DESARROLLO

WEB FULL STACK

-INTERMEDIO

Autor de contenido

Andrés Fernando Pineda Guerra

EMPEZAR

Tabla de Contenido

Presentación

En el curso de desarrollador Full Stack como componente intermedio, podrán adquirir las habilidades y lenguajes necesarios para el desarrollo web, enfocándose en sus grandes pilares, como lo son Front End, Back End, Diseño y modelamiento de aplicaciones y documentación de código.
El curso trata temas emergentes tales como, la seguridad informática, desarrollo de aplicaciones móviles, gestión de base de datos, todo esto basado en la metodología Scrum. De la misma manera, se hace énfasis en el manejo de proyectos tanto en los módulos de desarrollo como los módulos de gestión de proyectos de TI.

Objetivos del curso (competencias)

Objetivo general

Formar a los participantes en el desarrollo web en todo el ciclo de vida del software, en donde adquieran los conocimientos básicos para implementar soluciones web.

Objetivo específico

- Conocer los conceptos y teoría básica del desarrollo web.

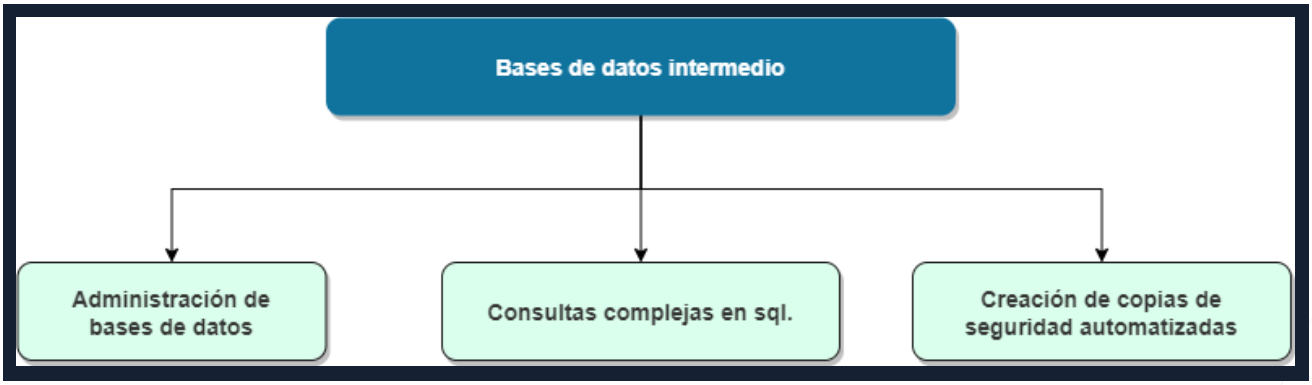
- Identificar y conocer los diferentes lenguajes de programación y herramientas para el desarrollo web.

- Aplicar las diferentes tecnologías web, tendencias y herramientas en el desarrollo de soluciones web enfocadas a proyectos.

- Diseñar, desarrollar e implementar soluciones web básicas en donde se integren los componentes de Front End, Back End, seguridad, redes y buenas prácticas utilizando metodologías ágiles.

- Identificar y conocer los conceptos básicos para el desarrollo móvil, así como aplicar su desarrollo en aplicaciones básicas.

Mapa de contenido de la unidad



Módulo 7

Bases de datos intermedio

Ideas clave

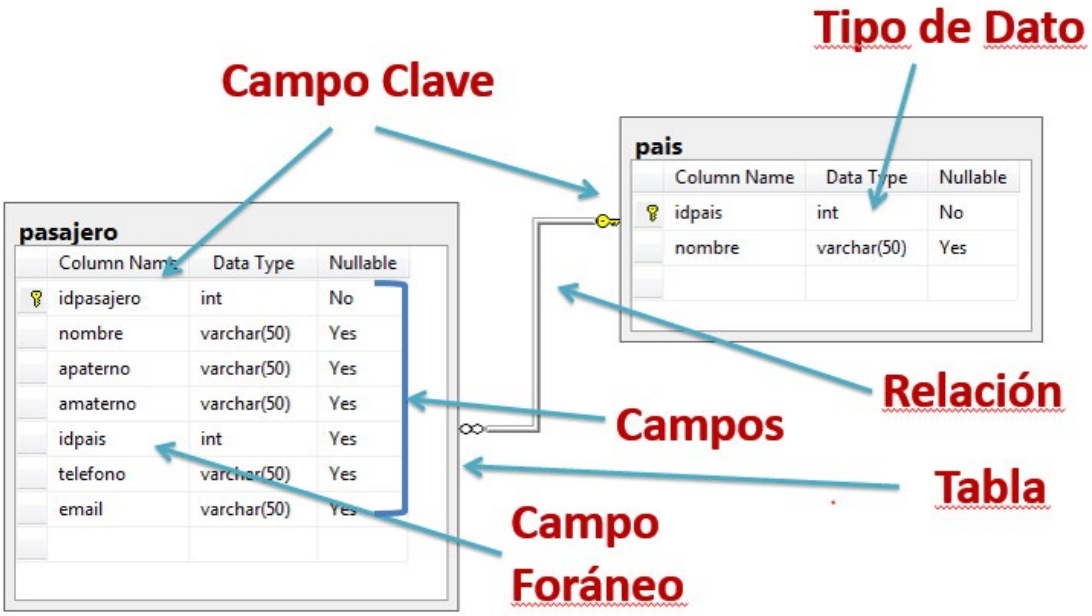
Conceptos básicos de bases de datos, lenguaje SQL, tipos de bases de datos, estructuras de datos, tipos de datos, comandos de manipulación de datos y comandos de definición de datos.

Consultas avanzadas de SQL y automatización copias de seguridad de bases de datos.

7.1. Administración de bases de datos

Estructura de una base de datos.

Elementos de una base de datos



Tipos de datos

Datos numéricos

TINYINT: Define un valor entero pequeño, cuyo rango es de -128 a 127. El tipo sin signo va de 0 a 255.

SMALLINT: Es un tipo de datos entero que ocupa 2 Bytes y puede guardar números de -215 a +215. (aproximadamente en el rango de -32768 a +32768).

MEDIUMINT: Número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607.

INT o INTEGER: El tipo de datos int es el principal tipo de datos de valores enteros de SQL Server. su rango es de -2000000000 a 2000000000 aprox.

BIGINT: Número entero De –9000000000000000000 hasta 9000000000000000000 aprox.

Datos para guardar cadenas de caracteres (alfanuméricos)

CHAR [(n)]: Datos de cadena de tamaño fijo. n define el tamaño de la cadena en bytes y debe ser un valor entre 1 y 8000. Para juegos de caracteres de codificación de byte único, como el Latin , el tamaño de almacenamiento es n bytes y el número de caracteres que se pueden almacenar también es n.

VARCHAR: El tipo de datos VARCHAR de SQL Server se utiliza para almacenar datos de tipo cadena no Unicode de longitud variable.

BINARY: Datos binarios de longitud fija con una longitud de n bytes, donde n es un valor entre 1 y 8.000. El tamaño de almacenamiento es de n bytes.

VARBINARY [(n | max)]: Datos binarios de longitud variable. n puede ser un valor de 1 a 8000. max indica que el tamaño máximo de almacenamiento es de 2^31-1 bytes. El tamaño de almacenamiento es la longitud real de los datos especificados + 2 bytes.

TINYBLOB: Puede almacenar un volumen variable de datos longitud máxima de 255 caracteres no diferencia entre mayúsculas y minúsculas.

TINYTEXT: Puede almacenar un volumen variable de datos longitud máxima de 255 caracteres y diferencia mayúsculas y minúsculas.

BLOB: El tipo de datos de gran objeto binario (BLOB) es un tipo de datos de MySQL que puede almacenar datos binarios como los de archivos de imagen, multimedia y PDF.

TEXT: Guarda datos binarios de longitud variable, puede contener hasta 2000000000 caracteres. No admite argumento para especificar su longitud.

MEDIUMBLOB: Longitud de 16777215 caracteres.

MEDIUMTEXT: Longitud de 16777215 caracteres.

LONGBLOB: Longitud para 4294967295 caracteres.

LONGTEX: Longitud para 4294967295 caracteres.

ENUM: El tipo de dato enum en mysql representa una enumeración. Puede tener un máximo de 65535 valores distintos y es una cadena cuyo valor se elige de una lista numerada de valores permitidos que se especifica al definir el campo y puede ser una cadena vacía e incluso null.

SET: Puede usar la instrucción SET para asignar a una variable declarada un valor distinto de NULL. La instrucción SET que asigna un valor a la variable devuelve un solo valor.

Datos para almacenar fechas y horas

DATE: Es el tipo de datos de fecha más simple, permite registrar únicamente el día (año, mes, día) sin horas.

DATETIME: DateTime es una función de texto que devuelve la fecha y la hora en que un informe se rellenará con datos.

TIME: Es un tipo de datos en SQL Server que solo guarda la hora, si el día (año, mes y día). Guarda hora, minuto, segundo y fracciones de segundo de hasta 7 dígitos de precisión, es decir, el rango de horas que maneja son de 00:00:00.0000000 a las 23:59:59.9999999.

TIMESTAMP: Tipo de datos contiene una FECHA y una HORA en años, meses, días, horas, minutos, segundos y fracciones de un segundo. El formato de un TIMESTAMP literal es la palabra TIMESTAMP seguido de un espacio, seguido de una indicación de fecha y hora entre comillas simples en el formato 'aaaa-MM-dd HH:mm:ss.

YEAR: Devuelve un entero que representa el año de la fecha especificada. Esta función es equivalente a DATEPART(yy, fecha).

Atributos de los campos

NULL: Se utiliza un valor NULL en una base de datos relacional cuando el valor de una columna es desconocido o falta. Un valor NULL no es una cadena vacía (para tipos de datos de caracteres o de fecha y hora) ni un valor cero (para tipos de datos numéricos). La especificación ANSI SQL-92 indica que un valor NULL debe ser el mismo para todos los tipos de datos, de modo que todos los valores NULL se trate de manera uniforme.

DEFAULT: Un valor por default -o por omisión- es aquel valor que se le dará a un campo si al momento de insertar su registro no se le da un valor específico.

BINARY: Datos binarios de longitud fija con una longitud de n bytes, donde n es un valor entre 1 y 8.000. El tamaño de almacenamiento es de n bytes.

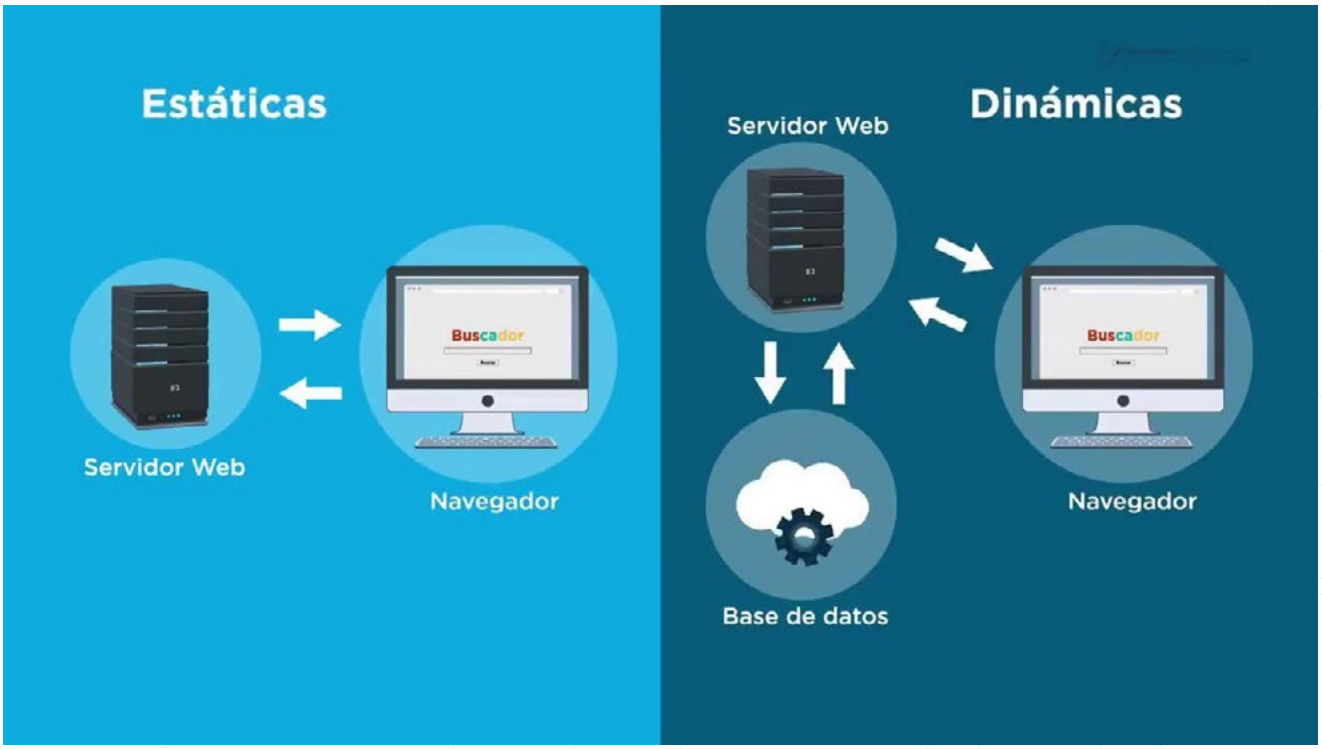
PRIMARY KEY: Una llave primaria o PRIMARY KEY es una columna o un grupo de columnas que identifica de forma exclusiva cada fila de una tabla.Se puede crear una llave primaria para una tabla utilizando la restricción PRIMARY KEY.

AUTO_INCREMENT: Auto Increment es una función que opera en tipos de datos numéricos. Genera automáticamente valores numéricos secuenciales cada vez que se inserta un registro en una tabla para un campo definido como incremento automático.

UNIQUE: UNIQUE es una restricción en SQL se utiliza para garantizar que no se inserten valores duplicados en una columna específica o combinación de columnas que participen en la restricción UNIQUE y no formen parte de la CLAVE PRIMARIA.

FULLTEXT: Es un tipo de índice que permite ejecutar búsquedas optimizadas de colecciones de palabras contenidas en el texto de una columna, este tipo de índice se aplica a columnas de tipo CHAR, VARCHAR y TEXT (Gabillaud, 2013).

Tipos de bases de datos MySQL.



Base de datos estática: Es de sólo lectura. Es decir, es la base que no permite que sus datos sean modificados o eliminados una vez que son almacenados.

Base de datos dinámica: Este tipo de bases de datos permite que sus datos sean modificados o eliminados una vez que son almacenados.

Base de datos relacional: Es una forma de estructurar información en tablas, filas y columnas. Un RDB tiene la capacidad de establecer vínculos (o relaciones) entre infor-

mación mediante la unión de tablas, lo que facilita la comprensión y la obtención de estadísticas sobre la relación entre varios datos.

Base de datos bibliográfica: Es la que guarda metadatos bibliográficos de materiales como libros, artículos, periódicos, revistas, tesis, patentes y conferencias. Es decir, son datos que brindan información bibliográfica, pero no el contenido en sí (la información completa). Ejemplo de ello sería nombres de autores, títulos, resúmenes, índices o palabras claves. Estas bases de datos pueden ser en formato físico o digital.

Base de datos de texto completo: Es la que almacena el contenido completo de libros, artículos, diarios, disertaciones, periódicos, revistas, tesis y otros documentos escritos.

Base de datos orientada a objetos: Es aquella base en la cual los datos son representados como objetos.

Base de datos jerárquica: Es la que almacena información de forma jerárquica. Es decir, es la colección que comprende una arquitectura del tipo arbórea, en la cual un nodo se ramifica en otros nodos, y estos se dividen en otros.

Base de datos de red: Es aquella en la que los datos contenidos están conectados entre sí a través de ciertos enlaces.

Base de datos transaccional: Es la que permite llevar a cabo transacciones cortas de carácter digital a gran velocidad.

Base de datos multidimensional: Al igual que la base de datos relacional, este tipo de base almacena la información utilizando tablas, la diferencia se encuentra en su forma estructural, que es organizada en una especie de cubo de información.

Base de datos deductiva: Es el banco que permite establecer conclusiones partiendo de ciertas premisas.

Base de datos documental: Es la base que ordena la información de forma semiestructurada utilizando archivos como JSON y XML.

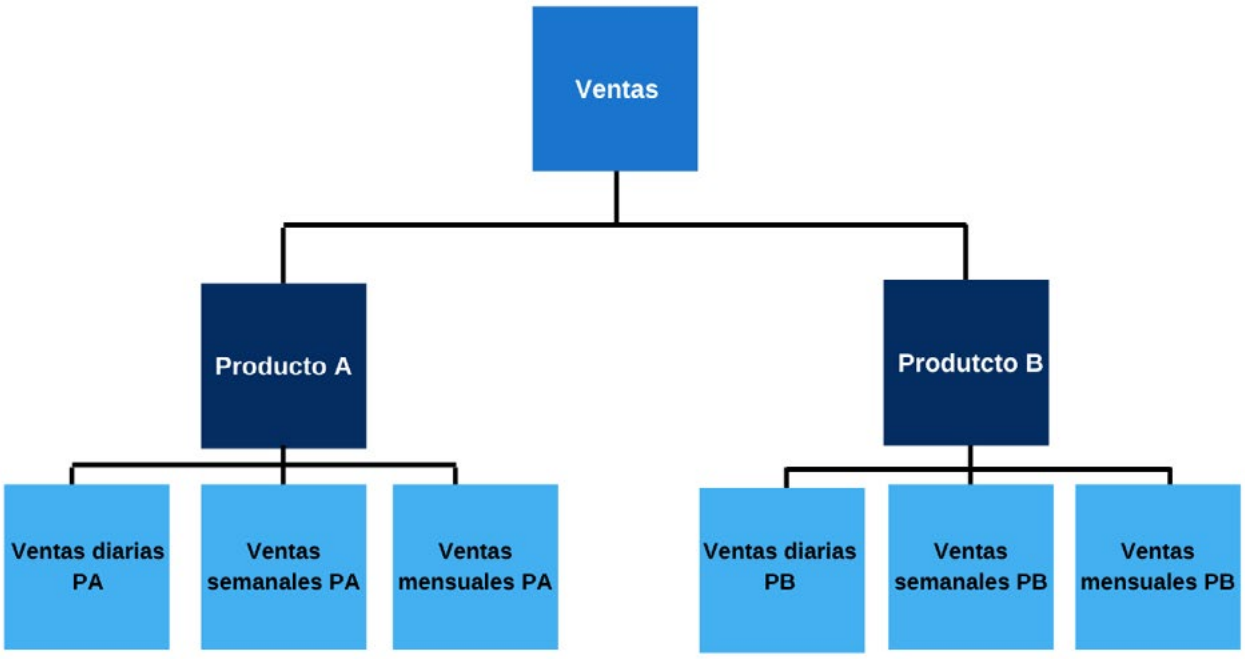
Modelos de bases de datos.

Los modelos son las estructuras lógicas que se adoptan en las bases de datos, incluyendo las limitaciones que determinan cómo se almacenan y organizan la forma de acceder a los datos.

Modelo jerárquico

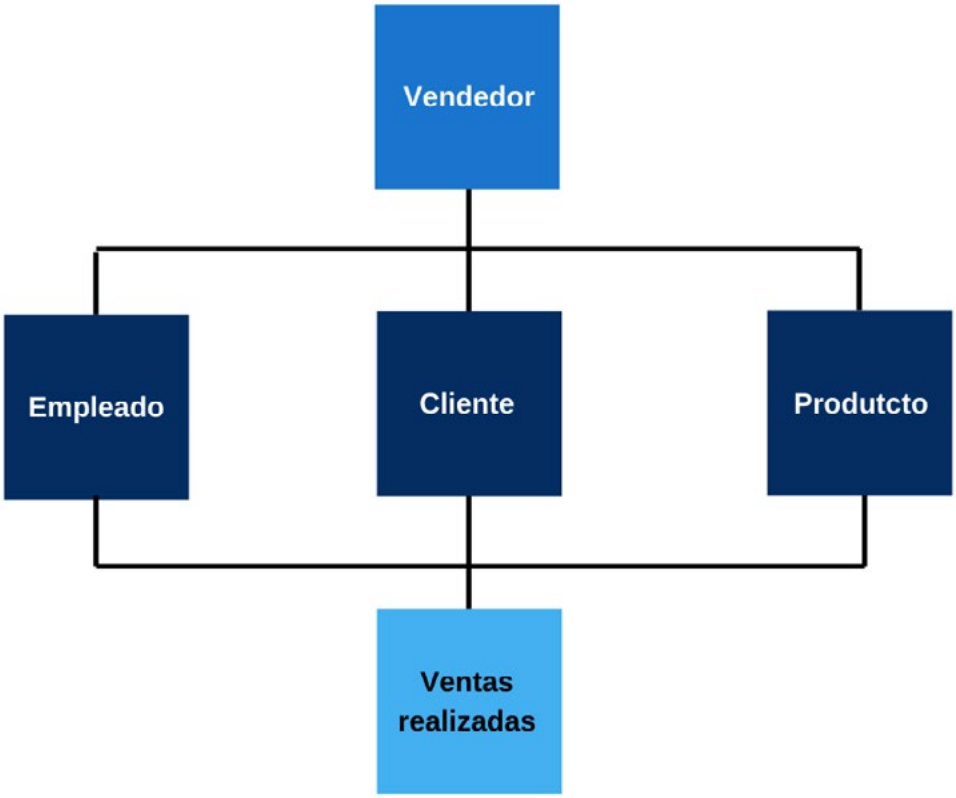
Si vamos a emplear una base datos jerárquica, el modelo de datos que emplearemos será el jerárquico, la cual se caracteriza por presentar los datos en una estructura de árbol invertido, donde cada registro tiene un único nodo raíz, del que surgen otros nodos (registros); los nodos en un mismo nivel son nodos padre, cada nodo padre tiene el mismo nodo raíz, y puede tener nodos hijos, pero los nodos hijos solo pueden tener un nodo padre. Este modelo se emplea poco actualmente.

En este modelo, los registros de un mismo nivel se clasifican en un orden específico.



Modelo de red

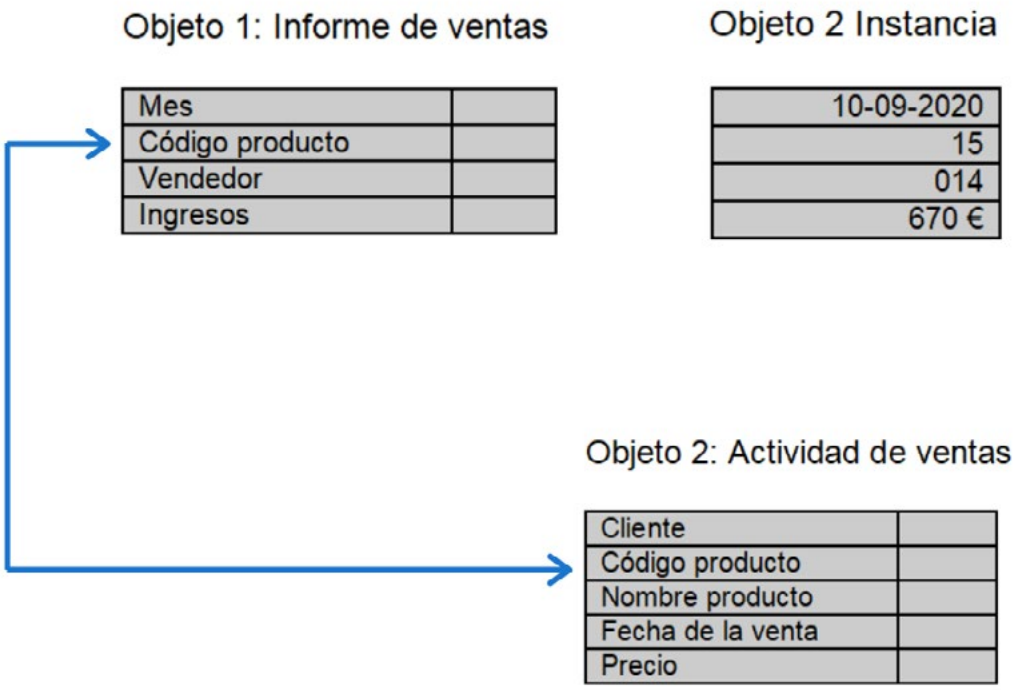
El modelo en red de base de datos parte del modelo jerárquico, pero aquí se permiten las relaciones de uno a muchos o de muchos a muchos entre registros vinculados, teniendo registros principales múltiples. El modelo se crea a través de conjuntos de registros relacionados; cada uno de estos conjuntos consiste en un registro propietario o principal y uno o más registros miembros o secundarios. Además, un registro puede ser miembro o secundario en diferentes conjuntos. Es decir, que en este modelo se permite que los nodos hijos tengan más de uno nodo padre, de manera que se pueden representar relaciones más complejas.



Modelo orientado a objetos

El modelo de la base de datos orientada a objetos define la base de datos como una colección de objetos utilizados en la programación orientada a objetos es decir, que emplear lenguajes como C++ o Java, o por el estilo. Este modelo de base de datos utiliza tablas también, pero no solo se limita a ellas y permite almacenar información muy detallada sobre cada objeto.

Los objetos se dotan de un conjunto de características propias, que a su vez les diferencian de objetos similares. Los objetos similares pueden agruparse en una clase y cada objeto de esta es una instancia. Las clases intercambian datos entre sí a través métodos (mensajes).

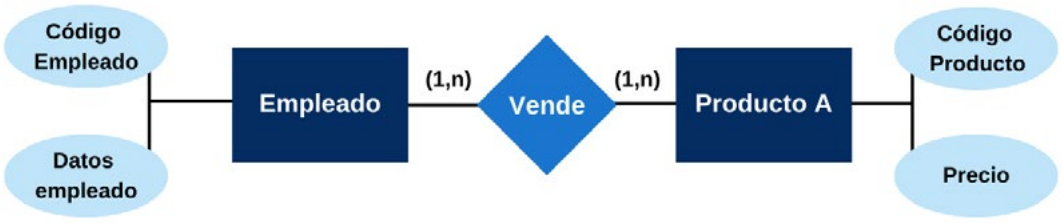


Modelo entidad-relación

El modelo entidad-relación se realiza previo a un modelo de bases datos relacional, ya que se trata de un diagrama elaborado a través de unos elementos básicos y su relación entre ellos:

- Entidades (son los objetos que se representan en la base de datos).
- Atributos (son el contenido de la entidad, sus características). A los atributos se les asigna un clave para distinguirlos de los demás registros.
- Relación (el vínculo que define la dependencia entre varias entidades).
- Cardinalidad (es la participación entre entidades, que pueden ser uno a uno, uno a varios o varios a varios).

En el diagrama las entidades se representan con un rectángulo, las relaciones con un rombo y los atributos con un óvalo. Lo vemos en un ejemplo:



Modelo plano

El modelo de bases de datos plano, los datos se estructura en dos dimensiones, en la que todos los objetos en una columna concreta tienen valores del mismo tipo y todos los objetos de la misma fila están relacionados entre ellos.

Usuario	Contraseña
0001	xxxx
0002	yyyy
0003	zzzz
0004	vvvv

7.2. Consultas complejas en sql

SELECT, DISTINCT, ORDER BY

Devolver todos los campos de una tabla (SELECT *)

```
1    SELECT *
2    FROM CLIENTES
```

Con el * indicamos que queremos devolver todos los campos. Si CLIENTES dispone de los campos idCliente, nombre y descripcion, lo anterior sería equivalente a:
Obviamente, al querer todos los campos, esto es innecesario y es por tanto más conveniente emplear el asterisco (*). También sería equivalente emplear la notación completa:

```
5    SELECT idCliente, nombre, descripcion
6    FROM CLIENTES
```

Al tener únicamente una tabla involucrada, podemos referirnos a los campos sin calificar, dado que no hay duda de a qué tabla se refiere. Cuando veamos consultas sobre varias tablas comprenderemos la necesidad de incluir esta notación calificada (TABLA.campo).
Devolver un subconjunto de los campos de una tabla (SELECT DISTINCT)

```
14   SELECT cp, ciudad
15   FROM DIRECCION
```

Esta consulta devolverá únicamente los campos cp (código postal) y ciudad de la tabla DIRECCION. Al tener un subconjunto de los campos, éstos no tienen por qué incluir a la clave de la tabla, por lo que no tienen por qué ser únicos. Así, si tenemos muchos registros referidos a distintas calles y números de ese mismo código postal y ciudad, nos encontraremos muchos registros repetidos. Esto puede evitarse haciendo:

```
18    SELECT DISTINCT cp, ciudad
19    FROM CLIENTES
```

Así se eliminan los registros repetidos, devolviendo únicamente una vez cada par cp, ciudad.
Esta selección de un subconjunto de los datos de la tabla y excluyendo repetidos se denomina en álgebra relacional proyección.

Ordenar según criterios (ORDER BY)

Permite ordenar los registros devueltos por una consulta por el campo o campos que estimemos oportunos:

```
1    SELECT *
2    FROM CIUDAD
3    ORDER BY provincia ASC, numhabitantes DESC
```

Esta consulta devolverá todas las ciudades ordenadas por provincia en orden ascendente, y dentro de los de la misma provincia ordenaría las ciudades por orden descendente del número de habitantes. Si no indicamos ASC ni DESC, el comportamiento por defecto será el orden ascendente (ASC).

WHERE

```
1    SELECT numero, calle
2    FROM DIRECCION
3    WHERE ciudad = 'Bogotá'
```

Esta consulta devolvería el número y la dirección de todas las direcciones pertenecientes a la ciudad de Bogotá. Como vemos, con WHERE indicamos la condición que deben cumplir los registros de la tabla para ser devueltos en la consulta. En este caso tenemos una condición simple dada por la comparación de igualdad (=) entre al campo (ciudad) y un literal de tipo cadena, entre comillas simples ('Bogotá').

```
1   SELECT calle, ciudad
2   FROM DIRECCION
3   WHERE numero = 12
```

Esta otra consulta devolvería la calle y ciudad de todos los registros de la tabla con el número 12, en este caso un literal numérico. Las condiciones empleadas pueden ser mucho más complejas incluyendo otro tipo de operadores y combinaciones de los mismos.

Operadores relacionales

Al margen del signo de igualdad empleado anteriormente, se pueden usar n las condiciones simples de las consultas los operadores relacionales habituales, devolviendo siempre un valor booleano (lógico):

Operador	Símbolo
igual	==
distinto	<>
menor que	<
menor que o igual	<=
mayor que	>
mayor o igual	>=

Ejemplo:

```
1   SELECT nombre
2   FROM CLIENTES
3   WHERE edad <= 32
```

AND, OR, NOT

Operador	Símbolo
AND	Y lógico
OR	O lógico
NOT	Negación lógica

La precedencia y asociatividad es la habitual definida en la Lógica. En cualquier caso, cuando incluya expresiones que emplean varios de estos operadores es recomendable usar paréntesis para evitar errores. Por ejemplo:

```
SELECT *
FROM DIRECCION
WHERE ciudad = 'Bogotá' AND cp = 11014141009 OR ciudad = 'Medellin' AND NOT cp = 050005
```

Devuelve los registros pertenecientes a direcciones que tengan el código postal 110141 de Bogotá o bien que no tengan el 050005 de Medellin. La mayor precedencia la adopta el operador NOT sobre la condición cp = 050005; a continuación los AND se aplican sobre ciudad = 'Bogotá' AND cp =110141 y ciudad = 'Medellin' AND NOT cp = 050005; por último se aplica el OR sobre la fórmula completa. La misma consulta se puede expresar de forma más clara con paréntesis:

```
SELECT *
FROM DIRECCION
WHERE (ciudad = 'Bogotá' AND cp = 110141) OR
(ciudad = 'Medellin' AND (NOT cp = 050005))
```

O bien si el NOT nos parece más evidente, podemos excluir el paréntesis interior, a nuestra gusto siempre conservando el significado que queríamos dar la operación.

IN

Se utiliza para comparar con una lista de valores fijados de modo que devuelva True si el campo indicado pertenece a la lista.

```
SELECT num, calle, direccion
FROM DIRECCION
WHERE ciudad IN ('Bogota', 'Medellin', 'Cali', 'Santa Marta')
```

BETWEEN

Es utilizado para indicar un intervalo de valores, es utilizado frecuentemente con valores numéricos o fechas.

```
1   SELECT nombre
2   FROM CLIENTES
3   WHERE edad BETWEEN 20 AND 35
```

LIKE

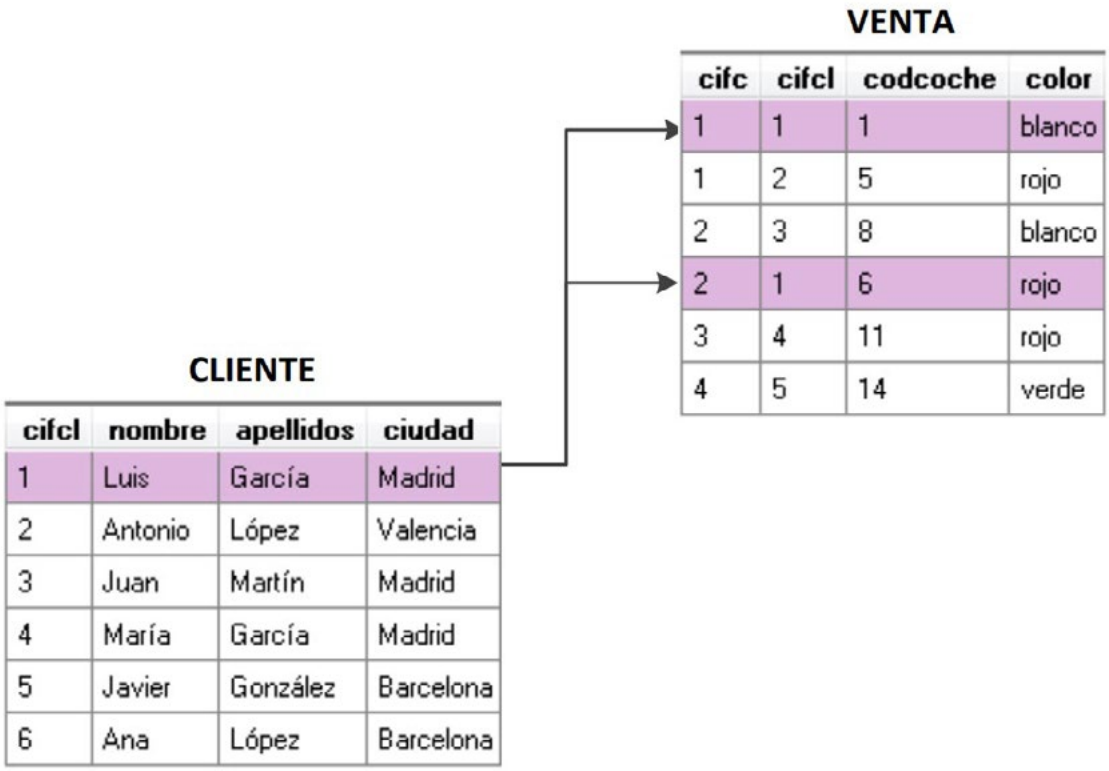
Se utiliza para comparar patrones de texto pudiendo incluir comodines como los siguientes:

Descripción	Símbolo
Sustituto para cero o más caracteres.	%
Sustituto para exactamente un carácter	_
Cualquier carácter de la lista	[lista caracteres]
Cualquier carácter que no esté en la lista	[^lista caracteres]
Cualquier carácter que no esté en la lista	[!lista caracteres]

```
1   SELECT num, calle, cp
2   FROM DIRECCION
3   WHERE ciudad LIKE 'Val%'
```

INNER JOIN, LEFT JOIN, RIGHT JOIN

Podemos generar una consulta que obtenga datos de varias tablas, pudiendo establecer a su vez criterios sobre otras. Por ejemplo, dados los clientes de nuestro concesionario y nuestras ventas de coches a clientes en los diversos concesionarios.



VENTA

cifc	cifcl	codcoche	color
1	1	1	blanco
1	2	5	rojo
2	3	8	blanco
2	1	6	rojo
3	4	11	rojo
4	5	14	verde

CLIENTE

cifcl	nombre	apellidos	ciudad
1	Luis	García	Madrid
2	Antonio	López	Valencia
3	Juan	Martín	Madrid
4	María	García	Madrid
5	Javier	González	Barcelona
6	Ana	López	Barcelona

podemos obtener una consulta que obtenga datos del cliente y de la venta. Por ejemplo, nombre, apellidos, codcoche y color, mediante una consulta del tipo:

```
SELECT nombre, apellidos, codcoche, color
FROM CLIENTE, VENTA
WHERE CLIENTE.cifcl = VENTA.idCliente
```

INNER JOIN implícito

Por ejemplo, para obtener los datos del cliente y el pedido en la misma consulta:

```
SELECT nombreCliente, idPedido, fechaPedido
FROM CLIENTE, PEDIDO
WHERE CLIENTE.idCliente = PEDIDOpedido.idCliente
```

Como vemos, podemos empezar escribiendo tal cuál qué datos nos piden (SELECT), de dónde podemos obtenerlos (FROM) y qué criterio (WHERE). Esta es la versión más an-

tigua de SQL, aunque se sigue empleando, conociéndose como JOIN implícito ya que no se usa por ningún lado la palabra JOIN, pero se está haciendo la intersección por la foreign key, en este caso la columna idCliente.

Si se omite en el caso anterior el WHERE:

```
1   SELECT nombreCliente, idPedido, fechaPedido
2   FROM CLIENTE, PEDIDO
```

Obtendremos los datos solicitados para todos los pares de registros de CLIENTE y PEDIDO, el producto cartesiano, de forma que si hay 3 registros en CLIENTE y 4 en PEDIDO devolveremos 12 registros. Este tipo de join se conoce como CROSS JOIN, y en este caso se ha hecho de forma implícita.

LEFT JOIN

El resultado de esta operación siempre contiene todos los registros de la relación izquierda (primera tabla que indicamos), y aquellos de la tabla derecha que cumplen la condición establecida. Para el resto aparecerá en los campos correspondientes a dicha tabla un NULL.

```
1   SELECT nombreCliente, idPedido, fechaPedido
2   FROM CLIENTE LEFT JOIN PEDIDO
3   ON cliente.idCliente = pedido.idCliente
```

Esta consulta devolverá todos los clientes con sus pedidos, y un registro por cada cliente que no tenga pedidos.

RIGHT JOIN

El RIGHT JOIN es análogo al LEFT JOIN, pero devolviendo todos los registros de la relación derecha (segunda tabla que aparece), y únicamente aquellos de la tabla izquierda que cumplen la condición del JOIN. El resultado de esta operación siempre contiene todos los registros de la relación derecha (segunda tabla que indicamos), de modo que en aquellos sin equivalente en la parte izquierda tendrán en los campos correspondientes a dicha tabla un NULL.

```
1   SELECT nombreCliente, idPedido, fechaPedido
2   FROM CLIENTE RIGHT JOIN PEDIDO
3   ON cliente.idCliente = pedido.idCliente
```

Asumiendo que pudiéramos tener en la base de datos pedidos sin cliente asociado, esta consulta devolverá todos los pedidos con sus clientes, y en caso de que el cliente no aparezca el nombreCliente sería NULL.

COUNT, AVG, SUM, MAX, MIN

COUNT

Estas funciones se denominan “funciones de agrupamiento” porque operan sobre conjuntos de registros, no con datos individuales.

Tenga en cuenta que no debe haber espacio entre el nombre de la función y el paréntesis, porque puede confundirse con una referencia a una tabla o campo. Las siguientes sentencias son distintas:

```
1   SELECT count(*) FROM libros;
2   SELECT count (*) FROM libros;
```

AVG

La función avg() retorna el valor promedio de los valores del campo especificado. Por ejemplo, queremos saber el promedio del precio de los libros referentes a “PHP”:

```
1   SELECT avg(precio) FROM libros
2    WHERE titulo LIKE '%PHP%';
```

SUM

La función “sum()” retorna la suma de los valores que contiene el campo especificado. Por ejemplo, queremos saber la cantidad de libros que tenemos disponibles para la venta:

```
1   SELECT sum(cantidad) FROM libros;
```

Esta función permite combinarse con la sentencia"where". Por ejemplo, queremos saber cuántos libros tenemos de la editorial "Planeta":

```
1   SELECT sum(cantidad) FROM libros
2    WHERE editorial ='Planeta';
```

MAX - MIN

Es utilizada para averiguar el valor máximo o mínimo de un campo usamos las funciones "max()" y "min()" respectivamente. Ejemplo, queremos saber cuál es el mayor precio de todos los libros:

```
1   SELECT max(precio) FROM libros;
```

```
2   SELECT min(precio) FROM libros
3    WHERE autor LIKE '%Rowling%';
```

GROUP BY

Las consultas anteriores recuperaban, trabajaban con, y mostraban información a nivel de cada registro individual de la base de datos. Así, si tenemos un producto con un determinado precio, podemos devolver el precio mediante SELECT precioLinea o bien operar sobre él como en SELECT precioLinea * 0.85.

Ahora bien, podemos querer obtener información que no proviene de un registro individual sino de la agrupación de información, como es el caso de contar el número de líneas de pedido, sumar el precio de todas las líneas por cada pedido, etc. Para ello, debemos emplear funciones agregadas y en la mayoría de los casos agrupar por algún campo.
Así, para ver el número total de registros podemos hacer:

```
1   SELECT COUNT(*)
2   FROM LINEAPEDIDO
```

Si por el contrario deseamos obtener el total de líneas por pedido, debemos indicar que agrupe por idPedido, lo que contará todos los registros con el mismo idPedido y calculará su cuenta:

```
5    SELECT idPedido, COUNT(*)
6    FROM LINEAPEDIDO
7    GROUP BY idPedido
```

Lo mismo se puede aplicar a otras funciones como la suma, indicando en ese caso aparte de la agrupación el campo que queremos sumar:

```
10   SELECT idPedido, SUM(precioLinea)
11   FROM LINEAPEDIDO
12   GROUP BY idPedido
```

A continuación, se muestra un ejemplo de la media de los precios por cada uno de los pedidos, En ese caso necesitamos de Anuevo agrupar (GROUP BY) por pedido.

```
16   SELECT idPedido, AVG(precioLinea)
17   FROM LINEAPEDIDO
18   GROUP BY idPedido
```

Igualmente, podríamos aplicar un redondeo (ROUND) sobre la media, para dejar 4 decimales, y aplicarle un alias (AS) para el nombre del dato de salida.

```
21   SELECT idPedido, ROUND(AVG(precioLinea),4) AS media
22   FROM LINEAPEDIDO
23   GROUP BY idPedido
```

7.3. Creación de copias de seguridad automatizadas

¿Qué es la copia de seguridad y cuántos tipos hay?

Una copia de seguridad o backup es un respaldo que hacemos de archivos físico o virtuales en su lugar secundario como un disco duro externo o nube para su conservación y posterior uso en caso de necesidad.

Hay que tener en cuenta que nuestro ordenador puede estar expuesto a que deje de funcionar en cualquier momento provocando que los datos de nuestro disco duro puedan llegar a corromperse o perderse en caso de fallo. En el momento de que el hardware que hay en el interior del PC deje de funcionar, los datos pueden llegar a perderse, por lo que hacer una copia de seguridad de nuestros archivos más importantes para evitar la pérdida de datos y garantizar que podamos recuperarlos si fuera necesario.

Tipos de Copias de seguridad

- Copia de seguridad completa: Este es el tipo de copia de seguridad más común que incluye todo, incorporando objetos, datos de tablas del sistema y transacciones que ocurren durante la copia de seguridad. Con una copia de seguridad completa, se puede restaurar su base de datos al estado original en el que realizó la copia de seguridad. Las copias de seguridad completas no truncarán su registro de transacciones, pero si su base de datos está en recuperación será completa, adicionalmente usted también debe considerar las copias de seguridad del registro de transacciones

- Copia de seguridad diferencial: Este tipo de copia de seguridad ofrece un medio para mantener un historial completo de su base de datos, pero sin almacenar los datos redundantes. Una copia de seguridad diferencial retiene datos desde la última copia de seguridad completa. Una copia de seguridad diferencial solo es útil si la misma se usa en conjunto con una copia de seguridad completa, pero le permite eliminar copias de seguridad diferenciales anteriores ya que las mismas son redundantes

- Copia de seguridad incremental: si solo queremos realizar una copia de los archivos que han sido modificados desde la realización de la última copia, esta será la opción que debemos elegir, siendo la más rápida para realizar nuestro backup.

- Registro de transacciones. Este tipo de copia de seguridad realizará una copia adicional de seguridad de todas las transacciones que se hayan producido desde la última copia de seguridad o truncamiento del registro, luego truncará al registro de transacciones. Esto capturará toda la información de la transacción, tanto DML como también DDL, que haya ocurrido en la base de datos. Con una copia de se-

guridad del registro de transacciones, se puede restaurar una base de datos a un punto particular en el tiempo, también conocido como recuperación en un punto en el tiempo, como para fijar puntualmente justo antes de un evento de pérdida de datos

- Copia de seguridad espejo: este modo de copia de seguridad es bastante similar al de la copia completa, con la salvedad de que los archivos no se pueden comprimir, por lo que además de ser menos seguro también ocupará más espacio de almacenamiento.

Importancia de automatizar copias de seguridad

Un sistema automatizado y periódico de backup permite que la empresa, entidad, organización mantenga siempre a disposición una copia actualizada de su información. Si por alguna circunstancia necesitas recuperar los datos, las pérdidas de información serán mínimas.

Ventajas:

Rapidez en el acceso para recuperarlos tras su pérdida, por causas ajenas a la empresa u otros incidentes, cuándo y cómo quieras. El objetivo siempre será minimizar el tiempo de inoperancia y restauración de la información perdida.

Seguridad de los archivos. No solo están salvaguardados en todo momento, sino que pueden enviarse de manera segura a los sistemas donde se les requiera.
Ahorro de backups físicos. Son mucho más costosos y corren mayor riesgo de sufrir daños. El uso de almacenamiento online de forma simple o múltiple es más efectivo y seguro y rápido si se dispone de conexiones de alta velocidad.

Copias de seguridad diarias. Ni siquiera necesitarás poner a tus empleados a trabajar en ello, las copias de seguridad se harán automáticamente todos los días, o en frecuencias mayores dependiendo de cada negocio y del tiempo disponible para una restauración.

Otros materiales para profundizar

Recursos de video

Sandro Arenas (Director). (2019, septiembre 1). SQL SERVER 2012—BACKUP AUTOMATICO - 05. https://www.youtube.com/watch?v=VnUR7Ro3ebY

Syntax Team (Director). (2022, junio 2). Como crear una copia de seguridad programada en SQL Server 2019. https://www.youtube.com/watch?v=Y1melJzyEF0

Yesica Yineth Ruiz Villamil (Director). (2018). ¿Que es una Base de Datos? https://www.youtube.com/watch?v=J4ipZ4u-mrY

Referencias bibliográficas de la unidad

Arias, Á. (2016). Fundamentos de Programación y Bases de Datos: 2a Edición. IT Campus Academy.

Gamazo, A. A. (2006). Diseño y administración de bases de datos. Universitat Politècnica de Catalunya. Iniciativa Digital Politècnica.

Machado Salazar, G., & Machado Salazar, G. (2020). Marco de testing metamórfico para consultas SQL con valores nulos [Info:eu-repo/semantics/masterThesis]. https://eprints.ucm.es/id/eprint/62473/

Mena Chuquimia, O. (2020). TÉCNICAS DE SEGURIDAD INFORMÁTICA PARA REDUCIR LAS VULNERABILIDADES POR INYECCIÓN SQL EN APLICACIÓN WEB. http://repositorio.upea.bo/handle/123456789/210

Spona, H. (2010). Programación de bases de datos con MySQL y PHP. Marcombo.

Cabello, M. V. N. (2010). Introducción a las Bases de Datos relacionales. Vision Libros.

Elmasri, R. (s. f.). Fundamentos de Sistemas de Bases de Datos.

Gabillaud, J. (2013). SQL Server 2012 - SQL, Transact SQL: Diseño y creación de una base de datos. Ediciones ENI.

ALCALDÍA MAYOR
DE BOGOTÁ D.C.
SECRETARÍA DE EDUCACIÓN

ATENEA
AGENCIA DISTRITAL PARA LA EDUCACIÓN
SUPERIOR LA CIENCIA Y LA TECNOLOGÍA

BOGOTÁ

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS
Acreditación Institucional de Alta Calidad