

Normalización de bases de datos relacionales

Indice del contenido

- [Pasos para crear una base de datos relacional](#).
- [¿Qué es un RDBMS?](#).
- [Características y beneficios de los RDBMS](#).
- [Diagrama del proceso de creación de una base de datos y ventajas de la normalización](#).
- [La normalización de bases de datos explicada de forma sencilla](#).
- [Reglas normales](#).
- [Primera forma normal](#).
- [Segunda forma normal](#).
- [Tercera forma normal](#).
- [Las 12 reglas de Codd para el modelo relacional](#).
- [Requerimientos, características de transacciones](#). (ACID).
- [Atomicidad](#).
- [Consistencia](#).
- [Aislamiento \(Isolation\)](#).
- [Durabilidad](#).

Habiendo comprendido el funcionamiento del **modelo relacional** es momento de conocer el concepto de **normalización de bases de datos**.

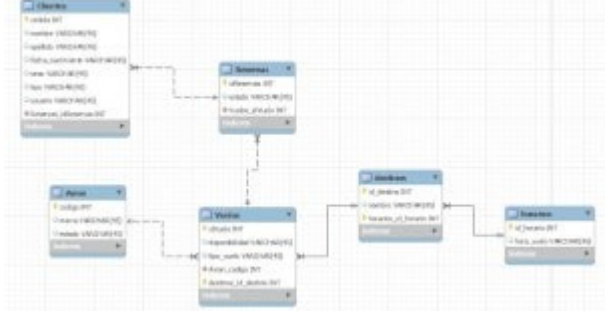
Para introducirte a la normalización de bases de datos debes conocer primero el proceso para diseñar una base de datos. Como ya sabrás las bases de datos son muy útiles para gestionar información de forma eficiente, ya sea en una empresa, pyme, o simplemente los datos de usuarios de un sitio web, **el desarrollador debe diseñar la estructura de la base de datos siguiendo un conjunto de reglas a las relaciones de los datos**.

Primeramente antes de crear una base de datos relacional en algún gestor se debe diseñar un boceto, un diagrama de cómo se relacionan los datos y donde se especificarán las relaciones, claves foráneas, claves primarias, etc.

De esta forma tendremos una vista previa de cómo estará organizada nuestra DB y cómo se estructuran los datos que vamos a almacenar dentro de ella.

Una vez creado este diagrama (**Entidad – Relación**) se procede a crear la base de datos utilizando un **Gestor de Bases de Datos Relacionales(RDBMS o RDBG)**.

Pasos para crear una base de datos relacional:

1. **Crear un diagrama Entidad-Relación:** el diagrama de entidad-relación es un clásico diagrama o modelo similar al [diagrama de flujos](#) que hemos visto anteriormente en programación en Python, pero con la diferencia de que **ilustra cómo las entidades, personas, objetos, datos se relacionan entre sí dentro del sistema.**
2. **Exportar nuestro diagrama a un RDBMS:** Normalmente estos diagramas son realizados utilizando algún software diseñado específicamente para estos propósitos como por ejemplo: Lucidchart, Workbench, etc. Mediante estos podemos exportar nuestro diseño a algún (**Relational Data Base Management System (RDBMS)**) como podría ser MYSQL, SQLITE, etc. Y aquí es donde abarca el concepto de Normalización, ya que son esas reglas las que debemos respetar y definir claramente, ***desde el momento de diseñar nuestra base de datos creando un diagrama hasta que lo exportamos a nuestro RDBMS.***
3. **Comenzar a añadir datos a nuestra base de datos** respetando las reglas y relaciones establecidas anteriormente protegiendo la **integridad de los datos.**

¿Que aprenderemos?

Primeramente aprenderemos las **reglas de normalización** para luego, en la siguiente entrada o post aprender a crear nuestro **diagrama de Entidad Relación** y exportar nuestra base de datos. Es importante que te concentres en aprender estas reglas y los beneficios que otorga respetarlas!.

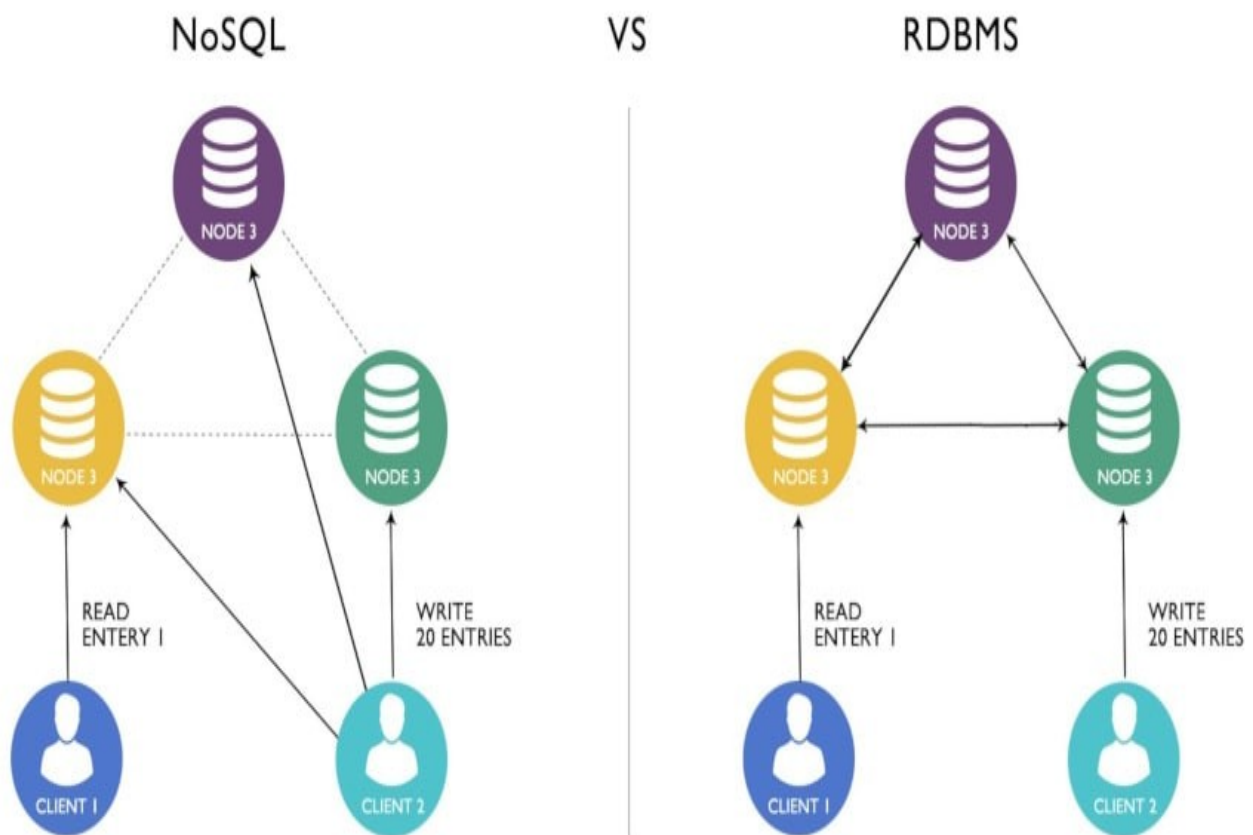
- Pero antes es importante comprender que es un RDBMS y que beneficios nos otorgan!!

¿Qué es un RDBMS?:

El modelo relacional es APARTE del gestor del sistema de bases de datos (Relational Data Base Managment System, RDBMS de ahora en más).

Podemos usar **MYSQL, PostgreSQL, SQLite, MariaDB**, etc (que son **RDBMS** de los cuales seguro habrás oído o manipulado).

Todos los **RDBMS** tienen sus pequeñas variaciones, sin embargo **nuestro modelo relacional** basado en un diseño mediante el **diagrama entidad-relación** que veremos más adelante debe poder aplicarse a cualquier **DBMS relacional (Mysql, Sql, PostgreSQL, etc..)**.



Simplemente un DBMS o DBGS es un **software de gestión de bases de datos** cuya función es servir de **interfaz** entre la base de datos, el usuario y las distintas aplicaciones utilizadas. Además de simplificarnos el manejo de los datos en conjunto y brindarnos numerosas herramientas que optimizan la creación, actualización, mantenimiento, consultas, etc de la información. En el caso de los que se aplican al modelo relacional se los denomina **RDBMS** (Relational Data Base Managment System).

Características y beneficios de los RDBMS

Un Software de gestión de bases de datos nos asegura:

- **Independencia.** (De la información respecto de nuestro programa de aplicación).
- **Seguridad de los datos.** (Además de una gestión segura se nos permite realizar backups automáticos entre otras.).
- **Mínima redundancia.** (Evitar datos repetidos en lugares diferentes).
- **Consistencia de la información.** (La **información** está completa, los datos se mantienen idénticos durante cualquier operación, como transferencia, almacenamiento y recuperación).
- **Abstracción de la información sobre su almacenamiento físico.** (Poder aislar la información del contexto físico de almacenamiento, trasladarla a otro sistema, a otro servidor, etc.).
- **Adopción de medidas necesarias para mantener la integridad de la información.** (Nos proporciona y nos obliga a respetar ciertas normas que permiten mantener la información íntegra, sin inconsistencias).
- **Medidas de seguridad y gestión de permisos de usuario.** (Admite y permite a más de un usuario leer, modificar, actualizar, mantener la información controlando sus privilegios y asegurando los beneficios anteriormente mencionado para cada uno de ellos).
- **Nos brinda un lenguaje DCL (Lenguaje de control de datos).** (Así cada **RDBMS** tiene su lenguaje y variación del mismo, normalmente **SQL** que nos permite realizar acciones con los datos almacenados de forma segura, conjunta, ágil y normalmente no está ligado a ningún proveedor específico. Por lo que aprender este lenguaje permite manejar muchos y diversos **RDBMS**)

Sin embargo, **el modelo relacional está sujeto a ciertas restricciones, reglas y recomendaciones** que debemos respetar a la hora de **diseñarlo, manipularlo y mantenerlo**. A estas reglas son las que llamamos normalmente **Normalización de la base de datos**.

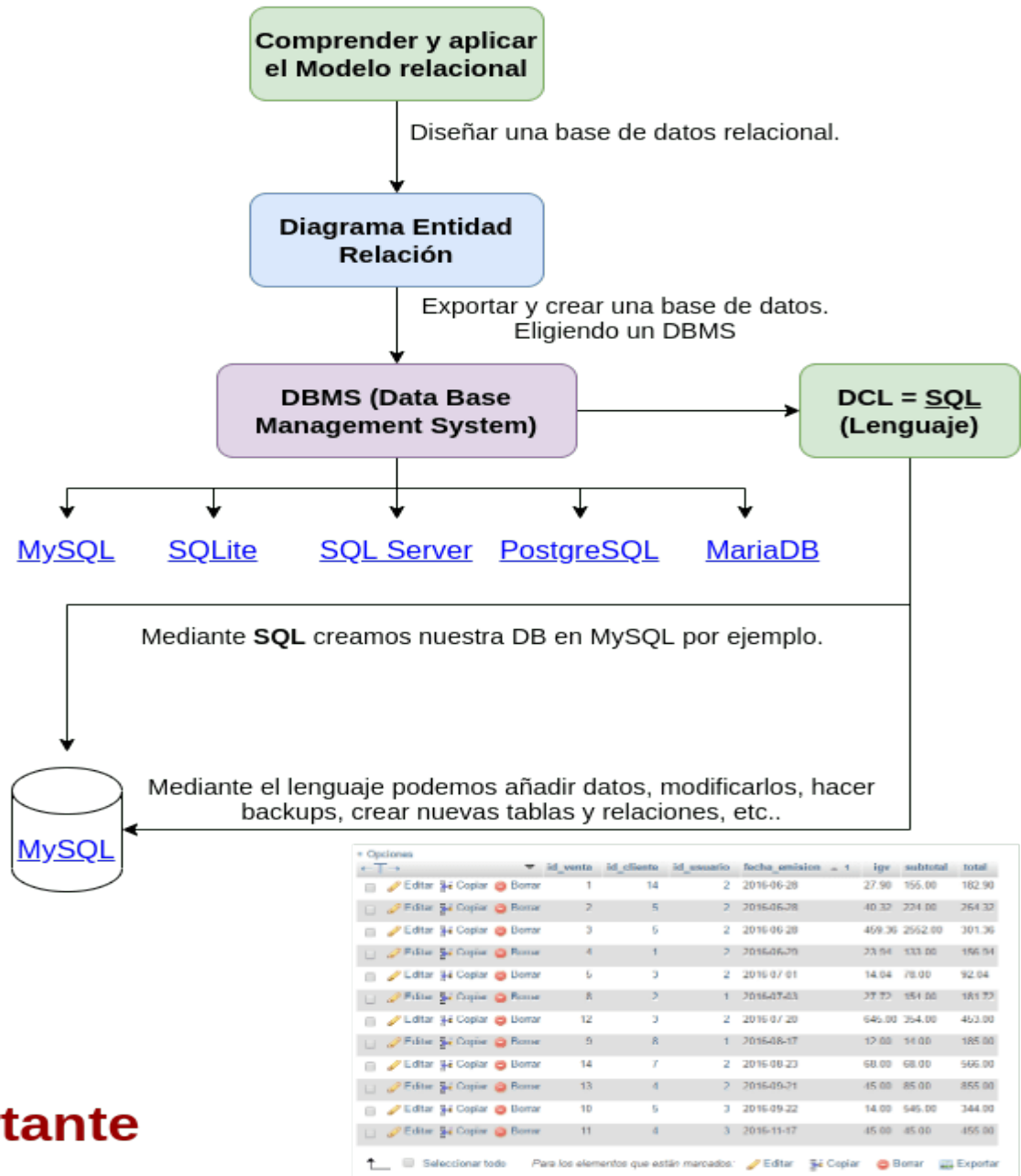
Por un lado nosotros podemos diseñar y crear nuestra propia base de datos para trabajar con ella o, en otros casos, puede que nos toque interactuar con una base de datos diseñada por otra persona, pero basada en el modelo relacional. Tanto en nuestros diseños como en la manipulación de cualquier base de datos correspondiente a este modelo debemos seguir ciertas reglas, o **podríamos cargarnos todo el modelo y con el tiempo este podría quedar insostenible, incongruente, redundante, atomizado y nos arrastraría a tener que diseñar una nueva base de datos. Y es que una DB bien diseñada y en la que se respete su estructura y las reglas citadas, no debería tener problemas nunca, salvo su tamaño en algún determinado momento de su vida o duración. Y en esto además de reglas específicas también nos ayuda el RDBMS y el diagrama Entidad – Relación al no permitirnos (en criollo “meter la pata hasta el fondo”) o lo que es lo mismo, desobedecerlas.**

Diagrama del proceso de creación de una base de datos y ventajas de la normalización.

Como siempre, te apporto mis humildes diagramas para que comprendas mejor todo lo que estoy diciendo y logres ordenar estos conocimientos en tu mente. Y fíjate que también te permite **tener un orden mental de lo que estás aprendiendo..** En este caso, veremos esos requerimientos que son explícitos, que están escritos por personas que quieren que no comentamos errores graves con el manejo de la información. Luego, cada paso del procedimiento de crear una base de datos consiste en respetar y tener en cuenta estos **requerimientos explícitos** y también los **implícitos de cada software** utilizado, entre otros.

Crear y manipular bases de datos relacionales

NORMALIZACIÓN DE BASE DE DATOS



Importante

Beneficios de la normalización de la base de datos:

- Minimizar la redundancia de los datos.
- Disminuir problemas de actualización de los datos en las tablas.
- Proteger la integridad de datos.

Mediante:

- Aplicar Formas normales
- 12 reglas de Codd.
- Respetar ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad)

La normalización de bases de datos explicada de forma sencilla:

En el momento de diseñar nuestra base de datos relacional debemos ir pensando en tablas y relaciones entre los datos y para ello optamos por diseñar y seguir ciertas reglas y formas normales como ya te había explicado para evitar redundancias e inconsistencias en los datos.

Como también te dije existen varias reglas definidas como "formas normales" que aunque resulta difícil siempre debemos de tratar de lograr seguir al menos las primeras tres y en el caso de no hacerlo, deberíamos prever posibles soluciones para los posibles datos redundantes, duplicados, o problemas en cuanto a relaciones, etc.

Reglas normales:

Las formas normales pueden verse como niveles, si se cumplen las tres primeras en una base de datos podemos afirmar que esta está en la tercera forma normal.

Primera forma normal:

Se dice que una base de datos relacional está en la "primera forma normal" si cumple que:

- Se eliminan grupos de repetición en tablas individuales.
- Se crea una tabla independiente para cada conjunto de datos relacionados.
- Identificamos cada conjunto de datos relacionados con una clave principal.

Esto en criollo (como suelo explicarlo yo) se refiere a que no habrá repeticiones de grupos de datos dentro de una misma tabla y se creará una tabla independiente para cada grupo o conjunto de datos que estén relacionados, por ejemplo sería lo normal tener una tabla "empleado", con los "nombres", "edades", "salarios", etc. Y no sería correcto tener dos tablas empleado1 y empleado2 con los nombres en uno y los salarios en otro, es absolutamente innecesario y engorroso. Y como ya sabíamos refiere también a identificar cada tupla (fila o conjunto de datos) con una llave o clave principal, por ejemplo un ID único o un DNI, (en el caso de tratarse de empleados)...

Segunda forma normal:

Se dice que una base de datos está en segunda forma normal si cumple los requerimientos de la primera y además:

- Se crean tablas independientes para conjuntos de valores que se apliquen a varios registros.
- Se relacionan estas tablas mediante una clave externa.
Básicamente eliminaremos datos redundantes..

Nada difícil de interpretar y como hemos hablado anteriormente se refiere a no almacenar un mismo valor en varias tablas, por ejemplo podemos pensar en la dirección de un empleado que no sólo es utilizada por la tabla "Empleados" sino también en la tabla "Recibo", en vez de almacenar la dirección como entradas diferentes en dos tablas, la almacenaremos en una sola ya sea en "Empleados", en "Recibo" o en una tabla aparte independiente por ejemplo "Direcciones", **pero nunca en ambas a la vez**. Porque al momento de hacer un cambio habrá que hacerlo en diversas tablas y si lo olvidamos generamos "inconsistencia y redundancia" y tal vez no sepamos luego cuál es la dirección verdadera.. Podemos también crear relaciones entre las tablas mediante una clave externa donde, por ejemplo, podríamos tener una clave en la tabla "Empleados" que referencie la dirección alojada en "Direcciones" o bien en "Recibo" se cree una relación con "Empleados" para evitar crear una tabla aparte "Direcciones". Lo importante es evitar tener entradas duplicadas en diferentes tablas.

Tercera forma normal:

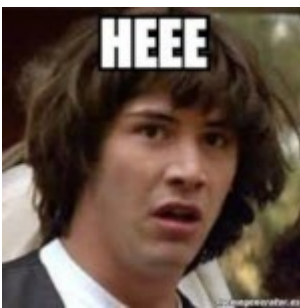
Decimos que una base de datos está en tercera forma normal si cumple las dos primeras y además:

- Se eliminan los campos que no dependen de la clave.

Aplicar al 100% la tercera forma normal a veces resulta complejo y deriva en crear muchas tablas pequeñas que podría acarrear problemas de rendimiento. Pero básicamente consiste en que si hay atributos que no tienen relación con la clave primaria, hay que eliminarlos y colocarlos en una tabla separada, relacionando ambas tablas por medio de una clave externa. Es decir, no debería haber ninguna dependencia transitiva.

Si te has quedado..

Si te has quedado algo descolocado no te preocupes que solo estoy exponiendo la teoría.



Esto llevado a la práctica en realidad se hace aún más fácil de comprender y recordar, en las próximas entradas cuando comencemos a diseñar bases de datos recordaremos y aplicaremos estas formas normales.

Por lo pronto he encontrado un blog que lo explica muy claro y me gustaría que lo leyeras para asegurarte una mejor comprensión de los conceptos y reglas de normalización: Formas normales

Las 12 reglas de Codd para el modelo relacional.

Recordarás que **Edgar Codd** es el creador del modelo relacional, este tipo vino a cambiar las cosas, pero al ver que muchos programadores y diseñadores de las bases de datos cometían errores y eran muy cochinos, decidió solucionarles la vida una vez más **especificando una docena de reglas que determinan un modelo relacional exitoso**. Y nos evita futuros problemas de **inconsistencia, redundancia** y dolores de cabeza o llamadas de la vecina de la suegra de la tía de tu ex novia (para que veas lo molestas que se pueden volver las relaciones ^^).



Las 12 reglas resumidas y explicadas con palabras terrícolas, dicen así:

1. **Información:** En el modelo relacional toda la información de la base de datos debe estar representada **explícitamente** en el esquema lógico. (**Nada de datos ocultos o "escondidos, implícitos, etc."**. **Toda la jodida información debe estar en la base de datos**).
2. **Acceso garantizado:** En el modelo relacional todo dato debe ser accesible conociendo el valor de su clave primaria y el nombre de la columna (Atributo) que contiene el dato. **Con esto queremos decir, que no deben existir datos sin claves ni atributos que dificulten su rápido y fácil acceso**.
3. **Tratamiento sistemático de valores nulos:** El sistema de bases de datos debe permitir el tratamiento adecuado de los valores **"null"** y estos deben estar presentes en aquellos atributos no especificados. **Además de especificar correctamente cuales permiten valores "null" y cuáles no**.

Por ejemplo, es obvio que no deberías jamás permitir valores nulos en un atributo que es clave primaria, porque en ese caso estarías no sólo violando esta regla sino también las dos anteriores, en el caso que luego quisieras acceder a ese dato. ¿Cómo carajos lo ibas a hacer si la clave primaria es "null?".

4. **Catálogo en línea basado en el modelo relacional:** Esta regla refiere a que todos los datos del modelo deben soportar un catálogo basado en relaciones que le permitan al usuario acceder a la estructura relacional total del modelo.
El usuario autorizado debe poder ver completa la estructura de las relaciones entre los datos!.
5. **Sublenguaje de datos completos:** el modelo debe contar con un sublenguaje capaz de manipular los datos de la base de datos. Este debe permitir realizar operaciones. ¿Recuerdas que hablamos de que antes los programadores debían desarrollar un lenguaje aparte para administrar los datos?, pues en este caso se utiliza **SQL** que veremos más adelante.
6. **Actualización de vistas: una vista es una tabla cuyo contenido ha sido generado por una consulta de un usuario,** un resultado.. por ejemplo “Muéstrame los nombres de los profesores de la tabla profesores”. Esta vista debe mostrar la información más actualizada siempre.
7. **Inserciones, modificaciones, eliminaciones de alto nivel:** Cualquier operación de este tipo, ya sea modificar, insertar, eliminar datos o registros debe ser en conjunto de filas o registros y no registro a registro. Con esto queremos decir que si queremos modificar el atributo “año” de todos los alumnos debemos hacerlo mejor en conjunto mediante una sentencia y no uno por uno como unos capullos. Al igual que si quisiéramos eliminar un registro, tabla, etc.
8. **Independencia física:** Los datos deben poder ser accesibles desde la lógica de la BD aunque se modifique el almacenamiento físico de la misma.
9. **Independencia lógica: Los programas no deben verse afectados por cambios en los datos de la base de datos.** La base de datos es independiente del programa de gestión o software que la consulte. No podemos utilizar la base de datos para almacenar código o variables vitales para el funcionamiento de un programa, ya que si son borrados dejaría de funcionar. Esto es bastante obvio, y de hecho es una gran ventaja que el tratamiento de los datos sea independiente del sistema o software que los gestiona.
10. **Independencia de Integridad:** Las reglas de integridad deben almacenarse en la base de datos (en el diccionario) y no en los programas de aplicación. Es obvio que quien especifica la integridad de los datos es la base de datos, no el programa que la consulta!!.

11. **Independencia de la distribución:** Anteriormente en otro post te dije que las bases de datos pueden estar distribuidas, una misma base de datos puede estar fragmentada y alojada en diferentes servidores para algún propósito en específico como mejorar la velocidad de acceso, etc. Pues, esta regla hace hincapié en que para el usuario que visualiza la base de datos esta debe mostrarse íntegra a pesar de estar fragmentada!. Al igual que cuando se distribuyen los datos lo hacen por todo el sistema..
12. **No subversión:** Si un sistema relacional tiene un (solo registro a la vez) de bajo nivel de lenguaje, ese bajo nivel no puede ser utilizado para subvertir o pasar por alto las reglas de integridad y las limitaciones expresadas en el lenguaje relacional de alto nivel. Se refiere a violar las reglas de integridad y limitaciones de SQL mediante otro lenguaje, etc.. Lo que podría causar estragos..

Estas reglas las debes respetar!. Y no son las únicas, existen diversos requisitos a cumplir a la hora de trabajar con el modelo relacional!

Requerimientos, características de transacciones: ACID

IMPORTANTE: Toda base de datos debe cumplir estos requerimientos!

Los requisitos **ACID**, (acrónimo en inglés de Atomicity, Consistency, Isolation an Durability), deben ser cumplidos por toda base de datos relacional con la que trabajemos. Es importante que lo comprendas a fondo, cada jodida palabra!.

Transacción: Interacción con una estructura de datos compleja que está compuesta por un conjunto de órdenes o instrucciones que se ejecutan una después de la otra, de una sola vez.

Podemos en un principio (porque estamos empezando con bases de datos) pensar en una transacción, "como un intercambio mediante una instrucción o sentencia".

Las características ACID son un requerimiento muy normal a la hora de pensar en cambios de datos dentro de bases de datos mediante transacciones **no cumplirlos implica grandes conflictos que normalmente no son de fácil solución.**

Sin embargo la mayoría de los **RDBMS** mediante el lenguaje **DCL** (Lenguaje de control de datos 'SQL'), cumplen con estos requerimientos y nos obligan a cumplirlos.

Estos son:

Atomicidad:

La palabra se puede usar para referirse a agrupar o unificar.

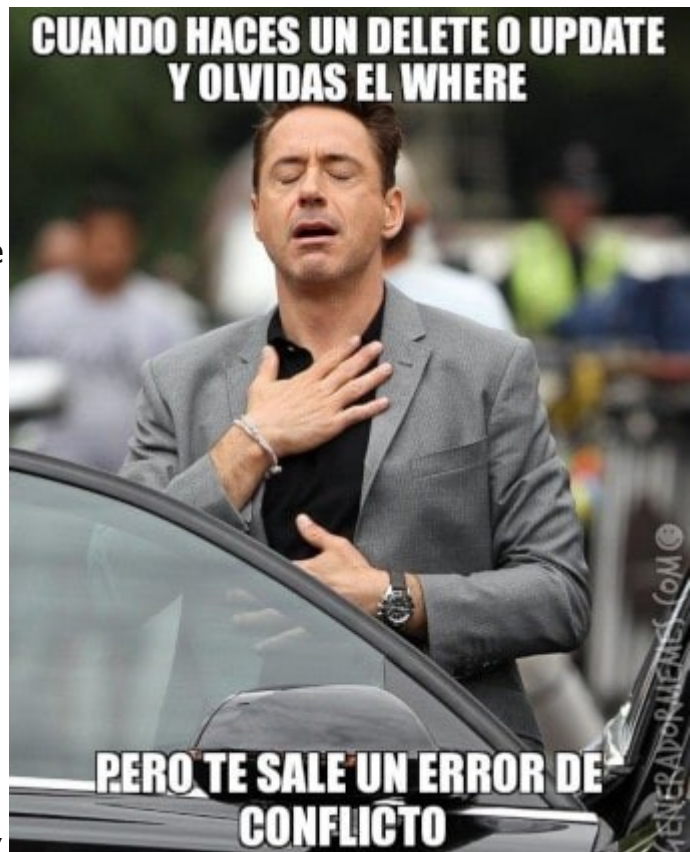
La atomización de la información consiste en tratarla como un conjunto, no cumplir la **atomicidad de información** sería como el ejemplo brindando cuando expliqué el modelo relacional, como tener todos los datos regados por todas partes dificultando su comprensión. Pero en este caso también se refiere a las **transacciones** que se realizan con los datos.

Mediante el lenguaje SQL podemos realizar consultas, transferir, manipular, añadir, actualizar lo que se conoce como TRANSACCIÓN, o lo que es lo mismo, cada ejecución que implique una modificación de datos en nuestra DB.

Cumplir con el principio de atomicidad significa realizar esas transacciones u operaciones en forma de conjunto (" O se ejecutan todas o Ninguna").

Suponiendo que tuviésemos que actualizar diversas columnas de nuestra base de datos mediante una transacción y se produjera un **error** en una sola de ellas, nuestra base de datos **no debería sufrir ningún cambio, en caso contrario, si la transacción completa se pudiera ejecutar sin errores, entonces si, se ejecuta completa y se aplican los cambios!.**

Con esto queremos decir que si tienes un error en una línea de una transacción en **SQL**, no pasará nada.. No vas a arruinarlo todo. **Ahora el problema surge cuando la transacción está mal diseñada, pero no contiene errores :**



El clásico "Delete sin Where" (eliminar sin ninguna condición).. Es un meme.. Y fíjate, si haces un delete sin where pero surge algún error, no se ejecuta. Y la **atomicidad** propiedad de **SQL** y el **RDBMS** te salva el culo!!.

Consistencia (Integridad):

Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de Integridad de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido. "La Integridad de la Base de Datos nos permite asegurar que los datos son exactos y consistentes, es decir que estén siempre intactos, sean siempre los esperados y que de ninguna manera cambien ni se deformen. De esta manera podemos garantizar que la información que se presenta al usuario será siempre la misma."

Aislamiento (Isolation):

El aislamiento implica que cada transacción se debe ejecutar de forma simultánea (una tras otra) de tal manera que **el estado intermedio de una transacción no sea visible por otra y de esa manera se mantengan aisladas unas de otras**. Lo cual podremos ver en profundidad más adelante a la hora de trabajar con transacciones en SQL!.

Durabilidad (Persistencia):

La durabilidad significa que una vez que se confirmó una transacción (**commit**), esta persistirá, incluso ante eventos como pérdida de alimentación eléctrica, errores o caídas del sistema. Por ejemplo, en las bases de datos relacionales, una vez que se ejecuta un grupo de sentencias SQL, los resultados tienen que almacenarse inmediatamente (incluso si la base de datos se cae inmediatamente después). Esto nos asegura que la transacción fue realizada y los cambios guardados de forma persistente en la base de datos a pesar de posibles errores externos surgidos después de aplicarlos.

RDBMS:

La mayoría de los **RDBMS** y lenguajes de manejo de datos como **SQL** cumplen con estos requisitos. Pero es importante tenerlos en cuenta a la hora de realizar transacciones para comprender su funcionamiento y elaboración.



En la actualidad es muy común utilizar software automatizado para mejorar y facilitarnos casi todo el proceso y estos abarcan y obligan a respetar la mayoría de

estas reglas y conceptos. Pero como bien tu y yo sabemos, ningún software puede salvarnos de cometer errores humanos. Así que si los has comprendido medianamente y los tendrás presente en el futuro, es momento de comenzar a crear nuestra base de datos en la siguiente lección.