Fundamentos Tecnológicos para el Tratamiento y Análisis de Datos

Tema 3. Bases de datos relacionales

Índice

Esquema

Ideas clave

3.1. Introducción y objetivos

3.2. Introducción al modelo relacional

3.3. Estructura del modelo relacional

3.4. Lenguaje SQL

3.5. Consultas SQL

3.6. Consulta en base de datos

3.7. Vistas y procedimientos almacenados

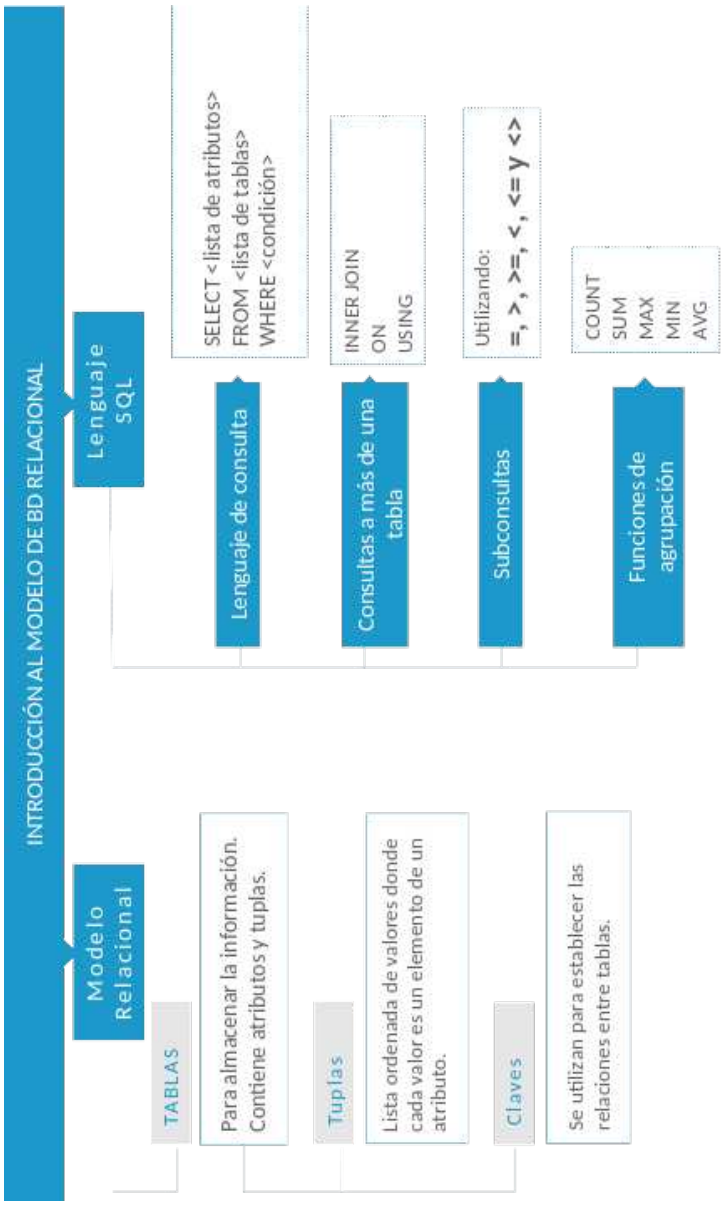
A fondo

Bases de datos NoSQL

Data warehouse

Test

Esquema



3.1. Introducción y objetivos

El modelo relacional es el más utilizado en todas las organizaciones para gestionar sus datos y utilizarlos estratégicamente en la toma de decisiones. Como parte esencial de la inteligencia de negocios basada en datos es importante conocer qué es el modelo relacional, por qué se creó y para qué sirve. La información aprendida en este y otros temas que tratan el modelo relacional permitirá al alumno tener los conocimientos necesarios para ser parte del proyecto de implantación y gestión de la inteligencia de negocios en la empresa.

En concreto, en este tema se desarrollarán conocimientos acerca de la estructura de este modelo, es decir, cómo se almacenan los datos en un modelo relacional y se aprenderá el lenguaje de tratamiento de las bases de datos relacionales, lenguaje SQL.

Conocer la estructura del lenguaje SQL permitirá familiarizarse con el lenguaje que utilizan los creadores de la base de datos. Además, este facilitará el realizar consultas directamente sobre la BD de cierto grado de complejidad, sin la necesidad de intermediación por parte de un informático.

Los principales objetivos del tema son:

- Conocer qué es un modelo relacional, sus características y para qué sirve.
- Aprender la estructura del modelo relacional.
- Conocer cómo se almacenan y se gestionan los datos en un modelo relacional.
- Aprender la terminología del lenguaje SQL.

Ideas clave

- Conocer el lenguaje SQL necesario para realizar consultas en una base de datos relacional.

3.2. Introducción al modelo relacional

Cuando hablamos de bases de datos, el modelo relacional es el principal modelo que utilizan los SGBD y las bases de datos más reconocidas en el mundo. Este modelo fue presentado por primera vez por Edgar F. Codd de IBM Research en 1970. El objetivo era **mantener la independencia de la estructura lógica** de la BD respecto a la forma de almacenamiento y otras características físicas. Esto fundamentalmente hace que su uso por parte de usuarios no expertos en creación y mantenimiento de BD sea más sencillo.

El modelo está basado en la teoría de las relaciones, es decir, utiliza el concepto de relación matemática como forma de representación de los datos almacenados. Así, los datos son representados mediante un conjunto de relaciones, las cuales se denominan **tablas**. Estas representan las relaciones existentes entre las diferentes variables y estructuras de la BD y con ellas se consigue crear el universo de la realidad que corresponde a dicha BD.

El elemento básico del modelo relacional es la **relación**, que se representa mediante una tabla con dos dimensiones (filas y columnas). El conjunto de una base de datos puede ser representada mediante un conjunto de tablas que, relacionadas entre sí, deben crear una estructura que sea válida para representar la realidad para la cual fue creada.

Dentro del modelo relacional se pueden distinguir los siguientes elementos que componen una tabla:

- **Nombre** de la tabla único que la distingue de las demás.
- Un conjunto de columnas denominadas **atributos**, que representan las propiedades de una tabla y se distinguen entre ellas por su nombre.

Ideas clave

- Un **dominio** que contiene el tipo de valores posibles que puede tomar cada uno de los atributos.
- Un conjunto de filas llamadas **tuplas** y que son distintas observaciones de la tabla y los atributos.
- Un número de tuplas (filas) que se denomina **cardinalidad** de la tabla.
- Un número de atributos (columnas) que se denomina **grado de la tabla**.

Todos estos elementos necesarios para crear una tabla y, por tanto, una BD serán analizados en profundidad a continuación.

3.3. Estructura del modelo relacional

Tablas

La tabla 1 muestra un ejemplo de una tabla extraída de una base de datos que almacena información acerca de una universidad. En la tabla, el título (ALUMNO) aparecerá debidamente colocado en la parte superior.

En esta se muestran una serie de **atributos** (columnas):

- nAl (número de identificación del alumno).
- dni (el DNI de los alumnos).
- Nombre.
- fechaNac.
- lugar (que hace referencia a la procedencia del alumno).

Estos atributos muestran el grado de la tabla, en este caso 5. También aparecen una serie de **tuplas** (filas) que indican cada una de las observaciones que se poseen de dichos atributos, en este caso los alumnos 145, 214, 112 y 220 que conjuntamente forman la **cardinalidad** de la tabla, en este caso 4.

ALUMNO

nAl	DNI	nombre	fechaNac	lugar
145	29.254.123	Manuel López González	15/03/75	Huelva
214	29.147.158	Mercedes Gómez Martín	20/06/76	Huelva
112	44.125.325	Francisco Gallego Macías	01/01/75	Cádiz
220	29.555.247	Beatriz Rico Vázquez	12/11/74	Cádiz

Tabla 1. Tabla de BD relacional ALUMNO.

Es un ejemplo de las muchas tablas que puede formar la base de datos de la universidad en cuestión. En concreto recoge información de los alumnos. La realidad de una universidad y la necesidad de almacenamiento podrían crear un conjunto de tablas y atributos, como se detalla a continuación:

ALUMNO (nAl, dni, nombre, fechaNac, etc.)
PROFESOR (nPr, dni, nombre, despacho, etc.)
ASIGNATURA (idAsig, nombre, créditos, etc.)
MATRICULA (nAl, idAsig, nPr, año, trimestre, etc.)

Este conjunto de tablas representa una base de datos de una universidad, que podría estar destinada a almacenar las matriculaciones de los alumnos en las distintas asignaturas y las asignaturas que imparte cada profesor.

Dominio

Como hemos dicho, el dominio (D) es un conjunto finito de valores homogéneos, del mismo tipo que toma un atributo. Un dominio se compone de:

- Nombre: para referenciarlo y ayudar a interpretar los valores que toma dicho dominio.
- Tipo de datos: que puede ser simples (carácter, entero, decimal, etc.) o definidos (número de teléfono, metros, kilogramos, etc.)

Además, existen dos formas por las cuales pueden ser definidos los dominios:

- **Intensión**: especificando el tipo de datos, por ejemplo, números enteros entre 0 y 100.
- **Extensión**: indicando el conjunto de valores posibles, por ejemplo, valores entre las siguientes ciudades (Logroño, Madrid, Badajoz o Huelva).

Hay que tener claro que cada atributo toma distintos valores de un solo dominio específico. Sin embargo, varios atributos pueden tener el mismo dominio en el caso de que sus valores sean los mismos.

Tupla y relación

Como hemos visto, una tupla es una lista ordenada de valores donde cada valor es un elemento de un determinado atributo que forma parte de un dominio, o bien, es un valor nulo.

Las relaciones de una tabla son definidas por el conjunto de tuplas donde cada una toma valores dentro del dominio de cada atributo. Cada relación (tabla) tendrá un nombre diferente al resto de relaciones y debe guardar las siguientes **restricciones**:

- No pueden existir tuplas repetidas.
- No existe orden establecido entre las tuplas.
- No pueden existir dos atributos con el mismo nombre.
- El orden de los atributos no es relevante.
- Los valores de los atributos son atómicos, es decir, cada atributo toma un solo valor del dominio.
- Pueden existir tuplas con valores desconocidos (nulos).

Las tablas (relaciones) pueden representar hechos sobre entidades (alumnos, profesores, asignaturas, etc.), o bien, pueden representar relaciones entre varias entidades, por ejemplo, la relación entre los alumnos y las asignaturas.

Claves

Como hemos dicho, todas las tuplas deben ser diferentes. Esta restricción hace que uno o varios atributos de cada tabla (relación) sean específicos de esa tabla. En general, siempre deben existir diferentes subconjuntos de atributos formados por uno o varios atributos que hacen cumplir la propiedad de que dos tuplas no pueden tener los mismos valores.

A este c**onjunto no vacío de atributos que identifica unívocamente cada tabla** se denomina clave. Cabe destacar que este subconjunto debe ser mínimo, es decir, debe contener el menor número de atributos por el cual se pueda identificar una tabla (relación), por ejemplo, si con el nPr de los profesores podemos identificarlos no es necesario incluir también su nombre.

En concreto, una relación (tabla) puede tener más de una clave que se denominarán **claves candidatas**, pero solo existirá una clave que elegirá el diseñador de la BD y se la denominará clave **primaria**. El resto de las claves candidatas que no sean elegidas se denominarán claves **alternativas**.

Para establecer relaciones entre diferentes entidades (tablas) se utilizan las claves primarias.

A continuación, ilustraremos lo explicado mediante un ejemplo.

Ejemplo: Clave ajena, clave primaria

La tabla 2 representa la información recogida en la base de datos de la entidad COMPUTADOR. Como vemos, el único atributo que identifica cada tupla de forma

individual es idOrd. Este atributo será, por tanto, elegido como clave primaria para identificar esta tabla

COMPUTADOR

nAl	nombre	lugar
Ord025	Servidor NT	Control
Ord008	PC Prácticas	Aula 7
Ord009	PC Prácticas	Aula 7
Ord010	PC Prácticas	Aula 8
Ord040	Beatriz Rico Vázquez	Aula 10

Tabla 2. Tabla de BD relacional COMPUTADOR.

Por otro lado, la tabla 3 representa a los alumnos y contiene atributos de distinto tipo (identificación, DNI, nombre, fecha de nacimiento, lugar de procedencia, número de hermanos). Además, se ha incluido un último atributo que hace referencia al computador que utiliza cada alumno en las clases.

Como vemos, para identificarlos se ha incorporado un nuevo atributo denominado computador , que toma los valores de la clave primaria elegida en la tabla 2 para identificar inequívocamente a cada uno. A esta clave añadida a la tabla ALUMNO para relacionarla con la tabla COMPUTADOR se la conoce como **clave ajena**.

ALUMNO

nAl	DNI	nombre	fechaNac	lugar	nH	computador
145	29.254.123	Manuel López González	15/03/75	Huelva	0	Ord009
214	29.147.158	Mercedes Gómez Martín	20/06/76	Huelva	1	Ord010
112	44.125.325	Francisco Gallego Macías	01/01/75	Cádiz	2	Ord009
088	29.214.856	Teresa Díaz Camacho	25/10/71	Madrid	4	Ord010
220	29.555.247	Beatriz Rico Vázquez	12/11/74	Cádiz	2	Ord025

Tabla 3. Tabla ALUMNO con clave ajena.

Gracias a este proceder, cada alumno está perfectamente relacionado con un computador.

Con este ejemplo se ha ilustrado el modo en que el modelo de datos relacional consigue crear relaciones entre distintas entidades para así, poder reflejar la realidad de una entidad. En concreto, seleccionando una **clave primaria** que identifique a cada una de las tuplas de una tabla e introduciendo esta como clave ajena en otra tabla, se consigue crear una relación entre ambas. En nuestro ejemplo, conseguimos conocer qué computador está utilizando cada alumno.

Ahora veremos la forma en la que se expresa toda la información que contiene una tabla:

ALUMNO (nAI, dni, nombre, fechaNac, lugar, nH, ordenador)
CP: nAI
CAj: ordenador ORDENADOR (idOrd)

- En negrita aparece el **título** de la tabla y todos los atributos que contiene, entre paréntesis.
- CP hace referencia a la **clave primaria** de esa tabla.

- CAj hace referencia a la **clave ajena** incorporada, indicando el nombre del atributo en la tabla ALUMNO y, con una flecha, indicando a la clave primaria de la tabla correspondiente.

Existen otra serie de restricciones necesarias para que el universo de la BD se ajuste lo máximo posible a la realidad:

- Restricción de **obligatoriedad**: no permite que existan valores nulos en determinados atributos.
- Restricción de **unicidad**: no permite que en determinados atributos existan valores repetidos.
- Restricción de **integridad semántica**: para introducir solo valores deseados.

3.4. Lenguaje SQL

La creación de las BD relacionales trajo consigo la necesidad de crear un lenguaje que permitiese definir, manipular y hacer consultas a los datos almacenados en dichas bases. Este lenguaje es SQL.

El **lenguaje de definición de datos** se utiliza para crear la estructura de la BD y crear las vistas externas que se van a realizar sobre esta. Mediante este lenguaje se describen y nombran las entidades, atributos y relaciones y se definen las restricciones de integridad y seguridad.

Toda esta información queda almacenada en los **metadatos**, que describen los objetos de la BD y que hacen más fácil su acceso y manipulación.

Un ejemplo de la sintaxis del lenguaje de definición sería:

```
CREATE TABLE Alumno(
        nAl INT PRIMARY KEY,
        DNI INT,
        NombreAlumno VARCHAR(100),
        fechaNac DATE,
        lugar VARCHAR(100),
        nH INT,
        computador VARCHAR(100));
```

También se puede manipular datos de dos formas diferentes, extrayéndolos o actualizándolos. A la parte relacionada con la extracción de datos se la denomina **lenguaje de consulta** y sirve para realizar extracciones de datos a medida.

Entre las instrucciones que se pueden dar para actualizar los datos están:

- INSERT : sirve para insertar nuevos datos a la BD.
- Ejemplo: insertar una serie de filas en la tabla Computador.

```
INSERT INTO Computador (nAIID, nAI, Nombre, lugar)
VALUES (25 ,'Ord025', 'Servidor NT', 'Control'),
(8 ,'Ord008' , 'PC Practicas', 'Aula 7'),
(9,'Ord009' , 'PC Practicas', 'Aula 7'),
(10,'Ord010' , 'PC Practicas', 'Aula 8'),
(40,'Ord040' , 'Beatriz Rico Vazquez', 'Aula 10');
```

- DELETE **:** se utiliza para eliminar datos que contiene la BD.

- Ejemplo: eliminar una fila concreta basada en el valor del DNI.

```
DELETE FROM Alumno
WHERE DNI = 29555247;
```

- **UPDATE:** se utiliza para actualizar el valor de un determinado dato.

- Ejemplo: eliminar las filas o tuplas con ordenadores que estén en Aula 10, y el número del ordenador sea 040.

```
UPDATE Computador
SET Nombre = 'Ordenador Personal'
WHERE Computador = 'Ord040' AND lugar = 'Aula 10';
```

Normalmente este tipo de operaciones está solo permitido a las personas encargadas de mantener la BD, que suelen ser informáticos, pero existe una serie de operaciones que pueden ser realizadas por los usuarios de la información sin modificar los datos: las consultas SQL.

Lenguaje de consulta SQL

El lenguaje de consulta SQL se utiliza para realizar **consultas específicas** a la BD:

- El usuario introduce el código SQL que realiza dicha consulta específica.

- El SGBD específico realiza la tarea de buscar dicha información, extraerla de la BD y presentársela al usuario.

La forma de realizar una consulta se resume en los siguientes **comandos** que son las instrucciones básicas:

```
SELECT <lista de atributos>
FROM <lista de tablas>
WHERE <condición>
```

Donde:

- La instrucción SELECT sirve para indicar qué atributos queremos recopilar, es decir, el objetivo de nuestra consulta y esta será la información que obtengamos. Siempre debemos incluir el alias delante del nombre del atributo; por ejemplo, A. Nombre .
- La instrucción FROM hace referencia a la tabla exacta de dónde deben extraerse los atributos indicados previamente. Es importante asignarle un ALIAS para estandarizar la nomenclatura de las tablas; por ejemplo, dbo.Asignatura AS A .
- La instrucción WHERE se trata de una expresión lógica que condiciona la consulta a las tuplas que cumplan dicha condición. En caso de existir alias de tablas, estos deben ser incluidos en la selección de atributos; por ejemplo, A.Curso = '3' .

Consulta SQL

Imaginemos que tenemos una BD con información de una universidad y se desea conocer el código, el nombre y la especialidad de las asignaturas de tercer curso que tengan más de 4.5 créditos. La consulta SQL que recogerá dicha información es la siguiente:

```
SELECT asi.idAsig AS CodAsignatura, asi.Nombre, asi.EspecialidadFROM dbo.Asignatura AS asiWHERE asi.Curso = '3' AND asi.Creditos > 4.5;
```

3.5. Consultas SQL

En este apartado estudiaremos algunas peculiaridades y posibilidades del lenguaje SQL para realizar consultas.

- Es habitual asignar un alias a los nombres de las tablas y cualificar los atributos para eliminar la ambigüedad y facilitar la escritura. Esto se consigue anteponiendo el nombre de la tabla o el alias a los atributos y separando ambos con un punto.
- Los nombres de los atributos pueden renombrarse escribiendo el nuevo detrás de la sentencia `AS` en la cláusula `SELECT`.
- La cláusula `WHERE` se puede omitir para obtener una selección de tuplas sin ninguna condición.
- Para obtener todos los valores de una determinada tabla se utiliza el carácter *.
- Si se obtienen tuplas repetidas como resultado de la consulta, SQL no las sintetiza en una solo; para esto se debe utilizar la cláusula `DISTINCT`.

Consultas SQL

Obtener toda la información de las asignaturas:

```
SELECT*
FROM dbo.Asignatura;
```

Obtener los distintos tipos de computadores que existen:

```
SELECT DISTINCT com.Nombre
FROM dbo.Computador AS com;
```

Obtener el número de descuento a realizar por alumno, sabiendo que el descuento es de 300 € por número de alumnos. Ordenar el resultado de mayor a menor descuento y, en caso de igualdad, ordenarlo

alfabéticamente.

```
SELECT al.nH * 300 AS Descuento, al.NombreAlumno
FROM dbo.Alumno AS al
ORDER BY Descuento DESC, al.NombreAlumno ASC;
```

Operaciones aritméticas

También se pueden calcular **operaciones aritméticas** sobre los campos a extraer:

- Para ordenar los resultados se utiliza la cláusula `ORDER BY` y las palabras `DESC` o `ASC` para ordenarlo de forma ascendente o descendente.
- La cláusula `BETWEEN` sirve para seleccionar valores entre dos valores.
- La cláusula `IN` comprueba que un valor pertenece a un conjunto de valores.

Si no conocemos toda la información:

- Para extraer determinados datos, podemos utilizar el operador `LIKE` .
- Para la descripción de dichos patrones, que sigue una determinada búsqueda que no conocemos exactamente, se pueden utilizar dos caracteres especiales:
  - **%** sustituye un número arbitrario de caracteres.
  - _ sustituye un solo carácter.
- `NOT` delante del operador `LIKE` busca elementos que no sean similares a lo indicado.

Consultas SQL con operaciones aritméticas

Obtener el DNI de los alumnos nacidos entre 1990 y 2000 cuya localidad de procedencia sea Murcia:

```
SELECT al.DNI
FROM dbo.Alumno AS al
WHERE al.fechaNac BETWEEN '19700101' AND '20001231'
AND al.lugar IN ('Madrid');
```

Obtener el DNI y los nombres de los alumnos cuyo nombre comience por la letra A y su procedencia sea de una ciudad que tenga al menos 6 caracteres, pero no comience por la letra C.

```
SELECT al.DNI, al.NombreAlumno
FROM dbo.Alumno AS al
WHERE al.NombreAlumno NOT LIKE 'M%'
AND al.lugar LIKE '%_____%'
AND al.lugar NOT LIKE 'C%';
```

Consultas a más de una tabla

- En ocasiones la información que deseamos obtener parte de la relación entre dos tablas; por eso es necesario utilizar ciertas cláusulas que permitan esto. Existen diversas tipologías, siendo las cláusulas INNER JOIN y LEFT JOIN las más utilizadas.
- Para ello, será necesario indicar cuáles son la clave primaria y ajena utilizando la sentencia ON .
- En caso de que el nombre del atributo sea el mismo en ambas tablas, utilizamos la cláusula USING .

INNER JOIN

Al utilizar cláusulas de combinación buscamos conectar ambas tablas en función de la concordancia entre las claves. En el caso de INNER JOIN la combinación de las tablas se realiza mediante la coincidencia total de registros entre ambas tablas.

Consultas SQL a más de una tabla

Obtener el nombre de las asignaturas y profesores responsables:

```
SELECT asi.Nombre, pr.Nombre
FROM Asignatura AS asi
INNER JOIN Profesor AS pr ON pr.ProfID = asi.ProfID;
```

Obtener el número de alumnos que tiene la asignatura «Fundamentos Tecnológicos para el Tratamiento de Datos» en 2019:

```
SELECT ma.nAl
FROM Matricula AS ma
INNER JOIN Asignatura AS asi ON asI.idAsig = ma.idAsig
WHERE asi.Nombre = 'Fundamentos Tecnológicos para el Tratamiento de Datos' AND
ma.An=2019;
```

En la primera consulta, si ejecutamos ambas tablas por separado veremos que en la tabla Asignatura hay un total de 9 registros, de los cuales 7 tienen un profesor asignado. Al combinar ambas tablas el resultado es de 7, puesto que las asignaturas «Trabajo Fin de Master» y «Prácticas» no tienen un profesor asignado.

En el caso de la segunda consulta, observamos que hay una **coincidencia total** entre la clave primaria y la clave ajena entre Matricula y Asignatura.

LEFT JOIN / LEFT OUTER JOIN

La cláusula LEFT JOIN permite la combinación entre dos o más tablas, de manera que muestra todos los registros de la primera, así como los que coincidan de la segunda. En caso de no coincidencia, el valor procedente de la segunda tabla toma

valor nulo. Teniendo en consideración la anterior consulta, si reemplazamos el INNER JOIN por LEFT JOIN obtenemos los nueve registros de la tabla Asignaturas, entre los que se incluyen en esta ocasión «Trabajo Fin de Master» y «Prácticas». Si observamos el atributo «nombre» de la tabla Profesor, los valores para esos dos registros son nulos, puesto que no existe correspondencia en la tabla Profesor para dichas asignaturas. Existe una variación de esta cláusula denominada RIGHT JOIN, donde se mostrarían todos los registros de la tabla 2 y los correspondientes, si existen, de la tabla 1.

Obtener el nombre de todas las asignaturas y profesores responsables:

```
SELECT asi.Nombre, pr.Nombre
FROM dbo.Asignatura AS asi
LEFT JOIN dbo.Profesor AS pr ON pr.ProfID = asi.ProfID;
```

Obtener el nombre del alumno, trimestre, curso, convocatoria, calificación, nombre de asignatura, especialidad, créditos y nombre del profesor para la convocatoria de junio y que se imparte en el trimestre 3 o 4:

```
SELECT al.NombreAlumno, ma.Curso,ma.Calificacion, asi.Nombre, asi.Especialidad,
asi.Creditos, pr.Nombre
FROM dbo.Matricula AS ma
INNER JOIN dbo.Asignatura AS asi ON asi.idAsig = ma.idAsig
INNER JOIN dbo.Alumno AS al ON al.nAI = ma.nAI
LEFT JOIN dbo.Profesor AS pr ON pr.ProfID = ma.ProfID
WHERE ma.Convocatoria = 'Junio' AND ma.Trimestre IN (3,4);
```

Obtener todos los valores de las tablas Alumno y Matriculo, teniendo en consideración que se ha agregado un código de alumno nuevo en la tabla de Matricula, pero no Alumno.

```
SELECT *
FROM dbo.Alumno AS al
```

```
RIGHT JOIN dbo.Matricula AS ma ON ma.nAI=al.nAI
```

En función de la estructura de la base de datos y sus tablas optaremos en la utilización de un tipo de JOIN o de otro. Por ejemplo, cuando queremos darle más énfasis a una tabla que a otra (mostrar todos los registros) deberemos utilizar LEFT o RIGHT JOIN .

Subconsultas

En ocasiones, para seleccionar valores dentro de una base de datos sin conocerlos y poder utilizarlos, como condición es necesario realizar una consulta previa y conocer estos valores.

SQL permite realizar una consulta dentro de la condición para evitar el exceso de trabajo. Se puede utilizar los comandos de comparación ($, >, \geq, <, \leq y \neq$) para contrastar un valor de la tupla que se está examinando con un único valor producido por la subconsulta. Esta debe devolver un único valor si se utilizan operadores de comparación.

Subconsultas SQL

Obtener el nombre de los alumnos que procedan de la misma ciudad que Francisco Galle y tengan el mismo número de hermanos que Beatriz Rico:

```
SELECT al.NombreAlumno
FROM Alumno AS al
WHERE al.lugar = (SELECT Lugar FROM Alumno WHERE NombreAlumno = 'Francisco
Gallego Macias')
AND al.nH = (SELECT nH FROM Alumno
```

Funciones de agrupación

Estas funciones sirven para **sintetizar la información** extraída en la consulta para que sea más fácil su uso. Por ejemplo, es probable que deseemos conocer el número de personas que realizan una acción o el valor máximo de un determinado valor.

Las funciones más habituales son:

- COUNT : devuelve el número de filas del resultado de la consulta.
- SUM : suma.
- MAX : máximo.
- MIN : mínimo.
- AVG : promedio.

Consultas SQL con funciones

Obtener el número de matriculados, la nota mínima y la nota media de la asignatura «Fundamentos Tecnológicos para el Tratamiento de Datos» en la convocatoria de diciembre de 2019.

```
SELECT COUNT(*) AS NumMatri,   MIN(Calificacion) AS Minima,
AVG(Calificacion) AS Media
FROM Asignatura AS asi
INNER JOIN Matricula AS ma ON ma.idAsig = asi.idAsig
WHERE asi.Nombre = 'Fundamentos Tecnológicos para el Tratamiento de Datos' AND
ma.An=2019;
```

A veces, es necesario aplicar estas funciones integradas a subgrupos de tuplas. Esto se logra a través de la cláusula GROUP BY .

En otras ocasiones, solo queremos utilizar estas funciones con grupos de tuplas que

satisfagan ciertas condiciones. Para ello, se puede utilizar la cláusula HAVING , la cual aparece junto con la cláusula GROUP BY . Esta cláusula especifica una condición y el resultado solo mostrará aquellos grupos que la satisfagan.

Consultas SQL con funciones a subgrupos de tuplas

Para cada asignatura, mostrar el nombre de esta, el número de alumnos que se han presentado y la nota media obtenida en la convocatoria de junio:

```
SELECT asi.Nombre, COUNT(*) AS Presentados, AVG(Calificacion) AS Media
FROM Asignatura AS asi
 INNER JOIN Matricula AS ma ON asi.idAsig = ma.idAsig
WHERE ma.Convocatoria = 'Junio' AND ma.An = 2020
GROUP BY asi.Nombre;
```

Para cada asignatura, mostrar el nombre de esta, la convocatoria, el número de alumnos que se han presentado y la nota media obtenida.

```
SELECT ma.Convocatoria, asi.Nombre, COUNT(*) as Presentados,
AVG(Calificacion) AS Media
FROM Asignatura AS asi
INNER JOIN Matricula AS ma ON asi.idAsig = ma.idAsig
GROUP BY ma.Convocatoria, asi.Nombre;
```

Nombre de los alumnos que hayan sacado más de un 5 de nota media en junio del 2019.

```
SELECT al.NombreAlumno, AVG(ma.Calificacion)
FROM Alumno AS al
INNER JOIN Matricula AS ma ON ma.nAI = al.Nai
WHERE ma.An = 2019 AND ma.Convocatoria='Junio'
```

```
GROUP BY al.NombreAlumno
HAVING AVG(ma.Calificacion) > 5
```

3.6. Consulta en base de datos

A continuación, procederemos a realizar consultas SQL en una base de datos *real*. En este caso utilizaremos AdventureWorks de Microsoft, que ofrece un entorno de base de datos simulado de una empresa real, la cual se dedica a la fabricación y distribución de bicicletas y accesorios, tanto a clientes individuales como a empresas.

La presente base de datos ofrece diferentes posibilidades de análisis, desde del departamento de recursos humanos hasta la venta a clientes individuales o tiendas.

Es un ejemplo simulado de la realidad de una base de datos empresarial, donde los diferentes departamentos son representados por medio de datos y tablas.

Antes de comenzar, debemos reflejar que la estructura inherente de la base de datos considera a los individuos, ya sean personas o tiendas, como entidades de negocio (BusinessEntity). Por lo tanto, todos y cada uno de los individuos tiene un código único referido como BusinessEntityID , que está presente en las tablas oportunas.

Recursos humanos

Comenzaremos desde el punto de vista de los RRHH. Para ello necesitaremos **diferentes tablas**:

- HumanResources.Employee : tabla principal que contiene los datos de los empleados, en la que BusinessEntityID es la clave primaria, que, en este caso, hace referencia a empleados.
- Person.Person : todos los humanos o individuos y sus datos principales. La clave primaria es BusinessEntityID.

- HumanResources.EmployeePayHistory : hace referencia al salario del empleado, a la frecuencia de salario (1=mensual y 2=bisemanal) y a las fechas de última modificación. La clave primaria es conjunta, a partir de la combinación de BusinessEntityID y RateChangeDate.

- HumanResources.EmployeeDepartmentHistory y HumanResources.Department : tablas que hacen referencia a la organización de empleados, a su turno, al departamento en el que trabajan y, si fuera caso, al cambio de departamento. Para identificar los empleados que están trabajando actualmente en un departamento específico es necesario incluir la condición de EndDate IS NULL .

Empezaremos con una consulta relativamente sencilla, que devuelva información sobre la cantidad de empleados por departamento. Para ello debemos combinar la tabla HumanResources.Employee con EmployeeDepartmentHistory y HumanResources.Department . Hay que tener en cuenta que debemos filtrar de tal forma que aparezca solo el departamento actual, es decir, que el atributo EndDate de la tabla DeparmentHistory sea nulo.

Además, necesitamos filtrar de forma descendente, en función del número de empleados:

```
SELECT hrd.Name AS Departamento, COUNT(hre.BusinessEntityID) AS NEmpleados
FROM HumanResources.Employee AS hre
INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh ON edh.BusinessEntityID =
hre.BusinessEntityID
INNER JOIN HumanResources.Department AS hrd ON hrd.DepartmentID = edh.DepartmentID
WHERE edh.EndDate IS NULL
GROUP BY hrd.Name
ORDER BY NEmpleados DESC
```

Teniendo en consideración dichas tablas, necesitamos ver la lista de empleados, con sus atributos principales (290 empleados) y combinando todas las tablas adecuadamente a través de las claves primarias:

```
SELECT
```

```
hre.NationalIDNumber,
hre.LoginID,
hre.OrganizationLevel,
hre.JobTitle,
hre.BirthDate,
hre.MaritalStatus,
hre.Gender,
hre.HireDate,
pp.FirstName,
pp.LastName,
hrph.*,
hrph.Rate * 40 * 30 AS MonthlySalary,
hrd.*
FROM HumanResources.Employee AS hre
 INNER JOIN Person.Person AS pp ON pp.BusinessEntityID = hre.BusinessEntityID
 INNER JOIN (SELECT BusinessEntityID, MAX(RateChangeDate) AS RateChangeDate, MAX(Rate)
AS Rate, PayFrequency FROM HumanResources.EmployeePayHistory GROUP BY BusinessEntityID,
PayFrequency) AS hrph ON hrph.BusinessEntityID = hre.BusinessEntityID
 INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh ON edh.BusinessEntityID =
hre.BusinessEntityID
 INNER JOIN HumanResources.Department AS hrd ON hrd.DepartmentID = edh.DepartmentID
WHERE edh.EndDate IS NULL
ORDER BY hre.BusinessEntityID
```

Analicemos la consulta paso a paso:

- Debemos combinar la tabla `HumanResources.Employee` , datos y listado de los empleados, con `Person.Person` para obtener el nombre y el apellido del empleado, uniéndolas a través de la CP B `usinessEntityID` .

- Debemos combinar nuestra consulta con `EmployeePayHistory` , donde viene reflejado el salario por hora del empleado. Debemos tener cuidado, puesto que en la tabla existe un registro histórico de los salarios de los empleados; por lo tanto, un empleado tiene varios registros. Para ello realizamos una subconsulta obteniendo la última modificación del salario y el montante del mismo por hora.

```
SELECT BusinessEntityID, MAX(RateChangeDate) AS RateChangeDate, MAX(Rate) AS Rate,
```

```
PayFrequency
FROM HumanResources.EmployeePayHistory
GROUP BY BusinessEntityID, PayFrequency
```

- Combinamos `EmployeeDepartmentHistory` y `Department` para obtener los departamentos a los que pertenecen los empleados actualmente. Ciertamente, no necesitamos ningún atributo de la tabla `DeparmentHistory`, pero la necesitamos como enlace para combinar nuestra consulta con la tabla de departamento. Debemos incluir en la cláusula `WHERE`, que el `EndDate` sea nulo para combinar solo los registros de los empleados que pertenecen en estos momentos a un cierto departamento. En caso de duda, es posible lanzar la siguiente consulta:

```
SELECT *
FROM HumanResources.EmployeeDepartmentHistory
WHERE BusinessEntityID = 4
```

Donde observamos que el empleado con ID 4 ha trabajo en un departamento y en la fecha indicada dejó de trabajar para el mismo y cambió.

Clientes

Ahora abordaremos la información relevante sobre los clientes. Existen dos tipologías principales: individuos y tiendas. En las siguientes consultas buscaremos la información referente a sus datos demográficos. Para ello, utilizaremos tablas y una **vista**, concepto que aprenderemos en el siguiente apartado. A grandes rasgos, una vista no es más que una tabla virtual que creamos a partir de una consulta especifica.

- La tabla `Sales.Customer` contiene el ID de cliente, información respecto a si es cliente individual (`PersonID`) o tienda (`StoreID`) y al territorio del mismo, entre otros.

- Necesitamos combinar la consulta con la tabla `Person.Person` para conseguir la información relevante de los clientes individuales. Para ello utilizamos la clave

primaria BusinessEntityID de la presente tabla y la conectamos con PersonID de la tabla Sales.Customer . Además, debemos incluir la condición que el tipo de persona (PersonType) sea 'IN', que hace referencia a clientes individuos.

- Para conseguir información demográfica debemos combinar la vista Sales.vPersonDemographics utilizando como conectores BusinessEntityID y PersonID.

```
SELECT sc.CustomerID, sc.PersonID, pp.PersonType, pp.FirstName, pp.LastName, vpd.*
FROM Sales.Customer AS sc
 INNER JOIN Person.Person AS pp ON pp.BusinessEntityID = sc.PersonID
 INNER JOIN Sales.vPersonDemographics AS vpd ON vpd.BusinessEntityID = sc.PersonID
WHERE pp.PersonType = 'IN'
```

A continuación, también queremos obtener la información respecto a clientes-tiendas: su nombre, sus sedes, así como los montantes de las ventas realizadas, lo cual incluye la diferencia entre el presente año y el pasado.

Para ello volveremos a utilizar la tabla Sales.Customers , pero en esta ocasión la combinaremos con:

- La tabla Sales.Store , para obtener el nombre de la tienda mediante la unión de BusinessEntityID con el StoreID de la tabla Sales.Customer .
- La tabla Sales.SalesTerritory , para obtener la localización de las tiendas en distintos grados utilizando la clave TerritoryID de ambas tablas.

```
SELECT sc.CustomerID,
sc.StoreID,
sc.AccountNumber,
ss.Name,
sst.Name AS Territory,
sst.CountryRegionCode,
sst.[Group],
sst.SalesYTD,
```

```
sst.SalesLastYear,
sst.SalesYTD - sst.SalesLastYear AS Diff
FROM sales.Customer AS sc
INNER JOIN Sales.Store AS ss ON ss.BusinessEntityID = sc.StoreID
INNER JOIN Sales.SalesTerritory AS sst ON sst.TerritoryID = sc.TerritoryID
```

Supongamos que nos solicitan ahora que rediseñemos la consulta y arrojemos los valores de venta agrupados por territorio y región del país. Para ello mantendremos las tablas, pero debemos modificar las cláusulas introduciendo funciones de agregación como SUM () y GROUP BY .

```
SELECT sst.Name AS Terrotory,
sst.CountryRegionCode,
SUM(sst.SalesYTD) AS TotalCurrentYear,
SUM(sst.SalesLastYear) AS TotalLastYear
FROM sales.Customer AS sc
 INNER JOIN Sales.Store AS ss ON ss.BusinessEntityID = sc.StoreID
 INNER JOIN Sales.SalesTerritory AS sst ON sst.TerritoryID = sc.TerritoryID
GROUP BY sst.Name, sst.CountryRegionCode
ORDER BY SUM(sst.SalesYTD) DESC
```

Ventas

La información de mayor relevancia a la hora de medir las KPI de una empresa está fuertemente relacionada con las **ventas de artículos o prestación de servicios**. La estructura de la base de datos en referencia a las ventas es compleja y contiene numerosas tablas. Para facilitar el aprendizaje y la comprensión utilizaremos solo algunas de ellas.

- La tabla Sales.SalesOrderHeader contiene la información general de las ventas, siendo la clave primaria SalesOrderID , que básicamente es el número de factura. Entre otros atributos presentes podemos encontrar CustomerID , territorio, cuantía y tipo de moneda, entre otros.

- La tabla Sales.SalesOrderDetail contiene los productos que pertenecen a cada factura, su cantidad, su precio y el descuento, entre otros. En este caso la clave primaria es SalesORderDetailID.

Ideas clave

Desde el departamento de ventas nos solicitan el listado de todos los productos vendidos, lo cual incluye su cantidad, precio, nombre y categoría a la que pertenecen. El período temporal solicitado esta entre el año 2012 y 2014. Para ello debemos conectar la tabla SalesOrderHeade r con la tabla Product y ProductSubcategory , a través de la clave ProductID en el primer caso y ProductSubCategoryID en el segundo. Además, tenemos que añadir la condición que permita filtrar entre los años, con lo cual la consulta queda de la siguiente forma:

```
SELECT sod.SalesOrderID,
 sod.CarrierTrackingNumber,
 sod.OrderQty,
 sod.UnitPrice,
 sod.UnitPriceDiscount,
 sod.LineTotal,
 sod.ModifiedDate,
 PP.Name AS ProductName,
 pp.Color,
 pps.Name AS [Group]
FROM Sales.SalesOrderDetail AS sod
 INNER JOIN Production.Product AS pp ON pp.ProductID = sod.ProductID
 INNER JOIN Production.ProductSubcategory AS pps ON pps.ProductSubCategoryID =
pp.ProductSubcategoryID
WHERE YEAR(sod.ModifiedDate) BETWEEN 2012 AND 2014
```

Sin embargo, los jefes de departamento no están del todo convencidos y nos solicitan modificar la consulta para arrojar el *top ten* de artículos por grupos con mayores ventas en dicho periodo temporal, lo cual incorpore además la cantidad de los mismos. Por lo tanto, procedemos a modificar la consulta de la siguiente forma:

```
 SELECT TOP 10       pps.Name AS [Group],
      SUM(sod.OrderQty) AS TotalQty,
      SUM(sod.LineTotal) AS TotalAmount
FROM Sales.SalesOrderDetail AS sod
      INNER JOIN Production.Product AS pp ON pp.ProductID = sod.ProductID
      INNER JOIN Production.ProductSubcategory AS pps ON pps.ProductSubCategoryID =
pp.ProductSubcategoryID
WHERE YEAR(sod.ModifiedDate) BETWEEN 2012 AND 2014
```

```
GROUP BY pps.Name
ORDER BY SUM(sod.LineTotal) DESC
```

Ya hemos analizado la venta de los productos y su tipología, pero para entender mejor el proceso comercial es necesario obtener los datos de las facturas y los clientes asociados a las mismas. Recordemos que existen dos tipos de clientes, tiendas e individuos. Para ello necesitaremos las siguientes tablas:

- Sales.SalesOrderHeader contiene la información principal de las facturas.
- Sales.Customer contiene la información de la tipología de clientes y la clave primaria C ustomerID es el medio de conexión con la tabla anterior.
- Person.Person contiene la información de los individuos y la combinaríamos con la consulta actual mediante la clave primaria BusinessEntityID y PersonID de la tabla Sales.Customer .
- Sales.Store contiene la información de las tiendas y la combinaríamos con la consulta actual mediante la clave primaria BusinessEntityID y StoreID de la tabla Sales.Customer .
- Sales.SalesTerritory contiene la localización y ubicación de los clientes, tanto individuales como tiendas. Para poder combinarla con nuestra consulta utilizaremos TerritoryID , la cual está presente en la tabla de facturas.
- Sales.vPersonDemographics contiene información demográfica de los clientes individuales.

Debemos señalar que para la realización de las combinaciones debemos utilizar LEFT JOIN en vez de INNER JOIN, debido a que combinamos tanto a los clientes individuales como a las tiendas, cuya información proviene de diferentes tablas. En caso de utilizar INNER JOIN para conectar SalesOrderHeader y Person perderíamos las filas referentes a las tiendas.

```
SELECT sog.SalesOrderID AS Factura,sog.OrderDate,
sog.DueDate,
sog.ShipDate,
sog.[Status],
sog.CustomerID,
sog.TerritoryID,
sst.[Name],
sst.CountryRegionCode,
sst.[Group],
sog.SubTotal,
sog.TaxAmt,
sog.TotalDue,
sc.TerritoryID,
sc.PersonID,
pp.PersonType,
pp.FirstName,
pp.LastName,
sc.StoreID,
ss.[Name],
VPD.*
FROM Sales.SalesOrderHeader AS sog
 LEFT JOIN sales.Customer AS sc ON sc.CustomerID = sog.CustomerID
 LEFT JOIN Person.Person AS pp ON pp.BusinessEntityID=sc.PersonID
 LEFT JOIN Sales.Store AS ss ON ss.BusinessEntityID = sc.StoreID
 LEFT JOIN Sales.SalesTerritory AS sst ON sst.TerritoryID = sOG.TerritoryID
 LEFT JOIN Sales.vPersonDemographics AS vpd ON vpd.BusinessEntityID = sc.PersonI
```

3.7. Vistas y procedimientos almacenados

A lo largo del presente tema hemos visto y empleado el lenguaje de consultas para obtener la información necesaria para el desempeño de un analista de inteligencia de negocio. A continuación, incrementaremos nuestro conocimiento en la presente materia con dos funcionalidades adicionales, que incrementan la eficiencia en la obtención de datos. Estos son la vista y el procedimiento almacenado.

Vista

Su terminología en inglés es *view* y, básicamente, es una tabla virtual que creamos a partir de una consulta específica, mediante la que optimizamos la carga de los datos y reducimos la cantidad de código en nuestras consultas. Entre las ventajas de una vista frente a consultas tradicionales destacan la centralidad, simplificación y centralización de la percepción de la base de datos para los usuarios, así como su funcionalidad en términos de seguridad. Esto se debe a que una vista puede contener tablas a las que el usuario en cuestión no tiene acceso explícito. Para crear una vista primero debemos definir la consulta de datos que necesitamos y, a continuación, agregar la cláusula de creación de esta.

```
CREATE VIEW dbo.vVistaEjemplo
AS
SELECT <lista de atributos>
FROM <lista de tablas>
WHERE <condición>
GO
```

Realicemos un caso práctico real mediante la base de datos Adventure Works. Para ello utilizaremos la consulta anterior referente a los artículos correspondientes a cada factura, sin incluir ninguna condición en la cláusula WHERE.

```
CREATE VIEW dbo.VSoldProducts
AS
   SELECT sod.SalesOrderID,
```

```
        sod.CarrierTrackingNumber,
        sod.OrderQty,
        sod.UnitPrice,
        sod.UnitPriceDiscount,
        sod.LineTotal,
        sod.ModifiedDate,
        PP.Name AS ProductName,
        pp.Color,
        pps.Name AS [Group]
    FROM Sales.SalesOrderDetail AS sod
        INNER JOIN Production.Product AS pp ON pp.ProductID = sod.ProductID
        INNER JOIN Production.ProductSubcategory AS pps ON pps.ProductSubCategoryID =
pp.ProductSubcategoryID
GO
SELECT * FROM dbo.VSoldProducts
SELECT *
FROM dbo.VSoldProducts
WHERE YEAR(ModifiedDate) BETWEEN 2012 AND 2014
```

Una vez creada la vista podemos hacer las consultas como si de una única tabla se tratarse. Por lo tanto, podemos condicionar los resultados utilizando la cláusula `WHERE` y arrojar los resultados entre el año 2012 y el 2014. De hecho, es posible crear vistas que contengan otras. Para modificar o eliminar una vista disponemos de los comandos `ALTER VIEW` y `DROP VIEW`.

Procedimientos almacenados

Los procedimientos almacenados, tal y como su nombre indica, se centran en la definición de una serie de comandos, ya sean de definición o de lectura, los cuales son llamados a través de unos parámetros que definen la misma. A la hora de trabajar en proyectos de inteligencia de negocio, los procedimientos almacenados nos ofrecen diferentes ventajas para la obtención y transformación de los datos. Además, son grandes aliados a la hora de optimizar la obtención de los mismos debido a la definición de parámetros. En un lenguaje menos técnico, imaginemos que tenemos la consulta anterior referente a las facturas y los artículos relacionados. En función de las necesidades de información, un procedimiento almacenado nos permitirá seleccionar y filtrar la información antes de ejecutar las consultas, con lo que la dotará de mayor eficiencia en tiempo y recursos del sistema —por ejemplo,

definiendo parámetros de fecha y grupo de producto—. Para ello, debemos crear el procedimiento mediante los siguientes comandos:

```
CREATE PROCEDURE dbo.Procedimiento
@Parametro1 INT
@Parametro2 VARCHAR(100)
AS
SELECT <lista de atributos>
FROM <lista de tablas>
WHERE <condición1> = @Parametro1 AND <condición2> = @Parametro2
GO

EXEC dbo.Procedimiento
@Parametro1 = 2012
@Parametro2 = 'Bicicletas'
```

Siguiendo con el ejemplo empleado, vamos a diseñar el procedimiento. Primero incluimos el comando `CREATE PROCEDURE` y le daremos un nombre al procedimiento. A continuación, definiremos los parámetros indicando de qué tipo son (`INT`, `VARCHAR`, `DATETIME` , etc…). Se incluirá la consulta que habíamos desarrollado anteriormente incorporando los parámetros dentro de la cláusula `WHERE` .

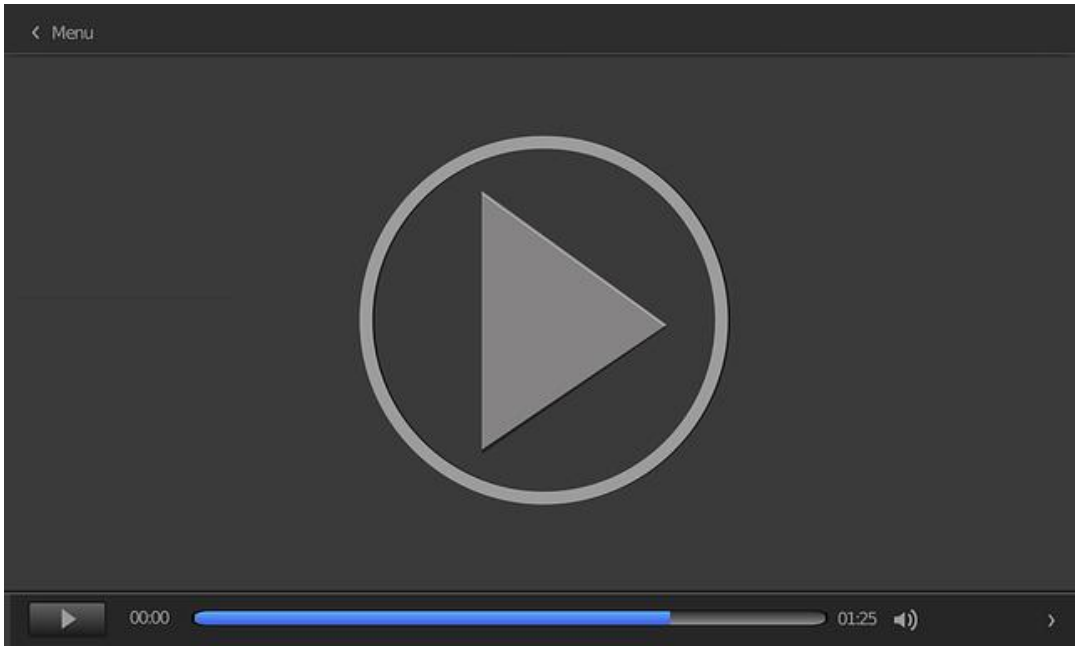
```
CREATE PROCEDURE dbo.SPSoldProducts
 @StartYear INT,
    @EndYear INT,
    @Group VARCHAR(100)
AS
   SELECT sod.SalesOrderID,
      sod.CarrierTrackingNumber,
      sod.OrderQty,
      sod.UnitPrice,
      sod.UnitPriceDiscount,
      sod.LineTotal,
      sod.ModifiedDate,
      PP.Name AS ProductName,
      pp.Color,
      pps.Name AS [Group]
   FROM Sales.SalesOrderDetail AS sod
      INNER JOIN Production.Product AS pp ON pp.ProductID = sod.ProductID
      INNER JOIN Production.ProductSubcategory AS pps ON pps.ProductSubCategoryID =
pp.ProductSubcategoryID
   WHERE YEAR(sod.ModifiedDate) BETWEEN @StartYear AND @EndYear
```

```
    AND pps.Name = @Group
GO
```

Una vez ejecutado el código, el procedimiento se ha creado. No obstante, a diferencia de las vistas, en este caso no utilizamos `SELECT`, sino `EXEC`, para ejecutar el procedimiento y obtener la información solicitada en base a los parámetros elegidos.

```
EXEC dbo.SPSoldProducts
 @StartYear = 2012,
   @EndYear = 2014,
   @Group = 'Mountain Bikes'
```

En este caso hemos incluido solo tres parámetros, pero podemos añadir tantos como consideremos oportunos. Asimismo, también disponemos de cláusulas para eliminar un procedimiento o modificarlo: `ALTER PROCEDURE` y `DROP PROCEDURE` respectivamente.



Introducción al modelo de bases de datos relacional.

Ideas clave

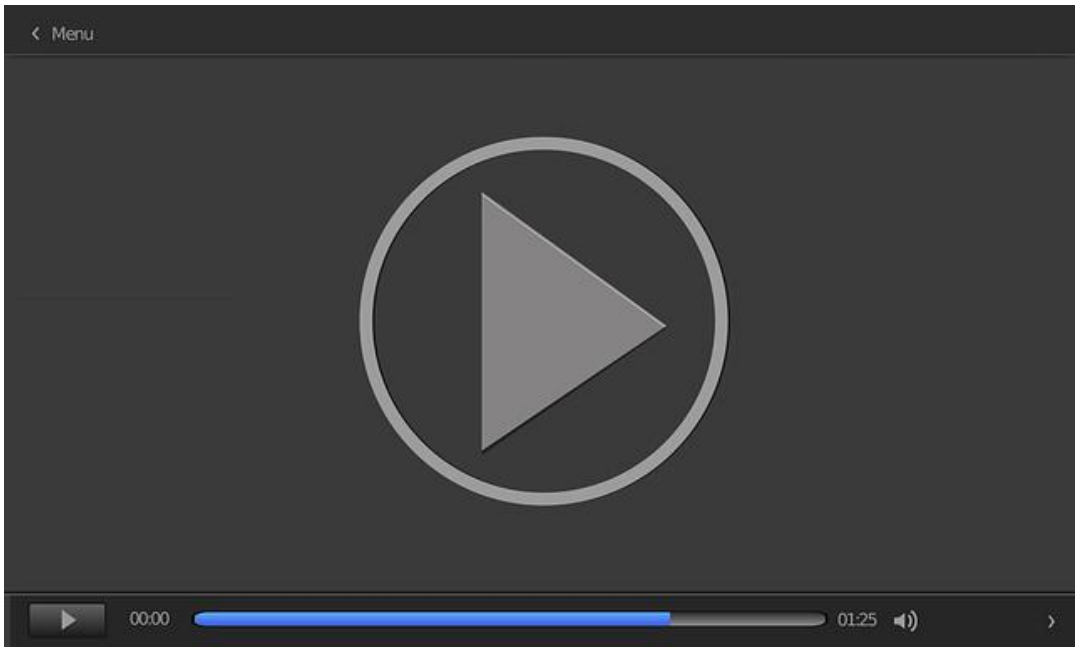
Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=e0ca4c74-9649-4800-b2f7-b16c0095a4d3

Bases de datos NoSQL

IntelDigital. (7 de mayo de 2018). *Los 4 tipos más populares de base de datos NoSQL* [Archivos de vídeo]. Recuperado de https://www.youtube.com/watch?v=bZPZIwdIu8U

Se han creado varios tipos de bases de datos NoSQL para soportar necesidades específicas y casos de uso.



Accede al vídeo:

https://www.youtube.com/embed/bZPZIwdIu8U

Data warehouse

Power Data. Página web oficial. https://www.powerdata.es/data-warehouse

Los *data warehouse* son alternativas a las bases de datos para almacenar una gran cantidad de datos y gestionar un negocio desde ellos. Como posible alternativa puede ser útil conocer qué son y cuál es su utilidad. En la sección de esta página que recomendamos encontrarás información sobre ellos.

Test

1. ¿Cuál fue el objetivo por el cual fueron creadas las bases de datos relacionales?

   A. Crear el universo de la realidad de los datos.

   B. Crear un dominio que contenga toda la información disponible.

   C. Mantener la independencia de la estructura lógica.

   D. Todas las anteriores son correctas.

2. ¿Qué elementos componen las tablas de las bases de datos relacionales?

   A. Cardinalidad.

   B. Dominio.

   C. Tuplas.

   D. Todas las anteriores.

3. Las tuplas:

   A. Forman el grado de la tabla y contiene los atributos.

   B. Forman la cardinalidad de la tabla y contiene observaciones.

   C. Forman el grado de la tabla y contiene observaciones.

   D. Forman la cardinalidad de la tabla y contiene los atributos.

4. ¿Qué necesita una clave para poder ser candidata?

   A. Debe ser un conjunto no vacío.

   B. Debe contener claves alternativas.

   C. Debe tener al menos un atributo repetido.

   D. Todas las anteriores son incorrectas.

5. ¿Para qué se utiliza la clave primaria?

A. Para crear las relaciones entre tablas.

B. Para ser la clave ajena de otra tabla.

C. Para identificar a cada tupla.

D. Todas las anteriores son correctas.

6. Un comando básico para realizar una consulta es:

A. FROM

B. AS

C. DISTINCT

D. Todas las anteriores son correctas.

7. ¿Cuál de las siguientes cláusulas se utiliza para unir dos tablas?

A. INNER JOIN.

B. ON .

C. USING.

D. Todas las respuestas anteriores son correctas.

8. ¿Cuál de las siguientes cláusulas se utiliza para unir dos tablas? Hay que tener en cuenta que necesitamos obtener únicamente las coincidencias entre ambas:

A. INNER JOIN

B. LEFT JOIN.

C. RIGHT JOIN

D. LEFT OUTER JOIN

9. Entre las funciones de agrupación están:

A. AVG, MIN y MAX.

B. ON, USING y DISTINCT .

C. IN, LIKE y ORDER BY .

D. AVG, COUNT y HAVING .

10. La cláusula LIKE sirve cuando:

A. Sabemos algún resultado previamente.

B. No conocemos toda la información, solo parte.

C. Quieres encontrar de forma exacta un resultado determinado.

D. Ninguna de las anteriores es correcta.