PROGRAMA EDUCATIVO

INFORMATICA ADMINISTRATIVA

UNIDAD DE APRENDIZAJE
BASES DE DATOS RELACIONALES

Unidad de competencia IV

LENGUAJE DE MANIPULACIÓN DE DATOS

ELABORACION
ADRIAN TRUEBA ESPINOSA

FECHA DE ELABORACIÓN
MARZO DE 2019

PRESENTACIÓN DEL CURSO

La unidad de aprendizaje “Bases de Datos Relacionales”, se imparte en el 7° semestre de la Licenciatura de Informática Administrativa. Tiene la finalidad de desarrollar las competencias en los alumnos para la implementación de bases de datos, con del modelo de bases de datos relacional y realizar consultas Manipulando datos. Para ello, es necesario sentar las bases teóricas y metodológicas para la manejar el Lenguaje de Manipulación de Datos (LMD) del modelo relacional.

CONTENIDO DEL CURSO

-Identificar los conceptos principales de bases de datos relacionales para el manejo

-Aprender los principios fundamentales para el diseño de una base de datos bajo el modelo relacional

-Aprender el proceso de normalización en el modelo relacional

-Aprender el funcionamiento de la implementación de una base de datos relacional

-Identificar los diferentes lenguajes de consulta de gráficos para **realizar manipulación de información desde dichos lenguajes**

-Aprender los principios de sistemas de bases de datos distribuidas

METAS A ALCANZAR

Que el alumno desarrolle las competencias técnicas y de especialidad para la manipulación de datos con el lenguaje SQL.

Manejo de consultas con MYSQL

OBJETIVO DEL MATERIAL DIDÁCTICO

Que el alumno conozca y adquiera las competencias para manipular datos en una base de datos relacional

METODOLOGÍA DEL CURSO

El curso se desarrollará bajo el siguiente proceso de estudio:

1. Exposición de parte del profesor mediante la utilización de este material en diapositivas.
2. Control de lecturas selectas que el profesor asignará para complementar la clase.
3. Tareas donde se investigarán temas, conceptos, procesos y métodos de los temas por ver.
4. Participación en clases
5. Prácticas de laboratorio

UTILIZACIÓN DEL MATERIAL DE DIAPOSITIVAS

El material didáctico visual es una herramienta de estudio que sirve como una guía para que el alumno repase los temas más significativos de “el modelo entidad relacion”, cabe aclarar que será un tutor el cual proporcionará las ideas generales del tema, asiendo ejercicios en el salón de clase.

UNIDAD DE COMPETENCIA IV

CONOCIMIENTO

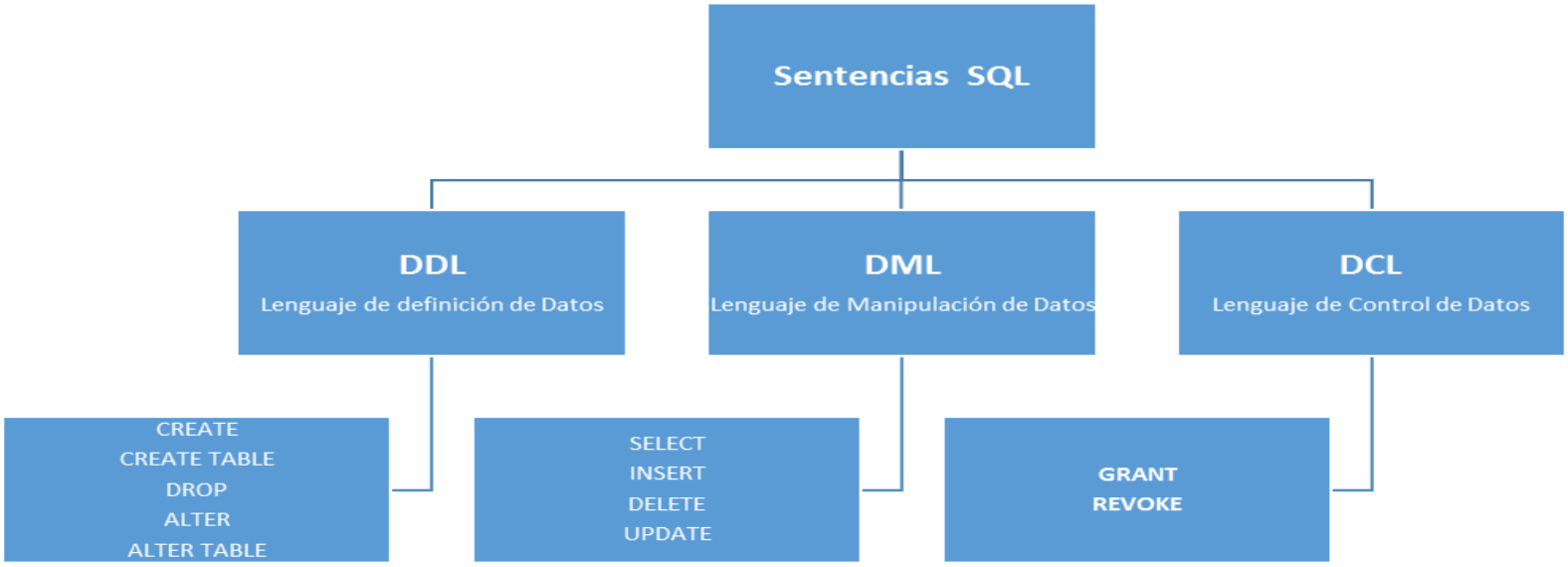
MANIPULACIÓN DE DATOS

Introducción

Aprender la implementación y funcionamiento de una base de datos relacional sirve para desarrollar las capacidades del uso de los elementos teóricos adquiridos del álgebra y cálculo relacional.

La implementación de una base de datos se hace por medio del Lenguaje de Definición de datos (LDD). La herramienta que se utiliza es un Sistema Gestor de Bases de Datos (SGBD) relacional, donde se puedan implementar bases de datos físicas, usando el Lenguaje de Estructurado de Consultas (SQL) (Structured Query Language) es un lenguaje diseñado para almacenar, manipular y recuperar datos almacenados en bases de datos relacionales.

Existen estándares para SQL. Sin embargo, el SQL que puede utilizarse en cada uno de los principales SGBD actuales viene en distintas formas. Esto se debe a dos razones: 1) el estándar SQL es bastante complejo, y no es práctico implementar el estándar completo, y 2) cada proveedor de base de datos necesita una forma de diferenciar su producto de otros.
Existen tres tipos de comandos SQL:

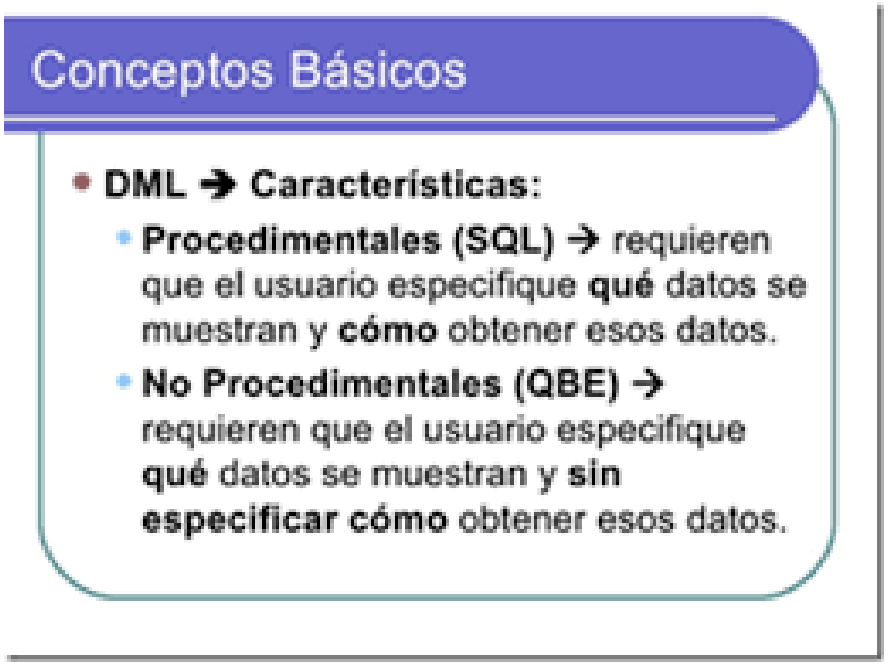


En este material se abordara el Lenguaje de Manipulación de Datos (LMD)

Lenguaje de Manipulación de Datos (Data Manipulation Language, DML) es un lenguaje proporcionado por los sistemas gestores de bases de datos, que permite a los usuarios, llevar a cabo las tareas de consulta o modificación de los datos contenidos en las Bases de Datos del Sistema Gestor de Bases de Datos.

En general a las operaciones básicas de manipulación de datos que se realizan con SQL se les denomina **operaciones CRUD** (de **C***reate,* **R***ead,* **U***pdate and* **D***elete*, o sea, *Crear, Leer, Actualizar y Borrar*, en español seria CLAB, pero no se usa). Lo verás utilizado de esta manera en muchos sitios, así que hay que aprenderse ese acrónimo.

El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional. Otros ejemplos de DML son los usados por bases de datos IMS/DL1, CODASYL.



Principales comandos de LMD

SELECT Es utilizado para consultar registros de la base de datos que satisfagan un criterio determinado

INSERT Se utiliza para cargar lotes de datos en la base de datos en una única operación

UPDATE Sirve para modificar los valores de los campos y registros especificados Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos

DELETE Es Usado para eliminar registros de una tabla

Ejemplos

INSERT

Una sentencia INSERT de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

Forma básica:

```
INSERT INTO ''tabla'' (''columna1'', [''columna2,... ''])
VALUES (''valor1'', [''valor2,...''])
```

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos).

Ejemplo:

Considere la tabla usuarios que tiene los atributos nombre, dirección teléfono email y ocupación y se requiere insertar a la tabla los siguientes datos : Roberto Jeldrez , que tiene dirección en Aldama 1100, con telefono 4886850, email : rj@gmail.com, de ocupación carpintero carpintero

```
INSERT INTO usuario (nombre, dirección, teléfono, email, ocupación
VALUES (‘Roberto Jeldrez’,’Aldama 1100’, ‘4886850’,’rj@gmail.com’,´
Carpintero’);
```

Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

```
INSERT INTO ''VALUES (''valor1'', [''valor2,...'']);
```

Ejemplo:

(Asumiendo que ‘nombre’ y ‘teléfono’ son las únicas columnas de la tabla ‘agenda’):

```
INSERT INTO agenda VALUES (‘Alma Altamirano’, ‘595473968’);
```

(Inserto valores alumno pepe en la materia matemática a la tabla cursos):

```
INSERT INTO ''cursos'' (''alumno'', ''materia'') VALUES (''pepe'', ''matemáticas');
```

UPDATE

Una sentencia UPDATE de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

Ejemplo:

```
UPDATE mi_tabla SET campo1 = 'nuevo valor campo1' WHERE campo2 = 'N';
```

Ejemplo (modifico la materia donde el alumno sea Roberto):

```
UPDATE ''curso'' SET ''materia''= ''matemáticas'' WHERE ''alumno''= ''Roberto';
```

DELETE

Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla.
Forma básica:

```
DELETE FROM 'tabla' WHERE 'columna1' = 'valor1'
```

Ejemplo:

Ejemplo (borro todos los valores de las columnas alumno y materia donde la materia sea matemáticas, de la tabla materia):

```
DELETE FROM cursada WHERE “materia” = 'matemáticas';
```

CONSULTAS

Para realizar consultas sobre las tablas de las bases de datos disponemos de la instrucción SELECT.

Con ella podemos consultar una o varias tablas. Este comando es el más versátil del lenguaje SQL.
Existen muchas cláusulas asociadas a la sentencia SELECT (GROUP BY, ORDER, HAVING, UNION).

Vamos a empezar viendo las consultas simples, basadas en una sola tabla.

Veremos cómo obtener tuplas y atributos de una tabla en el orden en que nos haga falta.

SELECT Es utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
El resultado de una consulta SELECT, devuelve **una tabla lógica o una tabla en memoria**.

La sintaxis básica de una consulta SELECT es la siguiente (los valores opcionales van entre corchetes):

```
SELECT [ ALL / DISTINC ] [ * ] / [Lista de atributos] AS
[Expresion]FROM Nombre_Tabla_Vista WHERE CondicionesORDER BY
ListaColumnas [ ASC / DESC ];
```

SELECT
Permite seleccionar las columnas que se van a mostrar y en el orden en que lo van a hacer. Simplemente es la instrucción que la base de datos interpreta como que vamos a solicitar información.

ALL / DISTINCT
ALL es el valor predeterminado, especifica que el conjunto de resultados puede incluir filas duplicadas. Por regla general nunca se utiliza.

DISTINCT especifica que el conjunto de resultados sólo puede incluir filas únicas. Es decir, si al realizar una consulta hay registros exactamente iguales que aparecen más de una vez, éstos se eliminan. Muy útil en muchas ocasiones.

Nombres de atributos

Se debe especificar una lista de nombres de atributos de la tabla o tablas que nos interesan consultar.

Se puede anteponer el nombre de la tabla al nombre del atributo, utilizando el formato *Tabla.atributo.* Además de nombres de atributos, en la lista se pueden poner constantes, expresiones aritméticas, y funciones, para obtener campos calculados de manera dinámica.

Si queremos que devuelva todos los atributos de la tabla se utiliza el comodín "*" (asterisco).

AS
Permite renombrar los atributos si lo utilizamos en la cláusula SELECT, o renombrar tablas si lo utilizamos en la cláusula FROM. Es opcional. Con ello podremos crear diversos alias de atributos y tablas.

FROM
Esta cláusula permite indicar las tablas o vistas de las cuales vamos a obtener la información.

WHERE
Especifica la **condición de filtro** de las filas devueltas. Se utiliza cuando no se desea que se devuelvan todas las filas de una tabla, sino sólo las que cumplen ciertas condiciones. Lo habitual es utilizar esta cláusula en la mayoría de las consultas.

Condiciones

Son **expresiones lógicas** a comprobar para ser usadas como un segundo filtro, que tras su aplicación devuelven para cada atributo TRUE o FALSE, en función de que se cumplan o no la condición.
Se puede utilizar cualquier expresión lógica y en ella utilizar **diversos operadores** como:

- (Mayor)

>= (Mayor o igual)
< (Menor)
<= (Menor o igual)
= (Igual)
<> o **!=** (Distinto)

IS [NOT] NULL (para comprobar si el valor del atributo es o no es nula, es decir, si contiene o no contiene algún valor)

LIKE

Para la comparación de un modelo se utiliza los caracteres comodines especiales: “%” y “_”.

Con “%”se indica que en su lugar puede ir cualquier cadena de caracteres, y con “_” se puede ir cualquier carácter individual (un solo carácter). Con la combinación de estos caracteres se pueden obtener múltiples patrones de búsqueda.

Por ejemplo:

El nombre empieza por A: Nombre LIKE ‘A%’

El nombre acaba por A:Nombre LIKE ‘%A’

El nombre contiene la letra A: Nombre LIKE ‘%A%’

El nombre empieza por A y después contiene un solo carácter cualquiera: Nombre LIKE ‘A_’

El nombre empieza una A, después cualquier carácter, luego una E y al final cualquier cadena de caracteres: Nombre LIKE ‘A_E%’

BETWEEN: para un intervalo de valores.
Por ejemplo:

Usuarios entre el 30 y el 100: Usuario BETWEEN 30 AND 100
Usuarios nacidos entre 1970 y 1979: FechaNac BETWEEN ‘1970-01-01’ AND ‘1979-12-31’

IN( ): para especificar una relación de valores concretos. Por ejemplo: Préstamo de usuarios 10, 15, 30 y 75: Usuario IN(10, 15, 30, 75)

Es posible combinar varias condiciones simples de los operadores anteriores utilizando los operadores lógicos OR, AND y NOT, así como el uso de paréntesis para controlar la prioridad de los operadores (como en matemáticas).

ORDER BY

Define el orden de las tuplas del conjunto de resultados. Se especifica el atributo o atributos (separados por comas) por los cuales queremos ordenar los resultados.

ASC / DESC

ASC es el valor predeterminado, especifica que el atributo indicado en la cláusula **ORDER BY** se ordenará de menor a mayor. Si por el contrario se especifica **DESC** se ordenará de mayor a menor.

Algunos ejemplos

Mostrar todos los datos de los empleados de una empresa:

```
SELECT * FROM empleados;
```

Mostrar apellido, ciudad y región (LastName, city, region) de los empleados de USA (nótese el uso de AS para darle el nombre en español a los campos devueltos):

```
SELECT E.LastName AS Apellido, City AS Ciudad, Region FROM Employees
AS EWHERE Country = 'USA';
```

Mostrar los clientes que no sabemos a qué región pertenecen (o sea, que no tienen asociada ninguna región):

```
SELECT * FROM Clientes WHERE Region IS NULL;
```

Mostrar las distintas regiones de las que tenemos algún cliente, accediendo sólo a la tabla de clientes:

```
SELECT DISTINCT Region FROM Clientes WHERE Region IS NOT NULL;
```

Mostrar los clientes que pertenecen a las regiones CA, MT o WA, ordenados por región ascendentemente y por nombre descendentemente.

```
SELECT * FROM Clientes WHERE Region IN('CA', 'MT', 'WA')ORDER BY
Region, CompañiaNom DESC;
```

Mostrar los clientes cuyo nombre empieza por la letra "W":

```
SELECT * FROM Clientes WHERE CompañiaNom LIKE 'W%';
```

Mostrar los empleados cuyo código está entre el 2 y el 9:

```
SELECT * FROM Usuario WHERE codigo BETWEEN 2 AND 9;
```

Mostrar los clientes cuya dirección contenga “ki”:

```
SELECT * FROM Clientes WHERE Direccion LIKE ‘%ki%’;
```

Mostrar las Ventas del producto 65 con cantidades entre 5 y 10, o que no tengan descuento:

```
SELECT * FROM [Ventas] WHERE (ProductoID = 65 AND Cantidad BETWEEN
5 AND 10) OR SinDescuento = 0;
```

CONSULTAS A MULTIPLES TABLAS

La consulta a múltiples tablas al mismo tiempo, en las bases de datos a veces se puede volver un poco complicada, sobre todo con el tema de las tablas relacionales. Este tipo de consultas son las más usadas por las empresas ya que se requiere cruzar datos para poder obtener información.

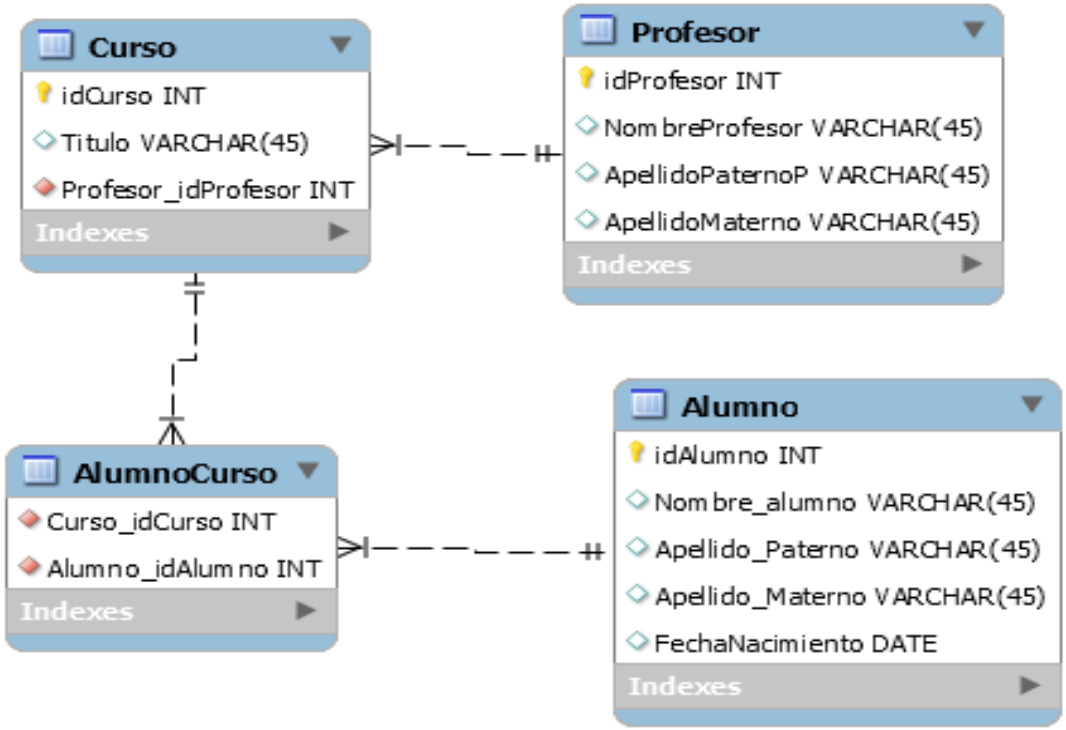
Las consultas pueden ser entre dos tablas, tres tablas o más. En estos casos es recomendable usar las relaciones entre tablas y las sentencias JOIN de SQL.

Con Relaciones

El modo de relacionar registros entre tablas es por tanto mediante referencias, para lo cual se usan los identificadores definidos como claves primarias y foráneas.

Supongamos una academia donde se imparten clases, habrá cursos, profesores y alumnos.

En una base de datos, se diseña una tabla para cada entidad, es decir, para alumnos, profesores y cursos. Veamos cómo se relacionan entre si



Ejemplos

Para obtener la consulta con **dos tablas** se puede plantear lo siguiente que cursos imparte el profesor se usa la siguiente sentencia

```
SELECT Curso.Titulo, Profesor.NombreProfesor,
Profesor.ApellidoPaternoP , Profesor.ApellidoMAterno
FROM Curso,Profesor
WHERE
Curso.Profesor_idProfesor = Profesor.IdProfesor;
```

Observe como que las relaciones que se dan entre los atributos llave primaria y llave foránea son los que permiten una consulta

Para **tres tablas** se puede plantear la siguiente consulta

Se desea saber qué curso ha cursado cada alumno.

Se Plantea la siguiente sentencia

```
SELECT Curso.Titulo, Alumno.Nombre_alumno, Alumno.Apellido_Paterno,
Alumno.Apellido_Materno
FROM
    Curso,Alumno,AlumnoCurso
WHERE
Curso.Profesor_idProfesor = Profesor.IdProfesor
AND Curso.idCurso = AlumnoCurso.Curso_idCurso
AND AlumnoCurso.Alumno_idAlumno = Alumno.idAlumno;
```

Con JOIN

El INNER JOIN cláusula MySQL
El INNER JOIN sirve para juntar dos o más tablas
Tiene la siguiente semántica

```
SELECT Selecciona lista de tributos
FROM t1 (tabla 1)
INNER JOIN t2 (tabla 2) ON join_condition1
INNER JOIN t3 (tabla 3) ON join_condition2;
```

En esta sintaxis:
Primero, especifique la tabla principal que aparece en la FROM cláusula (t1).
En segundo lugar, especificar la tabla que se combina con la tabla principal, que aparece en la INNER JOIN cláusula (t2, t3, ...).
Tercero, especifique una condición de unión después de la ON palabra clave de la INNER JOIN cláusula. La condición de unión especifica la regla para hacer coincidir filas entre la tabla principal y la tabla que aparece en la INNER JOIN cláusula.

Considerando la base de datos de alumno cursos el planteamiento seria el siguiente

```
SELECT Curso.Titulo, Profesor.NombreProfesor,
Profesor.ApellidoPaternoP , Profesor.ApellidoMAterno
FROM Curso
INNER JOIN Profesor ON
Curso.Profesor_idProfesor=Profesor.IdProfesor;
```

Con tres tablas seria

```
SELECT Curso.Titulo, Alumno.Nombre_alumno,
Alumno.Apellido_Paterno, Alumno.Apellido_Materno
FROM Curso
INNER JOIN AlumnoCurso ON AlumnoCurso.Alumno_idAlumno =
Alumno.idAlumno
INNER JOIN Profesor ON Curso.Profesor_idProfesor =
Profesor.IdProfesor
```

A estas consulta se pueden agregar los filtros que se vieron en la practica 3 para hacer consultas más específicas.

Esto es intuitivo y se puede plantear para más de tres tablas basta con hacer la igualdad de las relaciones que hay entre las tablas.

Para practicar plantee los siguientes ejercicios

Ejercicio 1
Construya una consulta que devuelva los cursos en que se ha matriculado el alumno con identificador 1

Modifique la anterior consulta para que devuelva los nombres y apellidos de los alumnos, y los cursos en que se han matriculado, tales que el nombre de pila del alumno contenga una E.

Ejercicio 2
¿Cuantos cursos imparte cada profesor? Construya una consulta que responda a esta cuestión de modo que el resultado muestre el nombre completo del profesor acompañado del número de cursos que imparte.

Ejercicio 3
¿Cuántos alumnos hay matriculados en cada uno de los cursos? Construya una consulta que responda a esta cuestión de modo que el resultado muestre el título del curso acompañado del número de alumnos matriculados.

Modifique la anterior consulta de modo que muestre aquellos cursos que el número de alumnos matriculados sea exactamente de dos alumnos.

Ejercicio 4

Obtenga el nombre de usuario, ocupación, numero de ejemplar prestado y fecha de entrega

Ejercicio 4
Obtenga el nombre de usuario, ocupación, numero de ejemplar prestado y fecha de entrega y ordénelos por fecha de entrega

Ejercicio 5
Si ahora a usted le pidiesen que adaptara la BD, que consta de las tres tablas presentadas en esta lección, a la siguiente necesidad: A todo alumno se le asignara un profesor que lo tutele. ¿Que cambios realizaría en la BD?

LECTURAS RECOMENDADAS

INTRODUCCION A LAS BASES DE DATOS RELACIONALES MARIA ANTONIA NEVADO CABELLO , VISION NET, 2010. ISBN 9788498868098

BASES DE DATOS RELACIONALES Y MODELADO DE DATOS. JOSE MANUEL PIÑEIRO GOMEZ , S.A. EDICIONES PARANINFO, 2013. ISBN 9788428333566

BASES DE DATOS RELACIONALES: FUNDAMENTOS Y DISEÑO LOGICO VV.AA. , UNIVERSIDAD PONTIFICIA COMILLAS, 2005. ISBN 9788484681724

Gracias por su atención

contacto
atruebae@hotmail.com