Chapter 7

How to code subqueries

Objectives

Applied

1. Code SELECT statements that require subqueries.
2. Code SELECT statements that use common table expressions (CTEs) to define the subqueries, including recursive CTEs.

Knowledge

1. Describe the way subqueries can be used in the WHERE, HAVING, FROM and SELECT clauses of a SELECT statement.
2. Describe the difference between a correlated subquery and a noncorrelated subquery.
3. 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 invoice_number, invoice_date, invoice_total
FROM invoices
WHERE invoice_total >
    (SELECT AVG(invoice_total)
     FROM invoices)
ORDER BY invoice_total
```

The value returned by the subquery

```
1879.741316
```

The result set

invoice_number	invoice_date	invoice_total
989319-487	2022-06-20	1927.54
97/522	2022-06-28	1962.13
989319-417	2022-07-23	2051.59
989319-427	2022-06-16	2115.81
989319-477	2022-06-08	2184.11

```
(21 rows)
```

A query that uses an inner join

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices JOIN vendors
    ON invoices.vendor_id = vendors.vendor_id
WHERE vendor_state = 'CA'
ORDER BY invoice_date
```

The result set

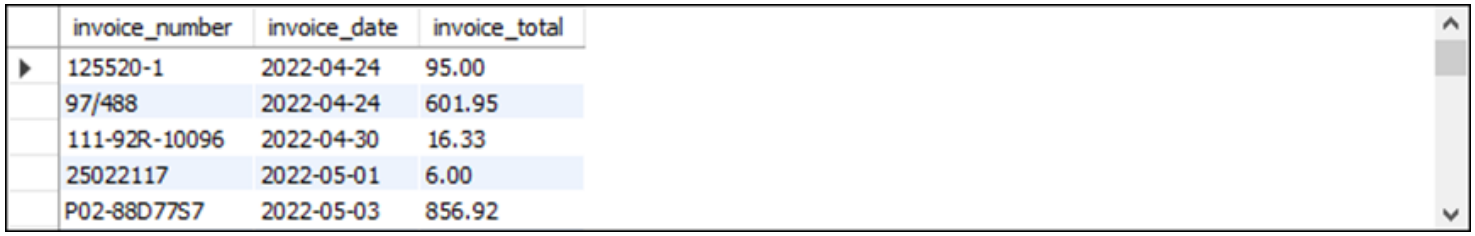
invoice_number	invoice_date	invoice_total
125520-1	2022-04-24	95.00
97/488	2022-04-24	601.95
111-92R-10096	2022-04-30	16.33
25022117	2022-05-01	6.00
P02-88D77S7	2022-05-03	856.92

```
(40 rows)
```

The same query restated with a subquery

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
WHERE vendor_id IN
    (SELECT vendor_id
     FROM vendors
     WHERE vendor_state = 'CA')
ORDER BY invoice_date
```

The same result set



invoice_number	invoice_date	invoice_total
125520-1	2022-04-24	95.00
97/488	2022-04-24	601.95
111-92R-10096	2022-04-30	16.33
25022117	2022-05-01	6.00
P02-88D77S7	2022-05-03	856.92

(40 rows)

Advantages of joins

- A join can include columns from both tables.
- A join is more intuitive when it uses an existing relationship.

Advantages of subqueries

- A subquery can pass an aggregate value to the main query.
- A subquery is more intuitive when it uses an ad hoc relationship.
- Long, complex queries can be easier to code using subqueries.

The syntax of a WHERE clause that uses an IN phrase

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

A query that gets vendors without invoices

```
SELECT vendor_id, vendor_name, vendor_state
FROM vendors
WHERE vendor_id NOT IN
    (SELECT DISTINCT vendor_id
     FROM invoices)
ORDER BY vendor_id
```

The result of the subquery that gets distinct vendor ids with invoices

vendor_id
34
37
48
72
80
81
82

```
(34 rows)
```

The result set that gets vendors without invoices

vendor_id	vendor_name	vendor_state
33	Nielson	OH
35	Cal State Termite	CA
36	Graylift	CA
38	Venture Communications Int'l	NY
39	Custom Printing Company	MO
40	Nat Assoc of College Stores	OH

```
(88 rows)
```

The query restated without a subquery

```
SELECT v.vendor_id, vendor_name, vendor_state
FROM vendors v LEFT JOIN invoices i
    ON v.vendor_id = i.vendor_id
WHERE i.vendor_id IS NULL
ORDER BY v.vendor_id
```

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 a WHERE condition

```
SELECT invoice_number, invoice_date,
    invoice_total - payment_total - credit_total
    AS balance_due
FROM invoices
WHERE invoice_total - payment_total - credit_total > 0
  AND invoice_total - payment_total - credit_total <
    (
     SELECT AVG(invoice_total - payment_total -
                credit_total)
     FROM invoices
     WHERE invoice_total - payment_total - credit_total > 0
    )
ORDER BY invoice_total DESC
```

The value returned by the subquery

```
2910.947273
```

The result set

invoice_number	invoice_date	balance_due
31361833	2022-07-21	579.42
9982771	2022-07-24	503.20
547480102	2022-08-01	224.00
134116	2022-07-28	90.36
39104	2022-07-10	85.31

```
(9 rows)
```

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 NOT IN (1, 2)`

A query that uses ALL

```
SELECT vendor_name, invoice_number, invoice_total
FROM invoices i JOIN vendors v ON i.vendor_id = v.vendor_id
WHERE invoice_total > ALL
     (SELECT invoice_total
      FROM invoices
      WHERE vendor_id = 34)
ORDER BY vendor_name
```

The result of the subquery

invoice_total
116.54
1083.58

The result set

vendor_name	invoice_number	invoice_total
Bertelsmann Industry Svcs. Inc	509786	6940.25
Cahners Publishing Company	587056	2184.50
Computerworld	367447	2433.00
Data Reproductions Corp	40318	21842.00

(25 rows)

How the ANY keyword works

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

A query that uses ANY

```
SELECT vendor_name, invoice_number, invoice_total
FROM vendors JOIN invoices
  ON vendors.vendor_id = invoices.vendor_id
WHERE invoice_total < ANY
    (SELECT invoice_total
     FROM invoices
     WHERE vendor_id = 115)
```

The result of the subquery with invoice totals for vendor 115

invoice_total
6.00
6.00
25.67
6.00

The result set for invoices with totals less than any invoice for vendor 115

vendor_name	invoice_number	invoice_total
Federal Express Corporation	963253251	15.50
Pacific Bell	111-92R-10096	16.33
Roadway Package System, Inc	25022117	6.00
Compuserve	21-4748363	9.95
Federal Express Corporation	4-321-2596	10.00

```
(17 rows)
```

A query that uses a correlated subquery

```
SELECT vendor_id, invoice_number, invoice_total
FROM invoices i
WHERE invoice_total >
     (SELECT AVG(invoice_total)
      FROM invoices
      WHERE vendor_id = i.vendor_id)
ORDER BY vendor_id, invoice_total
```

The value returned by the subquery for vendor 95

```
28.501667
```

The result set

vendor_id	invoice_number	invoice_total
83	31359783	1575.00
95	111-92R-10095	32.70
95	111-92R-10093	39.77
95	111-92R-10092	46.21
110	P-0259	26881.40

```
(36 rows)
```

The syntax of a subquery that uses the EXISTS operator

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

A query that gets vendors without invoices

```
SELECT vendor_id, vendor_name, vendor_state
FROM vendors
WHERE NOT EXISTS
    (SELECT *
     FROM invoices
     WHERE vendor_id = vendors.vendor_id)
```

The result set

vendor_id	vendor_name	vendor_state
33	Nielson	OH
35	Cal State Termite	CA
36	Graylift	CA
38	Venture Communications Int'l	NY
39	Custom Printing Company	MO
40	Nat Assoc of College Stores	OH

```
(88 rows)
```

A subquery in the SELECT clause

```
SELECT vendor_name,
    (SELECT MAX(invoice_date) FROM invoices
     WHERE vendor_id = vendors.vendor_id) AS latest_inv
FROM vendors
ORDER BY latest_inv DESC
```

The result set

vendor_name	latest_inv
Federal Express Corporation	2022-08-02
Blue Cross	2022-08-01
Malloy Lithographing Inc	2022-07-31
Cardinal Business Media, Inc.	2022-07-28
Zylka Design	2022-07-25

```
(122 rows)
```

The same query restated using a join

```
SELECT vendor_name, MAX(invoice_date) AS latest_inv
FROM vendors v
    LEFT JOIN invoices i ON v.vendor_id = i.vendor_id
GROUP BY vendor_name
ORDER BY latest_inv DESC
```

The same result set

vendor_name	latest_inv
Federal Express Corporation	2022-08-02
Blue Cross	2022-08-01
Malloy Lithographing Inc	2022-07-31
Cardinal Business Media, Inc.	2022-07-28
Zylka Design	2022-07-25

(122 rows)

A query that uses an inline view

```
SELECT vendor_state,
    MAX(sum_of_invoices) AS max_sum_of_invoices
FROM
(
    SELECT vendor_state, vendor_name,
        SUM(invoice_total) AS sum_of_invoices
    FROM vendors v JOIN invoices i
        ON v.vendor_id = i.vendor_id
    GROUP BY vendor_state, vendor_name
) t
GROUP BY vendor_state
ORDER BY vendor_state
```

The result of the subquery (an inline view)

vendor_state	vendor_name	sum_of_invoices
NV	United Parcel Service	23177.96
TN	Federal Express Corporation	4378.02
CA	Evans Executone Inc	95.00
CA	Zylka Design	6940.25
AZ	Wells Fargo Bank	662.00
CA	Pacific Bell	171.01
CA	Roadway Package System, Inc	43.67
CA	Fresno County Tax Collector	856.92

```
(34 rows)
```

The result set

vendor_state	max_sum_of_invoices
AZ	662.00
CA	7125.34
DC	600.00

```
(10 rows)
```

A complex query that uses three subqueries

```
SELECT t1.vendor_state, vendor_name, t1.sum_of_invoices
FROM
    (
        -- sum of invoice totals by vendor
        SELECT vendor_state, vendor_name,
            SUM(invoice_total) AS sum_of_invoices
        FROM vendors v JOIN invoices i
            ON v.vendor_id = i.vendor_id
        GROUP BY vendor_state, vendor_name
    ) t1
```

A complex query (continued)

```
JOIN
        (
            -- top sum of invoice totals by state
            SELECT vendor_state,
                   MAX(sum_of_invoices)
                   AS sum_of_invoices
            FROM
            (
                 -- sum of invoice totals by vendor
                 SELECT vendor_state, vendor_name,
                     SUM(invoice_total)
                     AS sum_of_invoices
                 FROM vendors v JOIN invoices i
                     ON v.vendor_id = i.vendor_id
                 GROUP BY vendor_state, vendor_name
            ) t2
            GROUP BY vendor_state
        ) t3
    ON t1.vendor_state = t3.vendor_state AND
       t1.sum_of_invoices = t3.sum_of_invoices
ORDER BY vendor_state
```

The result set of the complex query

vendor_state	vendor_name	sum_of_invoices
AZ	Wells Fargo Bank	662.00
CA	Digital Dreamworks	7125.34
DC	Reiter's Scientific & Pro Books	600.00
MA	Dean Witter Reynolds	1367.50

```
(10 rows)
```

A procedure for building complex queries

1. State the problem to be solved by the query in plain language.
2. Use pseudocode to outline the query.
3. Code the subqueries and test them to be sure that they return the correct data.
4. Code and test the final query.

Pseudocode for the query

```
SELECT vendor_state, vendor_name, sum_of_invoices
FROM (subquery returning vendor_state, vendor_name,
      sum_of_invoices)
JOIN (subquery returning vendor_state,
      largest_sum_of_invoices)
    ON vendor_state AND sum_of_invoices
ORDER BY vendor_state
```

The code for the first subquery

```
SELECT vendor_state, vendor_name,
    SUM(invoice_total) AS sum_of_invoices
FROM vendors v JOIN invoices i
    ON v.vendor_id = i.vendor_id
GROUP BY vendor_state, vendor_name
```

The result set for the first subquery

vendor_state	vendor_name	sum_of_invoices
NV	United Parcel Service	23177.96
TN	Federal Express Corporation	4378.02
CA	Evans Executone Inc	95.00

```
(34 rows)
```

The code for the second subquery

```
SELECT vendor_state,
    MAX(sum_of_invoices) AS sum_of_invoices
FROM
(
     SELECT vendor_state, vendor_name,
         SUM(invoice_total) AS sum_of_invoices
     FROM vendors v JOIN invoices i
         ON v.vendor_id = i.vendor_id
     GROUP BY vendor_state, vendor_name
) t
GROUP BY vendor_state
```

The result set for the second subquery

vendor_state	sum_of_invoices
NV	23177.96
TN	4378.02
CA	7125.34

(10 rows)

The syntax of a CTE

```
WITH [RECURSIVE] cte_name1 AS (subquery1)
[, cte_name2 AS (subquery2)]
[...]
sql_statement
```

Two CTEs and a query that uses them

```
WITH summary AS
(
    SELECT vendor_state, vendor_name,
        SUM(invoice_total) AS sum_of_invoices
    FROM vendors v JOIN invoices i
        ON v.vendor_id = i.vendor_id
    GROUP BY vendor_state, vendor_name
),
top_in_state AS
(
    SELECT vendor_state,
        MAX(sum_of_invoices) AS sum_of_invoices
    FROM summary
    GROUP BY vendor_state
)
```

Two CTEs and a query that uses them (continued)

```
SELECT summary.vendor_state, summary.vendor_name,
       top_in_state.sum_of_invoices
FROM summary JOIN top_in_state
    ON summary.vendor_state = top_in_state.vendor_state AND
       summary.sum_of_invoices = top_in_state.sum_of_invoices
ORDER BY summary.vendor_state
```

The result set

vendor_state	vendor_name	sum_of_invoices
AZ	Wells Fargo Bank	662.00
CA	Digital Dreamworks	7125.34
DC	Reiter's Scientific & Pro Books	600.00
MA	Dean Witter Reynolds	1367.50
MI	Malloy Lithographing Inc	119892.41
NV	United Parcel Service	23177.96
OH	Edward Data Services	207.78
PA	Cardinal Business Media, Inc.	265.36

```
(10 rows)
```

The Employees table

employee_id	last_name	first_name	department_number	manager_id
1	Smith	Cindy	2	NULL
2	Jones	Elmer	4	1
3	Simonian	Ralph	2	2
4	Hernandez	Olivia	1	9
5	Aaronsen	Robert	2	4
6	Watson	Denise	6	8
7	Hardy	Thomas	5	2
8	O'Leary	Rhea	4	9
9	Locario	Paulo	6	1

A recursive CTE that returns hierarchical data

```
WITH RECURSIVE employees_cte AS
(
        -- Nonrecursive query
        SELECT employee_id,
            CONCAT(first_name, ' ', last_name) AS employee_name,
            1 AS ranking
        FROM employees
        WHERE manager_id IS NULL
    UNION ALL
        -- Recursive query
        SELECT employees.employee_id,
            CONCAT(first_name, ' ', last_name),
            ranking + 1
        FROM employees
            JOIN employees_cte
            ON employees.manager_id = employees_cte.employee_id
)
SELECT *
FROM employees_cte
ORDER BY ranking, employee_id
```

The final result set

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