Chapter 3

How to retrieve data from a single table

Objectives

Applied

- Code and run SELECT statements that use any of the language elements presented in this chapter.

Knowledge

- Distinguish between the base table values and the calculated values in SELECT statements.
- Describe the use of a column alias.
- Describe the order of precedence and the use of parentheses for arithmetic expressions.
- Describe the use of the DISTINCT keyword and the TOP clause.
- Describe the use of comparison operators, logical operators, and parentheses in WHERE clauses.

Objectives (cont.)

Knowledge

- Describe the use of the IN, BETWEEN, and LIKE operators in WHERE clauses.
- Describe the use of the IS NULL clause in a WHERE clause.
- Describe the use of column names, aliases, calculated values, and column numbers in ORDER BY clauses.
- Describe the use of the OFFSET and FETCH clauses in ORDER BY clauses.

The simplified syntax of the SELECT statement

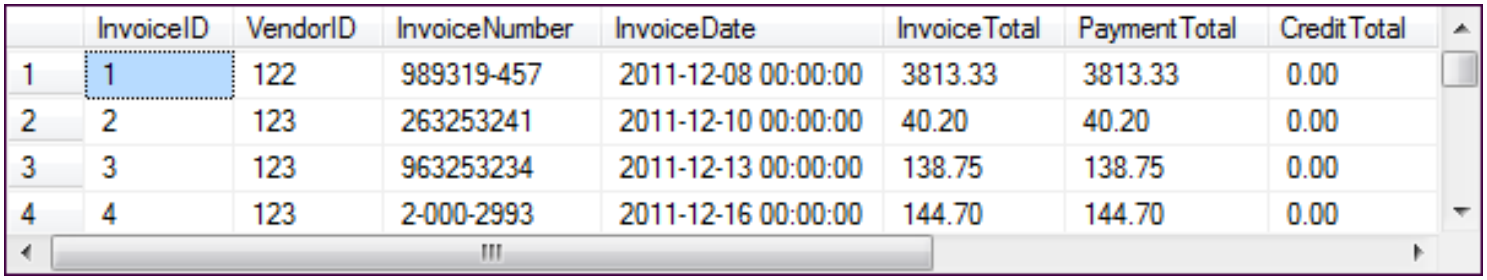
```
SELECT select_list
FROM table_source
[WHERE search_condition]
[ORDER BY order_by_list]
```

The four clauses of the SELECT statement

- SELECT
- FROM
- WHERE
- ORDER BY

A simple SELECT statement

```
SELECT *
FROM Invoices;
```



	InvoiceID	VendorID	InvoiceNumber	InvoiceDate	InvoiceTotal	PaymentTotal	CreditTotal
1	1	122	989319-457	2011-12-08 00:00:00	3813.33	3813.33	0.00
2	2	123	263253241	2011-12-10 00:00:00	40.20	40.20	0.00
3	3	123	963253234	2011-12-13 00:00:00	138.75	138.75	0.00
4	4	123	2-000-2993	2011-12-16 00:00:00	144.70	144.70	0.00

```
(114 rows)
```

A SELECT statement that retrieves and sorts rows

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
FROM Invoices
ORDER BY InvoiceTotal;
```

	InvoiceNumber	InvoiceDate	InvoiceTotal
1	25022117	2012-01-01 00:00:00	6.00
2	24863706	2012-01-10 00:00:00	6.00
3	24780512	2012-02-22 00:00:00	6.00
4	21-4923721	2012-01-13 00:00:00	9.95

```
(114 rows)
```

A SELECT statement that retrieves a calculated value

```
SELECT InvoiceID, InvoiceTotal, CreditTotal + PaymentTotal
AS TotalCredits
FROM Invoices
WHERE InvoiceID = 17;
```

	InvoiceID	InvoiceTotal	TotalCredits
1	17	10.00	10.00

A SELECT statement that retrieves all invoices between given dates

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
FROM Invoices
WHERE InvoiceDate BETWEEN '2012-01-01' AND '2012-05-31'
ORDER BY InvoiceDate;
```

	InvoiceNumber	InvoiceDate	InvoiceTotal
1	25022117	2012-01-01 00:00:00	6.00
2	P02-88D77S7	2012-01-03 00:00:00	856.92
3	21-4748363	2012-01-03 00:00:00	9.95
4	4-321-2596	2012-01-05 00:00:00	10.00

```
(101 rows)
```

A SELECT statement that returns an empty result set

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
FROM Invoices
WHERE InvoiceTotal > 50000;
```

	InvoiceNumber	InvoiceDate	InvoiceTotal

The expanded syntax of the SELECT clause

```
SELECT [ALL|DISTINCT] [TOP n [PERCENT] [WITH TIES]]
    column_specification [[AS] result_column]
    [, column_specification [[AS] result_column]] ...
```

Five ways to code column specifications

- All columns in a base table
- Column name in a base table
- Arithmetic expression
- String expression
- Function

Column specifications that use base table values

The * is used to retrieve all columns

```
SELECT *
```

Column names are used to retrieve specific columns

```
SELECT VendorName, VendorCity, VendorState
```

Column specifications that use calculated values

An arithmetic expression is used to calculate BalanceDue

```
SELECT InvoiceNumber,
    InvoiceTotal - PaymentTotal - CreditTotal
    AS BalanceDue
```

A string expression is used to calculate FullName

```
SELECT VendorContactFName + ' ' + VendorContactLName
    AS FullName
```

A function is used to calculate CurrentDate

```
SELECT InvoiceNumber, InvoiceDate,
    GETDATE() AS CurrentDate
```

Two ways to name the columns in a result set

Using the AS keyword (the preferred technique)

```
SELECT InvoiceNumber AS [Invoice Number],
       InvoiceDate AS Date, InvoiceTotal AS Total
FROM Invoices;
```

Using the equal operator (an older technique)

```
SELECT [Invoice Number] = InvoiceNumber, Date =
InvoiceDate,
    Total = InvoiceTotal
FROM Invoices;
```

The result set for both SELECT statements

	Invoice Number	Date	Total
1	989319-457	2011-12-08 00:00:00	3813.33
2	263253241	2011-12-10 00:00:00	40.20
3	963253234	2011-12-13 00:00:00	138.75
4	2-000-2993	2011-12-16 00:00:00	144.70
5	963253251	2011-12-16 00:00:00	15.50

A SELECT statement that doesn't name a calculated column

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
    InvoiceTotal - PaymentTotal - CreditTotal
FROM Invoices;
```

	InvoiceNumber	InvoiceDate	InvoiceTotal	(No column name)
1	989319-457	2011-12-08 00:00:00	3813.33	0.00
2	263253241	2011-12-10 00:00:00	40.20	0.00
3	963253234	2011-12-13 00:00:00	138.75	0.00
4	2-000-2993	2011-12-16 00:00:00	144.70	0.00
5	963253251	2011-12-16 00:00:00	15.50	0.00

How to concatenate string data

```
SELECT VendorCity, VendorState, VendorCity + VendorState
FROM Vendors;
```

	VendorCity	VendorState	(No column name)
1	Madison	WI	MadisonWI
2	Washington	DC	WashingtonDC
3	Washington	DC	WashingtonDC

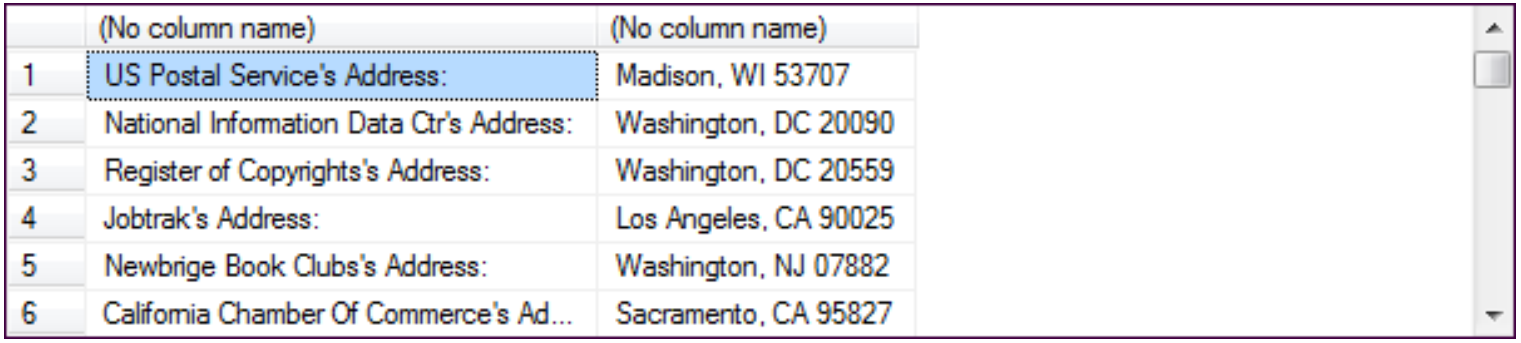
How to format string data using literal values

```
SELECT VendorName,
    VendorCity + ', ' + VendorState + ' ' + VendorZipCode
    AS Address
FROM Vendors;
```

	VendorName	Address
1	US Postal Service	Madison, WI 53707
2	National Information Data Ctr	Washington, DC 20090
3	Register of Copyrights	Washington, DC 20559
4	Jobtrak	Los Angeles, CA 90025

How to include apostrophes in literal values

```
SELECT VendorName + '''s Address: ',
    VendorCity + ', ' + VendorState + ' ' + VendorZipCode
FROM Vendors;
```



	(No column name)	(No column name)
1	US Postal Service's Address:	Madison, WI 53707
2	National Information Data Ctr's Address:	Washington, DC 20090
3	Register of Copyrights's Address:	Washington, DC 20559
4	Jobtrak's Address:	Los Angeles, CA 90025
5	Newbrige Book Clubs's Address:	Washington, NJ 07882
6	California Chamber Of Commerce's Ad...	Sacramento, CA 95827

The arithmetic operators in order of precedence

`*`	Multiplication
`/`	Division
`%`	Modulo (Remainder)
`+`	Addition
`-`	Subtraction

A SELECT statement that calculates the balance due

```
SELECT InvoiceTotal, PaymentTotal, CreditTotal,
    InvoiceTotal - PaymentTotal - CreditTotal AS BalanceDue
FROM Invoices;
```

	InvoiceTotal	PaymentTotal	CreditTotal	BalanceDue
1	3813.33	3813.33	0.00	0.00
2	40.20	40.20	0.00	0.00
3	138.75	138.75	0.00	0.00

A SELECT statement that uses parentheses to control the sequence of operations

```
SELECT InvoiceID,
    InvoiceID + 7 * 3 AS OrderOfPrecedence,
    (InvoiceID + 7) * 3 AS AddFirst
FROM Invoices
ORDER BY InvoiceID;
```

	InvoiceID	OrderOfPrecedence	AddFirst
1	1	22	24
2	2	23	27
3	3	24	30

What determines the sequence of operations

- Order of precedence
- Parentheses

A SELECT statement that uses the LEFT function

```
SELECT VendorContactFName, VendorContactLName,
    LEFT(VendorContactFName, 1) +
    LEFT(VendorContactLName, 1) AS Initials
FROM Vendors;
```

	VendorContactFName	VendorContactLName	Initials
1	Francesco	Alberto	FA
2	Ania	Irvin	AI
3	Lukas	Liana	LL

A SELECT statement that uses the CONVERT function

```
SELECT 'Invoice: #' + InvoiceNumber
    + ', dated ' + CONVERT(char(8), PaymentDate, 1)
    + ' for $' + CONVERT(varchar(9), PaymentTotal, 1)
FROM Invoices;
```

	(No column name)
1	Invoice: #989319-457, dated 01/07/12 for $3,813.33
2	Invoice: #263253241, dated 01/14/12 for $40.20

A SELECT statement that computes the age of an invoice

```
SELECT InvoiceDate,
    GETDATE() AS 'Today''s Date',
    DATEDIFF(day, InvoiceDate, GETDATE()) AS Age
FROM Invoices;
```

	InvoiceDate	Today's Date	Age
1	2012-04-02 00:00:00	2012-05-07 12:13:38.637	35
2	2012-04-01 00:00:00	2012-05-07 12:13:38.637	36
3	2012-03-31 00:00:00	2012-05-07 12:13:38.637	37

A SELECT statement that returns all rows

```
SELECT VendorCity, VendorState
FROM Vendors
ORDER BY VendorCity;
```

	VendorCity	VendorState
1	Anaheim	CA
2	Anaheim	CA
3	Ann Arbor	MI
4	Auburn Hills	MI
5	Boston	MA
6	Boston	MA
7	Boston	MA
8	Brea	CA

A SELECT statement that eliminates duplicate rows

```
SELECT DISTINCT VendorCity, VendorState
FROM Vendors;
```

	VendorCity	VendorState
1	Anaheim	CA
2	Ann Arbor	MI
3	Auburn Hills	MI
4	Boston	MA
5	Brea	CA
6	Carol Stream	IL
7	Charlotte	NC
8	Chicago	IL

```
(53 rows)
```

A SELECT statement with a TOP clause

```
SELECT TOP 5 VendorID, InvoiceTotal
FROM Invoices
ORDER BY InvoiceTotal DESC;
```

	VendorID	InvoiceTotal
1	110	37966.19
2	110	26881.40
3	110	23517.58
4	72	21842.00
5	110	20551.18

A SELECT statement with a TOP clause and the PERCENT keyword

```
SELECT TOP 5 PERCENT VendorID, InvoiceTotal
FROM Invoices
ORDER BY InvoiceTotal DESC;
```

	VendorID	InvoiceTotal
1	110	37966.19
2	110	26881.40
3	110	23517.58
4	72	21842.00
5	110	20551.18
6	110	10976.06

A SELECT statement with a TOP clause and the WITH TIES keyword

```
SELECT TOP 5 WITH TIES VendorID, InvoiceDate
FROM Invoices
ORDER BY InvoiceDate ASC;
```

	VendorID	InvoiceDate
1	122	2011-12-08 00:00:00
2	123	2011-12-10 00:00:00
3	123	2011-12-13 00:00:00
4	123	2011-12-16 00:00:00
5	123	2011-12-16 00:00:00
6	123	2011-12-16 00:00:00

The syntax of the WHERE clause with comparison operators

```
WHERE expression_1 operator expression_2
```

The comparison operators

- =
- >
- <
- <=
- >=
- <>

Examples of WHERE clauses that retrieve…

Vendors located in Iowa

```
WHERE VendorState = 'IA'
```

Invoices with a balance due (two variations)

```
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
```

```
WHERE InvoiceTotal > PaymentTotal + CreditTotal
```

Vendors with names from A to L

```
WHERE VendorName < 'M'
```

Invoices on or before a specified date

```
WHERE InvoiceDate <= '2012-05-31'
```

Invoices on or after a specified date

```
WHERE InvoiceDate >= '5/1/12'
```

Invoices with credits that don't equal zero

```
WHERE CreditTotal <> 0
```

The syntax of the WHERE clause with logical operators

```
WHERE [NOT] search_condition_1 {AND|OR}
      [NOT] search_condition_2 ...
```

Examples of queries using logical operators

The AND operator

```
WHERE VendorState = 'NJ' AND YTDPurchases > 200
```

The OR operator

```
WHERE VendorState = 'NJ' OR YTDPurchases > 200
```

The NOT operator

```
WHERE NOT (InvoiceTotal >= 5000 OR
      NOT InvoiceDate <= '2012-07-01')
```

The same condition without the NOT operator

```
WHERE InvoiceTotal < 5000 AND InvoiceDate <= '2012-07-01'
```

A compound condition without parentheses

```
WHERE InvoiceDate > '01/01/2012'
    OR InvoiceTotal > 500
    AND InvoiceTotal - PaymentTotal - CreditTotal > 0
```

	InvoiceNumber	InvoiceDate	InvoiceTotal	BalanceDue
1	P02-88D77S7	2012-01-03 00:00:00	856.92	0.00
2	21-4748363	2012-01-03 00:00:00	9.95	0.00
3	4-321-2596	2012-01-05 00:00:00	10.00	0.00
4	963253242	2012-01-06 00:00:00	104.00	0.00

```
(100 rows)
```

The order of precedence for compound conditions

- NOT
- AND
- OR

The same compound condition with parentheses

```
WHERE (InvoiceDate > '01/01/2012'
    OR InvoiceTotal > 500)
    AND InvoiceTotal - PaymentTotal - CreditTotal > 0
```

	InvoiceNumber	InvoiceDate	InvoiceTotal	BalanceDue
1	39104	2012-03-10 00:00:00	85.31	85.31
2	963253264	2012-03-18 00:00:00	52.25	52.25
3	31361833	2012-03-21 00:00:00	579.42	579.42
4	263253268	2012-03-21 00:00:00	59.97	59.97

```
(11 rows)
```

The syntax of the WHERE clause with an IN phrase

```
WHERE test_expression [NOT] IN
      ({subquery|expression_1 [, expression_2]...})
```

Examples of the IN phrase

An IN phrase with a list of numeric literals

```
WHERE TermsID IN (1, 3, 4)
```

An IN phrase preceded by NOT

```
WHERE VendorState NOT IN ('CA', 'NV', 'OR')
```

An IN phrase with a subquery

```
WHERE VendorID IN
    (SELECT VendorID
     FROM Invoices
     WHERE InvoiceDate = '2012-05-01')
```

The syntax of the WHERE clause with a BETWEEN phrase

```
WHERE test_expression [NOT] BETWEEN
      begin_expression AND end_expression
```

Examples of the BETWEEN phrase

A BETWEEN phrase with literal values

```
WHERE InvoiceDate BETWEEN '2012-05-01' AND '2012-05-31'
```

A BETWEEN phrase preceded by NOT

```
WHERE VendorZipCode NOT BETWEEN 93600 AND 93799
```

A BETWEEN phrase with a test expression coded as a calculated value

```
WHERE InvoiceTotal - PaymentTotal - CreditTotal
    BETWEEN 200 AND 500
```

A BETWEEN phrase with calculated values

```
WHERE InvoiceDueDate BETWEEN GetDate() AND GetDate() + 30
```

Warning about date comparisons

- All columns that have the datetime data type include both a date and time, and so does the value returned by the GetDate function.
- When you code a date literal without a time, the time defaults to 12:00 AM (midnight). As a result, a date comparison may not yield the results you expect.

The syntax of the WHERE clause with a LIKE phrase

```
WHERE match_expression [NOT] LIKE pattern
```

Wildcard symbols

- `%`
- `_`
- `[ ]`
- `[ - ]`
- `[ ^ ]`

WHERE clauses that use the LIKE phrase

Example 1

```
WHERE VendorCity LIKE 'SAN%'
```

Cities that will be retrieved

"San Diego" and "Santa Ana"

Example 2

```
WHERE VendorName LIKE 'COMPU_ER%'
```

Vendors that will be retrieved

"Compuserve" and "Computerworld"

Example 3

```
WHERE VendorContactLName LIKE 'DAMI[EO]N'
```

Names that will be retrieved

"Damien" and "Damion"

WHERE clauses that use the LIKE phrase (cont.)

Example 4

```
WHERE VendorState LIKE 'N[A-J]'
```

States that will be retrieved

"NC" and "NJ" but not "NV" or "NY"

Example 5

```
WHERE VendorState LIKE 'N[^K-Y]'
```

States that will be retrieved

"NC" and "NJ" but not "NV" or "NY"

Example 6

```
WHERE VendorZipCode NOT LIKE '[1-9]%'
```

Zip codes that will be retrieved

"02107" and "08816"

The syntax of the WHERE clause with the IS NULL clause

```
WHERE expression IS [NOT] NULL
```

The contents of the NullSample table

```
SELECT *
FROM NullSample;
```

	InvoiceID	InvoiceTotal
1	1	125.00
2	2	0.00
3	3	NULL
4	4	2199.99
5	5	0.00

A SELECT statement that retrieves rows with zero values

```
SELECT *
FROM NullSample
WHERE InvoiceTotal = 0;
```

	InvoiceID	InvoiceTotal
1	2	0.00
2	5	0.00

A SELECT statement that retrieves rows with non-zero values

```
SELECT *
FROM NullSample
WHERE InvoiceTotal <> 0;
```

	InvoiceID	InvoiceTotal
1	1	125.00
2	4	2199.99

A SELECT statement that retrieves rows with null values

```
SELECT *
FROM NullSample
WHERE InvoiceTotal IS NULL;
```

	InvoiceID	InvoiceTotal
1	3	NULL

A SELECT statement that retrieves rows without null values

```
SELECT *
FROM NullSample
WHERE InvoiceTotal IS NOT NULL;
```

	InvoiceID	InvoiceTotal
1	1	125.00
2	2	0.00
3	4	2199.99
4	5	0.00

The expanded syntax of the ORDER BY clause

```
ORDER BY expression [ASC|DESC] [, expression [ASC|DESC]]...
```

An ORDER BY clause that sorts by one column

```
SELECT VendorName,
    VendorCity + ', ' + VendorState + ' ' + VendorZipCode
    AS Address
FROM Vendors
ORDER BY VendorName;
```

	VendorName	Address
1	Abbey Office Furnishings	Fresno, CA 93722
2	American Booksellers Assoc	Tarrytown, NY 10591
3	American Express	Los Angeles, CA 90096

The default sequence for an ascending sort

- Nulls
- Special characters
- Numbers
- Letters

An ORDER BY clause that sorts by one column in descending sequence

```
SELECT VendorName,
    VendorCity + ', ' + VendorState + ' ' + VendorZipCode
    AS Address
FROM Vendors
ORDER BY VendorName DESC;
```

	VendorName	Address
1	Zylka Design	Fresno, CA 93711
2	Zip Print & Copy Center	Fresno, CA 93777
3	Zee Medical Service Co	Washington, IA 52353

An ORDER BY clause that sorts by three columns

```
SELECT VendorName,
    VendorCity + ', ' + VendorState + ' ' + VendorZipCode
    AS Address
FROM Vendors
ORDER BY VendorState, VendorCity, VendorName;
```

	VendorName	Address
1	AT&T	Phoenix, AZ 85062
2	Computer Library	Phoenix, AZ 85023
3	Wells Fargo Bank	Phoenix, AZ 85038
4	Aztek Label	Anaheim, CA 92807
5	Blue Shield of C...	Anaheim, CA 92850
6	Diversified Printi...	Brea, CA 92621
7	Abbey Office Fu...	Fresno, CA 93722
8	ASC Signs	Fresno, CA 93703
9	BFI Industries	Fresno, CA 93792

An ORDER BY clause that uses an alias

```
SELECT VendorName,
    VendorCity + ', ' + VendorState + ' ' + VendorZipCode
    AS Address
FROM Vendors
ORDER BY Address, VendorName;
```

	VendorName	Address
1	Aztek Label	Anaheim, CA 92807
2	Blue Shield of California	Anaheim, CA 92850
3	Malloy Lithographing Inc	Ann Arbor, MI 48106

An ORDER BY clause that uses an expression

```
SELECT VendorName,
    VendorCity + ', ' + VendorState + ' ' + VendorZipCode AS
Address
FROM Vendors
ORDER BY VendorContactLName + VendorContactFName;
```

	VendorName	Address
1	Dristas Groom & McCormick	Fresno, CA 93720
2	Internal Revenue Service	Fresno, CA 93888
3	US Postal Service	Madison, WI 53707

An ORDER BY clause that uses column positions

```
SELECT VendorName,
    VendorCity + ', ' + VendorState + ' ' + VendorZipCode
    AS Address
FROM Vendors
ORDER BY 2, 1;
```

	VendorName	Address
1	Aztek Label	Anaheim, CA 92807
2	Blue Shield of California	Anaheim, CA 92850
3	Malloy Lithographing Inc	Ann Arbor, MI 48106

The syntax of the ORDER BY clause for retrieving a range of rows

```
ORDER BY order_by_list
    OFFSET offset_row_count {ROW|ROWS}
   [FETCH {FIRST|NEXT} fetch_row_count {ROW|ROWS} ONLY]
```

An ORDER BY clause that retrieves the first five rows

```
SELECT VendorID, InvoiceTotal
FROM Invoices
ORDER BY InvoiceTotal DESC
    OFFSET 0 ROWS
    FETCH FIRST 5 ROWS ONLY;
```

	VendorID	InvoiceTotal
1	110	37966.19
2	110	26881.40
3	110	23517.58
4	72	21842.00
5	110	20551.18

An ORDER BY clause that retrieves rows 11 through 20

```
SELECT VendorName, VendorCity, VendorState, VendorZipCode
FROM Vendors
WHERE VendorState = 'CA'
ORDER BY VendorCity
    OFFSET 10 ROWS
    FETCH NEXT 10 ROWS ONLY;
```

	VendorName	VendorCity	VendorState	VendorZipCode
1	Robbins Mobile Lock And Key	Fresno	CA	93726
2	BFI Industries	Fresno	CA	93792
3	California Data Marketing	Fresno	CA	93721
4	Yale Industrial Trucks-Fresno	Fresno	CA	93706
5	Costco	Fresno	CA	93711
6	Graylift	Fresno	CA	93745
7	Shields Design	Fresno	CA	93728
8	Fresno County Tax Collector	Fresno	CA	93715
9	Gary McKeighan Insurance	Fresno	CA	93711
10	Ph Photographic Services	Fresno	CA	93726

Chapter 4

How to retrieve data from two or more tables

Objectives

Applied

- Use the explicit syntax to code an inner join that returns data from a single table or multiple tables.
- Code a union that combines data from a single table or multiple tables.

Knowledge

- Explain when column names need to be qualified.
- Describe the proper use of correlation names.
- Describe the differences between an inner join, a left outer join, a right outer join, a full outer join, and a cross join.
- Explain why you don't need to use right outer joins.
- Describe the use of the implicit syntax for coding inner joins.

Objectives (cont.)

- Describe the use of unions including the use of the EXCEPT and INTERSECT operators.

The explicit syntax for an inner join

```
SELECT select_list
FROM table_1
    [INNER] JOIN table_2
        ON join_condition_1
   [[INNER] JOIN table_3
        ON join_condition_2]...
```

An inner join of the Vendors and Invoices tables

```
SELECT InvoiceNumber, VendorName
FROM Vendors JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID;
```

The result set

	InvoiceNumber	VendorName
1	QP58872	IBM
2	Q545443	IBM
3	547481328	Blue Cross
4	547479217	Blue Cross
5	547480102	Blue Cross
6	P02-88D77S7	Fresno County Tax Collector
7	40318	Data Reproductions Corp

```
(114 rows)
```

The syntax for an inner join that uses correlation names

```
SELECT select_list
FROM table_1 [AS] n1
    [INNER] JOIN table_2 [AS] n2
        ON n1.column_name operator n2.column_name
   [[INNER] JOIN table_3 [AS] n3
        ON n2.column_name operator n3.column_name]...
```

Correlation names that make the query more difficult to read

```
SELECT InvoiceNumber, VendorName, InvoiceDueDate,
    InvoiceTotal - PaymentTotal - CreditTotal AS BalanceDue
FROM Vendors AS v JOIN Invoices AS i
    ON v.VendorID = i.VendorID
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
ORDER BY InvoiceDueDate DESC;
```

The result set

	InvoiceNumber	VendorName	InvoiceDueDate	BalanceDue
1	0-2436	Malloy Lithographing Inc	2012-04-30 00:00:00	10976.06
2	547480102	Blue Cross	2012-04-30 00:00:00	224.00
3	9982771	Ford Motor Credit Company	2012-04-23 00:00:00	503.20

```
(11 rows)
```

A correlation name that simplifies the query

```
SELECT InvoiceNumber, InvoiceLineItemAmount,
InvoiceLineItemDescription
FROM Invoices JOIN InvoiceLineItems AS LineItems
    ON Invoices.InvoiceID = LineItems.InvoiceID
WHERE AccountNo = 540
ORDER BY InvoiceDate;
```

The result set

	InvoiceNumber	InvoiceLineItemAmount	InvoiceLineItemDescription
1	I77271-001	478.00	Publishers Marketing
2	972110	207.78	Prospect list
3	133560	175.00	Card deck advertising

```
(6 rows)
```

The syntax of a fully-qualified object name

```
linked_server.database.schema.object
```

A join with fully-qualified table names

```
SELECT VendorName, CustLastName, CustFirstName,
    VendorState AS State, VendorCity AS City
FROM DBServer.AP.dbo.Vendors AS Vendors
    JOIN DBServer.ProductOrders.dbo.Customers AS Customers
    ON Vendors.VendorZipCode = Customers.CustZip
ORDER BY State, City;
```

The result set

	VendorName	CustLastName	CustFirstName	State	City
1	Wells Fargo Bank	Marissa	Kyle	AZ	Phoenix
2	Aztek Label	Irvin	Ania	CA	Anaheim
3	Gary McKeighan Insurance	Neftaly	Thalia	CA	Fresno
4	Gary McKeighan Insurance	Holbrooke	Rashad	CA	Fresno
5	Shields Design	Damien	Deborah	CA	Fresno

```
(37 rows)
```

The same join with partially-qualified table names

```
SELECT VendorName, CustLastName, CustFirstName,
    VendorState AS State, VendorCity AS City
FROM Vendors
    JOIN ProductOrders..Customers AS Customers
    ON Vendors.VendorZipCode = Customers.CustZip
ORDER BY State, City;
```

An inner join with two conditions

```
SELECT InvoiceNumber, InvoiceDate,
    InvoiceTotal, InvoiceLineItemAmount
FROM Invoices JOIN InvoiceLineItems AS LineItems
    ON (Invoices.InvoiceID = LineItems.InvoiceID) AND
       (Invoices.InvoiceTotal >
           LineItems.InvoiceLineItemAmount)
ORDER BY InvoiceNumber;
```

The result set

	InvoiceNumber	InvoiceDate	InvoiceTotal	InvoiceLineItemAmount
1	97/522	2012-02-28 00:00:00	1962.13	1197.00
2	97/522	2012-02-28 00:00:00	1962.13	765.13
3	I77271-001	2011-12-26 00:00:00	662.00	50.00
4	I77271-001	2011-12-26 00:00:00	662.00	75.60
5	I77271-001	2011-12-26 00:00:00	662.00	58.40
6	I77271-001	2011-12-26 00:00:00	662.00	478.00

The same join with the second condition coded in a WHERE clause

```
SELECT InvoiceNumber, InvoiceDate,
    InvoiceTotal, InvoiceLineItemAmount
FROM Invoices JOIN InvoiceLineItems AS LineItems
    ON Invoices.InvoiceID = LineItems.InvoiceID
WHERE Invoices.InvoiceTotal >
    LineItems.InvoiceLineItemAmount
ORDER BY InvoiceNumber;
```

The same result set

	InvoiceNumber	InvoiceDate	InvoiceTotal	InvoiceLineItemAmount
1	97/522	2012-02-28 00:00:00	1962.13	1197.00
2	97/522	2012-02-28 00:00:00	1962.13	765.13
3	I77271-001	2011-12-26 00:00:00	662.00	50.00
4	I77271-001	2011-12-26 00:00:00	662.00	75.60
5	I77271-001	2011-12-26 00:00:00	662.00	58.40
6	I77271-001	2011-12-26 00:00:00	662.00	478.00

A self-join that returns vendors from cities in common with other vendors

```
SELECT DISTINCT Vendors1.VendorName, Vendors1.VendorCity,
    Vendors1.VendorState
FROM Vendors AS Vendors1 JOIN Vendors AS Vendors2
    ON (Vendors1.VendorCity = Vendors2.VendorCity) AND
       (Vendors1.VendorState = Vendors2.VendorState) AND
       (Vendors1.VendorID <> Vendors2.VendorID)
ORDER BY Vendors1.VendorState, Vendors1.VendorCity;
```

The result set

	VendorName	VendorCity	VendorState
1	AT&T	Phoenix	AZ
2	Computer Library	Phoenix	AZ
3	Wells Fargo Bank	Phoenix	AZ
4	Aztek Label	Anaheim	CA
5	Blue Shield of California	Anaheim	CA
6	Abbey Office Furnishings	Fresno	CA
7	ASC Signs	Fresno	CA
8	BFI Industries	Fresno	CA

```
(84 rows)
```

A SELECT statement that joins four tables

```
SELECT VendorName, InvoiceNumber, InvoiceDate,
    InvoiceLineItemAmount AS LineItemAmount,
    AccountDescription
FROM Vendors
    JOIN Invoices ON Vendors.VendorID = Invoices.VendorID
    JOIN InvoiceLineItems
      ON Invoices.InvoiceID = InvoiceLineItems.InvoiceID
    JOIN GLAccounts
      ON InvoiceLineItems.AccountNo = GLAccounts.AccountNo
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
ORDER BY VendorName, LineItemAmount DESC;
```

The first interim table

	VendorName	InvoiceNumber	InvoiceDate
1	Blue Cross	547480102	2012-04-01 00:00:00
2	Cardinal Business Media, Inc.	134116	2012-03-28 00:00:00
3	Data Reproductions Corp	39104	2012-03-10 00:00:00
4	Federal Express Corporation	963253264	2012-03-18 00:00:00
5	Federal Express Corporation	263253268	2012-03-21 00:00:00
6	Federal Express Corporation	263253270	2012-03-22 00:00:00
7	Federal Express Corporation	263253273	2012-03-22 00:00:00

```
(11 rows)
```

The second interim table

	VendorName	InvoiceNumber	InvoiceDate	LineItemAmount
1	Blue Cross	547480102	2012-04-01 00:00:00	224.00
2	Cardinal Business Media, Inc.	134116	2012-03-28 00:00:00	90.36
3	Data Reproductions Corp	39104	2012-03-10 00:00:00	85.31
4	Federal Express Corporation	263253273	2012-03-22 00:00:00	30.75
5	Federal Express Corporation	963253264	2012-03-18 00:00:00	52.25
6	Federal Express Corporation	263253268	2012-03-21 00:00:00	59.97
7	Federal Express Corporation	263253270	2012-03-22 00:00:00	67.92

```
(11 rows)
```

The final result set

	VendorName	InvoiceNumber	InvoiceDate	LineItemAmount	AccountDescription
1	Blue Cross	547480102	2012-04-01 00:00:00	224.00	Group Insurance
2	Cardinal Business Media, Inc.	134116	2012-03-28 00:00:00	90.36	Direct Mail Advertising
3	Data Reproductions Corp	39104	2012-03-10 00:00:00	85.31	Book Printing Costs
4	Federal Express Corporation	263253270	2012-03-22 00:00:00	67.92	Freight
5	Federal Express Corporation	263253268	2012-03-21 00:00:00	59.97	Freight
6	Federal Express Corporation	963253264	2012-03-18 00:00:00	52.25	Freight
7	Federal Express Corporation	263253273	2012-03-22 00:00:00	30.75	Freight

```
(11 rows)
```

The implicit syntax for an inner join

```
SELECT select_list
FROM table_1, table_2 [, table_3]...
WHERE table_1.column_name operator table_2.column_name
 [AND table_2.column_name operator table_3.column_name]...
```

A join of the Vendors and Invoices tables

```
SELECT InvoiceNumber, VendorName
FROM Vendors, Invoices
WHERE Vendors.VendorID = Invoices.VendorID;
```

The result set

	InvoiceNumber	VendorName
1	QP58872	IBM
2	Q545443	IBM
3	547481328	Blue Cross
4	547479217	Blue Cross
5	547480102	Blue Cross
6	P02-88D77S7	Fresno County Tax Collector
7	40318	Data Reproductions Corp

A statement that joins four tables

```
SELECT VendorName, InvoiceNumber, InvoiceDate,
    InvoiceLineItemAmount AS LineItemAmount,
    AccountDescription
FROM Vendors, Invoices, InvoiceLineItems, GLAccounts
WHERE Vendors.VendorID = Invoices.VendorID
  AND Invoices.InvoiceID = InvoiceLineItems.InvoiceID
  AND InvoiceLineItems.AccountNo = GLAccounts.AccountNo
  AND InvoiceTotal - PaymentTotal - CreditTotal > 0
ORDER BY VendorName, LineItemAmount DESC;
```

The result set

	VendorName	InvoiceNumber	InvoiceDate	LineItemAmount	AccountDescription
1	Blue Cross	547480102	2012-04-01 00:00:00	224.00	Group Insurance
2	Cardinal Business Media, Inc.	134116	2012-03-28 00:00:00	90.36	Direct Mail Advertising
3	Data Reproductions Corp	39104	2012-03-10 00:00:00	85.31	Book Printing Costs
4	Federal Express Corporation	263253270	2012-03-22 00:00:00	67.92	Freight
5	Federal Express Corporation	263253268	2012-03-21 00:00:00	59.97	Freight
6	Federal Express Corporation	963253264	2012-03-18 00:00:00	52.25	Freight
7	Federal Express Corporation	263253273	2012-03-22 00:00:00	30.75	Freight

The explicit syntax for an outer join

```
SELECT select_list
FROM table_1
    {LEFT|RIGHT|FULL} [OUTER] JOIN table_2
        ON join_condition_1
   [{LEFT|RIGHT|FULL} [OUTER] JOIN table_3
        ON join_condition_2]...
```

What outer joins do

Joins of this type	Keep unmatched rows from
Left outer join	The first (left) table
Right outer join	The second (right) table
Full outer join	Both tables

A SELECT statement that uses a left outer join

```
SELECT VendorName, InvoiceNumber, InvoiceTotal
FROM Vendors LEFT JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
ORDER BY VendorName;
```

	VendorName	InvoiceNumber	InvoiceTotal
1	Abbey Office Furnishings	203339-13	17.50
2	American Booksellers Assoc	NULL	NULL
3	American Express	NULL	NULL
4	ASC Signs	NULL	NULL
5	Ascom Hasler Mailing Systems	NULL	NULL
6	AT&T	NULL	NULL

```
(202 rows)
```

The Departments table

	DeptName	DeptNo
1	Accounting	1
2	Payroll	2
3	Operations	3
4	Personnel	4
5	Maintenance	5

The Employees table

	EmployeeID	LastName	FirstName	DeptNo
1	1	Smith	Cindy	2
2	2	Jones	Elmer	4
3	3	Simonian	Ralph	2
4	4	Hernandez	Olivia	1
5	5	Aaronsen	Robert	2
6	6	Watson	Denise	6
7	7	Hardy	Thomas	5
8	8	O'Leary	Rhea	4
9	9	Locario	Paulo	6

The Projects table

	ProjectNo	EmployeeID
1	P1011	8
2	P1011	4
3	P1012	3
4	P1012	1
5	P1012	5
6	P1013	6
7	P1013	9
8	P1014	10

A left outer join

```
SELECT DeptName, Departments.DeptNo, LastName
FROM Departments LEFT JOIN Employees
    ON Departments.DeptNo = Employees.DeptNo;
```

The result set

	DeptName	DeptNo	LastName
1	Accounting	1	Hernandez
2	Payroll	2	Smith
3	Payroll	2	Simonian
4	Payroll	2	Aaronsen
5	Operations	3	NULL
6	Personnel	4	Jones
7	Personnel	4	O'Leary
8	Maintenance	5	Hardy

A right outer join

```
SELECT DeptName, Employees.DeptNo, LastName
FROM Departments RIGHT JOIN Employees
    ON Departments.DeptNo = Employees.DeptNo;
```

The result set

	DeptName	DeptNo	LastName
1	Payroll	2	Smith
2	Personnel	4	Jones
3	Payroll	2	Simonian
4	Accounting	1	Hernandez
5	Payroll	2	Aaronsen
6	NULL	6	Watson
7	Maintenance	5	Hardy
8	Personnel	4	O'Leary
9	NULL	6	Locario

A full outer join

```
SELECT DeptName, Departments.DeptNo,
    Employees.DeptNo, LastName
FROM Departments FULL JOIN Employees
       ON Departments.DeptNo = Employees.DeptNo;
```

The result set

	DeptName	DeptNo	DeptNo	LastName
1	Accounting	1	1	Hernandez
2	Payroll	2	2	Smith
3	Payroll	2	2	Simonian
4	Payroll	2	2	Aaronsen
5	Operations	3	NULL	NULL
6	Personnel	4	4	Jones
7	Personnel	4	4	O'Leary
8	Maintenance	5	5	Hardy
9	NULL	NULL	6	Watson
10	NULL	NULL	6	Locario

Join three tables using left outer joins

```
SELECT DeptName, LastName, ProjectNo
FROM Departments
    LEFT JOIN Employees
        ON Departments.DeptNo = Employees.DeptNo
    LEFT JOIN Projects
        ON Employees.EmployeeID = Projects.EmployeeID
ORDER BY DeptName, LastName, ProjectNo;
```

The result set

	DeptName	LastName	ProjectNo
1	Accounting	Hernandez	P1011
2	Maintenance	Hardy	NULL
3	Operations	NULL	NULL
4	Payroll	Aaronsen	P1012
5	Payroll	Simonian	P1012
6	Payroll	Smith	P1012
7	Personnel	Jones	NULL
8	Personnel	O'Leary	P1011

Join three tables using full outer joins

```
SELECT DeptName, LastName, ProjectNo
FROM Departments
    FULL JOIN Employees
        ON Departments.DeptNo = Employees.DeptNo
    FULL JOIN Projects
        ON Employees.EmployeeID = Projects.EmployeeID
ORDER BY DeptName;
```

The result set

	DeptName	LastName	ProjectNo
1	NULL	Watson	P1013
2	NULL	Locario	P1013
3	NULL	NULL	P1014
4	Accounting	Hernandez	P1011
5	Maintenance	Hardy	NULL
6	Operations	NULL	NULL
7	Payroll	Smith	P1012
8	Payroll	Simonian	P1012
9	Payroll	Aaronsen	P1012
10	Personnel	Jones	NULL
11	Personnel	O'Leary	P1011

Combine an outer and an inner join

```
SELECT DeptName, LastName, ProjectNo
FROM Departments
    JOIN Employees
        ON Departments.DeptNo = Employees.DeptNo
    LEFT JOIN Projects
        ON Employees.EmployeeID = Projects.EmployeeID
ORDER BY DeptName;
```

The interim table

	DeptName	LastName	EmployeeID
1	Payroll	Smith	1
2	Personnel	Jones	2
3	Payroll	Simonian	3
4	Accounting	Hernandez	4
5	Payroll	Aaronsen	5
6	Maintenance	Hardy	7
7	Personnel	O'Leary	8

The result set

	DeptName	LastName	ProjectNo
1	Accounting	Hernandez	P1011
2	Maintenance	Hardy	NULL
3	Payroll	Smith	P1012
4	Payroll	Simonian	P1012
5	Payroll	Aaronsen	P1012
6	Personnel	Jones	NULL
7	Personnel	O'Leary	P1011

How to code a cross join using the explicit syntax

The explicit syntax for a cross join

```
SELECT select_list
FROM table_1 CROSS JOIN table_2
```

A cross join that uses the explicit syntax

```
SELECT Departments.DeptNo, DeptName,
    EmployeeID, LastName
FROM Departments CROSS JOIN Employees
ORDER BY Departments.DeptNo;
```

The result set created by the cross joins

	DeptNo	DeptName	EmployeeID	LastName
1	1	Accounting	1	Smith
2	1	Accounting	2	Jones
3	1	Accounting	3	Simonian
4	1	Accounting	4	Hernandez
5	1	Accounting	5	Aaronsen
6	1	Accounting	6	Watson
7	1	Accounting	7	Hardy

```
(45 rows)
```

Terms

- Outer join
- Left outer join
- Right outer join
- Cross join
- Cartesian product

The syntax for a union operation

```
    SELECT_statement_1
UNION [ALL]
    SELECT_statement_2
[UNION [ALL]
    SELECT_statement_3]...
[ORDER BY order_by_list]
```

Rules for unions

- Each result set must return the same number of columns.
- The corresponding columns in each result set must have compatible data types.
- The column names in the final result set are taken from the first SELECT clause.

A union that combines data from two different tables

```
    SELECT 'Active' AS Source, InvoiceNumber,
        InvoiceDate, InvoiceTotal
    FROM ActiveInvoices
    WHERE InvoiceDate >= '02/01/2012'
UNION
    SELECT 'Paid' AS Source, InvoiceNumber,
        InvoiceDate, InvoiceTotal
    FROM PaidInvoices
    WHERE InvoiceDate >= '02/01/2012'
ORDER BY InvoiceTotal DESC;
```

	Source	InvoiceNumber	InvoiceDate	InvoiceTotal
1	Paid	P-0259	2012-03-19 00:00:00	26881.40
2	Paid	0-2060	2012-03-24 00:00:00	23517.58
3	Paid	40318	2012-02-01 00:00:00	21842.00
4	Active	P-0608	2012-03-23 00:00:00	20551.18
5	Active	0-2436	2012-03-31 00:00:00	10976.06
6	Paid	509786	2012-02-18 00:00:00	6940.25
7	Paid	989319-447	2012-03-24 00:00:00	3689.99
8	Paid	989319-437	2012-02-01 00:00:00	2765.36
9	Paid	367447	2012-02-11 00:00:00	2433.00

```
(72 rows)
```

A union that combines data from the same table

```
    SELECT 'Active' AS Source, InvoiceNumber,
        InvoiceDate, InvoiceTotal
    FROM Invoices
    WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
UNION
    SELECT 'Paid' AS Source, InvoiceNumber,
        InvoiceDate, InvoiceTotal
    FROM Invoices
    WHERE InvoiceTotal - PaymentTotal - CreditTotal <= 0
ORDER BY InvoiceTotal DESC;
```

The result set

	Source	InvoiceNumber	InvoiceDate	InvoiceTotal
1	Paid	0-2058	2012-01-28 00:00:00	37966.19
2	Paid	P-0259	2012-03-19 00:00:00	26881.40
3	Paid	0-2060	2012-03-24 00:00:00	23517.58
4	Paid	40318	2012-02-01 00:00:00	21842.00
5	Active	P-0608	2012-03-23 00:00:00	20551.18

```
(114 rows)
```

A union that combines payment data from the same joined tables

```
    SELECT InvoiceNumber, VendorName,
        '33% Payment' AS PaymentType,
        InvoiceTotal AS Total,
       (InvoiceTotal * 0.333) AS Payment
    FROM Invoices JOIN Vendors
        ON Invoices.VendorID = Vendors.VendorID
    WHERE InvoiceTotal > 10000
UNION
    SELECT InvoiceNumber, VendorName,
        '50% Payment' AS PaymentType,
        InvoiceTotal AS Total,
       (InvoiceTotal * 0.5) AS Payment
    FROM Invoices JOIN Vendors
        ON Invoices.VendorID = Vendors.VendorID
    WHERE InvoiceTotal BETWEEN 500 AND 10000
```

A union that combines payment data from the same joined tables (continued)

```
UNION
    SELECT InvoiceNumber, VendorName,
        'Full amount' AS PaymentType,
        InvoiceTotal AS Total,
        InvoiceTotal AS Payment
    FROM Invoices JOIN Vendors
        ON Invoices.VendorID = Vendors.VendorID
    WHERE InvoiceTotal < 500
ORDER BY PaymentType, VendorName, InvoiceNumber;
```

The result set

	InvoiceNumber	VendorName	PaymentType	Total	Payment
6	P-0608	Malloy Lithographing Inc	33% Payment	20551.18	6843.5429400
7	509786	Bertelsmann Industry S...	50% Payment	6940.25	3470.1250000
8	587056	Cahners Publishing Co...	50% Payment	2184.50	1092.2500000
9	367447	Computerworld	50% Payment	2433.00	1216.5000000

```
(114 rows)
```

The syntax for the EXCEPT and INTERSECT operators

```
    SELECT_statement_1
{EXCEPT|INTERSECT}
    SELECT_statement_2
[ORDER BY order_by_list]
```

The Customers table

	CustomerFirst	CustomerLast
1	Maria	Anders
2	Ana	Trujillo
3	Antonio	Moreno
4	Thomas	Hardy
5	Christina	Berglund
6	Hanna	Moos

```
(24 rows)
```

The Employees table

	FirstName	LastName
4	Olivia	Hernandez
5	Robert	Aaronsen
6	Denise	Watson
7	Thomas	Hardy
8	Rhea	O'Leary
9	Paulo	Locario

```
(9 rows)
```

Exclude rows from the first query if they also occur in the second query

```
    SELECT CustomerFirst, CustomerLast
    FROM Customers
EXCEPT
    SELECT FirstName, LastName
    FROM Employees
ORDER BY CustomerLast;
```

The result set

	CustomerFirst	CustomerLast
4	Donna	Chelan
5	Fred	Citeaux
6	Karl	Jablonski
7	Yoshi	Latimer

```
(23 rows)
```

Only include rows that occur in both queries

```
    SELECT CustomerFirst, CustomerLast
    FROM Customers
INTERSECT
    SELECT FirstName, LastName
    FROM Employees;
```

The result set

	CustomerFirst	CustomerLast
1	Thomas	Hardy

```
(1 row)
```

Terms

- Union
- Set operators