstante, el tiempo real de dedicación necesario para realizar un trabajo final de carrera de buena calidad sobrepasa esta dedicación teórica. Con casi toda probabilidad el tiempo de dedicación real estará entre las 180 y las 240 horas.

Distribuyendo uniformemente estas horas de trabajo entre los días de duración del proyecto, según la planificación realizada, obtendríamos una dedicación media diaria aproximada de dos horas. Como ya hemos dicho antes, la dedicación no será uniforme dado que la disponibilidad de tiempo será mayor en días festivos y fines de semana que en días laborables.

| Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|
| **Trabajo final de carrera** | **97 días** | **mié 09/10/13** | **mar 14/01/14** |
| **Elaboración del plan de trabajo** | **12 días** | **mié 09/10/13** | **lun 21/10/13** |
| Lectura del enunciado del trabajo | 1 día | mié 09/10/13 | jue 10/10/13 |
| Identificación de tareas a realizar y estimación tiempos | 3 días | jue 10/10/13 | dom 13/10/13 |
| Descarga e instalación de *Microsoft Project* | 3 días | dom 13/10/13 | mié 16/10/13 |
| Elaboración diagrama de *Gantt* | 3 días | mié 16/10/13 | sáb 19/10/13 |
| Redacción documento plan de trabajo | 2 días | sáb 19/10/13 | lun 21/10/13 |
| Entrega PAC1 | 0 días | lun 21/10/13 | lun 21/10/13 |
| **Análisis y diseño** | **22 días** | **lun 21/10/13** | **mar 12/11/13** |
| Identificación de entidades y relaciones | 6 días | lun 21/10/13 | dom 27/10/13 |
| Descarga e instalación de *Magic Draw* | 3 días | dom 27/10/13 | mié 30/10/13 |
| Elaboración diagrama Entidad-Relación | 7 días | mié 30/10/13 | mié 06/11/13 |
| Redacción documento PAC2 | 6 días | mié 06/11/13 | mar 12/11/13 |
| Entrega PAC2 | 0 días | mar 12/11/13 | mar 12/11/13 |
| **Implementación y pruebas** | **30 días** | **mar 12/11/13** | **jue 12/12/13** |
| **Preparación entorno de trabajo** | **5 días** | **mar 12/11/13** | **dom 17/11/13** |
| Creación máquina virtual + instalación S.O. | 1 día | mar 12/11/13 | mié 13/11/13 |
| Descarga e instalación de *Oracle* | 4 días | mié 13/11/13 | dom 17/11/13 |
| **Programación *SQL*** | **16 días** | **dom 17/11/13** | **mar 03/12/13** |
| Creación base de datos | 1 día | dom 17/11/13 | lun 18/11/13 |
| Creación de tablas | 1 día | lun 18/11/13 | mar 19/11/13 |
| Programación de *triggers* | 7 días | mar 19/11/13 | mar 26/11/13 |
| Programación consultas *SQL* | 7 días | mar 26/11/13 | mar 03/12/13 |
| **Pruebas** | **4 días** | **mar 03/12/13** | **sáb 07/12/13** |
| Programación y ejecución *scripts* carga de datos | 2 días | mar 03/12/13 | jue 05/12/13 |
| Programación y ejecución *scripts* juegos de pruebas | 2 días | jue 05/12/13 | sáb 07/12/13 |
| **Redacción documento** | **5 días** | **sáb 07/12/13** | **jue 12/12/13** |
| Redacción documento PAC3 | 5 días | sáb 07/12/13 | jue 12/12/13 |
| Entrega PAC3 | 0 días | jue 12/12/13 | jue 12/12/13 |
| **Elaboración documentación final a entregar** | **33 días** | **jue 12/12/13** | **mar 14/01/14** |
| Elaboración memoria del proyecto | 21 días | jue 12/12/13 | jue 02/01/14 |
| Elaboración presentación proyecto | 12 días | jue 02/01/14 | mar 14/01/14 |
| Entrega documentación final | 0 días | mar 14/01/14 | mar 14/01/14 |

Figura 2. Planificación de tareas a realizar.

## 3.2. Hitos.

Los hitos naturales de este proyecto son las entregas parciales y la entrega final fijadas en el enunciado del mismo.

- El día 20 de Octubre de 2013 se entregará la PAC1 cuyo contenido será el plan de trabajo
- El día 11 de Noviembre de 2013 se entregará la PAC2 cuyo contenido será el diseño lógico de la base de datos.
- El día 11 de Diciembre de 2013 se entregará la PAC3 cuyo contenido será la implementación de la base de datos, los *scripts* de carga de datos y los *scripts* de juegos de pruebas.
- Finalmente, el día 13 de Enero de 2014 se entregará la documentación y el producto final.

En la planificación realizada dichos hitos aparecen un día después del aquí expuesto, ello es debido a la semántica utilizada por el programa *Microsoft Project*, en la que si se trabaja en una tarea en una fecha determinada, la fecha de finalización de la tarea debe ser el día siguiente.

Si bien las fechas de estos hitos ya están establecidas a priori podrían experimentar variaciones si se produjeran acontecimientos que justificasen modificaciones en las fechas de entrega.

## 3.3. Posibles incidencias.

Con el objeto de disminuir la posible aparición de riesgos relacionados con la capacidad de dedicación al trabajo final de carrera por cursar otros estudios simultáneamente este semestre no me he matriculado de ninguna otra asignatura.

No obstante, mis obligaciones, sobre todo laborales, podrían suponer un riesgo en cualquier momento en mi capacidad de dedicación al trabajo final de carrera.

También podrían presentarse incidencias debido a motivos de salud. De hecho, por cuestiones de salud, el trabajo ya ha sufrido un retraso de dos semanas. No obstante, estos problemas de salud ya están siendo tratados y no es de esperar que vuelvan a ocasionar retrasos.

## 3.4. Diagrama de *Gantt*.

| Id | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1 | **Trabajo final de carrera** | **97 días** | **mié 09/10/13** | **mar 14/01/14** |
| 2 | **Elaboración del plan de trabajo** | **12 días** | **mié 09/10/13** | **lun 21/10/13** |
| 3 | Lectura del enunciado del trabajo | 1 día | mié 09/10/13 | jue 10/10/13 |
| 4 | Identificación de tareas a realizar y estimación tiempos | 3 días | jue 10/10/13 | dom 13/10/13 |
| 5 | Descarga, instalación, lectura documentación Microsoft Project | 3 días | dom 13/10/13 | mié 16/10/13 |
| 6 | Elaboración diagrama de Gantt | 3 días | mié 16/10/13 | sáb 19/10/13 |
| 7 | Redacción documento plan de trabajo | 2 días | sáb 19/10/13 | lun 21/10/13 |
| 8 | Entrega PAC1 | 0 días | lun 21/10/13 | lun 21/10/13 |
| 9 | **Análisis y diseño** | **22 días** | **lun 21/10/13** | **mar 12/11/13** |
| 10 | Identificación de entidades y relaciones | 6 días | lun 21/10/13 | dom 27/10/13 |
| 11 | Descarga, instalación, lectura documentación Magic Draw | 3 días | dom 27/10/13 | mié 30/10/13 |
| 12 | Elaboración diagrama E-R | 7 días | mié 30/10/13 | mié 06/11/13 |
| 13 | Redacción documento PAC2 | 6 días | mié 06/11/13 | mar 12/11/13 |
| 14 | Entrega PAC2 | 0 días | mar 12/11/13 | mar 12/11/13 |
| 15 | **Implementación y pruebas** | **30 días** | **mar 12/11/13** | **jue 12/12/13** |
| 16 | **Preparación entorno de trabajo** | **5 días** | **mar 12/11/13** | **dom 17/11/13** |
| 17 | Creación máquina virtual + instalación S.O. | 1 día | mar 12/11/13 | mié 13/11/13 |
| 18 | Descarga, instalación, lectura documentación Oracle | 4 días | mié 13/11/13 | dom 17/11/13 |
| 19 | **Programación SQL** | **16 días** | **dom 17/11/13** | **mar 03/12/13** |
| 20 | Creación base de datos | 1 día | dom 17/11/13 | lun 18/11/13 |
| 21 | Creación de tablas | 1 día | lun 18/11/13 | mar 19/11/13 |
| 22 | Programación de triggers | 7 días | mar 19/11/13 | mar 26/11/13 |
| 23 | Programación consultas SQL | 7 días | mar 26/11/13 | mar 03/12/13 |
| 24 | **Pruebas** | **4 días** | **mar 03/12/13** | **sáb 07/12/13** |
| 25 | Programación y ejecución scripts carga de datos | 2 días | mar 03/12/13 | jue 05/12/13 |
| 26 | Programación y ejecución scripts juegos de pruebas | 2 días | jue 05/12/13 | sáb 07/12/13 |
| 27 | **Redacción documento** | **5 días** | **sáb 07/12/13** | **jue 12/12/13** |
| 28 | Redacción documento PAC3 | 5 días | sáb 07/12/13 | jue 12/12/13 |
| 29 | Entrega PAC3 | 0 días | jue 12/12/13 | jue 12/12/13 |
| 30 | **Elaboración documentación final a entregar** | **33 días** | **jue 12/12/13** | **mar 14/01/14** |
| 31 | Elaboración memoria del proyecto | 21 días | jue 12/12/13 | jue 02/01/14 |
| 32 | Elaboración presentación proyecto | 12 días | jue 02/01/14 | mar 14/01/14 |
| 33 | Entrega documentación final | 0 días | mar 14/01/14 | mar 14/01/14 |

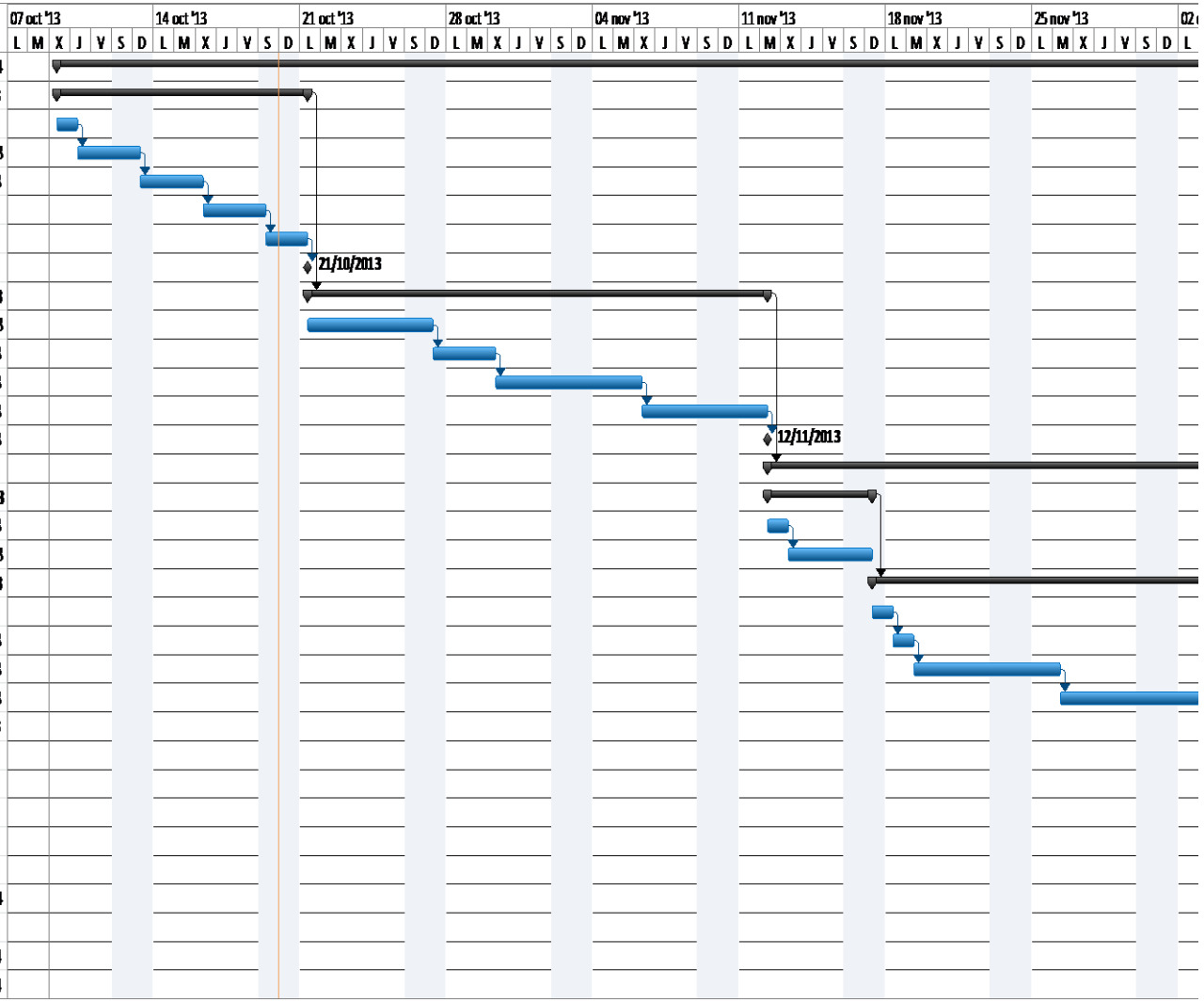


Figura 3. Diagrama de *Gantt* (parte 1).

| Id | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1 | **Trabajo final de carrera** | **97 días** | **mié 09/10/13** | **mar 14/01/14** |
| 2 | **Elaboración del plan de trabajo** | **12 días** | **mié 09/10/13** | **lun 21/10/13** |
| 3 | Lectura del enunciado del trabajo | 1 día | mié 09/10/13 | jue 10/10/13 |
| 4 | Identificación de tareas a realizar y estimación tiempos | 3 días | jue 10/10/13 | dom 13/10/13 |
| 5 | Descarga, instalación, lectura documentación Microsoft F | 3 días | dom 13/10/13 | mié 16/10/13 |
| 6 | Elaboración diagrama de Gantt | 3 días | mié 16/10/13 | sáb 19/10/13 |
| 7 | Redacción documento plan de trabajo | 2 días | sáb 19/10/13 | lun 21/10/13 |
| 8 | Entrega PAC1 | 0 días | lun 21/10/13 | lun 21/10/13 |
| 9 | **Análisis y diseño** | **22 días** | **lun 21/10/13** | **mar 12/11/13** |
| 10 | Identificación de entidades y relaciones | 6 días | lun 21/10/13 | dom 27/10/13 |
| 11 | Descarga, instalación, lectura documentación Magic Draw | 3 días | dom 27/10/13 | mié 30/10/13 |
| 12 | Elaboración diagrama E-R | 7 días | mié 30/10/13 | mié 06/11/13 |
| 13 | Redacción documento PAC2 | 6 días | mié 06/11/13 | mar 12/11/13 |
| 14 | Entrega PAC2 | 0 días | mar 12/11/13 | mar 12/11/13 |
| 15 | **Implementación y pruebas** | **30 días** | **mar 12/11/13** | **jue 12/12/13** |
| 16 | **Preparación entorno de trabajo** | **5 días** | **mar 12/11/13** | **dom 17/11/13** |
| 17 | Creación máquina virtual + instalación S.O. | 1 día | mar 12/11/13 | mié 13/11/13 |
| 18 | Descarga, instalación, lectura documentación Oracle | 4 días | mié 13/11/13 | dom 17/11/13 |
| 19 | **Programación SQL** | **16 días** | **dom 17/11/13** | **mar 03/12/13** |
| 20 | Creación base de datos | 1 día | dom 17/11/13 | lun 18/11/13 |
| 21 | Creación de tablas | 1 día | lun 18/11/13 | mar 19/11/13 |
| 22 | Programación de triggers | 7 días | mar 19/11/13 | mar 26/11/13 |
| 23 | Programación consultas SQL | 7 días | mar 26/11/13 | mar 03/12/13 |
| 24 | **Pruebas** | **4 días** | **mar 03/12/13** | **sáb 07/12/13** |
| 25 | Programación y ejecución scripts carga de datos | 2 días | mar 03/12/13 | jue 05/12/13 |
| 26 | Programación y ejecución scripts juegos de pruebas | 2 días | jue 05/12/13 | sáb 07/12/13 |
| 27 | **Redacción documento** | **5 días** | **sáb 07/12/13** | **jue 12/12/13** |
| 28 | Redacción documento PAC3 | 5 días | sáb 07/12/13 | jue 12/12/13 |
| 29 | Entrega PAC3 | 0 días | jue 12/12/13 | jue 12/12/13 |
| 30 | **Elaboración documentación final a entregar** | **33 días** | **jue 12/12/13** | **mar 14/01/14** |
| 31 | Elaboración memoria del proyecto | 21 días | jue 12/12/13 | jue 02/01/14 |
| 32 | Elaboración presentación proyecto | 12 días | jue 02/01/14 | mar 14/01/14 |
| 33 | Entrega documentación final | 0 días | mar 14/01/14 | mar 14/01/14 |

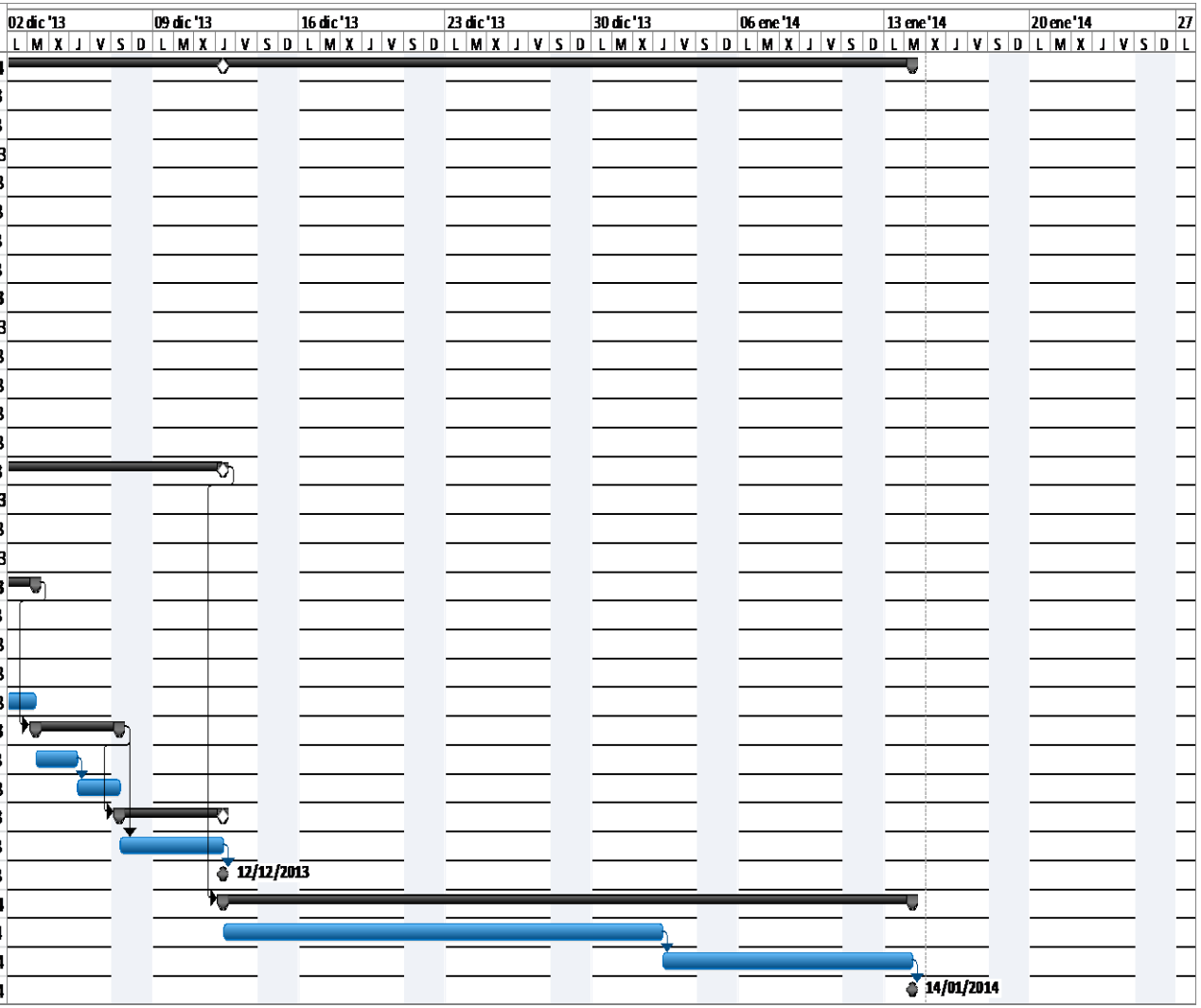


Figura 4. Diagrama de *Gantt* (parte 2).

## 3.5. Recursos necesarios.

**Recursos humanos.**

Este trabajo final de carrera será desarrollado únicamente por una persona que asumirá los diferentes roles de cada una de las diversas fases de desarrollo del proyecto. Como consecuencia en la planificación del proyecto realizada podrá observarse que algunas tareas que podrían haberse realizado en paralelo, en caso de que fuera posible delegarlas en otras personas, deben realizarse necesariamente de forma secuencial.

**Recursos de *hardware* y *software*.**

Por una parte será necesario contar con un ordenador, estación de trabajo, donde se realizarán:

a) La planificación del proyecto utilizando *Microsoft Project*.
b) El diagrama Entidad-Relación utilizando *Magic Draw*.
c) La redacción de los documentos correspondientes a las diferentes PACs y la memoria final con *Microsoft Word*.
d) La presentación en diapositivas a modo de resumen con *Microsoft Powerpoint*.

Por otra parte será necesario contar con un ordenador, servidor, donde se instalará y ejecutará todo el software del sistema de gestión de base de datos *Oracle*.

## 3.6. Valoración económica del proyecto.

| Tareas | Categoría | Horas | Precio | Coste |
|---|---|---|---|---|
| Análisis, coordinación, documentación | Jefe proyecto | 52 | 60,00 € | 3.120,00 € |
| Análisis y diseño lógico, creación base de datos | Analista | 24 | 60,00 € | 1.440,00 € |
| Programación *PL/SQL*, documentación interna | Programador | 55 | 45,00 € | 2.475,00 € |
| Pruebas | Probador | 30 | 30,00 € | 900,00 € |
| Documentación | Documentador | 32 | 30,00 € | 960,00 € |
| | **Totales** | **193** | | **8.895,00 €** |

**Figura 5. Valoración económica del proyecto.**

# 4. Diseño.

## 4.1. Requerimientos funcionales.

La base de datos a diseñar e implementar posteriormente deberá almacenar la información histórica y estadística de los gastos realizados por los diferentes parlamentos europeos, tanto aquellos gastos que sean atribuibles directamente a los parlamentarios (e indirectamente al parlamento al cual están adscritos en el momento de producirse el gasto) como aquellos gastos generales que sean atribuibles directamente a los distintos parlamentos.

En el presente documento iremos estudiando los diferentes requerimientos funcionales presentes en el enunciado del ejercicio y veremos cómo afectan al diseño de la base de datos que posteriormente tendremos que implementar.

Los nombres de las entidades, atributos y relaciones que a continuación se describen cambiarán ligeramente su nomenclatura en el momento de ser incluidos en la representación gráfica del diagrama Entidad-Relación y posteriormente en la implementación de la base de datos. En particular, para construir los nombres identificadores de entidades, atributos y relaciones se suprimirán los acentos y espacios en

blanco que puedan separar las palabras que los componen y se utilizarán mayúsculas intercaladas en el identificador para mejorar la legibilidad de los identificadores compuestos por varias palabras.

Asimismo, sin perjuicio de que puedan existir otras claves únicas naturales cuyo uso pudiera resultar lógico, salvo que expresamente se indique lo contrario, cada entidad incluirá un atributo identificador que será una secuencia autoincrementada de tipo entero. Este campo constituirá posteriormente la clave primaria identificativa de las filas de cada tabla en la base de datos relacional y servirá para relacionar las filas de unas tablas con las filas de otras utilizándolas como claves foráneas.

## 4.2. Identificación, justificación y descripción de entidades y relaciones.

Los diversos requerimientos funcionales presentes en el enunciado (puntos [R1], [R2], [R3], [R6] y [R7]) determinan la aparición en el diseño lógico de la base de datos de diferentes entidades, su conjunto de atributos y las relaciones con otras entidades, que veremos a continuación junto a una justificación de las mismas.

El requerimiento funcional [R1] determina la aparición de las entidades Parlamento, Tipo Parlamento, Ciudad, Región y País.

El requerimiento funcional [R2] determina la aparición de las entidades Parlamentario y Partido Político.

El requerimiento funcional [R3] determina la aparición de las entidades Ejercicio, Gasto, Tipo IVA, Categoría Gasto, Tipo Gasto, Proveedor, Estado y Forma Pago.

El requerimiento funcional [R7] determina la aparición de las relaciones Estadística Parlamento y Estadística Parlamentario.

Debido a algún error en la redacción del enunciado, no aparecen en éste los puntos [R4] ni [R5] como sería de esperar.

Algunas entidades identificadas podrían ser sustituidas por tipos de datos enumerativos en vez de dar lugar a la aparición de una tabla en la base de datos. No obstante, como se verá más adelante, hemos preferido modelar dichas entidades mediante tablas en la base de datos para dotar al sistema de una mayor flexibilidad. De esta manera se permiten ampliaciones futuras en la funcionalidad que, en muchos casos, podrían hacerse simplemente añadiendo una fila en una tabla y no requerirían, por tanto, de programación.

### 4.2.1. Entidad Parlamento.

La entidad Parlamento es necesaria para almacenar los datos básicos correspondientes a las diferentes cámaras parlamentarias.

**Atributos**

- **Identificador:** Número único de identificación del parlamento.
- **Nombre:** Nombre de la cámara parlamentaria.
  Por ejemplo: Congreso de los diputados, Senado, *Parlament de Catalunya, Parlament de les illes Balears, House of the commons, House of Lords*, etc…
- **Dirección:** Nombre y número de la calle donde el parlamento tiene su sede.
- **Código postal:** Código postal de la sede del parlamento.
- **Teléfono:** Teléfono de la sede del parlamento.
- **Web:** URL de la página web del parlamento.
- **Número de parlamentarios:** Número de parlamentarios que tiene cada parlamento.

**Relaciones con otras entidades**

- **Relación con Tipo Parlamento**
  Semántica: Relaciona cada parlamento con el tipo de parlamento al que pertenece.
  Por ejemplo: Europeo, Nacional, Regional.
  Cardinalidad [1..1]: Un parlamento tiene que ser obligatoriamente de uno y sólo uno de los tipos de parlamento existentes.

- **Relación con Ciudad**
  Semántica: Relaciona cada parlamento con la ciudad en la que tiene su sede.
  Cardinalidad [1..1]: Un parlamento tiene su sede en una y sólo una ciudad.

- **Relación con Parlamentario**
  Semántica: Relaciona cada parlamento con los parlamentarios que realizan su labor en él.
  Cardinalidad [0..*]: En un parlamento determinado puede haber cero o más parlamentarios. Podría ocurrir que un parlamento, en un momento determinado, no tuviera adscrito ningún parlamentario, por ejemplo: en el período entre legislaturas.

- **Relación con Gasto**
  Semántica: Relaciona cada parlamento con los gastos atribuibles a este parlamento.
  Cardinalidad [0..*]: A un parlamento determinado pueden serle atribuidos cero o más gastos registrados en la base de datos.

- **Relación con Parlamentario y Ejercicio (Estadística Parlamentario)**
  Semántica: Relaciona el resumen estadístico de los gastos de un parlamentario determinado en un parlamento determinado con un ejercicio.
  Cardinalidad con Parlamentario [0..*]: De un parlamento determinado, en un ejercicio determinado puede haber registrados gastos de cero o más parlamentarios.
  Cardinalidad con Ejercicio [0..*]: De un parlamento determinado y un parlamentario determinado adscrito a ese parlamento puede haber registrados gastos en cero o más ejercicios.

- **Relación con Ejercicio (Estadística Parlamento)**
  Semántica: Relaciona con un ejercicio el resumen estadístico de los gastos de un parlamento determinado.
  Cardinalidad [0..*]: De un parlamento determinado puede haber registrados gastos en cero o más ejercicios.

### 4.2.2. Entidad Tipo Parlamento.

No todos los parlamentos presentes en la base de datos serán del mismo tipo. Existirán diversos tipos de parlamentos dependiendo de si su ámbito geográfico de representación es europeo, nacional o regional.

Esta entidad podría ser un tipo de datos enumerativo {Europeo, Nacional, Regional}. Podría considerarse, como posible ampliación futura, añadir el tipo de parlamento “Local” para extender el control del gasto público a los Ayuntamientos, o bien modificar el tipo de datos enumerativo que quedaría como {Europeo, Nacional, Regional, Local}. En cualquier caso, esta ampliación no será implementada en el presente diseño de la base de datos.

**Atributos**

- **Identificador:** Número único de identificación del tipo de parlamento.
- **Descripción:** Nombre descriptivo del tipo de parlamento.
  Por ejemplo: Europeo, Nacional, Regional.

**Relaciones con otras entidades**

- **Relación con Parlamento**
  Semántica: Relaciona cada tipo de parlamento con los parlamentos existentes de ese tipo.
  Cardinalidad [1..*]: De un tipo de parlamento pueden existir en la base de datos uno o más parlamentos.

### 4.2.3. Entidad Ciudad.

Cada uno de los distintos parlamentos tendrá su única sede en una ciudad europea. No obstante, en una misma ciudad europea podrán coincidir las sedes de diversos parlamentos, en general de diferentes tipos de parlamento. En cualquier caso una restricción que obligue a que los parlamentos que tengan su sede en la misma ciudad sean de tipo distinto no se implementará en el diseño de la base de datos aunque podría considerarse como mejora.

**Atributos**

- **Identificador:** Número único de identificación de la ciudad.
- **Nombre:** Nombre de la ciudad.

**Relaciones con otras entidades**

- **Relación con Parlamento**
  Semántica: Relaciona cada ciudad con todos los parlamentos que tienen su sede en ella.
  Cardinalidad [1..*]: En una ciudad pueden tener su sede uno o más parlamentos.

- **Relación con Región**
  Semántica: Relaciona cada ciudad con la región en la que está situada.
  Cardinalidad [1..1]: Una ciudad está situada en una y sólo una región.

### 4.2.4. Entidad Región.

Cada ciudad está situada en una región. En una misma región puede haber varias ciudades. Estas regiones corresponden a las divisiones geográficas de los diferentes estados miembros de la Unión Europea que tengan cámaras parlamentarias. En algunos países, como por ejemplo en España, estas regiones corresponderían a las comunidades autónomas y en otros países, como por ejemplo Alemania, corresponderían a los *Lander*. Dado que los tipos de parlamento a tener en cuenta en el diseño son europeo, nacionales y regionales no hace falta definir un mayor detalle jerárquico geográfico y no son necesarios, por tanto, conceptos como provincia o municipio.

**Atributos**

- **Identificador:** Número único de identificación de la región.
- **Nombre:** Nombre de la región.

**Relaciones con otras entidades**

- **Relación con Ciudad**
  Semántica: Relaciona cada región con las ciudades que están situadas en ella.
  Cardinalidad [1..*]: En una región pueden existir una o más ciudades que sean sede de una cámara parlamentaria.

- **Relación con País**
  Semántica: Relaciona una región con cámara parlamentaria con el país al cual pertenece.
  Cardinalidad [1..1]: Una región está situada en un y sólo un país.

### 4.2.5. Entidad País.

Cada región con cámara parlamentaria está situada en un país. En un mismo país puede haber varias regiones que tengan cámara parlamentaria. Estos países se corresponden con los estados miembros de la Unión Europea.

**Atributos**

- **Identificador:** Número único de identificación de un país.
- **Nombre:** Nombre del país.

**Relaciones con otras entidades**

- **Relación con Región**
  Semántica: Relaciona cada país con todas sus regiones que tengan cámaras parlamentarias.
  Cardinalidad [1..*]: En un país pueden existir una o más regiones con cámaras parlamentarias.

### 4.2.6. Entidad Parlamentario.

La entidad Parlamentario es necesaria para almacenar los datos básicos correspondientes a los parlamentarios adscritos a las diferentes cámaras parlamentarias.

**Atributos**

- **Identificador:** Número único de identificación de un parlamentario.
- **Nombre:** Nombre del parlamentario.
- **Apellidos:** Apellidos del parlamentario.

**Relaciones con otras entidades**

- **Relación con Partido Político**
  Semántica: Relaciona a cada parlamentario con los partidos políticos en que ha militado a lo largo de su carrera política.
  Cardinalidad [0..*]: Un parlamentario puede militar en un solo partido político en un momento determinado. Sin embargo, un parlamentario puede militar en varios partidos políticos a lo largo de su carrera política.
  No obstante, un parlamentario podría, en un momento dado, no estar en ningún partido político, por ejemplo si abandona su partido y, sin ingresar en ningún otro, se pasa al grupo mixto. Más adelante veremos cómo puede reflejarse esta circunstancia en la base de datos.

- **Relación con Parlamento**
  Semántica: Relaciona a cada parlamentario con los parlamentos en los que realiza o ha realizado su labor parlamentaria.
  Cardinalidad [0..*]: Un parlamentario, en un momento determinado, estará adscrito a un solo parlamento.
  No obstante, es posible que un parlamentario pase a ejercer su labor parlamentaria a otro parlamento, por ejemplo, un diputado del Congreso de los Diputados español que se presente a las elecciones al Parlamento Europeo y consiga un escaño allí pasará de estar adscrito a un parlamento a estar adscrito a otro.
  En un momento determinado, un parlamentario podría no estar adscrito a ningún parlamento, por ejemplo si pierde su escaño.

- **Relación con Partido Político y Gasto**

Semántica: Relaciona a cada parlamentario con los gastos atribuibles a ese parlamentario mientras milita en un partido determinado.
Cardinalidad con Partido Político [0..1]: Dado un parlamentario determinado que esté relacionado con un gasto determinado, este gasto se relacionará también con un partido político y sólo uno, en el cual milita el parlamentario cuando se produce el gasto. La cardinalidad cero se utiliza cuando un gasto no es atribuible a un parlamentario (porque se trate de un gasto general del parlamento) en cuyo caso tampoco estará relacionado con ningún partido político.
Cardinalidad con Gasto [0..*]: Dado un parlamentario determinado que milita en un partido político determinado podrán existir cero o más gastos atribuibles a ese parlamentario y partido político registrados en la base de datos.

- **Relación con Parlamento y Ejercicio (Estadística Parlamentario)**
  Semántica: Relaciona el resumen estadístico de los gastos de un parlamentario determinado en un parlamento determinado con un ejercicio.
  Cardinalidad con Parlamento [0..*]: De un parlamentario determinado, en un ejercicio determinado puede haber registrados gastos en cero (si todavía no hay ningún gasto imputable al parlamentario en ningún ejercicio) o más parlamentos (si el parlamentario pasa por varios parlamentos en un mismo ejercicio).
  Cardinalidad con Ejercicio [0..*]: De un parlamentario determinado adscrito a un parlamento determinado puede haber registrados gastos registrados en cero o más ejercicios.

### 4.2.7. Entidad Partido Político.

La entidad Partido Político es necesaria para almacenar los datos básicos correspondientes a los partidos políticos en los cuales militan los parlamentarios.

**Atributos**

- **Identificador:** Número único de identificación del partido político.
- **Nombre:** Nombre del partido político.

**Relaciones con otras entidades**

- **Relación con Parlamentario**
  Semántica: Relaciona cada partido político con los parlamentarios que militan en él.
  Cardinalidad [0..*]: En un partido político pueden militar uno o más parlamentarios.
  Si bien lo habitual es que en un momento dado en un partido exista al menos un parlamentario también puede darse el caso de que no hubiera ninguno.

- **Relación con Parlamentario y Gasto**
  Semántica: Relaciona un partido político con un gasto determinado atribuible a un parlamentario determinado.
  Cardinalidad con Parlamentario [0..1]: Dado un partido político determinado que esté relacionado con un gasto determinado, este pertenecerá a un parlamentario y sólo a uno. La cardinalidad cero se utiliza cuando un gasto no es atribuible a un parlamentario (porque se trate de un gasto general del parlamento) en cuyo caso tampoco estará relacionado con ningún partido político.
  Cardinalidad con Gasto [0..*]: Dado un partido político determinado que esté relacionado con un parlamentario determinado podrán existir cero o más gastos atribuibles a ese parlamentario y partido.

### 4.2.8. Entidad Ejercicio.

De la redacción del enunciado queda patente la necesidad de utilizar el concepto o entidad de Ejercicio (que hace referencia al ejercicio fiscal al cual son imputables los gastos registrados en la base de datos). Este

concepto, presente en el diseño lógico y en el modelo Entidad-Relación, generará una tabla en la base de datos final puesto que resulta útil para la funcionalidad del modelo lógico.

**Atributos**

- **Identificador:** A diferencia de los demás casos este número no será una secuencia autoincrementada sino que se corresponderá, como hemos dicho, al año (ejercicio) al que sean imputables los gastos y en que deban ser asignados a efectos estadísticos.
- **Fecha inicio:** Fecha en que se inicia el ejercicio. Permite considerar ejercicios fiscales que no coincidan con el año natural.
- **Fecha final:** Fecha en que finaliza el ejercicio. Permite considerar ejercicios fiscales que no coincidan con el año natural.
- **Cerrado S/N:** Indica si el ejercicio está cerrado y, por tanto, ya no pueden imputársele gastos.

**Relaciones con otras entidades**

- **Relación con Gasto**
  Semántica: Relaciona con un ejercicio los gastos realizados y registrados en la base de datos.
  Cardinalidad [0..*]: De un ejercicio determinado podrán existir en la base de datos cero o más gastos registrados.

- **Relación con Parlamento y Parlamentario (Estadística Parlamentario)**
  Semántica: Relaciona con un ejercicio el resumen estadístico de los gastos de un parlamentario determinado en un parlamento determinado.
  Cardinalidad con Parlamento [0..*]: En un ejercicio determinado y de un parlamentario determinado puede haber registrados gastos en cero o más parlamentos (con uno si el parlamentario ha estado adscrito a un solo parlamento en ese ejercicio o con más de uno si el parlamentario ha estado adscrito a varios parlamentos en ese ejercicio).
  Cardinalidad con Parlamentario [0..*]: En un ejercicio determinado y de un parlamento determinado puede haber registrados gastos de cero o más parlamentarios.

- **Relación con Parlamento (Estadística Parlamento)**
  Semántica: Relaciona con un ejercicio el resumen estadístico de los gastos de un parlamento determinado.
  Cardinalidad [0..*]: En un ejercicio determinado puede haber registrados gastos de cero o más parlamentos.

### 4.2.9. Entidad Gasto.

La entidad Gasto es necesaria para almacenar los datos básicos correspondientes a los gastos registrados.

**Atributos**

- **Identificador:** Número único de identificación del gasto.
- **Fecha:** Fecha en la que se ha producido el gasto.
- **Base imponible:** Importe del gasto sin IVA.
- **Descripción:** Texto descriptivo del gasto realizado (opcional).
- **Justificación:** Texto justificativo del gasto realizado (opcional).
- **Presupuesto:** URL del presupuesto escaneado (opcional).
- **Factura:** URL de la factura escaneada (opcional).
- **Contrato:** URL del contrato escaneado (opcional).

**Relaciones con otras entidades**

- **Relación con Ejercicio**

Semántica: Relaciona los gastos realizados y registrados en la base de datos con el ejercicio al que son imputables. El ejercicio entre cuyas fechas de inicio y final se encuentre la fecha del gasto será el ejercicio al que corresponderá el gasto.
Cardinalidad [1..1]: Un gasto determinado es imputable a un solo ejercicio.

- **Relación con Parlamento**
  Semántica: Relaciona un gasto con el parlamento al cual es imputable.
  Cardinalidad [1..1]: Un gasto determinado es imputable a un solo parlamento.

- **Relación con Estado**
  Semántica: Relaciona un gasto con el estado en que se encuentra.
  Cardinalidad [1..1]: En un momento determinado un gasto determinado se encuentra en un solo estado.

- **Relación con Forma de pago**
  Semántica: Relaciona un gasto con la forma de pago utilizada para su pago.
  Cardinalidad [0..1]: Un gasto determinado será pagado mediante una sola forma de pago. La cardinalidad cero se utiliza mientras el gasto no haya sido pagado.

- **Relación con Parlamentario y Partido Político**
  Semántica: Relaciona cada gasto con el parlamentario al que pueda ser imputado y el partido político en el que milita ese parlamentario cuando se produce el gasto.
  Cardinalidad con Parlamentario [0..1]: Dado un gasto determinado que esté relacionado con un partido determinado, este gasto estará relacionado con un solo parlamentario (que milita en ese partido político en el momento de producirse el gasto). La cardinalidad cero se utiliza cuando un gasto no es atribuible a un parlamentario (porque se trate de un gasto general del parlamento) en cuyo caso tampoco estará relacionado con ningún partido político.
  Cardinalidad con Partido Político [0..1]: Dado un gasto determinado que sea atribuible a un parlamentario determinado, este estará relacionado también con un partido político y sólo uno, en el cual milita el parlamentario cuando se produce el gasto. La cardinalidad cero se utiliza cuando un gasto no es atribuible a un parlamentario (porque se trate de un gasto general del parlamento) en cuyo caso tampoco estará relacionado con ningún partido político.

- **Relación con Tipo de IVA**
  Semántica: Relaciona cada gasto con el tipo de IVA aplicable a ese gasto.
  Cardinalidad [1..1]: A un gasto determinado puede aplicársele un solo tipo de IVA.

- **Relación con Categoría de gasto**
  Semántica: Relaciona cada gasto con la categoría de gasto a la que pertenece.
  Cardinalidad [1..1]: Un gasto determinado pertenece solo a una categoría de gasto.

- **Relación con Proveedor**
  Semántica: Relaciona cada gasto con el proveedor que ha suministrado dicho bien o servicio, si existe.
  Cardinalidad [0..1]: Un gasto determinado estará relacionado con cero o un proveedor.

### 4.2.10. Entidad Tipo IVA.

La entidad Tipo IVA es necesaria para almacenar los datos básicos correspondientes a los distintos porcentajes de IVA que gravan los gastos registrados.

**Atributos**

- **Identificador:** Número único de identificación del tipo de IVA.
- **Porcentaje:** Porcentaje de IVA aplicable

**Relaciones con otras entidades**

- **Relación con Gasto**
  Semántica: Relaciona los tipos de IVA con los gastos registrados a los que son aplicables.
  Cardinalidad [0..*]: Un tipo de IVA determinado puede ser aplicable a cero o más gastos registrados en la base de datos.

### 4.2.11. Entidad Categoría Gasto.

La entidad Categoría Gasto es necesaria para almacenar las distintas descripciones básicas normalizadas en que se clasifican los distintos gastos registrados. Estas categorías pueden pertenecer a su vez a dos tipos de gasto: los que son directamente imputables a un parlamentario (e indirectamente imputables al parlamento al que está adscrito en el momento en que se realiza el gasto) y los que son directamente imputables a un parlamento no siendo atribuibles a ningún parlamentario en concreto.

**Atributos**

- **Identificador:** Número único de identificación de la categoría de gasto.
- **Descripción:** Nombre descriptivo de la categoría de gasto.
  Por ejemplo: Sueldo parlamentario, móvil parlamentario, taxi parlamentario, billete de avión parlamentario, escolta parlamentario, chófer parlamentario, escolta parlamento, chófer parlamento, etc…

**Relaciones con otras entidades**

- **Relación con Gasto**
  Semántica: Relaciona cada categoría de gasto con todos los gastos registrados pertenecientes a esa categoría.
  Cardinalidad [0..*]: Una categoría de gasto determinada puede aparecer en cero o más gastos registrados.

- **Relación con Tipo gasto**
  Semántica: Relaciona cada categoría de gasto con el tipo de gasto al que pertenece.
  Cardinalidad [1..1]: Una categoría de gasto determinada pertenece a un solo tipo de gasto.

### 4.2.12. Entidad Tipo Gasto.

La entidad Tipo Gasto es necesaria porque permite diferenciar si las categorías de gasto son imputables a un parlamentario directamente (e indirectamente al parlamento al cual está adscrito) o si son imputables directamente a un parlamento (sin ser imputables a ningún parlamentario concreto).

Esta entidad podría ser un tipo de datos enumerativo {Gasto imputable a parlamento, Gasto imputable a parlamentario}.

**Atributos**

- **Identificador:** Número único de identificación del tipo de gasto.
- **Descripción:** Nombre descriptivo del tipo de gasto
- **ParlamentarioSN:** Indica si el tipo de gasto requiere que se impute a un parlamentario o no.

**Relaciones con otras entidades**

- **Relación con Categoría Gasto**
  Semántica: Relaciona cada categoría de gasto con el tipo de gasto al que pertenece.
  Cardinalidad [0..*]: De un tipo de gasto determinado pueden existir cero o más categorías de gasto en la base de datos.

### 4.2.13. Entidad Proveedor.

La entidad Proveedor es necesaria para almacenar los datos básicos correspondientes a los distintos proveedores que suministran los bienes y/o servicios correspondientes a los gastos registrados, cuando éste haya sido suministrado por un proveedor. Cabe destacar que indicar a qué proveedor pertenece el gasto es obligatorio a partir de un determinado importe.

**Atributos**

- **Identificador:** Número único de identificación del proveedor.
- **Nombre:** Nombre del proveedor
- **CIF:** Código de Identificación fiscal del proveedor.

**Relaciones con otras entidades**

- **Relación con Gasto**
  Semántica: Relaciona cada proveedor con los gastos registrados de los que ha suministrado el bien y/o servicio correspondiente.
- Cardinalidad [0..*]: Un proveedor determinado puede haber suministrado los bienes y/o servicios de cero o más gasto registrados en la base de datos.

### 4.2.14. Entidad Estado.

Esta entidad es necesaria para representar en la base de datos los distintos estados en que se encuentra cada uno de los gastos registrados.

Esta entidad podría ser un tipo de datos enumerativo {Pendiente de aprobación, Aprobado, Pendiente de pago, Pagado} si bien es preferible que sea posible añadir nuevos estados en el futuro.

**Atributos**

- **Identificador:** Número único de identificación del estado.
- **Descripción:** Nombre descriptivo del estado.
  Por ejemplo: Pendiente de aprobación, Aprobado, Pendiente de pago, Pagado.
- **Forma Pago S/N:** Indica si este estado requiere especificar una forma de pago porque el hecho de que un gasto se encuentre en este estado significa que ha sido pagado.

**Relaciones con otras entidades**

- **Relación con Gasto**
  Semántica: Relaciona cada gasto con el estado en que se encuentra.
  Cardinalidad [0..*]: En un estado determinado pueden encontrase cero o más gastos registrados.

### 4.2.15. Entidad Forma Pago.

Esta entidad es necesaria para representar en la base de datos las diversas formas de pago mediante las cuales se pueden pagar los gastos registrados.

Esta entidad podría ser un tipo de datos enumerativo {Transferencia bancaria, Tarjeta de crédito} si bien es preferible que sea posible añadir nuevas formas de pago en el futuro.

**Atributos**

- **Identificador:** Número único de identificación de la forma de pago.
- **Descripción:** Nombre descriptivo de la forma de pago.
  Por ejemplo: Transferencia bancaria, Tarjeta de crédito, etc…

**Relaciones con otras entidades**

- **Relación con Gasto**
  Semántica: Relaciona cada gasto con la forma de pago utilizada para su pago, una vez se ha realizado el pago.
  Cardinalidad [0..*]: Una forma de pago determinada podrá ser utilizada para realizar el pago de cero o más gastos.

### 4.2.16. Relación ParlamentoDH.

La relación entre las entidades Parlamento y Parlamentario (ParlamentoDH) tiene una importancia especial que explicaremos a continuación.

Como hemos dicho antes, es posible que un parlamentario pase a ejercer su labor parlamentaria de uno a otro parlamento a lo largo de su carrera política. Por ejemplo, un diputado del Congreso de los Diputados español que se presente a las elecciones al Parlamento Europeo y consiga un escaño allí pasará de estar adscrito en el Congreso de los Diputados de España a estar adscrito al Parlamento Europeo.

Puesto que el diseño que vamos a realizar contemplará la posibilidad de que un parlamentario pase a ejercer su labor parlamentaria de un parlamento a otro, la relación entre las entidades Parlamento y Parlamentario ha de ser una relación N:M.

Como consecuencia, la relación entre Parlamento y Parlamentario tiene dos atributos propios que son la fecha de alta del parlamentario en un parlamento y su fecha de baja. Es decir, un parlamentario está adscrito a un parlamento desde (D) una fecha determinada hasta (H) otra fecha determinada, de ahí que para componer el nombre de la relación se haya utilizado ParlamentoDH.

La fecha de alta de un parlamentario en un parlamento nunca podrá ser nula. Sin embargo la fecha de baja de un parlamentario en un parlamento sí podrá ser nula, circunstancia que se producirá durante todo el tiempo en que dicho parlamentario esté adscrito a dicho parlamento.

Un parlamentario, en un momento determinado, estará adscrito a un solo parlamento. No obstante, es posible que un parlamentario ejerza su labor parlamentaria en más de un parlamento a lo largo de su carrera política.

En un momento determinado, un parlamentario podría no estar adscrito a ningún parlamento, por ejemplo si pierde su escaño. Este caso especial podría ser reflejado en la base de datos si todas las instancias de la relación entre las entidades Parlamento y Parlamentario tuvieran valores en el atributo fecha de baja, es decir, cuando el parlamentario ha causado baja en todos los parlamentos en los que ha estado adscrito hasta el momento. Puesto que dicho parlamentario podría tener información histórica en la base de datos no procedería su eliminación.

Por otra parte, en un momento determinado, podría darse el caso de que en un parlamento no tuviera adscrito ningún parlamentario, por ejemplo, en el período entre legislaturas. Puesto que dicho parlamento podría tener información histórica en la base de datos no procedería su eliminación.

### 4.2.17. Relación ParlamentoST.

La relación entre las entidades Parlamento y Ejercicio tiene una importancia especial que explicaremos a continuación.

La relación llamada Estadística Parlamento (ParlamentoST) es una relación binaria entre las entidades Ejercicio y Parlamento que recopila información estadística sobre el gasto atribuible a un parlamento determinado durante un ejercicio.

Esta relación tiene diversos atributos propios que recogen información estadística del gasto atribuible a un parlamento determinado durante un ejercicio determinado. Esta relación se convertirá posteriormente en una tabla de la base de datos.

### 4.2.18. Relación ParlamentarioST.

La relación entre las entidades Parlamento, Ejercicio y Parlamentario tiene una importancia especial que explicaremos a continuación.

La relación llamada Estadística Parlamentario (ParlamentarioST) es una relación ternaria entre las entidades Ejercicio, Parlamento y Parlamentario que recopila los gastos totales que cada parlamentario ha realizado durante un ejercicio en un parlamento.

De acuerdo con el enunciado, esta relación podría ser una relación binaria que obviara la entidad Parlamento. Sin embargo, desde el momento en que permitimos que un parlamentario pueda cambiar de parlamento (lo cual puede ocurrir en mitad de un ejercicio) es necesario tener también en consideración el parlamento a la hora de totalizar los gastos registrados de un parlamentario en cada ejercicio. En cualquier caso, hemos creído que haciéndolo de esta manera se mejora el diseño.

Esta relación ternaria tiene un atributo propio que totaliza el gasto atribuible a un parlamentario determinado adscrito a un parlamento determinado durante un ejercicio determinado. Esta relación se convertirá posteriormente en una tabla de la base de datos.

### 4.2.19. Relación PartidoDH.

La relación entre las entidades Partido y Parlamentario (PartidoDH) tiene una importancia especial que explicaremos a continuación.

De la redacción del requerimiento funcional [R3] donde se habla de “partido político del parlamentario en el momento de almacenar el gasto” y del requerimiento funcional [R6] donde se habla de “partido político actual” se desprende que la relación entre las entidades Partido Político y Parlamentario no debe ser una relación 1:N sino que ha de ser una relación N:M de manera que permita tener en cuenta los posibles cambios de partido que pueda protagonizar un parlamentario a lo largo de su carrera política.

Como consecuencia, la relación entre las entidades Partido Político y Parlamentario tiene dos atributos propios que son la fecha de alta del parlamentario en dicho partido político y su fecha de baja. Es decir, un parlamentario militará en un partido político desde (D) una fecha determinada hasta (H) otra fecha determinada, de ahí que para componer el nombre de la relación se haya utilizado PartidoDH.

La fecha de alta de un parlamentario en un partido político nunca podrá ser nula. Sin embargo la fecha de baja de un parlamentario en un partido político sí podrá ser nula, circunstancia que se producirá durante todo el tiempo en que dicho parlamentario milite en dicho partido.

Un parlamentario puede militar en un solo partido político en un momento determinado. Sin embargo, un parlamentario puede militar en varios partidos políticos a lo largo de su carrera política.

En un momento determinado, un parlamentario podría no estar en ningún partido político, por ejemplo si abandona su partido y, sin ingresar en ningún otro. Este caso especial podría ser reflejado en la base de datos si todas las instancias de la relación entre Partido Político y Parlamentario tuvieran valores en el atributo fecha de baja, es decir, cuando el parlamentario ha causado baja en todos los partidos en los que ha militado hasta el momento. Puesto que dicho parlamentario podría tener información histórica en la base de datos no procedería su eliminación.

Por otra parte, en un momento determinado, podría darse el caso de que en un partido no existiera ningún parlamentario, por ejemplo porque todos sus parlamentarios lo hubieran abandonado. Puesto que dicho partido podría tener información histórica en la base de datos no procedería su eliminación.

## 4.3. Diagrama Entidad-Relación.

A continuación se muestra el diagrama Entidad-Relación que surge de las entidades y relaciones cuya identificación, descripción y justificación hemos realizado.

Para confeccionar este diagrama Entidad-Relación se ha utilizado la herramienta de diseño *CASE Magic Draw UML Personal Edition version 17.0 SP5*. Esta herramienta no incluye el módulo para la realización de diagramas Entidad-Relación, que sólo se incluye como módulo opcional en las versiones *Professional* superiores. Por esta razón, el diagrama Entidad-Relación siguiente es en realidad un diagrama de clases *UML* adaptado.

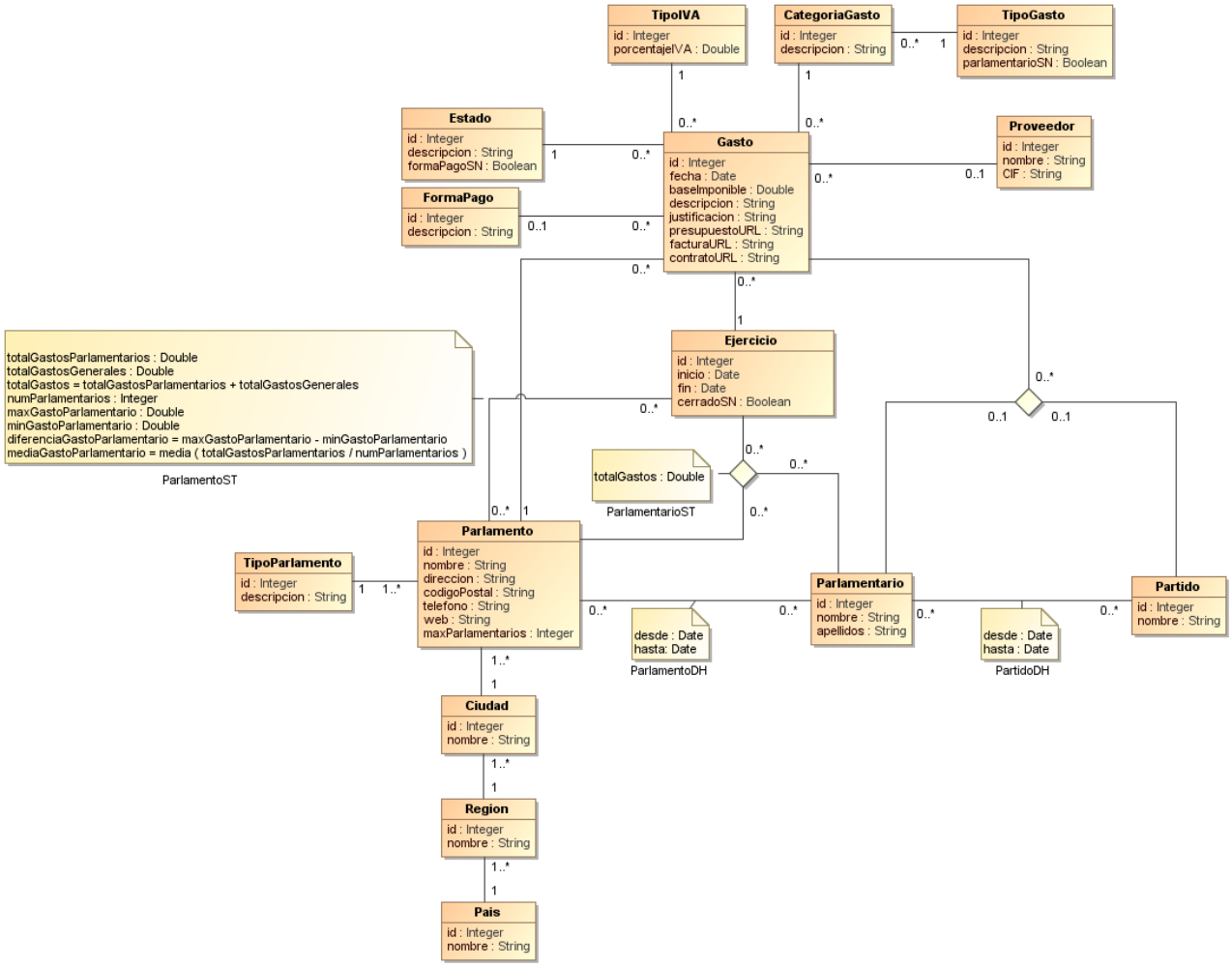


**Figura 6. Diagrama Entidad-Relación.**

# 5. Implementación.

## 5.1. Diseño de la base de datos.

A partir del diagrama Entidad-Relación anterior se obtiene una base de datos que se muestra en la siguiente figura.

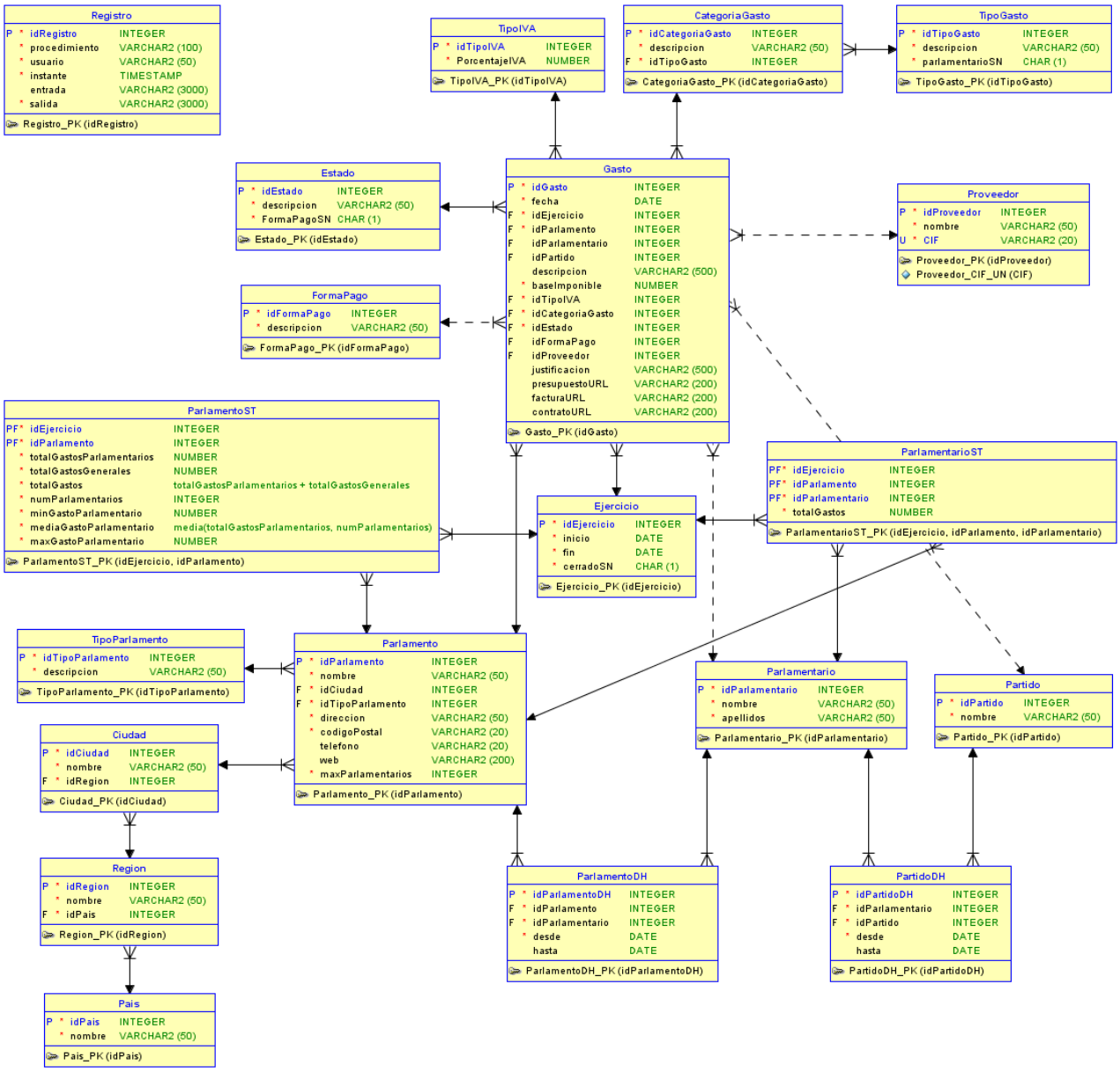


Figura 7. Diagrama de tablas de la base de datos.

## 5.2. Tablas de la base de datos.

A continuación se detallan las veinte tablas que forman la base de datos. Dichas tablas surgen de las quince entidades anteriormente detalladas más las cuatro relaciones con atributos que también dan lugar a tablas en la base de datos. A éstas se añade una tabla llamada Registro que no corresponde a ninguna entidad ni relación del diseño lógico sino que corresponde a un requerimiento de la implementación.

Las columnas de las tablas de la base de datos son las resultantes de la trasposición de los atributos de las entidades o relaciones del diseño lógico a las que se añaden los identificadores de otras tablas que actuarán como claves foráneas para implementar las relaciones entre tablas.

### 5.2.1. Tabla Registro.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idRegistro | NUMBER | No | Identificador de la entrada en el registro de ejecuciones de procedimientos (secuencia autoincrementada). |
| 2 | procedimiento | VARCHAR | No | Nombre del procedimiento ejecutado. |
| 3 | usuario | VARCHAR | No | Nombre del usuario que ejecutó el procedimiento. |
| 4 | instante | TIMESTAMP | No | Instante de tiempo en que se ejecutó el procedimiento. |
| 5 | entrada | VARCHAR | Yes | String con el nombre de los parámetros de entrada y sus valores. |
| 6 | salida | VARCHAR | No | String con el nombre de los parámetros de salida y sus valores. |

### 5.2.2. Tabla Parlamento.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idParlamento | NUMBER | No | Identificador del parlamento (secuencia autoincrementada). |
| 2 | nombre | VARCHAR | No | Nombre del parlamento. |
| 3 | idCiudad | NUMBER | No | Identificador de la ciudad donde está situado el parlamento. |
| 4 | idTipoParlamento | NUMBER | No | Identificador del tipo de parlamento. |
| 5 | direccion | VARCHAR | No | Dirección del parlamento. |
| 6 | codigoPostal | VARCHAR | No | Código postal del parlamento. |
| 7 | telefono | VARCHAR | Yes | Teléfono (centralita) del parlamento. |
| 8 | web | VARCHAR | Yes | URL de la página web del parlamento. |
| 9 | maxParlamentarios | NUMBER | No | Número máximo de parlamentarios del parlamento. |

### 5.2.3. Tabla TipoParlamento.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idTipoParlamento | NUMBER | No | Identificador del tipo de parlamento (secuencia autoincrementada). |
| 2 | descripcion | VARCHAR | No | Texto descriptivo del tipo de parlamento. |

### 5.2.4. Tabla Ciudad.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idCiudad | NUMBER | No | Identificador de la ciudad (secuencia autoincrementada). |
| 2 | nombre | VARCHAR | No | Nombre de la ciudad. |
| 3 | idRegion | NUMBER | No | Identificador de la región a la que pertenece. |

### 5.2.5. Tabla Region.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idRegion | NUMBER | No | Identificador de la región (secuencia autoincrementada). |
| 2 | nombre | VARCHAR | No | Nombre de la región. |
| 3 | idPais | NUMBER | No | Identificador del país al que pertenece la región. |

### 5.2.6. Tabla Pais.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idPais | NUMBER | No | Identificador del país (secuencia autoincrementada). |
| 2 | nombre | VARCHAR | No | Nombre del país. |

### 5.2.7. Tabla Parlamentario.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idParlamentario | NUMBER | No | Identificador del parlamentario (secuencia autoincrementada). |
| 2 | nombre | VARCHAR | No | Nombre del parlamentario. |
| 3 | apellidos | VARCHAR | No | Apellidos del parlamentario. |

### 5.2.8. Tabla Partido.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idPartido | NUMBER | No | Identificado del partido (secuencia autoincrementada). |
| 2 | nombre | VARCHAR | No | Nombre del partido. |

### 5.2.9. Tabla Ejercicio.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idEjercicio | NUMBER | No | Identificador del ejercicio. Secuencia manual que debe seguir el patrón: 2012, 2013, 2014, 2015, ... |
| 2 | inicio | DATE | No | Fecha de inicio del ejercicio. |
| 3 | fin | DATE | No | Fecha de fin del ejercicio. |
| 4 | cerradoSN | BOOLEAN | No | Indica si el ejercicio esta cerrado. Si el ejercicio está cerrado ya no se le pueden imputar gastos. |

### 5.2.10. Tabla Gasto.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idGasto | NUMBER | No | Identificador del gasto (secuencia autoincrementada). |
| 2 | fecha | DATE | No | Fecha del gasto. |
| 3 | idEjercicio | NUMBER | No | Ejercicio al que pertenece el gasto (es función de la fecha). |
| 4 | idParlamento | NUMBER | Yes | Identificador del parlamento al que debe imputarse el gasto. |
| 5 | idParlamentario | NUMBER | Yes | Identificador del parlamentario (si lo hay) al que debe imputarse el gasto. |
| 6 | idPartido | NUMBER | Yes | Identificador del partido al que pertenece el parlamentario (si lo hay) al que debe imputarse el gasto. |
| 7 | descripcion | VARCHAR | Yes | Texto descriptivo del gasto. |
| 8 | baseImponible | NUMBER | No | Base imponible del gasto. |
| 9 | idTipoIVA | NUMBER | No | Identificador del tipo de IVA aplicable al gasto. |
| 10 | idCategoriaGasto | NUMBER | No | Identificador de la categoría de gasto al que pertenece el gasto. |
| 11 | idEstado | NUMBER | No | Identificador del estado del gasto. |
| 12 | idFormaPago | NUMBER | Yes | Identificador de la forma de pago asociada al gasto (es función del estado del gasto). |
| 13 | idProveedor | NUMBER | Yes | Identificador del proveedor del gasto. |
| 14 | justificacion | VARCHAR | Yes | Texto (opcional) para justificar la necesidad del gasto. |
| 15 | presupuestoURL | VARCHAR | Yes | URL (opcional) al documento que contiene el presupuesto del gasto. |
| 16 | facturaURL | VARCHAR | Yes | URL (opcional) al documento que contiene la factura del gasto. |
| 17 | contratoURL | VARCHAR | Yes | URL (opcional) al documento que contiene el contrato del gasto. |

### 5.2.11. Tabla TipoIVA.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idTipoIVA | NUMBER | No | Identificador del tipo de IVA (secuencia autoincrementada). |
| 2 | porcentajeIVA | NUMBER | No | Porcentaje de IVA. |

### 5.2.12. Tabla CategoriaGasto.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idCategoriaGasto | NUMBER | No | Identificador de la categoría de gasto (secuencia autoincrementada). |
| 2 | descripcion | VARCHAR | No | Descripción de la categoría de gasto. |
| 3 | idTipoGasto | NUMBER | No | Identificador del tipo de gasto al que pertenece. |

### 5.2.13. Tabla TipoGasto.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idTipoGasto | NUMBER | No | Identificador del tipo de gasto (secuencia autoincrementada). |
| 2 | descripcion | VARCHAR | No | Texto descriptivo del tipo de gasto. |
| 3 | parlamentarioSN | BOOLEAN | No | Indica si el tipo de gasto requiere que el gasto sea imputado a un parlamentario. |

### 5.2.14. Tabla Proveedor.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idProveedor | NUMBER | No | Identificador del proveedor (secuencia autoincrementada). |
| 2 | nombre | VARCHAR | No | Nombre del proveedor. |
| 3 | CIF | VARCHAR | No | CIF del proveedor. |

### 5.2.15. Tabla Estado.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idEstado | NUMBER | No | Identificador del estado de un gasto (secuencia autoincrementada). |
| 2 | descripcion | VARCHAR | No | Descripción del estado de un gasto. |
| 3 | formaPagoSN | BOOLEAN | No | Indica si el estado del gasto requiere una forma de pago asociada. |

### 5.2.16. Tabla FormaPago.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idFormaPago | NUMBER | No | Identificador de la forma de pago (secuencia autoincrementada). |
| 2 | descripcion | VARCHAR | No | Descripción de la forma de pago. |

### 5.2.17. Tabla ParlamentoDH.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idParlamentoDH | NUMBER | No | Identificador del período del parlamentario en el parlamento (secuencia autoincrementada). |
| 2 | idParlamento | NUMBER | No | Identificador del parlamento. |
| 3 | idParlamentario | NUMBER | No | Identificador del parlamentario. |
| 4 | desde | DATE | No | Fecha de inicio del período de este parlamentario en este parlamento. |
| 5 | hasta | DATE | Yes | Fecha de finalización del período de este parlamentario en este parlamento. |

### 5.2.18. Tabla ParlamentoST.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idEjercicio | NUMBER | No | Identificador del ejercicio al que pertenece la estadística. |
| 2 | idParlamento | NUMBER | No | Identificador del parlamento al que pertenece la estadística. |
| 3 | totalGastosParlamentarios | NUMBER | No | Importe total de los gastos imputables a parlamentarios en este ejercicio y este parlamento. |
| 4 | totalGastosGenerales | NUMBER | No | Importe total de los gastos generales (no imputables a parlamentarios) de este ejercicio y este parlamento. |
| 5 | totalGastos | NUMBER | No | Suma del importe total de gastos de los parlamentarios e importe total de gastos generales. |
| 6 | numParlamentarios | NUMBER | No | Número de parlamentarios a los que se les han imputado gastos en este ejercicio y este parlamento. |
| 7 | minGastoParlamentario | NUMBER | No | Importe mínimo de gastos imputados a un parlamentario en este ejercicio y este parlamento. |
| 8 | mediaGastoParlamentario | NUMBER | No | Importe medio de gastos imputados a un parlamentario en este ejercicio y este parlamento. |
| 9 | maxGastoParlamentario | NUMBER | No | Importe máximo de gastos imputados a un parlamentario en este ejercicio y este parlamento. |

### 5.2.19. Tabla ParlamentarioST.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idEjercicio | NUMBER | No | Identificador del ejercicio al que pertenece la estadística. |
| 2 | idParlamento | NUMBER | No | Identificador del parlamento al que pertenece la estadística. |
| 3 | idParlamentario | NUMBER | No | Identificador del parlamentario al que pertenece la estadística. |
| 4 | totalGastos | NUMBER | No | Importe total de los gastos imputables a este parlamentario en este ejercicio en este parlamento. |

### 5.2.20. Tabla PartidoDH.

| | COLUMNA | TIPO | NULL | COMENTARIO |
|---|---|---|---|---|
| 1 | idPartidoDH | NUMBER | No | Identificador del período del parlamentario en el partido (secuencia autoincrementada). |
| 2 | idParlamentario | NUMBER | No | Identificador del parlamentario. |
| 3 | idPartido | NUMBER | No | Identificador del partido. |
| 4 | desde | DATE | No | Fecha de inicio del período de este parlamentario en este partido. |
| 5 | hasta | DATE | Yes | Fecha de finalización del período de este parlamentario en este partido. |

## 5.3. *Script* de creación y eliminación de la base de datos.

El *script* 1_1_CREAR_TABLESPACE_Y_USUARIO.SQL crea la base de datos y el usuario de la base de datos.

```
--
-- Conectado como SYS
--
-- Eliminamos el tablespace TFC si existiera.
DROP TABLESPACE TFC INCLUDING CONTENTS AND DATAFILES;
--
-- Creamos el tablespace TFC
--
CREATE TABLESPACE TFC
    DATAFILE 'TFC.DBF' SIZE 50 M
    DEFAULT NOCOMPRESS
    ONLINE
    EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
--
-- Eliminamos el usuario TFC si existiera.
--
DROP USER TFC CASCADE;
--
-- Creamos el usuario TFC
--
CREATE USER TFC
IDENTIFIED BY TFC
DEFAULT TABLESPACE TFC
QUOTA UNLIMITED ON TFC
TEMPORARY TABLESPACE TEMP
ACCOUNT UNLOCK ;
--
GRANT DBA TO TFC ;
--
```

El *script* 9_1_ELIMINAR_TABLESPACE_Y_USUARIO elimina finalmente la base de datos y el usuario.

```
--
-- Conectado como SYS
--
--
-- Eliminamos el tablespace TFC.
DROP TABLESPACE TFC INCLUDING CONTENTS AND DATAFILES;
--
-- Eliminamos el usuario TFC.
DROP USER TFC CASCADE;
--
```

## 5.4. *Script* de creación de las tablas de la base de datos.

El *script* 2_1_CREAR_TABLAS.SQL crea las tablas de la base de datos así como los disparadores que gestionan las secuencias autoincrementadas utilizadas en los identificadores de la mayor parte de las tablas.

```
--
-- Server version: Oracle Database 11g Express Edition Release 11.2.0.2.0 - Production
--

CREATE TABLE EJERCICIO
(
  IDEJERCICIO NUMBER(*, 0) NOT NULL,
  INICIO      DATE         NOT NULL,
  FIN         DATE         NOT NULL,
  CERRADOSN   CHAR(1 BYTE) NOT NULL,
  CONSTRAINT EJERCICIO_PK PRIMARY KEY (IDEJERCICIO)
);

COMMENT ON COLUMN EJERCICIO.CERRADOSN IS 'Indica si el ejercicio esta cerrado. Si el ejercicio está cerrado ya no se le pueden imputar gastos.';
COMMENT ON COLUMN EJERCICIO.FIN IS 'Fecha de fin del ejercicio.';
COMMENT ON COLUMN EJERCICIO.IDEJERCICIO IS 'Identificador del ejercicio. Secuencia manual que debe seguir el patrón: 2012, 2013, 2014, 2015, ...';
COMMENT ON COLUMN EJERCICIO.INICIO IS 'Fecha de inicio del ejercicio.';

CREATE TABLE ESTADO
(
  IDESTADO    NUMBER(*, 0)      NOT NULL,
  DESCRIPCION VARCHAR2(50 BYTE) NOT NULL,
  FORMAPAGOSN CHAR(1 BYTE)      NOT NULL,
  CONSTRAINT ESTADO_PK PRIMARY KEY (IDESTADO)
);

COMMENT ON COLUMN ESTADO.DESCRIPCION IS 'Descripción del estado de un gasto.';
COMMENT ON COLUMN ESTADO.FORMAPAGOSN IS 'Indica si el estado del gasto requiere una forma de pago asociada.';
COMMENT ON COLUMN ESTADO.IDESTADO IS 'Identificador del estado de un gasto (secuencia autoincrementada).';

CREATE TABLE FORMAPAGO
(
  IDFORMAPAGO NUMBER(*, 0)      NOT NULL,
  DESCRIPCION VARCHAR2(50 BYTE) NOT NULL,
  CONSTRAINT FORMAPAGO_PK PRIMARY KEY (IDFORMAPAGO)
);

COMMENT ON COLUMN FORMAPAGO.DESCRIPCION IS 'Descripción de la forma de pago.';
COMMENT ON COLUMN FORMAPAGO.IDFORMAPAGO IS 'Identificador de la forma de pago (secuencia autoincrementada).';

CREATE TABLE PAIS
(
  IDPAIS NUMBER(*, 0)      NOT NULL,
  NOMBRE VARCHAR2(50 BYTE) NOT NULL,
  CONSTRAINT PAIS_PK PRIMARY KEY (IDPAIS)
);
```

```
COMMENT ON COLUMN PAIS.IDPAIS IS 'Identificador del país (secuencia autoincrementada).';
COMMENT ON COLUMN PAIS.NOMBRE IS 'Nombre del país.';

CREATE TABLE PARLAMENTARIO
(
  IDPARLAMENTARIO NUMBER(*, 0)      NOT NULL,
  NOMBRE          VARCHAR2(50 BYTE) NOT NULL,
  APELLIDOS       VARCHAR2(50 BYTE) NOT NULL,
  CONSTRAINT PARLAMENTARIO PK PRIMARY KEY (IDPARLAMENTARIO)
);

COMMENT ON COLUMN PARLAMENTARIO.APELLIDOS IS 'Apellidos del parlamentario.';
COMMENT ON COLUMN PARLAMENTARIO.IDPARLAMENTARIO IS 'Identificador del parlamentario (secuencia autoincrementada).';
COMMENT ON COLUMN PARLAMENTARIO.NOMBRE IS 'Nombre del parlamentario.';

CREATE TABLE PARTIDO
(
  IDPARTIDO NUMBER(*, 0)      NOT NULL,
  NOMBRE    VARCHAR2(50 BYTE) NOT NULL,
  CONSTRAINT PARTIDO PK PRIMARY KEY (IDPARTIDO)
);

COMMENT ON COLUMN PARTIDO.IDPARTIDO IS 'Identificado del partido (secuencia autoincrementada).';
COMMENT ON COLUMN PARTIDO.NOMBRE IS 'Nombre del partido.';

CREATE TABLE PROVEEDOR
(
  IDPROVEEDOR NUMBER(*, 0)      NOT NULL,
  NOMBRE      VARCHAR2(50 BYTE) NOT NULL,
  CIF         VARCHAR2(20 BYTE) NOT NULL,
  CONSTRAINT PROVEEDOR PK PRIMARY KEY (IDPROVEEDOR),
  CONSTRAINT PROVEEDOR CIF UN UNIQUE (CIF)
);

COMMENT ON COLUMN PROVEEDOR.CIF IS 'CIF del proveedor.';
COMMENT ON COLUMN PROVEEDOR.IDPROVEEDOR IS 'Identificador del proveedor (secuencia autoincrementada).';
COMMENT ON COLUMN PROVEEDOR.NOMBRE IS 'Nombre del proveedor.';

CREATE TABLE REGISTRO
(
  IDREGISTRO    NUMBER(*, 0)        NOT NULL,
  PROCEDIMIENTO VARCHAR2(100 BYTE)  NOT NULL,
  USUARIO       VARCHAR2(50 BYTE)   NOT NULL,
  INSTANTE      TIMESTAMP(6)        NOT NULL,
  ENTRADA       VARCHAR2(3000 BYTE),
  SALIDA        VARCHAR2(3000 BYTE) NOT NULL,
  CONSTRAINT REGISTRO PK PRIMARY KEY (IDREGISTRO)
);

COMMENT ON COLUMN REGISTRO.ENTRADA IS 'String con el nombre de los parámetros de entrada y sus valores.';
COMMENT ON COLUMN REGISTRO.IDREGISTRO IS 'Identificador de la entrada en el registro de ejecuciones de procedimientos (secuencia autoincrementada).';
COMMENT ON COLUMN REGISTRO.INSTANTE IS 'Instante de tiempo en que se ejecutó el procedimiento.';
```

```
COMMENT ON COLUMN REGISTRO.PROCEDIMIENTO IS 'Nombre del procedimiento ejecutado.';
COMMENT ON COLUMN REGISTRO.SALIDA IS 'String con el nombre de los parámetros de salida y sus valores.';
COMMENT ON COLUMN REGISTRO.USUARIO IS 'Nombre del usuario que ejecutó el procedimiento.';

CREATE TABLE TIPOGASTO
(
  IDTIPOGASTO     NUMBER(*, 0)       NOT NULL,
  DESCRIPCION     VARCHAR2(50 BYTE) NOT NULL,
  PARLAMENTARIOSN CHAR(1 BYTE)       NOT NULL,
  CONSTRAINT TIPOGASTO PK PRIMARY KEY (IDTIPOGASTO)
);

COMMENT ON COLUMN TIPOGASTO.DESCRIPCION IS 'Texto descriptivo del tipo de gasto.';
COMMENT ON COLUMN TIPOGASTO.IDTIPOGASTO IS 'Identificador del tipo de gasto (secuencia autoincrementada).';
COMMENT ON COLUMN TIPOGASTO.PARLAMENTARIOSN IS 'Indica si el tipo de gasto requiere que el gasto sea imputado a un parlamentario.';

CREATE TABLE TIPOIVA
(
  IDTIPOIVA     NUMBER(*, 0) NOT NULL,
  PORCENTAJEIVA NUMBER       DEFAULT 0.0 NOT NULL,
  CONSTRAINT TIPOIVA PK PRIMARY KEY (IDTIPOIVA)
);

COMMENT ON COLUMN TIPOIVA.IDTIPOIVA IS 'Identificador del tipo de IVA (secuencia autoincrementada).';
COMMENT ON COLUMN TIPOIVA.PORCENTAJEIVA IS 'Porcentaje de IVA.';

CREATE TABLE TIPOPARLAMENTO
(
  IDTIPOPARLAMENTO NUMBER(*, 0)       NOT NULL,
  DESCRIPCION      VARCHAR2(50 BYTE) NOT NULL,
  CONSTRAINT TIPOPARLAMENTO PK PRIMARY KEY (IDTIPOPARLAMENTO)
);

COMMENT ON COLUMN TIPOPARLAMENTO.DESCRIPCION IS 'Texto descriptivo del tipo de parlamento.';
COMMENT ON COLUMN TIPOPARLAMENTO.IDTIPOPARLAMENTO IS 'Identificador del tipo de parlamento (secuencia autoincrementada).';

CREATE TABLE CATEGORIAGASTO
(
  IDCATEGORIAGASTO NUMBER(*, 0)       NOT NULL,
  DESCRIPCION      VARCHAR2(50 BYTE) NOT NULL,
  IDTIPOGASTO      NUMBER(*, 0)       NOT NULL,
  CONSTRAINT CATEGORIAGASTO PK PRIMARY KEY (IDCATEGORIAGASTO),
  CONSTRAINT CATEGORIAGASTO TIPOGASTO FK FOREIGN KEY (IDTIPOGASTO)
  REFERENCES TIPOGASTO (IDTIPOGASTO)
);

COMMENT ON COLUMN CATEGORIAGASTO.DESCRIPCION IS 'Descripción de la categoría de gasto.';
COMMENT ON COLUMN CATEGORIAGASTO.IDCATEGORIAGASTO IS 'Identificador de la categoría de gasto (secuencia autoincrementada).';
COMMENT ON COLUMN CATEGORIAGASTO.IDTIPOGASTO IS 'Identificador del tipo de gasto al que pertenece.';

CREATE TABLE PARTIDODH
(
  IDPARTIDODH     NUMBER(*, 0) NOT NULL,
```

```
  IDPARLAMENTARIO NUMBER(*, 0) NOT NULL,
  IDPARTIDO       NUMBER(*, 0) NOT NULL,
  DESDE           DATE         NOT NULL,
  HASTA           DATE,
  CONSTRAINT PARTIDODH PK PRIMARY KEY (IDPARTIDODH),
  CONSTRAINT PARTIDODH PARLAMENTARIO FK FOREIGN KEY (IDPARLAMENTARIO)
  REFERENCES PARLAMENTARIO (IDPARLAMENTARIO),
  CONSTRAINT PARTIDODH_PARTIDO_FK FOREIGN KEY (IDPARTIDO)
  REFERENCES PARTIDO (IDPARTIDO)
);

COMMENT ON COLUMN PARTIDODH.DESDE IS 'Fecha de inicio del período de este parlamentario en este partido.';
COMMENT ON COLUMN PARTIDODH.HASTA IS 'Fecha de finalización del período de este parlamentario en este partido.';
COMMENT ON COLUMN PARTIDODH.IDPARLAMENTARIO IS 'Identificador del parlamentario.';
COMMENT ON COLUMN PARTIDODH.IDPARTIDO IS 'Identificador del partido.';
COMMENT ON COLUMN PARTIDODH.IDPARTIDODH IS 'Identificador del período del parlamentario en el partido (secuencia autoincrementada).';

CREATE TABLE REGION
(
  IDREGION NUMBER(*, 0)      NOT NULL,
  NOMBRE   VARCHAR2(50 BYTE) NOT NULL,
  IDPAIS   NUMBER(*, 0)      NOT NULL,
  CONSTRAINT REGION_PK PRIMARY KEY (IDREGION),
  CONSTRAINT REGION_PAIS_FK FOREIGN KEY (IDPAIS)
  REFERENCES PAIS (IDPAIS)
);

COMMENT ON COLUMN REGION.IDPAIS IS 'Identificador del país al que pertenece la región.';
COMMENT ON COLUMN REGION.IDREGION IS 'Identificador de la región (secuencia autoincrementada).';
COMMENT ON COLUMN REGION.NOMBRE IS 'Nombre de la región.';

CREATE TABLE CIUDAD
(
  IDCIUDAD NUMBER(*, 0)      NOT NULL,
  NOMBRE   VARCHAR2(50 BYTE) NOT NULL,
  IDREGION NUMBER(*, 0)      NOT NULL,
  CONSTRAINT CIUDAD PK PRIMARY KEY (IDCIUDAD),
  CONSTRAINT CIUDAD REGION FK FOREIGN KEY (IDREGION)
  REFERENCES REGION (IDREGION)
);

COMMENT ON COLUMN CIUDAD.IDCIUDAD IS 'Identificador de la ciudad (secuencia autoincrementada).';
COMMENT ON COLUMN CIUDAD.IDREGION IS 'Identificador de la región a la que pertenece.';
COMMENT ON COLUMN CIUDAD.NOMBRE IS 'Nombre de la ciudad.';

CREATE TABLE PARLAMENTO
(
  IDPARLAMENTO      NUMBER(*, 0)      NOT NULL,
  NOMBRE            VARCHAR2(50 BYTE) NOT NULL,
  IDCIUDAD          NUMBER(*, 0)      NOT NULL,
  IDTIPOPARLAMENTO  NUMBER(*, 0)      NOT NULL,
  DIRECCION         VARCHAR2(50 BYTE) NOT NULL,
  CODIGOPOSTAL      VARCHAR2(20 BYTE) NOT NULL,
```

```
  TELEFONO          VARCHAR2(20 BYTE),
  WEB               VARCHAR2(200 BYTE),
  MAXPARLAMENTARIOS NUMBER(*, 0)      NOT NULL,
  CONSTRAINT PARLAMENTO PK PRIMARY KEY (IDPARLAMENTO),
  CONSTRAINT PARLAMENTO CIUDAD FK FOREIGN KEY (IDCIUDAD)
  REFERENCES CIUDAD (IDCIUDAD),
  CONSTRAINT PARLAMENTO_TIPOPARLAMENTO_FK FOREIGN KEY (IDTIPOPARLAMENTO)
  REFERENCES TIPOPARLAMENTO (IDTIPOPARLAMENTO)
);

COMMENT ON COLUMN PARLAMENTO.CODIGOPOSTAL IS 'Código postal del parlamento.';
COMMENT ON COLUMN PARLAMENTO.DIRECCION IS 'Dirección del parlamento.';
COMMENT ON COLUMN PARLAMENTO.IDCIUDAD IS 'Identificador de la ciudad donde está situado el parlamento.';
COMMENT ON COLUMN PARLAMENTO.IDPARLAMENTO IS 'Identificador del parlamento (secuencia autoincrementada).';
COMMENT ON COLUMN PARLAMENTO.IDTIPOPARLAMENTO IS 'Identificador del tipo de parlamento.';
COMMENT ON COLUMN PARLAMENTO.MAXPARLAMENTARIOS IS 'Número máximo de parlamentarios del parlamento.';
COMMENT ON COLUMN PARLAMENTO.NOMBRE IS 'Nombre del parlamento.';
COMMENT ON COLUMN PARLAMENTO.TELEFONO IS 'Teléfono (centralita) del parlamento.';
COMMENT ON COLUMN PARLAMENTO.WEB IS 'URL de la página web del parlamento.';

CREATE TABLE GASTO
(
  IDGASTO          NUMBER(*, 0) NOT NULL,
  FECHA            DATE         NOT NULL,
  IDEJERCICIO      NUMBER(*, 0) NOT NULL,
  IDPARLAMENTO     NUMBER(*, 0),
  IDPARLAMENTARIO  NUMBER(*, 0),
  IDPARTIDO        NUMBER(*, 0),
  DESCRIPCION      VARCHAR2(500 BYTE),
  BASEIMPONIBLE    NUMBER       DEFAULT 0.0 NOT NULL,
  IDTIPOIVA        NUMBER(*, 0) NOT NULL,
  IDCATEGORIAGASTO NUMBER(*, 0) NOT NULL,
  IDESTADO         NUMBER(*, 0) NOT NULL,
  IDFORMAPAGO      NUMBER(*, 0),
  IDPROVEEDOR      NUMBER(*, 0),
  JUSTIFICACION    VARCHAR2(500 BYTE),
  PRESUPUESTOURL   VARCHAR2(200 BYTE),
  FACTURAURL       VARCHAR2(200 BYTE),
  CONTRATOURL      VARCHAR2(200 BYTE),
  CONSTRAINT GASTO_PK PRIMARY KEY (IDGASTO),
  CONSTRAINT GASTO_CATEGORIAGASTO_FK FOREIGN KEY (IDCATEGORIAGASTO)
  REFERENCES CATEGORIAGASTO (IDCATEGORIAGASTO),
  CONSTRAINT GASTO EJERCICIO FK FOREIGN KEY (IDEJERCICIO)
  REFERENCES EJERCICIO (IDEJERCICIO),
  CONSTRAINT GASTO_ESTADO_FK FOREIGN KEY (IDESTADO)
  REFERENCES ESTADO (IDESTADO),
  CONSTRAINT GASTO_FORMAPAGO_FK FOREIGN KEY (IDFORMAPAGO)
  REFERENCES FORMAPAGO (IDFORMAPAGO),
  CONSTRAINT GASTO PARLAMENTARIO FK FOREIGN KEY (IDPARLAMENTARIO)
  REFERENCES PARLAMENTARIO (IDPARLAMENTARIO),
  CONSTRAINT GASTO_PARLAMENTO_FK FOREIGN KEY (IDPARLAMENTO)
  REFERENCES PARLAMENTO (IDPARLAMENTO),
  CONSTRAINT GASTO_PARTIDO_FK FOREIGN KEY (IDPARTIDO)
```

```
  REFERENCES PARTIDO (IDPARTIDO),
  CONSTRAINT GASTO_PROVEEDOR_FK FOREIGN KEY (IDPROVEEDOR)
  REFERENCES PROVEEDOR (IDPROVEEDOR),
  CONSTRAINT GASTO TIPOIVA FK FOREIGN KEY (IDTIPOIVA)
  REFERENCES TIPOIVA (IDTIPOIVA)
);

COMMENT ON COLUMN GASTO.BASEIMPONIBLE IS 'Base imponible del gasto.';
COMMENT ON COLUMN GASTO.CONTRATOURL IS 'URL (opcional) al documento que contiene el contrato del gasto.';
COMMENT ON COLUMN GASTO.DESCRIPCION IS 'Texto descriptivo del gasto.';
COMMENT ON COLUMN GASTO.FACTURAURL IS 'URL (opcional) al documento que contiene la factura del gasto.';
COMMENT ON COLUMN GASTO.FECHA IS 'Fecha del gasto.';
COMMENT ON COLUMN GASTO.IDCATEGORIAGASTO IS 'Identificador de la categoría de gasto al que pertenece el gasto.';
COMMENT ON COLUMN GASTO.IDEJERCICIO IS 'Ejercicio al que pertenece el gasto (es función de la fecha).';
COMMENT ON COLUMN GASTO.IDESTADO IS 'Identificador del estado del gasto.';
COMMENT ON COLUMN GASTO.IDFORMAPAGO IS 'Identificador de la forma de pago asociada al gasto (es función del estado del gasto).';
COMMENT ON COLUMN GASTO.IDGASTO IS 'Identificador del gasto (secuencia autoincrementada).';
COMMENT ON COLUMN GASTO.IDPARLAMENTARIO IS 'Identificador del parlamentario (si lo hay) al que debe imputarse el gasto.';
COMMENT ON COLUMN GASTO.IDPARLAMENTO IS 'Identificador del parlamento al que debe imputarse el gasto.';
COMMENT ON COLUMN GASTO.IDPARTIDO IS 'Identificador del partido al que pertenece el parlamentario (si lo hay) al que debe imputarse el gasto.';
COMMENT ON COLUMN GASTO.IDPROVEEDOR IS 'Identificador del proveedor del gasto.';
COMMENT ON COLUMN GASTO.IDTIPOIVA IS 'Identificador del tipo de IVA aplicable al gasto.';
COMMENT ON COLUMN GASTO.JUSTIFICACION IS 'Texto (opcional) para justificar la necesidad del gasto.';
COMMENT ON COLUMN GASTO.PRESUPUESTOURL IS 'URL (opcional) al documento que contiene el presupuesto del gasto.';

CREATE TABLE PARLAMENTARIOST
(
  IDEJERCICIO     NUMBER(*, 0) NOT NULL,
  IDPARLAMENTO    NUMBER(*, 0) NOT NULL,
  IDPARLAMENTARIO NUMBER(*, 0) NOT NULL,
  TOTALGASTOS     NUMBER       DEFAULT 0.0 NOT NULL,
  CONSTRAINT PARLAMENTARIOST PK PRIMARY KEY (IDEJERCICIO,
  IDPARLAMENTO,
  IDPARLAMENTARIO),
  CONSTRAINT PARLAMENTARIOST_EJERCICIO_FK FOREIGN KEY (IDEJERCICIO)
  REFERENCES EJERCICIO (IDEJERCICIO),
  CONSTRAINT PARLAMENTARIOST ID FK FOREIGN KEY (IDPARLAMENTARIO)
  REFERENCES PARLAMENTARIO (IDPARLAMENTARIO),
  CONSTRAINT PARLAMENTARIOST PARLAMENTO FK FOREIGN KEY (IDPARLAMENTO)
  REFERENCES PARLAMENTO (IDPARLAMENTO)
);

COMMENT ON COLUMN PARLAMENTARIOST.IDEJERCICIO IS 'Identificador del ejercicio al que pertenece la estadística.';
COMMENT ON COLUMN PARLAMENTARIOST.IDPARLAMENTARIO IS 'Identificador del parlamentario al que pertenece la estadística.';
COMMENT ON COLUMN PARLAMENTARIOST.IDPARLAMENTO IS 'Identificador del parlamento al que pertenece la estadística.';
COMMENT ON COLUMN PARLAMENTARIOST.TOTALGASTOS IS 'Importe total de los gastos imputables a este parlamentario en este ejercicio en este parlamento.';

CREATE TABLE PARLAMENTODH
(
  IDPARLAMENTODH  NUMBER(*, 0) NOT NULL,
  IDPARLAMENTO    NUMBER(*, 0) NOT NULL,
  IDPARLAMENTARIO NUMBER(*, 0) NOT NULL,
  DESDE           DATE         NOT NULL,
```

```
  HASTA           DATE,
  CONSTRAINT PARLAMENTODH_PK PRIMARY KEY (IDPARLAMENTODH),
  CONSTRAINT PARLAMENTODH_PARLAMENTARIO_FK FOREIGN KEY (IDPARLAMENTARIO)
  REFERENCES PARLAMENTARIO (IDPARLAMENTARIO),
  CONSTRAINT PARLAMENTODH PARLAMENTO FK FOREIGN KEY (IDPARLAMENTO)
  REFERENCES PARLAMENTO (IDPARLAMENTO)
);

COMMENT ON COLUMN PARLAMENTODH.DESDE IS 'Fecha de inicio del período de este parlamentario en este parlamento.';
COMMENT ON COLUMN PARLAMENTODH.HASTA IS 'Fecha de finalización del período de este parlamentario en este parlamento.';
COMMENT ON COLUMN PARLAMENTODH.IDPARLAMENTARIO IS 'Identificador del parlamentario.';
COMMENT ON COLUMN PARLAMENTODH.IDPARLAMENTO IS 'Identificador del parlamento.';
COMMENT ON COLUMN PARLAMENTODH.IDPARLAMENTODH IS 'Identificador del período del parlamentario en el parlamento (secuencia autoincrementada).';

CREATE TABLE PARLAMENTOST (
  IDEJERCICIO NUMBER(*, 0) NOT NULL,
  IDPARLAMENTO NUMBER(*, 0) NOT NULL,
  TOTALGASTOSPARLAMENTARIOS NUMBER DEFAULT 0.0 NOT NULL,
  TOTALGASTOSGENERALES NUMBER DEFAULT 0.0 NOT NULL,
  TOTALGASTOS AS (TOTALGASTOSPARLAMENTARIOS+TOTALGASTOSGENERALES) NOT NULL,
  NUMPARLAMENTARIOS NUMBER(*, 0) DEFAULT 0 NOT NULL,
  MINGASTOPARLAMENTARIO NUMBER DEFAULT 0.0 NOT NULL,
  MEDIAGASTOPARLAMENTARIO AS (MEDIA(TOTALGASTOSPARLAMENTARIOS,NUMPARLAMENTARIOS)) NOT NULL,
  MAXGASTOPARLAMENTARIO NUMBER DEFAULT 0.0 NOT NULL,
  CONSTRAINT PARLAMENTOST_PK PRIMARY KEY (IDEJERCICIO, IDPARLAMENTO),
  CONSTRAINT PARLAMENTOST_EJERCICIO_FK FOREIGN KEY (IDEJERCICIO)
    REFERENCES EJERCICIO(IDEJERCICIO),
  CONSTRAINT PARLAMENTOST PARLAMENTO FK FOREIGN KEY (IDPARLAMENTO)
    REFERENCES PARLAMENTO(IDPARLAMENTO))
);

COMMENT ON COLUMN PARLAMENTOST.IDEJERCICIO IS 'Identificador del ejercicio al que pertenece la estadística.';
COMMENT ON COLUMN PARLAMENTOST.IDPARLAMENTO IS 'Identificador del parlamento al que pertenece la estadística.';
COMMENT ON COLUMN PARLAMENTOST.MAXGASTOPARLAMENTARIO IS 'Importe máximo de gastos imputados a un parlamentario en este ejercicio y este parlamento.';
COMMENT ON COLUMN PARLAMENTOST.MEDIAGASTOPARLAMENTARIO IS 'Importe medio de gastos imputados a un parlamentario en este ejercicio y este parlamento.';
COMMENT ON COLUMN PARLAMENTOST.MINGASTOPARLAMENTARIO IS 'Importe mínimo de gastos imputados a un parlamentario en este ejercicio y este parlamento.';
COMMENT ON COLUMN PARLAMENTOST.NUMPARLAMENTARIOS IS 'Número de parlamentarios a los que se les han imputado gastos en este ejercicio y este parlamento.';
COMMENT ON COLUMN PARLAMENTOST.TOTALGASTOS IS 'Suma del importe total de gastos de los parlamentarios e importe total de gastos generales.';
COMMENT ON COLUMN PARLAMENTOST.TOTALGASTOSGENERALES IS 'Importe total de los gastos generales (no imputables a parlamentarios) de este ejercicio y este
parlamento.';
COMMENT ON COLUMN PARLAMENTOST.TOTALGASTOSPARLAMENTARIOS IS 'Importe total de los gastos imputables a parlamentarios en este ejercicio y este parlamento.';

CREATE OR REPLACE FUNCTION media (
  numerador   IN NUMBER,
  denominador IN NUMBER)
  RETURN NUMBER
  DETERMINISTIC
AS
BEGIN
  IF (denominador <> 0)
  THEN
    RETURN (numerador / denominador);
  ELSE
```

```
    RETURN (0);
  END IF;
END;

CREATE OR REPLACE TRIGGER IDCATEGORIAGASTO
  BEFORE INSERT ON CATEGORIAGASTO
  FOR EACH ROW
  WHEN (NEW.idCategoriaGasto IS NULL)
BEGIN
  :NEW.idCategoriaGasto := idCategoriaGasto SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDCIUDAD
  BEFORE INSERT ON CIUDAD
  FOR EACH ROW
  WHEN (NEW.idCiudad IS NULL)
BEGIN
  :NEW.idCiudad := idCiudad_SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDESTADO
  BEFORE INSERT ON ESTADO
  FOR EACH ROW
  WHEN (NEW.idEstado IS NULL)
BEGIN
  :NEW.idEstado := idEstado_SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDFORMAPAGO
  BEFORE INSERT ON FORMAPAGO
  FOR EACH ROW
  WHEN (NEW.idFormaPago IS NULL)
BEGIN
  :NEW.idFormaPago := idFormaPago SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDGASTO
  BEFORE INSERT ON GASTO
  FOR EACH ROW
  WHEN (NEW.idGasto IS NULL)
BEGIN
  :NEW.idGasto := idGasto SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDPAIS
  BEFORE INSERT ON PAIS
  FOR EACH ROW
  WHEN (NEW.idPais IS NULL)
BEGIN
  :NEW.idPais := idPais SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDPARLAMENTARIO
```

```
  BEFORE INSERT ON PARLAMENTARIO
  FOR EACH ROW
  WHEN (NEW.idParlamentario IS NULL)
BEGIN
  :NEW.idParlamentario := idParlamentario SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDPARLAMENTO
  BEFORE INSERT ON PARLAMENTO
  FOR EACH ROW
  WHEN (NEW.idParlamento IS NULL)
BEGIN
  :NEW.idParlamento := idParlamento_SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDPARLAMENTODH
  BEFORE INSERT ON PARLAMENTODH
  FOR EACH ROW
  WHEN (NEW.idParlamentoDH IS NULL)
BEGIN
  :NEW.idParlamentoDH := idParlamentoDH SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDPARTIDO
  BEFORE INSERT ON PARTIDO
  FOR EACH ROW
  WHEN (NEW.idPartido IS NULL)
BEGIN
  :NEW.idPartido := idPartido_SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDPARTIDODH
  BEFORE INSERT ON PARTIDODH
  FOR EACH ROW
  WHEN (NEW.idPartidoDH IS NULL)
BEGIN
  :NEW.idPartidoDH := idPartidoDH SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDPROVEEDOR
  BEFORE INSERT ON PROVEEDOR
  FOR EACH ROW
  WHEN (NEW.idProveedor IS NULL)
BEGIN
  :NEW.idProveedor := idProveedor_SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDREGION
  BEFORE INSERT ON REGION
  FOR EACH ROW
  WHEN (NEW.idRegion IS NULL)
BEGIN
  :NEW.idRegion := idRegion_SQ.NEXTVAL;
```

```
END;

CREATE OR REPLACE TRIGGER ID
  BEFORE INSERT ON REGISTRO
  FOR EACH ROW
  WHEN (NEW.idRegistro IS NULL)
BEGIN
  :NEW.idRegistro := idRegistro_SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDTIPOGASTO
  BEFORE INSERT ON TIPOGASTO
  FOR EACH ROW
  WHEN (NEW.idTipoGasto IS NULL)
BEGIN
  :NEW.idTipoGasto := idTipoGasto SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDTIPOIVA
  BEFORE INSERT ON TIPOIVA
  FOR EACH ROW
  WHEN (NEW.idTipoIVA IS NULL)
BEGIN
  :NEW.idTipoIVA := idTipoIVA_SQ.NEXTVAL;
END;

CREATE OR REPLACE TRIGGER IDTIPOPARLAMENTO
  BEFORE INSERT ON TIPOPARLAMENTO
  FOR EACH ROW
  WHEN (NEW.idTipoParlamento IS NULL)
BEGIN
  :NEW.idTipoParlamento := idTipoParlamento SQ.NEXTVAL;
END;
```

## 5.5. Procedimientos *PL/SQL*.

Los procedimientos *PL/SQL* desarrollados están organizados en seis paquetes:

- pkg_parlamento: Paquete de procedimientos de altas, bajas y modificaciones en la tabla Parlamento.
- pkg_parlamentario: Paquete de procedimientos de altas, bajas y modificaciones en la tabla Parlamentario y tablas relacionadas ParlamentoDH y PartidoDH.
- pkg_gasto: Paquete de procedimientos de altas, bajas y modificaciones en la tabla Gasto y las tablas relacionadas ParlamentoST y ParlamentarioST.
- pkg_listado: Paquete de procedimientos que realizan los listados solicitados.
- pkg_estadistica: Paquete de procedimientos que calculan las estadísticas solicitadas.
- pkg_comun: Paquete de procedimientos comunes de utilidad.

La utilización de paquetes introduce la modularidad en el diseño que a su vez favorece la organización y reduce la complejidad. La creación de paquetes también favorece la reutilización de código y permite cierto grado de encapsulamiento, es decir, ocultación de los detalles de la implementación a los potenciales usuarios del paquete reduciendo la probabilidad de la aparición de errores al utilizarlo.

También se ha utilizado el polimorfismo en la programación de algunos procedimientos. Mediante el uso del polimorfismo, diversos procedimientos que comparten un mismo nombre, pueden tener implementaciones distintas de la misma tarea en función de que reciban un número distinto de parámetros o parámetros de distinto tipo.

Por ejemplo, se ha utilizado el polimorfismo en la implementación de algunos procedimientos del paquete pkg_comun en el cual existen tres implementaciones del procedimiento existe_parlamentarioST:

- existe_parlamentarioST ( idEjercicio, idParlamento, idParlamentario )
- existe_parlamentarioST ( idEjercicio, idParlamentario )
- existe_parlamentarioST ( idEjercicio )

De no existir la posibilidad de utilizar polimorfismo se tendrían que haber utilizado nombres distintos para procedimientos que realizan la misma tarea.

Por otra parte, en la programación de los procedimientos se utiliza una estructura interna común en todos ellos compuesta de dos bloques anidados para el tratamiento de excepciones. El bloque externo inicializa al principio el indicador de error y al final construye los *strings* entrada y salida antes de registrar el resultado de la ejecución del procedimiento. El bloque interno realiza las tareas propias del procedimiento y gestiona las posibles excepciones modificando el indicador de error si se captura alguna excepción. Así obtenemos procedimientos más compactos y sin repetición de código.

Finalmente, en la cabecera de especificación de la interfaz de cada paquete se ha documentado cada procedimiento conforme a las reglas que permiten utilizar *PLDOC*, una utilidad que permite generar automáticamente documentación en formato *HTML* del código escrito en *Oracle PL/SQL*.

### 5.5.1. Paquete pkg_parlamento.

El *script* 3_6_CREAR_PKG_PARLAMENTO.SQL crea en la base de datos el paquete pkg_parlamento de procedimientos de altas, bajas y modificaciones en la tabla Parlamento.

```
-- PKG PARLAMENTO specification
CREATE OR REPLACE PACKAGE TFC.pkg parlamento AS

ErrorNumeroParlamentarios EXCEPTION;

/**
Procedimiento para dar de alta un parlamento en la base de datos.

Al insertar una fila en la base de datos, el atributo idParlamento
se asigna automáticamente mediante una secuencia.

@param p nombre                        Nombre del parlamento.
@param p idCiudad                      Identificador de la ciudad donde tiene la sede el parlamento.
@param p_idTipoParlamento              Identificador del tipo de parlamento.
@param p_direccion                     Dirección de la sede del parlamento.
@param p_codigoPostal                  Código postal de la sede del parlamento.
@param p telefono                      Teléfono de la centralita de la sede del parlamento.
@param p web                           URL de la web del parlamento.
@param p maxParlamentarios             Número máximo de parlamentarios del parlamento.
@param p_rsp                           Resultado de la ejecución del procedimiento.
*/
PROCEDURE alta (
  p nombre            IN  Parlamento.nombre % TYPE,
  p idCiudad          IN  Parlamento.idCiudad % TYPE,
  p idTipoParlamento  IN  Parlamento.idTipoParlamento % TYPE,
  p_direccion         IN  Parlamento.direccion % TYPE,
  p_codigoPostal      IN  Parlamento.codigoPostal % TYPE,
  p_telefono          IN  Parlamento.telefono % TYPE,
  p web               IN  Parlamento.web % TYPE,
  p maxParlamentarios IN  Parlamento.maxParlamentarios % TYPE,
  p_rsp               OUT VARCHAR2);

/**
Procedimiento para dar de baja un parlamentario en la base de datos.

@param p idParlamento                  Identificador del parlamento a dar de baja.
@param p_rsp                           Resultado de la ejecución del procedimiento.
*/
PROCEDURE baja (
  p idParlamento IN  Parlamento.idParlamento % TYPE,
  p rsp          OUT VARCHAR2);

/**
Procedimiento para modificar un parlamento en la base de datos.

@param p idParlamento                  Identificador del parlamento a modificar.
```

```
@param p_nombre                      Nombre del parlamento.
@param p_idCiudad                    Identificador de la ciudad donde tiene la sede el parlamento.
@param p_idTipoParlamento            Identificador del tipo de parlamento.
@param p direccion                   Dirección de la sede del parlamento.
@param p codigoPostal                Código postal de la sede del parlamento.
@param p telefono                    Teléfono de la centralita de la sede del parlamento.
@param p_web                         URL de la web del parlamento.
@param p_maxParlamentarios           Número máximo de parlamentarios del parlamento.
@param p_rsp                         Resultado de la ejecución del procedimiento.

@throws ErrorNumeroParlamentarios    Si el número de parlamentarios nuevo es inferior al número de parlamentarios activos.
*/
PROCEDURE modificacion (
  p_idParlamento      IN  Parlamento.idParlamento % TYPE,
  p_nombre            IN  Parlamento.nombre % TYPE,
  p idCiudad          IN  Parlamento.idCiudad % TYPE,
  p_idTipoParlamento  IN  Parlamento.idTipoParlamento % TYPE,
  p_direccion         IN  Parlamento.direccion % TYPE,
  p_codigoPostal      IN  Parlamento.codigoPostal % TYPE,
  p_telefono          IN  Parlamento.telefono % TYPE,
  p web               IN  Parlamento.web % TYPE,
  p maxParlamentarios IN  Parlamento.maxParlamentarios % TYPE,
  p_rsp               OUT VARCHAR2);

END pkg_parlamento;
```

#### 5.5.1.1. Procedimiento pkg_parlamento.alta.

```
--
-- Procedimiento para dar de alta un parlamento en la base de datos.
--
PROCEDURE alta (
  p_nombre            IN  Parlamento.nombre % TYPE,
  p_idCiudad          IN  Parlamento.idCiudad % TYPE,
  p idTipoParlamento  IN  Parlamento.idTipoParlamento % TYPE,
  p direccion         IN  Parlamento.direccion % TYPE,
  p codigoPostal      IN  Parlamento.codigoPostal % TYPE,
  p_telefono          IN  Parlamento.telefono % TYPE,
  p_web               IN  Parlamento.web % TYPE,
  p_maxParlamentarios IN  Parlamento.maxParlamentarios % TYPE,
  p rsp               OUT VARCHAR2)
AS
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p_rsp := 'OK';
  BEGIN
    INSERT INTO Parlamento
        (nombre,
         IdCiudad,
         IdTipoParlamento,
         direccion,
         codigoPostal,
         telefono,
         web,
         maxParlamentarios)
      VALUES
        (p_nombre,
         p idCiudad,
         p idTipoParlamento,
         p_direccion,
         p_codigoPostal,
         p_telefono,
         p_web,
         p maxParlamentarios);
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN OTHERS THEN
      p rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'nombre = [' || p nombre || '] + idCiudad = [' || p idCiudad || '] + idTipoParlamento = [' || p idTipoParlamento || '] + direccion = [' ||
p direccion || '] + codigoPostal = [' || p codigoPostal || '] + telefono = [' || p telefono || '] + web = [' || p web || '] + maxParlamentarios = [' ||
p maxParlamentarios || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p_rsp;
```

```
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_parlamento.alta', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END alta;
```

### 5.5.1.2. Procedimiento pkg_parlamento.baja.

```
--
-- Procedimiento para dar de baja un parlamento en la base de datos.
--
PROCEDURE baja (
  p_idParlamento IN  Parlamento.idParlamento % TYPE,
  p_rsp          OUT VARCHAR2)
AS
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p_rsp := 'OK';
  BEGIN
    -- Controla que el Parlamento a dar de baja exista en la base de datos
    pkg comun.existe parlamento (p idParlamento);
    DELETE FROM
        Parlamento
      WHERE
        idParlamento = p_idParlamento;
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN pkg_comun.ErrorParlamentoNoExiste THEN
      p_rsp := 'ERROR: El parlamento [' || p_idParlamento || '] no existe.';
    WHEN OTHERS THEN
      p rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idParlamento = [' || p_idParlamento || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_parlamento.baja', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END baja;
```

### 5.5.1.3. Procedimiento pkg_parlamento.modificacion.

```
--
-- Procedimiento para modificar un parlamento en la base de datos.
--
PROCEDURE modificacion (
  p_idParlamento      IN  Parlamento.idParlamento % TYPE,
  p_nombre            IN  Parlamento.nombre % TYPE,
  p idCiudad          IN  Parlamento.idCiudad % TYPE,
  p idTipoParlamento  IN  Parlamento.idTipoParlamento % TYPE,
  p direccion         IN  Parlamento.direccion % TYPE,
  p_codigoPostal      IN  Parlamento.codigoPostal % TYPE,
  p_telefono          IN  Parlamento.telefono % TYPE,
  p_web               IN  Parlamento.web % TYPE,
  p maxParlamentarios IN  Parlamento.maxParlamentarios % TYPE,
  p rsp               OUT VARCHAR2)
AS
  numParlamentariosActuales NUMBER;
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p rsp := 'OK';
  BEGIN
    -- Controla que el Parlamento a modificar exista en la base de datos
    pkg_comun.existe_parlamento (p_idParlamento);
    -- Controla que el nuevo número máximo de Parlamentarios p_maxParlamentarios no sea menor que el número de Parlamentarios activos actualmente en ese
Parlamento
    SELECT
      COUNT (*)
    INTO
      numParlamentariosActuales
    FROM
      ParlamentoDH
    WHERE
      idParlamento = p_idParlamento
      AND
      hasta IS NULL;

    IF (p maxParlamentarios < numParlamentariosActuales)
    THEN
      RAISE ErrorNumeroParlamentarios;
    END IF;

    -- Realiza el update en la tabla Parlamento
    UPDATE Parlamento SET
      nombre = p_nombre,
      IdCiudad = p_idCiudad,
      IdTipoParlamento = p_idTipoParlamento,
      direccion = p direccion,
      codigoPostal = p codigoPostal,
      telefono = p telefono,
      web = p_web,
      maxParlamentarios = p_maxParlamentarios
```

```
    WHERE
      idParlamento = p_idParlamento;
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN ErrorNumeroParlamentarios THEN
      p_rsp := 'ERROR: El número de parlamentarios del parlamento [' || p_idParlamento || '] no puede ser inferior a [' || numParlamentariosActuales || ']
parlamentarios actuales.';
    WHEN pkg_comun.ErrorParlamentoNoExiste THEN
      p rsp := 'ERROR: El parlamento [' || p idParlamento || '] no existe.';
    WHEN OTHERS THEN
      p rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idParlamento = [' || p_idParlamento || '] + nombre = [' || p_nombre || '] + idCiudad = [' || p_idCiudad || '] + idTipoParlamento = [' ||
p_idTipoParlamento || '] + direccion = [' || p_direccion || '] + codigoPostal = [' || p_codigoPostal || '] + telefono = [' || p_telefono || '] + web = [' ||
p_web || '] + maxParlamentarios = [' || p_maxParlamentarios || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_parlamento.modificacion', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END modificacion;
```

### 5.5.2. Paquete pkg_parlamentario.

El *script* 3_5_CREAR_PKG_PARLAMENTARIO.SQL crea en la base de datos el paquete pkg_parlamentario de procedimientos de altas, bajas y modificaciones en la tabla Parlamentario y tablas relacionadas ParlamentoDH y PartidoDH.

La solución implementada mediante las tablas ParlamentoDH (que relaciona las tablas Parlamento y Parlamentario) y PartidoDH (que relaciona las tablas Partido y Parlamentario) es una generalización del modelo de datos solicitado. Permite controlar los diversos períodos que un parlamentario está en un parlamento y en un partido. Un parlamentario puede pasar períodos en distintos parlamentos. Puede estar incluso en el mismo parlamento en varios períodos distintos. Pero en una fecha determinada sólo puede estar en un Parlamento como máximo. Análogamente, un parlamentario puede pasar períodos en distintos partidos. Puede estar incluso en el mismo partido en varios períodos distintos. Pero en una fecha determinada sólo puede estar en un Partido como máximo.

```
-- PKG_PARLAMENTARIO specification
CREATE OR REPLACE PACKAGE TFC.pkg_parlamentario AS

ErrorParlamentoDistinto EXCEPTION;
ErrorPartidoDistinto EXCEPTION;
ErrorDesdeParlamentoDH EXCEPTION;
ErrorHastaParlamentoDH EXCEPTION;
ErrorDesdePartidoDH EXCEPTION;
ErrorHastaPartidoDH EXCEPTION;
ErrorExisteGasto EXCEPTION;

/**
Procedimiento para dar de alta un parlamentario en la base de datos.

Al insertar una fila en la base de datos, el atributo idParlamentario
se asigna automáticamente mediante una secuencia.

@param p nombre             Nombre del parlamentario.
@param p_apellidos          Apellidos del parlamentario.
@param p_rsp                Resultado de la ejecución del procedimiento.
*/
PROCEDURE alta (
  p nombre    IN  Parlamentario.nombre % TYPE,
  p_apellidos IN  Parlamentario.apellidos % TYPE,
  p_rsp       OUT VARCHAR2);

/**
Procedimiento para dar de baja un parlamentario en la base de datos.

@param p_idParlamentario    Identificador del parlamentario a dar de baja.
@param p_rsp                Resultado de la ejecución del procedimiento.
*/
PROCEDURE baja (
  p idParlamentario IN  Parlamentario.idParlamentario % TYPE,
  p rsp             OUT VARCHAR2);

/**
Procedimiento para modificar un parlamentario en la base de datos.
```

```
@param p_idParlamentario    Identificador del parlamentario a modificar.
@param p_nombre             Nombre del parlamentario.
@param p apellidos          Apellidos del parlamentario.
@param p rsp                Resultado de la ejecución del procedimiento.
*/
PROCEDURE modificacion (
  p_idParlamentario IN  Parlamentario.idParlamentario % TYPE,
  p_nombre          IN  Parlamentario.nombre % TYPE,
  p apellidos       IN  Parlamentario.apellidos % TYPE,
  p rsp             OUT VARCHAR2);

/**
Procedimiento para asignar un parlamentario a un parlamento.

@param p idParlamentario          Identificador del parlamentario.
@param p_idParlamento             Identificador del parlamento.
@param p_desde                    Fecha de inicio del período del parlamentario en el parlamento.
@param p_rsp                      Resultado de la ejecución del procedimiento.

@throws ErrorDesdeParlamentoDH    Si la fecha de inicio del período es anterior a la fecha de finalización del último período.
*/
PROCEDURE parlamento_desde (
  p_idParlamentario IN  ParlamentoDH.idParlamentario % TYPE,
  p_idParlamento    IN  ParlamentoDH.idParlamento % TYPE,
  p_desde           IN  ParlamentoDH.desde % TYPE,
  p rsp             OUT VARCHAR2);

/**
Procedimiento para desvincular a un parlamentario de un parlamento.

@param p idParlamentario          Identificador del parlamentario.
@param p idParlamento             Identificador del parlamento.
@param p hasta                    Fecha de finalización del período del parlamentario en el parlamento.
@param p_rsp                      Resultado de la ejecución del procedimiento.

@throws ErrorParlamentoDistinto   Si el parlamentario ya estaba vinculado a un parlamento distinto al especificado.
@throws ErrorHastaParlamentoDH    Si la fecha de final del período es anterior a la fecha de inicio del mismo.
@throws ErrorExisteGasto          Si existe algún gasto imputado al parlamentario posterior a la fecha de final del período.
*/
PROCEDURE parlamento_hasta (
  p idParlamentario IN  ParlamentoDH.idParlamentario % TYPE,
  p idParlamento    IN  ParlamentoDH.idParlamento % TYPE,
  p hasta           IN  ParlamentoDH.hasta % TYPE,
  p_rsp             OUT VARCHAR2);

/**
Procedimiento para asignar un parlamentario a un partido.

@param p idParlamentario          Identificador del parlamentario.
@param p_idPartido                Identificador del partido.
@param p_desde                    Fecha de inicio del período del parlamentario en el partido.
@param p_rsp                      Resultado de la ejecución del procedimiento.
```

```
@throws ErrorDesdePartidoDH       Si la fecha de inicio del período es anterior a la fecha de finalización del último período.
*/
PROCEDURE partido desde (
  p idParlamentario IN  PartidoDH.idParlamentario % TYPE,
  p idPartido       IN  PartidoDH.idPartido % TYPE,
  p_desde           IN  PartidoDH.desde % TYPE,
  p_rsp             OUT VARCHAR2);

/**
Procedimiento para desvincular a un parlamentario de un partido.

@param p_idParlamentario          Identificador del parlamentario.
@param p_idPartido                Identificador del partido.
@param p_hasta                    Fecha de finalización del período del parlamentario en el partido.
@param p rsp                      Resultado de la ejecución del procedimiento.

@throws ErrorPartidoDistinto      Si el parlamentario ya estaba vinculado a un partido distinto al especificado.
@throws ErrorHastaPartidoDH       Si la fecha de final del período es anterior a la fecha de inicio del mismo.
@throws ErrorExisteGasto          Si existe algún gasto imputado al parlamentario posterior a la fecha de final del período.
*/
PROCEDURE partido hasta (
  p_idParlamentario IN  PartidoDH.idParlamentario % TYPE,
  p_idPartido       IN  PartidoDH.idPartido % TYPE,
  p_hasta           IN  PartidoDH.hasta % TYPE,
  p_rsp             OUT VARCHAR2);

END pkg_parlamentario;
```

### 5.5.2.1. Procedimiento pkg_parlamentario.alta.

```
--
-- Procedimiento para dar de alta un parlamentario en la base de datos.
--
PROCEDURE alta (
  p_nombre    IN  Parlamentario.nombre % TYPE,
  p_apellidos IN  Parlamentario.apellidos % TYPE,
  p rsp       OUT VARCHAR2)
AS
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p_rsp := 'OK';
  BEGIN
    -- Inserta un Parlamentario en la base de datos usando los parámetros recibidos.
    -- El atributo idParlamentario se asigna automáticamente mediante una secuencia.
    INSERT INTO Parlamentario
        (nombre,
         apellidos)
      VALUES
        (p nombre,
         p apellidos);
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN OTHERS THEN
      p rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'nombre = [' || p nombre || '] + apellidos = [' || p apellidos || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p_rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_parlamentario.alta', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END alta;
```

### 5.5.2.2. Procedimiento pkg_parlamentario.baja.

```
--
-- Procedimiento para dar de baja un parlamentario en la base de datos.
--
PROCEDURE baja (
  p_idParlamentario IN  Parlamentario.idParlamentario % TYPE,
  p_rsp             OUT VARCHAR2)
AS
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p_rsp := 'OK';
  BEGIN
    -- Controla que el parlamentario a dar de baja exista en la base de datos.
    pkg comun.existe parlamentario (p idParlamentario);
    DELETE FROM
        Parlamentario
      WHERE
        idParlamentario = p_idParlamentario;
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN pkg_comun.ErrorParlamentarioNoExiste THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no existe.';
    WHEN OTHERS THEN
      p rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idParlamentario = [' || p_idParlamentario || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_parlamentario.baja', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END baja;
```

### 5.5.2.3. Procedimiento pkg_parlamentario.modificación.

```
--
-- Procedimiento para modificar un parlamentario en la base de datos.
--
PROCEDURE modificacion (
  p_idParlamentario IN  Parlamentario.idParlamentario % TYPE,
  p_nombre          IN  Parlamentario.nombre % TYPE,
  p apellidos       IN  Parlamentario.apellidos % TYPE,
  p rsp             OUT VARCHAR2)
AS
  existe NUMBER;
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p rsp := 'OK';
  BEGIN
    -- Controla que el parlamentario a modificar exista en la base de datos.
    pkg_comun.existe_parlamentario (p_idParlamentario);
    -- Realiza el update en la tabla Parlamentario.
    UPDATE Parlamentario SET
      nombre = p nombre,
      apellidos = p apellidos
    WHERE
      idParlamentario = p_idParlamentario;
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN pkg comun.ErrorParlamentarioNoExiste THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no existe.';
    WHEN OTHERS THEN
      p_rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idParlamentario = [' || p_idParlamentario || '] + nombre = [' || p_nombre || '] + apellidos = [' || p_apellidos || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_parlamentario.modificacion', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END modificacion;
```

### 5.5.2.4. Procedimiento pkg_parlamentario.parlamento_desde.

```
--
-- Procedimiento para asignar un parlamentario a un parlamento.
--
PROCEDURE parlamento_desde (
  p_idParlamentario IN  ParlamentoDH.idParlamentario % TYPE,
  p_idParlamento    IN  ParlamentoDH.idParlamento % TYPE,
  p desde           IN  ParlamentoDH.desde % TYPE,
  p rsp             OUT VARCHAR2)
AS
  existe   NUMBER;
  maxHasta ParlamentoDH.hasta % TYPE;
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p rsp := 'OK';
  BEGIN
    -- Controla que el parlamentario exista en la base de datos.
    pkg_comun.existe_parlamentario (p_idParlamentario);
    -- Controla que el parlamento exista en la base de datos.
    pkg comun.existe Parlamento (p idParlamento);
    -- Controla que el parlamentario no esté ya en algún parlamento en la actualidad.
    pkg_comun.parlamentario_no_en_parlamento (p_idParlamentario);
    -- Controla que el período (ParlamentoDH) de este parlamentario en este parlamento empiece después de finalizado cualquier período
    -- anterior (si existe alguno) de este parlamentario en cualquier parlamento.
    SELECT
      MAX (hasta)
    INTO
      maxHasta
    FROM
      ParlamentoDH
    WHERE
      idParlamentario = p idParlamentario;

    IF (maxHasta IS NOT NULL
        AND
        maxHasta >= p_desde)
    THEN
      RAISE ErrorDesdeParlamentoDH;
    END IF;

    -- Si todas la comprobaciones anteriores fueron satisfactorias realiza una inserción de un nuevo período en la tabla ParlamentoDH
    -- correspondiente al inicio de un período de este parlamentario en este parlamento desde esta fecha y con
    -- fecha de finalización indefinida.

    INSERT INTO ParlamentoDH
        (idParlamentario,
         idParlamento,
         desde,
         hasta)
      VALUES
        (p_idParlamentario,
```

```
        p_idParlamento,
        p_desde,
        NULL);
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN pkg_comun.ErrorParlamentarioNoExiste THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no existe.';
    WHEN pkg_comun.ErrorParlamentoNoExiste THEN
      p_rsp := 'ERROR: El parlamento [' || p idParlamento || '] no existe.';
    WHEN pkg comun.ErrorParlamentarioParlamento THEN
      p rsp := 'ERROR: El parlamentario [' || p idParlamentario || '] ya está en un parlamento actualmente.';
    WHEN ErrorDesdeParlamentoDH THEN
      p_rsp := 'ERROR: El inicio del período del parlamentario [' || p_idParlamentario || '] en el parlamento [' || p_idParlamento || '] debe ser posterior a
[' || maxHasta || '].';
    WHEN OTHERS THEN
      p_rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idParlamentario = [' || p idParlamentario || '] + idParlamento = [' || p idParlamento || '] + desde = [' || p desde || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p_rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_parlamentario.parlamento_desde', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END parlamento_desde;
```

### 5.5.2.5. Procedimiento pkg_parlamentario.parlamento_hasta.

```
--
-- Procedimiento para desvincular a un parlamentario de un parlamento.
--
PROCEDURE parlamento_hasta (
  p_idParlamentario IN  ParlamentoDH.idParlamentario % TYPE,
  p_idParlamento    IN  ParlamentoDH.idParlamento % TYPE,
  p hasta           IN  ParlamentoDH.hasta % TYPE,
  p rsp             OUT VARCHAR2)
AS
  existe            NUMBER;
  r_idParlamentoDH  ParlamentoDH.idParlamentoDH % TYPE;
  r_idParlamentario ParlamentoDH.idParlamentario % TYPE;
  r idParlamento    ParlamentoDH.idParlamento % TYPE;
  r desde           ParlamentoDH.desde % TYPE;
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p_rsp := 'OK';
  BEGIN
    -- Controla que el parlamentario exista en la base de datos.
    pkg comun.existe parlamentario (p idParlamentario);
    -- Controla que el parlamento exista en la base de datos.
    pkg_comun.existe_Parlamento (p_idParlamento);
    -- Controla que el parlamentario esté en algún parlamento en la actualidad.
    pkg comun.parlamentario en parlamento (p idParlamentario);
    -- Consulta el período (ParlamentoDH) en curso de este parlamentario.
    SELECT
      idParlamentoDH,
      idParlamentario,
      idParlamento,
      desde
    INTO
      r_idParlamentoDH,
      r_idParlamentario,
      r_idParlamento,
      r_desde
    FROM
      ParlamentoDH
    WHERE
      idParlamentario = p_idParlamentario
      AND
      hasta IS NULL;

    -- Controla si el parlamentario está actualmente en el parlamento del que queremos cerrar el período.
    IF (r_idParlamento <> p_idParlamento)
    THEN
      RAISE ErrorParlamentoDistinto;
    END IF;

    -- Controla que la fecha de finalización del período sea posterior a la fecha de inicio del período.
    IF (r_desde > p_hasta)
```

```
    THEN
      RAISE ErrorHastaParlamentoDH;
    END IF;

    -- Controla que no exista ningún gasto imputado a este parlamentario posterior a la fecha en que se quiere
    -- cerrar el período.
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      Gasto
    WHERE
      idParlamentario = p_idParlamentario
      AND
      idParlamento = p idParlamento
      AND
      fecha > p_hasta;

    IF (existe <> 0)
    THEN
      RAISE ErrorExisteGasto;
    END IF;

    -- Si todas la comprobaciones anteriores fueron satisfactorias actualiza el período en la tabla ParlamentoDH
    -- para indicar la finalización en esta fecha del período en curso de este parlamentario en este parlamento.

    UPDATE ParlamentoDH SET
      hasta = p_hasta
    WHERE
      idParlamentoDH = r idParlamentoDH;
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN pkg_comun.ErrorParlamentarioNoExiste THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no existe.';
    WHEN pkg comun.ErrorParlamentoNoExiste THEN
      p rsp := 'ERROR: El parlamento [' || p idParlamento || '] no existe.';
    WHEN pkg comun.ErrorParlamentarioNoParlamento THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no está en ningún parlamento actualmente.';
    WHEN ErrorParlamentoDistinto THEN
      p rsp := 'ERROR: El parlamentario [' || p idParlamentario || '] actualmente está en el parlamento [' || r idParlamento || '].';
    WHEN ErrorHastaParlamentoDH THEN
      p rsp := 'ERROR: El final del período del parlamentario [' || p idParlamentario || '] en el parlamento [' || p idParlamento || '] no puede ser anterior
a [' || r_desde || '].';
    WHEN ErrorExisteGasto THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] del parlamento [' || p_idParlamento || '] tiene gastos imputados posteriores a [' ||
p hasta || '].';
    WHEN OTHERS THEN
      p rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
```

```
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idParlamentario = [' || p_idParlamentario || '] + idParlamento = [' || p_idParlamento || '] + hasta = [' || p_hasta || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg comun.entrada registro ('pkg parlamentario.parlamento hasta', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END parlamento hasta;
```

### 5.5.2.6. Procedimiento pkg_parlamentario.partido_desde.

```
--
-- Procedimiento para asignar un parlamentario a un partido.
--
PROCEDURE partido_desde (
  p_idParlamentario IN  PartidoDH.idParlamentario % TYPE,
  p_idPartido       IN  PartidoDH.idPartido % TYPE,
  p desde           IN  PartidoDH.desde % TYPE,
  p rsp             OUT VARCHAR2)
AS
  existe   NUMBER;
  maxHasta PartidoDH.hasta % TYPE;
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p rsp := 'OK';
  BEGIN
    -- Controla que el parlamentario exista en la base de datos.
    pkg_comun.existe_parlamentario (p_idParlamentario);
    -- Controla que el partido exista en la base de datos.
    pkg comun.existe partido (p idPartido);
    -- Controla que el parlamentario no esté ya en algún partido en la actualidad.
    pkg_comun.parlamentario_no_en_partido (p_idParlamentario);
    -- Controla que el período (PartidoDH) de este parlamentario en este partido empiece después de finalizado cualquier período
    -- anterior (si existe alguno) de este parlamentario en cualquier partido.
    SELECT
      MAX (hasta)
    INTO
      maxHasta
    FROM
      PartidoDH
    WHERE
      idParlamentario = p idParlamentario;

    IF (maxHasta IS NOT NULL
        AND
        maxHasta >= p_desde)
    THEN
      RAISE ErrorDesdePartidoDH;
    END IF;

    -- Si todas la comprobaciones anteriores fueron satisfactorias realiza una inserción de un período en la tabla PartidoDH
    -- correspondiente al inicio de un período de este parlamentario en este partido desde esta fecha y con
    -- fecha de finalización indefinida.

    INSERT INTO PartidoDH
        (idParlamentario,
         idPartido,
         desde,
         hasta)
      VALUES
        (p_idParlamentario,
```

```
        p_idPartido,
        p_desde,
        NULL);
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN pkg_comun.ErrorParlamentarioNoExiste THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no existe.';
    WHEN pkg_comun.ErrorPartidoNoExiste THEN
      p_rsp := 'ERROR: El partido [' || p idPartido || '] no existe.';
    WHEN pkg comun.ErrorParlamentarioPartido THEN
      p rsp := 'ERROR: El parlamentario [' || p idParlamentario || '] ya está en un partido actualmente.';
    WHEN ErrorDesdePartidoDH THEN
      p_rsp := 'ERROR: El inicio del período del parlamentario [' || p_idParlamentario || '] en el partido [' || p_idPartido || '] debe ser posterior a [' ||
maxHasta || '].';
    WHEN OTHERS THEN
      p_rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idParlamentario = [' || p idParlamentario || '] + idPartido = [' || p idPartido || '] + desde = [' || p desde || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p_rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_parlamentario.partido_desde', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END partido_desde;
```

### 5.5.2.7. Procedimiento pkg_parlamentario.partido_hasta.

```
--
-- Procedimiento para desvincular a un parlamentario de un partido.
--
PROCEDURE partido_hasta (
  p_idParlamentario IN  PartidoDH.idParlamentario % TYPE,
  p_idPartido       IN  PartidoDH.idPartido % TYPE,
  p hasta           IN  PartidoDH.hasta % TYPE,
  p rsp             OUT VARCHAR2)
AS
  existe            NUMBER;
  r_idPartidoDH     PartidoDH.idPartidoDH % TYPE;
  r_idParlamentario PartidoDH.idParlamentario % TYPE;
  r idPartido       PartidoDH.idPartido % TYPE;
  r desde           PartidoDH.desde % TYPE;
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p_rsp := 'OK';
  BEGIN
    -- Controla que el parlamentario exista en la base de datos.
    pkg comun.existe parlamentario (p idParlamentario);
    -- Controla que el partido exista en la base de datos.
    pkg_comun.existe_partido (p_idPartido);
    -- Controla que el parlamentario esté en algún partido en la actualidad.
    pkg comun.parlamentario en partido (p idParlamentario);
    -- Consulta el período (PartidoDH) en curso de este parlamentario.
    SELECT
      idPartidoDH,
      idParlamentario,
      idPartido,
      desde
    INTO
      r_idPartidoDH,
      r_idParlamentario,
      r_idPartido,
      r_desde
    FROM
      PartidoDH
    WHERE
      idParlamentario = p_idParlamentario
      AND
      hasta IS NULL;

    -- Controla si el parlamentario está actualmente en el partido del que queremos cerrar el período.
    IF (r_idPartido <> p_idPartido)
    THEN
      RAISE ErrorPartidoDistinto;
    END IF;

    -- Controla que la fecha de finalización del período sea posterior a la fecha de inicio del período.
    IF (r_desde > p_hasta)
```

```
    THEN
      RAISE ErrorHastaPartidoDH;
    END IF;

    -- Controla que no exista ningún gasto imputado a este parlamentario y partido posterior a la fecha en
    -- que se quiere cerrar el período.
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      Gasto
    WHERE
      idParlamentario = p_idParlamentario
      AND
      idPartido = p idPartido
      AND
      fecha > p_hasta;

    IF (existe <> 0)
    THEN
      RAISE ErrorExisteGasto;
    END IF;

    -- Si todas la comprobaciones anteriores fueron satisfactorias actualiza el período en la tabla PartidoDH
    -- para indicar la finalización en esta fecha del período de este parlamentario en este partido.

    UPDATE PartidoDH SET
      hasta = p_hasta
    WHERE
      idPartidoDH = r idPartidoDH;
    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN pkg_comun.ErrorParlamentarioNoExiste THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no existe.';
    WHEN pkg comun.ErrorPartidoNoExiste THEN
      p rsp := 'ERROR: El partido [' || p idPartido || '] no existe.';
    WHEN pkg comun.ErrorParlamentarioNoPartido THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no está en ningún partido actualmente.';
    WHEN ErrorPartidoDistinto THEN
      p rsp := 'ERROR: El parlamentario [' || p idParlamentario || '] actualmente está en el partido [' || r idPartido || '].';
    WHEN ErrorHastaPartidoDH THEN
      p rsp := 'ERROR: El final del período del parlamentario [' || p idParlamentario || '] en el partido [' || p idPartido || '] no puede ser anterior a ['
|| r_desde || '].';
    WHEN ErrorExisteGasto THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] del partido [' || p_idPartido || '] tiene gastos imputados posteriores a [' || p_hasta
|| '].';
    WHEN OTHERS THEN
      p rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
```

```
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idParlamentario = [' || p_idParlamentario || '] + idPartido = [' || p_idPartido || '] + hasta = [' || p_hasta || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg comun.entrada registro ('pkg parlamentario.partido hasta', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END partido hasta;
```

### 5.5.3. Paquete pkg_gasto.

El *script* 3_3_CREAR_PKG_GASTO.SQL crea en la base de datos el paquete pkg_gasto de procedimientos de altas, bajas y modificaciones en la tabla Gasto.

```
-- PKG_GASTO specification
CREATE OR REPLACE PACKAGE TFC.pkg gasto AS

ErrorFecha EXCEPTION;
ErrorEjercicioCerrado EXCEPTION;
ErrorParlamentario EXCEPTION;
ErrorNoParlamentario EXCEPTION;
ErrorBaseImponible EXCEPTION;
ErrorBaseImponibleNegativa EXCEPTION;
ErrorProveedor EXCEPTION;
ErrorFormaPago EXCEPTION;
ErrorNoFormaPago EXCEPTION;
ErrorPagado EXCEPTION;

/**
Procedimiento para dar de alta un gasto en la base de datos.

Al insertar una fila en la base de datos, el atributo idGasto se asigna automáticamente mediante una secuencia.

@param p fecha                        Fecha en la que se ha producido el gasto.
@param p_idParlamento                 Identificador del parlamento al que se imputa el gasto.
@param p_idParlamentario              Identificador del parlamentario al que se imputa el gasto (si lo hay).
@param p_baseImponible                Base imponible del importe del gasto.
@param p_idTipoIVA                    Identificador del tipo de IVA aplicable al gasto.
@param p idCategoriaGasto             Identificador de la categoría del gasto.
@param p idEstado                     Identificador del estado del gasto.
@param p_idFormaPago                  Identificador de la forma de pago del gasto (si ha sido pagado).
@param p_idProveedor                  Identificador del proveedor del bien o servicio (si lo hay).
@param p descripcion                  Texto descriptivo del gasto.
@param p justificacion                Texto justificativo del gasto.
@param p presupuestoURL               URL al documento que contiene el presupuesto del gasto (si lo hay).
@param p_facturaURL                   URL al documento que contiene la factura del gasto (si la hay).
@param p_contratoURL                  URL al documento que contiene el contrato del gasto (si lo hay).
@param p rsp                          Resultado de la ejecución del procedimiento.

@throws ErrorFecha                    Si se especifica una fecha NULL para el gasto.
@throws ErrorEjercicioCerrado         Si se especifica una fecha perteneciente a un ejercicio cerrado.
@throws ErrorParlamentario            Si se especifica un idParlamentario NULL cuando no está permitido.
@throws ErrorNoParlamentario          Si se especifica un idParlamentario cuando no está permitido.
@throws ErrorBaseImponible            Si se especifica una base imponible NULL.
@throws ErrorBaseImponibleNegativa    Si se especifica una base imponible negativa para el gasto.
@throws ErrorProveedor                Si se especifica un idProveedor NULL cuando no está permitido.
@throws ErrorFormaPago                Si se especifica un idFormaPago NULL cuando no está permitido.
@throws ErrorNoFormaPago              Si se especifica un idFormapago cuando no está permitido.
*/
PROCEDURE alta (
  p fecha            IN  Gasto.fecha % TYPE,
  p_idParlamento     IN  Gasto.idParlamento % TYPE,
```

```
  p_idParlamentario  IN  Gasto.idParlamentario % TYPE,
  p_baseImponible    IN  Gasto.baseImponible % TYPE,
  p_idTipoIVA        IN  Gasto.idTipoIVA % TYPE,
  p_idCategoriaGasto IN  Gasto.idCategoriaGasto % TYPE,
  p_idEstado         IN  Gasto.idEstado % TYPE,
  p_idFormaPago      IN  Gasto.idFormaPago % TYPE,
  p_idProveedor      IN  Gasto.idProveedor % TYPE,
  p_descripcion      IN  Gasto.descripcion % TYPE,
  p_justificacion    IN  Gasto.justificacion % TYPE,
  p_presupuestoURL   IN  Gasto.presupuestoURL % TYPE,
  p_facturaURL       IN  Gasto.facturaURL % TYPE,
  p_contratoURL      IN  Gasto.contratoURL % TYPE,
  p_rsp              OUT VARCHAR2);

/**
Procedimiento para dar de baja un gasto en la base de datos.

@param p_idGasto                      Identificador del gasto a dar de baja.
@param p_rsp                          Resultado de la ejecución del procedimiento.

@throws ErrorPagado                   Si el gasto que se intenta dar de baja está pagado.
@throws ErrorEjercicioCerrado         Si el gasto que se intenta dar de baja pertenece a un ejercicio cerrado.
*/
PROCEDURE baja (
  p_idGasto IN  Gasto.idGasto % TYPE,
  p_rsp     OUT VARCHAR2);

/**
Procedimiento para modificar un gasto en la base de datos.

@param p idGasto                      Identificador del gasto a modificar.
@param p fecha                        Fecha en la que se ha producido el gasto.
@param p idParlamento                 Identificador del parlamento al que se imputa el gasto.
@param p idParlamentario              Identificador del parlamentario al que se imputa el gasto (si lo hay).
@param p_baseImponible                Base imponible del importe del gasto.
@param p_idTipoIVA                    Identificador del tipo de IVA aplicable al gasto.
@param p idCategoriaGasto             Identificador de la categoría del gasto.
@param p idEstado                     Identificador del estado del gasto.
@param p idFormaPago                  Identificador de la forma de pago del gasto (si ha sido pagado).
@param p_idProveedor                  Identificador del proveedor del bien o servicio (si lo hay).
@param p_descripcion                  Texto descriptivo del gasto.
@param p justificacion                Texto justificativo del gasto.
@param p presupuestoURL               URL al documento que contiene el presupuesto del gasto (si lo hay).
@param p facturaURL                   URL al documento que contiene la factura del gasto (si la hay).
@param p_contratoURL                  URL al documento que contiene el contrato del gasto (si lo hay).
@param p_rsp                          Resultado de la ejecución del procedimiento.

@throws ErrorPagado                   Si el gasto que se intenta modificar está pagado.
@throws ErrorEjercicioCerrado         Si se especifica una fecha perteneciente a un ejercicio cerrado.
@throws ErrorFecha                    Si se especifica una fecha NULL para el gasto.
@throws ErrorParlamentario            Si se especifica un idParlamentario NULL cuando no está permitido.
@throws ErrorNoParlamentario          Si se especifica un idParlamentario cuando no está permitido.
@throws ErrorBaseImponible            Si se especifica una base imponible NULL.
```

```
@throws ErrorBaseImponibleNegativa    Si se especifica una base imponible negativa para el gasto.
@throws ErrorProveedor                Si se especifica un idProveedor NULL cuando no está permitido.
@throws ErrorFormaPago                Si se especifica un idFormaPago NULL cuando no está permitido.
@throws ErrorNoFormaPago              Si se especifica un idFormapago cuando no está permitido.
*/
PROCEDURE modificacion (
  p_idGasto          IN  Gasto.idGasto % TYPE,
  p_fecha            IN  Gasto.fecha % TYPE,
  p_idParlamento     IN  Gasto.idParlamento % TYPE,
  p idParlamentario  IN  Gasto.idParlamentario % TYPE,
  p baseImponible    IN  Gasto.baseImponible % TYPE,
  p idTipoIVA        IN  Gasto.idTipoIVA % TYPE,
  p_idCategoriaGasto IN  Gasto.idCategoriaGasto % TYPE,
  p_idEstado         IN  Gasto.idEstado % TYPE,
  p_idFormaPago      IN  Gasto.idFormaPago % TYPE,
  p idProveedor      IN  Gasto.idProveedor % TYPE,
  p_descripcion      IN  Gasto.descripcion % TYPE,
  p_justificacion    IN  Gasto.justificacion % TYPE,
  p_presupuestoURL   IN  Gasto.presupuestoURL % TYPE,
  p_facturaURL       IN  Gasto.facturaURL % TYPE,
  p contratoURL      IN  Gasto.contratoURL % TYPE,
  p rsp              OUT VARCHAR2);

/**
Procedimiento para cerrar un ejercicio.

Una vez cerrado un ejercicio ya no se pueden dar de alta nuevos gastos en ese ejercicio, ni dar de baja
ni modificar los gastos imputados en ese ejercicio.

@param p_idEjercicio                  Identificador del ejercicio a cerrar.
@param p rsp                          Resultado de la ejecución del procedimiento.
*/
PROCEDURE cerrar ejercicio (
  p idEjercicio IN  Gasto.idEjercicio % TYPE,
  p_rsp         OUT VARCHAR2);

END pkg gasto;
```

### 5.5.3.1. Procedimiento pkg_gasto.alta.

```
--
-- Procedimiento para dar de alta un gasto en la base de datos.
--
PROCEDURE alta (
  p_fecha             IN  Gasto.fecha % TYPE,
  p_idParlamento      IN  Gasto.idParlamento % TYPE,
  p idParlamentario   IN  Gasto.idParlamentario % TYPE,
  p baseImponible     IN  Gasto.baseImponible % TYPE,
  p idTipoIVA         IN  Gasto.idTipoIVA % TYPE,
  p_idCategoriaGasto IN  Gasto.idCategoriaGasto % TYPE,
  p_idEstado          IN  Gasto.idEstado % TYPE,
  p_idFormaPago       IN  Gasto.idFormaPago % TYPE,
  p idProveedor       IN  Gasto.idProveedor % TYPE,
  p descripcion       IN  Gasto.descripcion % TYPE,
  p justificacion     IN  Gasto.justificacion % TYPE,
  p_presupuestoURL    IN  Gasto.presupuestoURL % TYPE,
  p_facturaURL        IN  Gasto.facturaURL % TYPE,
  p contratoURL       IN  Gasto.contratoURL % TYPE,
  p rsp               OUT VARCHAR2)
AS
  x_idEjercicio     Ejercicio.idEjercicio % TYPE;
  x_inicio          Ejercicio.inicio % TYPE;
  x_fin             Ejercicio.fin % TYPE;
  x cerradoSN       Ejercicio.cerradoSN % TYPE;
  x ParlamentarioSN TipoGasto.ParlamentarioSN % TYPE;
  x idPartido       Partido.idPartido % TYPE;
  x_formaPagoSN     Estado.FormaPagoSN % TYPE;
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p rsp := 'OK';
  BEGIN
    -- Inicializa los campos que calcula este procedimiento.
    x_idEjercicio := NULL;
    x_idPartido := NULL;

    -- Controla la fecha del Gasto.
    IF (p fecha IS NULL)
    THEN
      -- Debe especificarse una fecha válida para el Gasto.
      RAISE ErrorFecha;
    END IF;

    -- Controla que la fecha esté entre el rango de fechas de algún Ejercicio dado de alta en la base de datos.
    pkg_comun.existe_ejercicio (p_fecha);
    -- Consulta el Ejercicio al que pertenece la fecha del Gasto.
    SELECT
      idEjercicio,
      inicio,
      fin,
      cerradoSN
```

```
    INTO
      x_idEjercicio,
      x_inicio,
      x fin,
      x cerradoSN
    FROM
      Ejercicio
    WHERE
      inicio <= p_fecha
      AND
      p fecha <= fin;

    -- Controla que el Ejercicio al que pertenece la fecha del Gasto no esté cerrado.
    IF (x_cerradoSN = 'T')
    THEN
      RAISE ErrorEjercicioCerrado;
    END IF;

    -- Controla que el Parlamento al que se imputará el Gasto exista en la base de datos.
    pkg_comun.existe_parlamento (p_idParlamento);

    -- Controla que la CategoriaGasto exista en la base de datos.
    pkg_comun.existe_categoriaGasto (p_idCategoriaGasto);

    -- Consulta la CategoriaGasto y su TipoGasto.
    SELECT
      ParlamentarioSN
    INTO
      x_ParlamentarioSN
    FROM
      CategoriaGasto,
      TipoGasto
    WHERE
      CategoriaGasto.idCategoriaGasto = p idCategoriaGasto
      AND
      TipoGasto.idTipoGasto = CategoriaGasto.idTipoGasto;

    -- Controla si la CategoriaGasto y su TipoGasto requieren que el Gasto sea asignado a un Parlamentario.
    IF (x ParlamentarioSN = 'T')
    THEN
      -- La CategoriaGasto requiere que el Gasto sea asignado a un Parlamentario.
      IF (p idParlamentario IS NULL)
      THEN
        RAISE ErrorParlamentario;
      ELSE -- Comprobaciones asociadas al Parlamentario.
        -- Controla que el Parlamentario exista en la base de datos.
        pkg_comun.existe_parlamentario (p_idParlamentario);
        -- Controla que el Parlamentario estuviera en este Parlamento en la fecha en que se produjo el Gasto.
        pkg comun.parlamentario en parlamento (p idParlamento, p idParlamentario, p fecha);
        -- Comprueba que el Parlamentario estuviera en algún Partido en la fecha en que se produjo el Gasto.
        -- Se considera que para poder imputar un Gasto a un Parlamentario en un Parlamento este debe estar en un Partido.
        -- Si no milita en ningún Partido entonces ha de pertenecer a un Partido ficticio llamado GRUPO MIXTO.
        pkg_comun.parlamentario_en_partido (p_idParlamentario, p_fecha);
```

```
      -- Consulta ese período (PartidoDH) para obtener el identificador del Partido.
      SELECT
        idPartido
      INTO
        x idPartido
      FROM
        PartidoDH
      WHERE
        idParlamentario = p_idParlamentario
        AND
        desde <= p fecha
        AND
        (p_fecha <= hasta
          OR
          hasta IS NULL);
  END IF; -- Comprobaciones asociadas al Parlamentario.
ELSE
  -- La CategoriaGasto no permite que el Gasto sea asignado a un Parlamentario.
  IF (p_idParlamentario IS NOT NULL)
  THEN
    RAISE ErrorNoParlamentario;
  END IF;
END IF;

-- Controla la baseImponible del Gasto.
IF (p_baseImponible IS NULL)
THEN
  RAISE ErrorBaseImponible;
ELSE
  IF (p_baseImponible < 0.00)
  THEN
    RAISE ErrorBaseImponibleNegativa;
  END IF;
END IF;

-- Controla que el TipoIVA exista en la base de datos.
pkg comun.existe tipoIVA (p idTipoIVA);

-- Controla el Proveedor.
IF (p_idProveedor IS NULL)
THEN
  IF (p baseImponible >= RequiereProveedor)
  THEN
    -- Se considera que el Proveedor es OBLIGATORIO si la base imponible del gasto es mayor o igual a RequiereProveedor.
    RAISE ErrorProveedor;
  END IF;
ELSE
  -- Se considera que el Proveedor es OPCIONAL si la base imponible del gasto es menor que RequiereProveedor.
  -- Si se especifica Proveedor controla su existencia en la base de datos.
  pkg comun.existe PROVEEDOR (p idProveedor);
END IF;

-- Controla que el Estado del Gasto exista en la base de datos.
```

```
    pkg_comun.existe_ESTADO (p_idEstado);
    -- Consulta el Estado del Gasto.
    SELECT
      FormaPagoSN
    INTO
      x formaPagoSN
    FROM
      Estado
    WHERE
      idEstado = p idEstado;

    -- Controla si el Estado del Gasto requiere que se especifique una FormaPago.
    IF (x_formaPagoSN = 'T')
    THEN
      -- El Estado del Gasto requiere una FormaPago.
      IF (p idFormaPago IS NULL)
      THEN
        RAISE ErrorFormaPago;
      ELSE
        -- Controla que la FormaPago exista en la base de datos.
        pkg comun.existe FORMAPAGO (p idFormaPago);
      END IF;
    ELSE
      -- El Estado del Gasto no permite una FormaPago.
      IF (p_idFormaPago IS NOT NULL)
      THEN
        RAISE ErrorNoFormaPago;
      END IF;
    END IF;

    -- No se realizan comprobaciones sobre los parámetros: p descripcion, p justificacion, p presupuestoURL, p facturaURL, p contratoURL.

    -- Actualización Estadísticas.
    IF (x ParlamentarioSN = 'T')
    THEN
      -- Si el Gasto es imputable a un Parlamentario entonces se suma la base imponible del gasto
      -- en la tabla de estadística ParlamentarioST que a su vez se suma en la tabla de estadística ParlamentoST.
      estadistica ParlamentarioST (x idEjercicio, p idParlamento, p idParlamentario, p baseImponible);
    ELSE
      -- Si el Gasto no es imputable a un Parlamentario entonces se suma la base imponible del gasto
      -- en la tabla de estadística ParlamentoST.
      estadistica ParlamentoST (x idEjercicio, p idParlamento, p baseImponible);
    END IF;

    -- Realiza una inserción en la tabla Gasto.
    INSERT INTO Gasto
        (fecha,
         idEjercicio,
         idParlamento,
         idParlamentario,
         idPartido,
         descripcion,
         baseImponible,
```

```
        idTipoIVA,
        idCategoriaGasto,
        idEstado,
        idFormaPago,
        idProveedor,
        justificacion,
        presupuestoURL,
        facturaURL,
        contratoURL)
    VALUES
      (p_fecha,
       x_idEjercicio,
       p_idParlamento,
       p_idParlamentario,
       x_idPartido,
       p_descripcion,
       p_baseImponible,
       p_idTipoIVA,
       p_idCategoriaGasto,
       p_idEstado,
       p_idFormaPago,
       p_idProveedor,
       p_justificacion,
       p_presupuestoURL,
       p_facturaURL,
       p_contratoURL);

  -- Si no ha habido excepciones confirma los cambios en la base de datos.
  COMMIT;
EXCEPTION
  WHEN ErrorFecha THEN
    p_rsp := 'ERROR: Debe especificarse una fecha válida para el gasto.';
  WHEN ErrorEjercicioCerrado THEN
    p_rsp := 'ERROR: El ejercicio [' || x_idEjercicio || '] desde [' || x_inicio || '] hasta [' || x_fin || '] está cerrado.';
  WHEN ErrorParlamentario THEN
    p_rsp := 'ERROR: La categoría de gasto [' || p_idCategoriaGasto || '] requiere que el gasto sea asignado a un parlamentario.';
  WHEN ErrorNoParlamentario THEN
    p_rsp := 'ERROR: La categoría de gasto [' || p_idCategoriaGasto || '] no permite que el gasto sea asignado a un parlamentario.';
  WHEN ErrorBaseImponible THEN
    p_rsp := 'ERROR: Debe especificarse la base imponible del gasto.';
  WHEN ErrorBaseImponibleNegativa THEN
    p_rsp := 'ERROR: La base imponible del gasto no puede ser negativa.';
  WHEN ErrorProveedor THEN
    p_rsp := 'ERROR: Debe especificarse un proveedor obligatoriamente por ser un gasto superior a [' || RequiereProveedor || '] euros.';
  WHEN ErrorFormaPago THEN
    p_rsp := 'ERROR: El estado [' || p_idEstado || '] requiere que se especifique una forma de pago en el gasto.';
  WHEN ErrorNoFormaPago THEN
    p_rsp := 'ERROR: El estado [' || p_idEstado || '] no permite que se especifique una forma de pago en el gasto.';
  WHEN pkg_comun.ErrorEjercicioNoExiste THEN
    p_rsp := 'ERROR: La fecha [' || p_fecha || '] no pertenece a ningún ejercicio.';
  WHEN pkg_comun.ErrorParlamentoNoExiste THEN
    p_rsp := 'ERROR: El parlamento [' || p_idParlamento || '] no existe.';
  WHEN pkg_comun.ErrorCategoriaGastoNoExiste THEN
```

```
      p_rsp := 'ERROR: La categoría de gasto [' || p_idCategoriaGasto || '] no existe.';
    WHEN pkg_comun.ErrorParlamentarioNoExiste THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no existe.';
    WHEN pkg comun.ErrorParlamentarioNoParlamento THEN
      p rsp := 'ERROR: El parlamentario [' || p idParlamentario || '] no estaba en el parlamento [' || p idParlamento || '] en fecha [' || p fecha || '].';
    WHEN pkg comun.ErrorParlamentarioNoPartido THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no estaba en ningún partido en fecha [' || p_fecha || '].';
    WHEN pkg_comun.ErrorProveedorNoExiste THEN
      p_rsp := 'ERROR: El proveedor [' || p_idProveedor || '] no existe.';
    WHEN pkg comun.ErrorTipoIVANoExiste THEN
      p rsp := 'ERROR: El tipo de IVA [' || p idTipoIVA || '] no existe.';
    WHEN pkg comun.ErrorEstadoNoExiste THEN
      p_rsp := 'ERROR: El estado [' || p_idEstado || '] no existe.';
    WHEN pkg_comun.ErrorFormaPagoNoExiste THEN
      p_rsp := 'ERROR: La forma de pago [' || p_idFormaPago || '] no existe.';
    WHEN OTHERS THEN
      p_rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'fecha = [' || p fecha || '] + idParlamento = [' || p idParlamento || '] + idParlamentario = [' || p idParlamentario || '] + baseImponible = ['
|| p_baseImponible || '] + idTipoIVA = [' || p_idTipoIVA || '] + idCategoriaGasto = [' || p_idCategoriaGasto || '] + idEstado = [' || p_idEstado || '] +
idFormaPago = [' || p_idFormaPago || '] + idProveedor = [' || p_idProveedor || '] + descripcion = [' || p_descripcion || '] + justificacion = [' ||
p_justificacion || '] + presupuestoURL = [' || p_presupuestoURL || '] + facturaURL = [' || p_facturaURL || '] + contratoURL = [' || p_contratoURL || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_gasto.alta', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END alta;
```

### 5.5.3.2. Procedimiento pkg_gasto.baja.

```
--
-- Procedimiento para dar de baja un gasto en la base de datos.
--
PROCEDURE baja (
  p_idGasto IN  Gasto.idGasto % TYPE,
  p_rsp     OUT VARCHAR2)
AS
  r idEjercicio     Gasto.idEjercicio % TYPE;
  r idParlamento    Gasto.idParlamento % TYPE;
  r_idParlamentario Gasto.idParlamentario % TYPE;
  r_idFormaPago     Gasto.idFormaPago % TYPE;
  r_baseImponible   Gasto.baseImponible % TYPE;
  r idTipoIVA       Gasto.idTipoIVA % TYPE;

  x inicio          Ejercicio.inicio % TYPE;
  x_fin             Ejercicio.fin % TYPE;
  x_cerradoSN       Ejercicio.cerradoSN % TYPE;
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p rsp := 'OK';
  BEGIN
    -- Controla que el Gasto a dar de baja exista en la base de datos.
    pkg_comun.existe_gasto (p_idGasto);
    -- Consulta el Gasto de la base de datos.
    SELECT
      idEjercicio,
      idParlamento,
      idParlamentario,
      idFormaPago,
      baseImponible,
      idTipoIVA
    INTO
      r_idEjercicio,
      r_idParlamento,
      r_idParlamentario,
      r idFormaPago,
      r baseImponible,
      r_idTipoIVA
    FROM
      Gasto
    WHERE
      idGasto = p idGasto;

    -- Controla que el Gasto no esté pagado.
    IF (r_idFormaPago IS NOT NULL)
    THEN
      RAISE ErrorPagado;
    END IF;

    -- Consulta el Ejercicio al que pertenece el Gasto.
```

```
    SELECT
      inicio,
      fin,
      cerradoSN
    INTO
      x inicio,
      x_fin,
      x_cerradoSN
    FROM
      Ejercicio
    WHERE
      idEjercicio = r idEjercicio;
    -- Controla que el Ejercicio no esté cerrado.
    IF (x_cerradoSN = 'T')
    THEN
      RAISE ErrorEjercicioCerrado;
    END IF;

    -- Actualiza Estadística.
    IF (r_idParlamentario IS NOT NULL)
    THEN
      -- Si el Gasto es imputable a un Parlamentario entonces se resta la base imponible del gasto
      -- en la tabla de estadística ParlamentarioST que a su vez se resta en la tabla de estadística ParlamentoST.
      estadistica_ParlamentarioST (r_idEjercicio, r_idParlamento, r_idParlamentario, -r_baseImponible);
    ELSE
      -- Si el Gasto no es imputable a un Parlamentario entonces se resta la base imponible del gasto
      -- en la tabla de estadística ParlamentoST.
      estadistica ParlamentoST (r idEjercicio, r idParlamento, -r baseImponible);
    END IF;

    -- Realiza un borrado en la tabla Gasto.
    DELETE FROM
        Gasto
      WHERE
        idGasto = p_idGasto;

    -- Si no ha habido excepciones confirma los cambios en la base de datos.
    COMMIT;
  EXCEPTION
    WHEN pkg_comun.ErrorGastoNoExiste THEN
      p_rsp := 'ERROR: El gasto [' || p_idGasto || '] no existe.';
    WHEN ErrorPagado THEN
      p rsp := 'ERROR: El gasto [' || p idGasto || '] no puede eliminarse porque está pagado.';
    WHEN ErrorEjercicioCerrado THEN
      p_rsp := 'ERROR: El ejercicio [' || r_idEjercicio || '] desde [' || x_inicio || '] hasta [' || x_fin || '] está cerrado.';
    WHEN OTHERS THEN
      p_rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idGasto = [' || p_idGasto || ']';
  -- Construye el string con los parámetros de salida y sus valores.
```

```
  salida := 'rsp = ' || p_rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_gasto.baja', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END baja;
```

### 5.5.3.3. Procedimiento pkg_gasto.modificacion.

```
--
-- Procedimiento para modificar un gasto en la base de datos.
--
PROCEDURE modificacion (
  p_idGasto           IN  Gasto.idGasto % TYPE,
  p_fecha             IN  Gasto.fecha % TYPE,
  p idParlamento      IN  Gasto.idParlamento % TYPE,
  p idParlamentario   IN  Gasto.idParlamentario % TYPE,
  p baseImponible     IN  Gasto.baseImponible % TYPE,
  p_idTipoIVA         IN  Gasto.idTipoIVA % TYPE,
  p_idCategoriaGasto IN  Gasto.idCategoriaGasto % TYPE,
  p_idEstado          IN  Gasto.idEstado % TYPE,
  p idFormaPago       IN  Gasto.idFormaPago % TYPE,
  p idProveedor       IN  Gasto.idProveedor % TYPE,
  p descripcion       IN  Gasto.descripcion % TYPE,
  p_justificacion     IN  Gasto.justificacion % TYPE,
  p_presupuestoURL    IN  Gasto.presupuestoURL % TYPE,
  p facturaURL        IN  Gasto.facturaURL % TYPE,
  p contratoURL       IN  Gasto.contratoURL % TYPE,
  p rsp               OUT VARCHAR2)
AS
  r_fecha               Gasto.fecha % TYPE;
  r_idEjercicio         Gasto.idEjercicio % TYPE;
  r idParlamento        Gasto.idParlamento % TYPE;
  r idParlamentario     Gasto.idParlamentario % TYPE;
  r idPartido           Gasto.idPartido % TYPE;
  r_baseImponible       Gasto.baseImponible % TYPE;
  r_idTipoIVA           Gasto.idTipoIVA % TYPE;
  r_idCategoriaGasto    Gasto.idCategoriaGasto % TYPE;
  r idEstado            Gasto.idEstado % TYPE;
  r idFormaPago         Gasto.idFormaPago % TYPE;
  r_idProveedor         Gasto.idProveedor % TYPE;
  r_descripcion         Gasto.descripcion % TYPE;
  r_justificacion       Gasto.justificacion % TYPE;
  r_presupuestoURL      Gasto.presupuestoURL % TYPE;
  r facturaURL          Gasto.facturaURL % TYPE;
  r contratoURL         Gasto.contratoURL % TYPE;

  idEjercicioCambia     BOOLEAN := FALSE;
  idParlamentoCambia    BOOLEAN := FALSE;
  idParlamentarioCambia BOOLEAN := FALSE;
  importeGastoCambia    BOOLEAN := FALSE;

  x_idEjercicio         Ejercicio.idEjercicio % TYPE;
  x_inicio              Ejercicio.inicio % TYPE;
  x fin                 Ejercicio.fin % TYPE;
  x cerradoSN           Ejercicio.cerradoSN % TYPE;
  x idTipoGasto         CategoriaGasto.idTipoGasto % TYPE;
  x_ParlamentarioSN     TipoGasto.ParlamentarioSN % TYPE;
  x_idPartido           Partido.idPartido % TYPE;
```

```
  x_formaPagoSN         Estado.FormaPagoSN % TYPE;
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p_rsp := 'OK';
  BEGIN
    -- Controla que el Gasto a dar de baja exista en la base de datos.
    pkg_comun.existe_gasto (p_idGasto);
    -- Consulta el Gasto de la base de datos.
    SELECT
      fecha,
      idEjercicio,
      idParlamento,
      idParlamentario,
      idPartido,
      descripcion,
      baseImponible,
      idTipoIVA,
      idCategoriaGasto,
      idEstado,
      idFormaPago,
      idProveedor,
      justificacion,
      presupuestoURL,
      facturaURL,
      contratoURL
    INTO
      r_fecha,
      r_idEjercicio,
      r_idParlamento,
      r_idParlamentario,
      r_idPartido,
      r_descripcion,
      r_baseImponible,
      r_idTipoIVA,
      r_idCategoriaGasto,
      r_idEstado,
      r_idFormaPago,
      r_idProveedor,
      r_justificacion,
      r_presupuestoURL,
      r_facturaURL,
      r_contratoURL
    FROM
      Gasto
    WHERE
      idGasto = p_idGasto;

    -- Inicializa los campos que calcula este procedimiento.
    x_idEjercicio := r_idEjercicio;
    x_idPartido := NULL;

    -- Se controlan las condiciones sobre los atributos actuales del Gasto (igual que en la baja).
```

```
  -- Controla que el Gasto no esté pagado.
  IF (r_idFormaPago IS NOT NULL)
  THEN
    RAISE ErrorPagado;
  END IF;

  -- Consulta el Ejercicio al que pertenece la fecha del Gasto.
  SELECT
    inicio,
    fin,
    cerradoSN
  INTO
    x_inicio,
    x_fin,
    x_cerradoSN
  FROM
    Ejercicio
  WHERE
    inicio <= r_fecha
    AND
    r fecha <= fin;

  -- Controla que el Ejercicio al que pertenece la fecha del Gasto no esté cerrado.
  IF (x_cerradoSN = 'T')
  THEN
    RAISE ErrorEjercicioCerrado;
  END IF;

  -- Se controlan las condiciones sobre los parámetros nuevos.

  -- Controla la nueva fecha del Gasto.
  IF (p fecha IS NULL)
  THEN
    -- Debe especificarse una fecha válida para el Gasto.
    RAISE ErrorFecha;
  END IF;

  -- Controla si la fecha se modifica.
  IF (p fecha <> r fecha)
  THEN
    r_fecha := p_fecha;
    -- Controla que la nueva fecha esté entre el rango de fechas de algún Ejercicio dado de alta en la base de datos.
    pkg comun.existe ejercicio (p fecha);
    -- Consulta el Ejercicio al que pertenece la nueva fecha del Gasto.
    SELECT
      idEjercicio,
      inicio,
      fin,
      cerradoSN
    INTO
      x_idEjercicio,
      x_inicio,
      x_fin,
```

```
      x_cerradoSN
    FROM
      Ejercicio
    WHERE
      inicio <= p fecha
      AND
      p_fecha <= fin;

    -- Controla si el idEjercicio se modifica.
    IF (r idEjercicio <> x idEjercicio)
    THEN
      -- La fecha cambia y el idEjercicio cambia.
      idEjercicioCambia := TRUE;
      -- Controla que el nuevo Ejercicio al que pertenece la nueva fecha del Gasto no esté cerrado.
      IF (x_cerradoSN = 'T')
      THEN
        RAISE ErrorEjercicioCerrado;
      END IF;
    END IF;
  END IF;

  -- Controla que el nuevo Parlamento al que se imputará el Gasto exista en la base de datos.
  pkg_comun.existe_parlamento (p_idParlamento);

  -- Controla si el idParlamento se modifica.
  IF (p_idParlamento <> r_idParlamento)
  THEN
    idParlamentoCambia := TRUE;
  END IF;

  -- Controla que la nueva CategoriaGasto exista en la base de datos.
  pkg comun.existe categoriaGasto (p idCategoriaGasto);
  -- Controla si idCategoriaGasto se modifica.
  IF (p idCategoriaGasto <> r idCategoriaGasto)
  THEN
    r_idCategoriaGasto := p_idCategoriaGasto;
    -- Consulta la nueva CategoriaGasto y su TipoGasto.
    SELECT
      CategoriaGasto.idTipoGasto,
      TipoGasto.ParlamentarioSN
    INTO
      x idTipoGasto,
      x ParlamentarioSN
    FROM
      CategoriaGasto,
      TipoGasto
    WHERE
      CategoriaGasto.idCategoriaGasto = p idCategoriaGasto
      AND
      TipoGasto.idTipoGasto = CategoriaGasto.idTipoGasto;

    -- Controla si la nueva CategoriaGasto y su TipoGasto requieren que el Gasto sea asignado a un Parlamentario.
    IF (x_ParlamentarioSN = 'T')
```

```
    THEN
      -- La nueva CategoriaGasto requiere que el Gasto sea asignado a un Parlamentario.
      IF (p_idParlamentario IS NULL)
      THEN
        RAISE ErrorParlamentario;
      ELSE
        -- Controla que el Parlamentario exista en la base de datos.
        pkg_comun.existe_parlamentario (p_idParlamentario);
        -- Controla que el nuevo Parlamentario estuviera en este Parlamento en la fecha en que se produjo el Gasto.
        pkg comun.parlamentario en parlamento (p idParlamento, p idParlamentario, p fecha);
        -- Comprueba que el nuevo Parlamentario estuviera en algún Partido en la fecha en que se produjo el Gasto.
        -- Se considera que para poder imputar un Gasto a un Parlamentario en un Parlamento este debe estar en un Partido.
        -- Si no milita en ningún Partido entonces ha de pertenecer a un Partido ficticio llamado GRUPO MIXTO.
        pkg_comun.parlamentario_en_partido (p_idParlamentario, p_fecha);
        -- Consulta ese período (PartidoDH) para obtener el identificador del Partido.
        SELECT
          idPartido
        INTO
          x_idPartido
        FROM
          PartidoDH
        WHERE
          idParlamentario = p_idParlamentario
          AND
          desde <= p_fecha
          AND
          (p fecha <= hasta
            OR
            hasta IS NULL);
      END IF;
    ELSE
      -- La CategoriaGasto no permite que el Gasto sea asignado a un Parlamentario.
      IF (p idParlamentario IS NOT NULL)
      THEN
        RAISE ErrorNoParlamentario;
      END IF;
    END IF;
  END IF;

  -- Controla si idParlamentario se modifica.
  IF (p_idParlamentario IS NULL)
  THEN
    IF (r idParlamentario IS NOT NULL)
    THEN
      idParlamentarioCambia := TRUE;
    END IF;
  ELSE
    IF (r idParlamentario IS NOT NULL
        AND
        p idParlamentario <> r idParlamentario)
    THEN
      idParlamentarioCambia := TRUE;
    END IF;
```

```
    END IF;

    -- Controla si idPartido se modifica.
    IF (x idPartido IS NULL)
    THEN
      IF (r idPartido IS NOT NULL)
      THEN
        r_idPartido := x_idPartido;
      END IF;
    ELSE
      IF (r idPartido IS NOT NULL
          AND
          x_idPartido <> r_idPartido)
      THEN
        r_idPartido := x_idPartido;
      END IF;
    END IF;

    -- Controla la nueva baseImponible del Gasto.
    IF (p_baseImponible IS NULL)
    THEN
      RAISE ErrorBaseImponible;
    ELSE
      IF (p_baseImponible < 0.00)
      THEN
        RAISE ErrorBaseImponibleNegativa;
      END IF;
    END IF;

    -- Controla si baseImponible se modifica.
    IF (p baseImponible <> r baseImponible)
    THEN
      r baseImponible := p baseImponible;
    END IF;

    -- Controla que el nuevo TipoIVA exista en la base de datos.
    pkg comun.existe tipoIVA (p idTipoIVA);

    -- Controla si idTipoIVA se modifica.
    IF (p_idTipoIVA <> r_idTipoIVA)
    THEN
      r idTipoIVA := p idTipoIVA;
    END IF;

    -- Controla si el importe del Gasto se ha modificado.
    IF (r_baseImponible <> p_baseImponible)
    THEN
      importeGastoCambia := TRUE;
    END IF;

    -- Controla el Proveedor.
    IF (p_idProveedor IS NULL)
    THEN
```

```
    IF (p_baseImponible >= RequiereProveedor)
    THEN
      -- Se considera que el Proveedor es OBLIGATORIO si la base imponible del gasto es mayor o igual a RequiereProveedor.
      RAISE ErrorProveedor;
    END IF;
  ELSE
    -- Se considera que el Proveedor es OPCIONAL si la base imponible del gasto es menor que RequiereProveedor.
    -- Si se especifica Proveedor controla su existencia en la base de datos.
    pkg_comun.existe_PROVEEDOR (p_idProveedor);
  END IF;

  -- Controla si idProveedor se modifica.
  IF (p_idProveedor IS NULL)
  THEN
    IF (r_idProveedor IS NOT NULL)
    THEN
      r_idProveedor := p_idProveedor;
    END IF;
  ELSE
    IF (r_idProveedor IS NOT NULL
        AND
        p idProveedor <> r idProveedor)
    THEN
      r_idProveedor := p_idProveedor;
    END IF;
  END IF;

  -- Controla que el Estado del Gasto exista en la base de datos.
  pkg_comun.existe_ESTADO (p_idEstado);

  -- Controla si idEstado se modifica.
  IF (p idEstado <> r idEstado)
  THEN
    r idEstado := p idEstado;
  END IF;

  -- Consulta el Estado del Gasto.
  SELECT
    FormaPagoSN
  INTO
    x_formaPagoSN
  FROM
    Estado
  WHERE
    idEstado = p_idEstado;
  -- Controla si el Estado del Gasto requiere que se especifique una FormaPago.
  IF (x_formaPagoSN = 'T')
  THEN
    -- El Estado del Gasto requiere una FormaPago.
    IF (p idFormaPago IS NULL)
    THEN
      RAISE ErrorFormaPago;
    ELSE
```

```
      -- Controla que la FormaPago exista en la base de datos.
      pkg_comun.existe_FORMAPAGO (p_idFormaPago);
    END IF;
  ELSE
    -- El Estado del Gasto no permite una FormaPago.
    IF (p idFormaPago IS NOT NULL)
    THEN
      RAISE ErrorNoFormaPago;
    END IF;
  END IF;

  -- Controla si idFormaPago se modifica.
  IF (p_idFormaPago IS NULL)
  THEN
    IF (r_idFormaPago IS NOT NULL)
    THEN
      r_idFormaPago := p_idFormaPago;
    END IF;
  ELSE
    IF (r_idFormaPago IS NOT NULL
        AND
        r idFormaPago <> p idFormaPago)
    THEN
      r_idFormaPago := p_idFormaPago;
    END IF;
  END IF;

  -- No se realizan comprobaciones sobre los parámetros: p descripcion, p justificacion, p presupuestoURL, p facturaURL, p contratoURL.

  r_descripcion := p_descripcion;
  r justificacion := p justificacion;
  r presupuestoURL := p presupuestoURL;
  r facturaURL := p facturaURL;
  r contratoURL := p contratoURL;

  -- Actualiza Estadística.
  -- Si ha cambiado Ejercicio, Parlamento, Parlamentario o importeGasto.

  IF (idEjercicioCambia
      OR
      idParlamentoCambia
      OR
      idParlamentarioCambia
      OR
      importeGastoCambia)
  THEN
    -- Desacumula estadísticas antiguas.
    IF (r idParlamentario IS NOT NULL)
    THEN
      -- Si el Gasto era imputable a un Parlamentario entonces se resta la base imponible del gasto
      -- en la tabla de estadística ParlamentarioST que a su vez se resta en la tabla de estadística ParlamentoST.
      estadistica_ParlamentarioST (r_idEjercicio, r_idParlamento, r_idParlamentario, -r_baseImponible);
    ELSE
```

```
      -- Si el Gasto no era imputable a un Parlamentario entonces se resta la base imponible del gasto
      -- en la tabla de estadística ParlamentoST.
      estadistica_ParlamentoST (r_idEjercicio, r_idParlamento, -r_baseImponible);
    END IF;

    -- Acumula estadísticas nuevas.
    IF (p_idParlamentario IS NOT NULL)
    THEN
      -- Si el Gasto ahora es imputable a un Parlamentario entonces se suma la base imponible del gasto
      -- en la tabla de estadística ParlamentarioST que a su vez se suma en la tabla de estadística ParlamentoST.
      estadistica ParlamentarioST (x idEjercicio, p idParlamento, p idParlamentario, p baseImponible);
    ELSE
      -- Si el Gasto ahora no es imputable a un Parlamentario entonces se suma la base imponible del gasto
      -- en la tabla de estadística ParlamentoST.
      estadistica_ParlamentoST (x_idEjercicio, p_idParlamento, p_baseImponible);
    END IF;
  END IF;

  -- Realiza un update en la tabla Gasto.
  UPDATE Gasto SET
    fecha = r fecha,
    idEjercicio = x idEjercicio,
    idParlamento = p_idParlamento,
    idParlamentario = p_idParlamentario,
    idPartido = r_idPartido,
    descripcion = r_descripcion,
    baseImponible = r baseImponible,
    idTipoIVA = r idTipoIVA,
    idCategoriaGasto = r_idCategoriaGasto,
    idEstado = r_idEstado,
    idFormaPago = r idFormaPago,
    idProveedor = r idProveedor,
    justificacion = r justificacion,
    presupuestoURL = r presupuestoURL,
    facturaURL = r_facturaURL,
    contratoURL = r_contratoURL
  WHERE
    idGasto = p idGasto;

  -- Si no ha habido excepciones confirma los cambios en la base de datos.
  COMMIT;
EXCEPTION
  WHEN ErrorFecha THEN
    p rsp := 'ERROR: Debe especificarse una fecha válida para el gasto.';
  WHEN ErrorPagado THEN
    p_rsp := 'ERROR: El gasto [' || p_idGasto || '] no puede eliminarse porque está pagado.';
  WHEN ErrorEjercicioCerrado THEN
    p rsp := 'ERROR: El ejercicio [' || x idEjercicio || '] desde [' || x inicio || '] hasta [' || x fin || '] está cerrado.';
  WHEN ErrorParlamentario THEN
    p rsp := 'ERROR: La categoría de gasto [' || p idCategoriaGasto || '] requiere que el gasto sea asignado a un parlamentario.';
  WHEN ErrorNoParlamentario THEN
    p_rsp := 'ERROR: La categoría de gasto [' || p_idCategoriaGasto || '] no permite que el gasto sea asignado a un parlamentario.';
  WHEN ErrorBaseImponible THEN
```

```
      p_rsp := 'ERROR: Debe especificarse la base imponible del gasto.';
    WHEN ErrorBaseImponibleNegativa THEN
      p_rsp := 'ERROR: La base imponible del gasto no puede ser negativa.';
    WHEN ErrorProveedor THEN
      p rsp := 'ERROR: Debe especificarse un proveedor obligatoriamente en los gastos superiores a [' || RequiereProveedor || '] euros.';
    WHEN ErrorFormaPago THEN
      p_rsp := 'ERROR: El estado [' || p_idEstado || '] requiere que se especifique una forma de pago en el gasto.';
    WHEN ErrorNoFormaPago THEN
      p_rsp := 'ERROR: El estado [' || p_idEstado || '] no permite que se especifique una forma de pago en el gasto.';
    WHEN pkg comun.ErrorGastoNoExiste THEN
      p rsp := 'ERROR: El gasto [' || p idGasto || '] no existe.';
    WHEN pkg comun.ErrorEjercicioNoExiste THEN
      p_rsp := 'ERROR: La fecha [' || p_fecha || '] no pertenece a ningún ejercicio.';
    WHEN pkg_comun.ErrorParlamentoNoExiste THEN
      p_rsp := 'ERROR: El parlamento [' || p_idParlamento || '] no existe.';
    WHEN pkg comun.ErrorCategoriaGastoNoExiste THEN
      p_rsp := 'ERROR: La categoría de gasto [' || p_idCategoriaGasto || '] no existe.';
    WHEN pkg_comun.ErrorParlamentarioNoExiste THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no existe.';
    WHEN pkg_comun.ErrorParlamentarioNoParlamento THEN
      p rsp := 'ERROR: El parlamentario [' || p idParlamentario || '] no estaba en el parlamento [' || p idParlamento || '] en fecha [' || p fecha || '].';
    WHEN pkg comun.ErrorParlamentarioNoPartido THEN
      p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no estaba en ningún partido en fecha [' || p_fecha || '].';
    WHEN pkg_comun.ErrorProveedorNoExiste THEN
      p_rsp := 'ERROR: El proveedor [' || p_idProveedor || '] no existe.';
    WHEN pkg_comun.ErrorTipoIVANoExiste THEN
      p rsp := 'ERROR: El tipo de IVA [' || p idTipoIVA || '] no existe.';
    WHEN pkg comun.ErrorEstadoNoExiste THEN
      p_rsp := 'ERROR: El estado [' || p_idEstado || '] no existe.';
    WHEN pkg_comun.ErrorFormaPagoNoExiste THEN
      p rsp := 'ERROR: La forma de pago [' || p idFormaPago || '] no existe.';
    WHEN OTHERS THEN
      p rsp := 'ERROR: ' || SQLERRM;
      -- Si ha habido excepciones distintas de las anteriores deshace los posibles cambios en la base de datos.
      ROLLBACK;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idGasto = [' || p idGasto || '] + fecha = [' || p fecha || '] + idParlamento = [' || p idParlamento || '] + idParlamentario = [' ||
p idParlamentario || '] + baseImponible = [' || p baseImponible || '] + idTipoIVA = [' || p idTipoIVA || '] + idCategoriaGasto = [' || p idCategoriaGasto ||
'] + idEstado = [' || p_idEstado || '] + idFormaPago = [' || p_idFormaPago || '] + idProveedor = [' || p_idProveedor || '] + descripcion = [' || p_descripcion
|| '] + justificacion = [' || p_justificacion || '] + presupuestoURL = [' || p_presupuestoURL || '] + facturaURL = [' || p_facturaURL || '] + contratoURL = ['
|| p contratoURL || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg_comun.entrada_registro ('pkg_gasto.modificacion', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END modificacion;
```

### 5.5.3.4. Procedimiento pkg_gasto.cerrar_ejercicio.

```
--
-- Procedimiento para cerrar un ejercicio.
--
PROCEDURE cerrar_ejercicio (
  p_idEjercicio IN  Gasto.idEjercicio % TYPE,
  p_rsp         OUT VARCHAR2)
AS
BEGIN
  -- Inicializa el resultado de ejecución del procedimiento a OK.
  p_rsp := 'OK';
  BEGIN
    -- Controla que el Ejercicio a cerrar exista en la base de datos.
    pkg comun.existe ejercicio (p idEjercicio);
    -- Actualiza el Ejercicio
    UPDATE Ejercicio SET
      cerradoSN = 'T'
    WHERE
      idEjercicio = p idEjercicio;
  EXCEPTION
    WHEN pkg comun.ErrorEjercicioNoExiste THEN
      p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
    WHEN OTHERS THEN
      p_rsp := 'ERROR: ' || SQLERRM;
  END;
  -- Construye el string con los parámetros de entrada y sus valores.
  entrada := 'idEjercicio = [' || p idEjercicio || ']';
  -- Construye el string con los parámetros de salida y sus valores.
  salida := 'rsp = ' || p_rsp;
  -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
  pkg comun.entrada registro ('pkg gasto.cerrar ejercicio', USER, SYSTIMESTAMP, entrada, salida);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
END cerrar_ejercicio;
```

### 5.5.3.5. Procedimientos privados del paquete pkg_gasto.

```
/**
Dados un identificador de ejercicio, un identificador de parlamento y un importe, actualiza la columna TotalGastosGenerales de la estadística
ParlamentoST correspondiente.

Este procedimiento es de uso interno dentro del paquete pkg_gasto.

@param p idEjercicio        Identificador del ejercicio de la estadística a actualizar.
@param p idParlamento       Identificador del parlamento de la estadística a actualizar.
@param p importeGasto       Importe a actualizar en la estadística.
*/
PROCEDURE estadistica_ParlamentoST (
  p_idEjercicio  IN Gasto.idEjercicio % TYPE,
  p idParlamento IN Gasto.idParlamento % TYPE,
  p importeGasto IN Gasto.baseImponible % TYPE);

/**
Dados un identificador de ejercicio, un identificador de parlamento, un identificador de parlamentario y un importe, actualiza las columnas
TotalGastosParlamentarios, NumParlamentarios, MinGastoParlamentario y MaxGastoParlamentario de la estadística ParlamentoST correspondiente.

Este procedimiento es de uso interno dentro del paquete pkg gasto.

@param p_idEjercicio        Identificador del ejercicio de la estadística a actualizar.
@param p_idParlamento       Identificador del parlamento de la estadística a actualizar.
@param p idParlamentario    Identificador del parlamentario de la estadística a actualizar.
@param p importeGasto       Importe a actualizar en la estadística.
*/
PROCEDURE estadistica_ParlamentoST (
  p_idEjercicio     IN Gasto.idEjercicio % TYPE,
  p_idParlamento    IN Gasto.idParlamento % TYPE,
  p idParlamentario IN Gasto.idParlamentario % TYPE,
  p importeGasto    IN Gasto.baseImponible % TYPE);

/**
Dados un identificador de ejercicio, un identificador de parlamento, un identificador de parlamentario y un importe, actualiza la columna
TotalGastos de la estadística ParlamentarioST correspondiente.

Este procedimiento es de uso interno dentro del paquete pkg gasto.

@param p_idEjercicio        Identificador del ejercicio de la estadística a actualizar.
@param p_idParlamento       Identificador del parlamento de la estadística a actualizar.
@param p idParlamentario    Identificador del parlamentario de la estadística a actualizar.
@param p importeGasto       Importe a actualizar en la estadística.
*/
PROCEDURE estadistica_ParlamentarioST (
  p_idEjercicio     IN Gasto.idEjercicio % TYPE,
  p idParlamento    IN Gasto.idParlamento % TYPE,
  p idParlamentario IN Gasto.idParlamentario % TYPE,
  p importeGasto    IN Gasto.baseImponible % TYPE);
```

### 5.5.3.5.1. Procedimiento pkg_gasto.estadistica_ParlamentoST.

```
--
-- Procedimiento que actualiza la columna TotalGastosGenerales de la estadística ParlamentoST.
--
PROCEDURE estadistica_ParlamentoST (
  p_idEjercicio  IN Gasto.idEjercicio % TYPE,
  p_idParlamento IN Gasto.idParlamento % TYPE,
  p importeGasto IN Gasto.baseImponible % TYPE)
AS
  existe INTEGER;
BEGIN
  -- Puesto que este procedimiento es privado sólo se puede invocar desde dentro del paquete y por tanto
  -- podemos confiar en que los parámetros vienen comprobados.
  -- Controla si ya existe la estadística de este Ejericio y Parlamento en la tabla ParlamentoST de la base de datos
  SELECT
    COUNT (*)
  INTO
    existe
  FROM
    ParlamentoST
  WHERE
    (idEjercicio = p_idEjercicio
      AND
      idParlamento = p_idParlamento);

  IF (existe = 0)
  THEN
    -- Si no existe se crea la estadística ParlamentoST de este Ejericio y Parlamento.
    -- Este es el primer Gasto acumulado en esta estadística.
    -- La estadística totalGastosGenerales coincidirá con el importeGasto de este primer Gasto acumulado.
    -- La estadística totalGastosParlamentarios será inicializada al valor 0.00 por la base de datos.
    -- La estadística numParlamentarios será inicializada al valor 0 por la base de datos.
    -- La estadística minGastoParlamentario será inicializada al valor 0.00 por la base de datos.
    -- La estadística maxGastoParlamentario será inicializada al valor 0.00 por la base de datos.
    -- Se inserta la estadística ParlamentoST.
    INSERT INTO ParlamentoST
        (idEjercicio,
         idParlamento,
         totalGastosGenerales)
      VALUES
        (p_idEjercicio,
         p idParlamento,
         p importeGasto);
  ELSE
    -- Si existe se actualiza la estadística ParlamentoST de este Ejericio y Parlamento.
    UPDATE ParlamentoST SET
      totalGastosGenerales = totalGastosGenerales + p importeGasto
    WHERE
      (idEjercicio = p idEjercicio
        AND
        idParlamento = p_idParlamento);
```

```
  END IF;

END estadistica_ParlamentoST;

--
-- Procedimiento que actualiza las columnas TotalGastosParlamentarios, NumParlamentarios, MinGastoParlamentario y MaxGastoParlamentario de la estadística
ParlamentoST.
--
PROCEDURE estadistica_ParlamentoST (
  p idEjercicio     IN Gasto.idEjercicio % TYPE,
  p idParlamento    IN Gasto.idParlamento % TYPE,
  p idParlamentario IN Gasto.idParlamentario % TYPE,
  p_importeGasto    IN Gasto.baseImponible % TYPE)
AS
  x_numParlamentarios     ParlamentoST.numParlamentarios % TYPE;
  x minGastoParlamentrio  ParlamentoST.minGastoParlamentario % TYPE;
  x_maxGastoParlamentario ParlamentoST.maxGastoParlamentario % TYPE;
  existe                  INTEGER;
BEGIN
  -- Puesto que este procedimiento es privado sólo se puede invocar desde dentro del paquete y por tanto
  -- podemos confiar en que los parámetros vienen comprobados.
  -- Controla si ya existe la estadística de este Ejericio y Parlamento en la tabla ParlamentoST de la base de datos.
  SELECT
    COUNT (*)
  INTO
    existe
  FROM
    ParlamentoST
  WHERE
    (idEjercicio = p_idEjercicio
      AND
      idParlamento = p idParlamento);

  IF (existe = 0)
  THEN
    -- Si no existe se crea la estadística ParlamentoST de este Ejericio y Parlamento.
    -- Este es el primer Gasto acumulado en esta estadística.
    -- La estadística totalGastosParlamentarios coincidirá con el importeGasto de este primer Gasto acumulado.
    -- La estadística numParlamentarios será igual a 1 (el Parlamentario de este primer Gasto acumulado).
    -- La estadística del minGastoParlamentario coincidirá con el importeGasto de este primer Gasto acumulado.
    -- La estadística del maxGastoParlamentario también coincidirá con el importeGasto de este primer Gasto acumulado.
    -- La estadística totalGastosGenerales será inicializada al valor 0.00 por la base de datos.
    -- Se inserta la estadística ParlamentoST.
    INSERT INTO ParlamentoST
        (idEjercicio,
         idParlamento,
         totalGastosParlamentarios,
         numParlamentarios,
         minGastoParlamentario,
         maxGastoParlamentario)
      VALUES
        (p_idEjercicio,
         p_idParlamento,
```

```
        p_importeGasto,
        1,
        p_importeGasto,
        p importeGasto);
  ELSE
    -- Se calcula de nuevo numParlamentarios: numero de Parlamentarios de este Parlamento en este Ejercicio que tienen totalGastos <> 0.00
    SELECT
      COUNT (*)
    INTO
      x numParlamentarios
    FROM
      ParlamentarioST
    WHERE
      (idEjercicio = p_idEjercicio
        AND
        idParlamento = p idParlamento
        AND
        totalGastos <> 0.00);
    -- Se busca cual es ahora el Parlamentario con el minGastoParlamentario este Ejercicio en este Parlamento.
    SELECT
      MIN (totalGastos)
    INTO
      x_minGastoParlamentrio
    FROM
      ParlamentarioST
    WHERE
      (idEjercicio = p idEjercicio
        AND
        idParlamento = p_idParlamento);
    -- Se busca cual es ahora el Parlamentario con el maxGastoParlamentario este Ejercicio en este Parlamento.
    SELECT
      MAX (totalGastos)
    INTO
      x maxGastoParlamentario
    FROM
      ParlamentarioST
    WHERE
      (idEjercicio = p idEjercicio
        AND
        idParlamento = p_idParlamento);
    -- Finalmente se actualiza la estadística ParlamentoST.
    UPDATE ParlamentoST SET
      totalGastosParlamentarios = totalGastosParlamentarios + p importeGasto,
      numParlamentarios = x numParlamentarios,
      minGastoParlamentario = x_minGastoParlamentrio,
      maxGastoParlamentario = x_maxGastoParlamentario
    WHERE
      (idEjercicio = p idEjercicio
        AND
        idParlamento = p idParlamento);
  END IF;

END estadistica_ParlamentoST;
```

### 5.5.3.5.2. Procedimiento pkg_gasto.estadistica_ParlamentarioST.

```
--
-- Procedimiento que actualiza la columna TotalGastos de la estadística ParlamentarioST.
--
PROCEDURE estadistica_ParlamentarioST (
  p_idEjercicio     IN Gasto.idEjercicio % TYPE,
  p_idParlamento    IN Gasto.idParlamento % TYPE,
  p idParlamentario IN Gasto.idParlamentario % TYPE,
  p importeGasto    IN Gasto.baseImponible % TYPE)
AS
  existe INTEGER;
BEGIN
  -- Puesto que este procedimiento es privado sólo se puede invocar desde dentro del paquete y por tanto
  -- podemos confiar en que los parámetros vienen comprobados.
  -- Controla si ya existe la estadística de este Ejericio, Parlamento y Parlamentario en la tabla ParlamentarioST.
  SELECT
    COUNT (*)
  INTO
    existe
  FROM
    ParlamentarioST
  WHERE
    (idEjercicio = p_idEjercicio
      AND
      idParlamento = p idParlamento
      AND
      idParlamentario = p idParlamentario);

  IF (existe = 0)
  THEN
    -- Si no existe se crea la estadística ParlamentarioST de este Ejericio, Parlamento y Parlamentario.
    -- Este es el primer Gasto acumulado en esta estadística de este Ejercicio, Parlamento y Parlamentario.
    INSERT INTO ParlamentarioST
        (idEjercicio,
         idParlamento,
         idParlamentario,
         totalGastos)
      VALUES
        (p_idEjercicio,
         p_idParlamento,
         p_idParlamentario,
         p importeGasto);
  ELSE
    -- Si existe se actualiza la estadística ParlamentarioST de este Ejercicio, Parlamento y Parlamentario.
    UPDATE ParlamentarioST SET
      totalGastos = totalGastos + p_importeGasto
    WHERE
      (idEjercicio = p idEjercicio
        AND
        idParlamento = p_idParlamento
        AND
```

```
        idParlamentario = p_idParlamentario);
  END IF;
  -- Finalmente se acumula también el importeGasto en la tabla de estadística ParlamentoST correspondiente.

  estadistica ParlamentoST (p idEjercicio, p idParlamento, p idParlamentario, p importeGasto);
END estadistica_ParlamentarioST;
```

### 5.5.4. Paquete pkg_estadistica.

El *script* 3_2_CREAR_PKG_ESTADISTICA.SQL crea en la base de datos el paquete pkg_estadistica de procedimientos que calculan las estadísticas solicitadas.

La solución implementada mediante las tablas ParlamentoST y ParlamentarioST es una generalización de las estadísticas solicitadas.

Esta modificación implica que **las consultas estadísticas no son siempre estrictamente en tiempo constante 1** pero se realizan siempre sobre un subconjunto reducido de filas de la tabla ParlamentoST o de la tabla ParlamentarioST y por tanto no se penaliza el tiempo de cálculo de las estadísticas de forma significativa. Es decir, no se se tienen que hacer consultas sobre la tabla Gasto que involucrarían cientos de miles o millones de filas (lo cual penalizaría el tiempo de cálculo significativamente) sino que se realizan consultas sobre la tabla ParlamentoST o ParlamentarioST que involucran un número reducido de filas que puede oscilar entre 1 o varios cientos (lo cual no penaliza el tiempo de cálculo significativamente).

La mayor versatilidad de esta implementación justifica sobradamente la modificación realizada a los requerimientos. Entre otras mejoras permite la obtención de estadísticas sobre un número arbitrario de ejercicios y no sólo sobre los últimos cuatro. Además esta solución tiene la virtud de almacenar datos estadísticos durante un período arbitrario de ejercicios (característica deseable de un sistema estadístico) y no tener que implementar un procedimiento de limpieza (y pérdida) de datos estadísticos cada cuatro años.

En cualquier caso cabe destacar que **esta modificación fue consensuada con el consultor.**

```
-- PKG_ESTADISTICA specification
CREATE OR REPLACE PACKAGE TFC.pkg_estadistica AS

/**
Dado un parlamento, un último ejercicio y el número de ejercicios anteriores a tener en cuenta calcula la suma total de gastos
de ese parlamento durante ese período.

@param p_idUltimoEjercicio               Identificador del último ejercicio.
@param p numEjerciciosAnteriores         Número de ejercicios anteriores al último a tener en cuenta.
@param p idParlamento                    Identificador del parlamento.
@param p_sumaTotalGastos                 Devuelve la suma total de gastos.
@param p_rsp                             Devuelve el resultado de la ejecución del procedimiento.
*/
PROCEDURE totalGastos parlamento (
  p idUltimoEjercicio       IN  ParlamentoST.idEjercicio % TYPE,
  p numEjerciciosAnteriores IN  NATURAL,
  p_idParlamento            IN  ParlamentoST.idParlamento % TYPE,
  p_sumaTotalGastos         OUT ParlamentoST.totalGastos % TYPE,
  p_rsp                     OUT VARCHAR2);

/**
Dado un ejercicio calcula qué parlamento ha tenido un total de gastos más bajo.

@param p_idEjercicio                     Identificador del ejercicio.
@param p_idParlamento                    Devuelve el identificador del parlamento con el total de gastos más bajo.
@param p minTotalGastos                  Devuelve la suma total de gastos más baja.
@param p_rsp                             Devuelve el resultado de la ejecución del procedimiento.
```

```
*/
PROCEDURE minTotalGastos_parlamentos (
  p_idEjercicio    IN  ParlamentoST.idEjercicio % TYPE,
  p idParlamento   OUT ParlamentoST.idParlamento % TYPE,
  p minTotalGastos OUT ParlamentoST.totalGastos % TYPE,
  p rsp            OUT VARCHAR2);

/**
Dado un ejercicio calcula qué parlamento ha tenido un total de gastos más alto.

@param p idEjercicio                    Identificador del ejercicio a tener en cuenta.
@param p idParlamento                   Devuelve el identificador del parlamento con el total de gastos más alto.
@param p_minTotalGastos                 Devuelve la suma total de gastos más alta.
@param p_rsp                            Devuelve el resultado de la ejecución del procedimiento.
*/
PROCEDURE maxTotalGastos parlamentos (
  p_idEjercicio    IN  ParlamentoST.idEjercicio % TYPE,
  p_idParlamento   OUT ParlamentoST.idParlamento % TYPE,
  p_maxTotalGastos OUT ParlamentoST.totalGastos % TYPE,
  p_rsp            OUT VARCHAR2);

/**
Dado un ejercicio calcula la suma total de los gastos de todos los parlamentos durante ese ejercicio.

@param p_idEjercicio                    Identificador del ejercicio.
@param p_maxTotalGastos                 Devuelve la suma total de gastos de todos los parlamentos ese ejercicio.
@param p rsp                            Devuelve el resultado de la ejecución del procedimiento.
*/
PROCEDURE sumaTotalGastos_parlamentos (
  p_idEjercicio     IN  ParlamentoST.idEjercicio % TYPE,
  p sumaTotalGastos OUT ParlamentoST.totalGastos % TYPE,
  p rsp             OUT VARCHAR2);

/**
Dado un parlamentario, un último ejercicio y el número de ejercicios anteriores a tener en cuenta calcula la suma total de gastos
atribuibles a ese parlamentario durante ese período.

@param p idUltimoEjercicio              Identificador del último ejercicio.
@param p numEjerciciosAnteriores        Número de ejercicios anteriores al último a tener en cuenta.
@param p_idParlamentario                Identificador del parlamentario.
@param p_sumaTotalGastos                Devuelve la suma total de gastos del parlamentario.
@param p rsp                            Devuelve el resultado de la ejecución del procedimiento.
*/
PROCEDURE totalGastos parlamentario (
  p_idUltimoEjercicio       IN  ParlamentarioST.idEjercicio % TYPE,
  p_numEjerciciosAnteriores IN  NATURAL,
  p_idParlamentario         IN  ParlamentarioST.idParlamentario % TYPE,
  p sumaTotalGastos         OUT ParlamentarioST.totalGastos % TYPE,
  p rsp                     OUT VARCHAR2);

/**
Dado un ejercicio calcula qué parlamentario ha tenido un total de gastos más bajo.
```

```
@param p_idEjercicio                      Identificador del ejercicio.
@param p_idParlamentario                  Devuelve el identificador del parlamentario con el total de gastos más bajo.
@param p_minTotalGastos                   Devuelve la suma total de gastos más baja.
@param p rsp                              Devuelve el resultado de la ejecución del procedimiento.
*/
PROCEDURE minTotalGastos parlamentario (
  p_idEjercicio     IN  ParlamentarioST.idEjercicio % TYPE,
  p_idParlamentario OUT ParlamentarioST.idParlamentario % TYPE,
  p_minTotalGastos  OUT ParlamentarioST.totalGastos % TYPE,
  p rsp             OUT VARCHAR2);

/**
Dado un ejercicio calcula qué parlamentario ha tenido un total de gastos más alto.

@param p_idEjercicio                      Identificador del ejercicio.
@param p idParlamentario                  Devuelve el identificador del parlamentario con el total de gastos más alto.
@param p_maxTotalGastos                   Devuelve la suma total de gastos más alta.
@param p_rsp                              Devuelve el resultado de la ejecución del procedimiento.
*/
PROCEDURE maxTotalGastos_parlamentario (
  p idEjercicio     IN  ParlamentarioST.idEjercicio % TYPE,
  p idParlamentario OUT ParlamentarioST.idParlamentario % TYPE,
  p_maxTotalGastos  OUT ParlamentarioST.totalGastos % TYPE,
  p_rsp             OUT VARCHAR2);

/**
Dado un ejercicio y un parlamento, calcula la diferencia entre el parlamentario que tiene más gastos imputados y el
parlamentario que tiene menos gastos imputados.

@param p_idEjercicio                      Identificador del ejercicio.
@param p idParlamento                     Identificacor del parlamento.
@param p minGastoParlamentario            Devuelve el importe del gasto mínimo de un parlamentario en ese ejercicio y parlamento.
@param p maxGastoParlamentario            Devuelve el importe del gasto máximo de un parlamentario en ese ejercicio y parlamento.
@parma p diferenciaGastoParlamentario     Devuelve la diferencia en entre el gasto máximo y mínimo de un parlamentario en ese ejercicio y parlamento.
@param p_rsp                              Devuelve el resultado de la ejecución del procedimiento.
*/
PROCEDURE diferenciaGastoParlamentario (
  p idEjercicio                  IN  ParlamentoST.idEjercicio % TYPE,
  p idParlamento                 IN  ParlamentoST.idParlamento % TYPE,
  p_minGastoParlamentario        OUT ParlamentoST.minGastoParlamentario % TYPE,
  p_maxGastoParlamentario        OUT ParlamentoST.maxGastoParlamentario % TYPE,
  p diferenciaGastoParlamentario OUT ParlamentoST.totalGastos % TYPE,
  p rsp                          OUT VARCHAR2);

/**
Dado un ejercicio, calcula la media del gasto de un parlamentario durante ese ejercicio teniendo en cuenta a todos los
parlamentarios de todos los parlamentos.

@param p idEjercicio                      Identificador del ejercicio.
@param p sumaGastosParlamentarios         Devuelve la suma de todos los gastos imputables a parlamentarios durante el ejercicio.
@parma p_sumaParlamentarios               Devuelve el número de todos los parlamentarios durante el ejercicio.
@param p_mediaGastosParlamentarios        Devuelve la media del gasto imputable a parlamentarios durante el ejercicio.
*/
```

```
PROCEDURE mediaGastosParlamentarios (
  p_idEjercicio               IN  ParlamentoST.idEjercicio % TYPE,
  p_sumaGastosParlamentarios  OUT ParlamentoST.totalGastosParlamentarios % TYPE,
  p_sumaParlamentarios        OUT ParlamentoST.numParlamentarios % TYPE,
  p_mediaGastosParlamentarios OUT ParlamentoST.totalGastosParlamentarios % TYPE,
  p_rsp                       OUT VARCHAR2);

/**
Dado un ejercicio, calcula el parlamento con la media de gasto por parlamentario más baja durante ese ejercicio.

@param p_idEjercicio                    Identificador del ejercicio.
@param p_idParlamento                   Devuelve el identificador del parlamento con una media de gasto por parlamentario más baja.
@param p_nombre                         Devuelve el nombre del parlamento con una media de gasto por parlamentario más baja.
@param p_minMediaGastoParlamentario     Devuelve la media de gasto por parlamentario más baja.
@param p_rsp                            Devuelve el resultado de la ejecución del procedimiento.
*/
PROCEDURE minMediaGastoParlamentario (
  p_idEjercicio               IN  ParlamentoST.idEjercicio % TYPE,
  p_idParlamento              OUT ParlamentoST.idParlamento % TYPE,
  p_nombre                    OUT Parlamento.nombre % TYPE,
  p_minMediaGastoParlamentario OUT ParlamentoST.mediaGastoParlamentario % TYPE,
  p_rsp                       OUT VARCHAR2);

/**
Dado un ejercicio, calcula el parlamento con la media de gasto por parlamentario más alta durante ese ejercicio.

@param p_idEjercicio                    Identificador del ejercicio.
@param p_idParlamento                   Devuelve el identificador del parlamento con una media de gasto por parlamentario más alta.
@param p_nombre                         Devuelve el nombre del parlamento con una media de gasto por parlamentario más alta.
@param p_maxMediaGastoParlamentario     Devuelve la media de gasto por parlamentario más alta.
@param p_rsp                            Devuelve el resultado de la ejecución del procedimiento.
*/
PROCEDURE maxMediaGastoParlamentario (
  p_idEjercicio               IN  ParlamentoST.idEjercicio % TYPE,
  p_idParlamento              OUT ParlamentoST.idParlamento % TYPE,
  p_nombre                    OUT Parlamento.nombre % TYPE,
  p_maxMediaGastoParlamentario OUT ParlamentoST.mediaGastoParlamentario % TYPE,
  p_rsp                       OUT VARCHAR2);

END pkg_estadistica;
```

### 5.5.4.1. Procedimiento pkg_estadistica.totalGastos_parlamento.

Esta estadística corresponde a la pregunta del enunciado siguiente: “1. Dado un parlamento: la suma de todos los gastos de los últimos 4 años (incluyendo los gastos generales de los parlamentos y los gastos asociados a los parlamentarios).”

La solución aportada es una generalización que permite fijados un parlamento, un último ejercicio y el número de ejercicios anteriores a éste (valor natural entre 0 y n-1) obtener la suma de los gastos de un parlamento durante los últimos n ejercicios fijado el último.

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de n filas.

```
--
-- Dado un parlamento, un último ejercicio y el número de ejercicios anteriores a tener en cuenta calcula la suma total de gastos
-- de ese parlamento durante ese período.
--
PROCEDURE totalGastos parlamento (
  p_idUltimoEjercicio       IN  ParlamentoST.idEjercicio % TYPE,
  p_numEjerciciosAnteriores IN  NATURAL,
  p_idParlamento            IN  ParlamentoST.idParlamento % TYPE,
  p_sumaTotalGastos         OUT ParlamentoST.totalGastos % TYPE,
  p rsp                     OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idParlamento exista en la base de datos.
      pkg_comun.existe_parlamento (p_idParlamento);
      -- Controla que p_idUltimoEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idUltimoEjercicio);
      -- Controla que existan estadisticas para este Ejercicio y este Parlamento.
      pkg comun.existe ParlamentoST (p idUltimoEjercicio, p idParlamento);
      -- Calcula la suma de totalGastos de este Parlamento en los últimos ejercicios solicitados existentes.
      SELECT
        SUM (totalGastos)
      INTO
        p_sumaTotalGastos
      FROM
        ParlamentoST
      WHERE
        idParlamento = p idParlamento
        AND
        (p idUltimoEjercicio - p numEjerciciosAnteriores) <= idEjercicio
        AND
        idEjercicio <= p_idUltimoEjercicio;
    EXCEPTION
      WHEN pkg comun.ErrorParlamentoNoExiste THEN
        p rsp := 'ERROR: El parlamento [' || p idParlamento || '] no existe.';
      WHEN pkg comun.ErrorEjercicioNoExiste THEN
```

```
        p_rsp := 'ERROR: El último ejercicio indicado [' || p_idUltimoEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorEstadisticaNoExiste THEN
        p_rsp := 'ERROR: No existe estadística para el ejercicio [' || p_idUltimoEjercicio || '] y parlamento [' || p_idParlamento || ']';
      WHEN OTHERS THEN
        p rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idUltimoEjercicio = [' || p_idUltimoEjercicio || '] + numEjerciciosAnteriores = [' || p_numEjerciciosAnteriores || '] + idParlamento = [' ||
p_idParlamento || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'sumaTotalGastos = [' || p sumaTotalGastos || '] + rsp = ' || p rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg_comun.entrada_registro ('pkg_estadistica.totalGastos_parlamento', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END totalGastos_parlamento;
```

### 5.5.4.2. Procedimiento pkg_estadistica.minTotalGastos_parlamentos.

Esta estadística corresponde a la pregunta del enunciado siguiente: “3. Dado un año concreto: el parlamento que ha tenido un total de gastos más alto (incluyendo los gastos generales de los parlamentos y los gastos asociados a los parlamentarios).”

Como solución a esta pregunta se han aportado el cálculo del parlamento con un total de gastos más alto y más bajo en dos funciones análogas (ésta calcula el más bajo).

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de un número de filas igual al número de parlamentos existentes en el ejercicio solicitado.

```
--
-- Dado un ejercicio calcula qué parlamento ha tenido un total de gastos más bajo.
--
PROCEDURE minTotalGastos parlamentos (
  p_idEjercicio    IN  ParlamentoST.idEjercicio % TYPE,
  p_idParlamento   OUT ParlamentoST.idParlamento % TYPE,
  p_minTotalGastos OUT ParlamentoST.totalGastos % TYPE,
  p rsp            OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idEjercicio exista en la base de datos.
      pkg comun.existe ejercicio (p idEjercicio);
      -- Controla que existan estadisticas para este Ejercicio.
      pkg_comun.existe_ParlamentoST (p_idEjercicio);
      -- Consulta la estadística este Ejercicio.
      SELECT
        MIN (totalGastos)
      INTO
        p_minTotalGastos
      FROM
        ParlamentoST
      WHERE
        idEjercicio = p idEjercicio;
      -- Busca a qué Parlamento pertenece ese minTotalGastos.
      -- Podría existir más de un Parlamento cuyo totalGastos este Ejercicio coincida con minTotalGastos.
      -- En ese caso elegiremos el de menor idParlamento.
      SELECT
        MIN (idParlamento)
      INTO
        p_idParlamento
      FROM
        ParlamentoST
      WHERE
        idEjercicio = p_idEjercicio
```

```
        AND
        totalGastos = p_minTotalGastos;
    EXCEPTION
      WHEN pkg comun.ErrorParlamentoNoExiste THEN
        p rsp := 'ERROR: El parlamento [' || p idParlamento || '] no existe.';
      WHEN pkg comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorEstadisticaNoExiste THEN
        p_rsp := 'ERROR: No existe estadística para el ejercicio [' || p_idEjercicio || ']';
      WHEN OTHERS THEN
        p rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'idParlamento = [' || p idParlamento || '] + minTotalGastos = [' || p minTotalGastos || '] + rsp = ' || p rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg_comun.entrada_registro ('pkg_estadistica.minTotalGastos_parlamentos', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END minTotalGastos_parlamentos;
```

### 5.5.4.3. Procedimiento pkg_estadistica.maxTotalGastos_parlamentos.

Esta estadística corresponde a la pregunta del enunciado siguiente: “3. Dado un año concreto: el parlamento que ha tenido un total de gastos más alto (incluyendo los gastos generales de los parlamentos y los gastos asociados a los parlamentarios).”

Como solución a esta pregunta se han aportado el cálculo del parlamento con un total de gastos más alto y más bajo en dos funciones análogas (ésta calcula el más alto).

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de un número de filas igual al número de parlamentos existentes en el ejercicio solicitado.

```
--
-- Dado un ejercicio calcula qué parlamento ha tenido un total de gastos más alto.
--
PROCEDURE maxTotalGastos parlamentos (
  p_idEjercicio    IN  ParlamentoST.idEjercicio % TYPE,
  p_idParlamento   OUT ParlamentoST.idParlamento % TYPE,
  p_maxTotalGastos OUT ParlamentoST.totalGastos % TYPE,
  p rsp            OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idEjercicio exista en la base de datos.
      pkg comun.existe ejercicio (p idEjercicio);
      -- Controla que existan estadisticas para este Ejercicio.
      pkg_comun.existe_ParlamentoST (p_idEjercicio);
      -- Consulta la estadística este Ejercicio.
      SELECT
        MAX (totalGastos)
      INTO
        p_maxTotalGastos
      FROM
        ParlamentoST
      WHERE
        idEjercicio = p idEjercicio;
      -- Busca a qué Parlamento pertenece ese maxTotalGastos.
      -- Podría existir más de un Parlamento cuyo totalGastos este Ejercicio coincida con maxTotalGastos.
      -- En ese caso elegiremos el de menor idParlamento.
      SELECT
        MIN (idParlamento)
      INTO
        p_idParlamento
      FROM
        ParlamentoST
      WHERE
        idEjercicio = p_idEjercicio
```

```
        AND
        totalGastos = p_maxTotalGastos;
    EXCEPTION
      WHEN pkg comun.ErrorParlamentoNoExiste THEN
        p rsp := 'ERROR: El parlamento [' || p idParlamento || '] no existe.';
      WHEN pkg comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorEstadisticaNoExiste THEN
        p_rsp := 'ERROR: No existe estadística para el ejercicio [' || p_idEjercicio || ']';
      WHEN OTHERS THEN
        p rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'idParlamento = [' || p idParlamento || '] + maxTotalGastos = [' || p maxTotalGastos || '] + rsp = ' || p rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg_comun.entrada_registro ('pkg_estadistica.maxTotalGastos_parlamentos', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END maxTotalGastos_parlamentos;
```

### 5.5.4.4. Procedimiento pkg_estadistica.sumaTotalGastos_parlamentos.

Esta estadística corresponde a la pregunta del enunciado siguiente: “8. Dado un año concreto: la suma de todos los gastos de todos los parlamentos (incluyendo los gastos generales de los parlamentos y los gastos asociados a los parlamentarios).”

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de un número de filas igual al número de parlamentos existentes en el ejercicio solicitado.

```
--
-- Dado un ejercicio calcula la suma total de los gastos de todos los parlamentos durante ese ejercicio.
--
PROCEDURE sumaTotalGastos parlamentos (
  p idEjercicio     IN  ParlamentoST.idEjercicio % TYPE,
  p sumaTotalGastos OUT ParlamentoST.totalGastos % TYPE,
  p_rsp             OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p rsp := 'OK';
    BEGIN
      -- Controla que p_idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Controla que existan estadisticas para este Ejercicio.
      pkg comun.existe ParlamentoST (p idEjercicio);
      -- Consulta la estadística este Ejercicio.
      SELECT
        SUM (totalGastos)
      INTO
        p sumaTotalGastos
      FROM
        ParlamentoST
      WHERE
        idEjercicio = p_idEjercicio;
    EXCEPTION
      WHEN pkg comun.ErrorEjercicioNoExiste THEN
        p rsp := 'ERROR: El ejercicio [' || p idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorEstadisticaNoExiste THEN
        p_rsp := 'ERROR: No existe estadística para el ejercicio [' || p_idEjercicio || ']';
      WHEN OTHERS THEN
        p rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'sumaTotalGastos = [' || p_sumaTotalGastos || '] + rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg_comun.entrada_registro ('pkg_estadistica.sumaTotalGastos_parlamentos', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
```

```
      DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END sumaTotalGastos_parlamentos;
```

### 5.5.4.5. Procedimiento pkg_estadistica.totalGastos_parlamentario.

Esta estadística corresponde a la pregunta del enunciado siguiente: “4. Dado un parlamentario: el gasto total que ha tenido los últimos cuatro años.”

La solución aportada es una generalización que permite fijados un parlamentario, un último ejercicio y el número de ejercicios anteriores a éste (valor natural entre 0 y n-1) obtener la suma de los gastos de un parlamentario durante los últimos n ejercicios fijado el último.

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de n filas.

```
--
-- Dado un parlamentario, un último ejercicio y el número de ejercicios anteriores a tener en cuenta calcula la suma total de gastos
-- atribuibles a ese parlamentario durante ese período.
--
PROCEDURE totalGastos parlamentario (
  p idUltimoEjercicio       IN  ParlamentarioST.idEjercicio % TYPE,
  p numEjerciciosAnteriores IN  NATURAL,
  p idParlamentario         IN  ParlamentarioST.idParlamentario % TYPE,
  p_sumaTotalGastos         OUT ParlamentarioST.totalGastos % TYPE,
  p_rsp                     OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idParlamentario exista en la base de datos.
      pkg comun.existe parlamentario (p idParlamentario);
      -- Controla que p idUltimoEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idUltimoEjercicio);
      -- Controla que existan estadisticas para este Ejercicio y este Parlamentario.
      pkg_comun.existe_ParlamentarioST (p_idUltimoEjercicio, p_idParlamentario);
      -- Calcula la suma de totalGastos de este Parlamentario en los últimos ejercicios solicitados existentes.
      SELECT
        SUM (totalGastos)
      INTO
        p_sumaTotalGastos
      FROM
        ParlamentarioST
      WHERE
        idParlamentario = p_idParlamentario
        AND
        (p_idUltimoEjercicio - p_numEjerciciosAnteriores) <= idEjercicio
        AND
        idEjercicio <= p idUltimoEjercicio;
    EXCEPTION
      WHEN pkg_comun.ErrorParlamentarioNoExiste THEN
        p_rsp := 'ERROR: El parlamentario [' || p_idParlamentario || '] no existe.';
      WHEN pkg_comun.ErrorEjercicioNoExiste THEN
        p rsp := 'ERROR: El último ejercicio indicado [' || p idUltimoEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorEstadisticaNoExiste THEN
```

```
        p_rsp := 'ERROR: No existe estadística para el ejercicio [' || p_idUltimoEjercicio || '] y parlamentario [' || p_idParlamentario || ']';
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idUltimoEjercicio = [' || p idUltimoEjercicio || '] + numEjerciciosAnteriores = [' || p numEjerciciosAnteriores || '] + idParlamentario = ['
|| p_idParlamentario || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'sumaTotalGastos = [' || p_sumaTotalGastos || '] + rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg comun.entrada registro ('pkg estadistica.totalGastos parlamentario', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END totalGastos parlamentario;
```

### 5.5.4.6. Procedimiento pkg_estadistica.minTotalGastos_parlamentario.

Esta estadística corresponde a la pregunta del enunciado siguiente: “6. Dado un año concreto: el parlamentario que ha tenido mayor gasto asociado y el de menor (teniendo en cuenta todos los parlamentos).”

Como solución a esta pregunta se han aportado el cálculo del parlamentario con un mayor y menor gasto asociado en dos funciones análogas (ésta calcula el más bajo).

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de un número de filas igual al número de parlamentarios existentes en todos los parlamentos en el ejercicio solicitado.

Podría haberse reducido significativamente el número de filas a consultar identificando al parlamentario con menor gasto asociado en cada parlamento en la tabla ParlamentoST. Esta podría considerarse una mejora posible al diseño.

```
--
-- Dado un ejercicio calcula qué parlamentario ha tenido un total de gastos más bajo.
--
PROCEDURE minTotalGastos parlamentario (
  p idEjercicio     IN  ParlamentarioST.idEjercicio % TYPE,
  p_idParlamentario OUT ParlamentarioST.idParlamentario % TYPE,
  p_minTotalGastos  OUT ParlamentarioST.totalGastos % TYPE,
  p_rsp             OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Controla que existan estadisticas para este Ejercicio.
      pkg_comun.existe_ParlamentarioST (p_idEjercicio);
      -- Calcula el mínimo totalGastos de este Ejercicio.
      SELECT
        MIN (totalGastos)
      INTO
        p_minTotalGastos
      FROM
        ParlamentarioST
      WHERE
        idEjercicio = p idEjercicio;
      -- Busca a qué Parlamentario pertenece ese minTotalGastos.
      -- Podría existir más de un Parlamentario cuyo totalGastos este Ejercicio coincida con minTotalGastos.
      -- En ese caso elegiremos el de menor idParlamentario.
      SELECT
        MIN (idParlamentario)
      INTO
        p_idParlamentario
      FROM
```

```
        ParlamentarioST
      WHERE
        idEjercicio = p_idEjercicio
        AND
        totalGastos = p minTotalGastos;
    EXCEPTION
      WHEN pkg_comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorEstadisticaNoExiste THEN
        p rsp := 'ERROR: No existe estadística para el ejercicio [' || p idEjercicio || ']';
      WHEN OTHERS THEN
        p rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'idParlamentario = [' || p_idParlamentario || '] + minTotalGastos = [' || p_minTotalGastos || '] + rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg_comun.entrada_registro ('pkg_estadistica.minTotalGastos_parlamentario', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END minTotalGastos_parlamentario;
```

### 5.5.4.7. Procedimiento pkg_estadistica.maxTotalGastos_parlamentario.

Esta estadística corresponde a la pregunta del enunciado siguiente: “6. Dado un año concreto: el parlamentario que ha tenido mayor gasto asociado y el de menor (teniendo en cuenta todos los parlamentos).”

Como solución a esta pregunta se han aportado el cálculo del parlamentario con un mayor y menor gasto asociado en dos funciones análogas (ésta calcula el más alto).

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de un número de filas igual al número de parlamentarios existentes en todos los parlamentos en el ejercicio solicitado.

Podría haberse reducido significativamente el número de filas a consultar identificando al parlamentario con mayor gasto asociado en cada parlamento en la tabla ParlamentoST. Esta podría considerarse una mejora posible al diseño.

```
--
-- Dado un ejercicio calcula qué parlamentario ha tenido un total de gastos más alto.
--
PROCEDURE maxTotalGastos parlamentario (
  p idEjercicio     IN  ParlamentarioST.idEjercicio % TYPE,
  p_idParlamentario OUT ParlamentarioST.idParlamentario % TYPE,
  p_maxTotalGastos  OUT ParlamentarioST.totalGastos % TYPE,
  p_rsp             OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Controla que existan estadisticas para este Ejercicio.
      pkg_comun.existe_ParlamentarioST (p_idEjercicio);
      -- Calcula el máximo totalGastos de este Ejercicio.
      SELECT
        MAX (totalGastos)
      INTO
        p_maxTotalGastos
      FROM
        ParlamentarioST
      WHERE
        idEjercicio = p idEjercicio;
      -- Busca a qué Parlamentario pertenece ese maxTotalGastos.
      -- Podría existir más de un Parlamentario cuyo totalGastos este Ejercicio coincida con maxTotalGastos.
      -- En ese caso elegiremos el de menor idParlamentario.
      SELECT
        MIN (idParlamentario)
      INTO
        p_idParlamentario
      FROM
```

```
        ParlamentarioST
      WHERE
        idEjercicio = p_idEjercicio
        AND
        totalGastos = p maxTotalGastos;
    EXCEPTION
      WHEN pkg_comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorEstadisticaNoExiste THEN
        p rsp := 'ERROR: No existe estadística para el ejercicio [' || p idEjercicio || ']';
      WHEN OTHERS THEN
        p rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'idParlamentario = [' || p_idParlamentario || '] + maxTotalGastos = [' || p_maxTotalGastos || '] + rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg_comun.entrada_registro ('pkg_estadistica.maxTotalGastos_parlamentario', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END maxTotalGastos_parlamentario;
```

### 5.5.4.8. Procedimiento pkg_estadistica.diferenciaGastoParlamentario.

Esta estadística corresponde a la pregunta del enunciado siguiente: “2. Dado un parlamento y un año: la diferencia en euros entre el parlamentario que más gastos asociados tiene y el que menos.”

La respuesta se obtiene en tiempo constante 1 puesto que involucra la consulta de una sola fila.

```
--
-- Dado un ejercicio y un parlamento, calcula la diferencia entre el parlamentario que tiene más gastos imputados y el
-- parlamentario que tiene menos gastos imputados.
--
PROCEDURE diferenciaGastoParlamentario (
  p_idEjercicio                 IN  ParlamentoST.idEjercicio % TYPE,
  p_idParlamento                IN  ParlamentoST.idParlamento % TYPE,
  p_minGastoParlamentario       OUT ParlamentoST.minGastoParlamentario % TYPE,
  p_maxGastoParlamentario       OUT ParlamentoST.maxGastoParlamentario % TYPE,
  p_diferenciaGastoParlamentario OUT ParlamentoST.totalGastos % TYPE,
  p_rsp                         OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idParlamento exista en la base de datos.
      pkg_comun.existe_parlamento (p_idParlamento);
      -- Controla que p_idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Controla que existan estadisticas para este Ejercicio y este Parlamento.
      pkg comun.existe ParlamentoST (p idEjercicio, p idParlamento);
      -- Consulta la estadística este Ejercicio y Parlamento.
      SELECT
        minGastoParlamentario,
        maxGastoParlamentario
      INTO
        p minGastoParlamentario,
        p maxGastoParlamentario
      FROM
        ParlamentoST
      WHERE
        idParlamento = p idParlamento
        AND
        idEjercicio = p_idejercicio;
      p_diferenciaGastoParlamentario := p_maxGastoParlamentario - p_minGastoParlamentario;
    EXCEPTION
      WHEN pkg comun.ErrorParlamentoNoExiste THEN
        p rsp := 'ERROR: El parlamento [' || p idParlamento || '] no existe.';
      WHEN pkg_comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorEstadisticaNoExiste THEN
        p rsp := 'ERROR: No existe estadística para el ejercicio [' || p idEjercicio || '] y parlamento [' || p idParlamento || ']';
```

```
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p idEjercicio || '] + idParlamento = [' || p idParlamento || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'minGastoParlamentario = [' || p_minGastoParlamentario || '] + maxGastoParlamentario = [' || p_maxGastoParlamentario || '] +
maxTotalGastos_parlamentario = [' || p_diferenciaGastoParlamentario || ' + rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg comun.entrada registro ('pkg estadistica.maxTotalGastos parlamentario', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END diferenciaGastoParlamentario;
```

### 5.5.4.9. Procedimiento pkg_estadistica.mediaGastosParlamentarios.

Esta estadística corresponde a la pregunta del enunciado siguiente: “5. Dado un año concreto: La media del gasto de un parlamentario durante aquel año (teniendo en cuenta todos los parlamentarios de todos los parlamentos).”

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de un número de filas igual al número de parlamentos en el ejercicio solicitado.

```
--
-- Dado un ejercicio, calcula la media del gasto de un parlamentario durante ese ejercicio teniendo en cuenta a todos los
-- parlamentarios de todos los parlamentos.
--
PROCEDURE mediaGastosParlamentarios (
  p_idEjercicio               IN  ParlamentoST.idEjercicio % TYPE,
  p_sumaGastosParlamentarios  OUT ParlamentoST.totalGastosParlamentarios % TYPE,
  p_sumaParlamentarios        OUT ParlamentoST.numParlamentarios % TYPE,
  p_mediaGastosParlamentarios OUT ParlamentoST.totalGastosParlamentarios % TYPE,
  p_rsp                       OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p_idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Controla que existan estadísticas para este Ejercicio.
      pkg_comun.existe_ParlamentoST (p_idEjercicio);
      -- Consulta la estadística este Ejercicio.
      SELECT
        SUM (totalGastosParlamentarios),
        SUM (numParlamentarios)
      INTO
        p_sumaGastosParlamentarios,
        p_sumaParlamentarios
      FROM
        ParlamentoST
      WHERE
        idEjercicio = p_idEjercicio;
      p_mediaGastosParlamentarios := MEDIA (p_sumaGastosParlamentarios, p_sumaParlamentarios);
    EXCEPTION
      WHEN pkg_comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorEstadisticaNoExiste THEN
        p_rsp := 'ERROR: No existe estadística para el ejercicio [' || p_idEjercicio || ']';
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || ']';
```

```
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'sumaGastosParlamentarios = [' || p_sumaGastosParlamentarios || '] + sumaParlamentarios = [' || p_sumaParlamentarios || '] +
mediaGastosParlamentarios = [' || p_mediaGastosParlamentarios || '] + rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg comun.entrada registro ('pkg estadistica.mediaGastosParlamentarios', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END mediaGastosParlamentarios;
```

### 5.5.4.10. Procedimiento pkg_estadistica.minMediaGastoParlamentario.

Esta estadística corresponde a la pregunta del enunciado siguiente: “7. Dado un año concreto: El nombre del parlamento con una media de gasto asociado a sus parlamentarios más alta y el valor de esta media de gasto.”

Como solución a esta pregunta se han aportado el cálculo del parlamento con una media de gasto por parlamentario más alta y más baja en dos funciones análogas (ésta calcula la media más baja).

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de un número de filas igual al número de parlamentos en el ejercicio solicitado.

```
--
-- Dado un ejercicio, calcula el parlamento con la media de gasto por parlamentario más baja durante ese ejercicio.
--
PROCEDURE minMediaGastoParlamentario (
  p_idEjercicio                IN  ParlamentoST.idEjercicio % TYPE,
  p_idParlamento               OUT ParlamentoST.idParlamento % TYPE,
  p_nombre                     OUT Parlamento.nombre % TYPE,
  p minMediaGastoParlamentario OUT ParlamentoST.mediaGastoParlamentario % TYPE,
  p rsp                        OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p rsp := 'OK';
    BEGIN
      -- Controla que p idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Controla que existan estadísticas para este Ejercicio.
      pkg_comun.existe_ParlamentoST (p_idEjercicio);
      -- Consulta la estadística este Ejercicio.
      SELECT
        MIN (mediaGastoParlamentario)
      INTO
        p_minMediaGastoParlamentario
      FROM
        ParlamentoST
      WHERE
        idEjercicio = p_idEjercicio;
      -- Busca a qué Parlamento pertenece ese minMediaGastoParlamentario.
      -- Podría existir más de un Parlamento cuya mediaGastoParlamentario este Ejercicio coincida con minMediaGastoParlamentario.
      -- En ese caso elegiremos el de menor idParlamento.
      SELECT
        MIN (idParlamento)
      INTO
        p_idParlamento
      FROM
        ParlamentoST
      WHERE
```

```
        idEjercicio = p_idEjercicio
        AND
        mediaGastoParlamentario = p_minMediaGastoParlamentario;
      -- Consulta el parlamento.
      SELECT
        nombre
      INTO
        p_nombre
      FROM
        Parlamento
      WHERE
        idParlamento = p idParlamento;
    EXCEPTION
      WHEN pkg_comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN pkg comun.ErrorEstadisticaNoExiste THEN
        p_rsp := 'ERROR: No existe estadística para el ejercicio [' || p_idEjercicio || ']';
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p idEjercicio || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'idParlamento = [' || p_idParlamento || '] + nombre = [' || p_nombre || '] + minMediaGastoParlamentario = [' || p_minMediaGastoParlamentario ||
'] + rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg comun.entrada registro ('pkg estadistica.minMediaGastoParlamentario', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END minMediaGastoParlamentario;
```

### 5.5.4.11. Procedimiento pkg_estadistica.maxMediaGastoParlamentario.

Esta estadística corresponde a la pregunta del enunciado siguiente: “7. Dado un año concreto: El nombre del parlamento con una media de gasto asociado a sus parlamentarios más alta y el valor de esta media de gasto.”

Como solución a esta pregunta se han aportado el cálculo del parlamento con una media de gasto por parlamentario más alta y más baja en dos funciones análogas (ésta calcula la media más alta).

La respuesta no se obtiene en tiempo constante 1 puesto que no involucra la consulta de una sola fila sino de un número de filas igual al número de parlamentos en el ejercicio solicitado.

```
--
-- Dado un ejercicio, calcula el parlamento con la media de gasto por parlamentario más alta durante ese ejercicio.
--
PROCEDURE maxMediaGastoParlamentario (
  p_idEjercicio                IN  ParlamentoST.idEjercicio % TYPE,
  p_idParlamento               OUT ParlamentoST.idParlamento % TYPE,
  p_nombre                     OUT Parlamento.nombre % TYPE,
  p maxMediaGastoParlamentario OUT ParlamentoST.mediaGastoParlamentario % TYPE,
  p rsp                        OUT VARCHAR2)
AS
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p rsp := 'OK';
    BEGIN
      -- Controla que p idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Controla que existan estadísticas para este Ejercicio.
      pkg_comun.existe_ParlamentoST (p_idEjercicio);
      -- Consulta la estadística este Ejercicio.
      SELECT
        MAX (mediaGastoParlamentario)
      INTO
        p_maxMediaGastoParlamentario
      FROM
        ParlamentoST
      WHERE
        idEjercicio = p_idEjercicio;
      -- Busca a qué Parlamento pertenece esta maxMediaGastoParlamentario.
      -- Podría existir más de un Parlamento cuya mediaGastoParlamentario este Ejercicio coincida con maxMediaGastoParlamentario.
      -- En ese caso elegiremos el de menor idParlamento.
      SELECT
        MIN (idParlamento)
      INTO
        p_idParlamento
      FROM
        ParlamentoST
      WHERE
```

```
        idEjercicio = p_idEjercicio
        AND
        mediaGastoParlamentario = p_maxMediaGastoParlamentario;
      -- Consulta el parlamento.
      SELECT
        nombre
      INTO
        p_nombre
      FROM
        Parlamento
      WHERE
        idParlamento = p idParlamento;
    EXCEPTION
      WHEN pkg_comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN pkg comun.ErrorEstadisticaNoExiste THEN
        p_rsp := 'ERROR: No existe estadística para el ejercicio [' || p_idEjercicio || ']';
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p idEjercicio || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'idParlamento = [' || p_idParlamento || '] + nombre = [' || p_nombre || '] + maxMediaGastoParlamentario = [' || p_maxMediaGastoParlamentario ||
'] + rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg comun.entrada registro ('pkg estadistica.maxMediaGastoParlamentario', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END maxMediaGastoParlamentario;
```

### 5.5.5. Paquete pkg_listado.

El *script* 3_4_CREAR_PKG_LISTADO.SQL crea en la base de datos el paquete pkg_listado de procedimientos que realizan los listados solicitados.

```
-- PKG_LISTADO specification
CREATE OR REPLACE PACKAGE TFC.pkg listado AS

/**
Dados un ejercicio, un parlamento y un parlamentario, obtiene un listado de todos los gastos asociados a dicho
parlamentario en ese ejercicio y ese parlamento en orden descendente por el importe de cada gasto.

En cada línea se muestran el identificador del gasto, la fecha en que se produjo, su base imponible, el porcentaje de IVA y el importe con IVA.

@param p_idEjercicio        Identificador del ejercicio.
@param p_idParlamento       Identificador del parlamento.
@param p_idParlamentario    Identificador del parlamentario.
@param p rsp                Resultado de la ejecución del procedimiento.
*/
PROCEDURE gastos_parlamentario (
  p_idEjercicio     IN  Gasto.idEjercicio % TYPE,
  p_idParlamento    IN  Gasto.idParlamento % TYPE,
  p_idParlamentario IN  Gasto.idParlamentario % TYPE,
  p rsp             OUT VARCHAR2);

/**
Dados un ejercicio y un parlamento, obtiene un listado del total de gastos asociado a cada parlamentario de ese parlamento en ese ejercicio.

En cada línea se muestran el nombre y apellidos de cada parlamentario, el nombre del partido al que pertenece (o del último al que perteneció
si ya no milita en ninguno), y el total de sus gastos.

@param p_idEjercicio        Identificador del ejercicio.
@param p_idParlamento          Identificador del parlamento.
@param p rsp                Resultado de la ejecución del procedimiento.
*/
PROCEDURE totalGastos parlamentarios (
  p_idEjercicio  IN  Gasto.idEjercicio % TYPE,
  p_idParlamento IN  Gasto.idParlamento % TYPE,
  p rsp          OUT VARCHAR2);

/**
Dado un ejercicio, obtiene un listado con el total de gastos de cada parlamento.

En cada línea se muestran el nombre del parlamento, los gastos atribuibles a los parlamentarios, los gastos generales y los gastos totales de ese
parlamento, el número de parlamentarios, la media de gasto por parlamentario, el gasto mínimo y el gasto máximo realizados por un parlamentario de ese
parlamento.

@param p_idEjercicio        Identificador del ejercicio.
@param p_rsp                Resultado de la ejecución del procedimiento.
*/
PROCEDURE totalGastos parlamentos (
  p idEjercicio IN  Gasto.idEjercicio % TYPE,
  p_rsp         OUT VARCHAR2);
```

```
/**
Dados un ejercicio y un parlamento, obtiene un listado de los gastos segmentado por las categorías de gasto.

En cada línea se muestran la categoría del gasto, la suma de la base imponible de los gastos de esa categoría en ese parlamento y el importe con IVA.

@param p_idEjercicio         Identificador del ejercicio.
@param p_idParlamento           Identificador del parlamento.
@param p_rsp                Resultado de la ejecución del procedimiento.
*/
PROCEDURE gastos por categoria (
  p idEjercicio  IN  Gasto.idEjercicio % TYPE,
  p_idParlamento IN  Gasto.idParlamento % TYPE,
  p_rsp          OUT VARCHAR2);

/**
Dado un ejercicio y un parlamento, obtiene un listado de los parlamentarios que superan el gasto medio por parlamentario.

En cada línea se muestran el nombre y apellidos del parlamentario, el nombre del partido al que pertenece (o del último al que perteneció), el importe total
de
los gastos atribuibles a dicho parlamentario en ese ejercicio en ese parlamento, la desviación respecto a la media del gasto por parlamentario absoluta y en
porcentaje.

@param p_idEjercicio         Identificador del ejercicio.
@param p_idParlamento           Identificador del parlamento.
@param p_rsp                Resultado de la ejecución del procedimiento.
*/
PROCEDURE parlamentarios gasto superior (
  p_idEjercicio  IN  Gasto.idEjercicio % TYPE,
  p_idParlamento IN  Gasto.idParlamento % TYPE,
  p rsp          OUT VARCHAR2);

/**
Dado un ejercicio, obtiene un listado del estado contable de todos los parlamentos.

En cada línea se muestran el nombre del parlamento, el importe de los gastos pendientes de aprobación diferenciando los pagados y los pendientes de pago,
y el importe de los gastos aprobados diferenciando los pagados de los pendientes de pago.

idEstado = 1 ( Pendiente de aprobación. No pagado. )
idEstado = 2 ( Pendiente de aprobación. Pagado. Gasto sujeto a posible devolución. )
idEstado = 3 ( Aprobado. No pagado. )
idEstado = 4 ( Aprobado. Pagado. )

@param p idEjercicio         Identificador del ejercicio.
@param p_rsp                Resultado de la ejecución del procedimiento.
*/
PROCEDURE estado_contable_parlamentos (
  p idEjercicio IN  Gasto.idEjercicio % TYPE,
  p rsp         OUT VARCHAR2);

END pkg_listado;
```

### 5.5.5.1. Procedimiento pkg_listado.gastos_parlamentario.

Este listado corresponde al listado solicitado en el enunciado siguiente: “a. Dado un parlamento, un año y un parlamentario: el listado de todos los gastos asociados al parlamentario en aquel año. Todo esto ordenado de forma descendente por el valor de cada gasto”.

```
--
-- Dados un ejercicio, un parlamento y un parlamentario, obtiene un listado de todos los gastos asociados a dicho
-- parlamentario en ese ejercicio y ese parlamento en orden descendente por el importe de cada gasto.
--
PROCEDURE gastos parlamentario (
  p idEjercicio     IN  Gasto.idEjercicio % TYPE,
  p idParlamento    IN  Gasto.idParlamento % TYPE,
  p_idParlamentario IN  Gasto.idParlamentario % TYPE,
  p_rsp             OUT VARCHAR2)
AS
  CURSOR C GASTOS IS
    SELECT
      idGasto,
      fecha,
      baseImponible,
      TipoIVA.PorcentajeIVA,
      baseImponible * (1 + TipoIVA.PorcentajeIVA / 100) AS importeGasto
    FROM
      Gasto,
      TipoIVA
    WHERE
      idEjercicio = p_idEjercicio
      AND
      idParlamento = p idParlamento
      AND
      idParlamentario = p_idParlamentario
      AND
      TipoIVA.idTipoIVA = Gasto.idTipoIVA
    ORDER BY importeGasto DESC;

  Parlamento_nombre       Parlamento.nombre % TYPE;
  Parlamentario_nombre    Parlamentario.nombre % TYPE;
  Parlamentario_apellidos Parlamentario.apellidos % TYPE;

  Gasto id                Gasto.idGasto % TYPE;
  Gasto fecha             Gasto.fecha % TYPE;
  Gasto_baseImponible     Gasto.baseImponible % TYPE;
  Gasto_importeGasto      Gasto.baseImponible % TYPE;
  TipoIVA_porcentaje      TipoIVA.PorcentajeIVA % TYPE;

  baseImponible           Gasto.baseImponible % TYPE;
  importeGasto            Gasto.baseImponible % TYPE;
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p rsp := 'OK';
```

```
    BEGIN
      -- Controla que p_idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Controla que p idParlamento exista en la base de datos.
      pkg comun.existe parlamento (p idParlamento);
      SELECT
        nombre
      INTO
        Parlamento_nombre
      FROM
        Parlamento
      WHERE
        idParlamento = p_idParlamento;
      -- Controla que p_idParlamentario exista en la base de datos.
      pkg_comun.existe_parlamentario (p_idParlamentario);
      SELECT
        nombre,
        apellidos
      INTO
        Parlamentario_nombre,
        Parlamentario apellidos
      FROM
        Parlamentario
      WHERE
        idParlamentario = p_idParlamentario;
      -- Imprime cabecera del listado.
      DBMS OUTPUT.ENABLE (BUFFER SIZE => NULL);
      DBMS OUTPUT.NEW LINE;
      DBMS_OUTPUT.PUT_LINE ('       L I S T A D O    G A S T O S    P A R L A M E N T A R I O       ');
      DBMS_OUTPUT.NEW_LINE;
      DBMS OUTPUT.PUT LINE ('EJERCICIO     = [' || TO CHAR (p idEjercicio, 'FM9999') || ']');
      DBMS OUTPUT.PUT LINE ('PARLAMENTO    = [' || TO CHAR (p idParlamento, 'FM0999') || ']    [' || Parlamento nombre || ']');
      DBMS OUTPUT.PUT LINE ('PARLAMENTARIO = [' || TO CHAR (p idParlamentario, 'FM0999999') || '] [' || Parlamentario nombre || ' ' || Parlamentario apellidos
|| ']');
      DBMS_OUTPUT.PUT_LINE ('================================================================================');
      DBMS_OUTPUT.PUT_LINE ('GASTO      FECHA           BASE IMPONIBLE   %IVA                IMPORTE');
      DBMS_OUTPUT.PUT LINE ('--------------------------------------------------------------------------------');
      OPEN C GASTOS;
      LOOP
        FETCH C_GASTOS
        INTO Gasto_id,
        Gasto fecha,
        Gasto baseImponible,
        TipoIVA porcentaje,
        Gasto_importeGasto;
        EXIT
        WHEN C_GASTOS % NOTFOUND;
        -- Imprime cada línea.
        DBMS OUTPUT.PUT LINE (TO CHAR (Gasto id, 'FM09999999') || ' ' || TO CHAR (Gasto fecha, 'DD-MM-YYYY') || ' ' || TO CHAR (Gasto baseImponible,
'99G999G990D00U') || ' ' || TO CHAR (TipoIVA porcentaje, '90D00') || '% ' || TO CHAR (Gasto importeGasto, '99G999G990D00U'));
      END LOOP;
      CLOSE C_GASTOS;
      DBMS_OUTPUT.PUT_LINE ('================================================================================');
```

```
      SELECT
        SUM (baseImponible),
        SUM (baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
      INTO
        baseImponible,
        importeGasto
      FROM
        Gasto,
        TipoIVA
      WHERE
        idEjercicio = p idEjercicio
        AND
        idParlamento = p_idParlamento
        AND
        idParlamentario = p_idParlamentario
        AND
        TipoIVA.idTipoIVA = Gasto.idTipoIVA;
      -- Imprime totales.
      DBMS_OUTPUT.PUT_LINE ('                    ' || TO_CHAR (baseImponible, '99G999G990D00U') || '         ' || TO_CHAR (importeGasto, '99G999G990D00U'));
      DBMS_OUTPUT.NEW_LINE;
    EXCEPTION
      WHEN pkg comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorParlamentoNoExiste THEN
        p_rsp := 'ERROR: El parlamento [' || p_idParlamento || '] no existe.';
      WHEN pkg_comun.ErrorParlamentarioNoExiste THEN
        p rsp := 'ERROR: El parlamentario [' || p idParlamentario || '] no existe.';
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p idEjercicio || '] + idParlamento = [' || p idParlamento || '] + idParlamentario = [' || p idParlamentario || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'rsp = ' || p rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg_comun.entrada_registro ('pkg_listado.gastos_parlamentario', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END gastos_parlamentario;
```

### 5.5.5.2. Procedimiento pkg_listado.totalGastos_parlamentarios.

Este listado corresponde al listado solicitado en el enunciado siguiente: “b. Dado un parlamento y un año: el listado de los parlamentarios con el gasto directo que ha realizado cada uno, incluyendo: su nombre y apellidos, su partido político actual, la suma de todos los gastos asociados. Ordenado de forma descendente por el valor de la suma de los gastos”.

```
--
-- Dados un ejercicio y un parlamento, obtiene un listado del total de gastos asociado a cada parlamentario de ese parlamento en ese ejercicio.
--
PROCEDURE totalGastos_parlamentarios (
  p_idEjercicio  IN  Gasto.idEjercicio % TYPE,
  p_idParlamento IN  Gasto.idParlamento % TYPE,
  p_rsp          OUT VARCHAR2)
AS
  CURSOR C_TOTALGASTOS IS
    SELECT
      ParlamentarioST.idParlamentario,
      ParlamentarioST.totalGastos,
      Parlamentario.nombre,
      Parlamentario.apellidos,
      Partido.idPartido,
      Partido.nombre
    FROM
      ParlamentarioST,
      Parlamentario,
      Partido
    WHERE
      (
      ParlamentarioST.idEjercicio = p_idEjercicio
        AND
        ParlamentarioST.idParlamento = p_idParlamento
        )
      AND
      (
      Parlamentario.idParlamentario = ParlamentarioST.idParlamentario
      )
      AND
      (
      Partido.idPartido =
      (SELECT PartidoDH.idPartido
          FROM PartidoDH
        WHERE (
          PartidoDH.idParlamentario = ParlamentarioST.idParlamentario
          AND (
          (
          -- El parlamentario pertenece a un partido en la actualidad.
          hasta IS NULL
          )
          OR (
          -- Cogemos el último partido al que perteneció el parlamentario.
          hasta =
```

```
          (SELECT MAX (hasta)
              FROM PartidoDH
             WHERE PartidoDH.idParlamentario = ParlamentarioST.idParlamentario))
          )
          )))
    ORDER BY ParlamentarioST.totalGastos DESC;

  Parlamento_nombre         Parlamento.nombre % TYPE;
  Parlamentario_id          Parlamentario.idParlamentario % TYPE;
  Parlamentario nombre      Parlamentario.nombre % TYPE;
  Parlamentario apellidos   Parlamentario.apellidos % TYPE;
  Parlamentario totalGastos ParlamentarioST.totalGastos % TYPE;
  Partido_id                Partido.idPartido % TYPE;
  Partido_nombre            Partido.nombre % TYPE;
  totalGastos               ParlamentarioST.totalGastos % TYPE;
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idEjercicio exista en la base de datos.
      pkg comun.existe ejercicio (p idEjercicio);
      -- Controla que p_idParlamento exista en la base de datos.
      pkg_comun.existe_parlamento (p_idParlamento);
      SELECT
        nombre
      INTO
        Parlamento nombre
      FROM
        Parlamento
      WHERE
        idParlamento = p idParlamento;
      -- Imprime cabecera del listado.
      DBMS OUTPUT.ENABLE (BUFFER SIZE => NULL);
      DBMS_OUTPUT.NEW_LINE;
      DBMS_OUTPUT.PUT_LINE ('         L I S T A D O    T O T A L    G A S T O S    P A R L A M E N T A R I O S         ');
      DBMS OUTPUT.NEW LINE;
      DBMS OUTPUT.PUT LINE ('EJERCICIO     = [' || TO CHAR (p idEjercicio, 'FM9999') || ']');
      DBMS OUTPUT.PUT LINE ('PARLAMENTO    = [' || TO CHAR (p idParlamento, 'FM0999') || '] [' || Parlamento nombre || ']');
      DBMS_OUTPUT.PUT_LINE ('=============================================================================================');
      DBMS_OUTPUT.PUT_LINE ('PARLAMENTARIO                          PARTIDO                                TOTAL GASTOS');
      DBMS OUTPUT.PUT LINE ('---------------------------------------------------------------------------------------------');
      OPEN C TOTALGASTOS;
      LOOP
        FETCH C_TOTALGASTOS
        INTO Parlamentario_id,
        Parlamentario_totalGastos,
        Parlamentario nombre,
        Parlamentario apellidos,
        Partido id,
        Partido_nombre;
        EXIT
        WHEN C_TOTALGASTOS % NOTFOUND;
```

```
        -- Imprime las líneas.
        DBMS_OUTPUT.PUT_LINE (TO_CHAR (Parlamentario_id, 'FM0999999') || ' ' || RPAD (Parlamentario_nombre || ' ' || Parlamentario_apellidos, 30, ' ') || ' '
|| TO_CHAR (Partido_id, 'FM0999999') || ' ' || RPAD (Partido_nombre, 20, ' ') || ' ' || TO_CHAR (Parlamentario_totalGastos, '99G999G990D00U'));
      END LOOP;
      CLOSE C TOTALGASTOS;
      DBMS OUTPUT.PUT LINE ('=============================================================================================');
      SELECT
        SUM (ParlamentarioST.totalGastos)
      INTO
        totalGastos
      FROM
        ParlamentarioST
      WHERE
        ParlamentarioST.idEjercicio = p_idEjercicio
        AND
        ParlamentarioST.idParlamento = p idParlamento;
      -- Imprime totales.
      DBMS_OUTPUT.PUT_LINE ('                                                                    ' || TO_CHAR (totalGastos, '99G999G990D00U'));
      DBMS_OUTPUT.NEW_LINE;
    EXCEPTION
      WHEN pkg comun.ErrorEjercicioNoExiste THEN
        p rsp := 'ERROR: El ejercicio [' || p idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorParlamentoNoExiste THEN
        p_rsp := 'ERROR: El parlamento [' || p_idParlamento || '] no existe.';
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || '] + idParlamento = [' || p_idParlamento || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'rsp = ' || p rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg comun.entrada registro ('pkg estadistica.totalGastos parlamentarios', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END totalGastos parlamentarios;
```

### 5.5.5.3. Procedimiento pkg_listado.totalGastos_parlamentos.

Este listado corresponde al listado solicitado en el enunciado siguiente: “c. Dado un año: el listado del gasto total de todos los parlamentos, incluyendo: nombre del parlamento, número de parlamentarios, gasto medio de sus parlamentarios (teniendo en cuenta sólo los gastos directos realizados por los parlamentarios), la suma de los gastos totales asociados a los parlamentarios, la suma de los gastos asociados al parlamento (es decir, los que no están asociados a un parlamentario en concreto), y la suma de todos los gastos.

```
--
-- Dado un ejercicio, obtiene un listado con el total de gastos de cada parlamento.
--
PROCEDURE totalGastos parlamentos (
  p_idEjercicio IN  Gasto.idEjercicio % TYPE,
  p_rsp         OUT VARCHAR2)
AS
  CURSOR C_TOTALGASTOS IS
    SELECT
      ParlamentoST.idParlamento,
      ParlamentoST.totalGastosParlamentarios,
      ParlamentoST.totalGastosGenerales,
      ParlamentoST.totalGastos,
      ParlamentoST.numParlamentarios,
      ParlamentoST.mediaGastoParlamentario,
      ParlamentoST.minGastoParlamentario,
      ParlamentoST.maxGastoParlamentario,
      Parlamento.nombre
    FROM
      ParlamentoST,
      Parlamento
    WHERE
      (
      ParlamentoST.idEjercicio = p_idEjercicio
      )
      AND
      (
      Parlamento.idParlamento = ParlamentoST.idParlamento
      )
    ORDER BY ParlamentoST.idParlamento;

  Parlamento id                  ParlamentoST.idParlamento % TYPE;
  Parlamento_nombre              Parlamento.nombre % TYPE;
  Parlamento_totalParlamentarios ParlamentoST.totalGastosParlamentarios % TYPE;
  Parlamento totalGenerales      ParlamentoST.totalGastosGenerales % TYPE;
  Parlamento totalGastos         ParlamentoST.totalGastos % TYPE;
  Parlamento numParlamentarios   ParlamentoST.numParlamentarios % TYPE;
  Parlamento mediaParlamentario  ParlamentoST.mediaGastoParlamentario % TYPE;
  Parlamento_minParlamentario    ParlamentoST.minGastoParlamentario % TYPE;
  Parlamento_maxParlamentario    ParlamentoST.maxGastoParlamentario % TYPE;

  totalGastosParlamentarios      ParlamentoST.totalGastosParlamentarios % TYPE;
  totalGastosGenerales           ParlamentoST.totalGastosGenerales % TYPE;
```

```
  totalGastos                   ParlamentoST.totalGastos % TYPE;
  numParlamentarios             ParlamentoST.numParlamentarios % TYPE;
  mediaGastoParlamentario       ParlamentoST.mediaGastoParlamentario % TYPE;
  minGastoParlamentario         ParlamentoST.minGastoParlamentario % TYPE;
  maxGastoParlamentario         ParlamentoST.maxGastoParlamentario % TYPE;

BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p_idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Imprime cabecera del listado.
      DBMS_OUTPUT.ENABLE (BUFFER_SIZE => NULL);
      DBMS_OUTPUT.NEW LINE;
      DBMS_OUTPUT.PUT_LINE ('                                                    L I S T A D O   T O T A L   G A S T O S   P A R L A M E N T O S
');
      DBMS_OUTPUT.NEW_LINE;
      DBMS_OUTPUT.PUT_LINE ('EJERCICIO     = [' || TO_CHAR (p_idEjercicio, 'FM9999') || ']');
      DBMS_OUTPUT.PUT LINE
('=============================================================================================================================================
================');
      DBMS_OUTPUT.PUT_LINE ('                                                                                G A S T O S                                P A R
L A M E N T A R I O S                        ');
      DBMS_OUTPUT.PUT_LINE ('                                                 ------------------------------------------------------------  ----------------------
-------------------------------------------');
      DBMS_OUTPUT.PUT LINE ('PARLAMENTO                                       PARLAMENTARIOS             GENERALES               TOTALES  NUMERO       GASTO
MINIMO         GASTO MEDIO        GASTO MAXIMO');
      DBMS_OUTPUT.PUT_LINE ('-----------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------');
      OPEN C TOTALGASTOS;
      LOOP
        FETCH C TOTALGASTOS
        INTO Parlamento_id,
        Parlamento_totalParlamentarios,
        Parlamento totalGenerales,
        Parlamento totalGastos,
        Parlamento numParlamentarios,
        Parlamento_mediaParlamentario,
        Parlamento_minParlamentario,
        Parlamento maxParlamentario,
        Parlamento nombre;
        EXIT
        WHEN C_TOTALGASTOS % NOTFOUND;
        -- Imprime las líneas.
        DBMS_OUTPUT.PUT_LINE (TO_CHAR (Parlamento_id, 'FM0999') || ' ' || RPAD (Parlamento_nombre, 30, ' ') || ' ' || TO_CHAR
(Parlamento totalParlamentarios, '99G999G990D00U') || ' ' || TO CHAR (Parlamento totalGenerales, '99G999G990D00U') || ' ' || TO CHAR (Parlamento totalGastos,
'99G999G990D00U') || '    ' || TO CHAR (Parlamento numParlamentarios, '999') || ' ' || TO CHAR (Parlamento minParlamentario, '999G999D00U') || ' ' || TO CHAR
(Parlamento mediaParlamentario, '999G999D00U') || ' ' || TO CHAR (Parlamento maxParlamentario, '999G999D00U'));
      END LOOP;
      CLOSE C_TOTALGASTOS;
      DBMS_OUTPUT.PUT_LINE
```

```
('================================================================================================================================================================
================');
      SELECT
        SUM (totalGastosParlamentarios),
        SUM (totalGastosGenerales),
        SUM (totalGastos),
        SUM (numParlamentarios),
        MEDIA (SUM (totalGastosParlamentarios), SUM (numParlamentarios)),
        MIN (minGastoParlamentario),
        MAX (maxGastoParlamentario)
      INTO
        totalGastosParlamentarios,
        totalGastosGenerales,
        totalGastos,
        numParlamentarios,
        mediaGastoParlamentario,
        minGastoParlamentario,
        maxGastoParlamentario
      FROM
        ParlamentoST
      WHERE
        ParlamentoST.idEjercicio = p idEjercicio;
      -- Imprime totales.
      DBMS_OUTPUT.PUT_LINE ('                                        ' || TO_CHAR (totalGastosParlamentarios, '99G999G990D00U') || ' ' || TO_CHAR
(totalGastosGenerales, '99G999G990D00U') || ' ' || TO_CHAR (totalGastos, '99G999G990D00U') || '    ' || TO_CHAR (numParlamentarios, '999') || ' ' || TO_CHAR
(minGastoParlamentario, '999G999D00U') || ' ' || TO_CHAR (mediaGastoParlamentario, '999G999D00U') || ' ' || TO_CHAR (maxGastoParlamentario, '999G999D00U'));
      DBMS OUTPUT.NEW LINE;
    EXCEPTION
      WHEN pkg_comun.ErrorEjercicioNoExiste THEN
        p_rsp := 'ERROR: El ejercicio [' || p_idEjercicio || '] no existe.';
      WHEN OTHERS THEN
        p rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'rsp = ' || p rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg comun.entrada registro ('pkg estadistica.totalGastos Parlamentos', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END totalGastos parlamentos;
```

### 5.5.5.4. Procedimiento pkg_listado.gastos_por_categoria.

Este listado corresponde al listado solicitado en el enunciado siguiente: “d. Dado un año y un parlamento: el listado de los gastos segmentado por categorías de gasto, es decir, un listado que devuelva las columnas siguientes: código de la categoría de gasto, descripción de la categoría de gasto, suma de los gastos totales asociados a la categoría de gasto durante el año indicado y en el parlamento indicado”.

```
--
-- Dados un ejercicio y un parlamento, obtiene un listado de los gastos segmentado por las categorías de gasto.
--
PROCEDURE gastos_por_categoria (
  p_idEjercicio  IN  Gasto.idEjercicio % TYPE,
  p_idParlamento IN  Gasto.idParlamento % TYPE,
  p_rsp          OUT VARCHAR2)
AS
  CURSOR C_GASTOS IS
    SELECT
      Gasto.idCategoriaGasto,
      CategoriaGasto.descripcion,
      SUM (Gasto.baseImponible),
      SUM (Gasto.baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
    FROM
      Gasto,
      CategoriaGasto,
      TipoIVA
    WHERE
      (
      idEjercicio = p_idEjercicio
        AND
        idParlamento = p_idParlamento
        )
      AND
      (
      CategoriaGasto.idCategoriaGasto = Gasto.idCategoriaGasto
      )
      AND
      (
      TipoIVA.idTipoIVA = Gasto.idTipoIVA
      )
    GROUP BY Gasto.idCategoriaGasto, CategoriaGasto.descripcion;

  Parlamento_nombre          Parlamento.nombre % TYPE;
  Parlamentario_nombre       Parlamentario.nombre % TYPE;

  Gasto idCategoriaGasto     Gasto.idCategoriaGasto % TYPE;
  Gasto baseImponible        Gasto.baseImponible % TYPE;
  Gasto_importeGasto         Gasto.baseImponible % TYPE;

  CategoriaGasto_descripcion CategoriaGasto.descripcion % TYPE;
  TipoIVA porcentaje         TipoIVA.PorcentajeIVA % TYPE;

  baseImponible              Gasto.baseImponible % TYPE;
```

```
  importeGasto              Gasto.baseImponible % TYPE;

BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p rsp := 'OK';
    BEGIN
      -- Controla que p_idEjercicio exista en la base de datos.
      pkg_comun.existe_ejercicio (p_idEjercicio);
      -- Controla que p idParlamento exista en la base de datos.
      pkg comun.existe parlamento (p idParlamento);
      SELECT
        nombre
      INTO
        Parlamento_nombre
      FROM
        Parlamento
      WHERE
        idParlamento = p_idParlamento;
      -- Imprime cabecera del listado.
      DBMS OUTPUT.ENABLE (BUFFER SIZE => NULL);
      DBMS OUTPUT.NEW LINE;
      DBMS_OUTPUT.PUT_LINE ('                  L I S T A D O   G A S T O S   P O R   C A T E G O R I A S                  ');
      DBMS_OUTPUT.NEW_LINE;
      DBMS_OUTPUT.PUT_LINE ('Ejercicio     = [' || TO_CHAR (p_idEjercicio, 'FM9999') || ']');
      DBMS_OUTPUT.PUT_LINE ('Parlamento    = [' || TO_CHAR (p_idParlamento, 'FM0999') || ']    [' || Parlamento_nombre || ']');
      DBMS OUTPUT.PUT LINE ('=============================================================================================================');
      DBMS OUTPUT.PUT LINE ('CATEGORIA                                                    BASE IMPONIBLE              IMPORTE');
      DBMS_OUTPUT.PUT_LINE ('-------------------------------------------------------------------------------------------------------------');
      OPEN C_GASTOS;
      LOOP
        FETCH C GASTOS
        INTO Gasto idCategoriaGasto,
        CategoriaGasto descripcion,
        Gasto_baseImponible,
        Gasto_importeGasto;
        EXIT
        WHEN C GASTOS % NOTFOUND;
        DBMS OUTPUT.PUT LINE (TO CHAR (Gasto idCategoriaGasto, 'FM0999') || ' ' || RPAD (CategoriaGasto descripcion, 50, ' ') || ' ' || TO CHAR
(Gasto_baseImponible, '99G999G990D00U') || TO_CHAR (Gasto_importeGasto, '99G999G990D00U'));
      END LOOP;
      CLOSE C GASTOS;
      DBMS OUTPUT.PUT LINE ('=============================================================================================================');
      SELECT
        SUM (baseImponible),
        SUM (baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
      INTO
        baseImponible,
        importeGasto
      FROM
        Gasto,
        TipoIVA
      WHERE
```

```
        (
        idEjercicio = p_idejercicio
          AND
          idParlamento = p idParlamento
          )
        AND
        (
        TipoIVA.idTipoIVA = Gasto.idTipoIVA
        );
      -- Imprime totales.
      DBMS OUTPUT.PUT LINE ('                                                    ' || TO CHAR (baseImponible, '99G999G990D00U') || TO CHAR (importeGasto,
'99G999G990D00U'));
      DBMS_OUTPUT.NEW_LINE;
    EXCEPTION
      WHEN pkg_comun.ErrorEjercicioNoExiste THEN
        p rsp := 'ERROR: El ejercicio [' || p idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorParlamentoNoExiste THEN
        p_rsp := 'ERROR: El parlamento [' || p_idParlamento || '] no existe.';
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || '] + idParlamento = [' || p_idParlamento || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg comun.entrada registro ('pkg listado.gastos por categoria', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END gastos por categoria;
```

### 5.5.5.5. Procedimiento pkg_listado.parlamentarios_gasto_superior.

Este listado corresponde al listado solicitado en el enunciado siguiente: “e. Dado un parlamento y un año: el listado de los parlamentarios que superen la media de gasto de todos los parlamentarios de aquel parlamento durante el año indicado, incluyendo: Su nombre y apellidos, su partido político actual, suma de todos los gastos directos asociados al parlamentario durante aquel año, porcentaje en que supera el gasto medio”.

```
--
-- Dado un ejercicio y un parlamento, obtiene un listado de los parlamentarios que superan el gasto medio por parlamentario.
--
PROCEDURE parlamentarios_gasto_superior (
  p_idEjercicio  IN  Gasto.idEjercicio % TYPE,
  p_idParlamento IN  Gasto.idParlamento % TYPE,
  p_rsp          OUT VARCHAR2)
AS
  CURSOR C_PARLAMENTARIOS IS
    SELECT
      ParlamentarioST.idParlamentario,
      ParlamentarioST.totalGastos,
      Parlamentario.nombre,
      Parlamentario.apellidos,
      Partido.idPartido,
      Partido.nombre,
      (ParlamentarioST.totalGastos - ParlamentoST.mediaGastoParlamentario),
      ((ParlamentarioST.totalGastos - ParlamentoST.mediaGastoParlamentario) / ParlamentoST.mediaGastoParlamentario * 100.00)
    FROM
      ParlamentarioST,
      ParlamentoST,
      Parlamentario,
      Partido
    WHERE
      (
      ParlamentarioST.idEjercicio = p_idEjercicio
        AND
        ParlamentarioST.idParlamento = p_idParlamento
        )
      AND
      (
      ParlamentoST.idEjercicio = p_idEjercicio
        AND
        ParlamentoST.idParlamento = p_idParlamento
        )
      AND
      (
      ParlamentarioST.totalGastos > ParlamentoST.mediaGastoParlamentario
      )
      AND
      (
      Parlamentario.idParlamentario = ParlamentarioST.idParlamentario
      )
      AND
      (
```

```
      Partido.idPartido =
      (SELECT PartidoDH.idPartido
          FROM PartidoDH
        WHERE (
          PartidoDH.idParlamentario = ParlamentarioST.idParlamentario
          AND (
          (
          -- El parlamentario pertenece a un Partido en la actualidad.
          hasta IS NULL
          )
          OR (
          -- Cogemos el último Partido al que perteneció el parlamentario.
          hasta =
          (SELECT MAX (hasta)
              FROM PartidoDH
            WHERE PartidoDH.idParlamentario = ParlamentarioST.idParlamentario))
          )
          )));

  Parlamento_nombre          Parlamento.nombre % TYPE;
  Parlamento mediagasto      ParlamentoST.mediaGastoParlamentario % TYPE;

  Parlamentario_id           Parlamentario.idParlamentario % TYPE;
  Parlamentario_nombre       Parlamentario.nombre % TYPE;
  Parlamentario_apellidos    Parlamentario.apellidos % TYPE;
  Parlamentario_totalGastos ParlamentarioST.totalGastos % TYPE;
  Parlamentario desviacion   ParlamentarioST.totalGastos % TYPE;
  Parlamentario porcentaje   ParlamentarioST.totalGastos % TYPE;

  Partido_id                 Partido.idPartido % TYPE;
  Partido nombre             Partido.nombre % TYPE;

  totalGastos                ParlamentarioST.totalGastos % TYPE;
  desviacion                 ParlamentarioST.totalGastos % TYPE;
  porcentaje                 ParlamentarioST.totalGastos % TYPE;

BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idEjercicio exista en la base de datos.
      pkg comun.existe ejercicio (p idEjercicio);
      -- Controla que p idParlamento exista en la base de datos.
      pkg_comun.existe_parlamento (p_idParlamento);
      -- Consulta el parlamento.
      SELECT
        nombre
      INTO
        Parlamento nombre
      FROM
        Parlamento
      WHERE
```

```
        idParlamento = p_idParlamento;
      -- Consulta la estadística ParlamentoST de este Ejercicio y Parlamento.
      SELECT
        mediaGastoParlamentario
      INTO
        Parlamento mediagasto
      FROM
        ParlamentoST
      WHERE
        idEjercicio = p idEjercicio
        AND
        idParlamento = p idParlamento;
      -- Imprime cabecera del listado
      DBMS_OUTPUT.ENABLE (BUFFER_SIZE => NULL);
      DBMS_OUTPUT.NEW_LINE;
      DBMS_OUTPUT.PUT LINE ('               L I S T A D O   P A R L A M E N T A R I O S   G A S T O   S U P E R I O R   M E D I A                 ');
      DBMS_OUTPUT.NEW_LINE;
      DBMS_OUTPUT.PUT_LINE ('EJERCICIO     = [' || TO_CHAR (p_idEjercicio, 'FM9999') || ']');
      DBMS_OUTPUT.PUT_LINE ('PARLAMENTO    = [' || TO_CHAR (p_idParlamento, 'FM0999') || '] [' || Parlamento_nombre || ']');
      DBMS_OUTPUT.PUT_LINE ('MEDIA GASTO   = [' || TO_CHAR (Parlamento_mediagasto, 'FM99G999G990D00U') || ']');
      DBMS_OUTPUT.PUT LINE ('=============================================================================================================================');
      DBMS_OUTPUT.PUT LINE ('PARLAMENTARIO                          PARTIDO                                TOTAL GASTOS            DESVIACION  PORCENTAJE');
      DBMS_OUTPUT.PUT_LINE ('-----------------------------------------------------------------------------------------------------------------------------');
      OPEN C_PARLAMENTARIOS;
      LOOP
        FETCH C_PARLAMENTARIOS
        INTO Parlamentario id,
        Parlamentario totalGastos,
        Parlamentario_nombre,
        Parlamentario_apellidos,
        Partido id,
        Partido nombre,
        Parlamentario desviacion,
        Parlamentario porcentaje;
        EXIT
        WHEN C_PARLAMENTARIOS % NOTFOUND;
        -- Imprime las líneas.
        DBMS OUTPUT.PUT LINE (TO CHAR (Parlamentario id, 'FM0999999') || ' ' || RPAD (Parlamentario nombre || ' ' || Parlamentario apellidos, 30, ' ') || ' '
|| TO CHAR (Partido id, 'FM0999999') || ' ' || RPAD (Partido nombre, 20, ' ') || ' ' || TO CHAR (Parlamentario totalGastos, '99G999G990D00U') || ' ' ||
TO_CHAR (Parlamentario_desviacion, '99G999G990D00U') || ' ' || TO_CHAR (Parlamentario_porcentaje, '99G990D00') || '%');
      END LOOP;
      CLOSE C PARLAMENTARIOS;
      DBMS_OUTPUT.PUT LINE ('=============================================================================================================================');
      SELECT
        SUM (ParlamentarioST.totalGastos),
        SUM ((ParlamentarioST.totalGastos - ParlamentoST.mediaGastoParlamentario))
      INTO
        totalGastos,
        desviacion
      FROM
        ParlamentarioST,
        ParlamentoST
      WHERE
```

```
        (
        ParlamentarioST.idEjercicio = p_idejercicio
          AND
          ParlamentarioST.idParlamento = p idParlamento
          )
        AND
        (
        ParlamentoST.idEjercicio = p_idejercicio
          AND
          ParlamentoST.idParlamento = p idParlamento
          )
        AND
        (
        ParlamentarioST.totalGastos > ParlamentoST.mediaGastoParlamentario
        );
      -- Imprime totales.
      DBMS_OUTPUT.PUT_LINE ('                                                                    ' || TO_CHAR (totalGastos, '99G999G990D00U') || ' ' ||
TO_CHAR (desviacion, '99G999G990D00U'));
      DBMS_OUTPUT.NEW_LINE;
    EXCEPTION
      WHEN pkg comun.ErrorEjercicioNoExiste THEN
        p rsp := 'ERROR: El ejercicio [' || p idEjercicio || '] no existe.';
      WHEN pkg_comun.ErrorParlamentoNoExiste THEN
        p_rsp := 'ERROR: El parlamento [' || p_idParlamento || '] no existe.';
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p_idEjercicio || '] + idParlamento = [' || p_idParlamento || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'rsp = ' || p rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg comun.entrada registro ('pkg estadistica.parlamentarios gasto superior', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END parlamentarios gasto superior;
```

### 5.5.5.6. Procedimiento pkg_listado.estado_contable_parlamentos.

Este listado corresponde al listado solicitado en el enunciado siguiente: “f. Dado un año: el listado de todos los parlamentos con su estado contable, incluyendo: Nombre del parlamento, numero de parlamentarios, suma de todos los gastos pendientes de aprobación, suma de todos los gastos aprobados, suma de todos los gastos pendientes de abono y suma de todos los gastos abonados”.

```
--
-- Dado un ejercicio, obtiene un listado del estado contable de todos los parlamentos.
--
-- idEstado = 1 ( Pendiente de aprobación. No pagado. )
-- idEstado = 2 ( Pendiente de aprobación. Pagado. Gasto sujeto a posible devolución. )
-- idEstado = 3 ( Aprobado. No pagado. )
-- idEstado = 4 ( Aprobado. Pagado. )
--
PROCEDURE estado contable parlamentos (
  p idEjercicio IN  Gasto.idEjercicio % TYPE,
  p rsp         OUT VARCHAR2)
AS
  CURSOR C_PARLAMENTOS IS
    SELECT
      ParlamentoST.idParlamento,
      Parlamento.nombre,
      ParlamentoST.numParlamentarios,
      (SELECT SUM (Gasto.baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
          FROM Gasto, TipoIVA
        WHERE Gasto.idEjercicio = p idEjercicio
          AND Gasto.idParlamento = ParlamentoST.idParlamento
          AND TipoIVA.idTipoIVA = Gasto.idTipoIVA
          AND Gasto.idEstado = 1),
      (SELECT SUM (Gasto.baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
          FROM Gasto, TipoIVA
        WHERE Gasto.idEjercicio = p idEjercicio
          AND Gasto.idParlamento = ParlamentoST.idParlamento
          AND TipoIVA.idTipoIVA = Gasto.idTipoIVA
          AND Gasto.idEstado = 2),
      (SELECT SUM (Gasto.baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
          FROM Gasto, TipoIVA
        WHERE Gasto.idEjercicio = p idEjercicio
          AND Gasto.idParlamento = ParlamentoST.idParlamento
          AND TipoIVA.idTipoIVA = Gasto.idTipoIVA
          AND Gasto.idEstado = 3),
      (SELECT SUM (Gasto.baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
          FROM Gasto, TipoIVA
        WHERE Gasto.idEjercicio = p idEjercicio
          AND Gasto.idParlamento = ParlamentoST.idParlamento
          AND TipoIVA.idTipoIVA = Gasto.idTipoIVA
          AND Gasto.idEstado = 4)
    FROM
      ParlamentoST,
      Parlamento
    WHERE
```

```
      ParlamentoST.idEjercicio = p_idEjercicio
      AND
      Parlamento.idParlamento = ParlamentoST.idParlamento;

  Parlamento id                 ParlamentoST.idParlamento % TYPE;
  Parlamento nombre             Parlamento.nombre % TYPE;
  Parlamento_numParlamentarios  ParlamentoST.numParlamentarios % TYPE;
  Parlamento_noAprobadoNoPagado ParlamentoST.totalGastos % TYPE;
  Parlamento_noAprobadoPagado   ParlamentoST.totalGastos % TYPE;
  Parlamento AprobadoNoPagado   ParlamentoST.totalGastos % TYPE;
  Parlamento aprobadoPagado     ParlamentoST.totalGastos % TYPE;
  noAprobadoNoPagado            ParlamentoST.totalGastos % TYPE;
  noAprobadoPagado              ParlamentoST.totalGastos % TYPE;
  aprobadoNoPagado              ParlamentoST.totalGastos % TYPE;
  aprobadoPagado                ParlamentoST.totalGastos % TYPE;
BEGIN
  BEGIN
    -- Inicializa el resultado de ejecución del procedimiento a OK.
    p_rsp := 'OK';
    BEGIN
      -- Controla que p idEjercicio exista en la base de datos.
      pkg comun.existe ejercicio (p idEjercicio);
      -- Imprime cabecera del listado.
      DBMS_OUTPUT.ENABLE (BUFFER_SIZE => NULL);
      DBMS_OUTPUT.NEW_LINE;
      DBMS_OUTPUT.PUT_LINE ('                          L I S T A D O   E S T A D O   C O N T A B L E   P A R L A M E N T O S
');
      DBMS OUTPUT.NEW LINE;
      DBMS_OUTPUT.PUT_LINE ('EJERCICIO     = [' || TO_CHAR (p_idEjercicio, 'FM9999') || ']');
      DBMS_OUTPUT.PUT_LINE
('=============================================================================================================================');
      DBMS OUTPUT.PUT LINE ('                                                    GASTOS PENDIENTES DE APROBACIÓN                  GASTOS APROBADOS
');
      DBMS OUTPUT.PUT LINE ('                                              ----------------------------------------  ----------------------------------------
');
      DBMS_OUTPUT.PUT_LINE ('PARLAMENTO                                        NO PAGADOS              PAGADOS            NO PAGADOS
PAGADOS');
      DBMS OUTPUT.PUT LINE ('-----------------------------------------------------------------------------------------------------------------------------------------
');
      OPEN C_PARLAMENTOS;
      LOOP
        FETCH C PARLAMENTOS
        INTO Parlamento id,
        Parlamento nombre,
        Parlamento_numParlamentarios,
        Parlamento_noAprobadoNoPagado,
        Parlamento_noAprobadoPagado,
        Parlamento AprobadoNoPagado,
        Parlamento aprobadoPagado;
        EXIT
        WHEN C_PARLAMENTOS % NOTFOUND;

        IF (Parlamento_noAprobadoNoPagado IS NULL)
```

```
        THEN
          Parlamento_noAprobadoNoPagado := 0.0;
        END IF;

        IF (Parlamento noAprobadoPagado IS NULL)
        THEN
          Parlamento_noAprobadoPagado := 0.0;
        END IF;

        IF (Parlamento AprobadoNoPagado IS NULL)
        THEN
          Parlamento AprobadoNoPagado := 0.0;
        END IF;

        IF (Parlamento_aprobadoPagado IS NULL)
        THEN
          Parlamento_aprobadoPagado := 0.0;
        END IF;
        -- Imprime las líneas.
        DBMS_OUTPUT.PUT_LINE (TO_CHAR (Parlamento_id, 'FM0999') || ' ' || RPAD (Parlamento_nombre, 30, ' ') || ' ' || TO_CHAR (Parlamento_numParlamentarios,
'990') || TO CHAR (Parlamento noAprobadoNoPagado, '99G999G990D00U') || TO CHAR (Parlamento noAprobadoPagado, '99G999G990D00U') || TO CHAR
(Parlamento AprobadoNoPagado, '99G999G990D00U') || TO CHAR (Parlamento aprobadoPagado, '99G999G990D00U'));
      END LOOP;
      CLOSE C_PARLAMENTOS;
      DBMS_OUTPUT.PUT_LINE
('================================================================================================================================================');
      SELECT
        SUM (
        (SELECT SUM (Gasto.baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
            FROM Gasto, TipoIVA
          WHERE Gasto.idEjercicio = p idEjercicio
            AND Gasto.idParlamento = ParlamentoST.idParlamento
            AND TipoIVA.idTipoIVA = Gasto.idTipoIVA
            AND Gasto.idEstado = 1)),
        SUM (
        (SELECT SUM (Gasto.baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
            FROM Gasto, TipoIVA
          WHERE Gasto.idEjercicio = p idEjercicio
            AND Gasto.idParlamento = ParlamentoST.idParlamento
            AND TipoIVA.idTipoIVA = Gasto.idTipoIVA
            AND Gasto.idEstado = 2)),
        SUM (
        (SELECT SUM (Gasto.baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
            FROM Gasto, TipoIVA
          WHERE Gasto.idEjercicio = p_idEjercicio
            AND Gasto.idParlamento = ParlamentoST.idParlamento
            AND TipoIVA.idTipoIVA = Gasto.idTipoIVA
            AND Gasto.idEstado = 3)),
        SUM (
        (SELECT SUM (Gasto.baseImponible * (1 + TipoIVA.PorcentajeIVA / 100))
            FROM Gasto, TipoIVA
          WHERE Gasto.idEjercicio = p_idEjercicio
            AND Gasto.idParlamento = ParlamentoST.idParlamento
```

```
            AND TipoIVA.idTipoIVA = Gasto.idTipoIVA
            AND Gasto.idEstado = 4))
      INTO
        noAprobadoNoPagado,
        noAprobadoPagado,
        aprobadoNoPagado,
        aprobadoPagado
      FROM
        ParlamentoST
      WHERE
        ParlamentoST.idEjercicio = p idEjercicio;

      IF (noAprobadoNoPagado IS NULL)
      THEN
        noAprobadoNoPagado := 0.0;
      END IF;

      IF (noAprobadoPagado IS NULL)
      THEN
        noAprobadoPagado := 0.0;
      END IF;

      IF (aprobadoNoPagado IS NULL)
      THEN
        aprobadoNoPagado := 0.0;
      END IF;

      IF (aprobadoPagado IS NULL)
      THEN
        aprobadoPagado := 0.0;
      END IF;
      -- Imprime totales.
      DBMS OUTPUT.PUT LINE ('                                    ' || TO CHAR (noAprobadoNoPagado, '99G999G990D00U') || TO CHAR (noAprobadoPagado,
'99G999G990D00U') || TO CHAR (aprobadoNoPagado, '99G999G990D00U') || TO CHAR (aprobadoPagado, '99G999G990D00U'));
      DBMS_OUTPUT.NEW_LINE;
    EXCEPTION
      WHEN pkg comun.ErrorEjercicioNoExiste THEN
        p rsp := 'ERROR: El ejercicio [' || p idEjercicio || '] no existe.';
      WHEN OTHERS THEN
        p_rsp := 'ERROR: ' || SQLERRM;
    END;
    -- Construye el string con los parámetros de entrada y sus valores.
    entrada := 'idEjercicio = [' || p idEjercicio || ']';
    -- Construye el string con los parámetros de salida y sus valores.
    salida := 'rsp = ' || p_rsp;
    -- Registra la ejecución del procedimiento junto con sus parámetros de entrada y salida.
    pkg_comun.entrada_registro ('pkg_estadistica.estado_contable_parlamentos', USER, SYSTIMESTAMP, entrada, salida);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS OUTPUT.PUT LINE ('ERROR INSERTANDO ENTRADA EN REGISTRO: ' || SQLERRM);
  END;
END estado_contable_parlamentos;
```

### 5.5.6. Paquete pkg_comun.

El *script* 3_1_CREAR_PKG_COMUN.SQL crea en la base de datos el paquete pkg_comun de procedimientos comunes de utilidad utilizados por el resto de paquetes.

```
-- PKG_COMUN specification
CREATE OR REPLACE PACKAGE TFC.pkg comun AS

ErrorParlamentoNoExiste EXCEPTION;
ErrorParlamentarioNoExiste EXCEPTION;
ErrorPartidoNoExiste EXCEPTION;
ErrorParlamentarioParlamento EXCEPTION;
ErrorParlamentarioNoParlamento EXCEPTION;
ErrorParlamentarioPartido EXCEPTION;
ErrorParlamentarioNoPartido EXCEPTION;
ErrorEjercicioNoExiste EXCEPTION;
ErrorCategoriaGastoNoExiste EXCEPTION;
ErrorTipoIVANoExiste EXCEPTION;
ErrorProveedorNoExiste EXCEPTION;
ErrorEstadoNoExiste EXCEPTION;
ErrorFormaPagoNoExiste EXCEPTION;
ErrorGastoNoExiste EXCEPTION;
ErrorEstadisticaNoExiste EXCEPTION;

/**
Registra la ejecución de un procedimiento, sus parámetros de entrada y sus parámetros de salida.

Inserta una fila (entrada) en la tabla Registro.

En cada entrada de la tabla Registro quedan registrados:
- El nombre del procedimiento ejecutado.
- El nombre del usuario que invocó la ejecución del procedimiento.
- El instante de tiempo en que se produjo la ejecución del procedimiento.
- La lista de parámetros de entrada y sus valores.
- La lista de parámetros de salida y sus valores.

@param p_procedimiento                  Nombre del procedimiento ejecutado.
@param p_usuario                        Nombre del usuario que invocó el procedimiento.
@param p instante                       Instante de tiempo en que fue ejecutado el procedimiento.
@param p entrada                        Parámetros de entrada y sus valores.
@param p salida                         Parámetros de salida y sus valores.
*/
PROCEDURE entrada_registro (
  p_procedimiento IN Registro.PROCEDIMIENTO % TYPE,
  p usuario       IN Registro.USUARIO % TYPE,
  p instante      IN Registro.INSTANTE % TYPE,
  p entrada       IN Registro.ENTRADA % TYPE,
  p_salida        IN Registro.SALIDA % TYPE);

/**
Dado un identificador de Parlamento, comprueba si ese Parlamento existe en la base de datos.

@param p_idParlamento                   Identificador del Parlamento a buscar en la base de datos.
```

```
@throws ErrorParlamentoNoExiste         Si el Parlamento buscado no existe en la base de datos.
*/
PROCEDURE existe parlamento (
  p idParlamento IN Parlamento.idParlamento % TYPE);

/**
Dado un identificador de Parlamentario, comprueba si ese Parlamentario existe en la base de datos.

@param p idParlamentario                Identificador del Parlamentario a buscar en la base de datos.

@throws ErrorParlamentarioNoExiste      Si el Parlamentario buscado no existe en la base de datos.
*/
PROCEDURE existe_parlamentario (
  p_idParlamentario IN Parlamentario.idParlamentario % TYPE);

/**
Dado un identificador de partido, comprueba si ese partido existe en la base de datos.

@param p_idPartido                      Identificador del partido a buscar en la base de datos.

@throws ErrorParlamentarioNoExiste      Si el Parlamentario buscado no existe en la base de datos.
*/
PROCEDURE existe_partido (
  p_idPartido IN Partido.idPartido % TYPE);

/**
Dado un identificador de Parlamentario comprueba que ese Parlamentario no esté activo ya en algún Parlamento.

La comprobación a realizar consiste en buscar la existencia de alguna fila en la tabla ParlamentoDH que
corresponda a ese Parlamentario y cuya fecha de finalización sea NULL.

La existencia de esa fila provocará el lanzamiento de la excepción ErrorParlamentarioParlamento.

@param p_idParlamentario                Identificador del Parlamentario que debe comprobarse.

@throws ErrorParlamentarioParlamento    Si el Parlamentario ya está activo en algún Parlamento.
*/
PROCEDURE parlamentario no en parlamento (
  p_idParlamentario IN Parlamentario.idParlamentario % TYPE);

/**
Dado un identificador de Parlamentario, comprueba que ese Parlamentario esté activo en algún Parlamento.

La comprobación a realizar consiste en buscar la existencia de alguna fila en la tabla ParlamentoDH que
corresponda a ese Parlamentario y cuya fecha de finalización sea NULL.

La no existencia de esa fila provocará el lanzamiento de la excepción ErrorParlamentarioNoParlamento.

@param p idParlamentario                Identificador del Parlamentario que debe comprobarse.

@throws ErrorParlamentarioNoParlamento  Si el Parlamentario no está activo en ningún Parlamento.
*/
```

```
PROCEDURE parlamentario_en_parlamento (
  p_idParlamentario IN Parlamentario.idParlamentario % TYPE);

/**
Dados un identificador de Parlamento, un identificador de Parlamentario y una fecha, comprueba si ese Parlamentario
estaba activo en ese Parlamento en esa fecha.

La comprobación a realizar consiste en buscar la existencia de alguna fila en la tabla ParlamentoDH que
corresponda a ese Parlamento y a ese Parlamentario y cuya fecha de inicio sea anterior o igual a la fecha
buscada, y a la vez la fecha buscada sea anterior o igual a la fecha de finalización o esta última sea NULL.

La no existencia de esa fila provocará el lanzamiento de la excepción ErrorParlamentarioNoParlamento.

@param p_idParlamento                   Identificador del Parlamento que debe comprobarse.
@param p_idParlamentario                Identificador del Parlamentario que debe comprobarse.
@param p fecha                          Fecha que debe comprobarse.

@throws ErrorParlamentarioNoParlamento Si el Parlamentario no estaba activo en ese Parlamento en esa fecha.
*/
PROCEDURE parlamentario_en_parlamento (
  p idParlamento    IN Parlamento.idParlamento % TYPE,
  p idParlamentario IN Parlamentario.idParlamentario % TYPE,
  p_fecha           IN ParlamentoDH.desde % TYPE);

/**
Dado un identificador de Parlamentario, comprueba que ese Parlamentario no esté activo ya en algún partido.

La comprobación a realizar consiste en buscar la existencia de alguna fila en la tabla PartidoDH que
corresponda a ese Parlamentario y cuya fecha de finalización sea NULL.

La existencia de esa fila provocará el lanzamiento de la excepción ErrorParlamentarioPartido.

@param p idParlamentario                Identificador del Parlamentario que debe comprobarse.

@throws ErrorParlamentarioPartido       Si el Parlamentario ya está activo en algún partido.
*/
PROCEDURE parlamentario no en partido (
  p idParlamentario IN Parlamentario.idParlamentario % TYPE);

/**
Dado un identificador de Parlamentario, comprueba que ese Parlamentario esté activo en algún partido.

La comprobación a realizar consiste en buscar la existencia de alguna fila en la tabla PartidoDH que
corresponda a ese Parlamentario y cuya fecha de finalización sea NULL.

La no existencia de esa fila provocará el lanzamiento de la excepción ErrorParlamentarioNoPartido.

@param p idParlamentario                Identificador del Parlamentario que debe comprobarse.

@throws ErrorParlamentarioNoPartido     Si el Parlamentario no está activo en ningún partido.
*/
PROCEDURE parlamentario_en_partido (
  p_idParlamentario IN Parlamentario.idParlamentario % TYPE);
```

```
/**
Dado un identificador de Parlamentario y una fecha, comprueba si ese Parlamentario estaba activo en algún
partido en esa fecha.

La comprobación a realizar consiste en buscar la existencia de alguna fila en la tabla PartidoDH que
corresponda a ese Parlamentario y cuya fecha de inicio sea anterior o igual a la fecha buscada, y a la vez la fecha
buscada sea anterior o igual a la fecha de finalización o esta última sea NULL.

La no existencia de esa fila provocará el lanzamiento de la excepción ErrorParlamentarioNoParlamento.

@param p idParlamentario                Identificador del Parlamentario que debe comprobarse.
@param p_fecha                          Fecha que debe comprobarse.

@throws ErrorParlamentarioNoPartido     Si el Parlamentario no está activo en ningún partido en esa fecha.
*/
PROCEDURE parlamentario_en_partido (
  p_idParlamentario IN Parlamentario.idParlamentario % TYPE,
  p_fecha           IN PartidoDH.desde % TYPE);

/**
Dada una fecha, comprueba que exista un ejercicio en la base de datos al que pertenezca esa fecha.

La comprobación a realizar consiste en buscar la existencia de alguna fila en la tabla Ejercicio cuya fecha de
inicio sea anterior o igual a la fecha buscada y cuya fecha de finalización sea igual o posterior a la fecha
buscada.

La no existencia de esa fila provocará el lanzamiento de la excepción ErrorEjercicioNoExiste.

@param p_fecha                          Fecha cuyo ejercicio hay que buscar en la base de datos.

@throws ErrorEjercicioNoExiste          Si no existe el ejercicio al que pertenece esa fecha en la base de datos.
*/
PROCEDURE existe ejercicio (
  p_fecha IN Ejercicio.inicio % TYPE);

/**
Dado un identificador de ejercicio, comprueba que ese ejercicio exista en la base de datos.

@param p_idEjercicio                    Identificador del ejercicio a buscar en la base de datos.

@throws ErrorEjercicioNoExiste          Si el ejercicio buscado no existe en la base de datos.
*/
PROCEDURE existe ejercicio (
  p_idEjercicio IN Ejercicio.idEjercicio % TYPE);

/**
Dado un identificador de categoría de gasto, comprueba que esa categoría de gasto exista en la base de datos.

@param p idCategoriaGasto               Identificador de la categoría de gasto a buscar en la base de datos.

@throws ErrorCategoriaGastoNoExiste     Si la categoría de gasto buscada no existe en la base de datos.
*/
```

```
PROCEDURE existe_categoriaGasto (
  p_idCategoriaGasto IN CategoriaGasto.idCategoriaGasto % TYPE);

/**
Dado un identificador de tipo de IVA, comprueba que ese tipo de IVA exista en la base de datos.

@param p_idTipoIVA                      Identificador del tipo de IVA a buscar en la base de datos.

@throws ErrorTipoIVAnoExiste            Si el tipo de IVA buscado no existe en la base de datos.
*/
PROCEDURE existe tipoIVA (
  p idTipoIVA IN TipoIVA.idTipoIVA % TYPE);

/**
Dado un identificador de proveedor, comprueba que ese proveedor exista en la base de datos.

@param p_idProveedor                    Identificador del proveedor a buscar en la base de datos.

@throws ErrorProveedorNoExiste          Si el proveedor buscado no existe en la base de datos.
*/
PROCEDURE existe proveedor (
  p idProveedor IN Proveedor.idProveedor % TYPE);

/**
Dado un identificador de estado (de un gasto), comprueba que ese estado exista en la base de datos.

@param p idEstado                       Identificador del estado a buscar en la base de datos.

@throws ErrorEstadoNoExiste             Si el estado buscado no existe en la base de datos.
*/
PROCEDURE existe estado (
  p idEstado IN Estado.idEstado % TYPE);

/**
Dado un identificador de forma de pago, comprueba que esa forma de pago exista en la base de datos.

@param p idFormaPago                    Identificador de la forma de pago a buscar en la base de datos.

@throws ErrorFormaPagoNoExiste          Si la forma de pago buscada no existe en la base de datos.
*/
PROCEDURE existe_formaPago (
  p idFormaPago IN FormaPago.idFormaPago % TYPE);

/**
Dado un identificador de gasto, comprueba que ese gasto exista en la base de datos.

@param p_idGasto                        Identificador del gasto a buscar en la base de datos.

@throws ErrorGastoNoExiste              Si el gasto buscado no existe en la base de datos.
*/
PROCEDURE existe_gasto (
  p_idGasto IN Gasto.idGasto % TYPE);
```

```
/**
Dado un identificador de ejercicio y un identificador de Parlamento, comprueba si la estadística del Parlamento
para ese ejercicio exista en la base de datos.

@param p idEstado                       Identificador del ejercicio que debe comprobarse.
@param p idParlamento                   Identificador del Parlamento que debe comprobarse.

@throws ErrorEstadisticaNoExiste        Si la estadística de ese Parlamento en ese ejercicio no existe en la base de datos.
*/
PROCEDURE existe parlamentoST (
  p idEjercicio  IN ParlamentoST.idEjercicio % TYPE,
  p idParlamento IN ParlamentoST.idParlamento % TYPE);

/**
Dado un identificador de ejercicio, comprueba si existen estadísticas de Parlamentos para ese Ejercicio.

@param idEjercicio                      Identificador del ejercicio que debe comprobarse.

@throws ErrorEstadisticaNoExiste        Si no existen estadísticas de Parlamento para ese Ejercicio.
*/
PROCEDURE existe parlamentoST (
  p idEjercicio IN ParlamentoST.idEjercicio % TYPE);

/**
Dado un identificador de ejercicio, un identificador de Parlamento y un identificador de Parlamentario, comprueba
si existe la estadística de ese Parlamentario para ese ejercicio y ese Parlamento.

@param idEjercicio                      Identificador del ejercicio que debe comprobarse.
@param idParlamento                     Identificador del Parlamento que debe comprobarse.
@param idParlamentario                  Identificador del Parlamentario que debe comprobarse.

@throws ErrorEstadisticaNoExiste        Si no existen estadísticas para ese Parlamentario en ese ejercicio en ese Parlamento.
*/
PROCEDURE existe parlamentarioST (
  p_idEjercicio     IN ParlamentarioST.idEjercicio % TYPE,
  p_idParlamento    IN ParlamentarioST.idParlamento % TYPE,
  p idParlamentario IN ParlamentarioST.idParlamentario % TYPE);

/**
Dados un identificador de ejercicio y un identificador de Parlamentario, comprueba si existen estadísticas de
Parlamentario para ese ejercicio en algún Parlamento.

@param idEjercicio                      Identificador del ejercicio que debe comprobarse.
@param idParlamentario                  Identificador del Parlamentario que debe comprobarse.

@throws ErrorEstadisticaNoExiste        Si no existen estadísticas para ese Parlamentario en ese Ejercicio.
*/
PROCEDURE existe parlamentarioST (
  p idEjercicio     IN ParlamentarioST.idEjercicio % TYPE,
  p idParlamentario IN ParlamentarioST.idParlamentario % TYPE);

/**
Dado un identificador de ejercicio, comprueba si existen estadísticas de Parlamentario para ese Ejercicio.
```

```
@param idEjercicio                      Identificador del ejercicio que debe comprobarse.

@throws ErrorEstadisticaNoExiste        Si no existen estadísticas de Parlamentario para ese Ejercicio.
*/
PROCEDURE existe parlamentarioST (
  p_idEjercicio IN ParlamentarioST.idEjercicio % TYPE);

END pkg_comun;
```

### 5.5.6.1. Procedimientos del paquete pkg_comun.

```
--
-- Registra la ejecución de un procedimiento, sus parámetros de entrada y sus parámetros de salida.
--
PROCEDURE entrada_registro (
  p_procedimiento IN Registro.PROCEDIMIENTO % TYPE,
  p_usuario       IN Registro.USUARIO % TYPE,
  p instante      IN Registro.INSTANTE % TYPE,
  p entrada       IN Registro.ENTRADA % TYPE,
  p salida        IN Registro.SALIDA % TYPE)
AS
BEGIN
  INSERT INTO REGISTRO
      (PROCEDIMIENTO,
       USUARIO,
       INSTANTE,
       ENTRADA,
       SALIDA)
    VALUES
      (p procedimiento,
       p usuario,
       p_instante,
       p_entrada,
       p_salida);
END entrada registro;

--
-- Dado un identificador de Parlamento, comprueba si ese Parlamento existe en la base de datos.
--
PROCEDURE existe_parlamento (
  p idParlamento IN Parlamento.idParlamento % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idParlamento IS NULL)
  THEN
    RAISE ErrorParlamentoNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      Parlamento
    WHERE
      idParlamento = p_idParlamento;

    IF (existe = 0)
    THEN
      RAISE ErrorParlamentoNoExiste;
    END IF;
```

```
  END IF;
END existe_parlamento;

--
-- Dado un identificador de Parlamentario, comprueba si ese Parlamentario existe en la base de datos.
--
PROCEDURE existe_parlamentario (
  p_idParlamentario IN Parlamentario.idParlamentario % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p idParlamentario IS NULL)
  THEN
    RAISE ErrorParlamentarioNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      Parlamentario
    WHERE
      idParlamentario = p_idParlamentario;

    IF (existe = 0)
    THEN
      RAISE ErrorParlamentarioNoExiste;
    END IF;
  END IF;
END existe_parlamentario;

--
-- Dado un identificador de partido, comprueba si ese partido existe en la base de datos.
--
PROCEDURE existe_partido (
  p_idPartido IN Partido.idPartido % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idPartido IS NULL)
  THEN
    RAISE ErrorPartidoNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      PARTIDO
    WHERE
      idPartido = p_idPartido;

    IF (existe = 0)
```

```
    THEN
      RAISE ErrorPartidoNoExiste;
    END IF;
  END IF;
END existe partido;

--
-- Dado un identificador de Parlamentario comprueba que ese Parlamentario no esté activo ya en algún Parlamento.
--
PROCEDURE parlamentario no en parlamento (
  p idParlamentario IN Parlamentario.idParlamentario % TYPE)
AS
  existe NATURAL;
BEGIN
  SELECT
    COUNT (*)
  INTO
    existe
  FROM
    ParlamentoDH
  WHERE
    idParlamentario = p idParlamentario
    AND
    hasta IS NULL;

  IF (existe <> 0)
  THEN
    RAISE ErrorParlamentarioParlamento;
  END IF;
END parlamentario_no_en_parlamento;

--
-- Dado un identificador de Parlamentario, comprueba que ese Parlamentario esté activo en algún Parlamento.
--
PROCEDURE parlamentario_en_parlamento (
  p_idParlamentario IN Parlamentario.idParlamentario % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idParlamentario IS NULL)
  THEN
    RAISE ErrorParlamentarioNoParlamento;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      ParlamentoDH
    WHERE
      idParlamentario = p_idParlamentario
      AND
      hasta IS NULL;
```

```
    IF (existe = 0)
    THEN
      RAISE ErrorParlamentarioNoParlamento;
    END IF;
  END IF;
END parlamentario_en_parlamento;

--
-- Dados un identificador de Parlamento, un identificador de Parlamentario y una fecha, comprueba si ese Parlamentario estaba activo en ese Parlamento en esa
fecha.
--
PROCEDURE parlamentario_en_parlamento (
  p_idParlamento    IN Parlamento.idParlamento % TYPE,
  p_idParlamentario IN Parlamentario.idParlamentario % TYPE,
  p fecha           IN ParlamentoDH.desde % TYPE)
AS
  existe NATURAL;
BEGIN
  SELECT
    COUNT (*)
  INTO
    existe
  FROM
    ParlamentoDH
  WHERE
    idParlamento = p idParlamento
    AND
    idParlamentario = p_idParlamentario
    AND
    desde <= p fecha
    AND
    (
    p fecha <= hasta
      OR
      hasta IS NULL
      );

  IF (existe = 0)
  THEN
    RAISE ErrorParlamentarioNoParlamento;
  END IF;
END parlamentario en parlamento;

--
-- Dado un identificador de Parlamentario, comprueba que ese Parlamentario no esté activo ya en algún partido.
--
PROCEDURE parlamentario no en partido (
  p idParlamentario IN Parlamentario.idParlamentario % TYPE)
AS
  existe NATURAL;
BEGIN
  SELECT
```

```
    COUNT (*)
  INTO
    existe
  FROM
    PartidoDH
  WHERE
    idParlamentario = p_idParlamentario
    AND
    hasta IS NULL;

  IF (existe <> 0)
  THEN
    RAISE ErrorParlamentarioPartido;
  END IF;
END parlamentario_no_en_partido;

--
-- Dado un identificador de Parlamentario, comprueba que ese Parlamentario esté activo en algún partido.
--
PROCEDURE parlamentario_en_partido (
  p idParlamentario IN Parlamentario.idParlamentario % TYPE)
AS
  existe NATURAL;
BEGIN
  SELECT
    COUNT (*)
  INTO
    existe
  FROM
    PartidoDH
  WHERE
    idParlamentario = p idParlamentario
    AND
    hasta IS NULL;

  IF (existe = 0)
  THEN
    RAISE ErrorParlamentarioNoPartido;
  END IF;
END parlamentario_en_partido;

--
-- Dado un identificador de Parlamentario y una fecha, comprueba si ese Parlamentario estaba activo en algún partido en esa fecha.
--
PROCEDURE parlamentario_en_partido (
  p_idParlamentario IN Parlamentario.idParlamentario % TYPE,
  p_fecha           IN PartidoDH.desde % TYPE)
AS
  existe NATURAL;
BEGIN
  SELECT
    COUNT (*)
  INTO
```

```
    existe
  FROM
    PartidoDH
  WHERE
    idParlamentario = p idParlamentario
    AND
    desde <= p_fecha
    AND
    (
    p fecha <= hasta
      OR
      hasta IS NULL
      );

  IF (existe = 0)
  THEN
    RAISE ErrorParlamentarioNoPartido;
  END IF;
END parlamentario_en_partido;

--
-- Dada una fecha, comprueba que exista un ejercicio en la base de datos al que pertenezca esa fecha.
--
PROCEDURE existe_ejercicio (
  p_fecha IN Ejercicio.inicio % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_fecha IS NULL)
  THEN
    RAISE ErrorEjercicioNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      Ejercicio
    WHERE
      inicio <= p_fecha
      AND
      p fecha <= fin;

    IF (existe = 0)
    THEN
      RAISE ErrorEjercicioNoExiste;
    END IF;
  END IF;
END existe ejercicio;

--
-- Dado un identificador de ejercicio, comprueba que ese ejercicio exista en la base de datos.
--
```

```
PROCEDURE existe_ejercicio (
  p_idEjercicio IN Ejercicio.idEjercicio % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p idEjercicio IS NULL)
  THEN
    RAISE ErrorEjercicioNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      Ejercicio
    WHERE
      idEjercicio = p_idEjercicio;

    IF (existe = 0)
    THEN
      RAISE ErrorEjercicioNoExiste;
    END IF;

  END IF;
END existe_ejercicio;

--
-- Dado un identificador de categoría de gasto, comprueba que esa categoría de gasto exista en la base de datos.
--
PROCEDURE existe_categoriaGasto (
  p idCategoriaGasto IN CategoriaGasto.idCategoriaGasto % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idCategoriaGasto IS NULL)
  THEN
    RAISE ErrorCategoriaGastoNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      CategoriaGasto
    WHERE
      idCategoriaGasto = p_idCategoriaGasto;

    IF (existe = 0)
    THEN
      RAISE ErrorCategoriaGastoNoExiste;
    END IF;
  END IF;
END existe_categoriaGasto;
```

```
--
-- Dado un identificador de tipo de IVA, comprueba que ese tipo de IVA exista en la base de datos.
--
PROCEDURE existe tipoIVA (
  p idTipoIVA IN TipoIVA.idTipoIVA % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p idTipoIVA IS NULL)
  THEN
    RAISE ErrorTipoIVANoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      TipoIVA
    WHERE
      idTipoIVA = p idTipoIVA;

    IF (existe = 0)
    THEN
      RAISE ErrorTipoIVANoExiste;
    END IF;
  END IF;
END existe tipoIVA;

--
-- Dado un identificador de proveedor, comprueba que ese proveedor exista en la base de datos.
--
PROCEDURE existe proveedor (
  p idProveedor IN Proveedor.idProveedor % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p idProveedor IS NULL)
  THEN
    RAISE ErrorProveedorNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      Proveedor
    WHERE
      idProveedor = p idProveedor;

    IF (existe = 0)
    THEN
      RAISE ErrorProveedorNoExiste;
```

```
    END IF;
  END IF;
END existe_proveedor;

--
-- Dado un identificador de estado (de un gasto), comprueba que ese estado exista en la base de datos.
--
PROCEDURE existe_estado (
  p_idEstado IN Estado.idEstado % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idEstado IS NULL)
  THEN
    RAISE ErrorEstadoNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      Estado
    WHERE
      idEstado = p_idEstado;

    IF (existe = 0)
    THEN
      RAISE ErrorEstadoNoExiste;
    END IF;
  END IF;
END existe estado;

--
-- Dado un identificador de forma de pago, comprueba que esa forma de pago exista en la base de datos.
--
PROCEDURE existe_formaPago (
  p idFormaPago IN FormaPago.idFormaPago % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idFormaPago IS NULL)
  THEN
    RAISE ErrorFormaPagoNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      FormaPago
    WHERE
      idFormaPago = p_idFormaPago;
```

```
    IF (existe = 0)
    THEN
      RAISE ErrorFormaPagoNoExiste;
    END IF;
  END IF;
END existe formaPago;

--
-- Dado un identificador de gasto, comprueba que ese gasto exista en la base de datos.
--
PROCEDURE existe gasto (
  p idGasto IN Gasto.idGasto % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p idGasto IS NULL)
  THEN
    RAISE ErrorGastoNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      Gasto
    WHERE
      idGasto = p idGasto;

    IF (existe = 0)
    THEN
      RAISE ErrorGastoNoExiste;
    END IF;
  END IF;
END existe gasto;

--
-- Dado un identificador de ejercicio y un identificador de Parlamento, comprueba si la estadística del Parlamento para ese ejercicio exista en la base de
datos.
--
PROCEDURE existe_parlamentoST (
  p_idEjercicio  IN ParlamentoST.idEjercicio % TYPE,
  p idParlamento IN ParlamentoST.idParlamento % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idEjercicio IS NULL
      OR
      p idParlamento IS NULL)
  THEN
    RAISE ErrorEstadisticaNoExiste;
  ELSE
    SELECT
      COUNT (*)
```

```
    INTO
      existe
    FROM
      ParlamentoST
    WHERE
      idEjercicio = p idEjercicio
      AND
      idParlamento = p_idParlamento;

    IF (existe = 0)
    THEN
      RAISE ErrorEstadisticaNoExiste;
    END IF;
  END IF;
END existe_parlamentoST;

--
-- Dado un identificador de ejercicio, comprueba si existen estadísticas de Parlamentos para ese Ejercicio.
--
PROCEDURE existe_parlamentoST (
  p idEjercicio IN ParlamentoST.idEjercicio % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idEjercicio IS NULL)
  THEN
    RAISE ErrorEstadisticaNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      ParlamentoST
    WHERE
      idEjercicio = p_idEjercicio;

    IF (existe = 0)
    THEN
      RAISE ErrorEstadisticaNoExiste;
    END IF;
  END IF;
END existe parlamentoST;

--
-- Dado un identificador de ejercicio, un identificador de Parlamento y un identificador de Parlamentario, comprueba
-- si existe la estadística de ese Parlamentario para ese ejercicio y ese Parlamento.
--
PROCEDURE existe parlamentarioST (
  p idEjercicio     IN ParlamentarioST.idEjercicio % TYPE,
  p_idParlamento    IN ParlamentarioST.idParlamento % TYPE,
  p_idParlamentario IN ParlamentarioST.idParlamentario % TYPE)
AS
```

```
  existe NATURAL;
BEGIN
  IF (p_idEjercicio IS NULL
      OR
      p idParlamento IS NULL
      OR
      p_idParlamentario IS NULL)
  THEN
    RAISE ErrorEstadisticaNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      ParlamentarioST
    WHERE
      idEjercicio = p_idEjercicio
      AND
      idParlamento = p_idParlamento
      AND
      idParlamentario = p idParlamentario;

    IF (existe = 0)
    THEN
      RAISE ErrorEstadisticaNoExiste;
    END IF;
  END IF;
END existe_parlamentarioST;

--
-- Dados un identificador de ejercicio y un identificador de Parlamentario, comprueba si existen estadísticas de
-- Parlamentario para ese ejercicio en algún Parlamento.
--
PROCEDURE existe_parlamentarioST (
  p_idEjercicio     IN ParlamentarioST.idEjercicio % TYPE,
  p idParlamentario IN ParlamentarioST.idParlamentario % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idEjercicio IS NULL
      OR
      p idParlamentario IS NULL)
  THEN
    RAISE ErrorEstadisticaNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      ParlamentarioST
    WHERE
```

```
      idEjercicio = p_idEjercicio
      AND
      idParlamentario = p_idParlamentario;

    IF (existe = 0)
    THEN
      RAISE ErrorEstadisticaNoExiste;
    END IF;
  END IF;
END existe parlamentarioST;

--
-- Dado un identificador de ejercicio, comprueba si existen estadísticas de Parlamentario para ese Ejercicio.
--
PROCEDURE existe_parlamentarioST (
  p idEjercicio IN ParlamentarioST.idEjercicio % TYPE)
AS
  existe NATURAL;
BEGIN
  IF (p_idEjercicio IS NULL)
  THEN
    RAISE ErrorEstadisticaNoExiste;
  ELSE
    SELECT
      COUNT (*)
    INTO
      existe
    FROM
      ParlamentarioST
    WHERE
      idEjercicio = p idEjercicio;

    IF (existe = 0)
    THEN
      RAISE ErrorEstadisticaNoExiste;
    END IF;
  END IF;
END existe parlamentarioST;
```

## 6. Productos obtenidos.

Como resultado del trabajo realizado se han producido una serie de documentos:

a) Una memoria del proyecto en la que se detalla todo el trabajo realizado.

   Se trata de un documento cuya extensión prevista era de un máximo de 60 páginas y que debía incluir el diagrama Entidad-Relación, las definiciones de tablas, procedimientos almacenados, *scripts* de carga de la base de datos, *scripts* de juegos de pruebas y resultados obtenidos en las pruebas realizadas.

   La extensión del documento final es muy superior a la prevista. Sin embargo ello responde a la extensión del código de los procedimientos almacenados que se ha formateado y documentado internamente de manera que se facilite al máximo su lectura y comprensión. Podría haberse optado por un formato mucho más compacto que hubiera permitido respetar la extensión máxima prevista de la memoria pero se ha valorado más mejorar la legibilidad del código

b) Un anexo que muestra los *scripts* de juegos de pruebas desarrollados así como los resultados obtenidos en las pruebas realizadas.

c) Una presentación en diapositivas a modo de resumen del trabajo realizado cuya extensión prevista era de un máximo de 20 diapositivas.

   La extensión de la presentación final si se ha mantenido en el tamaño previsto puesto que no es sino un resumen.

d) El producto final: Un fichero comprimido que contiene todo el código ejecutable desarrollado para la creación de la base de datos, creación de las tablas, índices, *triggers*, procedimientos almacenados, *scripts* de carga de datos en la base de datos y *scripts* de juegos de pruebas.

## 7. Conclusiones.

Como conclusiones a la realización de este trabajo final de carrera pueden destacarse aspectos positivos y aspectos negativos.

Como aspectos positivos cabe destacar que se han cumplido los objetivos del mismo.

- Se ha desarrollado una base de datos que mejora los requerimientos solicitados en el enunciado.
- Se han puesto en práctica los conocimientos adquiridos a lo largo de la carrera en asignaturas como programación estructurada, programación orientada a objetos, estructuras de la información y, en especial, en las asignaturas de bases de datos.
- Se han adquirido nuevos conocimientos y utilizado herramientas que no había usado hasta ahora, acumulando experiencia y conocimientos que son aplicables en el mundo laboral.

Como aspectos negativos cabe mencionar que no pudo realizarse conforme a la planificación prevista inicialmente y ha tenido que finalizarse en el semestre siguiente.

- Problemas personales provocaron un retraso en el inicio que se propagó a lo largo del las etapas del proyecto.
- La curva de aprendizaje de la peculiar implementación *SQL* de *Oracle*, así como la falta de funcionalidades e insuficiente documentación de la herramienta *CASE SQL Developer* también supusieron una serie de contratiempos que afectaron a la planificación de forma importante.

## 8. Bibliografía.


- ***Oracle® Database Express Edition Documentation, Version 11g Release 2***
  (publicación en línea) http://docs.oracle.com/cd/E17781_01/index.htm

- ***Oracle® SQL Developer Documentation, Release 4.0***
  (publicación en línea) http://docs.oracle.com/cd/E39885_01/index.htm