

Clase 1



Introducción a Bases de datos



Introducción a SQL



Introducción a Consultas

Base de datos

- Estructura en memoria secundaria que permite almacenar y manipular de manera concurrente grandes volúmenes de información.
- Múltiples usuarios pueden consultar y modificar simultáneamente la información contenida en la base de datos.
- El contenido debe mantenerse actualizado.

Estructura de una base de datos

- Un conjunto de **tablas**
- Cada tabla tiene un **nombre**
- Cada tabla está compuesta por **registros**
- Cada registro tiene **campos**
- Cada campo tiene un **nombre** y un **tipo**
- Un registro tiene una **llave primaria de acceso** (rápido)

Estructura de una base de datos

```
Tabla = cursos  
Campo = codigo varchar(10)  
Campo = curso varchar(20)  
Campo = estudiantes int  
PRIMARY KEY = codigo
```

cursos

registros

codigo	curso	estudiantes
"ISIS-1204"	"APO1"	23
"ISIS-1205"	"APO2"	25
"FISI-1012"	"Fisica1"	67
"FISI-1013"	"Fisica2"	78
"IIND-2101"	"Probabilidad"	56
"IIND-2102"	"Estadistica"	47

Estructura de una base de datos

```
Tabla = salones  
Campo = nombre varchar(32)  
Campo = capacidad int  
PRIMARY KEY = nombre
```

registros

salones

nombre	capacidad
"AU-306"	35
"AU-102"	30
"AU-201"	30
"W-560"	20
"O-101"	90
"O-102"	90
"R-201"	120

Operaciones básicas de una BD

- Crear una tabla
 - Descripción de sus campos (nombre y tipo)
- Insertar un registro
 - Tabla y valores de los campos
- Modificar un registro
 - Tabla, registro, modificación
- Eliminar un registro
 - Tabla, registro
- Buscar todos los registros con una característica

SQL: Structured Query Language

- Es un lenguaje estándar para manejar bases de datos.
- Conjunto simple de instrucciones textuales.
- Permite realizar todas las operaciones de manipulación de la información.
- Funciona con cualquier base de datos (SQL Server, ORACLE, Access, DB2, Derby).

Las instrucciones de SQL no hacen parte de los lenguajes de programación. Son instrucciones que llegan a la base de datos por algún medio.

SQL: Structured Query Language

- Agregar (crear) una tabla a una base de datos:

```
CREATE TABLE cursos  
(  
    codigo varchar(10),  
    curso varchar(20),  
    estudiantes int,  
    PRIMARY KEY ( codigo )  
)
```

cursos		
codigo	curso	estudiantes

SQL: Structured Query Language

- Agregar un registro a una tabla:

```
INSERT INTO cursos VALUES  
(  
    "ISIS-1204",  
    "APO1",  
    23  
)
```

cursos

codigo	curso	Estudiantes
"ISIS-1204"	"APO1"	23

SQL: Structured Query Language

- Agregar un registro a una tabla:

```
INSERT INTO cursos VALUES  
(  
    "FISI-1012",  
    "Fisica1",  
    98  
)
```

cursos

codigo	curso	Estudiantes
"ISIS-1204"	"APO1"	23
"FISI-1012"	"Fisica1"	98

SQL: Structured Query Language

- Agregar un registro a una tabla:

```
INSERT INTO cursos VALUES  
(  
    "ISIS-1205",  
    "APO2",  
    30  
)
```

cursos

codigo	curso	Estudiantes
"ISIS-1204"	"APO1"	23
"FISI-1012"	"Fisica1"	98

SQL: Structured Query Language

- Agregar un registro a una tabla:

```
INSERT INTO cursos VALUES  
(  
    "FISI-1013",  
    "Fisica2",  
    102  
)
```

cursos

codigo	curso	Estudiantes
"ISIS-1204"	"APO1"	23
"FISI-1012"	"Fisica1"	98

SQL: Structured Query Language

- Agregar un registro a una tabla:

```
INSERT INTO cursos VALUES  
(  
    "FISI-1014",  
    "Fisica3",  
    105  
)
```

cursos

codigo	curso	Estudiantes
"ISIS-1204"	"APO1"	23
"FISI-1012"	"Fisica1"	98

SQL: Structured Query Language

- Eliminar un registro de una tabla:

```
DELETE FROM cursos WHERE codigo = "ISIS-1204"
```

Expresión que permite localizar el registro que se quiere eliminar

cursos

codigo	curso	Estudiantes
"FISI-1012"	"Fisica1"	98

SQL: Structured Query Language

- Modificar un registro de una tabla:

```
UPDATE cursos  
SET estudiantes = 50  
WHERE codigo = "FISI-1012"
```

cursos

codigo	curso	Estudiantes
"FISI-1012"	"Fisica1"	50

- Si hay varias modificaciones, se separan por coma

SQL: Structured Query Language

- Recuperar información de una base de datos:

SELECT **estudiantes**

FROM **cursos**

WHERE **codigo** = "FISI-1012"

50



Retorna los estudiantes de un solo registro, el del curso cuyo
codigo es "FISI-1012"

SQL: Structured Query Language

- Recuperar información de una base de datos:

SELECT curso, estudiantes

FROM cursos

WHERE codigo = "FISI-1012"

"Fisica1"	50
-----------	----



Retorna un conjunto de campos del registro cuyo código es "FISI-1012"

SQL: Structured Query Language

- Recuperar información de una base de datos:

SELECT *

FROM cursos

WHERE codigo = "FISI-1012"

"FISI-1012"	"Fisica1"	50
-------------	-----------	----



Retorna TODOS los campos del registro cuyo código es "FISI-1012"

SQL: Structured Query Language

- Buscar un subconjunto de campos de un conjunto de registros:

```
SELECT curso, estudiantes
```

```
FROM cursos
```



Retorna el curso y el número de estudiantes de TODOS los registros (no hay WHERE, es decir, no hay condición de selección)

Ejercicio

- Crear una tabla llamada resultados que tenga tres campos:
 - nombre: cadena de máximo 32 caracteres
 - ganados: valor entero
 - perdidos: valor entero
- La llave primaria es el campo nombre.

Ejercicio

- Insertar en la tabla anterior un registro para indicar que el jugador llamado “Barbanegra” lleva 10 partidos ganados y 5 perdidos

Ejercicio

- Insertar en la tabla anterior un registro para indicar que el jugador llamado “Drake” lleva 0 partidos ganados y 25 perdidos

Ejercicio

- Insertar en la tabla anterior un registro para indicar que el jugador llamado “Nelson” lleva 20 partidos ganados y 2 perdidos

Ejercicio

- Eliminar de la tabla de resultados al jugador con nombre “Drake”

Ejercicio

- Aumentar en uno el número de partidos ganados del jugador “Barbanegra”

Ejercicio

- Multiplicar por dos el número de partidos perdidos del jugador “Nelson”

¿Qué es Access?

Microsoft Access es un programa de Microsoft Office cuyo propósito es la gestión de bases de datos.

Microsoft Access permite:

- Crear y manejar bases de datos
- Recopilación masiva de datos
- Crear formularios

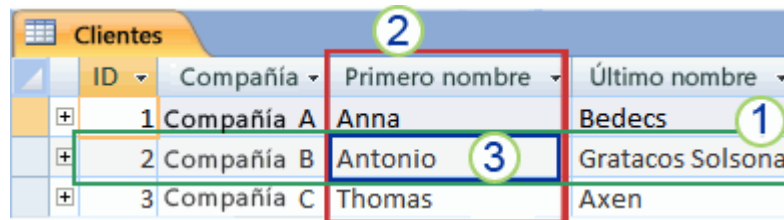
¿Por qué Access?

Al ser un programa de Microsoft Office, ya lo tenemos instalado! Además, su manejo nos puede resultar intuitivo si ya conocemos Excel, Word o PowerPoint.

Lo utilizaremos para entender mejor la estructura de las bases de datos y sus consultas antes de pasar nuevamente a Java.

Tablas

Las Tablas son elementos fundamentales para una base de datos. Aquí es donde se recopila la información de la base de datos. Cada una de estas tablas se caracteriza por tener tanto un Registro (filas) como distintos Campos (columnas).



ID	Compañía	Primero nombre	Último nombre
1	Compañía A	Anna	Bedecs
2	Compañía B	Antonio	Gratacos Solsona
3	Compañía C	Thomas	Axen

- 1.Un registro: contiene datos específicos, como información acerca de un determinado empleado o un producto.
- 2.Un campo: contiene datos sobre un aspecto del asunto de la tabla, como el nombre o la dirección de correo electrónico.
3. Un valor de campo: cada registro tiene un valor de campo.

Consultas

- Con las consultas es más fácil ver, agregar, eliminar y cambiar datos en una base de datos. Sus usos principales son:
 - Encontrar datos específicos rápidamente, filtrándolos según criterios.
 - Calcular o resumir datos.
- Para las consultas se usa el lenguaje SQL (Structured Query Language). Se divide en:
 - Data Manipulation Language(DML) { SELECT,UPDATE,DELETE}
 - Data Definition Language (DDL) { (CREATE|ALTER)DATABASE,
 - (CREATE|ALTER|DROP)
 - TABLE, (CREATE|DROP) INDEX }

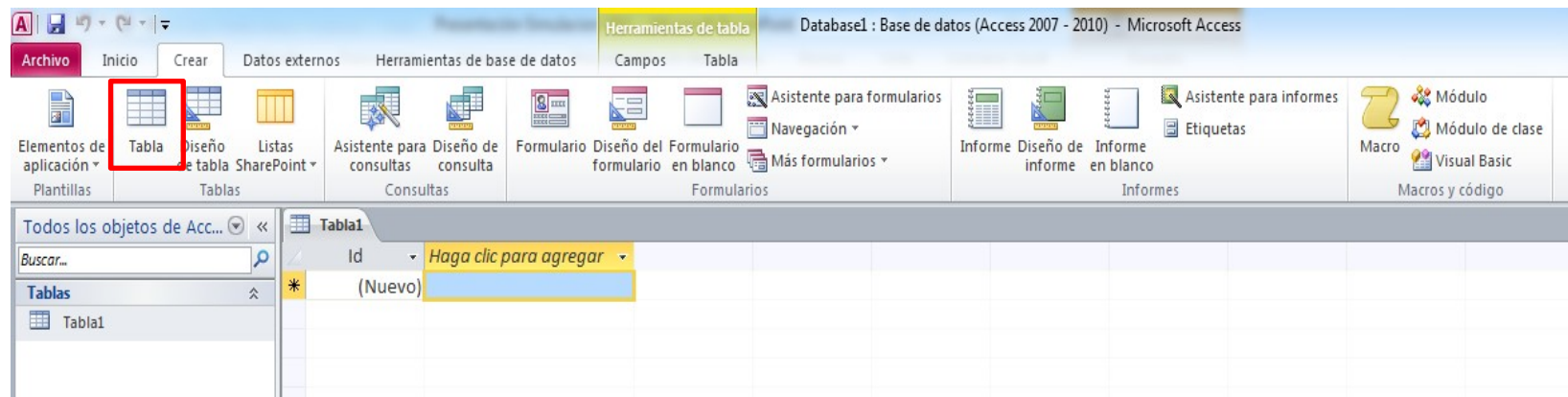
Consultas

- Con una consulta, se puede extraer información de diversas tablas y juntarla para mostrarla en un formulario o informe.
- Una consulta puede servir para pedir resultados de datos de la base de datos, para llevar a cabo una acción relativa a los datos o para ambas cosas.
- Sirve para obtener una respuesta a una pregunta sencilla, efectuar cálculos, combinar datos de distintas tablas o agregar, cambiar o eliminar datos de una base de datos.

Tablas

Creación

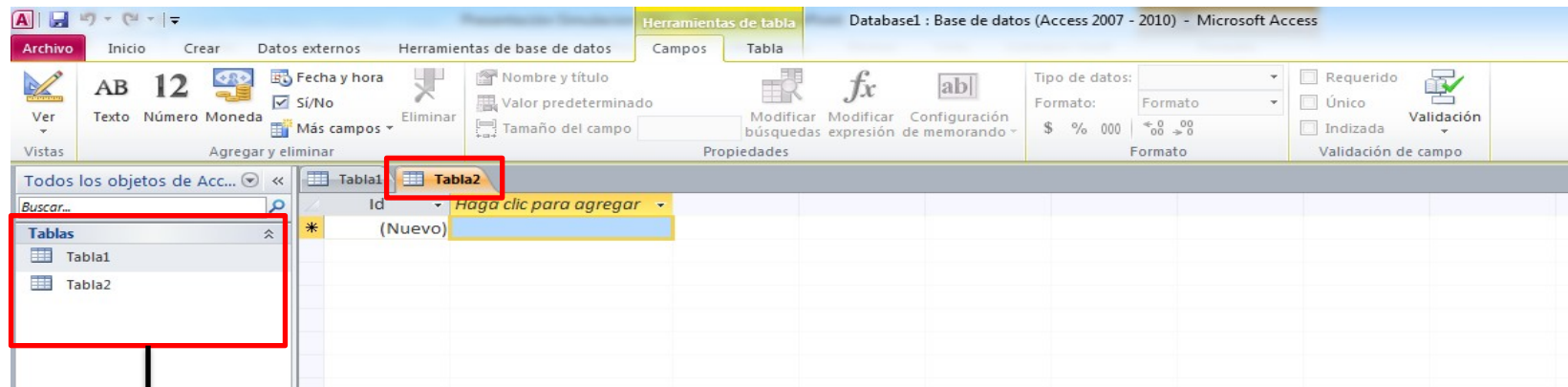
Seleccionamos la opción “Tabla”, en la pestaña “Crear”.



Tablas

Creación

Se crea una nueva Tabla llamada “Tabla2”

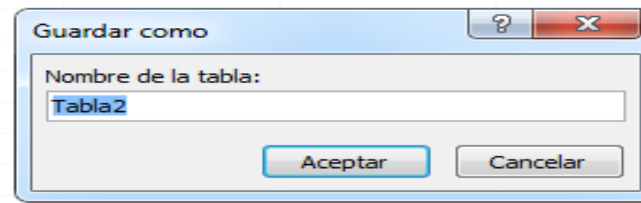
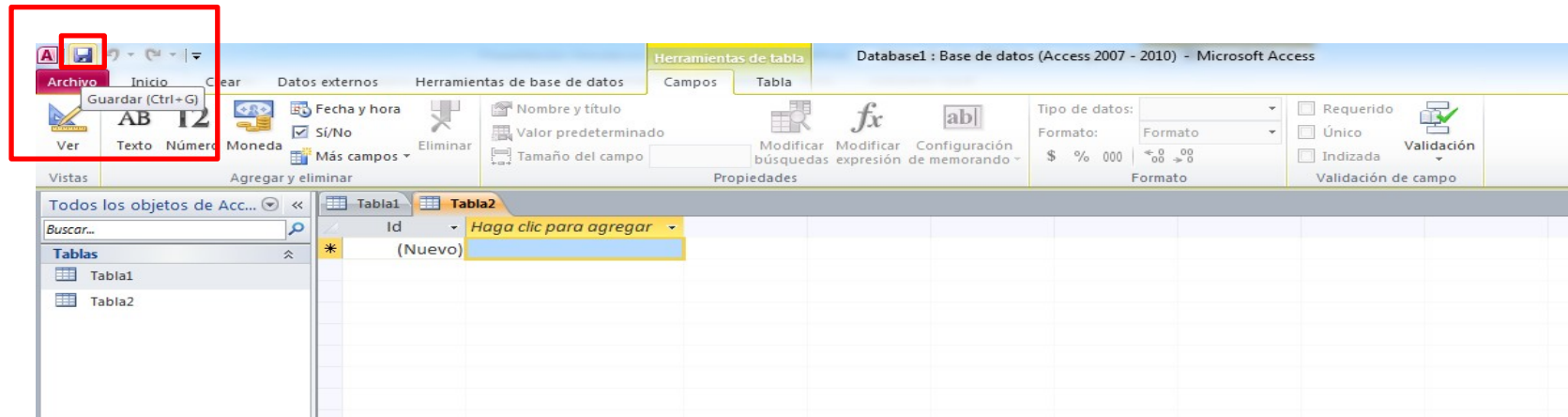


En esta ventana se
pueden apreciar las
diversas tablas existentes.

Tablas

Creación

Posteriormente, se debe guardar la Tabla y asignarle un nombre

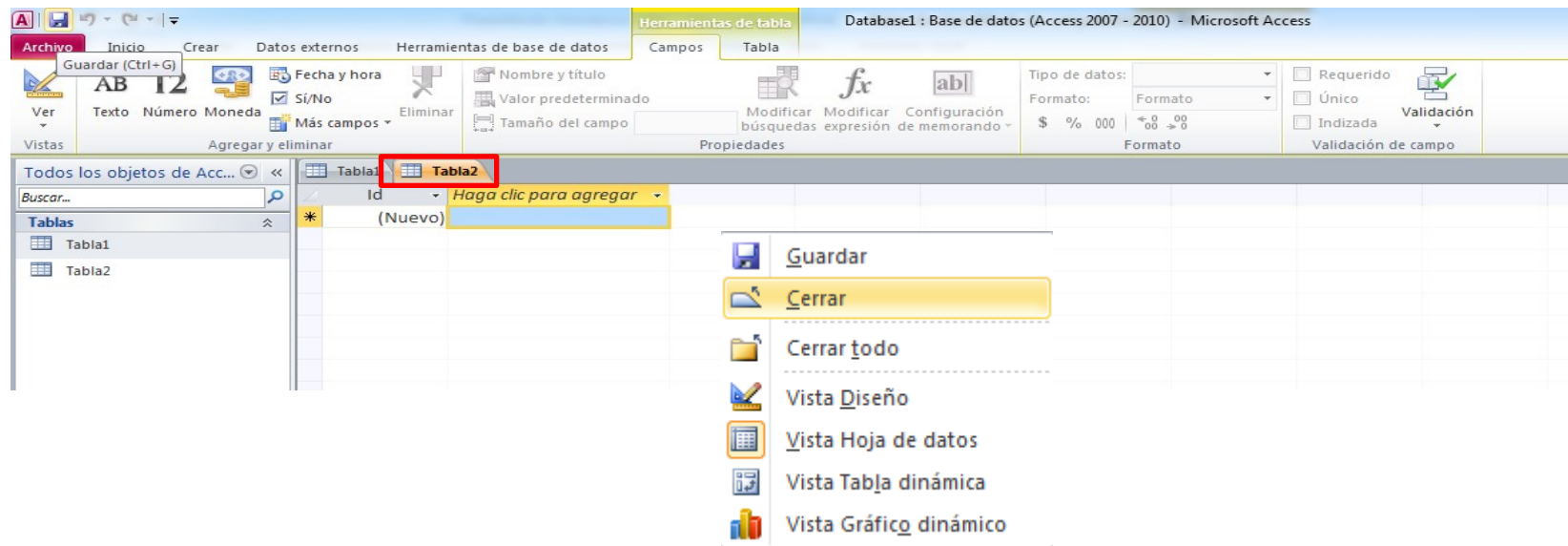


Tablas

Creación

Si se desea cambiar el nombre de una tabla ya existente, esta se debe cerrar en primera instancia.

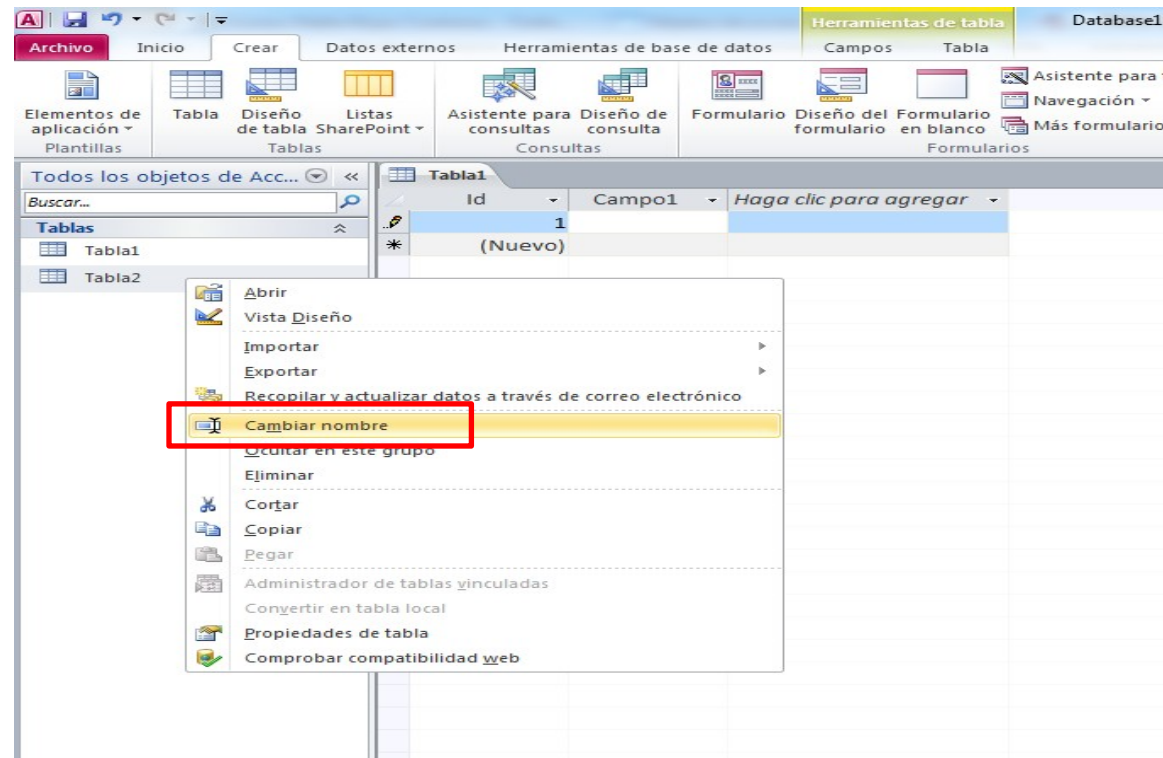
Para cerrar una tabla, presione click Derecho sobre la Pestaña de la tabla deseada. Seleccione “Cerrar”



Tablas

Creación

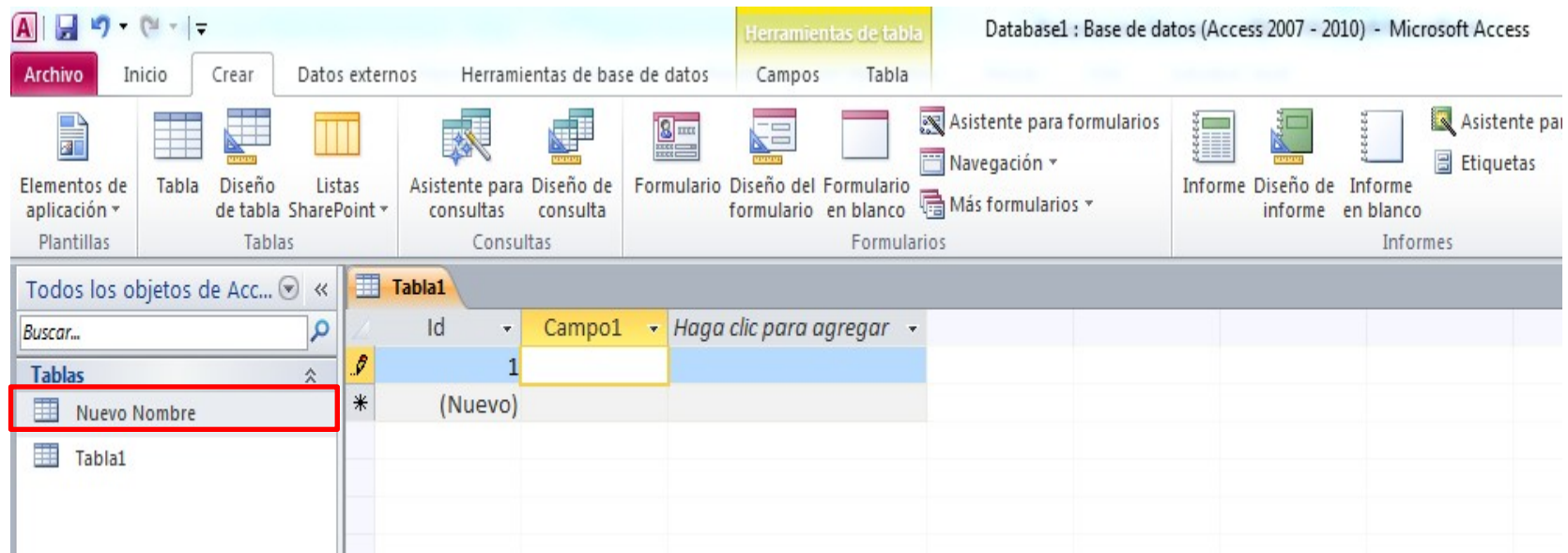
Posteriormente, oprima click derecho sobre la tabla que desea modificar. Seguidamente, seleccione la opción “Cambiar nombre”.



Tablas

Creación

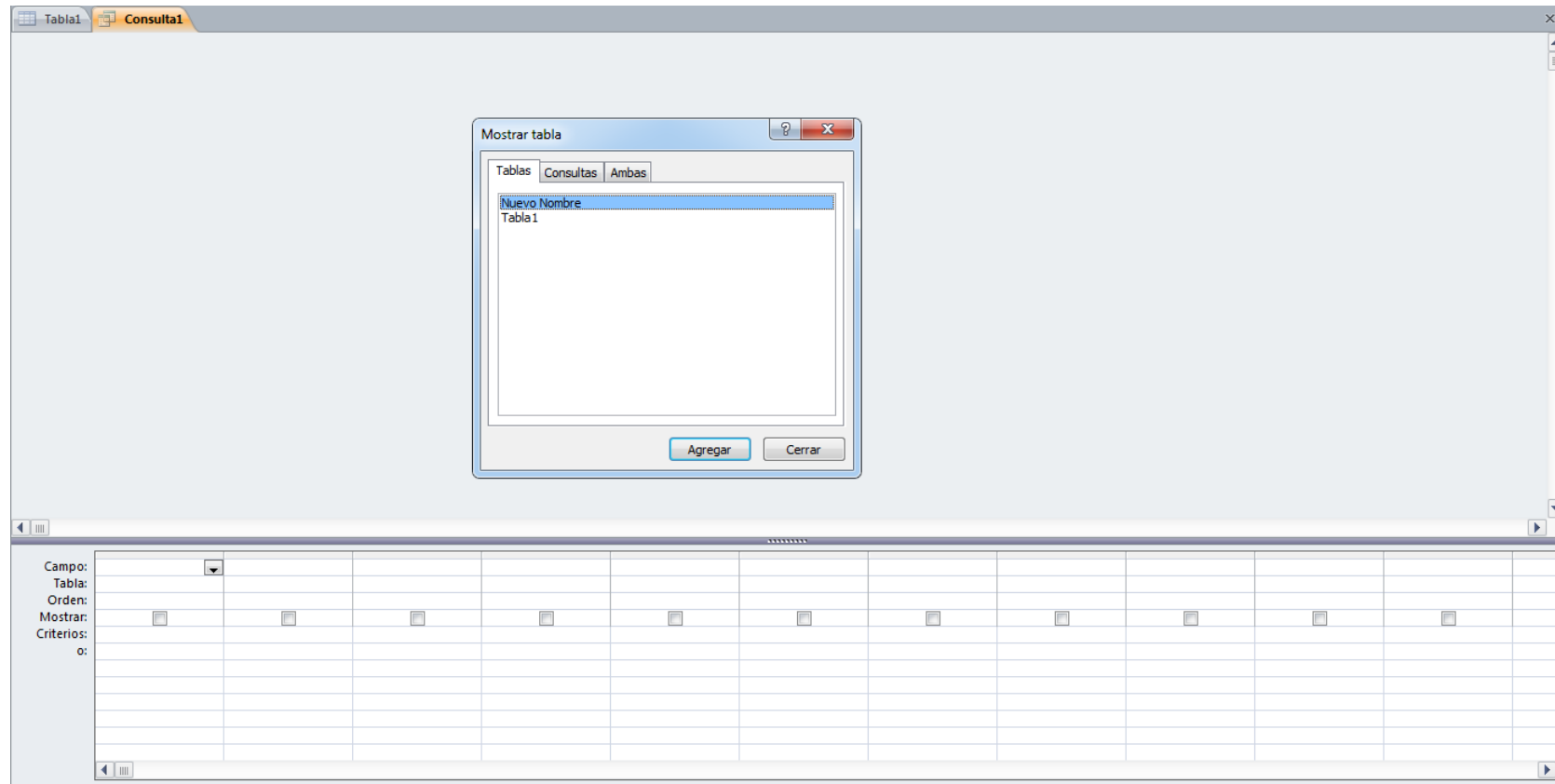
La tabla ahora cambia de nombre



Consultas

Creación

En la pestaña crear se selecciona la opción de crear una consulta presionando “Diseño de Consulta”



Consultas

Creación – Consulta de selección

Estando en la vista diseño.

En el cuadro Mostrar tabla, en la pestaña Tablas, haga doble clic en la tabla de la cual desee obtener la información y luego cierre el cuadro de diálogo.

Supongamos que en la tabla que seleccionó tenemos los campos Nombre y Promedio. Haga doble clic en Nombre y Promedio para agregar estos campos a la cuadrícula de diseño de la consulta.

En la ficha Diseño, haga clic en Ejecutar. La consulta se ejecuta y muestra una lista de los estudiantes con sus respectivos nombres y promedios

Herramientas - Vistas

Access cuenta con dos esquemas diferentes en los que el usuario puede introducir y manejar la información para cada una de sus tablas.

- Vista Hoja de Datos
- Vista Diseño

Herramientas - Vistas

- Vista Hoja de Datos

En esta opción:

- El usuario puede acceder directamente a los datos de las tablas.
- El usuario puede modificar los datos de las tablas
 - Agregar o quitar registros
 - Agregar o quitar campos
 - Modificar valores y los tipos de campo

Herramientas - Vistas

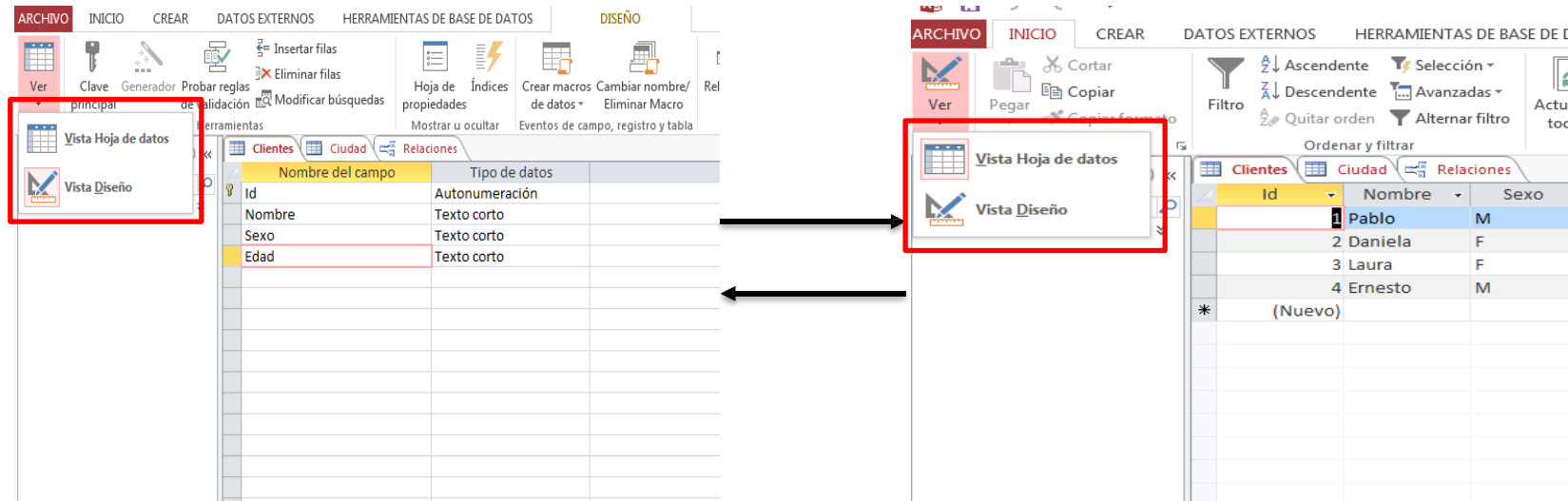
- Vista Diseño

En esta opción:

- El usuario puede cambiar fácilmente el nombre de los campos
- El usuario puede modificar fácilmente el tipo de los valores de los campos
- El usuario puede acceder a las propiedades del campo

Herramientas - Vistas

Una manera de cambiar el tipo de Vista es seleccionando la opción “Ver”, que se encuentra en la pestaña Inicio



Tipos de Datos

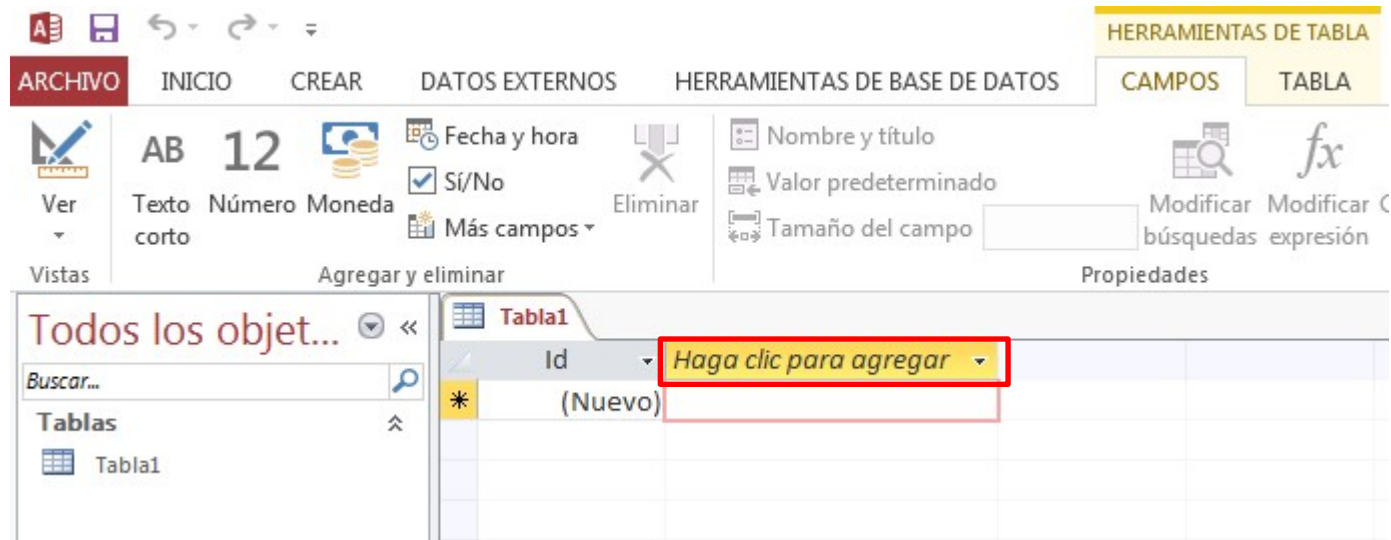
Al igual que en Excel y en VBA, en Access existen diferentes tipos de datos.

Tipos de Datos:

- Texto
- Numérico
- Fecha / Hora
- Moneda
- Sí/ No
- Entre otros

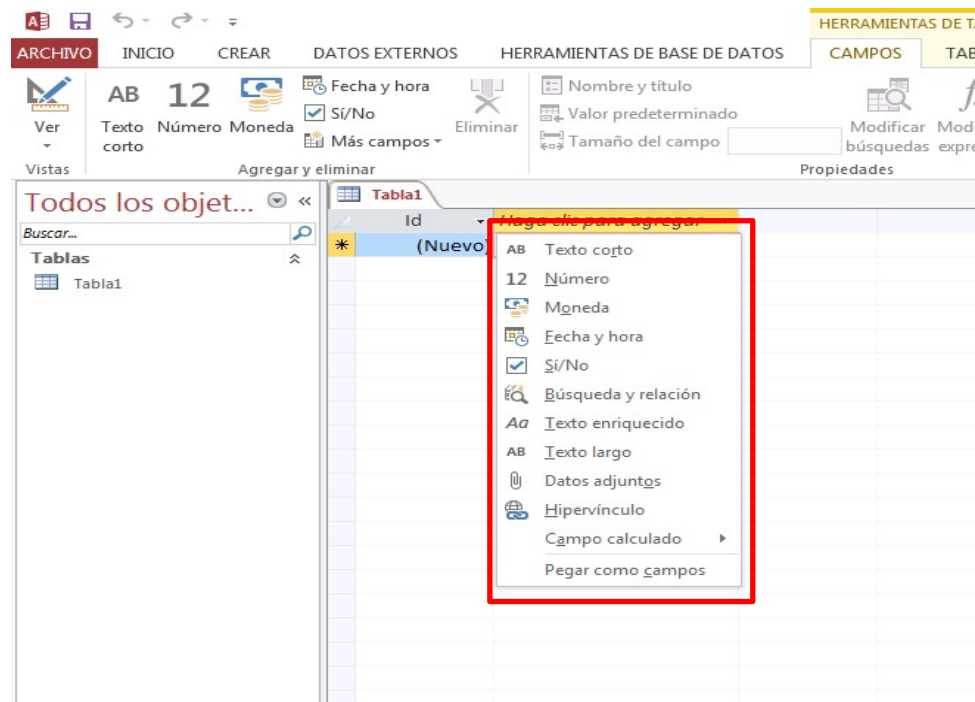
Tipos de Datos

Para seleccionar el tipo de dato a utilizar en cada uno de los campos de las tablas, haga clic sobre la región indicada a continuación:



Tipos de Datos

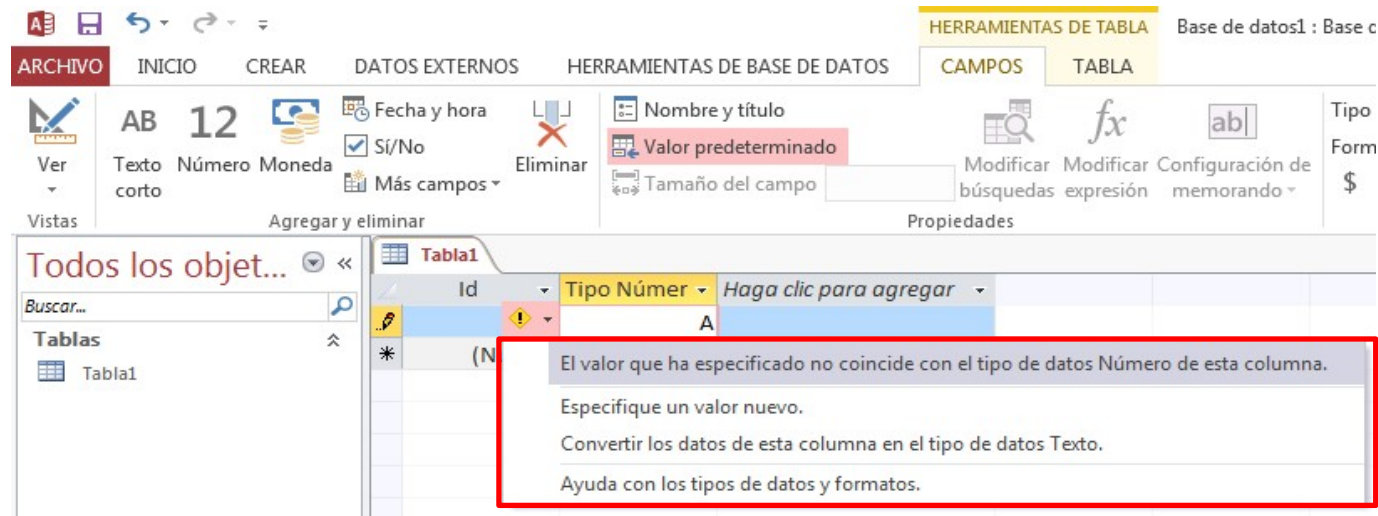
Como resultado, se extiende una ventana con los posibles tipos de valores que pueden adquirir los valores del campo



Tipos de Datos

Si el tipo de valor ingresado por el usuario no coincide con el del campo, Access informará un error.

Aparecerá una ventana en donde la herramienta le brindará soluciones al Usuario.




Clave Principal

Una clave principal es un campo o conjunto de campos en donde se registran valores únicos en las tablas. En otras palabras, la clave principal determina cuales datos no cuentan con duplicidad.


Clave Principal

Para crear una Clave, primero se debe ingresar a Vista Diseño. Ahora, procedemos a establecer cuál o cuáles campos poseen valores únicos para cada registro.

Tabla1		
	Nombre del campo	Tipo de datos
	Id	Autonumeración
	Nombre	Número
	Sexo	Texto corto
	Edad	Texto corto



Clave Principal

Predeterminadamente, Access creará la Clave Principal en el primer campo (Id).

Tabla1		
	Nombre del campo	Tipo de datos
	Id	Autonumeración
	Nombre	Número
	Sexo	Texto corto
	Edad	Texto corto

Clave Principal

Si se desea cambiar el campo de la Clave Principal, tan sólo se debe hacer clic derecho a la izquierda del campo deseado

Tabla1		
	Nombre del campo	Tipo de datos
	Id	Autonumeración
	Nombre	Número
	Sexo	Texto corto
	Edad	Texto corto

Clave Principal

Así aparecerá la opción de fijar la Clave en la nueva posición.

Tabla1		
	Nombre del campo	Tipo de datos
	Id	Autonumeración
🔑	Nombre	Número
	Sexo	Texto corto
	Edad	Texto corto

Propiedades del Campo

Cada campo de cada tabla posee una serie de diversas características. Estas características le proporcionan al usuario control adicional sobre la forma como se comporta y funciona el campo.

Propiedades del campo

En primera instancia, para acceder a las Propiedades del Campo se debe seleccionar la Vista Diseño.

Así, aparecerá en la parte inferior de la pantalla un panel en donde se especifican las propiedades del campo

The screenshot shows a database design tool interface. At the top, there are tabs for 'Clientes', 'Ciudad', and 'Relaciones'. Below these, a table lists fields: 'Id' (Autonumeración), 'Nombre' (Texto corto), 'Sexo' (Texto corto), and 'Edad' (Texto corto). The 'Edad' field is selected, and a red box highlights the 'Propiedades del campo' (Field Properties) panel at the bottom. This panel has two tabs: 'General' and 'Búsqueda'. The 'General' tab is active, showing various properties for the 'Edad' field.

Propiedades del campo	
Tamaño del campo	255
Formato	
Máscara de entrada	
Título	Edad
Valor predeterminado	
Regla de validación	
Texto de validación	
Requerido	No
Permitir longitud cero	Sí
Indexado	No
Compresión Unicode	Sí
Modo IME	Sin Controles
Modo de oraciones IME	Nada
Alineación del texto	General

Propiedades del campo

En la pestaña General, algunas de las características más influyentes que el usuario puede manipular son

Tamaño

Formato

Título

Valor

entre otras...

Propiedades del campo

General	Búsqueda
Tamaño del campo	255
Formato	
Máscara de entrada	
Título	Edad
Valor predeterminado	
Regla de validación	
Texto de validación	
Requerido	No
Permitir longitud cero	Sí
Indexado	No
Compresión Unicode	Sí
Modo IME	Sin Controles
Modo de oraciones IME	Nada
Alineación del texto	General

Filtro de Datos

Al igual que en las tablas de Excel, Access permite filtrar los datos de los campos deseados.

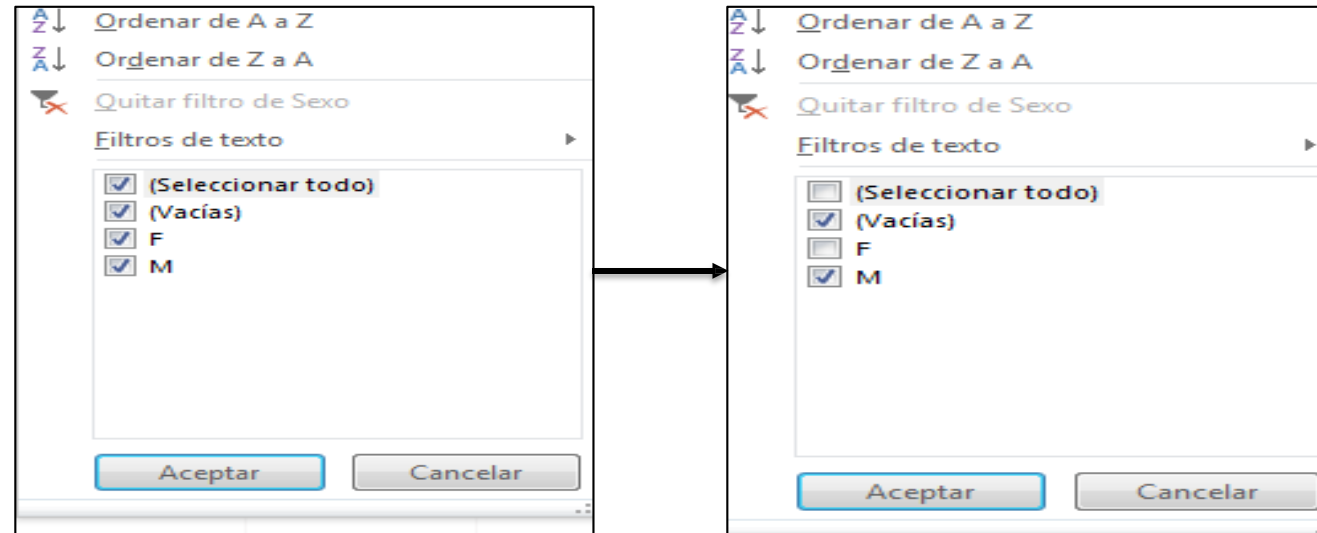
Para esto, nos paramos sobre el campo que deseamos filtrar. Posteriormente, seleccionamos la opción “Filtro” en la pestaña de Inicio



El campo se selecciona haciendo clic sobre el nombre del campo




Filtro de Datos

Si se desea filtrar por sexo, por ejemplo, se selecciona en la ventana el género que se desea filtrar.



Filtro de Datos

Al hacer clic en “Aceptar”, se filtrarán los datos de la tabla según se establezca por parámetro.

 Clientes  Ciudad  Relaciones				
	Id ▼	Nombre ▼	Sexo ▼	Edad ▼
	1	Pablo	M	45
	4	Ernesto	M	22
*	(Nuevo)			

Relación Entre Tablas

Cada tabla puede estar relacionada con una o más tablas. Esto último, mediante los datos que posee cada una de ellas

Al crear relaciones entre diversas tablas, cada uno de los datos de las tablas se debe relacionar con por lo menos un dato de las demás tablas.

Relación Entre Tablas

Creación de Relaciones

Imaginemos que contamos con las siguientes dos tablas

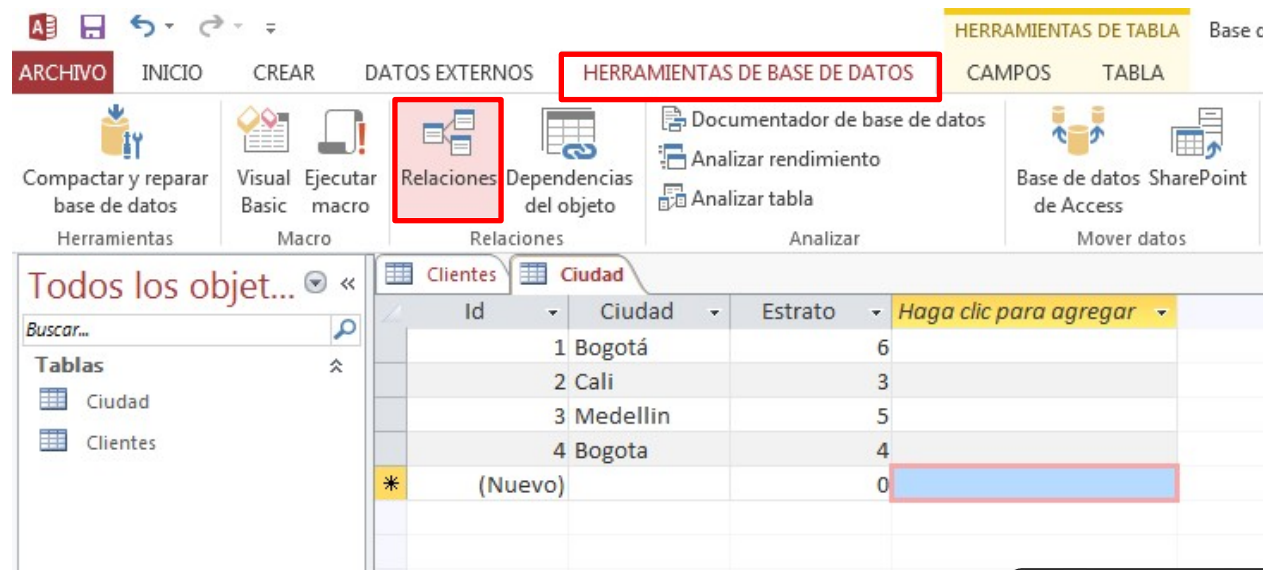
Clientes		Ciudad		
Id	Nombre	Sexo	Edad	
1	Pablo	M	45	
2	Daniela	F	26	
3	Laura	F	30	
4	Ernesto	M	22	

Clientes		Ciudad	
Id	Ciudad	Estrato	
1	Bogotá	6	
2	Cali	3	
3	Medellin	5	
4	Bogota	4	

Relación Entre Tablas

Creación de Relaciones

Para crear la relación, seleccione la opción “Relaciones”, en la pestaña “Herramientas de bases de datos”

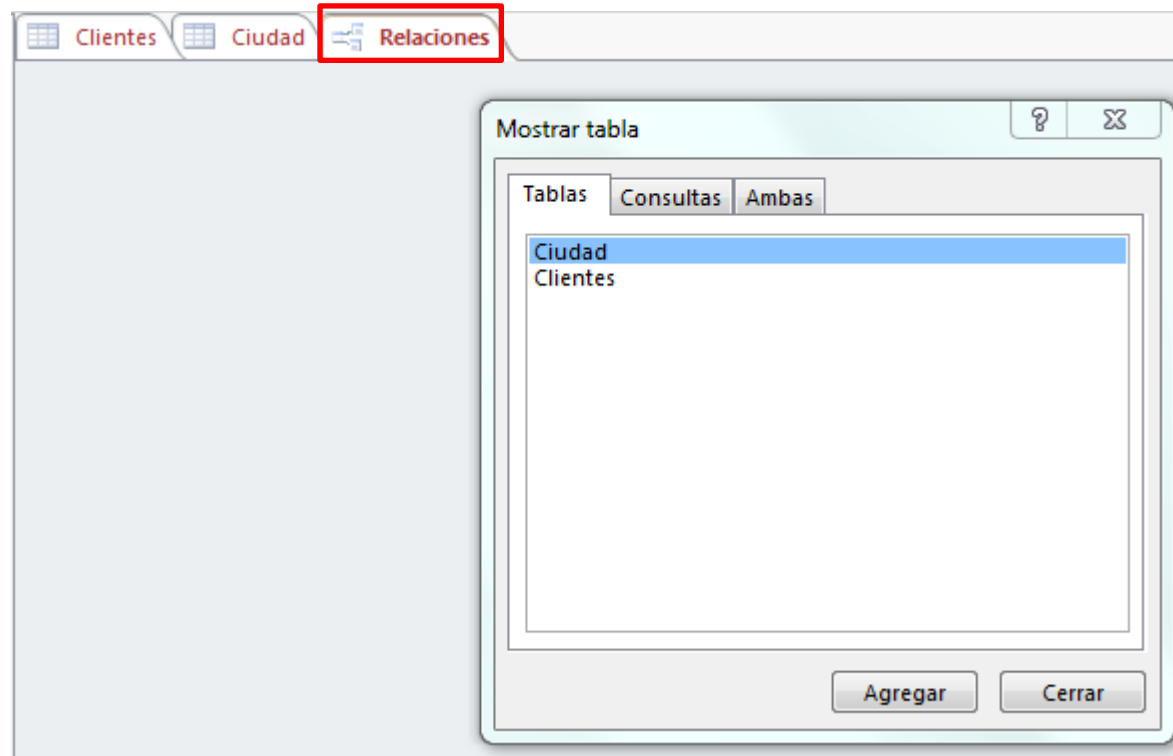


Relaciones
Define cómo se relacionan los datos en las tablas, como hacer coincidir campos de identificación o campos de nombre en diferentes tablas.

Relación Entre Tablas

Creación de Relaciones

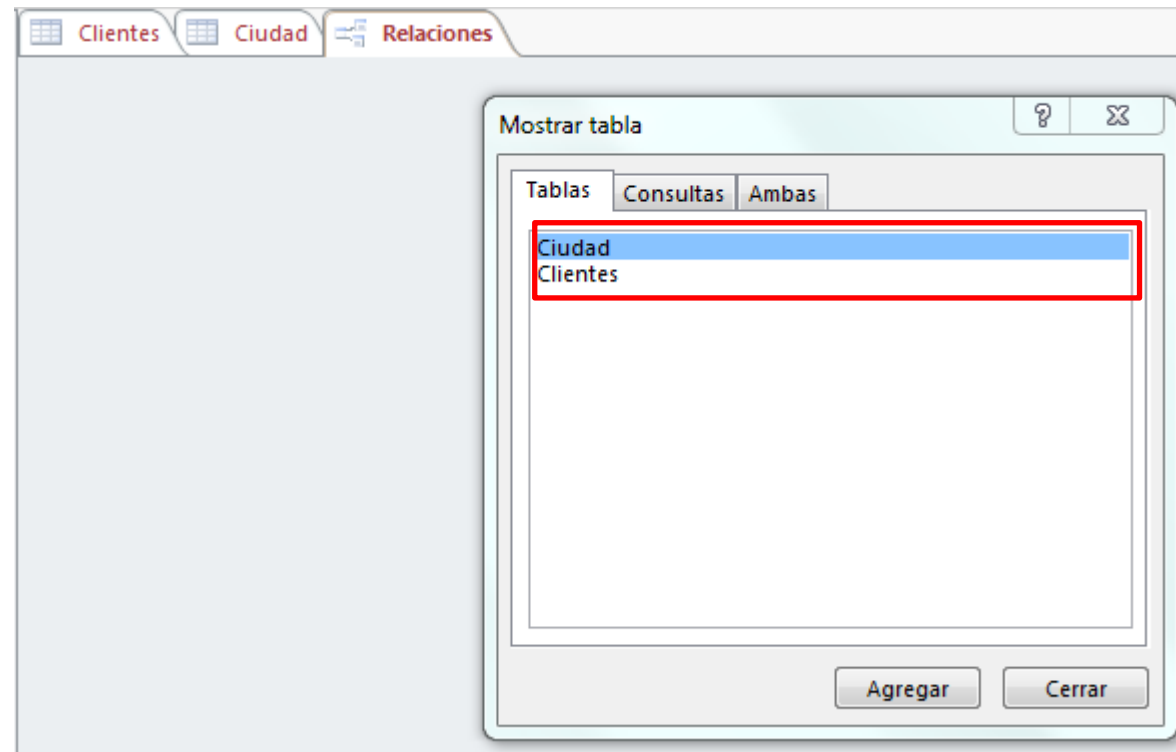
Se creará una nueva pestaña llamada Relaciones



Relación Entre Tablas

Creación de Relaciones

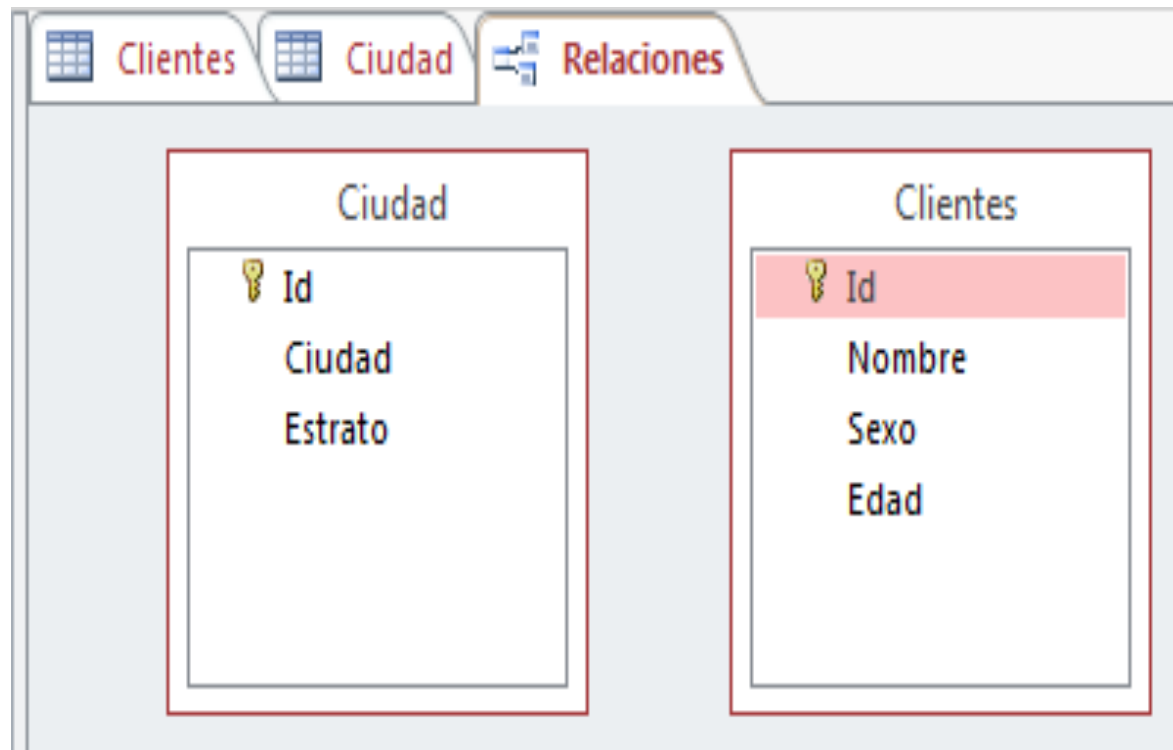
En la ventana aparecerán las diferentes tablas existentes



Relación Entre Tablas

Creación de Relaciones

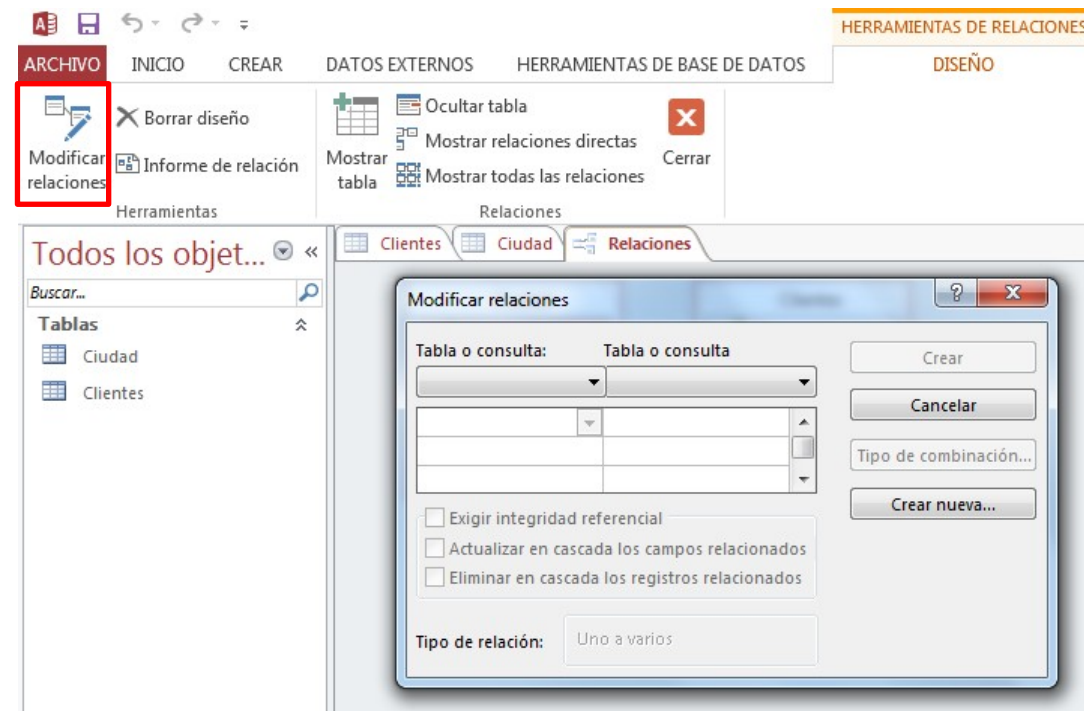
Por cada tabla que se seleccione y agregue, en la pestaña Relaciones aparecerá una ventana que hace referencia a la tabla deseada.



Relación Entre Tablas

Creación de Relaciones

Seleccione la opción “Modificar relaciones” para que aparezca una ventana en donde se establecerán las conexiones entre las tablas.



Relación Entre Tablas

Creación de Relaciones

Seleccione las respectivas tablas sobre las cuales se realizará la relación. Posteriormente, seleccione los campos de cada tabla que desea relacionar.

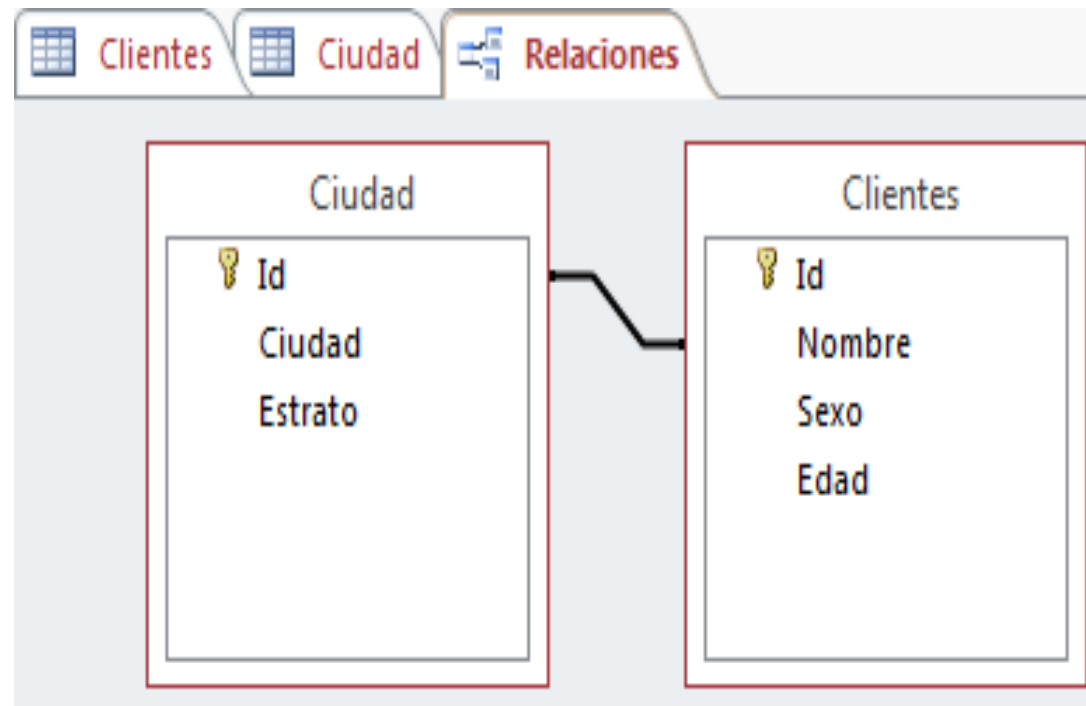
The screenshot shows a dialog box titled "Modificar relaciones" (Modify relationships). It contains the following elements:

- Table selection:** Two dropdown menus labeled "Tabla o consulta:". The first is set to "Ciudad" and the second to "Clientes".
- Field selection:** Two columns of dropdown menus. The first column has "Id" selected, and the second column has "Nombre" selected.
- Buttons:** "Aceptar" (Accept), "Cancelar" (Cancel), "Tipo de combinación..." (Relationship type...), and "Crear nueva..." (Create new...).
- Options:** Three checkboxes:
 - ☐ Exigir integridad referencial (Require referential integrity)
 - ☐ Actualizar en cascada los campos relacionados (Cascade update related fields)
 - ☐ Eliminar en cascada los registros relacionados (Cascade delete related records)
- Relationship type:** A dropdown menu labeled "Tipo de relación:" set to "Uno a varios" (One to many).

Relación Entre Tablas

Creación de Relaciones

Ahora aparecerá una conexión entre las dos ventanas. Dicha conexión representa la relación entre el elemento de la primera tabla con el de la segunda



Introducción a Consultas

Usando SQL

- **SELECT**
 - Extrae todas las columnas de la tabla usando (*)
 - Extrae una(s) columna(s) de la tabla por su(s) nombres(s)
- **FROM**
 - Define de dónde se obtienen los datos (a partir de tablas)
 - Uso de joins para mezclar tablas

- Ejemplo
Store_Information

Table *Store_Information*

store_name	Sales	Date
Los Angeles	\$1500	Jan-05-1999
San Diego	\$250	Jan-07-1999
Los Angeles	\$300	Jan-08-1999
Boston	\$700	Jan-08-1999

SELECT store_name FROM



store_name
Los Angeles
San Diego
Los Angeles
Boston

Introducción a Consultas

Usando SQL

- WHERE
 - Filtra por tuplas
 - Operadores Permitidos
 - =, <>, >, <, <=, >=
 - BETWEEN(i)
 - IN(ii)

(i)

```
SELECT *  
FROM Store_Information  
WHERE Date BETWEEN 'Jan-05-1999' AND 'Jan-07-1999'
```

store_name	Sales	Date
Los Angeles	\$1500	Jan-05-1999
San Diego	\$250	Jan-07-1999

Table *Store_Information*

store_name	Sales	Date
Los Angeles	\$1500	Jan-05-1999
San Diego	\$250	Jan-07-1999
Los Angeles	\$300	Jan-08-1999
Boston	\$700	Jan-08-1999

Introducción a Consultas

Usando SQL

- WHERE
 - Filtra por tuplas
 - Operadores Permitidos
 - =, <>, >, <, <=, >=
 - BETWEEN(i)
 - IN(ii)

(ii)

```
SELECT *  
FROM Store_Information  
WHERE store_name IN ('Los Angeles', 'San Diego')
```

store_name	Sales	Date
Los Angeles	\$1500	Jan-05-1999
San Diego	\$250	Jan-07-1999
Los Angeles	\$300	Jan-08-1999

Table *Store_Information*

store_name	Sales	Date
Los Angeles	\$1500	Jan-05-1999
San Diego	\$250	Jan-07-1999
Los Angeles	\$300	Jan-08-1999
Boston	\$700	Jan-08-1999

Clase 2: Instrucciones SQL de no agrupamiento



Objetivos



Revisión de conceptos anteriores



Uso de operadores en la instrucción WHERE



Uso de funciones de no agrupamiento



Ordenamiento de datos

Objetivos

- Revisar los conceptos anteriores: tablas, bases de datos, SQL, principales sentencias
- Entender y analizar el uso de distintas funciones de no agrupamiento
- Entender y analizar los operadores condicionales de la sentencia WHERE
- Aprender y aplicar el uso de la sentencia ORDER BY

Revisión de Conceptos

Introducción

Tablas

- Arreglos de datos organizadas en filas y columnas.

Bases de Datos

- Colección de tablas en las que se almacena un conjunto específico de datos estructurados.

SQL

- Lenguaje de programación estándar para el acceso y modificación de bases de datos.

Revisión de Conceptos

Introducción

Select

- Permite seleccionar las columnas a usar de una base de datos.

From

- Permite seleccionar la base de datos de donde salen los datos.

Where

- Permite condicionar los datos mostrados.

Operadores en la instrucción WHERE

La instrucción WHERE nos permite utilizar condicionales para establecer criterios de decisión sobre los datos que se mostraran en las consultas que se creen. Para esto se pueden utilizar una serie de operadores condicionales que permiten filtrar los datos, durante el curso se analizarán y utilizarán las siguientes.

- Operadores lógicos.
- And
- Or
- Like
- Between
- In

Operadores en la instrucción WHERE

Descripción:

Existen seis tipos de operadores lógicos que se presentan en la tabla a continuación:

Operador	Descripción	Sintaxis
=	Igual a	WHERE nombre_columna = valor
≠	Diferente de	WHERE nombre_columna ≠ valor
>	Mayor que	WHERE nombre_columna > valor
<	Menor que	WHERE nombre_columna < valor
≥	Mayor o igual a	WHERE nombre_columna ≥ valor
≤	Menor o igual a	WHERE nombre_columna ≤ valor

Operadores en la instrucción WHERE

AND

Descripción:

Permite filtrar datos basado en más de una condición donde TODAS las condiciones se cumplan.

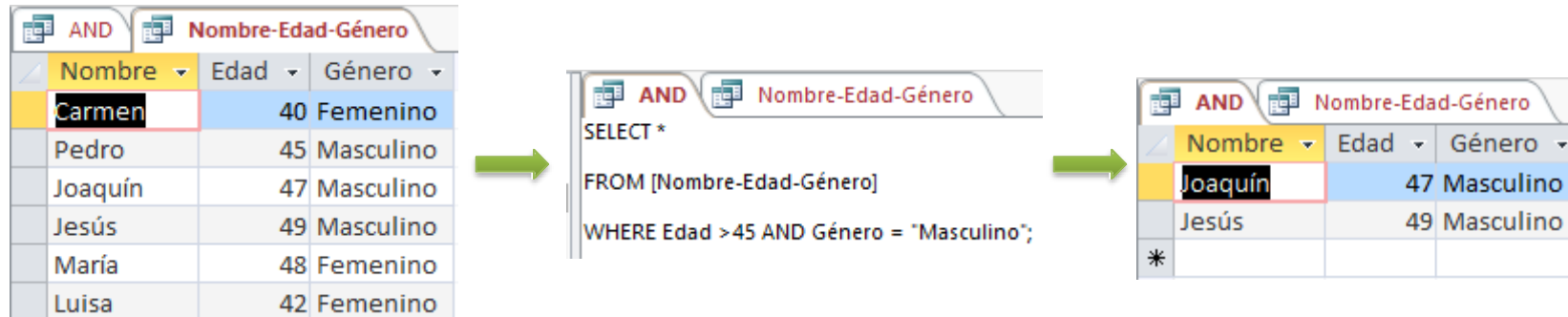
Sintaxis:

SELECT nombre_columna, ...

FROM nombre_tabla

WHERE condición1 AND condición2 AND ...;

Ejemplo:



Operadores en la instrucción WHERE

OR

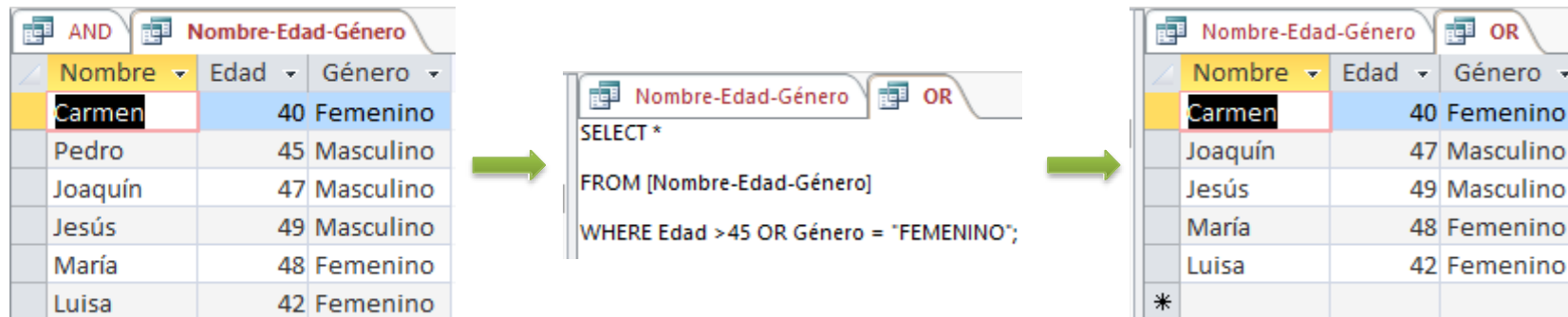
Descripción:

Permite filtrar datos basado en más de una condición donde ALGUNAS de las condiciones se cumpla.

Sintaxis:

```
SELECT nombre_columna, ...  
FROM nombre_tabla  
WHERE condición1 OR condición2 OR ...;
```

Ejemplo:



Operadores en la instrucción WHERE

Wildcards

Descripción:

Los *wildcards* permiten substituir diferentes tipos de caracteres dentro de un campo.

Carácter	Descripción	Ejemplo	Resultado
*	Sustituye un número indeterminado de caracteres.	A* *Z *O*	Andrés, A1, Alto, ... Arroz, Oz, ... Caoba, Oveja, Cetro, ...
?	Sustituye un único carácter.	?ed M?d No?	Ted, Led, Red, ... Mid, Mod, Med, ... Nox, Not, Nod, ..
#	Sustituye un carácter numérico por un número indeterminado.	#23 12#34 12#	123, 223, 323, 423, ... 12134, 12234, 12334, 12434, ... 121, 122, 123, 124, ...

Operadores en la instrucción WHERE

Carácter	Descripción	Ejemplo	Resultado
[xx]	Sustituye un carácter por un rango de caracteres.	[DSR]edución	D edución, S edución, R edución, ...
[!xx]	Sustituye un carácter por un carácter que no pertenezca al rango.	[!AEGIJKOQUW XYZ]e	B e, C e, D e, F e, H e, L e, M e, N e, P e, R e, S e, T e, V e
-	Genera rangos de caracteres.	[A-D]x	A x, B x, C x, D x

Operadores en la instrucción WHERE

LIKE

Descripción:

Permite buscar un patrón específico de datos dentro de una columna.

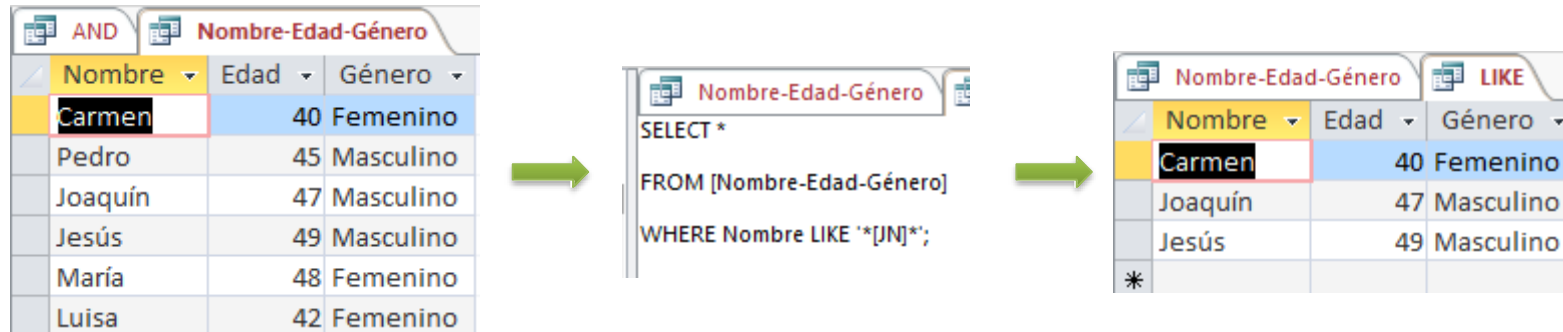
Sintaxis:

```
SELECT nombre_columna, ...
```

```
FROM nombre_tabla
```

```
WHERE nombre_columna LIKE 'patrón_de_busqueda';
```

Ejemplo:



Operadores en la instrucción WHERE

BETWEEN

Descripción:

Permite condicionar los valores de una columna a un rango. Estos rangos pueden ser numéricos, de texto o fechas. Para los rangos de fecha el formato establecido por SQL es #mm/dd/yyyy# o #dd/mm/yyyy#.

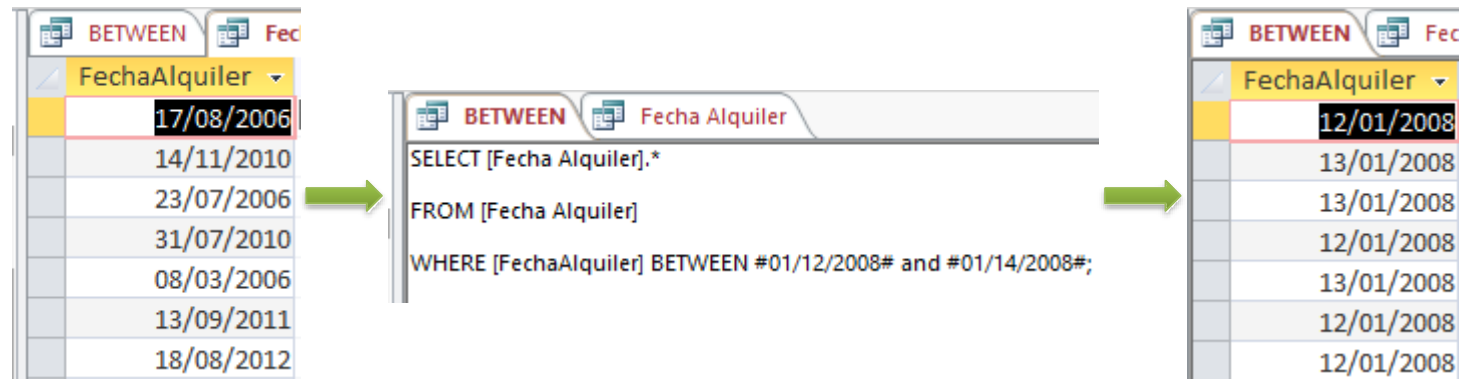
Sintaxis:

```
SELECT nombre_columna, ...
```

```
FROM nombre_tabla
```

```
WHERE nombre_columna BETWEEN cota_inferior AND cota_superior;
```

Ejemplo:



Operadores en la instrucción WHERE

IN

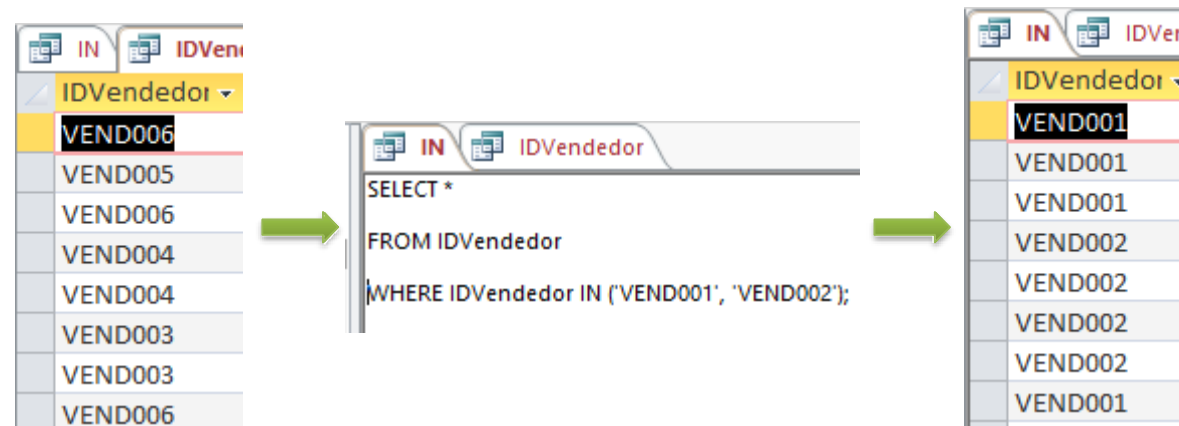
Descripción:

Permite condicionar los valores de una columna a una serie específica de valores.

Sintaxis:

```
SELECT nombre_columna, ...  
FROM nombre_tabla  
WHERE nombre_columna IN (valor1, valor2, ...);
```

Ejemplo:



Uso de funciones de no agrupamiento

Funciones

Las funciones en SQL nos permiten hacer modificaciones y transformaciones de los datos que se encuentran en las tablas pertenecientes a nuestra base de datos. Las funciones de no agrupamiento que se estudiarán y utilizarán durante el curso son:

- UCase()
- LCase()
- Mid()
- Len()
- Round()
- Concatenar
- First()
- Last()
- Now()
- Format()
- InStr()
- InStrRev()
- Left()
- Right()

Funciones de no agrupamiento

UCASE()

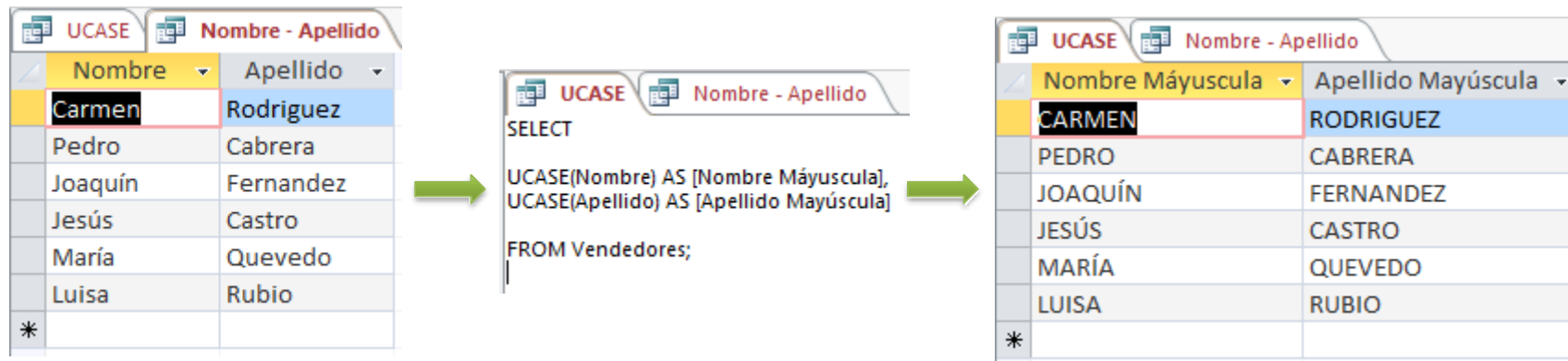
Descripción:

Permite convertir los valores de un rango en mayúscula.

Sintaxis:

```
SELECT UCASE (nombre_columna)  
FROM nombre_tabla;
```

Ejemplo:



Funciones de no agrupamiento

LCASE()

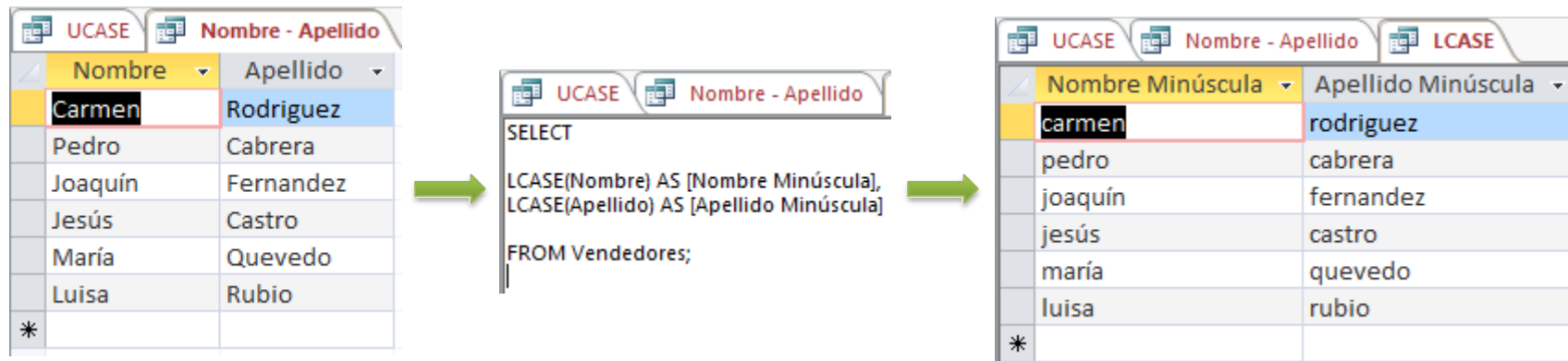
Descripción:

Permite convertir los valores de un rango en minúscula.

Sintaxis:

```
SELECT LCASE (nombre_columna)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

MID()

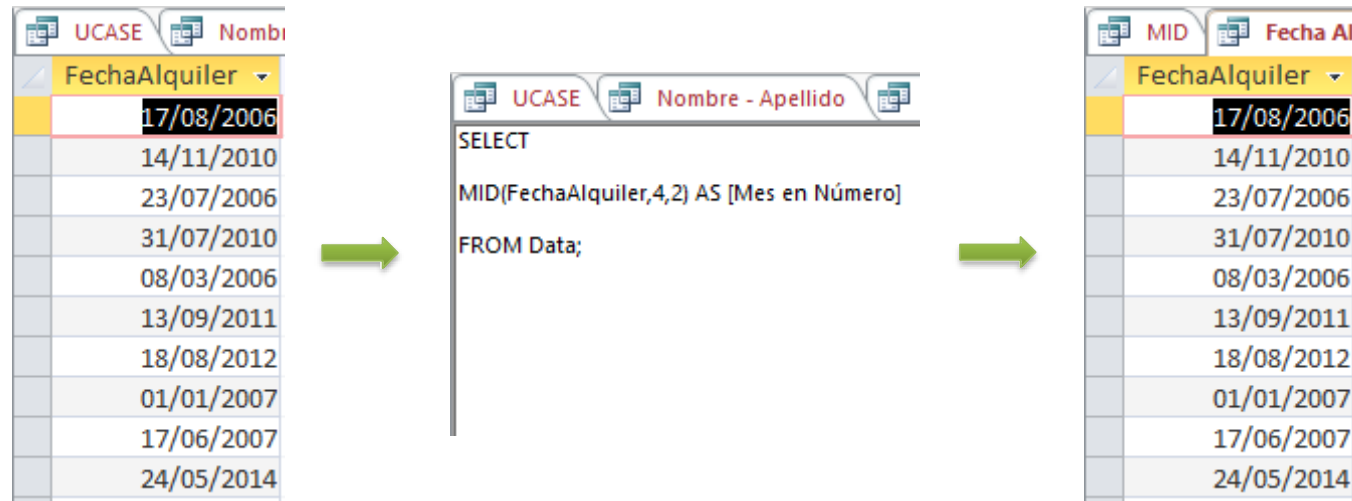
Descripción:

Permite extraer determinado número de caracteres de un campo de texto.

Sintaxis:

```
SELECT MID(nombre_columna, posición_inicial, largo)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

LEN()

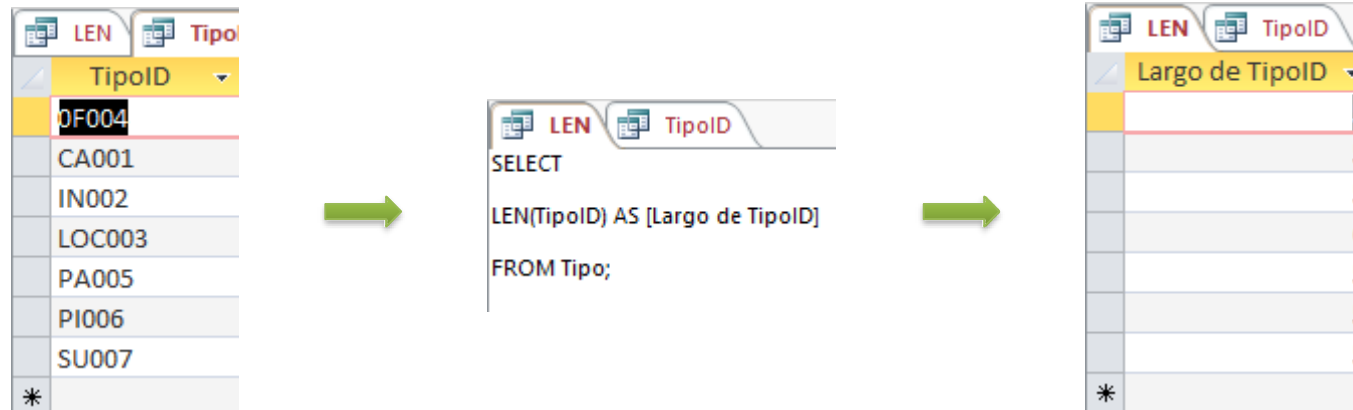
Descripción:

Permite determinar el número de caracteres de un campo de texto.

Sintaxis:

```
SELECT LEN(nombre_columna)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

ROUND()

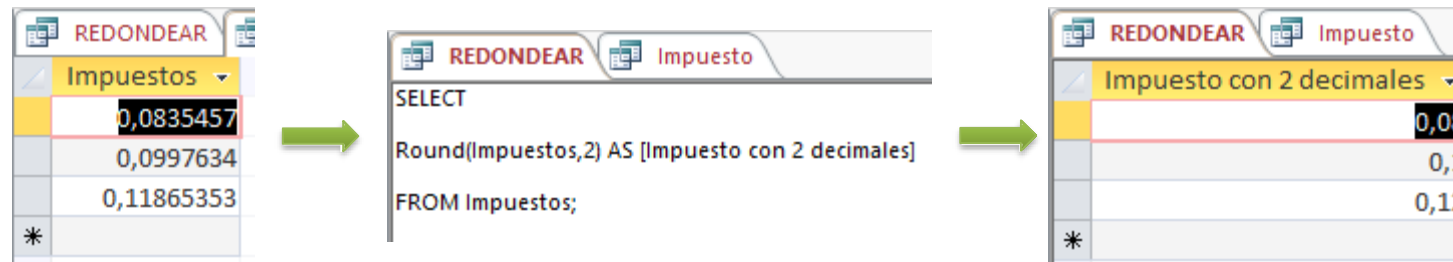
Descripción:

Permite redondear un campo numérico a un número determinado de decimales.

Sintaxis:

```
SELECT ROUND(nombre_columna, número_de_decimales)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

Concatenar

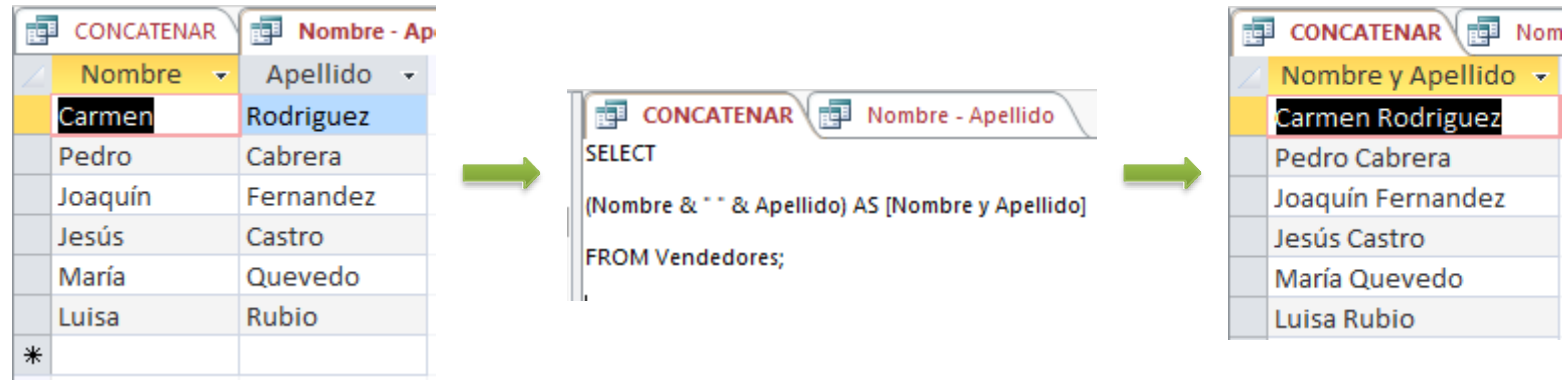
Descripción:

Permite unir dos o más campos de texto en uno solo.

Sintaxis:

```
SELECT nombre_columna & nombre_columna2 & ... & nombre_columna(n)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

FORMAT()

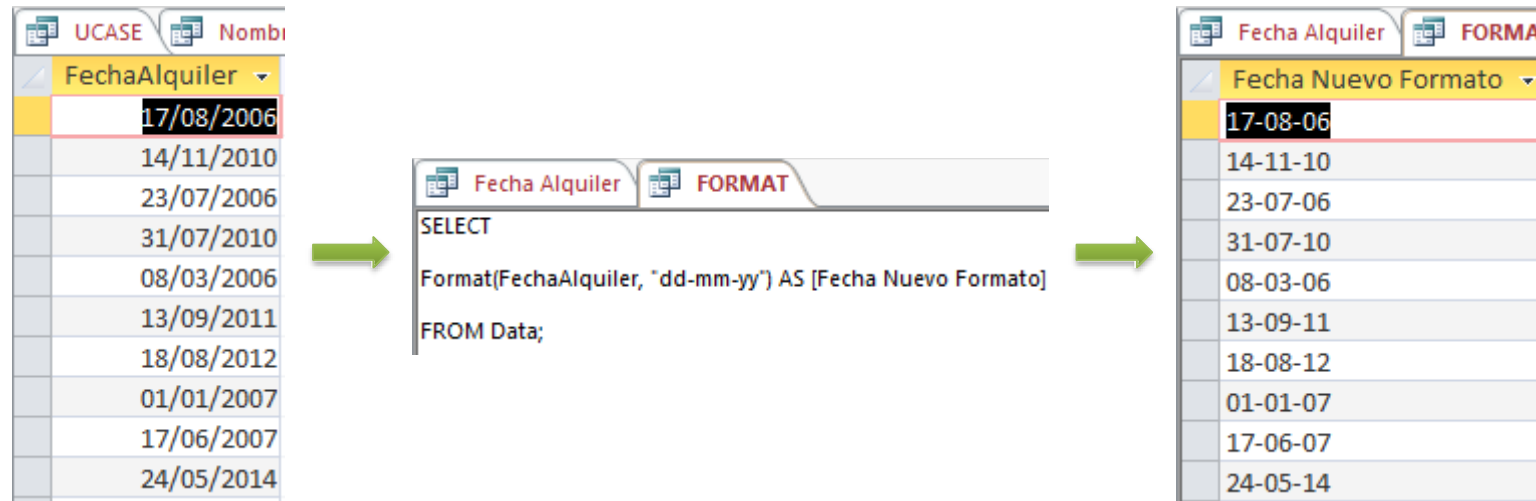
Descripción:

Devuelve un dato que contiene una expresión con un formato seleccionado.

Sintaxis:

```
SELECT FORMAT(nombre_columna, formato)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

FIRST()

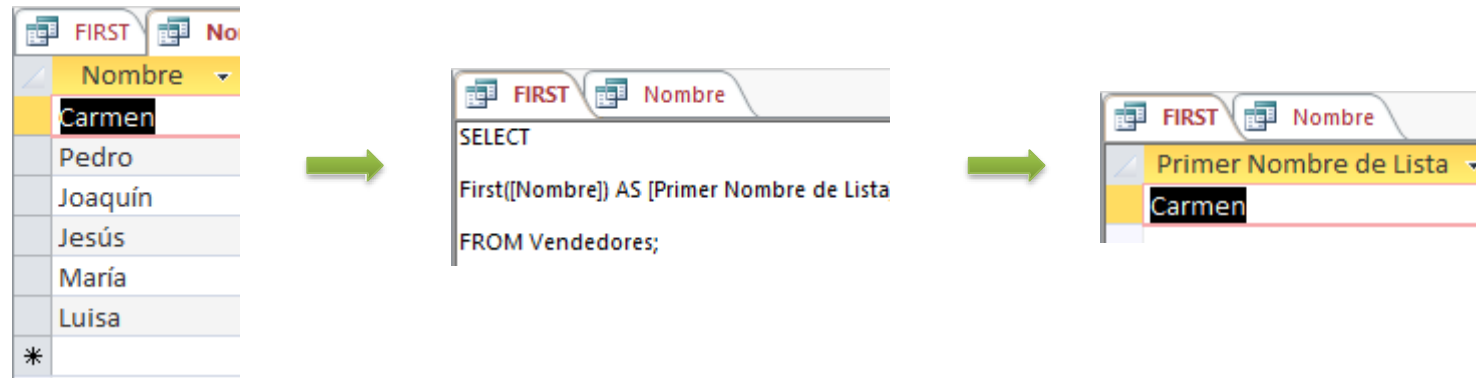
Descripción:

Permite obtener el primer valor de una columna seleccionada sin importar el orden.

Sintaxis:

```
SELECT FIRST(nombre_columna)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

LAST()

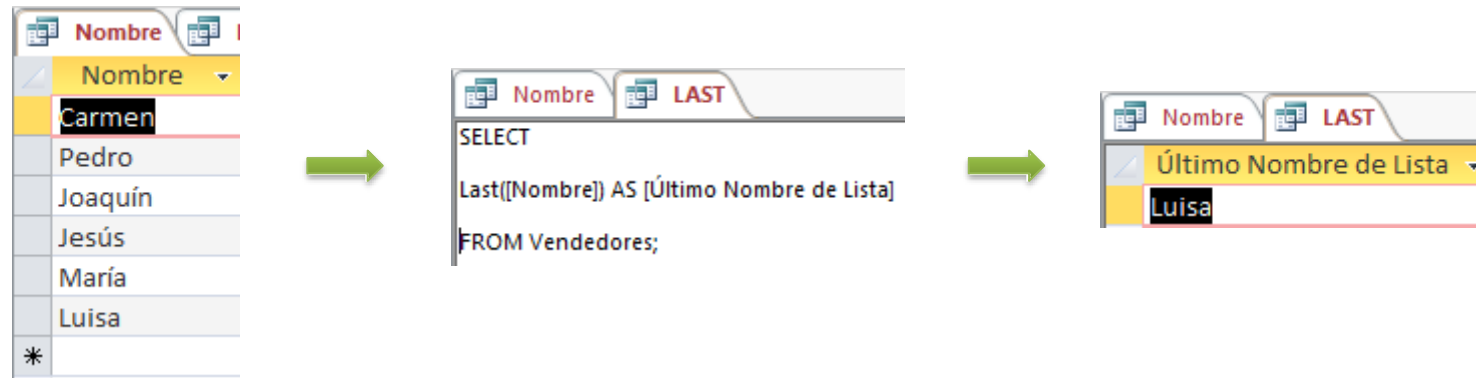
Descripción:

Permite obtener el último valor de una columna seleccionada sin importar el orden.

Sintaxis:

```
SELECT LAST(nombre_columna)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

NOW()

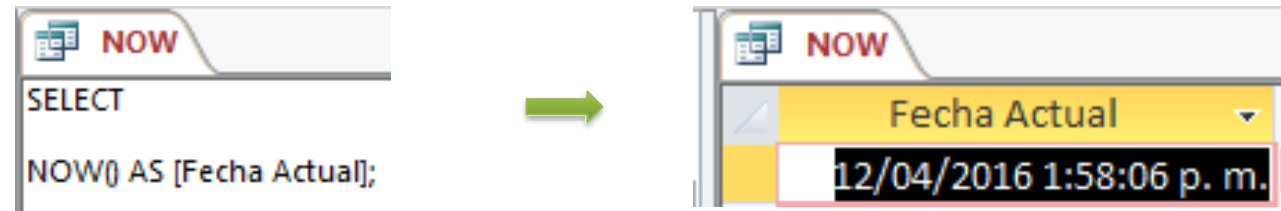
Descripción:

Permite obtener la fecha y hora actual del sistema.

Sintaxis:

```
SELECT NOW( )  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

INSTR()

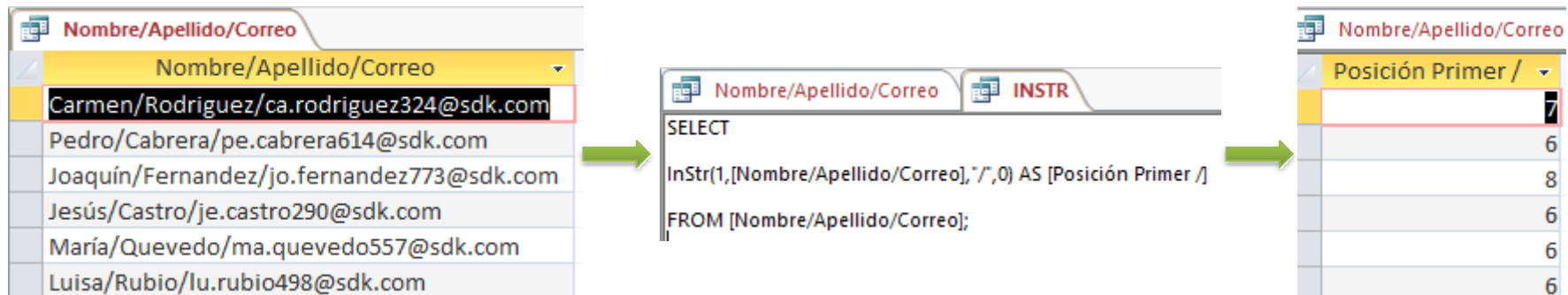
Descripción:

Permite encontrar la posición de una cadena de texto buscado de izquierda a derecha.

Sintaxis:

```
SELECT INSTR(posición_inicial, nombre_columna, texto_buscado,  
tipo_coincidencia)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

INSTRREV()

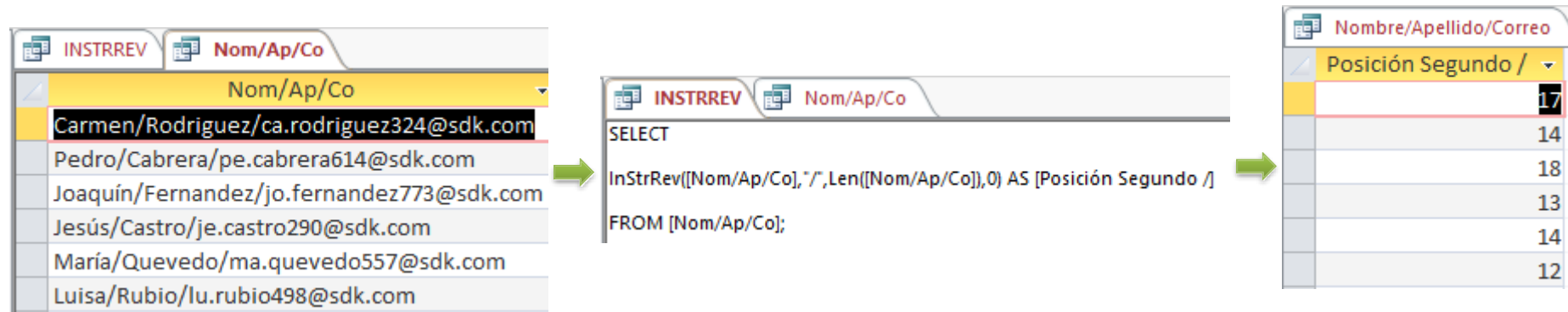
Descripción:

Permite encontrar la posición de una cadena de texto buscado de derecha a izquierda.

Sintaxis:

```
SELECT INSTRREV(nombre_columna, texto_buscado, posición_inicial,  
tipo_coincidencia)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

LEFT()

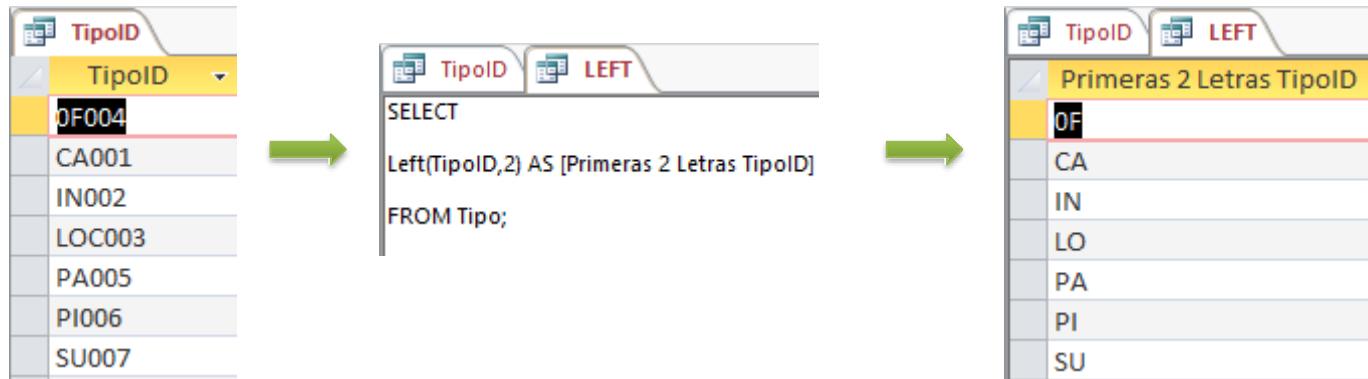
Descripción:

Permite extraer determinado número de caracteres del lado izquierdo de un campo de texto.

Sintaxis:

```
SELECT LEFT(nombre_columna, largo)  
FROM nombre_tabla;
```

Ejemplo:



Uso de funciones de no agrupamiento

RIGHT()

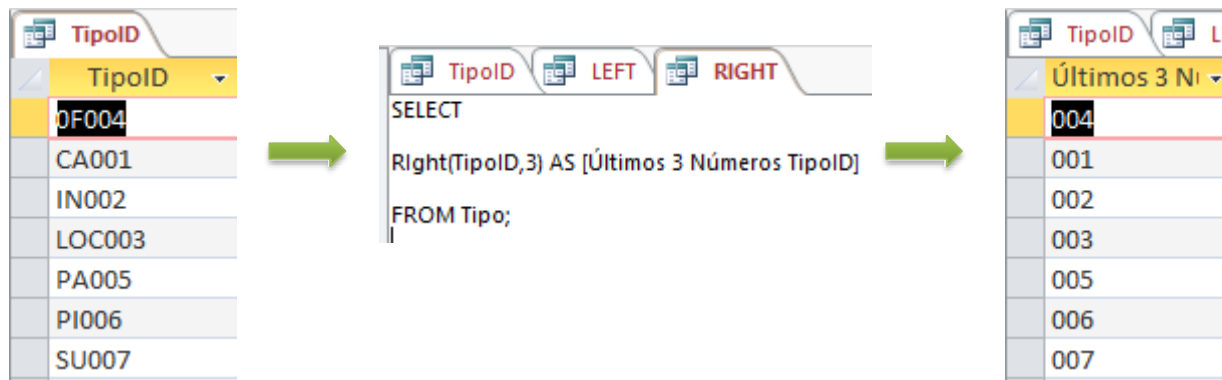
Descripción:

Permite extraer determinado número de caracteres del lado derecho de un campo de texto.

Sintaxis:

```
SELECT RIGHT(nombre_columna, largo)  
FROM nombre_tabla;
```

Ejemplo:



Ordenamiento de datos

ORDER BY

Descripción:

Permite ordenar los datos de una tabla con respecto a una o más columnas.

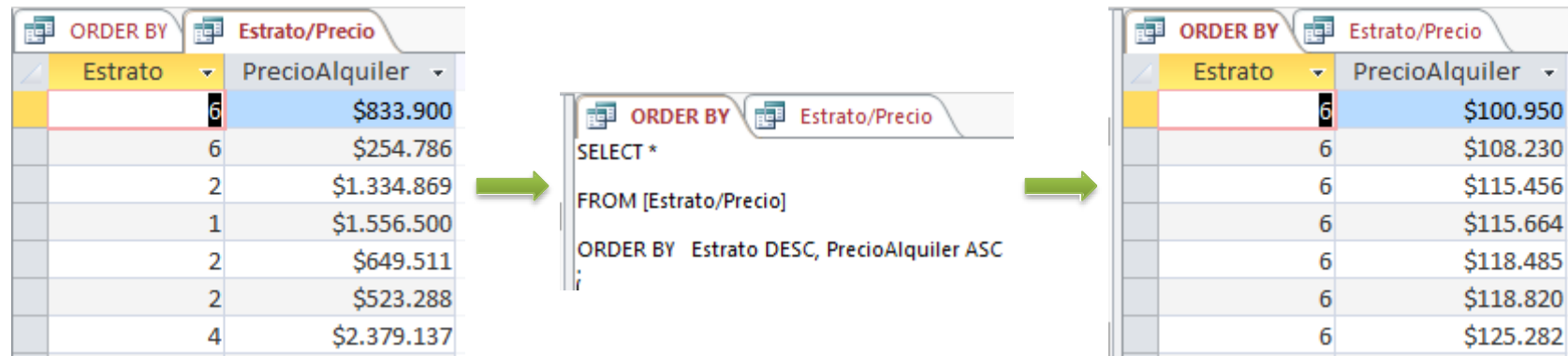
Sintaxis:

```
SELECT nombre_columna, ...
```

```
FROM nombre_tabla
```

```
ORDER BY nombre_columna [ASC/DESC], nombre_columna [ASC/DESC];
```

Ejemplo:



The diagram illustrates the process of sorting data using an SQL query. It shows an initial table, a SQL query, and the resulting sorted table.

Initial Table:

Estrato	PrecioAlquiler
6	\$833.900
6	\$254.786
2	\$1.334.869
1	\$1.556.500
2	\$649.511
2	\$523.288
4	\$2.379.137

SQL Query:

```
SELECT *  
FROM [Estrato/Precio]  
ORDER BY Estrato DESC, PrecioAlquiler ASC  
;
```

Resulting Table:

Estrato	PrecioAlquiler
6	\$100.950
6	\$108.230
6	\$115.456
6	\$115.664
6	\$118.485
6	\$118.820
6	\$125.282

Ordenamiento de datos

ASC

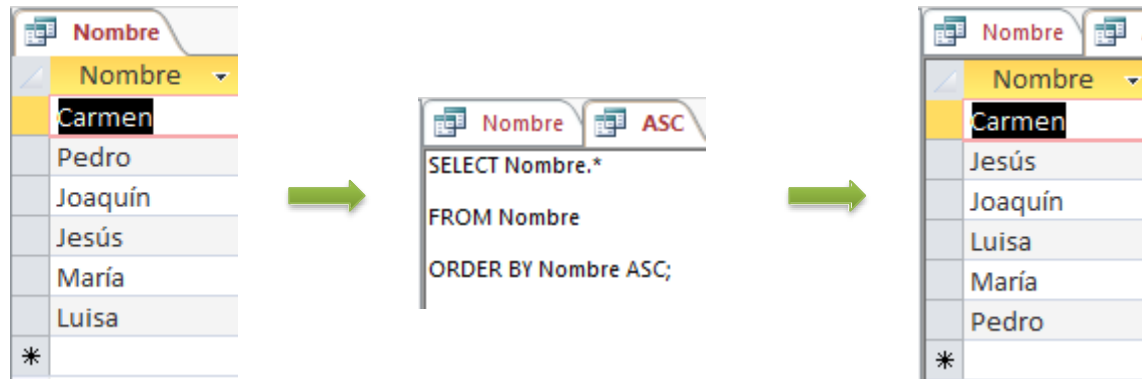
Descripción:

Permite ordenar los datos de una tabla con respecto a una o más columnas **ascendentemente**.

Sintaxis:

```
SELECT nombre_columna, ...  
FROM nombre_tabla  
ORDER BY nombre_columna ASC;
```

Ejemplo:



Ordenamiento de datos

DESC

Descripción:

Permite ordenar los datos de una tabla con respecto a una o más columnas **descendentemente**.

Sintaxis:

```
SELECT nombre_columna, ...  
FROM nombre_tabla  
ORDER BY nombre_columna DESC;
```

Ejemplo:



Ordenamiento de datos

TOP #

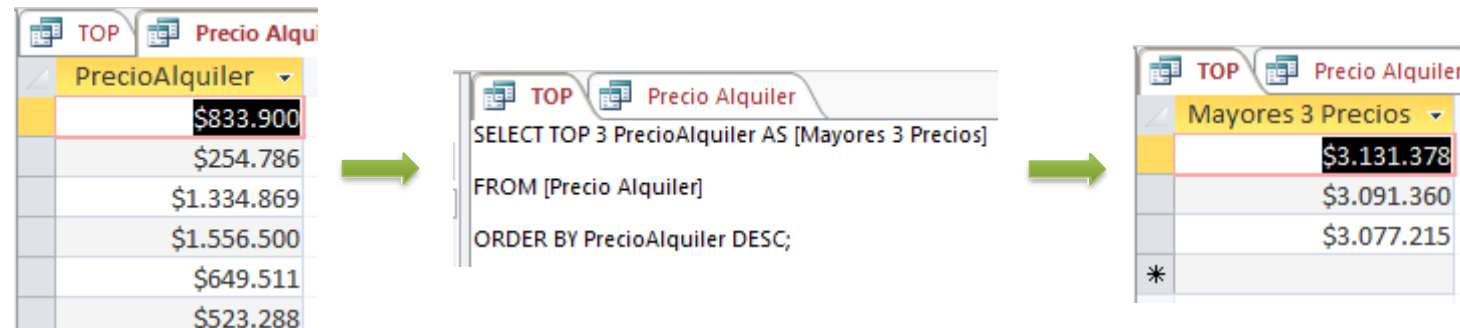
Descripción:

Permite especificar el número de datos que se muestran de una tabla. Los datos mostrados **salen de la parte de arriba de la tabla**, por lo cual debe estar en el orden deseado.

Sintaxis:

```
SELECT TOP # nombre_columna, ...  
FROM nombre_tabla  
ORDER BY nombre_columna [ASC/DESC];
```

Ejemplo:





Clase 3: Funciones de agrupamiento y taller en clase.

Agenda

- Objetivos
- Sentencia GROUP BY
- Funciones de agrupamiento
- Sentencia HAVING
- Taller

Objetivos

- Aprender a hacer consultas de agrupamiento y de resumen
- Utilizar efectivamente las sentencias GROUP BY y HAVING

Orden de las sentencias

Hasta el momento las sentencias vistas se ordenan según se muestra a continuación:

```
SELECT  
FROM  
WHERE  
ORDER BY
```

Orden de las sentencias

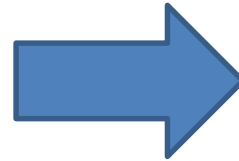
Las nuevas sentencias tienen también un orden específico:

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

GROUP BY

Imaginen la sentencia GROUP BY como un campo de fila o columna en una tabla dinámica, donde se agrupan campos iguales para calcular métricas importantes, tales como contar, máximo, mínimo, promedio, etc.

Nombre	Género	Ciudad	Edad
Laura	Femenino	Bogotá	23
Esteban	Masculino	Cali	29
Daniel	Masculino	Bogotá	38
Alejandra	Femenino	Cartagena	40
María	Femenino	Cartagena	32
Leonardo	Masculino	Bogotá	51
Ismael	Masculino	Bogotá	41
Lina	Femenino	Cali	55
Lorena	Femenino	Cartagena	43
Alberto	Masculino	Cali	24
Jimena	Femenino	Cartagena	21
Daniela	Femenino	Bogotá	36

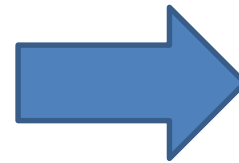


Ciudad	Edad_Promedio
Bogotá	37.8
Cali	36
Cartagena	34
Total general	36.0833

GROUP BY

Para escribir una sentencia de GROUP BY en SQL, debe tener en cuenta que campos desea mostrar, por cuales quiere agrupar y finalmente que campos quiere calcular. Por ejemplo la sentencia para nuestro ejemplo sería:

Nombre	Género	Ciudad	Edad
Laura	Femenino	Bogotá	23
Esteban	Masculino	Cali	29
Daniel	Masculino	Bogotá	38
Alejandra	Femenino	Cartagena	40
María	Femenino	Cartagena	32
Leonardo	Masculino	Bogotá	51
Ismael	Masculino	Bogotá	41
Lina	Femenino	Cali	55
Lorena	Femenino	Cartagena	43
Alberto	Masculino	Cali	24
Jimena	Femenino	Cartagena	21
Daniela	Femenino	Bogotá	36



Ciudad	Edad_Promedio
Bogotá	37.8
Cali	36
Cartagena	34
Total general	36.0833

```
SELECT Ciudad, AVG(Edad) AS Edad_Promedio FROM Tabla GROUP BY Ciudad;
```

GROUP BY

```
SELECT Ciudad, AVG(Edad) AS Edad_Promedio FROM Tabla GROUP BY Ciudad;
```

En la consulta se tienen dos tipos de campos, los que están dentro de una función de agrupamiento y los que no. Hay que tener en cuenta que si se utiliza una función de agrupamiento, **todo los campos no agrupado deben aparecer en el GROUP BY en el mismo orden que en la sentencia SELECT.**

GROUP BY

Pueden existir cualquier número de campos agrupados como no agrupados, con tal que se llamen de acuerdo a como fueron escritos.

Ciudad	Género	Edad_Promedio	Edad_Máxima	Cuenta_Nombre
Bogotá	Femenino	29.5	36	2
Bogotá	Masculino	43.33333333	51	3
Cali	Femenino	55	55	1
Cali	Masculino	26.5	29	2
Cartagena	Femenino	34	43	4

```
SELECT Ciudad, Género, AVG(Edad) AS Edad_Promedio, MAX(Edad) as
Edad_Máxima, COUNT(Nombre) as Cuenta_Nombre
FROM Tabla
GROUP BY Ciudad, Género;
```

Funciones de Agrupamiento

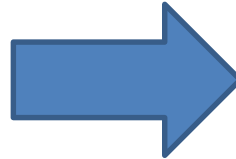
Existen varias funciones de agrupamiento:

- COUNT: Contar datos
- MAX: El máximo de los datos
- MIN: El mínimo de los datos
- AVG: El promedio de los datos
- SUM: La suma de los datos
- VAR: La varianza de los datos
- STDEV: La desviación estándar de los datos

HAVING

La sentencia HAVING es igual a una sentencia WHERE, pero es sólo usada para campos calculados.

Ciudad	Edad_Promedio
Bogotá	37.8
Cali	36
Cartagena	34



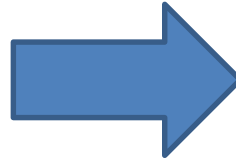
Ciudad	Edad_Promedio
Bogotá	37.8
Cali	36

```
SELECT Ciudad, AVG(Edad) AS Edad_Promedio FROM Tabla GROUP BY Ciudad  
HAVING AVG(Edad) > 34;
```

HAVING

Tener en cuenta que si una condición sobre un campo calculado se utiliza en un WHERE, ésta no funcionará.

Ciudad	Edad_Promedio
Bogotá	37.8
Cali	36
Cartagena	34



Ciudad	Edad_Promedio
Bogotá	37.8
Cali	36

```
SELECT Ciudad, AVG(Edad) AS Edad_Promedio FROM Tabla GROUP BY Ciudad  
HAVING AVG(Edad) > 34;
```



Taller

- Descargar de la plataforma



Clase 4: Manejo de bases de datos en Java

Base de datos

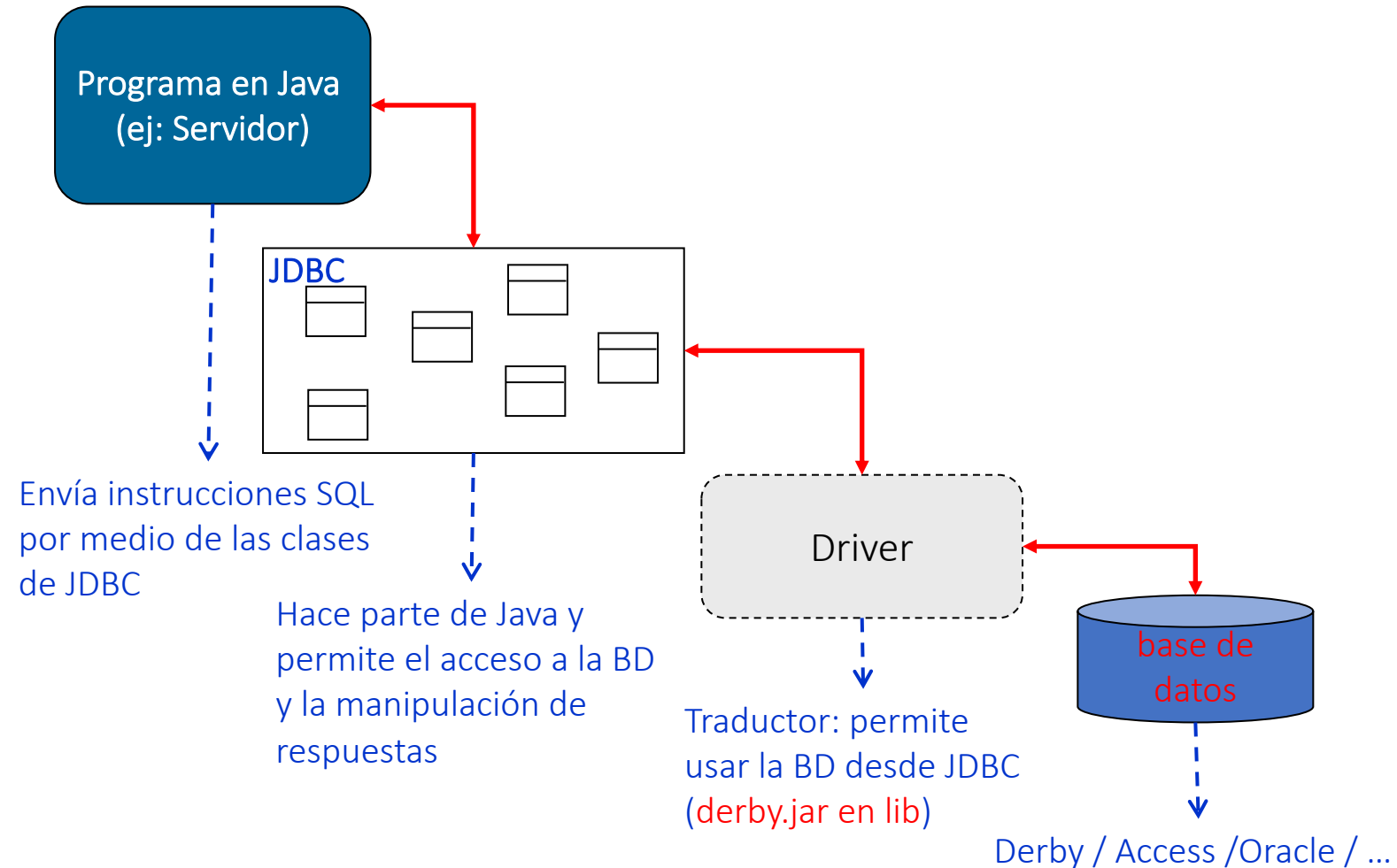
Para manejar una BD desde Java

- Se necesita un manejador de bases de datos (producto fuera del lenguaje):
 - ORACLE
 - ACCESS
 - SQL Server
 - DB2
 - MySQL
 - DERBY
- Libre
 - Implementado en java
 - Desarrollado por APACHE
 - <http://db.apache.org/derby/>

JDBC

- JDBC: *Java DataBase Connectivity*
- Framework: conjunto de clases Java que permiten usar SQL para manipular bases de datos:
 - Conectarse a una BD
 - Enviar a la BD instrucciones escritas en SQL
- Genera independencia entre el programa y el manejador de bases de datos que se utilice (el producto específico)

JDBC



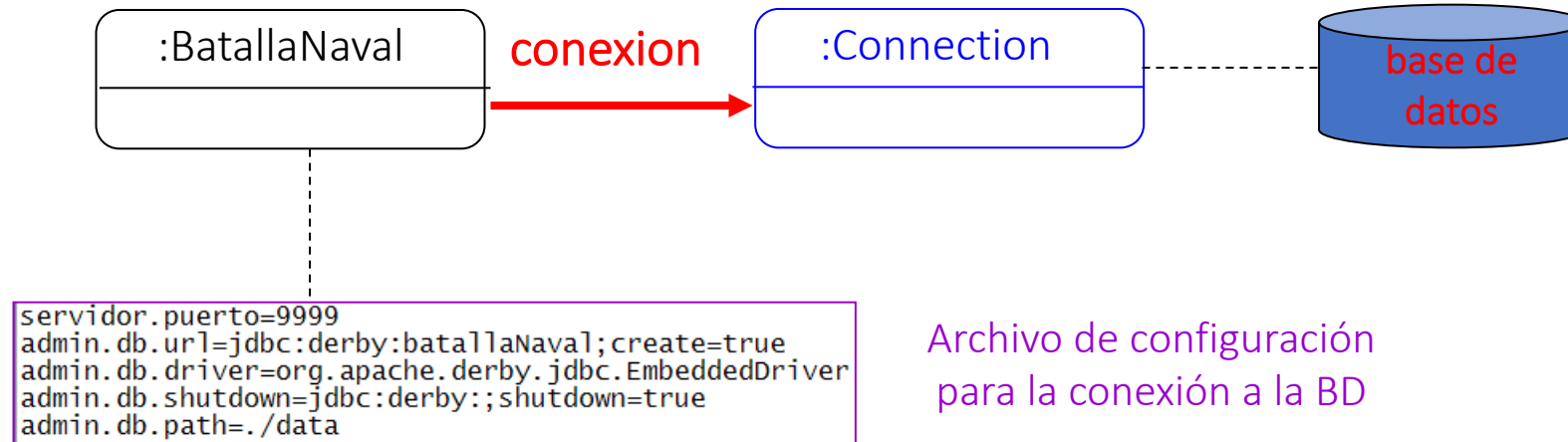
JDBC: Uso básico

Dos etapas:

1. Crear una conexión a la base de datos
2. Utilizar la conexión para enviar por ahí las instrucciones en SQL

JDBC: Conexión a la BD

Objeto de la clase Connection
(JDBC) por medio del cual se
puede interactuar con la BD



JDBC: Cómo crear una instrucción SQL

Una instrucción SQL se crea por medio del método `createStatement` de la clase `Connection`:

```
try
{
    Statement s = conexion.createStatement( );
    ...
    s.close( );
}
catch( SQLException e )
{
    ...
}
```

Retorna un objeto de la clase `Statement`,
por medio del cual se pueden enviar
instrucciones SQL a la BD
Para dejar de utilizar el objeto

JDBC: Cómo crear una instrucción SQL

Una instrucción SQL se crea por medio del método `createStatement` de la clase `Connection`:

```
Statement s = conexion.createStatement( );  
String sql = "SELECT ...";  
  
ResultSet resultado = s.executeQuery( sql );
```

```
Statement s = conexion.createStatement( );  
String sql = "UPDATE resultados ...";  
  
int modificados = s.executeUpdate( sql );
```

JDBC: Cómo crear una instrucción SQL

Una instrucción SQL se crea por medio del método `createStatement` de la clase `Connection`:

```
Statement s = conexion.createStatement( );  
String sql = "CREATE ...";  
  
s.execute( sql );
```

JDBC: Cómo crear una instrucción SQL

Ejemplo executeUpdate

```
Statement st = conexión.createStatement( );
```

```
String sql = "UPDATE resultados  
            SET perdidos=perdidos+1  
            WHERE nombre='Barbanegra'";
```

```
st.executeUpdate( sql );
```

JDBC: Cómo crear una instrucción SQL

Ejemplo execute

```
Statement st = conexión.createStatement( );
```

```
String sql = "CREATE TABLE resultados  
    ( nombre varchar(32),  
      ganados int,  
      perdidos int,  
      PRIMARY KEY (nombre))";
```

```
st.execute( sql );
```

JDBC: Cómo crear una instrucción SQL

Ejemplo executeQuery

```
Statement st = conexión.createStatement( );
```

```
String sql = "SELECT * FROM resultados WHERE  
            nombre = 'Barbanegra'";
```

```
ResultSet resultado = st.executeQuery( sql );
```

JDBC: Cómo manipular la respuesta de una consulta

Se manipula la respuesta de una consulta por medio de la clase `ResultSet`

- Respuesta = secuencia de elementos
- Cada elemento tiene uno o varios campos
- Con el método `next()` se avanza sobre la secuencia.
- Con los métodos `getString()` y `getInt()` se extrae la información de los campos del elemento actual de la secuencia.

JDBC: Cómo manipular la respuesta de una consulta

```
String sql = "SELECT * FROM resultados WHERE nombre = 'Barbanegra'";
```

```
Statement st = conexion.createStatement( );  
ResultSet resultado = st.executeQuery( sql );  
if (resultado.next( ))  
{  
    String nombre = resultado.getString( 1 );  
    int ganados = resultado.getInt( 2 );  
    int perdidos = resultado.getInt( 3 );  
}  
resultado.close( );  
st.close( );
```

Retorna el nombre, el número de partidos ganados y el número de partidos perdidos de UN SOLO registro (cuyo nombre es Barbanegra). Se usa if (resultado.next()) para acceder al elemento retornado por la sentencia sql

JDBC: Cómo manipular la respuesta de una consulta

```
String sql = "SELECT * FROM resultados";

Statement st = conexion.createStatement( );
ResultSet resultado = st.executeQuery( sql );
while ( resultado.next( ) )
{
    String nombre = resultado.getString( 1 );
    int ganados = resultado.getInt( 2 );
    int perdidos = resultado.getInt( 3 );
}
resultado.close( );
st.close( );
```

Retorna el nombre, el número de partidos ganados y el número de partidos perdidos de TODOS los registros (no hay WHERE en la consulta). Se usa while (resultado.next()) para acceder a los elementos retornados por la sentencia sql

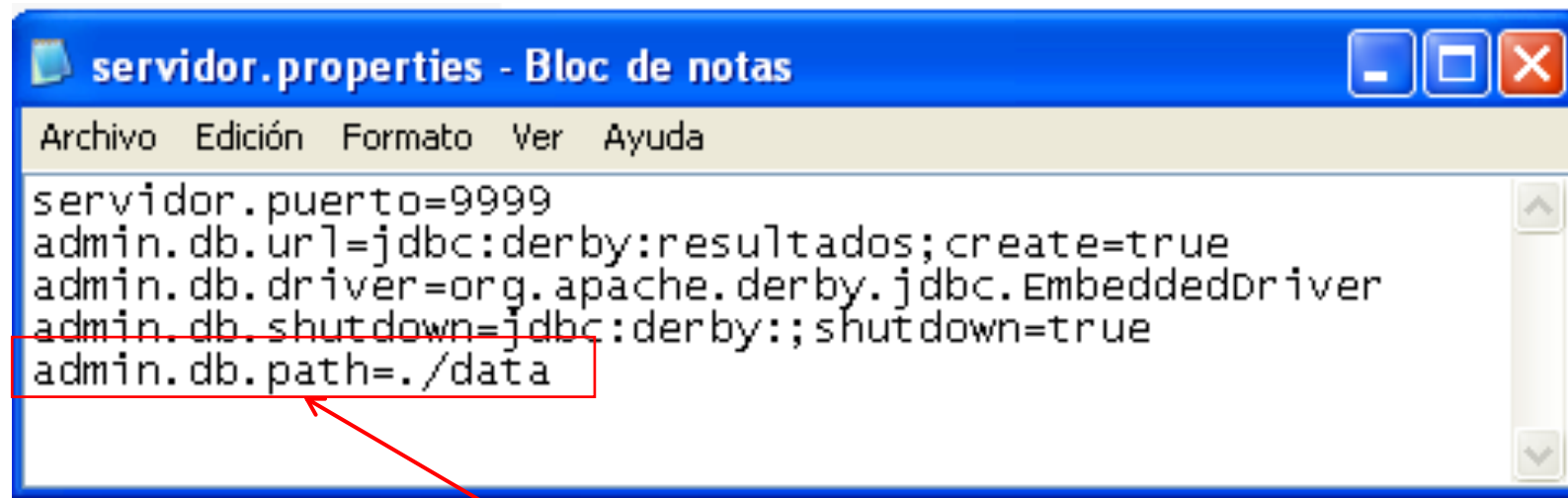
JDBC: Cómo establecer la conexión con la base de datos

- ❶ Especificar la localización física de la base de datos
- ❷ Crear un controlador (driver) para la base de datos
- ❸ Crear la conexión

JDBC: Cómo establecer la conexión con la base de datos

① Especificar la localización física de la BD

Archivo de propiedades: `servidor.properties`



```
servidor.puerto=9999
admin.db.url=jdbc:derby:resultados;create=true
admin.db.driver=org.apache.derby.jdbc.EmbeddedDriver
admin.db.shutdown=jdbc:derby:;;shutdown=true
admin.db.path=./data
```

Directorio donde se encuentra la BD

JDBC: Cómo establecer la conexión con la base de datos

① Especificar la localización física de la BD

- 👉 Establecer la ruta donde va a estar la base de datos.
- 👉 Derby utiliza la variable del sistema `derby.system.home` para saber dónde están los datos

Atributo de la clase Properties donde se leyó el
archivo de configuración

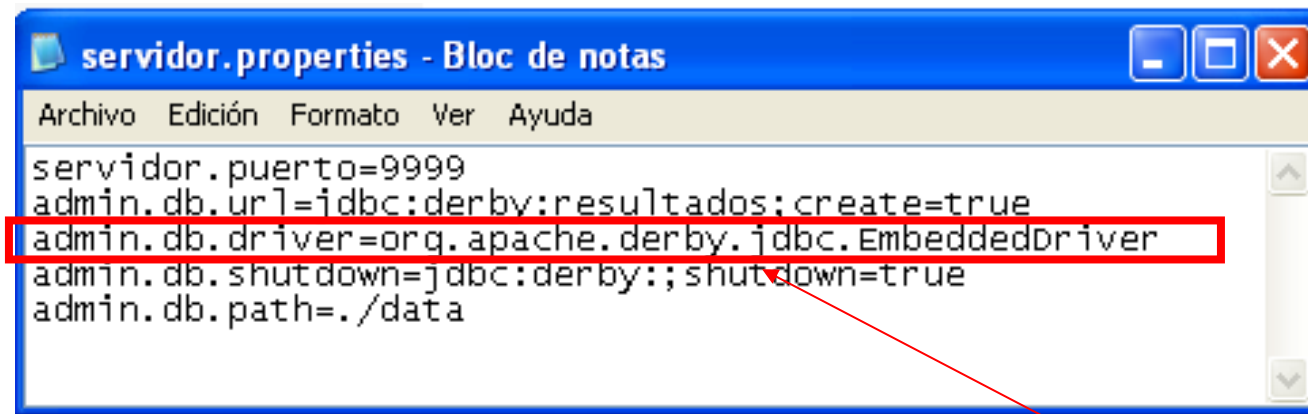
```
File data = new File( config.getProperty( "admin.db.path" );
```

```
System.setProperty( "derby.system.home", data.getAbsolutePath( ) );
```

Instrucción de java que permite asignar un valor
a una variable del sistema

JDBC: Cómo establecer la conexión con la base de datos

② Crear un controlador (driver) para la BD



```
servidor.puerto=9999
admin.db.url=jdbc:derby:resultados:create=true
admin.db.driver=org.apache.derby.jdbc.EmbeddedDriver
admin.db.shutdown=jdbc:derby:;;shutdown=true
admin.db.path=./data
```

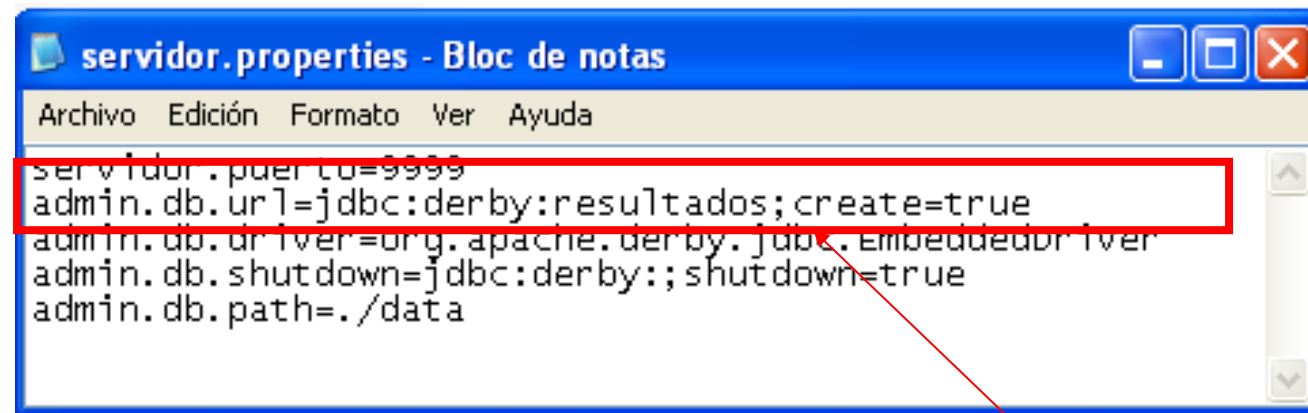
```
String driver = config.getProperty( "admin.db.driver");
```

```
Class.forName( driver ).newInstance( );
```

Instrucción para crear el controlador

JDBC: Cómo establecer la conexión con la base de datos

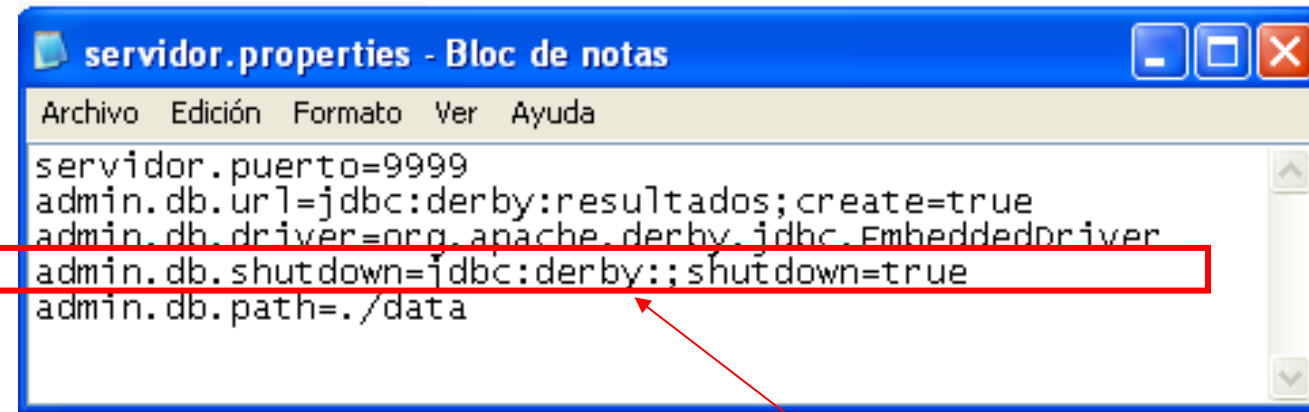
3 Crear la conexión



```
String url = config.getProperty( "admin.db.url" );  
conexion = DriverManager.getConnection( url );
```

Instrucción para crear la conexión a la BD. Que se va a almacenar en el directorio data/resultados

JDBC: Cómo desconectarse de la BD?



```
// Cierra la conexión a la base de datos y detiene luego la base de datos
public void desconectarBD( ) throws SQLException
{
    conexion.close( );
    String down = configuracion.getProperty( "admin.db.shutdown" );
    DriverManager.getConnection( down );
}
```




Clase 5: Joins

- Identificar los diferentes tipos de JOIN y su utilidad.
- Mostar como se puede extraer la información de más de dos tablas, mediante el uso de múltiples JOINS.
- Enseñar el uso de la instrucción UNION All y como se puede utilizar para realizar un FULL OUTER JOIN.

Introducción

- En SQL la instrucción JOIN se utiliza para extraer la información de varias tablas.
- Por ejemplo, si un hospital guarda la información de los pacientes, doctores y citas de la siguiente manera:

Tabla Pacientes

id	nombre	apellido
1	Catalina	Ferreira
2	Sergio	López
3	María Dolores	González
4	Francisco	Álvarez
5	Pascual	López
6	Luis	Sastre
7	Pedro	Hernández

Tabla Citas

idPaciente	idDoctor	Lugar	dia
1	1	101	Lunes
2	1	211	Martes
3	2	300	Martes
4	3	300	Jueves
5	3	101	Lunes
5	3	211	Jueves
5	3	300	Miércoles

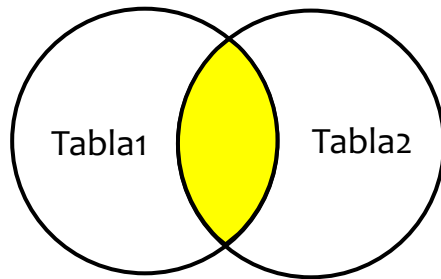
Tabla Doctores

id	Nombre	Apellido	Especialidad
1	José	Roldan	Pediatra
2	Laura	Riquelme	Nefrólogo
3	Josefina	Álvarez	Cirujano
4	Manuel	Ortega	General

Se puede extraer información de cada una de las tablas por medio de la relación que existe entre las tablas.

Inner Join

Notación:



```
SELECT Campo1, Campo2, ...  
FROM Tabla1  
INNER JOIN Tabla2  
ON Tabla1.Id = Tabla2.Id;
```

Ejemplo:

- El nombre y apellido de los pacientes que tienen asignada una cita para el Martes.

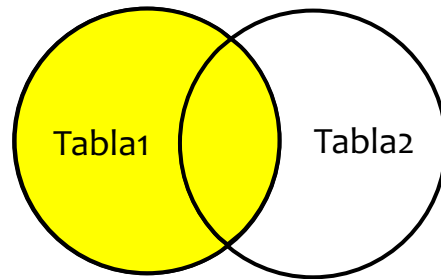
```
SELECT Pacientes.nombre, Pacientes.apellido  
FROM Pacientes  
INNER JOIN Citas  
ON Pacientes.id = Citas.idPaciente  
WHERE Citas.dia = "Martes";
```



nombre	apellido
Sergio	López
María Dolores	González

Left Join

Notación:



```
SELECT Campo1, Campo2, ...  
FROM Tabla1  
LEFT JOIN Tabla2  
ON Tabla1.Id = Tabla2.Id;
```

Ejemplo:

- El apellido de todos los doctores y la cantidad de citas que tienen asignados.

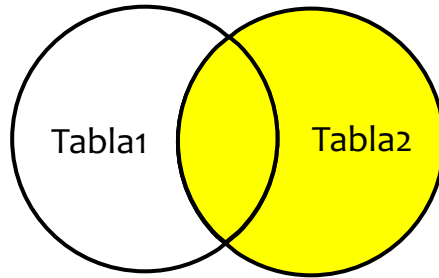
```
SELECT Doctores.apellido,  
COUNT(Citas.lugar) AS cantidad  
FROM Doctores  
LEFT JOIN Citas ON Doctores.id = Citas.idDoctor  
GROUP BY Doctores.apellido;
```



Nombre	Número
Alvaréz	4
Riquelme	1
Roldan	2
Ortega	0

Right Join

Notación:

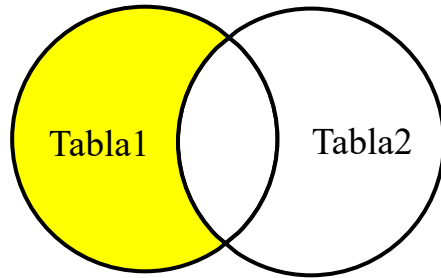


```
SELECT Campo1, Campo2, ...  
FROM Tabla1  
RIGHT JOIN Tabla2  
ON Tabla1.Id = Tabla2.Id;
```

- Observe que el **RIGHT JOIN** y el **LEFT JOIN** son equivalentes, es decir se puede obtener el mismo resultado cambiando el orden de las tablas.

Left Join sin Intercepción

Notación:



```
SELECT Campo1, Campo2, ...  
FROM Tabla1  
LEFT JOIN Tabla2  
ON Tabla1.Id = Tabla2.Id  
WHERE Tabla2.id IS NULL;
```

Ejemplo:

- El id y el nombre de los pacientes que tienen asignados un doctor de la lista de doctores.

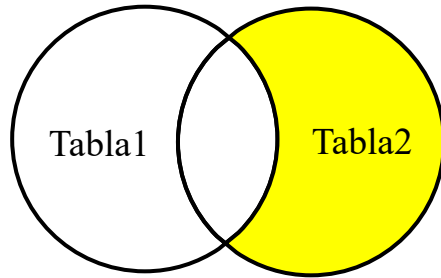
```
SELECT Pacientes.nombre, Pacientes.apellido  
FROM Pacientes  
LEFT JOIN Citas  
ON Pacientes.id = Citas.idPaciente  
WHERE Citas.idPaciente IS NULL;
```



nombre	apellido
Luis	Sastre
Pedro	Hernández

Right Join sin Intercepto

Notación:



```
SELECT Campo1, Campo2, ...  
FROM Tabla1  
Right JOIN Tabla2  
ON Tabla1.Id = Tabla2.Id  
WHERE Tabla1.id IS NULL;
```

- También en este caso el **RIGHT JOIN** y el **LEFT JOIN** son equivalentes, es decir se puede obtener el mismo resultado cambiando el orden de las tablas.

Múltiples Joins

Notación:

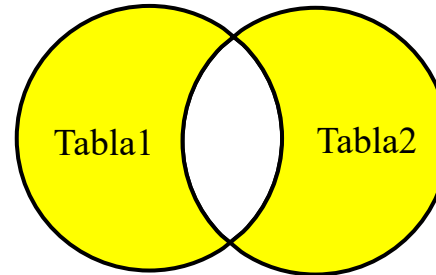
- También se puede anidar más de un JOIN para utilizar la información de varias de tablas, por ejemplo:

```
SELECT Pacientes.nombre, Pacientes.apellido, Doctores.especialidad, Citas.lugar  
FROM Pacientes  
INNER JOIN (Citas INNER JOIN Doctores ON Citas.idDoctor = Doctores.id)  
ON Citas.idPaciente = Pacientes.id;
```

- En el caso anterior se extrajeron los nombres y apellidos de la tabla Pacientes, la especialidad de la tabla Doctores y el lugar de la tabla Citas.

nombre	apellido	especialidad	lugar
Catalina	Ferreira	Pediatra	101
Sergio	López	Pediatra	211
María Dolores	González	Nefrólogo	300
Francisco	Álvarez	Cirujano	300
Pascual	López	Cirujano	101
Pascual	López	Cirujano	211
Pascual	López	Cirujano	300

Full Outer Join sin Intercepto



Ejemplo:

- La lista de los nombres y apellidos de los pacientes y doctores que no tienen asignada una cita, junto con una columna que indique si es doctor o paciente.

```
SELECT Pacientes.nombre, Pacientes.apellido, "Paciente" AS tipo
FROM Pacientes
LEFT JOIN Citas ON Citas.idPaciente = Pacientes.id
WHERE Citas.idPaciente IS NULL;
UNION ALL
SELECT Doctores.nombre, Doctores.apellido, "Doctor"
FROM Doctores
LEFT JOIN Citas ON Citas.idDoctor = Doctores.id
WHERE Citas.idDoctor IS NULL;
```