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 SELECT statement that joins the 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;
```

	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

A SELECT statement that joins the 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;
```

	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

Figure 4-10 Outer joins that join more than two tables

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 SELECT statement that combines 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

Description

- You can combine inner and outer joins within a single SELECT statement using the explicit join syntax. You can't combine inner and outer joins using the implicit syntax.

Figure 4-11 How to combine inner and outer joins

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;
```

How to code a cross join using the implicit syntax

The implicit syntax for a cross join

```
SELECT select_list
FROM table_1, table_2
```

A cross join that uses the implicit syntax

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

The result set created by the statements above

	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)
```

Description

- A *cross join* joins each row from the first table with each row from the second table. The result set returned by a cross join is known as a *Cartesian product*.
- To code a cross join using the explicit syntax, use the CROSS JOIN keywords in the FROM clause.
- To code a cross join using the implicit syntax, list the tables in the FROM clause and omit the join condition from the WHERE clause.

Figure 4-12 How to use cross joins

The syntax for a union operation

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

A union that combines invoice data from two different tables

```
    SELECT 'Active' AS Source, InvoiceNumber, InvoiceDate, InvoiceTotal
    FROM ActiveInvoices
    WHERE InvoiceDate >= '01/01/2020'
UNION
    SELECT 'Paid' AS Source, InvoiceNumber, InvoiceDate, InvoiceTotal
    FROM PaidInvoices
    WHERE InvoiceDate >= '01/01/2020'
ORDER BY InvoiceTotal DESC;
```

The result set

	Source	InvoiceNumber	InvoiceDate	InvoiceTotal
1	Paid	P-0259	2020-01-19 00:00:00	26881.40
2	Paid	0-2060	2020-01-24 00:00:00	23517.58
3	Active	P-0608	2020-01-23 00:00:00	20551.18
4	Active	0-2436	2020-01-31 00:00:00	10976.06
5	Paid	989319-447	2020-01-24 00:00:00	3689.99
6	Paid	989319-467	2020-01-01 00:00:00	2318.03
7	Paid	989319-417	2020-01-23 00:00:00	2051.59
8	Paid	97/222	2020-01-25 00:00:00	1000.46
9	Paid	963253230	2020-01-07 00:00:00	739.20

```
(35 rows)
```

Description

- A *union* combines the result sets of two or more SELECT statements into one result set.
- Each result set must return the same number of columns, and the corresponding columns in each result set must have compatible data types.
- By default, a union eliminates duplicate rows. If you want to include duplicate rows, code the ALL keyword.
- The column names in the final result set are taken from the first SELECT clause. Column aliases assigned by the other SELECT clauses have no effect on the final result set.
- To sort the rows in the final result set, code an ORDER BY clause after the last SELECT statement. This clause must refer to the column names assigned in the first SELECT clause.

Figure 4-13 How to combine data from different tables

union that combines information from the Invoices 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	2019-11-28	37966.19
2	Paid	P-0259	2020-01-19	26881.40
3	Paid	0-2060	2020-01-24	23517.58
4	Paid	40318	2019-12-01	21842.00
5	Active	P-0608	2020-01-23	20551.18

```
(114 rows)
```

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
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 Svcs. Inc	50% Payment	6940.25	3470.1250000
8	587056	Cahners Publishing Company	50% Payment	2184.50	1092.2500000
9	367447	Computerworld	50% Payment	2433.00	1216.5000000

```
(114 rows)
```

gure 4-14 Unions that combine data from the same table

The syntax for the EXCEPT and INTERSECT operations

```
    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)

A query that excludes 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)

A query that only includes 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)

Description

- The number of columns must be the same in both SELECT statements.
- The data types for each column must be compatible.
- The column names in the final result set are taken from the first SELECT statement.

Figure 4-15 How to use the EXCEPT and INTERSECT operators

Exercises

Unless otherwise stated, use the explicit join syntax.

1. Write a SELECT statement that returns all columns from the Vendors table inner-joined with the Invoices table.

2. Write a SELECT statement that returns four columns:

VendorName	From the Vendors table
InvoiceNumber	From the Invoices table
InvoiceDate	From the Invoices table
Balance	InvoiceTotal minus the sum of PaymentTotal and CreditTotal

   The result set should have one row for each invoice with a non-zero balance. Sort the result set by VendorName in ascending order.

3. Write a SELECT statement that returns three columns:

VendorName	From the Vendors table
DefaultAccountNo	From the Vendors table
AccountDescription	From the GLAccounts table

   The result set should have one row for each vendor, with the account number and account description for that vendor's default account number. Sort the result set by AccountDescription, then by VendorName.

5. Write a SELECT statement that returns five columns from three tables, all using column aliases:

Vendor	VendorName column
Date	InvoiceDate column
Number	InvoiceNumber column
#	InvoiceSequence column
LineItem	InvoiceLineItemAmount column

Assign the following correlation names to the tables:

v	Vendors table
i	Invoices table
li	InvoiceLineItems table

Sort the final result set by Vendor, Date, Number, and #.

6. Write a SELECT statement that returns three columns:

VendorID	From the Vendors table
VendorName	From the Vendors table
Name	A concatenation of VendorContactFName and VendorContactLName, with a space in between

The result set should have one row for each vendor whose contact has the same first name as another vendor's contact. Sort the final result set by Name.

Hint: Use a self-join.

7. Write a SELECT statement that returns two columns from the GLAccounts table: AccountNo and AccountDescription. The result set should have one row for each account number that has never been used. Sort the final result set by AccountNo.

Hint: Use an outer join to the InvoiceLineItems table.

8. Use the UNION operator to generate a result set consisting of two columns from the Vendors table: VendorName and VendorState. If the vendor is in California, the VendorState value should be "CA"; otherwise, the VendorState value should be "Outside CA." Sort the final result set by VendorName.

scalar function
aggregate function
column function
summary query
scalar aggregate
vector aggregate
cumulative total
moving average

Exercises

1. Write a SELECT statement that returns two columns from the Invoices table: VendorID and PaymentSum, where PaymentSum is the sum of the PaymentTotal column. Group the result set by VendorID.

2. Write a SELECT statement that returns two columns: VendorName and PaymentSum, where PaymentSum is the sum of the PaymentTotal column. Group the result set by VendorName. Return only 10 rows, corresponding to the 10 vendors who've been paid the most.

   Hint: Use the TOP clause and join Vendors to Invoices.

3. Write a SELECT statement that returns three columns: VendorName, InvoiceCount, and InvoiceSum. InvoiceCount is the count of the number of invoices, and InvoiceSum is the sum of the InvoiceTotal column. Group the result set by vendor. Sort the result set so the vendor with the highest number of invoices appears first.

4. Write a SELECT statement that returns three columns: AccountDescription, LineItemCount, and LineItemSum. LineItemCount is the number of entries in the InvoiceLineItems table that have that AccountNo. LineItemSum is the sum of the InvoiceLineItemAmount column for that AccountNo. Filter the result set to include only those rows with LineItemCount greater than 1. Group the result set by account description, and sort it by descending LineItemCount.

   Hint: Join the GLAccounts table to the InvoiceLineItems table.

5. Modify the solution to exercise 4 to filter for invoices dated from October 1, 2019 to December 31, 2019.

   Hint: Join to the Invoices table to code a search condition based on InvoiceDate.

6. Write a SELECT statement that answers the following question: What is the total amount invoiced for each AccountNo? Use the ROLLUP operator to include a row that gives the grand total.

   Hint: Use the InvoiceLineItemAmount column of the InvoiceLineItems table.

7. Write a SELECT statement that returns four columns: VendorName, AccountDescription, LineItemCount, and LineItemSum. LineItemCount is the row count, and LineItemSum is the sum of the InvoiceLineItemAmount column. For each vendor and account, return the number and sum of line items, sorted first by vendor, then by account description.

   Hint: Use a four-table join.

8. Write a SELECT statement that answers this question: Which vendors are being paid from more than one account? Return two columns: the vendor name and the total number of accounts that apply to that vendor's invoices.

   Hint: Use the DISTINCT keyword to count InvoiceLineItems.AccountNo.

9. Write a SELECT statement that returns six columns:

VendorID	From the Invoices table
InvoiceDate	From the Invoices table
InvoiceTotal	From the Invoices table
VendorTotal	The sum of the invoice totals for each vendor
VendorCount	The count of invoices for each vendor
VendorAvg	The average of the invoice totals for each vendor

   The result set should include the individual invoices for each vendor.