Utilización de versiones de tablas en la docencia de SQL interactivo

Mª Belén Vaquerizo García, Angel Arroyo Puente, Jesús Manuel Maudes Raedo

Escuela Politécnica Superior, Departamento de Ingeniería Civil,

Área de Lenguajes y Sistemas Informáticos, Universidad de Burgos,

Av. Cantabria s/n., 09006, Burgos (España)

e-mail: {belvagar, aarroyo, jmaudes}@ubu.es

Resúmen

La docencia de SQL en los cursos iniciales de un currículo en informática es una materia con una gran componente práctica. En este artículo se discute como plantear el esquema de base de datos sobre el que hacer prácticas en función de la metodología docente y contenidos a impartir, teniendo en cuenta la dicotomía entre dotar de privilegios al alumno para que experimente, o acotárselos para tenerlo más controlado. Finalmente, se discutirán las ventajas e inconvenientes de una aproximación basada en versiones de tablas que puede servir de punto de equilibrio en cuanto a permisos sobre el sistema para los alumnos.

1. Introducción

La docencia en Bases de Datos requiere, ya en sus fases más tempranas, que el alumno se enfrente a los lenguajes de bases de datos practicando con ellos. En [1], [2] se propone un esquema secuencial de asignaturas de Bases de Datos distribuidas a lo largo de las titulaciones de informática. Estos documentos recomiendan que la primera asignatura a la que se enfrenten los alumnos (*Bases de Datos I*) abarque entre sus contenidos, el trabajo con SQL interactivo.

La experiencia de los que subscriben el presente documento es que la docencia de estos contenidos tiene unas componentes más pedagógicas dentro del aula de prácticas que dentro de una clase tipo lección magistral. Esta opinión es refrendada en las encuestas de calidad rellenadas por los alumnos y tiene, probablemente, sus orígenes en los siguientes motivos:

- Los grupos de prácticas son más reducidos que los grupos correspondientes a las lecciones magistrales, lo que permite al profesor dar una mayor atención individualizada a cada alumno.
- El alumno tiene la oportunidad de experimentar, y aprender por sí mismo; sin dedicar la mayor parte del tiempo a tomar apuntes.
- El profesor puede observar la marcha de sus alumnos, teniendo la oportunidad de reaccionar e incidir sobre aquello que ha quedado menos claro.
- Los alumnos tienen una mayor motivación, al tender a identificar las prácticas con el que es, para la mayoría de ellos, objetivo último de sus estudios, que no es otro que la preparación para insertarse convenientemente en el mercado de trabajo.

Todos estos puntos, hacen necesario cuidar (i) los contenidos, (ii) la metodología y (iii) los medios utilizados en la presentación de contenidos

en las prácticas en general, y en nuestro caso concreto, en las prácticas de SQL interactivo en Bases de Datos.

En cuanto a los contenidos de SQL interactivo, es interesante que el alumno pueda experimentar con el DML (*Data Management Language*) de SQL por completo, y con el DDL (*Data Definition Language*) a nivel de creación de tablas y vistas. Las operaciones de actualización en DML deben de permitir al alumno experimentar con la concurrencia a nivel transaccional, y referente a las vistas, ha de ser posible trabajar con vistas actualizables.

Con relación a la metodología en el aula de prácticas, hay varias posibilidades: (i) explicaciones del profesor, (ii) problemas a resolver por el alumno solo, (iii) problemas a resolver por el conjunto de los alumnos guiados por el profesor y discutiendo cada una de las posibles soluciones. La explicación de los problemas por parte del profesor no fomenta la participación del alumno, pero tiene como ventaja que es la forma de avanzar más deprisa en el temario y puede ser útil en aquellos puntos en los que el profesor estime que no es interesante incidir demasiado. Dejar al alumno que resuelva los problemas por sí mismo suele consumir demasiado tiempo, pero el alumno suele sacar bastante partido de su trabajo. Esta fórmula es idónea para aquellas horas en las que el aula de prácticas esté libre para uso de los alumnos, o para un acceso web desde casa del alumno al Sistema de Base de Datos utilizado en prácticas. Finalmente, el resolver los problemas por el conjunto de los alumnos guiados por del profesor (de ahora en adelante *prácticas guiadas*) es una fórmula intermedia que fomenta la participación y el aprendizaje, optimizando el tiempo dedicado. Si bien, nosotros nos decantamos a abusar de esta última fórmula, el profesor debe además utilizar las otras dos allí donde estime que los contenidos, la aptitud y actitud del alumno puedan hacerlas más interesantes.

Finalmente, en cuanto a los medios, históricamente ha habido aproximaciones de Sistemas de Bases de Datos orientados a docencia, como WinRDBI[1] [3] y [4]. El inconveniente principal de éste y otros sistemas similares, es que aunque intenta abarcar otros lenguajes como Álgebra y Cálculo Relacional, la parte de SQL no es muy fiel al SQL estándar, especialmente en lo concerniente a la definición de tablas, restricciones y vistas. Para nosotros es necesario que el sistema sea lo más fiel posible al estándar ISO de SQL, preferiblemente a la versión de 1992 [6], por ser más sencilla y más extendida en la industria. Otro inconveniente de WinRDBI es que es monousuario impidiendo experimentar con la concurrencia.

La solución a concurrencia y acercamiento al estándar viene de la mano de la utilización de Sistemas Gestores de Bases de Datos (SGBDs) comerciales. Aunque en la actualidad hay buenas ofertas económicas para docencia de este tipo de sistemas, hay que valorar la adecuación de productos de libre distribución como por ejemplo *PostgreSQL*[2] y *MySQL*[3].

Sin embargo, los SGBDs comerciales plantean el problema de cómo articular los privilegios de los alumnos para que puedan compartir datos con el profesor en una práctica guiada, pero a la vez puedan modificarlos para experimentar, sin que por ello arruinen la labor del profesor, ni de sus compañeros. En la medida que al alumno se le provee independencia y privilegios sobre el entorno, el profesor pierde el control sobre el contenido de las bases de datos de los alumnos, lo que puede impedirle impartir adecuadamente unas prácticas guiadas. Por el contrario, recortar demasiado las posibilidades del alumno, puede ir en contra de que experimente por su cuenta, lo que también es perjudicial.

Este artículo trata sobre cómo plantear soluciones a este problema para cada uno de los posibles escenarios que aparecerán en función del método y contenidos empleados. Para ello se estructura de la siguiente forma: En el apartado 2 plantearemos las diversas posibilidades de

[1] http://www.eas.asu.edu/~winrdbi/

[2] http://postgresql.org/

[3] http://www.mysql.com/ Desafortunadamente la versión actual de *MySQL* no soporta transacciones, por lo que es poco ideal para DML con concurrencia.

esquemas y privilegios en función del tipo de contenidos. En el apartado 3 describiremos la aproximación basada en versiones de tablas, adecuada para prácticas guiadas. En el apartado 4, describiremos los puntos más notables de una implementación real en *PostgreSQL* de la aproximación basada en versiones de tablas. Finalmente, en el apartado 5 se comentarán las conclusiones y trabajos futuros.

2. Planteamiento del Esquema de Base de Datos y Permisos en función de los Contenidos

Los contenidos posibles de SQL interactivo en una asignatura introductoria a las Bases de Datos deben de cubrir el DML por completo, y el DDL a nivel de definición de tablas y vistas. La experimentación con concurrencia y con vistas actualizables también debe de estar amparada. En este apartado mostraremos cómo plantear los esquemas y privilegios de la base de datos para cada uno de estos contenidos.

2.1. Las Consultas SELECT

La configuración más habitual para una práctica guiada sobre consultas SELECT es hacer que todos los alumnos compartan las tablas del profesor, desde un usuario distinto de forma que los alumnos no puedan modificar su contenido, sino que tan solo puedan consultarlo. Las tablas del profesor tendrían únicamente permiso de lectura para los usuarios alumnos. Los alumnos podrían acceder perfectamente desde un único usuario del sistema para facilitar el mantenimiento. Este usuario alumno no necesitaría privilegios que le permitiesen crear objetos de la base de datos como tablas o vistas. En esta situación el alumno puede experimentar consultas con las tablas del profesor:

1 Sin temor a que el alumno cree algún tipo de problema por excederse en la ocupación de disco, pues no puede crear tablas, ni modificar las del profesor.

2 Sin temor a que bloquee las operaciones de otros compañeros, pues todos están accediendo en modo consulta.

3 El profesor puede cambiar la extensión de las tablas sobre la marcha para conseguir efectos particulares en las consultas (introducir valores nulos para ver comportamientos atípicos etc …) teniendo la certeza de que esos efectos son inmediatamente visibles a todos los alumnos. Dado que los alumnos y el profesor acceden a los mismos datos, es conveniente que el sistema admita relajar el control de la concurrencia de forma que en la misma transacción del alumno se puedan obtener distintas lecturas de las tablas del profesor, las cuales pueden ir cambiando por acción de transacciones cometidas de éste. Este nivel de concurrencia relajado es el conocido como *Read Commited* en el estándar SQL/ISO[6]. Esta configuración viene activada por defecto en algunos sistemas como *Oracle*. Otra solución posible al mismo problema, es que los alumnos tengan activada la opción de *autocommit*, habitual en casi todos los sistemas, de forma que cada una de sus consultas sea una transacción en sí misma.

Desde el punto de vista estético, este planteamiento puede resultar feo en sistemas como *Oracle* [5] que tienen un espacio de nombre de tablas distinto para cada usuario. Esto obliga a que el alumno para referenciar a la tabla *X* del usuario *profesor*, tenga que escribir *profesor.X*. Este problema puede ocultarse definiendo una vista[4] con el mismo nombre que la tabla (*X*) en el alumno (CREATE VIEW X AS SELECT * FROM Profesor.X).

Desde el punto de vista funcional, el principal inconveniente de esta aproximación, es que el alumno no puede experimentar por si sólo, creando sus propios casos (crear sus propias tablas, modificar la extensiones de las tablas existentes) para ver cómo varían las respuestas a una misma consulta.

Esta aproximación sólo es válida para cuando el profesor explica sin que el alumno participe, y –aparentemente- para la clase de prácticas guiada. Nótese que si en una clase guiada el profesor, en

[4] O bien un sinónimo (SYNONYM) de la tabla del profesor en el usuario alumno en el caso de aquellos sistemas que dispongan de este tipo de objetos, como por ejemplo *Oracle* [5].

lugar de pedir que dado el enunciado de una consulta los alumnos encuentren la respuesta, pide que dada una SELECT -que parece resolver un enunciado- los alumnos encuentren una extensión de la tabla para la que dicha SELECT no cumple su cometido, los alumnos no pueden realizar dicho ejercicio utilizando el sistema.

Por lo tanto, en este caso se necesita una aproximación que de más libertad al alumno. Para que el alumno pueda cambiar las extensiones de las tablas del profesor necesita poder hacer INSERT-UPDATE-DELETE. Si el profesor comparte sus tablas permitiendo estas operaciones a los alumnos, se verá obligado a pasar todo el tiempo restaurando su copia de seguridad de las mismas, con el disgusto de los alumnos que estuviesen interesados en mantener sus cambios. Por el contrario, si cada alumno tiene sus propias tablas, el profesor no tiene un mecanismo sencillo de propagar sus cambios sobre las mismas a los alumnos. Además los alumnos, serían responsables de recuperar las tablas con los contenidos que tenga el profesor en cada momento.

Por todo ello, vemos que ninguna de las soluciones planteadas son del todo satisfactorias como para impartir clases prácticas que versen sobre el comando SELECT.

2.2. Insert-Delete-Update y la Concurrencia

Los comandos de modificación del contenido de las tablas exigen que los alumnos tengan los permisos correspondientes.

Además, es necesario que el sistema disponga de, al menos, el nivel control de concurrencia *serializable* de SQL/ISO[6] para que los alumnos puedan experimentar con la concurrencia manteniendo -cada alumno- varias sesiones abiertas que accedan sobre los mismos datos. Si los alumnos comparten las mismas tablas, pueden bloquearse entre ellos si no cierran las transacciones, dejando en espera a sus compañeros. Por ello, parece más indicado que cada alumno tenga sus propias tablas.

Sin embargo, como ya se ha comentado en 2.1, si los alumnos trabajan con sus propias tablas, el profesor pierde el control sobre el estado de la base de datos de cada alumno, por lo que le es difícil proponer ejercicios en los que de antemano se sepa que el sistema va a producir el mismo resultado en todos los alumnos. De otro lado, si los alumnos comparten las mismas tablas que el profesor, pudiendo modificar su estado, además los problemas de concurrencia ya mencionados, tampoco el profesor controlaría el estado de la base de datos. Nótese que este caso es muy similar al de la SELECT, y tampoco se tiene una solución adecuada.

Tanto si se comparten o no las tablas del profesor, el crecimiento de las tablas y su ocupación de espacio de disco debe de estar acotado para no perjudicar a otros posibles usuarios del sistema, quizás alumnos de otras asignaturas.

2.3. DDL

Los contenidos a impartir sobre DDL son la creación de tablas, incluyendo restricciones, y vistas. Para que el alumno pueda experimentar se hace necesario que tenga privilegios para crear y eliminar este tipo de objetos. Debido a que los alumnos tenderán a crear tablas con nombres parecidos a los de sus compañeros, habría que habilitar algún mecanismo para que cada alumno tuviese un especio de nombres distinto. En sistemas como *Oracle*[5], donde cada usuario tiene su propio espacio de nombres, la solución es dotar a cada alumno con una cuenta distinta de usuario.

Asimismo, hay que controlar que las tablas que creen los alumnos no puedan crecer indefinidamente, asociándolas una cuota de disco máxima por defecto, y un espacio de disco que no interfiera con otros posibles usos de la base de datos. En este sentido, nuevamente es recomendable que cada alumno sea un usuario distinto; de lo contrario, un alumno puede eliminar o hacer crecer demasiado tablas de sus compañeros intencionadamente o por descuido.

Finalmente, el sistema que elijamos debe de permitir actualizar vistas fácilmente, pues la actualización de vistas es parte de los contenidos a impartir.

Como conclusión, las prácticas DDL *si* que tienen una respuesta adecuada, simplemente dando determinados privilegios a los alumnos, pero sin que puedan tocar las tablas del profesor. La única cuestión relevante es tener un sistema con espacios

de nombres diferenciados por cada usuario, y con posibilidad de actualización de vistas.

3. La Aproximación Basada en Versiones de Tablas

En el apartado anterior se ha visto que el dotar a los alumnos de sus propias tablas origina ciertos problemas, pero que el compartirlas también. La aproximación basada en versiones de tablas que se propone resuelve en buena parte el problema.

Una versión de tabla *T* (tabla base) consiste en otra tabla *T'* (tabla/versión derivada) con la misma estructura y restricciones que *T*, pero con distinta extensión [7]. Las versiones de tablas ya son contempladas por algunos sistemas como *Postgres* [7]. En cierta forma, las vistas son muy parecidas a las versiones. Los cambios de extensión en la tabla base se propagan a las vistas y versiones derivadas. Lo que diferencia a versiones y vistas, es que los cambios en la versión no se propagan a la tabla base, a diferencia de las vistas actualizables.

Nuestra aportación consiste en la aplicación de estos conceptos para obtener un sistema de prácticas de SQL, sobre el que el profesor pueda tener un cierto control sin constreñir demasiado las posibilidades de experimentación libre del alumno. Para ello, las tablas del profesor serán tablas base, mientras que las tablas sobre las que los alumnos experimenten serán versiones derivadas de las tablas del profesor. Los alumnos podrán consultar, pero no modificar, las tablas base; mientras que sí podrán modificar el estado de sus versiones.

Con las versiones de tablas no hay problemas de bloqueos indeseados por concurrencia, pues cada alumno sólo hace cambios sobre sus versiones, y las tablas del profesor se acceden sólo en lectura. Sin embargo, se puede experimentar con concurrencia, sin más que el alumno mantenga varias sesiones simultáneas sobre el mismo usuario, esto es, sobre sus propias versiones.

El profesor tiene control absoluto sobre sus tablas, que no pueden modificarse por los alumnos, pero además puede introducir cambios en las versiones de los alumnos, como efecto de la propagación de los cambios que él haga sobre sus tablas base. Puede introducir filas nuevas, cambiar contenidos de las filas de sus tablas, o eliminarlas, y esos cambios van a propagarse a las versiones que tengan los alumnos, con lo que el profesor puede hacer ciertas suposiciones sobre el estado de las versiones, que le permitan, con mucha probabilidad, saber cual va a ser de antemano la respuesta del sistema, incluso en las distintas versiones de los alumnos.

Por tanto, la aproximación de versiones de tablas es la más adecuada para poder impartir clases guiadas de SQL interactivo, el problema es que no cualquier sistema permite implementarla.

4. Implementación Genérica

La implementación de un sistema basado en versiones de tablas ya viene descrita por encima en [7]. La idea es que por cada tabla que se versione (por ejemplo *Clientes*), existan dos nuevas tablas soporte (en nuestro caso *vadd_Clientes* y *vdel_Clientes*).

La tabla *vadd* tiene la misma estructura que la tabla base, y contiene:

- Las filas que se inserten por el alumno en la versión de la tabla.
- Las filas que el alumno haya cambiado con los valores nuevos.

La tabla *vdel* sólo contiene un identificador que haga referencia a las filas de la tabla base. En el caso de los sistemas objeto-relacionales, como *Postgres*, dicho identificador es el OID (*Object Identifier*). Este identificador indicará:

- Las referencias a filas que los alumnos han eliminado de la tabla base.
- Las referencias a filas de la tabla base que los alumnos hayan cambiado.

Todo ello, permite definir la versión como una vista tal y como muestra la figura.

```
CREATE VIEW VersionClientes AS
(SELECT OID as _OID, CIF, Nombre,etc…
FROM Clientes
WHERE NOT EXISTS
       (SELECT * FROM vdel
       WHERE oid=Clientes.oid)
UNION
(SELECT OID as _OID, CIF, Nombre,etc…
FROM Clientes);
```

Ilustración 1. Vista que implementa una versión de la tabla Clientes.

Mediante *triggers* se controla la actualización de esta vista haciendo:

Caso 1. Que cuando el alumno realice una inserción sobre la versión, el sistema realmente lo que hace es insertar la fila sobre *vadd*. Ver **Ilustración 2**.

Caso 2. Cuando se realiza una eliminación de una fila de la tabla base, lo que se hace realmente es apuntar esa fila como borrada en *vdel*. Ver **Ilustración 3**.

Caso 3. Si se modifica una fila de la tabla base por un alumno, por primera vez, se trata como una baja de la fila con los valores viejos (la fila antes de la modificación es registrada en *vdel*), más un alta de la misma con los valores nuevos (la fila que se obtiene después de la modificación es apuntada en *vadd*). Ver **Ilustración 4**.

Caso 4. Si se elimina una fila que se haya insertado o modificado desde la versión (está en *vadd*), se elimina directamente de *vadd*. Ver **Ilustración 3**.

Caso 5. Si se modifica una fila insertada por el alumno en la versión, o una fila de la tabla base ya modificada por el alumno en la versión (está en *vadd*),simplemente se hace el cambio directamente sobre la fila en *vadd*. Ver **Ilustración 4**.

El tipo de *triggers* que se necesitan para hacer esta implementación necesitan el modo de disparo *instead*, que permite hacer que la acción del *trigger* se ejecute sustituyendo al evento que lo dispara. Es decir, si el evento es una operación de modificación (p.e. una inserción) sobre la vista, esta operación no se ejecuta, y en su lugar se ejecuta la acción especificada en el *trigger* (p.e. la inserción sobre *vadd*). Los *triggers instead* no están disponibles en la mayoría de los sistemas actuales, lo que limita el número de gestores en el que se puede aplicar la aproximación de versiones de tablas.

2. Implementación de la Aproximación de Versiones en PostgreSQL

En el presente apartado se muestra la implementación en PostgreSQL.

Regla de inserción:

Implementa el Caso 1 del apartado anterior.

```
CREATE RULE versionClientes_ins AS
ON INSERT TO versionClientes
DO INSTEAD
//Caso 1
INSERT INTO vadd_Clientes
  CIF, nombre, etc…) VALUES
  (NEW.CIF, NEW.nombre, etc…);
```

Ilustración 2. Regla de Inserción

Regla de borrado:

Implementa los Casos 2 y 4 del apartado anterior.

```
CREATE RULE versionClientes_del AS
ON DELETE TO versionClientes
DO INSTEAD (
//Caso 2
INSERT INTO vdel_Clientes(DOID)
   SELECT Clientes.OID FROM Clientes
   WHERE OLD._OID = Clientes.OID
     AND OLD._OID NOT IN
    (SELECT DOID FROM vdel_Clientes);
//Caso 4
DELETE FROM vadd_Clientes
   WHERE vadd_Clientes.oid = OLD._oid;
);
```

Ilustración 3. Regla de Borrado

Un inconveniente de la implementación *PostgreSQL* es que las vistas no tienen OID. Ni siquiera tienen el OID de la tabla de la que se derivan, por lo que es necesario definir un campo en la vista que deliberadamente contenga el valor del OID de las filas que la componen (el campo _OID en **Ilustración 1**). Este campo lo llamaremos _OID y como se puede ver es utilizado en la Regla

de Borrado y en la de Actualización (ver más adelante) por los *triggers*.

Otra observación interesante es que *vdel* como tabla, ya tiene su propio OID, por ello es necesario que su único campo, que contiene los OIDs de las filas borradas/modificadas, tenga un nombre distinto de OID. En nuestro cado hemos elegido como nombre DOID, que también se utiliza en las Reglas de Borrado y Actualización.

Regla de actualización:

Implementa los Casos 3 y 5 del apartado anterior.

```
CREATE RULE versionClientes_upd AS
ON UPDATE TO versionClientes
DO INSTEAD(

//Caso 3
INSERT INTO vadd_CLIENTES
  (CIF, nombre, telefono)
   select NEW.CIF, NEW.nombre, etc…
   FROM Clientes
   WHERE old._OID = Clientes.OID;

INSERT INTO vdel_Clientes(DOID)
  SELECT oid FROM Clientes
  WHERE old._oid = Clientes.oid;

//Caso 5
UPDATE vadd_Clientes
SET CIF=NEW.CIF, nombre=NEW.nombre,
telefono=NEW.telefono
WHERE vadd_Clientes.oid=OLD._oid;
);
```

Ilustración 4. Regla de Actualización

La implementación mostrada sólo concierne a una de las tablas y a uno de los usuarios. Para que el entorno sea eficaz, deberá de crear automáticamente las vistas y *triggers* correspondientes para cada alumno y versión de tabla del profesor. Este proceso se puede automatizar a través de *scripts* del sistema operativo que consulten el catálogo de la base de datos del profesor. Estos *scripts* se ejecutarían al dar de alta a cada usuario alumno, al principio del curso.

Así, para una tabla *Clientes* del profesor, este *script* crearía automáticamente la versión *Clientes_1* al dar de alta al alumno 1, *Clientes_2* al dar de alta el alumno 2, etc… De esta forma el alumno *n* puede disponer de la versión del profesor –*Clientes*- y de la suya propia –*Clientes_n*.

Un problema que hemos pasado por alto es el de las restricciones. En principio, parece que para que las versiones tengan las mismas restricciones que sus tablas base, basta con volver a declarar dichas restricciones en *vadd*. Sin embargo, esto no es así para las restricciones SQL de PRIMARY KEY, UNIQUE y FOREIGN KEY. Las claves primarias y candidatas han de mantener su unicidad no sólo en la tabla base y en *vadd* por separado, sino que deben tener valores únicos para la unión de las extensiones de ambas tablas. Con las claves foráneas ocurre lo mismo. Si por ejemplo una fila de la tabla *Pedidos* referencia a la tabla *Clientes*, no basta con declarar que *vadd_Pedidos* referencie a *vadd_Clientes*, pues debe poder referenciar además a un cliente de la tabla base *Cientes*.

La solución a estos problemas es controlar las violaciones de estas restricciones, nuevamente a través de *triggers* declarados, en esta ocasión, sobre eventos en *vadd*. En el caso de las claves foráneas los *triggers* controlaran que las inserciones y modificaciones de *Pedidos* y las eliminaciones de *Clientes* no dejan a los pedidos sin cliente asociado. Esta implementación mediante *triggers* es muy flexible, permitiendo implementar distintas semánticas (rechazos, acciones en cascada etc.). En el caso de las claves primarias y candidatas, los *triggers* controlarán que las inserciones y modificaciones de la clave en *vadd* no generen duplicados.

Todos estos *triggers* también pueden ser creados directamente en el *script* de creación de usuario. Aunque las malas noticias son que todos estos controles por *trigger* acaban por degradar el rendimiento del sistema, las buenas noticias son que con el escaso número de filas que suelen tener las bases de datos de docencia, este efecto no es apreciable.

Parte de la materia de SQL interactivo, es el trabajo con vistas actualizables. Un problema de la versión actual de *PostgreSQL* es que las vistas que definan los alumnos no son actualizables, a no ser que se especifiquen vía *triggers*, las acciones asociadas a la propagación de las modificaciones sobre la vista a las tablas base. Sin embargo, este

problema tiene solución, como se indica en el apartado de trabajos futuros.

5. Conclusiones y Trabajos Futuros

La docencia en SQL es recomendable abordarla a través de clases prácticas. El entorno en el que se realicen las prácticas y su configuración es muy importante para poder aplicar metodologías docentes que permitan al alumno experimentar por sí mismo y que sean compatibles con la compartición de un esquema de base de datos inicial del profesor. Se han identificado posibles problemas y soluciones de las aproximaciones tradicionales de cómo abordar las prácticas de SQL, teniendo en cuenta si se trata de explicar el comando SELECT, los comandos INSERT-DELETE-UPDATE o el DDL. Los problemas detectados tienen su orígen en que unas soluciones dan demasiada libertad al alumno, y otras son demasiado restrictivas.

Las versiones de tablas han demostrado que son una solución que mantienen un punto de equilibrio entre dar demasiada libertad al alumno, y otorgar un cierto control al profesor sobre el contenido de los datos.

Un problema de esta aproximación es que no es posible implementarla en cualquier tipo de Sistema de Base de Datos, siendo necesario que el sistema soporte *triggers* en modo *instead*, y que los triggers puedan definirse a partir de eventos que surjan durante el intento de actualización de una vista. Se ha dado una implementación para *PotgreSQL* que es un sistema gratuito y fiable que sí tiene estas características en cuanto a *triggers*.

Un inconveniente de la aproximación que se ha implementado es que no es directamente utilizable desde web, por lo que no sirve para que el alumno pueda practicar desde casa, salvo que le proporcionemos una entrada *Telnet* a lo que probablemente se opongan los responsables de seguridad de nuestro sistema. Por ello, estamos trabajando en un cliente web que permita interactuar con el sistema. Nuestro propósito final es que dicho cliente web admita las consultas en álgebra y cálculo relacional; lenguajes de consulta que le son propios a la misma asignatura de Bases de Datos en la que se revisa el SQL interactivo. Para poder implementar esta mejora desarrollaremos un *parser* en el lado del servidor que traduzca las consultas a *PostgreSQL*. Otra función futura del *parser* será la creación transparente al alumno de los *triggers* necesarios para la actualización de vistas definidas por los alumnos sobre tablas y versiones de su usuario.

Agradecimientos

Por su colaboración a los alumnos Laura Izquierdo Rodrigo y Alberto Tovar González.

Referencias

[1] P. Blesa, N. Brisaboa, V. Canivel, J. Garbajosa, J. Maudes, M. Piattini, A. Urpi. *Contenidos en Bases de Datos de los Planes de Estudio de Informática*. Actas de la III JIDB. Eds. J.C. Casamayor, M. Celma, L. Mota, M.A. Pastor. DSIC, Universidad Politécnica de Valencia. Valencia, 1998.

[2] P. Blesa, N. Brisaboa, V. Canivel, J. Garbajosa, J. Maudes, M. Piattini. *Propuesta de Contenidos en Bases de Datos de los Planes de Estudio de Informática*. Novática, Enero/Febrero 1999. pp 60-63.

[3] Dietrich, S. W. *An Educational Tool for Formal Relational Query Languages*, Computer Science Education. Vol 4, pp.157-184, 1993.

[4] Dietrich, S. W., Eckert, E., Piscator, K., *WinRDBI: A Windows-Based Relational Database Educational Tool*, Proceedings of the 28th ACM SIGCSE Technical Symposium on Computer Science Education, San Jose, California, pp. 126-130, de 27-02 a 01-03 de 1997.

[5] Loney K., Koch G. *Oracle 8i the Complete Reference*. Oracle Press 2000.

[6] Melton J., Simon A.R. *Understanding the new SQL: a Complete Guide*. Morgan Kaufmann 1993.

[7] Potaminos S. & Stonebraker M. *The Postgres Rule* System. En Widom, J & Ceri, S. *Active Database Systems. Triggers and Rules For Advanced Database Processing*, Morgan Kaufmann Publishers, Inc. San Francisco, California. 1996.