Paradigmas de Modelado de Base de Datos I. CI5311. Tecnología Objeto-Relacional: SQL3 y Traducción de OMT a OR

Prof. Soraya Abad Mota

Actualización Junio 2012

En la primera porción de estos apuntes de clases se cubrió desde los objetivos del curso hasta la introducción a la tecnología objeto-relacional. En esta segunda parte completamos el tópico Objeto-Relacional, cubriendo el estándar SQL3 y una estrategia de traducción de esquemas conceptuales expresados en notación UML, a estructuras objeto-relacionales ilustradas en el manejador Oracle.

Para preservar la numeración de secciones y subsecciones, copiamos la tabla de contenidos de las secciones de la primera porción de los apuntes.

6.3. SQL3 (SQL:1999)

Después de la definición oficial del modelo relacional por parte del Dr. Codd (1970) había que especificar un lenguaje de definición y manipulación de datos para el modelo. El primer lenguaje de esta naturaleza se llamó *SEQUEL* y sus siglas significaban, en inglés, Structured English Query Language, este lenguaje era el *API (application program interface)* del Sistema R, que era el manejador relacional que estaba desarrollando IBM a principios de los años 70. El primer prototipo de SEQUEL salió entre 1974 y 1975, después hubo un par de versiones más y se le cambió el nombre a SQL, pero se seguía pronunciando "sequel". El proyecto que desarrollaba el Sistema R estuvo activo desde 1971 hasta 1979 y después evolucionó en el Sistema R* que era un sistema de base de datos distribuidos.

Pero el Sistema R era un sistema de investigación, después IBM desarrolló un manejador comercial, llamado SQL/DS, que introdujo al mercado en 1981 y en 1983 liberó la primera versión de su manejador DB2. En particular, otra empresa: Relational Software, Inc. sacó otro producto al mercado que le ganó en ventas al producto de IBM, esta empresa luego se convirtió en Oracle Corporation, Inc. En paralelo, otras empresas estaban desarrollando productos relacionales.

En 1986 SQL se convirtió en un estándar legal establecido por el *ANSI (American National Standards Institute)*, la especificación de este estándar se llamó *SQL-86*, después vino otro en 1989: *SQL-89*, y uno en 1992, que fue una revisión importante del anterior, llamado *SQL-92*. A partir de este estándar se sabía que existían muchos aspectos por resolver para que realmente SQL pudiera ofrecer las capacidades de base datos requeridas y para poder satisfacer las necesidades de los usuarios. Durante siete años se produjeron varios estándares que atacaban aspectos específicos de SQL, hasta que en 1999, se produjo el siguiente estándar importante de SQL, el SQL:1999 o SQL3. Estos estándares intermedios fueron:

CLI-95 El estándar SQL/CLI (CLI por *Call Level Interface*), cuya implementación más conocida es el estándar *ODBC (Open Database Connectivity)*.

PSM-96 El estándar SQL/PSM (PSM por *Persistent Stored Modules*), el cual especifica la sintaxis de la lógica procedimental de los módulos del servidor de SQL. A pesar de que los manejadores comerciales tenían la capacidad de usar "stored procedures" no había ningún estándar al respecto.

OLB-98 El estándar SQL/OLB (OLB por *Object Language Bindings*) provee la habilidad de incluir comandos de SQL en programas de Java y está basado en el paradigma de JDBC, el cual utiliza iteradores (iterators) que son "fuertemente tipeados".

En 1999, después de esta secuencia de estándares intermedios, finalmente fue aceptado y publicado el SQL3 o SQL:1999. En él participaron las dos organizaciones oficialmente activas en la estandarización de SQL, estas organizaciones son: ANSI e ISO (*International Organization for Standardization*), la primera es una organización estadounidense, la segunda está basada en Europa. Dentro de estas organizaciones hay comités específicos que trabajan en este desarrollo. Dentro de ISO está el ISO/IEC JTCI (Joint Technical Committee 1), que depende del International Electrotechnical Commission, y cuya responsabilidad

es el desarrollo y mantenimiento de estándares relacionados con las Tecnologías de la Información. Dentro del JCT1, el subcomité SC32 se formó recientemente su título es "Data Management and Interchange" y tiene que ver con estándares relacionados a base de datos y a metadatos, desarrollados por otras organizaciones. El SC32 además formó grupos de trabajo (Working Groups) que son los que realmente hacen el trabajo técnico, en particular el WG3 de "Database Languages", es el responsable por el estándar SQL, y el WG4 es el encargado de SQL/MM (SQL Multimedia).

En los E.E.U.U. los estándares de tecnología de la información están a cargo del comité acreditado por ANSI que es un Accredited Standards Development Committee llamado *NCITS (National Committee for Information Technology Standardization)*, que antes se conocía como "X3". Dentro de NCITS el Comité Técnico H2 (antes "X3H2") es el responsable de varios estándares relacionados con el manejo de datos, en particular de SQL y SQL/MM.

A partir de 1999, la concepción del estándar se dividió en porciones, cada una de las cuales aborda un aspecto específico de SQL, estas porciones son:

- *Foundation*: constituye el corazón del estándar.
- *SQL/CLI*: Call Level Interfaced.
- *SQL/PSM*: Persistent Stored Modules.
- *SQL/Bindings*.
- *SQL/XA*.
- *SQL/Temporal*.

En el año 2003 salió un nuevo estándar de SQL, SQL:2003, el cual introdujo la noción de XML y estandarizó los generadores de secuencias o valores auto generados, esto incluye columnas que se utilizan como identificadores.

6.4. Características Objeto-Relacionales

A SQL3 se le ha llamado informalmente "SQL orientado por objetos", pero SQL3 va más allá de SQL-92 más la tecnología de objetos. Este nuevo estádar provee muchas mejoras sobre SQL-92 que no se limitan a la inclusión de los conceptos OO. Aparte de algunos nuevos tipos de datos y nuevos predicados, hay tres aspectos fundamentales de SQL:1999 que vamos a reseñar en este curso:

- Orientación por objetos, aquí se concentran las características denominadas objeto relacionales y es donde nosotros hacemos énfasis. Veremos en detalle estas características.
- Bases de Datos Activas, lo cual se refleja en la especificación de triggers y assertions en el estándar. Este aspecto lo cubrimos luego, cuando veamos el modelo dinámico y el paradigma de base de datos activas.

- Nueva semántica, nos referimos específicamente a la habilidad de SQL3 de definir consultas recursivas, con lo cual se puede computar la clausura transitiva, eliminando así la restricción que existía en el álgebra relacional. El paradigma envuelto en esta nueva habilidad de SQL es el de Bases de Datos Deductivas del cual se incluye una visión global al final de este curso.

Comenzamos nuestra cobertura de los conceptos de OO en SQL3 con las estructuras orientadas por objetos, las cuales son extensiones de los tipos. Estas estructuras son:

1. Tipos definidos por el usuario (UDT's, user-defined types), en esta categoría se encuentran:

   a) *Distinct types*: declaran tipos diferentes basados en un tipo ya definido, formalmente, a pesar de que el tipo base es el mismo, los nuevos tipos declarados son diferentes y las operaciones que los combinan dan error.
   CREATE DISTINCT TYPE usDollars AS decimal(9,2)
   CREATE DISTINCT TYPE canadianDollars AS decimal(9,2)
   A pesar de que los dos tipos definidos en este ejemplo son números reales, no se pueden sumar valores en usDollars con valores en canadianDollars, pues da error.
   Se pueden definir funciones para hacer las conversiones de tipo con la opción de CAST dentro de la definición del tipo (dentro del CREATE TYPE).
   CAST (SOURCE AS DISTINCT) WITH usd_to_real
   especificado dentro de la creación del tipo usDollars, dice que para convertir un valor en usDollars a un real se utilice la función usd_to_real, la cual debe estar definida previamente. Cuando se quiera operar sobre dos tipos diferentes, primero se hace cast de uno al otro o de cada uno a un tipo base y luego si se puede operar sobre los valores. Si se declaran las siguientes variables:
   DECLARE VARIABLE X decimal(9,2)
   DECLARE VARIABLE y usDollars
   No se puede sumar directamente X + Y, sin embargo si se puede hacer
   X + CAST(Y AS decimal(9,2)).

   b) *Row type*: se utiliza para definir un registro que contiene campos de varios tipos, es un tipo compuesto por otros tipos. Por ejemplo, se puede definir el tipo compuesto telefono de la siguiente forma:

```
create row type telefono_t (
       codigoArea    varchar(3),
       numTelefono   varchar(7));

create table TablaTelefonos of type telefono\_t;
```

   Como se ilustra en el ejemplo anterior, una vez definido un row type éste se puede usar para definir una tabla con filas del tipo de ese row type. De hecho, esta es la forma de darle persistencia al tipo. Los *row type* también se pueden utilizar para definir el tipo de una columna de una tabla y que esa tabla tenga otras columnas.
   Adicionalmente, se pueden definir funciones que devuelvan valores del tipo de un *row type*.

Con los *row type* se puede utilizar la notación de punto (dot notation) para acceder a elementos del registro compuesto.

c) *Abstract data type (ADT)* (tipo abstracto de dato definido por el usuario). La definición del ADT incluye la estructura del tipo, las operaciones que definen igualdad y ordenamiento, y las operaciones que definen el comportamiento del ADT. Las operaciones se implementan con procedimientos llamados "rutinas".

```
create type empleado_t as object (
              nombre VARCHAR(100),
              apellido VARCHAR(100),
              telefE   telefono_t,
              direccion direccion_t,
              salario decimal(9,2));

create table tablaEmpleados of empleado_t;
```

La manera de almacenar persistentemente un ADT en una base de datos es declarándolo como el tipo de una de las columnas de una tabla. Los atributos y las operaciones de un ADT pueden ser públicos o privados, los públicos son los únicos que pueden ser accedidos fuera del ADT. Adicionalmente, para cada atributo se definen automáticamente dos funciones: una observador y una "mutator". Se pueden definir atributos virtuales dentro de un ADT, para los cuales no se almacenan valores, sino que se calculan cuando se necesitan a través de las funciones definidas por el usuario de observador y "mutator". Las instancias de un ADT se crean con las funciones constructoras del sistema. Se puede utilizar la notación de punto o la funcional (solo para las funciones) para acceder las partes de un ADT.

En el ejemplo de empleado, sólo definimos la estructura del tipo, pero podemos definir allí las funciones apropiadas para manipular instancias del tipo.

2. *REFerence Type*. Es un tipo especial de datos para definir identificadores de objetos. En términos de lenguajes de programación imperativos es un apuntador a un tipo. Todo tipo complejo o estructurado tiene la posibilidad de definirle un ref para los apuntadores a instancias de ese tipo.

```
create type empleado_t as object (
              nombre VARCHAR(100),
              apellido VARCHAR(100),
              telefE   telefono_t,
              direccion direccion_t,
              salario decimal(9,2),
              trabajaEn REF(depto_t));
```

3. *Collection Types*: arreglos, conjuntos, listas y multiconjuntos. SQL:2003 es donde se han definido ampliamente los multiconjuntos.

4. BLOB y CLOB. Objetos grandes de los tipos binario o caracter.

5. Uso de subtipos (con la cláusula UNDER). Herencia de atributos y operaciones.

6.5. Ejemplo de DBMS objeto relacional

El ejemplo de manejador objeto relacional que veremos en esta edición del curso es Oracle 9i o 10g. Primero cubrimos las características de objetos de ese manejador y luego nos concentramos en cómo traducir esquemas de objetos de OMT con notación UML a Oracle.

En Oracle hay varios aspectos nuevos que implementan las nociones de objeto definidas en SQL:1999. Estos aspectos se describen a continuación.

- *Object types*, son tipos abstractos de datos (TADs) definidos por el usuario o por el sistema. Tienen dos componentes: los atributos, en cuya definición se pueden usar los tipos definidos en Oracle (*built-in types*) u otros TADs, y los métodos, que son funciones o procedimientos escritos en PL/SQL o en algún otro lenguaje soportado por Oracle. Un ejemplo sencillo de definición de tipo de objeto es:

```
CREATE OR REPLACE TYPE departmento AS OBJECT (
    Code          VARCHAR(20),
    Name          VARCHAR(40),
    MEMBER FUNCTION getStudents     RETURN personsArray,
    MEMBER FUNCTION getFaculty      RETURN personsArray,
)
/
< definicion de las dos funciones >
/
```

  Los *object types* se usan para definir:

  - *object tables* (tablas objeto) cuyas tuplas tienen identificadores de objeto (OID), un ejemplo de esta definición es:
    CREATE TABLE departmentos OF departamento;
  - objetos contenidos (embedded objects), los cuales no tienen OID y definen estructuras de registro complejas, y
  - tipo de atributos en otras tablas; en una tabla relacional las columnas pueden ser del tipo de un TAD o de un tipo complejo. Por ejemplo:

```
CREATE OR REPLACE TYPE persona_t AS OBJECT (
    Nombre          VARCHAR(40),
    Telefono        VARCHAR(20) );
CREATE TABLE contactos (
    contacto     persona_t,
    fecha        DATE) ;
```

  En Oracle 8i no hay jerarquías de tipos. Pero a partir de la versión 9i, Oracle si permite definir unos tipos como subtipos de otros ya definidos, utilizando la cláusula *UNDER*; con esta cláusula se pueden establecer jerarquías de tipos de varios niveles, pero sólo se permite herencia simple.

- *Object views*, para ver esquemas relacionales como objetos.

- Nuevos tipos, adicionalmente a los tipos tradicionales predefinidos en Oracle.
  - VARRAYs y nested tables, dos nuevos tipos que permiten que las columnas de una tabla sean una colección estructurada de datos. Sirven para representar atributos compuestos y multivaluados, entre otros usos.
  - REFs, son referencias a objetos, se utilizan para almacenar apuntadores lógicos a objetos.
  - LOBs, arreglos muy grandes de bytes sobre los cuales están definidas una serie de operaciones por parte del sistema.

6.6. Traducción del modelo de objetos de OMT a Oracle

En este curso hemos descrito cómo producir el esquema conceptual de una base de datos utilizando el modelo de objetos de OMT con notación UML. Una vez construido el esquema conceptual es necesario traducirlo al modelo lógico objeto relacional para poder implementar la base de datos en un DBMS que siga ese modelo de datos.

Como vimos en la sección anterior, Oracle es un ejemplo de manejador que implementa aspectos del modelo lógico objeto relacional. En esta sección tratamos de dar lineamientos sobre cómo traducir los esquemas de objetos de OMT a objeto relacional.

Para escribir esta sección utilizamos las siguientes fuentes:

- Libro [OBJ DB]: Capítulo 2 (Mapping UML Diagrams to Object-Relational Schemas in Oracle 8i) de S. Dietrich y S. Urban.
- Libro "Object-Oriented Oracle" de W. Rahayu, D. Taniar y E. Pardede. CyberTech Publishing 2006.

Hemos estudiado y analizado lo que proponen los autores de estas dos fuentes y tomando ideas de cada uno, proponemos los siguientes lineamientos de traducción de un esquema conceptual escrito en notación UML a conceptos del modelo lógico objeto relacional. Con esta traducción se puede escribir un *script* basado en la sintaxis de un DBMS que utilice este modelo lógico.

1. **Creación de tipos.** Se definen tantos tipos de datos (*object types*) como haga falta para representar los atributos y las operaciones de las clases del esquema conceptual. Por ejemplo, para los atributos compuestos se puede definir un tipo estructurado que contega los atributos simples que lo conforman; para los atributos multivaluados (o con multiplicidad mayor a 1) se puede utilizar un VARRAY o un nested table.
2. **Creación de tablas.** Las *clases* se pueden traducir en *object tables* (tablas objeto), donde el tipo del *object table* debe definirse primero como un *object type*. También se puede utilizar una tabla relacional tradicional para traducir las clases y para los atributos de esa tabla se pueden utilizar los tipos definidos en el punto anterior. En cualquiera de los dos casos, es importante que toda tabla tenga definido un *primary key* y también se debe tratar de utilizar la noción de clave foránea para controlar la integridad de los datos.
3. **Representación de asociaciones.** Para las asociaciones debemos distinguir entre los diferentes tipos.
    - **Asociaciones Binarias 1:1 o 1:n**. Dependiendo de cómo se haya implementado cada una de las clases que participan en la asociación, en la clase que tiene una participación máxima de 1 en la asociación, se puede definir un atributo tipo REF que apunte al tipo de la otra clase. Para garantizar la simetría en la consulta de la asociación es necesario definir una función en la otra clase (la que no tiene el atributo tipo REF) que permita conocer las instancias de la clase con participación máxima de 1, con las cuales cada instancia está asociada.

- **Asociaciones n:m** Las autoras del capítulo del libro [OBJ DB] (ver bibliografía en el cronograma del curso) sugieren que se utilicen colecciones estructuradas, como *VARRAY*s o *nested tables* para implementar como un atributo (multivaluado y compuesto) de una clase, a las instancias de la otra clase que están asociadas con ella. Consideramos que esta implementación esconde a la asociación y preferimos una implementación que haga más explícita la asociación. Hay situaciones, sin embargo, donde es conveniente usar esta representación, por ejemplo, si la clase que se está representando como una colección (*nested table* o *VARRAY*) no tiene asociaciones con otras clases y su existencia depende de la única clase con la cual está asociada.

  Por ello sugerimos definir una tabla para la asociación, que contenga un atributo tipo REF por cada clase que participa en la asociación. Adicionalmente, esta tabla debería tener definida su clave primaria y establecerse restricciones de clave foránea sobre los atributos REF que apuntan a las clases asociadas. También se debe establecer el alcance de cada atributo tipo REF con la cláusula de *SCOPE*, aunque es preferible usar el foreign key y references en la medida de lo posible, para controlar mejor cuando se elimine algún objeto apuntado por otro.

4. **Generalizaciones**. Para traducir las generalizaciones es útil recordar las cuatro opciones de traducción del modelo ERE al modelo relacional. En el contexto de estas cuatro opciones se puede utilizar una tabla para cada superclase y las subclases o se pueden representar todas las clases en una sóla tabla. En cualquier caso, es importante que se explicite la asociación 1:1 que existe entre la superclase y cada una de las subclases. Esto se puede realizar colocando atributos tipo REF en la superclase para apuntar a las superclases o viceversa.

   El concepto de la tecnología objeto relacional que se adapta naturalmente a las generalizaciones es el de la definición de subtipos (*UNDER*) y el aprovechamiento del mecanismo de la herencia. De esta forma, los tipos de las subclases pueden definirse como subtipos del tipo de la clase general o superclase.

   En el libro "Object-Oriented Oracle" se dan sugerencias particulares para cada tipo de generalización. Por ejemplo si es solapada o disjunta, si es parcial o total y sus combinaciones posibles.

5. **Agregaciones**. Los autores del libro "Object-Oriented Oracle" sugieren tres opciones para traducir las agregaciones, dependiendo de su naturaleza. Una de estas opciones es aprovechar una característica de las estructuras físicas de las cuales dispone Oracle, que es la noción de un *cluster de varias tablas*, según la cual, las tuplas de diferentes tablas se almacenan juntas en el mismo bloque de datos. Otra opción es representar en la tabla en la cual se traduce el agregado (el todo) con un *nested table*; este *nested table* contiene todas las instancias del objeto componente del todo. Una tercera opción es crear una nueva tabla que represente al todo agregado y en esa tabla colocar atributos que indiquen cuál es la instancia de la clase agregada (la superclase) y cuáles son las instancias de las clases componentes que participan en el todo.

Adicionalmente, se pueden destacar los siguientes consejos:

- Si en cada clase que participa en una asociación se colocan atributos tipo REF que apuntan a las instancias de la otra clase, eso constituye un tipo de redundancia. Esta

redundancia debe controlarse construyendo mecanismos que garanticen al crear una nueva instancia de la asociación, que se coloquen las referencias apropiadas automáticamente.

- Se pueden crear *object types* para representar estructuras definidas por el usuario u otros tipos complejos que se puedan necesitar en una aplicación. Esos tipos se pueden entonces utilizar para crear objetos contenidos dentro de tablas relacionales o tablas objeto.

Es importante aclarar que en estos lineamientos de traducción se destacan los aspectos nuevos, propios de la tecnología objeto relacional, pero también se pueden utilizar los procedimientos de traducción del modelo relacional para traducir algunos elementos del esquema conceptual.