

El paradigma de la Orientación a Objetos en SQL:1999 y Oracle 8i.

Miguel Romero V.
Universidad del Bío-Bío
mromero@ubiobio.cl

Gilberto Gutiérrez R.
Universidad del Bío Bío
ggutier@ubiobio.cl

Resumen

Al desarrollar aplicaciones bajo el paradigma de Orientación a Objetos (OO), nos encontramos con el problema de la *persistencia*, es decir, permitir que los objetos "sobrevivan" al programa que los creó. Existen varias estrategias para solucionar esto, pasando por aquellas que se apoyan en simples archivos, hasta aquellas que utilizan Sistemas de Administración de Bases de Datos Orientadas a Objetos (SABDOO). Por otra parte, el modelo relacional, definido en el estándar SQL-92, no entrega soporte para el almacenamiento de Objetos, sin embargo, en el actual (SQL:1999) se proponen formas que apoyan el paradigma OO. En el presente trabajo se analizan las capacidades y limitaciones del estándar SQL:1999 y del Sistema de Administración de Bases de Datos (SABD) Oracle 8i, para soportar el paradigma OO.

Como resultado de este trabajo, podemos decir que el estándar SQL:1999 implementa alrededor de un 78% el paradigma OO, mientras que el SABD Oracle 8i lo implementa en aproximadamente un 70%. La limitación más importante que tiene Oracle 8i frente a SQL:1999 es la falta de herencia y el deficiente soporte para la evolución de esquemas.

Palabras claves: Orientación a Objetos, Bases de datos, persistencia.

1 Introducción

La OO, desde sus orígenes ha sido un paradigma muy prometedor para el desarrollo de software. En estos últimos años, hemos visto aparecer varios productos, estándares y herramientas que han marcado un hito en la historia de la OO. Lenguajes como JAVA, C++, etc.; herramientas para el análisis y diseño (UML), entre otros. El paradigma OO también se ha extendido hacia el área de las bases de datos, dando origen a los Sistemas de Administración de Bases de Datos Orientadas a Objetos (SABDOO), las cuales nacen para solucionar el problema de la persistencia de los objetos, en forma eficiente, requisito indispensable para aplicaciones como: CASE (Ingeniería de software asistida por computador), CAD/CAM (Diseño / fabricación asistidos por computador), SIG (Sistemas de información geográficos), multimedia, gestión de redes; donde los sistemas relacionales han sido incapaces de satisfacer los requerimientos de dichas aplicaciones (Piattini M., 1999).

Asociados a las bases de datos OO, existen dos estándares: SQL:1999 y ODMG v2.0 (Piattini M., 1995), el primero "de Jure" (y de facto) y el segundo "de facto". Estos dos estándares siguen tendencias distintas, pero están convergiendo. Estas tendencias son:

1. Extender los SABD relacionales para que soporten OO(SQL:1999);
2. Confeccionar un SABDOO basado en un modelo de objetos "puro" que no extienda sistemas relacionales (ODMG-93).

El objetivo de este trabajo es evaluar las capacidades y limitaciones del primer enfoque (SQL:1999) y del producto Oracle 8i para incorporar el paradigma OO. El resto del artículo está organizado de la siguiente manera: en la sección 2 se detallan y explican los criterios utilizados para realizar la evaluación. La sección 3 analiza en detalle cada uno de los criterios para el estándar (SQL:1999). La sección 4, por su parte, analiza los criterios para Oracle 8i. La sección 5 resume la evaluación de las capacidades y limitaciones de SQL:1999 y Oracle 8i para soportar el paradigma OO. Finalmente en la sección 6 se entregan las conclusiones.

2 Criterios utilizados en la evaluación

Debido a la necesidad de almacenar objetos, los SABDOO deben satisfacer un conjunto de requisitos que están estrechamente ligados a los conceptos básicos del paradigma OO.

1. **Definición de clases y extensión de los tipos de datos primitivos.** La definición de clases y su jerarquía debe formar parte del catálogo, además es deseable que estas clases pasen a formar parte del sistema de tipos del SABDOO.
2. **Lenguajes de programación computacionalmente completos.** Los SABDOO deben contar con leguajes de programación o interfaces con lenguajes para implementar los métodos de los objetos.
3. **Objetos complejos.** Para poder implementar este tipo de objetos es necesario contar con mecanismos que permitan tener atributos multivaluados (como listas, conjuntos, arreglos, etc.), identificadores únicos y referencias a otros objetos.
4. **Herencia, Encapsulamiento y Polimorfismo.** Estos son pilares fundamentales de la OO, por lo que deben considerarse.
5. **Manejar versiones de objetos y configuraciones.** Cuando cambiamos los valores de los atributos, estos son reemplazados por los nuevos. En algunas aplicaciones es necesario conocer los valores históricos que han asumido los objetos, esto se logra manejando distintas versiones del objeto. Una configuración establece enlaces entre una versión de un objeto compuesto y las correspondientes versiones de sus objetos componentes.
6. **Evolución de esquemas y de instancias.** Los cambios en este tipo de sistemas son la regla más que una excepción. Para ello es necesario proveer de mecanismos que permitan modificar la definición de las clases y sus jerarquías (Evolución de esquemas), además de soportar la migración de instancias entre clases.
7. **Transacciones de larga duración.** Como los objetos y sus atributos pueden ser bastante grandes (Ej.: un video en una base de datos multimedia), las transacciones pueden ser bastante largas. Por eso se deben reconsiderar los mecanismos de recuperación ante un fallo y el control de la consistencia.
8. **Mecanismos de autorización basados en la noción de objetos.** En los sistemas tradicionales, los mecanismos de autorización están asociados a una relación o a una vista. Estos mecanismos no son del todo adecuados para un modelo orientado a objeto, el que necesita mecanismos asociados a la noción de objetos, considerando la herencia, versiones, objetos compuestos, etc.
9. **Interactuar con sistemas existentes.** Es necesario que estos nuevos sistemas sean capaces de acceder y manipular datos en sistemas existentes como bases de datos relacionales, debido al impacto que tiene en una organización la migración de datos.

Para evaluar objetivamente el paradigma OO en SQL-1999 y Oracle 8i, definimos un conjunto de criterios basados en las características del paradigma OO y en los requerimientos para las bases de datos OO. Con el propósito de precisar la evaluación algunos criterios fueron desagregados (figura 8). El porcentaje asignado a cada criterio se realizó de la siguiente manera:

- Si la característica está presente, se asignó 100%
- Si la característica no está presente, pero se puede simular, se asignó 60%
- Si la característica no está presente, se asignó 0%

Los criterios utilizados para la evaluación se encuentran en la tabla 8

3 El paradigma OO en SQL:1999

Tipo, abstracción y clases. El estándar unifica estos conceptos bajo el nombre de tipo estructurado. Un tipo estructurado encapsula atributos y métodos en una única entidad lógica, pero físicamente separados. Los atributos de un objeto persistente se almacenan en una tabla, y los métodos, junto con los demás procedimientos almacenados. Este nuevo tipo, pasará a ser uno más del sistema, pudiendo ser utilizado del mismo modo que un tipo primitivo. La cláusula **Not Final**, permite que el tipo sea utilizado como supertipo de otro (ver herencia y polimorfismo).

Todo tipo estructurado tiene asociado un correspondiente **tipo de referencia**. Este tipo almacena una secuencia de bytes, que indica el lugar donde está almacenado un objeto del tipo estructurado. Un

```
CREATE TYPE punto as (X integer, Y integer ) Instantiable not Final
    Method moverA(X Integer, Y Integer), Method distanciaA(P Punto) return float,
    Static Method crear(X Integer, Y Integer) return Punto
```

Figura 1: Definición de un tipo de datos Punto

objeto de tipo referencia puede ser utilizado en los mismos lugares donde se puede utilizar un objeto del tipo estructurado. Con este tipo de datos, podemos crear redes de objetos. Por ejemplo, podemos crear una lista enlazada, un árbol binario, etc.

Al definir un tipo estructurado, podemos indicar la manera en que el sistema construirá sus referencias. Para eso, debemos escoger entre tres formas de representación:

1. **User Generated.** La referencia se basa en un tipo primitivo, cuyo valor será proporcionado por el usuario.
2. **System Generated.** La referencia es generada automáticamente por el sistema. Este es el valor por defecto.
3. **Derived.** La referencia se basa en uno o más atributos. Es semejante a una clave primaria.

Al definir los métodos en la creación de un tipo estructurado, sólo se declara su interfaz (función prototipo o encabezado). Para definir su implementación o cuerpo, se utiliza la sentencia **Create Method**. El código que implementa al método, puede ser escrito en SQL (usando las sentencias computacionalmente completas del SQL/PSM (ANSI X3H2-99-269, 1999) o en cualquiera de los lenguajes de programación más tradicionales, incluyendo JAVA (Piattini M., 1999). Los métodos pueden ser:

- **Métodos de instancia.** Son aquellos que serán invocados a partir de un objeto concreto.
- **Métodos estáticos.** Son aquellos que serán invocados a partir de un tipo estructurado y no de un objeto.
- **Constructores.** Cuando se crea un tipo estructurado con la cláusula **Instantiable** el sistema define un método constructor por defecto (ANSI X3H2-99-462,1999). Este método, es una función cuyo nombre coincide con el del tipo estructurado, no posee argumentos, y el valor de retorno es una instancia del tipo al que pertenece, donde el valor de cada atributo corresponde al valor por defecto, especificado al momento de su creación. El estándar SQL:1999, no permite los constructores definidos por el usuario, pero el borrador de trabajo de la futura versión del estándar (SQL:200n), ya lo incorpora. Existe un operador llamado **New** que permite invocar a un método de instancia que haga las veces de constructor. El método debe tener el mismo nombre que su tipo estructurado, y puede ser sobrecargado.

En SQL:1999 es posible definir colecciones. Una **colección** es una estructura que permite almacenar cero o más elementos de distinto o del mismo tipo de datos(ANSI X3H2-99-463,1999). El término colección es genérico, y comprende varios tipos de colecciones como: arreglos, conjuntos, árboles, etc. El estándar SQL:1999 solamente soporta los arreglos.

Encapsulamiento. El encapsulamiento abarca tres aspectos:

1. Agrupar atributos y métodos en una sola entidad.
2. Distinguir entre interfaz(pública) e implementación (privada)
3. Asignar niveles de acceso individual a los atributos y métodos.

El primer aspecto, es cubierto con el predicado **Create Type**. Esta es la parte pública del método. Con el predicado **Create Method** es cubierto el segundo punto. Gracias a esta división (pública y privada) podemos realizar cambios a la implementación de los métodos, sin que estos afecten a las aplicaciones que los utilizan. El tercer punto es cubierto medianamente las funciones observadoras y mutadoras definidas por el sistema, puesto que estas impiden que los usuarios accedan a los atributos directamente. Lamentablemente, no es posible sobrescribir dichas funciones. Lo que falta para que el soporte al encapsulamiento sea completo, es la posibilidad de definir niveles de acceso a los atributos y métodos, como **public**, **protected** y **privada** de JAVA. Sin embargo, el modelo de seguridad de SQL-1999, permite obtener una funcionalidad similar, pero no completa de lo anterior.

Modularidad. SQL:1999 cuenta con la noción de módulos, pero estos sólo proveen modularidad para las funciones y procedimientos almacenados, pero no para tipos. Sin embargo, podemos agrupar tipos basándonos en los esquemas de SQL, logrando así una funcionalidad similar.

Jerarquía de clases y tipos.

Asociación. En el estándar es posible mantener la relación de **asociación** entre entidades de manera similar a la establecida en el modelo entidad relación. Por eso podemos utilizar las mismas estrategias para su implementación, pero con la ventaja de la existencia de las referencias de objeto, que son más eficientes que el manejo de clave foránea. Es posible implementar asociones **uno a uno** y **muchos a muchos**.

Agregación. La implementación de la agregación también es posible en el estándar. Consideremos por ejemplo la relación de agregación entre una carrera y sus asignaturas, donde una carrera puede tener muchas asignaturas y una asignatura puede pertenecer a varias carreras. La implementación sería como se muestra en la figura 2.

```
CREATE TYPE carrera AS( codigo CHAR(12), nombre CHAR(40),
  Asignaturas REF(asignatura) ARR [100])INSTANTIABLE NOT FINAL

CREATE TYPE asignatura AS( codigo CHAR(12), nombre CHAR(40),
  creditos NUMBER)INSTANTIABLE NOT FINAL
```

Figura 2: Implementación de la relación de agregación

Si la agregación es **uno a uno** no necesitaremos un arreglo, bastará con una referencia a la parte.

Herencia y polimorfismo. SQL:1999 permite la herencia simple de tipos, es decir, un tipo puede poseer un único supertipo. Consideremos un tipo llamado **FiguraGeométrica** (figura 3).

```
CREATE TYPE FiguraGeometrica AS( color_linea CHAR(10),
  color_fondo CHAR(10))NOT FINAL
METHOD area() RETURNS REAL, METHOD perimetro() RETURNS REAL
```

Figura 3: Definición del tipo de datos **FiguraGeometrica**

A partir de este tipo, podemos definir el subtipo **cuadrado**, según se muestra en la figura 4.

```
CREATE TYPE cuadrado UNDER FiguraGeometrica AS (a punto, b punto)
INSTANTIABLE NOT FINAL
OVERRIDING METHOD area() RETURNS REAL,
OVERRIDING METHOD perimetro() RETURNS REAL
```

Figura 4: Definición del subtipo **cuadrado**. *a* y *b* son los puntos de los extremos opuestos

Una característica esencial de la herencia es que los atributos y métodos del supertipo pasan a formar parte del subtipo. Por lo anterior, no se puede definir el nombre de un atributo igual a otro que se haya heredado. Con respecto a los métodos, estos pueden ser redefinidos en el subtipo. Por ejemplo, la implementación del método **area()** del tipo **cuadrado** sería;

```
CREATE OVERRIDING METHOD area() RETURNS REAL FOR cuadrado
BEGIN RETURN (b.X - a.X) * (b.X - a.X); END;
```

La redefinición y la sobrecarga de métodos son la base del polimorfismo soportado por el estándar. Otra característica de la herencia es que toda instancia de un subtipo, es una instancia de su supertipo directo y también de los indirectos.

Concurrencia. Como las versiones anteriores, SQL:1999 soporta la concurrencia de usuarios. Desde el punto de vista de la orientación a objetos, podríamos decir que varios usuarios pueden manipular un mismo objeto a la vez o que varios objetos pueden ser manipulados por varios usuarios simultáneamente. Otro aspecto de la concurrencia es que los métodos pueden ser concurrentes, ya sea multihilo o multiproceso. Si desarrollamos los métodos con SQL/PSM, no podremos contar con esta característica. Pero si lo implementamos como una rutina externa, podremos utilizar algún lenguaje que soporte concurrencia, como JAVA o C.

Persistencia. Para hacer persistir los objetos tenemos dos alternativas:

1. Almacenarlos como columnas en tablas relacionales.
2. Almacenarlos en una tabla especial denominada **Type table**, donde cada fila es un objeto y cada columna un atributo del mismo.

Los objetos almacenados como columnas no pueden ser referenciados ya que no es posible asignarles un identificador (OID). Solamente los objetos almacenados en una **Typed Table** pueden ser referenciados. La figura 5 muestra la definición de una tabla para almacenar objetos en columnas. En esta tabla (**Persona**) en el atributo **domicilio** se almacenarán objetos de tipo **Dirección**.

```
CREATE TABLE persona (rut CHAR(12) PRIMARY KEY, nombre CHAR(40),
    fecha_nacim DATE, domicilio direccion)
```

Figura 5: Definición de tabla para almacenar objetos como columnas

Una **typed table**, es una tabla o vista construida en base a un tipo estructurado definido por el usuario. Para cada atributo del tipo base, es creada una columna con el mismo nombre y el mismo tipo de datos. Para ejemplificarlo consideremos el tipo **Empleado** (figura 6). Para crear una tabla **typed table** llamada **Funcionarios** basada en el tipo **Empleado** necesitamos hacer lo siguiente:

```
create table Funcionarios of Empleado (Ref is oid System Generated)
```

oid, es el nombre de la columna adicional que almacenará el identificador de objeto. El nombre es puesto por el usuario y puede ser cualquier identificador válido de columna. El tipo de datos de **oid** es **REF(Empleado)**, este no puede ser asignado por el usuario. El valor que asumirá la columna **oid**, será asignado al insertar el objeto en la tabla, y no podrá ser modificado posteriormente.

Manejo de Versiones de objetos y configuraciones. No soportado en el estándar, pero existe una especificación llamada SQL/Temporal prevista para el 2003 aproximadamente, que se encargará del manejo de bases de datos históricas, en ella debería estar soportada esta característica.

Evolución de esquemas y de Instancias. La evolución de esquemas se refiere a la capacidad de modificar la estructura de los tipos, agregando, modificando o eliminando métodos o atributos y que estos cambios sean reflejados en todas las estructuras que dependan de él. La evolución de instancia apunta a dos cosas, la primera es que al alterar un tipo, dicho cambio debe verse reflejado en las instancias y la otra es contar con la posibilidad de migrar instancias entre tipos. La evolución de esquemas está completamente soportada mediante la sentencia **ALTER TYPE**. La evolución de instancias no es soportada completamente, debido a las restricciones impuestas al predicado **ALTER TYPE** con respecto a los atributos. Pero utilizando tablas temporales, lo podemos simular. La migración de clases no es soportada directamente, pero podemos sacar una copia idéntica de él e insertarla en otra tabla, y eliminar el original. Esto es posible, gracias a la posibilidad de sobrescribir el OID.

```
CREATE TYPE empleado AS( rut CHAR(12), nombre CHAR(40), sueldo INTEGER ,
    anticipo INTEGER DEFAULT 0, fecha_ing DATE DEFAULT CURRENT_DATE,
    domicilio direccion )INSTANTIABLE NOT FINAL REF IS SYSTEM GENERATED
    METHOD SueldoLiquido() RETURNS INTEGER
```

Figura 6: Definición del tipo de datos empleado

Transacciones y recuperación ante fallos. El estándar implementa un robusto sistema de transacciones, incorporando en esta versión la noción de **SAVEPOINT**. Con respecto a la recuperación ante fallos, es una materia dependiente de las implementaciones del estándar.

Mecanismos de Autorización. Los mecanismos de autorización están basados en las tablas y en los tipos. Se autoriza o restringe el acceso y manipulación de las tablas o tipos, como un todo, pero no se puede restringir el acceso sobre un objeto o una fila.

Compatibilidad con el modelo relacional. El estándar ofrece una alta compatibilidad con el modelo relacional, no sólo por soportarlo directamente, sino por permitir mezclar tablas relacionales y tablas que almacenan objetos. Además podemos definir vistas de objetos sobre tablas relacionales, permitiendo definir una "capa" de orientación a objeto sobre nuestros datos relacionales. Las vistas de objeto también pueden ser creadas jerárquicamente.

4 El paradigma OO en Oracle 8i

Introducción. Oracle fue el primer servidor de bases de datos relacional, y desde entonces ha sido uno de los líderes de la industria, realizando grandes inversiones en investigación y desarrollo, además, participa activamente en la estandarización del lenguaje SQL.

A partir de la versión 8 de Oracle, comenzaron a incorporar características orientadas a objeto, como definición de tipos, creación de tablas y vistas de objeto, etc. En esta sección se revisan y analizan los aspectos más importantes de Oracle 8i relacionados con la tecnología de objetos.

Para analizar el soporte de la orientación a objeto en ORACLE, como lo hicimos con SQL:1999, nos basaremos en los conceptos básicos de la orientación a objeto y en los requerimientos para las bases de datos orientadas a objeto. En lo posible, mantendremos la misma estructura de presentación utilizada al analizar SQL:1999 para facilitar su comparación con Oracle.

Tipos, abstracción y clases. En Oracle es posible la creación de tipos definidos por el usuario por medio de la cláusula **create typed**, que luego de ser creado, pasa a formar parte del sistema de tipos del lenguaje, quedando disponible para todos los usuarios que tengan los privilegios de acceso. Un tipo objeto implementa una abstracción y por ende está compuesto por (Oracle, 1999): **Nombre, Atributos y Métodos**.

Un tipo objeto tiene dos partes: una especificación y un cuerpo. La especificación es la interfaz pública del tipo y en ella se declaran sus atributos y la especificación de sus métodos (sin su implementación). En el cuerpo se especifican e implementan los métodos que aparecieron en la especificación del tipo. El cuerpo es la parte privada del tipo. Es posible declarar métodos estáticos.

Todo tipo objeto debe poseer a lo menos un atributo, y como máximo 1.000. Un atributo consta de un nombre y un tipo de dato, donde el nombre debe ser único dentro del tipo objeto, y el tipo de datos del atributo puede ser otro tipo definido por el usuario o un tipo primitivo, con las excepciones ROWID, UROWID, LONG, LONG ROW, NCLOB, NCHAR y NCHAR2. Entre los tipos de datos primitivos existe uno denominado REF, que permite almacenar una referencia a un tipo objeto. Gracias a él podemos definir un tipo objeto recursivo. Una referencia (REF), almacena un puntero lógico a un objeto, lo que nos permite acceder a él directamente. El manejo de las referencias en Oracle, es semejante al de JAVA. La figura 7 muestra la creación de un tipo en oracle

```
CREATE TYPE empleado AS OBJECT (rut VARCHAR2(10), nombre VARCHAR2(30),  
fecha_ingreso DATE, salario NUMBER );
```

Figura 7: Creación de un tipo en Oracle 8i

Los métodos no son más que funciones o procedimientos almacenados que se asocian a un tipo específico, y no pueden ser invocados independientemente. Existen cuatro categorías para los métodos estas son: **Constructor, miembro, estático y comparación**.

- **Constructor** Todos los tipos objeto definidos por el usuario poseen un método constructor generado por el sistema, que es una función con el mismo nombre que el tipo. Los parámetros formales de esta función coinciden exactamente con los atributos del tipo, es decir, son declarados en el mismo

orden, con el mismo nombre y del mismo tipo de dato que los atributos. El valor de retorno del constructor es un nuevo objeto del mismo tipo al que pertenece el constructor, donde los valores de los atributos fueron inicializados con los valores pasados por parámetro al momento de llamar al constructor. Como el método constructor es una función, sólo puede ser utilizar dentro de una expresión en la que se espera un objeto de ese tipo.

- **Miembro (member)** Para especificar un método miembro se antepone la palabra reservada MEMBER, y como mencionamos anteriormente, estos son invocados a partir de un objeto.
- **Estático (Static)** Un método estático al igual que un método miembro, puede ser un procedimiento o una función, pero a diferencia de este, no podemos declarar el parámetro SELF, implícita o explícitamente. El uso más común de los métodos estáticos es crear funciones que hagan las veces de constructor, supliendo así la carencia de constructores definidos por el usuario.
- **MAP** Un método de comparación del tipo MAP es una función miembro que no posee argumentos y cuyo valor de retorno es de uno de los siguientes tipos de datos escalares: DATE, NUMBER, VARCHAR2, o un tipo ANSI SQL como CHARACTER o REAL. Un método del tipo MAP deberá entregar la posición relativa que ocuparía el objeto si se ordenara el universo completo de objetos del mismo tipo.

La sobrecarga de métodos es soportada por Oracle 8i, únicamente, para los métodos miembro y estático. No podemos sobrecargar métodos si sus parámetros formales difieren únicamente en el modo (IN, OUT, IN OUT). Tampoco podemos sobrecargar funciones miembro, que difieran únicamente en el tipo de dato del valor de retorno. Existen tres alternativas para implementar los métodos, PL/SQL, JAVA o C.

En oracle 8i también es posible definir **colecciones** de objetos el cual es otro tipo de dato definido por el usuario, que permite almacenar un número indefinido de elementos, todos de un mismo tipo. En Oracle, existen dos tipos de colecciones: arreglos (VARRAYs) y tablas anidadas (nested table).

Encapsulamiento. El encapsulamiento abarca tres aspectos:

1. Agrupar atributos y métodos en una sola entidad
2. Distinguir entre interfaz (pública) e implementación (privada)
3. Asignar niveles de acceso individual a los atributos y métodos.

Los aspectos 1 y 2 están cubiertos en Oracle 8i, a través de los predicados **CREATE TYPE** y **CREATE TYPE BODY**. Lamentablemente, el tercer aspecto no es soportado directamente, pero a través de las vistas podemos restringir el acceso a los atributos.

Modularidad. En Oracle existe una estructura denominada paquete, que brinda modularidad a los programas escritos en PL/SQL. Lamentablemente, no podemos incluir un tipo objeto en estos paquetes. Sin embargo, podemos agrupar los tipos objeto a través de los esquemas de usuarios, logrando así la funcionalidad de un paquete.

Persistencia. En todas las bases de datos relacionales la estructura encargada de persistir o almacenar información es la tabla. En Oracle podemos persistir objetos en columnas de tablas relacionales, o en filas de una tabla de objetos. Una **tabla de objeto**, es una clase especial de tabla que es construida en base a un tipo objeto, donde cada fila de la tabla será una instancia de él, y cada atributo del tipo, será una columna de la tabla.

Jerarquía de Clases.

Asociación. Para implementar las asociaciones utilizamos las referencias de objetos.

Agregación y Composición. La agregación podemos implementarla de la misma manera que una asociación. Pero, como la asociación refleja una relación todo-parte, en la mayoría de los casos, será más conveniente implementarla a través de un tipo colección que almacene referencias de objetos.

Herencia y Polimorfismo. En Oracle 8i la herencia no es soportada, y la única forma de polimorfismo es la sobrecarga de métodos. En todo caso, si necesitamos modelar relaciones de herencia, podemos simularla a través de agregación o composición, y tendremos que dar un soporte manual al polimorfismo. Existen varias estrategias para realizar esto, por ejemplo, si *A* es la clase padre, y *B*, *C* sus subclases, podríamos crear un tipo objeto para cada uno, donde *B* y *C* tendrían un atributo llamado *super*, que sería una referencia a *A*, el que nos permitiría acceder a sus atributos y métodos. El único problema con esta implementación es que no satisface una característica esencial de la herencia, donde una instancia de *B* o *C* también es de *A*, en la práctica, si tenemos una colección que almacena instancias de *A*, una instancia de *B* debería ser posible de almacenar. Una solución a esto es crear el tipo *A* como una composición, es decir, con un atributo para cada subclase, y una variable que indique el tipo particular de la instancia almacenada. Este último enfoque fue empleado por Oracle para implementar el SQL espacial, que es un conjunto de tipos objeto que dan soporte a la implementación de sistemas de información geográficos.

Concurrencia. La concurrencia podemos verla desde dos puntos de vista:

1. Como un método con múltiples hilos de ejecución.
2. Como varios usuarios manipulando el mismo objeto.

La concurrencia desde el primer punto de vista está resuelta, si implementamos el método con JAVA, puesto que este lenguaje implementa un robusto sistema para el manejo y sincronización de múltiples hilos de control.

Desde el segundo punto de vista, la concurrencia también es soportada, permitiendo que varios usuarios utilicen y modifiquen el mismo objeto, permitiendo bloqueos manuales o automáticos que garantizan la consistencia y la integridad del objeto.

Manejo de Versiones de objetos y Configuraciones. Lamentablemente, Oracle no provee mecanismos para manejar versiones y configuraciones de objetos.

Evolución de esquemas y de Instancias. La evolución de esquemas se refiere a la capacidad de modificar la estructura de los tipos, agregando, modificando o eliminando métodos o atributos y que estos cambios sean reflejados en todas las estructuras que dependan de él. La evolución de instancia apunta a dos cosas, la primera es que al alterar un tipo, dicho cambio debe verse reflejado en las instancias y la otra es contar con la posibilidad de migrar instancias entre tipos. En Oracle 8i, existe una forma restringida de evolución. A diferencia de una tabla relacional, un tipo no puede ser alterado, solamente podemos agregar métodos, cambio que no afecta a los tipos, tablas o vistas que dependen de él. Esta carencia junto con la de herencia son las limitaciones más significativas que tiene Oracle, aunque con un poco más de trabajo, podremos lograr esta funcionalidad. Una solución a este problema es crear una tabla relacional con los atributos del objeto como columna y una vista de objeto sobre la tabla relacional. Luego utilizaremos la vista de objeto como si fuera una tabla de objetos, y cuando necesitemos agregar, modificar o eliminar la definición de un atributo, lo haremos sobre la tabla relacional, y luego modificaremos nuestra vista de objeto. No se pueden migrar instancias entre tipos en Oracle 8i.

Transacciones y recuperación ante fallos. Estas características están presentes en Oracle desde varias versiones atrás, y de hecho son uno de sus puntos más fuertes. Una transacción, es una unidad lógica de trabajo que contiene una o más sentencias SQL que pueden ser todas aplicadas a la base de datos o ninguna. La utilidad de una transacción es que nos permite impedir que los cambios producidos por las sentencias SQL ya ejecutadas se apliquen a la base de datos, al ocurrir alguna excepción, manteniendo así la consistencia y la integridad. En lo referente a la recuperación ante fallos, existen varios errores que Oracle soluciona automáticamente y otros requieren de una mínima intervención del usuario. Otra característica importante, es que la mayoría de los fallos pueden ser superados sin la necesidad de bajar el servidor.

Mecanismos de autorización. Al igual que en SQL:1999, los mecanismos de autorización están basados en las tablas y en los tipos objeto. Se autoriza o restringe el acceso y manipulación de las tablas o tipos objeto, pero no se puede restringir el acceso sobre un objeto o una fila.

Compatibilidad con el modelo relacional. Oracle posee una alta compatibilidad con el modelo relacional, permitiendo que interactúen las estructuras relacionales con las nuevas estructuras. Además permite la creación de vistas de objeto sobre tablas relacionales. Una vista de objetos, se basa en el resultado de una consulta y en un tipo objeto, que definirá las columnas de la vista. La consulta puede estar basada en varias tablas, sean relacionales o de objeto. Al consultar una vista de objetos, se obtendrá un conjunto de instancias del tipo objeto en el que se base la vista.

5 Evaluación

Limitaciones encontradas en SQL:1999.

1. No podemos sobrecargar las funciones observadoras y mutadoras
2. No podemos especificar un nivel de encapsulamiento (public, private, protected) sobre los atributos y métodos.
3. No podemos definir constructores
4. No podemos definir atributos estáticos
5. Sólo contamos con los arreglos para definir colecciones.
6. No podemos definir arreglos de múltiples dimensiones (por ejemplo matrices).
7. Los tipos de referencia sólo pueden almacenar direcciones de objetos persistentes.
8. No podemos crear módulos o paquetes de objetos.
9. No contamos con herencia múltiple.
10. No está soportado el manejo de versiones y configuraciones de objetos.
11. No podemos modificar la definición de un atributo ni la interfaz de los métodos.

Limitaciones encontradas en Oracle 8i. En Oracle 8i, encontramos las siguientes limitaciones, referente al manejo de objetos:

1. Un tipo no puede tener más de 1000 atributos
2. No podemos sobrecargar el constructor definido por el sistema, ni redefinirlo. Tampoco podrá crear nuevos constructores.
3. Los procedimientos miembro de un tipo, no pueden ser invocados desde SQL.
4. Las funciones miembro de un tipo, que posean uno o más parámetros pasados como INOUT o OUT, no pueden ser invocados desde SQL.
5. No podemos modificar la definición de atributos o métodos existentes de un tipo objeto. Sólo podemos agregar nuevos métodos.
6. No podemos replicar tablas de objetos.
7. No podemos tener colecciones que almacenen colecciones, ni colecciones de objetos que posean un atributo de un tipo de colección.
8. No podemos definir colecciones en base a ciertos tipos de datos como por ejemplo: BOOLEAN, BLOB, CLOB, LONG, LONG RAW, y otros.
9. No podemos especificar un nivel de encapsulamiento (private, public, protected) para los atributos, ni para los métodos.
10. No podemos definir Jerarquías de Herencia, ni simple, ni múltiple.
11. No podemos duplicar las tablas de objeto en Oracle 8i

6 Conclusiones

A pesar de las limitaciones del estándar para el manejo de objetos, este obtiene una buena calificación en nuestra evaluación, un 78%. Por otro lado, Oracle 8i obtiene una calificación de 70%, más baja que el estándar, esto se debe principalmente a la falta de herencia, un encapsulamiento incompleto, y el deficiente soporte a la evolución de esquemas. El estándar, con sus capacidades y limitaciones, es lo suficientemente robusto para permitir que los desarrolladores lo utilicen para hacer persistir sus objetos,

	Oracle 8i		SQL:1999	
Criterio	Parcial(%)	Final(%)	Parcial(%)	Final(%)
1. Definición de tipos o clases		95		60
(a) Definición de atributos estáticos	60		60	
(b) Definición de atributos de instancia	100		100	
(c) Constructores definidos por el usuario	60		100	
(d) Métodos estáticos	100		100	
(e) Métodos de instancia	100		100	
(f) Sobrecarga de métodos	100		100	
(g) Definición de colecciones	100		100	
(h) Clases abstractas	60		100	
2. Encapsulamiento		80		80
(a) Atributos y métodos en una sola entidad	100		100	
(b) Distinguir entre interfaz e implementación	100		100	
(c) Visibilidad de métodos	60		60	
(d) Visibilidad de atributos	60		60	
3. Modularidad		60		60
4. Jerarquía de clases o tipos		82		92
(a) Asociación	100		100	
(b) Agregación y Composición	100		100	
(c) Herencia Simple	60		100	
(d) Herencia múltiple	60		60	
(e) Polimorfismo	60		100	
5. Concurrencia		100		100
(a) Métodos con múltiples hilos	100		100	
(b) Concurrencia de usuarios	100		100	
6. Objetos Persistentes		100		100
(a) Creación	100		100	
(b) Modificación	100		100	
(c) Eliminación	100		100	
(d) Consulta	100		100	
(e) Identificadores de objetos	100		100	
7. Versiones y configuraciones de objetos		0		0
8. Evolución de instancias		0		60
9. Evolución de esquemas		80		92
(a) Agregar atributos	60		100	
(b) Agregar métodos	100		100	
(c) Agregar restricciones sobre objetos	100		100	
(d) Eliminar atributos	60		100	
(e) Eliminar métodos	60		100	
(f) Eliminar restricciones	100		100	
(g) Modificar definición de atributos	60		60	
(h) Modificar interfaz de métodos	60		60	
(i) Modificar implementación de métodos	100		100	
(j) Modificar restricciones	100		100	
10. Transacciones y recuperación ante fallas		100		100
11. Mecanismos de autorización basados en objetos		60		60
12. Compatibilidad con el modelo relacional		100		100
Promedio		70.6		78.3

Figura 8: Resultados de la evaluación

puesto que las limitaciones son pocas y en áreas no fundamentales. Pero Oracle 8i, por su falta de herencia, hace que el desarrollo de aplicaciones OO en él sea muy complejo y poco productivo.

Uno de los beneficios de la tecnología de objetos incluida en el estándar SQL:1999 y en Oracle 8i es permitir que los usuarios incrementen el conjunto de tipos de datos provisto por el sistema. Esto permite la reutilización del código, lo que disminuye el tiempo de desarrollo y la complejidad. Además, trae como consecuencia una mayor eficiencia a la hora de mantener sistemas, porque bastaría con modificar un tipo (ALTER TYPE) para que automáticamente se modifiquen todas las tablas y tipos que dependen de él. Lamentablemente dicha actualización no está presente en Oracle 8i aunque cuenta con el predicado ALTER TYPE, este sólo permite agregar nuevos métodos, lo cual es muy limitado. En SQL:1999 esto sí es posible, pero tiene algunas restricciones. La herencia y el polimorfismo, también traen beneficios a las bases de datos, puesto que permiten extender el sistema fácilmente para que se adapte a los cambios de requerimientos. Permitiendo así un desarrollo incremental, lo que disminuye la complejidad del desarrollo. Estas características están presentes en SQL:1999, pero no en Oracle 8i.

Otra de las capacidades de SQL:1999 y de Oracle 8i, son las vistas de objetos. Estas permiten definir una capa de orientación a objeto sobre datos relacionales, lo que elimina la necesidad de migrar las tablas relacionales a tablas de objetos.

Una limitación del estándar SQL:1999 y Oracle 8i es que carece de la funcionalidad para manejar versiones y configuraciones de objetos. Esta es la desventaja más grande entre las bases de datos objeto-relacional y las bases de datos orientadas a objeto puras.

Referencias

- [1] Booch Grady (1996) Análisis y Diseño Orientado a Objetos. Addison Wesley/Díaz. (1996).
- [2] Piattini M. (1999) El futuro de las Bases de Datos. Tutorial VII Encuentro Chileno de computación, (1999).
- [3] Piattini M. SABD (1995) Orientados a Objetos. Revista algoritmo <http://www.algoritmodigital.com>, abril de 1995
- [4] ANSI X3H2-99-269/WG3:RTM-006, (ANSI-ISO Working Draft) Persistent Stored Modules (SQL/PSM), agosto 1999. En: ftp://sqlstandards.org/SC32/WG3/Progression_Documents/Informali_working_drafts/wd-psm-1999-07.pdf
- [5] ANSI X3H2-99-462/WG3:SAF-003, (ANSI-ISO Working Draft) Framework (SQL/Framework), agosto 1999. En: <ftp://jerry.ece.umassd.edu/SC32/WG3>
- [6] ANSI X3H2-99-463/WG3:SAF-004, (ANSI/ISO Working Draft) Foundation (SQL/Foundation), agosto 1999. En: <ftp://jerry.ece.umassd.edu/SC32/WG3>
- [7] Oracle. "Oracle 8i Concepts". Número del documento A67781-01 en: <http://technet.Oracle.com/docs>