

Facultad de Ciencias Exactas y Naturales Escuela de Informática y computación

EIF 211 - Diseño e Implementación de Bases de Datos

Proyecto #3

**Estudiantes:**

**Kevin Chaves Granados**

**William Mora Barahona**

**Jean Abarca Bermúdez**

**Andrés Carballo Esquivel**

**Profesor: Msc Johnny Villalobos Murillo**

**I Ciclo 2016**

**Lagunilla, Heredia 12 de Junio, 2016**

Introducción

Este manual está basado en el caso de la municipalidad de Coronado, San José, Costa Rica y toma curso sobre el camino del software libre más precisamente PostgreSQL.

Objetivos

-Proporcionar una guía para el lector y a la organización patrocinadora del manual.

-Promover el conocimiento básico para la administración, mantenimiento y optimización de una base de datos.

-Ser capaz de realizar el modelo de una base de datos relacional, a partir de la especificación de requerimientos de la organización, comprendiendo y aplicando los conceptos y transformaciones implicados.

Descripción de la organización

Esta organización corre con el sistema de archivos DOS uno de los más usados en el mundo, hay una muy buena razón para que aún se mantenga funcionando, este está “Bien” logrado pero aun así sabemos que estos sistemas presentan grandes desventajas en comparación a los sistemas gestores de bases de datos (SGBD).

Es importante destacar que los sistemas que actualmente se usan en esta organización están en su estado “terminal” y es necesario dar paso a una implementación de un sistema gestor de bases de datos.

En nuestra reunión con los principales entes administrativos del departamento de TI en la municipalidad se nos hizo saber que esta quería comenzar a utilizar un sistema de software libre, por lo tanto recomendamos el uso de PostgreSQL.

Características importantes de PostgreSQL.

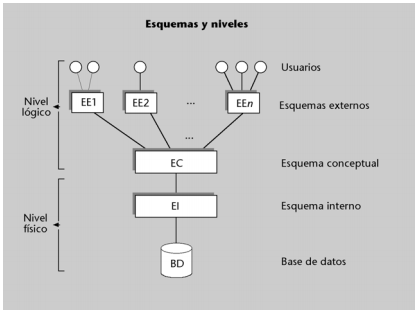
Como se mencionó anteriormente este manual está enfocado a la utilización de postgreSQL, por esta razón debemos conocer el porque es un SGBD ampliamente usado y preferido en el área del software libre. Unas de sus principales características son:

**PostgreSQL**

* Se rige bajo una licencia BSD la cual no nos limita en ningún aspecto. Se puede comercializar libremente aunque debemos poner una nota sobre la misma en nuestro software.
* No ofrece soporte directamente pero sí ofrece los medios necesarios para que la comunidad sea la que se encargue de dicha tarea. También existe una lista (corta) de empresas que brindan soporte profesional.
* PHP se acopla muy bien con este SGBD ya que ofrece funciones especializadas para aprovechar al máximo sus características. Ampliamente usado en aplicaciones web.
* Su administración se basa en usuarios y privilegios.
* Es altamente confiable en cuanto a estabilidad se refiere.
* Para la creación de estructuras (tablas, índices, etc) PostgreSQL ofrece prácticamente todas las prestaciones contempladas en el estándar, y tiene un fantástico sistema de extensión.
* Los tipos de datos y operadores que soporta son más que suficientes para su uso en la municipalidad.
* Para la administración soporta bastante bien el estándar en cuanto a la creación de usuarios y la gestión de sus privilegios con GRANT y REVOKE. Su sistema múltiple de autenticación lo hace demasiado completo en este aspecto.
* Para la gestión de copias de seguridad disponemos tanto de herramientas de volcado, como de la posibilidad de copia binaria de la base de datos.
* Posee una documentación muy extensa y muy bien desarrollada.
* Es extensible mediante FDU (Funciones definidas por el usuario) fuerte, mediana y débilmente acopladas. Además de soportar extensiones para tipo de datos compuestos o nativos así como los operadores para manejar los nuevos tipos de datos.

Como se puede ver es un SGBD muy completo y totalmente ACID (Atomicity, Consistency, Isolation and Durability) lo cual significa Atomicidad, Consistencia, Aislamiento y Durabilidad.

Arquitectura Fisica y logica



**Arquitectura física:**

En este nivel se debe tener en cuenta que el trabajo del DBA será responder a las cuestiones de rendimiento (espacio y tiempo) planteadas al hacer el diseño físico de la BD y al ajustarlo posteriormente a las necesidades cambiantes de la municipalidad.

Debemos lograr un sistema robusto en el que la BD y los sistemas de aplicación que depende de ella no se vean comprometidos por un cambio en la estructura física debido a que un disco falló o los recursos se están quedando cortos debido a la gran cantidad de datos procesados.

En este aspecto hay que señalar que la Municipalidad no tiene en este momento un estimado de los picos de solicitudes a la base de datos lo cual hace que se tome en cuenta la cantidad de registros y la cantidad de personas que tienen acceso a la base de datos para crear una cifra muy aproximada y se pueda llegar mejor a las necesidades físicas que va a requerir la base de datos.

Conceptos y recomendaciones que debemos conocer para lograr la labor descrita:

**Tablespaces:** Son unidades de almacenamiento lógico que permite al DBA definir las rutas en el sistema de archivos donde los ficheros que representan los objetos(tablas, índices, etc) de la BD pueden estar guardados en el disco. (La creación de tablespaces y su sintaxis se verá en la sección de parte lógica del manual.)

Esto nos brinda 2 **ventajas:**

* Si la partición o volumen donde el clúster fue inicializado se queda sin espacio y no se puede extender, podríamos crear un tablespace en otra partición y usarlo hasta que el sistema pueda ser actualizado con los discos necesarios.
* Los tablespace permite al DBA usar el conocimiento del patrón de uso de los objetos de la base de datos para mejorar el rendimiento. Por ejemplo: Consideremos un índice que es usado muy frecuentemente en la municipalidad; digamos una lista de contribuyentes. Se podría pensar en ubicarlo en un disco de mayor velocidad (SSD) y en sentido inverso ubicar los datos que son raramente usados en discos de menor prioridad y no tan caros.

**Discos:** Son los encargados de almacenar los datos de manera persistente pero son lentos en relación a la memoria RAM. Aumentar la memoria puede ayudar para bases de datos pequeñas ya que con más memoria tendremos que acceder menos al disco pero esta solución no aplica para bases de datos grandes. Algunas consideraciones:

* De ser posible usar una configuración RAID para lograr una mayor integridad, mayor tolerancia a fallos, mayor rendimiento y por supuesto mayor capacidad de almacenamiento. **Nota:** El nivel de RAID a utilizar debe ser elegido en base a la necesidad de la municipalidad.
* Es recomendable mantener el directorio **/pg\_xlog(**donde se encuentran los archivos WAL**)** en un disco aparte para que la escrituras de estos archivos no generen un costo mayor en el rendimiento general de la base.
* Es recomendable mantener en discos separados los índices y datos.

**Memoria:** La memoria RAM tiene unos de los roles más importante en el funcionamiento optimizado de la base de datos. Es por esta razón que se considera importante conocer los siguientes parámetros del archivo postgresql.conf:

* **shared\_buffers:** Determina la memoria dedicada que usará PostgreSQL para mantener datos en caché. Es recomendable empezar probando con un ¼ de la memoria RAM disponible ya que si se empieza con un valor más alto puede ocasionar problemas en el servidor al tomar memoria que el SO utiliza para realizar sus procesos.
* **work\_mem:** Este parámetro sirve para establecer la cantidad de memoria que vamos a utilizar de los shared\_buffers para realizar operaciones relacionadas a ordenamientos, joins, hash, etc. Esto quiere decir que se usará X veces por consulta(s) realizada por lo tanto debemos tener en cuenta el siguiente parámetro.
* **max\_connections:** Con ese parámetro vamos a establecer el número máximo de conexiones a la BD, si en la municipalidad se tienen muchos usuarios trabajando simultáneamente se debe tomar en consideración la cantidad de memoria de trabajo(work\_mem) asignada para que la memoria no se sature y las consultas, funciones y procedimientos trabajen de la mejor manera.
* **wal\_buffers:** Establece la cantidad de memoria compartida que se usará para mantener los registros antes de ser escritos en disco. La escritura se realiza antes de realizar un commit. El valor por defecto de este parámetro suele tomar 3% de memoria del shared\_buffers con esto suele ser suficiente el la mayoría de los casos. Pero puede ser ajustado a la necesidad.

·

**Arquitectura Logica:**

**Criterio de distribución:**

La organización cuenta con un servidor donde los archivos son distribuidos en los discos duros del mismo, una especificación de donde se encuentran estos archivos no fue discutida con la organización pero el sistema de archivos DOS utiliza un nombramiento de los archivos de 8 bytes y los directorios son acomodados en una estructura jerárquica de directorios.

**Requerimientos de espacio de disco:** la organización no cuenta con una taza de transacciones diaria, lo cual tiende a ser un problema a resolver en la nueva implementación de la arquitectura postgreSQL para definir los nuevos requerimientos de espacio en disco, los cuales recomendamos que deben de tener un espacio aislado para usuarios, uno para el sistema gestor de bases de datos, un espacio de almacenamiento temporal, uno con los índices de las tablas, esto para maximizar eficiencia y por último un espacio dedicado a los “Rollback” estos son útiles para y cuando en el sistema gestor de bases de datos se ha presentado un problema de integridad, de funcionamiento o simplemente un problema tecnico en medio de una o muchas transacciones, por lo tanto un espacio dedicado a esta funcionalidad puede dar una solución para poder “volver en el tiempo” en caso de que se presente un problema de este tipo y este pueda ser resuelto de manera efectiva.

En cuanto a un ejemplo de como hacer el método para el análisis de consumo de disco, tenemos:

Select \* from Patentes where Patentes.propiedad\_numPK = ‘001’;

Utilizaremos el modelo entidad-relación que hemos considerado en la implementación del modelo de datos en el manual, el cual es una aproximación, y hemos considerado 13 diferentes tablas.

Donde la tabla donde buscamos hacer nuestra transaccion cuenta con una llave primaria de tipo Integer( 4 bytes) la cual es el número de patente, la fecha de tipo Date ( 8 bytes ) y el estado el cual es un booleano (1 byte).

(13 Tables) \* (13 bytes) \* 365 \* (Prom. Consultas diarias(1)) = 60.164 Kb

Por lo tanto solo hacer una consulta de patentes diariamente con un criterio de búsqueda básico nos da esta estimacion de consumo, a la cual hay que multiplicar la cantidad de consultas diarias, en el cual está contemplado el peor de los casos, esto se debe a que para hacer el cálculo se ha utilizado el modelo entidad-relación, este es solo un ejemplo básico pero nos adentra a la cantidad de poder computacional requerido para mantener una organización como esta y su funcionamiento óptimo.

**Buenas prácticas y recomendaciones para la implementación de una base de datos:**

**La identificación del modelo de datos de una organización**: Es el paso que hay que tomar en cuenta en la creación de una base de datos, se debe investigar acerca de las reglas de negocio de la organización para crear una base de datos óptima que cubra todas las necesidades específicas de la organización, efectivamente las grandes organizaciones no tuvieron las mejores bases de datos en sus comienzos, esto tiende a ser porque en el entorno de la informática podemos ver que las prácticas tanto como el software están en una constante mejora que las impulsa a tener visiones en el mercado más amplias y con esto sus bases de datos deben de ser lo suficientemente genéricas y eficientes para poder acomodar las nuevas reglas de negocio que surjan en estas mejoras.

**Definición de cardinalidades**: Una vez que el modelo de datos ha sido identificado, hay que tener claro que las diferentes entidades de las bases de datos son relacionadas con otras, esto implica que en la base de datos las cardinalidades deben de ser altamente estudiadas para que un futuro no se presenten errores de integridad, un claro ejemplo de esta práctica es imaginarse un salón de clase, en este salón, ¿el alumno es quien tiene un salon?  
O ¿es el salon quien tiene a los estudiantes?  
En este caso entramos a temas muy filosóficos pero si utilizamos nuestra lógica como informáticos nos damos cuenta que es más eficiente que un alumno tenga asignado el número de un salón a su tabla de cursos donde se pueda saber con certeza que ese será el lugar donde el estudiante tenga clases, de la misma manera nos encontramos con el dilema del salón teniendo una cantidad N de estudiantes, donde sí se ve de una manera, el salon en teoria podria ser capaz de tener una tabla donde estan los codigos de todos los estudiantes que asisten al mismo, de ambas maneras consideramos que pueden ser prácticas válidas, dicho sea el caso, hay que tomar en cuenta las reglas de la organización, una organización que se centra en estudiantes no tiene porque darle tanta importancia a un salón de clases, pero por otro lado si este dilema fuese por seguridad y se quisiese saber exactamente qué estudiante y a que hora es cuando habita el salón podemos ver reflejada la segunda opción propuesta, todo se basa en las reglas de la organización y la manera en que nosotros como informáticos somos capaces de ver los diferentes problemas y sus respectivas soluciones.

**Creación del modelo Entidad - Relación:** Para la creación de un modelo entidad- relación se deben de identificar las diferentes entidades que van a conformar la base de datos, las entidades son la base de toda implementación de una base de datos, si como programadores podemos crear clases a partir del paradigma orientado a objetos donde cada objeto tiene atributos y funciones, una entidad pasa a ser uno de esos objetos que puede ser almacenado y utilizado en el futuro para su posterior uso en una base de datos, al mismo tiempo la identificación de sus diferentes atributos pasa a ser una parte importante porque estos pueden mantener referencias a las diferentes relaciones que se crean a partir de este modelo.  
Este modelo tiende a ser un poco lento pero tiene una implicación útil sin embargo este se asemeja más al mundo real y por eso este es mucho más fácil de comprender que los demás modelos de datos y se usa para calcular un nivel muy alto en la tasa de transacciones en la base de datos, una vez modeladas las entidades se deben establecer las relaciones que comparten las mismas, estas relaciones en este modelo pasan a ser tablas separadas a las entidades para confeccionar un modelo que es entendible a la vista; estas dos viniendo a ser las más grandes ventajas de este modelo. Una vez creado se pueden identificar las diferentes relaciones que en si no son necesarias para la siguiente implementación, el modelo Relacional.

**Creación del modelo Relacional:** El modelo relacional cumple con uno de los principales objetivos de las bases de datos, la eficiencia en sus consultas; este modelo debería ser creado a partir de un modelo entidad-relación, ya que este mantiene solo las relaciones que son puras de las reglas de negocio en la organización y las que no son necesarias se descomponen en atributos que referencian otras entidades, la aplicación de este modelo nos lleva a un punto de optimizacion mucho mas alto que el anterior modelo, aun así se recomienda empezar con el anterior para que se puedan hacer los cálculos necesarios para un mejor manejo del espacio requerido en la base de datos. Para que este modelo pueda funcionar de manera óptima se recomienda cumplir con las 4 restricciones de integridad:

* **Integridad relacional**: Esta dice “Toda relación requiere una única llave primaria” esta se refiere a que la identificación de cada una de estas entidades debería ser única e irrepetible.
* **Integridad referencial**: “No se pueden eliminar llaves primarias ligadas a llaves foráneas” esta se refiere a la eliminación de entidades que tienen un significado derivado por otra entidad.
* **Integridad de dominio**: “Si un atributo (a) solo permite un dominio, a este se le deberían establecer sólo los valores correctos”, esta integridad es importante ya que le da la oportunidad a la base de datos de ser más inteligente y poder atrapar excepciones por este lado reduciendo la cantidad de programación hecha por el informático.
* **Correcta dependencia funcional**: “Conocer la regla de negocio” este es un tema importante que viene a ser la base de todas base de datos, es importante conocer el negocio para que la base de datos sea adaptada a las necesidades de la organización.

**Normalizar el modelo Relacional**: La normalización es útil para llevar la implementación de la base de datos al siguiente nivel ya que asegura que no existan problemas de integridad en la misma.

-La Primer Forma normal: Es cuando una tabla está libre de atributos repetidos y sus dominios son autónomos aunque en la actualidad se permite el uso de atributos repetitivos.

-La Segunda Forma normal: Esta no debe tener dependencias parciales entre la llave primaria y los otros atributos, esto quiere decir que hay que dividir las tablas hasta que solo exista una llave primaria aplicando el teorema de descomposición de tablas.

-La Tercera Forma normal: Es cuando no existen dependencias transitivas entre la llave primaria y los demás atributos.

PL/pgSQL

Esta sección es muy importante para el funcionamiento óptimo de la base de datos. Conocer cómo crear un tablespace y saber que tipo de datos vamos a almacenar en él es crucial para mantener estable la BD y evitar que tengamos que detener el funcionamiento del sistema para hacer cambios en la estructura física (aumentar memoria, almacenamiento o reparaciones). Además los procedimientos y funciones almacenadas junto con los triggers son esenciales para simplificar el trabajo y ayudar a mantener la integridad de la BD. Ya que el único que debería “tocar” la DB es el DBA. Con esto en mente pasamos a la práctica.

**Creación de tablespaces.**

Sintaxis de la sentencia para crear un tablespace en PostgreSQL 9.3.

**Ejemplo 1:**

Para crear un tablespace llamado **espacioBD** en el directorio **/data/dbs:**

*CREATE TABLESPACE* ***espacioBD*** *LOCATION* ***'/data/dbs'****;*

**Ejemplo 2:**

Para crear un tablespace llamado **espacioIndices**en el directorio **/data/índices** perteneciente a usuario llamado **DBA:**

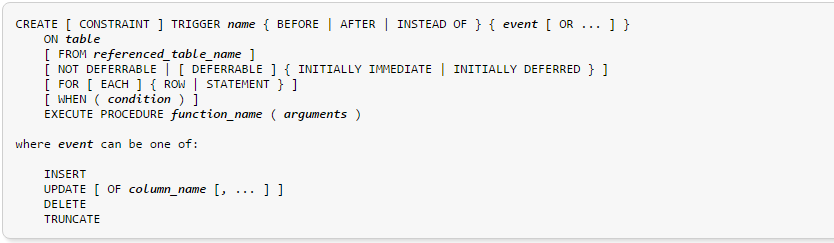
*CREATE TABLESPACE* ***espacioIndices*** *OWNER* ***DBA*** *LOCATION* ***'/data/indices'****;*

**Nota:** El parámetro **tablespace\_option** de momento solo puede tomar 2 valores **seq\_page\_cost** y **random\_page\_cost.** Definir este parámetro para un tablespace en particular hará que se sobreescriba la estimación de costo de lectura del *PLANNER* para leer las páginas de las tablas en ese tablespace. Esto puede ser útil si uno de los tablespace está ubicado en un disco que es más lento o más rápido que el resto del sistema.

Triggers

Los trigger son procesos que se disparan cuando ocurren algunos eventos en la base de datos, tales como UPDATE, DELETE y INSERT, ya que estos son los que alteran la base de datos y es cuando se necesita que se ejecuten.

PL/pgSQL puede ser usado para definir procedimientos de triggers.Los triggers son creados mediante la sentencia CREATE TRIGGER.

-Name: el nombre que se le va a ser asignado al trigger.  
-Before | After | Instead of: Este parámetro determina en qué momento va a ser ejecutado el trigger, ya sea antes, después, o en lugar del evento.

-From referenced table: El nombre de la tabla donde el trigger va a ser ejecutado.

-For each row | statement: para especificar si se debe ejecutar en cada una de las filas donde se haga un cambio o solamente por sentencia

-Condition: Determina si el trigger va a ser ejecutado si determinada condición se da o no.

-Execute procedure: Ejecuta la función que el usuario determinó para su ejecución en el momento especificado.

Ejemplo:

*CREATE TRIGGER verifica\_patente  
 BEFORE UPDATE ON patentes  
 FOR EACH ROW  
 EXECUTE PROCEDURE verifica\_fecha\_vencimiento();*

La utilización de las variables NEW y OLD son propias de los triggers, las cuales pueden ser implementadas para la utilización de los datos anteriores y posteriores al cambio del atributo para extender el uso de algoritmos en las funciones de la base de datos.

Funciones Almacenadas

Una función almacenada se puede definir como un programa almacenado en la base de datos y listo para ser usado.

Existen ventajas evidentes al utilizar funciones almacenadas como:

* La ejecución de la función ocurre en el servidor de bases de datos, esto aumentará el rendimiento.
* Al tener la lógica de la aplicación implementada en la base de datos no hay que implementarla en los clientes.
* Si hay diferentes tipos de usuarios implementados en diferentes sistemas o lenguajes de programación y accediendo a la misma base de datos. Hay una API a la lógica de la aplicación lista para usarse desde diferentes clientes.

Podemos definir una función en PL/pgSQL de la siguiente manera:

*CREATE [ OR REPLACE ] FUNCTION   
nombre\_funcion([ [ argmodo ] [ argnombre ] argtipo [, ...] ])   
RETURNS tipo AS $$  
  
[ DECLARE ]  
 [ declaraciones de variables ]  
  
BEGIN  
 codigo   
  
END;  
$$ LANGUAGE plpgsql,*

**Declaraciones de variables**: Las declaraciones de variables se pueden realizar de la siguiente manera:

*nombre\_variable ALIAS FOR $n;  
nombre\_variable [ CONSTANT ] tipo [ NOT NULL ] [ { DEFAULT | := } expresion ];*

**IMMUTABLE / STABLE / VOLATILE:**

IMMUTABLE: Indica que la función no puede alterar a la base de datos y que siempre devolverá el mismo resultado.

STABLE: Indica que la función no puede alterar a la base de datos y que siempre devolverá el mismo resultado en una consulta de una tabla en especifico.

VOLATILE: Indica que la función puede devolver diferentes valores.

**SECURITY INVOKER / SECURITY DEFINER:**

SECURITY INVOKER: Indica que la función se ejecutará con los privilegios del usuario que la ejecuta.

SECURITY DEFINER: Indica que la función se ejecutará con los privilegios del usuario que la creó.

A continuación un ejemplo sencillo de una función:

*CREATE OR REPLACE FUNCTION ejemplo() RETURNS integer AS $$  
BEGIN  
 RETURN 1;  
END;  
$$ LANGUAGE plpgsql;*

Ejemplo con dos argumentos y definiendo algunas variables:

*CREATE OR REPLACE FUNCTION ejemplo(integer, integer) RETURNS integer AS $$  
DECLARE  
 numero1 ALIAS FOR $1;  
 numero2 ALIAS FOR $2;  
  
 constante CONSTANT integer := 100;  
 resultado integer;  
  
BEGIN  
 resultado := (numero1 \* numero2) + constante;  
  
 RETURN resultado;  
END;  
$$ LANGUAGE plpgsql;*

Usando una sentencia IF, THEN en la función:

*CREATE OR REPLACE FUNCTION ejemplo\_txt(integer, integer) RETURNS text AS $$  
DECLARE  
 numero1 ALIAS FOR $1;  
 numero2 ALIAS FOR $2;  
  
 constante CONSTANT integer := 100;  
 resultado INTEGER;  
  
 resultado\_txt TEXT DEFAULT 'El resultado es 104';   
  
BEGIN  
 resultado := (numero1 \* numero2) + constante;  
  
 IF resultado <> 104 THEN  
 resultado\_txt := 'El resultado NO es 104';  
 END IF;  
  
 RETURN resultado\_txt;  
END;  
$$ LANGUAGE plpgsql;*

Una función no solo se tiene en cuenta el nombre del mismo para diferenciarlo de otros, los argumentos de la función también se tienen en cuenta. ejemplo(), ejemplo(integer), ejemplo(integer, integer) y ejemplo(text) son todos procedimientos diferentes, aunque se llamen igual.

**Seguridad**

Un usuario es todo aquel que tenga contacto con el sistema de bases de datos.

Se tienen 3 clases generales de usuarios:

**1. Programador de aplicaciones:** son aquellos profesionales en informática que interactúan con el sistema a través del DML(Lenguaje de Manipulación de Datos), los cuales se encuentran en un lenguaje de programación. Es el encargado de escribir programas de aplicación que usen Bases de Datos.

**2.** **Usuario Final:** accede a la base de datos desde un equipo en el cual puede utilizar lenguaje de consulta generado como parte del sistema o acude a un programa de aplicación suministrado por un programador.

**3.** **Administrador de Bases de Datos**: es el encargado del control general del sistema.

Todo usuario que ingrese o consulte una base de datos puede clasificarse como:

1. **Programador de Aplicaciones.**

* **Usuario sofisticado**: interactúa con el sistema sin escribir programas. Generan consultas en un lenguaje de bases de datos.
* **Usuario Especializado:** algunos usuarios sofisticados desarrollan aplicaciones de bases de datos especializadas.
* **Usuarios ingenuos**: es el usuario final que utiliza bases de datos sin saberlo.

**2. DataBase Administrators(DBA):** administran 2 recursos:

* + La base de datos
  + El SGBD(Sistema de Gestión de Bases de Datos) y con el software relacionado.

El Administrador de Base de Datos (DBA) es quien autoriza el acceso a la base de datos, vigilar el uso y adquirir hardware y software necesarios para su uso. También es el responsable de velar por la seguridad y el funcionamiento óptimo del sistema.

**3. Usuarios Finales**: son quienes requieren acceso a la base de datos para generar consultas e informes. Hay varios usuarios finales como son:

* **Usuarios finales esporádicos:** acuden de vez en cuando pero esto no significa que siempre requieren la misma información.
* **Usuarios finales simples o paramétricos:** su función gira en torno a consultas y actualizaciones de la base de datos.
* **Usuarios finales avanzados**: estos son ingenieros, analistas de negocios, científicos, son quienes conocen los recursos del SGBD para satisfacer requerimientos complejos.
* **Usuarios Autónomos**: utilizan bases de datos personalizadas basadas en programas comerciales que cuentan con interfaces de fácil uso.

Conociendo esto es necesaria la creación de roles en la base de datos para que cada uno de estos usuarios pueda acceder a la base manteniendo la integridad de la base para aclarar la creación de roles, se adjunta un ejemplo:

*CREATE ROLE* ***name*** *[ [ WITH ]* ***option*** *[ ... ] ]*

Donde en la parte que dice option puede ser cambiado por los siguientes roles:

* **SUPERUSER:** El puede anular todas las restricciones de acceso dentro de la base de datos, es sumamente peligrosa ya que puede alterar el funcionamiento del sistema entonces solo debe ser utilizado cuando sea realmente necesario.
* **CREATEDB:** Este usuario tiene la potestad de crear nuevas bases de datos.
* **CREATEROLE:** Este usuario tiene la potestad de crear nuevos roles.
* **CREATEUSER:** Este usuario tiene la potestad de crear nuevos usuarios.
* **INHERIT:** Este usuario puede usar los roles que se le ha dado además de otros que se le heredaron.
* **LOGIN:** Este usuario puede ingresar a la bases de datos.
* **CONNECTION LIMIT connlimit:** Este usuario tiene cierta cantidad de conexiones a la base.
* **[ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password':** Es un atributo que tiene el LOGIN, pero se puede añadir a la hora de hacer login.
* **VALID UNTIL 'timestamp' :** Usuario que tiene un tiempo de conexion, despues no puede ingresar.
* **IN ROLE rolename [, ...]:** Se le entrega el privilegio a los usuarios.
* **IN GROUP rolename [, ...]:** Se les da un grupo de privilegios a los usuarios.
* **ROLE rolename [, ...]:** Se añaden automáticamente a los usuarios.
* **ADMIN rolename [, ...]:** Este usuario puede conceder privilegios.

Modelo de datos

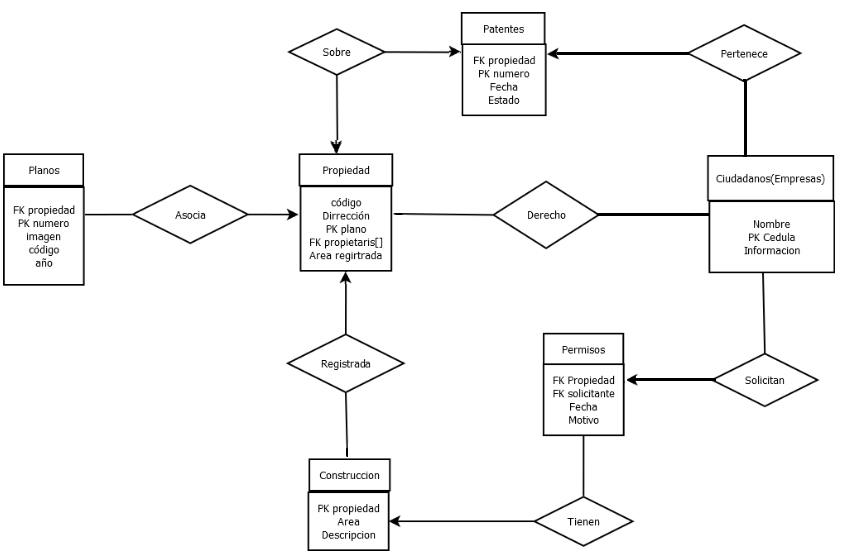
**Modelo Entidad Relación**

En este modelo se debe tener en cuenta que todas la tablas se relacionan, en este caso la relación aparece como una tabla más, la cual va a tener datos de esa relación, lo cual ayuda a entender cómo va a estar estructurada la base de datos y cómo va a estar ligada a las reglas de negocio.

En este caso la Municipalidad de Coronado nos proporcionó la información que ellos manejan lo que nos ayudó a crear el modelo Entidad Relación.

Hay que señalar que en una relación la llave primaria es donde esta el N. Esto quiere decir que por ejemplo, un alumno tiene n cursos, entonces en la relación que tiene la llave primaria sería “cursos”.

A continuación una representación que se aproxima a las especificaciones de la municipalidad:



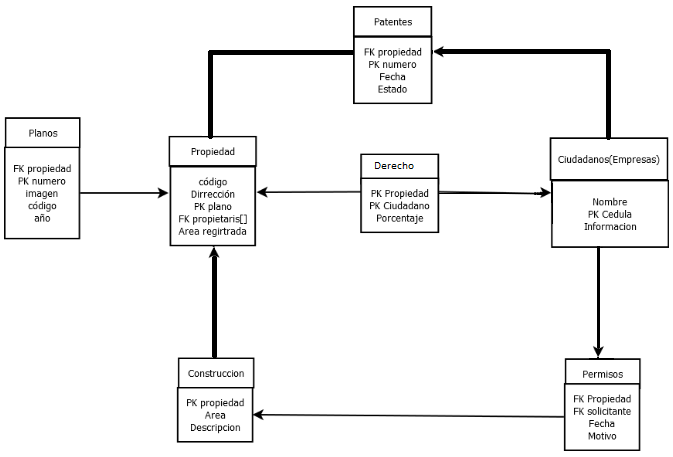
Hay que señalar que esto no es como deberia quedar la base de datos ya que se puede mejorar mucho más con el Modelo Relacional.

**Modelo Relacional**

Hay que señalar que para realizar este modelo se tiene que tener el Modelo Entidad Relación, donde, a partir de ese modelo se trata de eliminar las tablas de las relaciones, siempre y cuando se deje las relaciones de 1 a N, ya que asi queda mejor estructurada una base de datos, y se puede mantener mejor el uso de las llaves primarias.

Si al eliminar la relación no queda 1 a N, se mantiene la relación, la cual va a estabilizar la base.

En este caso basados en Modelo Entidad Relación, el Modelo Relacional quedó así:



**Algebra relacional**

Este apartado es de un orden muy tecnico pero esperamos dar un vistazo general para que cualquiera con un conocimiento mínimo logre entender su importancia. En resumen, optimizar una sentencia compleja no es tarea fácil, y saber la responsabilidad que requiere realizar una consulta SQL, por ejemplo, realizar una sentencia del tipo SELECT \* FROM TablaEjemplo, puede funcionar para unas tablas con un volumen de datos bajo pero, ¿que pasa si en estas tablas de las que hablamos se manejan millones de millones de registros?, el sistema puede verse comprometido y en el peor de los casos puede haber una caída.

La creación de un modelo relacional nos da un vistazo a la optimización de la base de datos aun así pueden haber maneras de incrementar la velocidad de las sentencias SQL con un entendimiento de el algebra relacional, esta es un lenguaje procedimental de alto nivel, para usuarios sofisticados de la base de datos.  
Este lenguaje consta de símbolos los cuales son llamados “Operadores” y de relaciones llamados “registros”.  
Como todo lenguaje se utilizan funciones de dos tipos, las funciones “Unitarias” y las funciones “Binarias” las cuales son utilizadas para interpretar las diferentes sentencias SQL.

Funciones Unitarias:

-Proyección (Π): Esta función es el equivalente a los “Select” donde un ejemplo para seleccionar el nombre y el apellido de la tabla de Personas viene a ser “Πnombre,apellido(Personas)” el cual es equivalente a “Select nombre, apellido from Personas”.

-Selección (σ): Esta función es equivalente a las restricciones de las consultas las cuales empiezan luego de un “where” por lo tanto si queremos hacer una selección de los nombres en la tabla de Personas donde los nombres sean iguales a “Juancho” se denota “Πnombre(σ nombre = “Juancho” (Personas))” el cual es el equivalente a una consulta “Select nombre from Personas where nombre = “Juancho”.

Funciones Binarias:

-Intersección(∩): Es el resultado de todas las entidades cuyos atributos pueden formarse de forma que ambos elementos puedan ser relacionados entre sí por el sistema gestor de bases de datos.  
-Producto cartesiano (x): Entrega la combinación de todas las tuplas que sean parte de las dos entidades que el usuario decida combinar.  
-División(/): Retorna todos los distintos valores de la relación entre ambas entidades.   
-Diferencia(-): Entrega todas las tuplas que estén en la primera entidad pero no en la segunda.  
-Producto Natural(⋈):Consiste en combinar las proyección, selección y producto cartesiano en una sola operación, eliminando columnas extras.

Una vez entendidas estas connotaciones recomendamos el uso del software RAT(Relational Algebra Translator) hecho por la Universidad Nacional de Costa Rica el cual es un software traductor de álgebra relacional a sentencias SQL para maximizar la velocidad de las sentencias a crear por los usuarios.

Ya que se tienen estos conceptos el DBA puede utilizar esta herramienta para crear sentencias SQL eficientes y efectivas, donde se asegura que no se desperdicie tanta memoria ni CPU y esos recursos se le puedan dar a otros usuarios del sistema. Esto quiere decir que para que una base sea óptima se tiene que tener en cuenta que siempre se puede mejorar, y dependiendo del equipo con el que se cuente hay que maximizar su funcionamiento.

Bibliografía:

**Silberschatz, A., Korth, H. F., & Sudarshan, S**. (1997). *Database system concepts* (Vol. 4). New York: McGraw-Hill.

**PostgreSQL**. (n.d.). Extraido Junio 12, 2016, de <https://www.postgresql.org/>

### Angeles, M. (2016). *Buenas Prácticas en bases de datos*.

**Camps Paré, R., Casillas Santillán, L., Costal Costa, D., Ginestà, M., Escofet, C. and Pérez Mora, O.** (2005). *Bases de datos - Software libre*.