Chapter 6

How to code subqueries

Objectives

Applied

- Code SELECT statements that require subqueries.
- Code SELECT statements that use common table expressions (CTEs) to define the subqueries.

Knowledge

- Describe the way subqueries can be used in the WHERE, HAVING, FROM and SELECT clauses of a SELECT statement.
- Describe the difference between a correlated subquery and a noncorrelated subquery.
- Describe the use of common table expressions (CTEs).

Four ways to introduce a subquery in a SELECT statement

1. In a WHERE clause as a search condition
2. In a HAVING clause as a search condition
3. In the FROM clause as a table specification
4. In the SELECT clause as a column specification

A subquery in the WHERE clause

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
FROM Invoices
WHERE InvoiceTotal >
     (SELECT AVG(InvoiceTotal)
      FROM Invoices)
ORDER BY InvoiceTotal;
```

The value returned by the subquery

```
1879.7413
```

The result set (21 rows)

	InvoiceNumber	InvoiceDate	InvoiceTotal
1	989319-487	2019-12-20	1927.54
2	97/522	2019-12-28	1962.13
3	989319-417	2020-01-23	2051.59
4	989319-427	2019-12-16	2115.81
5	989319-477	2019-12-08	2184.11

Where a subquery can be introduced

If a subquery returns…	It can be introduced…
A single value	Anywhere an expression is allowed
A result set with a single column	In place of a list of values
A result set with one or more columns	In place of a table in the FROM clause

A query that uses an inner join

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
FROM Invoices JOIN Vendors
    ON Invoices.VendorID = Vendors.VendorID
WHERE VendorState = 'CA'
ORDER BY InvoiceDate;
```

The result set (40 rows)

	InvoiceNumber	InvoiceDate	InvoiceTotal
1	125520-1	2019-10-24	95.00
2	97/488	2019-10-24	601.95
3	111-92R-10096	2019-10-30	16.33
4	25022117	2019-11-01	6.00

The same query restated with a subquery

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
FROM Invoices
WHERE VendorID IN
     (SELECT VendorID
     FROM Vendors
     WHERE VendorState = 'CA')
ORDER BY InvoiceDate;
```

The same result set (40 rows)

	InvoiceNumber	InvoiceDate	InvoiceTotal
1	125520-1	2019-10-24	95.00
2	97/488	2019-10-24	601.95
3	111-92R-10096	2019-10-30	16.33
4	25022117	2019-11-01	6.00

Advantages of joins

- The result of a join operation can include columns from both tables.
- A join tends to be more intuitive when it uses an existing relationship between two tables.
- A query with a join typically performs faster than the same query with a subquery.

Advantages of subqueries

- A subquery can pass an aggregate value to the outer query.
- A subquery tends to be more intuitive when it uses an ad hoc relationship between two tables.
- Long, complex queries can sometimes be easier to code using subqueries.

The syntax of a WHERE clause that uses an IN phrase with a subquery

```
WHERE test_expression [NOT] IN (subquery)
```

A query that returns vendors without invoices

```
SELECT VendorID, VendorName, VendorState
FROM Vendors
WHERE VendorID NOT IN
    (SELECT DISTINCT VendorID
    FROM Invoices);
```

The result of the subquery: 34 VendorIDs

	VendorID
1	34
2	37
3	48
4	72
5	80
6	81

The result set (88 rows)

	VendorID	VendorName	VendorState
32	33	Nielson	OH
33	35	Cal State Termite	CA
34	36	Graylift	CA
35	38	Venture Communications Int1	NY
36	39	Custom Printing Company	MO
37	40	Nat Assoc of College Stores	OH

The query restated without a subquery

```
SELECT Vendors.VendorID, VendorName, VendorState
FROM Vendors LEFT JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
WHERE Invoices.VendorID IS NULL;
```

The same result set (88 rows)

	VendorID	VendorName	VendorState
32	33	Nielson	OH
33	35	Cal State Termite	CA
34	36	Graylift	CA
35	38	Venture Communications Int1	NY
36	39	Custom Printing Company	MO
37	40	Nat Assoc of College Stores	OH

The syntax of a WHERE clause that uses a comparison operator

```
WHERE expression comparison_operator [SOME|ANY|ALL]
    (subquery)
```

A query with a subquery in the WHERE condition

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
    InvoiceTotal - PaymentTotal - CreditTotal AS BalanceDue
FROM Invoices
WHERE InvoiceTotal - PaymentTotal - CreditTotal  > 0
    AND InvoiceTotal - PaymentTotal - CreditTotal <
    (SELECT AVG(InvoiceTotal - PaymentTotal - CreditTotal)
    FROM Invoices
    WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0)
ORDER BY InvoiceTotal DESC;
```

The value returned by the subquery

```
2910.9472
```

The result set (9 rows)

	InvoiceNumber	InvoiceDate	InvoiceTotal	BalanceDue
1	31361833	2020-01-21	579.42	579.42
2	9982771	2020-01-24	503.20	503.20
3	547480102	2020-02-01	224.00	224.00
4	134116	2020-01-28	90.36	90.36
5	39104	2020-01-10	85.31	85.31

How the ALL keyword works

Condition	Equivalent expression
`x > ALL (1, 2)`	`x > 2`
`x < ALL (1, 2)`	`x < 1`
`x = ALL (1, 2)`	`(x = 1) AND (x = 2)`
`x <> ALL (1, 2)`	`(x <> 1) AND (x <> 2)`

A query that uses the ALL keyword

```
SELECT VendorName, InvoiceNumber, InvoiceTotal
FROM Invoices JOIN Vendors
  ON Invoices.VendorID = Vendors.VendorID
WHERE InvoiceTotal > ALL
    (SELECT InvoiceTotal
    FROM Invoices
    WHERE VendorID = 34)
ORDER BY VendorName;
```

The result of the subquery using the ALL keyword

	InvoiceTotal
1	116.54
2	1083.58

The result set (25 rows)

	VendorName	InvoiceNumber	InvoiceTotal
1	Bertelsmann Industry Svcs. Inc	509786	6940.25
2	Cahners Publishing Company	587056	2184.50
3	Computerworld	367447	2433.00
4	Data Reproductions Corp	40318	21842.00
5	Dean Witter Reynolds	75C-90227	1367.50

How the ANY and SOME keywords work

Condition	Equivalent expression
`x > ANY (1, 2)`	`x > 1`
`x < ANY (1, 2)`	`x < 2`
`x = ANY (1, 2)`	`(x = 1) OR (x = 2)`
`x <> ANY (1, 2)`	`(x <> 1) OR (x <> 2)`

A query that uses the ANY keyword

```
SELECT VendorName, InvoiceNumber, InvoiceTotal
FROM Vendors
    JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
WHERE InvoiceTotal < ANY
    (SELECT InvoiceTotal
    FROM Invoices
    WHERE VendorID = 115);
```

The result of the subquery using the ANY keyword

	InvoiceTotal
1	6.00
2	6.00
3	25.67
4	6.00

The result set (17 rows)

	VendorName	InvoiceNumber	InvoiceTotal
1	Abbey Office Furnishings	203339-13	17.50
2	Pacific Bell	111-92R-10096	16.33
3	Pacific Bell	111-92R-10097	16.33
4	Pacific Bell	111-92R-10094	19.67
5	Compuserve	21-4923721	9.95

A query that uses a correlated subquery

```
SELECT VendorID, InvoiceNumber, InvoiceTotal
FROM Invoices AS Inv_Main
WHERE InvoiceTotal >
     (SELECT AVG(InvoiceTotal)
     FROM Invoices AS Inv_Sub
     WHERE Inv_Sub.VendorID = Inv_Main.VendorID)
ORDER BY VendorID, InvoiceTotal;
```

The value returned by the subquery for vendor 95

```
28.5016
```

The result set (36 rows)

	VendorID	InvoiceNumber	InvoiceTotal
6	83	31359783	1575.00
7	95	111-92R-10095	32.70
8	95	111-92R-10093	39.77
9	95	111-92R-10092	46.21
10	110	P-0259	26881.40

Terms to know for subqueries

- Subquery
- Subquery predicate
- Correlated subquery
- Noncorrelated subquery
- Correlation name
- Derived table

The syntax of a subquery with EXISTS

```
WHERE [NOT] EXISTS (subquery)
```

A query that returns vendors without invoices

```
SELECT VendorID, VendorName, VendorState
FROM Vendors
WHERE NOT EXISTS
     (SELECT *
     FROM Invoices
     WHERE Invoices.VendorID = Vendors.VendorID);
```

The result set (88 rows)

	VendorID	VendorName	VendorState
32	33	Nielson	OH
33	35	Cal State Termite	CA
34	36	Graylift	CA
35	38	Venture Communications Int'l	NY
36	39	Custom Printing Company	MO
37	40	Nat Assoc of College Stores	OH

A subquery coded in the FROM clause

```
SELECT Invoices.VendorID, MAX(InvoiceDate) AS LatestInv,
    AVG(InvoiceTotal) AS AvgInvoice
FROM Invoices JOIN
    (SELECT TOP 5 VendorID, AVG(InvoiceTotal) AS AvgInvoice
    FROM Invoices
    GROUP BY VendorID
    ORDER BY AvgInvoice DESC) AS TopVendor
    ON Invoices.VendorID = TopVendor.VendorID
GROUP BY Invoices.VendorID
ORDER BY LatestInv DESC;
```

The derived table generated by the subquery

	VendorID	AvgInvoice
1	110	23978.482
2	72	10963.655
3	104	7125.34
4	99	6940.25
5	119	4901.26

The result set

	VendorID	LatestInv	AvgInvoice
1	110	2020-01-31	23978.482
2	72	2020-01-10	10963.655
3	99	2019-12-18	6940.25
4	104	2019-11-21	7125.34
5	119	2019-11-11	4901.26

A correlated subquery in the SELECT clause

```
SELECT DISTINCT VendorName,
    (SELECT MAX(InvoiceDate) FROM Invoices
    WHERE Invoices.VendorID = Vendors.VendorID) AS LatestInv
FROM Vendors
ORDER BY LatestInv DESC;
```

The result set (122 rows)

	VendorName	LatestInv
1	Federal Express Corporation	2020-02-02
2	Blue Cross	2020-02-01
3	Malloy Lithographing Inc	2020-01-31
4	Cardinal Business Media, Inc.	2020-01-28
5	Zylka Design	2020-01-25
6	Ford Motor Credit Company	2020-01-24
7	United Parcel Service	2020-01-24
8	Ingram	2020-01-21
9	Wakefield Co	2020-01-20

The same query restated using a join

```
SELECT VendorName, MAX(InvoiceDate) AS LatestInv
FROM Vendors LEFT JOIN Invoices ON Vendors.VendorID =
Invoices.VendorID
GROUP BY VendorName
ORDER BY LatestInv DESC;
```

The same result set (122 rows)

	VendorState	VendorName	SumOfInvoices
1	AZ	Wells Fargo Bank	662.00
2	CA	Digital Dreamworks	7125.34
3	DC	Reiter's Scientific & Pro Books	600.00
4	MA	Dean Witter Reynolds	1367.50
5	MI	Malloy Lithographing Inc	119892.41
6	NV	United Parcel Service	23177.96
7	OH	Edward Data Services	207.78
8	PA	Cardinal Business Media, Inc.	265.36

A complex query with three subqueries (part 1)

```
SELECT Summary1.VendorState, Summary1.VendorName,
TopInState.SumOfInvoices
FROM
        (SELECT V_Sub.VendorState, V_Sub.VendorName,
            SUM(I_Sub.InvoiceTotal) AS SumOfInvoices
        FROM Invoices AS I_Sub JOIN Vendors AS V_Sub
            ON I_Sub.VendorID = V_Sub.VendorID
        GROUP BY V_Sub.VendorState, V_Sub.VendorName)
        AS Summary1
```

A complex query with three subqueries (part 2)

```
    JOIN
        (SELECT Summary2.VendorState,
            MAX(Summary2.SumOfInvoices) AS SumOfInvoices
        FROM
            (SELECT V_Sub.VendorState, V_Sub.VendorName,
                SUM(I_Sub.InvoiceTotal) AS SumOfInvoices
            FROM Invoices AS I_Sub JOIN Vendors AS V_Sub
                ON I_Sub.VendorID = V_Sub.VendorID
            GROUP BY V_Sub.VendorState, V_Sub.VendorName)
            AS Summary2
        GROUP BY Summary2.VendorState) AS TopInState
    ON Summary1.VendorState = TopInState.VendorState AND
       Summary1.SumOfInvoices = TopInState.SumOfInvoices
ORDER BY Summary1.VendorState;
```

The partial result set of the complex vendor query (10 rows total)

	VendorState	VendorName	SumOfInvoices
1	AZ	Wells Fargo Bank	662.00
2	CA	Digital Dreamworks	7125.34
3	DC	Reiter's Scientific & Pro Books	600.00
4	MA	Dean Witter Reynolds	1367.50
5	MI	Malloy Lithographing Inc	119892.41
6	NV	United Parcel Service	23177.96
7	OH	Edward Data Services	207.78
8	PA	Cardinal Business Media, Inc.	265.36

A procedure for building complex queries

1. State the problem to be solved by the query in English.
2. Use pseudocode to outline the query.
3. If necessary, use pseudocode to outline each subquery.
4. Code the subqueries and test them to be sure that they return the correct data.
5. Code and test the final query.

Pseudocode for the query

```
SELECT Summary1.VendorState, Summary1.VendorName,
    TopInState.SumOfInvoices
FROM (Derived table returning VendorState, VendorName,
    SumOfInvoices) AS Summary1
    JOIN (Derived table returning VendorState,
        MAX(SumOfInvoices)) AS TopInState
    ON Summary1.VendorState = TopInState.VendorState AND
       Summary1.SumOfInvoices = TopInState.SumOfInvoices
ORDER BY Summary1.VendorState;
```

The code for the Summary1 and Summary2 subqueries

```
SELECT V_Sub.VendorState, V_Sub.VendorName,
    SUM(I_Sub.InvoiceTotal) AS SumOfInvoices
FROM Invoices AS I_Sub JOIN Vendors AS V_Sub
    ON I_Sub.VendorID = V_Sub.VendorID
GROUP BY V_Sub.VendorState, V_Sub.VendorName;
```

The result of the Summary1 and Summary2 subqueries (34 rows)

	VendorState	VendorName	SumOfInvoices
10	MA	Dean Witter Reynolds	1367.50
11	CA	Digital Dreamworks	7125.34
12	CA	Dristas Groom & McCormick	220.00
13	OH	Edward Data Services	207.78

Pseudocode for the TopInState subquery

```
SELECT Summary2.VendorState, MAX(Summary2.SumOfInvoices)
FROM (Derived table returning VendorState, VendorName,
      SumOfInvoices)
    AS Summary2
GROUP BY Summary2.VendorState;
```

The result of the TopInState subquery (10 rows)

	VendorState	SumOfInvoices
1	AZ	662.00
2	CA	7125.34
3	DC	600.00
4	MA	1367.50

The syntax of a CTE

```
WITH cte_name1 AS (query_definition1)
[, cte_name2 AS (query_definition2)]
[...]
sql_statement
```

Two CTEs and a query that uses them

```
WITH Summary AS
(
    SELECT VendorState, VendorName, SUM(InvoiceTotal)
        AS SumOfInvoices
    FROM Invoices JOIN Vendors
      ON Invoices.VendorID = Vendors.VendorID
    GROUP BY VendorState, VendorName
),
TopInState AS
(
    SELECT VendorState, MAX(SumOfInvoices) AS SumOfInvoices
    FROM Summary
    GROUP BY VendorState
)
SELECT Summary.VendorState, Summary.VendorName,
    TopInState.SumOfInvoices
FROM Summary JOIN TopInState
    ON Summary.VendorState = TopInState.VendorState AND
       Summary.SumOfInvoices = TopInState.SumOfInvoices
ORDER BY Summary.VendorState;
```

The partial result set of the query that uses CTEs (10 rows total)

	VendorState	VendorName	SumOfInvoices
1	AZ	Wells Fargo Bank	662.00
2	CA	Digital Dreamworks	7125.34
3	DC	Reiter's Scientific & Pro Books	600.00
4	MA	Dean Witter Reynolds	1367.50
5	MI	Malloy Lithographing Inc	119892.41
6	NV	United Parcel Service	23177.96
7	OH	Edward Data Services	207.78
8	PA	Cardinal Business Media, Inc.	265.36

The Employees table

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

A recursive CTE that returns hierarchical data

```
WITH EmployeesCTE AS
(
        -- Anchor member
        SELECT EmployeeID,
            FirstName + ' ' + LastName As EmployeeName,
            1 As Rank
        FROM Employees
        WHERE ManagerID IS NULL
    UNION ALL
        -- Recursive member
        SELECT Employees.EmployeeID,
            FirstName + ' ' + LastName,
            Rank + 1
        FROM Employees JOIN EmployeesCTE
          ON Employees.ManagerID = EmployeesCTE.EmployeeID
)
SELECT *
FROM EmployeesCTE
ORDER BY Rank, EmployeeID;
```

The final result set

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

Terms to know for CTEs

- Common table expression (CTE)
- Recursive query
- Recursive CTE