Capítulo 1

Introducción a bases de datos relacionales y SQL

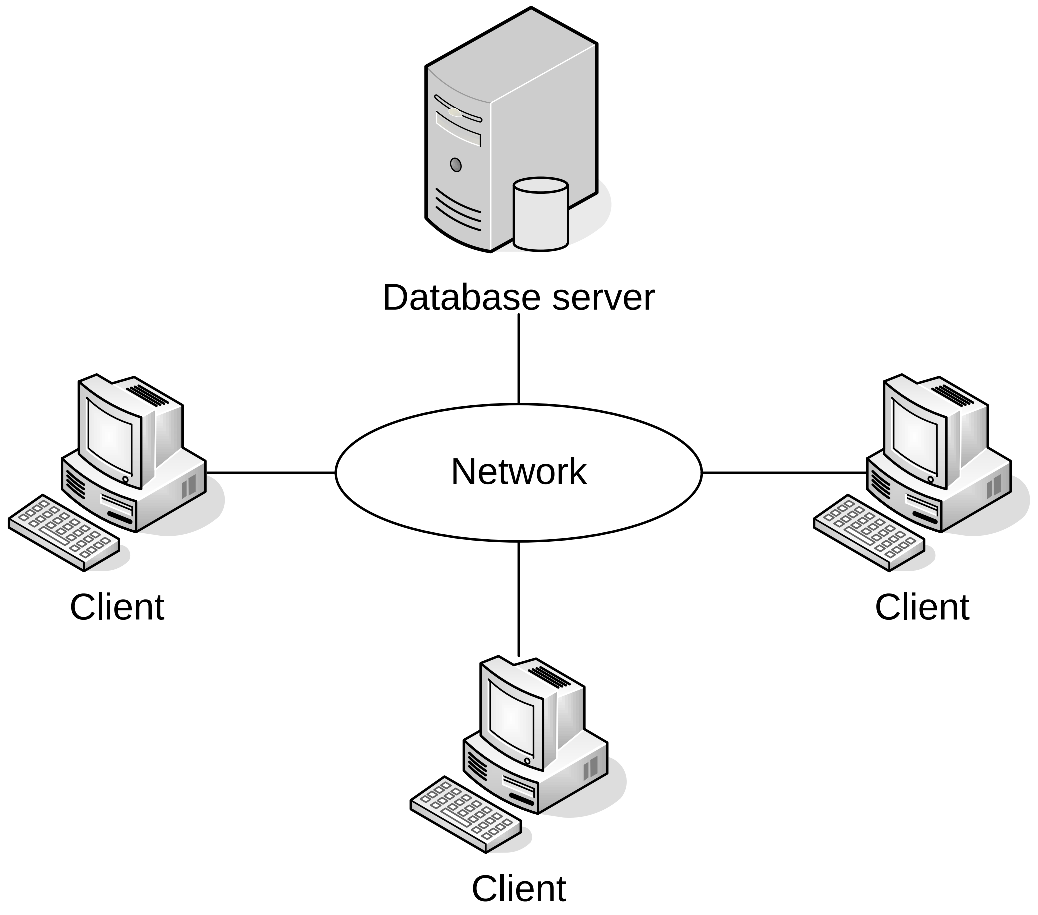
Objetivos
Conocimiento

- Identificar los tres componentes principales de hardware de un sistema cliente/servidor.
- Describa la forma en que un cliente accede a la base de datos en un servidor utilizando estos términos: software de aplicación, API de acceso a datos, sistema de administración de bases de datos, consulta SQL y resultados de consultas.
- Describa la forma en que se organiza una base de datos utilizando estos términos: tablas, columnas, filas y celdas.
- Describa cómo se relacionan las tablas de una base de datos relacional utilizando estos términos: clave principal y clave externa.
- Identifique los tres tipos de relaciones que pueden existir entre dos tablas.
- Describa la forma en que se definen las columnas de una tabla utilizando estos términos: tipo de datos, valor nulo, valor predeterminado y columna de identidad.

Objetivos (cont.)

- Describir la relación entre SQL estándar y Transact-SQL de Microsoft SQL Server.
- Describa la diferencia entre las instrucciones DML y las instrucciones DDL.
- Describa la diferencia entre una consulta de acción y una consulta SELECT.
- Enumere tres técnicas de codificación que pueden hacer que su código SQL sea más fácil de leer y mantener.
- Explique en qué se diferencian las vistas y los procedimientos almacenados de las sentencias SQL que se emiten desde un programa de aplicación.
- Describir el uso de objetos de comando, conexión y lector de datos cuando las aplicaciones .NET tienen acceso a una base de datos de SQL Server.

Un sistema cliente/servidor sencillo



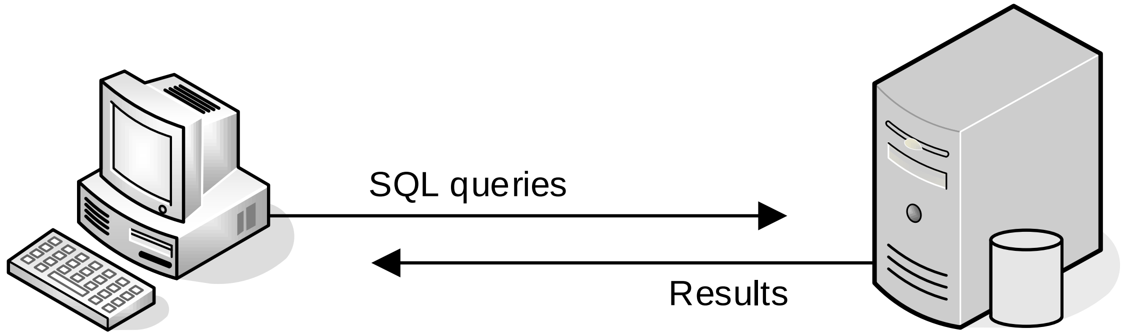
Los tres componentes de hardware de un sistema cliente/servidor

- Clientes
- Servidor
- Palabra de red

Términos que debes conocer

- Red de área local (LAN)
- Red de área extensa (WAN)
- Sistema empresarial

Software de cliente, software de servidor y la interfaz SQL



Client
Application software
Data access API

Database server
Database management system
Database

Software de servidor

- Sistema de gestión de bases de datos (DBMS)
- El DBMS realiza el procesamiento back-end

Software de cliente

- Aplicación
- API de acceso a datos (interfaz de programación de aplicaciones)
- El software cliente realiza el procesamiento front-end

La interfaz SQL

- El software de la aplicación se comunica con el DBMS mediante el envío de consultas SQL a través de la API de acceso a datos.
- Cuando el DBMS recibe una consulta, proporciona un servicio como devolver los datos solicitados (los resultados de la consulta) al cliente.
- SQL son las siglas de Structured Query Language, que es el lenguaje estándar para trabajar con una base de datos relacional.

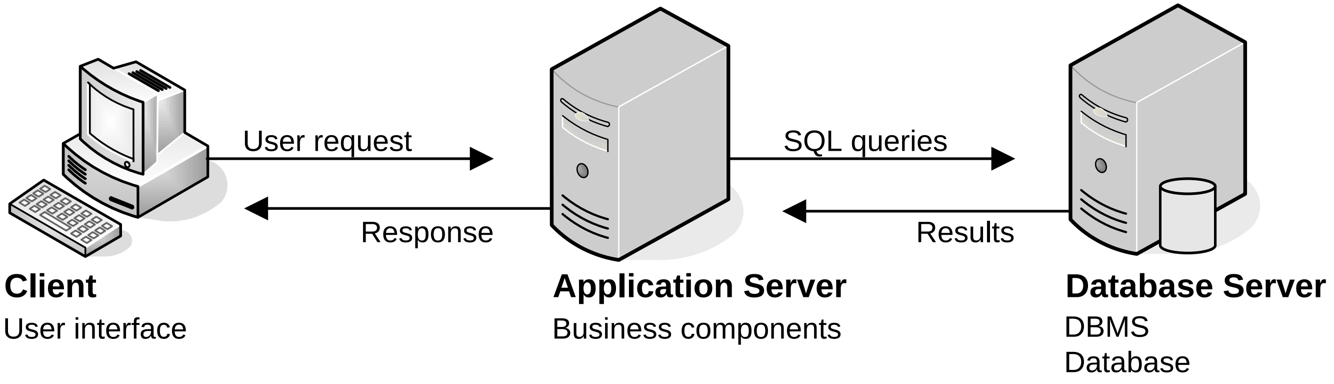
Sistema cliente/servidor

- El procesamiento se divide entre el cliente y el servidor

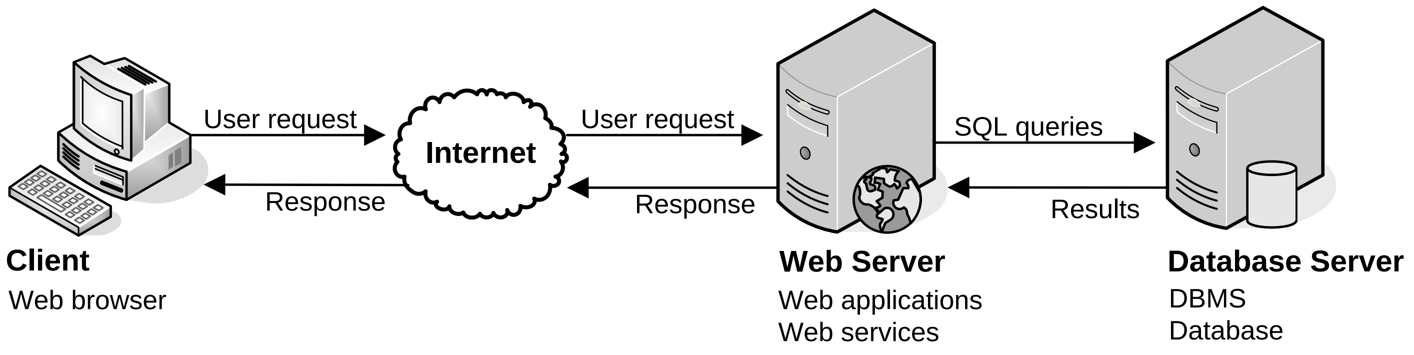
Sistema de manejo de archivos

- Todo el procesamiento se realiza en los clientes

Un sistema basado en Windows que utiliza un servidor de aplicaciones



Un sistema sencillo basado en la web



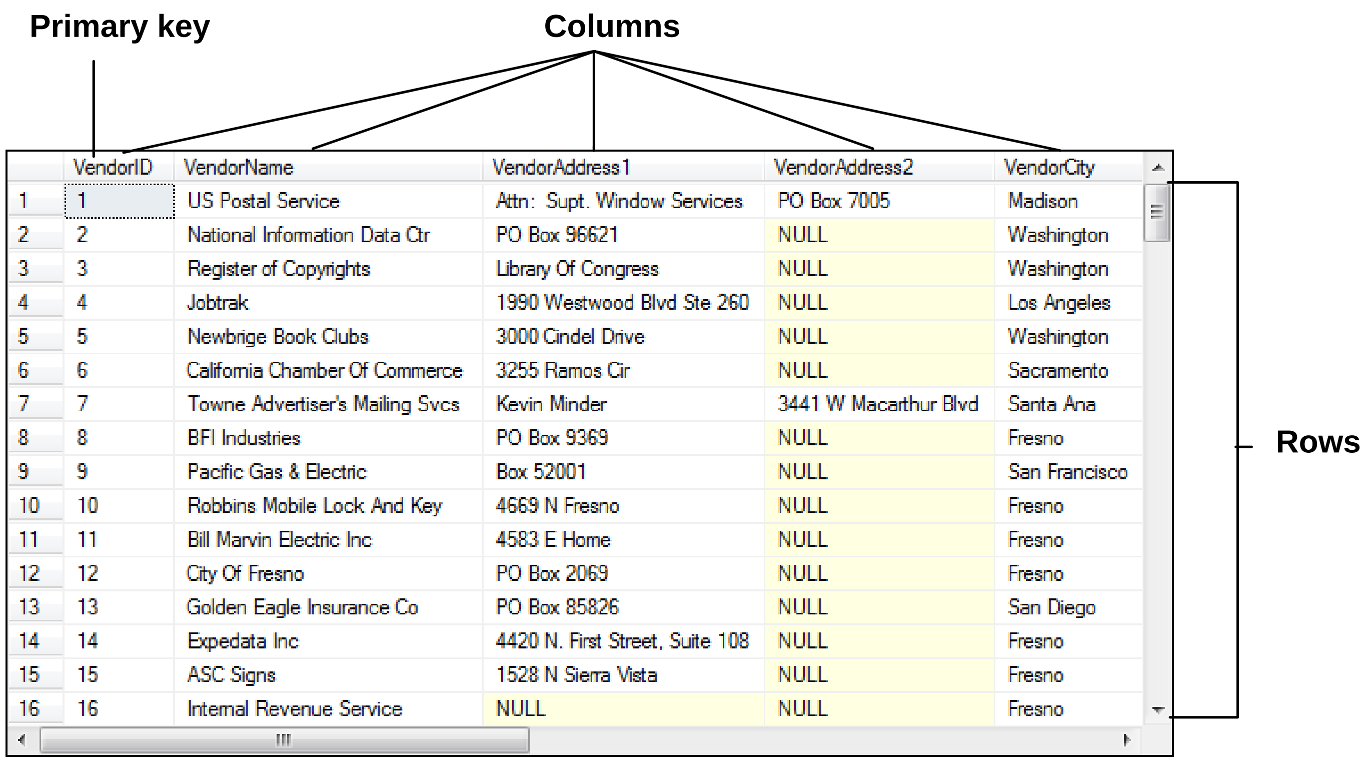
Otras arquitecturas de sistemas cliente/servidor

- Los servidores de aplicaciones almacenan componentes empresariales
- Los servidores web almacenan aplicaciones web y servicios web

Cómo funcionan las aplicaciones web

- Un explorador web de un cliente envía una solicitud a un servidor web.
- El servidor web procesa la solicitud.
- El servidor web pasa las solicitudes de datos al servidor de bases de datos.
- El servidor de bases de datos devuelve los resultados al servidor web.
- El servidor web devuelve una respuesta al navegador.

La tabla Proveedores en una base de datos de proveedores (AP)

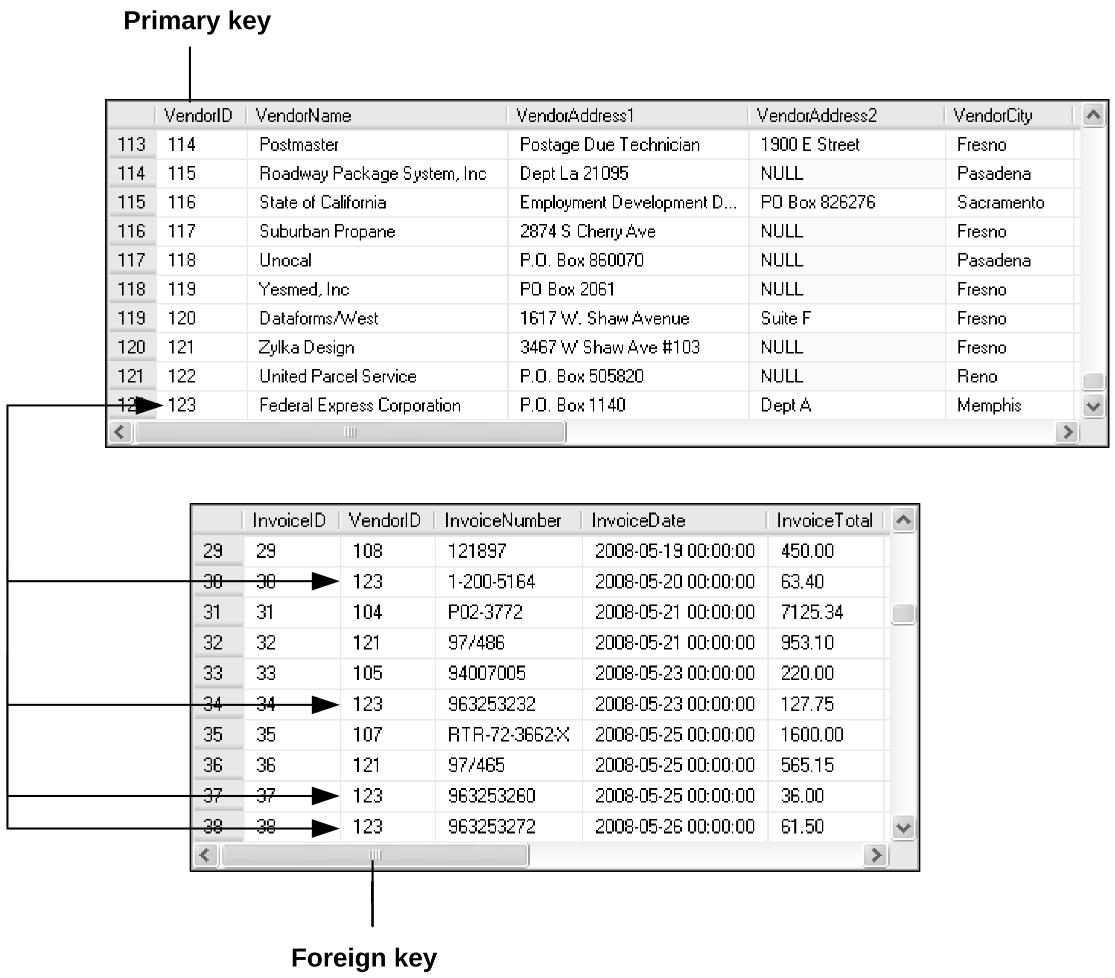


	VendorID	VendorName	VendorAddress1	VendorAddress2	VendorCity
1	1	US Postal Service	Attn: Supt. Window Services	PO Box 7005	Madison
2	2	National Information Data Ctr	PO Box 96621	NULL	Washington
3	3	Register of Copyrights	Library Of Congress	NULL	Washington
4	4	Jobtrak	1990 Westwood Blvd Ste 260	NULL	Los Angeles
5	5	Newbrige Book Clubs	3000 Cindel Drive	NULL	Washington
6	6	California Chamber Of Commerce	3255 Ramos Cir	NULL	Sacramento
7	7	Towne Advertiser's Mailing Svcs	Kevin Minder	3441 W Macarthur Blvd	Santa Ana
8	8	BFI Industries	PO Box 9369	NULL	Fresno
9	9	Pacific Gas & Electric	Box 52001	NULL	San Francisco
10	10	Robbins Mobile Lock And Key	4669 N Fresno	NULL	Fresno
11	11	Bill Marvin Electric Inc	4583 E Home	NULL	Fresno
12	12	City Of Fresno	PO Box 2069	NULL	Fresno
13	13	Golden Eagle Insurance Co	PO Box 85826	NULL	San Diego
14	14	Expedata Inc	4420 N. First Street, Suite 108	NULL	Fresno
15	15	ASC Signs	1528 N Sierra Vista	NULL	Fresno
16	16	Internal Revenue Service	NULL	NULL	Fresno

Terms

- Base de datos relacional
- Mesa
- Columna
- Fila
- Celda
- Clave principal
- Clave principal compuesta
- Clave no principal (clave única)
- Índice

La relación entre dos tablas



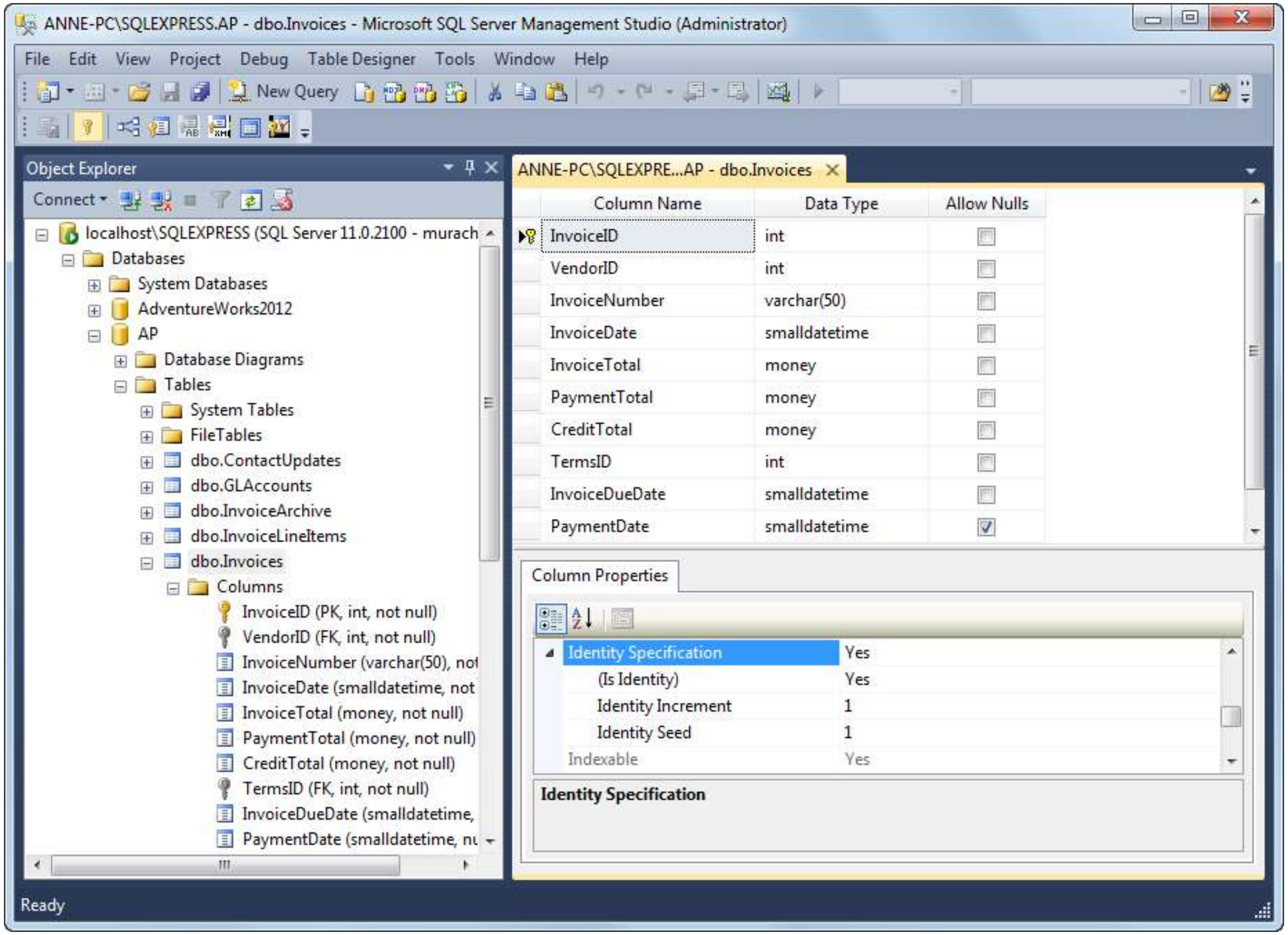
	VendorID	VendorName	VendorAddress1	VendorAddress2	VendorCity
113	114	Postmaster	Postage Due Technician	1900 E Street	Fresno
114	115	Roadway Package System, Inc	Dept La 21095	NULL	Pasadena
115	116	State of California	Employment Development D...	PO Box 826276	Sacramento
116	117	Suburban Propane	2874 S Cherry Ave	NULL	Fresno
117	118	Unocal	P.O. Box 860070	NULL	Pasadena
118	119	Yesmed, Inc	PO Box 2061	NULL	Fresno
119	120	Dataforms/West	1617 W. Shaw Avenue	Suite F	Fresno
120	121	Zylka Design	3467 W Shaw Ave #103	NULL	Fresno
121	122	United Parcel Service	P.O. Box 505820	NULL	Reno
122	123	Federal Express Corporation	P.O. Box 1140	Dept A	Memphis

	InvoiceID	VendorID	InvoiceNumber	InvoiceDate	InvoiceTotal
29	29	108	121897	2008-05-19 00:00:00	450.00
30	30	123	1-200-5164	2008-05-20 00:00:00	63.40
31	31	104	P02-3772	2008-05-21 00:00:00	7125.34
32	32	121	97/486	2008-05-21 00:00:00	953.10
33	33	105	94007005	2008-05-23 00:00:00	220.00
34	34	123	963253232	2008-05-23 00:00:00	127.75
35	35	107	RTR-72-3662-X	2008-05-25 00:00:00	1600.00
36	36	121	97/465	2008-05-25 00:00:00	565.15
37	37	123	963253260	2008-05-25 00:00:00	36.00
38	38	123	963253272	2008-05-26 00:00:00	61.50

Terms

- Clave foránea
- Relación de uno a varios
- Relación uno a uno
- Relación de varios a varios

Las columnas de la tabla Facturas


ANNE-PC\SQLEXPRESS.AP - dbo.Invoices - Microsoft SQL Server Management Studio (Administrator)
File Edit View Project Debug Table Designer Tools Window Help
New Query
Object Explorer
Connect
ANNE-PC\SQLEXPRE...AP - dbo.Invoices
Ready

localhost\SQLEXPRESS (SQL Server 11.0.2100 - murach
- Databases
  - System Databases
  - AdventureWorks2012
  - AP
    - Database Diagrams
    - Tables
      - System Tables
      - FileTables
      - dbo.ContactUpdates
      - dbo.GLAccounts
      - dbo.InvoiceArchive
      - dbo.InvoiceLineItems
      - dbo.Invoices
        - Columns
          - InvoiceID (PK, int, not null)
          - VendorID (FK, int, not null)
          - InvoiceNumber (varchar(50), not
          - InvoiceDate (smalldatetime, not
          - InvoiceTotal (money, not null)
          - PaymentTotal (money, not null)
          - CreditTotal (money, not null)
          - TermsID (FK, int, not null)
          - InvoiceDueDate (smalldatetime,
          - PaymentDate (smalldatetime, nu

Column Name	Data Type	Allow Nulls
InvoiceID	int	☐
VendorID	int	☐
InvoiceNumber	varchar(50)	☐
InvoiceDate	smalldatetime	☐
InvoiceTotal	money	☐
PaymentTotal	money	☐
CreditTotal	money	☐
TermsID	int	☐
InvoiceDueDate	smalldatetime	☐
PaymentDate	smalldatetime	✓

Column Properties

Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes

Identity Specification

Common SQL Server data types

- bit
- int, bigint, smallint, tinyint
- money, smallmoney
- decimal, numeric
- float, real
- datetime, smalldatetime
- char, varchar
- nchar, nvarchar

Terms

- Data type
- Null value
- Default value
- Identity column

A comparison of relational databases and conventional file systems

Feature	File system	Relational database
Definition	Each program must define the file and the layout of the records within the file	Tables, rows, and columns are defined within the database and can be accessed by name
Maintenance	If the definition of a file changes, each program that uses the file must be modified	Programs can be used without modification when the definition of a table changes
Validity checking	Each program that updates a file must include code to check for valid data	Can include checks for valid data

A comparison of relational databases and conventional file systems (continued)

Feature	File system	Relational database
Relationships	Each program must provide for and enforce relationships between files	Can enforce relationships between tables using foreign keys; ad hoc relationships can also be used
Data access	Each I/O operation targets a specific record based on its relative position in the file or its key value	A program can use SQL to access selected data in one or more tables of a database

A comparison of relational databases and other database systems

Feature	Hierarchical database	Network database	Relational database
Supported relationships	One-to-many only	One-to-many, one-to-one, and many-to-many	One-to-many, one-to-one, and many-to-many; ad hoc relationships can also be used
Data access	Programs must include code to navigate through the physical structure of the database	Programs must include code to navigate through the physical structure of the database	Programs can access data without knowing its physical structure

A comparison of relational databases and other database systems (continued)

Feature	Hierarchical database	Network database	Relational database
Maintenance	New and modified relationships can be difficult to implement in application programs	New and modified relationships can be difficult to implement in application programs	Programs can be used without modification when the definition of a table changes

Important events in the history of SQL

Year	Event
1970	Dr. E. F. Codd developed the relational database model.
1978	IBM developed the predecessor to SQL, called Structured English Query Language (SEQUEL).
1979	Relational Software, Inc. (later renamed Oracle) released the first relational DBMS, Oracle.
1982	IBM released their first relational database system, SQL/DS (SQL/Data System).
1985	IBM released DB2 (Database 2).
1987	Microsoft released SQL Server.
1989	ANSI published the first set of standards (ANSI/ISO SQL-89, or SQL1).
1992	ANSI revised standards (ANSI/ISO SQL-92, or SQL2).
1999	ANSI published SQL3 (ANSI/ISO SQL:1999).

Important events in the history of SQL (continued)

Year	Event
2003	ANSI published SQL:2003.
2006	ANSI published SQL:2006.
2008	ANSI published SQL:2008.
2011	Information on these standards is not yet freely available.

How knowing “standard SQL” helps you

- Basic SQL statements are the same for all SQL *dialects*.
- Once you know one SQL dialect, you can easily learn others.

How knowing “standard SQL” does not help you

- Any non-trivial application will require modification when moved from one SQL database to another.

First database releases

Oracle	1979
DB2	1985
MySQL	2000
SQL Server	1987

Primary platforms

Oracle	Unix OS/390 and z/OS
DB2	OS/390 and z/OS Unix
MySQL	Unix Windows Mac OS
SQL Server	Windows

SQL DML statements

- SELECT
- INSERT
- UPDATE
- DELETE

SQL DDL statements

- CREATE DATABASE, TABLE, INDEX
- ALTER TABLE, INDEX
- DROP DATABASE, TABLE, INDEX

A statement that creates a new database

```
CREATE DATABASE AP;
```

A statement that creates a new table

```
CREATE TABLE Invoices
(InvoiceID      INT           NOT NULL IDENTITY
                PRIMARY KEY,
VendorID        INT           NOT NULL
                REFERENCES Vendors(VendorID),
InvoiceNumber   VARCHAR(50)   NOT NULL,
InvoiceDate     SMALLDATETIME NOT NULL,
InvoiceTotal    MONEY         NOT NULL,
PaymentTotal    MONEY         NOT NULL DEFAULT 0,
CreditTotal     MONEY         NOT NULL DEFAULT 0,
TermsID         INT           NOT NULL
                REFERENCES Terms(TermsID),
InvoiceDueDate  SMALLDATETIME NOT NULL,
PaymentDate     SMALLDATETIME NULL);
```

A statement that adds a new column to the table

```
ALTER TABLE Invoices
ADD BalanceDue MONEY NOT NULL;
```

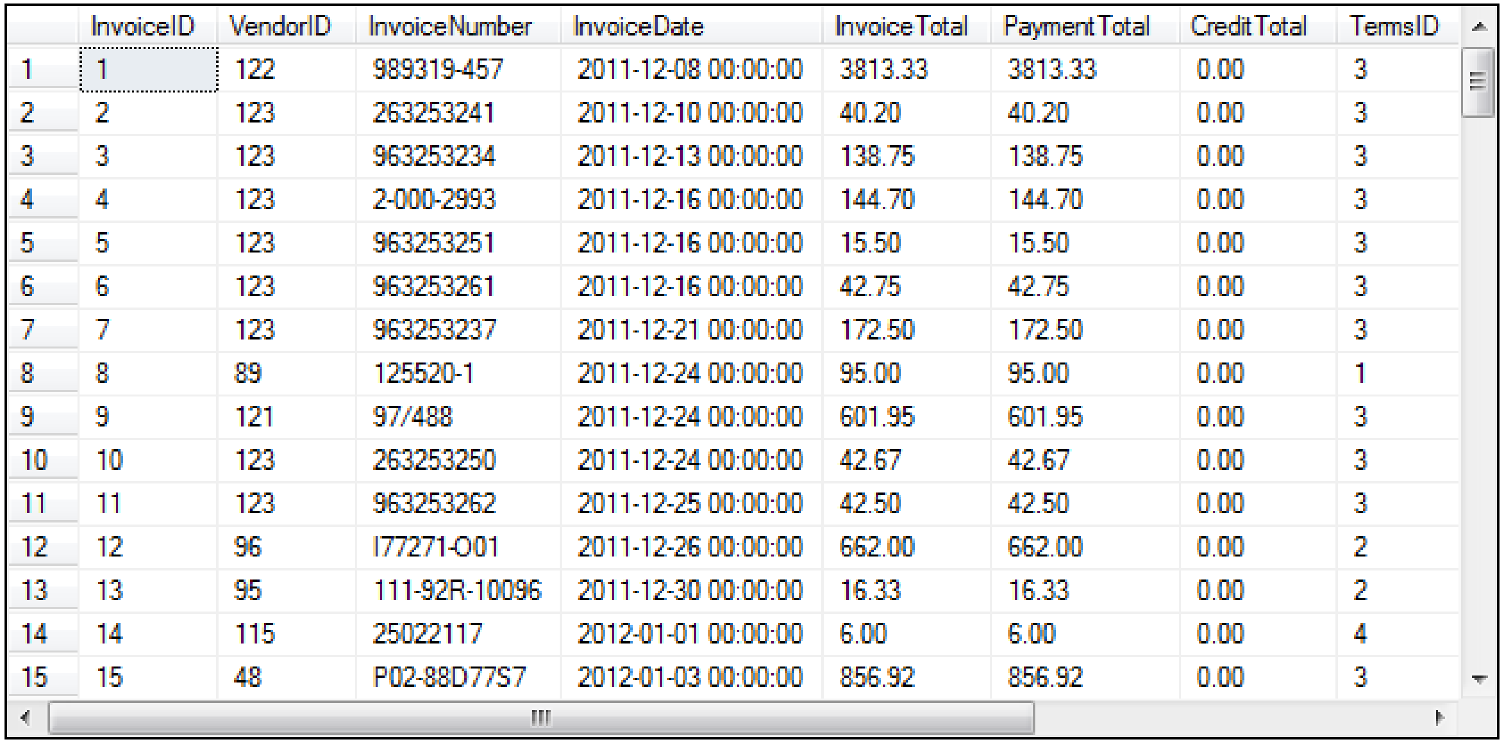
A statement that deletes the new column

```
ALTER TABLE Invoices
DROP COLUMN BalanceDue;
```

A statement that creates an index on the table

```
CREATE INDEX IX_Invoices_VendorID
    ON Invoices (VendorID);
```

The Invoices base table

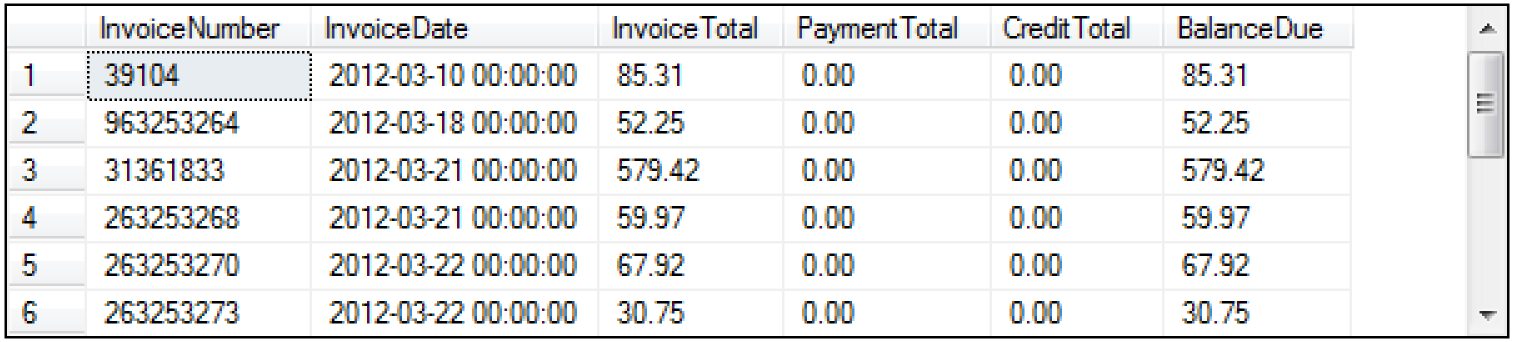


	InvoiceID	VendorID	InvoiceNumber	InvoiceDate	InvoiceTotal	PaymentTotal	CreditTotal	TermsID
1	1	122	989319-457	2011-12-08 00:00:00	3813.33	3813.33	0.00	3
2	2	123	263253241	2011-12-10 00:00:00	40.20	40.20	0.00	3
3	3	123	963253234	2011-12-13 00:00:00	138.75	138.75	0.00	3
4	4	123	2-000-2993	2011-12-16 00:00:00	144.70	144.70	0.00	3
5	5	123	963253251	2011-12-16 00:00:00	15.50	15.50	0.00	3
6	6	123	963253261	2011-12-16 00:00:00	42.75	42.75	0.00	3
7	7	123	963253237	2011-12-21 00:00:00	172.50	172.50	0.00	3
8	8	89	125520-1	2011-12-24 00:00:00	95.00	95.00	0.00	1
9	9	121	97/488	2011-12-24 00:00:00	601.95	601.95	0.00	3
10	10	123	263253250	2011-12-24 00:00:00	42.67	42.67	0.00	3
11	11	123	963253262	2011-12-25 00:00:00	42.50	42.50	0.00	3
12	12	96	I77271-O01	2011-12-26 00:00:00	662.00	662.00	0.00	2
13	13	95	111-92R-10096	2011-12-30 00:00:00	16.33	16.33	0.00	2
14	14	115	25022117	2012-01-01 00:00:00	6.00	6.00	0.00	4
15	15	48	P02-88D77S7	2012-01-03 00:00:00	856.92	856.92	0.00	3

A SELECT statement that retrieves and sorts selected columns and rows

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
    PaymentTotal, CreditTotal,
    InvoiceTotal - PaymentTotal - CreditTotal
    AS BalanceDue
FROM Invoices
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
ORDER BY InvoiceDate;
```

The result set defined by the SELECT statement

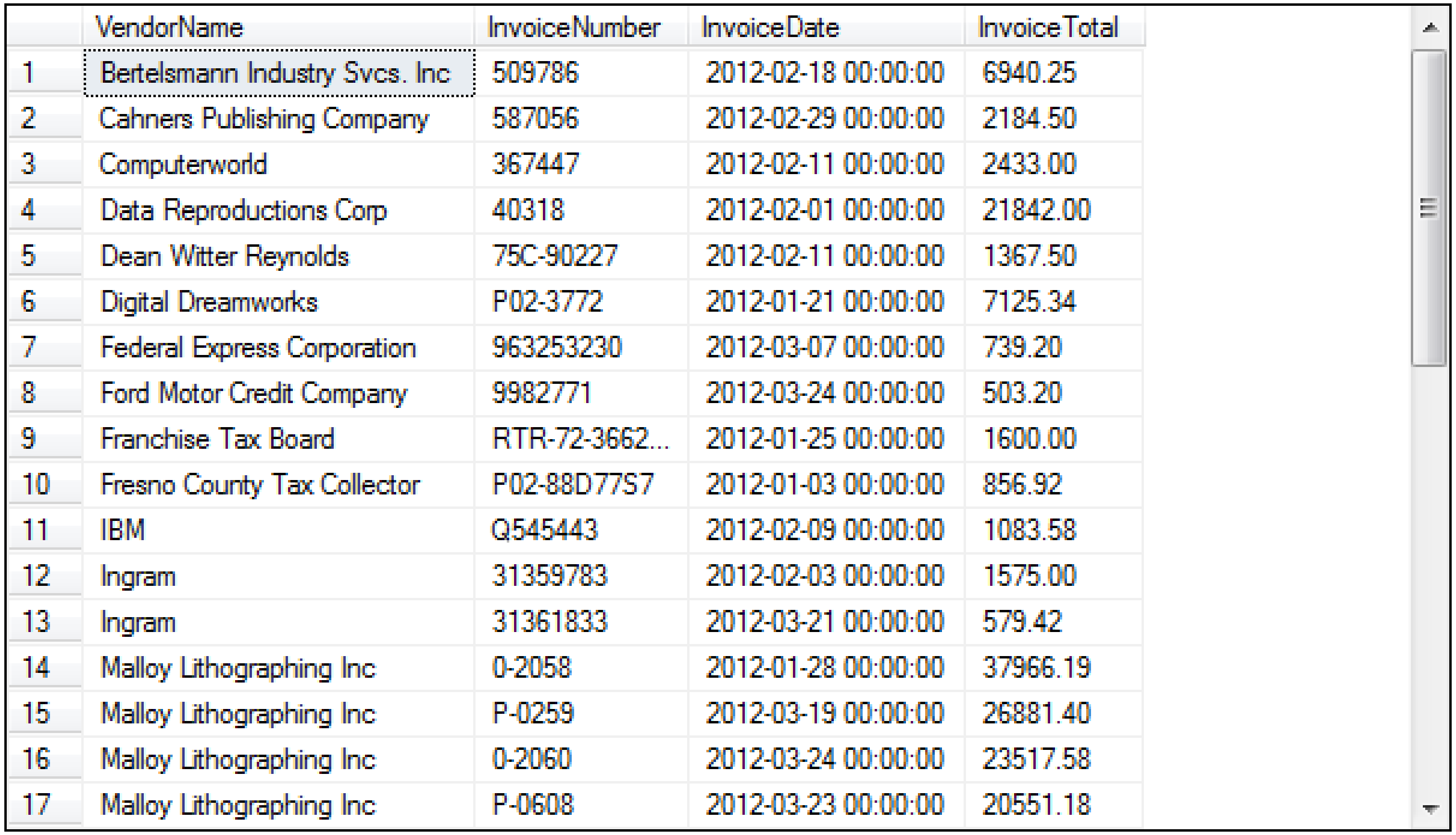


	InvoiceNumber	InvoiceDate	InvoiceTotal	PaymentTotal	CreditTotal	BalanceDue
1	39104	2012-03-10 00:00:00	85.31	0.00	0.00	85.31
2	963253264	2012-03-18 00:00:00	52.25	0.00	0.00	52.25
3	31361833	2012-03-21 00:00:00	579.42	0.00	0.00	579.42
4	263253268	2012-03-21 00:00:00	59.97	0.00	0.00	59.97
5	263253270	2012-03-22 00:00:00	67.92	0.00	0.00	67.92
6	263253273	2012-03-22 00:00:00	30.75	0.00	0.00	30.75

A SELECT statement that joins data from the Vendors and Invoices tables

```
SELECT VendorName, InvoiceNumber, InvoiceDate,
       InvoiceTotal
FROM Vendors INNER JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
WHERE InvoiceTotal >= 500
ORDER BY VendorName, InvoiceTotal DESC;
```

The result set defined by the SELECT statement



	VendorName	InvoiceNumber	InvoiceDate	InvoiceTotal
1	Bertelsmann Industry Svcs. Inc	509786	2012-02-18 00:00:00	6940.25
2	Cahners Publishing Company	587056	2012-02-29 00:00:00	2184.50
3	Computerworld	367447	2012-02-11 00:00:00	2433.00
4	Data Reproductions Corp	40318	2012-02-01 00:00:00	21842.00
5	Dean Witter Reynolds	75C-90227	2012-02-11 00:00:00	1367.50
6	Digital Dreamworks	P02-3772	2012-01-21 00:00:00	7125.34
7	Federal Express Corporation	963253230	2012-03-07 00:00:00	739.20
8	Ford Motor Credit Company	9982771	2012-03-24 00:00:00	503.20
9	Franchise Tax Board	RTR-72-3662...	2012-01-25 00:00:00	1600.00
10	Fresno County Tax Collector	P02-88D77S7	2012-01-03 00:00:00	856.92
11	IBM	Q545443	2012-02-09 00:00:00	1083.58
12	Ingram	31359783	2012-02-03 00:00:00	1575.00
13	Ingram	31361833	2012-03-21 00:00:00	579.42
14	Malloy Lithographing Inc	0-2058	2012-01-28 00:00:00	37966.19
15	Malloy Lithographing Inc	P-0259	2012-03-19 00:00:00	26881.40
16	Malloy Lithographing Inc	0-2060	2012-03-24 00:00:00	23517.58
17	Malloy Lithographing Inc	P-0608	2012-03-23 00:00:00	20551.18

Terms

- Base table
- Result set
- Calculated value
- Query
- Join
- Inner join
- Outer join

Add a row to the Invoices table

```
INSERT INTO Invoices (VendorID, InvoiceNumber, InvoiceDate,
    InvoiceTotal, TermsID, InvoiceDueDate)
VALUES (12, '3289175', '4/18/2012', 165, 3, '5/18/2012');
```

Change the value of a column for a selected row

```
UPDATE Invoices
SET CreditTotal = 35.89
WHERE InvoiceNumber = '367447';
```

Change the value in a column for all rows that satisfy the search condition

```
UPDATE Invoices
SET InvoiceDueDate = InvoiceDueDate + 30
WHERE TermsID = 4;
```

Delete a selected invoice from the Invoices table

```
DELETE FROM Invoices
WHERE InvoiceNumber = '4-342-8069';
```

Delete all paid invoices from the Invoices table

```
DELETE FROM Invoices
WHERE InvoiceTotal - PaymentTotal - CreditTotal = 0;
```

A SELECT statement that’s difficult to read

```
select invoicenumber, invoicedate, invoicetotal,
invoicetotal - paymenttotal - credittotal as balancedue
from invoices where invoicetotal - paymenttotal -
credittotal > 0 order by invoicedate
```

A SELECT statement that’s coded with a readable style

```
Select InvoiceNumber, InvoiceDate, InvoiceTotal,
    InvoiceTotal - PaymentTotal - CreditTotal
    As BalanceDue
From Invoices
Where InvoiceTotal - PaymentTotal - CreditTotal > 0
Order By InvoiceDate;
```

A SELECT statement with a block comment

```
/*
Author: Bryan Syverson
Date: 8/22/12
*/
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
    InvoiceTotal – PaymentTotal – CreditTotal
    AS BalanceDue
FROM Invoices;
```

A SELECT statement with a single-line comment

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
    InvoiceTotal – PaymentTotal – CreditTotal
    AS BalanceDue
    -- The fourth column calculates the balance due
FROM Invoices;
```

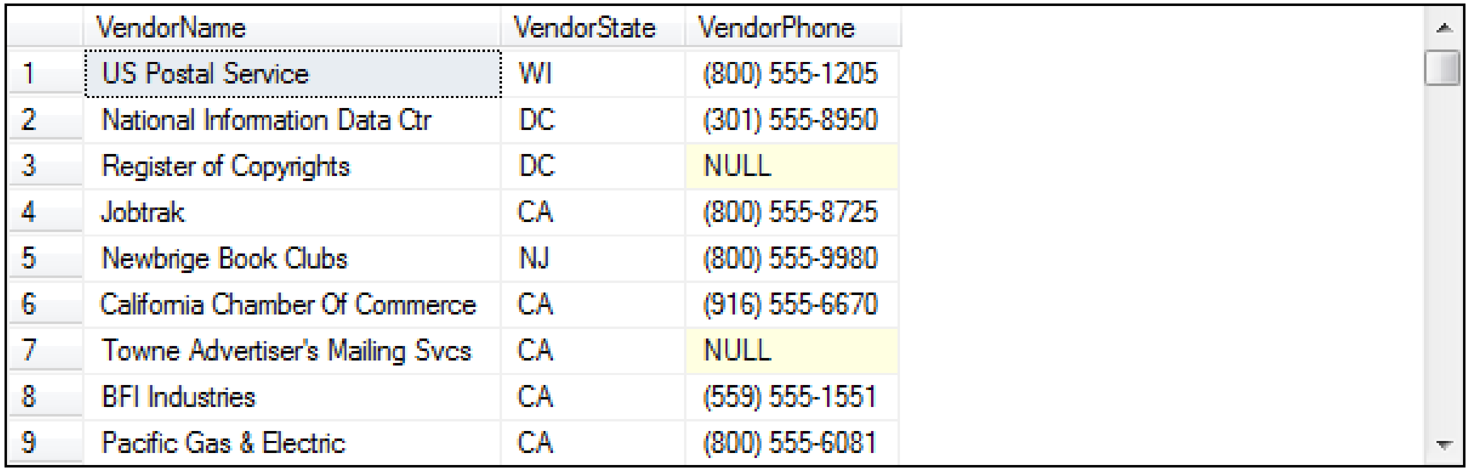
Recomendaciones de codificación SQL

- Comience cada nueva cláusula en una nueva línea.
- Divida las cláusulas largas en varias líneas y aplique sangría a las líneas continuas.
- Escriba en mayúscula la primera letra de cada palabra clave y cada palabra en los nombres de columnas y tablas.
- Termine cada instrucción con un punto y coma (;).
- Use comentarios solo para las partes del código que son difíciles de entender.

A CREATE VIEW statement for a view named VendorsMin

```
CREATE VIEW VendorsMin AS
    SELECT VendorName, VendorState, VendorPhone
    FROM Vendors;
```

The virtual table that's represented by the view

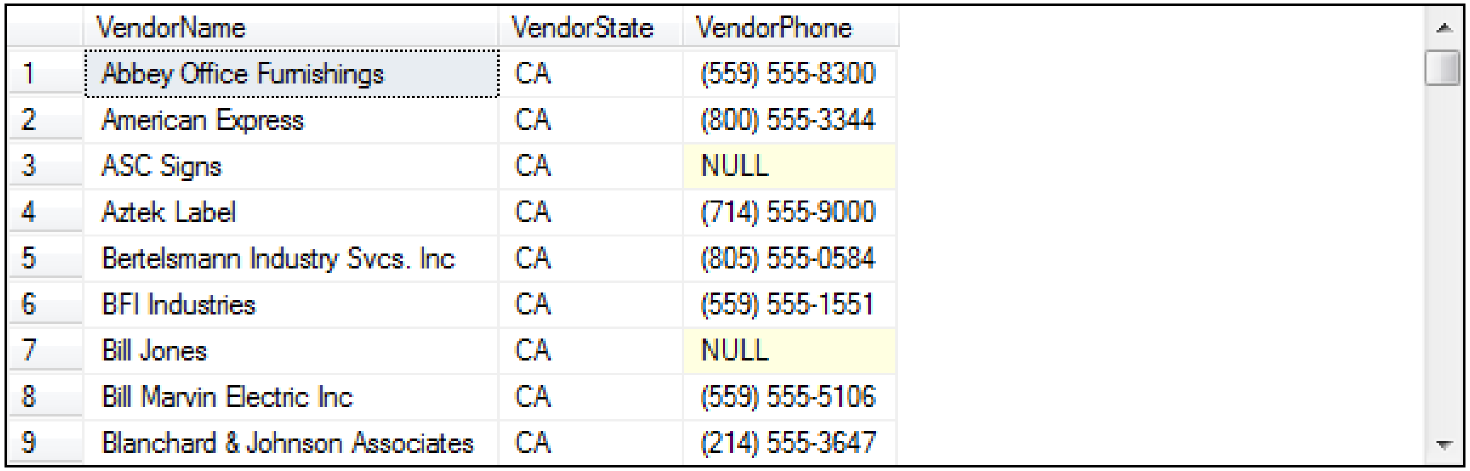


	VendorName	VendorState	VendorPhone
1	US Postal Service	WI	(800) 555-1205
2	National Information Data Ctr	DC	(301) 555-8950
3	Register of Copyrights	DC	NULL
4	Jobtrak	CA	(800) 555-8725
5	Newbrige Book Clubs	NJ	(800) 555-9980
6	California Chamber Of Commerce	CA	(916) 555-6670
7	Towne Advertiser's Mailing Svcs	CA	NULL
8	BFI Industries	CA	(559) 555-1551
9	Pacific Gas & Electric	CA	(800) 555-6081

A SELECT statement that uses the VendorsMin view

```
SELECT * FROM VendorsMin
WHERE VendorState = 'CA'
ORDER BY VendorName;
```

The result set that's returned by the SELECT statement



	VendorName	VendorState	VendorPhone
1	Abbey Office Furnishings	CA	(559) 555-8300
2	American Express	CA	(800) 555-3344
3	ASC Signs	CA	NULL
4	Aztek Label	CA	(714) 555-9000
5	Bertelsmann Industry Svcs. Inc	CA	(805) 555-0584
6	BFI Industries	CA	(559) 555-1551
7	Bill Jones	CA	NULL
8	Bill Marvin Electric Inc	CA	(559) 555-5106
9	Blanchard & Johnson Associates	CA	(214) 555-3647

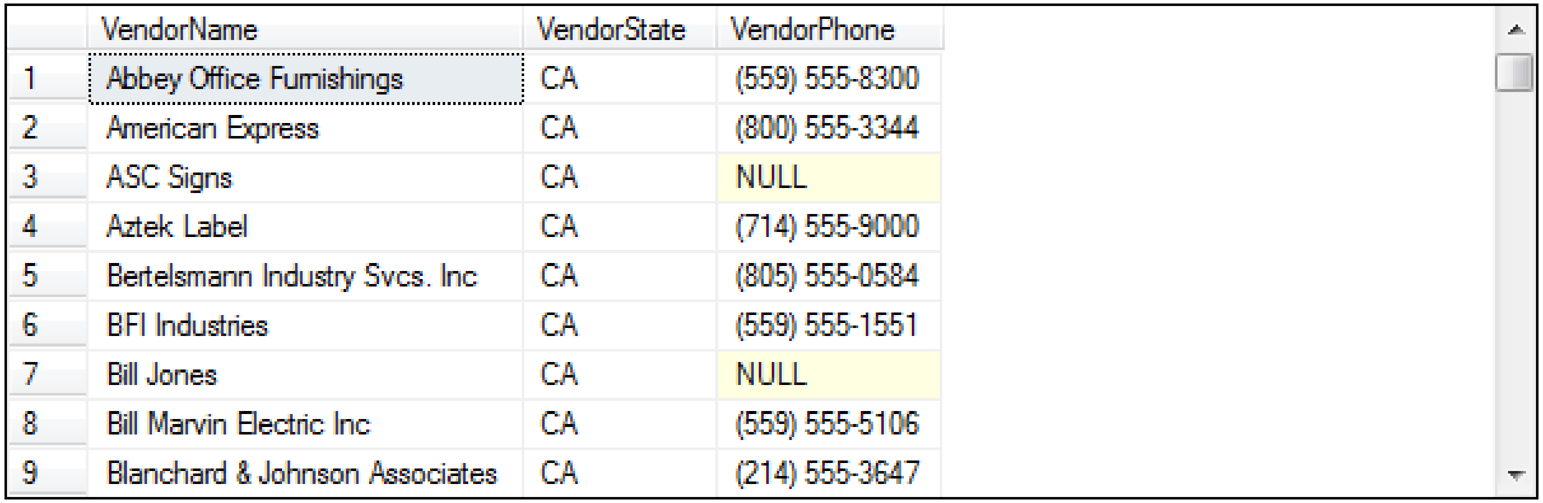
A CREATE PROCEDURE statement for a procedure named spVendorsByState

```
CREATE PROCEDURE spVendorsByState @State char(2) AS
    SELECT VendorName, VendorState, VendorPhone
    FROM Vendors
    WHERE VendorState = @State
    ORDER BY VendorName;
```

Instrucción que ejecuta el procedimiento almacenado spVendorsByState

```
EXEC spVendorsByState 'CA';
```

El conjunto de resultados

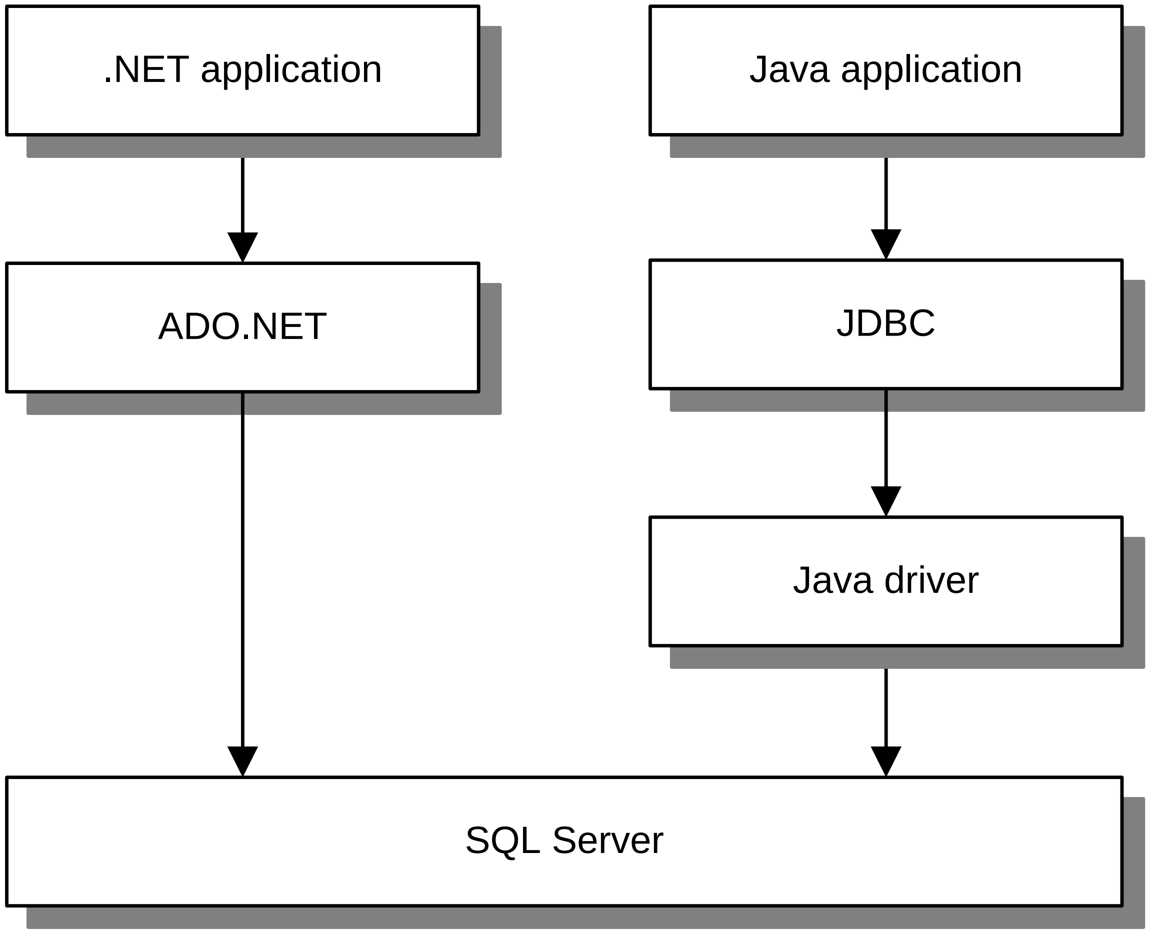


	VendorName	VendorState	VendorPhone
1	Abbey Office Furnishings	CA	(559) 555-8300
2	American Express	CA	(800) 555-3344
3	ASC Signs	CA	NULL
4	Aztek Label	CA	(714) 555-9000
5	Bertelsmann Industry Svcs. Inc	CA	(805) 555-0584
6	BFI Industries	CA	(559) 555-1551
7	Bill Jones	CA	NULL
8	Bill Marvin Electric Inc	CA	(559) 555-5106
9	Blanchard & Johnson Associates	CA	(214) 555-3647

Letra chica

- Procedimiento almacenado
- Lenguaje de control de flujo
- Detonante
- Función definida por el usuario (UDF)

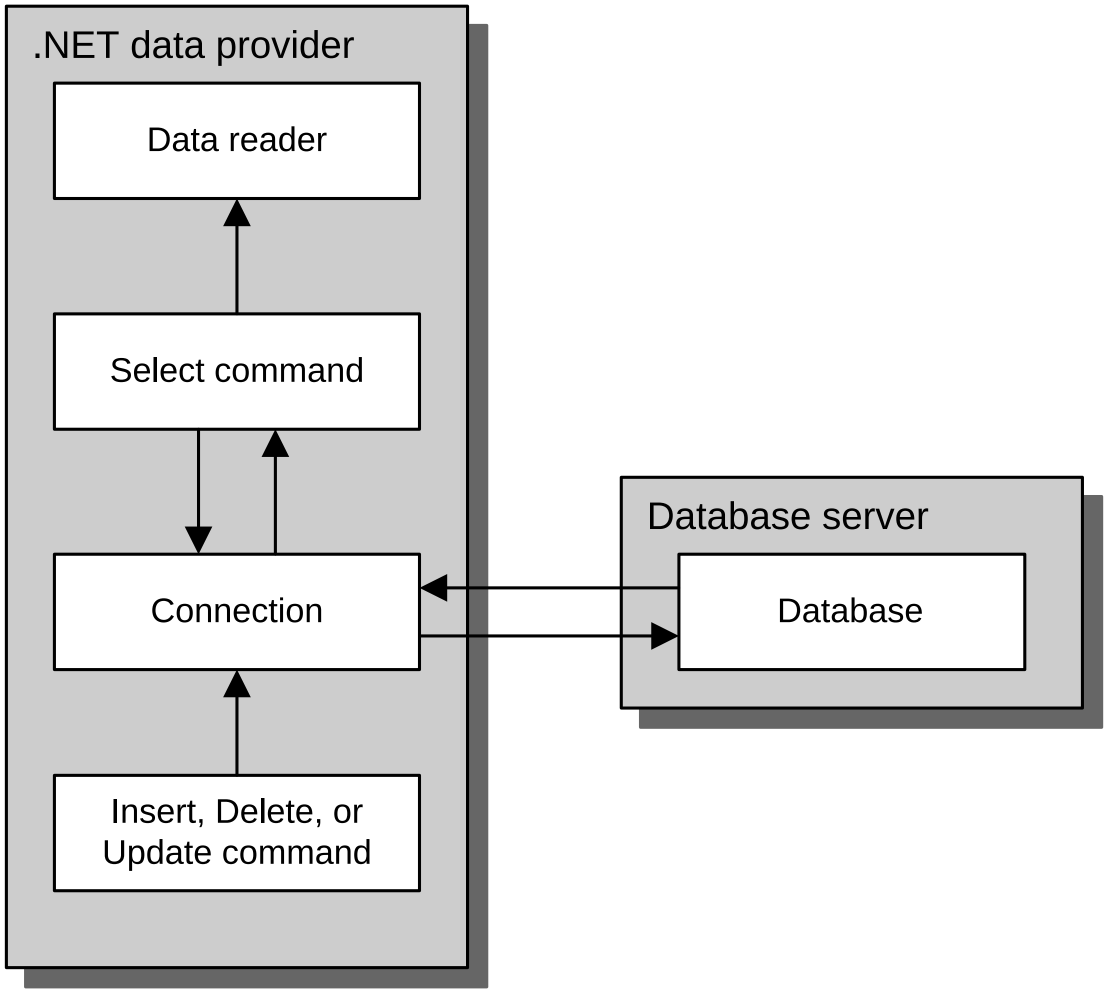
Opciones comunes para acceder a los datos de SQL Server



Letra chica

- Modelo de acceso a datos
- ADO.NET (para lenguajes .NET)
- JDBC (para Java)
- Controlador de base de datos

Basic ADO.NET objects in a .NET application



Letra chica

- Proveedor de datos .NET
- Objeto de comando
- Objeto de conexión
- Objeto lector de datos
- Arquitectura de datos desconectada

A Visual Basic function that uses ADO.NET to retrieve data from SQL Server

```
Public Shared Function GetVendor(
        vendorID As Integer) As Vendor
    Dim vendor As New Vendor

    ' Create the connection object
    Dim connection As New SqlConnection()
    connection.ConnectionString =
        "Data Source=localhost\SqlExpress;" &
        "Initial Catalog=AP;Integrated Security=True"

    ' Create the command object and set the connection,
    ' SELECT statement, and parameter value
    Dim selectCommand As New SqlCommand
    selectCommand.Connection = connection
    selectCommand.CommandText = "SELECT VendorID, " &
        "VendorName, VendorAddress1, VendorAddress2, " &
        "VendorCity, VendorState, VendorZipCode " &
        "FROM Vendors WHERE VendorID = @VendorID"
    selectCommand.Parameters.AddWithValue(
        "@VendorID", vendorID)
```

A Visual Basic function that uses ADO.NET to retrieve data from SQL Server (continued)

```
' Open the connection to the database
    connection.Open()

    ' Retrieve the row specified by the SELECT statement
    ' and load it into the Vendor object
    Dim reader As SqlDataReader =
        selectCommand.ExecuteReader
    If reader.Read Then
        vendor.VendorID = CInt(reader("VendorID"))
        vendor.VendorName = reader("VendorName").ToString
        vendor.VendorAddress1 =
            reader("VendorAddress1").ToString
        vendor.VendorAddress2 =
            reader("VendorAddress2").ToString
        vendor.VendorCity = reader("VendorCity").ToString
        vendor.VendorState = reader("VendorState").ToString
        vendor.VendorZipCode =
            reader("VendorZipCode").ToString
```

A Visual Basic function that uses ADO.NET to retrieve data from SQL Server (continued)

```
    Else
        vendor = Nothing
    End If
    reader.Close()

    ' Close the connection to the database
    connection.Close()

    Return vendor
End Function
```

A C# method that uses ADO.NET to retrieve data from SQL Server

```
public static Vendor GetVendor(int vendorID)
{
    Vendor vendor = new Vendor();

    // Create the connection object
    SqlConnection connection = new SqlConnection();
    connection.ConnectionString =
        "Data Source=localhost\\SqlExpress;" +
        "Initial Catalog=AP;Integrated Security=True";

    // Create the command object and set the connection,
    // SELECT statement, and parameter value
    SqlCommand selectCommand = new SqlCommand();
    selectCommand.Connection = connection;
    selectCommand.CommandText = "SELECT VendorID, " +
        "VendorName, VendorAddress1, VendorAddress2, " +
        "VendorCity, VendorState, VendorZipCode " +
        "FROM Vendors WHERE VendorID = @VendorID";
    selectCommand.Parameters.AddWithValue(
        "@VendorID", vendorID);
```

A C# method that uses ADO.NET to retrieve data from SQL Server (continued)

```
// Open the connection to the database
connection.Open();

// Retrieve the row specified by the SELECT statement
// and load it into the Vendor object
SqlDataReader reader = selectCommand.ExecuteReader();
if (reader.Read())
{
    vendor.VendorID = (int)reader["VendorID"];
    vendor.VendorName =
        reader["VendorName"].ToString();
    vendor.VendorAddress1 =
        reader["VendorAddress1"].ToString();
    vendor.VendorAddress2 =
       reader["VendorAddress2"].ToString();
    vendor.VendorCity =
       reader["VendorCity"].ToString();
    vendor.VendorState =
       reader["VendorState"].ToString();
```

A C# method that uses ADO.NET to retrieve data from SQL Server (continued)

```
            vendor.VendorZipCode =
                reader["VendorZipCode"].ToString();
        }
        else
        {
            vendor = null;
        }
        reader.Close();

        // Close the connection to the database
        connection.Close();

        return vendor;
    }
```

Capítulo 11

Cómo crear y mantener bases de datos, tablas y secuencias con sentencias SQL

Objetivos

Aplicado

- Dado un diseño de base de datos completo, escriba las instrucciones DDL de SQL para crear la base de datos, incluidas todas las tablas, relaciones, restricciones, índices y secuencias.

Conocimiento

- Describa cómo cada uno de estos tipos de restricciones restringe los valores que se pueden almacenar en una tabla: NOT NULL, PRIMARY KEY, UNIQUE, CHECK y FOREIGN KEY (o REFERENCES).
- Describa la diferencia entre una restricción de nivel de columna y una restricción de nivel de tabla.
- Explicar en qué se diferencian las opciones CASCADE y NO ACTION en la aplicación de la integridad referencial en las eliminaciones y actualizaciones.

Objetivos (cont.)

- Describa el uso de una secuencia.
- Describir el uso de un script que contiene uno o varios lotes para crear una base de datos.

Instrucciones DDL para crear, modificar y eliminar objetos

CREATE DATABASE

CREATE TABLE

CREATE INDEX

CREATE SEQUENCE

CREATE FUNCTION

CREATE PROCEDURE

CREATE TRIGGER

CREATE VIEW

ALTER TABLE

ALTER SEQUENCE

ALTER FUNCTION

Instrucciones DDL para crear, modificar y eliminar objetos (continuación)

ALTER PROCEDURE

ALTER TRIGGER

ALTER VIEW

DROP DATABASE

DROP TABLE

DROP SEQUENCE

DROP INDEX

DROP FUNCTION

DROP PROCEDURE

DROP TRIGGER

DROP VIEW

Reglas de formato para identificadores

- El primer carácter de un identificador debe ser una letra, tal como se define en el estándar Unicode 2.0, un carácter de subrayado (_), un signo de arroba (@) o un signo de número (#).
- Todos los caracteres después del primero deben ser una letra, tal como se define en el estándar Unicode 2.0, un número, un signo arroba, un signo de dólar ($), un signo numérico o un guión bajo.
- Un identificador no puede ser una palabra clave reservada de Transact-SQL.
- Un identificador no puede contener espacios ni caracteres especiales distintos de los ya mencionados.

Valid regular identifiers

```
Employees
#PaidInvoices
ABC$123
Invoice_Line_Items
@TotalDue
```

Valid delimited identifiers

```
[%Increase]
"Invoice Line Items"
[@TotalDue]
```

Basic syntax of the CREATE DATABASE statement

```
CREATE DATABASE database_name
    [ON [PRIMARY] (FILENAME = 'file_name')]
    [FOR ATTACH]
```

Create a new database

```
CREATE DATABASE New_AP;
```

The response from the system

```
Command(s) completed successfully.
```

Attach an existing database file

```
CREATE DATABASE Test_AP
    ON PRIMARY (FILENAME =
      'C:\Murach\SQL Server 2012\Databases\Test_AP.mdf')
    FOR ATTACH;
```

The response from the system

```
Command(s) completed successfully.
```

Basic syntax of the CREATE TABLE statement

```
CREATE TABLE table_name
(column_name_1 data_type [column_attributes]
[, column_name_2 data_type [column_attributes]]...
[, table_attributes])
```

Common column attributes

- `NULL|NOT NULL`
- `PRIMARY KEY|UNIQUE`
- `IDENTITY`
- `DEFAULT default_value`
- `SPARSE`

Create a table without column attributes

```
CREATE TABLE Vendors
(VendorID       INT,
VendorName      VARCHAR(50));
```

Create a table with column attributes

```
CREATE TABLE Invoices
(InvoiceID      INT             PRIMARY KEY IDENTITY,
VendorID        INT             NOT NULL,
InvoiceDate     SMALLDATETIME   NULL,
InvoiceTotal    MONEY           NULL DEFAULT 0);
```

A column definition that uses the SPARSE attribute

```
VendorAddress2  VARCHAR(50) SPARSE NULL
```

Basic syntax of the CREATE INDEX statement

```
CREATE [CLUSTERED|NONCLUSTERED] INDEX index_name
    ON table_name (col_name_1 [ASC|DESC]
                 [, col_name_2 [ASC|DESC]]...)
 [WHERE filter-condition]
```

Create a nonclustered index on a single column

```
CREATE INDEX IX_VendorID
    ON Invoices (VendorID);
```

Create a nonclustered index on two columns

```
CREATE INDEX IX_Invoices
    ON Invoices (InvoiceDate DESC, InvoiceTotal);
```

Note

- SQL Server automatically creates a clustered index for a table’s primary key.

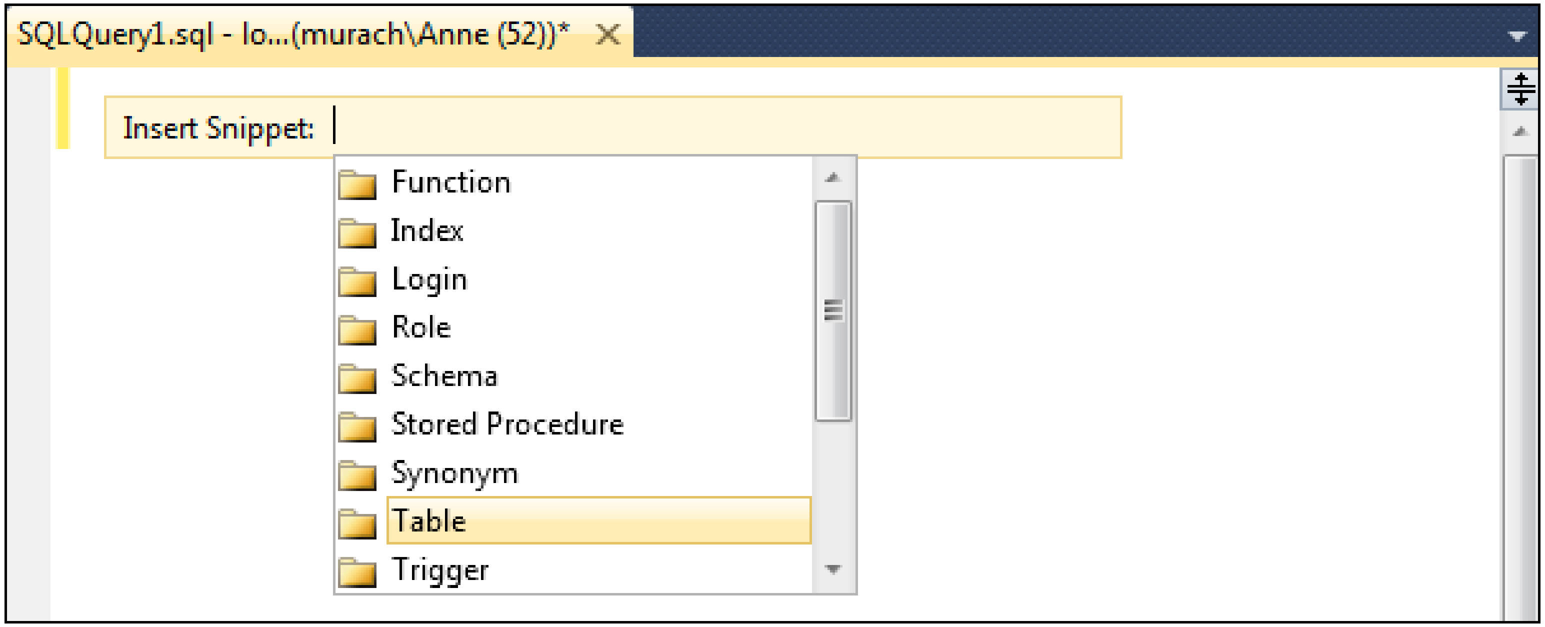
Create a filtered index for a subset of data in a column

```
CREATE INDEX IX_InvoicesPaymentFilter
    ON Invoices (InvoiceDate DESC, InvoiceTotal)
WHERE PaymentDate IS NULL;
```

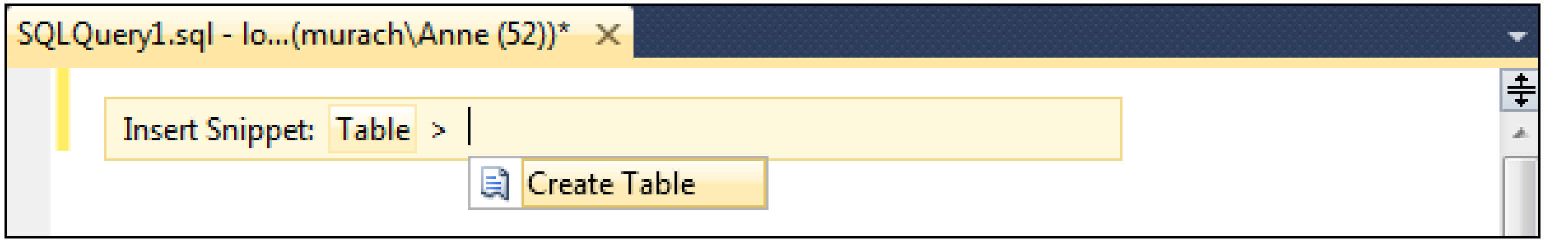
Create a filtered index for categories in a column

```
CREATE INDEX IX_InvoicesDateFilter
    ON Invoices (InvoiceDate DESC, InvoiceTotal)
WHERE InvoiceDate > '2012-02-01';
```

The snippet picker with a list of object folders



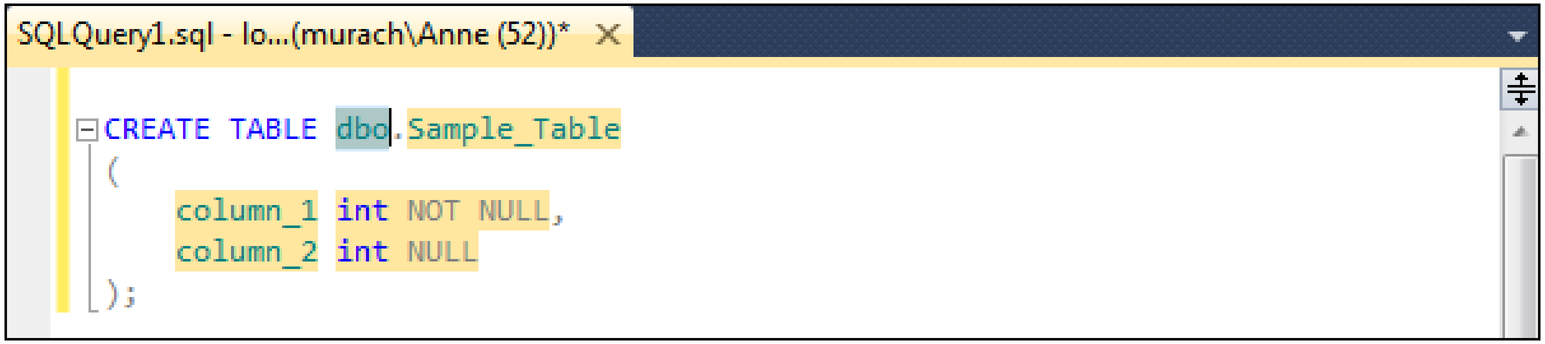
The snippet picker with the list of snippets for a table



Insert Snippet: Table >

Create Table

The CREATE TABLE snippet after it has been inserted



```
CREATE TABLE dbo.Sample_Table
(
    column_1 int NOT NULL,
    column_2 int NULL
);
```

Terms

- Archivo de registro de transacciones
- Adjuntar un archivo de base de datos
- Índice de tabla completa
- Índice filtrado

Column-level constraints

Constraint	Description
`NOT NULL`	Prevents null values from being stored in the column.
`PRIMARY KEY`	Requires that each row in the table have a unique value in the column. Null values are not allowed.
`UNIQUE`	Requires that each row in the table have a unique value in the column.
`CHECK`	Limits the values for a column.
`[FOREIGN KEY] REFERENCES`	Enforces referential integrity between a column in the new table and a column in a related table.

Table-level constraints

Constraint	Description
`PRIMARY KEY`	Requires that each row in the table have a unique set of values over one or more columns. Null values are not allowed.
`UNIQUE`	Requires that each row in the table have a unique set of values over one or more columns.
`CHECK`	Limits the values for one or more columns.
`[FOREIGN KEY] REFERENCES`	Enforces referential integrity between one or more columns in the new table and one or more columns in the related table.

Create a table with a two-column primary key constraint

```
CREATE TABLE InvoiceLineItems1
(InvoiceID                  INT           NOT NULL,
InvoiceSequence             SMALLINT      NOT NULL,
InvoiceLineItemAmount       MONEY         NOT NULL,
PRIMARY KEY (InvoiceID, InvoiceSequence));
```

Create a table with two column-level check constraints

```
CREATE TABLE Invoices1
(InvoiceID       INT   NOT NULL IDENTITY PRIMARY KEY,
InvoiceTotal     MONEY NOT NULL
                 CHECK (InvoiceTotal >= 0),
PaymentTotal     MONEY NOT NULL DEFAULT 0
                 CHECK (PaymentTotal >= 0));
```

The same check constraints coded at the table level

```
CREATE TABLE Invoices2
(InvoiceID       INT   NOT NULL IDENTITY PRIMARY KEY,
InvoiceTotal     MONEY NOT NULL,
PaymentTotal     MONEY NOT NULL DEFAULT 0,
CHECK ((InvoiceTotal >= 0) AND (PaymentTotal >= 0)));
```

The syntax of a check constraint

```
CHECK (condition)
```

A column-level check constraint

```
CREATE TABLE Invoices3
(InvoiceID      INT   NOT NULL IDENTITY PRIMARY KEY,
InvoiceTotal    MONEY NOT NULL CHECK (InvoiceTotal > 0));
```

An INSERT statement that fails due to the check constraint

```
INSERT Invoices3
VALUES (-100);
```

The response from the system

```
The INSERT statement conflicted with the CHECK constraint
"CK__Invoices3__Invoi__0BC6C43E". The conflict occurred in
database "New_AP", table "dbo.Invoices3", column
'InvoiceTotal'.
The statement has been terminated.
```

A table-level check constraint

```
CREATE TABLE Vendors1
(VendorCode      CHAR(6)     NOT NULL PRIMARY KEY,
VendorName       VARCHAR(50) NOT NULL,
CHECK ((VendorCode LIKE '[A-Z][A-Z][0-9][0-9][0-9][0-9]')
  AND  (LEFT(VendorCode,2) = LEFT(VendorName,2))));
```

An INSERT statement that fails due to the check constraint

```
INSERT Vendors1
VALUES ('Mc4559','Castle Printers, Inc.');
```

The response from the system

```
The INSERT statement conflicted with the CHECK constraint
"CK__Vendors1__164452B1". The conflict occurred in database
"New_AP", table "dbo.Vendors1".
The statement has been terminated.
```

The syntax of a column-level foreign key constraint

```
[FOREIGN KEY] REFERENCES ref_table_name (ref_column_name)
    [ON DELETE {CASCADE|NO ACTION}]
    [ON UPDATE {CASCADE|NO ACTION}]
```

The syntax of a table-level foreign key constraint

```
FOREIGN KEY (column_name_1 [, column_name_2]...)
    REFERENCES ref_table_name (ref_column_name_1
                               [, ref_column_name_2]...)
    [ON DELETE {CASCADE|NO ACTION}]
    [ON UPDATE {CASCADE|NO ACTION}]
```

A column-level foreign key constraint

A statement that creates the primary key table

```
CREATE TABLE Vendors9
(VendorID          INT NOT NULL PRIMARY KEY,
VendorName         VARCHAR(50) NOT NULL);
```

A statement that creates the foreign key table

```
CREATE TABLE Invoices9
(InvoiceID         INT NOT NULL PRIMARY KEY,
VendorID           INT NOT NULL REFERENCES Vendors9
(VendorID),
InvoiceTotal       MONEY NULL);
```

A column-level foreign key constraint (continued)

An INSERT statement that fails because a related row doesn’t exist

```
INSERT Invoices9
VALUES (1, 99, 100);
```

The response from the system

```
The INSERT statement conflicted with the FOREIGN KEY
constraint "FK__Invoices9__Vendo__1367E606". The conflict
occurred in database "New_AP", table "dbo.Vendors9",
column 'VendorID'.
The statement has been terminated.
```

Terms

- Constraint
- Column-level constraint
- Table-level constraint
- Check constraint
- Foreign key constraint
- Reference constraint
- Cascading delete
- Cascading update

The syntax of the DROP INDEX statement

```
DROP INDEX index_name_1 ON table_name_1
        [, index_name_2 ON table_name_2]...
```

Delete an index from the Invoices table

```
DROP INDEX IX_Invoices ON Invoices;
```

Note

- You can’t delete an index that’s based on a primary key or unique key constraint. To do that, you have to use the ALTER TABLE statement.

The syntax of the DROP TABLE statement

```
DROP TABLE table_name_1 [, table_name_2]...
```

Delete a table from the current database

```
DROP TABLE Vendors1;
```

Qualify the table to be deleted

```
DROP TABLE New_AP.dbo.Vendors1;
```

Notes

- You can’t delete a table if a foreign key constraint in another table refers to that table.
- When you delete a table, all of the data, indexes, triggers, and constraints are deleted. Any views or stored procedures associated with the table must be deleted explicitly.

The syntax of the DROP DATABASE statement

```
DROP DATABASE database_name_1 [, database_name_2]...
```

A statement that deletes a database

```
DROP DATABASE New_AP;
```

The basic syntax of the ALTER TABLE statement

```
ALTER TABLE table_name [WITH CHECK|WITH NOCHECK]
{ADD new_column_name data_type [column_attributes] |
 DROP COLUMN column_name |
 ALTER COLUMN column_name new_data_type [NULL|NOT NULL] |
 ADD [CONSTRAINT] new_constraint_definition |
 DROP [CONSTRAINT] constraint_name}
```

Add a new column

```
ALTER TABLE Vendors
ADD LastTranDate SMALLDATETIME NULL;
```

Drop a column

```
ALTER TABLE Vendors
DROP COLUMN LastTranDate;
```

Add a new check constraint

```
ALTER TABLE Invoices WITH NOCHECK
ADD CHECK (InvoiceTotal >= 1);
```

Add a foreign key constraint

```
ALTER TABLE InvoiceLineItems WITH CHECK
ADD FOREIGN KEY (AccountNo) REFERENCES
GLAccounts(AccountNo);
```

Change the data type of a column

```
ALTER TABLE InvoiceLineItems
ALTER COLUMN InvoiceLineItemDescription VARCHAR(200);
```

The syntax of the CREATE SEQUENCE statement

```
CREATE SEQUENCE sequence_name
    [AS integer_type]
    [START WITH starting_integer]
    [INCREMENT BY increment_integer]
    [{MINVALUE minimum_integer | NO MINVALUE}]
    [{MAXVALUE maximum_integer | NO MAXVALUE}]
    [{CYCLE|NOCYCLE}]
    [{CACHE cache_size|NOCACHE}]
```

Create a sequence that starts with 1

```
CREATE SEQUENCE TestSequence1
    START WITH 1;
```

Specify a starting value and an increment

```
CREATE SEQUENCE TestSequence2
    START WITH 10
    INCREMENT BY 10;
```

Specify all optional parameters

```
CREATE SEQUENCE TestSequence3
    AS int
    START WITH 100 INCREMENT BY 10
    MINVALUE 0 MAXVALUE 1000000
    CYCLE CACHE 10;
```

Create a table with a sequence column

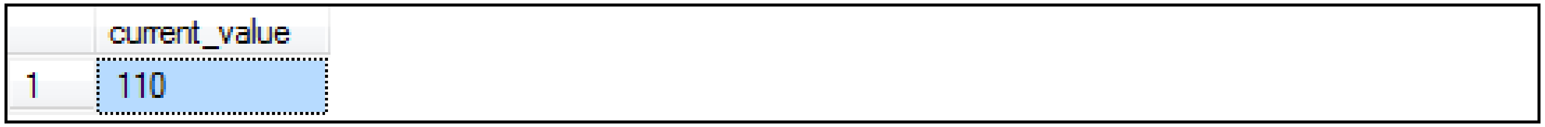
```
CREATE TABLE SequenceTable(
    SequenceNo      INT,
    Description     VARCHAR(50));
```

Insert the next value for a sequence

```
INSERT INTO SequenceTable
VALUES (NEXT VALUE FOR TestSequence3, 'First inserted row')
INSERT INTO SequenceTable
VALUES (NEXT VALUE FOR TestSequence3,
       'Second inserted row');
```

Get the current value of the sequence

```
SELECT current_value FROM sys.sequences
WHERE name = 'TestSequence3';
```



	current_value
1	110

The syntax of the DROP SEQUENCE statement

```
DROP SEQUENCE sequence_name1[, sequence_name2]...
```

A statement that drops a sequence

```
DROP SEQUENCE TestSequence2;
```

The syntax of the ALTER SEQUENCE statement

```
ALTER SEQUENCE sequence_name
    [RESTART [WITH starting_integer]]
    [INCREMENT BY increment_integer]
    [{MINVALUE minimum_integer | NO MINVALUE}]
    [{MAXVALUE maximum_integer | NO MAXVALUE}]
    [{CYCLE|NOCYCLE}]
    [{CACHE cache_size|NOCACHE}]
```

A statement that alters a sequence

```
ALTER SEQUENCE TestSequence1
    INCREMENT BY 9
    MINVALUE 1 MAXVALUE 999999
    CACHE 9
    CYCLE;
```

The SQL script that creates the AP database

```
CREATE DATABASE AP;
GO

USE AP;
CREATE TABLE Terms
(TermsID                 INT            NOT NULL PRIMARY KEY,
TermsDescription         VARCHAR(50)    NOT NULL,
TermsDueDays             SMALLINT       NOT NULL);

CREATE TABLE GLAccounts
(AccountNo               INT            NOT NULL PRIMARY KEY,
AccountDescription       VARCHAR(50)    NOT NULL);
```

The SQL script (cont.)

```
CREATE TABLE Vendors
(VendorID               INT             NOT NULL IDENTITY
PRIMARY KEY,
VendorName              VARCHAR(50)     NOT NULL,
VendorAddress1          VARCHAR(50)     NULL,
VendorAddress2          VARCHAR(50)     SPARSE NULL,
VendorCity              VARCHAR(50)     NOT NULL,
VendorState             CHAR(2)         NOT NULL,
VendorZipCode           VARCHAR(20)     NOT NULL,
VendorPhone             VARCHAR(50)     NULL,
VendorContactLName      VARCHAR(50)     NULL,
VendorContactFName      VARCHAR(50)     NULL,
DefaultTermsID          INT             NOT NULL
                        REFERENCES Terms(TermsID),
DefaultAccountNo        INT             NOT NULL
                        REFERENCES GLAccounts(AccountNo));
```

The SQL script (cont.)

```
CREATE TABLE Invoices
(InvoiceID              INT            NOT NULL IDENTITY
PRIMARY KEY,
VendorID                INT            NOT NULL
                        REFERENCES Vendors(VendorID),
InvoiceNumber           VARCHAR(50)    NOT NULL,
InvoiceDate             SMALLDATETIME  NOT NULL,
InvoiceTotal            MONEY          NOT NULL,
PaymentTotal            MONEY          NOT NULL DEFAULT 0,
CreditTotal             MONEY          NOT NULL DEFAULT 0,
TermsID                 INT            NOT NULL
                        REFERENCES Terms(TermsID),
InvoiceDueDate          SMALLDATETIME  NOT NULL,
PaymentDate             SMALLDATETIME  NULL);
```

The SQL script (cont.)

```
CREATE TABLE InvoiceLineItems
(InvoiceID                  INT              NOT NULL
                            REFERENCES Invoices(InvoiceID),
InvoiceSequence             SMALLINT         NOT NULL,
AccountNo                   INT              NOT NULL
                            REFERENCES GLAccounts(AccountNo),
InvoiceLineItemAmount       MONEY            NOT NULL,
InvoiceLineItemDescription VARCHAR(100) NOT NULL,
PRIMARY KEY (InvoiceID, InvoiceSequence));
```

The SQL script (cont.)

```
CREATE INDEX IX_Invoices_VendorID
    ON Invoices (VendorID);
CREATE INDEX IX_Invoices_TermsID
    ON Invoices (TermsID);
CREATE INDEX IX_Vendors_TermsID
    ON Vendors (DefaultTermsID);
CREATE INDEX IX_Vendors_AccountNo
    ON Vendors (DefaultAccountNo);
CREATE INDEX IX_InvoiceLineItems_AccountNo
    ON InvoiceLineItems (AccountNo);
CREATE INDEX IX_VendorName
    ON Vendors (VendorName);
CREATE INDEX IX_InvoiceDate
    ON Invoices (InvoiceDate DESC);
```

Terms

- Script
- Batch