1. Historia y Evolución de los SGBD (Resumen)

1. Introducción

El término bases de datos fue escuchado por primera vez en un simposio celebrado en California en 1963.
En una primera aproximación, se puede decir que una base de datos es un conjunto de información relacionada que se encuentra agrupada o estructurada.

Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos.
Por su parte, un sistema de Gestión de Bases de datos es un tipo de software muy específico dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan; es decir, una agrupación de programas que sirven para definir, construir y manipular una base de datos, permitiendo así almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

2. Orígenes

Los orígenes de las bases de datos se remontan a la Antigüedad donde ya existían bibliotecas y toda clase de registros, se utilizaban entre otras cosas para recoger información sobre las cosechas y censos. Sin embargo, su búsqueda era lenta y poco eficaz y no se contaba con la ayuda de máquinas que pudiesen reemplazar el trabajo manual.

Las bases de datos se desarrollan a partir de las necesidades de almacenar grandes cantidades de información o datos. El concepto de bases de datos ha estado siempre ligado a la informática y se empleó desde la aparición de las primeras computadoras.

En 1884 Herman Hollerith creó la máquina automática de tarjetas perforadas, se trataba de una máquina tabuladora o censadora, basada en tarjetas perforadas.

En la década de los cincuenta se da origen a las cintas magnéticas, para automatizar la información y hacer respaldos, a través de este mecanismo se comenzó

a automatizar información, con la desventaja de que solo se podía hacer de forma secuencial.
Las aplicaciones informáticas de los años sesenta acostumbraban a darse totalmente por lotes (batch) y estaban pensadas para una tarea muy específica.
Cada aplicación utilizaba ficheros de movimientos para actualizar y/o para consultar uno o dos ficheros maestros o, excepcionalmente, más de dos.
Cada vez que se le quería añadir una aplicación que requería el uso de algunos de los datos que ya existían y de otros nuevos, se diseñaba un fichero nuevo con todos los datos necesarios esto provocaba redundancia.
A medida que se fueron introduciendo las líneas de comunicación, los terminales y los discos, se fueron escribiendo programas que permitían a varios usuarios consultar los mismos ficheros on-line y de forma simultánea. Más adelante fue surgiendo la necesidad de hacer las actualizaciones también on-line.
A medida que se integraban las aplicaciones, se tuvieron que interrelacionar sus ficheros y fue necesario eliminar la redundancia.
El nuevo conjunto de ficheros se debía diseñar de modo que estuviesen interrelacionados.
El acceso on-line y la utilización eficiente de las interrelaciones exigían estructuras físicas que diesen un acceso rápido, como por ejemplo los índices, las multilistas, etc.

Estos conjuntos de ficheros interrelacionados, con estructuras complejas recibieron al principio el nombre de Data Banks, y después, a inicios de los años setenta, el de Data Bases.

La utilización de estos conjuntos de ficheros por parte de los programas de aplicación era excesivamente compleja, de modo que, especialmente durante la segunda mitad de los años setenta, fue saliendo al mercado software más sofisticado: los **Data Base Management Systems**, que aquí denominamos **Sistemas de Gestión de BD** (**SGBD**).
Una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única e integrada.

Podríamos agregar que una base de datos de un **SI** (**Sistema de Información**) es la representación integrada de los conjuntos de entidades instancia correspondientes a las diferentes entidades tipo del SI y de sus interrelaciones.
Esta representación informática (o conjunto estructurado de datos) debe poder ser utilizada de forma compartida por muchos usuarios de distintos tipos.

3. Década de los 60.

En los 60 se popularizara el uso de los discos, cosa que fue un adelanto muy efectivo en la época, debido a que a partir de este soporte se podía consultar la información directamente, sin tener que saber la ubicación exacta de los datos.

Nacen las primeras generaciones de bases de datos de red y las bases de datos jerárquicas, ya que era posible guardar estructuras de datos en listas y arboles.

Otro de los principales logros de los años sesenta fue la alianza de IBM y American Airlines para desarrollar **SABRE**, un sistema operativo que manejaba las reservas de vuelos, transacciones e informaciones sobre los pasajeros de la compañía American Airlines.

En esta misma década, se llevo a cabo el desarrollo del IDS desarrollado por Charles Bachman, esto supuso la creación de un nuevo tipo de sistema de bases de datos conocido como modelo en red que permitió la creación de un estándar en los sistemas de bases de datos gracias a la creación de nuevos lenguajes de sistemas de información.

CODASYL (**Conference on Data Systems Languages**) era un consorcio de industrias informáticas que tenían como objetivo la regularización de un lenguaje de programación estándar. Aunque trabajaron en varios lenguajes de programación como COBOL, nunca llegaron a establecer un estándar fijo, proceso que se llevo a cabo por ANSI.

4. Década de los 70 – Sistemas Centralizados.

Los primeros SGBD de los años sesenta todavía no se les denominaba así. Estaban orientados a facilitar la utilización de grandes conjuntos de datos en los que las interrelaciones eran complejas.
Estos sistemas trabajaban exclusivamente por lotes (batch).

Al aparecer los terminales de teclado, conectados al ordenador central (**Mainframes**) mediante una línea telefónica, se empiezan a construir grandes aplicaciones on-line transaccionales (**OLTP**).
Para escribir los programas de aplicación se utilizaban lenguajes de alto nivel como Cobol o PL/I, se disponía también de instrucciones y de subrutinas especializadas para tratar las BD que requerían que el programador conociese muchos detalles del diseño físico, y que hacían que la programación fuese muy compleja.
Puesto que los programas estaban relacionados con el nivel físico, se debían modificar continuamente cuando se hacían cambios en el diseño y la organización de la BD. La preocupación básica era maximizar el rendimiento: el tiempo de respuesta y las transacciones por segundo.

Edgar Frank Codd, científico informático inglés, definió el modelo relacional a la par que publicó una serie de reglas para los sistemas de datos relacionales a través de su artículo "**Un modelo relacional de datos para grandes bancos de datos compartidos**".

Este hecho dio paso al nacimiento de la segunda generación de los Sistemas Gestores de Bases de Datos.

Como consecuencia de esto, durante la década de 1970, Lawrence J. Ellison desarrolló el Relational Software System, o lo que es lo mismo, lo que actualmente se conoce como Oracle Corporation, desarrollando así un sistema de gestión de bases de datos relacional con el mismo nombre que dicha compañía.
Finalmente IBM desarrolló unas técnicas para construir un sistema de bases de datos relacionales eficientes, las cuales llamó System R; por otro lado Ingres se desarrolló en la UBC en los años de 1974 a 1977.
Ingres utilizaba un lenguaje de consulta, llamado QUEL, dando pie a la creación de sistemas como Ingres Corporación, MS SQL Server, Sybase, PACE Wang, y Britton Lee-. Por su parte, el Sistema R utilizó el lenguaje de consulta Secuela, el cual ha contribuido al desarrollo de SQL / DS, DB2, Allbase, Oracle y SQL Non-Stop.
Posteriormente en la época de los ochenta también se desarrollará el **SQL** (**Structured Query Language**) o lo que es lo mismo un lenguaje de consultas o lenguaje declarativo de acceso a bases de datos relacionales.

Cabe destacar que ORACLE es considerado como uno de los sistemas de bases de datos más completos que existen en el mundo, actualmente sufre la competencia del SQL Server de la compañía Microsoft y de la oferta de otros Sistemas Administradores de Bases de Datos Relacionales con licencia libre como es el caso de PostgreSQL, MySQL o Firebird que aparecerían posteriormente en la década de 1990.

2. Historia y evolución de los SGBD II (Resumen)

5. Década de los 80 – SGBD Relacionales.

A principios de los años ochenta comenzó el auge de la comercialización de los sistemas relacionales, y SQL comenzó a ser el estándar de la industria debido a que su nivel de programación era sencillo y relativamente bajo.
Los SGBD de los años setenta eran demasiado complejos e inflexibles, y sólo los podía utilizar un personal muy cualificado.

Durante la década de los 80 aparecen y se extienden muy rápidamente los ordenadores personales, también surgen software para equipos monousuario con los cuales es muy fácil utilizar y crear conjunto de datos, los cuales se denominan **personal data bases**.

En la década de los años 80', se desarrolló el SQL (Structured Query Language), un lenguaje de consultas que permite consultar, valga la redundancia, con el fin de recuperar información de una base de datos y a su vez, hacer cambios sobre esa misma base, de forma sencilla. Permitía analizar gran cantidad de información y especificar varios tipos de operaciones.
SQL comenzó a ser el modelo estándar de las industrias, con su base de datos bajo un sistema de tablas (filas y columnas), pudo competir con las bases jerárquicas y de redes, ya que su nivel de programación era sencillo y el nivel era relativamente bajo.
El ORACLE está considerado como uno de los sistemas de bases de datos más completos del mundo, su dominio en el mercado fue casi total hasta muchos años después, pero esto cambió con la aparición del SQL Server de Microsoft.

6. Década de los 90: Distribución, C/S y 4GL

A finales de los ochenta y principios de los noventa, las empresas se han encontrado con el hecho de que sus departamentos han ido comprando ordenadores departamentales y personales, y han ido haciendo aplicaciones con BD. El resultado ha sido que en el seno de la empresa hay numerosas BD y varios SGBD de diferentes tipos o proveedores.

Los SGDB actuales permiten que un programa pueda trabajar con diferentes BD como si se tratase de una sola, esto se lo conoce como bases de datos distribuida.

Esta distribución ideal se consigue cuando las diferentes BD son soportadas por una misma marca de SGBD, es decir, cuando hay homogeneidad.
En la actualidad, gracias principalmente a la estandarización del lenguaje SQL, los SGBD de marcas diferentes pueden darse servicio unos a otros y colaborar para dar servicio a un programa de aplicación.
Además de esta distribución "impuesta", al querer tratar de forma integrada distintas BD preexistentes, también se puede hacer una distribución "deseada", diseñando una BD distribuida físicamente, y con ciertas partes replicadas en diferentes sistemas.

Las razones básicas por las que interesa esta distribución son las siguientes:

Disponibilidad

La disponibilidad de un sistema con una BD distribuida puede ser más alta, porque si queda fuera de servicio uno de los sistemas, los demás seguirán funcionando.

Coste

Una BD distribuida puede reducir el coste. En el caso de un sistema centralizado, todos los equipos usuarios, que pueden estar distribuidos por distintas y lejanas áreas geográficas, están conectados al sistema central por medio de líneas de comunicación. El coste total de las comunicaciones se puede reducir haciendo que un usuario tenga más cerca los datos que utiliza con mayor frecuencia; por ejemplo, en un ordenador de su propia oficina o, incluso, en su ordenador personal.

Para esta distribución de datos se utiliza la arquitectura clirnte/servidor.
Un programa de aplicación que un usuario ejecuta en su PC pide ciertos datos de una BD donde, a su vez, se ejecuta el SGBD relacional que la gestiona. El programa de aplicación es el cliente y el SGBD es el servidor.

La facilidad para disponer de distribución de datos no es la única razón, ni siquiera la básica, del gran éxito de los entornos C/S en los años noventa.Tal vez el motivo fundamental ha sido la flexibilidad para construir y hacer crecer la configuración informática global de la empresa, así como de hacer modificaciones en ella, mediante hardware y software muy estándar y barato.
El éxito de las BD ha llevado a la aparición de los Fourth Generation Languages (4GL), lenguajes muy fáciles y potentes, especializados en el desarrollo de aplicaciones fundamentadas en BD.

7. Tendencias actuales

La orientación a objetos (OO) encuentra el éxito al final de los años ochenta, en el desarrollo de software básico, en las aplicaciones de ingeniería industrial y en la construcción de interfaces gráficas con los usuarios, ha hecho que durante la década de los noventa se extendiese en prácticamente todos los campos de la informática.
La utilización de lenguajes como C++ o Java requiere que los SGBD relacionales se adapten a ellos con interfaces adecuadas.
La rápida adopción de la web a los SI hace que los SGBD incorporen recursos para ser servidores de páginas web, como por ejemplo la inclusión de SQL en guiones HTML, SQL incorporado en Java, etc.
Durante estos últimos años se ha empezado a extender un tipo de aplicación de las BD denominado Data Warehouse, o almacén de datos, que también produce algunos cambios en los SGBD relacionales del mercado.

Los datos de este gran almacén, el Data Warehouse, se obtienen por una replicación más o menos elaborada de las que hay en las BD que se utilizan en el trabajo cotidiano de la empresa. Estos almacenes de datos se utilizan exclusivamente para hacer consultas, de forma especial para que lleven a cabo estudios los analistas financieros, los analistas de mercado, etc.

Actualmente, los SGBD se adaptan a este tipo de aplicación, incorporando, por ejemplo, herramientas como las siguientes:

La creación y el mantenimiento de réplicas, con una cierta elaboración de los datos.
La consolidación de datos de orígenes diferentes.
La creación de estructuras físicas que soporten eficientemente el análisis multidimensional.

Gestores de Bases de Datos

Las aplicaciones más usadas son para gestiones de empresas e instituciones públicas, así como en entornos científicos, para almacenar la información experimental.

Una base de datos es un sistema compuesto por un conjunto de datos, los cuales están almacenados en discos, a los que se accede directamente y un conjunto de programas que regulen o manejen ese conjunto de datos.
Mientras que un sistema de Gestión de Bases de Datos es un software que sirve de interfaz entre la base de datos, el usuario y las aplicaciones que se utilizan.

Los mejores gestores de base de datos

El principal lenguaje de base de datos y el más utilizado desde que se conoce la programación de gestión, es el Structured Query Language (SQL). Este, de consulta estructurada, facilita el acceso a la gestión de las bases de datos relaciones, lo que permite realizar tareas en ellas y realizar consultas, que sirvan para obtener, agregar, eliminar o modificar información.

Para el desarrollo de este lenguaje hay que utilizar un gestor de base de datos, de los que hay muchos, unos de acceso libre y otros de pago. Veamos cuáles son, primeramente, los gestores de base de datos de pago:

Oracle

El sistema de gestión de base de datos relacional más utilizado en el mercado.
Es propiedad de Oracle Corporation y fue desarrollado en 1977.
Su versatilidad le facilita ejecutarse en casi todas las plataformas existentes, Windows, Unix, Linux, MAC OS, entre otros.

SQL Server

En competencia directa a Oracle, está SQL Server de Microsoft. Los dos ocupan gran parte del mercado en el sector de base de datos. Son muy parecidos en algunas de sus características y funciones, aunque tienen sus marcadas diferencias.

Gestores de base de datos de acceso libre

MySQL

Este es de simple instalación y actúa de lado del cliente o servidor, es de código abierto y tiene licencia comercial disponible. Pertenece a Oracle Corporation y gestiona las bases de datos relacionales, con funciones multiusuario y es el más usado dentro del software libre.

Fire Bird

De gran potencia y muy sencillo a la vez, este sistema de gestión de base de datos relacional SQL, es uno de los mejores gestores Open Source (Código abierto) o libres. Es compatible con Windows y Linux.

Tiene buen soporte para los procedimientos almacenados, las transacciones compatibles con ACID y con los métodos de acceso múltiple como Nativo, Python, .NET, etc...

Como vemos, son múltiples las posibilidades que tenemos de acceso a gestores de base de datos, tanto adquiriendo licencias de pago como acudiendo a software libre. En función de los gustos, formas de trabajar y necesidades de cada uno, seguro encontraremos distintos gestores de base de datos que pueden satisfacernos en pro de nuestro trabajo.

METODOLOGÍAS AGILES Y GESTION DE PROYECTOS

> GITHUB y sus funciones:

Publicar repositorios de código, **colaborar en otros proyectos y publicar los tuyos propios**, de codigo abierto donde cualquiera puede utilizar el codigo que se comparte

> Crear cuenta en Github

Accede a **github.com** y crea una cuenta.

Selecciona el plan personal gratuito con repositorio público. No te olvides de terminar la verificación mediante correo electrónico.

Crea un proyecto en la opción “Create a Proyect”. Incluye el nombre de tu repositorio y selecciona la opción “Create repository”.

No cierres esta ventana porque vas a necesitar algunos de los datos que ahí se muestran para clonar el repositorio mediante URL.

> Clonar el repositorio con Visual Studio Code

$ git config --global user.name “tunombre"
$ git config --global user.email tumail@dominio.com

Clonar el proyecto creado en Github en Visual Studio Code.

Para ello, nos vamos al menú “view” seleccionamos “**command palette**”.
En ese punto buscamos “**git: Clone**”.

La herramienta nos pedirá la ruta y ahí es donde tenemos que pegar la URL del repositorio que hemos creado en el punto anterior en Github

Desde la pestaña de Git podemos hacer el “**commit**” de los ficheros. En primer lugar, incluiremos los ficheros a los que queremos hacer stage pulsando en “+” o “todos”. Además debemos escribir el mensaje del Commit e Intro para terminar.

Para hacer push de los cambios o pull para copiar los datos de servidor podemos hacerlo de dos maneras:
- Podemos pulsar en la parte derecha de la barra de git (icono tres puntos) y elegir la opción *push* o *pull*.
- Podemos pulsar en la parte inferior sobre sincronizar (en este caso haríamos *push* y *pull*).

Ver los cambios en Github

La URL de visualización tendrá el siguiente aspecto:
https://nombredetucuenta/github.io/nombredeturepositorio/ (La página mostrada es la página index.html).

MODULO PROGRAMADOR 2022
RESUMEN DEL BLOQUE I

lista de temas bloque 1

1. **INTRODUCCIÓN A BASE DE DATOS**
2. **METODOLOGÍAS ÁGILES Y GESTIÓN DE PROYECTOS**
3. **ÉTICA Y DEONTOLOGÍA PROFESIONAL**

1. INTRODUCCIÓN A BASE DE DATOS

1.1 Historia y Evolución de los SGBD

1.2 Historia y evolución de los SGBD II

1.3 Introducción a las Bases de Datos

1.4 Sistemas Gestores de Bases de Datos

Resumen de

1.5 Diseño de bases de datos relacionales

Etapas de diseño

1. Diseño conceptual.

Su objetivo es definir las entidades y las relaciones entre ellos de forma abstracta, sin centrarse en ningún modelo lógico en concreto (como el relacional, el orientado a objetos, el jerárquico o el de red).
Herramienta: Modelo conceptual de datos. Se usa alguna variante del modelo entidad-relación para las bases de datos relacionales.
Resultado: Esquema conceptual de la base de datos.

Conceptos

- **Entidad:** Es el menor objeto con signicado en una instancia.
- **Atributo:** Es cada uno de los componentes que determinan una entidad.
- **Atributos monovalorados y multivalorados:** uno o varios valores.
- **Atributos simples y compuestos:** es compuesto cuando puede descomponerse en otros componentes o atributos más pequeños.
- **Clave:** Es un atributo o conjunto de atributos cuyos valores identifican unívocamente cada entidad.

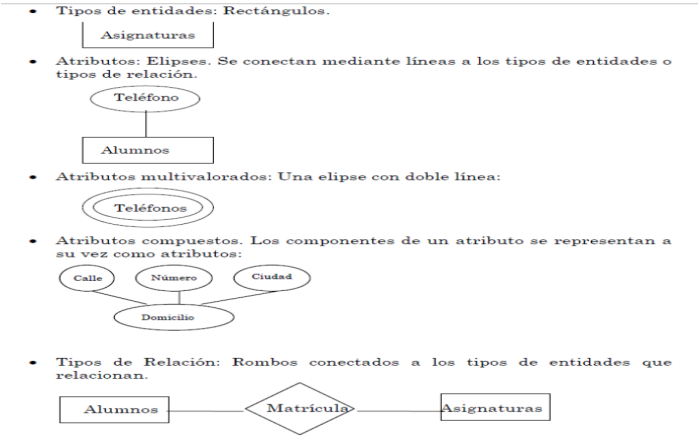
Superclave. Es cualquier conjunto de atributos que pueden identicar unívocamente a una tupla.

Clave candidata. Es el menor conjunto de atributos que puede formar clave. Puede haber varias en una tabla.

ClavePrimaria. Es la clave candidata que distingue el usuario para identicar unívocamente cada tupla.

- **Tipo de entidad.** Es el conjunto de entidades que comparten los mismos atributos (aunque con diferentes valores)
- **Relación.** Es una correspondencia entre dos o más entidades. Se habla de relaciones binarias cuando la correspondencia es entre dos entidades, ternarias cuando es entre tres, y así sucesivamente.
- **Tipos de relación.** Representan a todas las posibles relaciones entre entidades del mismo tipo.

Diagramas entidad-relación (E-R) y Elección de los tipos de entidad, sus atributos y relaciones



2. Diseño lógico.

Su objetivo es definir el esquema de la base de datos según el modelo que implementa el SGBD objetivo. Herramienta: Modelo lógico de datos. Se usa el modelo lógico que implemente el sistema de gestión de bases de datos objetivo, pero es independiente de los aspectos físicos. Se usan técnicas formales para verificar la calidad del esquema lógico; la más usual es la normalización. En el modelo relacional se usan las tablas.
Resultado: Esquema lógico de la base de datos.

El modelo relacional

El concepto principal de este modelo es la relación o tabla. Es importante no confundir la tabla con las relaciones del modelo E-R. Aquí las relaciones se aplican tanto a tipos de relaciones como a tipos de entidades. En este modelo no se distingue entre tipos de entidades y tipos de relaciones porque la idea es que una relación o tabla expresa la relación entre los tipos de valores que contiene.
Conceptos de este modelo:

- Entidad. Igual que en el modelo E-R. También se les llama tuplas o las de la relación.
- Atributo. Igual que en el modelo E-R. También se le llaman campos o columnas de la relación. El dominio de los atributos tiene que ser simple: no se admiten atributos multivalorados ni compuestos.
- Esquema de una relación. Viene dado por el nombre de la relación y una lista de atributos. Es el tipo de entidad.
- Conjunto de entidades. Relación o tabla.
- Clave.
- Instancia de una relación. Son conjuntos de entidades. Cada entidad se representa como una tupla. Cada componente de la tupla corresponde con el valor del atributo correspondiente, según el orden enunciado en el esquema de la relación.

Por ejemplo, una instancia de la relación Alumnos sería:

{ (01234567Z, Manuel Vázquez Prieto, Calle del Jazmín 7 4 Izq, 91-12345678, COU = SÍ), ...}

En el modelo relacional no se representan diagramas del esquema de la BD. Por el contrario, el esquema relacional se representa por los conjuntos de entidades como hemos visto antes (nombre de la tabla y entre paréntesis el nombre de sus atributos). Las instancias de una relación se representan con tablas, como se muestra en el ejemplo del conjunto de entidades Alumnos.

Relación o tabla

Según el modelo relacional (desde que Codd lo enunció) el elemento fundamental es lo que se conoce como **relación**, aunque más habitualmente se le llama **tabla** (o también array o matriz). Codd definió las relaciones utilizando un lenguaje matemático, pero se pueden asociar a la idea de tabla (de filas y columnas) ya que es más fácil de entender. No hay que confundir la idea de relación según el modelo de

Codd, con lo que significa una relación en el modelo Entidad/Relación de **Chen.** No tienen nada que ver. Las relaciones constan de:
ü**Atributos**. Referido a cada propiedad de los datos que se almacenan en la relación (nombre, dni,...).
ü**Tuplas.** Referido a cada elemento de la relación. Por ejemplo si una relación almacena personas, una tupla representaría a una persona en concreto. Puesto que una relación se representa como una tabla; podemos entender que las columnas de la tabla son los atributos; y las filas, las tuplas.

atributo 1	atributo 2	atributo 3	....	atributo n	
valor 1,1	valor 1,2	valor 1,3	....	valor 1,n	← tupla 1
valor 2,1	valor 2,2	valor 2,3	....	valor 2,n	← tupla 2
.....	.....	.....	....	.....	....
valor m,1	valor m,2	valor m,3	....	valor m,n	← tupla m

Tupla o registro.
Cada una de las filas de la relación. Se corresponde con la idea clásica de registro. Representa por tanto cada elemento individual de esa relación. Tiene que cumplir que:
üCada tupla se debe corresponder con un elemento del mundo real.
üNo puede haber dos tuplas iguales (con todos los valores iguales).
Atributo – Columnas.
Un Atributo en el Modelo Relacional representa una propiedad que posee esa Relación y equivale al atributo del Modelo E-R.
Se corresponde con la idea de campo o columna.
Dominio
Un dominio contiene todos los posibles valores que puede tomar un determinado atributo. Dos atributos distintos pueden tener el mismo dominio.
Un dominio en realidad es un conjunto finito de valores del mismo tipo. A los dominios se les asigna un nombre y así podemos referirnos a ese nombre en más de un atributo, facilitando la definición de los mismos.
La forma de indicar el contenido de un dominio se puede hacer utilizando dos posibles técnicas:

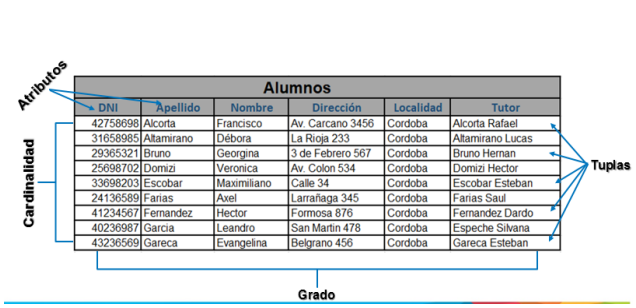
ü**Intensión.** Se define el nomino indicando la definición exacta de sus posibles valores. Por intensión se puede definir el dominio de edades de los trabajadores como: ***números enteros entre el 16 y el 65*** (un trabajador sólo podría tener una edad entre 16 y 65 años).

ü**Extensión**. Se indican algunos valores y se sobreentiende el resto gracias a que se autodefinen con los anteriores. Por ejemplo el dominio localidad se podría definir por extensión así: ***Valencia, Valladolid, Madrid,...***
Además pueden ser:

Generales. Los valores están comprendidos entre un máximo y un mínimo
Restringidos. Sólo pueden tomar un conjunto de valores.

Atributo	Nombre del Dominio	Descripción	Definición
noficina	NUM_OFICINA	Posibles valores de número de oficina	3 caracteres, rango 100 - 990
calle	NOM_CALLE	Nombres de calles y numero de Santiago donde se ubica la oficina	25 caracteres
area	NOM_AREA	Área de Santiago en la que se encuentra ubicada la oficina	20 caracteres
telefono	NUM_TEL_FAX	Números de teléfono de Santiago	9 caracteres
fax	NUM_TEL_FAX	Números de teléfono de Santiago	9 caracteres

Grado
Indica el tamaño de una relación en base al número de columnas (atributos) de la misma. Lógicamente cuanto mayor es el grado de una relación, mayor es su complejidad al manejarla.
Cardinalidad
Número de tuplas de una relación, o número de filas de una tabla.



Sinónimos

Los términos vistos anteriormente tienen distintos sinónimos según la nomenclatura utilizada. A ese respecto se utilizan tres nomenclaturas:

Terminología Relacional		Terminología de Tablas		Terminología de Archivo
Relación	=	Tabla	=	Archivo
Tupla	=	Fila	=	Registro
Atributo	=	Columna	=	Campo
Grado	=	Número de columnas	=	Número de campos
Cardinalidad	=	Número de filas	=	Número de registros

Se han subrayado en la tabla los términos que más se usan.

Restricciones

Se trata condiciones de obligado cumplimiento por las tuplas de la base de datos.
Las hay de varios tipos:
Inherentes

Son aquellas que no son determinadas por los usuarios, sino que son definidas por el hecho de que la base de datos sea relacional. Las más importantes son:

- ✓ **No puede haber dos tuplas iguales**
- ✓ **El orden de las tuplas no es significativo**
- ✓ **El orden de los atributos no es significativo**
- ✓ **Cada atributo sólo puede tomar un valor en la tupla y dominio en el que está inscrito**

Semánticas

El modelo relacional permite a los usuario incorporar restricciones personales a los datos. Se comentan las diferentes reglas semánticas a continuación:
Clave principal (primary key)
También llamada clave primaria. Marca uno o más atributos como identificadores de la tabla. De esa forma en esos atributos las filas de la tabla no podrán repetir valores ni tampoco dejarlos vacíos.
Unicidad (unique)
Impide que los valores de los atributos marcados de esa forma, puedan repetirse.
Esta restricción debe indicarse en todas las claves alternativas.
Al marcar una clave primaria se añade automáticamente sobre los atributos que forman la clave un criterio de unicidad.
Obligatoriedad (not null)
Prohíbe que el atributo marcado de esta forma quede vacío (es decir impide que pueda contener el valor nulo, null).
Regla de validación (check)
Condición lógica que debe de cumplir un dato concreto para darlo por válido. Por ejemplo restringir el campo sueldo para que siempre sea mayor de 1000, sería una regla de validación. También por ejemplo que la fecha de inicio sea mayor que la fecha final.
Integridad referencial (Foreign key)
Sirve para indicar una clave externa (también llamada secundaria y foránea) sobre uno o más atributos. Los atributos marcados de esta forma sólo podrán contener valores que estén relacionados con la clave principal de la tabla que relacionan (llamada tabla principal). Dichos atributos sí podrán contener valores nulos.
Es decir si hay una tabla de alquileres en la que cada fila es un *alquiler*, existirá un atributo *cod_cliente* que indicará el *código del cliente* y que estará relacionado con una tabla de *clientes*, en la que dicho atributo es la clave principal. De hecho no se podrá incluir un código que no esté en la tabla clientes; eso es lo que prohíbe la integridad referencial.
Paso de un esquema E-R a un esquema relacional

Transformación de las entidades fuertes

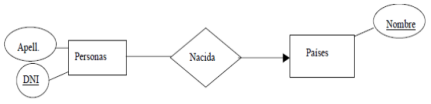
En principio las entidades fuertes del modelo Entidad Relación son transformados al modelo relacional siguiendo estas instrucciones:

- ✓ **Entidades.** Las entidades pasan a ser tablas
- ✓ **Atributos.** Los atributos pasan a ser columnas o atributos de la tabla.
- ✓ **Identificadores principales.** Pasan a ser claves primarias
- ✓ **Identificadores candidatos.** Pasan a ser claves candidatas.

Transformación de las relaciones en base a la cardinalidad

1. Si dos tablas tienen una relación entre ellas 1:1, entonces el campo clave de una de las tablas debe aparecer en la otra tabla.
2. Si dos tablas tienen una relación entre ellas 1:M, entonces el campo clave de la tabla (1) debe aparecer en la otra tabla (M).
3. Si dos tablas tienen una relación entre ellas M:M, entonces debe crearse una nueva tabla que contenga los campos clave de las dos tablas.

En ocasiones es posible combinar dos o más tablas en una sola. Generalmente se combinan por motivos de rendimiento. Por ejemplo, el ejemplo de personas nacidas en países:



La traducción de este esquema E-R al relacional sería:

Personas(DNI, Apell.)
Países(Nombre)
Nacida(DNI, Nombre)

Y se podrían combinar las dos primeras tablas en el nuevo esquema:

Personas(DNI,Apell, PaisNac)
Países(Nombre)

3. Diseño físico.

Su objetivo es definir el esquema físico de la base de datos de forma que se den todas las instrucciones para que un DBA pueda implementar la base de datos sin ninguna ambigüedad. Se considera el rendimiento como un aspecto que no se ha tratado en las etapas anteriores.
Herramienta: Modelo físico de datos. Se consideran todos los detalles de la implementación física: organización de archivos e índices para el SGBD considerado.
Resultado: Esquema físico de la base de datos.
Las restricciones de integridad proporcionan un medio de asegurar que las modicaciones hechas a la base de datos por los
usuarios autorizados no provoquen la pérdida de la consistencia de los datos. Protegen a la base de datos contra los daños accidentales (no
contra daños intencionados, de lo cual se ocupa la seguridad de las bases de datos). Los tipos de restricciones de integridad en una base de datos
se pueden resumir como sigue:

- Claves.
- Cardinalidad de la relación.
- Restricciones de los dominios.
- Integridad referencial.
- Participación total.
- Dependencias funcionales.
- Otras restricciones.

Normalización

Para resolver:
Redundancia. Se llama así a los datos que se repiten continua e innecesariamente por las tablas de las bases de datos. Cuando es excesiva es evidente que el diseño hay que revisarlo, es el primer síntoma de problemas y se detecta fácilmente.
Ambigüedades. Datos que no clarifican suficientemente el elemento al que representan. Los datos de cada fila podrían referirse a más de un ejemplar de esa tabla o incluso puede ser imposible saber a qué ejemplar exactamente se están refiriendo. Es un problema muy grave y difícil de detectar.
Pérdida de restricciones de integridad. Normalmente debido a dependencias funcionales. Más adelante se explica este problema. Se arreglan fácilmente siguiendo una serie de pasos concretos.
Anomalías en operaciones de modificación de datos. El hecho de que al insertar un solo elemento haya que repetir tuplas en una tabla para variar unos pocos datos. O que eliminar un elemento suponga eliminar varias tuplas necesariamente (por ejemplo que eliminar un cliente suponga borrar seis o siete filas de la tabla de clientes, sería un error muy grave y por lo tanto un diseño terrible).
Objetivos:
-Tener almacenado con el menor espacio posible
-Eliminar datos repetidos
-Eliminar errores lógicos
-Datos ordenados

<u>Primera Forma Normal</u>

Una tabla está en Primera Forma Normal si:

Todos los atributos son «atómicos». Por ejemplo, en el campo teléfono no tenemos varios teléfonos.
La tabla contiene una **clave primaria única**.
La clave primaria no contiene atributos nulos. No podemos tener filas para las que no haya clave

No debe existir variación en el número de columnas. Si algunas filas tienen 8 columnas y otras 3, pues no estamos en 1FN.

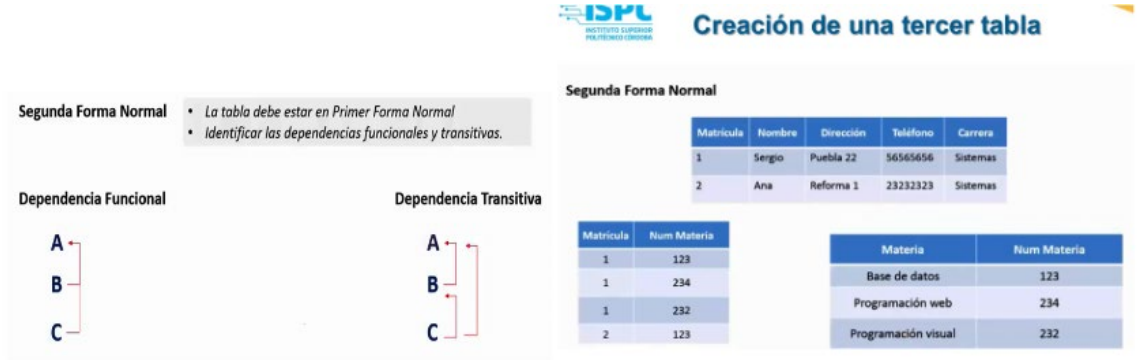
Los campos no clave deben identificarse por la clave. existe clave primaria.

Debe Existir una independencia del orden tanto de las filas como de las columnas, es decir, si los datos cambian de orden no deben cambiar sus significados.

Segunda Forma Normal

Una tabla está en 2FN si además de estar en 1FN cumple que los atributos no clave depende de TODA la clave principal.

●Por ejemplo, si tenemos una tabla con Personas, identificadas por su DNI y recogemos su empresa y dirección de trabajo, la clave sería CUIT-Empresa. Pero nos encontraremos con que una misma persona puede trabajar en varias empresas. Y vemos que la dirección de trabajo no depende de TODA la clave primaria, sino solo de la empresa. Por lo tanto, no estamos en 2FN



Tercera Forma Normal

Una tabla está en 3FN si además de estar en 2FN no existe ninguna dependencia transitiva entre los atributos que no son clave.

●

●Como dijo Bill Kent, «todo atributo no clave debe proporcionar información sobre la clave, sobre toda la clave y nada más que la clave… con la ayuda de Codd».

Identificar atributos no claves, no relacionados

Matrícula	Nombre	Dirección	Teléfono	No Carrera
1	Sergio	Puebla 22	56565656	1234
2	Ana	Reforma 1	23232323	1234

No Carrera	Carrera
1234	Sistemas
6789	Mecatrónica

Matrícula	Num Materia
1	123
1	234
1	232
2	123

Materia	Num Materia
Base de datos	123
Programación web	234
Programación visual	232
Base de datos	123

GIT

Acerca del Control de Versiones

RESUMEN

GitHub es un servidor remoto que permite subir repositorios de código para almacenarlo en el sistema de control de versiones Git.

¿Cómo funciona git?

1. Crear un "repositorio" (proyecto) con una herramienta de alojamiento de Git (por ejemplo, Git Hub)
2. Copia (o clona) el repositorio a tu máquina local
3. Añade un archivo a tu repositorio local y confirma ("commit") los cambios
4. Envía ("push") los cambios a la rama principal
5. Haz cambios en tu archivo con una herramienta de alojamiento de Git y confírmalos
6. Extrae ("pull") los cambios a tu máquina local
7. Crea una rama ("branch", versión), haz algún cambio y confírmalo
8. Abre una solicitud de incorporación de cambios ("pull request": propón cambios a la rama principal)
9. Fusiona ("merge") tu rama con la rama principal

GIT

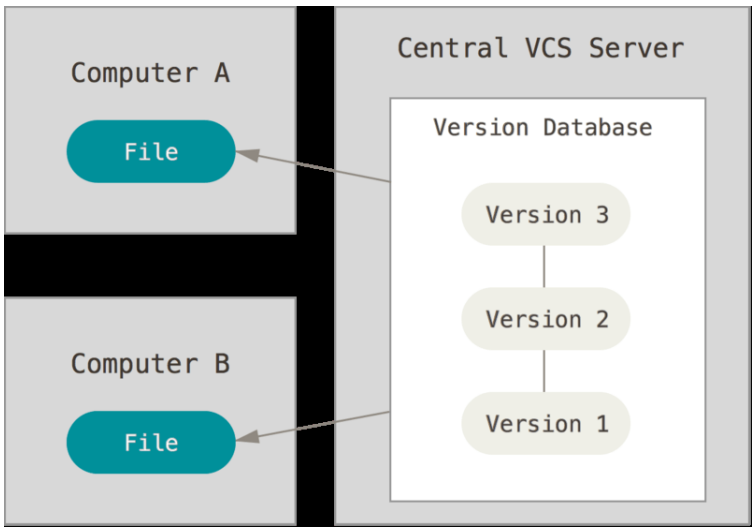
- Acerca del Control de Versiones -

¿Qué es un control de versiones? Es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Sistemas de Control de Versiones Centralizados

Ventajas de esta configuración:

- Todas las personas saben hasta cierto punto en que están trabajando los otros colaboradores del proyecto.
- Los administradores tienen control detallado sobre qué puede hacer cada usuario.
- Es más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.

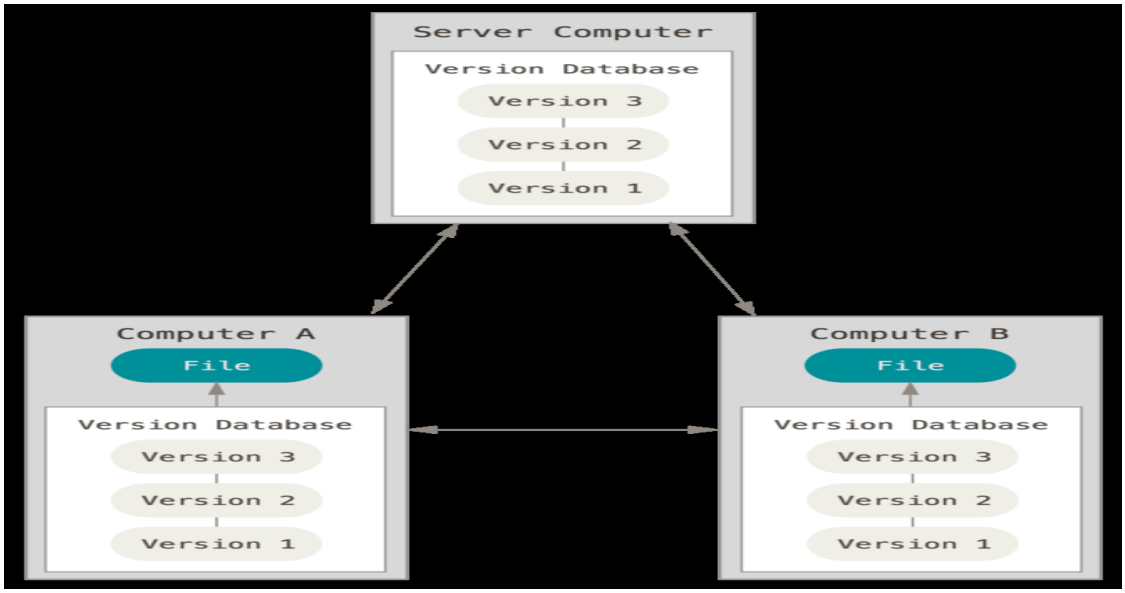


Desventajas.

- Si ese servidor se cae, entonces nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando.
- Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han realizado copias de seguridad adecuadamente, se perderá toda la información del proyecto, con excepción de las copias instantáneas que las personas tengan en sus máquinas locales.
- Los VCS locales sufren de este mismo problema: Cuando tienes toda la historia del proyecto en un mismo lugar, te arriesgas a perderlo todo.

Sistemas de Control de Versiones Distribuidos

Los sistemas de Control de Versiones Distribuidos (DVCS) ofrecen soluciones para los problemas que han sido mencionados. En un DVCS (como GIT), los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio. De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo. Cada clon es realmente una copia completa de todos los datos.



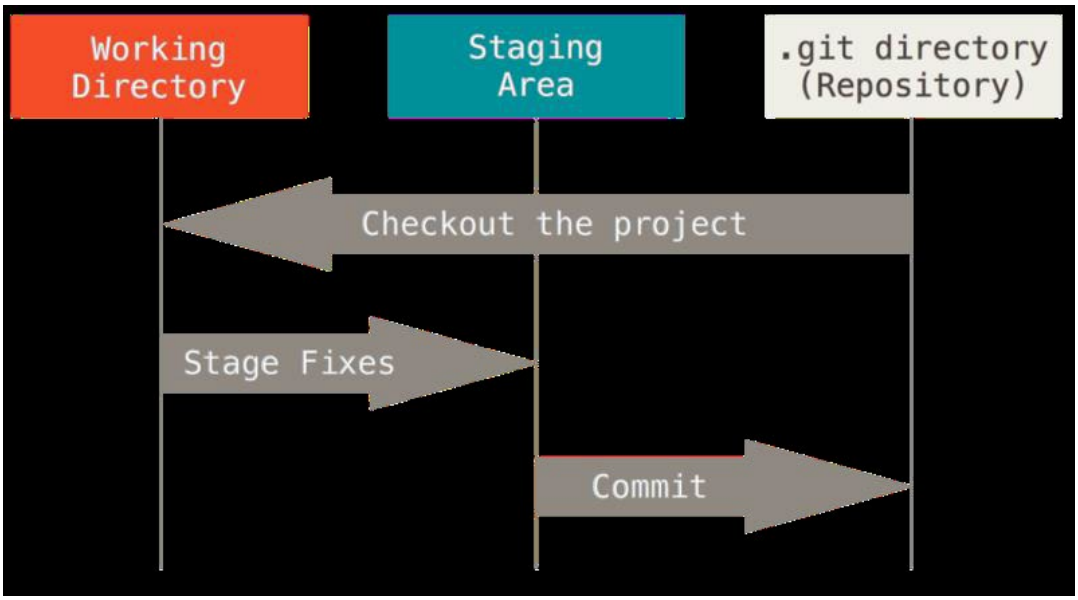
Los Tres Estados

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged).

Confirmado: significa que los datos están almacenados de manera segura en tu base de datos local.

Modificado: significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.
Preparado: significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto nos lleva a las tres secciones principales de un proyecto de Git: **El directorio de Git** (Git directory), **el directorio de trabajo** (working directory), y el **área de preparación** (staging area).



Etapas o Flujo de Trabajo

- LOCAL (Working Directory): etapa de trabajo en la computadora personal de cada participante, se puede modificar, crear o agregar o eliminar archivos.

  Se agregan con git add (selecciona los archivos que queramos para a stage)

- STAGE: etapa intermedia para verificar cuales son los cambios que nosotros efectuamos antes de pasar al repositorio. no se ve reflejado en el repositorio. Solo pasan los cambios que seleccionamos.

  Se agrega con git commit

- COMMIT: Los cambios que queremos subir al servidor remoto (GitHub) desde esta etapa lo realizamos usando el comando git push.

Configuración general de Git

CONFIGURAR NOMBRE

1) git config --global user.name +"NOMBRE PERSONAL"

CONFIGURAR E-MAIL

2) git config --globa user.email +dirrecion de email

CONFIGURAR EDITOR DE CODIGO

3) git config --global core.editor "code --wait" (espera hasta abrir el editor)

4) git config --global -e (abre el editor por defecto y una vez que se cierra, se asigna la configuración)

CONFIGURAR COMO GIT INTERACTUA CON LOS ARCHIVOS

5) git config --global core.autocrlf true (en windows) o git config --global core.autocrlf input (en mac o linux)

Comandos de navegación dentro de la terminal

- ls (lista todas las carpetas que hay donde estoy trabajando)
- pw (me ubica en donde me encuentro)
- cd + nombre la carpeta (me mueve a la carpeta que escribo)
- pwd (muestra mi nueva ubicación)
- cd .. (sale de la carpeta y me vuelve al lugar anterior)
- mkdir (crea una nueva carpeta) luego con cd ingreso a la nueva carpeta.
- git init (inicializa un repositorio vacío) si la dirección en consola aparece con .git al final, está oculto
- ls -a (hace visible el nuevo repositorio).
- cd .git (abre el nuevo repo)
- ls -a (muestra todos los archivo que utiliza git para poder gestionar nuestros proyectos)
- cd .. (me regresa a la carpeta donde estoy trabajando)

COMANDOS IMPORTANTES

Los siguientes comandos se ejecutan trabajando de forma local (GitBash)

- clear (comando que se utiliza para limpiar el Terminal / Git Bash. Simplemente colocan “clear” en la consola y todo lo que tienen allí desaparecerá, PERO continuarán en la carpeta/archivo en el que se encuentren al momento de ejecutar ese comando)

- git status (comando que muestra el estado del directorio de trabajo y del área del entorno de ensayo. Permite ver los cambios que se han preparado, los que no y los archivos en los que Git no va a realizar el seguimiento. El resultado del estado no muestra ninguna información relativa al historial del proyecto)

- git staus -s (solo muestra los archivos o cambios/modificaciones, pero ignora toda la informacion del contexto, presuponiendo que ya lo sabemos con anterioridad)

- git add (comando para añadir un cambio en el directorio de trabajo en el entorno de ensayo. De este modo, se indica a Git que quieres incluir actualizaciones en un archivo concreto en la próxima confirmación.

- git commit (comando para capturar una instantánea de los cambios preparados en ese momento del proyecto. Las instantáneas confirmadas pueden considerarse como versiones "seguras" de un proyecto: Git no las cambiará nunca a no ser que se lo pidas expresamente)

- git diff (comando que se ejecuta para establecer las diferencias en los orígenes de datos de Git. Dichos orígenes de datos pueden ser confirmaciones, ramas y archivos, entre otras posibilidades)

- git diff --stage (solo muestra los cambios que se encuentran en la etapa de stage, listo para commit)

- git log (comando que podemos usar para ver el historial de commits, estando situados en la carpeta de nuestro proyecto)

- git log --oneline (muestra un resumen del historial)

- git push (comando que se usa para cargar contenido del repositorio local a un repositorio remoto. El envío es la forma de transferir confirmaciones desde tu repositorio local a un repositorio remoto)

- git pull (comando para extraer y descargar contenido desde un repositorio remoto y actualizar al instante el repositorio local para reflejar ese contenido)

- git branch (comando que indica en que rama estamos trabajando, luego de crear una nueva rama, al ejecutarlo, nos confirma en que rama estamos)

- git checkout -b + rama"NOMBRE DEL USUARIO" (comando para crear rama)

- git checkout + "nombre de la rama" (comando para cambiar entre las diferentes ramas, por ejemplo: git checkout master, me lleva a la rama principal)

- git merge + "nombre de la rama individual" (agrega los cambios a repo general)

- git remote add origin + url del repositorio (comando para indicar a que servidor remoto se van a subir los cambios y desde donde se realizan esos cambios)

- git push -u origin master (comando para subir nuestros cambios desde nuestra rama)

ACLARACION: el origen de la terminal puede aparecer nombrado de diferentes maneras, como master, main, etc., prestar atención en este punto

- git push -u origin + "nombre de la rama" (comando para agregar rama al repo pero, sin merge)

SI QUIERO CLONAR UN REPOSITORIO REMOTO A LOCAL

- git clone + url del repo (comando para importar el repositorio a mi computadora, luego habría que crear la rama y todo lo demás)

scrum

Software

El software son los *programas*, los ***documentos*** *asociados* y la *configuración de datos* que se necesitan para hacer que estos programas operen de manera correcta.

La documentación se refiere a documentos que describen:
- la organización del sistema,
- documentos para el usuario que les explica cómo utilizar el sistema
- sitios web que permitan a los usuarios descargar la información de actualizaciones del producto

Caracteristicas del Software:
- Intangible
- Puede ser modificado
- Se diseña , planifica, programa y monitorea
-

Modelo de cascada

Proceso de desarrollo:

- Definición de requerimientos
- Diseño de software y del sistema
- Implementación y prueba de unidades
- Integración y Prueba del sistema
- Operación y Mantenimiento

Ciclo de vida:
El sistema de software pasa por un conjunto de fases que estan estandarizadas, que se involucran en todas sus facetas de desarrollo.

Metodologías ágiles

Sus valores:

- Valorar a los individuos por encima de los procesos.
- La colaboración del cliente tiene mas valor que la negociación contractual.

Los resultados esperados:

- Los estudiantes tienen Autonomía en su manera de aprender
- Mayor motivación en el aprendizaje
- El mundo camina hacia los trabajos en equipo.

Principios ágiles

- Colaboración estrecha con el cliente.
- Predisposición y respuesta al cambio.
- Desarrollo incremental con entregas frecuentes de funcionalidad.
- Comunicación verbal directa.
- Simplicidad, sólo los artefactos necesarios.

- Motivación, compromiso y responsabilidad del equipo por la autogestión, auto-organización.

SCRUM

Definición: Es un marco de trabajo a través del cual las personas pueden abordar problemas complejos adaptativos, a la vez que se entregan productos de forma eficiente y creativa con el máximo valor.

Es un enfoque ágil para la gestión de un proyecto es un Marco de Trabajo.
- Utiliza procesos iterativos/incrementales.
- Orientado a resultados y compromisos.
- No está restringido a proyectos de software solamente.
- Su visión es opuesta a la propuesta por la metodología en cascada.

Scrum y sus pilares:

- Transparencia: Aspectos significativos del proceso deben ser visibles y Definidos en base a un estándar común.

- Inspección: Deben inspeccionar frecuentemente los Artefactos de Scrum y el progreso hacia un objetivo para detectar variaciones indeseadas

- Adaptación: Si se determina que uno o más aspectos de un proceso se desvían de los límites aceptables y que el producto resultante será inaceptable, el proceso o el material que está siendo procesado deben ajustarse. Dicho ajuste deberá realizarse cuanto antes para minimizar desviaciones mayores.

SPRINT

El Sprint es un *período de corta duración* que debe finalizar con un prototipo operativo o producto parcialmente entregable. El mismo se repite veces a lo largo del proyecto y permite hacer entregas de producto en partes, donde cada entrega, es un incremento de funcionalidad respecto al anterior.
Esto difiere del conocido ciclo de vida en cascada muy utilizado en el campo de desarrollo de software en que, las fases del ciclo de vida (requisitos, análisis, diseño, codificación, testing, etc.) se realizan una única vez y, el inicio de cada fase no comienza hasta que termina la que precede.

Durante el Sprint:

- No se realizan cambios que puedan afectar al objetivo del Sprint (Sprint Goal);
- Los objetivos de calidad no disminuyen;
- El alcance puede clarificarse y renegociarse entre el Propietario del Producto (Product Owner) y el Equipo de Desarrollo a medida que se va aprendiendo más.

El proyecto y sus **Ceremonias:**

- **Sprint Planning Meeting** (reunión de planificación del sprint). Se produce al iniciar cada Sprint y tiene por objetivo decidir que se va a realizar en el Sprint.

- **Daily Scrum Meeting** (reunión periódica). Se produce diariamente, y tiene un máximo de 20 min. de duración. Tiene por objeto tratar qué es lo que se hizo, qué se va a hacer y qué problemas se han encontrado, esto a fines de encontrar soluciones en la diaria.

- **Sprint Review Meeting** (reunión de revisión del Sprint). Se produce al finalizar el Sprint y tiene por objeto mostrar qué es lo que se ha completado y qué no. Debe estar presente el Product Owner.

- **Sprint Retrospective Meeting** (reunión de retrospectiva del Sprint). Se produce también al finalizar el Sprint y tiene por objeto documentar qué ha funcionado y qué no ha funcionado

en el Sprint. La idea de dicha reunión es centrar al equipo en lo que salió bien y en lo que debe mejorar para la próxima iteración. De ninguna manera se centra en lo que salió mal.

Logros a conseguir:

Priorizar qué temas se van a trabajar primero, en qué orden y porqué. Cada una de las personas que componen el equipo deberá comprender, escuchar, respetar y sentir curiosidad, buscándose la responsabilidad conjunta y el talento colectivo, asumiendo que los resultados son el producto del trabajo en equipo. Los equipos se autogestionan. Cada integrante es el líder de su propio aprendizaje.
Todos los miembros del equipo pueden probar sus ideas en lugar de esperar a que sea otro compañero el que las tome por él. Son los integrantes quienes toman sus propias decisiones autodefiniendo como van a alcanzar los objetivos.

Scrum diario:

- ¿Qué hiciste ayer?
- ¿Qué vas a hacer hoy?
- ¿Hay algo que te impida hacer tu trabajo?

Sprint Review

- El equipo presenta la funcionalidad terminada al Product Owner y demás stakeholders.
- Los miembros del equipo responden preguntas de los stakeholders en relación a la demostración, toman nota de los cambios propuestos.
- Al finalizar la presentación, los stakeholders dan su impresión acerca del producto, cambios deseados y prioridad de esos cambios.

Sprint Retrospective

- Provee una visión de qué está funcionando y qué no está funcionando.
- Se realiza al finalizar el sprint.
- Participa todo el equipo.