Chapter 6

How to code subqueries

Objectives

Applied

- Code SELECT statements that require subqueries.
- Code SELECT statements that use subquery factoring clauses 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 subquery factoring clauses.
- Describe the use of a hierarchical query.

Four ways to introduce a subquery in a SELECT statement

- In a WHERE clause as a search condition
- In a HAVING clause as a search condition
- In the FROM clause as a table specification
- In the SELECT clause as a column specification

A subquery in a 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

```
1879.7413
```

The result set

	INVOICE_NUMBER	INVOICE_DATE	INVOICE_TOTAL
1	989319-487	18-APR-08	1927.54
2	97/522	30-APR-08	1962.13
3	989319-417	26-APR-08	2051.59
4	989319-427	25-APR-08	2115.81
5	989319-477	19-APR-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
1	QP58872	25-FEB-08	116.54
2	Q545443	14-MAR-08	1083.58
3	MABO1489	16-APR-08	936.93
4	97/553B	26-APR-08	313.55

(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
1	QP58872	25-FEB-08	116.54
2	Q545443	14-MAR-08	1083.58
3	MABO1489	16-APR-08	936.93
4	97/553B	26-APR-08	313.55

(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 outer query.
- A subquery is more intuitive when it uses an ad hoc relationship.
- Long, complex queries can be easier to code with 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 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

	VENDOR_ID
1	34
2	123
3	100
4	83
5	121
6	113

(34 rows)

The result set

	VENDOR_ID	VENDOR_NAME	VENDOR_STATE
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

(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 with an expression that uses the value returned by a subquery

```
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

```
1669.906
```

The result set

	INVOICE_NUMBER	INVOICE_DATE	BALANCE_DUE
1	31359783	23-MAY-08	1575
2	97/553	27-APR-08	904.14
3	I77271-O01	05-JUN-08	662
4	31361833	23-MAY-08	579.42
5	9982771	03-JUN-08	503.2
6	97/553B	26-APR-08	313.55

```
(33 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 <> 1) AND (x <> 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
1	116.54
2	1083.58

The result set

	VENDOR_NAME	INVOICE_NUMBER	INVOICE_TOTAL
1	Bertelsmann Industry Svcs. Inc	509786	6940.25
2	Cahners Publishing Company	587056	2184.5
3	Computerworld	367447	2433
4	Data Reproductions Corp	40318	21842

(25 rows)

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 ANY

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

The result of the subquery

	INVOICE_TOTAL
1	25.67
2	6
3	6
4	6

The result set

	VENDOR_NAME	INVOICE_NUMBER	INVOICE_TOTAL
1	Boucher Communications Inc	24863706	6
2	Reiter's Scientific & Pro Books	25022117	6
3	Champion Printing Company	24780512	6
4	Opamp Technical Books	21-4748363	9.95
5	Capital Resource Credit	21-4923721	9.95

A query that uses a correlated subquery

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

The value returned by the subquery for vendor 95

```
28.50166...
```

The result set

	VENDOR_ID	INVOICE_NUMBER	INVOICE_TOTAL
6	83	31359783	1575
7	95	111-92R-10095	32.7
8	95	111-92R-10093	39.77
9	95	111-92R-10092	46.21
10	110	P-0259	26881.4

(36 rows)

The syntax of a subquery with EXISTS

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

A query that returns vendors without invoices

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

The result set

	VENDOR_ID	VENDOR_NAME	VENDOR_STATE
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

(88 rows)

A query that uses an inline view

```
SELECT i.vendor_id,
    MAX(invoice_date) AS last_invoice_date,
    AVG(invoice_total) AS average_invoice_total
FROM invoices i JOIN
    (
    SELECT vendor_id,
        AVG(invoice_total) AS average_invoice_total
    FROM invoices
    HAVING AVG(invoice_total) > 4900
    GROUP BY vendor_id
    ) v
    ON i.vendor_id = v.vendor_id
GROUP BY i.vendor_id
ORDER BY MAX(invoice_date) DESC
```

The result of the subquery (an inline view)

	VENDOR_ID	AVERAGE_INVOICE_TOTAL
1	72	10963.655
2	119	4901.26
3	110	23978.482
4	99	6940.25
5	104	7125.34

The result set

	VENDOR_ID	LAST_INVOICE_DATE	AVERAGE_INVOICE_TOTAL
1	72	18-JUL-08	10963.655
2	119	04-JUN-08	4901.26
3	104	03-JUN-08	7125.34
4	99	31-MAY-08	6940.25
5	110	08-MAY-08	23978.482

A query that uses a correlated subquery

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

The result set

	VENDOR_NAME	LATEST_INV
1	IBM	14-MAR-08
2	Wang Laboratories, Inc.	16-APR-08
3	Reiter's Scientific & Pro Books	17-APR-08
4	United Parcel Service	26-APR-08
5	Wakefield Co	26-APR-08
6	Zylka Design	01-MAY-08
7	Abbey Office Furnishings	02-MAY-08

(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
```

The same result set

	VENDOR_NAME	LATEST_INV
1	IBM	14-MAR-08
2	Wang Laboratories, Inc.	16-APR-08
3	Reiter's Scientific & Pro Books	17-APR-08
4	United Parcel Service	26-APR-08
5	Wakefield Co	26-APR-08
6	Zylka Design	01-MAY-08
7	Abbey Office Furnishings	02-MAY-08

(122 rows)

A query that uses three subqueries

```
SELECT summary1.vendor_state, summary1.vendor_name,
       top_in_state.sum_of_invoices
FROM
    (
    SELECT v_sub.vendor_state, v_sub.vendor_name,
        SUM(i_sub.invoice_total) AS sum_of_invoices
    FROM invoices i_sub JOIN vendors v_sub
        ON i_sub.vendor_id = v_sub.vendor_id
    GROUP BY v_sub.vendor_state, v_sub.vendor_name
    ) summary1
```

The query (continued)

```
    JOIN
        (
        SELECT summary2.vendor_state,
            MAX(summary2.sum_of_invoices) AS sum_of_invoices
        FROM
            (
            SELECT v_sub.vendor_state, v_sub.vendor_name,
                SUM(i_sub.invoice_total) AS sum_of_invoices
            FROM invoices i_sub JOIN vendors v_sub
                ON i_sub.vendor_id = v_sub.vendor_id
            GROUP BY v_sub.vendor_state, v_sub.vendor_name
            ) summary2
        GROUP BY summary2.vendor_state
        ) top_in_state
    ON summary1.vendor_state =
            top_in_state.vendor_state AND
       summary1.sum_of_invoices =
           top_in_state.sum_of_invoices
ORDER BY summary1.vendor_state
```

The result set

	VENDOR_STATE	VENDOR_NAME	SUM_OF_INVOICES
1	AZ	Wells Fargo Bank	662
2	CA	Digital Dreamworks	7125.34
3	DC	Reiter's Scientific & Pro Books	600
4	MA	Dean Witter Reynolds	1367.5
5	MI	Malloy Lithographing Inc	119892.41
6	NV	United Parcel Service	23177.96
7	OH	Edward Data Services	207.78

(10 rows)

A procedure for building complex queries

- State the problem to be solved in English.
- Use pseudocode to outline the query.
- If necessary, use pseudocode to outline each subquery.
- Code the subqueries and test them.
- Code and test the final query.

Pseudocode for the query

```
SELECT summary1.vendor_state, summary1.vendor_name,
       top_in_state.sum_of_invoices
FROM (inline view returning
          vendor_state, vendor_name, sum_of_invoices)
          AS summary1
    JOIN (inline view returning vendor_state,
              max(sum_of_invoices))
              AS top_in_state
         ON summary1.vendor_state =
            top_in_state.vendor_state
        AND summary1.sum_of_invoices =
            top_in_state.sum_of_invoices
ORDER BY summary1.vendor_state
```

Pseudocode for the Top_In_State subquery

```
SELECT summary2.vendor_state,
       MAX(summary2.sum_of_invoices)
FROM (inline view returning vendor_state,
          vendor_name, sum_of_invoices)
          AS summary2
GROUP BY summary2.vendor_state
```

The Summary1 and Summary2 subqueries

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

The result of Summary1 and Summary2

	VENDOR_STATE	VENDOR_NAME	SUM_OF_INVOICES
1	AZ	Wells Fargo Bank	662
2	CA	Abbey Office Furnishings	17.5
3	CA	Bertelsmann Industry Svcs. Inc	6940.25

(34 rows)

The result of the Top_In_State subquery

	VENDOR_STATE	SUM_OF_INVOICES
1	CA	7125.34
2	MA	1367.5
3	OH	207.78

(10 rows)

The syntax of a subquery factoring clause

```
WITH query_name1 AS (query_definition1)
[, query_name2 AS (query_definition2)]
[...]
sql_statement
```

Two subfactoring query clauses

```
WITH summary AS
(
    SELECT vendor_state, vendor_name,
        SUM(invoice_total) AS sum_of_invoices
    FROM invoices
        JOIN vendors
            ON invoices.vendor_id = vendors.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
)
```

The query that uses the clauses

```
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
1	AZ	Wells Fargo Bank	662
2	CA	Digital Dreamworks	7125.34
3	DC	Reiter's Scientific & Pro Books	600
4	MA	Dean Witter Reynolds	1367.5
5	MI	Malloy Lithographing Inc	119892.41
6	NV	United Parcel Service	23177.96
7	OH	Edward Data Services	207.78

(10 rows)

A syntax for a query that returns hierarchical data

```
SELECT select_list
FROM table_source
[WHERE search_condition]
START WITH row_specification
CONNECT BY PRIOR connect_expression
[ORDER BY order_by_list]
```

The Employees table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_NUMBER	MANAGER_ID
1	Cindy	Smith	2	(null)
2	Elmer	Jones	4	1
3	Ralph	Simonian	2	2
4	Olivia	Hernandez	1	9
5	Robert	Aaronsen	2	4
6	Denise	Watson	6	8
7	Thomas	Hardy	5	2
8	Rhea	O'Leary	4	9
9	Paulo	Locario	6	1

A query that returns hierarchical data

```
SELECT employee_id,
   first_name || ' ' || last_name AS employee_name,
   LEVEL
FROM employees
START WITH employee_id = 1
CONNECT BY PRIOR employee_id = manager_id
ORDER BY LEVEL, employee_id
```

The result set

	EMPLOYEE_ID	EMPLOYEE_NAME	LEVEL
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