Buenas prácticas en el diseño de bases de datos

Base de Datos oñembosako'i hag̃ua hekópe

Best Practices in Database Design

José Alberto Giménez

Universidad Tecnológica Intercontinental

Nota del autor

Licenciatura en Análisis de Sistemas Informáticos

gimenezj@outlook.com

Resumen

Esta monografía presenta algunas técnicas orientadas a ayudar al diseñador a mejorar sus proyectos de construcción de bases de datos mediante el uso de estándares y prácticas de diseño que permitan mejorar el rendimiento y seguridad de una base de datos. Los métodos mencionados no tienen mucha dificultad para ser aplicados y forman parte de un conjunto de buenas prácticas que se recomiendan para el diseño de bases de datos y tienen como objetivo final obtener más facilidad en el mantenimiento de una base de datos y la mejora en la experiencia de uso por parte de los usuarios. Se exponen técnicas de uso de nomenclaturas estándar así como los principios de normalización de bases de datos hasta la tercera forma normal. Para el mejorar el rendimiento y la seguridad al momento de manipular y operar los datos se desataca la importancia de la utilización de procedimientos almacenados y no operar directamente los datos desde la aplicación. Las buenas prácticas recomendadas son independientes de las herramientas utilizadas para el diseño o gestión de una base de datos.

Palabras clave: bases de datos, buenas prácticas, normalización, procedimientos almacenados, under score, formas normales.

Mombykypyre

Ko jeporekapýpe ojekuaauka tapereko oporoipytyvõkuaáva omba'ediseñávape ojapoporãvévo marandu renda oiporúvo umi tembiapokue oĩmavavoi ha omba'apóvo diseño apópe hekópe, ikatu hagüaicha osẽporãve hembiapokue ha ta'isegurove umi marandu renda. Umi tapereko oñeñe'ẽha ndahasýi ojeporu hag̃ua ha oĩ umi pojoapy ojeporukuaáva ha ojejeruréva tojeporu oñembosako'i

potávo marandu renda apytépe, ha ikatukuaáva ombohasy'ỹve oñemantene hagua, ha iporuharakuéra oiporuporãve rekávo. Ojehechauka aporeko ojeporuhápe téra opavave oikuaáva, ha avei marandu renda jeporukatui renda, tercera forma peve. Jahechápa ojeporuporãve ha areve ha ta'isegurove umi marandu ojepoko jave hese, ojehechakuaa iporãveneha ojeporu umi marandu oñeñongatupyre ha ani pe aplicación guive. Oimeraẽ herramienta ikatu ojeporu hekópe, oñemoheñói ha ojeporu potávo umi marandu renda.

Mba'e mba'e rehepa oñeñe'ẽ: marandu renda, jeporu porã, jeporukatui, marandu ñeñongatu, under score, forma normal.

Abstract

This monograph presents some techniques aimed at helping designers to improve their database construction projects by the using of design standards and practices that improve performance and security of a database. The mentioned methods do not present much difficulty in being applied and are part of a set of good practices that are recommended for the design of databases and have as a final goal the easier obtaining of database management and improved user experience. The techniques of use of standard nomenclatures are exposed as well as the principles of database normalization up to the third normal form. To improve the performance and security when manipulating and operating the data, the importance of using stored procedures and not directly operating the data from the application is highlighted. The good practices recommended are independent of the tools used for the design or management of a database.

***Keywords*:** databases, good practices, standardization, stored procedures, under score, normal forms.

Fecha de recepción: 21/03/2019

Fecha de aprobación: 10/05/2019

Buenas prácticas en el diseño de bases de datos

El objetivo principal de este escrito es presentar un conjunto de reglas de buenas prácticas para el diseño de base datos. Se expondrán una serie de recomendaciones que se pueden aplicar al momento de crear objetos en una base de datos y técnicas que contribuyan a mejorar el rendimiento. Se destaca la importancia de no pensar en la implementación a nivel físico sino centrar los aspectos del diseño dentro de un marco conceptual.

Las técnicas expuestas aquí han sido recopiladas de diferentes fuentes bibliográficas. El trabajo está organizado en tres secciones: consejos sobre nomenclaturas, buenas prácticas de diseño para evitar inconsistencias en las operaciones y consejos para aumentar el rendimiento al momento de manipular los datos.

En el desarrollo del trabajo se plantean las siguientes preguntas:

- ¿Qué convenciones se pueden usar para mejorar la nomenclatura de objetos en una base de datos?
- ¿Qué técnicas deben usarse para evitar comportamientos anómalos en las operaciones y consultas de datos?
- ¿Qué prácticas de tratamientos de datos pueden mejorar el rendimiento y la seguridad una base de datos?

Los consejos están limitados al diseño, por lo que no se abordarán buenas prácticas relacionadas a otro tipo de operaciones como transacciones, funciones o creación de disparadores[1].

Convenciones para nomenclaturas de objetos en una base de datos

Importancia del uso de estándares en nomenclaturas

Una vez que se realiza el diseño de una base datos y se establecen cuáles son las tablas y los campos[2] que incluirá, es importante tener en cuenta la nomenclatura. En ocasiones, se tienen en cuenta preferencias personales más que reglas o convenciones, sin embargo, es importante mantener una

[1] Los triggers o disparadores son objetos que se asocian con tablas y se almacenan en la base de datos. Su nombre se deriva por el comportamiento que presentan en su funcionamiento, ya que se ejecutan cuando sucede algún evento sobre las tablas a las que se encuentra asociado. Los eventos que hacen que se ejecute un trigger son las operaciones de inserción (INSERT), borrado (DELETE) o actualización (UPDATE), ya que modifican los datos de una tabla.

[2] En una tabla, un campo es un campo es un espacio de almacenamiento para un dato en particular.

nomenclatura que permita coherencia y consistencia en las notaciones. Esto permitirá una mejor comprensión y autodocumentación de nuestros objetos de bases de datos.

En primer lugar debemos nombrarla tabla y luego sus campos. Para Martin (2009) es importante usar nombres que revelen intenciones. Se recomienda que los nombres de los campos de una tabla faciliten la comprensión del dato/objeto nombrado. Si un nombre requiere un comentario, significa que no revela su contenido (Martin, 2009). Esta regla siempre estará orientada a facilitar al diseñador o lector a saber de qué se trata el campo, qué datos está almacenando o qué entidad representa exactamente una tabla.

Al momento de seleccionar un estándar para las nomenclaturas, los equipos de trabajo pueden inventar su propio sistema de nomenclaturas. No se trata de imponer uno u otro estilo, sino contar con una estandarización. Existen varios estilos que han sido desarrollados como por ejemplo, la notación *Camel Case*, *Notación Húngara* y la notación *snake_case o under_score*. Estas convenciones o estándares de nomenclatura son un conjunto de normas para un lenguaje de programación específico y se recomiendan como buenas prácticas para facilitar la lectura del código y sea más fácilmente entendible y más sencillode mantener. Los mismos criterios se pueden aplicar a la notación de objetos de una base de datos.

En este trabajo se recomienda el uso de la notación *snake_case*, en la cual los nombres se definen con palabras que se separan por un guion bajo (_). Esta recomendación está basada en un estudio presentado en la 18º Conferencia Internacional sobre Comprensión de Programas de la IEEE (*Institute of Electrical and Electronics Engineers*[3]) del año 2010, en donde se presentó un estudio empírico para determinar si las convenciones afectan la comprensión de código.

En este estudio los resultados indicaron que si bien no hay diferencias en la precisión entre los estilos *Camel Case* y *under_score*, los sujetos reconocen los identificadores en el estilo de guion bajo más rápidamente (Sharif & Maletic, 2010).

La nomenclatura *under_score*

El uso de caracteres de subrayado como separadores de palabras en identificadores es antiguo, de finales de la década de 1960. Estuvo originalmente asociado al lenguaje de programación *C* y no tuvo un nombre

[3] El Instituto de Ingeniería Eléctrica y Electrónica (conocido por sus siglas IEEE) es una asociación mundial de ingenieros dedicada a la estandarización y el desarrollo en áreas técnicas.

específico hasta que Gavin Kistner la menciona en un foro del lenguaje de programación *Ruby* en 2004 como *"snake_case"* (imitando una serpiente y evitando el uso de espacios).Bajo esta nomenclatura los objetos se pueden nombrar haciendo uso de palabras separadas por un guion medio.

Ejemplos de nomenclaturas. Consideremos primero la nomenclatura de las tablas de una base de datos. Si existe una tabla que va almacenar datos de clientes, es conveniente llamarla según la entidad a la que representa: *Clientes*. Algunos diseñadores optan por no usar plural en la nomenclatura de tablas, pero no hay una regla fija sobre esto. Lo importante es usar un nombre con significado que sea lo más descriptivo posible sin importar que tan largo sea, siempre y cuando la longitud esté soportada por el motor de base de datos que utilizamos.En el uso de la notación *under_score*, considero práctico evitar el uso de acentos.

Por ejemplo, una tabla que almacena catálogos de productos puede ser nombrada *catologo_productos* en lugar de *catprod*. Aprecie que el primer nombre es mucho más significativo. Una tabla que guarde los teléfonos de clientes será nombrada como: *telefonos_de_clientes* o *telefonos_clientes*. Otros ejemplos de nomenclaturas de tablas usando la notación *under_score* para dar más significado pueden ser:

- *cabecera_ventas*, en lugar de *cabventa* (para almacenar los datos de la cabecera de una venta)
- *fotos_productos* en lugar de *fotprod*(almacena las fotos de productos)
- *resultados_analisis*, en lugar de *resanal*(almacena resultados de un análisis)
- *detalle_ventas*en lugar de *detvent*(para almacenar los datos detallados de una venta)
- *categorias_de_usuarios,* en lugar de *categusu*
- *tipos_de_productos*en lugar de *tprod*
- *fotos_de_productos, cuentas_en_bancos, correos_de_clientes,* etc.

Nombrando los campos de una tabla. Aplicamos el mismo criterio de nomenclatura que se expuso para nombrar tablas. Usamos nombres con significado y en la extensión que consideremos necesaria. Un detalle a tener en cuenta para la nomenclatura de campos, es que siempre debe hacerse en singular, esto surge como parte de la normalización, que se tratará más adelante; el campo siempre debe ser atómico: almacenar un solo valor. Si utilizamos el plural en el nombre de campos se puede dar a entender que no es

atómico.Se tomará como ejemplo un caso típico de diseño en el cual se necesita definir tablas de clientes y proveedores para ofrecer ejemplos de nomenclaturas de campos. No se analizará el contenido o campos de las tablas, ya que estos dependen de los requisitos del negocio, solamente se ofrecen ejemplos para las notaciones.

Tabla clientes: *clientes*

- *numero_de_cedula_cliente*
- *nombre_cliente*
- *apellido_cliente*
- *fecha_de_nacimiento*

Tabla Proveedores: *proveedores*

- *ruc_proveedor*
- *nombre_proveedor*
- *direccion_proveedor*
- *ciudad_proveedor*

Tabla para almacenar correos electrónicos de clientes: *correos_clientes*

- *numero_de_cedula_cliente*
- *correo_de_cliente*

Tabla para almacenar correos electrónicos de proveedores: *correos_proveedores*

- *ruc_proveedor*
- *correo_de_proveedor*

Tabla para almacenar los datos de cabecera de una compra: *cabecera_compras*

- *numero_ boleta_compra*
- *ruc_proveedor*
- *fecha_compra*
- *tipo_de_compra*

Tabla para almacenar los datos de detallede una compra: *detalle_compras*

- *numero_boleta_compra*
- *codigo_articulo*

- *cantidad_comprada*
- *precio_compra*

En estos ejemplos se puede ver cómo utilizar la notación *under_score*. Observe que al nombrar los campos se evita hacer uso de la preposición "de" en el caso de *fecha_compra*. Esto se debe a que el campo se encuentra dentro de la tabla *cabera_compras*, por lo tanto el dato se refiere a la fecha de la compra. Aquí podemos optar por usar como nombre de campo *fecha_de_compra* si considera este nombre con más significado.

Las convenciones de nombres no son difíciles de usar, hay que tomarse el tiempo para establecer una notación convencional en todos nuestros proyectos, usando las que se sugieren en la comunidad de diseñadores y programadores o bien creando nuestra propia convención. La clave es ser consistente.

Normalización de tablas en una base de datos

En esta sección se expondrán las técnicas que evitaran problemas al momento de tratar los datos en una base de datos. Edgar Frank Codd utilizó el término *anomalías* para referirse a los problemas de actualización y comportamientos inesperados en las consultas realizadas a los datos de las tablas de una base de datos (Opell, 2010).

Para evitar estos comportamientos Codd desarrolló una serie de técnicas a las que denominó *formas normales* a fin de evitar estas anomalías, evitar la redundancia de los datos y proteger la integridad de los datos. Con el proceso de normalización (Codd desarrolló originalmente tres) se hacen que las tablas sean menos vulnerables a inconsistencias y anomalías.

Un esquema normalizado es robusto y carece de redundancias, por lo que está libre de ciertas anomalías que las redundancias puede provocar cuando se actualiza una base de datos (Marqués, 2009). Además de reducir estas anomalías, una ventaja adicional del proceso de normalización es que el diseño normalizado hace que las estructuras de datos sean más estables y fáciles de mantener (Hueso, 2014). Estas formas normales deben ser aplicadas a todas las tablas que conforman la base de datos.

En las secciones siguientes se darán ejemplos de cómo normalizar tablas hasta la tercera forma normal.

Primera forma normal

En la primera forma normal, es necesario conocer el concepto de dependencia funcional. Una dependencia funcional en una relación que se da

entre dos atributos *x* e *y*, cuando cada valor de *x* tiene asociado un solo valor de *y* (Hueso, 2014). Por ejemplo, si queremos calcular la edad de una persona, en años, necesitamos conocer la fecha de nacimiento. En este caso existe una dependencia funcional del valor de la edad con la fecha de nacimiento, ya que de esta relación se obtiene solamente un valor.

Las reglas para normalizar una tabla de una base de datos son las siguientes:

- Todas la tablas deben tener una clave primaria (así evitamos redundancias).
- Todos los campos deben ser atómicos (almacenar un solo valor).
- No deben existir campos nulos.
- No debe existir variación en el número de columnas de la tabla.
- Los campos que no son clave deben identificarse por la clave (dependencia funcional).
- Debe existir una independencia del orden tanto de las filas como de las columnas (si cambiamos el orden de los datos, no debe cambiar su significado).

Ejemplo de normalización a la primera forma normal. Se tomará como ejemplo una tabla estándar que almacenará datos de clientes.

clientes
numero_de_cedula
nombre_cliente
apellido_cliente
notas_varias_clientes
telefonos_clientes
mails_clientes

Analizamos la tabla presentada y notamos que no está normalizada a la primera forma normal, ya que: no tiene establecida una clave primaria, el campo *notas_de_clientes* puede contener valores nulos, los campos *telefonos_clientes* y *mail_clientes* no son atómicos, pues el usuario puede cargar más de un teléfono o correo en cada campo. Cuando algún campo esté en contra de una forma normal, este debe ser separado a su propia tabla.

El campo *numero_de_cedula* será la clave primaria (usaremos la sigla *PK*, de *Primary Key* para notar que un campo forma la clave primaria). Así, la tabla clientes normalizada quedará estructurada de la siguiente forma:

clientes
*numero_de_cedula**(PK)***
nombre_cliente
apellido_cliente

Los campos *notas_varias_clientes*así como*telefono_clientes* y *mail_clientes* quedarán en tablas independientes. Nombraremos la tabla según las recomendaciones dadas, usando la nomenclatura *under_score.*

notas_de_clientes
*numero_de_cedula**(PK)***
notas_varias_clientes

telefonos_de_clientes
numero_de_cedula
telefono_clientes

correos_de_clientes
numero_de_cedula
mail_clientes

Las tablas resultantes deben pasar nuevamente por el proceso de normalización a la primera forma normal. Como los datos de los correos, notas y teléfonos corresponden a los clientes, el campo mediante el cual se identifica a cada cliente: *numero_de_cedula* debe estar en cada tabla y formamos de esta forma sus claves primarias:

notas_de_clientes
*numero_de_cedula**(PK)***
*notas_varias_clientes**(PK)***

telefonos_de_clientes
*numero_de_cedula**(PK)***
*telefono_clientes**(PK)***

correos_de_clientes
*numero_de_cedula**(PK)***
*mail_clientes**(PK)***

Esquema final, normalizado a la primera forma normal:

clientes
*numero_de_cedula**(PK)***
nombre_cliente
apellido_cliente

notas_de_clientes
*numero_de_cedula**(PK)***
*notas_varias_clientes**(PK)***

telefonos_de_clientes
*numero_de_cedula**(PK)***
*telefono_clientes**(PK)***

correos_de_clientes
*numero_de_cedula**(PK)***
*mail_clientes**(PK)***

Con la normalización a la primera forma normal evitamos que *notas_de_clientes* pueda ser nulo, ya que si el usuario no desea almacenar una nota u observación relacionada al cliente, este campo simplemente no existirá en la tabla *notas_de_clientes*

Los correos y teléfonos en este esquema normalizado pueden ser varios, es decir, si un cliente tiene más de un correo o teléfono, estos datos aparecerán como registros en la base de datos tantas veces como sea necesario, pero cada registro contendrá un solo dato, será atómico. Se finaliza el proceso por cada tabla para la primera forma normal y a continuación se verifican las reglas de la segunda forma normal.

Segunda forma normal

Para el proceso de normalización a la segunda forma normal, es requisito que todas las tablas se hayan normalizado a la primera forma normal. Se consideran las reglas:

- La tabla debe estar en primera forma normal
- Los campos que no forman parte de ninguna clave dependen de forma completa de la clave primaria. Es decir, no deben existir dependencias parciales. Una dependencia parcial es un término que describe a aquellos datos que no dependen de la clave primaria de la tabla para identificarlos.

Ejemplo de normalización a la segunda forma normal. Para comprender el proceso de normalización a la segunda forma normal, se presentará el siguiente ejemplo de una tabla que registra los pedidos realizados de unos artículos:

numero_pedido(PK)	**codigo_articulo**(PK)	**descripcion_articulo**	**cantidad**	**precio_articulo**
1971	86	RED	3	50000
1971	40	RAQUETA	6	65000
1971	32	PAQ-3	8	47500
1972	94	PAQ-6	4	50000
1973	40	RAQUETA	2	65000
1973	41	FUNDA	2	100000

En esta tabla se estableció que los campos *numero_pedido*y *codigo_articulo* conformaran la clave primaria. Los demás datos representan su descripción, cantidad de pedido y precio correspondiente.

Para aplicar la segunda forma normal se deben realizar los siguientes pasos:

- Determinar cuáles campos que no son clave primaria, no dependen de la clave primaria de la tabla.
- Eliminar esos campos de la tabla analizada.
- Crear una segunda tabla con esos campos y los campos de la *PK* de la cual dependen.

Los campos *precio_articulo* y *descripcion_articulo* dependen del campo*codigo_articulo*, más no así de *numero_pedido*, por lo que aquí tenemos campos que dependen de una parte de la clave primaria y no de la clave completa (dependencia parcial). Aplicamos la segunda forma normal y obtenemos la siguiente estructura:

numero_pedido(PK)	**codigo_articulo**(PK)	**cantidad**
1971	86	3
1971	40	6
1971	32	8
1972	94	4
1973	40	2
1973	41	2

codigo_articulo(PK)	**descripcion_articulo**	**precio_articulo**
86	RED	50000
40	RAQUETA	65000
32	PAQ-3	47500
94	PAQ-6	50000
40	RAQUETA	65000
41	FUNDA	100000

Estas tablas deben volver a pasar por el proceso de normalización de la primera y segunda forma normal. Se notará que cuando la tabla tiene como clave primaria a un solo campo, estará automáticamente en segunda forma

normal. Las tablas candidatas a ser normalizadas para la segunda forma normal serán aquellas que tienen claves compuestas, es decir, la clave primaria formada por más de un campo.

Tercera forma normal

La tercera forma normal establece que las tablas de una base de datos no deben contener dependencias transitivas, es decir, campos que dependan de campos que no son clave primaria.

Se usará como ejemplo para describir este proceso la siguiente tabla:

numero_pedido(PK)	**fecha_pedido**	**cliente**	**nombre_cliente**	**ciudad_cliente**
1971	23/02/2018	101	MARTINEZ	Asunción
1972	25/02/2018	107	QUIÑONEZ	Encarnación
1973	27/02/2018	110	AB-SPORTS	Asunción

Los campos *nombre_cliente* y *ciudad_cliente* dependen del campo *cliente*, no así del campo *numero_pedido* que es la clave primaria de la tabla. Aquí tenemos el caso de una dependencia transitiva (campos que dependen de campos que no son clave primaria) por lo que debemos aplicar el proceso de normalización a la tercera forma normal, obteniendo la estructura:

numero_pedido(PK)	**fecha_pedido**	**cliente**
1971	2/23/03	101
1972	2/25/03	107
1973	2/27/03	110

Cliente (PK)	**nombre_cliente**	**ciudad_cliente**
101	MARTINEZ	Asunción
107	QUIÑONEZ	Encarnación
110	AB-SPORTS	Asunción

A las tablas resultantes aplicamos nuevamente la primera y segunda forma normal, estableciendo la clave primaria para la tabla clientes.

La normalización hasta la tercera forma normal es un proceso que debe realizarse siempre luego de realizar un diseño preliminar de las tablas de

una base de datos. Recordemos que con la normalización se evitarán muchos problemas de anomalías y hará que nuestra base de datos sea más fácil de mantener al momento de ampliarla.

Prácticas de tratamientos de datos para mejorar el rendimiento y la seguridad de una base de datos

Para mejorar el rendimiento y la seguridad en una base de datos, una buena práctica es dejar la manipulación de los datos al motor de base de datos y no realizar las inserciones, modificaciones, eliminaciones y consultas mediante comandos enviados desde el lenguaje de programación. Sobre este aspecto la recomendación más importante es el uso de los procedimientos almacenados. Un procedimiento almacenado es un conjunto de comandos *SQL* que pueden guardarse en el servidor. Una vez que se hace, los clientes no necesitan lanzar cada comando individual, sino que pueden en su lugar llamar al procedimiento almacenado como un único comando (Hueso, 2014). En el sitio oficial de la empresa *Microsoft Corporation* se citan las siguientes ventajas en el uso de procedimientos almacenados:

- Tráfico de red reducido entre el cliente y el servidor.

Los comandos de un procedimiento se ejecutan en un único lote de código. Esto puede reducir significativamente el tráfico de red entre el servidor y el cliente porque únicamente se envía a través de la red la llamada que va a ejecutar el procedimiento. Sin la encapsulación de código que proporciona un procedimiento, cada una de las líneas de código tendría que enviarse a través de la red.

- Mayor seguridad.

  Varios usuarios y programas cliente pueden realizar operaciones en los objetos de base de datos subyacentes a través de un procedimiento, aunque los usuarios y los programas no tengan permisos directos sobre esos objetos subyacentes. El procedimiento controla qué procesos y actividades se llevan a cabo y protege los objetos de base de datos subyacentes. Esto elimina la necesidad de conceder permisos en cada nivel de objetos y simplifica los niveles de seguridad (Microsoft, 2017).

El código escrito en un procedimiento almacenado es más seguro ya que al encontrarse en el servidor, el usuario o la aplicación no tienen acceso directo a sus comandos o su estructura, aportando una encapsulación adicional que mejora la seguridad.

Al aportar mayor seguridad y rapidez, el uso de procedimientos almacenados es recomendable en bases de datos de las cuales se espera

velocidad y rendimiento. Es posible realizar unas pruebas de rendimiento usando los comandos directamente en el lenguaje de programación y haciéndolo con procedimientos almacenados en el servidor, de seguro notaremos un importante incremento en la velocidad con las que las peticiones de datos son realizadas en cada caso.

El procedimiento almacenado realiza un procesamiento intermedio en el servidor de bases de datos, sin transmitir los datos innecesarios a través de la red, dando como resultado un uso reducido de la red y un mejor rendimiento general.

Otra ventaja del uso de los procedimientos almacenados es que cuando necesitemos realizar alguna modificación, se hará en el podrecimiento almacenado, no en el lenguaje de programación, por lo que evitamos tener que volver a escribir o actualizar código en las interfaces de programación.

Todos los motores de bases de datos permiten conceder y revocar permisos a los procedimientos almacenados, de esta forma podemos construir una sola interface para todos los usuarios y controlar qué operaciones pueden ser ejecutadas por los usuarios simplemente concediendo o revocando el permiso de ejecución a un determinado procedimiento almacenado.

Las ventajas de rendimiento en el uso de procedimientos almacenados son más evidentes cuando la base de datos contiene miles o millones de registros. Por ello se recomienda contar con un conjunto de registros de pruebas para notar la velocidad de respuesta cuando se utilizan procedimientos almacenados. En este artículo no realizaremos estas pruebas pero se invita a los diseñadores a realizarlas para comprobar de forma personal las ventajas en el rendimiento.

Conclusión

Las técnicas expuestas en este trabajo son recomendaciones de buenas prácticas que ayudarán al diseñador a mejorar sus proyectos y ofrecer mejores rendimientos a las aplicaciones, cuando éstas deban soportar grandes volúmenes de registros y muchos usuarios haciendo uso de las aplicaciones al mismo tiempo.

Se sugiere la utilización de la nomenclatura *under_score* para mejorar estandarizar y mejorar la nomenclatura de objetos en una base de datos. Las técnicas de normalización expuestas serán utilizadas a fin de evitar comportamientos anómalos en las operaciones y consultas. El uso de los procedimientos almacenados al momento de manipular y consultar datos mejorará el rendimiento y la seguridad de la base datos. Todas estas buenas

prácticas se realizarán independiente de las herramientas utilizadas en el diseño de bases datos.

Las recomendaciones de estas buenas prácticas no son difíciles de seguir, el diseño y la calidad de los proyectos serán mejores y proporcionarán a los usuarios una experiencia más satisfactoria.

Referencias

Codd E. F. (1970). *A Relational Model of Data for Large Shared Data Banks Communications of the ACM*, Vol. 13, No. 6, 377-387.

Hueso I. (2014) *Bases de datos. Grado superior*. Madrid, España: RA-MA Editorial.

Hueso I. (2014) *Gestión de bases de datos*. Madrid, España: RA-MA Editorial.

Microsoft, C. (2017). *Procedimientos almacenados*. Recuperado de https://docs.microsoft.com/es-es/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-2017

Marqués M. (2009) *Bases de datos*. España: Universitat Jaume I. Servei de Comunicació i Publicacions.

Martin, R. (2009). *Código limpio*. Madrid, España: Prentice Hall.

Oppel A. (2010). Fundamentos de base de datos. Mexico, D.F.: Mc Graw Hill.

Sharif B., Maletic, J. (2010). An Eye Tracking Study on camelCase and under_score Identifier Styles. En *2010 IEEE 18th International Conference on Program Comprehension* (pp. 196-205).