Tema 5

Sistemas de gestión de bases de datos relacionales. Características y componentes. Sistemas de gestión de bases de datos orientados a objetos.

Guion-resumen

1. **Sistemas de gestión de bases de datos**
   - 1.1. Definiciones
   - 1.2. Arquitectura de los sistemas de bases de datos
   - 1.3. Lenguajes de los sistemas de gestión de bases de datos
   - 1.4. Funciones de los sistemas de gestión de bases de datos
   - 1.5. Componentes de un sistema de gestión de bases de datos
2. **Sistemas de gestión de bases de datos relacionales**
   - 2.1. El Modelo Relacional
   - 2.2. Conceptos fundamentales
   - 2.3. Claves
   - 2.4. Esquema de una base de datos relacional
   - 2.5. Restricciones del modelo relacional
   - 2.6. Componentes
   - 2.7. Vistas
   - 2.8. Algebra relacional
   - 2.9. Soluciones de Sistemas Gestores de Bases de Datos Relacionales (SGBDR)
3. **Mapeo Objeto/Relacional**
   - 3.1. Características
   - 3.2. Ventajas
   - 3.3. Soluciones
4. **Sistemas de gestión de bases de datos orientados a objetos**
   - 4.1. Introducción
   - 4.2. Definiciones
   - 4.3. Características de una BDOO según el manifiesto de Malcolm Atkinson
   - 4.4. Ventajas e inconvenientes
   - 4.5. ODMG: estándar de facto para modelos de objetos
   - 4.6. Soluciones de sistemas gestores de bases de datos orientados a objetos

1. Sistemas de gestión de bases de datos

1.1. Definiciones

1.1.1. Base de datos

El concepto de base de datos ha ido cambiando y configurándose a lo largo del tiempo y, en la actualidad, podemos definir una base de datos como: "Colección o depósito de datos integrados, con redundancia controlada y con una estructura que refleje las interrelaciones y restricciones existentes en el mundo real; los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de estas y su definición y descripción, únicas para cada tipo de datos, han de estar almacenadas junto con los mismos. Los procedimientos de actualización y recuperación, comunes y bien determinados, habrán de ser capaces de conservar la integridad, seguridad y confidencialidad del conjunto de los datos."

1.1.2. Modelos de datos

Una de las características fundamentales de los sistemas de bases de datos es que proporcionan cierto nivel de abstracción de datos, al ocultar las características sobre el almacenamiento físico que la mayoría de usuarios no necesita conocer. Los modelos de datos son el instrumento principal para ofrecer dicha abstracción.

Un modelo de datos es:

1. Un conjunto de conceptos que sirven para describir la estructura de una base de datos:

   a) Los datos.

   b) Las relaciones entre los datos.

   c) Las restricciones que deben cumplirse sobre los datos.

2. Contienen también un conjunto de operaciones básicas para la realización de consultas (lecturas) y actualizaciones de datos.

3. Además, los modelos de datos más modernos incluyen conceptos para especificar comportamiento, permitiendo especificar un conjunto de operaciones definidas por el usuario.

Los modelos de datos se pueden clasificar dependiendo de los tipos de conceptos que ofrecen para describir la estructura de la base de datos:

1. Los modelos de datos de alto nivel, o **modelos conceptuales**, disponen de conceptos muy cercanos al modo en que la mayoría de los usuarios percibe los datos. Los modelos conceptuales utilizan conceptos como entidades, atributos y relaciones. Una entidad representa un objeto o concepto del mundo real como, por ejemplo, un empleado de la

empresa inmobiliaria o una oficina. Un atributo representa alguna propiedad de interés de una entidad como, por ejemplo, el nombre o el salario del empleado. Una relación describe una interacción entre dos o más entidades, por ejemplo, la relación de trabajo entre un empleado y su oficina.

2. Mientras que los modelos de datos de bajo nivel, o **modelos físicos**, proporcionan conceptos que describen los detalles de cómo se almacenan los datos en el ordenador, los conceptos de los modelos físicos están dirigidos al personal informático, no a los usuarios finales. Los modelos físicos describen cómo se almacenan los datos en el ordenador: el formato de los registros, la estructura de los ficheros (desordenados, ordenados, etc.) y los métodos de acceso utilizados (índices, etc).

3. Entre estos dos extremos se encuentran los modelos lógicos, cuyos conceptos pueden ser entendidos por los usuarios finales, aunque no están demasiado alejados de la forma en que los datos se organizan físicamente. Los modelos lógicos ocultan algunos detalles de cómo se almacenan los datos, pero pueden implementarse de manera directa en un ordenador. Cada Sistema de Gestión de Bases de Datos (SGBD) soporta un modelo lógico, siendo los más comunes el relacional, el de red y el jerárquico. Estos modelos representan los datos valiéndose de estructuras de registros, por lo que también se denominan modelos orientados a registros. Los últimos en llegar son los modelos orientados a objetos, que están más próximos a los modelos conceptuales.

A la descripción de una base de datos mediante un modelo de datos se le denomina **esquema de la base de datos**. Este esquema se especifica durante el diseño, y no es de esperar que se modifique a menudo. Sin embargo, los datos que se almacenan en la base de datos pueden cambiar con mucha frecuencia: se insertan datos, se actualizan, etc. Los datos que la base de datos contiene en un determinado momento se denominan **estado de la base de datos** u **ocurrencia de la base de datos** o también ejemplar de la base de datos.

La distinción entre el esquema y el estado de la base de datos es muy importante. Cuando definimos una nueva base de datos, solo especificamos su esquema al SGBD. En ese momento, el estado de la base de datos es el **estado vacío**, sin datos. Cuando se cargan datos por primera vez, la base datos pasa al **estado inicial**. De ahí en adelante, siempre que se realice una operación de actualización de la base de datos, se tendrá un nuevo estado. El SGBD se encarga, en parte, de garantizar que todos los estados de la base de datos sean estados válidos que satisfagan la estructura y las restricciones especificadas en el esquema. Por lo tanto, es muy importante que el esquema que se especifique al SGBD sea correcto y se debe tener muchísimo cuidado al diseñarlo. El SGBD almacena el esquema en su catálogo o diccionario de datos, de modo que se pueda consultar siempre que sea necesario.

1.2. Arquitectura de los sistemas de bases de datos

Hay tres características importantes inherentes a los sistemas de bases de datos: la separación entre los programas de aplicación y los datos, el manejo de múltiples vistas por parte de los usuarios y el uso de un catálogo para

almacenar el esquema de la base de datos. En 1975, el comité ANSI-SPARC *(American National Standard Institute - Standards Planning and Requirements Committee)* propuso una arquitectura de tres niveles para los sistemas de bases de datos, que resulta muy útil a la hora de conseguir estas tres características.

El objetivo de la arquitectura de tres niveles es el de separar los programas de aplicación de la base de datos física. En esta arquitectura, el esquema de una base de datos se define en tres niveles de abstracción distintos:

1. En el **nivel externo** se describen varios esquemas externos o vistas de usuario. Cada esquema externo describe la parte de la base de datos que interesa a un grupo de usuarios determinado, y oculta a ese grupo el resto de la base de datos. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar los esquemas.

2. En el **nivel conceptual** se describe la estructura de toda la base de datos para una comunidad de usuarios (todos los de una empresa u organización), mediante un esquema conceptual. Este esquema oculta los detalles de las estructuras de almacenamiento y se concentra en describir entidades, atributos, relaciones, operaciones de los usuarios y restricciones. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar el esquema.

3. En el **nivel interno** se describe la estructura física de la base de datos mediante un esquema interno. Este esquema se especifica mediante un modelo físico y describe todos los detalles para el almacenamiento de la base de datos, así como los métodos de acceso.

Hay que destacar que los tres esquemas no son más que descripciones de los mismos datos pero con distintos niveles de abstracción. Los únicos datos que existen realmente están a nivel físico, almacenados en un dispositivo como puede ser un disco. En un SGBD basado en la arquitectura de tres niveles, cada grupo de usuarios hace referencia exclusivamente a su propio esquema externo. El SGBD debe poder garantizar la transferencia de los datos desde el nivel interno al nivel externo; a este proceso se le llama **transformación de datos o mapeo** *(data mapping)*. Para ello existen dos niveles de correspondencia:

— **Correspondencia conceptual/interna:** permite el paso de la vista conceptual a la vista interna y viceversa.

— **Correspondencia externa/conceptual:** permite el paso de una vista externa específica a la vista conceptual, y viceversa.

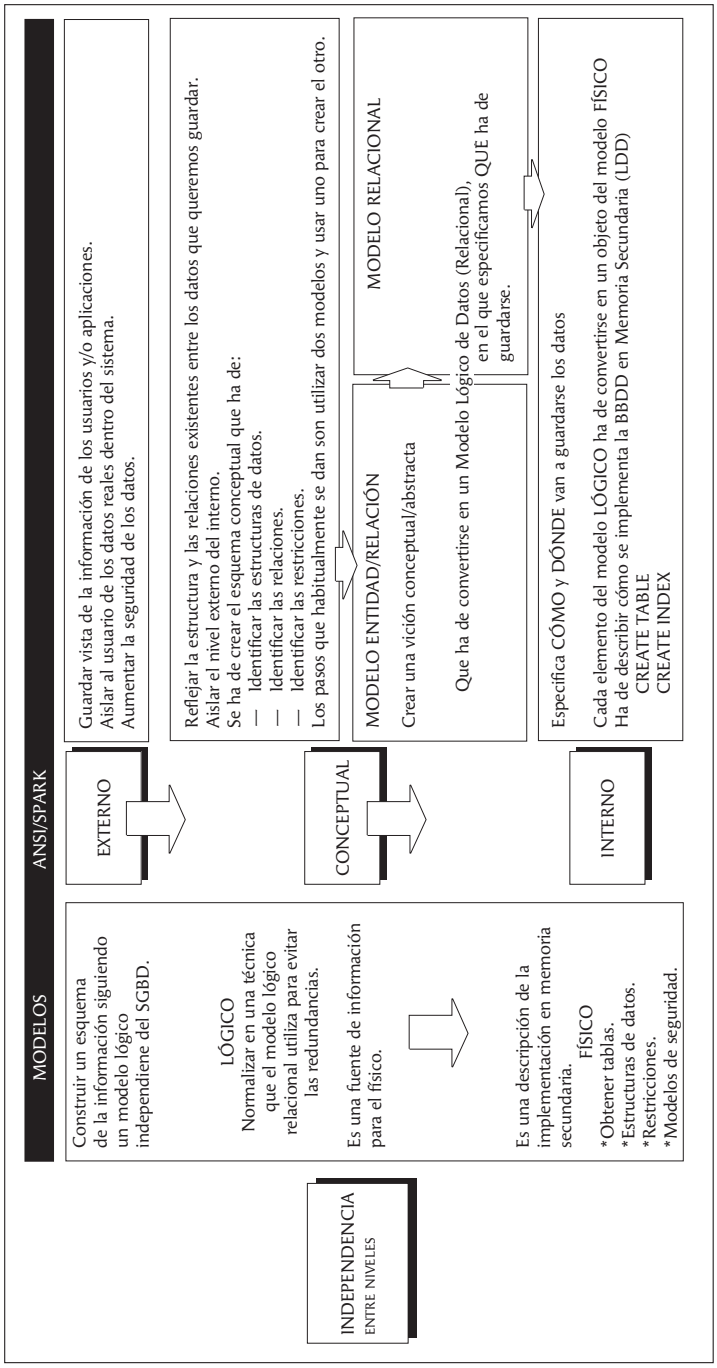
La arquitectura de tres niveles es útil para explicar el concepto de **independencia de datos** que podemos definir como la capacidad para modificar el esquema en un nivel del sistema sin tener que modificar el esquema del nivel inmediato superior. Se pueden definir dos tipos de independencia de datos:

— **La independencia lógica** es la capacidad de modificar el esquema conceptual sin tener que alterar los esquemas externos ni los pro-

gramas de aplicación. Se puede modificar el esquema conceptual para ampliar la base de datos o para reducirla. Si, por ejemplo, se reduce la base de datos eliminando una entidad, los esquemas externos que no se refieran a ella no deberán verse afectados.

— **La independencia física** es la capacidad de modificar el esquema interno sin tener que alterar el esquema conceptual (o los externos). Por ejemplo, puede ser necesario reorganizar ciertos ficheros físicos con el fin de mejorar el rendimiento de las operaciones de consulta o de actualización de datos

Por lo tanto, la arquitectura de tres niveles puede facilitar la obtención de la verdadera independencia de datos, tanto física como lógica.


MODELOS
ANSI/SPARK
EXTERNO
Guardar vista de la información de los usuarios y/o aplicaciones.
Aislar al usuario de los datos reales dentro del sistema.
Aumentar la seguridad de los datos.
CONCEPTUAL
Reflejar la estructura y las relaciones existentes entre los datos que queremos guardar.
Aislar el nivel externo del interno.
Se ha de crear el esquema conceptual que ha de:
— Identificar las estructuras de datos.
— Identificar las relaciones.
— Identificar las restricciones.
Los pasos que habitualmente se dan son utilizar dos modelos y usar uno para crear el otro.
MODELO ENTIDAD/RELACIÓN
Crear una vición conceptual/abstracta
Que ha de convertirse en un Modelo Lógico de Datos (Relacional),
en el que especificamos QUE ha de guardarse.
MODELO RELACIONAL
INTERNO
Especifica CÓMO y DÓNDE van a guardarse los datos
Cada elemento del modelo LÓGICO ha de convertirse en un objeto del modelo FÍSICO
Ha de describir cómo se implementa la BBDD en Memoria Secundaria (LDD)
CREATE TABLE
CREATE INDEX
INDEPENDENCIA
ENTRE NIVELES
Construir un esquema
de la información siguiendo
un modelo lógico
independiente del SGBD.
LÓGICO
Normalizar en una técnica
que el modelo lógico
relacional utiliza para evitar
las redundancias.
Es una fuente de información
para el físico.
Es una descripción de la
implementación en memoria
secundaria.
FÍSICO
*Obtener tablas.
*Estructuras de datos.
*Restricciones.
*Modelos de seguridad.

1.3. Lenguajes de los sistemas de gestión de bases de datos

Los SGBD deben ofrecer lenguajes e interfaces apropiadas para cada tipo de usuario: administradores de la base de datos, diseñadores, programadores de aplicaciones y usuarios finales.

1.3.1. Lenguaje de definición de datos

Los lenguajes de definición de datos, en adelante LDD (DDL en inglés), están orientados a la descripción de la base de datos dentro del SGBD, así como a las modificaciones de la base y cambios en su estructura física. Esto permite la definición de las tablas: nombres, campos y dominios, atributos, interrelaciones, autorizaciones de acceso, reglas de integridad, índices, así como todo lo necesario para asegurar la integración referencial y validaciones. Uno de los LDD más extensamente utilizados es el Abstract Syntax Notation 1 (ASN.1), definido por ISO en 1988 como ISO 8824.

1.3.2. Lenguaje de manipulación de datos

Los lenguajes de manipulación de datos, en adelante LMD (DML en inglés), están orientados al proceso y extracción de información almacenada en el SGBD. Se usan para añadir, borrar o modificar registros de las distintas entidades. Los objetos creados en la BD con un lenguaje de definición de datos se gestionan con un lenguaje de manipulación de datos. Los usuarios lo utilizan para realizar consultas, inserciones, eliminaciones y modificaciones sobre los datos de la BD.

Hay dos tipos de LMD: los procedurales y los no procedurales.

— Con un **LMD procedural** el usuario (normalmente será un programador) especifica qué datos se necesitan y cómo hay que obtenerlos. Esto quiere decir que el usuario debe especificar todas las operaciones de acceso a datos llamando a los procedimientos necesarios para obtener la información requerida. Estos lenguajes acceden a un registro, lo procesan y basándose en los resultados obtenidos, acceden a otro registro, que también deben procesar. Así se va accediendo a registros y se van procesando hasta que se obtienen los datos deseados. Las sentencias de un LMD procedural deben estar embebidas en un lenguaje de alto nivel, ya que se necesitan sus estructuras (bucles, condicionales, etc.) para obtener y procesar cada registro individual. A este lenguaje se le denomina **lenguaje anfitrión**. Las bases de datos jerárquicas y de red utilizan LMD procedurales.

— Un **LMD no procedural** se puede utilizar de manera independiente para especificar operaciones complejas sobre la base de datos de forma concisa. En muchos SGBD se pueden introducir interactivamente instrucciones del LMD desde un terminal o bien embeberlas en un lenguaje de programación de alto nivel. Los LMD no procedurales permiten especificar los datos a obtener en una consulta o los datos que se deben actualizar, mediante una sola y sencilla sentencia. El usuario o programador especifica qué datos quiere obtener sin decir cómo se debe acceder a ellos. El SGBD tra-

duce las sentencias del LMD en uno o varios procedimientos que manipulan los conjuntos de registros necesarios. Esto libera al usuario de tener que conocer cuál es la estructura física de los datos y qué algoritmos se deben utilizar para acceder a ellos. A los LMD no procedurales también se les denomina declarativos. Las bases de datos relacionales utilizan LMD no procedurales, como SQL *(Structured Query Language)* o QBE *(Query-By-Example)*. Los lenguajes no procedurales son más fáciles de aprender y de usar que los procedurales, y el usuario debe realizar menos trabajo, siendo el SGBD quien hace la mayor parte. La parte de los LMD no procedurales que realiza la obtención de datos es lo que se denomina un lenguaje de consultas. En general, las órdenes tanto de obtención como de actualización de datos de un LMD no procedural se pueden utilizar interactivamente, por lo que al conjunto completo de sentencias del LMD se le denomina lenguaje de consultas, aunque es técnicamente incorrecto.

1.3.3. Lenguaje de control de datos

Los lenguajes de control de datos, en adelante LCD (DCL en inglés), permiten conceder o suprimir privilegios a los usuarios, es decir, realizan el control del acceso a los datos. Con este lenguaje se establecen las vistas de los usuarios; de esta manera, a cada usuario se le permite manipular únicamente el conjunto de datos que le interesa y se le deniega el acceso a los datos que no necesita.

1.3.4. Lenguajes de cuarta generación

No existe consenso sobre lo que es un lenguaje de cuarta generación (4GL). Lo que en un lenguaje de tercera generación (3GL) como COBOL requiere cientos de líneas de código, tan solo necesita diez o veinte líneas en un 4GL. Comparado con un 3GL, que es procedural, un 4GL es un lenguaje no procedural: el usuario define qué se debe hacer, no cómo debe hacerse. Los 4GL se apoyan en unas herramientas de mucho más alto nivel denominadas herramientas de cuarta generación. El usuario no debe definir los pasos a seguir en un programa para realizar una determinada tarea, tan solo debe definir una serie de parámetros que estas herramientas utilizarán para generar un programa de aplicación. Se dice que los 4GL pueden mejorar la productividad de los programadores en un factor de 10, aunque se limita el tipo de problemas que pueden resolver. Los 4GL abarcan:

— Lenguajes de presentación, como lenguajes de consultas y generadores de informes.

— Lenguajes especializados, como hojas de cálculo y lenguajes de bases de datos.

— Generadores de aplicaciones que definen, insertan, actualizan y obtienen datos de la base de datos.

— Lenguajes de muy alto nivel que se utilizan para generar el código de la aplicación.

Los lenguajes SQL y QBE son ejemplos de 4GL. Hay otros tipos de 4GL:

— Un **generador de formularios** es una herramienta interactiva que permite crear rápidamente formularios de pantalla para introducir o visualizar datos. Los generadores de formularios permiten que el usuario defina el aspecto de la pantalla, qué información se debe visualizar y en qué lugar de la pantalla debe visualizarse. Algunos generadores de formularios permiten la creación de atributos derivados utilizando operadores aritméticos y también permiten especificar controles para la validación de los datos de entrada.

— Un **generador de informes** es una herramienta para crear informes a partir de los datos almacenados en la base de datos. Se parece a un lenguaje de consultas en que permite al usuario hacer preguntas sobre la base de datos y obtener información de ella para un informe. Sin embargo, en el generador de informes se tiene un mayor control sobre el aspecto de la salida. Se puede dejar que el generador determine automáticamente el aspecto de la salida o se puede diseñar esta para que tenga el aspecto que desee el usuario final.

— Un **generador de gráficos** es una herramienta para obtener datos de la base de datos y visualizarlos en un gráfico mostrando tendencias y relaciones entre datos. Normalmente se pueden diseñar distintos tipos de gráficos: barras, líneas, etc.

— Un **generador de aplicaciones** es una herramienta para crear programas que hagan de interface entre el usuario y la base de datos. El uso de un generador de aplicaciones puede reducir el tiempo que se necesita para diseñar un programa de aplicación. Los generadores de aplicaciones constan de procedimientos que realizan las funciones fundamentales que se utilizan en la mayoría de los programas. Estos procedimientos están escritos en un lenguaje de programación de alto nivel y forman una librería de funciones entre las que escoger. El usuario especifica qué debe hacer el programa y el generador de aplicaciones es quien determina cómo realizar la tarea.

1.4. Funciones de los sistemas de gestión de bases de datos

CODD, el creador del modelo relacional, ha establecido una lista con los ocho servicios que debe ofrecer todo SGBD.

1. Un SGBD debe proporcionar a los usuarios la capacidad de almacenar datos en la base de datos, acceder a ellos y actualizarlos. Esta es la función fundamental de un SGBD y, por supuesto, el SGBD debe ocultar al usuario la estructura física interna (la organización de los ficheros y las estructuras de almacenamiento).

2. Un SGBD debe proporcionar un catálogo en el que se almacenen las descripciones de los datos y que sea accesible por los usuarios. Este catálogo es lo que se denomina diccionario de datos, metabase o metadatos y contiene información que describe los datos de la base de datos. Normalmente, un diccionario de datos almacena:

— Nombre, tipo y tamaño de los datos.

— Nombre de las relaciones entre los datos.

— Restricciones de integridad sobre los datos.

— Nombre de los usuarios autorizados a acceder a la base de datos.

— Esquemas externos, conceptual e interno, y correspondencia entre los esquemas.

— Estadísticas de utilización, tales como la frecuencia de las transacciones y el número de accesos realizados a los objetos de la base de datos.

Algunos de los beneficios que reporta el diccionario de datos son los siguientes:

— La información sobre los datos se puede almacenar de un modo centralizado. Esto ayuda a mantener el control sobre los datos, como un recurso que son.

— El significado de los datos se puede definir, lo que ayudará a los usuarios a entender el propósito de los mismos.

— La comunicación se simplifica ya que se almacena el significado exacto. El diccionario de datos también puede identificar al usuario o usuarios que poseen los datos o que los acceden.

— Las redundancias y las inconsistencias se pueden identificar más fácilmente ya que los datos están centralizados.

— Se puede tener un historial de los cambios realizados sobre la base de datos.

— El impacto que puede producir un cambio se puede determinar antes de que sea implementado, ya que el diccionario de datos mantiene información sobre cada tipo de dato, todas sus relaciones y todos sus usuarios.

— Se puede hacer respetar la seguridad.

— Se puede garantizar la integridad.

— Se puede proporcionar información para auditorías.

3. Un SGBD debe proporcionar un mecanismo que garantice que todas las actualizaciones correspondientes a una determinada transacción se realicen, o que no se realice ninguna. Una transacción es un conjunto de acciones que cambian el contenido de la base de datos. Una transacción en el sistema informático de la empresa inmobiliaria sería dar de alta a un empleado o eliminar un inmueble. Una transacción un poco más complicada sería eliminar un empleado y reasignar sus inmuebles a otro empleado. En este caso hay que realizar varios cambios sobre la base de datos. Si la transacción falla durante su realización, por ejemplo, porque falla el hardware, la base de datos quedará en un estado inconsistente. Algunos de los cambios se habrán hecho y otros no, por lo tanto, los cambios realizados deberán ser deshechos

ADAMS

para devolver la base de datos a un estado consistente, a esto se le denomina atomicidad de las transacciones.

4. Un SGBD debe proporcionar un mecanismo que asegure que la base de datos se actualice correctamente cuando varios usuarios la están actualizando concurrentemente. Uno de los principales objetivos de los SGBD es el permitir que varios usuarios tengan acceso concurrente a los datos que comparten. El acceso concurrente es relativamente fácil de gestionar si todos los usuarios se dedican a leer datos, ya que no pueden interferir unos con otros. Sin embargo, cuando dos o más usuarios están accediendo a la base de datos y al menos uno de ellos está actualizando datos, pueden interferir de modo que se produzcan inconsistencias en la base de datos. El SGBD se debe encargar de que estas interferencias no se produzcan en el acceso simultáneo, a esto se le denomina aislamiento.

5. Un SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos en caso de que ocurra algún suceso que la dañe. Como se ha comentado antes, cuando el sistema falla en medio de una transacción, la base de datos se debe devolver a un estado consistente. Este fallo puede ser a causa de un fallo en algún dispositivo hardware o un error del software, que hagan que el SGBD aborte, o puede ser a causa de que el usuario detecte un error durante la transacción y la aborte antes de que finalice. En todos estos casos, el SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos llevándola a un estado consistente, a esto se le denomina consistencia.

6. Un SGBD debe proporcionar un mecanismo que garantice que solo los usuarios autorizados pueden acceder a la base de datos. La protección debe ser contra accesos no autorizados, tanto intencionados como accidentales.

7. Un SGBD debe ser capaz de integrarse con algún software de comunicación. Muchos usuarios acceden a la base de datos desde terminales. En ocasiones estos terminales se encuentran conectados directamente a la máquina sobre la que funciona el SGBD. En otras ocasiones los terminales están en lugares remotos, por lo que la comunicación con la máquina que alberga al SGBD se debe hacer a través de una red. En cualquiera de los dos casos, el SGBD recibe peticiones en forma de mensajes y responde de modo similar. Todas estas transmisiones de mensajes las maneja el gestor de comunicaciones de datos. Aunque este gestor no forma parte del SGBD, es necesario que el SGBD se pueda integrar con él para que el sistema sea comercialmente viable.

8. Un SGBD debe proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas. La integridad de la base de datos requiere la validez y consistencia de los datos almacenados. Se puede considerar como otro modo de proteger la base de datos, pero además de tener que ver con la seguridad, tiene otras implicaciones. La integridad se ocupa de la calidad de los datos. Normalmente se expresa mediante restricciones, que son una serie de reglas que la base de datos no puede violar. Por ejemplo, se

puede establecer la restricción de que cada empleado no puede tener asignados más de diez inmuebles. En este caso sería deseable que el SGBD controlara, que no se sobrepase este límite cada vez que se asigne un inmueble a un empleado.

Además de estos ocho servicios, es razonable esperar que los SGBD proporcionen un par de servicios más:

1. Un SGBD debe permitir que se mantenga la independencia entre los programas y la estructura de la base de datos. La independencia de datos se alcanza mediante las vistas o subesquemas. La independencia de datos física es más fácil de alcanzar, de hecho hay varios tipos de cambios que se pueden realizar sobre la estructura física de la base de datos sin afectar a las vistas. Sin embargo, lograr una completa independencia de datos lógica es más difícil. Añadir una nueva entidad, un atributo o una relación puede ser sencillo, pero no es tan sencillo eliminarlos.

2. Un SGBD debe proporcionar una serie de herramientas que permitan administrar la base de datos de modo efectivo. Algunas herramientas trabajan a nivel externo, por lo que habrán sido producidas por el administrador de la base de datos. Las herramientas que trabajan a nivel interno deben ser proporcionadas por el distribuidor del SGBD. Algunas de ellas son:

   — Herramientas para importar y exportar datos.

   — Herramientas para monitorizar el uso y el funcionamiento de la base de datos.

   — Programas de análisis estadístico para examinar las prestaciones o las estadísticas de utilización.

   — Herramientas para reorganización de índices.

   — Herramientas para aprovechar el espacio dejado en el almacenamiento físico por los registros borrados y que consoliden el espacio liberado para reutilizarlo cuando sea necesario.

1.5. Componentes de un sistema de gestión de bases de datos

Los SGBD son paquetes de software muy complejos y sofisticados que deben proporcionar los servicios comentados en la sección anterior. No se puede generalizar sobre los elementos que componen un SGBD ya que varían mucho unos de otros. Sin embargo, es muy útil conocer sus componentes y cómo se relacionan cuando se trata de comprender lo que es un sistema de bases de datos.

Un SGBD tiene varios módulos, cada uno de los cuales realiza una función específica. El sistema operativo proporciona servicios básicos al SGBD, que es construido sobre él.

— El **procesador de consultas** es el componente principal de un SGBD. Transforma las consultas en un conjunto de instrucciones de bajo nivel que se dirigen al gestor de la base de datos.

— El **gestor de la base de datos** es el interface con los programas de aplicación y las consultas de los usuarios. El gestor de la base de datos acepta consultas y examina los esquemas externo y conceptual para determinar qué registros se requieren para satisfacer la petición. Entonces el gestor de la base de datos realiza una llamada al gestor de ficheros para ejecutar la petición.

— El **gestor de ficheros** maneja los ficheros en disco en donde se almacena la base de datos. Este gestor establece y mantiene la lista de estructuras e índices definidos en el esquema interno. Si se utilizan ficheros dispersos, llama a la función de dispersión para generar la dirección de los registros. Pero el gestor de ficheros no realiza directamente la entrada y salida de datos. Lo que hace es pasar la petición a los métodos de acceso del sistema operativo que se encargan de leer o escribir los datos en el buffer del sistema.

— El **preprocesador del LMD** convierte las sentencias del LMD embebidas en los programas de aplicación, en llamadas a funciones estándar escritas en el lenguaje anfitrión. El preprocesador del LMD debe trabajar con el procesador de consultas para generar el código apropiado.

— **El compilador del LDD** convierte las sentencias del LDD en un conjunto de tablas que contienen metadatos. Estas tablas se almacenan en el diccionario de datos.

— **El gestor del diccionario** controla los accesos al diccionario de datos y se encarga de mantenerlo. La mayoría de los componentes del SGBD acceden al diccionario de datos.

Los principales componentes del gestor de la base de datos son los siguientes:

— **Control de autorización**. Este módulo comprueba que el usuario tiene los permisos necesarios para llevar a cabo la operación que solicita.

— **Procesador de comandos**. Una vez que el sistema ha comprobado los permisos del usuario, se pasa el control al procesador de comandos.

— **Control de la integridad**. Cuando una operación cambia los datos de la base de datos, este módulo debe comprobar que la operación a realizar satisface todas las restricciones de integridad necesarias.

— **Optimizador de consultas**. Este módulo determina la estrategia óptima para la ejecución de las consultas.

— **Gestor de transacciones**. Este módulo realiza el procesamiento de las transacciones.

— **Planificador** *(scheduler)*. Este módulo es el responsable de asegurar que las operaciones que se realizan concurrentemente sobre la base de datos tienen lugar sin conflictos.

— **Gestor de recuperación**. Este módulo garantiza que la base de datos permanece en un estado consistente en caso de que se produzca algún fallo.

— **Gestor de buffers**. Este módulo es el responsable de transferir los datos entre memoria principal y los dispositivos de almacenamiento secundario. A este módulo también se le denomina gestor de datos.

2. Sistemas de gestión de bases de datos relacionales

Un Sistema Gestor de Bases de Datos (SGBD) es, según ADORACIÓN DE MIGUEL, un conjunto coordinado de programas, procedimientos, lenguajes, etc., que suministra a todos los usuarios (analistas, programadores, usuarios no informáticos) los medios necesarios para describir, recuperar y manipular los datos almacenados en la base de datos, manteniendo su integridad, confidencialidad y seguridad.

Un Sistema Gestor de Bases de Datos Relacionales (SGBDR o RDBMS) se caracteriza por almacenar los datos bajo una única estructura de almacenamiento de datos llamada relación y que está basada en la teoría de conjuntos propuesta por Codd bajo la forma del cálculo y el álgebra relacional. Debe proporcionar herramientas en forma de lenguajes que permitan definir y controlar estas estructuras, junto con la integridad, seguridad y consistencia de los datos, sin olvidarnos de la inserción, recuperación y modificación de los datos.

El modelo relacional representa la segunda generación de los SGBD. En él, todos los datos están estructurados a nivel lógico como tablas formadas por filas y columnas, aunque a nivel físico pueden tener una estructura completamente distinta. Un punto fuerte del modelo relacional es la sencillez de su estructura lógica.

2.1. El Modelo Relacional

Es un modelo basado en la teoría de los conjuntos. Fue definido por Codd en 1970. Las características fundamentales del modelo relacional son:

1. Las estructura de datos son las relaciones, que presentan al usuario la información en forma de tablas bidimensionales independientemente del medio físico de almacenamiento de los datos.

2. Proporciona una base sólida para la seguridad de los datos, controlando su valor por la pertenencia a un dominio, evitando la duplicidad de la información gracias a los no duplicados y las claves, evitando la inconsistencia a través de las reglas de integridad, etc.

3. Permite la manipulación de las relaciones en forma de conjuntos, basándose bien en la teoría de conjuntos (álgebra relacional), bien en la lógica de predicados (cálculo relacional).

2.2. Conceptos fundamentales

El **esquema** de una Base de Datos Relacional se compone de uno o más esquemas de relación y de un conjunto de restricciones de integridad.

Un **esquema de relación** consiste en un nombre de relación, seguido de los nombres de los atributos: Clientes(Dni,Nombre,Apellidos).

Una **relación** (tabla) se define como un conjunto de tuplas (filas) compuestas a su vez de atributos pertenecientes cada uno a un dominio, tal que los dominios no han de ser necesariamente distintos.

Un **dominio** es un conjunto de valores que puede tomar un campo.

Un **atributo** es la unidad de información (propiedad con interés informativo de una relación) que está asociado a un dominio del que toma sus valores.

El número de atributos de una relación define su **grado**, mientras que el número de tuplas de la misma define su **cardinalidad**.

Las características de una relación serían:

— Cada relación tiene un único tipo de filas que viene dado por el esquema de la relación.

— El orden de las filas dentro de la relación es indistinto.

— Cada columna (atributo) debe estar identificada por un nombre específico.

— El orden de las columnas es indiferente.

— Cada columna ha de extraer sus valores de un dominio.

— Un mismo dominio puede valer para varios campos.

— El valor individual de un campo (intersección de cualquier fila y columna) ha de ser un único dato.

En un SGBD relacional pueden existir varios tipos de relaciones, aunque no todos manejan todos los tipos.

— **Relaciones base**. Son relaciones reales que tienen nombre y forman parte directa de la base de datos almacenada (son autónomas).

— **Vistas**. También denominadas relaciones virtuales, son relaciones con nombre y derivadas: se representan mediante su definición en términos de otras relaciones con nombre, no poseen datos almacenados propios.

— **Instantáneas**. Son relaciones con nombre y derivadas. Pero a diferencia de las vistas, son reales, no virtuales: están representadas no solo por su definición en términos de otras relaciones con nombre,

sino también por sus propios datos almacenados. Son relaciones de solo lectura y se refrescan periódicamente.

— **Resultados de consultas**. Son las relaciones resultantes de alguna consulta especificada. Pueden o no tener nombre y no persisten en la base de datos.

— **Resultados intermedios.** Son las relaciones que contienen los resultados de las subconsultas. Normalmente no tienen nombre y tampoco persisten en la base de datos.

— **Resultados temporales**. Son relaciones con nombre, similares a las relaciones base o a las instantáneas, pero la diferencia es que se destruyen automáticamente en algún momento apropiado.

2.3. Claves

Ya que en una relación no hay tuplas repetidas, estas se pueden distinguir unas de otras, es decir, se pueden identificar de modo único. La forma de identificarlas es mediante los valores de sus atributos.

Una **superclave** es un atributo o un conjunto de atributos que identifican de modo único las tuplas de una relación.

Una **clave candidata** es una superclave en la que ninguno de sus subconjuntos es una superclave de la relación. El atributo o conjunto de atributos ***K*** de la relación ***R*** es una clave candidata para ***R*** si, y solo si, satisface las siguientes propiedades:

— Unicidad: nunca hay dos tuplas en la relación ***R*** con el mismo valor de ***K***.

— Irreducibilidad (minimalidad): ningún subconjunto de ***K*** tiene la propiedad de unicidad, es decir, no se pueden eliminar componentes de ***K*** sin destruir la unicidad.

Cuando una clave candidata está formada por más de un atributo, se dice que es una **clave compuesta**. Una relación puede tener varias claves candidatas.

Para identificar las claves candidatas de una relación no hay que fijarse en un estado o instancia de la base de datos. El hecho de que en un momento dado no haya duplicados para un atributo o conjunto de atributos, no garantiza que los duplicados no sean posibles. Sin embargo, la presencia de duplicados en un estado de la base de datos sí es útil para demostrar que cierta combinación de atributos no es una clave candidata. El único modo de identificar las claves candidatas es conociendo el significado real de los atributos, ya que esto permite saber si es posible que aparezcan duplicados. Solo usando esta información semántica se puede saber con certeza si un conjunto de atributos forman una clave candidata.

La **clave primaria** de un relación es aquella clave candidata que se escoge para identificar sus tuplas de modo único. Ya que una relación no tiene tuplas duplicadas, siempre hay una clave candidata y, por lo tanto, la relación

siempre tiene clave primaria. En el peor caso, la clave primaria estará formada por todos los atributos de la relación, pero normalmente habrá un pequeño subconjunto de los atributos que haga esta función. Las claves candidatas que no son escogidas como clave primaria son denominadas **claves alternativas**.

Una **clave ajena** es un atributo o un conjunto de atributos de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación (puede ser la misma). Las claves ajenas representan relaciones entre datos. Se dice que un valor de clave ajena representa una referencia a la tupla que contiene el mismo valor en su clave primaria (tupla referenciada).

2.4. Esquema de una base de datos relacional

Una base de datos relacional es un conjunto de relaciones normalizadas. Para representar el esquema de una base de datos relacional se debe dar el nombre de sus relaciones, los atributos de estas, los dominios sobre los que se definen estos atributos, las claves primarias y las claves ajenas.

El esquema de la base de datos de la empresa inmobiliaria es el siguiente:

OFICINA	(Onum, Calle, Area, Población, Teléfono, Fax)
PLANTILLA	(Enum, Nombre, Apellido, Dirección, Teléfono, Puesto, Fecha_nac, Salario, DNI, Onum)
INMUEBLE	(Inum, Calle, Area, Población, Tipo, Hab, Alquiler, Pnum, Enum, Onum)
INQUILINO	(Qnum, Nombre, Apellido, Dirección, Teléfono, Tipo_pref, Alquiler_max)
PROPIETARIO	(Pnum, Nombre, Apellido, Dirección, Teléfono)
VISITA	(Qnum, Inum, Fecha, Comentario)

En el esquema, los nombres de las relaciones aparecen seguidos de los nombres de los atributos encerrados entre paréntesis. Las claves primarias son los atributos subrayados. Las claves ajenas se representan mediante los siguientes diagramas referenciales.

PLANTILLA	Onum →	OFICINA	:	Oficina a la que pertenece el empleado.
INMUEBLE	Pnum →	PROPIETARIO	:	Propietario del inmueble.
INMUEBLE	Enum →	PLANTILLA	:	Empleado encargado del inmueble.
INMUEBLE	Onum →	OFICINA	:	Oficina a la que pertenece el inmueble.
VISITA	Qnum →	INQUILINO	:	Inquilino que ha visitado el inmueble.
VISITA	Inum →	INMUEBLE	:	Inmueble que ha sido visitado.

2.5. Restricciones del modelo relacional

En primer lugar vamos a definir el valor nulo como un atributo cuyo valor es desconocido. No representa ni la cadena vacía ni el valor cero, sino que son valores que no tienen significado, ausencia total de información. No es lo mismo un campo vacío que un campo nulo.

— **Restricción de Dominios**. Al tener que definir cada atributo sobre un dominio se impone una restricción sobre el conjunto de valores permitidos

— **Regla de Integridad de la Entidad**. Se aplica a las **claves primarias** de las relaciones al evitar que los atributos que las componen tengan valores nulos. Se define clave primaria a un identificador irreducible que identifica de modo único las tuplas. Es irreducible porque ningún subconjunto de la clave principal sirve para identificar las tuplas de forma única, de forma que admitiríamos que algunos de los atributos no son necesarios. Esta regla no se aplica sobre los **índices secundarios** (o claves alternativas).

— **Regla de Integridad Referencial**. Se aplica sobre las **claves ajenas**. Si en una relación hay una clave ajena, los valores de la misma han de coincidir con los valores de la clave primaria a la que hacen referencia. Las claves ajenas nos crean una serie de cuestiones que debemos contestar a la hora de modificar o borrar los datos de la clave primaria. Si borramos la clave primaria, ¿Qué ocurre con los datos de la clave ajena que coinciden con ella? Podemos impedir el borrado, o bien propagarlo gracias al Borrado en Cascada. Si modificamos el valor de la clave primaria, podemos impedirlo a propagarlo gracias a la Actualización es Cascada.

Por lo tanto, para cada clave ajena de la base de datos habrá que contestar a tres preguntas:

— **Regla de los nulos:** ¿Tiene sentido que la clave ajena acepte nulos?

— **Regla de borrado:** ¿Qué ocurre si se intenta borrar la tupla referenciada por la clave ajena?

  • **Restringir:** no se permite borrar la tupla referenciada.

  • **Propagar:** se borra la tupla referenciada y se propaga el borrado a las tuplas que la referencian mediante la clave ajena.

  • **Anular:** se borra la tupla referenciada y las tuplas que la referenciaban ponen a nulo la clave ajena (solo si acepta nulos).

— **Regla de modificación:** ¿Qué ocurre si se intenta modificar el valor de la clave primaria de la tupla referenciada por la clave ajena?

  • **Restringir:** no se permite modificar el valor de la clave primaria de la tupla referenciada.

- **Propagar:** se modifica el valor de la clave primaria de la tupla referenciada y se propaga la modificación a las tuplas que la referencian mediante la clave ajena.
- **Anular:** se modifica la tupla referenciada y las tuplas que la referenciaban ponen a nulo la clave ajena (solo si acepta nulos).

2.6. Componentes

1. Un **catálogo** en el que se almacenan las descripciones de los datos también llamado Diccionario de Datos o Metabase. En el podremos encontrar los metadatos que describen los datos de la base de datos. Dentro de este diccionario podemos encontrar:
   a) Nombre, tipo y tamaño de los datos.
   b) Nombre de las relaciones.
   c) Restricciones de integridad.
   d) Nombre de los usuarios autorizados a acceder.
2. Capacidad de almacenar gran cantidad de datos, acceder a ellos y utilizarlos de modo transparente a su estructura física interna.
3. Acceso y actualización correcta ante el uso concurrente de la misma.
4. Actualización correcta de las transacciones (conjunto de acciones que representan una unidad, de tal modo que el gestor a la hora de ejecutarlas, o se actualizan todas las operaciones, o se deshacen todas).
5. Recuperación de los datos ante un daño.
6. Solo permitir accesos autorizados.
7. Asegurar la integridad.
8. Independencia entre niveles.
9. Herramientas de administración con una interfaz que nos permita acceder al motor para realizar cualquier tarea necesaria.
10. Un gestor de ficheros para controlar la grabación de los datos en disco.
11. Un lenguaje de definición de datos (DDL), otro para la manipulación de los datos (DML), otro para el control (DCL) y otro para la consulta de los datos (SQL).

12. Índices. Estructuras de datos auxiliares que permiten acelerar los procesos de ordenación y búsqueda. Un índice se define sobre un atributo o conjunto de atributos permitiendo o no los valores nulos y los valores duplicados. Son estructuras aparte de la tabla, de tal modo que se pueden crear y destruir sin que ello afecte a la misma.

13. Bloqueos para controlar el acceso concurrente y el mantenimiento de la integridad.

2.7. Vistas

En la arquitectura de tres niveles estudiada se describe una vista externa como la estructura de la base de datos tal y como la ve un usuario en particular. En el modelo relacional, el término "vista" tiene un significado un tanto diferente. En lugar de ser todo el esquema externo de un usuario, una vista es una relación virtual, una relación que en realidad no existe como tal. Una vista se puede construir realizando operaciones como las del álgebra relacional: restricciones, proyecciones, concatenaciones, etc., a partir de las relaciones base de la base de datos. Las relaciones base son aquellas que forman parte directa de la base de datos, las que se encuentran almacenadas físicamente. Un esquema externo puede tener tanto **relaciones base** como vistas derivadas de las relaciones base de la base de datos.

Una vista es el resultado dinámico de una o varias operaciones relacionales realizadas sobre las relaciones base. Una vista es una relación virtual que se produce cuando un usuario la consulta. Al usuario le parece que la vista es una relación que existe y la puede manipular como si se tratara de una relación base, pero la vista no está almacenada físicamente. El contenido de una vista está definido como una consulta sobre una o varias relaciones base. Cualquier operación que se realice sobre la vista se traduce automáticamente a operaciones sobre las relaciones de las que se deriva. Las vistas son **dinámicas** porque los cambios que se realizan sobre las tablas base que afectan a una vista se reflejan inmediatamente sobre ella. Cuando un usuario realiza un cambio sobre la vista (no todo tipo de cambios están permitidos), este cambio se realiza sobre las relaciones de las que se deriva.

Las vistas son útiles por varias razones:

— Proporcionan un poderoso mecanismo de seguridad, ocultando partes de la base de datos a ciertos usuarios. El usuario no sabrá que existen aquellos atributos que se han omitido al definir una vista.

— Permiten que los usuarios accedan a los datos en el formato que ellos desean o necesitan, de modo que los mismos datos pueden ser vistos con formatos distintos por distintos usuarios.

— Se pueden simplificar operaciones sobre las relaciones base que son complejas. Por ejemplo, se puede definir una vista como la concatenación de dos relaciones. El usuario puede hacer restricciones y proyecciones sobre la vista, que el SGBD traducirá en las operaciones equivalentes sobre la concatenación.

Se puede utilizar una vista para ofrecer un esquema externo a un usuario de modo que este lo encuentre "familiar". Por ejemplo:

— Un usuario puede necesitar los datos de los directores junto con los de las oficinas. Esta vista se crea haciendo una concatenación de las relaciones `PLANTILLA` y `OFICINA`, y proyectando sobre los atributos que se desee mantener.

— Otro usuario puede que necesite ver los datos de los empleados sin ver el salario. Para este usuario se realiza una proyección para crear una vista sin el atributo `salario`.

— Los atributos se pueden renombrar, de modo que cada usuario los vea del modo en que esté acostumbrado. También se puede cambiar el orden en que se visualizan las columnas.

— Un miembro de la plantilla puede querer ver solo los datos de aquellos inmuebles que tiene asignados. En este caso, se debe hacer una restricción para que solo se vea el subconjunto horizontal deseado de la relación `INMUEBLE`.

Como se ve, las vistas proporcionan independencia de datos a nivel lógico, que también se da cuando se reorganiza el nivel conceptual. Si se añade un atributo a una relación, los usuarios no se percatan de su existencia si sus vistas no lo incluyen. Si una relación existente se reorganiza o se divide en varias relaciones, se pueden crear vistas para que los usuarios la sigan viendo como al principio.

Cuando se actualiza una relación base, el cambio se refleja automáticamente en todas las vistas que la referencian. Del mismo modo, si se actualiza una vista, las relaciones base de las que se deriva deberían reflejar el cambio. Sin embargo, hay algunas restricciones respecto a los tipos de modificaciones que se pueden realizar sobre las vistas. A continuación, se resumen las condiciones bajo las cuales la mayoría de los sistemas determinan si se permite realizar una actualización:

— Se permiten las actualizaciones de vistas que se definen mediante una consulta simple sobre una sola relación base y que contienen la clave primaria de la relación base.

— No se permiten las actualizaciones de vistas que se definen sobre varias relaciones base.

— No se permiten las actualizaciones de vistas definidas con operaciones de agrupamiento (`GROUPBY`).

2.8. Algebra relacional

El álgebra relacional es un lenguaje formal con una serie de operadores que trabajan sobre una o varias relaciones para obtener otra relación resultado, sin que cambien las relaciones originales. Tanto los operandos como los resultados son relaciones, por lo que la salida de una operación puede

ser la entrada de otra operación. Esto permite anidar expresiones del álgebra, del mismo modo que se pueden anidar las expresiones aritméticas. A esta propiedad se le denomina **clausura**: las relaciones son cerradas bajo el álgebra, del mismo modo que los números son cerrados bajo las operaciones aritméticas.

Primero se describen los ocho operadores originalmente propuestos por Codd y después se estudian algunos operadores adicionales que añaden potencia al lenguaje.

De los ocho operadores, solo hay cinco que son fundamentales: *restricción, proyección, producto cartesiano, unión y diferencia,* que permiten realizar la mayoría de las operaciones de obtención de datos. Los operadores no fundamentales son la *concatenación (join), la intersección y la división,* que se pueden expresar a partir de los cinco operadores fundamentales.

La restricción y la proyección son operaciones *unarias* porque operan sobre una sola relación. El resto de las operaciones son binarias porque trabajan sobre pares de relaciones.

2.8.1. Restricción

```
:R WHERE condición
```

La restricción, también denominada selección, opera sobre una sola relación R y da como resultado otra relación cuyas tuplas son las tuplas de R que satisfacen la condición especificada. Esta condición es una comparación en la que aparece al menos un atributo de R, o una combinación booleana de varias de estas comparaciones.

Ejemplo: obtener todos los empleados con un salario anual superior a 15.000 euros.

```
PLANTILLA WHERE salario>15000
```

Ejemplo: obtener todos los inmuebles de Castellón con un alquiler mensual de hasta 350 euros.

```
INMUEBLE WHERE población=`Castellón' AND alquiler<=350
```

2.8.2. Proyección

```
:R[a^i, ..., a^k]
```

La proyección opera sobre una sola relación R y da como resultado otra relación que contiene un subconjunto vertical de R, extrayendo los valores de los atributos especificados y eliminando duplicados.

Ejemplo: obtener un listado de empleados mostrando su número, nombre, apellido y salario.

```
SELECT enum,nombre,apellido,salario FROM PLANTILLA
```

2.8.3. Producto cartesiano

```
:R X S
```

Dadas dos relaciones R y S, el producto cartesiano genera una relación resultante de combinar cada fila de R con todas las de S. Se representa como RxS El producto cartesiano obtiene una relación cuyas tuplas están formadas por la concatenación de todas las tuplas de R con todas las tuplas de S.

La restricción y la proyección son operaciones que permiten extraer información de una sola relación. Habrá casos en que sea necesario combinar la información de varias relaciones. El producto cartesiano "multiplica" dos relaciones, definiendo una nueva relación que tiene todos los pares posibles de tuplas de las dos relaciones. Si la relación R tiene P tuplas y N atributos y la relación S tiene Q tuplas y Matributos, la relación resultado tendrá P*Q tuplas y N+M atributos. Ya que es posible que haya atributos con el mismo nombre en las dos relaciones, el nombre de la relación se antepondrá al del atributo en este caso para que los nombres de los atributos sigan siendo únicos en la relación resultado.

Como se puede observar, la relación resultado contiene más información de la que se necesita. Para obtener el listado que se pide en el ejemplo, es necesario realizar una restricción para quedarse solamente con las tuplas en donde el campo común tenga valores iguales

```
(INQUILINO[qnum,nombre,apellido] X VISITA[qnum,inum,comentario])
                WHERE inquilino.qnum=visita.qnum
```

La combinación del producto cartesiano y la restricción del modo en que se acaba de realizar, se puede reducir a la operación de concatenación *(join)* que se presenta más adelante.

2.8.4. Unión

```
:R ∪ S
```

La unión de dos relaciones R y S, con P y Qtuplas respectivamente, es otra relación que tiene como mucho P+Q tuplas siendo estas las tuplas que se encuentran en R o en S o en ambas relaciones a la vez. En el caso de que una misma tupla esté en las dos relaciones que se unen, el resultado de la unión no la tendrá repetida. Para poder realizar esta operación, R y S deben ser compatibles para la unión.

Se dice que dos relaciones son **compatibles para la unión** si ambas tienen la misma cabecera, es decir, si tienen el mismo número de atributos y estos se encuentran definidos sobre los mismos dominios para cada uno de los atributos. En muchas ocasiones será necesario realizar proyecciones para hacer que dos relaciones sean compatibles para la unión.

Ejemplo: obtener un listado de las áreas en las que hay oficinas o inmuebles para alquilar.

```
OFICINA[área] ∪ INMUEBLE[área]
```

2.8.5. Diferencia

```
:R - S
```

La diferencia obtiene una relación que tiene las tuplas que se encuentran en R y no se encuentran en S. Para realizar esta operación, R y S deben ser compatibles para la unión.

Ejemplo: obtener un listado de todas las poblaciones en donde hay una oficina y no hay inmuebles para alquilar.

```
OFICINA[población] - INMUEBLE[población]
```

2.8.6. Concatenación Interna *(Inner-Join)*

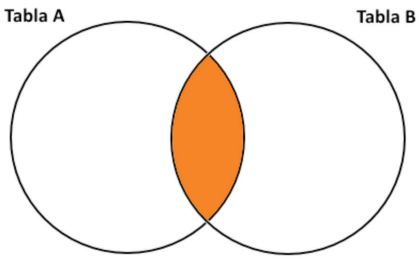
```
:R [INNER] JOIN S
```

La palabra INNER es opcional. La concatenación de dos relaciones R y S obtiene como resultado una relación cuyas tuplas son todas las tuplas de R concatenadas con todas las tuplas de S que en los atributos comunes (que se llaman igual) tienen los mismos valores. Estos atributos comunes aparecen una sola vez en el resultado.

Veamos un **ejemplo**: para seleccionar los registros comunes entre la Tabla1 y la Tabla2 que tengan correspondencia entre ambas tablas por el campo Col1, escribiríamos:

SELECT T1.Col1, T1.Col2, T1.Col3, T2.Col7

FROM Tabla1 T1 INNER JOIN Tabla2 T2 ON T1.Col1 = T2.Col1



INNER JOIN

2.8.7. Concatenación externa *(Outer-join)*

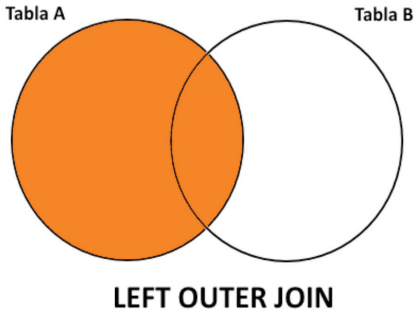
Las concatenaciones externas se realizan mediante la instrucción OUTER JOIN. Como más adelante se indica, esta operación devuelve todos los valores de la tabla que hemos puesto a la derecha (RIGHT), los de la tabla que hemos puesto a la izquierda (LEFT) o los de ambas tablas (FULL), según el

caso, devolviendo además valores nulos en las columnas de las tablas que no tengan el valor existente en la otra tabla. En todas estas combinaciones externas el uso de la palabra OUTER es opcional. Si utilizamos LEFT, RIGHT o FULL y la combinación de columnas, el sistema sobreentiende que estamos haciendo una combinación externa.

- **LEFT JOIN**

En esta variante se obtienen todas las filas de la tabla colocada a la izquierda, aunque no tengan correspondencia en la tabla de la derecha. Así, para seleccionar todas las filas de la Tabla1, aunque no tengan correspondencia con las filas de la Tabla2, suponiendo que se combinan por la columna Col1 de ambas tablas, escribiríamos:

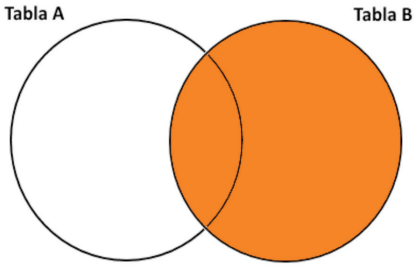
```
SELECT T1.Col1, T1.Col2, T1.Col3, T2.Col7
FROM Tabla1 T1 LEFT [OUTER] JOIN Tabla2 T2 ON T1.Col1 = T2.Col1
```



- **RIGHT JOIN**

Análogamente a la anterior, usando RIGHT JOIN se obtienen todas las filas de la tabla de la derecha, aunque no tengan correspondencia en la tabla de la izquierda. Así, para seleccionar todas las filas de la Tabla2, aunque no tengan correspondencia con las filas de la Tabla1, podemos utilizar la cláusula RIGHT. La sentencia asociada es:

```
SELECT T1.Col1, T1.Col2, T1.Col3, T2.Col7
FROM Tabla1 T1 RIGHT [OUTER] JOIN Tabla2 T2 ON T1.Col1 = T2.Col1
```



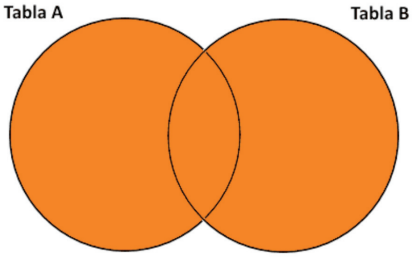
RIGHT OUTER JOIN

- **FULL JOIN**

La variante FULL obtiene todas las filas en ambas tablas, aunque no tengan correspondencia en la otra tabla. Es decir, todos los registros de A y de B, aunque no haya correspondencia entre ellos, rellenando con nulos los campos que falten. Es equivalente a obtener los registros comunes (con un INNER) y luego añadirle los de la tabla A que no tienen correspondencia en la tabla B, con los campos de la tabla vacíos, y los registros de la tabla B que no tienen correspondencia en la tabla A, con los campos de la tabla A vacíos. La sentencia asociada es:

```
SELECT T1.Col1, T1.Col2, T1.Col3, T2.Col7

FROM Tabla1 T1 FULL [OUTER] JOIN Tabla2 T2 ON T1.Col1 = T2.Col1
```



FULL OUTER JOIN

2.8.8. Intersección

```
:R ∩ S
```

La intersección obtiene como resultado una relación que contiene las tuplas de R que también se encuentran en S. Para realizar esta operación, R y S deben ser compatibles para la unión.

La intersección se puede expresar en términos de diferencias:

$$R \cap S = R - (R - S)$$

2.8.9. División

```
:R / S
```

Suponiendo que la cabecera de R es el conjunto de atributos A y que la cabecera de S es el conjunto de atributos B, tales que B es un subconjunto de A, y si C = A - B (los atributos de R que no están en S), la división obtiene una relación cuya cabecera es el conjunto de atributos C y que contiene las tuplas de R que están acompañadas de todas las tuplas de S.

Ejemplo: obtener los inquilinos que han visitado todos los inmuebles de tres habitaciones.

```
VISITA[qnum,inum] / (INMUEBLE WHERE hab = 3)[inum]
```

2.9. Soluciones de Sistemas Gestores de Bases de Datos Relacionales (SGBDR)

2.9.1. Oracle

Oracle es el sistema comercial de gestión de bases de datos relacionales más conocido del mercado y es el principal dominador del mercado dentro del mundo empresarial, a pesar de su alto precio. Es considerado como uno de los sistemas de bases de datos más completos; entre sus características se pueden destacar:

— El soporte de transacciones.

— La estabilidad.

— La escalabilidad.

— Que es multiplataforma.

2.9.2. DB2

El sistema de gestión de bases de datos relacionales que comercializa IBM se conoce como DB2. Al igual que Oracle, el origen del mismo es bastante antiguo, de hecho fue el primer gestor de bases de datos en utilizar el lenguaje SQL.

2.9.3. SQL Server

SQLServer lo presentó Microsoft como competencia de Oracle y DB2 en el mercado de los sistemas gestores de bases de datos relacionales. Una particularidad de este SGBDR es que utiliza una variación del lenguaje SQL denominado Transact SQL, una implementación de SQL-92 (el estándar ISO para SQL certificado en 1992) con bastantes extensiones. Proporciona sintaxis adicional en procedimientos almacenados.

2.9.4. PostgreSQL

PostgreSQL es un SGBDR-Orientado a objetos open-source. Se distribuye bajo licencia BSD y su desarrollo no es manejado por una sola compañía, sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales que trabajan en su desarrollo, denominada PGDG (PostgreSQL Global Development Group).

2.9.5. MySQL

MySQL es un sistema de gestión de bases de datos relacional, desarrollado bajo licencia GPL, que actualmente se encuentra en manos de Oracle Corporation y que se distribuye tanto con licencia GLP como con licencia comercial. Está considerado como la base datos open source más popular del mundo. MySQL fue inicialmente desarrollado por MySQL A.B. (empresa fundada por David Axmark, Allan Larsson y Michael Widenius). MySQL A.B. fue adquirida por Sun Microsystems en 2008 y esta, a su vez, fue comprada por Oracle Corporation en 2010, la cual ya era dueña desde 2005 de Innobase Oy, empresa finlandesa desarrolladora del motor InnoDB para MySQL.

2.9.6. Otros SGBDR

Si bien los anteriores son los más habituales, existen otros sistemas gestores de bases de datos relacionales como son: MaríaDB, MaxDB, SQLite, Firebird, Informix, HSLDB, Ingres, InterBase, SmallSQL...

3. Mapeo Objeto/Relacional

3.1. Características

Las herramientas de mapeo objeto/relacional (*Object/Relational Mapping*, ORM), también denominadas mediadores objeto-relacionales, envoltorios o wrappers, son productos que ofrecen capas de software orientado a objetos que permiten trabajar sobre una base de datos relacional. Estos sistemas ofrecen al desarrollador la sensación de estar trabajando con un SGBDOO (Sistema de Gestión de Bases de Datos Orientadas a Objetos), ocultando los mecanismos necesarios para traducir las clases en tablas relacionales y los objetos en tuplas. Estas herramientas generalmente emplean SQL para interactuar con la base de datos relacional.

Una solución ORM comprende 3 componentes básicos:

— Una API para ejecutar las operaciones CRUD (Create - Crear, Retrieve - Recuperar, Update - Actualizar, Delete - Borrar) básicas sobre los objetos de las clases persistentes.

— Una API o un lenguaje para especificar consultas que hagan referencia a las clases o a las propiedades de las clases.

— Un mecanismo para especificar los metadatos de mapeo. Las herramientas ORM necesitan un formato de metadatos para que la aplicación especifique cómo se realizará la traducción entre clases y tablas, propiedades y columnas, asociaciones y claves ajenas, tipos Java y tipos SQL.

Las herramientas ORM pueden ser implementadas de diversas maneras. Se definen los siguientes niveles de calidad para las herramientas de mapeo objeto/relacional:

— **Relacional pura (*Pure relational*).** Toda la aplicación, incluido el interfaz de usuario, se diseña siguiendo el modelo relacional y las operaciones relacionales basadas en SQL. Como principal ventaja destaca que la manipulación directa de código SQL permite muchas optimizaciones. Sin embargo, inconvenientes tales como la dificultad de mantenimiento y la falta de portabilidad son muy significativos. Las aplicaciones en esta categoría utilizan intensivamente procedimientos almacenados (stored procedures), desplazando parte de la carga de trabajo desde la capa de negocio hasta la base de datos.

— **Mapeo de objetos ligero (*Light object mapping*).** Las entidades se representan como clases que se mapean de forma manual sobre tablas relacionales. La codificación manual de SQL puede quedar separada de la lógica de negocio si se utilizan patrones de diseño como Data Access Object (DAO). Esta alternativa está ampliamente extendida para aplicaciones con un número pequeño de entidades o aplicaciones con un modelo de datos genérico dirigido por metadatos.

— **Mapeo de objetos medio (*Medium object mapping*).** La aplicación se diseña en torno a un modelo de objetos. SQL se genera en tiempo de compilación, utilizando una herramienta de generación de código, o en tiempo de ejecución, utilizando el código de un framework. El sistema de persistencia soporta las relaciones entre objetos y las consultas pueden especificarse utilizando un lenguaje de expresiones orientadas a objetos. Una gran parte de las soluciones ORM soportan este nivel de funcionalidad. Son adecuadas para aplicaciones de un tamaño medio, con algunas transacciones complejas, en particular, cuando la portabilidad entre diferentes bases de datos es un factor importante.

— **Mapeo de objetos total *(Full object mapping).*** Estas herramientas soportan características avanzadas del modelado de objetos: composición, herencia, polimorfismo y persistencia por alcance. La capa de persistencia implementa persistencia transparente y las clases persistentes no necesitan heredar ninguna clase base especial ni implementar un determinado interfaz. Además, se incluyen diversas optimizaciones que son transparentes para la aplicación, como varias estrategias para cargar los objetos de forma transparente desde el almacén utilizado cuando se navega por un árbol de objetos o la utilización de cachés de objetos.

3.2. Ventajas

— Incrementan la productividad al evitar que sea el propio programador el que tenga que hacer la conversión de clases y objetos a tablas y viceversa, evitando así posibles errores por parte de los equipos de desarrollo.

— El código a programar encargado de la traducción puede suponer entre un 25% y un 40% del total de la aplicación.

— Favorecen el mantenimiento de la aplicación, al separar el código que interactúa con la base de datos con el código correspondiente a la lógica de negocio de la aplicación.

— Permiten aprovechar las ventajas derivadas de la experiencia de los SGBD relacionales, del rendimiento que proporcionan, además de que favorecen la independencia de elección de fabricante, pues todos soportan el estándar SQL.

3.3. Soluciones

- **Hibernate**

Hibernate es una herramienta de mapeo objeto/relacional para la plataforma Java. Se trata de un proyecto de código abierto distribuido bajo licencia LGPL que fue desarrollado por un grupo de programadores Java de diversos países liderados por Gavin King. También hay una distribución para .NET conocida como NHibernate. La traducción la realiza mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

— Otras soluciones.

— Propel, escrito en PHP y que soporta el uso de SQLite, MySQL y PostgreSQL.

— Peewee, escrito en Python y que soporta el uso de SQLite, MySQL y PostgreSQL.

— Doctrine, escrito en PHP y que soporta, entre otros, MySQL.

4. Sistemas de gestión de bases de datos orientados a objetos

4.1. Introducción

Las Bases de Datos Orientadas a Objetos (BDOO) son aquellas cuyo modelo de datos está orientado a objetos y almacenan y recuperan objetos en los que se almacenan estado y comportamiento. Su origen se debe a que en los modelos clásicos de datos existen problemas para representar cierta información, puesto que aunque permiten representar gran cantidad de datos, las operaciones que se pueden realizar con ellos son bastante simples. Las clases utilizadas en un determinado lenguaje de programación orientado a objetos son las mismas clases que serán utilizadas en una BDOO, de tal manera que no es necesaria una transformación del modelo de objetos para ser utilizado por un SGBDOO. De forma contraria, el modelo relacional requiere abstraerse lo suficiente como para adaptar los objetos del mundo real a tablas. Las bases de datos orientadas a objetos surgen para evitar los problemas que aparecen al tratar de representar cierta información, aprovechar las ventajas del paradigma orientado a objetos

en el campo de las bases de datos y para evitar transformaciones entre modelos de datos (usar el mismo modelo de objetos).

4.2. Definiciones

— **Base de datos orientada a objetos (BDOO):** una colección persistente y compatible de objetos definida por un modelo de datos orientado a objetos.

— **Modelo de datos orientado a objetos:** un modelo de datos que captura la semántica de los objetos soportados en la programación orientada a objetos.

— **Sistema Gestor de Bases de Datos Orientadas a Objetos (SGBDOO):** el gestor de una base de datos orientada a objetos.

4.3. Características de una BDOO según el manifiesto de Malcolm Atkinson

En 1989 se hizo el Manifiesto de los sistemas de base de datos orientados a objetos, el cual propuso trece características obligatorias para un SGBDOO y cuatro opcionales. Las trece características obligatorias estaban basadas en dos criterios: debía tratarse de un sistema orientado a objetos y un SGBD.

Características obligatorias de orientación a objetos:

1. Deben soportarse objetos complejos.
2. Deben soportarse mecanismos de identidad de los objetos.
3. Debe soportarse la encapsulación.
4. Deben soportarse los tipos o clases.
5. Los tipos o clases deben ser capaces de heredar de sus ancestros.
6. Debe soportarse el enlace dinámico.
7. El DML debe ser computacionalmente complejo.
8. El conjunto de todos los tipos de datos debe ser ampliable.

Características obligatorias de SGBD:

1. Debe proporcionarse persistencia a los datos.
2. El SGBD debe ser capaz de gestionar bases de datos de muy gran tamaño.
3. El SGBD debe soportar a usuarios concurrentes.

4. El SGBD debe ser capaz de recuperarse de fallos de hardware y de software.
5. El SGBD debe proporcionar una forma simple de consultar los datos.

Características opcionales:

1. Herencia múltiple.
2. Comprobación de tipos e inferencia de tipos.
3. Sistema de base de datos distribuido.
4. Soporte de versiones.

4.4. Ventajas e inconvenientes

Las **ventajas** de un SGBDOO son:

— **Mayor capacidad de modelado.** El modelado de datos orientado a objetos permite modelar el "mundo real" de una manera mucho más fiel. Esto se debe a:

- Un objeto permite encapsular tanto un estado como un comportamiento.
- Un objeto puede almacenar todas las relaciones que tenga con otros objetos.
- Los objetos pueden agruparse para formar objetos complejos (herencia).

— **Ampliabilidad.** Esto se debe a:

- Se pueden construir nuevos tipos de datos a partir de los ya existentes.
- Se pueden realizar agrupaciones de propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
- Permite la reusabilidad de clases, lo que repercute en una mayor facilidad de mantenimiento y un menor tiempo de desarrollo.

— **Lenguaje de consulta más expresivo.** El acceso navegacional desde un objeto al siguiente es la forma más común de acceso a datos en un SGBDOO, mientras que SQL utiliza el acceso asociativo. El acceso navegacional es más adecuado para gestionar operaciones como los despieces, consultas recursivas, etc.

— **Adecuación a las aplicaciones avanzadas de base de datos.** Hay muchas áreas en las que los SGBD tradicionales no han tenido excesivo éxito, como el CAD, CASE, OIS, sistemas multimedia, etc., en los que las capacidades de modelado de los SGBDOO han hecho que esos sistemas sí resulten efectivos para este tipo de aplicaciones.

— **Mayores prestaciones.** Los SGBDOO proporcionan mejoras significativas de rendimiento con respecto a los SGBD relacionales, aunque hay autores que han argumentado que los bancos de prueba usados están dirigidos a aplicaciones de ingeniería donde los SGBDOO son más adecuados. También está demostrado que los SGBDR tienen un rendimiento mejor que los SGBDOO en las aplicaciones tradicionales de bases de datos, como el procesamiento de transacciones en línea (OLTP).

Los inconvenientes de un SGBDOO son:

— **Carencia de un modelo de datos universal.** No hay ningún modelo de datos que esté universalmente aceptado para los SGBDOO y la mayoría de los modelos carecen una base teórica.

— **Carencia de experiencia.** Todavía no se dispone del nivel de experiencia del que se dispone para los sistemas tradicionales.

— **Carencia de estándares.** Existe una carencia de estándares general para los SGBDOO.

— **Competencia.** Con respecto a los SGBDR y los SGBDOR, estos productos tienen una experiencia de uso considerable. SQL es un estándar aprobado y ODBC es un estándar de facto. Además, el modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales.

— **La optimización de consultas** compromete la encapsulación. La optimización de consultas requiere una compresión de la implementación de los objetos para poder acceder a la base de datos de manera eficiente. Sin embargo, esto compromete el concepto de encapsulación.

— El modelo de objetos aún **no tiene una teoría matemática coherente** que le sirva de base.

4.5. ODMG: estándar de facto para modelos de objetos

ODMG es un grupo de representantes de la industria de bases de datos que fue concebido en el verano de 1991 con el objetivo de definir estándares para los SGBDOO. Uno de sus estándares, el cual lleva el mismo nombre del grupo (ODMG), es el del modelo para la semántica de los objetos de una base de datos. El modelo de objetos ODMG es un superconjunto del modelo de objetos de OMG, que permite portar tanto los diseños como las implementaciones entre diversos sistemas compatibles.

La última versión del estándar, ODMG 3.0, propone los siguientes componentes principales de la arquitectura ODMG para un SGBDOO:

— **Modelo de objetos.** Permite que los diseños y las implementaciones sean portables entre los sistemas que lo soportan. Primitivas de modelado:

- Componentes básicos son objetos y literales. Un objeto es una instancia autocontenida de una entidad de interés del mundo real; tienen identificador único. Literal es un valor específico, no tiene identificadores; puede ser una estructura o un conjunto de valores relacionados.
- Se categorizan en tipos. Cada tipo tiene un dominio específico compartido por todos los objetos y literales de ese tipo. Los tipos también pueden tener comportamientos, que también comparten todos los objetos del mismo.
- Lo que un objeto sabe hacer son sus operaciones. Puede requerir datos de entrada y devolver algún valor de un tipo conocido.
- Las propiedades son sus atributos y las relaciones. El estado viene dado por los valores actuales de sus propiedades.
- Una base de datos es un conjunto de objetos almacenados que pueden ser accesibles por múltiples usuarios y aplicaciones.
- La definición de una base de datos está contenida en un esquema que se ha creado mediante el lenguaje de definición de objetos ODL.

— **Lenguaje de definición de objetos (ODL, *Object Definition Language*)**. Es el equivalente de DDL (*Data Definition Languaje* o lenguaje de definición de datos) de los DBMS tradicionales. Define los atributos y las relaciones entre tipos y especifica la signatura de las operaciones. La sintaxis de ODL extiende el lenguaje de definición de interfaces (IDL) de la arquitectura CORBA (*Common Object Request Broker Architecture*). Las declaraciones de atributos son sintácticamente idénticas a las declaraciones de miembros de C++.

— **Lenguaje de consulta de objetos (OQL, *Object Query Language*)**. Es un lenguaje declarativo del tipo de SQL que permite realizar consultas de modo eficiente sobre bases de datos orientadas a objetos, incluyendo primitivas de alto nivel para conjuntos de objetos y estructuras. Está basado en el estándar SQL-92, proporcionando un superconjunto de la sentencia SELECT. OQL no posee primitivas para modificar el estado de los objetos, ya que estas se deben realizar a través de los métodos que dichos objetos poseen. OQL es ortogonal respecto a la especificación de expresiones de caminos: atributos, relaciones y colecciones pueden ser utilizados en estas expresiones, siempre que el sistema de tipos de OQL no se vea comprometido. OQL tiene además otras características:

  - Especificación de vistas dando nombres a consultas.
  - Obtención como resultado de un solo elemento.
  - Uso de operadores de colecciones: funciones agregados (max, min, count, sum, avg) y cuantificadores (for all, exists).
  - Uso de group by.

— **Conexión con los lenguajes C++, Smalltalk y Java (al menos).**

4.6. Soluciones de sistemas gestores de bases de datos orientados a objetos

- **ObjectDB**

Es un sistema de gestión de base de datos orientada a objetos para Java. ObjectDB ofrece todos los servicios de gestión de base de datos estándar (almacenamiento y recuperación, operaciones, gestión de bloqueo, de procesamiento de consultas, etc.), pero de una manera que hace que el desarrollo sea más fácil y las aplicaciones más rápidas. Para ello incorpora dos API estándar de JAVA, JPA (Java Persistant API) y JDO (Java Data Objects).

- **Zope Object Database**

La Zope Object Database (ZODB) es una base de datos orientada a objetos para almacenar de forma transparente y persistente objetos en el lenguaje de programación Python. Ofrece transacciones, historial de transacciones y undo, almacenamiento conectable, almacenamiento en caché, control de concurrencia multiversión y escalabilidad.

- **DB4O**

DB4O (DataBase 4 (for) Objects) es un motor de base de datos orientada a objetos que, a su vez, es el nombre de la compañía que lo desarrolla: db4objects, Inc. Actualmente se ofrece para dos entornos de desarrollo: .NET y Java.

- **GemStone/S**

GemStone/S es una base de datos orientada a objetos, propiedad de GemTalk Systems, disponible para Smalltalk y Java.

- **Objectivity/DB**

Objectivity/DB es una base de datos orientada a objetos disponible para su uso con C ++, C #, Java u objetos Python. Además es compatible con lenguajes que utilicen el tándem SQL/ODBC y XML. Se ejecuta sobre plataformas Linux, Macintosh, UNIX y Windows.

- **ObjectStore**

Es una base de datos orientada a objetos escrita en C++ y Java que puede ejecutarse tanto en Windows como en Linux

- **Versant Object Database (VOD)**

Es un base de datos orientada a objetos escrita en C++, C#, Java, Smalltalk, Python, que puede ejecutarse en Windows, Linux, HP-UX, AIX...