UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Diplomatura en Bases de Datos

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 3: Herramientas de cara al futuro

Unidad 3: Bases de datos orientadas a objetos

Presentación:

En esta Unidad descubrimos qué es una base de orientada a objetos, para lo cual debemos entender en qué consiste el modelo de datos de objetos. Finalmente, recorremos todos los componentes del modelo orientado a objeto y su lenguaje, así como también el formato extendido de SQL, llamado SQL3.

Objetivos:

Que los participantes:

- Conozcan y entiendan qué es una base de datos orientada a objetos.
- Identifiquen en qué problemas conviene aplicar esta tecnología y sus limitaciones.
- Aprendan a seleccionar entre las distintas bases de datos disponibles.
- Se familiaricen con los usos básicos de una base de datos orientada a objetos y sus lenguajes asociados.

Bloques temáticos:

1. Descripción del concepto de base de datos orientada a objetos

2. ¿En qué consiste el modelo orientado a objetos?

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

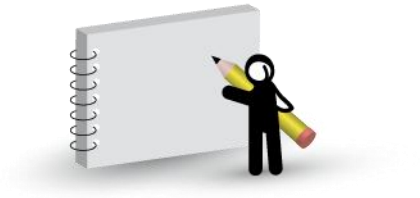
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***



Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Descripción del concepto de base de datos orientada a objetos:

OODBMS significa sistema de gestión de bases de datos orientadas a objetos. Este sistema de gestión de bases de datos orientado a objetos (también conocido simplemente como una base de datos de objetos) es un DBMS donde los datos se representan en forma de objetos, tal como se usan en la programación orientada a objetos. Es decir, que cuando se realiza una consulta, estas bases de datos devuelven los objetos en su totalidad. Un objeto devuelto significa que sus atributos y métodos son tan utilizables como lo eran antes de que el objeto fuera alguna vez almacenado en la base de datos.

Actualmente, la base de datos más popular en uso es una base de datos relacional. Estas almacenan sus datos en tablas, con cada fila en una tabla correspondiente a un registro y cada columna que representa las propiedades del registro. Su principal distinción es que solo pueden contener tipos de datos primitivos, como un entero o texto. No son capaces de almacenar tipos de datos más complejos a menos que los datos sean serializado o dividido en componentes primitivas.

De esta manera, un beneficio de las bases de datos orientadas a objetos es que, cuando se integra con un lenguaje de programación orientado a objetos, existe una consistencia mucho mayor entre la base de datos y el lenguaje de programación. Ambos usan el mismo modelo de representación para los datos y no existe la necesidad de dividir al objeto en componentes primitivas.

Esto está en contraste con una base de datos relacional, donde hay una clara diferencia entre el modelo de base de datos y el modelo de programación.

Algunos OODBMS están diseñados para funcionar con otros lenguajes de programación (como Java, Python, Perl, Delphi, Ruby, C #, Visual Basic .NET, C ++, etc.). Otros tienen su propio lenguaje de programación.

Algunos DBMS son un híbrido de OODBMS y RDBMS, por lo que se los denomina bases de datos relacionales de objetos (ORD) o sistemas de gestión de bases de datos relacionales de objetos (ORDBMS).

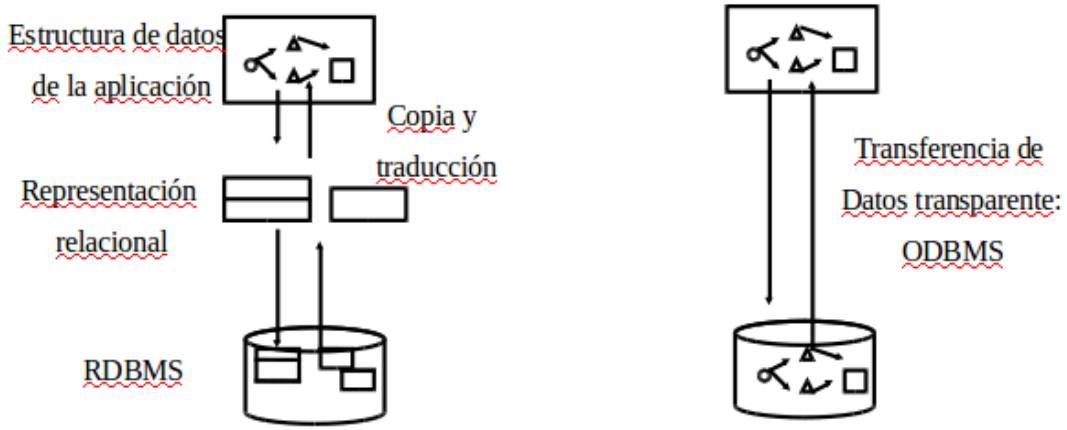
Sin embargo, a pesar de presentar una ventaja significativa cuando se trabaja con lenguajes de programación orientados a objetos también existen desventajas y, en algunos escenarios, la base de datos relacional sería la más adecuada.

En primer lugar, no hay duda de que las bases de datos relacionales son mucho más simple que las bases de datos orientadas a objetos; solo tienen datos primitivos que se almacenan de manera muy uniforme. Por lo tanto, cuando se trata de datos muy simplistas que puede estar contenida en solo una o dos tablas, es más eficiente usar una base de datos relacional. Otra desventaja de las bases de datos orientadas a objetos es el hecho de que no tienen estándares establecidos. Esto significa que hay menos herramientas de usuario para las bases de datos de objetos, y menos soporte en general para desarrolladores. La multitud de APIs de algún idioma específico también hace que sea más difícil portar una base de datos de objetos a una aplicación que está escrita en otro lenguaje de programación, porque probablemente implicaría el uso de una API diferente.

Programación orientada a objetos y las OODBMS

La programación orientada a objetos representó una gran ganancia en la productividad del programador, la confiabilidad de la aplicación, y rendimiento. A finales de los años ochenta y principios de los noventa, la mayoría de los lenguajes de programación se convirtieron en un modelo orientado a objetos, y surgieron muchos lenguajes nuevos importantes, como Java, que estaban orientados a objetos nativos. La revolución de la programación orientada a objetos preparó el escenario para el primer desafío serio a las base de datos relacionales, que surgió a mediados de la década de 1990: había una falta de concordancia entre las representaciones orientadas a objetos de sus datos dentro de los programas y la representación relacional dentro de la base de datos. Así, los primeros lenguajes de programación orientados a objetos, comenzando con Simula y Smalltalk, presentaron una forma alternativa de diseñar programas en la que las estructuras de datos y sus operaciones tenían una importancia primordial. La programación orientada a objetos es ampliamente reconocida como un método para producir código altamente confiable y reutilizable. La promesa de los primeros lenguajes orientados a objetos; la

popularidad de los lenguajes orientados a objetos C ++, C #, Python, Ruby y Java; y la adición de capacidades orientadas a objetos a otros lenguajes como Visual Basic.Net influenciaron el modelado de base de datos. El desafío de extender estos lenguajes a las bases de datos es moverse de los objetos temporales creados y manipulados por programas a objetos persistentes que se pueden almacenar en una base de datos. En estos programas orientados a objetos, todos los detalles relevantes a una unidad lógica de trabajo se almacenaría dentro de la clase directamente vinculado a esa clase. Por ejemplo, un objeto del cliente contendría todos los detalles sobre el cliente, con enlaces a objetos que contenían a pedidos de cliente, que a su vez tenían enlaces para ordenar artículos de línea. Esta representación era intrínsecamente no relacional; en efecto, la representación de los datos coincidió más estrechamente con las bases de datos de red de la era CODASYL (Conference on Data Systems Languages, un consorcio de industrias informáticas formado en 1959 con el objeto de regular el desarrollo de un lenguaje de programación estándar que pudiera ser utilizado en multitud de computadoras. De todos estos esfuerzos resultó el lenguaje COBOL). Cuando un objeto se almacenaba o se recuperaba de una base de datos relacional, varias operaciones de SQL eran requeridas convertir de la representación orientada a objetos a la representación relacional. Esto resultaba muy engorroso para el programador y generaba problemas de rendimiento o confiabilidad. El problema que se presentaba se puede visualizar en la siguiente figura:



Los defensores de la programación orientada a objetos comenzaron a ver la base de datos relacional como algo obsoleto, y esto llevó casi inevitablemente a la proposición de que un sistema de gestión de bases de datos orientadas a objetos (OODBMS) era más adecuado para satisfacer las demandas de las aplicaciones modernas. Un OODBMS almacenaría objetos del programa directamente sin normalización, y permitiría que las aplicaciones carguen y almacenen sus objetos fácilmente. Los proveedores de bases de datos relacionales incumbentes: en este punto, principalmente Oracle, Informix, Sybase e IBM, aceleraron rápidamente para implementar las características de OODBMS dentro de su RDBMS. Mientras tanto, se desarrollaron algunos sistemas OODBMS puros y obtuvieron fuerza inicial. Sin embargo, para el final de la década, los sistemas OODBMS habían fallado por completo en ganar participación en el mercado. Los proveedores de bases de datos tradicionales como Oracle e Informix han implementado con éxito muchas características de las OODBMS, pero estas características rara vez se utilizan.

Los programadores de OO se resignaron al uso de sistemas de RDBMS para almacenar objetos, y el dolor se alivió de alguna manera por marcos de mapeo relacional de objetos (ORM) que automatizan los aspectos más tediosos de la traducción. Existen explicaciones competitivas y no necesariamente contradictorias para el fracaso de la base de datos OO. Por una parte, los defensores del modelo OODBMS se concentraban sólo en las ventajas ofrecidas al desarrollador de la aplicación e ignoraban las desventajas que el nuevo modelo tenía para aquellos que deseaba consumir información para fines comerciales.

Las bases de datos no existen simplemente para el beneficio de programadores; representan activos significativos que deben ser accesibles para aquellos que quieren explotar la información para la toma de decisiones y la inteligencia empresarial. Implementar un modelo de datos que solo podría ser utilizado por el programador y privar al usuario comercial de una interfaz SQL utilizable, generó que OODBMS fallara en obtener apoyo fuera de los círculos de programación. Sin embargo, las motivaciones para un OODBMS influyeron fuertemente en algunas de las bases de datos no relacionales más populares de la actualidad.

A difrencia de los lenguajes orientados objetos, en los lenguajes de programación "procedimentales" tradicionales, los datos y la lógica estaba esencialmente separados. Los procedimientos se cargan y manipulan datos dentro de su lógica, pero el el procedimiento en sí no contiene los datos de ninguna manera significativa. La programación orientada a objetos (OO) fusionó atributos y comportamientos en un solo objeto. Entonces, por ejemplo, un objeto empleado podría representar la estructura de los registros de los empleados, así como las operaciones que se pueden realizar en aquellos registros: cambio de salario, promoción, jubilación, etc. Para los propósitos de las bases de datos, existen ciertas características elevantes:

• Complejidad: las OODBMS tienen la capacidad de representar la compleja estructura interna (del objeto) con una complejidad multinivel.

• Encapsulación: una clase de objeto encapsula datos y acciones (métodos) que se pueden realizar en esos datos. De hecho, un objeto puede restringir el acceso directo a los datos subyacentes, que requieren que las modificaciones a los datos sean posibles solo a través de un método de objetos. Por ejemplo, una clase de empleado puede incluir un método para recuperar salario y otro método para modificar el salario. El método de modificación salarial podría incluir restricciones sobre salarios mínimos y máximos, y la clase podría permitir que no se manipule el salario fuera de estos métodos.

• Herencia: las clases de objetos pueden heredar las características de una clase padre. Los integrantes de la clase de empleado puede heredar todas las propiedades de una clase de personas (fecha de nacimiento, nombre, etc.) al agregar propiedades y métodos, como salario, fecha de empleado, etc.

• Persistencia: se permite crear objetos persistentes (el objeto permanece en la memoria incluso después de la ejecución). Esta característica puede resolver automáticamente el problema de recuperación y concurrencia.

Impedancia no coincidente

Uno de los términos comunmente utilizado es el de impedancia no coincidente. Este término se usa para describir el problema de una aplicación orientada a objetos (OO) que aloja sus datos en bases de datos relacionales heredadas (RDBMS). Los programadores de C ++ lo han tratado durante años, y ahora es un problema familiar para Java y otros programadores de OO. La falta de coincidencia de impedancia surge de la falta de afinidad inherente entre el objeto y los modelos relacionales. Los problemas asociados con la falta de coincidencia de impedancias incluyen jerarquías de clases vinculantes a esquemas relacionales (asignación de clases de objetos a tablas relacionales), generación de ID, concurrencia, así como otros problemas que se describen a continuación.

El impacto de estos problemas está vinculado específicamente a la combinación de la aplicación OO y el esquema relacional. Pero las ramificaciones son claras en términos de tiempo de lanzamiento al mercado, costos de diseño, desarrollo y garantía de calidad, mantenimiento y extensibilidad del código comprometido, y el tamaño y la topología del hardware requerido para garantizar la respuesta esperada y los tiempos de producción.

Dada la prevalencia creciente del desajuste de impedancia de las aplicaciones OO y las RDBMS, como así también la falta de correspondencia entre las aplicaciones basadas en SQL y las bases de datos de objetos (OOBBMS), resulta oportuno y útil mostrar una posible solución a estos problemas.

SQL embebido

Cuando se quieren almacenar datos de aplicaciones que utilizan programas orientados a objetos a RDBMS se deben aplicar operaciones de lógica de negocios desde fuentes externas a la tabla, por ejemplo, mediante el uso de SQL incorporado o procedimientos precodificados. Para construir una aplicación eficaz y eficiente en el modelo relacional, el desarrollador debe tener un conocimiento exhaustivo de las tablas, las relaciones entre ellas y de estos componentes lógicos externos.

Para simplificar estos procedimientos se utiliza SQL embebido. Este tipo de consultas proporciona acceso a la base de datos desde un lenguaje de programación de propósito general requerido en los casos siguientes:

Cuando no todas las consultas se pueden expresar en SQL, por ejemplo, las consultas recursivas no se pueden escribir en SQL.

Existan acciones no declarativas: por ejemplo, imprimir informes no se puede hacer desde SQL.

El lenguaje de propósito general en el que se incrusta SQL es llamado lenguaje de host. En este lenguaje los comandos de SQL son embebidos y los datos son intercambiados entre el lenguaje de host y DBMS usando cursores. De esta forma, una consulta SQL pasa del lenguaje del host a una DBMS que calcula el conjunto de respuestas y un cursor se puede ver como un puntero en el conjunto de respuestas. Una vez obtenido este conjunto, la DBMS devuelve el cursor al lenguaje de programación, que puede usar el cursor para obtener un registro a la vez de acceso a la respuesta materializada.

```
#:dname = “juguete”;
#raise  = 0.1;
#EXEC SQL  SELECT dnum into :dnum
#                  FROM  Tienda
#           WHERE dname= :dname;
#EXEC SQL DECLARE Emp CURSOR FOR
#               SELECT *  FROM Empleado
#               WHERE dno = :dnum
#                FOR UPDATE
#EXEC SQL OPEN Emp;
#EXEC SQL FETCH Emp INTO :E.ssn, :E.dno, :E.name, :E.sal;
#while (SQLCODE == 0) {
#   EXEC SQL UPDATE WHERE CURRENT OF CURSOR
#       SET sal = sal * (1 + ::raise);
#      EXEC SQL FETCH Emp INTO :E.ssn, :E.dno, :E.name, :E.sal;
#  }
#EXEC SQL CLOSE CURSOR Emp

#/* SQL embebido en C para leer la lista de empleados que trabajan en la tienda
de juguetes y darles un aumento del 10% */
```

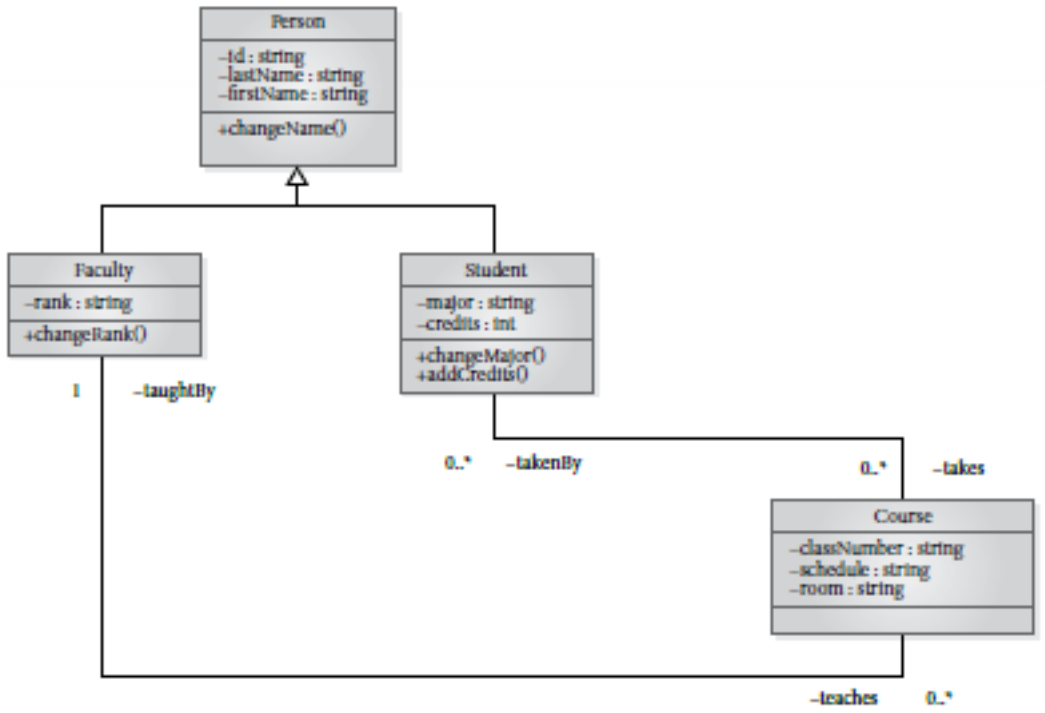
2. ¿En qué consiste el modelo orientado a objetos?

El modelo relacional tradicional, en el que los datos se representan como tablas que tienen filas de atributos de un solo valor, tiene una capacidad limitada para representar los datos complejos y las relaciones necesarias para aplicaciones avanzadas. Tal como se encuentra que el modelo E-R (modelo entity–relationship, que describe cosas de interés interrelacionadas en el dominio específico de conocimiento) carece de la capacidad de representar un modelo de concepto para aplicaciones avanzadas tales como ingeniería de software, redes sociales, y sistemas de información geográfica; el modelo relacional tradicional carece de las estructuras de datos para soportar los requisitos de información para estas aplicaciones. El modelo orientado a objetos es principalmente un modelo semántico que se basa en la noción de un objeto, que, como una entidad, es una representación de alguna persona, lugar, evento o concepto en el mundo real que deseamos representar en la base de datos. Si bien las entidades solo tienen atributos, los objetos tienen un estado y un comportamiento.

El estado de un objeto está determinado por los valores de sus atributos (variables de instancia). El comportamiento es el conjunto de operaciones (métodos) definidos para el objeto. Una clase es similar a un conjunto de entidades y consiste en el conjunto de objetos que tienen la misma estructura y comportamiento.

El modelo orientado a objetos usa encapsulación, incorporando datos y funciones en una unidad donde están protegidos de modificaciones desde el exterior. Las clases se pueden agrupar en jerarquías, los diagramas de clases de lenguaje de modelado unificado (UML) son usualmente utilizado para representar modelos de datos orientados a objetos (OO).

La figura siguiente muestra un ejemplo de un modelo de datos OO simple.



Databases Illuminated, 2017

En el diagrama se ve una clase Person con atributos id, firstName y lastName, y un método ChangeName. La clase Student y la clase Faculty son subclases de Person. Las subclases heredan atributos y métodos del padre, por lo que en este ejemplo, cada objeto de Student y cada objeto de Faculty tiene los atributos y métodos de Person. También pueden tener sus propios atributos y métodos. Por ejemplo, la clase de Student puede tener su propio atributo adicional major y un método ChangeMajor. Las clases también pueden participar en relaciones de varios tipos. Por ejemplo, la clase de Student se puede relacionar con la clase de Course. Cada objeto en una base de datos debe tener un identificador de objeto único que funcione como una clave primaria permanente, pero no toma su valor de ninguno de los atributos del objeto. Una diferencia importante entre los objetos del programa y los objetos de la base de datos es la persistencia. A diferencia de un objeto de programa que existe solo mientras el programa se está ejecutando, un objeto de base de datos permanece en existencia después que se completa la ejecución de un programa de aplicación.

Por eso, los sistemas de gestión de bases de datos orientado a objetos permiten al diseñador de la base de datos crear objetos complejos e interrelacionados y darles persistencia. Los proveedores de sistemas de bases de datos relacionales han respondido al desafío de los sistemas orientados a objetos mediante la extensión del modelo relacional y la incorporación de conceptos orientados a objetos mientras se mantienen estructuras relacionales subyacentes, creando sistemas híbridos objeto-relacionales (ORDBMS). Sin embargo, hay una sobrecarga considerable involucrada en el mapeo de los objetos utilizado en aplicaciones orientadas a objetos a las estructuras de tabla relacional que el modelo debe soportar. Debido a que los proveedores de bases de datos relacionales se han dirigido a algunas de las limitaciones de las bases de datos relacionales, y porque los usuarios que tienen una gran inversión en sistemas relacionales no están dispuestos a migrar a un paradigma completamente nuevo, la popularidad de las bases de datos orientadas a objetos ha sido limitado, como se mencionó anteriormente. Sin embargo, un sistema de gestión de bases de datos estrictamente orientada a objetos como Caché, Objectivity, GemStone, ObjectStore, y Versant, están en uso, particularmente en entornos que tienen esos requerimientos de datos.

Evolución de las ODBMS

Las bases de datos orientadas a objetos han evolucionado a lo largo de dos caminos diferentes:

Hacia lenguajes de programación orientados a objetos persistentes: (ODBMS puros). Como es el caso en el cual se comienza con un lenguaje OO (por ejemplo, C ++, Java, SMALLTALK) que tiene un sistema de tipo enriquecido en el cual se agrega persistencia a los objetos en el lenguaje de programación y los objetos son persistentemente almacenados en las bases de datos.

Sistemas de gestión de bases de datos relacionales de objetos (sistemas SQL3). En este caso se extienden los DBMS relacionales con el sistema de tipos enriquecidos y las funciones definidas por el usuario. Esto proporciona una ruta conveniente para que los usuarios de DBMS relacionales migren a la tecnología OO. Todos los proveedores principales (por ejemplo, Informix, Oracle) dan funciones de soporte de SQL3.

Grupo de administración de la base de datos de objetos (ODMG)

El Object Management Group (OMG) fue fundado en 1989 por un grupo de líderes de la industria informática con el propósito de desarrollar estándares para un modelo de objeto común que sería portátil e interoperable. El grupo sin fines de lucro se ha expandido a cientos de miembros, incluyendo tanto compañías como usuarios de computadoras. Un foco del grupo ha estado en desarrollo de estándares para el modelo de objetos y los servicios conectados con él, y otro en estándares de modelado. Estos estándares permiten que los clientes de ODBMS escriban aplicaciones portátiles y los mismos incluyen:

Modelo de objetos

Idiomas de especificación de objeto: entre los que se incluyen

Lenguaje de definición de objetos (ODL) para la definición de esquema

Formato de intercambio de objetos (OIF) para intercambiar objetos entre bases de datos

Lenguaje de consulta de objetos (OQL):

Lenguaje declarativo para consultar y actualizar objetos de base de datos

Enlaces de lenguaje (C ++, Java, Smalltalk): que vincula el lenguaje del programa con el lenguaje de la base de datos e incluye,

Lenguaje de manipulación de objetos

Mecanismos para invocar OQL desde el lenguaje

Procedimientos para operar en bases de datos y transacciones

Detalles del modelo de objetos

Dentro del modelo de datos (el cual ya se mencionó anteriormente) se pueden encontrar diferentes categorías para caracterizar a los objetos.

Clases

Objetos similares con el mismo conjunto de propiedades y que describen conceptos similares del mundo real se recopilan en una clase. Cada clase se puede definir mediante una declaración de ODL. Por ejemplo:

```
# interface Empleado {
#   attribute string nombre;
#   attribute integer salario;
#   attribute date  dia-de-nacimiento;
#   attribute integer empid;
#   relationship  Projects trabaja-para
#             inverse Projects::team;
#   age-type    edad();
#   }

#   interface Projects{
#attribute  string nombre;
#attribute integer projid;
#relationship Empleado team
#   inverse Empleado trabaja-para;
#int num-de-empleados();
#}
```

En este caso se define una clase Empleado y se listan todos los atributos comunes que poseen los integrantes de esa clase. Además se la relaciona con otra clase Projects que indica qué integrante de la clase Empleado trabaja para algún integrante de la clase Projects.

También, para cada clase de ODL, se puede declarar una extensión. Esta extensión es el conjunto actual de objetos que pertenecen a la clase. Esta noción similar a la relación que se establece en las bases de datos del modelo relacional. Aunque es necesario tener en cuenta que cuando se realizan consultas en OQL, en este caso, siempre se refieren a la extensión de una clase y no a la clase directamente. En el ejemplo que se está mostrando, una extensión se declara de la siguiente forma:

```
#interface Empleado  (extent Emp-set)
#   { attribute string nombre;
#   attribute integer salario;
#   attribute date  dia-de-nacimiento;
#   attribute integer empid;
#   relationship  Projects trabaja-para
```

```
#             inverse Projects::team;
#   age-type    edad(); }
```

Subclases y herencia

Una clase también puede declararse como una subclase de otra clase. En este caso, la subclase definida hereda todas las propiedades de la superclase:

atributos

relaciones

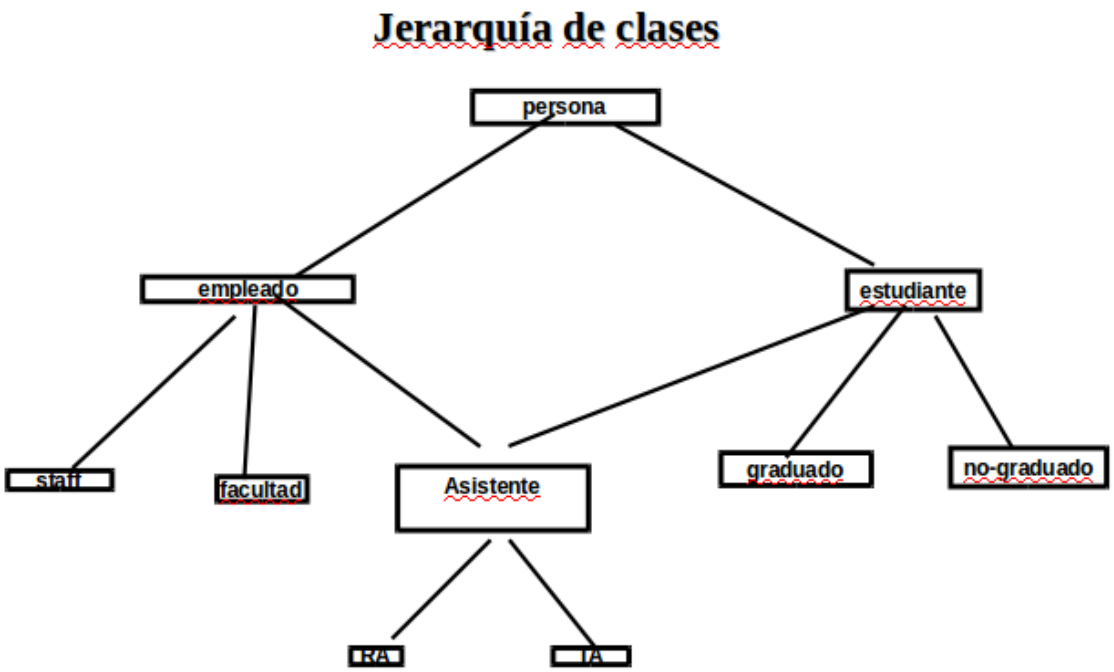
métodos

```
#                                   interface    Empleado-casado:    Empleado    {
#                                                      string          nombre-esposa;
#  }
```

Cuando se define una subclase, se da la propiedad de sustituibilidad, por la cual cualquier método de superclase se puede invocar sobre los objetos de cualquier subclase. Esto implica una reutilización del código.

Jerarquía

Dentro de las clases definidas se establece una especie de jerárquia, en la cual existe una clase principal y otras derivadas y/o relacionadas. En la siguiente figura se muestra un ejemplo en el cual existe una clase Persona y dos subclases derivada de esta, empleado y estudiante, que a su vez tienen otras subclases. Es importante notar qua la subclase asistente vincula las subclases empleado y estudiante .



Herencia Múltiple

En la figura anterior se ve que la subclase asistente tiene más de una superclase. Una clase de este tipo hereda propiedades de cada una de sus superclases. Aunque es importante ver que existe una potencial ambigüedad: variable con el mismo nombre heredado de dos superclases. Para solucionar esta ambigüedad se pueden seguir alguna de las siguientes tres alternativas:

marcar con un Flag y catalogarlo como error

cambiar el nombre de la variable

elegir alguna de las variables

Identidad del objeto

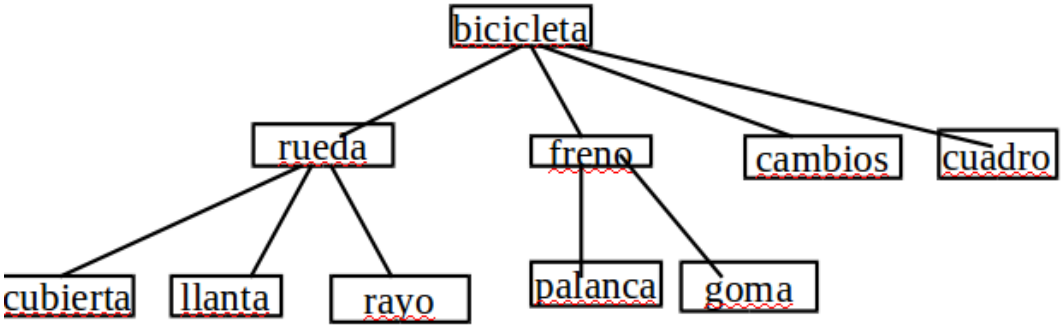
Cada objeto tiene una identidad que mantiene incluso si algunos o todos sus atributos cambian. Este concepto de identidad de objeto es una noción de identidad más sólida que en los DBMS relacionales, ya que en las DBMS está basada en el valor (clave principal),

mientras que en las ODBMS es un concepto que se haya integrado en el modelo de datos y no requiere un identificador especificado por el usuario.

El identificador de objeto (OID) es una noción similar al de puntero en el lenguaje de programación y se puede almacenar como atributo en el objeto para referirse a otro objeto. Las referencias a otros objetos a través de sus OID pueden dar como resultado una jerarquía de contención que difiere de la jerarquía de clases.

Jerarquía de contención

En la siguiente figura se muestra un ejemplo de la jerarquía de contención que puede darse en los sistemas orientados a objetos. Cada enlace en la jerarquía de contención debe leerse como "es parte de", ya que, por ejemplo en la figura, la clase bicicleta contiene todas las subclases (partes de una bicicleta) que salen de la misma.



Persistencia

Este concepto ya se mencionó anteriormente, pero entrando en más detalle es importante recalcar que los objetos creados pueden tener diferentes tiempos de vida:

transitorios: la memoria asignada es administrada por el sistema de tiempo de ejecución del lenguaje de programación. Por ejemplo, las variables locales en los procedimientos tienen una duración de ejecución de un procedimiento. También las variables globales tienen una duración de ejecución de un programa.

persistente: la memoria asignada y almacenada es administrada por el sistema de tiempo de ejecución de la ODBMS.

Todas las clases se deben declarar de alguna de las dos formas: capaces de persistencia o transitorias. También los diferentes lenguajes tienen diferentes mecanismos para hacer que los objetos sean persistentes:

tiempo de creación: objeto declarado persistente en el momento de la creación (por ejemplo, en enlace de C ++) (la clase debe ser persistente).

persistencia por accesibilidad: el objeto es persistente si se puede alcanzar desde un objeto persistente (p. ej., en enlace Java) (la clase debe ser compatible con la persistencia).

Lenguaje de consulta de objetos (OQL)

Los objetos persistentes se almacenan en la base de datos y se accede desde el lenguaje de programación. Mientras que las clases declaradas en ODL asignadas al sistema de tipo de lenguaje de programación lo hacen mediante el enlace ODL. Para poder realizar las consultas se utiliza un lenguaje de programación único para aplicaciones y gestión de datos. Este lenguaje evita tener que traducir datos hacia y desde el lenguaje de programación de aplicaciones y DBMS (eliminando el problema que se presentaba en la primer figura mostrada). Entre las ventajas principales que presenta tener este lenguaje se encuentran:

una implementación eficiente con menos código.

el programador no necesita escribir un código explícito para obtener datos hacia y desde la base de datos.

los objetos persistentes para el programador se ven exactamente igual que los objetos transitorios.

el sistema trae automáticamente los objetos hacia y desde la memoria al dispositivo de almacenamiento (punteo swizzling).

Sin embargo también existen algunas desventajas en la implementación de este tipo de lenguaje:

baja protección dado que los objetos persistentes manipulados directamente desde las aplicaciones, hay más cambios que los errores en las aplicaciones pueden violar la integridad de los datos.

interfaz no declarativa que hace que sea difícil de optimizar las consultas o que haya preguntas difíciles de expresar.

Sin embargo la mayoría de los ODBMS ofrecen un lenguaje de consulta declarativa del tipo de lenguaje de consulta de objetos (OQL) para superar el problema. Este OQL es un lenguaje de consulta estándar para bases de datos orientadas a objetos modelado a partir de SQL. OQL fue creado por ODMG como un intento de extender los lenguajes como C++ y SQL-like. Pero, debido a su complejidad, ningún creador de software ha implementado completamente OQL. Aún así, OQL ha influenciado el diseño de algunos lenguajes de consulta nuevos como JDOQL y EJBQL, pero estos no pueden ser considerados como versiones de OQL.

Volviendo a la explicación de OQL, el mismo tiene una definición de esquema similar a ODL. Además OQL es muy similar a SQL y se puede optimizar de manera efectiva y se puede invocar desde el lenguaje de programación ODBMS. Los objetos pueden manipularse tanto en OQL como en el lenguaje de programación sin transferir valores explícitamente entre los dos idiomas. La incorporación de OQL mantiene la simplicidad de la interfaz del lenguaje de programación ODBMS y, a la vez, proporciona acceso declarativo

Tiene tipos de datos básicos como strings, ints, reals, etc., además de nombres de clases. También tiene tipos de constructores como Struct para estructuras; y tipos de colecciones como set, bag, list, array.

Un ejemplo de OQL es como sigue:

En esta declaración se define la clase Empleado y la relación con la clase Projects

```
#interface Empleado {
#   attribute string nombre;
#   relationship
#       setof(Projects) trabaja-para
#        inverse Projects::team;
#   }
```

En esta declaración se define la clase Projects y su relación inversa con la clase Empleado,

```
#   Interface Projects{
#attribute  string nombre;
#relationship setof(Empleado) team
#   inverse Empleado trabaja-para;
#int num-de-empleados();
#}
```

Mediante la siguiente declaración se encuentra el número de empleados que trabajan en el proyecto "sharad"

```
#Select num-de-empleados()
#From Empleado e, e.trabaja-para
#where nombre = “sharad”
```

Migración de RDBMS hacia tecnologías OO

El SQL3 es una extensión del estándar de bases de datos SQL92 que incluye soporte para la administración de bases de datos orientadas a objetos. En particular, SQL3 es un estándar para productos y no un producto en s, ya que se encuentra fuera de la tradición de administración de bases de datos, pero está dentro de la de pensamiento de objetos. La meta de los comités que trabajan en el SQL3 ha sido el de describir un estándar que logre la compatibilidad con el SQL92. Esto significa que todas las características y funciones del SQL92 también funcionen con el SQL3.

Sin embargo no se debe considerar a SQL3 como una nueva facilidad de bases de datos orientadas a objetos, sino que es una facilidad de base de datos relacional con características de objetos agregadas. De esta forma el estándar SQL3 incorpora conceptos OO en el modelo relacional, por ejemplo una fila en una tabla es considerada como un objeto. También permite que un tipo sea declarado para tuplas (similar a la clase en ODBMS). Las relaciones son una colección de tuplas de un tipo de fila (similar a la extensión en ODBMS). Además, las filas en una relación pueden referirse entre sí utilizando un tipo de referencia (similar a la identidad del objeto en ODBMS). Una referencia puede ser desreferenciada para navegar entre las tablas. También los atributos en una relación pueden pertenecer a tipos de datos abstractos y los métodos y funciones

(expresados en SQL así como en el lenguaje de programación de host) pueden asociarse con tipos de datos abstractos.

En el SQL3 se incorporan tres (3) grupos de nuevas ideas, para que sea completo desde el punto de vista computacional:

Soporte para tipos de datos abstractos (ADT). Es una estructura que define el usuario, la cual es equivalente a un objeto OOP.

Mejoras a las definiciones de las tablas. El SQL3 amplía la definición de las tablas de muchas maneras:

Las tablas SLQ3 tiene un identificador de renglones, el cual es un identificador único de cada renglón de una tabla. Éste es igual a una llave sustituta.

Definición de tres tipos de tablas: CONJUNTO (no tiene renglones duplicados), MULTICONJUNTO (puede tener renglones duplicados) y LISTA (tiene un orden definido por una o más columnas).

Extensiones a las construcciones de lenguaje. Se añaden instrucciones de lenguaje que hacen posible desarrollar una lógica de un renglón a la vez dentro del mismo SQL, lo que lo hace cada vez más parecido a un lenguaje de programación tradicional.

Por ejemplo una declaración, como la que se hizo anteriormente, en SQL3 sería de la siguiente forma:

```
#  CREATE ROW TYPE Empleado-type {
#nombre CHAR(30)
#trabaja-para REF(Projects-type)
#}
#CREATE ROW TYPE Projects-type {
#nombre CHAR(30)
#team setof(REF(Empleado-type))
#}
#CREATE TABLE Emp OF TYPE Empleado-type
#CREATE TABLE Project of TYPE Project-type

#Select trabaja-para --> nombre
#From Emp
#Where nombre = 'sharad'
```

De la misma forma que antes esta consulta devuelve el nombre de los empleados que trabajan en el proyecto "sharad".

Principales OODBMS

Actualmente existen varias opciones de OODBMS, tanto propietarias como de código abierto. Entre las primeras se pueden nombrar:

Object Store

O2

Gemstone

Versant

Ontos

DB/Explorer ODBMS

Objectivity/DB

EyeDB

Mientras que entre las de código abierto se encuentran:

Ozone

Zope

FramerD

XL2

Bibliografía utilizada y sugerida

Libros y otros manuscritos

Catherine M. Ricardo, Susan D. Urban, Databases Illuminated, USA New York, Jones & Bartlett Learning, 2017.

Guy Harrison, Next Generation Databases, USA New York, APRESS 2016.

Lo que vimos:

En esta Unidad vimos los conceptos de bases de datos orientadas a objetos. Aprovechamos, también, para repasar clase, jerarquía y herencia, que son las piedras angulares del modelo de objetos, así como también discutir las implementaciones de la persistencia de los objetos.

Lo que viene:

En la próxima Unidad vamos a ver de qué se tratan los avances en Big Data y las bases de datos distribuidas, sus ventajas y desventajas, así como a generarnos un concepto de cuándo conviene usarlas.