Chapter 7 How to code subqueries

Objectives

Applied

Code SELECT statements that require 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.

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

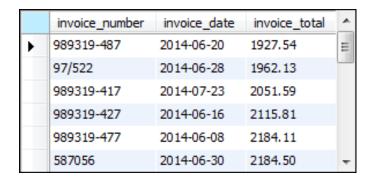
A subquery in a WHERE clause

```
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

1879,741316

The result set



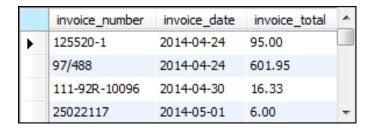
(21 rows)

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



(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	A
•	125520-1	2014-04-24	95.00	
	97/488	2014-04-24	601.95	
	111-92R-10096	2014-04-30	16.33	
	25022117	2014-05-01	6.00	Ŧ

(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.

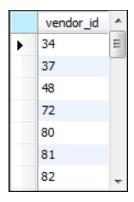
The syntax of a WHERE clause that uses an IN phrase

WHERE test_expression [NOT] IN (subquery)

A query that gets vendors without invoices

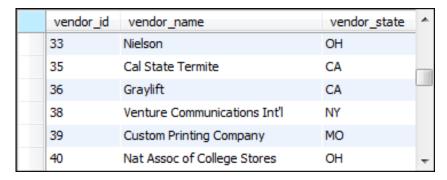
```
SELECT vendor_id, vendor_name, vendor_state
FROM vendors
WHERE vendor_id NOT IN
          (SELECT DISTINCT vendor_id
          FROM invoices)
ORDER BY vendor_id
```

The result of the subquery



(34 rows)

The result set



(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

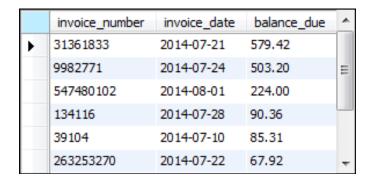
```
WHERE expression comparison_operator [SOME ANY ALL] (subquery)
```

A query with a subquery in a WHERE condition

The value returned by the subquery

2910.947273

The result set



(9 rows)

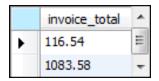
How the ALL keyword works

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)

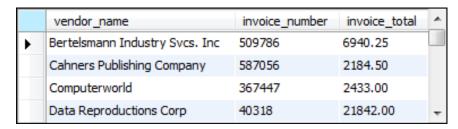
A query that uses ALL

```
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



The result set



(25 rows)

How the ANY keyword works

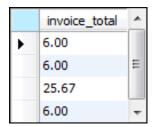
Condition	Equivalent expression
x > ANY (1, 2)	x > 1
x < ANY (1, 2)	x < 2
x = ANY (1, 2)	x IN (1, 2)
x <> ANY (1, 2)	(x <> 1) OR (x <> 2)

A query that uses ANY

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

Slide 16

The result of the subquery



The result set



(17 rows)

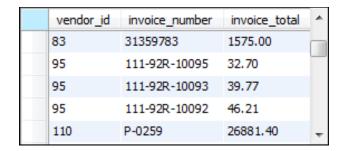
A query that uses a correlated subquery

```
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



(36 rows)

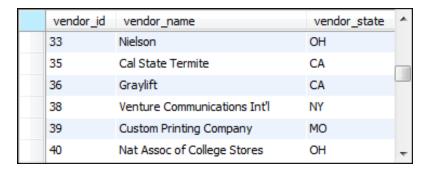
The syntax of a subquery that uses the EXISTS operator

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



(88 rows)

A subquery in the SELECT clause

The result set



(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



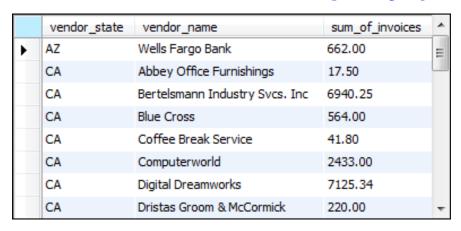
(122 rows)

Slide 21

A query that uses an inline view

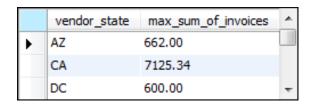
```
SELECT vendor_state,
    MAX(sum_of_invoices) AS max_sum_of_invoices
FROM
(
    SELECT vendor_state, vendor_name,
         SUM(invoice_total) AS sum_of_invoices
    FROM vendors v JOIN invoices i
        ON v.vendor_id = i.vendor_id
    GROUP BY vendor_state, vendor_name
) t
GROUP BY vendor_state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

The result of the subquery (an inline view)



(34 rows)

The result set



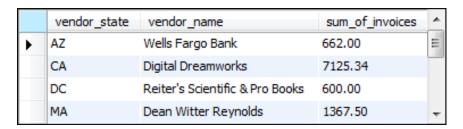
(10 rows)

A complex query that uses three subqueries

A complex query (continued)

```
JOIN
            -- top invoice totals by state
            SELECT vendor state,
                   MAX(sum of invoices)
                   AS sum of invoices
            FROM
                 -- 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



(10 rows)

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. 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

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
```

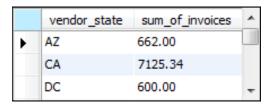
The result set for the first subquery



(34 rows)

The code for the second subquery

The result set for the second subquery



(10 rows)