ADMINISTRACIÓN DE BASE DE DATOS CON POSTGRESQL LABORATORIO 3. PROCEDIMIENTOS ALMACENADOS Y DISPARADORES

Luis Antonio Álvarez Oval
loval@unach.mx

Universidad Autónoma de Chiapas

Para citar este artículo:
Álvarez, L. (2015) Administración de base de datos con postgreSQL Laboratorio 3. Procedimientos almacenado y disparadores. *Espacio I+D Innovación más Desarrollo 4* (8) 193-208. Recuperado de http://espacioimasd.unach.mx/suplemento/espacioimasd_vol4_no_8.pdf

Esta tercera entrega de una serie de laboratorios de Administración de Base de Datos (ABD) enseña el uso de los procedimientos almacenados asociados a los disparadores (triggers) en una tabla de la base de datos. Este trabajo no pretende enseñar a programar procedimientos almacenados en la base de datos, por lo que el lector debe de investigar el tema denominado Lenguaje de Procedimientos de SQL para PostgreSQL (SQL Procedural Language, mejor conocido como PLPGSQL), aunque probablemente en una segunda serie de laboratorios se explique este tipo de programación.

Los laboratorios se han diseñado para proporcionar los conceptos y la experiencia necesarios para conocer detalladamente el sistema, se aprovecha la función de "copiar y pegar" que nos ofrece el sistema operativo Windows para disminuir el esfuerzo del lector en la preparación del ambiente de trabajo y en la solución de los problemas. En la sección denominada "trabajo adicional" se requiere que el lector aplique la experiencia obtenida en la solución de problemas relacionados al tema central del laboratorio. La sección de conceptos básicos muestra la sintaxis de los comandos y da algunas explicaciones del uso de los mismos, este material ha sido tomado del manual de usuario del sistema PostgreSQL el cual está disponible en la página oficial de la herramienta, en algunos casos se ha tomado del sitio oficial en Español. Los conceptos básicos se aplican en torno al mismo proyecto que usaremos en esta serie: "Universidad ACME", el cual es producto de la imaginación del autor, así como la solución práctica de los problemas planteados. Los libros que se ofrecen en la sección de referencias, sirven como consulta para apoyar algunos de los conceptos que se aplican en la solución práctica de problemas de administración de base de datos.

Estos laboratorios se han preparado para procurar experiencia práctica a los estudiantes de la materia Administración de Base de Datos de la Licenciatura en Sistemas Computacionales que se ofrece en la Facultad de Contaduría Pública (FCP) del Campus IV de la Universidad Autónoma de Chiapas (UNACH). En la FCP se tienen

por lo menos 14 años de experiencia en el uso de PostgreSQL en las aulas, proyectos de investigación y en sistemas que se han implementado para la automatización de las actividades cotidianas de la FCP. Como producto de esa experiencia académica e industrial se han obtenido estos laboratorios que se usan en las aulas para capacitar a nuestros estudiantes. También se tiene noticia de que son una fuente de consulta para egresados que laboran en el sector empresarial.

Como se ha mencionado previamente la herramienta tiene características y lenguajes de programación estándar que ofrecen sistemas propietarios, por lo que los ejemplos fácilmente pueden ser aplicados en otros sistemas de bases de datos del mercado, o pueden ser referencia para aplicar los conceptos en proyectos industriales. Por lo que puedan servir como consulta a profesionales de las Ciencias de la Computación.

OBJETIVO

El lector aprenderá a usar Procedimientos Almacenados y su relación con los Disparadores en la Base de Datos.

PRERREQUISITOS

Se espera que el lector tenga experiencia previa en el uso y conversión de diagramas Entidad-Relación (E-R), los temas asociados al Diseño de Base de Datos no se cubren en este documento. También se espera que el lector tenga conocimientos de programación en cualquier lenguaje de programación y si necesita información adicional del PLPGSQL, se sugiere que visite el sitio: http://www.postgresql.org/docs/9.3/static/plpgsql.html, o busque esta información en el libro “PostgreSQL” de los autores Susan y Korry Dougles (2005).

Finalmente, es necesario instalar la base de datos PostgreSQL versión 9.3 sobre el sistema operativo Windows, verifique los

requerimientos para instalación en la página oficial de la herramienta: www. postgresql.org. El sistema puede descargarse del sitio Web:

http://www.enterprisedb.com/products-services-training/pgdownload#windows

Si tiene alguna duda con respecto a PostgreSQL, le recomiendo visitar el sitio oficial con información publicada en idioma español: http://www.postgresql.org.es/primeros_pasos.

PARTES QUE COMPONEN ESTE LABORATORIO

1. Proyecto a desarrollar
2. Conceptos básicos
3. Preparación del ambiente de trabajo
4. Problemática a resolver
5. Trabajo adicional
6. Referencias

1. PROYECTO A DESARROLLAR

El ejercicio que se va a realizar consiste en un proyecto que describe el problema de una empresa dedicada a la prestación de servicios educativos: después de leer el texto se genera el diagrama E-R con la solución a este problema, se continúa con la creación de las tablas y población de las tablas, para finalmente trabajar con los permisos de grupos y usuarios.

Proyecto Universidad ACME

En UACME, se ofrecen dos tipos de cursos en el periodo especial de verano, en el cual se imparten cursos de verano y cursos extracurriculares. Los primeros son materias que un alumno regu-

lar que estudia una carrera cursa en este periodo, se le permite adelantar hasta dos materias; mientras que los segundos son cursos especiales de capacitación que se ofrecen a alumnos regulares como estudiantes o profesionistas externos.

Los docentes de la UACME, son los únicos a los que se les permite impartir estos cursos, por los cuales recibe un pago adicional, se les paga de acuerdo a un tabulador que indica el costo de la hora de estos cursos de acuerdo al nivel académico del docente. El pago se genera a partir del alta del curso y solo se permite expedir un cheque por cada curso. Además los estudiantes deben acudir a pagar adicionalmente al costo del semestre por asistir a ellos.

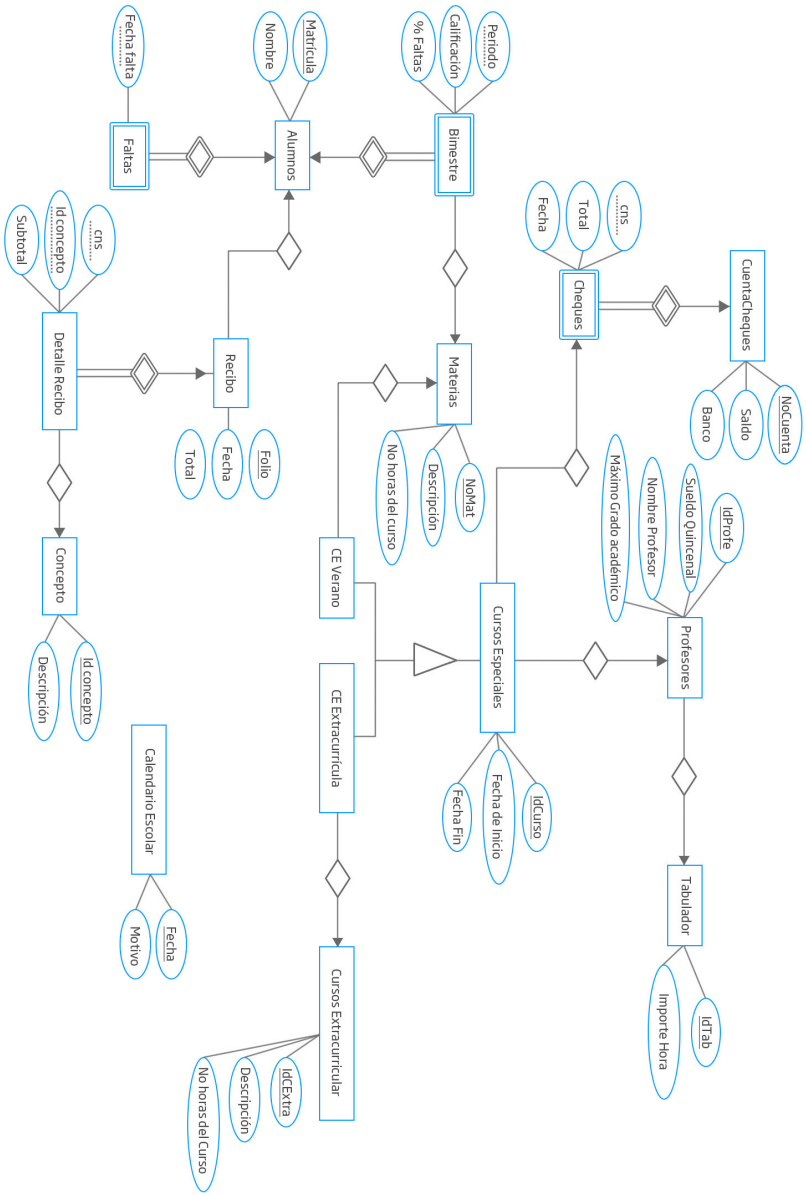
UACME tiene dos departamentos que intervienen en la administración de los cursos:

A) Departamento de Administración (DA) y B) Departamento de Control Escolar (DCE). Corresponde al DA, efectuar el pago a los docentes y los cobros a los alumnos. El DA es dirigido por el C.P. Ávila y es auxiliado por el Sr. Cancino. Mientras que el DCE, es dirigido por el Lic. Barroso y auxiliado por los Sras. Tirado, Martínez, Aquino y Ramos y es en este donde se decide cuales cursos se imparten en el periodo, quién los imparte, y se aceptan las solicitudes de los alumnos. Un caso especial, es el de los Profesores, ya que el DA es quién les puede modificar el sueldo quincenal, mientras que el DCE ni siquiera puede visualizar éste. Lo curioso radica en que, es el DCE quién acepta los docentes y los registra en el sistema, pero es el DA donde se captura el sueldo. Importante es para la administración de la UACME que esta política se aplique al pie de la letra, y que sea implementado directamente sobre la DB. A continuación se describen detalladamente las tablas a las cuales tiene acceso el personal de la Secretaría Administrativa: CuentaCheques, Cheque, Tabulador, Profesores, Concepto, Recibo, y DetalleRecibo. En casos especiales, este departamento podrá acceder a consultar las tablas de Cursos Especiales, Cursos Especiales Verano, Cursos Especiales Extracurriculares,

Cursos Extracurriculares y Materias. Explícitamente no se les permite modificar ningún campo o registro.

Tablas a las que se le permite el acceso al personal de la Secretaría Escolar: CursosEspeciales, CursosExtracurricular, Materias, CEVerano, CEExtracurricula, Alumnos, Bimestre, Faltas, CalendarioEscolar.

Figura 1. Diagrama E/R que resuelve el problema anterior.



CONCEPTOS BÁSICOS

Disparadores

Postgres tiene algunas interfaces cliente como Perl, Tcl, Python y C, así como dos *lenguajes Procedimentales* (PL). También es posible llamar a funciones C como acciones disparador. Actualmente es posible especificar BEFORE o AFTER en los INSERT, DELETE o UPDATE de un registro como un evento de disparo.

Creación de Disparadores

Si un evento disparo ocurre, el administrador de disparos (llamado Ejecutor) inicializa la estructura global TriggerData *CurrentTriggerData y llama a la función de disparo para procesar el evento. La función de disparo debe ser creada antes que el disparador, y debe hacerse como una función sin argumentos de entrada, y códigos de retorno opacos.

La sintaxis para la creación de disparadores es la siguiente:

CREATE TRIGGER <trigger name> <BEFORE|AFTER> <INSERT|DELETE|UPDATE> ON <relation name> FOR EACH <ROW|STATEMENT> EXECUTE PROCEDURE <procedure name> (<function args>);

<TRIGGER NAME> se usa para identificar al disparador y también se usa como argumento del comando DROP TRIGGER si se desea eliminarlo.
<BEFORE|AFTER> determina si la función debe ser llamada antes (BEFORE) o después (AFTER) del evento.
<INSERT|DELETE|UPDATE> determina en que evento/s será llamada la función. Es posible especificar múltiples eventos utilizado el operador OR.

<RELATION NAME> determinará la tabla afectada por el evento.
<FOR EACH> determina si el disparador se ejecutará para cada fila afectada o bien antes (o después) de que la secuencia se haya completado.
<PROCEDURE NAME> es la función invocada cuando se ejecute el disparador. Esta función puede ser escrita en cualquier de los lenguaje procedimentales o las interfaces cliente que maneja PostgreSQL.
<FUNCTION ARGS> Son los argumentos pasados a la función en la estructura CurrentTriggerData. El propósito de pasar los argumentos a la función es permitir que disparadores diferentes con requisitos similares invoquen a la misma función. Además, la función puede ser utilizada para disparar distintas relaciones (estas funciones son llamadas "general trigger functions").

Como ejemplo de utilización de lo antes descrito, se puede hacer una función general que toma como argumentos dos nombres de campo e inserta el nombre del usuario y la fecha (timestamp) actuales en ellos. Esto permite, por ejemplo, utilizar los disparadores en los eventos INSERT para realizar un seguimiento automático de la creación de registros en una tabla de transacciones. Se podría utilizar también para registrar actualizaciones si es utilizado en un evento UPDATE.

Las funciones de disparo retornan un área de tuplas (HeapTuple) al ejecutor. Esto es ignorado para disparadores lanzados después (AFTER) de una operación INSERT, DELETE o UPDATE, pero permite lo siguiente a los disparadores:

BEFORE: retornar NULL e ignorar la operación para la tupla actual (y de este modo la tupla no será insertada/actualizada/borrada); o devolver un puntero a otra tupla (solo en eventos INSERT y UPDATE) que serán insertados (como la nueva versión de la tupla actualizada en caso de UPDATE) en lugar de la tupla original.

Note que no hay inicialización por parte del manejador CREATE TRIGGER. Además, si más de un disparador es definido

para el mismo evento en la misma relación, el orden de su ejecución es impredecible. Si una función de disparo ejecuta consultas SQL, entonces estas funciones pueden disparar a otros disparadores. Esto es conocido como disparos en cascada. No hay ninguna limitación explicita en cuanto al número de niveles de cascada.

Si un disparador es lanzado por un INSERT e inserta una nueva tupla en la misma relación, el disparador será llamado de nuevo (por el nuevo INSERT). Actualmente, no se proporciona ningún mecanismo de sincronización para estos casos.

Los disparadores son parte de lo que se conoce como "elementos activos" de una base de datos. Así como lo son las restricciones tales como NOT NULL, FOREIGN KEY, PRIMARY KEY, CHECK. Una vez definidas ellas "se activaran" solo al ocurrir un evento que las viole, un valor nulo en un campo con NOT NULL, etc.

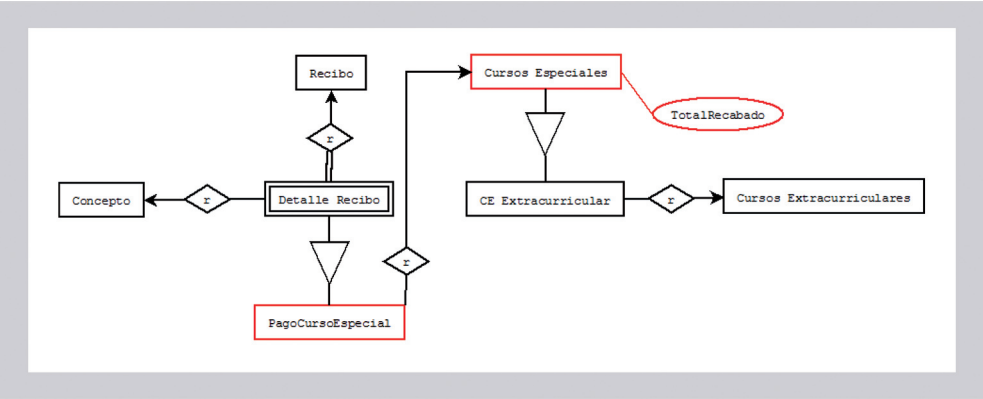
Con los disparadores se intenta dar más control al programador sobre los eventos que desencadenan un elemento activo, se les conoce en inglés como reglas ECA (event-condition-action). Es por ello que los disparadores tienen una clausula BEFORE, AFTER o INSTEAD y bajo que evento (INSERT, UPDATE, DELETE) pero de esta forma el disparador se ejecutará para cada tupla (o fila) sometido al evento (clausula FOR EACH ROW) pero el estándar dicta que puede ser también FOR EACH SENTENCE. Esto provoca que se ejecute el disparador para toda la relación (o tabla) para la cual se define (clausula ON).

La diferencia para los que lo han programado, por ejemplo en plpgsql, queda clara entonces: cuando es FOR EACH ROW en la función pgsql que implementa el disparador se tiene un objeto NEW y uno OLD que se refiere a la tupla completa, en el disparador de oraciones (statement) tiene un objeto NEW y OLD que son la relación completa. Está claro entonces que es un poco más difícil implementar un disparador para oraciones que para una fila.

Consultas, Procedimientos Almacenados y Disparadores

Para aplicar los disparadores a nuestra base de datos tenemos que hacer un ligero cambio en la base de datos. Este cambio consiste en que al momento de pagar un alumno su inscripción a un curso extracurricular se le debe preguntar el *idcurso* al que se va a inscribir, relacionarlo con el folio y consecutivo del detalle del recibo en el que ha pagado el mismo y automáticamente se debe de incrementar el campo *Total Recabado* en la tabla *Cursos Especiales*, esto con el fin de que el DCE vea el ingreso económico de este curso. La figura 2 muestra solo la pequeña modificación que debe hacerse al diagrama. Las entidades y relaciones coloreadas en rojo son las únicas modificaciones de este diagrama, las demás entidades permanecen intactas y solo se han incluido para referencia.

Figura 2. Cambio a la Base de Datos para el laboratorio 3.



El problema consiste que cuando se detecte que el alumno Juan Pérez ha pagado el concepto 50 (pago de cursos especiales) por el Curso 75, el cual es un curso extracurricular de Java Básico y que será impartido del 10 al 30 de Enero del 2008 y por el cual va a pagar $4,500 pesos. En el momento en el que el Recibo y el Detalle del Recibo se actualicen, también se actualizará la tabla

PagoCursoEspecial, entonces automáticamente se debe de incrementar el importe recabado para el curso que se ha pagado en la tabla *CursosEspeciales*. Asimismo, si el recibo es cancelado, entonces el importe recabado en la tabla *CursosEspeciales* debe ser disminuido. Los cambios efectuados se ven reflejados en las siguientes definiciones de tablas:

```
-- creando la nueva tabla
create table PagoCursoEspecial(
folio int,
cns int,
idcurso int references CursosEspeciales,
primary key(folio, cns),
foreign key(folio, cns) references
DetalleRecibo
);

-- borrando la tabla de CursosEspeciales para
después redefinirla, cuidado con el cascade
-- ya que si tiene datos relacionados con ella
puede llegar a borrar varias tablas y puede
-- tener que volver a crearlas a partir del
laboratorio 1.
drop table CursosEspeciales cascade;

-- redefiniendo la tabla de CursosEspeciales
con los cambios requeridos
create table CursosEspeciales(
idcurso int,
idprofe int,
fini varchar,
ffin varchar,
ncuenta int,
cns int,
TotalRecabado numeric(10,2),
CostoCurso numeric(10,2),
foreign key(idprofe) references Profesores,
foreign key(ncuenta, cns) references Cheque,
primary key (idcurso)
```

```
);

Ahora, preparamos el entorno para trabajar
con los elementos adecuados, ejecute los
siguientes comandos con el usuario postgres:

-- Insertando datos de alumnos
insert into alumnos values (100, 'Juan Perez');

-- Insertando datos de concepto
insert into concepto values (50, 'Pago de
Cursos Especiales');

-- Insertando datos del CursosEspeciales
insert into CursosEspeciales values ( 75, 5,
'2008-01-10', '2008-01-30', 3, 10, 0.0, 4500 );

-- Insertando datos del CEExtracurricula
insert into CEExtracurricula values ( 75, 4 );

-- Insertando datos de Recibo
insert into Recibo values(200, 100, '2008-02-
10', 4500 );

-- Insertando datos de Detalle de Recibo
insert into DetalleRecibo values ( 1, 50, 200,
4500 );
```

Ésta es la función denominada ActualizadorTotal, cuya tarea es la de hacer el incremento del total recabado. Está escrita con el lenguaje denominado PLPGSQL, y debe copiarse usando la cuenta del usuario postgres. Recuerde que para poder copiar este programa debe de tener instalado el plpgsql en su base de datos. Si aún no tiene instalado el lenguaje PLPGSQL, use el comando create language, para mayor información visite la página: http://www.postgresql.org/docs/9.3/static/sql-createlanguage.html

```
DROP FUNCTION IncrementadorTotal() CASCADE;
CREATE OR REPLACE FUNCTION
IncrementadorTotal() RETURNS trigger AS '
DECLARE
vreg record;
vrec record;
vnuevo numeric(10,2);
BEGIN
-- Selecciona el curso de la tabla
CursosEspeciales
select into vreg * from CursosEspeciales where
idcurso = new.idcurso;
-- Selecciona el importe pagado por cada curso
en el detalle de recibo
select into vrec * from DetalleRecibo where
folio = new.folio and cns = new.cns;
-- Incrementa el pago del alumno
vnuevo := vreg.TotalRecabado + vrec.subtotal;
-- Actualizar TotalRecabado
update CursosEspeciales set TotalRecabado =
vnuevo where idcurso = new.idcurso;
return new;
END;
' LANGUAGE 'plpgsql';
```

A continuación tenemos los comandos necesarios para la creación del disparador, en la definición de este se hace referencia a la función definida antes.

```
-- Eliminando el disparador, por si acaso.
Drop Trigger TgIncrementaRecabado ON
PagoCursoEspecial CASCADE;

-- Creación del disparador, cuyo nombre es
TgIncrementaRecabado. Este se dispara después
```

```
-- de hacer una inserción en la tabla
PagoCursoEspecial, para cada renglón insertado
se
-- debe ejecutar la función
IncrementadorTotal().

CREATE TRIGGER TgIncrementaRecabado AFTER
INSERT ON PagoCursoEspecial FOR EACH ROW
EXECUTE PROCEDURE IncrementadorTotal();

-- Consulte el total recabado para el curso
antes del pago
select * from CursosEspeciales;

-- Insertando datos en PagoCursoEspecial,
¡el alumno paga su curso! insert into
PagoCursoEspecial values( 200, 1, 75 );

-- Consulte el total recabado para el
curso después del pago select * from
CursosEspeciales;
```

Ésta es la función DecrementadorTotal(), cuya tarea es la de decrementar el total recabado. Está escrita con el lenguaje denominado PLPGSQL, y debe copiarse usando la cuenta del usuario postgres.

Finalmente, ahora construimos un disparador para cuando alguien que había pagado un curso y decide cancelarlo, ante esta situación el total recaudado debe de disminuir.

```
-- Eliminando el disparador, por si acaso.
Drop Trigger TgDecrementaRecabado ON
PagoCursoEspecial CASCADE;

-- Creación del disparador, cuyo nombre es
TgDecrementa Recabado. Este se dispara
-- después de efectuar el borrado de un regis-
tro en la tabla PagoCursoEspecial, para cada
-- renglón insertado se debe ejecutar la fun-
ción DecrementadorTotal().
```

```
CREATE TRIGGER TgDecrementaRecabado AFTER
DELETE ON PagoCursoEspecial FOR EACH ROW
EXECUTE PROCEDURE DecrementadorTotal();
-- Consulte el total recabado para el curso
antes del pago
select * from CursosEspeciales;

-- Borrando datos en PagoCursoEspecial, ¡se
cancela el recibo!
Delete from PagoCursoEspecial where folio =
200 and cns = 1;

-- -- Consulte el total recabado para el curso
después del pago
select * from CursosEspeciales;
```

TRABAJO ADICIONAL

Modifique el disparador para que cumpla con la siguiente condición: El alumno Juan Pérez ha decidido de última hora que va a asistir a dos cursos extracurriculares, “Java Básico” que es el curso 75 y “Programación Avanzada de Procedimientos Almacenados” que es el curso 100. Así que va a efectuar el pago y la cajera en el mismo recibo (el folio 600) le cobra los dos cursos. ¿Qué modificaciones debe hacer en el programa que incrementa el total recabado? Y ¿En el qué lo disminuye?

- En apariencia las dos funciones hacen casi lo mismo, ¿Puede optimizar el trabajo? Es decir, hacer una sola función que sea capaz de controlar tanto el incremento como el decremento del total recabado.

REFERENCIAS

Douglas Korry y Susan Douglas. (2005) *PostgreSQL A comprehensive guide to building, programming and administering Postgre SQL databases. (2nd. Edition)*. Sams

Elmasri, R.; Navathe, S.B. (2002) *Fundamentos de Sistemas de Bases de Datos*. 3ª Edición. Addison-Wesley

Garcia-Molina, Jeffrey Ullman y Jennifer Widom. (2008) *Database systems: the complete book*. Prentice-Hall.

Momjiam Bruce (2001) *PostgreSQL Introduction and Concepts*. Boston: Addison-Wesley Logman Publishing Co. Inc.

Silberschatz Abraham, Henry Korth y S. Sudarshan. (2006) *Fundamentos de Base de Datos* (5a. Edición). España: McGraw-Hill

The PostgreSQL Global Development Group. (2015) *Manual de postgreSQL*. Disponible en www.postgresql.org