UNIDAD 2: Manejo y Gestión de Bases de Datos

CONTENIDOS	
1. Conocer y clasificar las bases de datos. Concepto de Base de Datos, Clasificación de Base de Datos. Diseño Lógico y Normalización. Concepto de Diseño Lógico. Normalización: 1º, 2º y 3º Forma Normal. Diferencias entre diseño lógico y diseño físico, casos prácticos de diseño físico en Access y MySQL;	1 6
2. SQL (Lenguaje de Consultas Estructurado). Sintaxis SQL: Lenguaje de Definición de Datos (DDL de SQL) y Lenguaje de Manipulación de Datos (DML de SQL).	11 12
3. SQL – DML: Sentencia SELECT: básica, con proyección, con restricción y mixta; distintos casos sobre una tabla o más tablas. Utilización de cláusulas para ordenamiento y agrupamiento. Sentencia INSERT, Sentencia UPDATE y Sentencia DELETE.	16 16
4. SQL – DDL: Sentencias CREATE, ALTER, RENAME y DROP aplicadas a Tablas y Vistas.	16

ACTIVIDADES	
Unidad 2	Tareas
1/6	Lectura, trabajos, prácticas propuestas en el Website de la Materia, Sección Actividades.
1/6	Autoevaluación Nº 2 (en Website, Sección Alumnos)

BIBLIOGRAFÍA para el Alumno

1. Contenidos Unidad 2
2. SAROKA, Raúl Horacio**, Sistemas de Información en la Era Digital** (Capítulo 2, ebook, Fundación OSDE 2002) (*)
3. NOUSSAN LETTRY, Laura y otros, **Secuencia Didáctica de Comandos SQL - Ejemplos y Resultados** (Cátedra de Gestión de Datos, 3º Año, Ingeniería en Sistemas de Información UTN – FRM) (*)
4. BrowserSQL versión 3.1 y su Tutorial (*)

(*): En el Website de la materia.

(**): Libro impreso.

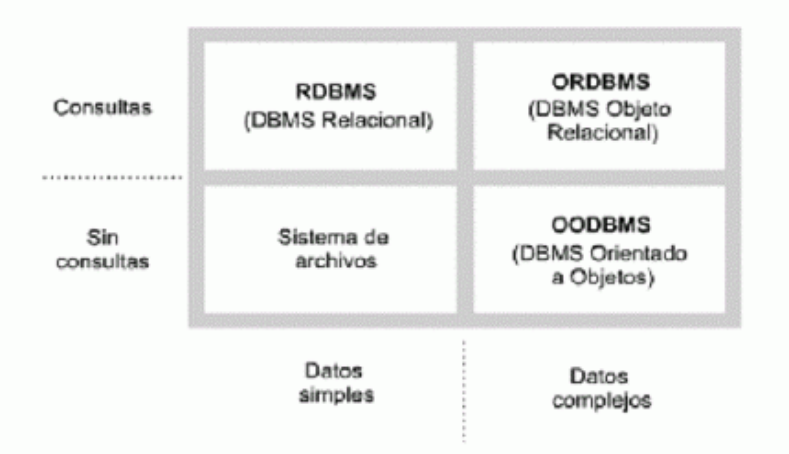
CONTENIDOS

1. Conocer y clasificar las bases de datos.

CONCEPTO DE BASE DE DATOS

En primer lugar lo fundamental es entender qué es una base de datos y cómo funciona. Nos interesa este concepto porque si bien el programa trata las Bases de Datos en Visual Basic los conceptos teóricos y prácticos sobre este tema en particular sirven incluso para su utilización y aprovechamiento con otros lenguajes de programación y de diseño de software que permiten manipular datos.

El fin básico de una Base de Datos tiene es ***almacenar información que tiene valor desde algún punto de vista para su propietario.*** La siguiente imagen indica la diferencia entre un Sistema de Base de Datos y otros Sistemas de almacenamiento de información. .



Los Sistemas de Bases de Datos han tenido un gran auge en los últimos años y ello se debe a sus ventajas en relación a los sistemas que se basan en Archivos como el Excel, por ejemplo. Las ventajas se resumen y explican seguidamente:

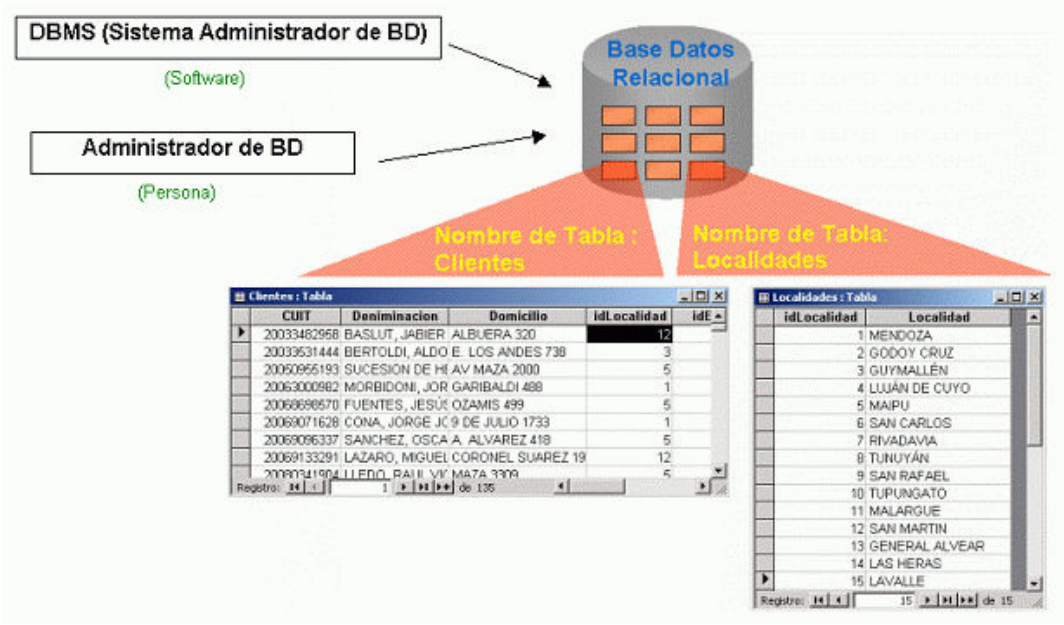
- las Bases de Datos nos permiten compartir datos e información;
- entre distintas aplicaciones y distintos tipos de usuarios;
- haciendo que la información esté disponibles en forma rápida;
- y fundamentalmente restringida a quienes debe y pueden tener acceso a ella.

Aquí no se trata de que un usuario decide compartir sus datos con otros sino más bien de que existe una ***política empresaria*** que decide qué datos estarán accesibles, en qué forma y en qué nivel de acceso a los diferentes usuarios.

Cuando realmente existe dicha política en una organización la misma se ve reflejada en la práctica en la existencia de una ***persona, el Administrador de Bases de Datos o DBA***, que en base a esa política o visión de negocio configura la Base de Datos en vista a sus objetivos.

También existe un administrador por software que se llama habitualmente ***Sistema Administrador de Bases de Datos o DBMS***.

En las bases de datos tenemos una serie de archivos relacionados entre sí que se llaman tablas. Además estos sistemas permiten establecer un orden de accesos a la información teniendo en cuenta la jerarquía del personal de la empresa. De esta forma la seguridad es fundamental y con ello se consigue compartir los datos pero también restringir su acceso. Por ejemplo, un empleado de almacenes debe poder consultar los datos de los artículos y mercaderías pero no debería por qué tener acceso a los datos de los sueldos del personal. Por otro lado la información puede estar disponible en forma muy rápida mediante la ejecución de consultas sobre la base de datos. Todo lo referente al manejo de archivos está gestionado por el Sistema Administrador de la Base de Datos (software) y por el Administrador de la Base de Datos (un ingeniero, analista de sistemas).



Estos sistemas de Bases de Datos tienen éxito en la medida que estén bien diseñados y en la medida que el Sistema se adapte a las necesidades de la empresa. Por ejemplo, en empresas grandes una Base de Datos como Access, por poner un ejemplo, no resulta del todo óptimo debido a que se necesita un sistema más potente que pueda manejar en manera óptima la gran cantidad de operaciones diarias que estas empresas requieren. Además los costos aumentan a medida que el Sistema de Base de Datos es más potente y estos costos no sólo tienen que ver con la compra del sistema sino con su mantenimiento.

Un aspecto fundamental para conseguir concomitantemente los objetivos de utilidad, rapidez, seguridad e integridad de los datos es mediante un buen diseño lógico mediante la ***normalización*** de las tablas.

CLASIFICACIÓN

Podemos hacer una clasificación de Sistemas de Bases de Datos teniendo en cuenta su concepción intrínseca; de esta forma podemos decir que existen los tres tipos vistos en la primera imagen:

- RDBMS: Sistema Administrador de Bases de Datos Relacional o Bases de Datos Relacionales;
- ORDBMS: Sistema Administrador de Bases de Datos Objeto Relacional o Bases de Datos Objeto Relacionales;
- OODBMS: Sistema Administrador de Bases de Datos Orientados a Objeto o Bases de Datos Orientadas a Objeto.

Los sistemas de bases de datos que actualmente existen son los Relaciones u Objeto Relacionales, ya que los orientados a objeto no han tenido éxito. Así tenemos a DB2 (de IBM), Oracle, MySQL, Postgress, Access, etc.

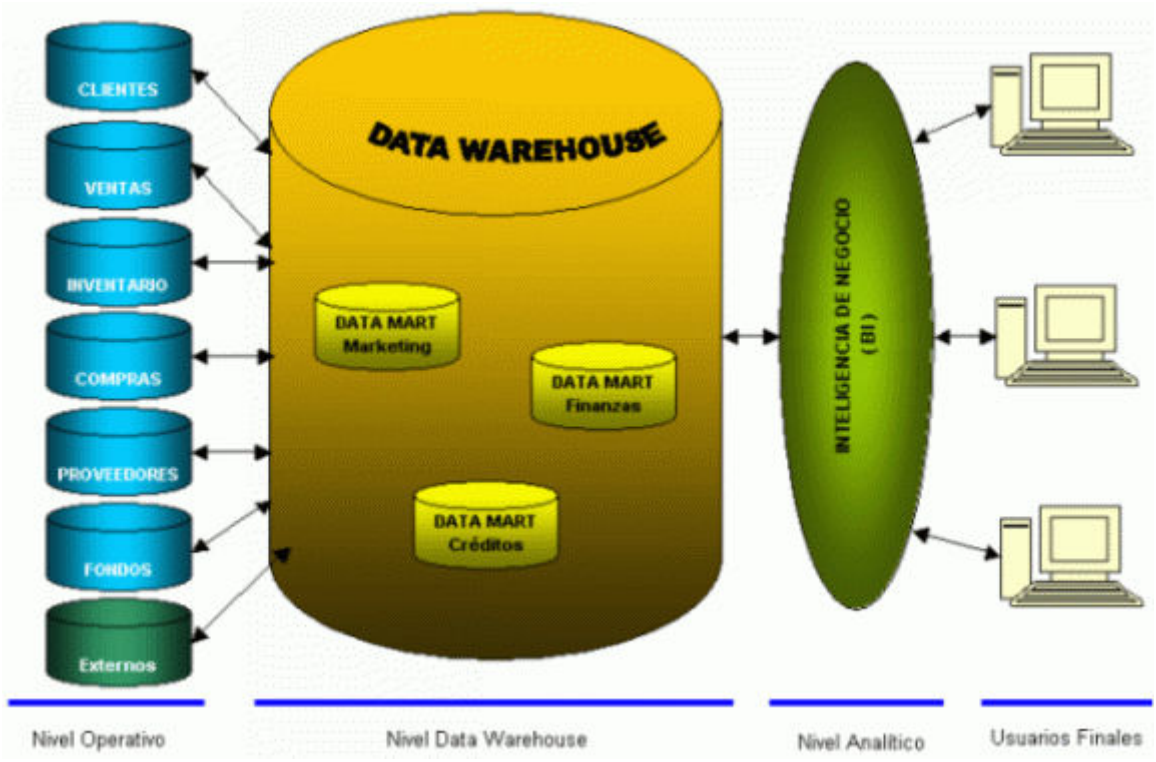
Están basados en la lógica del álgebra relacional y por lo tanto, cualquier diseño físico corresponderá a un previo diseño lógico que deberá ser consistente con su sustento teórico. Para ello es necesario que se cumplan ciertos requisitos:

- El diseño lógico es siempre independiente del tipo de aplicación, esto quiere decir que no interesa si la base de datos es operacional o es para apoyo a la toma de decisiones. Inclusive no debería haber mayores discrepancias entre un sistema comercial u otro.
- Las tablas que forman la Base de Datos deben estar normalizadas ya que es el requisito necesario para que puedan aplicarse las operaciones relacionales.
- Como consecuencia de ello deben poder establecerse reglas de integridad. Por integridad debe entenderse no sólo mantener la consistencia de los datos (el pegamento, si se quiere entre las tablas), sino también definir el significado de las tablas y de la base de datos en su totalidad.

La importancia del modelo lógico es tal, que los cambios en el sistema físico deberían siempre amoldarse a éste. En realidad el modelo físico puede cambiar sin cambiarse el modelo lógico cuando cuestiones de almacenamiento y eficiencia así lo indiquen.

Otra clasificación tiene en cuenta el uso del Sistema de Bases de Datos:

- Sistemas Operacionales;
- Sistemas para el apoyo a la toma de decisiones.



Los Sistemas de Bases de Datos ***Operacionales*** son aquellos que sirven de base para llevar las operaciones diarias de las empresas y organizaciones. En cambio los Sistemas de ***Apoyo a la Toma de Decisiones*** son sistemas que ayudan en el análisis de información de negocios.

Las bases de datos para la toma de decisiones tienen ciertas características propias. En general puede decirse que son prácticamente de sólo lectura. Cada tanto se hacen actualizaciones pero que consisten básicamente en INSERTs (casi nunca DELETEs y menos aún UPDATEs). Esto se debe a que se basan e las bases de datos operacionales que cada tanto son almacenadas en un repositorio para no perder la información histórica. Además suelen ser bases de datos grandes.

Los sistemas operacionales en cambio suelen ser bases de datos no tan grandes puesto que las cuestiones relativas al rendimiento son fundamentales, las cargas de datos (que involucran INSERTs, UPDATEs y DELETEs o altas, bajas y modificaciones) son generalmente predecibles y de gran volumen, de allí la importancia de que no estén sobrecargadas con datos de ejercicios anteriores; motivo por el cual suele guardarse sólo los datos del ejercicio actual y los de los ejercicios anteriores enviarse a un repositorio, que es en definitiva otra base de datos, fuente juntamente con bases de datos externas de la información que se utilizará en los niveles intermedios y superiores de la organización para la toma de decisiones.

Así para los Sistemas de Apoyo a la Toma de Decisiones tenemos bases de datos especiales como son los ***Data Warehouse***, los ***Data Marts***, etc.

Un ***Data Warehouse*** es una base de datos orientada a un tema, integrado y no volátil (quiere decir que una vez insertados los datos no pueden ser cambiados aunque sí borrados). Surgieron para prestar información específica sobre un tema teniendo una fuente única de información y para no afectar a la base de datos operacional.

En general un Data Warehouse está pensado para proporcionar una fuente de datos única a todas las actividades de apoyo a la toma de decisiones. En cambio, los ***Data Marts*** consideran un subconjunto, es decir son almacenes de datos especializados orientados a un tema, integrado, no volátil pero para apoyar a un subconjunto específico de decisiones de administración.

Asimismo cuentan con un ***Nivel Analítico*** que permite sacar conclusiones para la toma de decisiones.

En resumen un ***Sistema BI*** (Sistema de Inteligencia de Negocio) está conformado de la siguiente manera:

Los Sistemas BI permiten capturar datos de los sistemas de nivel operativo (OLTP, ERP, externos; sistemas que llevan a cabo decisiones repetitivas y rutinarias) para construir un repositorio de datos llamado Data Warehouse. Este almacén puede estar formado por diferentes Data Marts o almacenes de datos para distintos temas específicos. Las distintas metodologías de análisis de datos (OLAP, Minería de datos, etc.) brindan a los usuarios finales la información necesaria que requieren para la toma de decisiones no estructuradas.

DISEÑO LÓGICO Y NORMALIZACIÓN - Repaso de conceptos fundamentales

Diseño Lógico o Conceptual de una Base de Datos

El diseño conceptual tiende a identificar las entidades y relaciones existentes, aplicación de las reglas de normalización, de nombres inequívocos para los atributos de las entidades, establecimiento de reglas de integridad, etc.

Conceptualmente la Base de Datos se diseña por medio de un DER (Diagrama de Entidad-Relación), que se lleva a un MER (Modelo de Entidad-Relación), y un Diccionario de Datos Elemental (se especifica en el modelo el detalle conceptual de las entidades y relaciones; es decir cuál es su función u objetivo; Los elementos de datos o atributos necesarios y las reglas de integridad básicas para la base de datos).
Además, hay que tener en cuenta, que al normalizar el diseño, necesariamente las relaciones van a establecerse por medio de tablas y vínculos entre éstas.

Es muy importante no olvidar que el diseño lógico o conceptual de una Base de Datos debe ser independiente del diseño físico. Es decir, el diseño lógico debería poder aplicarse para cualquier DBMS (Oracle, DB2, MySQL, Access, etc.) sin alteraciones en su lógica, mientras que el diseño físico; es decir cómo se va a implementar este diseño lógico sobre el hardware puede ser distinto según las funcionalidades de cada DBMS o incluso variar en el tiempo en una misma organización dependiendo de factores como acceso a la base de datos, costo de mantenimiento, velocidad de accesos, etc. Por ejemplo la base de datos podría estar alojada en tres servidores distintos dependiendo de la performance que se necesite, pero el diseño lógico debería seguir siendo el mismo.

Normalización

Las tablas que forman la Base de Datos deben estar normalizadas ya que es el requisito necesario para que pueda aplicarse el álgebra relacional y sus operaciones.

La normalización consiste en establecer lo que se conoce como reglas de integridad. De esta forma se pueden aplicar las operaciones relacionales obteniendo los resultados que prevé el álgebra relacional y no cualquier otro. Algunas operaciones relacionales se efectúan sobre una misma tabla y otras sobre un conjunto. Por ejemplo, sobre una misma tabla: seleccionar, proyectar , restringir; sobre conjuntos (sobre más de una tabla): unión, intersección y diferencia entre conjuntos. En SQL a estas operaciones se las halla definidas en lo que se conoce como DML o Lenguaje de Manipulación de Datos.

Por integridad debería entenderse que no sólo implica mantener la consistencia de los datos (el pegamento, si se quiere entre las tablas), sino también definir el significado de las tablas y de la base de datos en su totalidad; es decir su diseño lógico. En SQL estas definiciones están representadas por lo que se conoce como DDL o Lenguaje de Definición de Datos y permite crear las tablas con sus atributos, tipos de datos y restricciones de integridad.

Con las tres primeras formas normales se alcanzarían estos objetivos. Estas tres formas normales fueron propuestas por Edgard Frank Codd y se basan el cálculo relacional.
Aquí debemos aclarar cuál es la ***correspondencia*** entre la terminología relacional y la utilizada en los distintos sistemas comerciales, ya que no es la misma:

- Relación = tabla
- Tupla = fila o registro
- Atributo = columna o campo
- Clave = llave o código de identificación
- Clave Primaria = es la superclave. [PRIMARY KEY en SQL]
- Clave Ajena = es una clave externa o clave foránea [FOREIGN KEY en SQL]

Primera forma Normal (1FN): siguiendo a J.C. Date, una tabla está en primera forma normal sí y sólo si es isomorfa a alguna relación. Lo cual significa que la tabla debe cumplir los siguientes requisitos:

1. No hay orden de arriba-a-abajo en las filas.
2. No hay orden de izquierda-a-derecha en las columnas.
3. No hay filas duplicadas.
4. Cada intersección de fila-y-columna contiene exactamente un valor del dominio aplicable (y nada más).
5. Todas las columnas son regulares [es decir, las filas no tienen componentes como IDs de fila, IDs de objeto, o timestamps ocultos].

Cabe hacer las siguientes aclaraciones:

- una tabla no estaría en 1ª Forma Normal si no tiene una clave primaria puesto que así no se podría asegurar que no aparecieran filas o tuplas duplicadas.
- en el punto 4 existen discrepancias entre distintos autores. Date considera que no se cumpliría la 1FN si se admiten los nulos (debe entenderse por nulo al campo vacío). Otros autores consideran que mientras la columna no sea una clave se pueden admitir nulos, salvo en la clave primaria.
- el punto 5 significa que una columna no puede tener múltiples valores porque cada valor que tome un atributo debe ser un dato atómico; es decir, a cada valor de X en la relación le pertenece un valor de Y, entonces a cada valor de Y le pertenece un valor de X. Por eso mismo Date aclara que no se permiten los IDs de objetos, filas, etc puesto que estos IDs son en realidad punteros a otra tabla; con lo cual no se respeta la atomicidad.
- respecto a las claves conviene aclarar lo siguiente:
  - Una ***clave primaria*** es aquella columna (pueden ser también dos columnas o más) que identifica únicamente a esa fila. La clave primaria es un identificador, y por lo tanto, los valores que debe tomar deben ser únicos para cada fila y no se admite el valor nulo. Es una costumbre colocar como primera columna a la clave primaria, pero ello no es necesario. Además muchos RDBMS suelen permitir la opción de declarar a esta clave como autonumérica.
  - Una ***clave foránea*** es aquella columna que existiendo como dependiente en una tabla, es a su vez clave primaria en otra tabla; es decir, que es lo que vincula a ambas tablas.
  - Asimismo una clave es ***compuesta*** cuando está formada por más de una columna.

Segunda forma Normal (2FN): Una relación está en 2FN si está en 1FN y si los atributos que no forman parte de ninguna clave dependen de forma completa de la clave principal.

El siguiente es un ejemplo tomado del libro del autor citado. La tabla PRIMERA contiene información de los proveedores (V) y las partes que los proveedores han enviado (P). Esta tabla está en 1FN ya que no se repiten las filas siendo su clave primaria compuesta (V#,P#)

PRIMERA

V#	STATUS	CIUDAD	P#	CANT
V1	20	Londres	P1	300
V1	20	Londres	P2	200
V1	20	Londres	P3	400
V1	20	Londres	P4	200
V1	20	Londres	P5	100
V1	20	Londres	P6	100
V2	10	París	P1	300
V2	10	París	P2	400
V3	10	París	P2	200
V4	20	Londres	P2	200
V4	20	Londres	P4	300
V4	20	Londres	P5	400

SEGUNDA

V#	STATUS	CIUDAD
V1	20	Londres
V2	10	París
V3	10	París
V4	20	Londres
V5	30	Atenas

VP

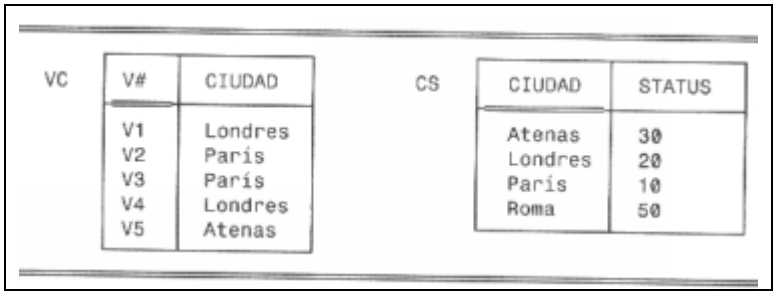
V#	P#	CANT
V1	P1	300
V1	P2	200
V1	P3	400
V1	P4	200
V1	P5	100
V1	P6	100
V2	P1	300
V2	P2	400
V3	P2	200
V4	P2	200
V4	P4	300
V4	P5	400

Pero PRIMERA No está en 2FN porque el campo Status no depende de la clave primaria sino de la ciudad y la ciudad sólo depende de la columna proveedor (V) pero no de la clave primaria. Esta situación trae consecuencias nefastas con las operaciones INSERT, UPDATE Y DELETE. ¿Por qué? Simplemente si consideramos la operación DELETE para el proveedor 3 (V3) eliminaremos también la información relativa sobre la parte que envío (P2). Esta situación se soluciona reduciendo PRIMERA a dos tablas que estén en 2FN: las tablas SEGUNDA y VP.

La clave primaria de SEGUNDA es V# y de la nueva tabla VP es (V#,p#).

Tercera forma Normal (3FN): Una relación se encuentra en 3FN si está en 2FN y cada atributo que no forma parte de ninguna clave, depende directamente y no transitivamente, de la clave primaria.

La tabla SEGUNDA está en 2FN pero no está en 3FN para llevarla la reducimos a una serie de tablas que estén en 3FN, lo que se muestra en la siguiente imagen:



En la tabla SEGUNDA la ciudad depende del proveedor (V) pero el estado depende de la ciudad, con lo cual se da un relación transitiva y no se cumple entonces con la 3FN.

Hay que notar que la tabla VP al estar en 2FN también está en 3FN ya que la cantidad depende directamente de la clave primaria que identifica a un envío en particular. La tabla VC relaciona los proveedores con la ciudad en la que se encuentran y la tabla CS muestra el estado para cada ciudad.
El modelo ha quedado reducido a tres tablas: VP, VC Y CS. Puede apreciarse a simple vista que para VP la clave primaria es (V#,P#). La tabla VC tiene a V# como clave primaria y la tabla CS tiene a CIUDAD como clave primaria.

Asimismo VP tiene una clave primaria (V#,p#) y dos claves foráneas: V# y P#. V# hace referencia a la tabla VC en la cual V# es primaria y la clave P# hace referencia a tabla P de partes, que aparece en los diagramas pero cuya clave primaria es P#. La tabla VC tiene como clave foránea a CIUDAD que se vincula con la tabla CS donde la clave primaria es CIUDAD.

Diferencias entre diseño lógico y diseño físico

El diseño lógico de un sistema de base de datos es más bien permanente en el tiempo, en cambio el diseño físico es variable ya que debe considerar no sólo el hardware sobre el que se va a instalar, sino también cambia según el DBMS que consideremos para utilizar. Por ejemplo Oracle, DB2, SQL Server (edición Empresarial) y MySQL permiten el particionamiento de tablas, la fragmentación de tablas y replicación.

Además los sistemas más modernos y potentes permiten crear sistemas de bases de datos distribuidos. Estos sistemas son distintos entre sí y también existe gran cantidad de software intermedio, conocido como Middleware, que permite que se puedan compartir datos e información de distintos sistemas de bases de datos, cuando el sistema es distribuido. En teoría, cuando un sistema es distribuido, podríamos encontrar en un computador un DBMS de DB2, en otro ORACLE, etc.

Casos prácticos de diseño físico en Access y MySQL

Los casos prácticos son desarrollados en el punto 4 donde se creará la base de datos, tanto en Access como en MySQL siguiendo el diseño lógico.

2. SQL (Lenguaje de Consultas Estructurado).

SQL. Concepto

SQL es un lenguaje de alto nivel creado para el tratamiento de datos almacenados en bases de datos relacionales.

Es un lenguaje de tipo declarativo; es decir, no es procedural o imperativo, y permite trabajar con datos a nivel de conjunto, a diferencia de los lenguajes imperativos que trabajan en torno al registro individual y que requieren de lógica procedural para el tratamiento de los datos.

Está basado en álgebra relacional y cálculo relacional orientado a tuplas.

Sus instrucciones básicas se pueden dividir en dos grupos:

1. DDL (Data Definition Language): conjunto de instrucciones que permiten definir tablas, índices, etc.
2. DML (Data Manipulation Language): se utilizan para actualizar información en la base de datos (insertar, eliminar o modificar filas en tablas) y para extraer información de la misma mediante consultas.

Además de estos grupos existen otros, cuyas instrucciones permiten realizar otras operaciones aplicadas al DBMS y/o a sus bases de datos.

Sintaxis SQL

Toda la definición de la sintaxis y uso de SQL está incluida en estándares que van actualizándose periódicamente. Actualmente el último estándar aceptado es SQL: 2003.

Muchos de los RDBMSs del mercado satisfacen gran parte de estándares anteriores (SQL/92, SQL/99) y otros este último estándar. Pero además, varios de los RDBMSs líderes en el mercado tales como DB2 de IBM, ORACLE, SQL Server de Microsoft y otros, tienen funcionalidades que van más allá de los estándares y, en base a estas funcionalidades, muchas veces se elaboran nuevos estándares.

Los ejemplos desarrollados en este material serán realizados sobre la Base de Datos Empresa (en Access) utilizando el programa BrowserSQL, para lo cual deberá descargarse el material indicado en el SiteWeb.

También se crearán las tablas de la Base de Datos Universidad, partiendo del diseño lógico con el MER yla descripción lógica de las tablas y llegando a un diseño físico con Access y con MySQL.

Los ejemplos y trabajos con MySQL serán desarrollados con posterioridad a Access, al finalizar el punto 4, de modo que se pueda lograr una comprensión real de la diferencia entre el diseño lógico y el diseño físico, en este caso utilizando dos DBMS relacionales distintos.

Lenguaje de Definición de Datos (DDL de SQL)

Las sentencias que encontramos aquí tienen por objetivo poder definir datos como Tablas, Vistas, índices y snapshots.

Las sentencias básicas que veremos son: CREATE, ALTER, RENAME y DROP.

Lenguaje de Manipulación de Datos (DML de SQL)

Las sentencias en este caso tienen por fin insertar, modificar o eliminar registros de tablas y vistas, así como la obtención de resultados mediante consultas.

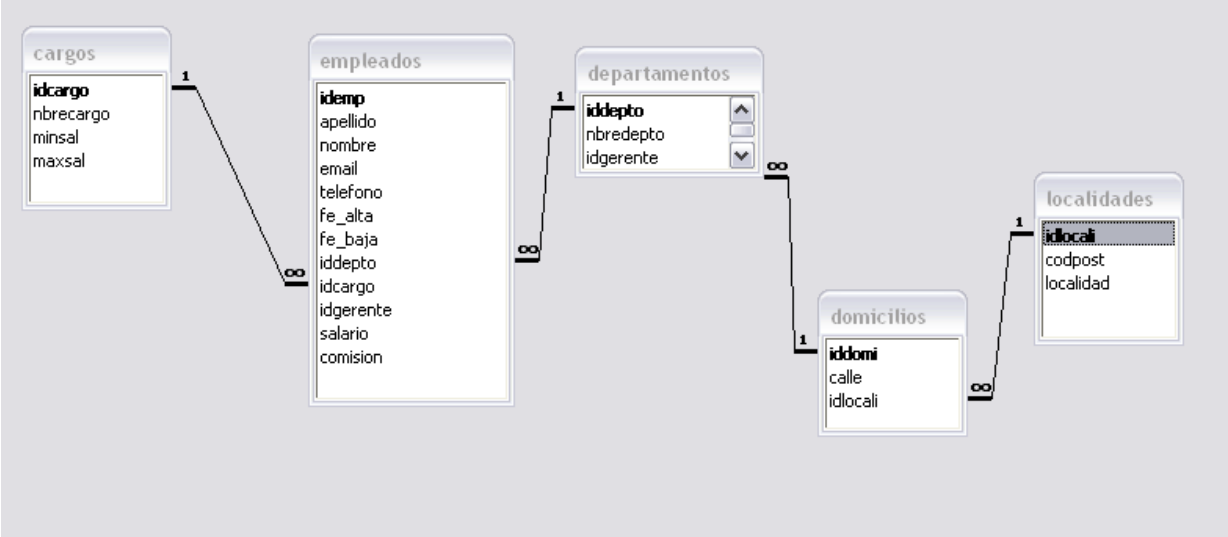
Las sentencias que veremos son: SELECT (en sus diversas opciones, incluyendo la unión y el join), INSERT, UPDATE y DELETE.

3. SQL – DML:

Las sentencias que utilizaremos y que mostramos en los diferentes ejemplos suponen la previa creación de los objetos *(ver punto 4 de la unidad, en lo relativo a la instalación del software BrowserSQL)*

Asimismo para los ejemplos hemos creado la conexión ODBC llamada bdAccess_1 que permite la conexión con la base de datos bdEmpresas.mdb *(ver en el punto 4 cómo crear una conexión ODBC en Windows 7)*

La siguiente imagen muestra el MER físico, en Acces, de la base de datos bdEmpresas capturado desde la vista de relaciones de Access:



Los ejemplos se han llevado a cabo utilizando el BrowserSQL

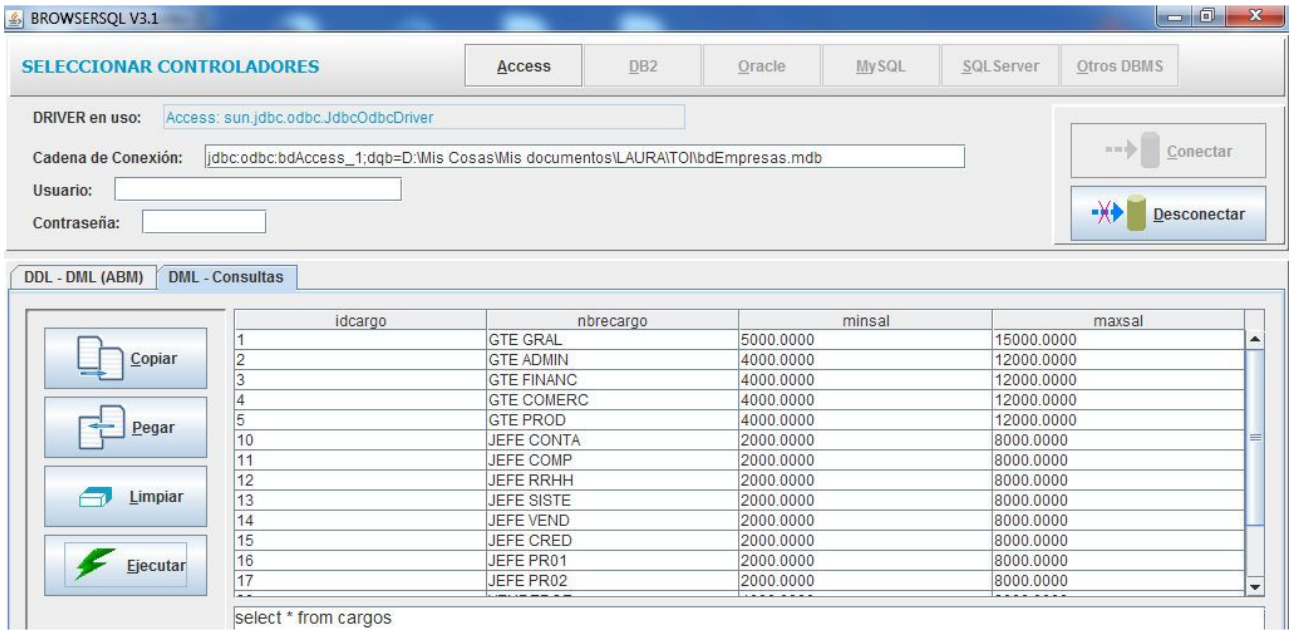
Sentencia SELECT sobre una tabla:

Sentencia básica

Sintaxis: SELECT * FROM tabla

El resultado de esta sentencia es el total de los registros de la tabla

Ejemplo: mostrar todos los registros o tuplas de la tabla Cargos

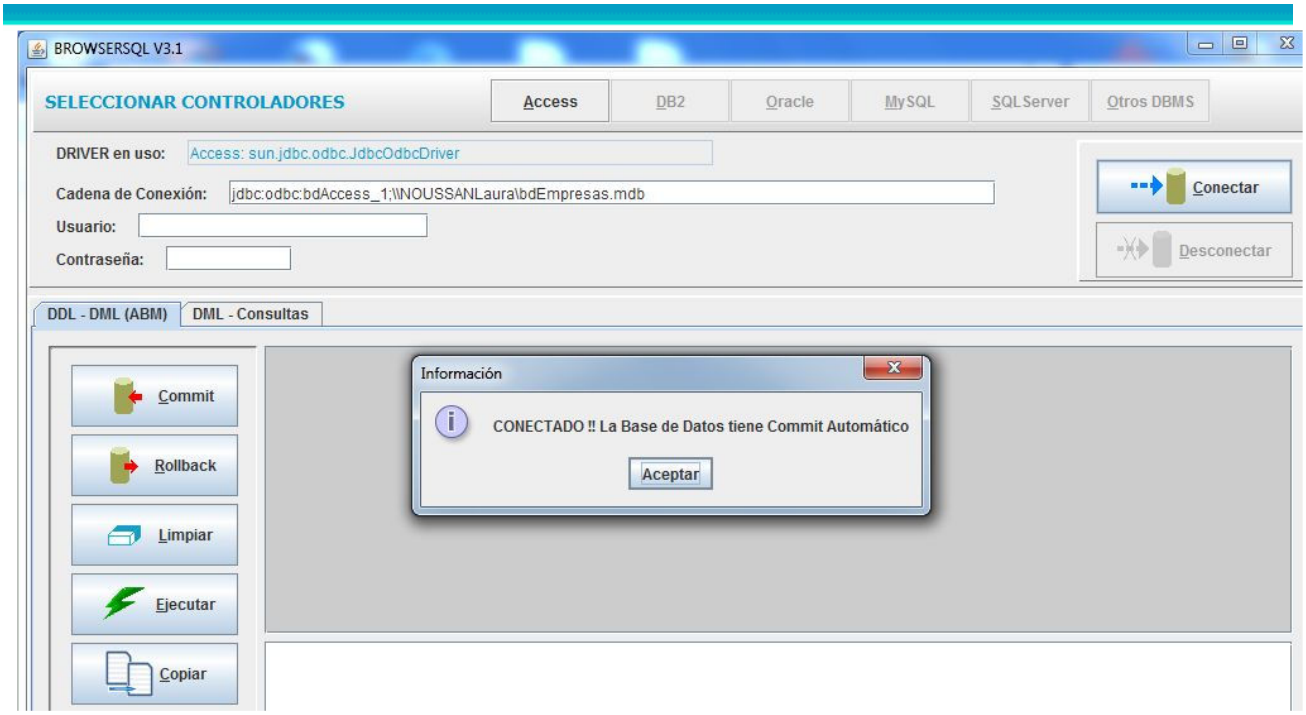


La cadena de conexión tiene la siguiente estructura

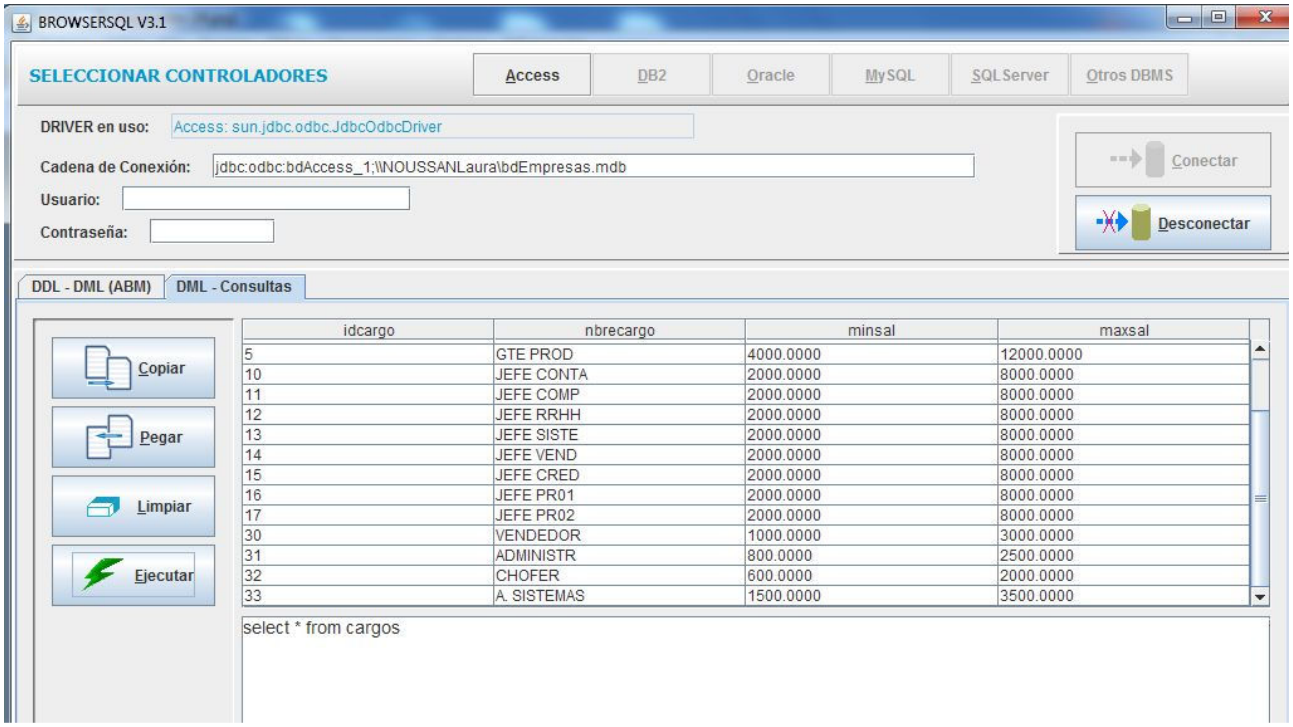
jdbc:odbc:bdAccess_1;dqb=D:\Mis Cosas\Mis documentos\LAURA\TOI\bdEmpresas.mdb

Otra alternativa es:
Jdbc:odbc:bdAccess_1;\\NOUSSANLaura\bdEmpresas.mdb

Esto es porque, después del punto y coma, hemos utilizado el dominio o nombre de la notebook



La conexión con la cadena utilizando el nombre del equipo



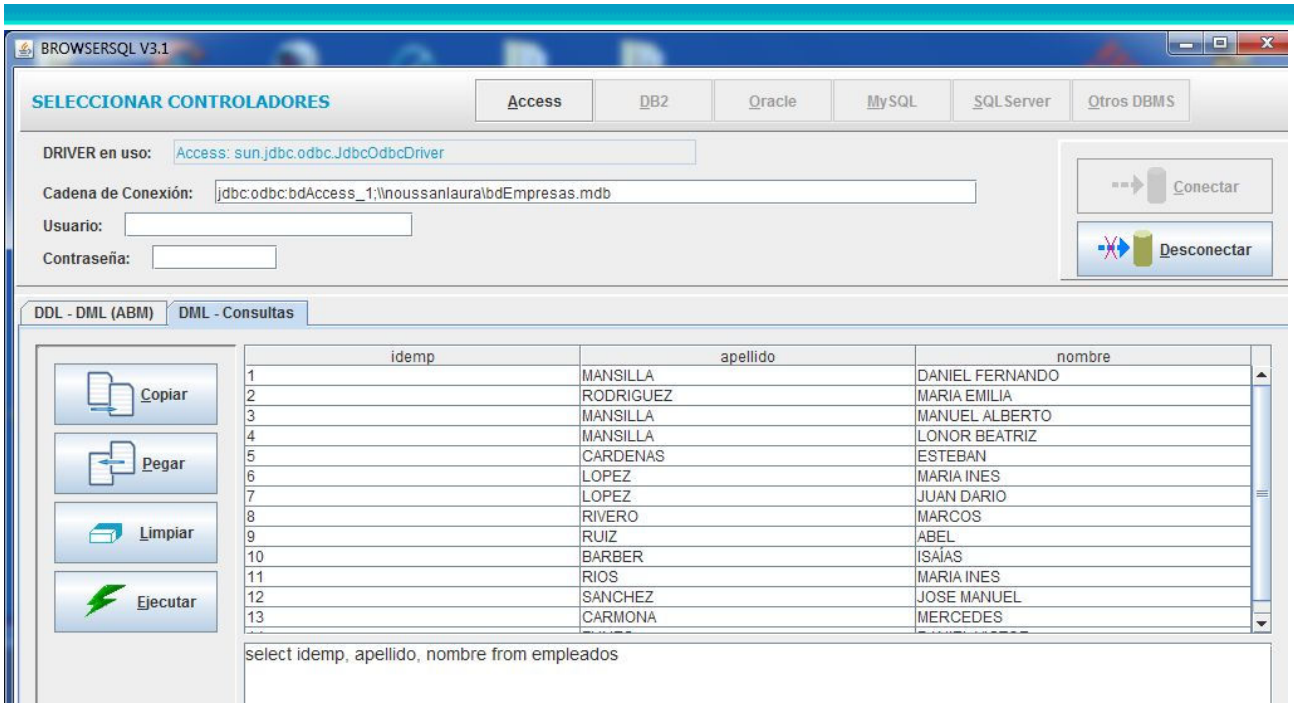
La misma consulta anterior

Sentencia SELECT con proyección

Sintaxis: SELECT col1,col4,col6 FROM tabla

El resultado de aplicar esta sentencia es que obtenemos todas las filas pero se muestran sólo las columnas indicadas.

Ejemplo: mostrar el apellido, nombre y legajo de los empleados

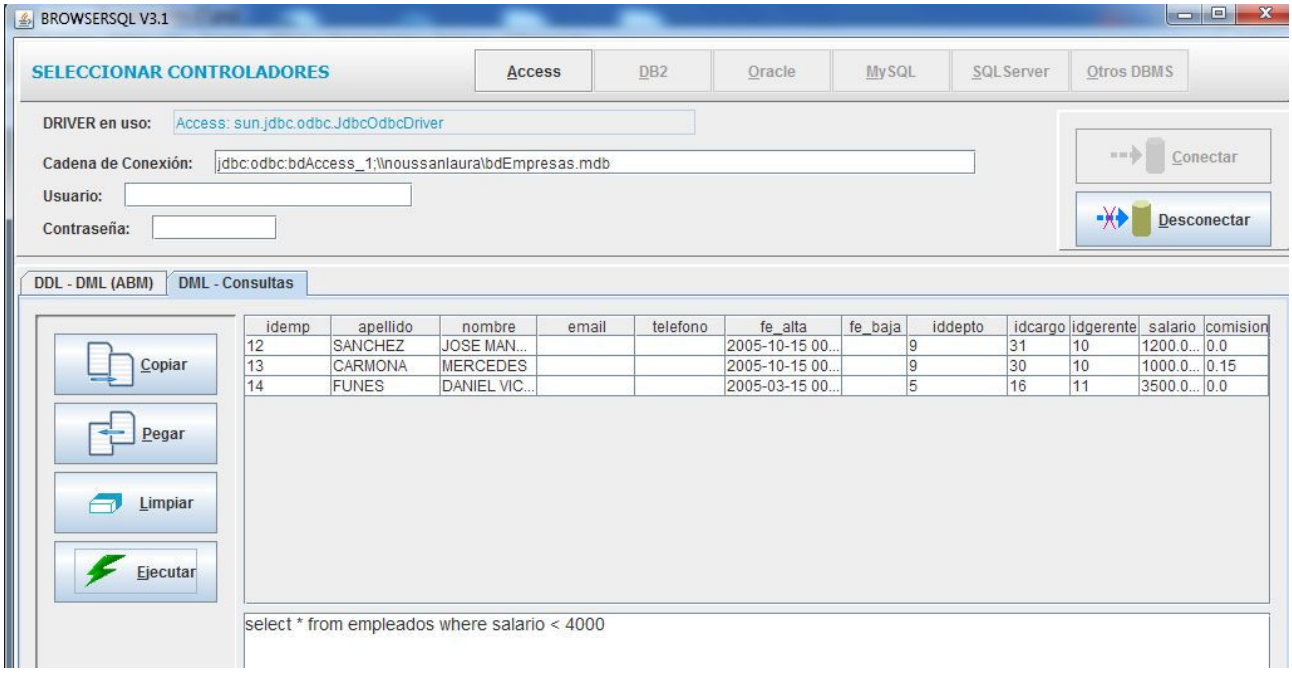


Sentencia SELECT con restricción

Sintaxis: SELECT * FROM tabla WHERE condición

El resultado de aplicar la sentencia es que se devuelven sólo las filas o tuplas que coinciden con la condición.

Ejemplo: mostrar los empleados salario es menor a 4000

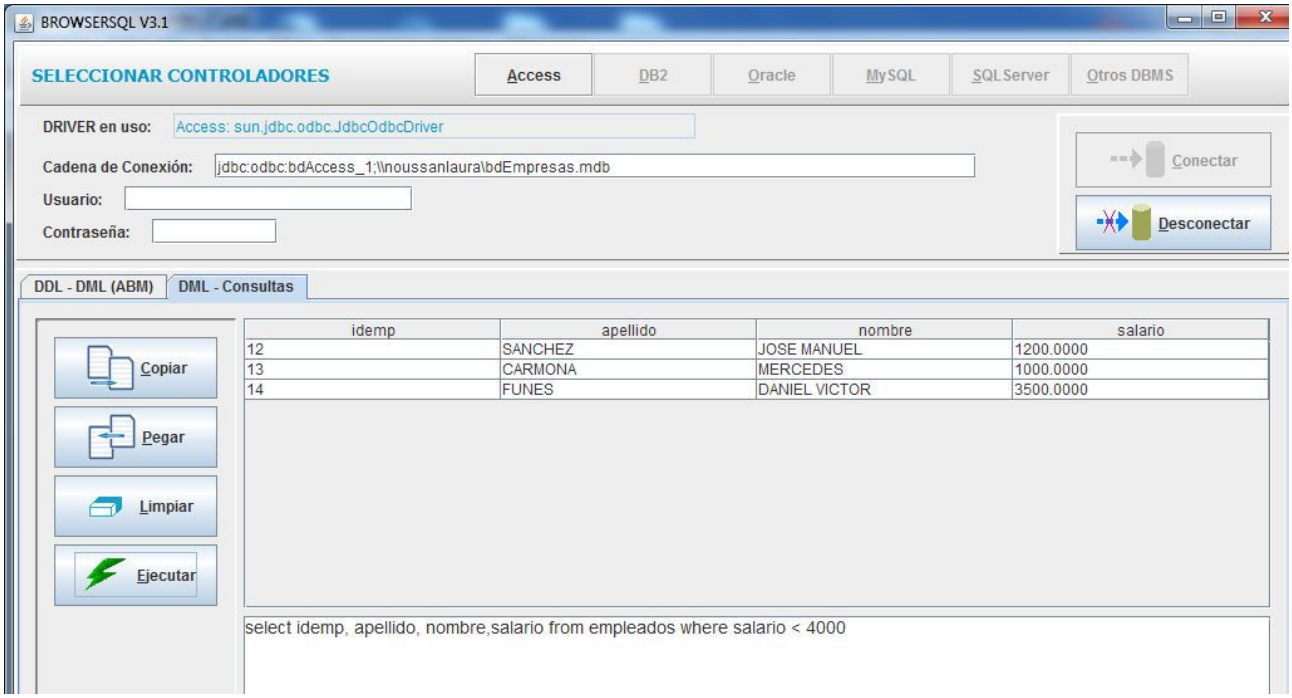


Sentencia SELECT mixta

Sintaxis: SELECT col1,col4,col6 FROM tabla WHERE condición

El resultado de esta sentencia es que se devuelven las tuplas que coinciden con la expresión de la condición y sólo se muestran los datos de las columnas o atributos indicados.

Ejemplo: mostrar el legajo, apellido, nombre y salario de los empleados cuyo salario es menor a 4000

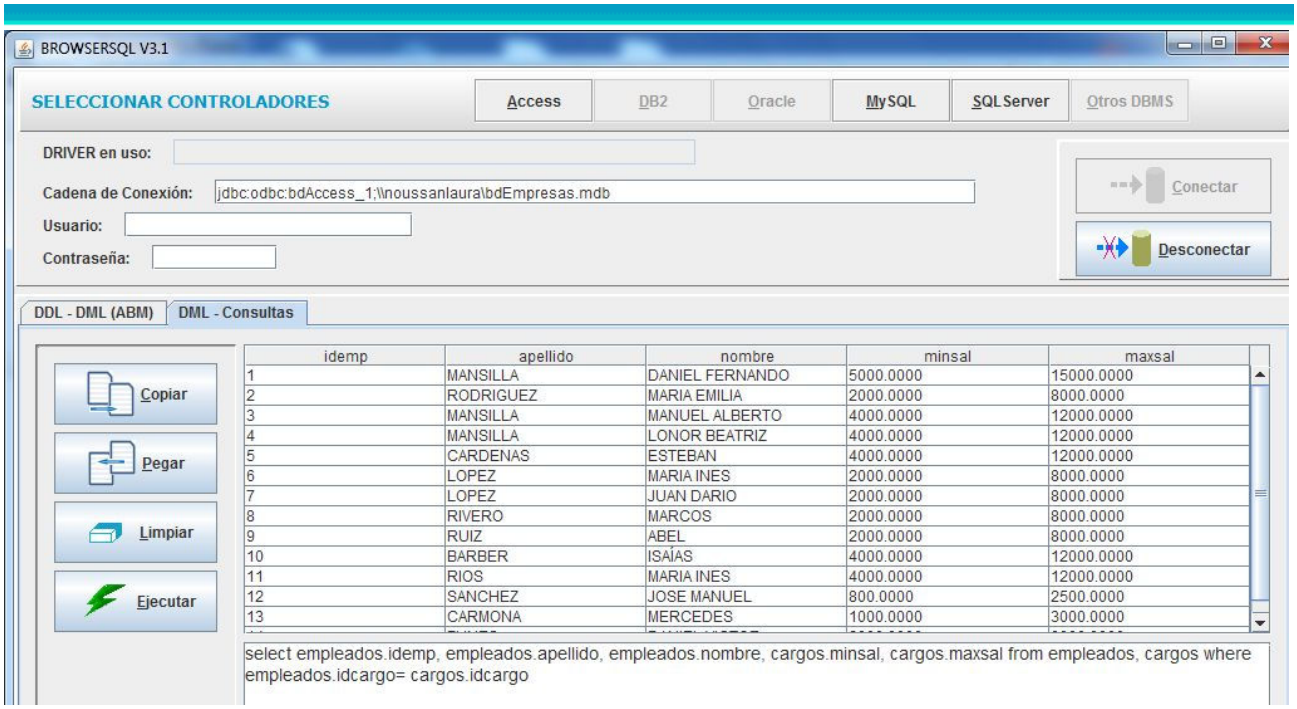


Sentencia SELECT sobre más de una tabla

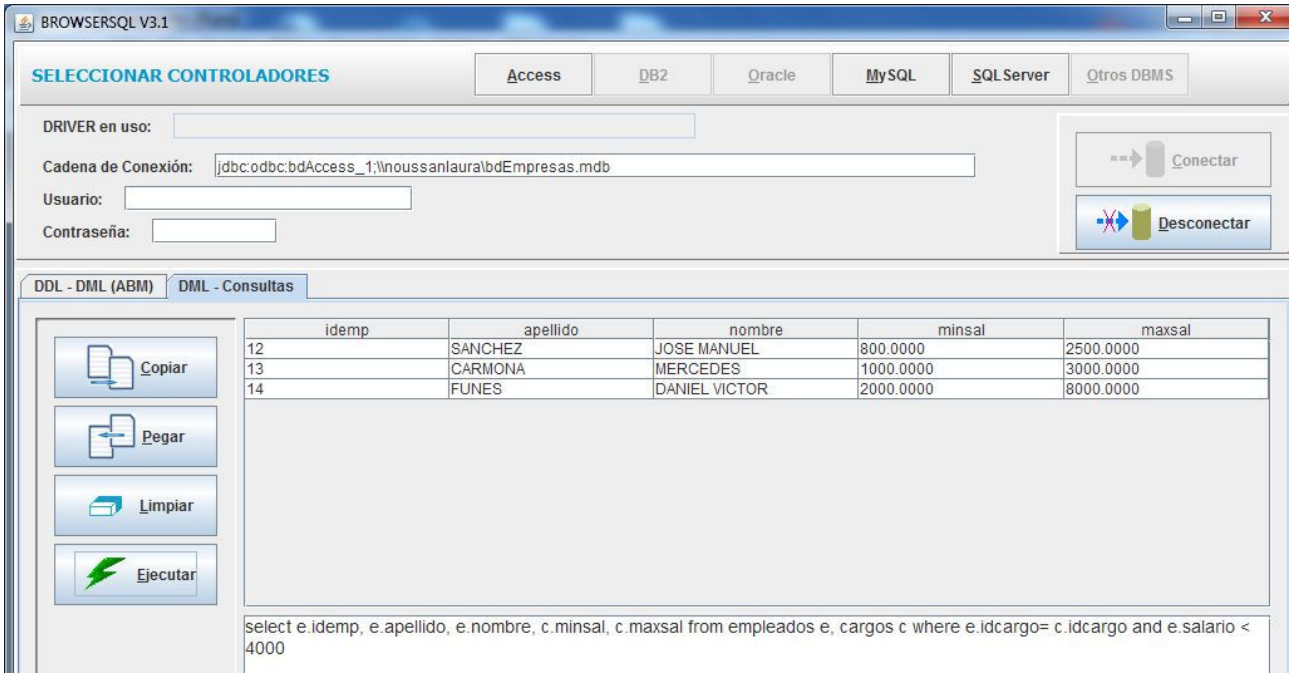
En este caso utilizaremos la Unión de tipo **join,** es decir aplicaremos las nociones teóricas vistas sobre claves primarias y foráneas.

Sintaxis: SELECT tabla1.col2, tabla2.col1, tabla3.col2 FROM tabla1, tabla2, tabla3 WHERE tabla1.id_clave = tabla2.id_clave AND tabla2.id_clave = tabla3.id_clave

Ejemplo: mostrar el legajo, apellido, nombre, el salario máximo y el salario mínimo de los empleados



Ejemplo: mostrar el legajo, apellido, nombre, el salario máximo y el salario mínimo de los empleados cuyo salario es menor a 4000

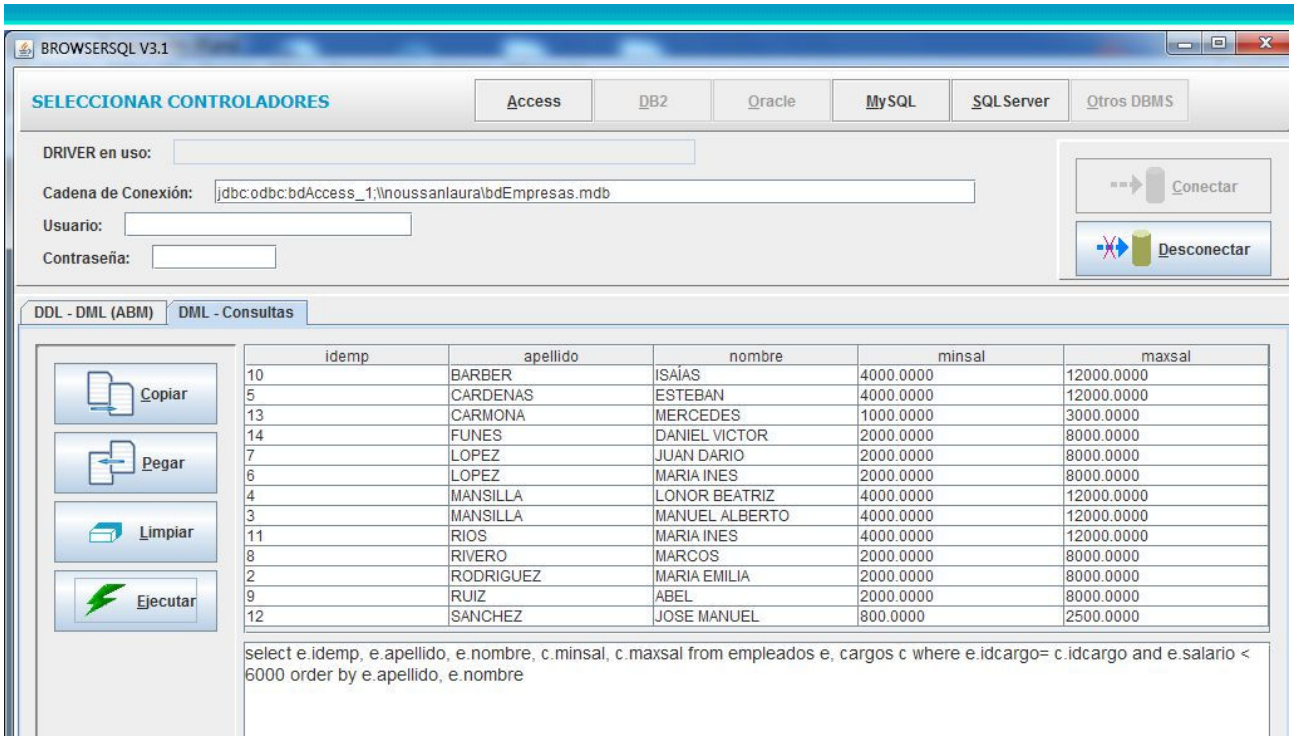


Notar que en el segundo ejemplo se han ***utilizado alias de tablas***, lo cual hace mucho más fácil la escritura y también la consulta más rápida.

Cláusula ORDER BY

La sintaxis de esta cláusula es … ORDER BY col1,col2 [ASC | DESC] siendo predeterminada la ordenación ascendente.

Ejemplo: mostrar el legajo, apellido, nombre, el salario máximo y el salario mínimo de los empleados cuyo salario es menor a 6000 ordenados por apellido y nombre ascendentes



Cláusula LIKE

Esta cláusula permite realizar una restricción donde la condición implica la comparación entre cadenas. Básicamente la sintaxis utiliza el carácter comodín %.

Hay tres casos básicos. Tomaremos como ejemplo la tabla empleados y el atributo apellido.

Caso 1: conocer los empleados cuyo apellido empieza con la letra M

SELECT idemp, apellido, nombre FROM empleados WHERE apellido LIKE 'M%'

Las comillas indican que es una cadena y el símbolo % después de la letra M indica al DBMS que no interesa el resto de la cadena.

Caso 2: conocer los empleados cuyo apellido termina en EZ, la consulta será:

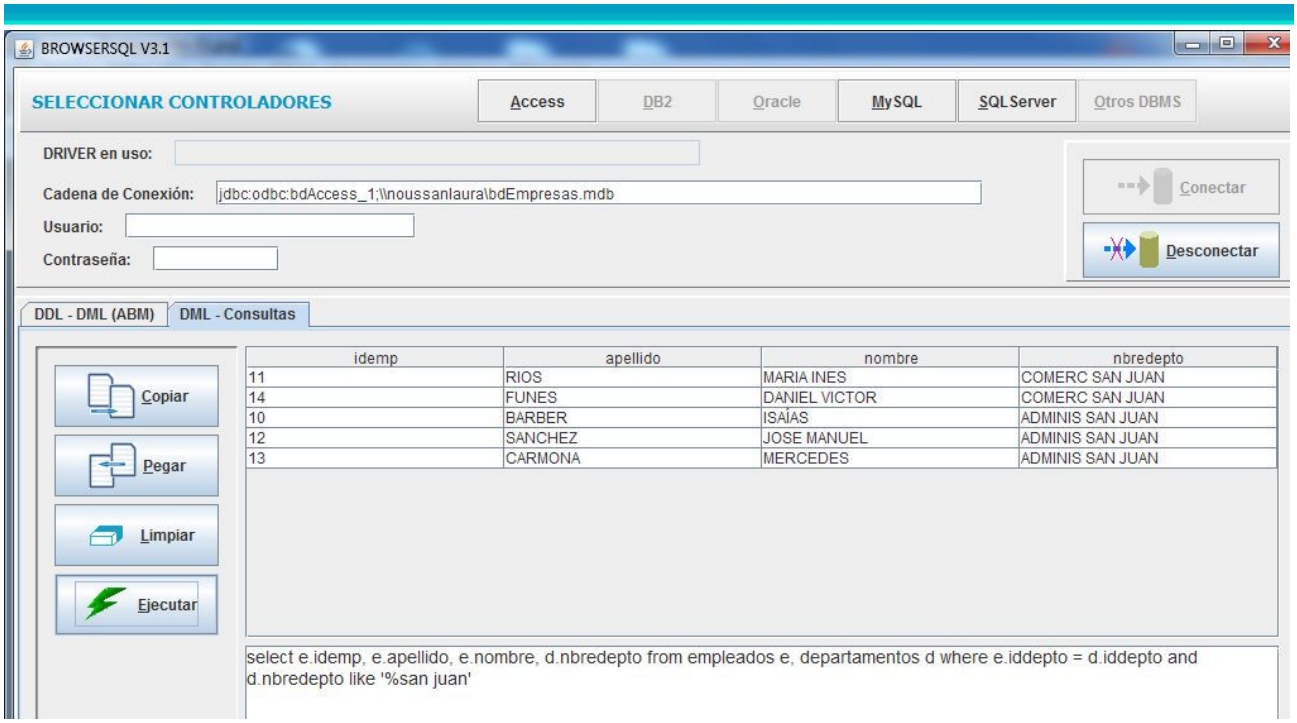
SELECT idemp, apellido, nombre FROM empleados WHERE apellido LIKE '%EZ'

Caso 3: si se quiere saber qué empleados tienen en el apellido la CH

SELECT idemp, apellido, nombre FROM empleados WHERE apellido LIKE '%CH%'

Ejemplo: mostrar el legajo, apellido, nombre de los empleados y el nombre del departamento en que trabajan y que esté en San Juan.

Si hiciéramos un SELECT sobre la tabla Departamentos conoceríamos que en San Juan y Mendoza hay varias oficinas (comercialización, administración); es decir funcionalmente son departamentos administrativos.



Teniendo en cuenta ese detalle se obtienen todos los empleados que trabajan en San Juan, a través del nombre del departamento. En este caso la consulta utilizó el LIKE conforme al segundo caso visto con anterioridad.

Sentencia INSERT: será analizada en profundidad con la base de datos Universidad, después de crear las tablas (punto 4)

Sentencia UPDATE: será analizada en profundidad con la base de datos Universidad, después de crear las tablas (punto 4)

Sentencia DELETE: será analizada en profundidad con la base de datos Universidad, después de crear las tablas (punto 4)

4. SQL – DDL: sentencias aplicadas a objetos Tabla y Vista

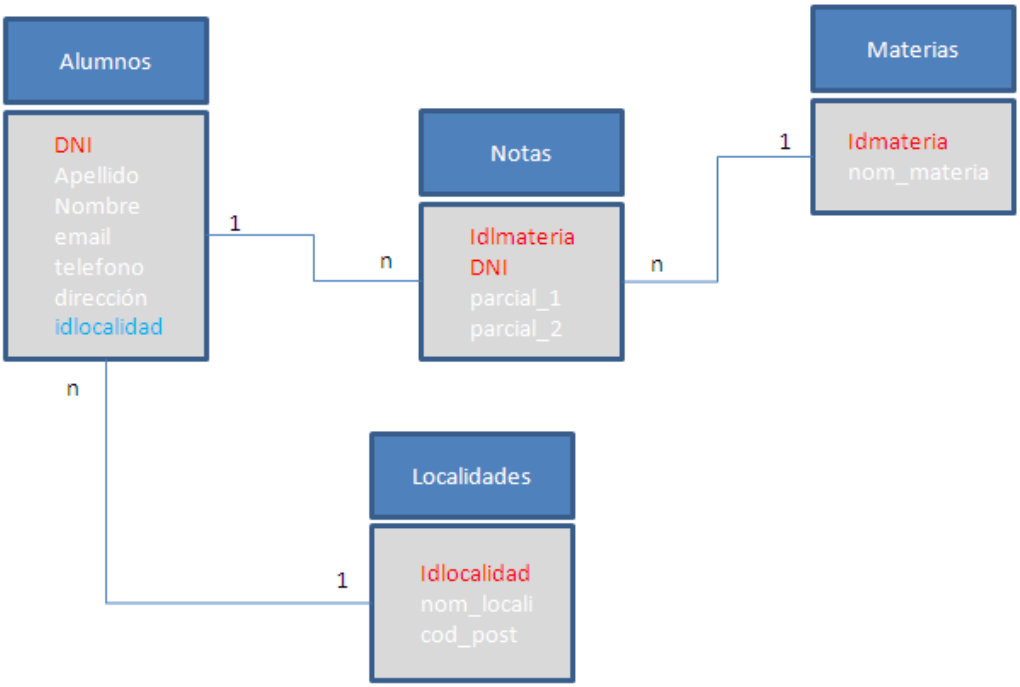
Para utilización de las diferentes sentencias se llevará a cabo la creación de la base de datos la Universidad, desarrollada en clase, en Access y en MySQL

Para llevar a cabo los ejemplos se deberá tener instalado en las netbooks:

1. Access
2. MySQL
3. BrowserSQL

El Diseño lógico corresponde al siguiente MER y a la siguiente estructura lógica de cada tabla

MER



DESCRIPCIÓN ESTRUCTURA LÓGICA

Tabla	Alumnos				
Atributos	**Dominio**	**¿es PK?**	**¿es FK?**	**¿obligat?**	**Descripción**
DNI	numerico	sí	no	sí	Es el número de documento
Apellido	alfabetico	no	no	si	apellido del alumno
Nombre	alfabetico	no	no	si	nombres del alumno
email	alfanumerico	no	no	no	correo electrónico
telefono	numerico	no	no	no	el teléfono del alumno
domicilio	alfanumerico	no	no	si	el domicilio comprende la calle y número
idlocalidad	numerico	no	sí	si	localidad del domicilio

Tabla	Materias				
Atributos	**Dominio**	**¿es PK?**	**¿es FK?**	**¿obligat?**	**Descripción**
Idmateria	numerico	sí	no	sí	identificador único de la materia
nom_materia	alfanumerico	no	no	sí	nombre de la materia

Tabla	Localidades				
Atributos	**Dominio**	**¿es PK?**	**¿es FK?**	**¿obligat?**	**Descripción**
Idlocalidad	numerico	sí	no	sí	identificador único de la localidad
nom_localidad	alfanumerico	no	no	si	nombre de la localidad
cod_post	numerico	no	no	sí	el código postal, puede ser clave alternativa

Tabla	Notas				
Atributos	**Dominio**	**¿es PK?**	**¿es FK?**	**¿obligat?**	**Descripción**
Idmateria	numerico	sí	sí	sí	En forma conjunta (Idmateria,DNI) es la clave primaria compuesta;
DNI	numerico		sí	sí	en forma separada cada atributo es una clave foránea
parcial_1	numerico	no	no	no	la nota del parcial 1
parcial_2	numerico	no	no	no	la nota del parcial 2

El diseño físico será llevado a cabo considerando dos alternativas: de Access y MySQL.
Primero se desarrollará la parte teórica-práctica y las prácticas utilizando Access y con posterioridad MySQL

Diseño Físico de la Base de Datos Universidad para Access

Ha sido diseñado considerando cuestiones importantes de tipos de datos (SQL) de Access, para lo cual se puede consultar la ayuda del producto. Asimismo es importante nombrar en forma específica las claves primarias y las foráneas, de lo contrario los DBMS asignan un nombre arbitrario.

El diseño puede observarse en la siguiente imagen:

Tabla	Alumnos								
Atributos	**Dominio**	**Tipo de Datos**	**Nombre interno en Access**	**¿es PK?**	**nombre pk**	**¿es FK?**	**nombre FK**	**¿obligat?**	**Descripción**
DNI	numerico	entero largo (autonumerico)	AUTOINCREMENT	sí	DNI	no		sí	Es el número de documento
Apellido	alfabetico	texto(40)	TEXT(40)	no		no		si	apellido del alumno
Nombre	alfabetico	texto (36)	TEXT(36)	no		no		si	nombres del alumno
email	alfanumerico	texto (30)	TEXT(30)	no		no		no	correo electrónico
telefono	numerico	entero largo	LONG	no		no		no	el teléfono del alumno
domicilio	alfanumerico	texto (50)	TEXT(50)	no		no		si	el domicilio comprende la calle y número
idlocalidad	numerico	entero largo	LONG	no		sí	FK_localidad	si	localidad del domicilio

Tabla	Materias								
Atributos	**Dominio**	**Tipo de Datos**	**Nombre interno en Access**	**¿es PK?**	**nombre pk**	**¿es FK?**	**nombre FK**	**¿obligat?**	**Descripción**
Idmateria	numerico	entero largo (autonumerico)	AUTOINCREMENT	sí	Idmateria	no		sí	identificador único de la materia
nom_materi	alfanumerico	texto (24)	TEXT(24)	no		no		sí	nombre de la materia

Tabla	Localidades								
Atributos	**Dominio**	**Tipo de Datos**	**Nombre interno en Access**	**¿es PK?**	**nombre pk**	**¿es FK?**	**nombre FK**	**¿obligat?**	**Descripción**
Idlocalidad	numerico	entero largo (autonumérico)	AUTOINCREMENT	sí	Idlocalidad	no		sí	identificador único de la localidad
nom_localid	alfanumerico	texto (60)	TEXT(60)	no		no		si	nombre de la localidad
cod_post	numerico	entero	INTEGER	no		no		sí	el código postal, puede ser clave alternativa

Tabla	Notas								
Atributos	**Dominio**	**Tipo de Datos**	**Nombre interno en Access**	**¿es PK?**	**nombre pk**	**¿es FK?**	**nombre FK**	**¿obligat?**	**Descripción**
Idmateria	numerico	entero largo (autonumérico)	LONG	sí	PK_notas	sí	FK_dni	sí	En forma conjunta (Idmateria,DNI) es la clave
DNI	numerico	entero largo	LONG			sí	FK_idmateria	sí	primaria compuesta; en forma separada cada
parcial_1	numerico	decimal (5,2)	DOUBLE	no		no		no	la nota del parcial 1
parcial_2	numerico	decimal (5,2)	DOUBLE	no		no		no	la nota del parcial 2

Utilización del BrowserSQL

Esta aplicación está totalmente desarrollada en Java. Permite la conexión y aplicación de sentencias DDL y DML para tres DBMS distintos: Access, MySQL y SQL SERVER.

En los dos últimos casos basta con seleccionar el DBMS y escribir la cadena de conexión a la base de datos. La base de datos puede estar alojada en la misma máquina o en otra máquina, igual funcionará, dependerá de indicar en forma apropiada el servidor.

En el caso de que el sistema físico se desarrolle con Access, la conexión no es directa porque, a la fecha en que se desarrolló la aplicación, Microsoft sólo había sacado un controlador java puro para SQL Server y no para Access. En este caso, la propia plataforma java, trae un controlador por omisión que en realidad es un puente entre Java y ODBC. Este tipo de Driver o controlador se denomina del tipo jdbc-odbc; es decir un puente entre java y odbc,

ODBC es una tecnología de conexión a distintos orígenes de datos de Microsoft. Es una tecnología que si bien, vieja, ha sido tenida en cuenta por otros fabricantes de Software, si querían que sus productos de bases de datos pudieran funcionar sobre Windows.

Por lo tanto, para poder conectarnos a una base de datos Access deberemos configurar la conexión ODBC sobre el sistema operativo Windows. Estas operaciones se muestran en las siguientes imágenes y han sido llevadas a cabo sobre el Sistema Operativo Windows 7. En el tutorial del BrowserSQL la explicación se basa en la configuración realizada sobre el Sistema Operativo Windows XP.

Creación de una Conexión ODBC

Antes que nada deberemos crear una base de datos Access en blanco o utilizar una base de datos en blanco como la que se encuentra en el Site Web.

Una vez bajada a la máquina anfitriona se le cambia el nombre de modo que sea representativo. En nuestro ejemplo le hemos llamado bdUniversidad.mdb.

Por lo tanto cuando terminamos de crear la conexión ODBC, es decir, indicando el DNS de Sistema, nos pedirá la ruta de la base datos y su nombre; se puede ver en la última imagen.

Seguidamente pasaremos a detallar los pasos necesarios para crear una conexión ODBC para esta base de datos.

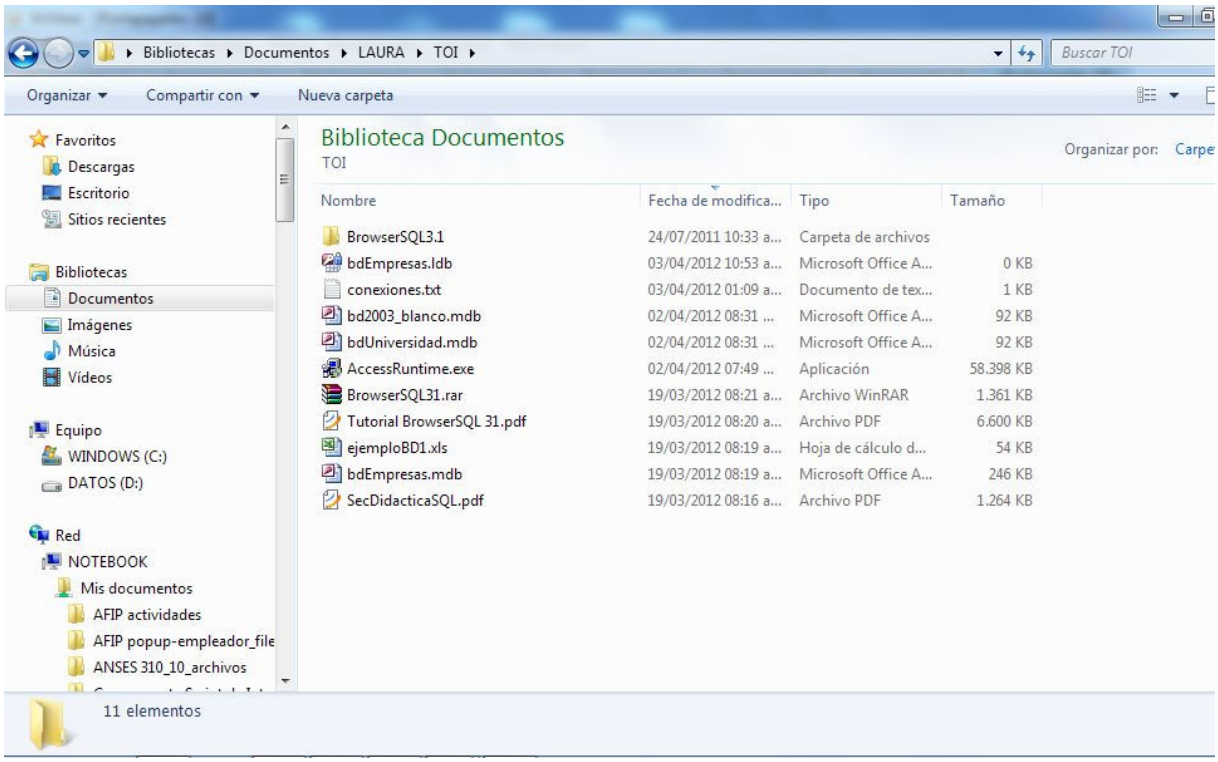


Figura 1: el directorio TOI en la netbook

Podemos ver que tenemos la bdEmpresas.mdb, bdUniversidad.mdb y la bd2003_blanco.mdb; estas dos últimas son bases de datos en blanco, sin ningún objeto.

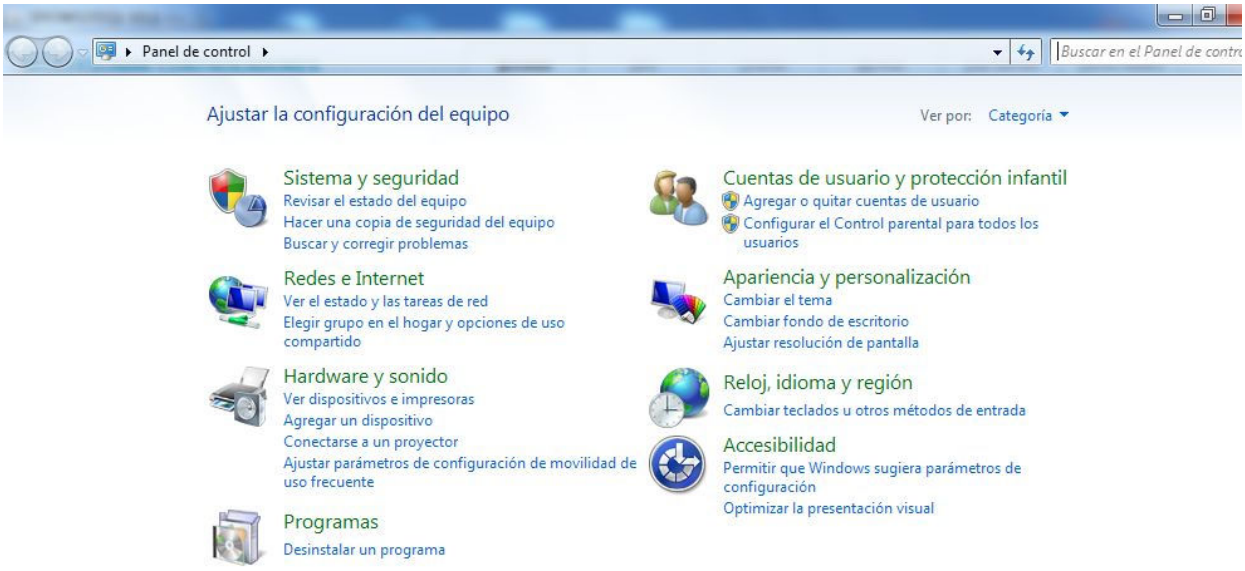


Figura 2: Panel Control Windows 7

También encontramos la aplicación AccessRuntime.exe que es el runtime que necesitamos para poder utilizar las bases de datos Access ya que la versión instalada en la Netbook no trae Access.

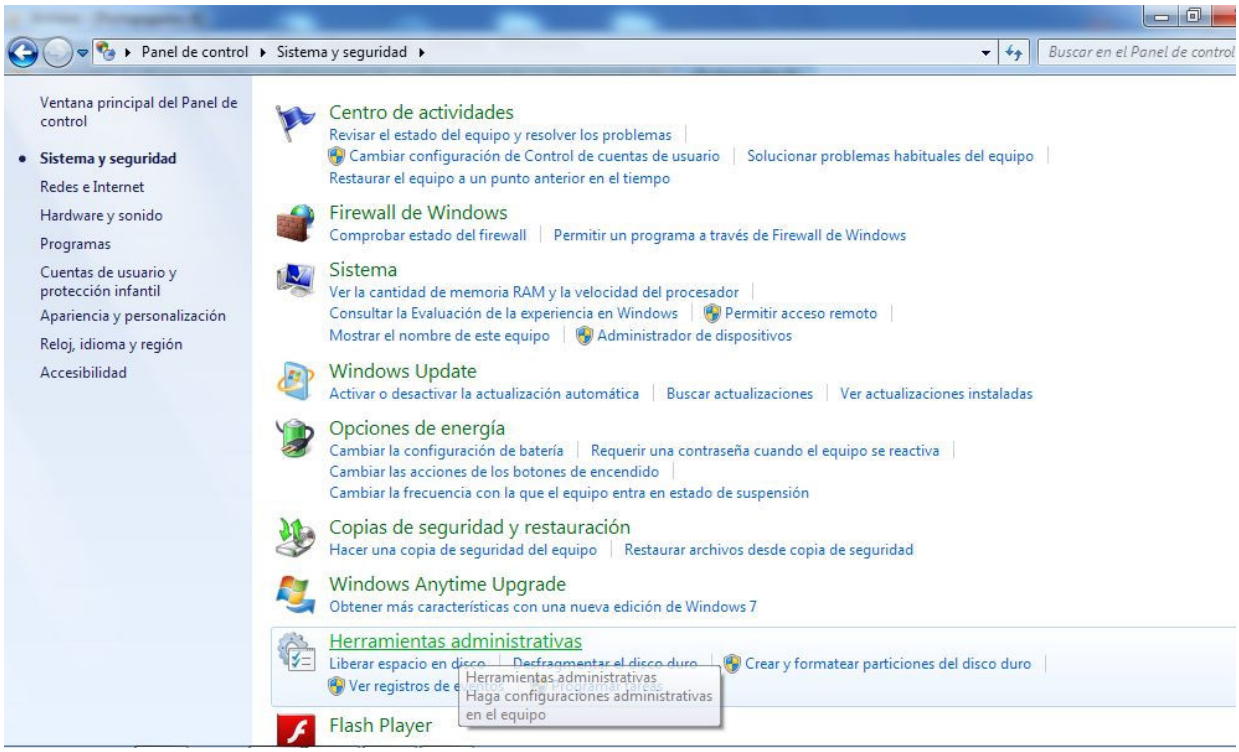


Figura 3: seleccionamos Herramientas Administrativas

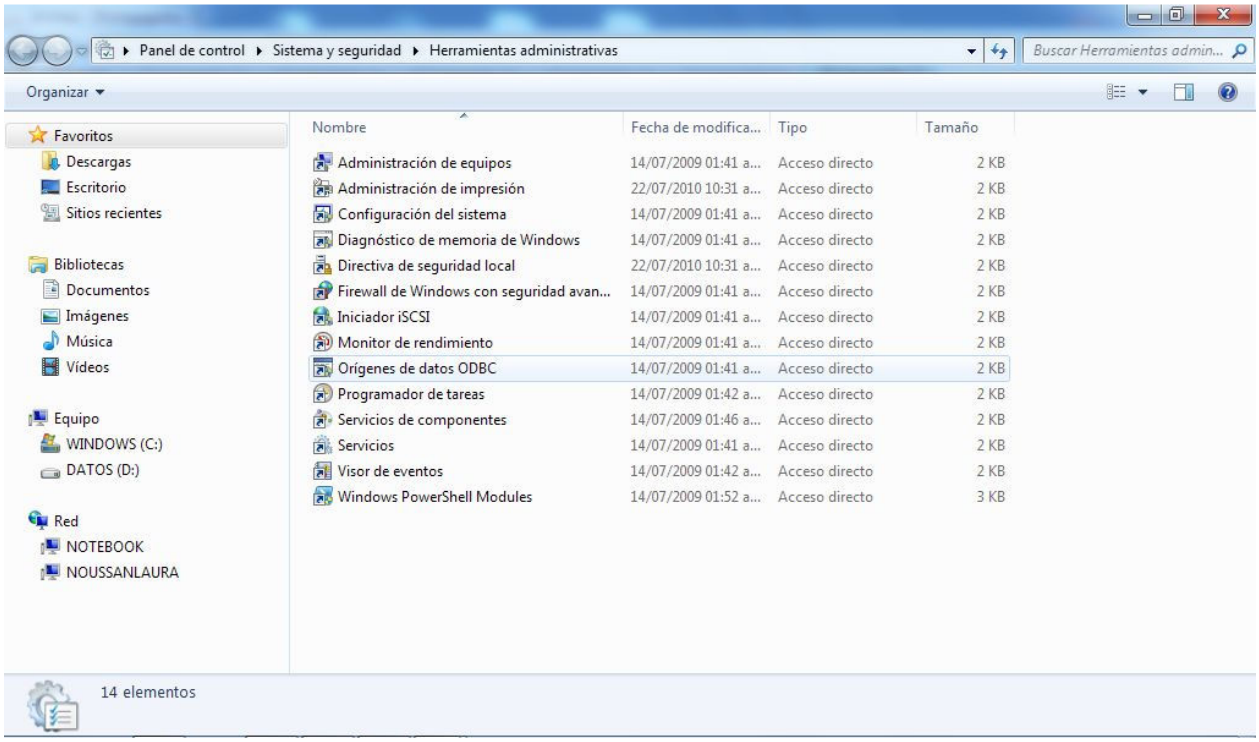


Figura 4: Seleccionamos Orígenes de datos ODBC

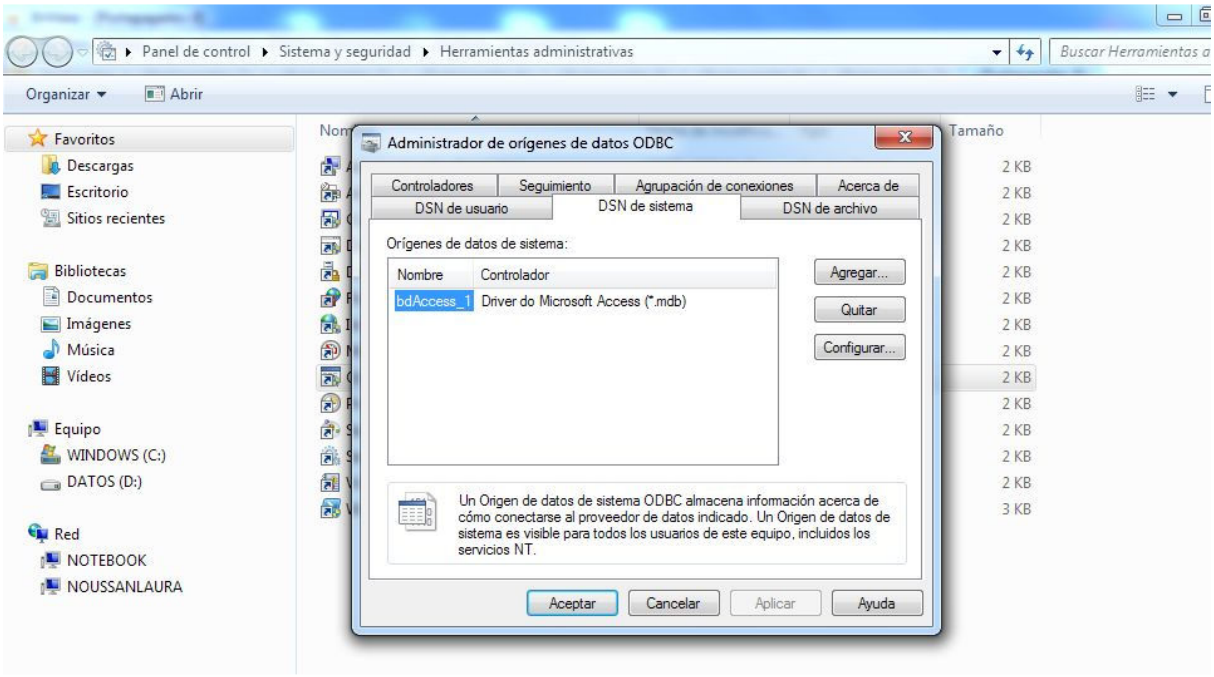


Figura 5: Aparece el Administrador de Orígenes de datos ODBC.
Vamos a pulsar el botón Agregar

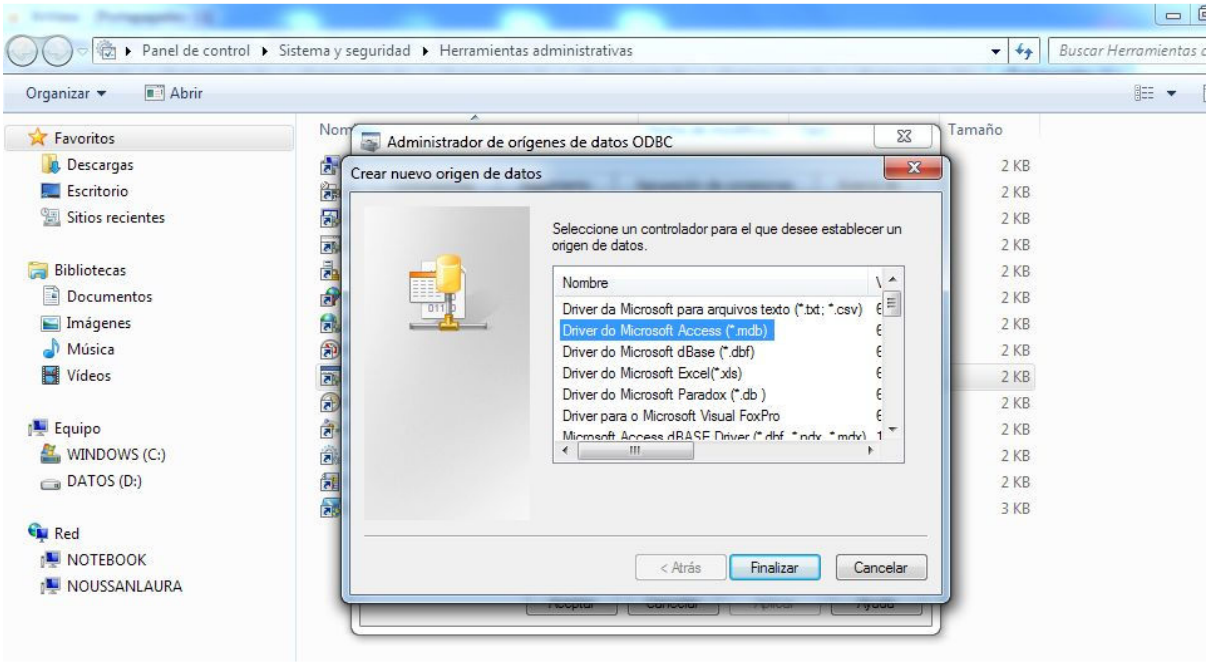


Figura 6: Creamos un nuevo origen de datos para Access (*.mdb)

Notar que seleccionamos el Driver de Microsoft Access para extensión mdb. Si nos deslizamos por la lista veremos que es extensa y permite conectarse a variados orígenes de datos, por ejemplo Oracle e incluso Access para la versión 2007 (extensión accdb).

Al pulsar Finalizar tendremos la posibilidad de nombrar la conexión y seleccionar la base de datos con la cual la utilizaremos, lo que se muestra en la siguiente imagen:

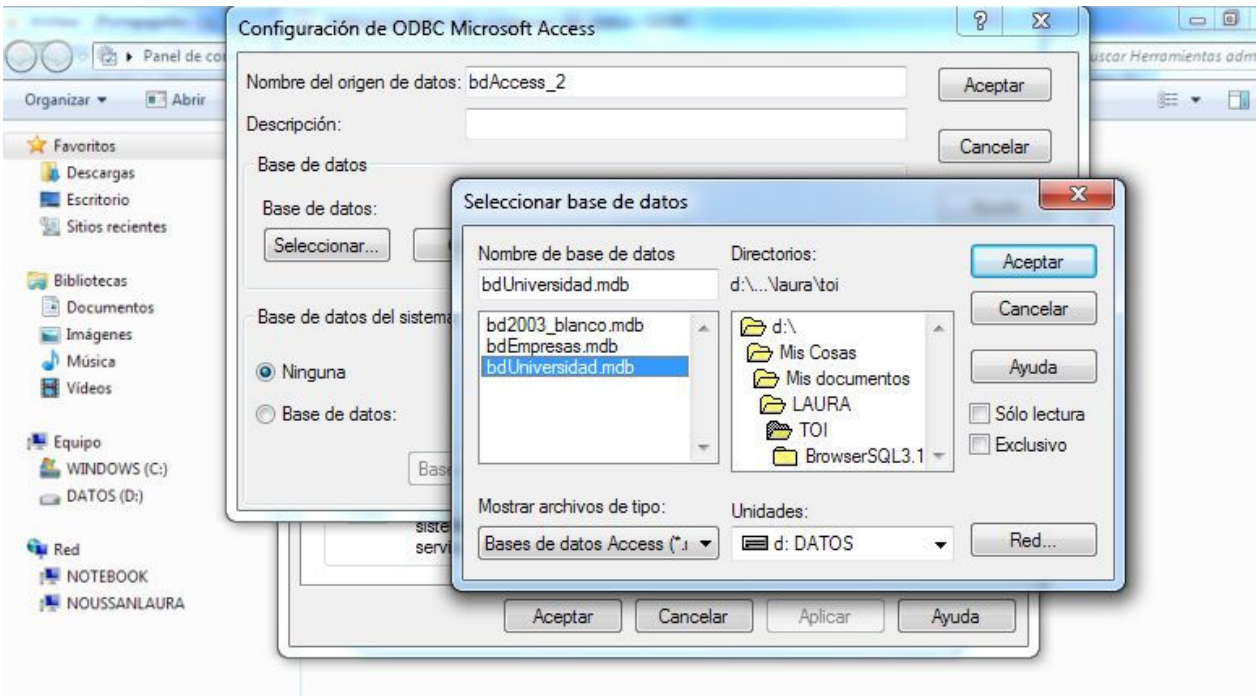


Figura 7: Nombramos la nueva conexión y seleccionamos la base de datos

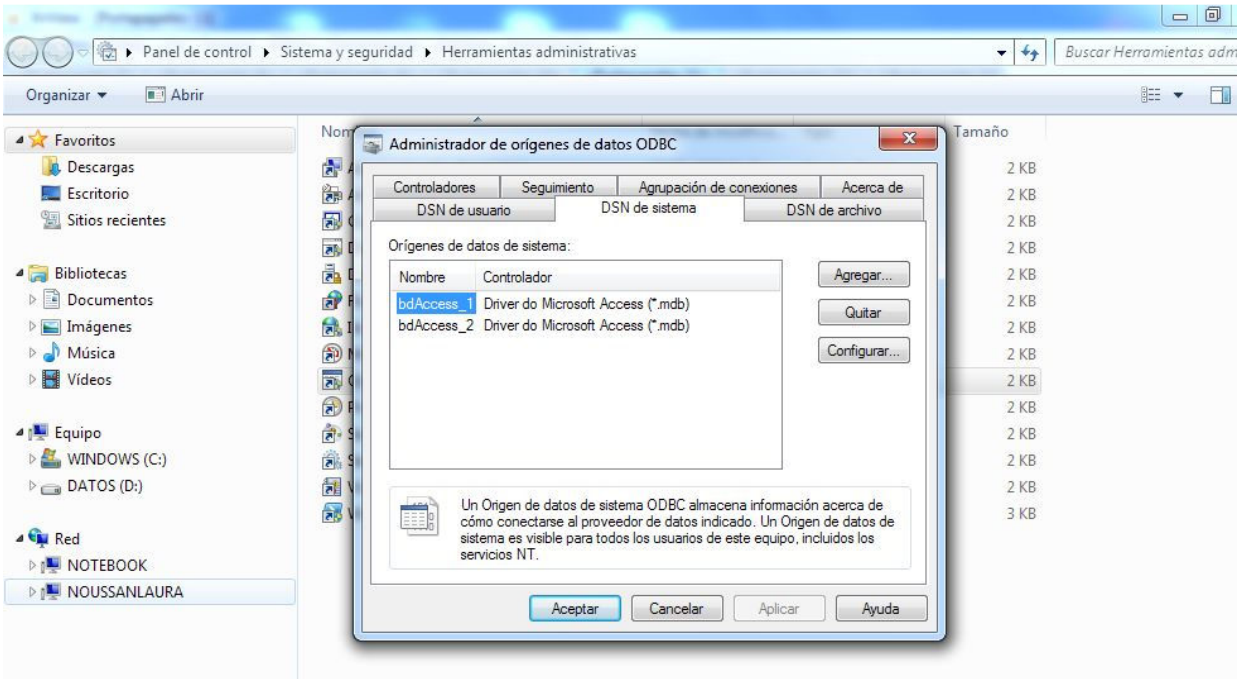


Figura 8: tenemos ahora dos conexiones ODBC.

La primera conexión, bdAccess_1, la utilizaremos para conectarnos a la base de datos bdEmpresas.mdb y la segunda para la base bdUniversidad.mdb.

Con la primera realizaremos los ejemplos de las sentencias SQL DML y con la segunda llevaremos a cabo un ejemplo para aplicar las sentencias SQL DDL.

Utilización de SQL – DDL en un entorno Access utilizando el BrowserSQL

Se utilizará la conexión ODBC bdAccess_2, salvo para la sentencia CREATE y la bdAccess_1 para crear una tabla a la que con posterioridad le modificaremos la estructura y la borraremos (ALTER y DROP)

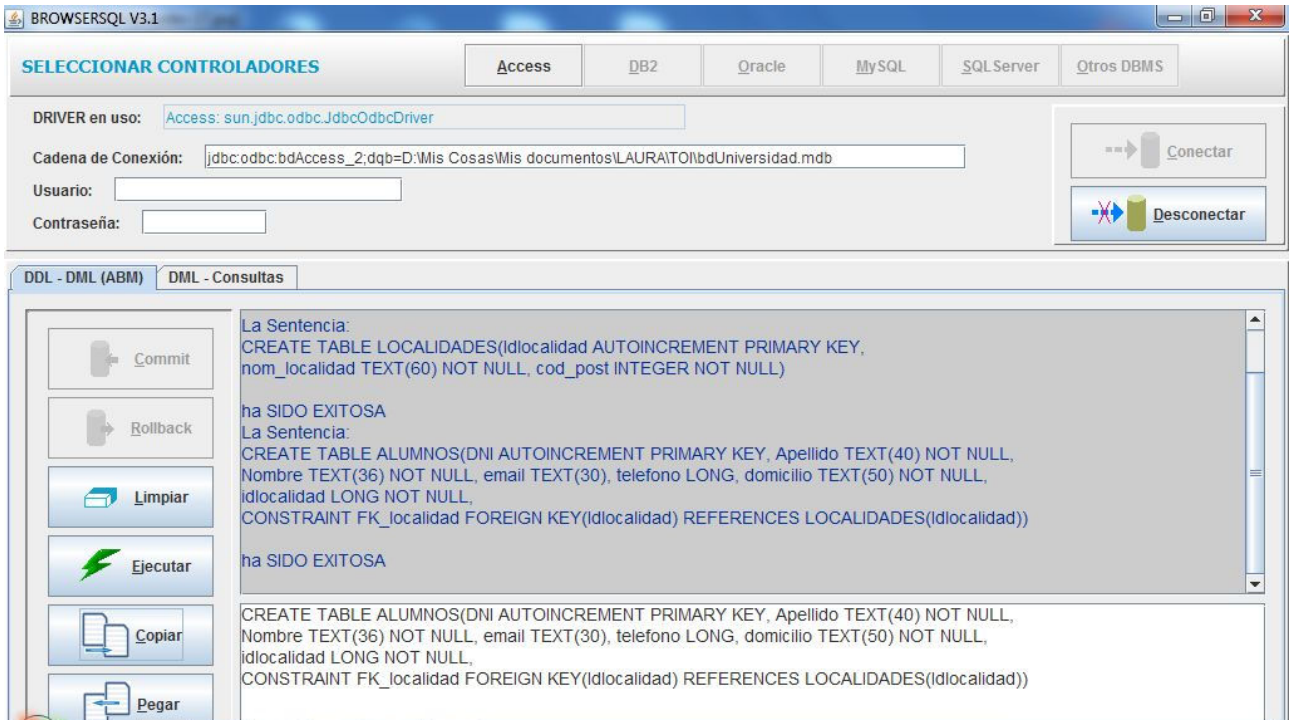
Sentencia CREATE

Con esta sentencia vamos a crear todas las tablas, como puede apreciarse en las siguientes imágenes.

Se comienza siempre por las tablas que sólo tienen claves primarias, es decir, aquellas tablas que en el lenguaje coloquial se denominan primarias; las tablas dependientes (aquellas que tienen claves foráneas) se crean con posterioridad.

En las imágenes se puede apreciar las sentencias de creación de los objetos conforme al diseño lógico y posterior diseño físico que se ha considerado para Access.

La primer tabla a crear es la tabla Localidades puesto que es primaria totalmente.



Notas tabla Localidades:

1) Los atributos van entre los paréntesis después del nombre de la tabla
2) Cada atributo tiene un nombre y un tipo de datos
3) La clave primaria es el atributo con nombre Idlocalidad que se indica con la cláusula PRIMARY KEY

Notas tabla Alumnos:

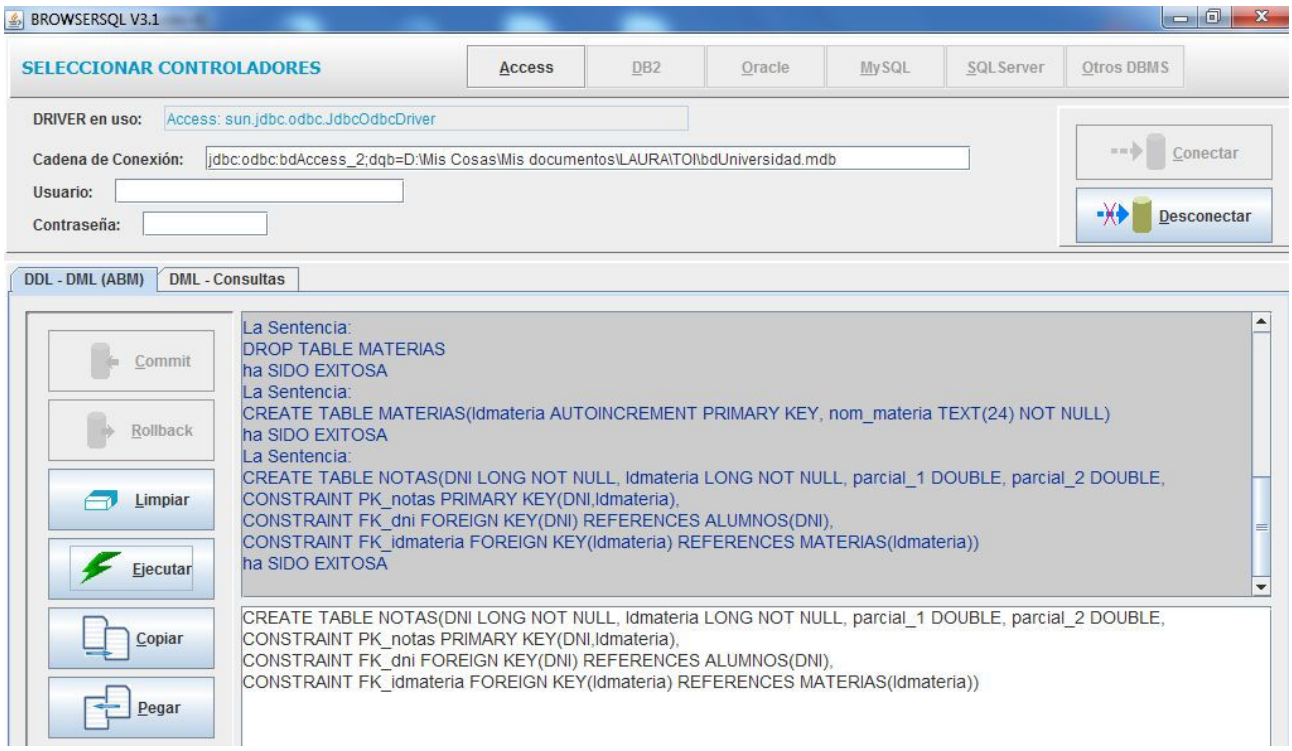
1) Lo mismo que en el caso anterior, salvo que la clave primara es DNI
2) La clave foránea se indica con la cláusula CONSTRAINT

El sentido de esta cláusula CONSTRAINT es totalmente compatible con la relación 1 a n que podemos ver en el MER: la relación 1 a n es desde la tabla Localidades (1) a la tabla Alumnos (n)

> CONSTRAINT nombre_clave_foránea FOREIGN KEY(columna_de_clave_foránea)
> REFERENCES tabla_que_contiene_a_la_clave_primaria_que_es_referenciada
> (columna_de_clave_primaria)

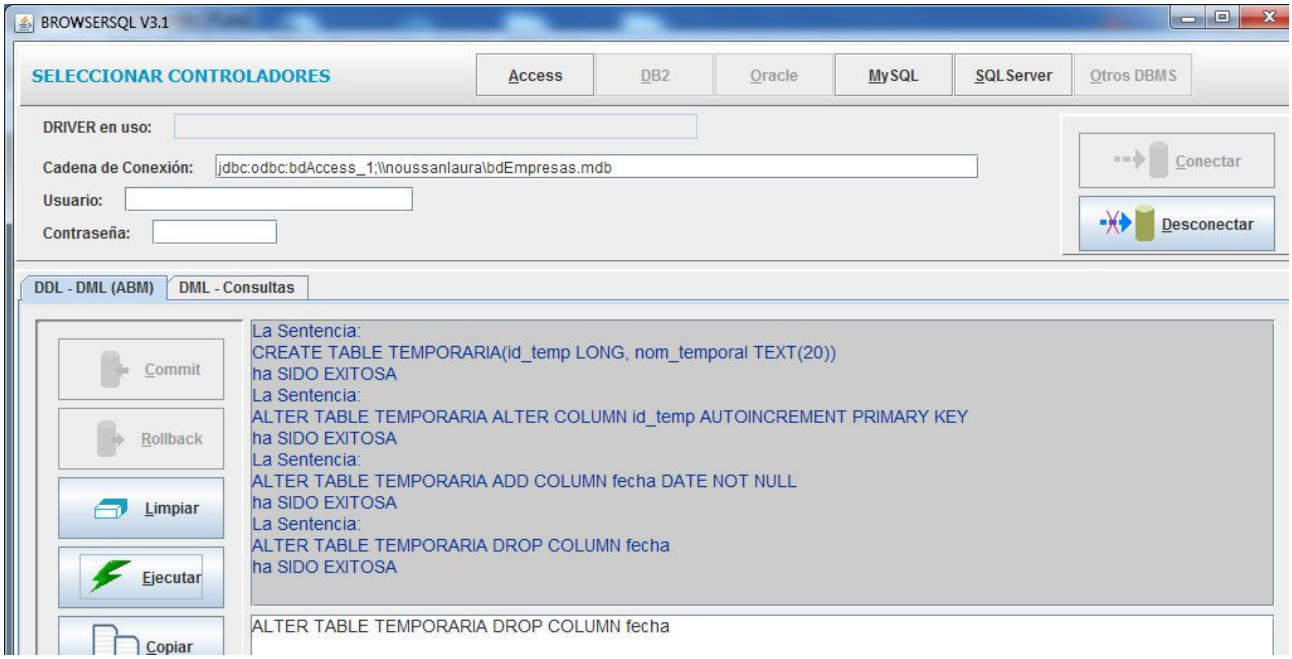
En resumen: la clave foránea referencia o apunta a una clave primaria de la otra tabla. Lógicamente no se trata de cualquier clave primaria ni de cualquier clave foránea, sino de aquellas que están vinculadas a través del diseño lógico en el MER!!

La siguiente figura muestra la creación de las otras dos tablas: Materias y Notas. Materias también se podría haber creado antes de la tabla Alumnos ya que es primaria y Notas, porque depende tanto de Alumnos como de Materias, necesariamente debe crearse después.



Sentencia ALTER

Creamos una tabla que se llamará temporaria, con clave primaria Id_temp y el atributo nom_teporal como se puede ver en la figura.



Como nos equivocamos al no establecer la clave primaria tenemos que modificar la columna id_temp y nombrarla como clave primaria. Para ello utilizamos ALTER tabla ALTER columna

Otro error es que nos olvidamos de otra columna que llamamos fecha, siendo su tipo de datos DATE, por lo tanto la agregamos. Para ello la sintaxis es ALTER tabla ADD columna

Es decir, ALTER implica modificar la estructura de un objeto, lo que puede ser, agregar o eliminar un atributo, cambiarle el nombre, el tipo de datos, etc. Conviene para ampliar revisar la sintaxis directamente desde el DBMS.

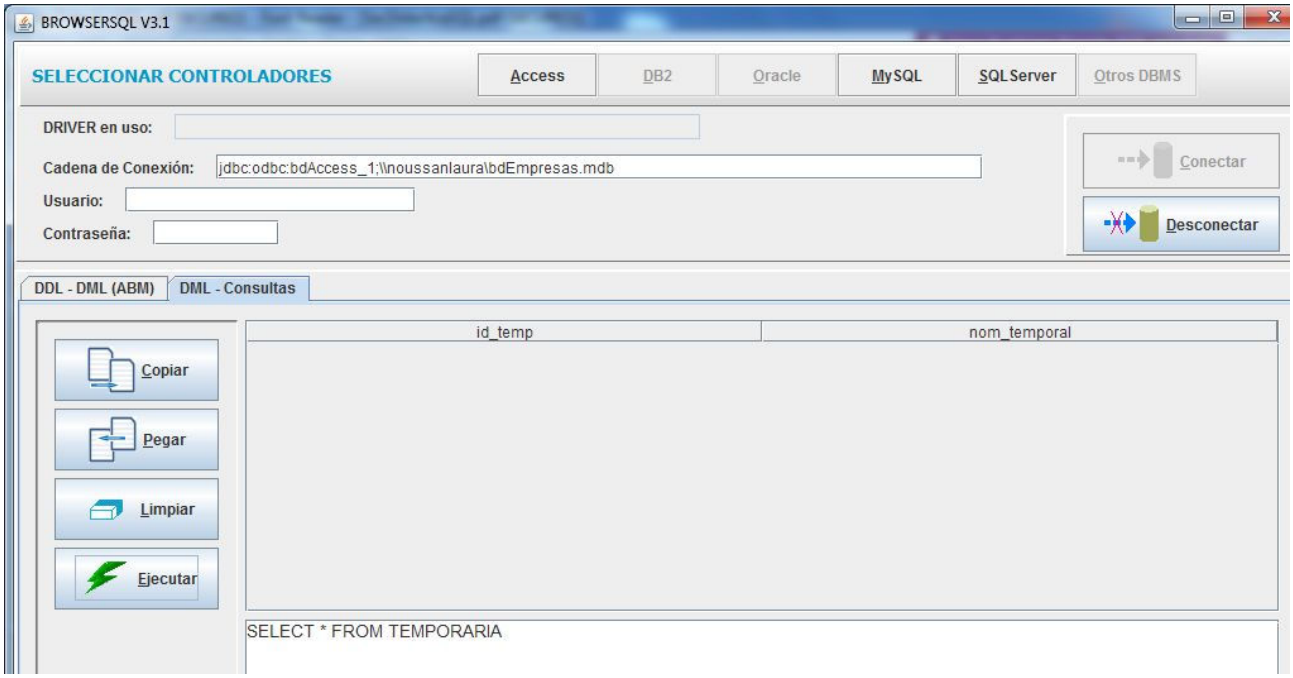
En la última sentencia podemos apreciar que eliminamos la columna fecha.

Sentencia RENAME

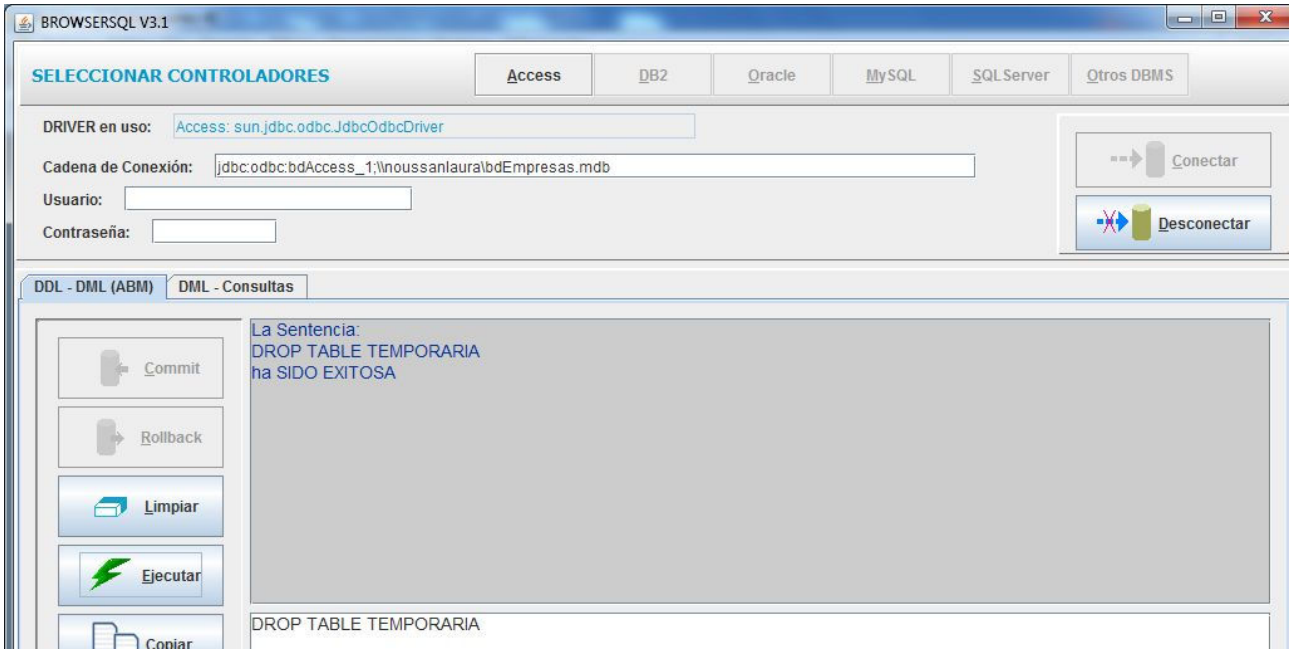
Si intentamos cambiar el nombre de la tabla anterior a Temporal2 en Access no podremos. No está contemplada dentro de este DBMS esta sentencia.

Sentencia DROP

Directamente eliminaremos esta tabla que no nos sirve para fines prácticos.

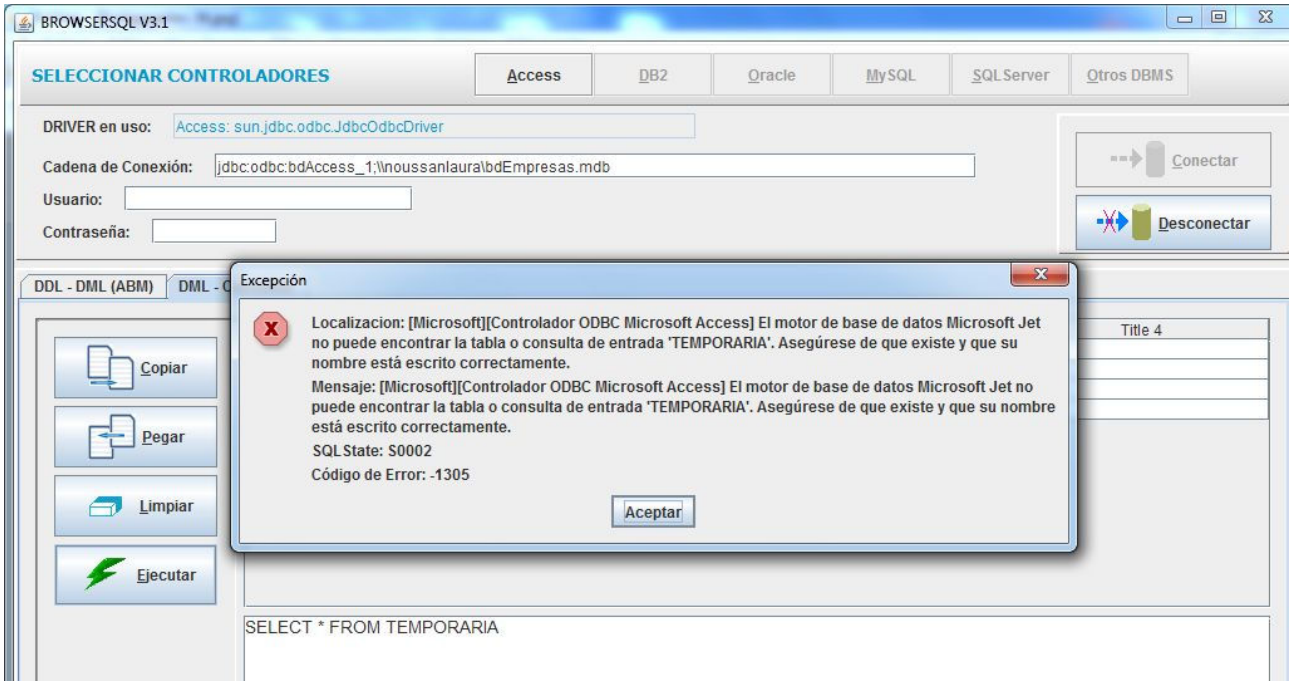


Vemos que la tabla existe, pero no tiene registros



Se elimina la tabla. En la siguiente figura volvemos a hacer un SELECT y el DBMS nos envía una Excepción SQL que indica que no encuentra el objeto

Cabe aclarar que el motor de base de datos de Access es el que envía a la aplicación dicho mensaje. Un aspecto que hace a otros DBMS mucho más poderosos es que justamente sus rutinas de Errores y Excepciones son mucho más específicas y descriptivas.



Diseño Físico de la Base de Datos Universidad para MySQL

El diseño físico para esta plataforma de DBMS todavía está pendiente.

Ambos diseños físicos se van a encontrar en el archivo Excel **Base Datos Universidad.xls** que puede descargarse del SiteWeb desde la solapa Material.