Subqueries

- 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 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
```

- 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 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_stat = 'CA')
ORDER BY invoice_date
```

- **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 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
```

- 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
```

- 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)
```

- 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 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)
```

- 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 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
```

- 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
```

- A complex query that uses three subqueries:

```
SELECT t1.vendor_state, vendor_name, t1.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) t1

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
```

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.

a. **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
```

b. **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
```

c. 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 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)

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
```

- 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
```